# EECS 2030

# ADVANCED OBJECT ORIENTED PROGRAMMING

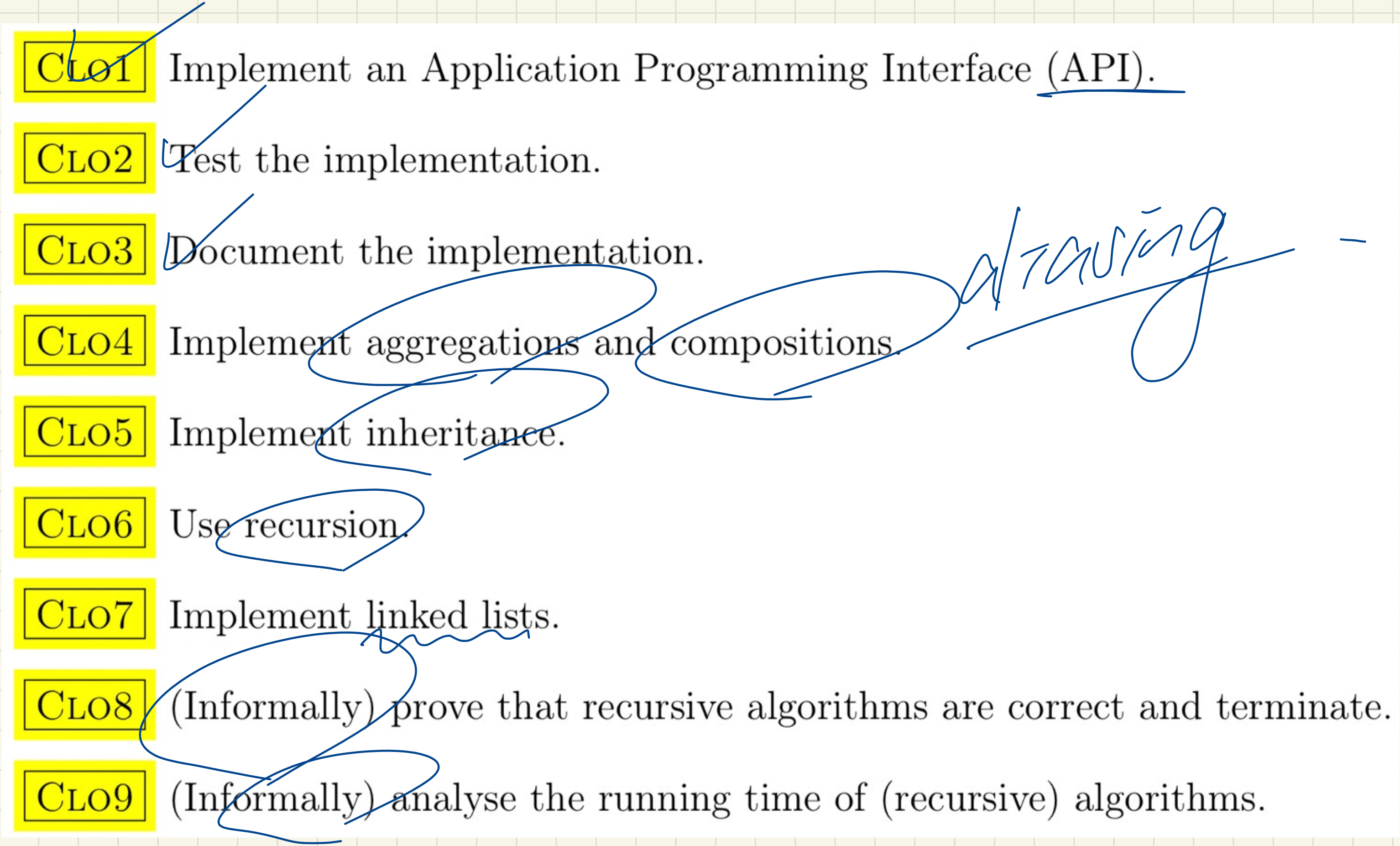
SESSION B FALL 2019

INSTRUCTOR:
JACKIE WANG

# LECTURE 1

Wednesday September 4

# Course Learning Outcomes



CLO1 Implement an Application Programming Interface (API).

CLO2 Test the implementation.

CLO3 Document the implementation.

CLO4 Implement aggregations and compositions. drawing

CLO5 Implement inheritance.

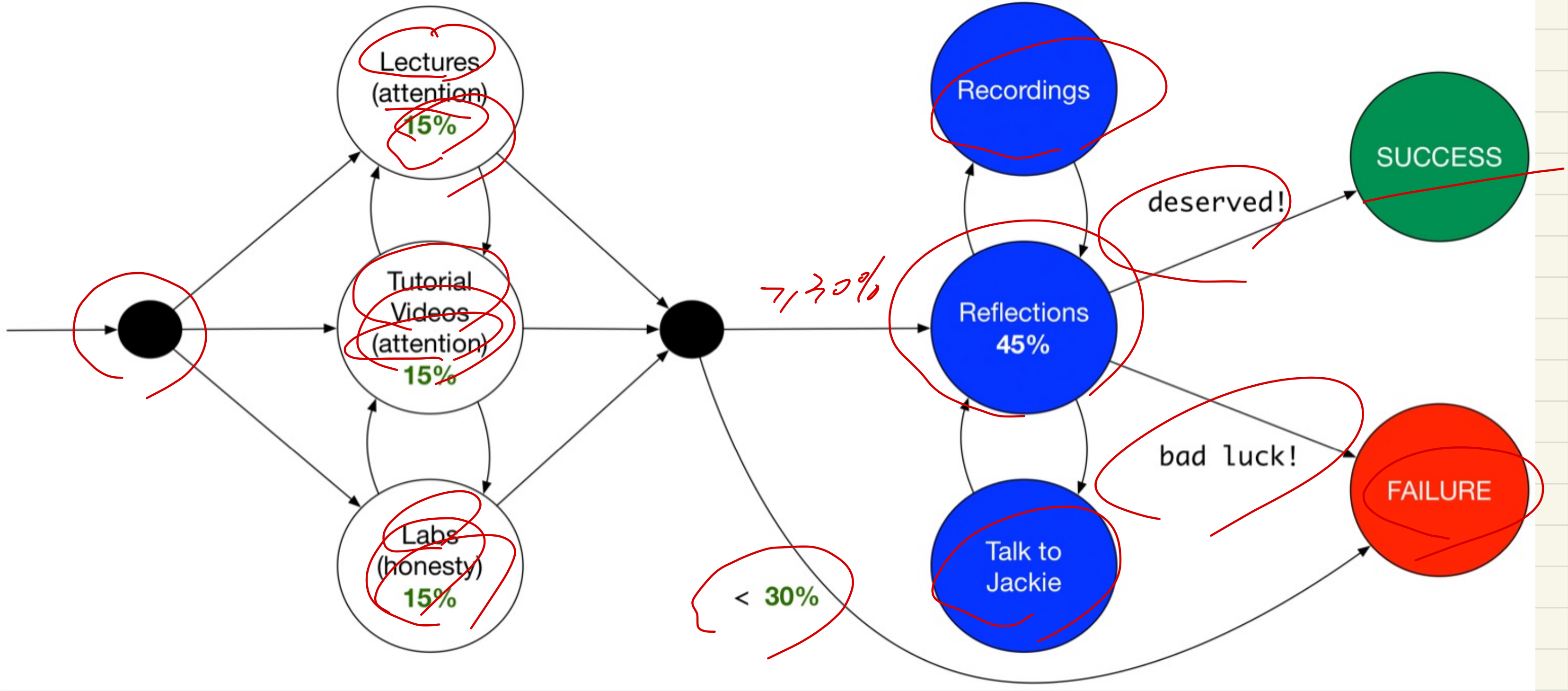
CLO6 Use recursion.

CLO7 Implement linked lists.

CLO8 (Informally) prove that recursive algorithms are correct and terminate.

CLO9 (Informally) analyse the running time of (recursive) algorithms.

# SURVIVAL PATTERN



# Object-Oriented Programming (OOP)

- Templates (Compile-time Java classes)
  - ~ attributes
  - ~ methods
    - Constructor
    - mutator
    - accessor
- Instances / Entities (runtime objects)
  - ~ calling constructor to create objects
  - ~ use of "dot notation" to
    - get attribute values
    - call accessor or mutator

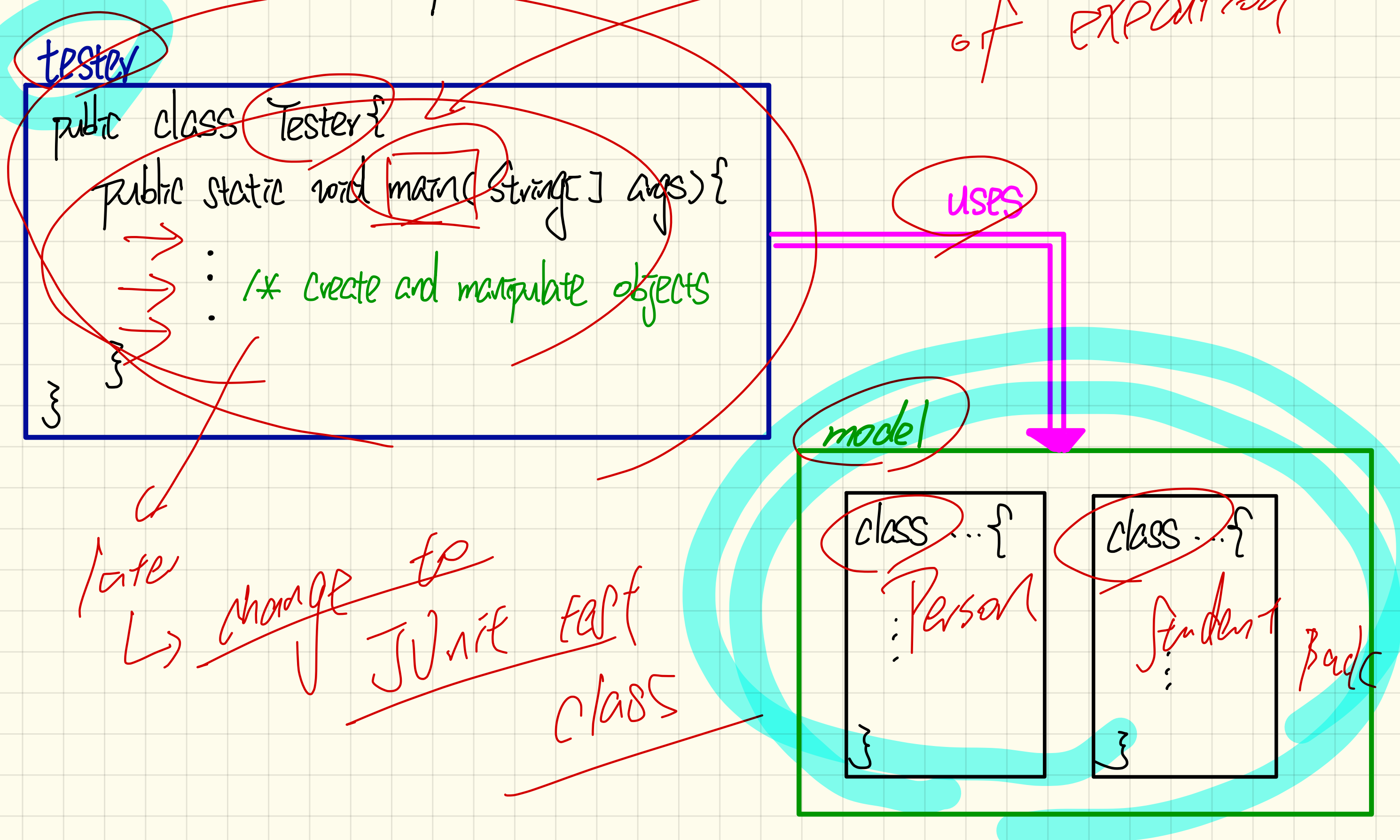
class vs. object

P. spouse. spouse

Person

P. getTuition()

# Test Driven Development (TDD)

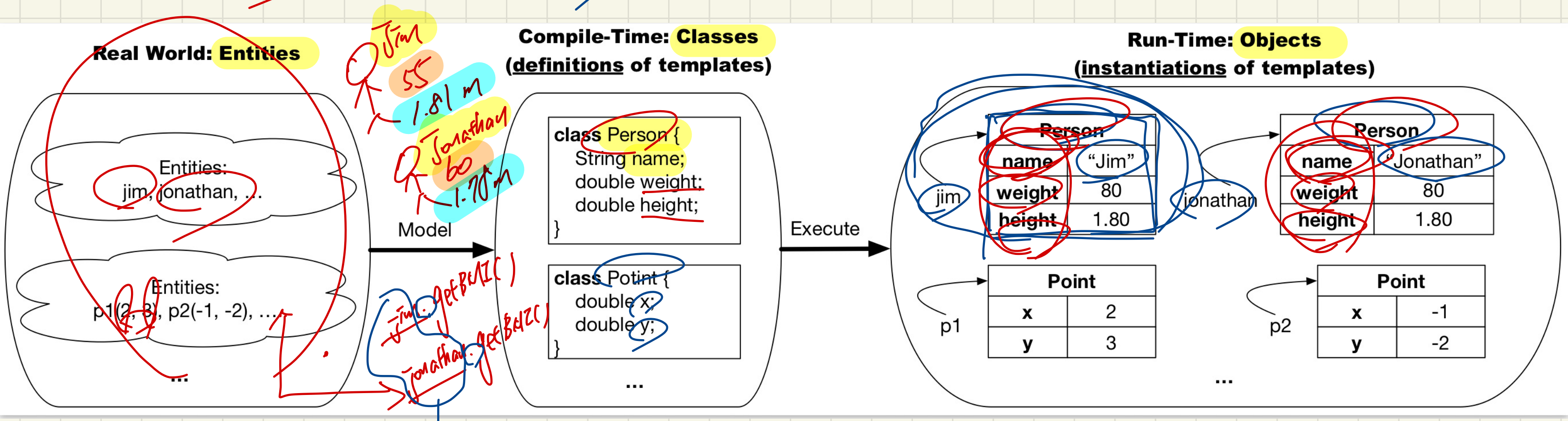


```
public class Tester{
    public static void main(String[] args){
        :
        /* create and manipulate objects
        :
    }
}
```

```
class ...{
    :
}
```

```
class ...{
    :
}
```

# The Observe-Model-Execute Process



```
class Person {
  String name;
  double weight;
  double height;
}
```

```
class Point {
  double x;
  double y;
}
```

...

| Person | |
|---|---|
| name | "Jim" |
| weight | 80 |
| height | 1.80 |

| Person | |
|---|---|
| name | "Jonathan" |
| weight | 80 |
| height | 1.80 |

| Point | |
|---|---|
| x | 2 |
| y | 3 |

| Point | |
|---|---|
| x | -1 |
| y | -2 |

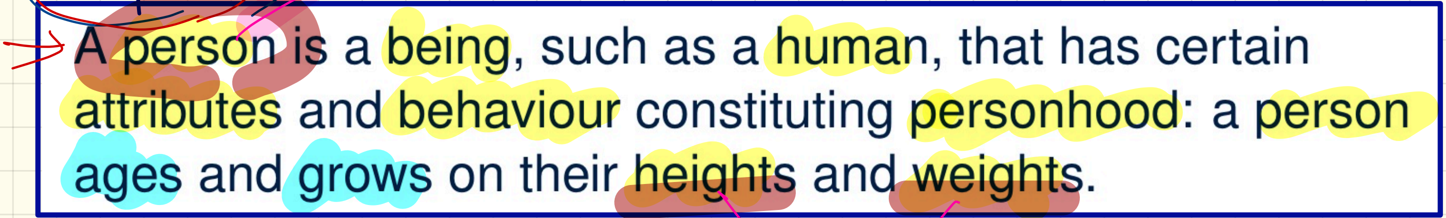
context objects

# Model: From Entities to Classes

Identify Critical Nouns & Verbs

Example 1

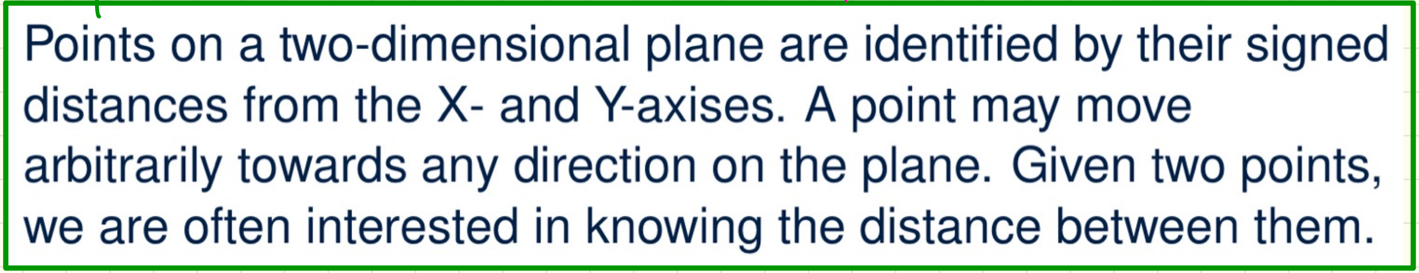
class



A person is a being, such as a human, that has certain attributes and behaviour constituting personhood: a person ages and grows on their heights and weights.

attributes.

Example 2



Points on a two-dimensional plane are identified by their signed distances from the X- and Y-axises. A point may move arbitrarily towards any direction on the plane. Given two points, we are often interested in knowing the distance between them.

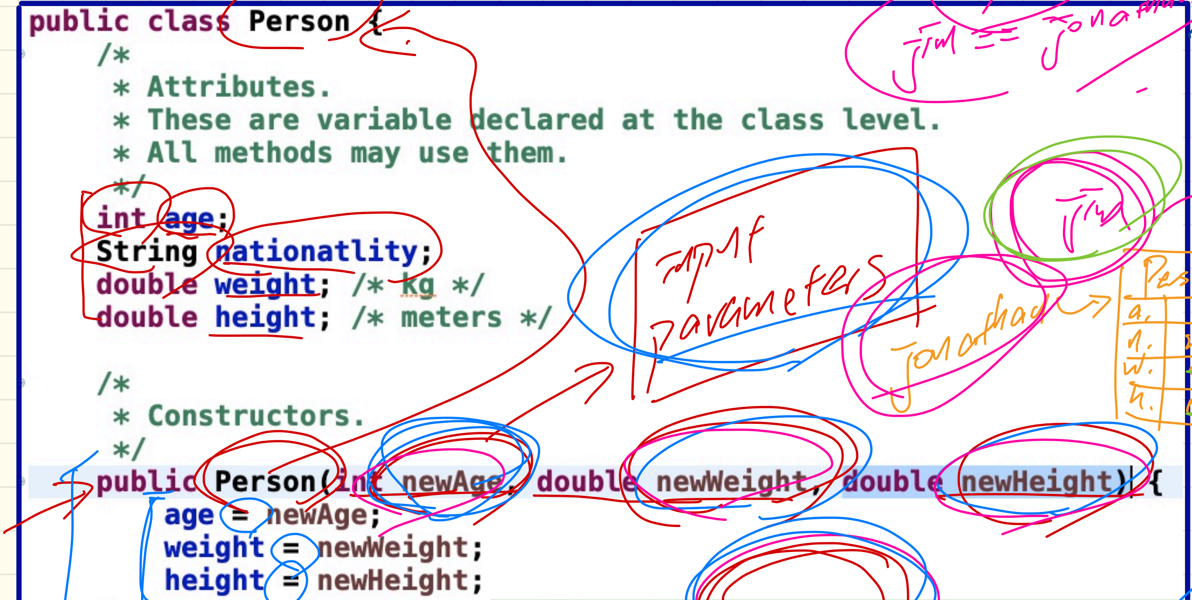
# Constructors

Perspectives :

1. Java
2. Debugger

F

jim == jonathan



```java
public class Person {
    /*
     * Attributes.
     * These are variable declared at the class level.
     * All methods may use them.
     */
    int age;
    String nationatlity;
    double weight; /* kg */
    double height; /* meters */

    /*
     * Constructors.
     */
    public Person(int newAge, double newWeight, double newHeight) {
        age = newAge;
        weight = newWeight;
        height = newHeight;
    }
}
```

input parameters

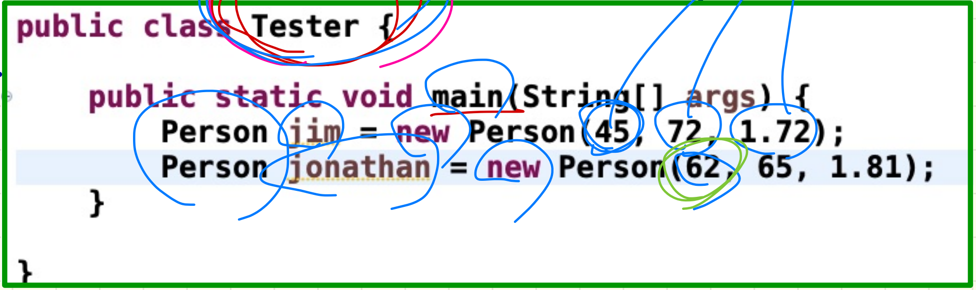
jim

jonathan

| Person | |
|---|---|
| a. | ~~0~~ 45 |
| n. | null |
| w | ~~0.0~~ 72 |
| h. | ~~0.0~~ 1.72 |

| Person | |
|---|---|
| a. | ~~0~~ 62 |
| n. | null |
| w. | ~~0.0~~ 65 |
| h. | ~~0.0~~ 1.81 |

actual arguments



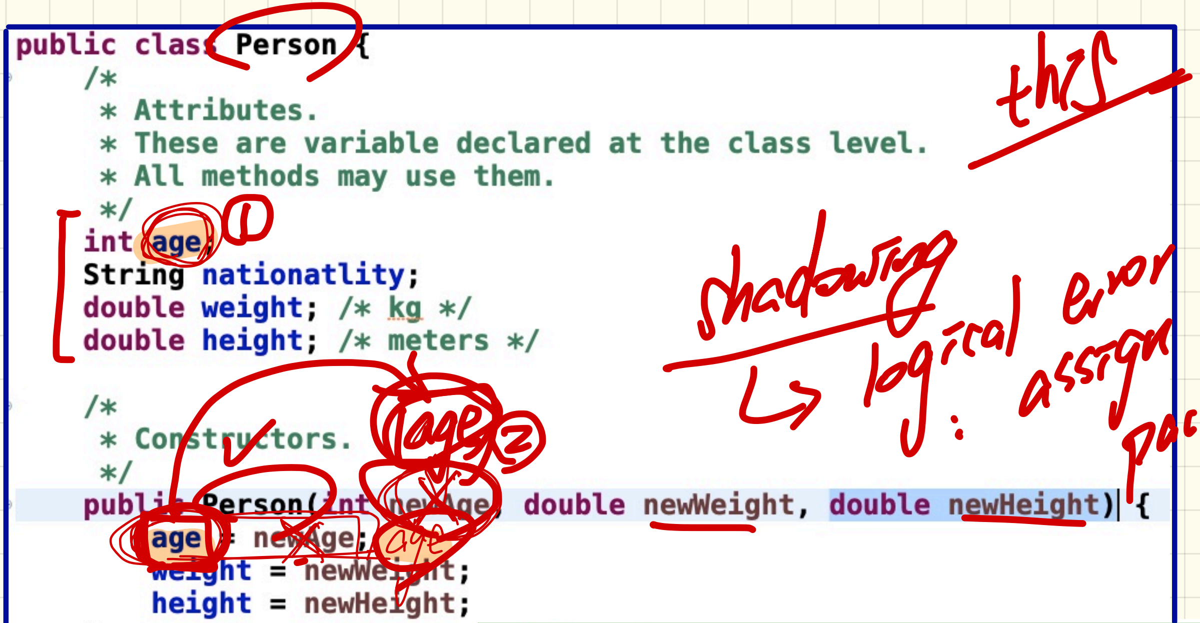
```java
public class Tester {

    public static void main(String[] args) {
        Person jim = new Person(45, 72, 1.72);
        Person jonathan = new Person(62, 65, 1.81);
    }

}
```

# LECTURE 2

MONDAY SEPTEMBER 9

- Waiting list?
- Lab 0
- Office Hours : 4 - 6 MON TUE WED
- Java Tutorial Series

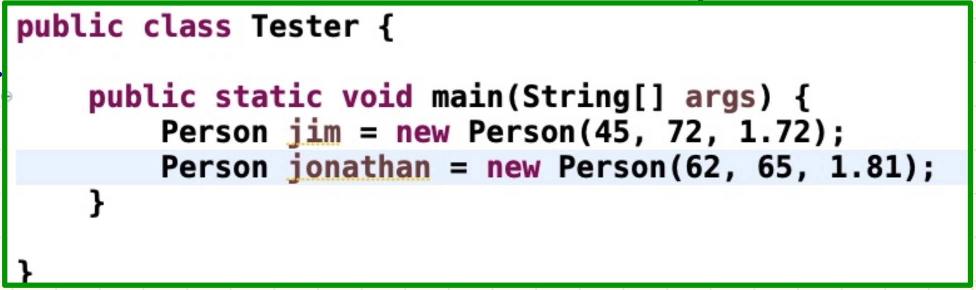
# Constructors not using this Keyword



```
public class Person {
    /*
     * Attributes.
     * These are variable declared at the class level.
     * All methods may use them.
     */
    int age;
    String nationatlity;
    double weight; /* kg */
    double height; /* meters */

    /*
     * Constructors.
     */
    public Person(int newAge, double newWeight, double newHeight) {
        age = newAge;
        weight = newWeight;
        height = newHeight;
    }
}
```

~~this~~

shadowing
↳ logical error: assign input param to itself.



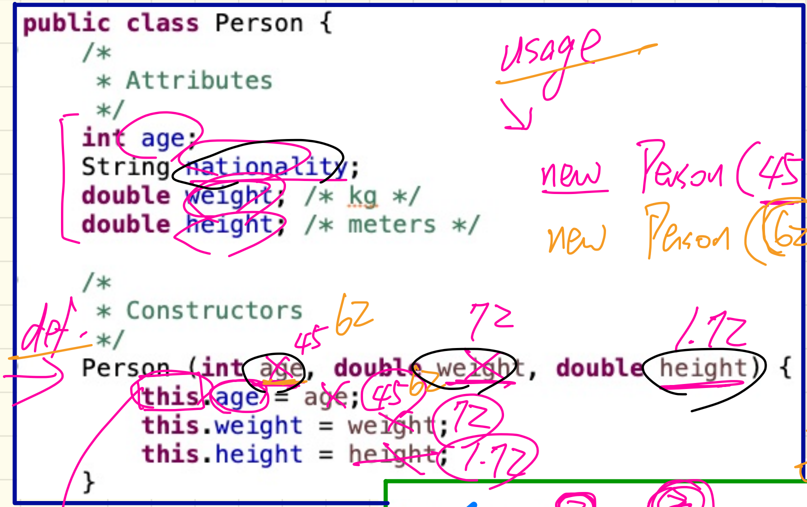
```
public class Tester {

    public static void main(String[] args) {
        Person jim = new Person(45, 72, 1.72);
        Person jonathan = new Person(62, 65, 1.81);
    }
}
```

1. Same parameter & attribute names?
2. implicit "this"

# Effect of Creating a New Object

(SEQUENCE OF BYTES) MEMORY

MODEL



```
public class Person {
    /*
     * Attributes
     */
    int age;
    String nationality;
    double weight; /* kg */
    double height; /* meters */

    /*
     * Constructors
     */
    Person (int age, double weight, double height) {
        this.age = age;
        this.weight = weight;
        this.height = height;
    }
```

usage

new Person(45, 72, 1.72) 0x11c

new Person(62, 65, 1.81)

jim

jonathan

Person

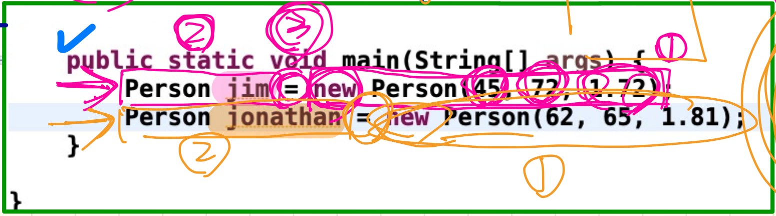
0x2034

def

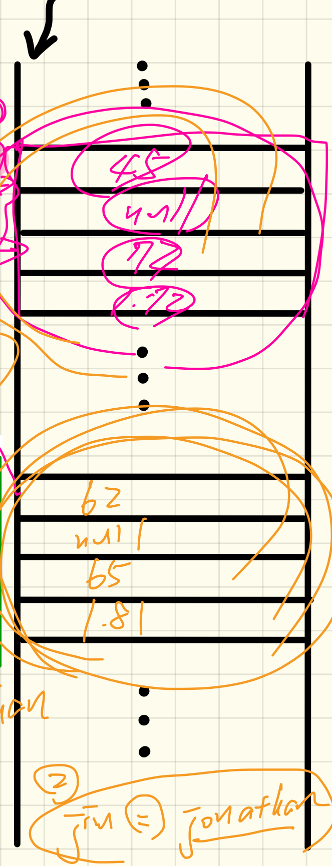
45 62 72 1.72

context object

TESTER



```
public static void main(String[] args) {
    Person jim = new Person(45, 72, 1.72);
    Person jonathan = new Person(62, 65, 1.81);
}
}
```

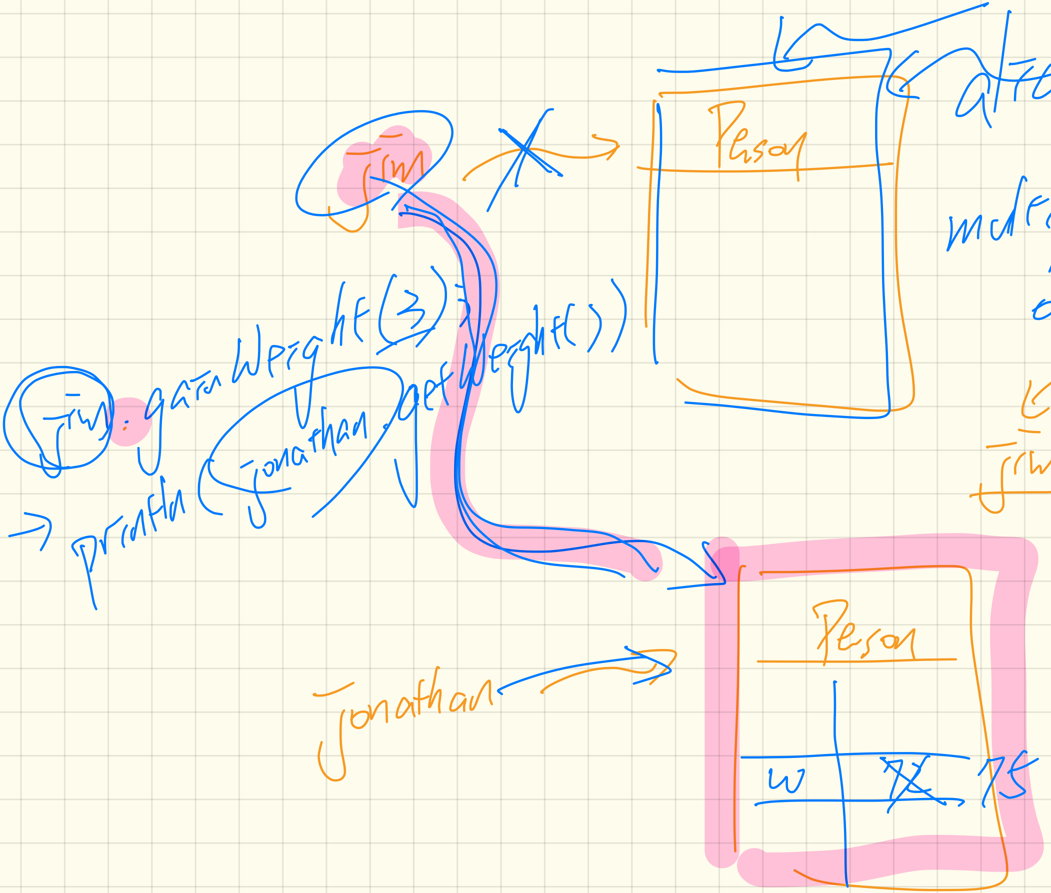


address jim 0x11c

address jonathan 0x2034

jim == jonathan F

jim = jonathan



jim.getWeight(3);

→ println(jonathan.getWeight())

aliasing: multiple copies of the same address.

jim = jonathan

```
Person alan = new Person(62);
Person mark = new Person(45);
Person tom = new Person(67);
```

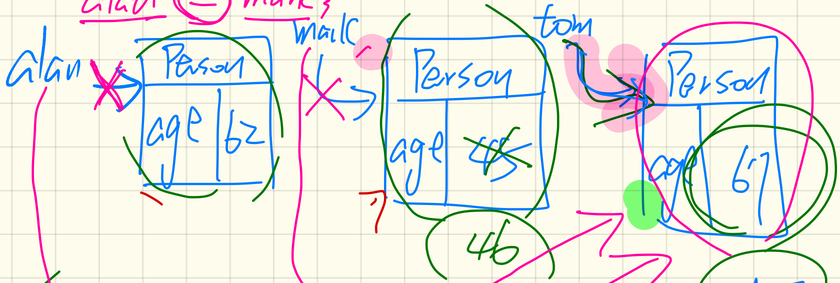
age

```
class Person{
    int age;
}
```

```
alan = mark;
mark = tom;
alan.getolder();
println(alan.age)
```

46

```
mark = tom;
alan = mark;
```



```
println(mark.age)    println(tom.age)
```

```
→ Person alan = new Person(45);

→ Person tom = new Person(62);
```

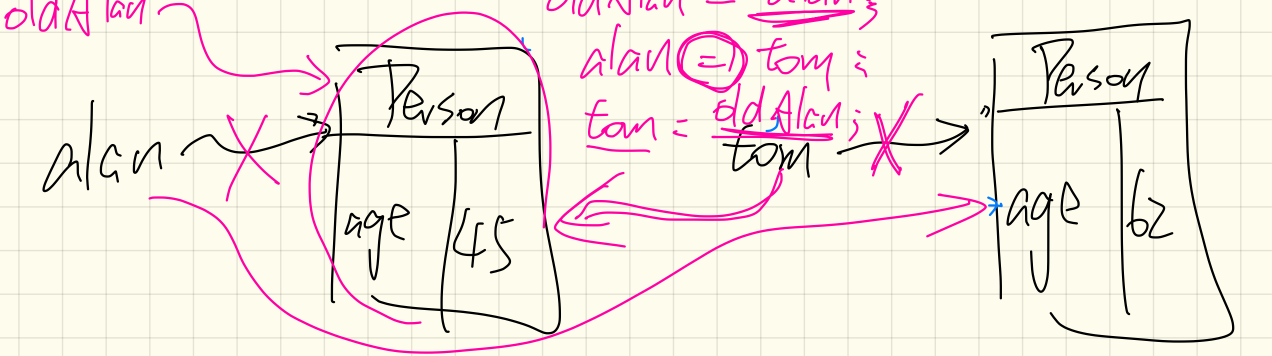
```
Peson oldAlan;
```

Swap alan and tom.

②
```
tom = alan
alan = tom
```

①
```
alan = tom
tom = alan
```

```
oldAlan = alan;
alan = tom;
tom = oldAlan;
```

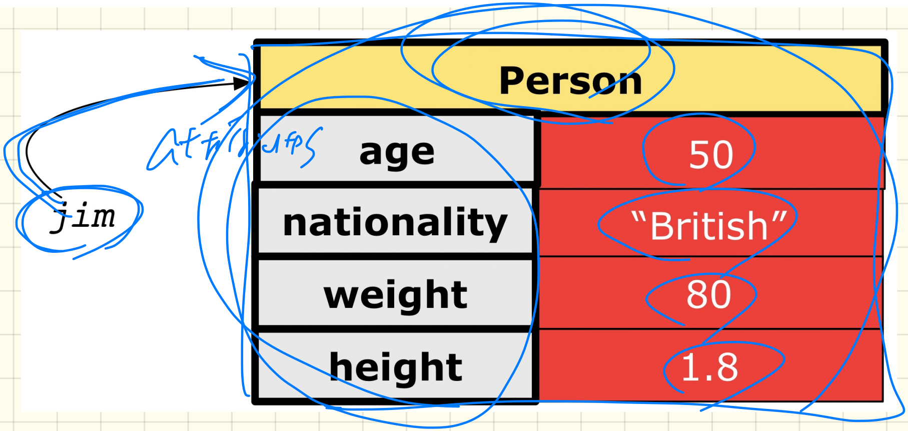


```
int i;

        Person  alan = new ...
        Person  mark = new ..
        Person  tom  = new ..


            alan = mark;
            alan = tom;
```

# Tracing OO Code: Visualizing Objects

To visualize an object:

- ○ Draw a rectangle box to represent *contents* of that object:
  - Title indicates the *name of class* from which the object is instantiated.
  - Left column enumerates *names of attributes* of the instantiated class.
  - Right column fills in *values* of the corresponding attributes.
- ○ Draw arrow(s) for *variable(s)* that store the object's *address*.



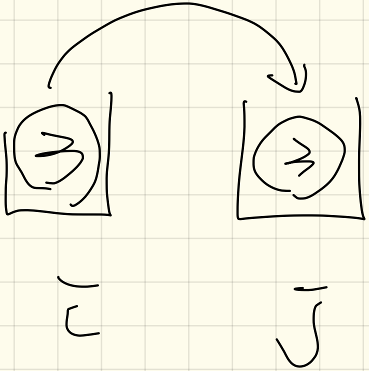
jim

attributes

| Person | |
|---|---|
| age | 50 |
| nationality | "British" |
| weight | 80 |
| height | 1.8 |

## short circuit -

```
int i = 3;
int j = i;
```

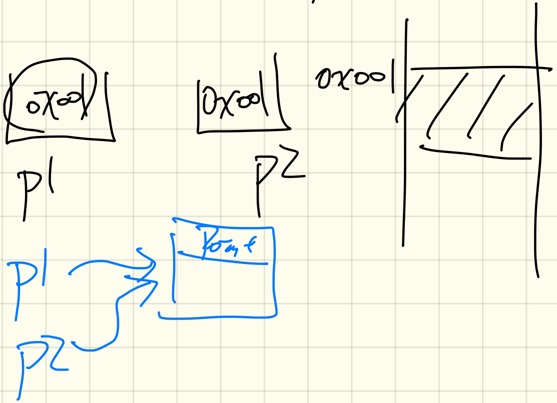


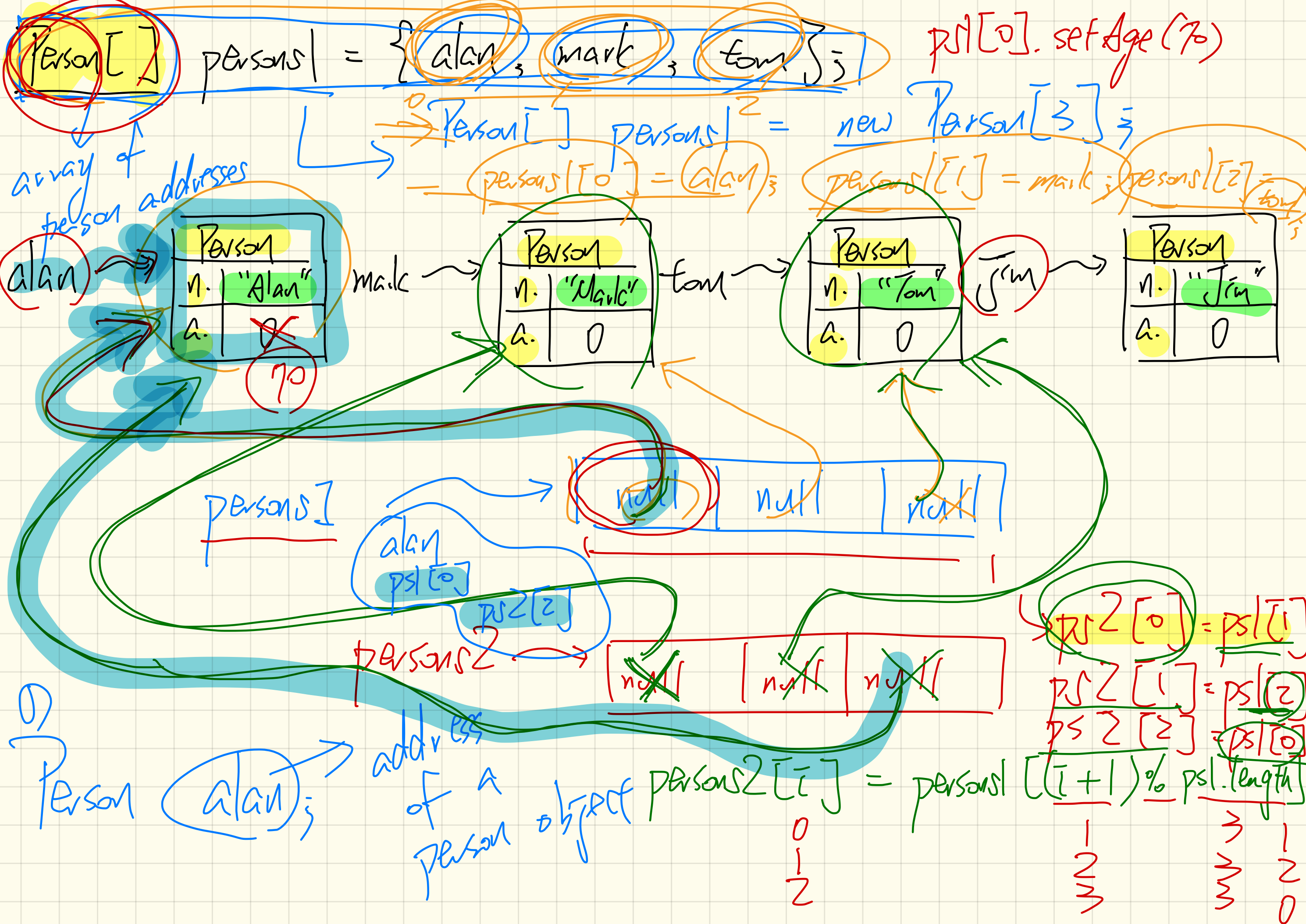
```
Point p1 = new Point(. . . —);
```

①

```
Point p2 (=) p1;
```

②





```
Person[] persons1 = {alan, mark, tom};
ps1[0].setAge(70)
Person[] persons2 = new Person[3];
persons1[0] = alan; persons1[1] = mark; persons1[2] = tom;
ps2[0] = ps1[1]
ps2[1] = ps1[2]
ps2[2] = ps1[0]
Person alan;
persons2[i] = persons1[(i+1) % ps1.length]
```

```
Person[] persons = new Person[3];
persons = { alan, mark, tom };
```

only usable in initialization,

not re-assignment.

Persons[] ps = - - -

× ps[0] = "Jackie";

String

# LECTURE 3

WEDNESDAY SEPTEMBER 11

- Notes on a Programming Pattern
  (Point, PointCollector, PointTester)
- Java Tutorial Series

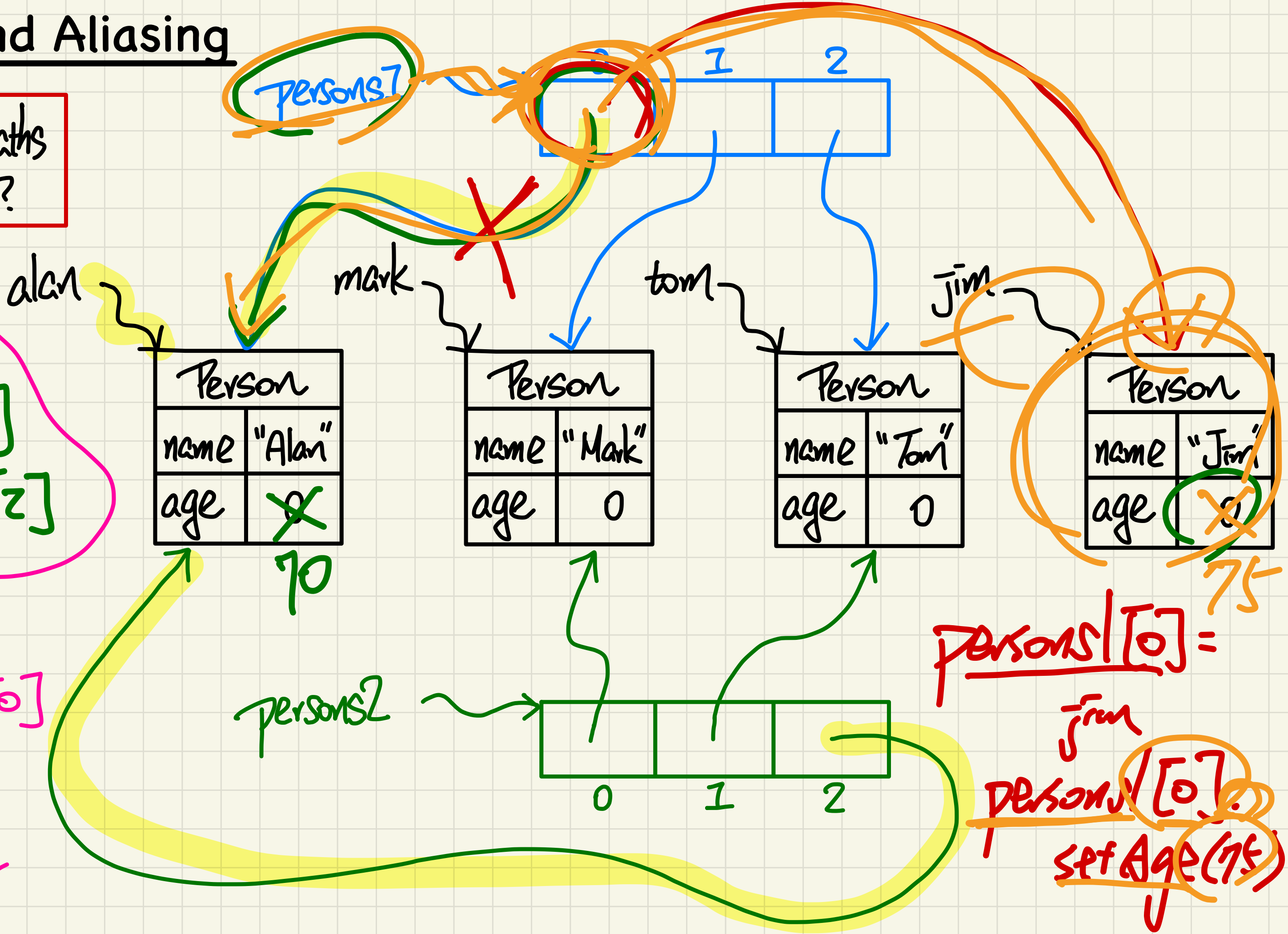
# Arrays and Aliasing

All alias paths
to "Alan" ?

alan
persons1[0]
persons2[2]

alan ==
persons1[0]

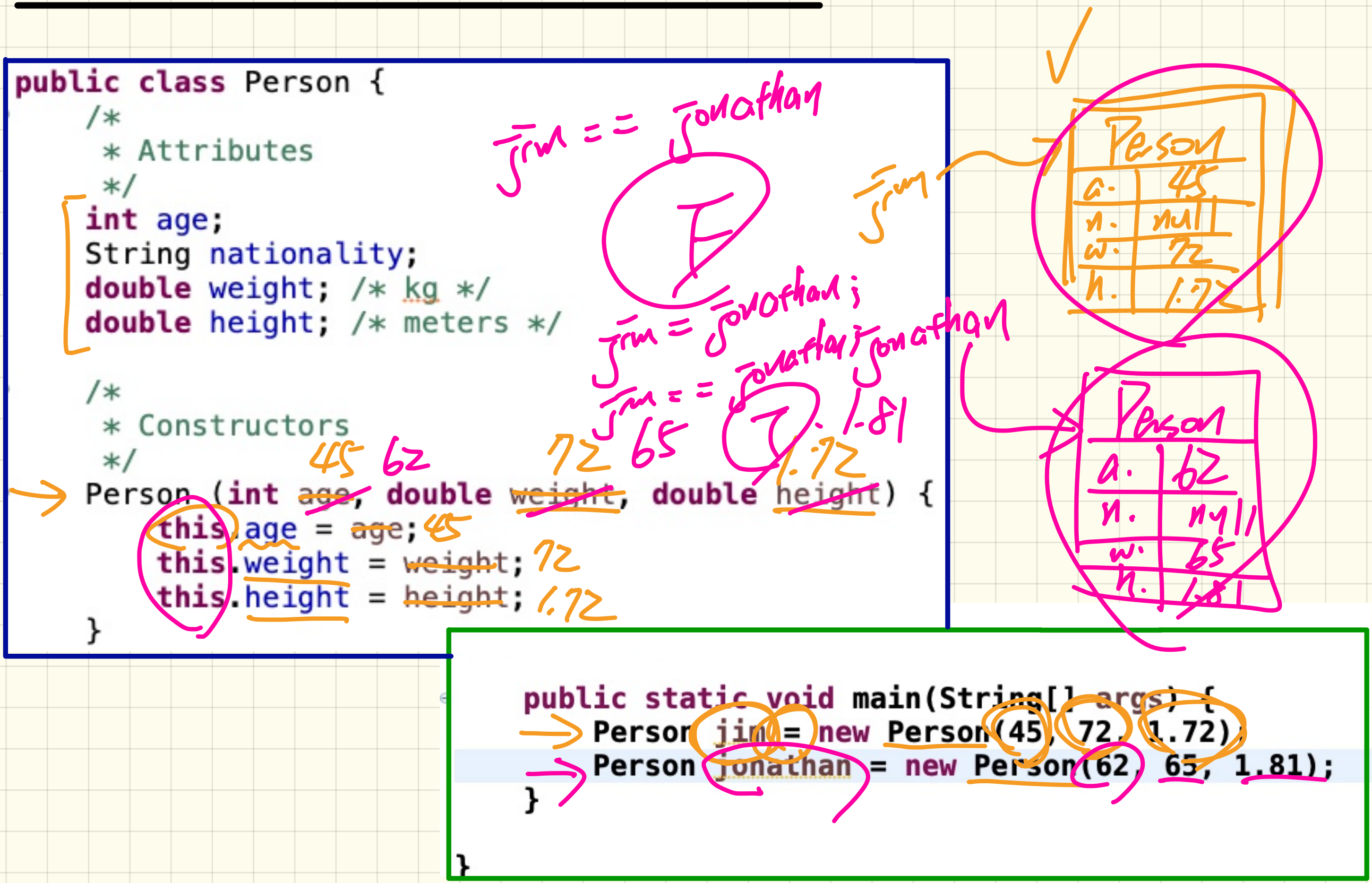
T



persons1[0] =
jim

persons1[0].
setAge(75)

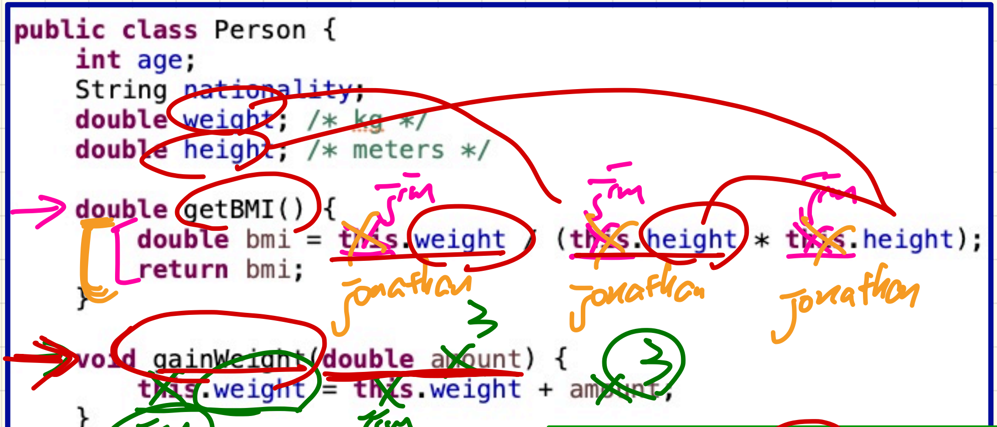
# Constructors using this Keyword



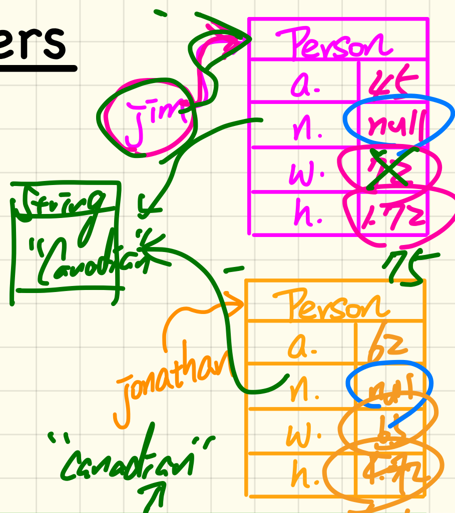
```
public class Person {
    /*
     * Attributes
     */
    int age;
    String nationality;
    double weight; /* kg */
    double height; /* meters */

    /*
     * Constructors
     */
    Person (int age, double weight, double height) {
        this.age = age;
        this.weight = weight;
        this.height = height;
    }
```

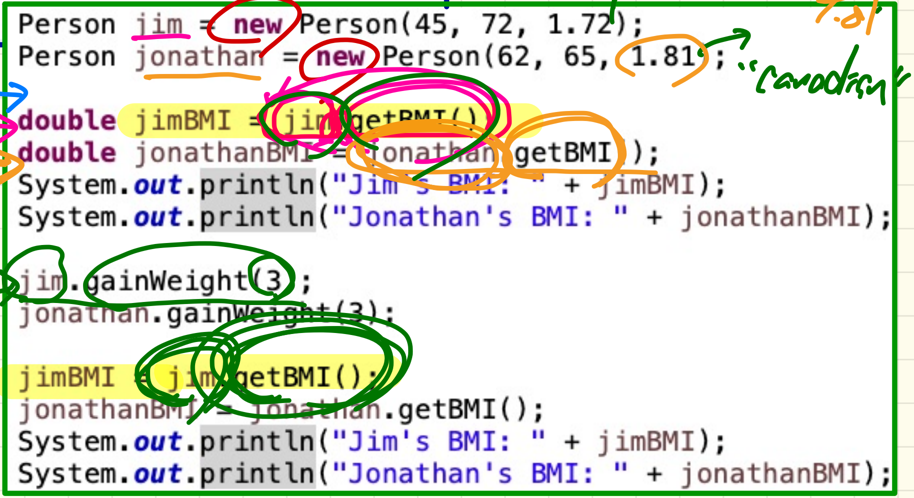
```
    public static void main(String[] args) {
        Person jim = new Person(45, 72, 1.72);
        Person jonathan = new Person(62, 65, 1.81);
    }

}
```

# Accessors/Getters vs. Mutators/Setters



```
public class Person {
    int age;
    String nationality;
    double weight; /* kg */
    double height; /* meters */

    double getBMI() {
        double bmi = this.weight / (this.height * this.height);
        return bmi;
    }

    void gainWeight(double amount) {
        this.weight = this.weight + amount;
    }
}
```





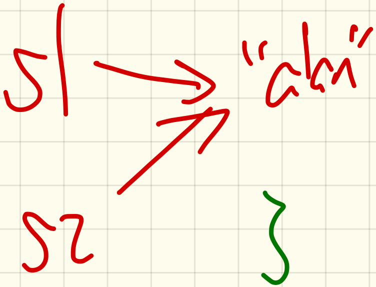
```
Person jim = new Person(45, 72, 1.72);
Person jonathan = new Person(62, 65, 1.81);

double jimBMI = jim.getBMI();
double jonathanBMI = jonathan.getBMI();
System.out.println("Jim's BMI: " + jimBMI);
System.out.println("Jonathan's BMI: " + jonathanBMI);

jim.gainWeight(3);
jonathan.gainWeight(3);

jimBMI = jim.getBMI();
jonathanBMI = jonathan.getBMI();
System.out.println("Jim's BMI: " + jimBMI);
System.out.println("Jonathan's BMI: " + jonathanBMI);
```

jim == jonathan F

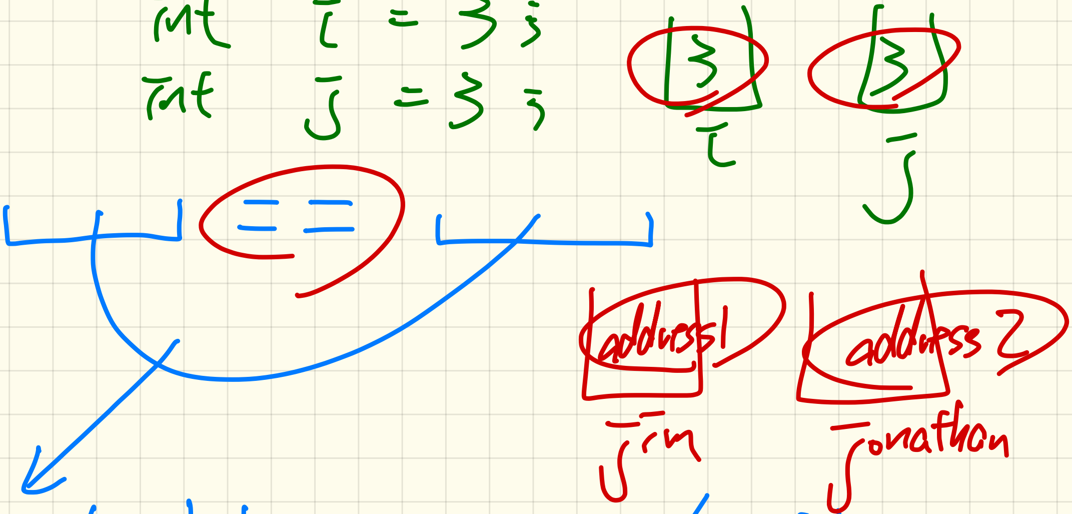
jim.age == jonathan.age T

jim.nationality == jonathan.nat. T

```
main( . . -  -){                double d1 = 2.0;
                                double d2 = 2.0;
      String s1 = "alan";       d1 == d2

      String s2 = "alan";

      s1 == s2     T ,

}
```



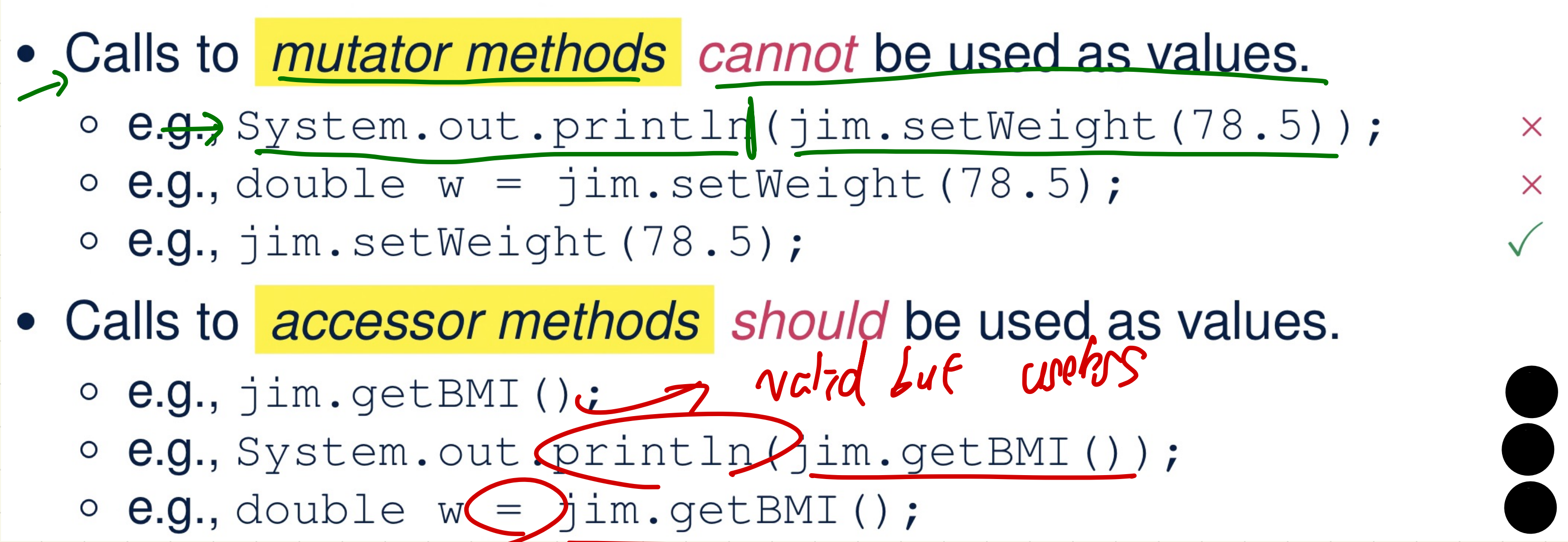
```
int i = 3;
int j = 3;
```



1. both are primitives (int)
   ↳ compare values

2. both are references (jim, jonathan)
   ↳ compare addresses

# OOP: Use of Accessors vs Use of Mutators

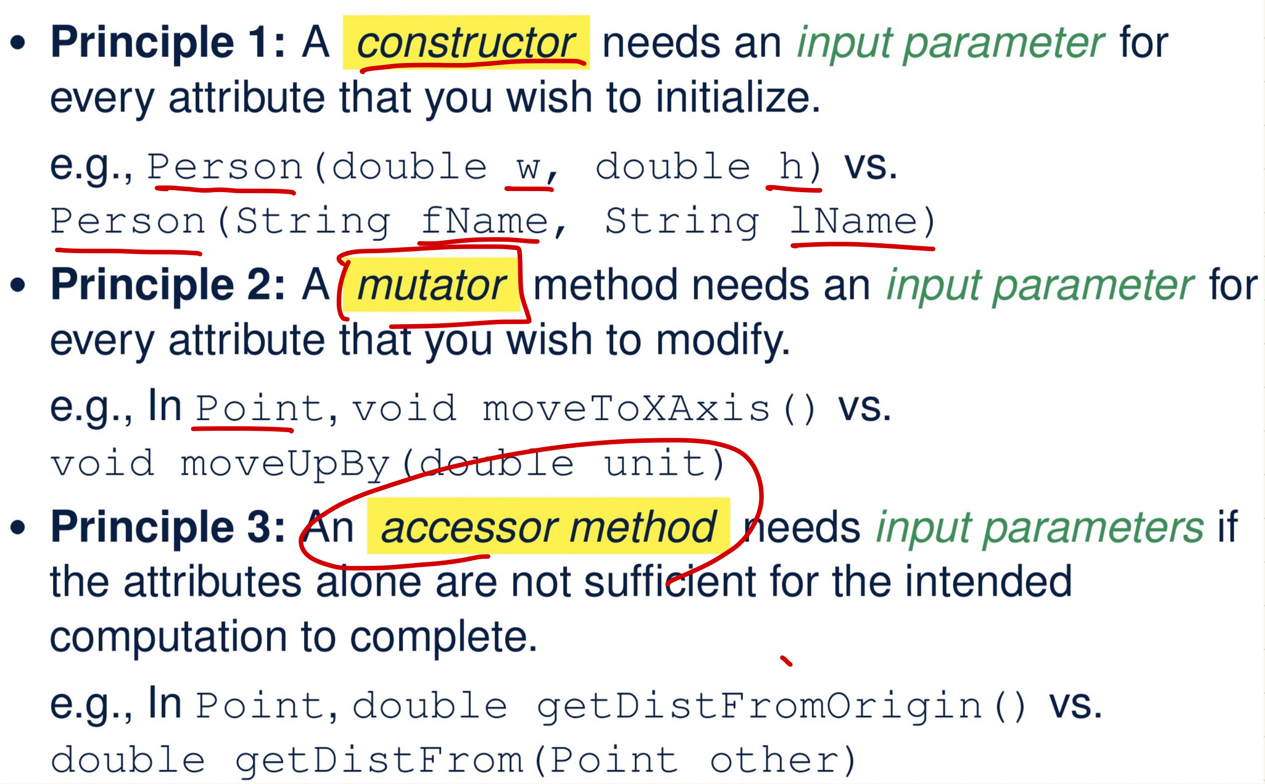


- Calls to *mutator methods* *cannot* be used as values.
  - e.g., `System.out.println(jim.setWeight(78.5));` ×
  - e.g., `double w = jim.setWeight(78.5);` ×
  - e.g., `jim.setWeight(78.5);` ✓
- Calls to *accessor methods* *should* be used as values.
  - e.g., `jim.getBMI();` [redacted]
  - e.g., `System.out.println(jim.getBMI());` [redacted]
  - e.g., `double w = jim.getBMI();` [redacted]

valid but useless

```
void   setWeight(...) {...}
double getBMI()  {...}
```

# OOP: Choice of Method Parameters



- **Principle 1:** A *constructor* needs an *input parameter* for every attribute that you wish to initialize.

  e.g., `Person(double w, double h)` vs. `Person(String fName, String lName)`

- **Principle 2:** A *mutator* method needs an *input parameter* for every attribute that you wish to modify.

  e.g., In `Point`, `void moveToXAxis()` vs. `void moveUpBy(double unit)`

- **Principle 3:** An *accessor method* needs *input parameters* if the attributes alone are not sufficient for the intended computation to complete.

  e.g., In `Point`, `double getDistFromOrigin()` vs. `double getDistFrom(Point other)`

# Return Type: Reference Type

```
class Point {
  Point(double x, double y) {...}

  void moveUpBy(double units) {
    this.y = this.y + units;
  }

  Point movedUpBy(double units) {
    Point np = new Point(this.x, this.y);
    np.moveUpBy(units);
    return np;
  }
}
```

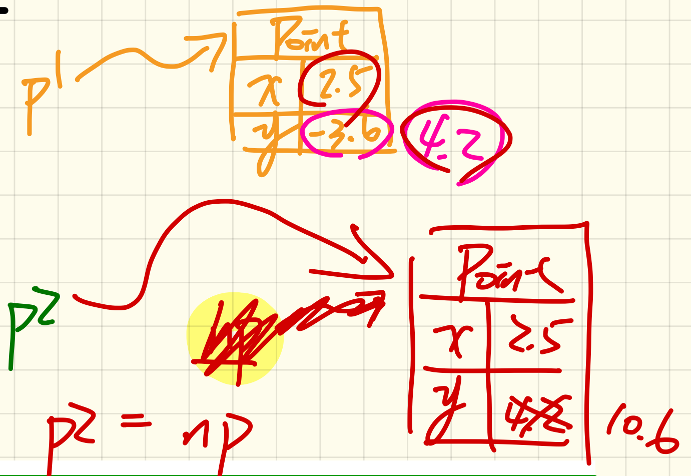
2.5 -3.6

7.8

p1 p1. 7.8

6.4

p1 p1 6.4

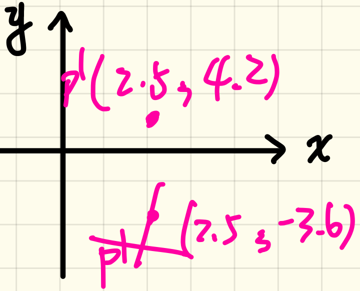


```
class PointTester {
  static void main(String[] args) {
    Point p1 = new Point(2.5, -3.6);
    p1.moveUp(7.8);
    Point p2 = p1.movedUpBy(6.4);
    System.out.println(p1 == p2);
  }
}
```

np

F

Point



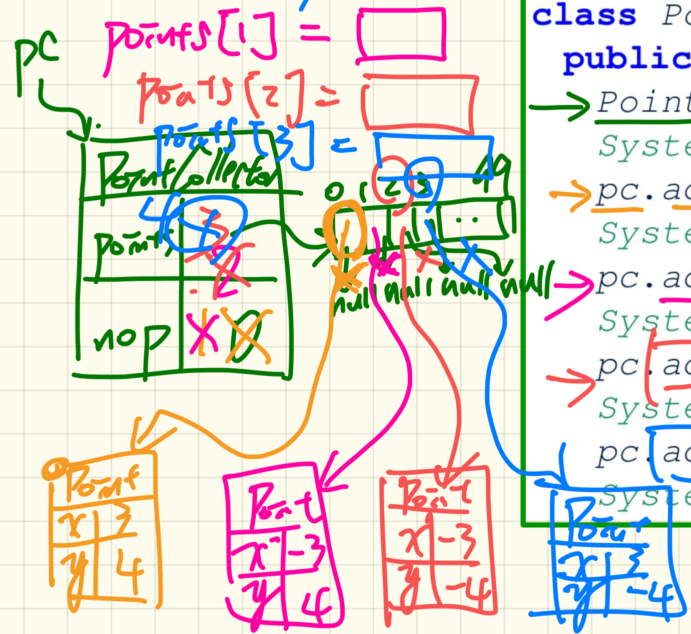
# Programming Pattern: Mutator

array of Point addresses.

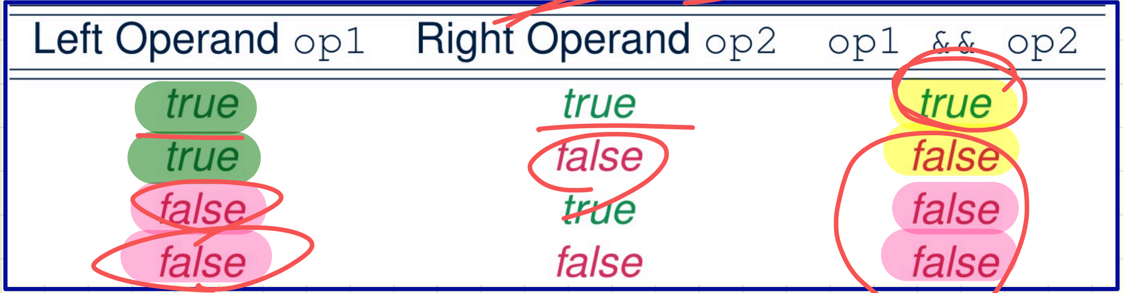
nop: 1. how points have been stored
2. where to store next point

```
class PointCollector {
  Point[] points; int nop; /* number of points */
  PointCollector() { this.points = new Point[100]; }
  void addPoint(double x, double y) {
    points[nop] = new Point(x, y); nop++; }
```

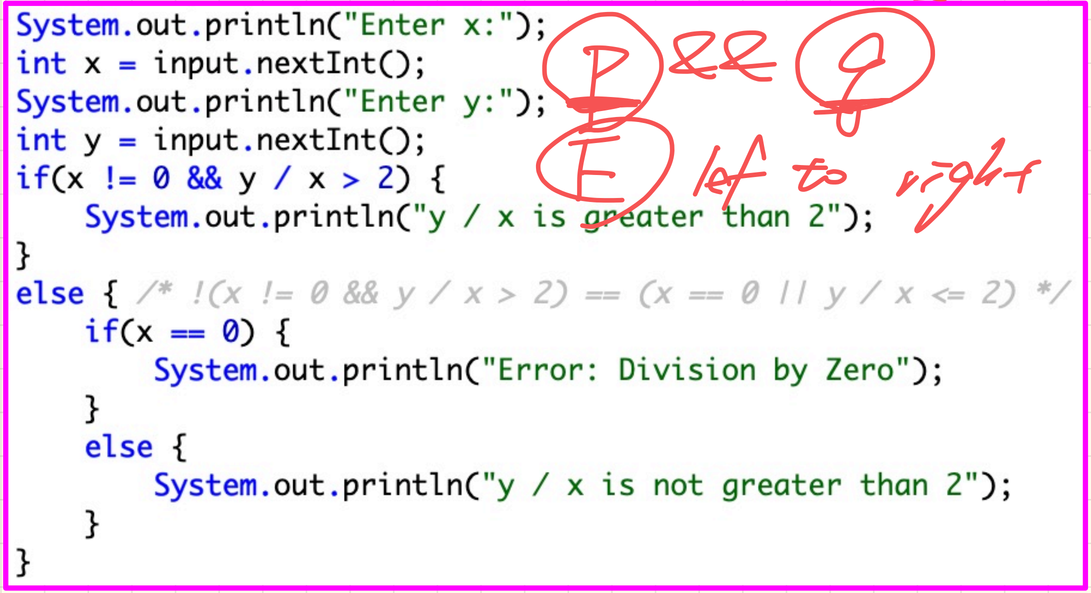
```
class PointCollectorTester {
  public static void main(String[] args) {
    PointCollector pc = new PointCollector();
    System.out.println(pc.nop);    /* 0 */
    pc.addPoint(3, 4);
    System.out.println(pc.nop);    /* 1 */
    pc.addPoint(-3, 4);
    System.out.println(pc.nop);    /* 2 */
    pc.addPoint(-3, -4);
    System.out.println(pc.nop);    /* 3 */
    pc.addPoint(3, -4);
    System.out.println(pc.nop);    /* 4 */
```



# Short-Circuit Evaluation: &&



| Left Operand `op1` | Right Operand `op2` | `op1 && op2` |
|---|---|---|
| true | true | true |
| true | false | false |
| false | true | false |
| false | false | false |



```
System.out.println("Enter x:");
int x = input.nextInt();
System.out.println("Enter y:");
int y = input.nextInt();
if(x != 0 && y / x > 2) {
    System.out.println("y / x is greater than 2");
}
else { /* !(x != 0 && y / x > 2) == (x == 0 || y / x <= 2) */
    if(x == 0) {
        System.out.println("Error: Division by Zero");
    }
    else {
        System.out.println("y / x is not greater than 2");
    }
}
```

P && Q

E lef to right

Test Case:

$x = 0$

$y = 10$

Test Case:

$x = 5$

$y = 10$

# Short-Circuit Evaluation: ||

| Left Operand op1 | Right Operand op2 | op1 \|\| op2 |
|---|---|---|
| *false* | *false* | *false* |
| *true* | *false* | *true* |
| *false* | *true* | *true* |
| *true* | *true* | *true* |

guard

```java
System.out.println("Enter x:");
int x = input.nextInt();
System.out.println("Enter y:");
int y = input.nextInt();
if(x == 0 || y / x > 2) {
    if(x == 0) {
        System.out.println("Error: Division by Zero");
    }
    else {
        System.out.println("y / x is greater than 2");
    }
}
else { /* !(x == 0 || y / x > 2) == (x != 0 && y / x <= 2) */
    System.out.println("y / x is not greater than 2");
}
```

P1 || P2

T

L → R

x

Test Case :

x = 0

y = 10

Test Case :

x = 5

y = 10

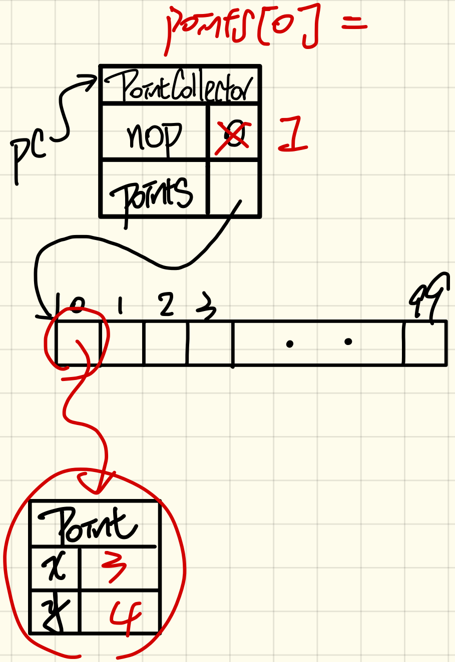
# LECTURE 4

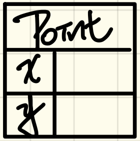
MONDAY SEPTEMBER 16

# Programming Pattern: Mutator

```
class PointCollector {
  Point[] points; int nop; /* number of points */
  PointCollector() { points = new Point[100]; }
  void addPoint(double x, double y) {
   points[nop] = new Point(x, y); nop++; }
```

```
class PointCollectorTester {
 public static void main(String[] args) {
  PointCollector pc = new PointCollector();
  System.out.println(pc.nop);   /* 0 */
  pc.addPoint(3, 4);
  System.out.println(pc.nop);   /* 1 */
  pc.addPoint(-3, 4);
  System.out.println(pc.nop);   /* 2 */
  pc.addPoint(-3, -4);
  System.out.println(pc.nop);   /* 3 */
  pc.addPoint(3, -4);
  System.out.println(pc.nop);   /* 4 */
```





nop:

1. number of Point objects stored

2. index to store the next Point object

# Programming Pattern: Accessor

i < points.length

```
Point[] getPointsInQuadrantI() {
  Point[] ps = new Point[nop];
  int count = 0; /* number of points in Quadrant I */
  for(int i = 0; i < nop; i ++) {
    Point p = points[i];
    if(p.x > 0 && p.y > 0) { ps[count] = p; count ++; } }
  Point[] q1Points = new Point[count];
  /* ps contains null if count < nop */
  for(int i = 0; i < count; i ++) { q1Points[i] = ps[i] }
  return q1Points;
} }
```

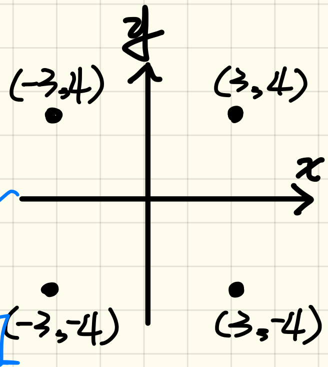
ps[0] = p; count ++;

p = points[1];

return ps;

q1Points

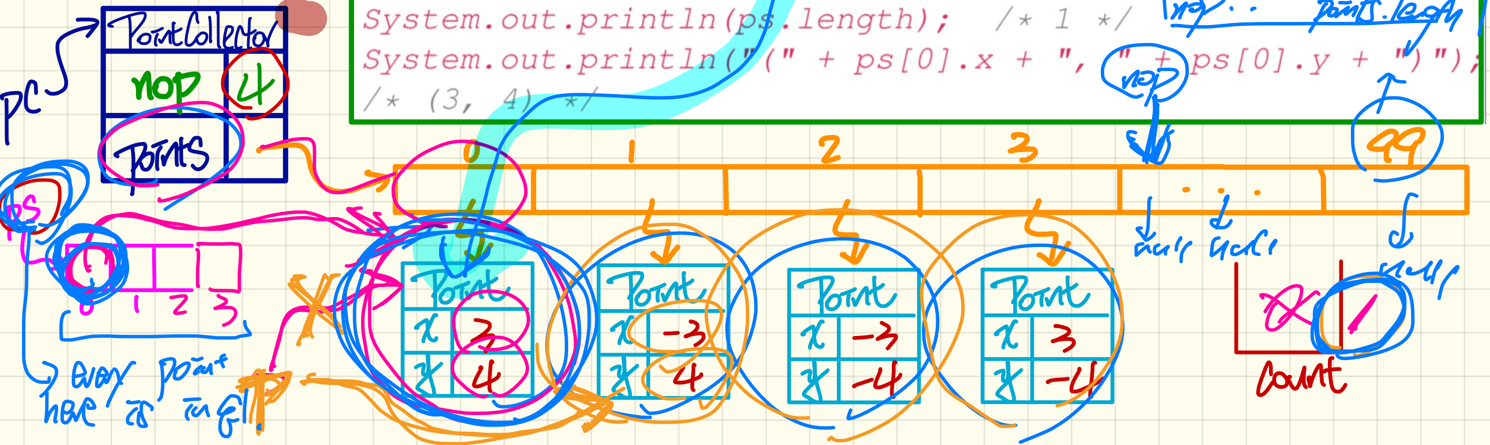
q1Points[0] = ps[0]



```
Point[] ps = pc.getPointsInQuadrantI();
System.out.println(ps.length); /* 1 */
System.out.println("(" + ps[0].x + ", " + ps[0].y + ")");
/* (3, 4) */
```

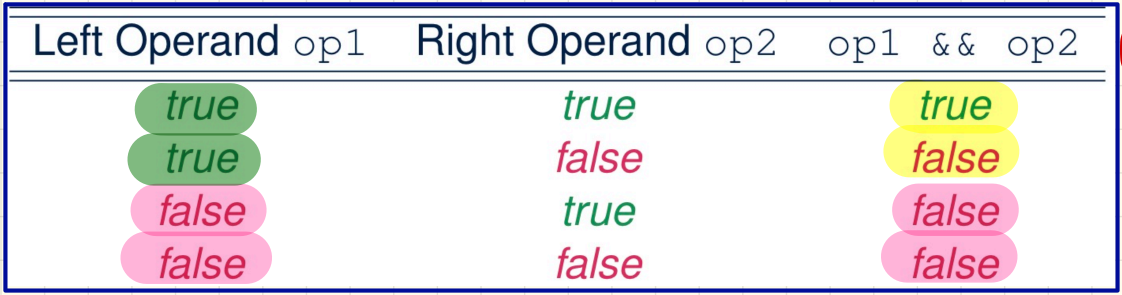
nop .. points.length

nop



# Short-Circuit Evaluation: &&

→ L → R



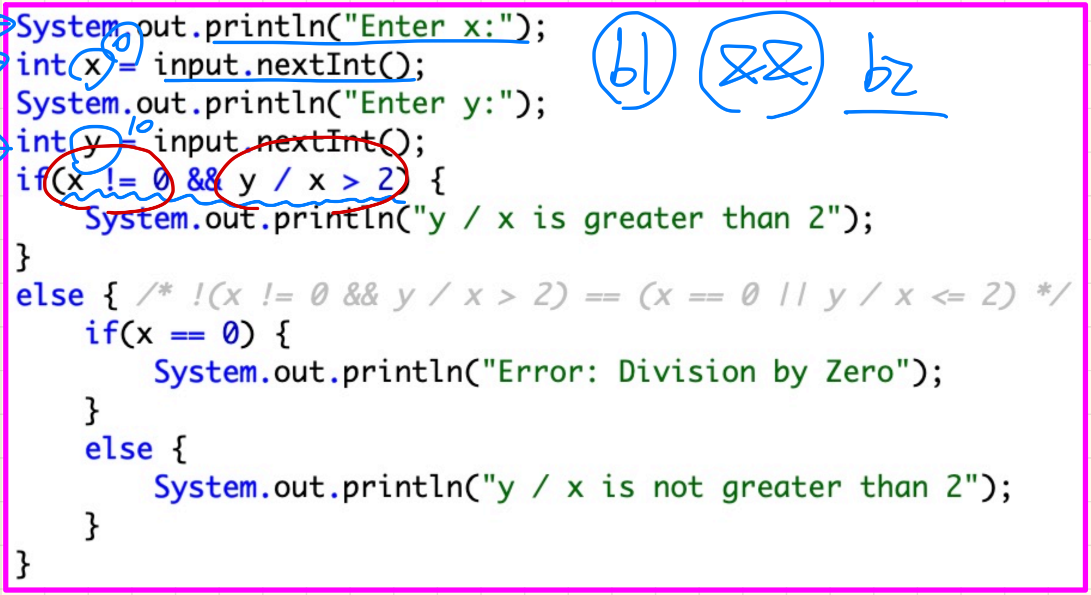
| Left Operand op1 | Right Operand op2 | op1 && op2 |
|---|---|---|
| true | true | true |
| true | false | false |
| false | true | false |
| false | false | false |

F

0 10 0

$0 != 0$ && $y / x > 2$

X



```
System.out.println("Enter x:");
int x = input.nextInt();
System.out.println("Enter y:");
int y = input.nextInt();
if(x != 0 && y / x > 2) {
    System.out.println("y / x is greater than 2");
}
else { /* !(x != 0 && y / x > 2) == (x == 0 || y / x <= 2) */
    if(x == 0) {
        System.out.println("Error: Division by Zero");
    }
    else {
        System.out.println("y / x is not greater than 2");
    }
}
```

0

10

b1 && b2

Test Case:

→ x = 0

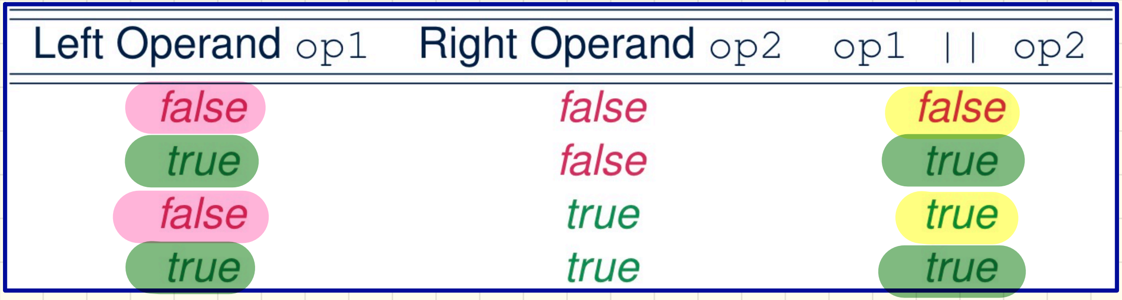
→ y = 10

Test Case:

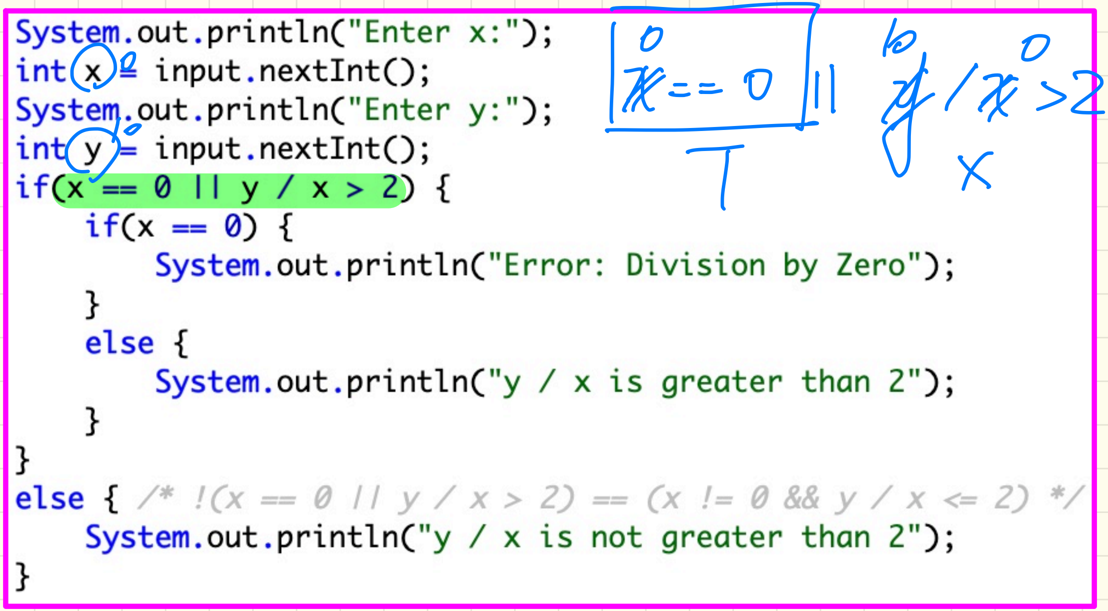
x = 5

y = 10

# Short-Circuit Evaluation: ||



| Left Operand op1 | Right Operand op2 | op1 \|\| op2 |
|---|---|---|
| false | false | false |
| true | false | true |
| false | true | true |
| true | true | true |



```
System.out.println("Enter x:");
int x = input.nextInt();
System.out.println("Enter y:");
int y = input.nextInt();
if(x == 0 || y / x > 2) {
    if(x == 0) {
        System.out.println("Error: Division by Zero");
    }
    else {
        System.out.println("y / x is greater than 2");
    }
}
else { /* !(x == 0 || y / x > 2) == (x != 0 && y / x <= 2) */
    System.out.println("y / x is not greater than 2");
}
```

0 10

$\overset{0}{x} == 0 \;||\; \overset{10}{y} / \overset{0}{x} > 2$

T X

Test Case:

$x = 0$

$y = 10$

Test Case:

$x = 5$

$y = 10$

# Short-Circuit Evaluation: Common Errors

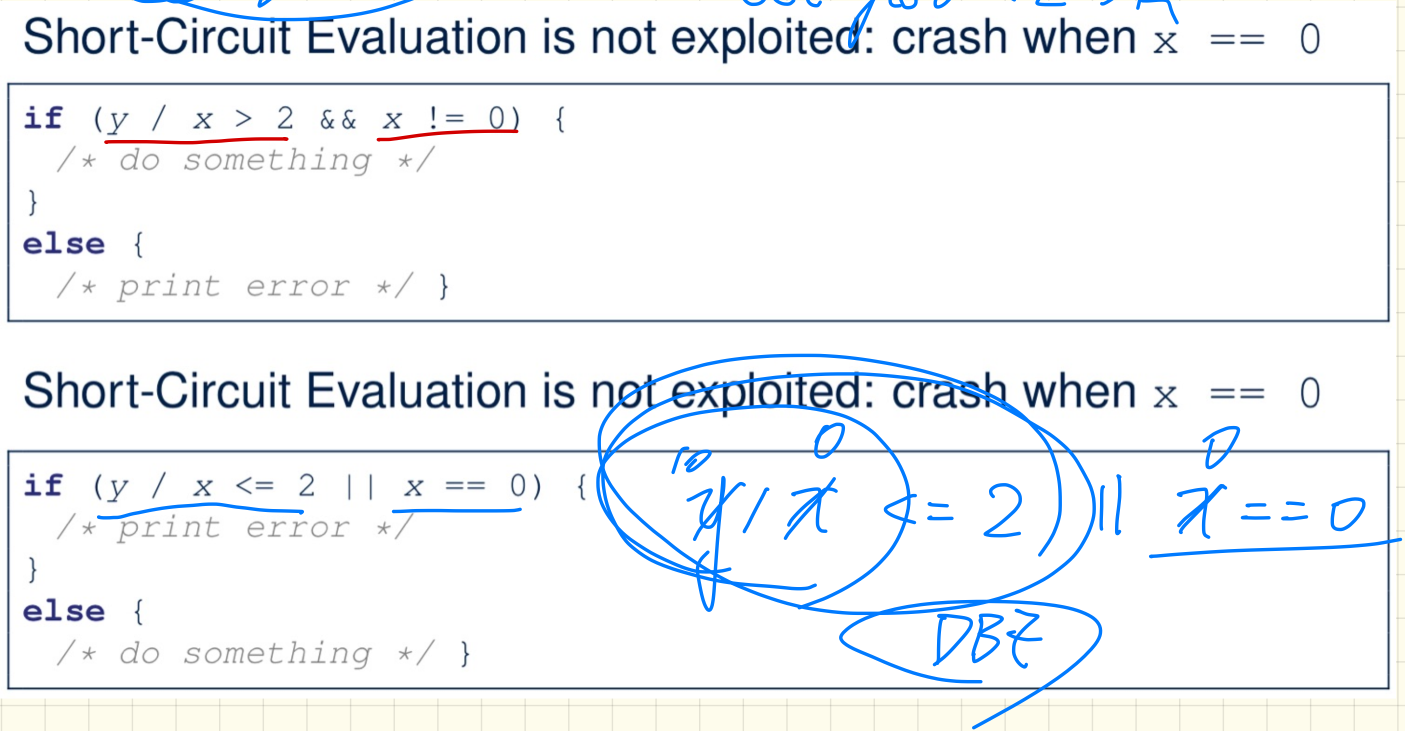
$\overset{10}{y} / \overset{0}{x} > 2$ && $\overset{0}{x} != 0$

Test Case :

$x = 0$

$y = 10$

SCE goes from L -> R



Short-Circuit Evaluation is not exploited: crash when `x == 0`

```
if (y / x > 2 && x != 0) {
  /* do something */
}
else {
  /* print error */ }
```

Short-Circuit Evaluation is not exploited: crash when `x == 0`

```
if (y / x <= 2 || x == 0) {
  /* print error */
}
else {
  /* do something */ }
```

$\overset{10}{y} / \overset{0}{x} <= 2$ || $\overset{0}{x} == 0$

DBZ

# Anonymous Objects

−3

```
1  double square(double x) {
2    double sqr = x * x;
3    return sqr; }
```

sqr

```
1  double square(double x) {
2    return x * x; }
```

```
1  Person getP(String n) {
2    Person p = new Person(n);
3    return p; }
```

```
1  Person getP(String n) {
2    return new Person(n); }
```

between line 2 and return line 3, p is never used.

```
class Member {
  Order[] orders;
  int noo;
  /* constructor ommitted */
  void addOrder(Order o) {
    orders[noo] = o;
    noo ++;
  }
  void addOrder(String n, double p, double q) {
    addOrder(new Order(n, p, q));
    /* Equivalent implementation:
     * orders[noo] = new Order(n, p, q);
     noo ++; */
  }
}
```

addPut

"Americano"

"2.3"

4

```
class MemberTester {
  public static void main(String[] args) {
    Member m = new Member("Alan");
    Order o = new Order("Americano", 4.7, 3);
    m.addOrder(o);
    m.addOrder( new Order("Cafe Latte", 5.1, 4) );
  }
}
```

# Overloadding

```
int divide(int x, int y)
double divide(double x, double y)
```

Tester:

```
Member m = ...
m.addOrder("A", 2.3, 4);
m.addOrder(
  new Order("A", 2.3, 4));
```

```
void addOrder(Order o) { ... }
void addOrder(String n, double p, int q)
```

compilation error ✓

→ void addOrder(String n, double p, int q)

→ void ✓ addOrder(double p, int q, String n)

X void addOrder(String name, double price, int quan)

```
m. addOrder("A", 2.3, 4);
m. addOrder(2.3, 4, "A");
```

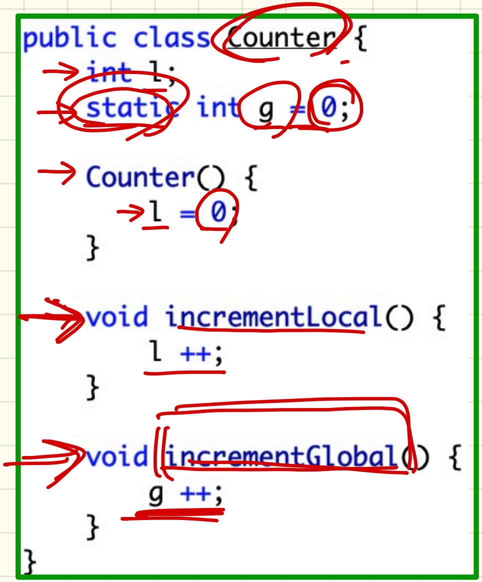
usage

# Overloading

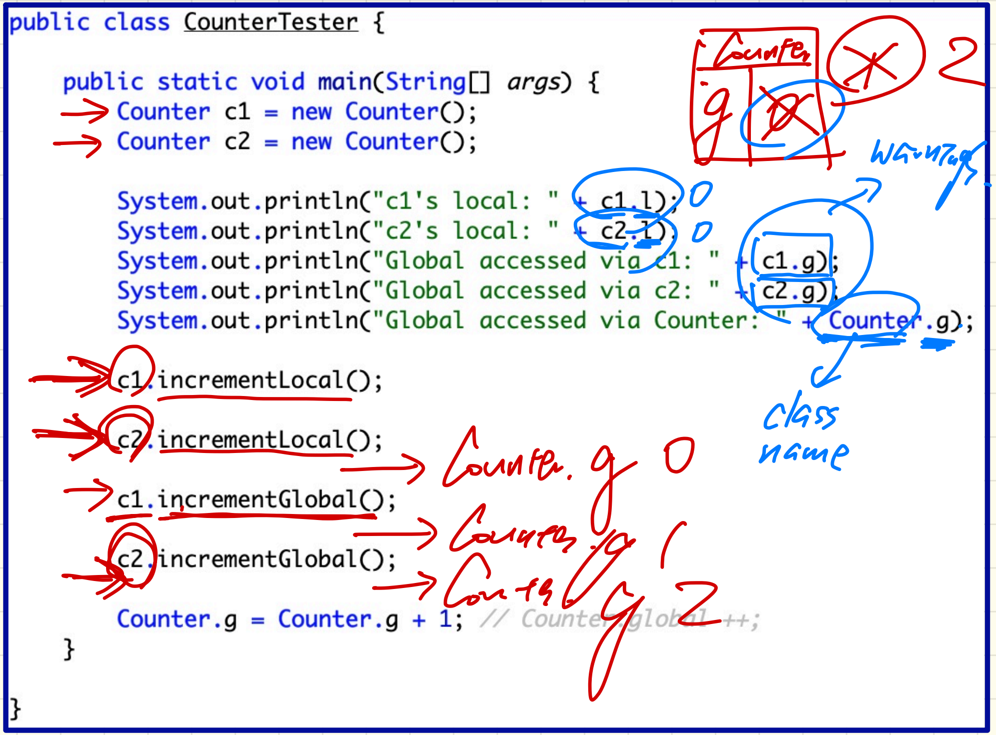
Methods with the same name:

1. different # of parameters
2. same # of parameters, types must be different.

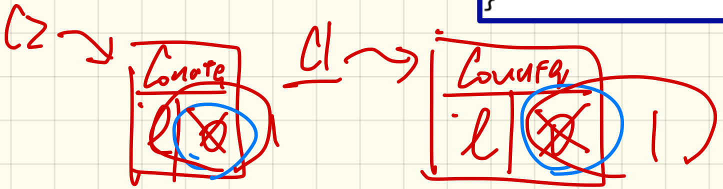
# Static vs Non-Static Variables



```
public class Counter {
  int l;
  static int g = 0;

  Counter() {
    l = 0;
  }

  void incrementLocal() {
    l ++;
  }

  void incrementGlobal() {
    g ++;
  }
}
```

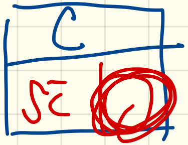


```
public class CounterTester {

    public static void main(String[] args) {
        Counter c1 = new Counter();
        Counter c2 = new Counter();

        System.out.println("c1's local: " + c1.l);
        System.out.println("c2's local: " + c2.l);
        System.out.println("Global accessed via c1: " + c1.g);
        System.out.println("Global accessed via c2: " + c2.g);
        System.out.println("Global accessed via Counter: " + Counter.g);

        c1.incrementLocal();

        c2.incrementLocal();

        c1.incrementGlobal();

        c2.incrementGlobal();

        Counter.g = Counter.g + 1; // Counter.global ++;
    }

}
```



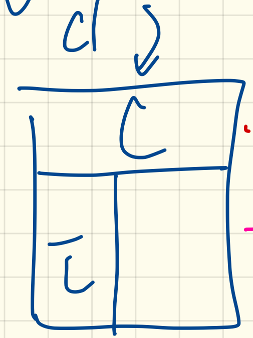
```
class C {
  static int sc;
  int i;
}
```

static variables



↳ 1. belongs to the class

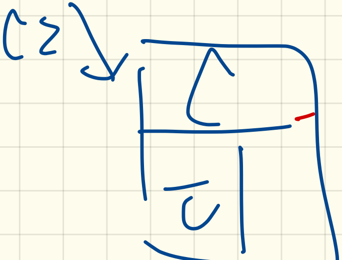
↳ 2. global to all objects



→ non-static variables (attributes)

↳ 1. belongs to an instance

2. local



# LECTURE 5

WEDNESDAY SEPTEMBER 17

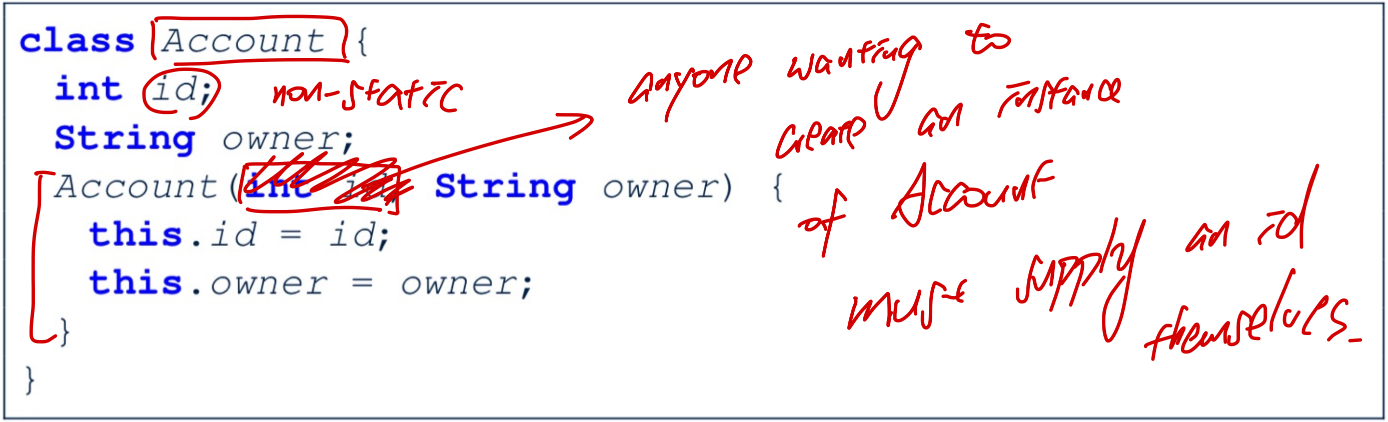
Tue Oct 1 Mini Test # 1 (10%)
(written)

Tue Oct 8 Lab Test # 1 (20%)
(programming)

static int i ] global to all instances

int j ] local to each instance.

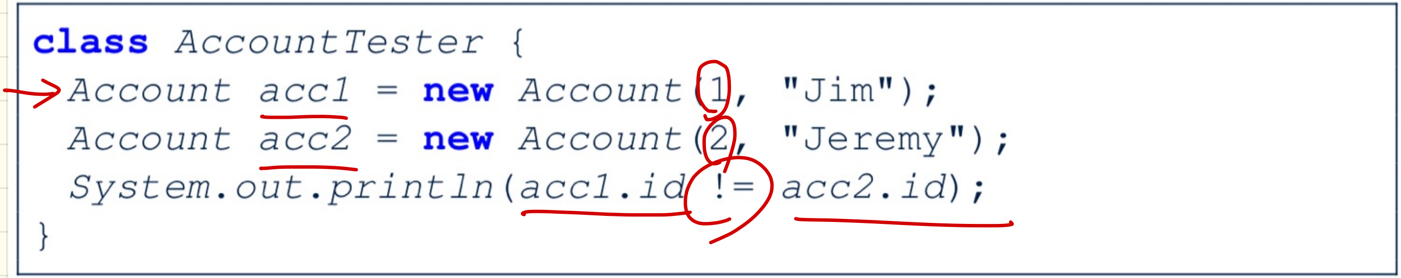
# Managing Account ID: Manually



```
class Account {
  int id;
  String owner;
  Account(~~int id~~, String owner) {
    this.id = id;
    this.owner = owner;
  }
}
```

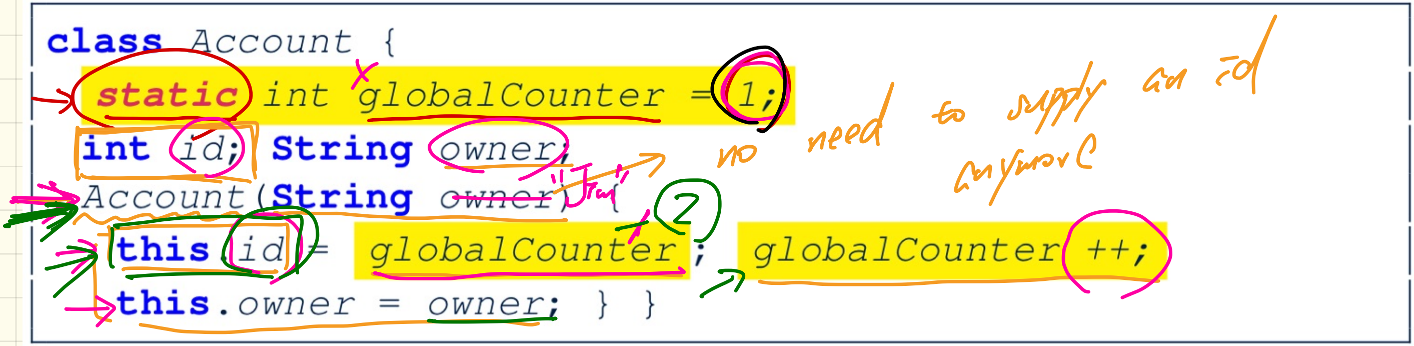
non-static

anyone wanting to create an instance of Account must supply an id themselves



```
class AccountTester {
  Account acc1 = new Account(1, "Jim");
  Account acc2 = new Account(2, "Jeremy");
  System.out.println(acc1.id != acc2.id);
}
```

# Managing Account ID: Automatically

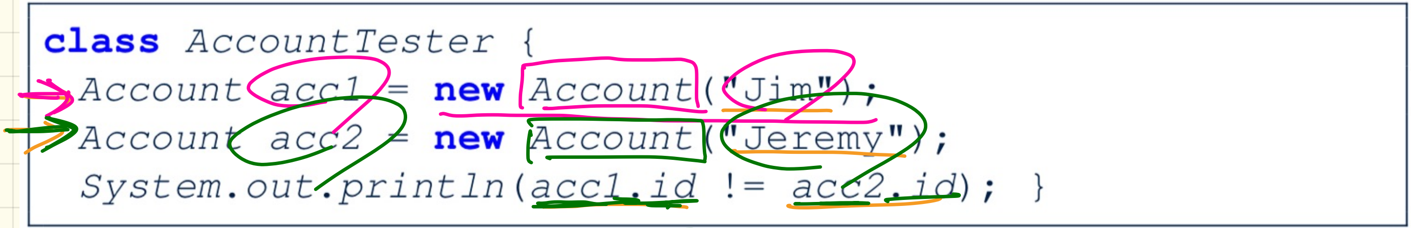


```
class Account {
  static int globalCounter = 1;
  int id; String owner;
  Account(String owner) {
    this.id = globalCounter ; globalCounter ++;
    this.owner = owner; } }
```

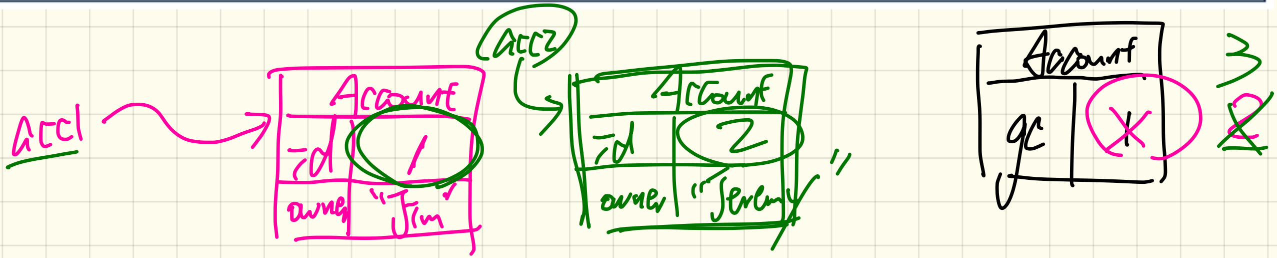
no need to supply an id anymore

"Jim"

2



```
class AccountTester {
  Account acc1 = new Account("Jim");
  Account acc2 = new Account("Jeremy");
  System.out.println(acc1.id != acc2.id); }
```



# Misuse of Static Variables

```
class Client {
  Account[] accounts;
  static int numberOfAccounts = 0;
void addAccount(Account acc) {
  accounts[numberOfAccounts] = acc;
  numberOfAccounts ++;
  } }
```

bill. accounts[0] = acc1;

acc1 acc2 ~~0~~ 1

acc2;

steve. accounts[1] =

```
class ClientTester {
Client bill = new Client("Bill");
Client steve = new Client("Steve");
Account acc1 = new Account();
Account acc2 = new Account();
bill.addAccount(acc1);
  /* correctly added to bill.accounts[0] */
steve.addAccount(acc2);
  /* mistakenly added to steve.accounts[1]! */
}
```

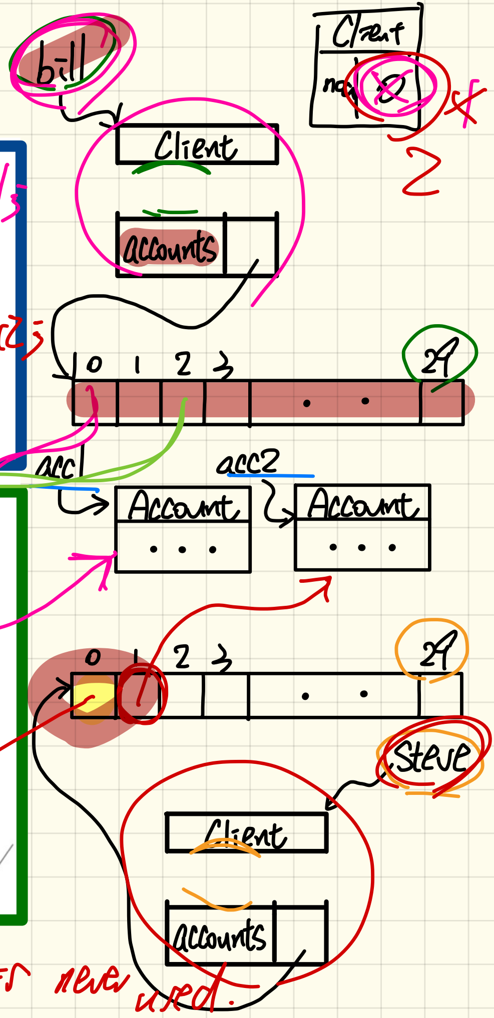
Account acc3 = new . .

bill. addAccount(acc3)

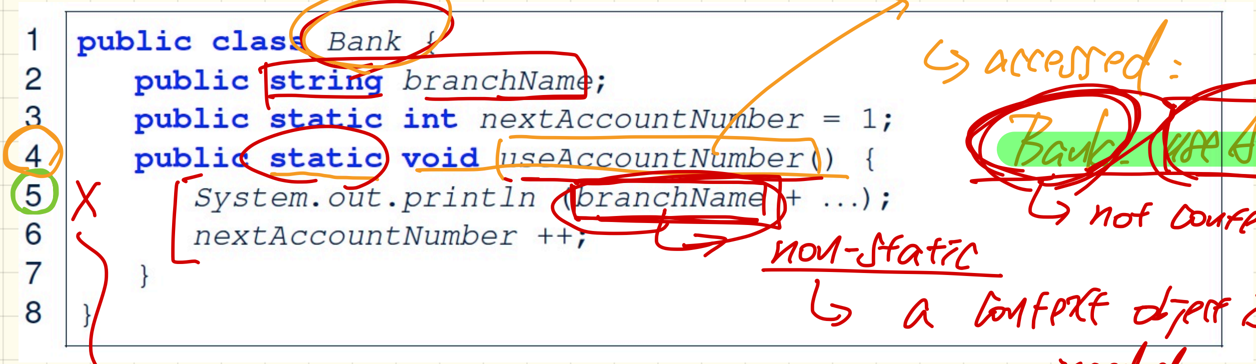
acc3 →

problem

steve. accounts[0] is never used.



# Use of Static Variables: Common Errors



```
public class Bank {
   public string branchName;
   public static int nextAccountNumber = 1;
   public static void useAccountNumber() {
      System.out.println (branchName + ...);
      nextAccountNumber ++;
   }
}
```

static method

↳ accessed :

Bank.useAccount()

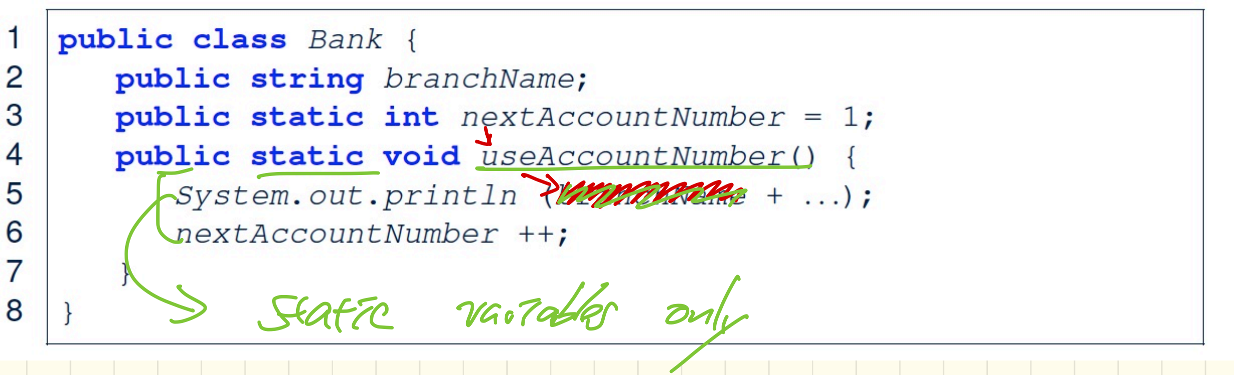
↳ not context object

X

non-static

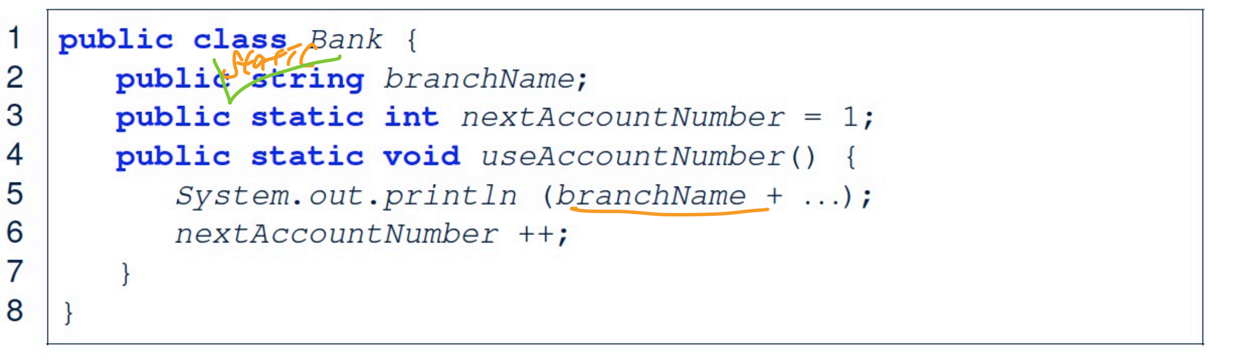
↳ a context object is needed

→ Illegal use of non-static variable branchName in a static context useAccountNumber .



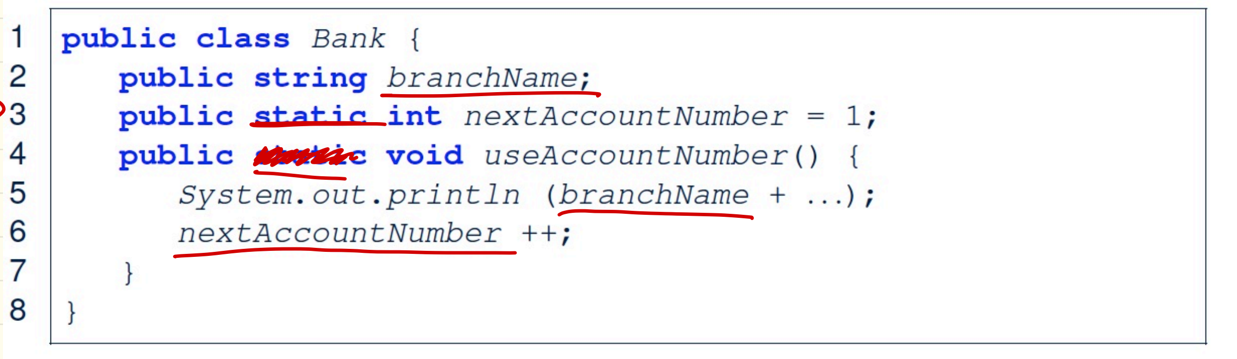
```
public class Bank {
   public string branchName;
   public static int nextAccountNumber = 1;
   public static void useAccountNumber() {
      System.out.println (~~branchName~~ + ...);
      nextAccountNumber ++;
   }
}
```

static variables only



```
public class Bank {
   public string branchName;
   public static int nextAccountNumber = 1;
   public static void useAccountNumber() {
      System.out.println (branchName + ...);
      nextAccountNumber ++;
   }
}
```

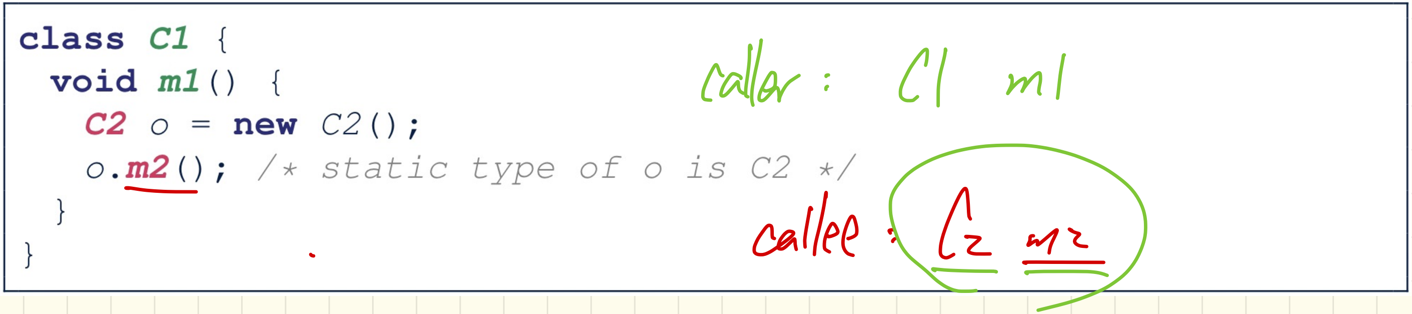
static



```
public class Bank {
   public string branchName;
   public static int nextAccountNumber = 1;
   public ~~static~~ void useAccountNumber() {
      System.out.println (branchName + ...);
      nextAccountNumber ++;
   }
}
```

# Caller vs. Callee

- **caller** is the **client** using the service provided by another method.
- **callee** is the **supplier** providing the service to another method.



```
class C1 {
  void m1() {
    C2 o = new C2();
    o.m2(); /* static type of o is C2 */
  }
}
```

caller : C1 m1

callee : C2 m2

Q: Can a method be a **caller** and a **callee** simultaneously?

C2 m2 to be caller and callee at the same time?

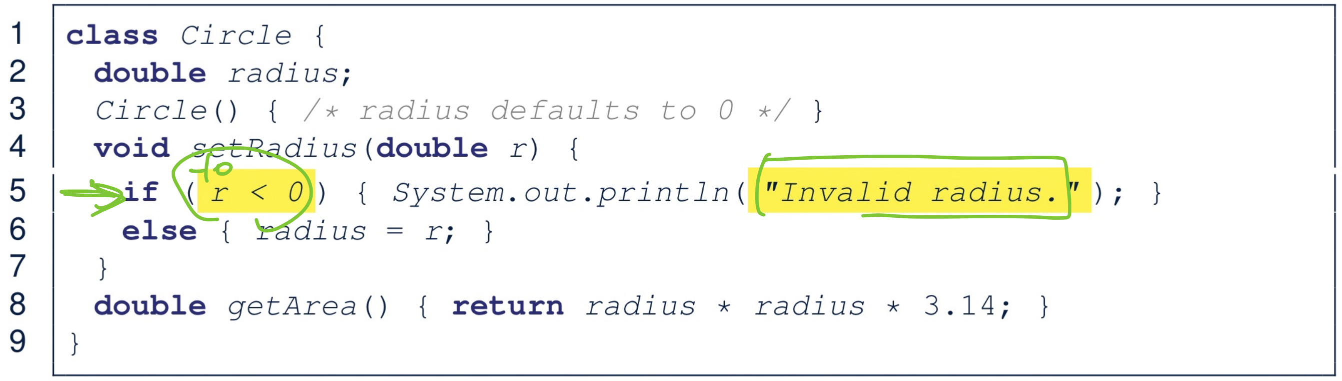
```
class C2 {
  m2(){
  }
}
```

C3 o2 = new C3();
o2. m3( );

# Error Handling via Console Messages: Circles

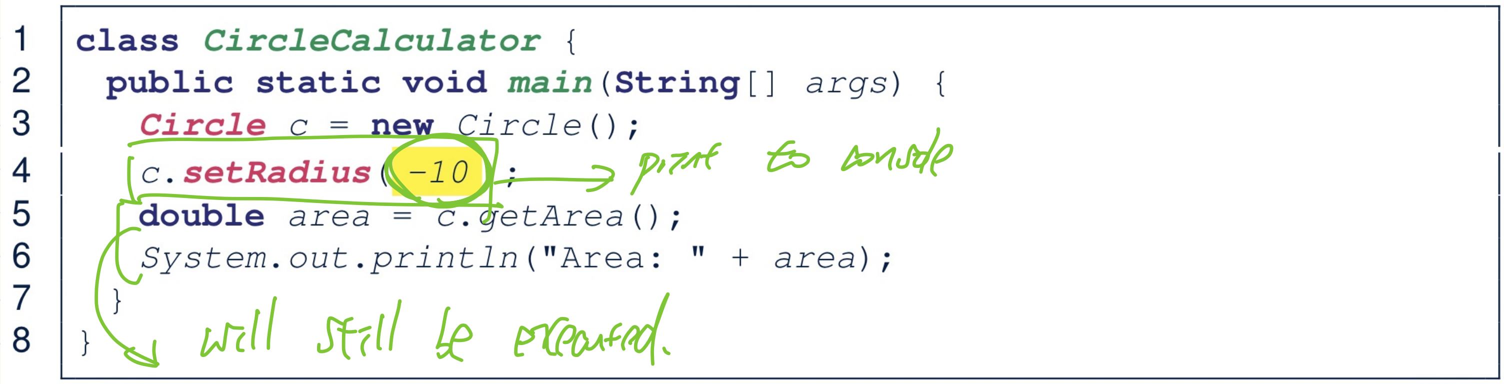
Caller?

Callee?



```java
class Circle {
  double radius;
  Circle() { /* radius defaults to 0 */ }
  void setRadius(double r) {
   if (r < 0) { System.out.println("Invalid radius."); }
   else { radius = r; }
  }
  double getArea() { return radius * radius * 3.14; }
}
```

-10



```java
class CircleCalculator {
 public static void main(String[] args) {
  Circle c = new Circle();
  c.setRadius(-10);
  double area = c.getArea();
  System.out.println("Area: " + area);
 }
}
```

print to console

will still be executed.

# Error Handling via Console Messages: Bank

```
class Account {
  int id; double balance;
  Account(int id) { this.id = id; /* balance defaults to 0 */ }
  void deposit(double a) {
    if ( a < 0 ) { System.out.println( "Invalid deposit." ); }
    else { balance += a; }
  }
  void withdraw(double a) {
    if ( a < 0 || balance - a < 0 ) {
      System.out.println( "Invalid withdraw." ); }
    else { balance -= a; }
  }
}
```

errors to console
not appropriate

```
class Bank {
  Account[] accounts; int numberOfAccounts;
  Account(int id) { ... }
  void withdrawFrom(int id, double a) {
    for(int i = 0; i < numberOfAccounts; i ++) {
      if(accounts[i].id == id) {
        accounts[i].withdraw( a );
      }
    } /
  } /*
}
```

```
class BankApplication {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    Bank b = new Bank(); Account acc1 = new Account(23);
    b.addAccount(acc1);
    double a = input.nextDouble();
    b.withdrawFrom(23, a);
}
```

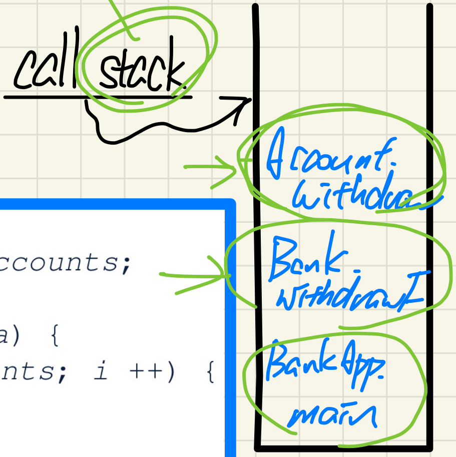
as Java Application

**Caller?**

**Callee?**

LIFO
Last In
First
Out

call stack



| context | caller | callee |
|---|---|---|
| Account | / | / |
| Bank | Bank.withdrawFrom | Account.withdraw |
| BankApp | BankApp.main | Bank.withdrawFrom |

catch-or-specify

When an exception is thrown:

throws

1. Specify: include "throws ___"
as part of the method
exception

try {
...
}
catch {
...
}

2. Catch: catch the exception that may be thrown; and do something about.

# LECTURE 6

MONDAY SEPTEMBER 23

- Guides
  - ~ Written Test
  - ~ Programming Test
- Review Session

  Tentatively: 10am ~ 12 noon

  Monday Sep. 30

# Error Handling via Console Messages: Circles

Caller?

Callee?

```
class Circle {
  double radius;
  Circle() { /* radius defaults to 0 */ }
  void setRadius(double r) {
    if ( r < 0 ) { System.out.println("Invalid radius."); }
    else { radius = r; }
  }
  double getArea() { return radius * radius * 3.14; }
}
```

-10

```
class CircleCalculator {
  public static void main(String[] args) {
    Circle c = new Circle();
    c.setRadius(-10);
    double area = c.getArea();
    System.out.println("Area: " + area);
  }
}
```

print error to console

when 'r' is invalid, these two lines should not be executed.

# Error Handling via Console Messages: Bank

```
class Account {
  int id; double balance;
  Account(int id) { this.id = id; /* balance defaults to 0 */ }
  void deposit(double a) {
    if ( a < 0 ) { System.out.println( "Invalid deposit." ); }
    else { balance += a; }
  }
  void withdraw(double a) {
    if ( a < 0 || balance - a < 0) {
      System.out.println( "Invalid withdraw." ); }
    else { balance -= a; }
  }
}
```

-10

-10

printed to console

```
class Bank {
  Account[] accounts; int numberOfAccounts;
  Account(int id) { ... }
  void withdrawFrom(int id, double a) {
    for(int i = 0; i < numberOfAccounts; i ++) {
      if(accounts[i].id == id) {
        accounts[i].withdraw( a );
      }
    }
  } /
}
```

-10

-10

```
class BankApplication {
  pubic static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    Bank b = new Bank(); Account acc1 = new Account(23);
    b.addAccount(acc1);
    double a = input.nextDouble();
    b.withdrawFrom(23, a);
    System.out.println("Transaction Completed.");
}
```

-10

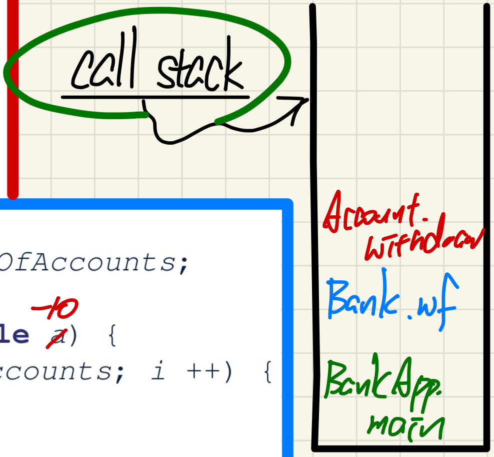
trigger console error

but this line is still executed

**Caller?**

**Callee?**

| context | caller | callee |
|---|---|---|
| Account | | |
| Bank | | |
| BankApp | | |



# Catch-or-Specify Requirement

1. **The "Catch" Solution:** A `try` statement that *catches and handles the exception*.

~~throws~~

```
main(...) {
  Circle c = new Circle();
  try {
    c.setRadius(-10);
  }
  catch(NegativeRaidusException e) {
    ...
  }
}
```

NVE

**The "Specify" Solution:** A method that specifies as part of its *signature* that it *can throw* the exception (without handling that exception).

```
class Bank {
  Account[] accounts; /* attribute */
  void withdraw (double amount)
      throws InvalidTransactionException {
    ...
    accounts[i].withdraw(amount);
    ...
  }
}
```

specify as part of the API

# Example: To Handle or Not To Handle?

| context | caller | callee |
|---|---|---|
|  |  |  |
|  |  |  |

throws NVE

throw new NVE("...");

```
class A {
  ma(int i) {
    if(i < 0) { /* Error */ }
    else { /* Do something. */ }
  } }
```

V2: handle in Tester

```
class B {
  mb(int i) {
    A oa = new A();
    oa.ma(i); /* Error occurs if i < 0 */
  } }
```

caller: B.mb

callee: A.ma

V1: handle exception here

```
class Tester {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    int i = input.nextInt();
    B ob = new B();
    ob.mb(i); /* Where can the error be handled? */
  } }
```

V2: handle exception here.

**Version 1:**
Handle it in `B.mb`

**Version 2:**
Pass it from `B.mb` and handle it in `Tester.main`

**Version 3:**
Pass it from `B.mb`, then from `Tester.main`, then throw it to the console.

do not handle ~~ftp~~ exception anywhere

```
class NegValException extends Exception {
  NegValException(String s) { super(s); }
}
```

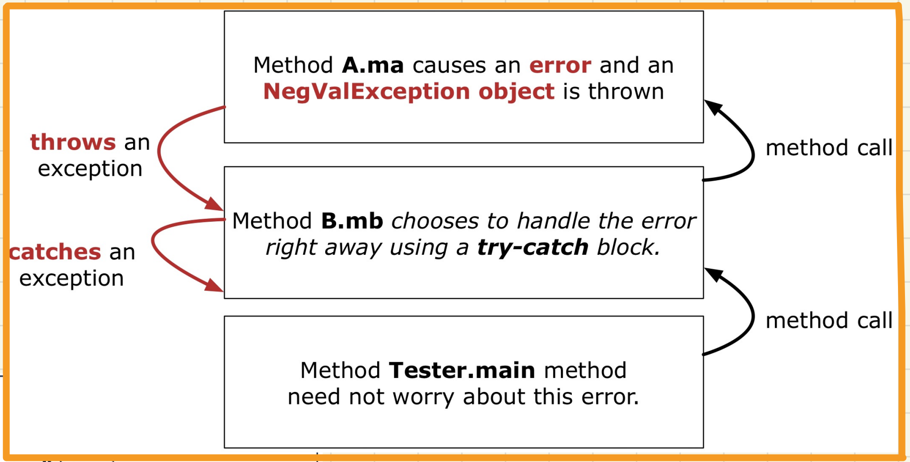
call stack

A.ma

B.mb

Tester.main

# Version 1:
# Handle the Exception in B.mb



Method **A.ma** causes an **error** and an **NegValException object** is thrown

**throws** an exception

method call

Method **B.mb** *chooses to handle the error right away using a* ***try-catch*** *block.*

**catches** an exception

method call

Method **Tester.main** method need not worry about this error.

```
class A {
  ma(int i) throws NegValException {
    if(i < 0) { throw new NegValException("Error."); }
    else { /* Do something. */ }
  } }
```

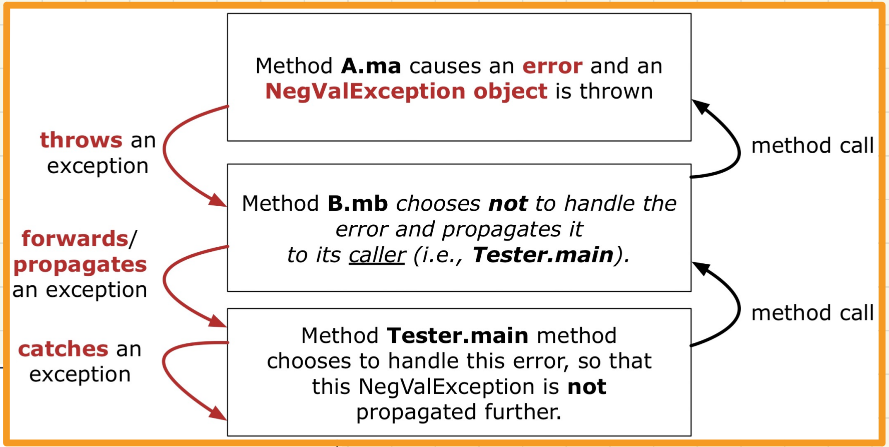
```
class B {
  mb(int i) {
    A oa = new A();
    try { oa.ma(i); }
    catch(NegValException nve) { /* Do something. */ }
  } }
```

```
class Tester {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    int i = input.nextInt();
    B ob = new B();
    ob.mb(i); /* Error, if any, would have been handled in B.mb. */
  } }
```

no exception handling is necessary ∵ it's handled in B already.

# Version 2:

# Handle the Exception in Tester.main



Method **A.ma** causes an **error** and an **NegValException object** is thrown

Method **B.mb** *chooses **not** to handle the error and propagates it to its caller (i.e., **Tester.main**).*

Method **Tester.main** method chooses to handle this error, so that this NegValException is **not** propagated further.

```
class A {
  ma(int i) throws NegValException {
    if(i < 0) { throw new NegValException("Error."); }
    else { /* Do something. */ }
  } }
```

where we signal the Error.

```
class B {
  mb(int i) throws NegValException {
    A oa = new A();
    oa.ma(i);
  } }
```

handled by specify

any caller of B.mb should handle this exception

```
class Tester {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    int i = input.nextInt();
    B ob = new B();
    try { ob.mb(i); }
    catch(NegValException nve) { /* Do something. */ }
  } }
```

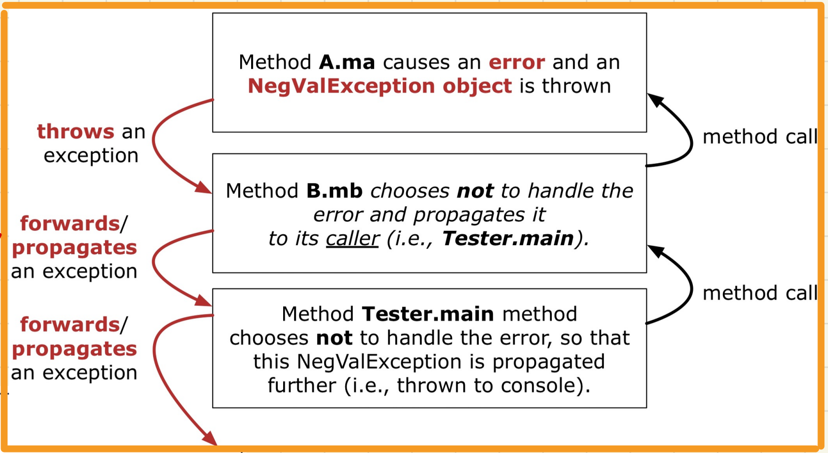
may throw NegValExcep

# Version 3:

# Handle in Neither Classes on Call Stack

throw : signal an error

throws : → warn any potential caller that they must handle this error .


Method **A.ma** causes an **error** and an **NegValException object** is thrown

**throws** an exception

Method **B.mb** *chooses **not** to handle the error and propagates it to its caller (i.e., **Tester.main**).*

**forwards**/ **propagates** an exception

Method **Tester.main** method chooses **not** to handle the error, so that this NegValException is propagated further (i.e., thrown to console).

**forwards**/ **propagates** an exception

```
class A {
  ma(int i) throws NegValException {
    if(i < 0) { throw new NegValException("Error."); }
    else { /* Do something. */ }
  } }
```

```
class B {
  mb(int i) throws NegValException {
    A oa = new A();
    oa.ma(i);
  } }
```

```
class Tester {
  public static void main(String[] args) throws NegValException {
    Scanner input = new Scanner(System.in);
    int i = input.nextInt();
    B ob = new B();
    ob.mb(i);
  } }
```

# Error Handling via Exceptions: Circles (Version 1)

```
public class InvalidRadiusException extends Exception {
  public InvalidRadiusException(String s) {
    super(s);
  }
}
```

→ Case 1: Valid radius 5.

Case 2: Invalid radius -4.

```
class Circle {
  double radius;
  Circle() { /* radius defaults to 0 */ }
  void setRadius(double r) throws InvalidRadiusException {
    if (r < 0) {
      throw new InvalidRadiusException("Negative radius.");
    }
    else { radius = r; }
  }
  double getArea() { return radius * radius * 3.14; }
}
```

5

-4

```
class CircleCalculator1 {
  public static void main(String[] args) {
    Circle c = new Circle();
    try {
      c.setRadius([illegible]);
      double area = c.getArea();
      System.out.println("Area: " + area);
    }
    catch(InvalidRadiusException e) {
      System.out.println(e);
    }
  } }
```

5 -4

IRE thrown

# Error Handling via Exceptions: Circles (Version 2)

```java
public class InvalidRadiusException extends Exception {
  public InvalidRadiusException(String s) {
    super(s);
  }
}
```

**Test Case:**

User enters -5

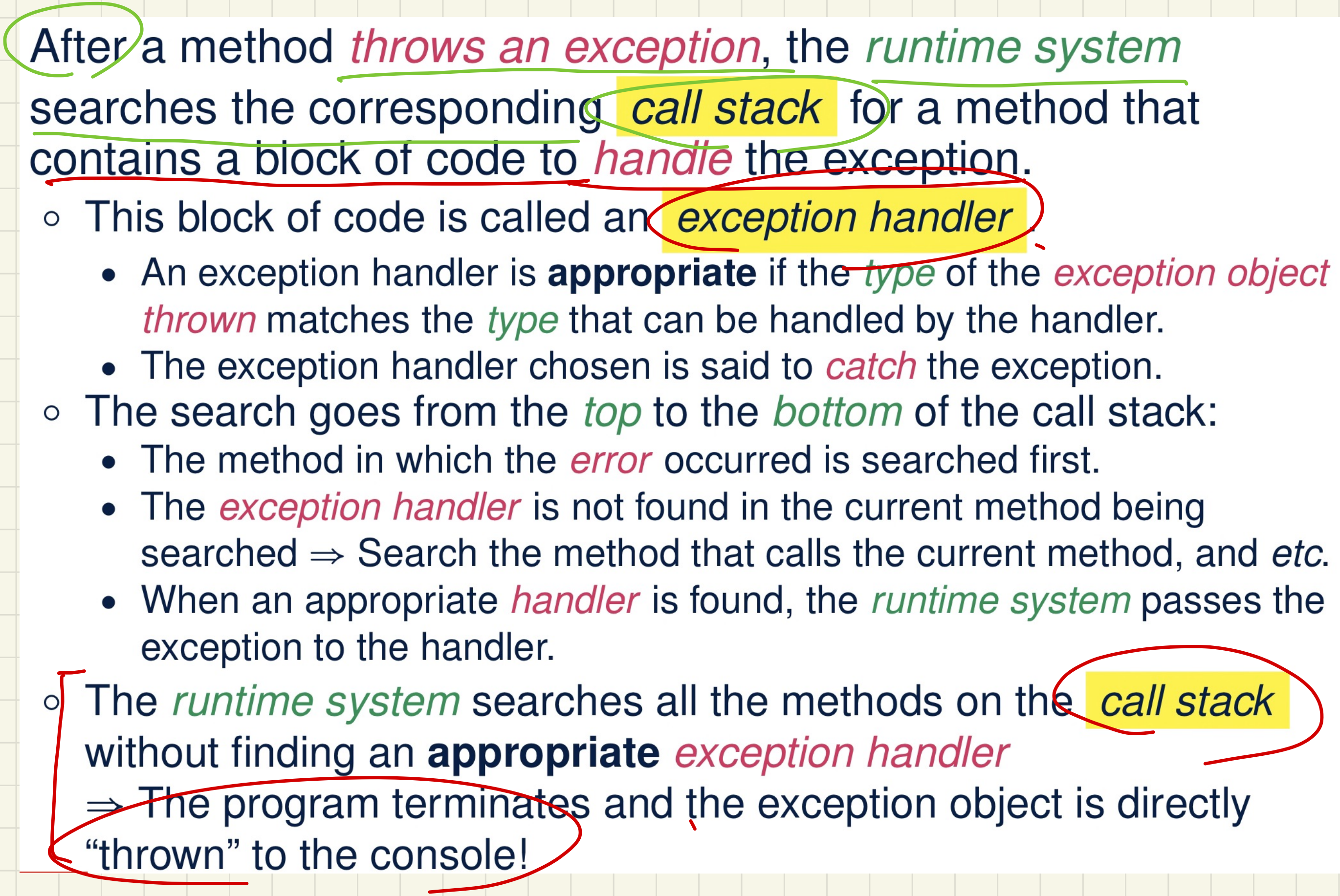
Then user enters 10

```java
class Circle {
  double radius;
  Circle() { /* radius defaults to 0 */ }
  void setRadius(double r) throws InvalidRadiusException {
    if (r < 0) {
      throw new InvalidRadiusException("Negative radius.");
    }
    else { radius = r; }
  }
  double getArea() { return radius * radius * 3.14; }
}
```

```java
public class CircleCalculator2 {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    boolean inputRadiusIsValid = false;
    while(!inputRadiusIsValid) {
      System.out.println("Enter a radius:");
      double r = input.nextDouble();
      Circle c = new Circle();
      try { c.setRadius(r); L1
            inputRadiusIsValid = true; L2
            System.out.print("Circle with radius " + r);
            System.out.println(" has area: "+ c.getArea()); }
      catch(InvalidRadiusException e) { print("Try again!"); }
    } } }
```

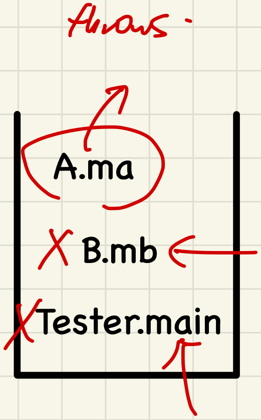
if we are able to move from L1 to L2 what does it mean?

# What to Do When an Exception is Thrown?



After a method *throws an exception*, the *runtime system* searches the corresponding *call stack* for a method that contains a block of code to *handle* the exception.

- This block of code is called an *exception handler*.
  - An exception handler is **appropriate** if the *type* of the *exception object thrown* matches the *type* that can be handled by the handler.
  - The exception handler chosen is said to *catch* the exception.
- The search goes from the *top* to the *bottom* of the call stack:
  - The method in which the *error* occurred is searched first.
  - The *exception handler* is not found in the current method being searched ⇒ Search the method that calls the current method, and *etc.*
  - When an appropriate *handler* is found, the *runtime system* passes the exception to the handler.
- The *runtime system* searches all the methods on the *call stack* without finding an **appropriate** *exception handler* ⇒ The program terminates and the exception object is directly "thrown" to the console!

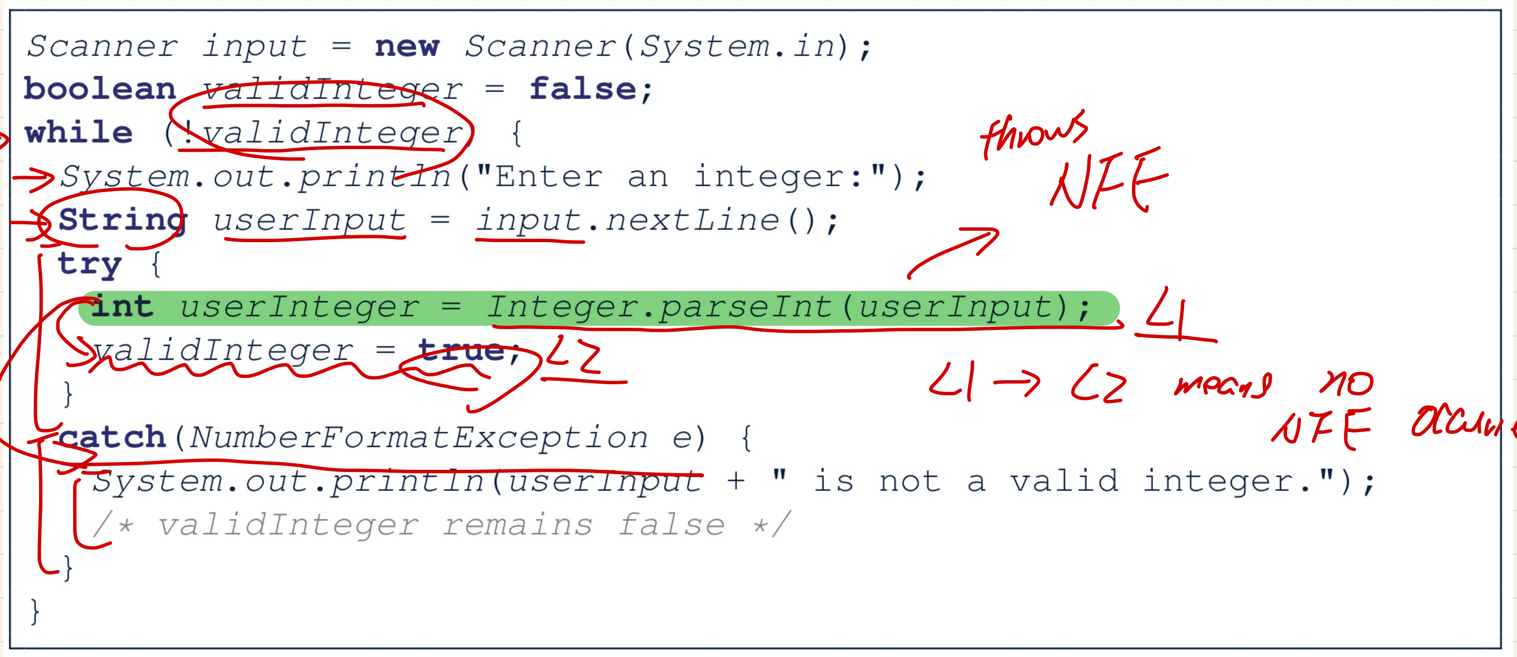


Read from user:

"23" string

↳ 23 int

# More Example: Parsing Strings as Integers

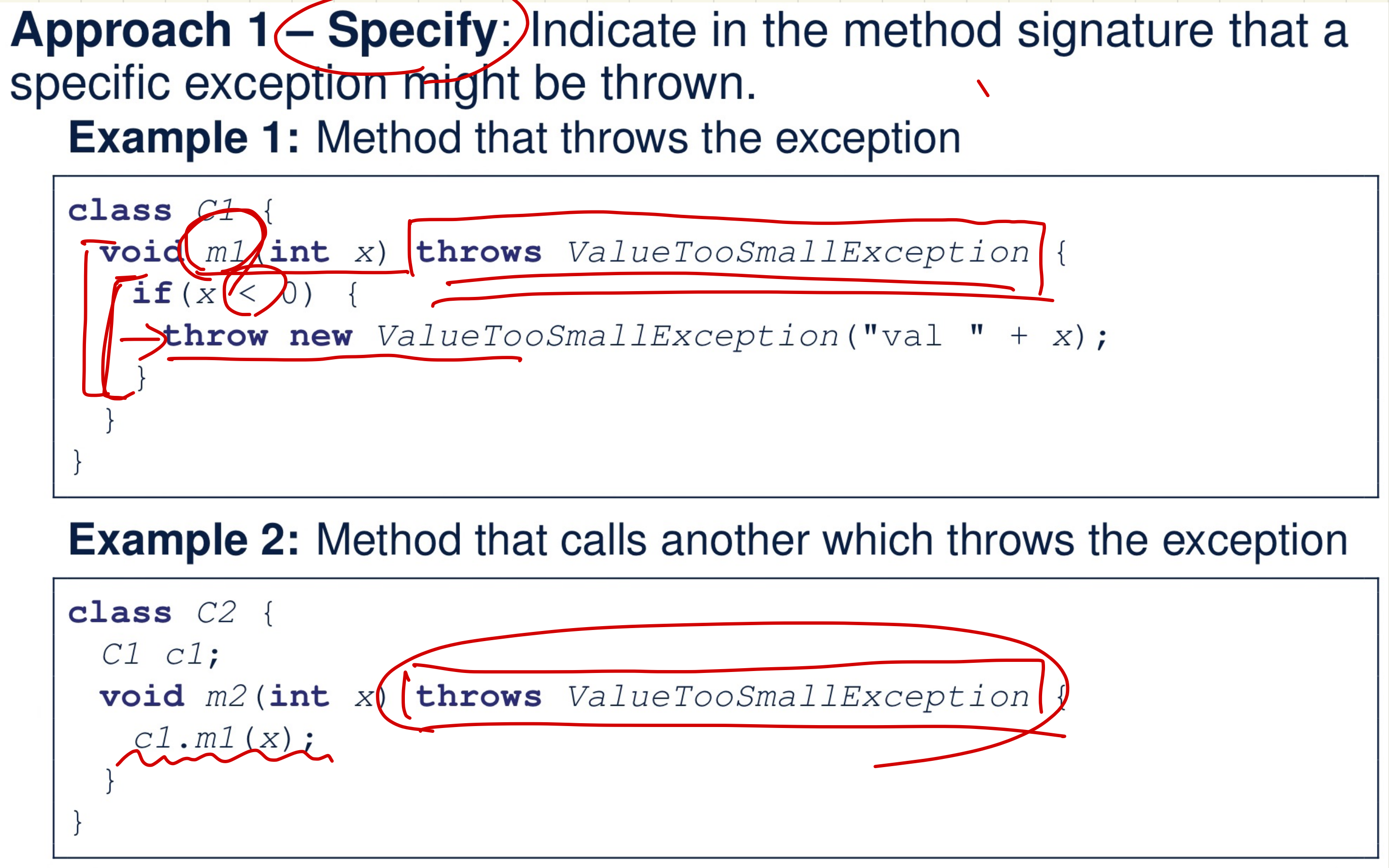


```java
Scanner input = new Scanner(System.in);
boolean validInteger = false;
while (!validInteger) {
  System.out.println("Enter an integer:");
  String userInput = input.nextLine();
  try {
    int userInteger = Integer.parseInt(userInput);   L1
    validInteger = true;   L2
  }
  catch(NumberFormatException e) {
    System.out.println(userInput + " is not a valid integer.");
    /* validInteger remains false */
  }
}
```

throws NFE

L1 → L2 means no NFE occured

# Review: Specify-or-Catch Principle



**Approach 1 – Specify**: Indicate in the method signature that a specific exception might be thrown.

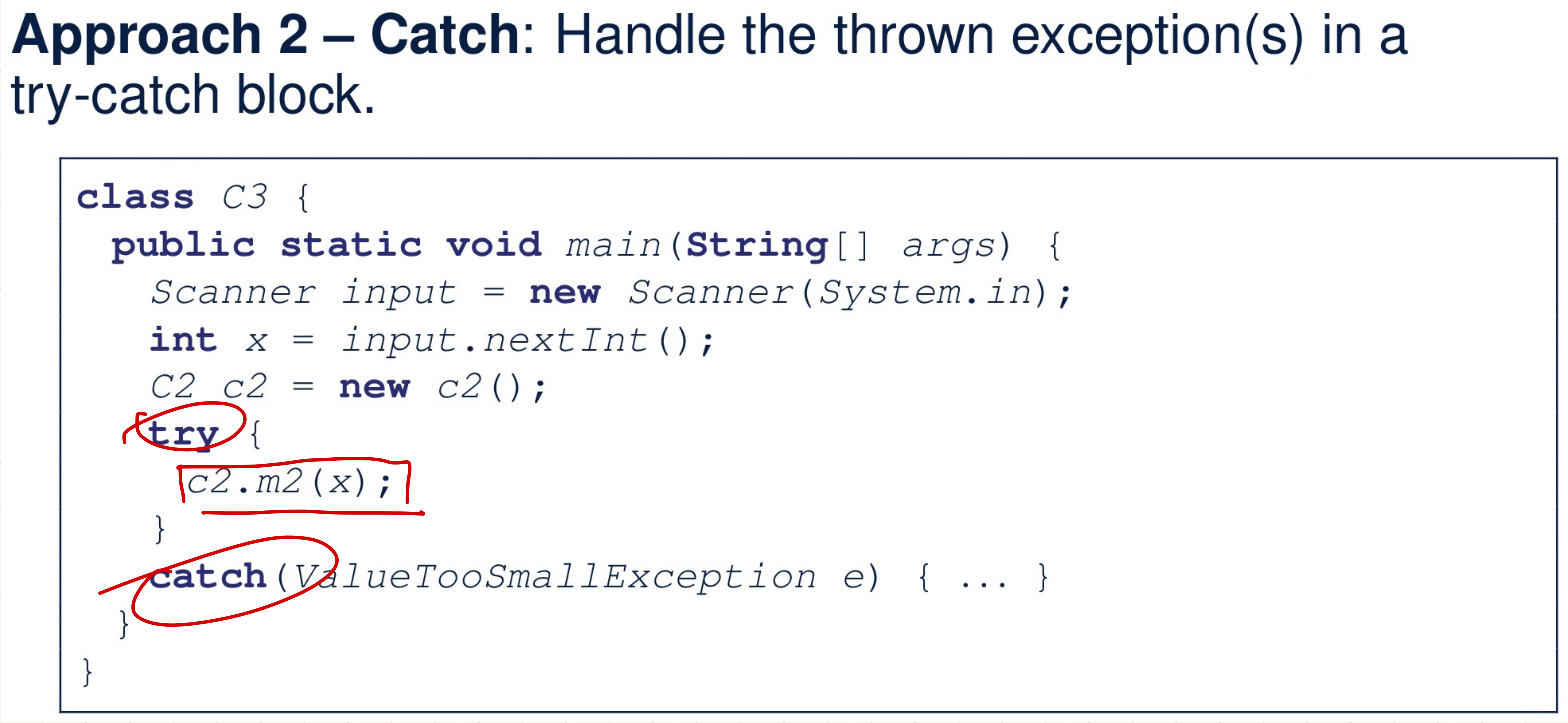
**Example 1:** Method that throws the exception

```
class C1 {
  void m1(int x) throws ValueTooSmallException {
    if(x < 0) {
      throw new ValueTooSmallException("val " + x);
    }
  }
}
```

**Example 2:** Method that calls another which throws the exception

```
class C2 {
  C1 c1;
  void m2(int x) throws ValueTooSmallException {
    c1.m1(x);
  }
}
```

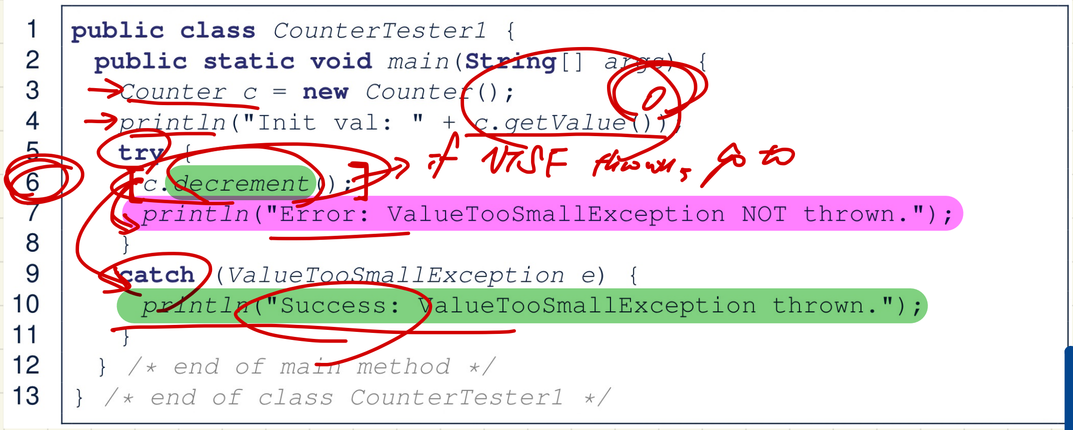
# Review: Specify-or-Catch Principle



**Approach 2 – Catch**: Handle the thrown exception(s) in a try-catch block.

```
class C3 {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    int x = input.nextInt();
    C2 c2 = new c2();
    try {
      c2.m2(x);
    }
    catch(ValueTooSmallException e) { ... }
  }
}
```

# Manual Test 1 from the Console



```java
public class CounterTester1 {
  public static void main(String[] args) {
    Counter c = new Counter();
    println("Init val: " + c.getValue());
    try {
      c.decrement();
      println("Error: ValueTooSmallException NOT thrown.");
    }
    catch (ValueTooSmallException e) {
      println("Success: ValueTooSmallException thrown.");
    }
  } /* end of main method */
} /* end of class CounterTester1 */
```

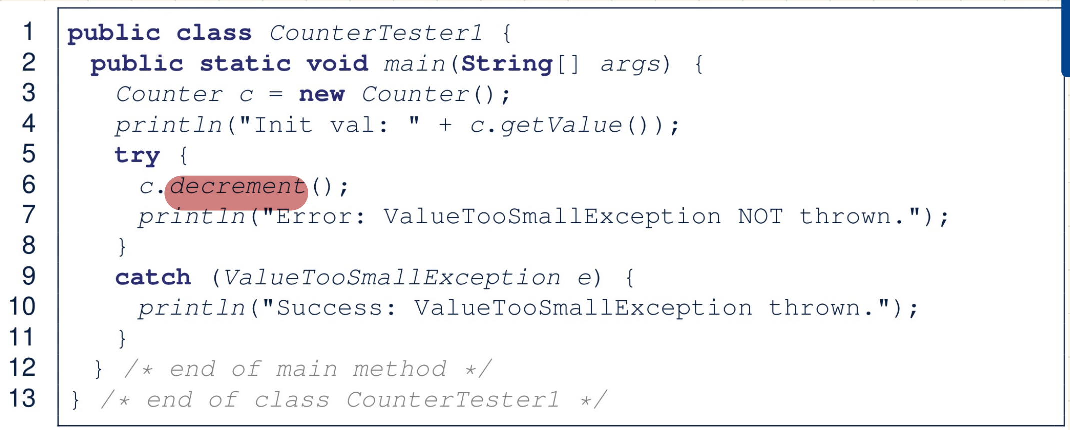
0

if VTSF thrown, go to

What if decrement is implemented correctly?

EXPECTED BEHAVOUR:

Calling c.decrement() when c.value is 0 should trigger a ValueTooSmallException.



```java
public class CounterTester1 {
  public static void main(String[] args) {
    Counter c = new Counter();
    println("Init val: " + c.getValue());
    try {
      c.decrement();
      println("Error: ValueTooSmallException NOT thrown.");
    }
    catch (ValueTooSmallException e) {
      println("Success: ValueTooSmallException thrown.");
    }
  } /* end of main method */
} /* end of class CounterTester1 */
```

What if decrement is implemented incorrectly?

# LECTURE 7

WEDNESDAY SEPTEMBER 25

- WRITTEN TEST REVIEW SESSION

  CLH M 10am ~ 12noon

  MONDAY SEPTEMBER 30

  POST YOUR QUESTIONS ON A GOOGLE DOC

- SEATING PLAN OF LAB TEST

# Recap of Exceptions

- Catch-or-Specify Requirement

## Normal Flow of Execution

```
... /* before, ouside try-catch block */
try {
  o.m(...); /* may throw SomeException */
  ... / * rest of try-block */
}
catch (SomeException se) {
  ... /* rest of catch-block */
}
... /* after, ouside try-catch block */
```

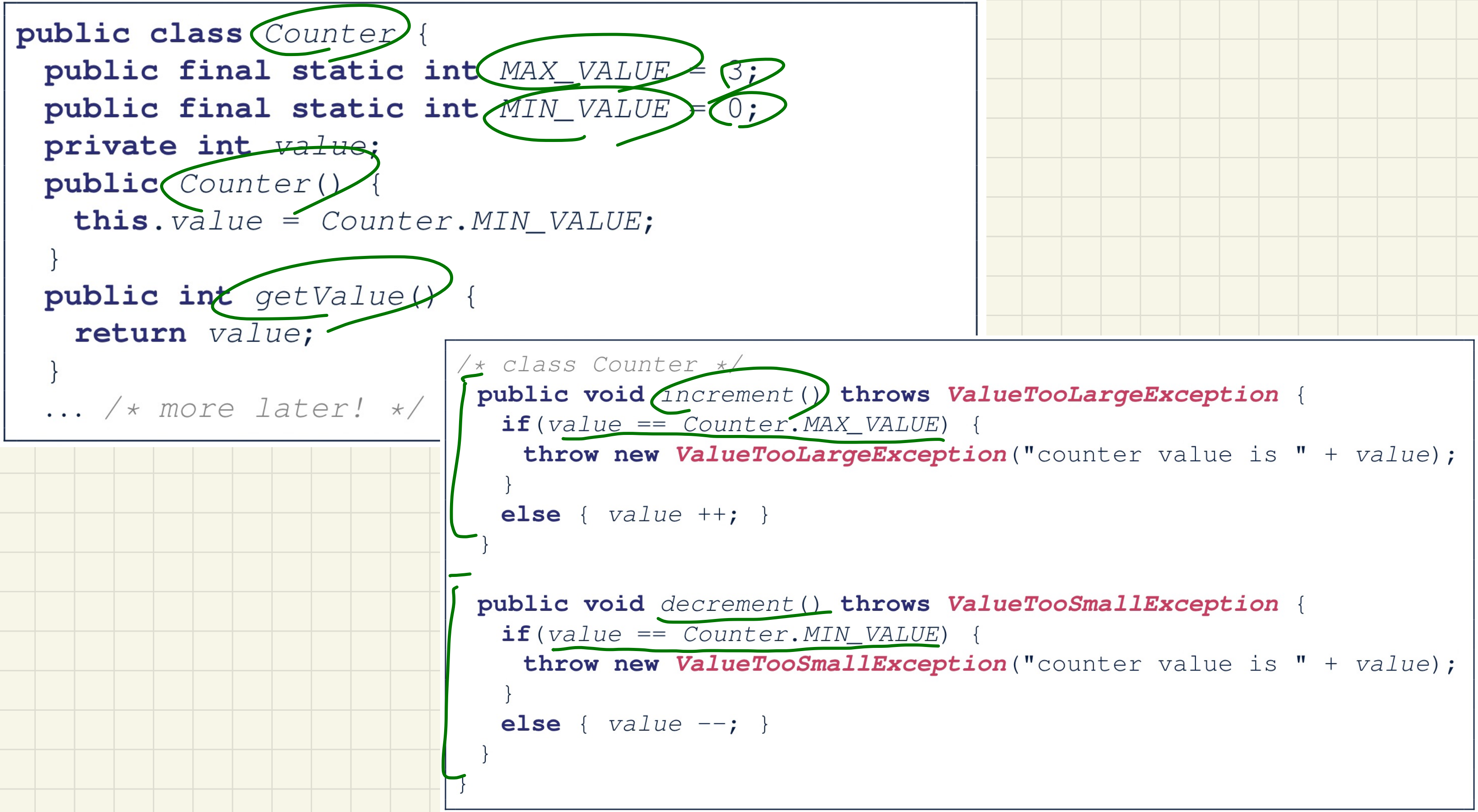
When the exception does not occur

## Abnormal Flow of Execution

```
... /* before, ouside try-catch block */
try {
  o.m(...); /* may throw SomeException */
  ... / * rest of try-block */
}
catch (SomeException se) {
  ... /* rest of catch-block */
}
... /* after, ouside try-catch block */
```

When the exception occurs

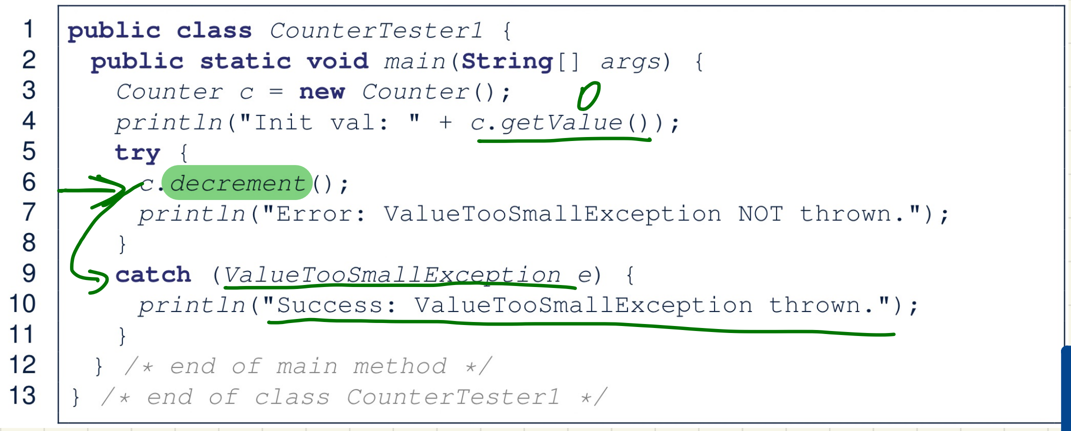
# Class for Bounded Counters



```
public class Counter {
  public final static int MAX_VALUE = 3;
  public final static int MIN_VALUE = 0;
  private int value;
  public Counter() {
    this.value = Counter.MIN_VALUE;
  }
  public int getValue() {
    return value;
  }
  ... /* more later! */
```

```
/* class Counter */
  public void increment() throws ValueTooLargeException {
    if(value == Counter.MAX_VALUE) {
      throw new ValueTooLargeException("counter value is " + value);
    }
    else { value ++; }
  }

  public void decrement() throws ValueTooSmallException {
    if(value == Counter.MIN_VALUE) {
      throw new ValueTooSmallException("counter value is " + value);
    }
    else { value --; }
  }
}
```

# Manual Tester 1 from the Console

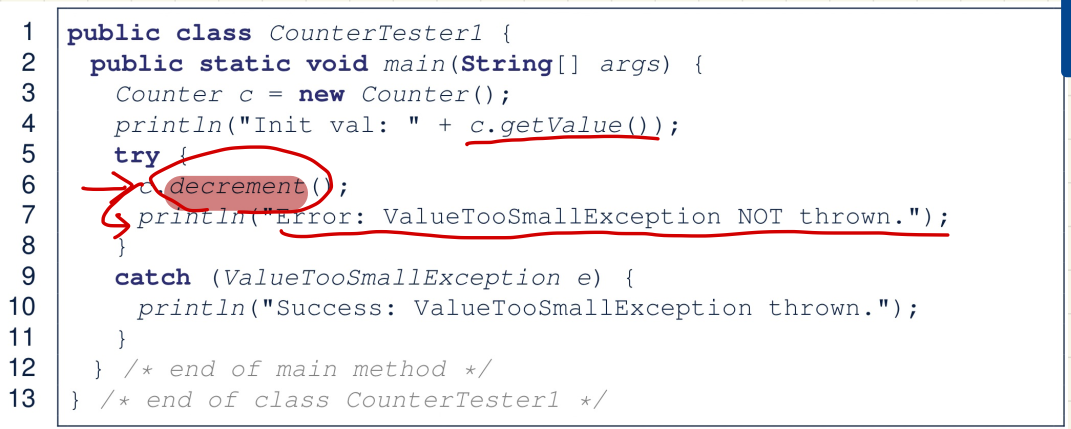


```java
public class CounterTester1 {
  public static void main(String[] args) {
    Counter c = new Counter();
    println("Init val: " + c.getValue());
    try {
      c.decrement();
      println("Error: ValueTooSmallException NOT thrown.");
    }
    catch (ValueTooSmallException e) {
      println("Success: ValueTooSmallException thrown.");
    }
  } /* end of main method */
} /* end of class CounterTester1 */
```

What if decrement is implemented correctly?

## EXPECTED BEHAVIOUR :

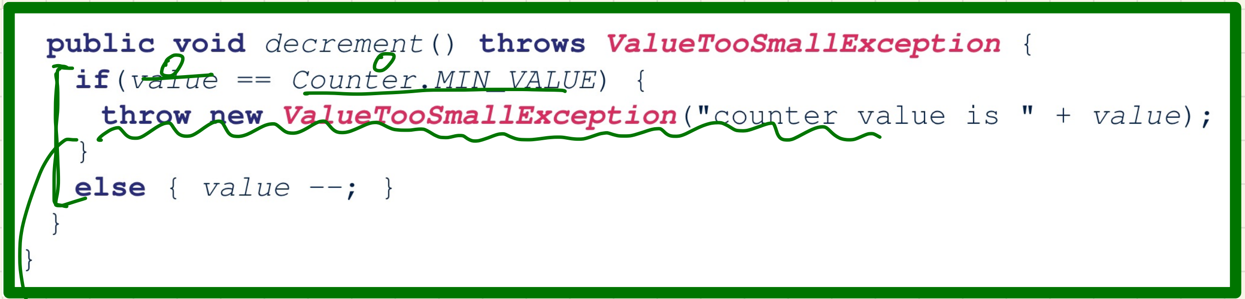
Calling c.decrement() when c.value is 0 should trigger a ValueTooSmallException.



```java
public class CounterTester1 {
  public static void main(String[] args) {
    Counter c = new Counter();
    println("Init val: " + c.getValue());
    try {
      c.decrement();
      println("Error: ValueTooSmallException NOT thrown.");
    }
    catch (ValueTooSmallException e) {
      println("Success: ValueTooSmallException thrown.");
    }
  } /* end of main method */
} /* end of class CounterTester1 */
```

What if decrement is implemented incorrectly?

# Running Console Tester 1 on Correct Implementation

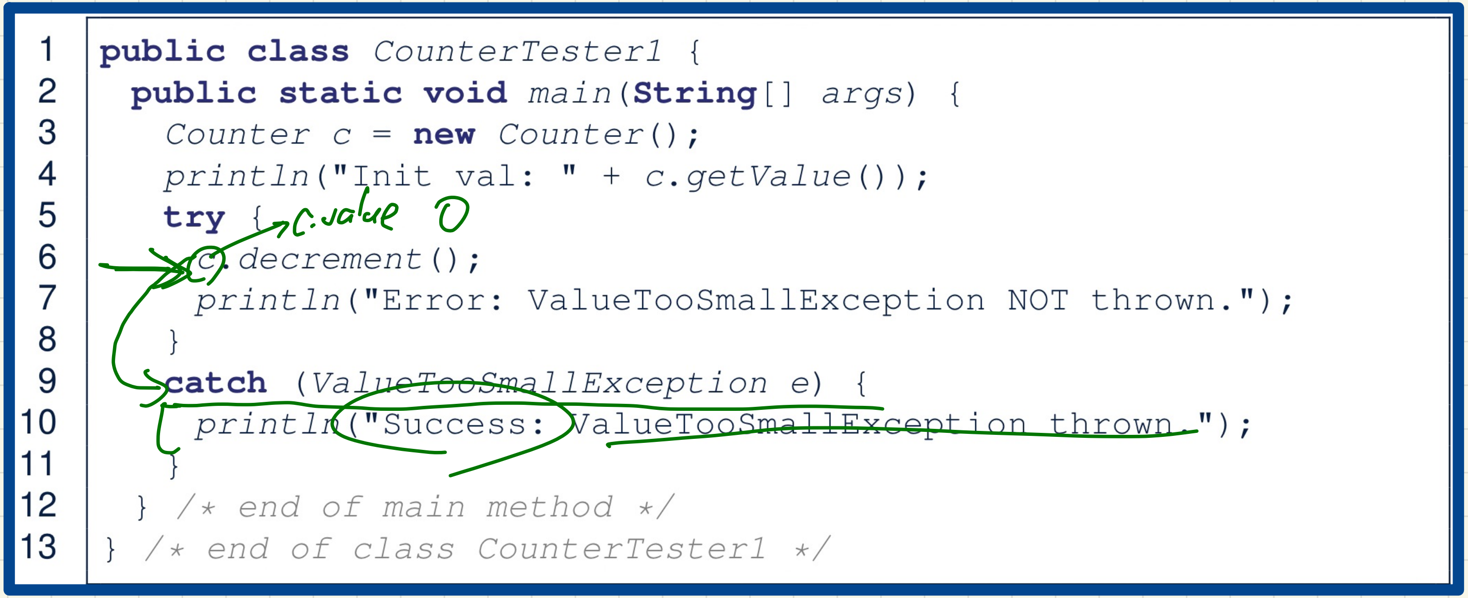


```
public void decrement() throws ValueTooSmallException {
  if(value == Counter.MIN_VALUE) {
    throw new ValueTooSmallException("counter value is " + value);
  }
  else { value --; }
 }
}
```

0

0

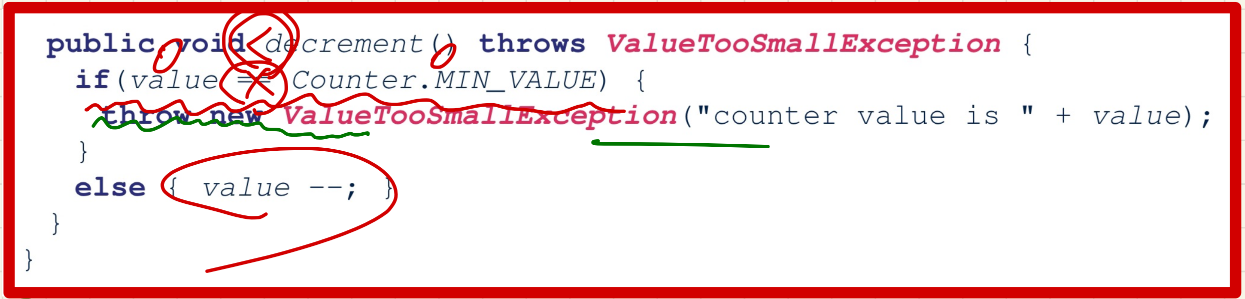
correct



```
public class CounterTester1 {
  public static void main(String[] args) {
    Counter c = new Counter();
    println("Init val: " + c.getValue());
    try {
      c.decrement();
      println("Error: ValueTooSmallException NOT thrown.");
    }
    catch (ValueTooSmallException e) {
      println("Success: ValueTooSmallException thrown.");
    }
  } /* end of main method */
} /* end of class CounterTester1 */
```

c.value 0

# Running Console Tester 1 on **Incorrect** Implementation



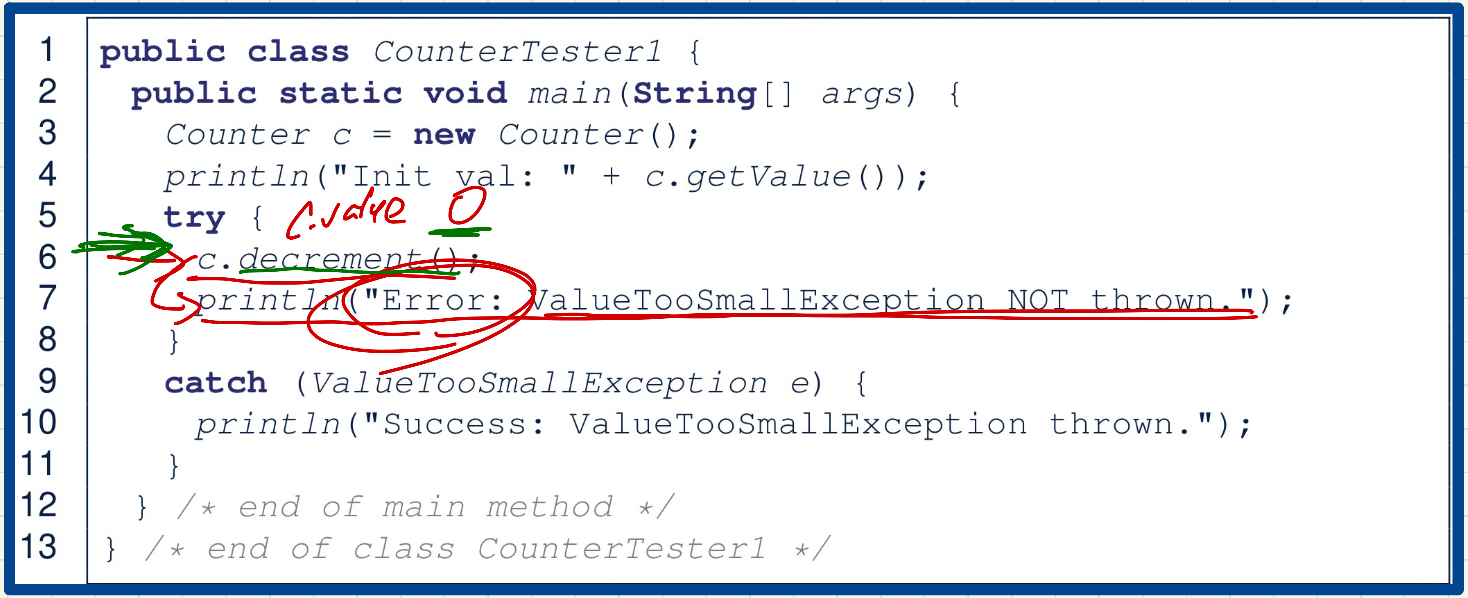
```
public void decrement() throws ValueTooSmallException {
  if(value == Counter.MIN_VALUE) {
    throw new ValueTooSmallException("counter value is " + value);
  }
  else { value --; }
 }
}
```

0 0

<

0 <= 0

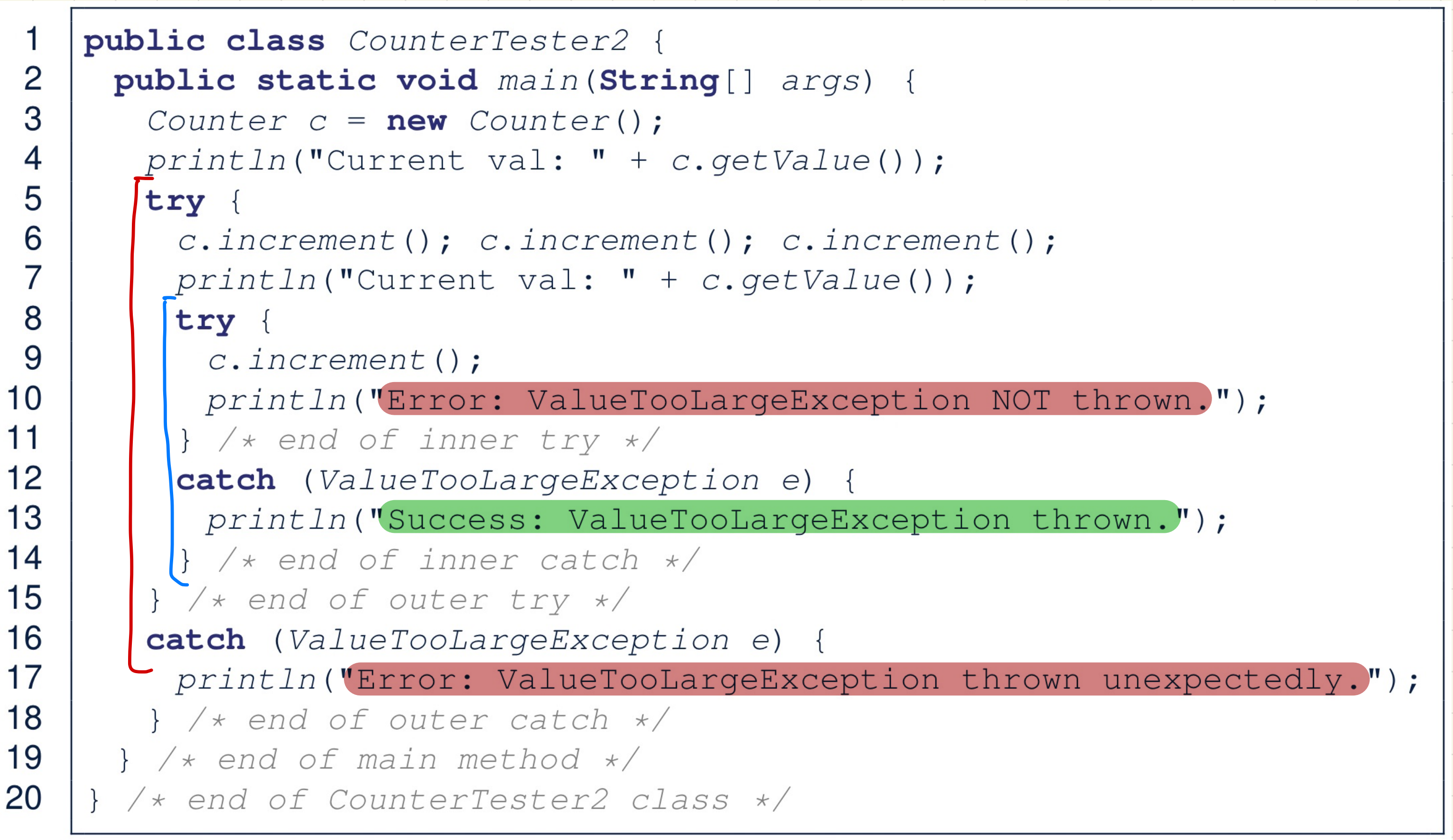
T



```
public class CounterTester1 {
  public static void main(String[] args) {
    Counter c = new Counter();
    println("Init val: " + c.getValue());
    try {
      c.decrement();
      println("Error: ValueTooSmallException NOT thrown.");
    }
    catch (ValueTooSmallException e) {
      println("Success: ValueTooSmallException thrown.");
    }
  } /* end of main method */
} /* end of class CounterTester1 */
```

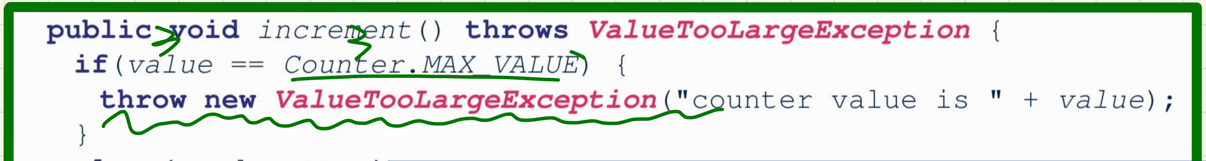
c.value 0

# Manual Tester 2 from the Console

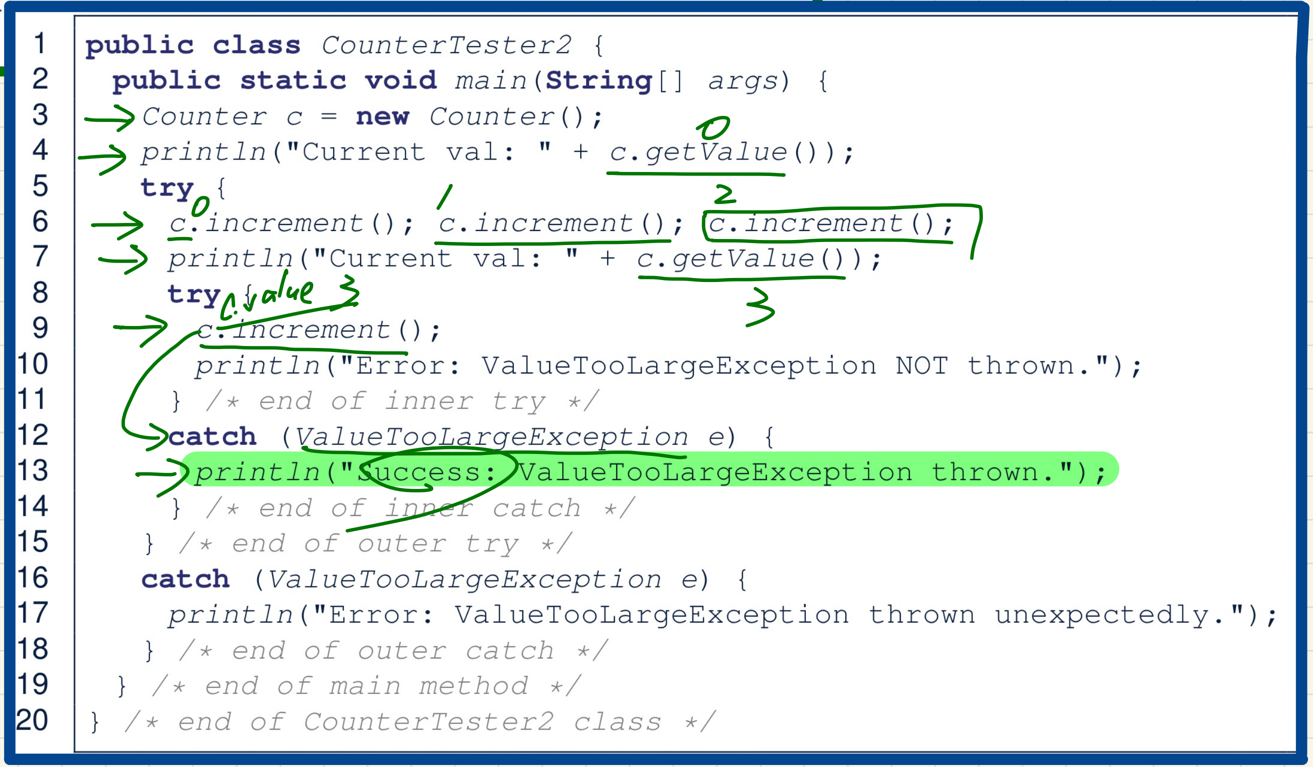


```java
public class CounterTester2 {
  public static void main(String[] args) {
    Counter c = new Counter();
    println("Current val: " + c.getValue());
    try {
      c.increment(); c.increment(); c.increment();
      println("Current val: " + c.getValue());
      try {
        c.increment();
        println("Error: ValueTooLargeException NOT thrown.");
      } /* end of inner try */
      catch (ValueTooLargeException e) {
        println("Success: ValueTooLargeException thrown.");
      } /* end of inner catch */
    } /* end of outer try */
    catch (ValueTooLargeException e) {
      println("Error: ValueTooLargeException thrown unexpectedly.");
    } /* end of outer catch */
  } /* end of main method */
} /* end of CounterTester2 class */
```

# Running Console Tester 2 on Correct Implementation

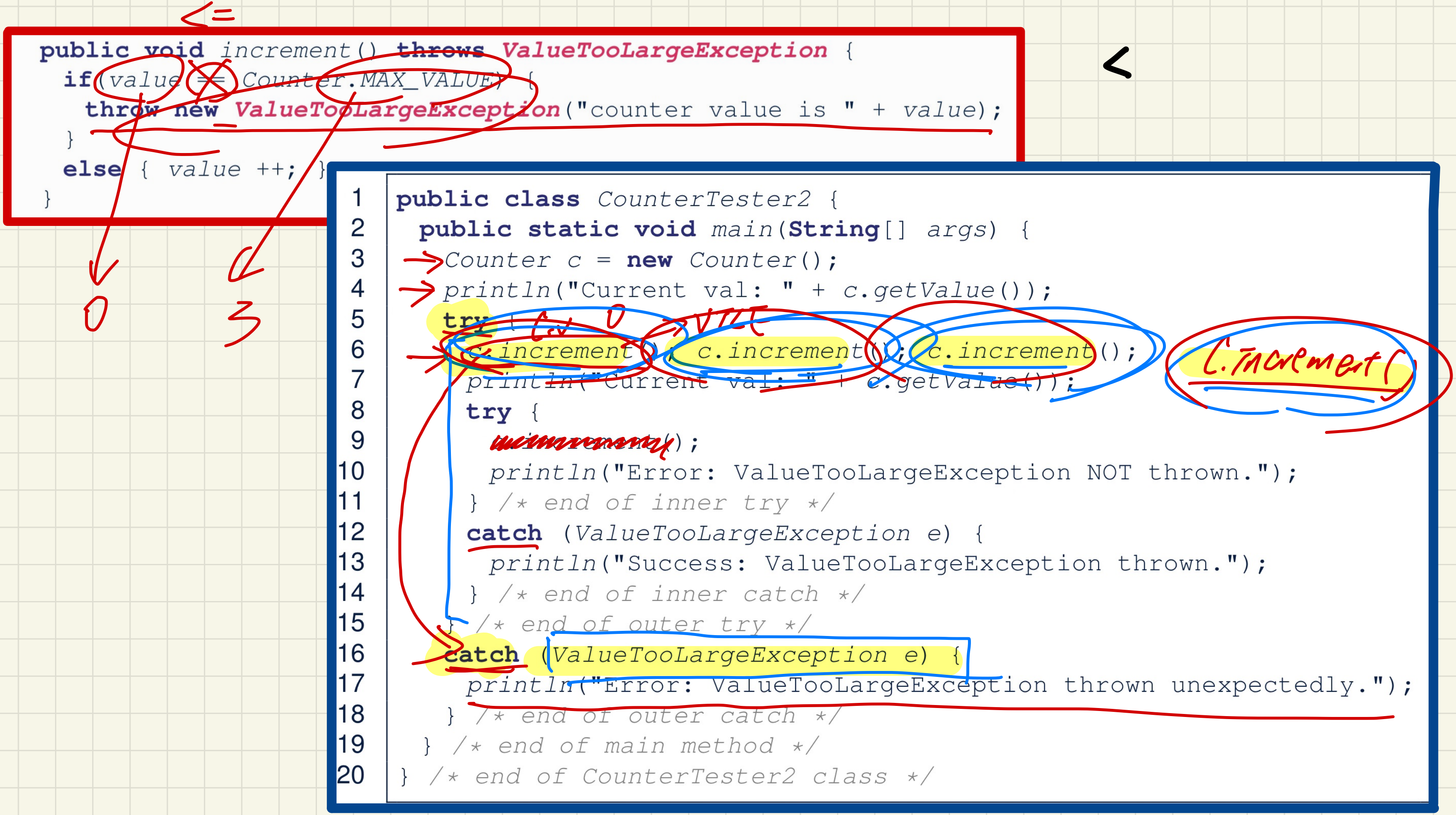


```java
public void increment() throws ValueTooLargeException {
  if(value == Counter.MAX_VALUE) {
    throw new ValueTooLargeException("counter value is " + value);
  }
  else { value ++; }
}
```



```java
public class CounterTester2 {
  public static void main(String[] args) {
    Counter c = new Counter();
    println("Current val: " + c.getValue());
    try {
      c.increment(); c.increment(); c.increment();
      println("Current val: " + c.getValue());
      try {
        c.increment();
        println("Error: ValueTooLargeException NOT thrown.");
      } /* end of inner try */
      catch (ValueTooLargeException e) {
        println("Success: ValueTooLargeException thrown.");
      } /* end of inner catch */
    } /* end of outer try */
    catch (ValueTooLargeException e) {
      println("Error: ValueTooLargeException thrown unexpectedly.");
    } /* end of outer catch */
  } /* end of main method */
} /* end of CounterTester2 class */
```

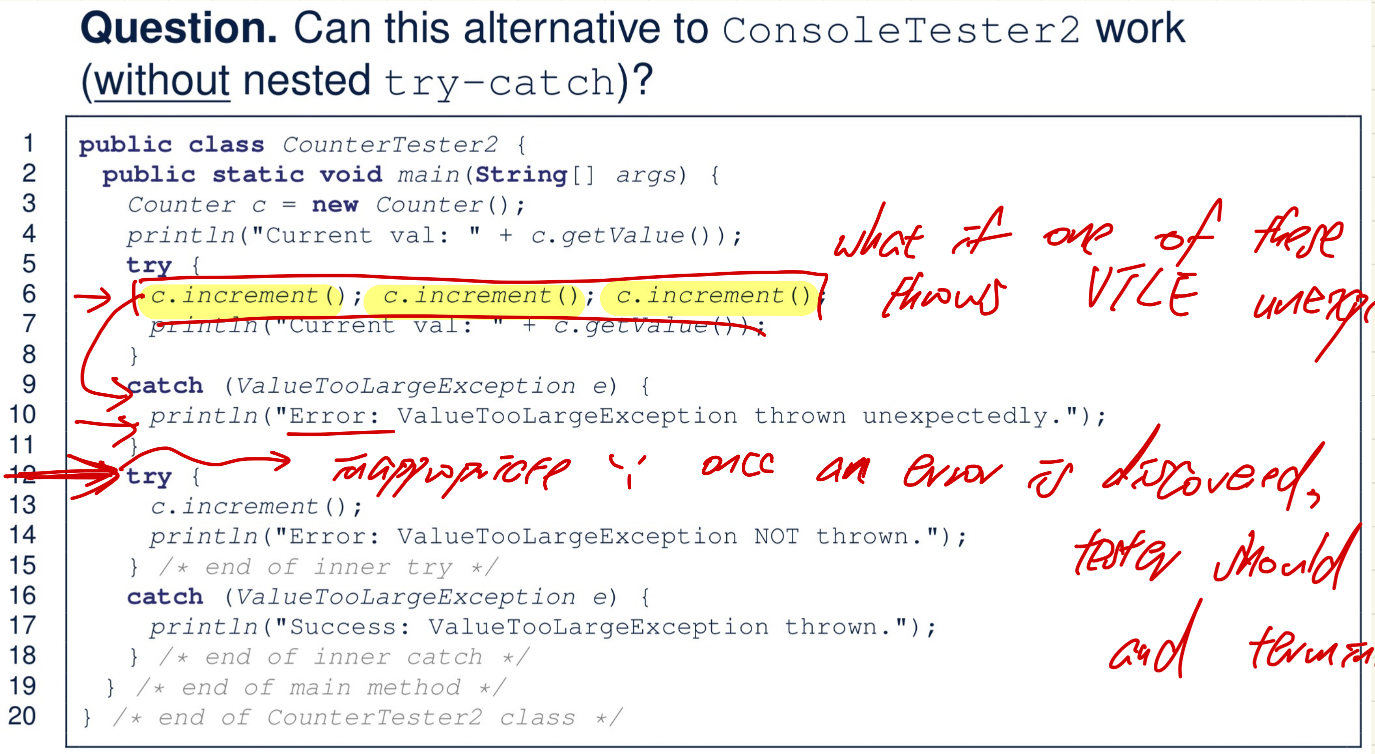
# Running Console Tester 2 on Incorrect Implementation 1



```
public void increment() throws ValueTooLargeException {
  if(value == Counter.MAX_VALUE) {
    throw new ValueTooLargeException("counter value is " + value);
  }
  else { value ++; }
}
```

```
public class CounterTester2 {
  public static void main(String[] args) {
    Counter c = new Counter();
    println("Current val: " + c.getValue());
    try {
      c.increment(); c.increment(); c.increment();
      println("Current val: " + c.getValue());
      try {
        ~~c.increment~~();
        println("Error: ValueTooLargeException NOT thrown.");
      } /* end of inner try */
      catch (ValueTooLargeException e) {
        println("Success: ValueTooLargeException thrown.");
      } /* end of inner catch */
    } /* end of outer try */
    catch (ValueTooLargeException e) {
      println("Error: ValueTooLargeException thrown unexpectedly.");
    } /* end of outer catch */
  } /* end of main method */
} /* end of CounterTester2 class */
```

# Exercise



**Question.** Can this alternative to `ConsoleTester2` work (without nested `try-catch`)?

```
public class CounterTester2 {
  public static void main(String[] args) {
    Counter c = new Counter();
    println("Current val: " + c.getValue());
    try {
      c.increment(); c.increment(); c.increment();
      println("Current val: " + c.getValue());
    }
    catch (ValueTooLargeException e) {
      println("Error: ValueTooLargeException thrown unexpectedly.");
    }
    try {
      c.increment();
      println("Error: ValueTooLargeException NOT thrown.");
    } /* end of inner try */
    catch (ValueTooLargeException e) {
      println("Success: ValueTooLargeException thrown.");
    } /* end of inner catch */
  } /* end of main method */
} /* end of CounterTester2 class */
```

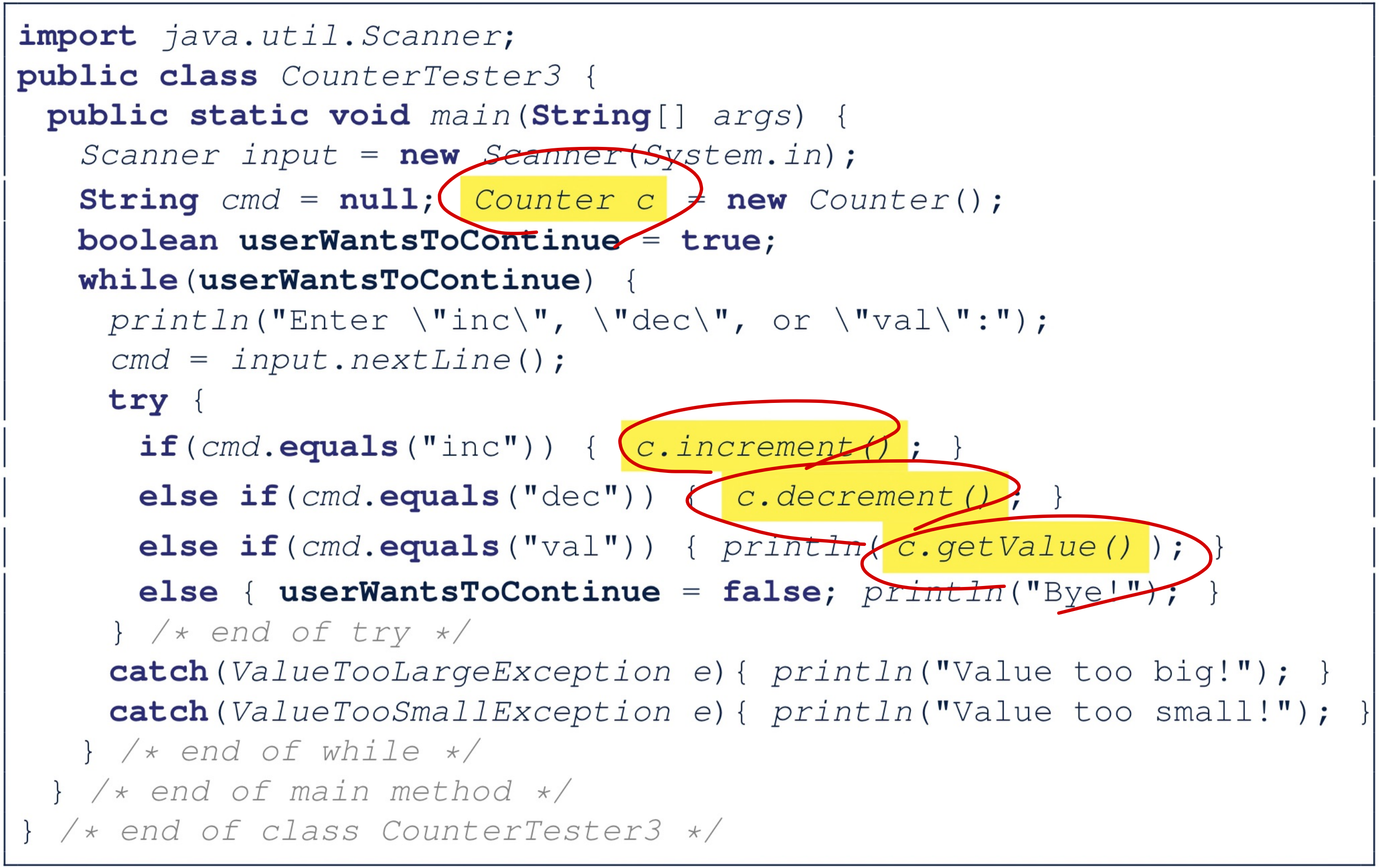
what if one of these throws VTLE unexpectedly

inappropriate ∵ once an error is discovered, tester should report and terminate right away

✓

**Hint:** What if one of the first 3 c.increment() **mistakenly** throws a ValueTooLargeException?

# A Manual, Iterative Console Tester



```
import java.util.Scanner;
public class CounterTester3 {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    String cmd = null; Counter c = new Counter();
    boolean userWantsToContinue = true;
    while(userWantsToContinue) {
      println("Enter \"inc\", \"dec\", or \"val\":");
      cmd = input.nextLine();
      try {
        if(cmd.equals("inc")) { c.increment() ; }
        else if(cmd.equals("dec")) { c.decrement() ; }
        else if(cmd.equals("val")) { println( c.getValue() ); }
        else { userWantsToContinue = false; println("Bye!"); }
      } /* end of try */
      catch(ValueTooLargeException e){ println("Value too big!"); }
      catch(ValueTooSmallException e){ println("Value too small!"); }
    } /* end of while */
  } /* end of main method */
} /* end of class CounterTester3 */
```

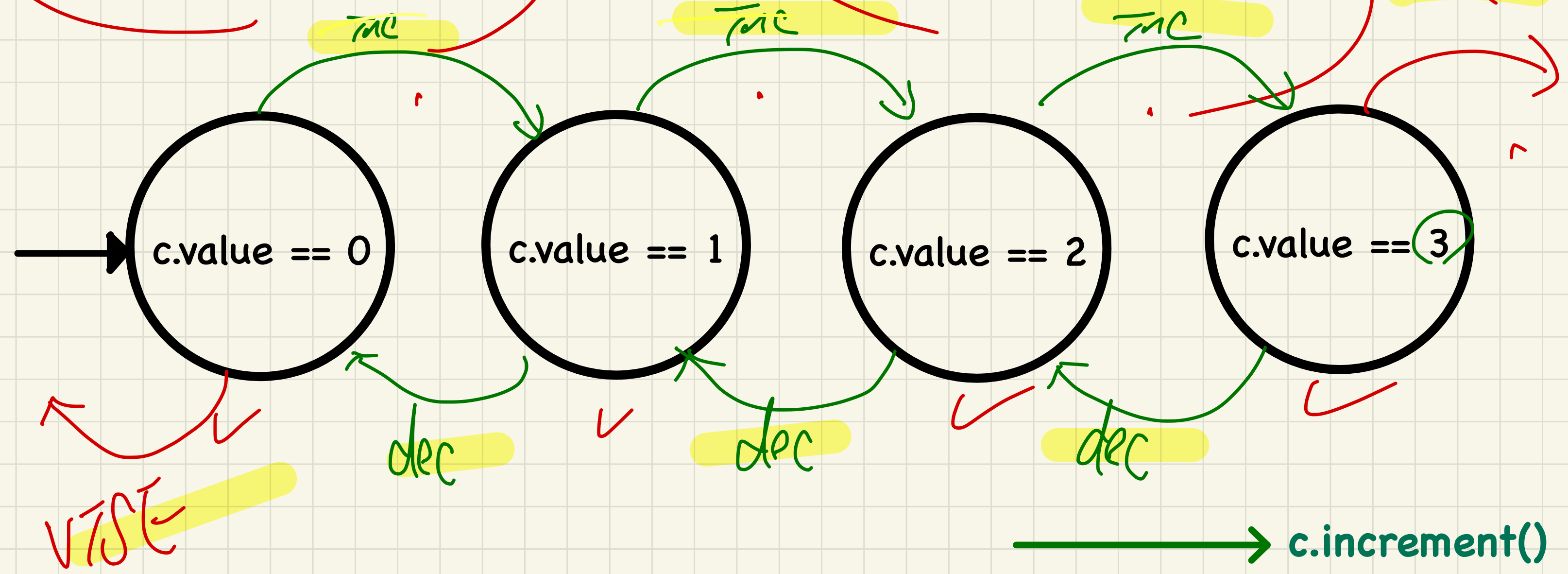
# Coming Up with Test Cases

0

3

Boundries:

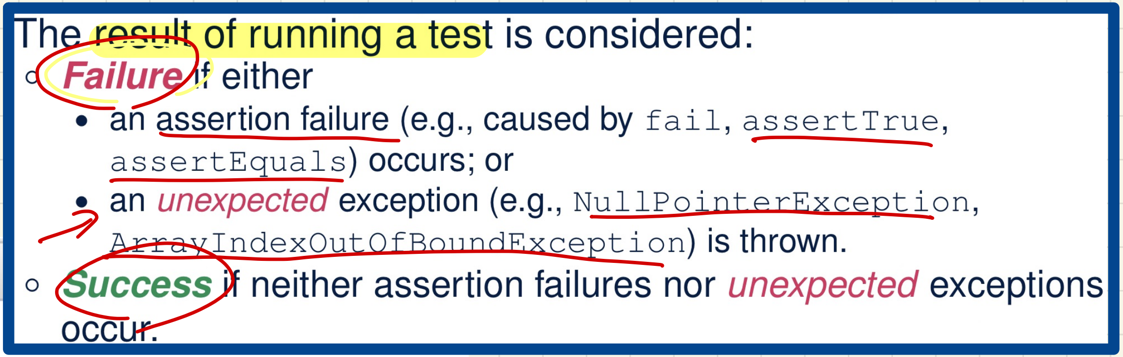
Counter.MIN_VALUE <= c.value <= Counter.MAX_VALUE



c.increment()

c.decrement()

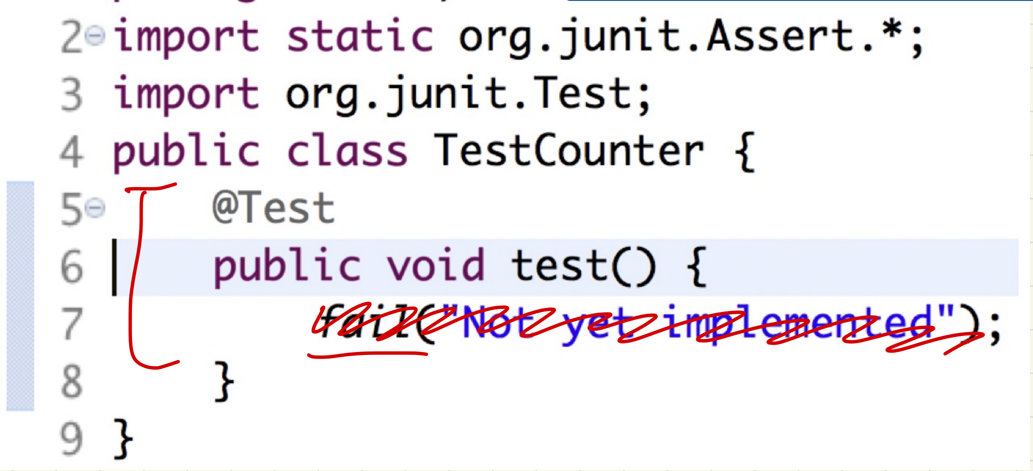
# A Default Test Case that Fails



The result of running a test is considered:

- *Failure* if either
  - an assertion failure (e.g., caused by `fail`, `assertTrue`, `assertEquals`) occurs; or
  - an *unexpected* exception (e.g., `NullPointerException`, `ArrayIndexOutOfBoundException`) is thrown.
- *Success* if neither assertion failures nor *unexpected* exceptions occur.

TestCounter.java



```
package tests;
import static org.junit.Assert.*;
import org.junit.Test;
public class TestCounter {
    @Test
    public void test() {
        ~~fail("Not yet implemented");~~
    }
}
```

What is the easiest way to making this test **pass**?

# JUnit Assertions Examples (2)

Consider the following class:

```
class Circle {
  double radius;
  Circle(double radius) { this.radius = radius; }
  int getArea() { return 3.14 * radius * radius; }
}
```

assertEquals(36.2984, c.getArea()) X

Then consider these assertions. Do they ***pass*** or ***fail***?

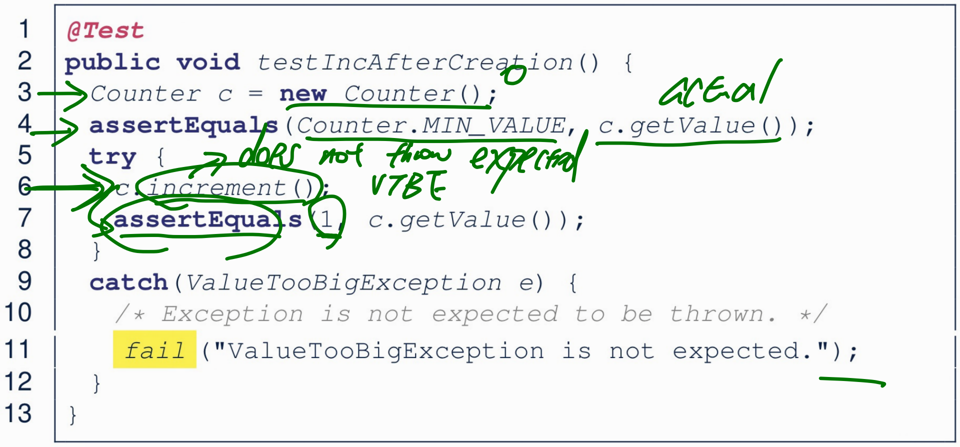
```
Circle c = new Circle(3.4);
assertTrue(36.2984, c.getArea(), 0.01); [redacted]
```

Equals

3.4 * 3.4 * 3.14

$36.2984 - 0.01 \le c.\text{getAr} \le 36.2984 + 0.01$

# JUnit where an Exception is Not Expected



```java
@Test
public void testIncAfterCreation() {
  Counter c = new Counter();
  assertEquals(Counter.MIN_VALUE, c.getValue());
  try {
    c.increment();
    assertEquals(1, c.getValue());
  }
  catch(ValueTooBigException e) {
    /* Exception is not expected to be thrown. */
    fail ("ValueTooBigException is not expected.");
  }
}
```

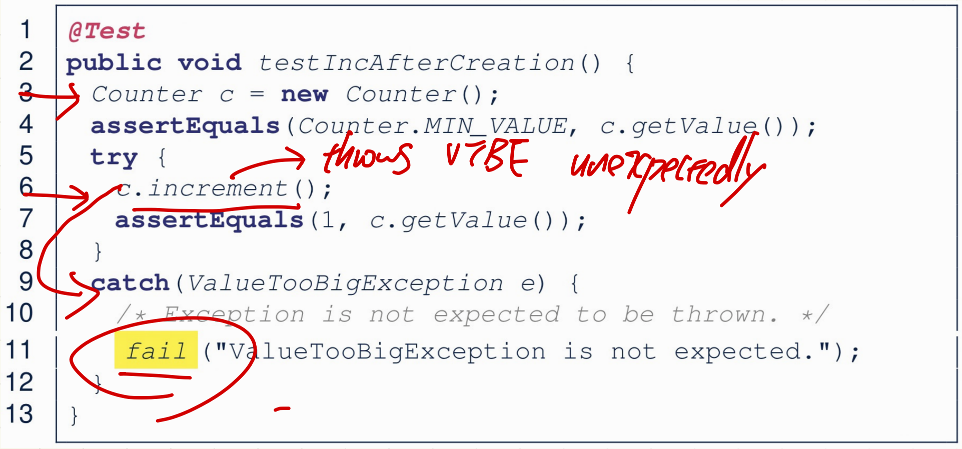
0

actual

does not throw expected

VTBE

What if method increment is implemented correctly?



```java
@Test
public void testIncAfterCreation() {
  Counter c = new Counter();
  assertEquals(Counter.MIN_VALUE, c.getValue());
  try {
    c.increment();
    assertEquals(1, c.getValue());
  }
  catch(ValueTooBigException e) {
    /* Exception is not expected to be thrown. */
    fail ("ValueTooBigException is not expected.");
  }
}
```

throws VTBE unexpectedly

What if method increment is implemented incorrectly?

# JUnit where an Exception is Expected (1)

JUnit Test

```java
@Test
public void testDecFromMinValue() {
  Counter c = new Counter();
  assertEquals(Counter.MIN_VALUE, c.getValue());
  try {
    c.decrement();
    fail ("ValueTooSmallException is expected.");
  }
  catch(ValueTooSmallException e) {
    /* Exception is expected to be thrown. */
  }
}
```

VTSE thrown as expected

VTSE not thrown

Console Tester

```java
public class CounterTester1 {
  public static void main(String[] args) {
    Counter c = new Counter();
    println("Init val: " + c.getValue());
    try {
      c.decrement();
      println("Error: ValueTooSmallException NOT thrown.");
    }
    catch (ValueTooSmallException e) {
      println("Success: ValueTooSmallException thrown.");
    }
  } /* end of main method */
} /* end of class CounterTester1 */
```

VTSE thrown

VTSE not thrown

WRITTEN TEST I
REVIEW SESSION
MONDAY SEPTEMBER 30

"F" String literal

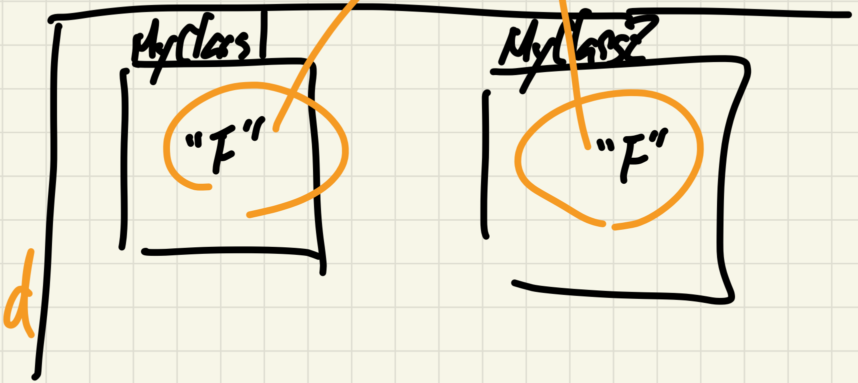
there's only a single object created for this literal

==

new String("F") anonymous object

every time a new object is created

a single object



# Anonymous Objects

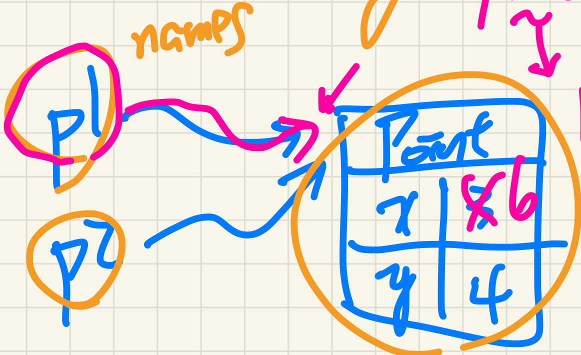
objects for which you do not store their addresses in variables

```
Point p1 = new Point(3, 4);
```

object in memory

name of variable storing that object's address

```
Point p2 = p1;
```



p1.setX(6)

look up address stored in p1.

```
class Point {
    Point newPoint(int x, int y) {
        Point np = new Point(x, y);
        return np;
    }
}
```

return the address of some Point object

no method calls on np.

When to use a.o.?

↳ When you only want to pass its address, without calling any methods on it.

```
return new Point(x, y);
```

Exceptions: Error handling

↳ force caller to “handle” errors when they occur.

catch-or-specify req.

```
class Account {
  int balance;
  String name; ①
  void withdraw(int a) throws NAE {
    if (a < 0 || a > balance) {
      throw new NAE("Error: neg. amount");
    }
  }
}
```

```
Account(String ② name) {
  ~~name~~ = name;
  this.name
}
```

header

very original source of NAE

has the potential of throwing an exception

```
class NAE extends Exception {
  :
}
```

1. Use eclipse to generate

✓ 2. Type this new class

catch option → usually not taken by the callee throwing the exception.

```
void withdraw( . . . ){
  try{ if(...){ throw new NAE(...)}
  } catch (NAE ...){ - - - }
}
```

```
class Client {
    Account acc;
```

```
Client(String name){
    this.acc = new Account(name);
}
```

```
    /* charge the account of this client by amount 'a' */
    void pay(int a){
        acc.withdraw(a);
    }
```

→ unhandle Exception

NAE

In order to see what methods we can call on "acc", go to the declare type

Specify → convenient for the current caller

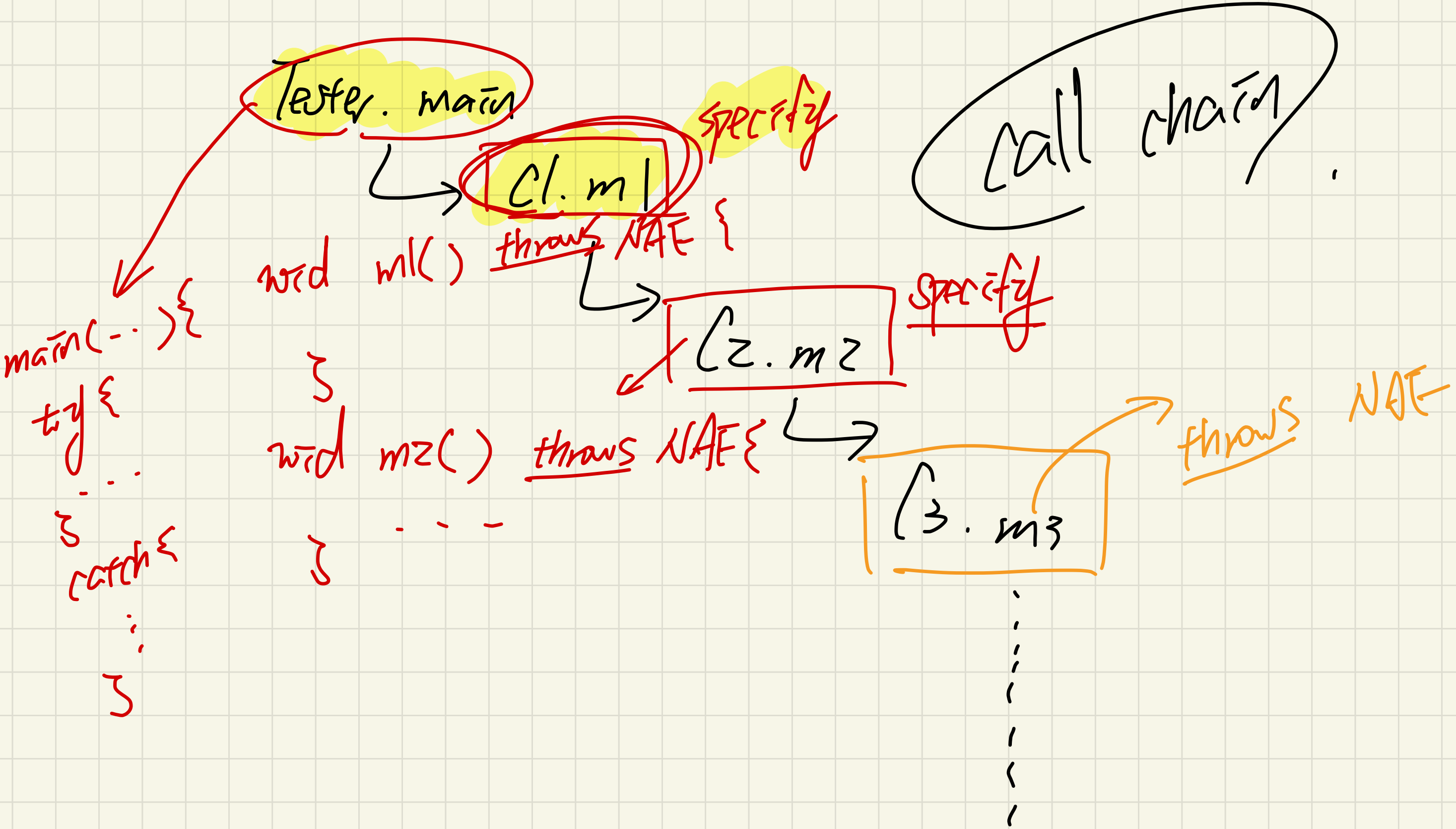
catch

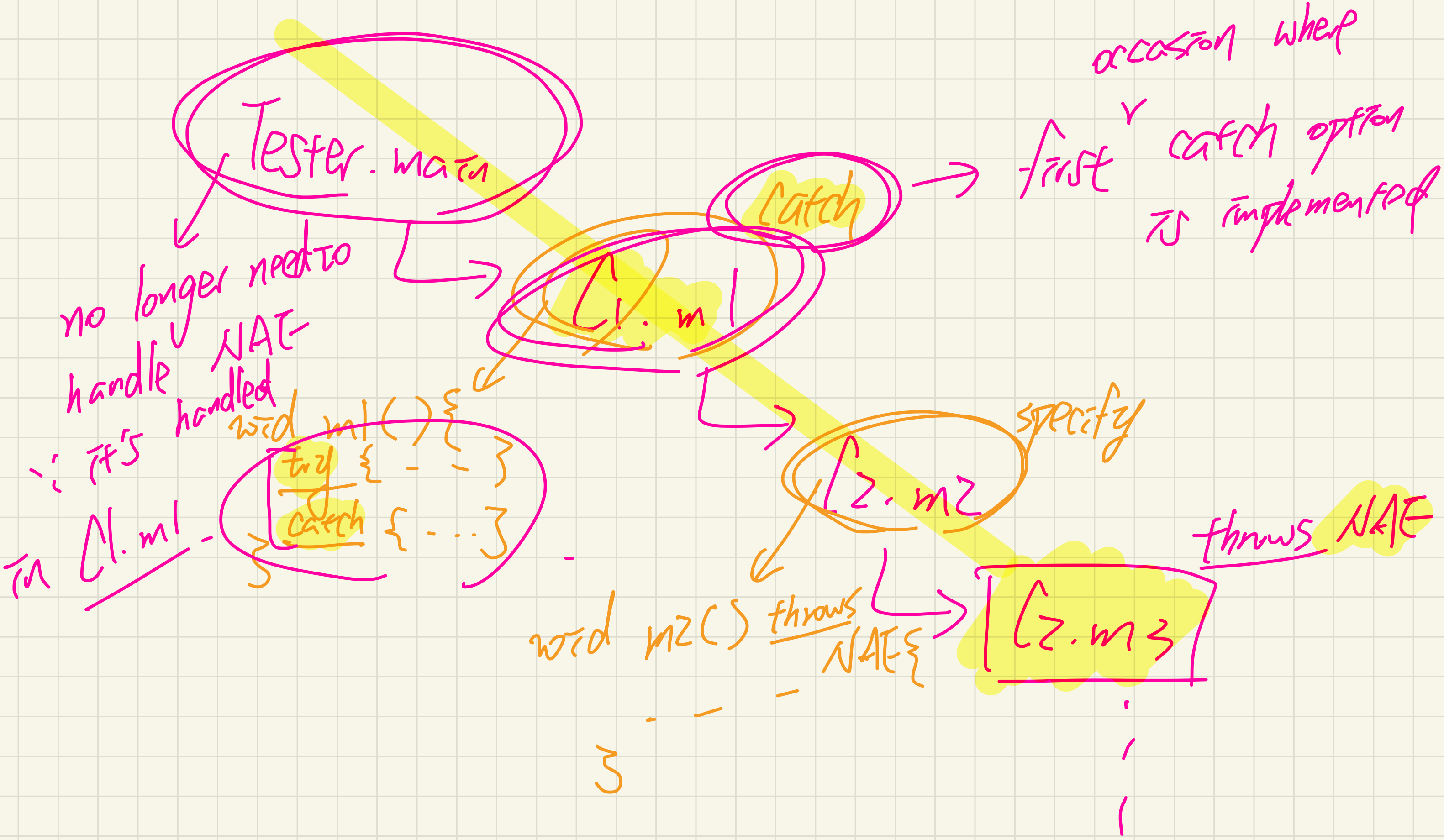
```
void pay(int a) throws NAE {
    acc.withdraw(a)
}
```

any caller of Client.pay must handle NAE

any caller of client.pay does not have to handle

```
void pay(int a){
    try{
        acc.withdraw(a)
    }catch (NAE e){ ... }
}
```


Tester. main
C1. m1
specify
call chain.
main(..){
try{
...
}
catch{
...
}
void m1() throws NAE {
}
C2. m2
specify
void m2() throws NAE{
- - -
}
C3. m3
throws NAE


Tester.main
C1.m1
catch
first occasion where catch option is implemented
no longer need to handle NAE ∵ it's handled in C1.m1
void m1() {
try { - - }
catch { . . . }
}
C2.m2
specify
void m2() throws NAE {
- - -
}
C2.m3
throws NAE

throws ___ , ___ , ___

C1.m1

```
void m1() throws NAE {
    C2 o = new C2();
    o.m2();
}
```

redundant X

```
try {
    o.m2();
} catch (NAE .. ) { ... }
```

C2.m2

```
void m2() throws NAE
{
    if (..) {
        throw new NAE(...);
    }
}
```

```
void m1 ( ) throws E1, E2 {
    if ( ... ) { throw new E1(..); }
    if ( ... ) { throw new E2(...); }
    :
}
```

Q7

Person[] persons = {p1, p2};

→ does not store Person addresses

Person[] → Array of

Person → Store address of a Person object

persons → Stores the starting address of an array

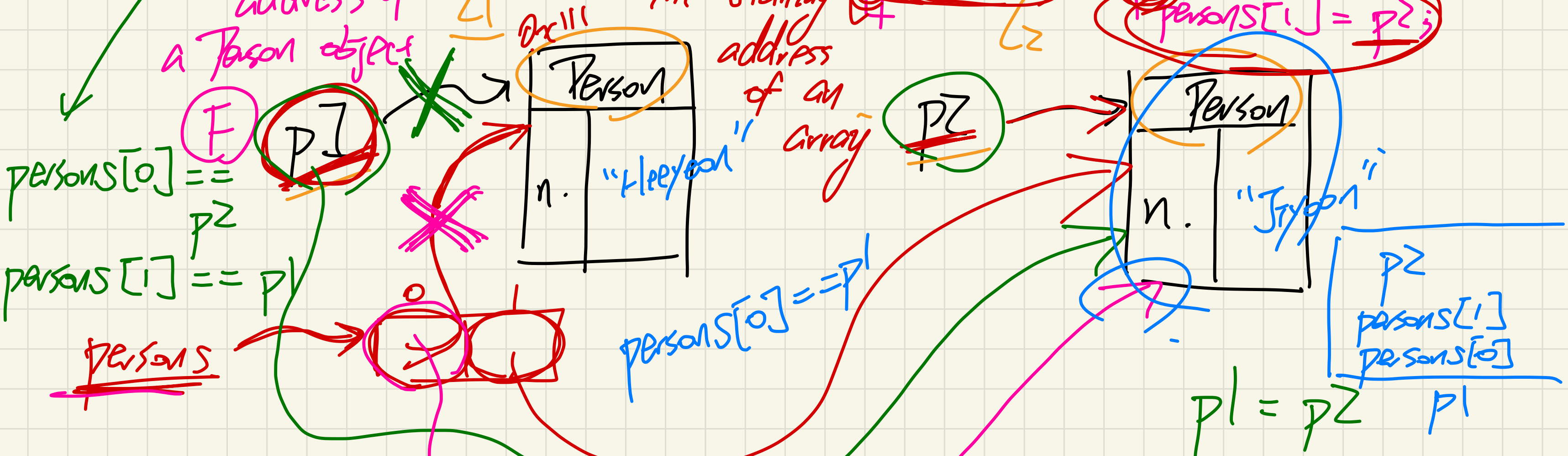
Person[] persons = new Person[2];

persons[0] = p1;

persons[1] = p2;

L1

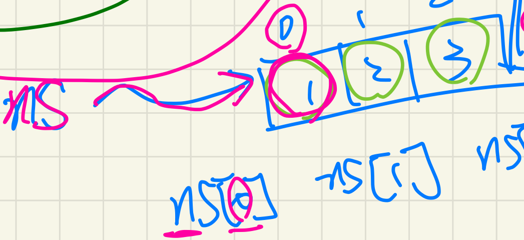
L2



persons[0] = p2

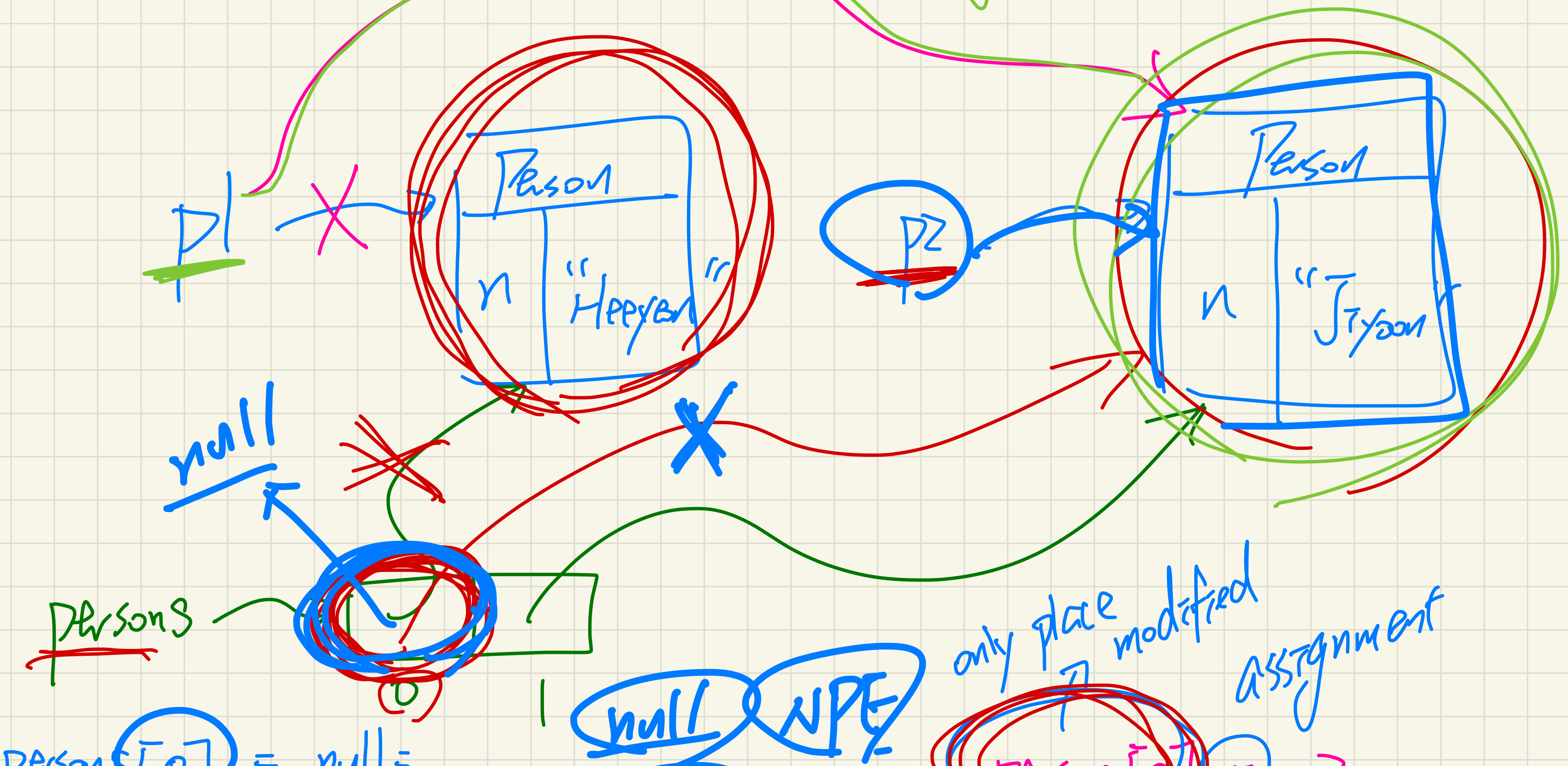
int[] ns = {1, 2, 3};

ns → store starting address of the array



p1 = p2

p1.getName()



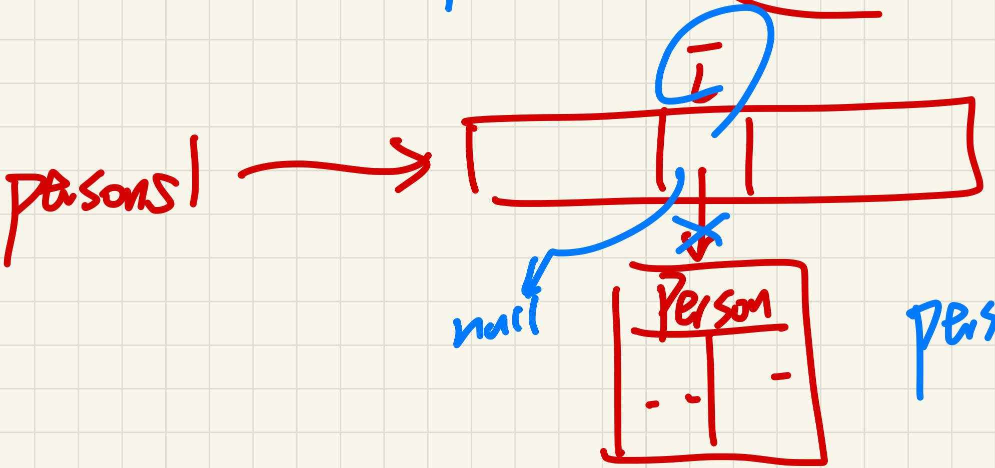
persons[0] = null ③

① p2.getName() Jryoon ② persons[0].getName()

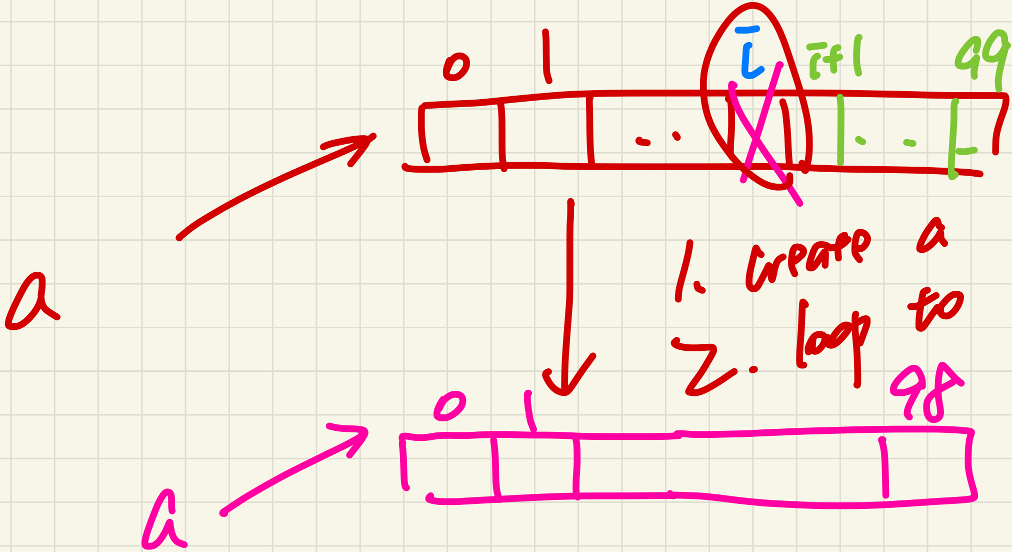
persons[0] = p2

Person[] persons1 = { - - - };

Persons[] pesons2 = new Person[3];



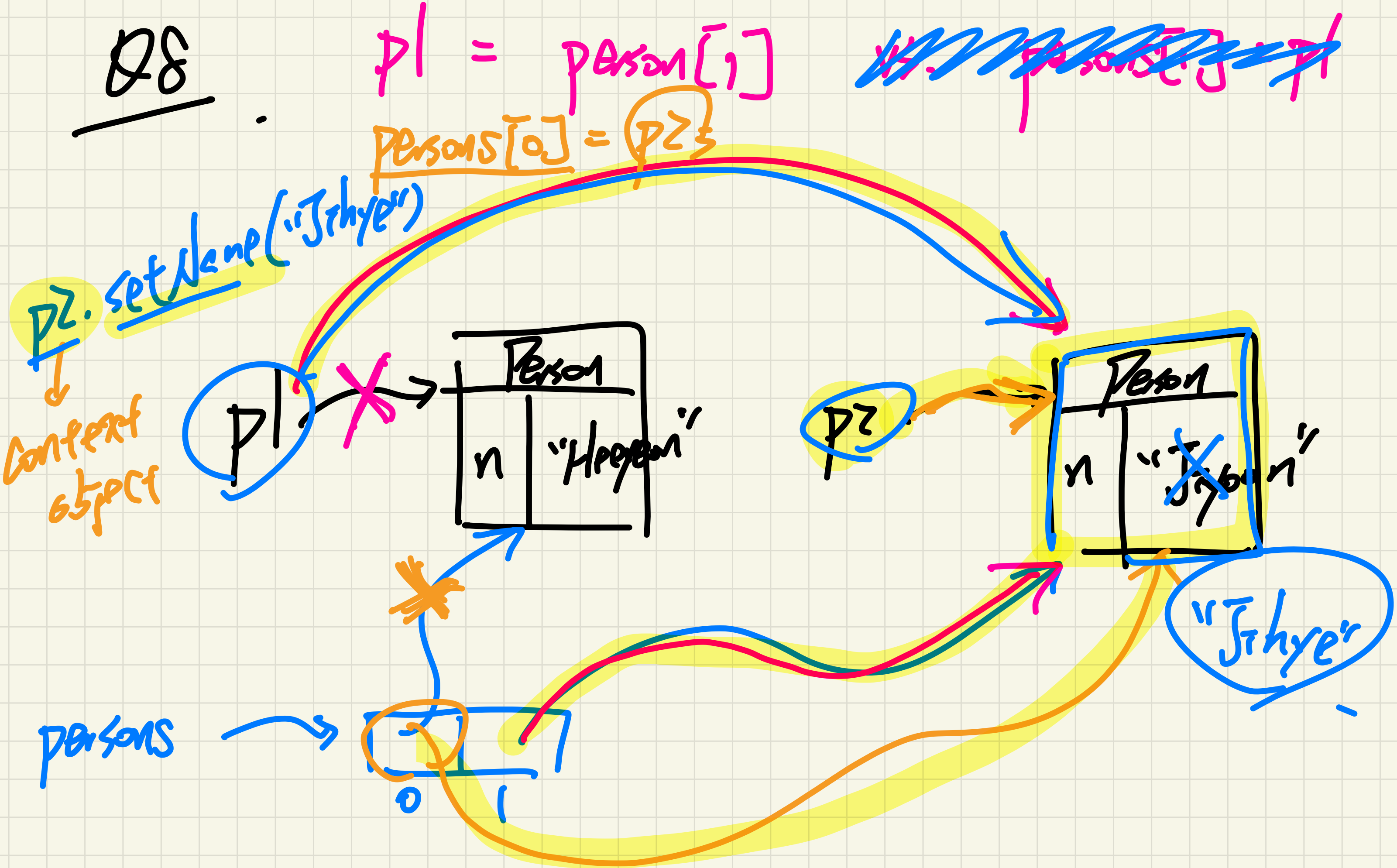
persons1[i] = null;



1. create a new, smaller array

2. loop to copy every el. except a[i] into new array.

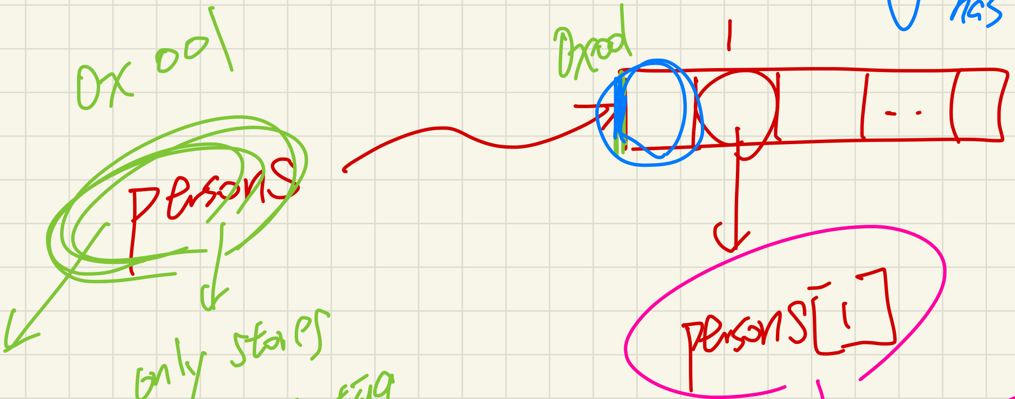
## Q8



p1 = person[1]
~~persons[1] = p1~~
persons[0] = p2
p2.setName("Jihye")
Context object
p1
Person
n
"Hayoon"
p2
Person
n
"~~Jiyoon~~"
"Jihye"
persons
0
1

Person[ ] persons = - - -

Starting address of the array has the same value as persons[0]



persons → does not store any Person object's address

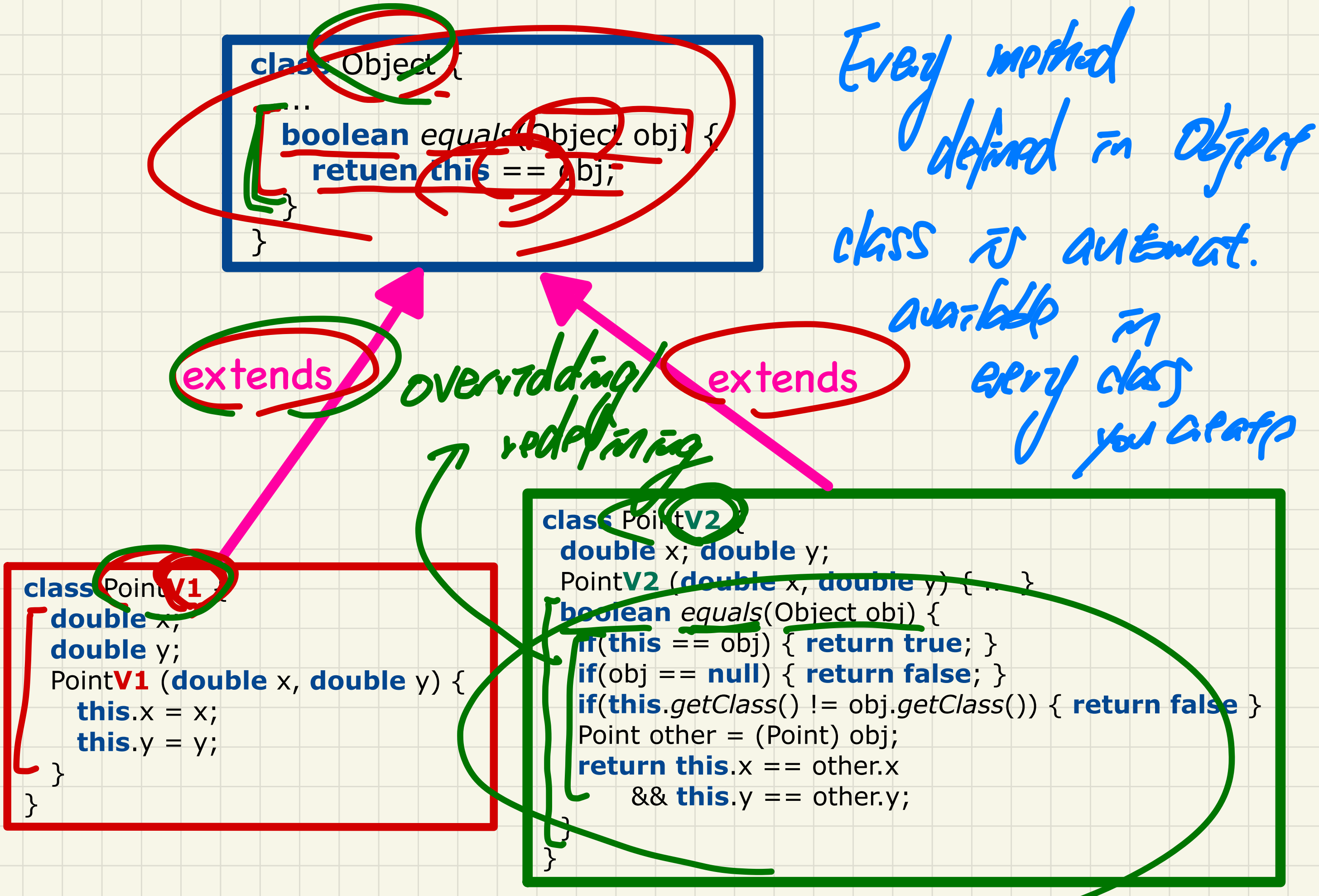
persons → only stores the starting address of the array

persons[1] → stores a Person object's address

# LECTURE 8

MONDAY SEPTEMBER 30

# The equals Method: To Override or Not?



```
class Object {
  ...
  boolean equals(Object obj) {
    retuen this == obj;
  }
}
```

```
class PointV1 {
  double x;
  double y;
  PointV1 (double x, double y) {
    this.x = x;
    this.y = y;
  }
}
```

```
class PointV2 {
  double x; double y;
  PointV2 (double x, double y) { ... }
  boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false }
    Point other = (Point) obj;
    return this.x == other.x
        && this.y == other.y;
  }
}
```

Every method defined in Object class is automat. available in every class you create

# The equals Method: Default Version

[redacted]

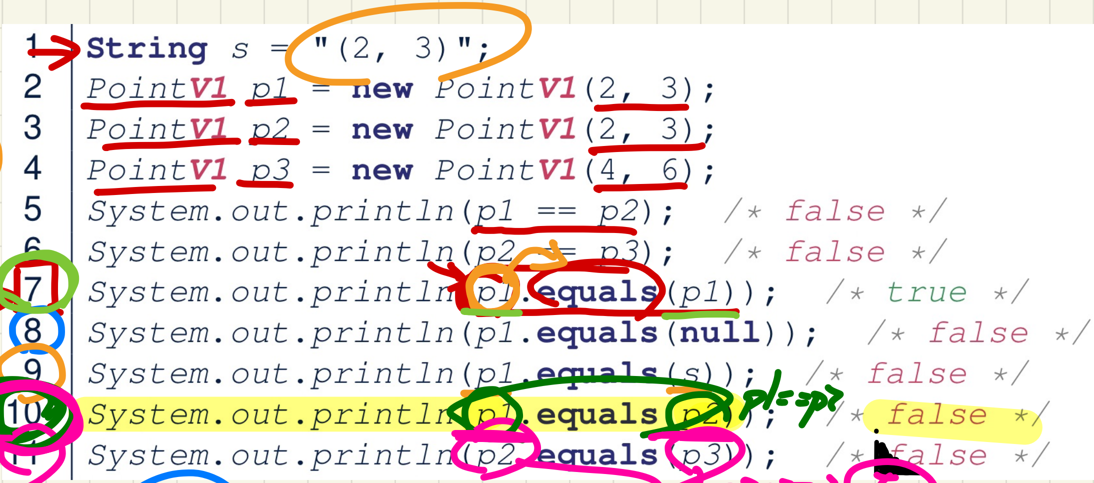
```
class Object {
  ...
  boolean equals(Object obj) {
    return this == obj;
  }
}
```

p1 p1 null

p1 == null

extends

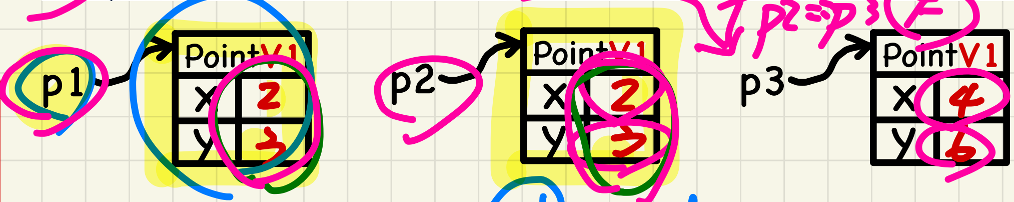
```
class PointV1 {
  double x;
  double y;
  PointV1 (double x, double y) {
    this.x = x;
    this.y = y;
  }
}
```



```
String s = "(2, 3)";
PointV1 p1 = new PointV1(2, 3);
PointV1 p2 = new PointV1(2, 3);
PointV1 p3 = new PointV1(4, 6);
System.out.println(p1 == p2);   /* false */
System.out.println(p2 == p3);   /* false */
System.out.println(p1.equals(p1));   /* true */
System.out.println(p1.equals(null));   /* false */
System.out.println(p1.equals(s));   /* false */
System.out.println(p1.equals(p2));   /* false */
System.out.println(p2.equals(p3));   /* false */
```

p1!=p2

p2==p3 F



p1.equals(null) F

↳

C.O.

p1.equals(p1)

p1.equals(s)

↳ p1 == s F

p1.equals(p2) F

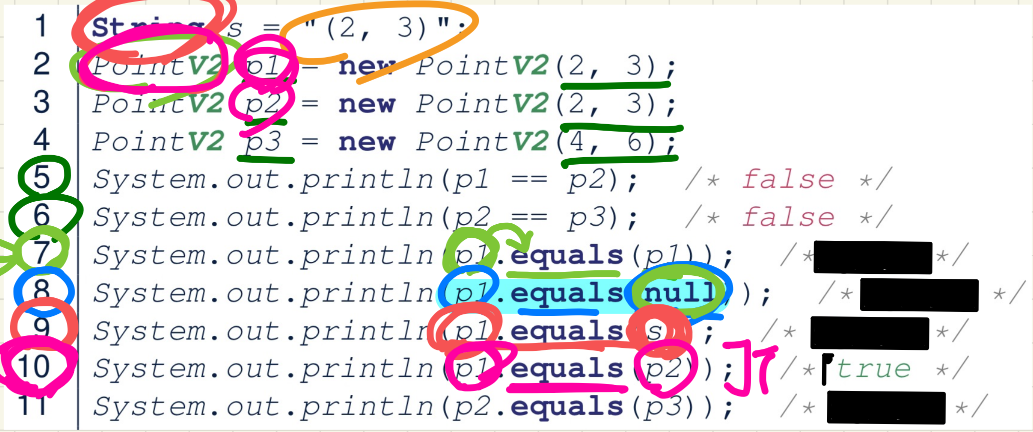
# The equals Method: Overridden Version    Example 1

[redacted]

```
class Object {
  ...
  boolean equals(Object obj) {
    retuen this == obj;
  }
}
```

extends

```
class PointV2 {
  double x; double y;
  PointV2 (double x, double y) { ... }
  boolean equals(Object obj) { null
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false }
    Point other = (Point) obj;
    return this.x == other.x
        && this.y == other.y;
  }
}
```

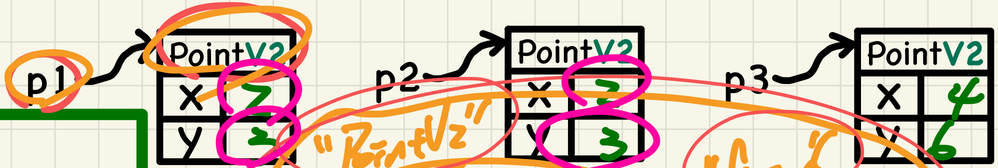


```
1  String s = "(2, 3)";
2  PointV2 p1 = new PointV2(2, 3);
3  PointV2 p2 = new PointV2(2, 3);
4  PointV2 p3 = new PointV2(4, 6);
5  System.out.println(p1 == p2);       /* false */
6  System.out.println(p2 == p3);       /* false */
7  System.out.println(p1.equals(p1));    /* [redacted] */
8  System.out.println(p1.equals(null));  /* [redacted] */
9  System.out.println(p1.equals(s));     /* [redacted] */
10 System.out.println(p1.equals(p2));    /* true */
11 System.out.println(p2.equals(p3));    /* [redacted] */
```

T

<7: p1 == p1

<8: null == null T



"PointV2" "String"

<9: p1.getClass() != s.getClass()

dynamic type

T

p1.x == p2.x
&& p1.y == p2.y

p1.equals(p1)
↳ type of c.o. p1: PointV2
∴ equals redefined ∴ redefined version called

PointV1 p1 = new . — -

PointV2 p2 = new . —

p1.equals(p2)

C.O. "PointV1"  "PointV2"

```
class PointV2 {
  double x; double y;
  PointV2 (double x, double y) { ... }
  boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false }
    Point other = (Point) obj;
    return this.x == other.x
        && this.y == other.y;
  }
}
```

F
F
F

PointV2

this != obj

obj != null

this.getClass() == obj.getClass()

all true if you can reach *

```
class PointV2 {
  double x; double y;
  PointV2 (double x, double y) { ... }
  boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false }
    PointV2 other = (PointV2) obj;
    return this.x == other.x
        && this.y == other.y;
  }
}
```

Object

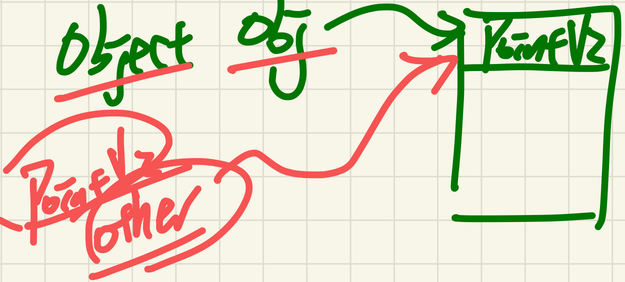
obj

obj. has the declare type Object.

this and obj are same type

V1

V2 [ return
this.x == obj.x
&&
this.y == obj.y



Java compiler only allows attributes/methods defined in the declared type of obj.
Object

```
class PointV2 {
 double x; double y;
 PointV2 (double x, double y) { ... }
 boolean equals(Object obj) {
   if(this == obj) { return true; }
   if(obj == null) { return false; }
   if(this.getClass() != obj.getClass()) { return false }
   PointV2 other = (PointV2) obj;
   return this.x == other.x
        && this.y == other.y;
 }
}
```

p1 → null

PointV2 p1 = new ...

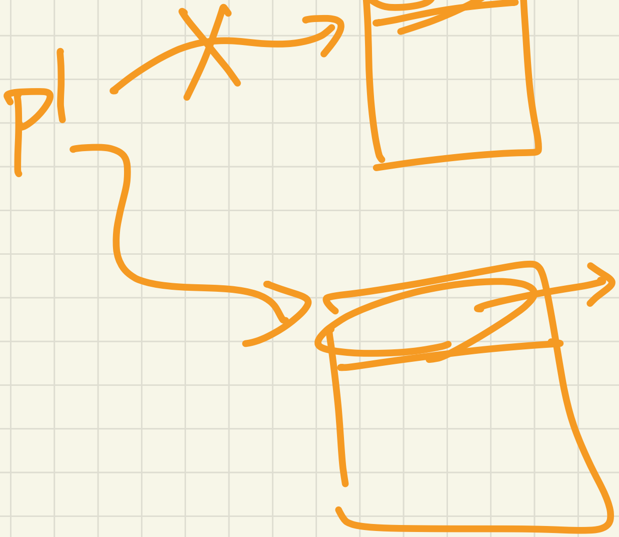
p1 = null;

p1.equals("junk")

look up where the object pointed by p1 is.

```
      p1
if (this == null) { return ? ; }
```

↳ redundat ∵ a NPE would've occured already.



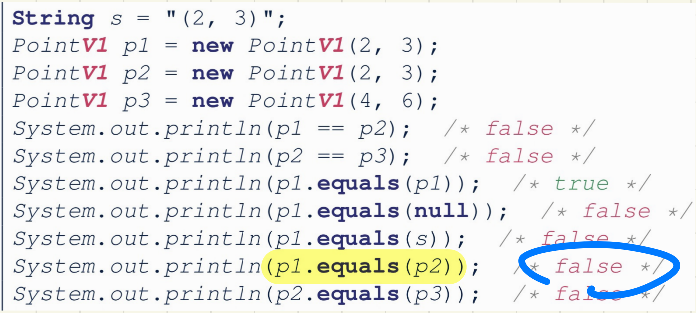
this type may be differe

o. getClass()

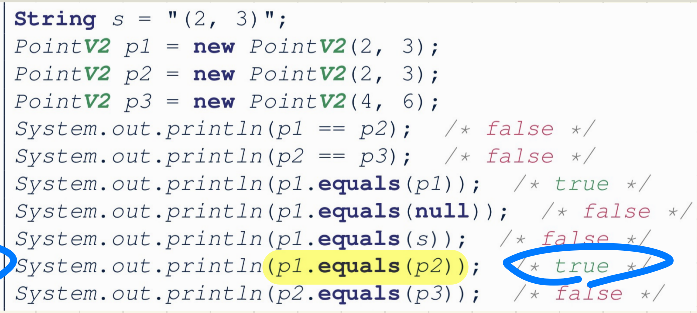
returns the type of object pointed by o currently.

# The equals Method:

# To Override or Not?



```
String s = "(2, 3)";
PointV1 p1 = new PointV1(2, 3);
PointV1 p2 = new PointV1(2, 3);
PointV1 p3 = new PointV1(4, 6);
System.out.println(p1 == p2);   /* false */
System.out.println(p2 == p3);   /* false */
System.out.println(p1.equals(p1));   /* true */
System.out.println(p1.equals(null));   /* false */
System.out.println(p1.equals(s));   /* false */
System.out.println(p1.equals(p2));   /* false */
System.out.println(p2.equals(p3));   /* false */
```



```
String s = "(2, 3)";
PointV2 p1 = new PointV2(2, 3);
PointV2 p2 = new PointV2(2, 3);
PointV2 p3 = new PointV2(4, 6);
System.out.println(p1 == p2);   /* false */
System.out.println(p2 == p3);   /* false */
System.out.println(p1.equals(p1));   /* true */
System.out.println(p1.equals(null));   /* false */
System.out.println(p1.equals(s));   /* false */
System.out.println(p1.equals(p2));   /* true */
System.out.println(p2.equals(p3));   /* false */
```

```
class Object {
  ...
  boolean equals(Object obj) {
    retuen this == obj;
  }
}
```

extends

extends

```
class PointV1 {
  double x;
  double y;
  PointV1 (double x, double y) {
    this.x = x;
    this.y = y;
  }
}
```

```
class PointV2 {
  double x; double y;
  PointV2 (double x, double y) { ... }
  boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false }
    Point other = (Point) obj;
    return this.x == other.x
        && this.y == other.y;
  }
}
```

```
class PointV2 {
  double x; double y;
  PointV2 (double x, double y) { ... }
  boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false }
    Point other = (Point) obj;
    return this.x == other.x
        && this.y == other.y;
  }
}
```

p2.equals(p3)

p2.x    p3.x

# The equals Method: Overridden Version

## Example 2

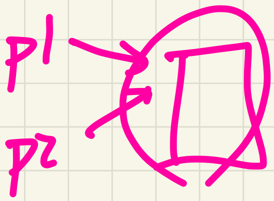
[redacted]

```
PointV2 p1 = new PointV2(3, 4);
PointV2 p2 = new PointV2(3, 4);
PointV2 p3 = new PointV2(4, 5);
System.out.println(p1 == p1);   /* [redacted] */
System.out.println(p1.equals(p1));   /* [redacted] */
System.out.println(p1 == p2);   /* [redacted] */
System.out.println(p1.equals(p2));   /* [redacted] */
System.out.println(p2 == p3);   /* [redacted] */
System.out.println(p2.equals(p3));   /* [redacted] */
```

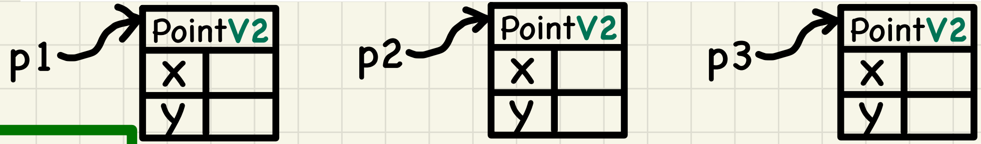
8 8

```
class Object {
  ...
  boolean equals(Object obj) {
    retuen this == obj;
  }
}
```

extends



```
class PointV2 {
  double x; double y;
  PointV2 (double x, double y) { ... }
  boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false }
    Point other = (Point) obj;
    return this.x == other.x
          && this.y == other.y;
  }
}
```



(A) Two objects are reference-equal.

(B) Two objects are contents-equal.

- If (A) is true, then (B) is true.

p1==p2    p1.equals(p2)

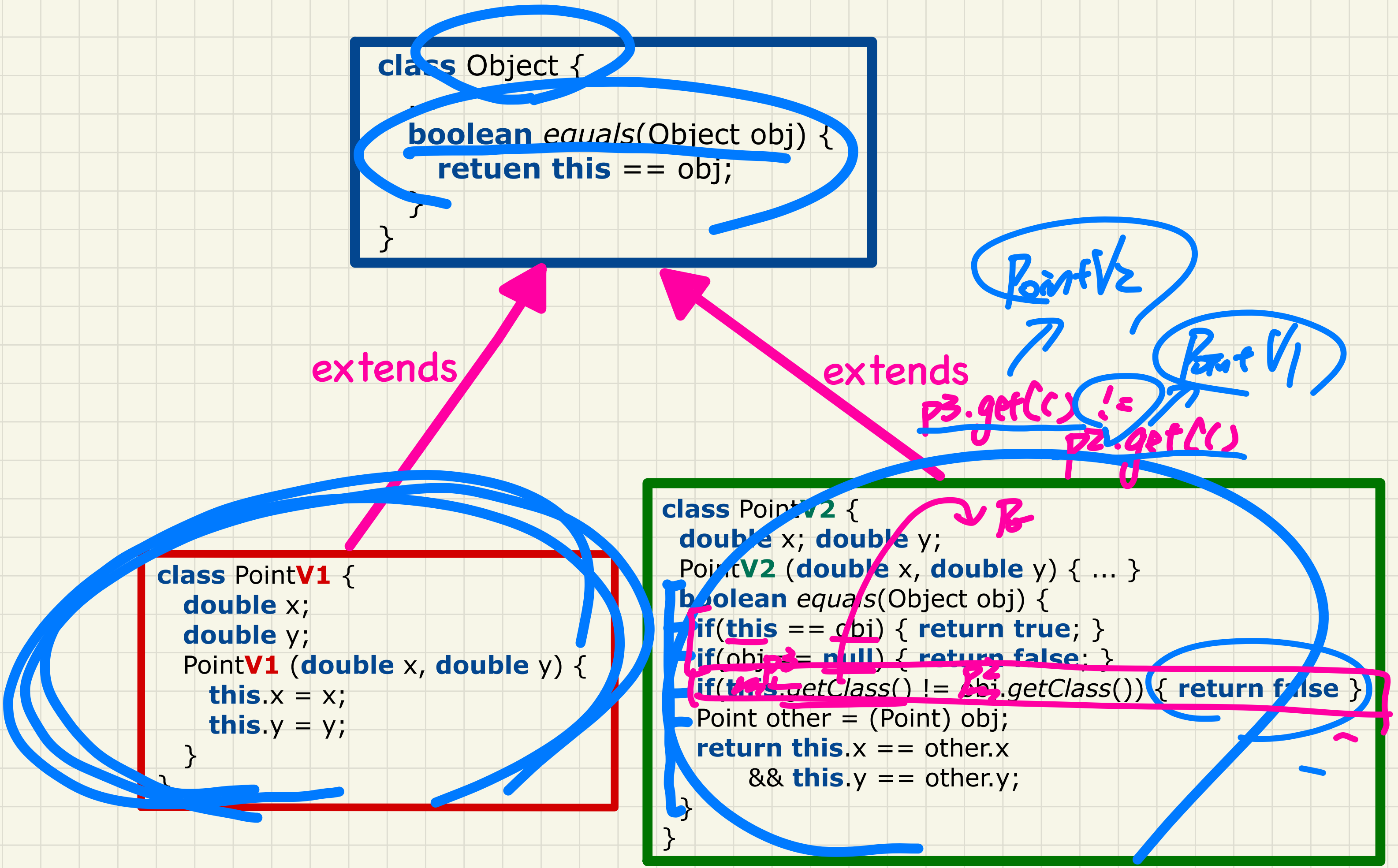
- If (B) is true, then (A) is true.

p1.equals(p2)    p1 == p2

Lecture 9

Wednesday October 2

# The equals Method: To Override or Not?



```
class Object {
  boolean equals(Object obj) {
    retuen this == obj;
  }
}
```

```
class PointV1 {
  double x;
  double y;
  PointV1 (double x, double y) {
    this.x = x;
    this.y = y;
  }
}
```

```
class PointV2 {
  double x; double y;
  PointV2 (double x, double y) { ... }
  boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false }
    Point other = (Point) obj;
    return this.x == other.x
        && this.y == other.y;
  }
}
```

c.o.

assertEquals(obj1, obj2)

↳ obj1.equals(obj2)

c.o

assetEquals(obj2, obj1)

↳ obj2.equals(obj1)

[redacted]

assertE(p3, p2)

↳ p3.equals(p2) → calling overridden version

not the same type

**assertEquals**(exp1, exp2)

- ≈ exp1.**equals**(exp2) if exp1 and exp2 are **reference** type

**Case 1:** If equals is not explicitly overridden in *obj*1's declared type
≈ **assertSame**(obj1, obj2)

PointV1 does not have equals redefined

```
PointV1 p1 = new PointV1(3, 4);
PointV1 p2 = new PointV1(3, 4);
PointV2 p3 = new PointV2(3, 4);
assertEquals(p1, p2);  × /* ∵ different PointV1 objects */
assertEquals(p2, p3);  × /* ∵ different types of objects */
```

p1 == p2

V1 p2.equals(p3) V2 → p2 == p3 F

**Case 2:** If equals is explicitly overridden in *obj*1's declared type
≈ obj1.**equals**(obj2)

PointV1 has default version

p2.equals(p3) → p2 == p3 ×

```
PointV1 p1 = new PointV1(3, 4);
PointV1 p2 = new PointV1(3, 4);
PointV2 p3 = new PointV2(3, 4);
assertEquals(p1, p2);  × /* ≈ [redacted]
assertEquals(p2, p3);  × /* ≈
assertEquals(p3, p2);  × /* ≈ [redacted]
```

equals (Object obj)

PointV1 p1 = new ...

PointV1 p2 = new ...

PointV2 p3 = new ...

at runtime
↳ p2 == p3

① p1 == p2 ✓ ∵ same type

ref type

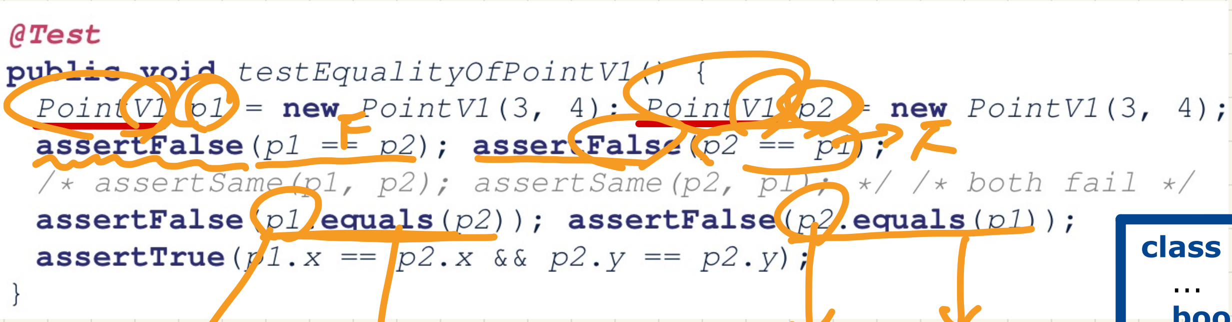
② p1.equals(p2) → PointV1 which is an object

③ p2 == p3 ✗ not compile ∵ diff. types.

p2 is a ref. type.

④ p2.equals(p3) → every ref type is an object

# Testing **Default** Equality of **Points** in JUnit



```
@Test
public void testEqualityOfPointV1() {
  PointV1 p1 = new PointV1(3, 4); PointV1 p2 = new PointV1(3, 4);
  assertFalse(p1 == p2); assertFalse(p2 == p1);
  /* assertSame(p1, p2); assertSame(p2, p1); */ /* both fail */
  assertFalse(p1.equals(p2)); assertFalse(p2.equals(p1));
  assertTrue(p1.x == p2.x && p2.y == p2.y);
}
```

F

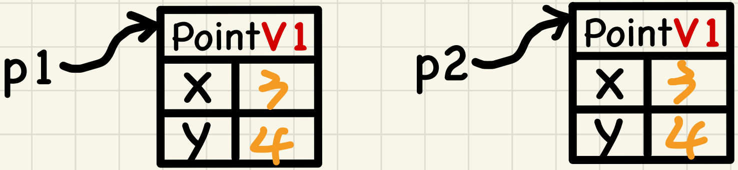
T

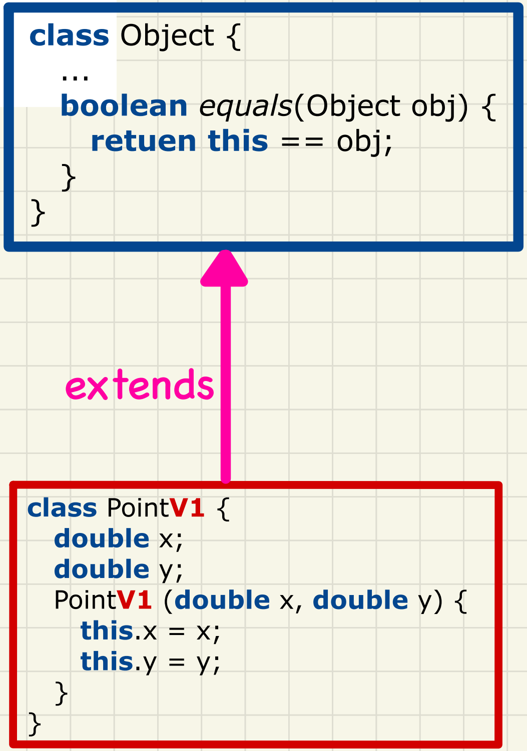
default

p1==p2

default

p2==p1

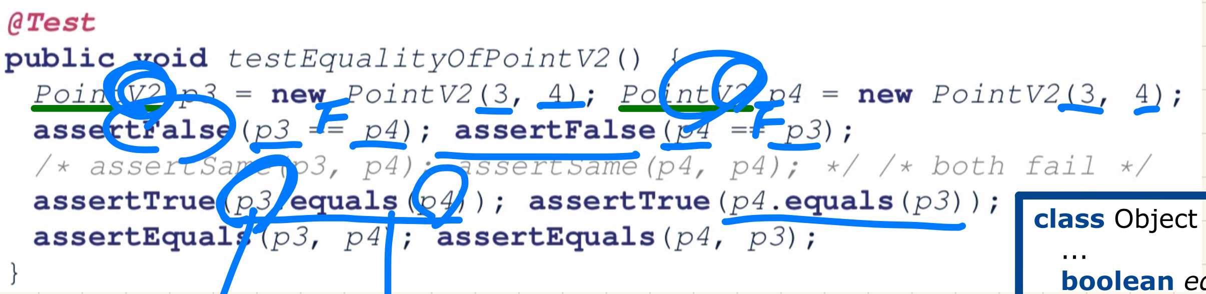




```
class Object {
  ...
  boolean equals(Object obj) {
    retuen this == obj;
  }
}
```

```
class PointV1 {
  double x;
  double y;
  PointV1 (double x, double y) {
    this.x = x;
    this.y = y;
  }
}
```

# Testing Overridden Equality of Points in JUnit

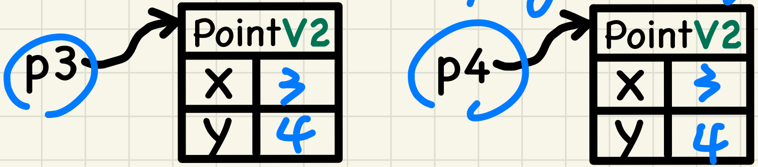


```
@Test
public void testEqualityOfPointV2() {
  PointV2 p3 = new PointV2(3, 4); PointV2 p4 = new PointV2(3, 4);
  assertFalse(p3 == p4); assertFalse(p4 == p3);
  /* assertSame(p3, p4); assertSame(p4, p4); */ /* both fail */
  assertTrue(p3.equals(p4)); assertTrue(p4.equals(p3));
  assertEquals(p3, p4); assertEquals(p4, p3);
}
```

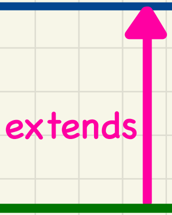
F F

overridden

p3.x == p4.x &&
p3.y == p4.y

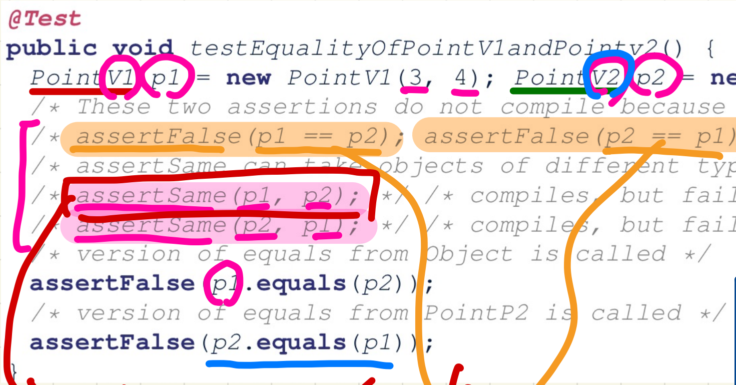


```
class Object {
  ...
  boolean equals(Object obj) {
    retuen this == obj;
  }
}
```



```
class PointV2 {
  double x; double y;
  PointV2 (double x, double y) { ... }
  boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false }
    Point other = (Point) obj;
    return this.x == other.x
        && this.y == other.y;
  }
}
```

# Testing Equality of Points in JUnit



```java
@Test
public void testEqualityOfPointV1andPointv2() {
  PointV1 p1 = new PointV1(3, 4); PointV2 p2 = new PointV2(3, 4);
  /* These two assertions do not compile because p1 and p2 are of different types. */
  /* assertFalse(p1 == p2); assertFalse(p2 == p1); */
  /* assertSame can take objects of different types and fail. */
  /* assertSame(p1, p2); */ /* compiles, but fails */
  /* assertSame(p2, p1); */ /* compiles, but fails */
  /* version of equals from Object is called */
  assertFalse(p1.equals(p2));
  /* version of equals from PointP2 is called */
  assertFalse(p2.equals(p1));
}
```

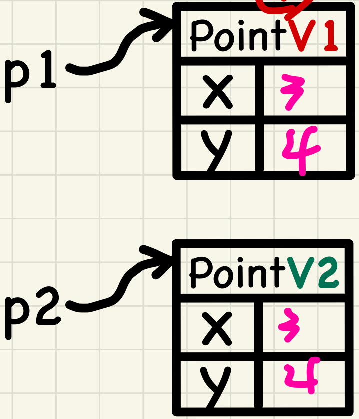
p1.equals(p2) → default → p1 == p2

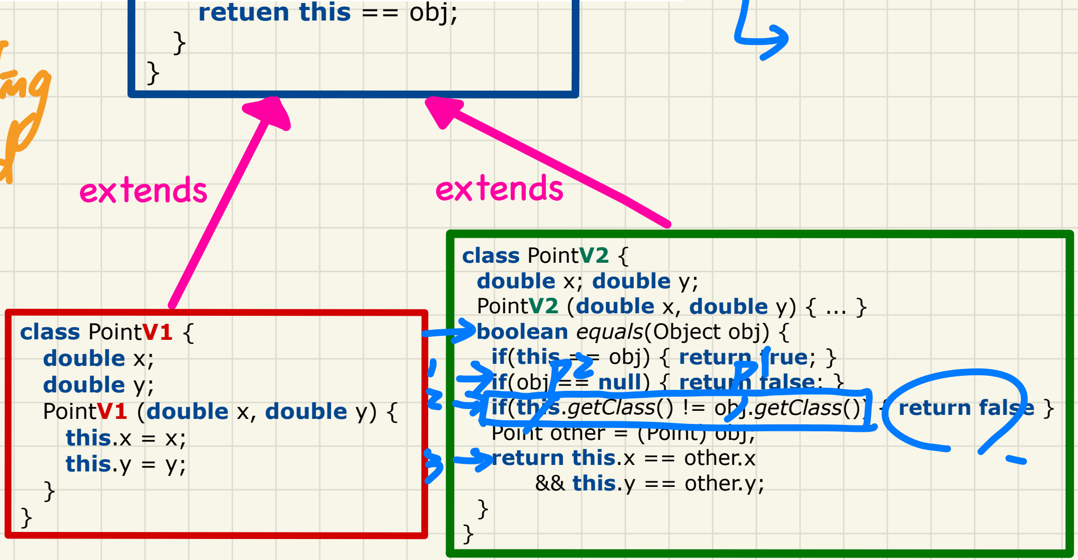
p2.equals(p1) → Overridden →

p1 == p2 ✓ compile

F

not compiling ∵ p1 and p2 different types





```java
class Object {
  ...
  boolean equals(Object obj) {
    retuen this == obj;
  }
}
```

```java
class PointV1 {
  double x;
  double y;
  PointV1 (double x, double y) {
    this.x = x;
    this.y = y;
  }
}
```

```java
class PointV2 {
  double x; double y;
  PointV2 (double x, double y) { ... }
  boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false }
    Point other = (Point) obj;
    return this.x == other.x
        && this.y == other.y;
  }
}
```

```
PointV1 p1 = new PointV1(3, 4);
PointV2 p2 = new PointV2(3, 4);
```

Known:

p1.equals(p2) → F (diffrent addresses)

p2.equals(p1) → F (different types)

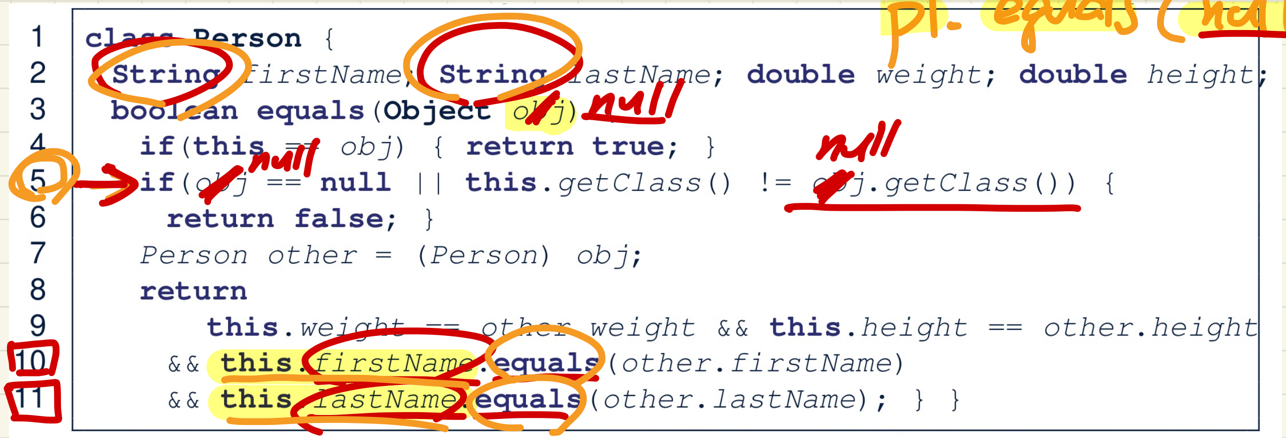
```
boolean CompareV1V2(PointV1 p1, PointV2 p2){

    return p1.x == p2.x
        && p1.y == p2.y;
}
```

```
class PointV2 {
  double x; double y;
  PointV2 (double x, double y) { ... }
  boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false }
    Point other = (Point) obj;
    return this.x == other.x
          && this.y == other.y;
  }
}
```

```
if (   obj == null   ||   this.getC != obj.getC){
    return  false ;
  }
```

# Exercise: Two Persons are equal if their names and measures are equal

p1.equals(null)



```
1  class Person {
2   String firstName; String lastName; double weight; double height;
3   boolean equals(Object obj) {
4     if(this == obj) { return true; }
5     if(obj == null || this.getClass() != obj.getClass()) {
6       return false; }
7     Person other = (Person) obj;
8     return
9         this.weight == other.weight && this.height == other.height
10      && this.firstName.equals(other.firstName)
11      && this.lastName.equals(other.lastName); } }
```

null

null

null

**Q1: At Lines 10 and 11 which version of the equals method is called?**

**Q2: At Line 5, will there be a NullPointerException if obj == null?**

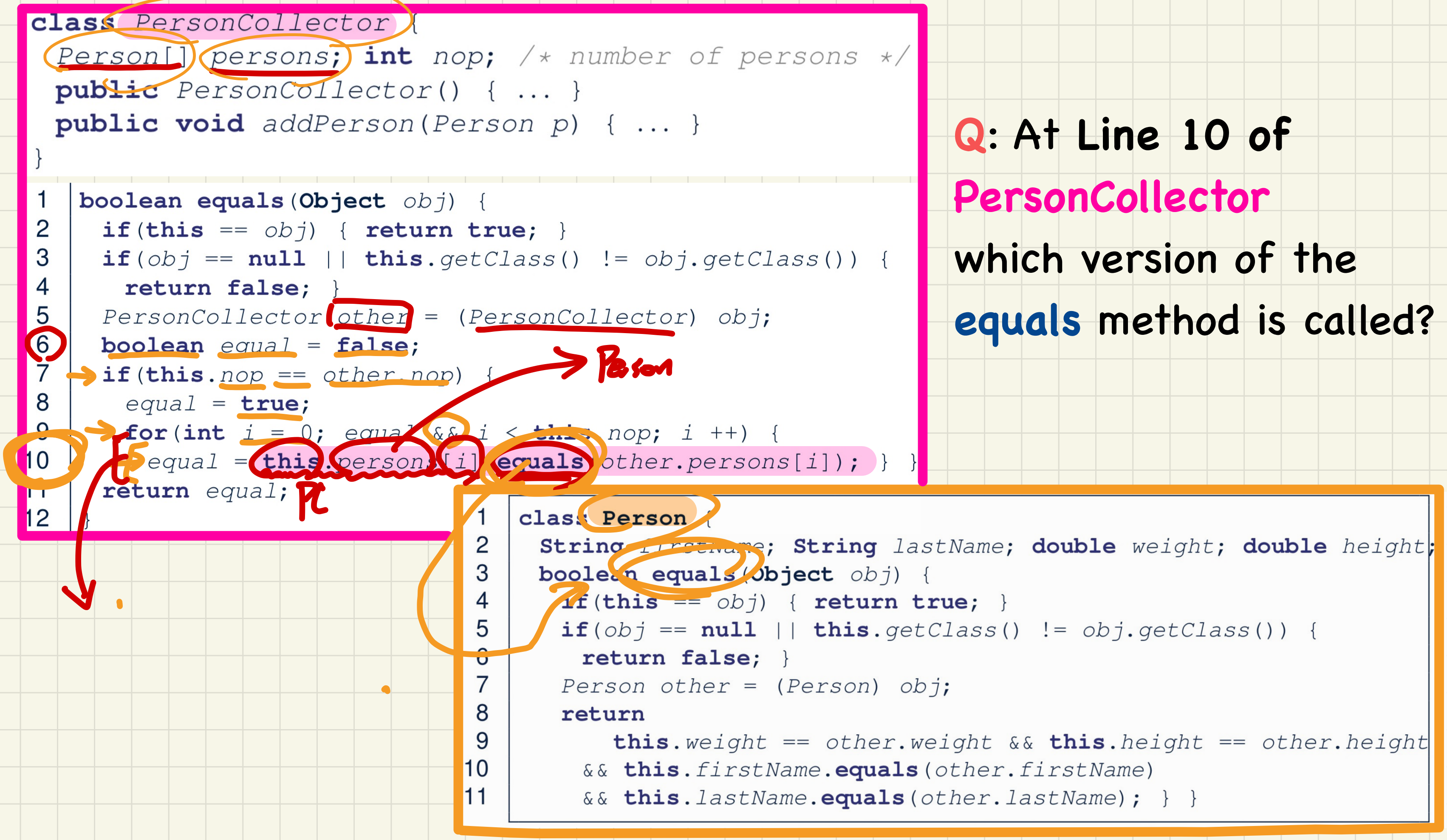
**Q3: At Line 5, what if we change it to:**

**if(this.getClass() != obj.getClass() || obj == null)**

NPE.

null

# Exercise: PersonCollectors are equal if their arrays of persons are equal



```
class PersonCollector {
  Person[] persons; int nop; /* number of persons */
  public PersonCollector() { ... }
  public void addPerson(Person p) { ... }
}
```

```
1  boolean equals(Object obj) {
2    if(this == obj) { return true; }
3    if(obj == null || this.getClass() != obj.getClass()) {
4      return false; }
5    PersonCollector other = (PersonCollector) obj;
6    boolean equal = false;
7    if(this.nop == other.nop) {
8      equal = true;
9      for(int i = 0; equal && i < this.nop; i ++) {
10       equal = this.persons[i].equals(other.persons[i]); } }
11   return equal;
12 }
```

```
1  class Person {
2    String firstName; String lastName; double weight; double height;
3    boolean equals(Object obj) {
4      if(this == obj) { return true; }
5      if(obj == null || this.getClass() != obj.getClass()) {
6        return false; }
7      Person other = (Person) obj;
8      return
9          this.weight == other.weight && this.height == other.height
10       && this.firstName.equals(other.firstName)
11       && this.lastName.equals(other.lastName); } }
```

**Q**: At Line 10 of **PersonCollector** which version of the **equals** method is called?

40

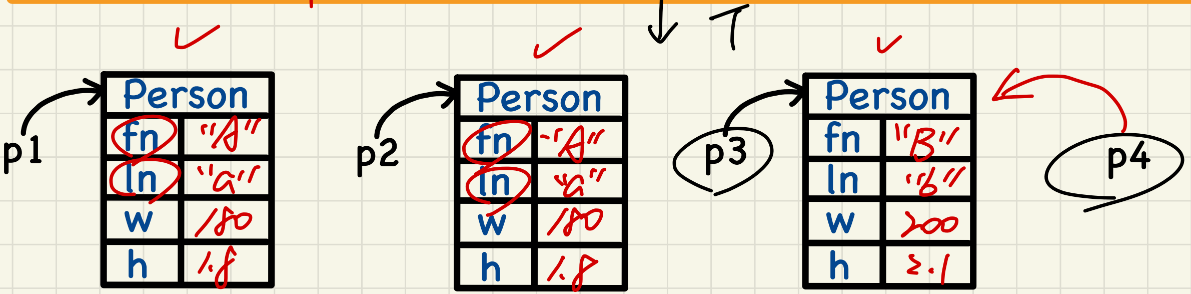
this.pesons[i].equals(othe.pesons[i])

this.pesons[i].weight ==
other.pesons[i].weight

&&

# Testing Equality of Person/PersonCollector in JUnit (1)

```
@Test
public void testPersonCollector() {
  Person p1 = new Person("A", "a", 180, 1.8); Person p2 = new Person("A", "a", 180, 1.8);
  Person p3 = new Person("B", "b", 200, 2.1); Person p4 = p3;
  assertFalse(p1 == p2); assertTrue(p1.equals(p2));
  assertTrue(p3 == p4); assertTrue(p3.equals(p4));
```

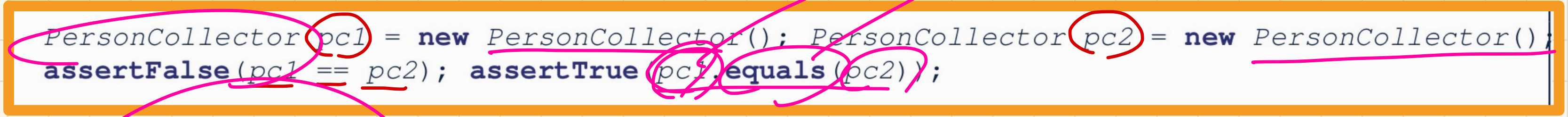


```
class Person {
  String firstName; String lastName; double weight; double height;
  boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null || this.getClass() != obj.getClass()) {
      return false; }
    Person other = (Person) obj;
    return
        this.weight == other.weight && this.height == other.height
      && this.firstName.equals(other.firstName)
      && this.lastName.equals(other.lastName); } }
```

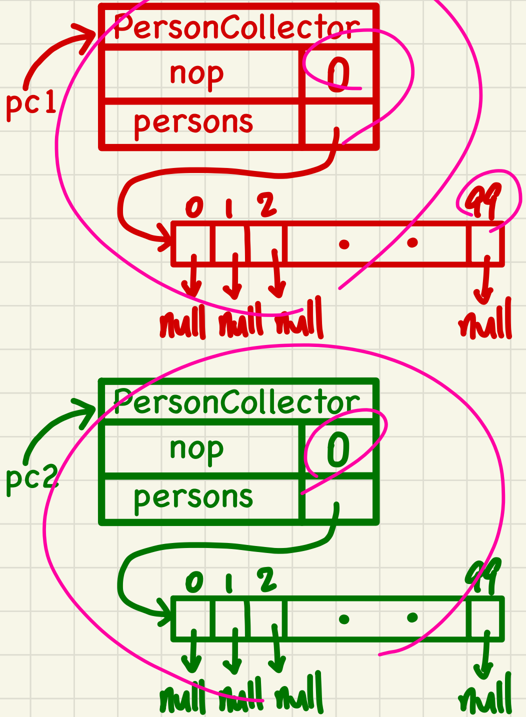
# Testing Equality of Person/PersonCollector in JUnit (2)

## (continued from testPersonCollector)

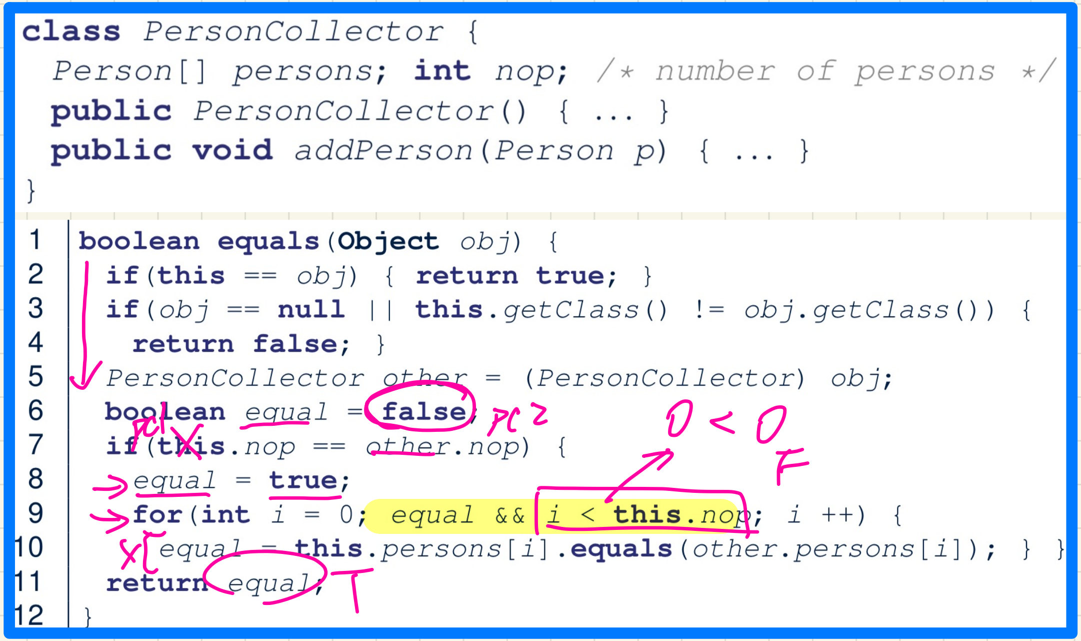
Overridden version



```
PersonCollector pc1 = new PersonCollector(); PersonCollector pc2 = new PersonCollector();
assertFalse(pc1 == pc2); assertTrue(pc1.equals(pc2));
```



**Q**: How about **assertTrue**(pc2.**equals**(pc1))?



```
class PersonCollector {
  Person[] persons; int nop; /* number of persons */
  public PersonCollector() { ... }
  public void addPerson(Person p) { ... }
}
```

```
boolean equals(Object obj) {
  if(this == obj) { return true; }
  if(obj == null || this.getClass() != obj.getClass()) {
    return false; }
  PersonCollector other = (PersonCollector) obj;
  boolean equal = false;
  if(this.nop == other.nop) {
  equal = true;
  for(int i = 0; equal && i < this.nop; i ++) {
    equal = this.persons[i].equals(other.persons[i]); } }
  return equal;
}
```

pc1 pc2

$0 < 0$

F

T

# Testing Equality of Person/PersonCollector in JUnit (3)

(continued from testPersonCollector)

```
pc1.addPerson(p1);
assertFalse(pc1.equals(pc2));

pc2.addPerson(p2);
assertFalse(pc1.persons[0] == pc2.persons[0]);
assertTrue(pc1.persons[0].equals(pc2.persons[0]));
assertTrue(pc1.equals(pc2));

pc1.addPerson(p3); pc2.addPerson(p4);
assertTrue(pc1.persons[1] == pc2.persons[1]);
assertTrue(pc1.persons[1].equals(pc2.persons[1]));
assertTrue(pc1.equals(pc2));
```

False

Person

F

**PersonCollector**

```
boolean equals(Object obj) {
 if(this == obj) { return true; }
 if(obj == null || this.getClass() != obj.getClass()) {
   return false; }
 PersonCollector other = (PersonCollector) obj;
 boolean equal = false;
 if(this.nop == other.nop) {
  equal = true;
  for(int i = 0; equal && i < this.nop; i ++) {
   equal = this.persons[i].equals(other.persons[i]); } }
 return equal;
}
```

F

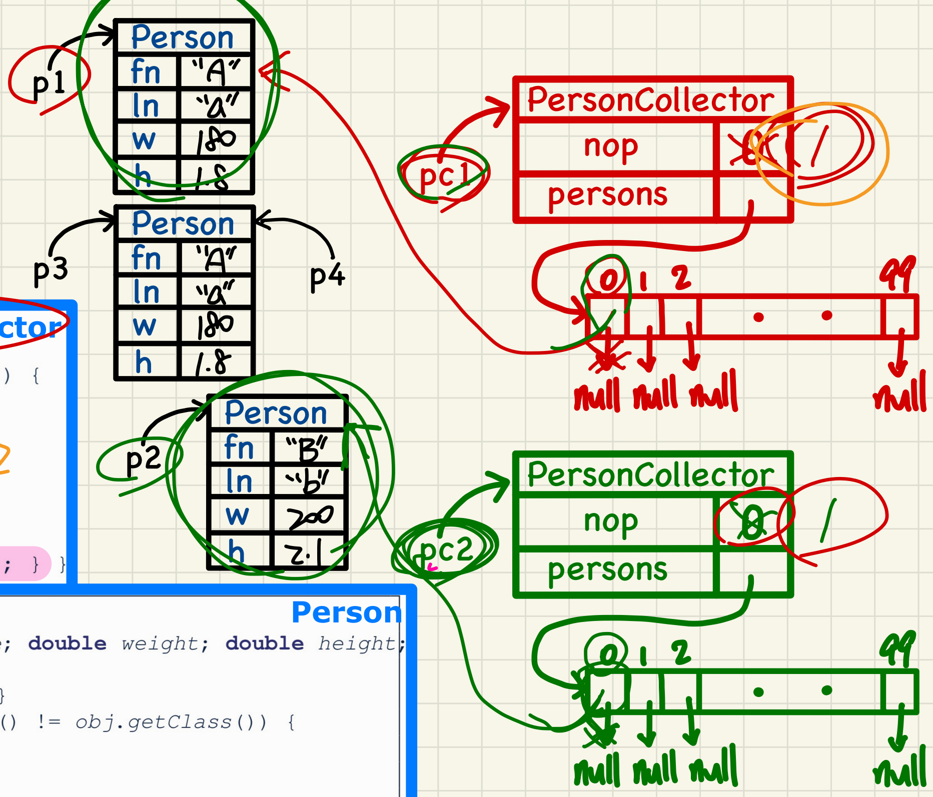
pc2

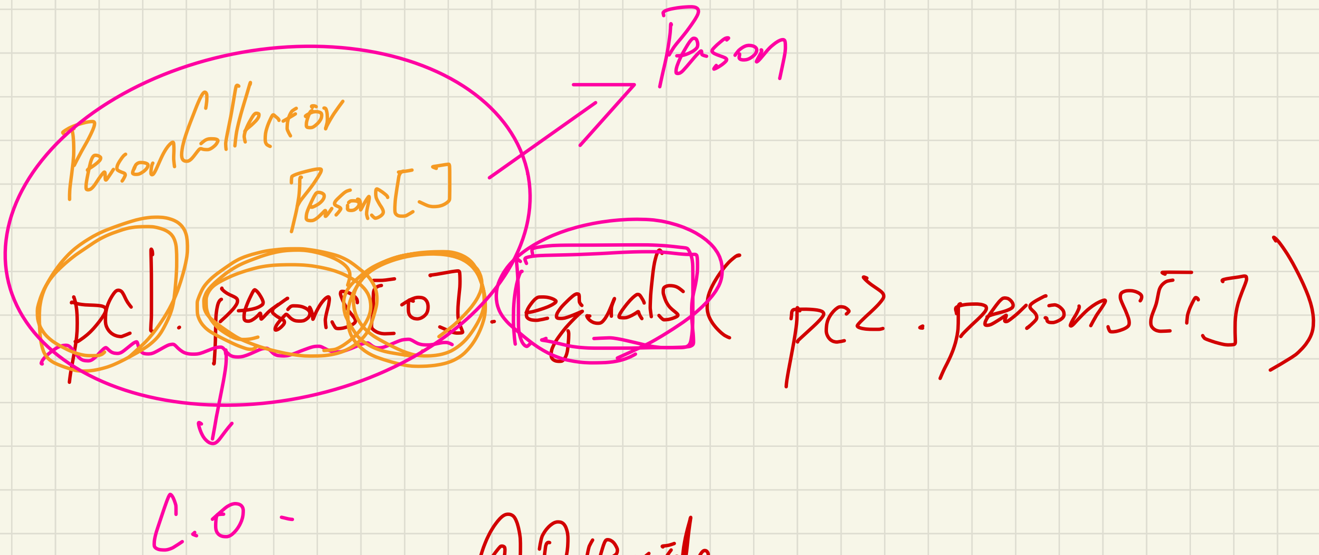
0

pc1

**Person**

```
class Person {
 String firstName; String lastName; double weight; double height;
 boolean equals(Object obj) {
   if(this == obj) { return true; }
   if(obj == null || this.getClass() != obj.getClass()) {
    return false; }
   Person other = (Person) obj;
   return
      this.weight == other.weight && this.height == other.height
    && this.firstName.equals(other.firstName)
    && this.lastName.equals(other.lastName); } }
```





```
pc1.persons[0].equals( pc2.persons[1])
```

```
@Override
boolean equals(. -)
```

# Ordering between Employees

| name | id | salary |
|---|---|---|
| alan | 2 | 4500.31 |
| mark | 3 | 3450.67 |
| tom | 1 | 3450.67 |

attributes

tom.id < alan.id < mark.id

~~mark~~.salary = mark.salary < alan.salary

tom

**Sorting**: from smallest to largest

Sorting based on id's
(smaller id's come first)

tom < alan < mark

Sorting based on salaries and id's
(higher salaries and smaller id's come first)

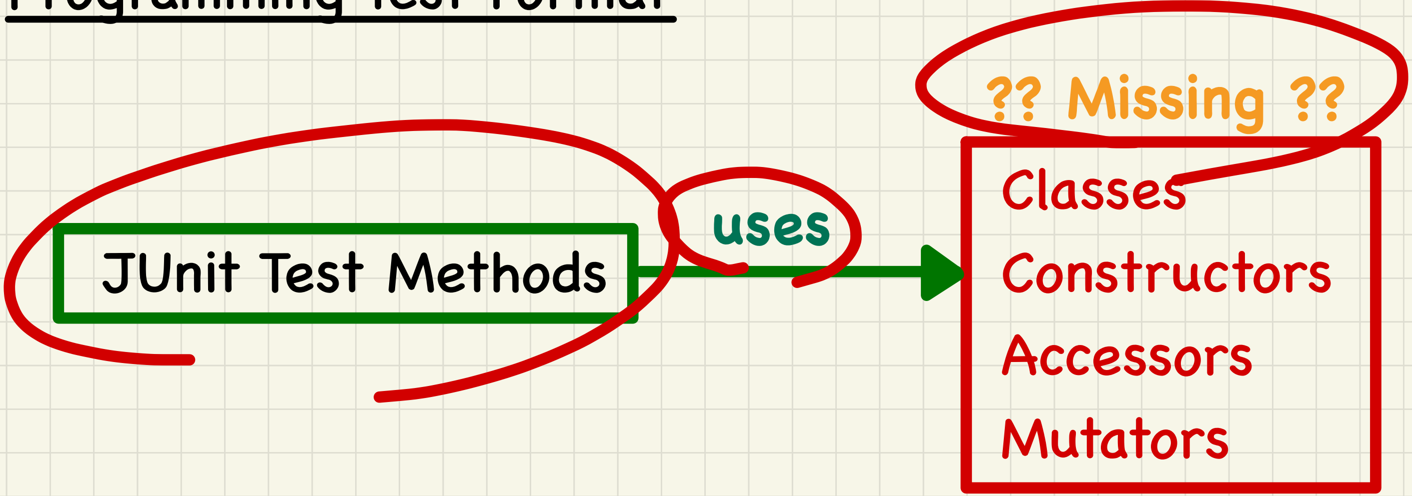
alan < tom < mark

# PROGRAMMING TEST I

REVIEW SESSION

FRIDAY OCTOBER 4

# Programming Test Format



JUnit Test Methods

uses

?? Missing ??

Classes

Constructors

Accessors

Mutators

## Your Tasks

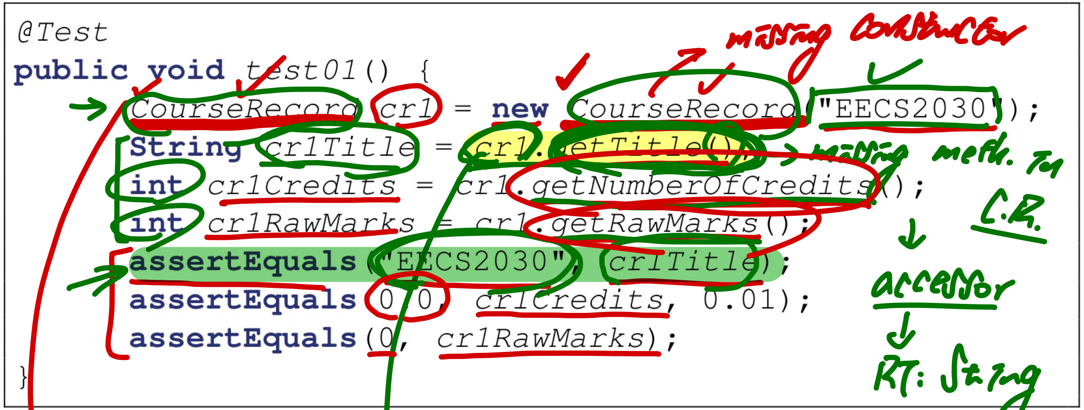
- **Read** the given JUnit tests
- **Infer** the necessary classes and methods
- **Write** suitable attributes and method implementations

## Requirements

[]

- Use primitive arrays and loops only to implement collections
- No use of any Java library classes (e.g., ArrayList, Arrays)
- No import statements at the beginning of your classes

# Programming Test Review Exercise: Test 1



```java
@Test
public void test01() {
    CourseRecord cr1 = new CourseRecord("EECS2030");
    String cr1Title = cr1.getTitle();
    int cr1Credits = cr1.getNumberOfCredits();
    int cr1RawMarks = cr1.getRawMarks();
    assertEquals("EECS2030", cr1Title);
    assertEquals(0.0, cr1Credits, 0.01);
    assertEquals(0, cr1RawMarks);
}
```

missing constructor

missing meth. for C.R.

↓

accessor

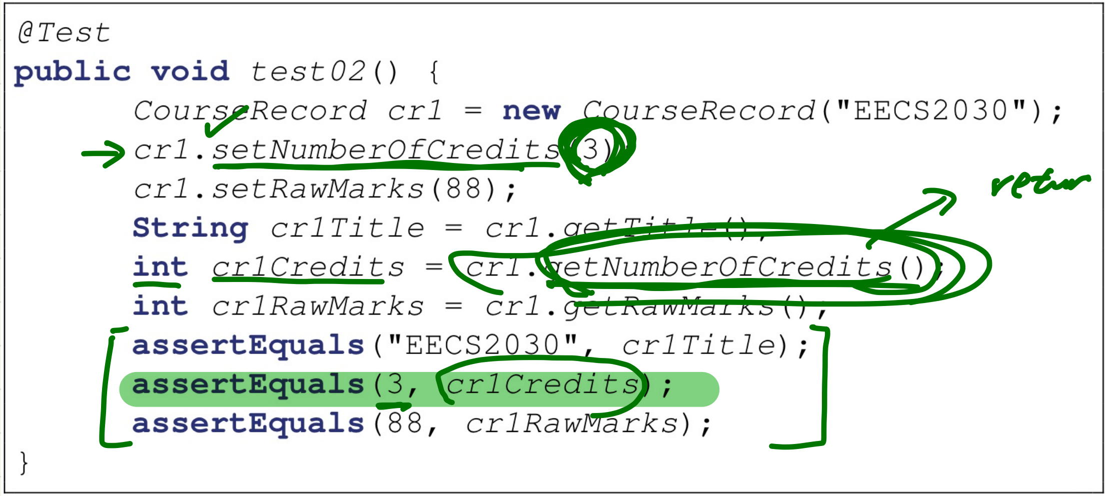
↓

RT: String

missing class

context object

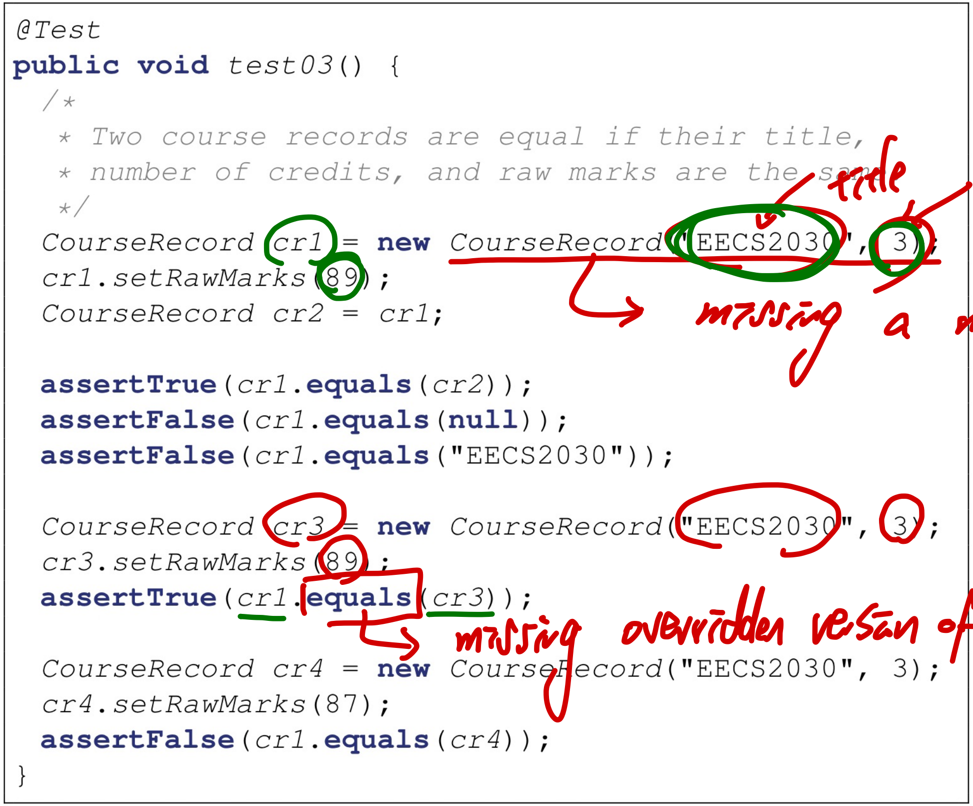
# Programming Test Review Exercise: Test 2



```
@Test
public void test02() {
      CourseRecord cr1 = new CourseRecord("EECS2030");
      cr1.setNumberOfCredits(3);
      cr1.setRawMarks(88);
      String cr1Title = cr1.getTitle();
      int cr1Credits = cr1.getNumberOfCredits();
      int cr1RawMarks = cr1.getRawMarks();
      assertEquals("EECS2030", cr1Title);
      assertEquals(3, cr1Credits);
      assertEquals(88, cr1RawMarks);
}
```

return this.credits

# Programming Test Review Exercise: Test 3



```java
@Test
public void test03() {
  /*
   * Two course records are equal if their title,
   * number of credits, and raw marks are the same
   */
  CourseRecord cr1 = new CourseRecord("EECS2030", 3);
  cr1.setRawMarks(89);
  CourseRecord cr2 = cr1;

  assertTrue(cr1.equals(cr2));
  assertFalse(cr1.equals(null));
  assertFalse(cr1.equals("EECS2030"));

  CourseRecord cr3 = new CourseRecord("EECS2030", 3);
  cr3.setRawMarks(89);
  assertTrue(cr1.equals(cr3));

  CourseRecord cr4 = new CourseRecord("EECS2030", 3);
  cr4.setRawMarks(87);
  assertFalse(cr1.equals(cr4));
}
```

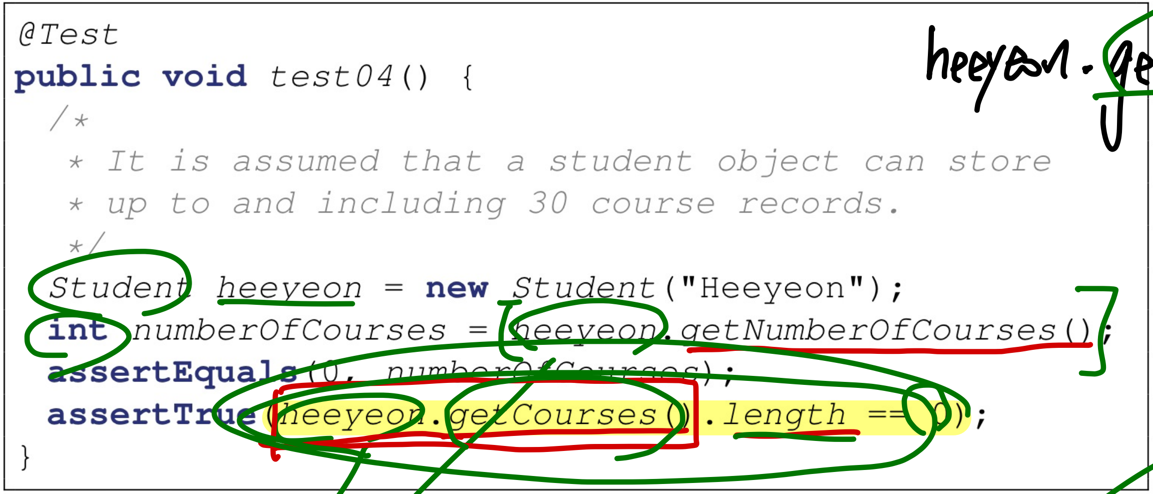
title

no.

missing a new version of constructor

missing overridden version of this method!

# Programming Test Review Exercise: Test 4



```java
@Test
public void test04() {
  /*
   * It is assumed that a student object can store
   * up to and including 30 course records.
   */
  Student heeyeon = new Student("Heeyeon");
  int numberOfCourses = heeyeon.getNumberOfCourses();
  assertEquals(0, numberOfCourses);
  assertTrue(heeyeon.getCourses().length == 0);
}
```

heeyeon.getCourses().length == 0

return null

Context object

missing methods from Student

length

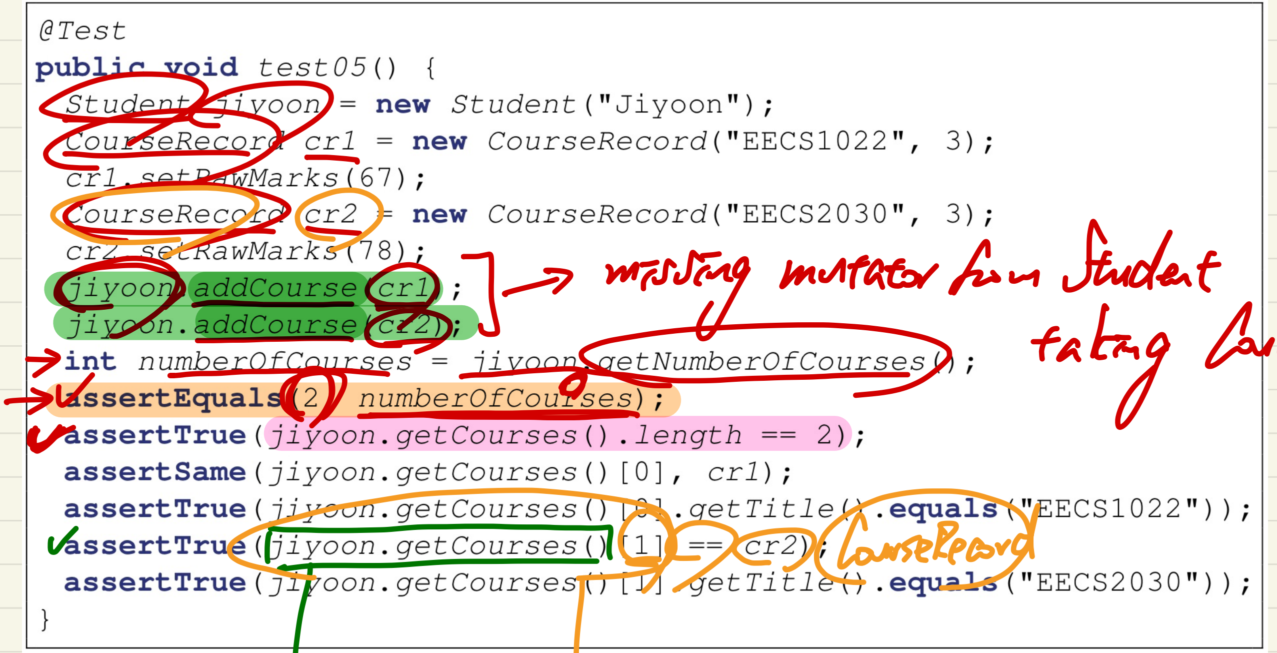
length() X

Student[ ]

CourseRecord[ ]

String[ ]

# Programming Test Review Exercise: Test 5



```
@Test
public void test05() {
  Student jiyoon = new Student("Jiyoon");
  CourseRecord cr1 = new CourseRecord("EECS1022", 3);
  cr1.setRawMarks(67);
  CourseRecord cr2 = new CourseRecord("EECS2030", 3);
  cr2.setRawMarks(78);
  jiyoon.addCourse(cr1);
  jiyoon.addCourse(cr2);
  int numberOfCourses = jiyoon.getNumberOfCourses();
  assertEquals(2, numberOfCourses);
  assertTrue(jiyoon.getCourses().length == 2);
  assertSame(jiyoon.getCourses()[0], cr1);
  assertTrue(jiyoon.getCourses()[0].getTitle().equals("EECS1022"));
  assertTrue(jiyoon.getCourses()[1] == cr2);
  assertTrue(jiyoon.getCourses()[1].getTitle().equals("EECS2030"));
}
```

missing mutator from Student taking CourseRecord

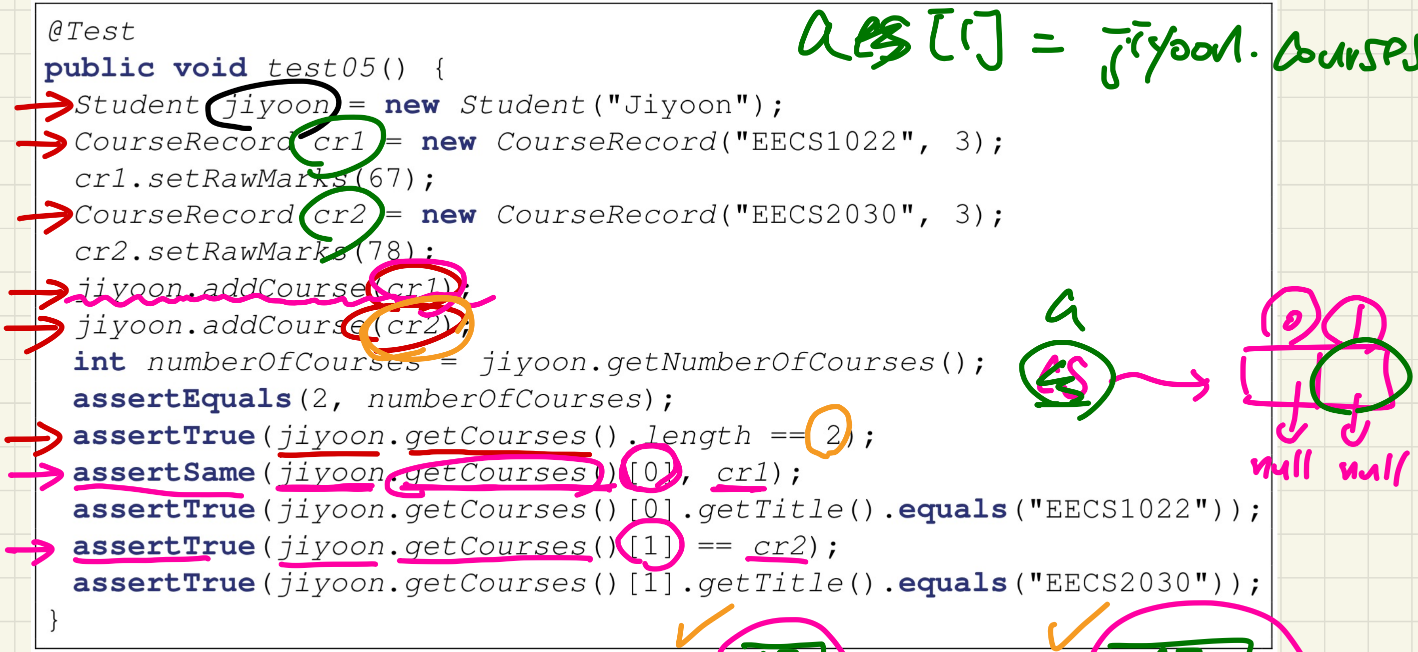
CourseRecord

~~Student~~[]
CourseRecord

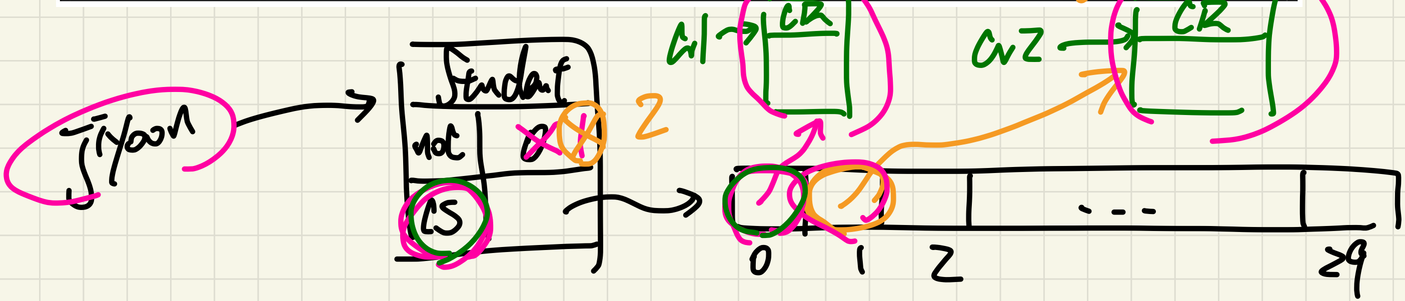
~~Student~~
CourseRecord

a cs[0] = jiyoon.courses[0]

a cs[i] = jiyoon.courses[i]



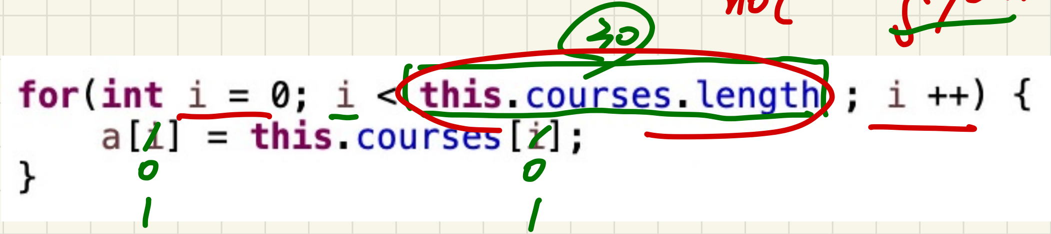
```
@Test
public void test05() {
  Student jiyoon = new Student("Jiyoon");
  CourseRecord cr1 = new CourseRecord("EECS1022", 3);
  cr1.setRawMarks(67);
  CourseRecord cr2 = new CourseRecord("EECS2030", 3);
  cr2.setRawMarks(78);
  jiyoon.addCourse(cr1);
  jiyoon.addCourse(cr2);
  int numberOfCourses = jiyoon.getNumberOfCourses();
  assertEquals(2, numberOfCourses);
  assertTrue(jiyoon.getCourses().length == 2);
  assertSame(jiyoon.getCourses()[0], cr1);
  assertTrue(jiyoon.getCourses()[0].getTitle().equals("EECS1022"));
  assertTrue(jiyoon.getCourses()[1] == cr2);
  assertTrue(jiyoon.getCourses()[1].getTitle().equals("EECS2030"));
}
```



noC

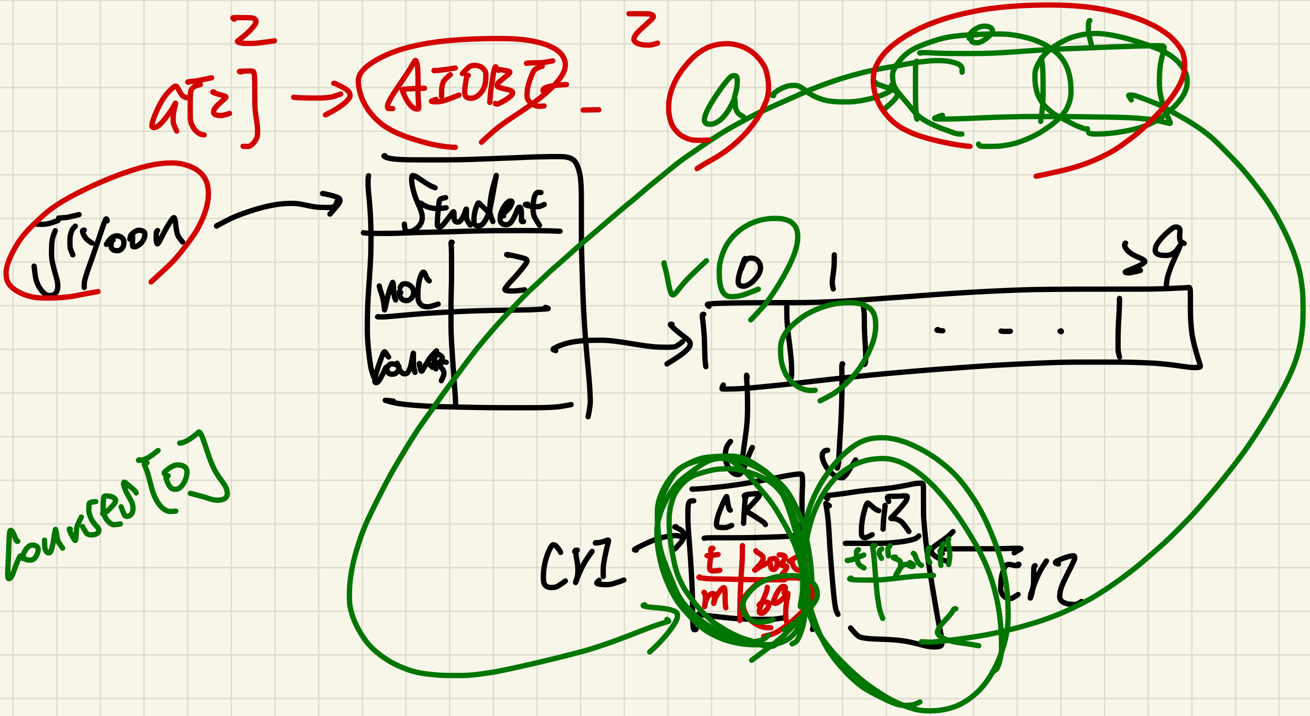
jiyoon.getLetterGrade("EECS2011")

("EECS2030")



```
for(int i = 0; i < this.courses.length ; i ++) {
    a[i] = this.courses[i];
}
```

"C"



# LECTURE 10

MONDAY OCTOBER 7

# Testing Equality of Person/PersonCollector in JUnit (3)

(continued from testPersonCollector)

```
pc1.addPerson(p1);
assertFalse(pc1.equals(pc2));

pc2.addPerson(p2);
assertFalse(pc1.persons[0] == pc2.persons[0]);
assertTrue(pc1.persons[0].equals(pc2.persons[0]));
assertTrue(pc1.equals(pc2));

pc1.addPerson(p3); pc2.addPerson(p4);
assertTrue(pc1.persons[1] == pc2.persons[1]);
assertTrue(pc1.persons[1].equals(pc2.persons[1]));
assertTrue(pc1.equals(pc2));
```

F

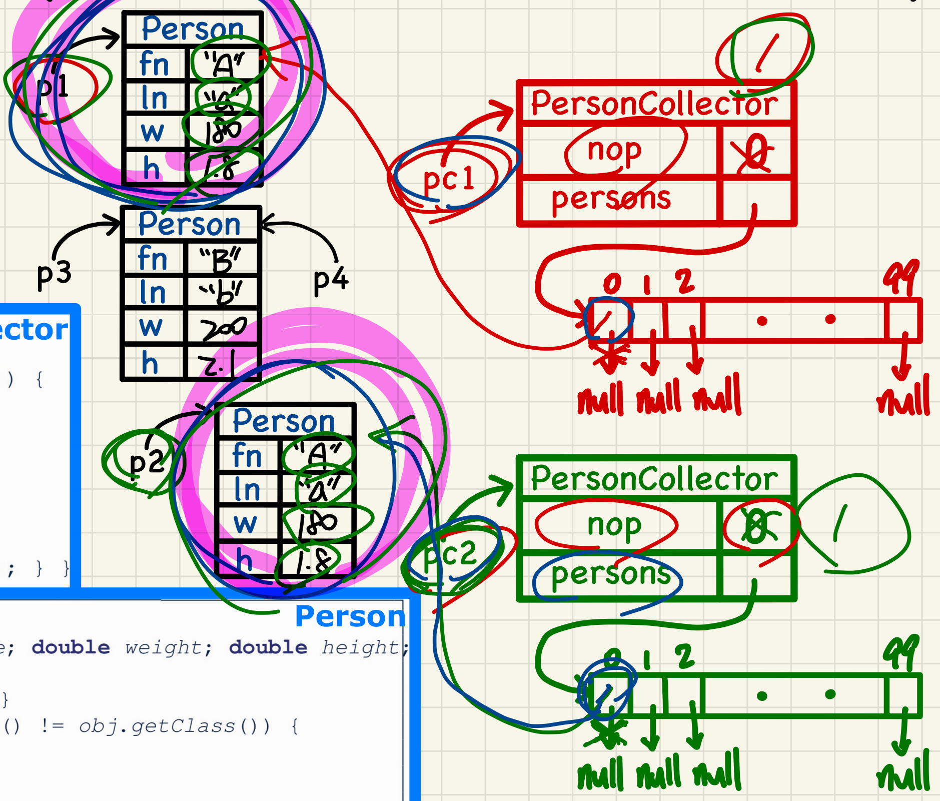
Person

**PersonCollector**

```
boolean equals(Object obj) {
  if(this == obj) { return true; }
  if(obj == null || this.getClass() != obj.getClass()) {
    return false; }
  PersonCollector other = (PersonCollector) obj;
  boolean equal = false;
  if(this.nop == other.nop) {
    equal = true;
    for(int i = 0; equal && i < this.nop; i ++) {
      equal = this.persons[i].equals(other.persons[i]); } }
  return equal;
}
```

**Person**

```
class Person {
  String firstName; String lastName; double weight; double height;
  boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null || this.getClass() != obj.getClass()) {
      return false; }
    Person other = (Person) obj;
    return
        this.weight == other.weight && this.height == other.height
      && this.firstName.equals(other.firstName)
      && this.lastName.equals(other.lastName); } }
```



# Ordering between Employees

| name | id | salary |
|---|---|---|
| alan | 2 | 4500.31 |
| mark | 3 | 3450.67 |
| tom | 1 | 3450.67 |

tom.id < alan.id < mark.id

mark.salary = mark.salary < alan.salary

**Sorting**: from smallest to largest

## Sorting based on id's

(smaller id's come first)

seq 1

tom < alan < mark

## Sorting based on salaries and id's

(higher salaries and smaller id's come first)

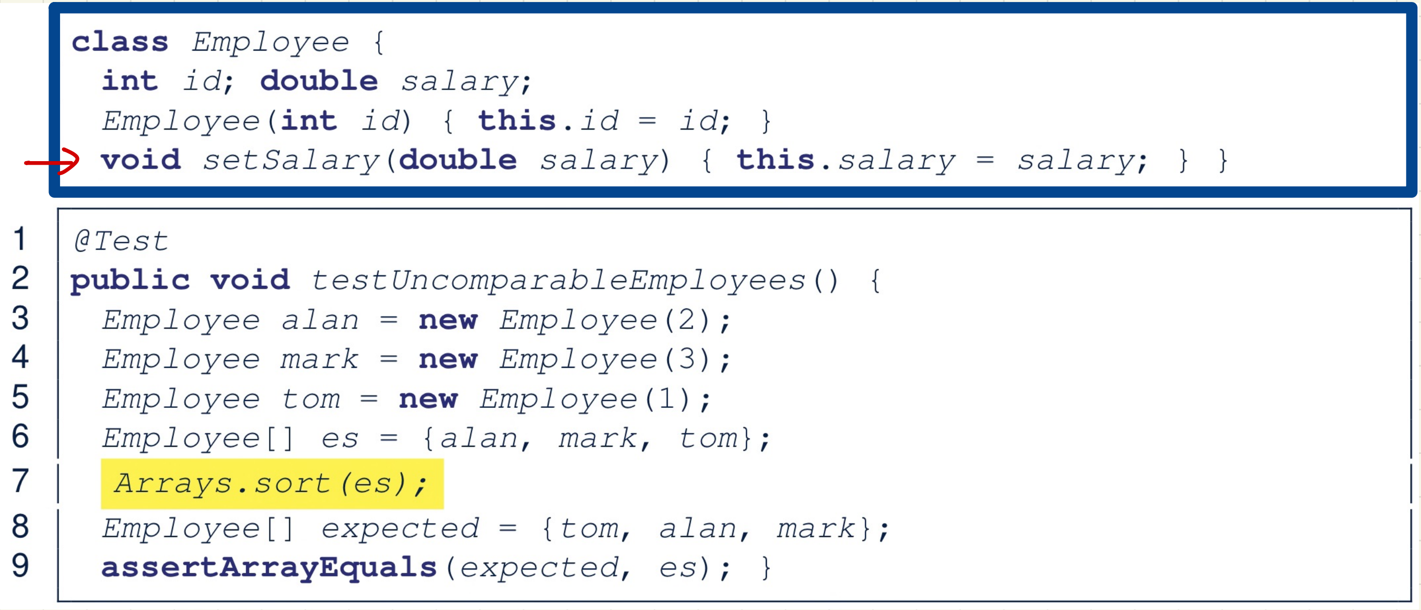
seq 2

alan < tom < mark

# Unknown Ordering between Employees

Say: Sorting based on ~~salaries and~~ id's

(higher salaries and smaller id's come first)



```
class Employee {
  int id; double salary;
  Employee(int id) { this.id = id; }
  void setSalary(double salary) { this.salary = salary; } }
```

```
@Test
public void testUncomparableEmployees() {
  Employee alan = new Employee(2);
  Employee mark = new Employee(3);
  Employee tom = new Employee(1);
  Employee[] es = {alan, mark, tom};
  Arrays.sort(es);
  Employee[] expected = {tom, alan, mark};
  assertArrayEquals(expected, es); }
```

# Comparable Employees: Version 1

```
class CEmployee1 implements Comparable<CEmployee1> {
  ... /* attributes, constructor, mutator similar to Employee */
  @Override
  public int compareTo(CEmployee1 e) { return this.id - e.id; }
}
```

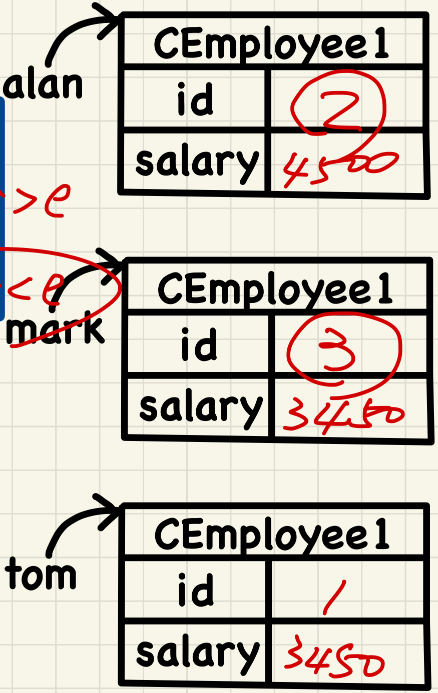
3 1 + this > e

1 3 − this < e

```
@Test
public void testComparableEmployees_1() {
  /*
   * CEmployee1 implements the Comparable interface.
   * Method compareTo compares id's only.
   */
  CEmployee1 alan = new CEmployee1(2);
  CEmployee1 mark = new CEmployee1(3);
  CEmployee1 tom = new CEmployee1(1);
  alan.setSalary(4500.34);
  mark.setSalary(3450.67);
  tom.setSalary(3450.67);
  CEmployee1[] es = {alan, mark, tom};
  /* When comparing employees,
   * their salaries are irrelevant.
   */
  Arrays.sort(es);
  CEmployee1[] expected = {tom, alan, mark};
  assertArrayEquals(expected, es);
}
```

alan "=" alan

merge sort ✓



alan → CEmployee1: id 2, salary 4500

mark → CEmployee1: id 3, salary 3450

tom → CEmployee1: id 1, salary 3450

| compareTo | alan | mark | tom |
|---|---|---|---|
| alan | 0 | < | |
| mark | | | |
| tom | | | |

# Comparable Employees: Version 2.1

```java
class CEmployee2  implements  Comparable <CEmployee2> {
  ... /* attributes, constructor, mutator similar to Employee */
  @Override
  public int compareTo(CEmployee2 other) {
    if (this.salary > other.salary) {
      return -1;
    }
    else if (this.salary < other.salary) {
      return 1;
    }
    else { /* equal salaries */
      return this.id - other.id;
    }
}
```

this should come first

why not -1 ?

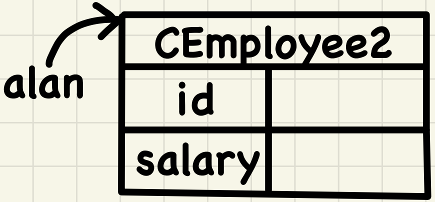
salary is a stronger criterion to consider first.

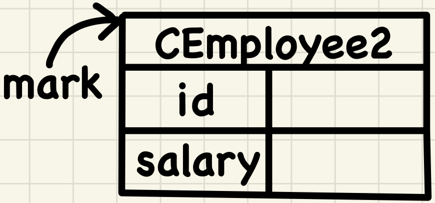
2

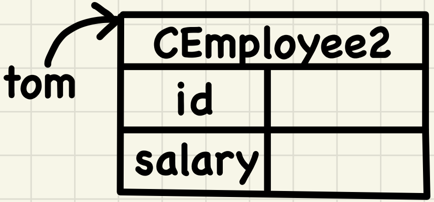
3

-1

```java
@Test
public void testComparableEmployees_2() {
  /*
   * CEmployee2 implements the Comparable interface.
   * Method compareTo first compares salaries, then
   * compares id's for employees with equal salaries.
   */
  CEmployee2 alan = new CEmployee2(2);
  CEmployee2 mark = new CEmployee2(3);
  CEmployee2 tom = new CEmployee2(1);
  alan.setSalary(4500.34);
  mark.setSalary(3450.67);
  tom.setSalary(3450.67);
  CEmployee2[] es = {alan, mark, tom};
  Arrays.sort(es);
  CEmployee2[] expected = {alan, tom, mark};
  assertArrayEquals(expected, es);
}
```







| compareTo | alan | mark | tom |
|---|---|---|---|
| alan | | | |
| mark | | | |
| tom | | | |

Smaller id's come first.

If equal id's, then higher salary come first.

|   | id | salary |
|---|---|---|
| A | 2 | 45000 |
| B | 1 | 30000 |

Original criterion:

A B

revised criterion:

B A

```
class CEmployee2  implements Comparable<CEmployee2> {
  ... /* attributes, constructor, mutator similar to Employee */
  @Override
  public int compareTo(CEmployee2 other) {
    if(this.~~salary~~ id > other.~~salary~~ id) {
→    return ~~-1~~; +1
    }
    else if (this.~~salary~~ id < other.~~salary~~ id) {
      return ~~1~~; -1
    }
    else { /* equal salaries */
      return this.~~id~~ - other.~~id~~;
    }
}
```

— this.salary - other.salary

45000 30000

if ( this.s - other.s < 0){

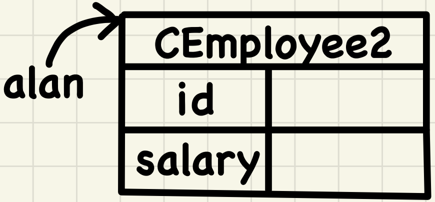
return +1 ;

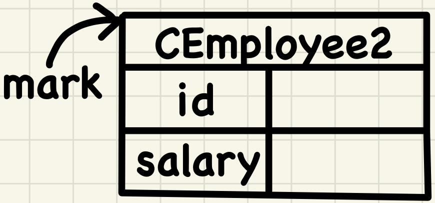
# Comparable Employees: Version 2.2

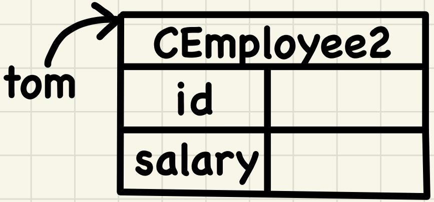
```
class CEmployee2  implements  Comparable<CEmployee2> {
  ... /* attributes, constructor, mutator similar to Employee */
  @Override
  public int compareTo(CEmployee2 other) {
    int salaryDiff = Double.compare(this.salary, other.salary);
    int idDiff = this.id - other.id;
    if(salaryDiff != 0) { return - salaryDiff; }
    else { return idDiff; } } }
```

45000 — 30000

```
@Test
public void testComparableEmployees_2() {
  /*
   * CEmployee2 implements the Comparable interface.
   * Method compareTo first compares salaries, then
   * compares id's for employees with equal salaries.
   */
  CEmployee2 alan = new CEmployee2(2);
  CEmployee2 mark = new CEmployee2(3);
  CEmployee2 tom = new CEmployee2(1);
  alan.setSalary(4500.34);
  mark.setSalary(3450.67);
  tom.setSalary(3450.67);
  CEmployee2[] es = {alan, mark, tom};
  Arrays.sort(es);
  CEmployee2[] expected = {alan, tom, mark};
  assertArrayEquals(expected, es);
}
```







| compareTo | alan | mark | tom |
|---|---|---|---|
| alan | | | |
| mark | | | |
| tom | | | |

# Design Principles of the compareTo Method

$4 < 2$

$2 < 4$

**Asymmetric** :

$c1 < c2$ $\quad$ $c2 < c1$

negation

$\neg(c1.compareTo(c2) < 0 \land c2.compareTo(c1) < 0)$
$\neg(c1.compareTo(c2) > 0 \land c2.compareTo(c1) > 0)$

$i < j$ $\quad$ $j < k$

$\hookrightarrow i < k$

**Transitive** : $c1 < c2$ $\quad$ $c2 < c3$ $\quad$ $c1 < c3$

$c1.compareTo(c2) < 0 \land c2.compareTo(c3) < 0 \Rightarrow c1.compareTo(c3) < 0$
$c1.compareTo(c2) > 0 \land c2.compareTo(c3) > 0 \Rightarrow c1.compareTo(c3) > 0$

a $\quad$ R

b $\quad$ S

c $\quad$ P

$a < b$

$b < c$

$a < c$ X

```
public class Entry {
  private int key;
  private String value;

  public Entry(int key, String value) {
    this.key = key;
    this.value = value;
  }
```

1 "D"

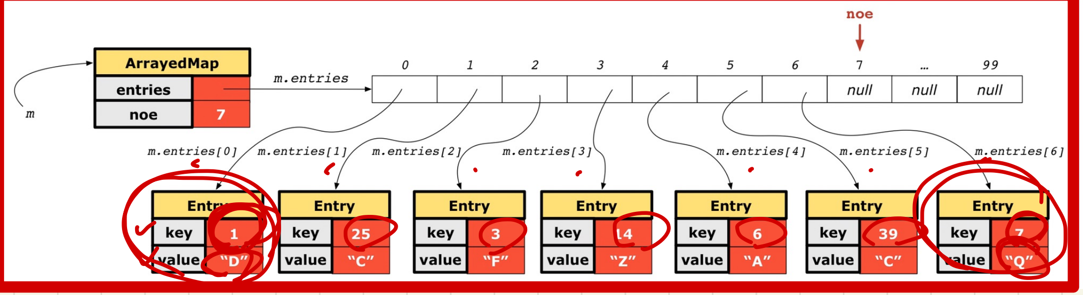
```
public class ArrayedMap {
  private final int MAX_CAPCAITY = 100;
  private Entry[] entries;
  private int noe; /* number of entries */
  public ArrayedMap() {
    entries = new Entry[MAX_CAPCAITY];
    noe = 0;
  }
  public int size() {
    return noe;
  }
  public void put(int key, String value) {
    Entry e = new Entry(key, value);
    entries[noe] = e;
    noe ++;
  }
```

100

```
@Test
public void testArrayedMap() {
  ArrayedMap m = new ArrayedMap();
  assertTrue(m.size() == 0);
  m.put(1, "D");
  m.put(25, "C");
  m.put(3, "F");
  m.put(14, "Z");
  m.put(6, "A");
  m.put(39, "C");
  m.put(7, "Q");
  assertTrue(m.size() == 7);
  /* inquiries of existing key */
  assertTrue(m.get(1).equals("D"));
  assertTrue(m.get(7).equals("Q"));
  /* inquiry of non-existing key */
  assertTrue(m.get(31) == null);
}
```

k v

no duplicates of keys



# Naive Implementation of a Map

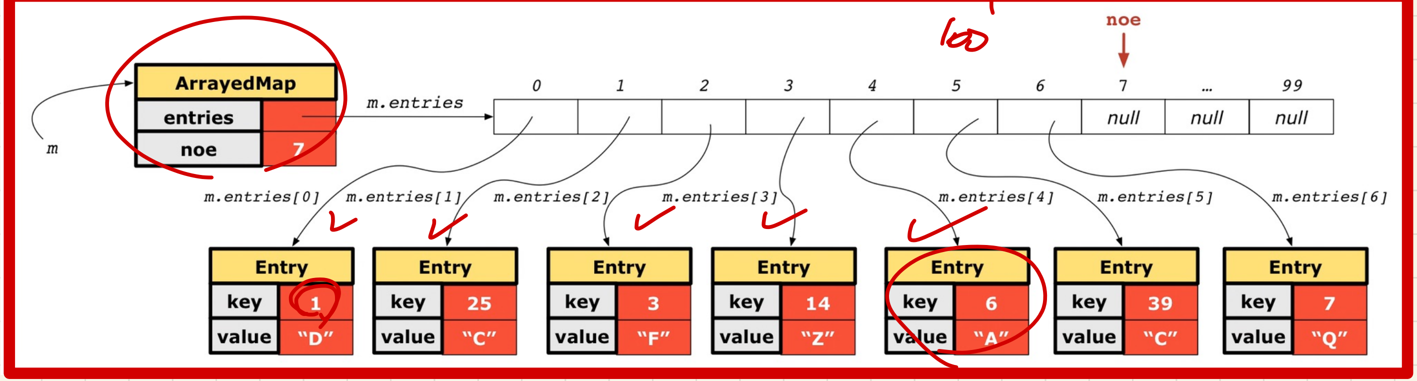
# Naive Implementation of a Map: Retrieval of an Entry

```
public class ArrayedMap {
  private final int MAX_CAPCAITY = 100;
  public String get (int key) {
    for(int i = 0; i < noe; i ++) {
      Entry e = entries[i];
      int k = e.getKey();
      if(k == key) { return e.getValue(); }
    }
    return null;
  }
```

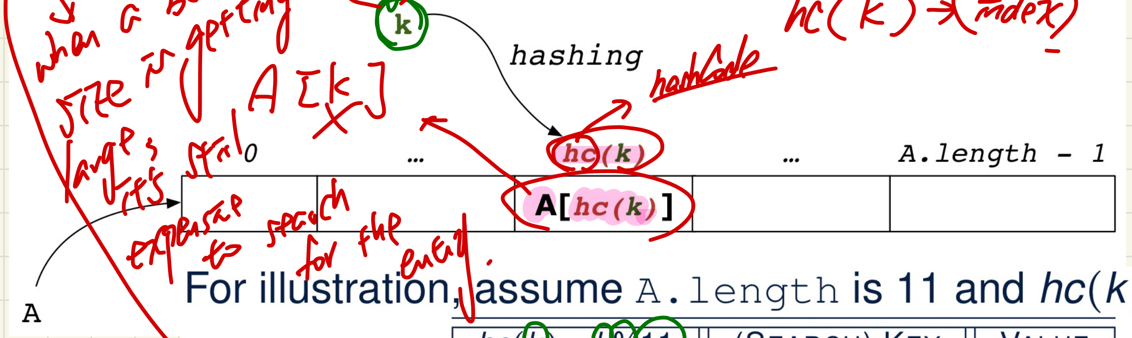
| | # iterations |
|---|---|
| m.get(1) | 1 |
| m.get(6) | 5 |

worst case: 7

100



# Implementing a Hash Table via Hashing


k
hashing
hc(k)
0
...
...
A.length - 1
A[hc(k)]
A
key
hashcode
A[k]
when a bucket's size is getting large, it's still expensive to search for the entry.

hc(k) → index

- Converting k to hc(k) → index
- Indexing into A[hc(k)] → index

For illustration, assume A.length is 11 and $hc(k) = k\%11$.

| $hc(k) = k\%11$ | (SEARCH) KEY | VALUE |
|---|---|---|
| [redacted] 1 | 1 | D |
| [redacted] 3 | 25 | C |
| [redacted] 3 | 3 | F |
| [redacted] 3 | 14 | Z |
| [redacted] 6 | 6 | A |
| [redacted] 6 | 39 | C |
| [redacted] 7 | 7 | Q |

equal hc.

unequal keys

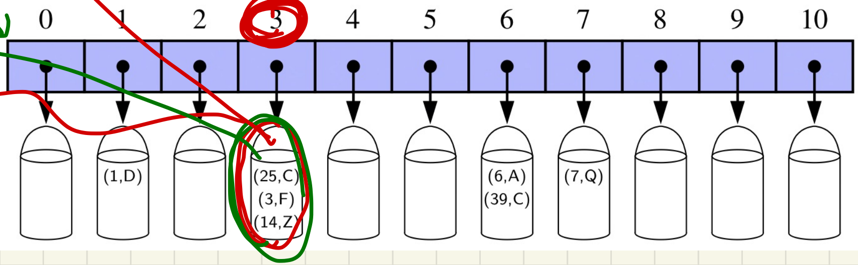
collision

bucket

25 % 11 == 3

where should they be stored?

index hc(k)



collision if ∴ keys 25, 3, 14 all have hc 3.

# Testing Overridden/Redefined hashCode()

```
@Test
public void testCustomizedHashFunction() {
  IntegerKey ik1 = new IntegerKey(1);
  /* 1 % 11 == 1 */
  assertTrue(ik1.hashCode() == 1);

  IntegerKey ik39_1 = new IntegerKey(39); /* 39 % 11 == 6 */
  IntegerKey ik39_2 = new IntegerKey(39);
  IntegerKey ik6 = new IntegerKey(6); /* 6 % 11 == 6 */

  assertTrue(ik39_1.hashCode() == 6);
  assertTrue(ik39_2.hashCode() == 6);
  assertTrue(ik6.hashCode() == 6);

  assertTrue(ik39_1.hashCode() == ik39_2.hashCode());
  assertTrue(ik39_1.equals(ik39_2));

  assertTrue(ik39_1.hashCode() == ik6.hashCode());
  assertFalse(ik39_1.equals(ik6));
}
```

39 ✓

① ik39_1.equals(ik6); F

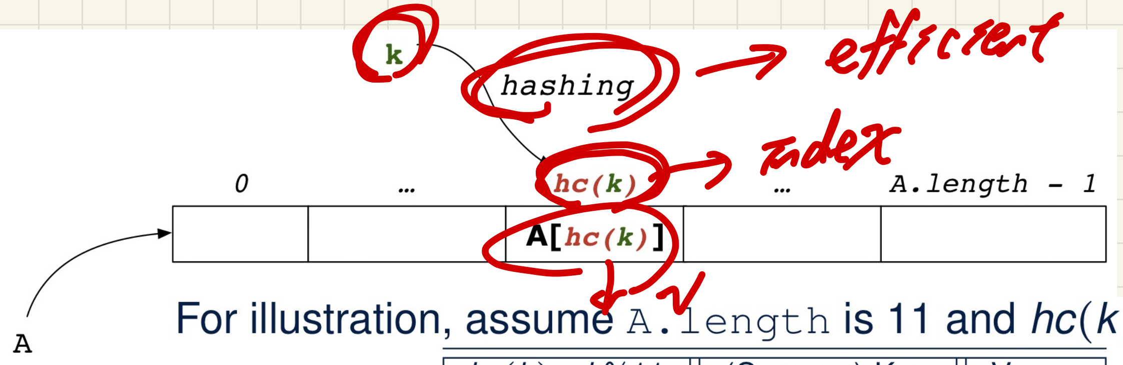
② ik39_1.hashCode() == ik6.hashCode T

```
public class IntegerKey {
  private int k;
  public IntegerKey(int k) { this.k = k; }
  @Override
  public int hashCode() { return k % 11; }
  @Override
  public boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false; }
    IntegerKey other = (IntegerKey) obj;
    return this.k == other.k;
} }
```

# Lecture 11

Wednesday October 9

# Implementing a Hash Table via Hashing



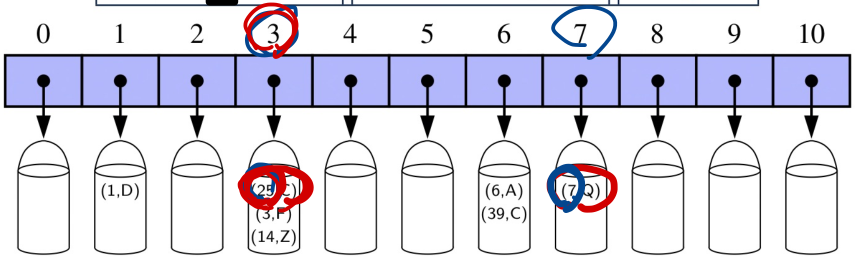
- Converting k to hc(k)
- Indexing into A[hc(k)]

For illustration, assume A.length is 11 and *hc*(*k*) = *k*%11.

| *hc*(*k*) = *k*%11 | (SEARCH) KEY | VALUE |
|---|---|---|
| [redacted] | 1 | D |
| [redacted] | 25 | C |
| [redacted] | 3 | F |
| [redacted] | 14 | Z |
| [redacted] | 6 | A |
| [redacted] | 39 | C |
| [redacted] | 7 | Q |

25 ≠ 7

3 ≠ 7



7 $\underline{k1}$ . equals(k2)

$hc(k) = k \% 11$

- $hc(k1) \underline{==} hc(k2)$

- $hc(k1) \mathrel{!=} hc(k2)$

k1 k2

25 3

collision

25 7

hc(k)=k % 11

hc(k1) == hc(k2)

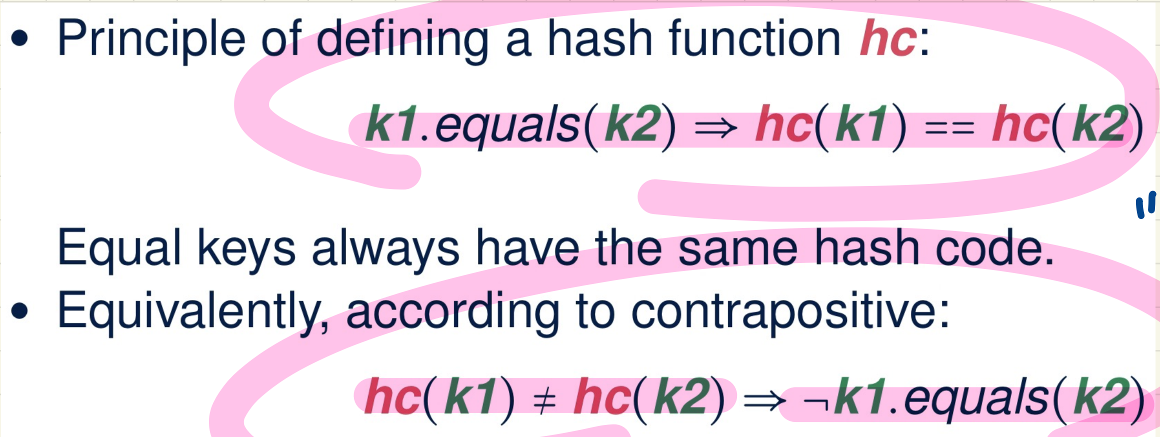
k1 k2

? k1.equals(k2) 25 3 3

k1.equals(k2) 25 25

hc ~~f~~ function
↑ found

# Contract of a **Hash Code** Function



- Principle of defining a hash function *hc*:

$$k1.equals(k2) \Rightarrow hc(k1) == hc(k2)$$

Equal keys always have the same hash code.

- Equivalently, according to contrapositive:

$$hc(k1) \neq hc(k2) \Rightarrow \neg k1.equals(k2)$$

$p \Rightarrow q \equiv$

$\neg q \Rightarrow \neg p$

"it rains" ⇒ "I bring um..."

¬ "I bring um..." ⇒ ¬ "it rains"

- What if k1.equals(k2) is **false**?
- What if hc(k1) == hc(k2) is **true**?

# Overridding/Redefining hashCode() from Object

```
1   public class IntegerKey {
2     private int k;
3     public IntegerKey(int k) { this.k = k; }
4     @Override
5     public int hashCode() { return k % 11; }
6     @Override
7     public boolean equals(Object obj) {
8       if(this == obj) { return true; }
9       if(obj == null) { return false; }
10      if(this.getClass() != obj.getClass()) { return false; }
11      IntegerKey other = (IntegerKey) obj;
12      return ~~this.k == other.k~~;
13  } }
```

this.hashCode() == other.hashCode()

1 1

- Principle of defining a hash function *hc*:

  ***k1**.equals(**k2**)* ⇒ ***hc**(**k1**) == **hc**(**k2**)*

  Equal keys always have the same hash code.
- Equivalently, according to contrapositive:

  ***hc**(**k1**) ≠ **hc**(**k2**)* ⇒ *¬**k1**.equals(**k2**)*

k1 k2

1 12

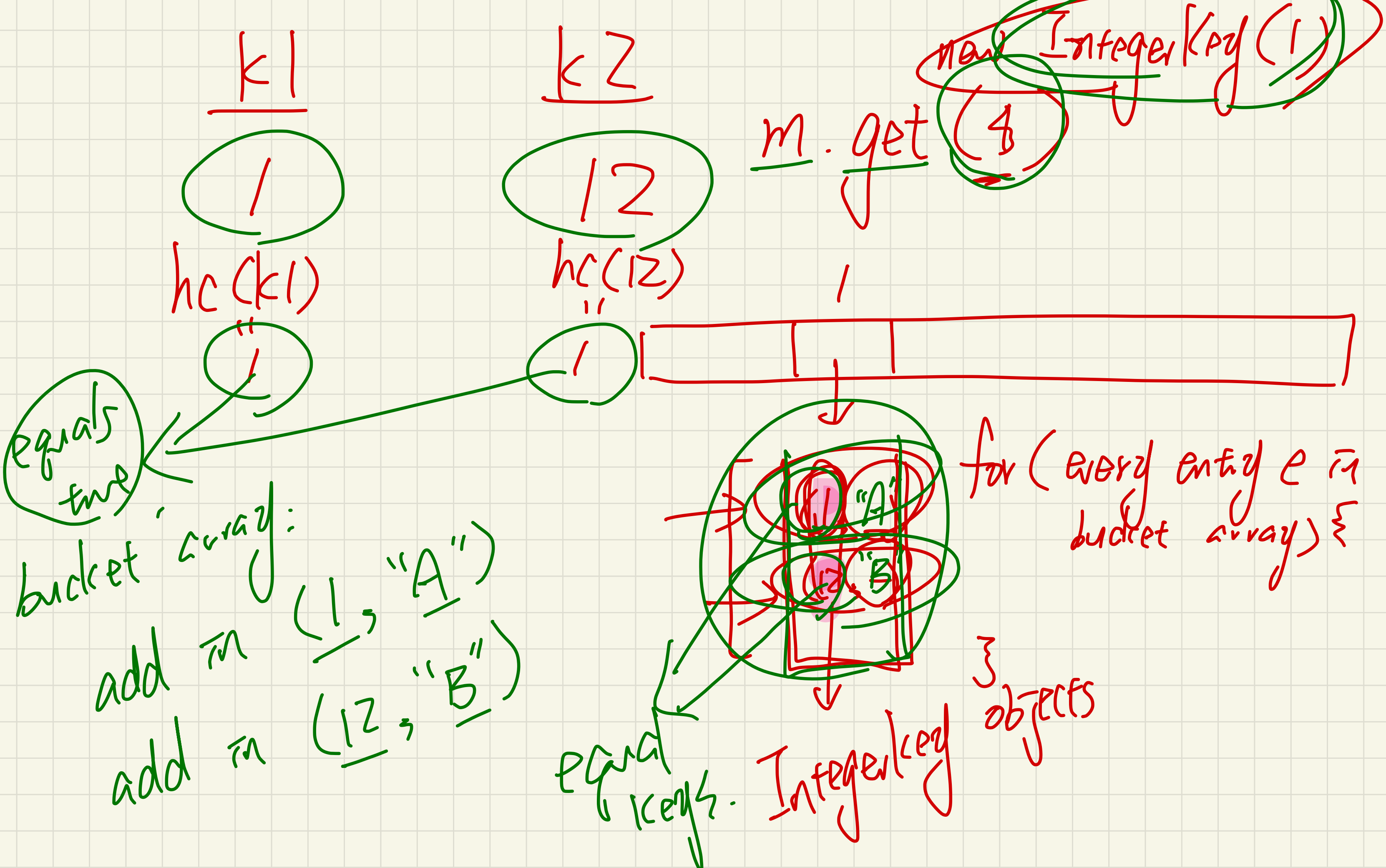
hc(k1) 1 hc(k2) 1

k

**Q**: Can we replace Line 12 by:

**return this**.hashCode() == other.hashCode();

return true ← k1. equals(k2) → hc(k1) == hc(k2)



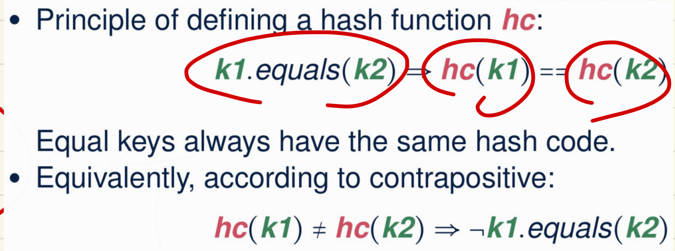
bucket array:

add in (1, "A")

add in (12, "B")

for (every entry e in bucket array) {
}

```
Ik ik1 = new Ik(3);

Ik ik2 = new Ik(3);
```



- Principle of defining a hash function ***hc***:

$$\textit{k1.equals}(\textit{k2}) \Rightarrow \textit{hc}(\textit{k1}) == \textit{hc}(\textit{k2})$$

Equal keys always have the same hash code.

- Equivalently, according to contrapositive:

$$\textit{hc}(\textit{k1}) \neq \textit{hc}(\textit{k2}) \Rightarrow \neg \textit{k1.equals}(\textit{k2})$$

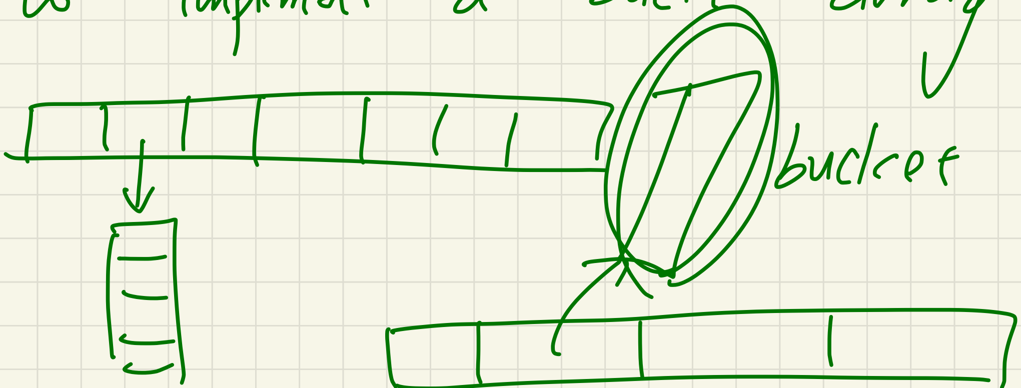
```
ik1.equals(ik2)

ik1.hc() == ik2.hc()
```

F

∵ ik1 != ik2

How to implement a bucket array?

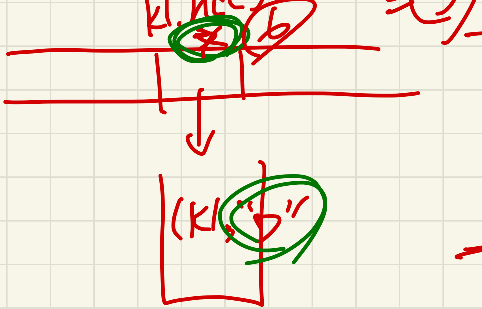
1.



2. ArrayList<ArrayList< >> Enty.

# Testing HashTable using Overridden/Redefined hashCode()

k1.hc() = 39%11 = 3



key

key

value

```
@Test
public void testHashTable() {
  Hashtable<IntegerKey, String> table = new Hashtable<>();
  IntegerKey k1 = new IntegerKey(39);
  IntegerKey k2 = new IntegerKey(39);
  assertTrue(k1.equals(k2));
  assertTrue(k1.hashCode() == k2.hashCode());
  table.put(k1, "D");
  assertTrue(table.get(k2).equals("D"));
}
```

T

table.put(k, v)

k.hashCode()

```
public class IntegerKey {
  private int k;
  public IntegerKey(int k) { this.k = k; }
  @Override
  public int hashCode() { return k % 11; }
  @Override
  public boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false; }
    IntegerKey other = (IntegerKey) obj;
    return this.k == other.k;
} }
```

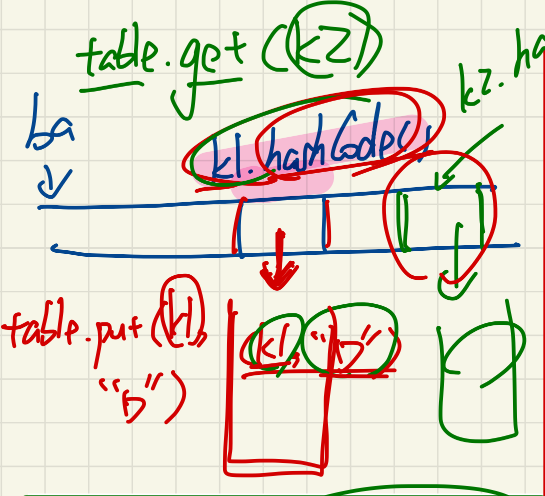
table.get(k2)

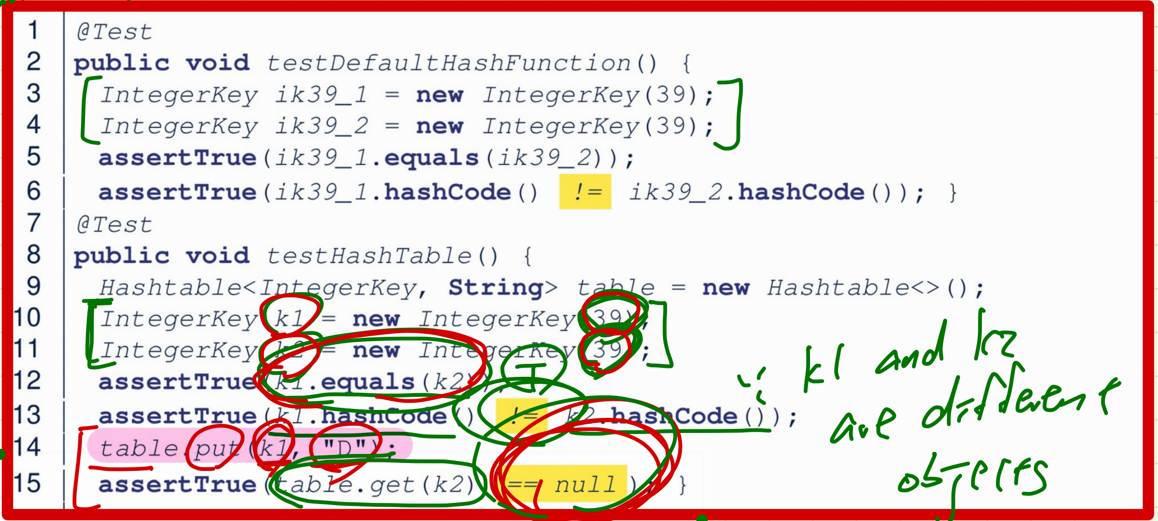
k2.hc()

||

~~3~~ 6

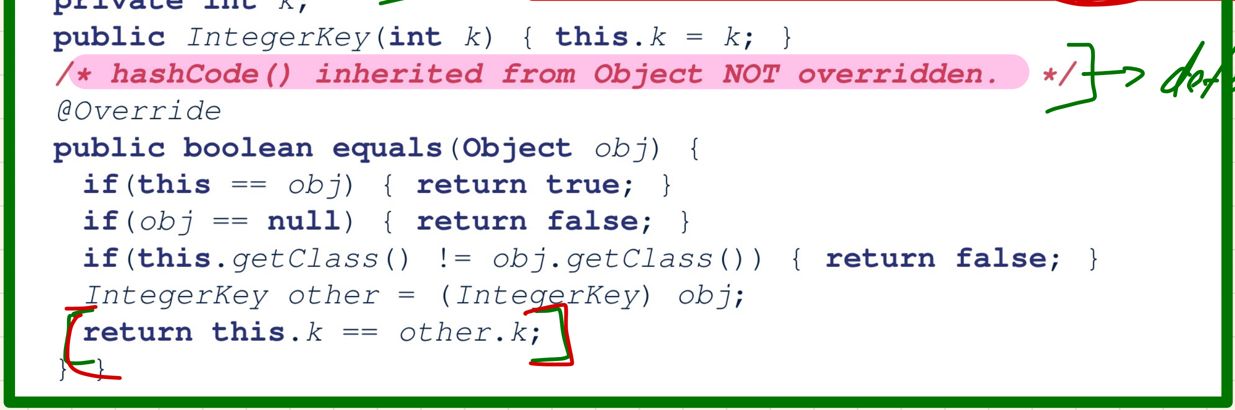
# Inconsistent equals and hashCode





```
@Test
public void testDefaultHashFunction() {
  IntegerKey ik39_1 = new IntegerKey(39);
  IntegerKey ik39_2 = new IntegerKey(39);
  assertTrue(ik39_1.equals(ik39_2));
  assertTrue(ik39_1.hashCode() != ik39_2.hashCode()); }
@Test
public void testHashTable() {
  Hashtable<IntegerKey, String> table = new Hashtable<>();
  IntegerKey k1 = new IntegerKey(39);
  IntegerKey k2 = new IntegerKey(39);
  assertTrue(k1.equals(k2));
  assertTrue(k1.hashCode() != k2.hashCode());
  table.put(k1, "D");
  assertTrue(table.get(k2) == null); }
```

∵ k1 and k2 are different objects



```
public class IntegerKey {
  private int k;
  public IntegerKey(int k) { this.k = k; }
  /* hashCode() inherited from Object NOT overridden. */
  @Override
  public boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false; }
    IntegerKey other = (IntegerKey) obj;
    return this.k == other.k;
} }
```

→ default:
hash code
→ based on
address

# Method Call: Callee vs. Caller

```
class A {
   ...
   void m(T param) {
     /* use of param */
   }
}
```

```
class B {
   ...
   void n(...){
     A co = new A();
     co.m(arg);
   }
}
```

primitive

or

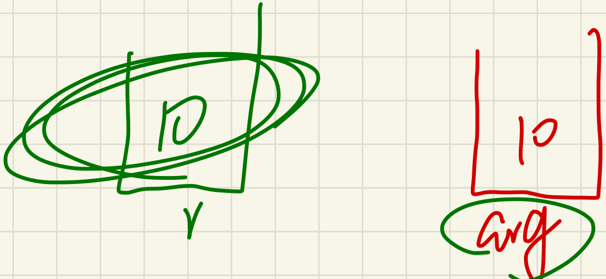
reference

# Call by Value: Primitive Argument

```
class Circle {
  int radius;
  void setRadius(int r) {
    this.radius = r;
  }
}
```

r = arg

```
class CircleUser {
  ...
  Circle c = new Circle();
  int arg = 10;
  c.setRadius(arg);
  }
}
```



# Call by Value: Reference Argument

```
class Circle {
  int radius;
  Circle() {}
  Circle(int r) {
    this.radius = r;
  }
  void setRadius(Circle c) {
    this.radius = c.radius;
  }
}
```

10

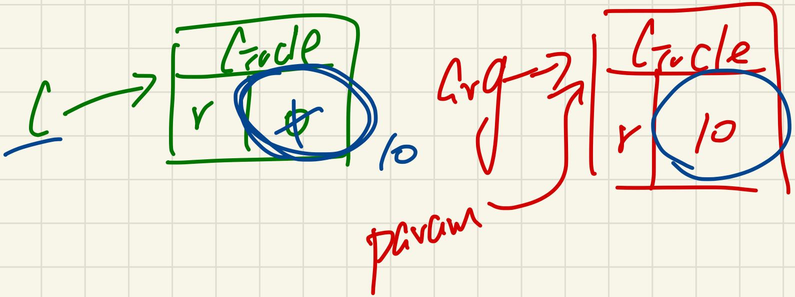
param = arg

param

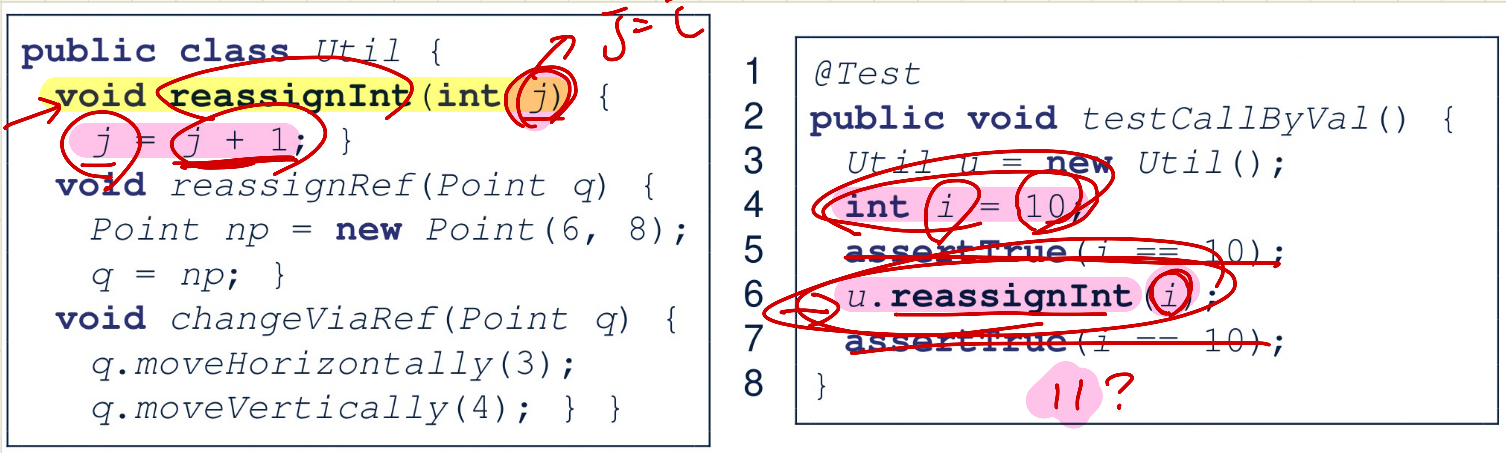
param

```
class CircleUser {
  ...
  Circle c = new Circle();
  Circle arg = new Circle(10);
  c.setRadius(arg);
  }
}
```

reference type



# Call by Value: Re-Assigning Primitive Parameter

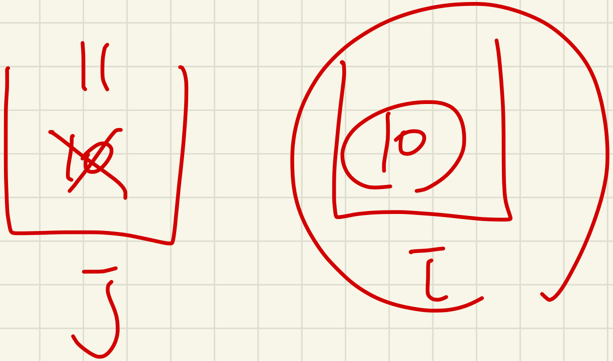


j = i

```
public class Util {
  void reassignInt(int j) {
    j = j + 1; }
  void reassignRef(Point q) {
    Point np = new Point(6, 8);
    q = np; }
  void changeViaRef(Point q) {
    q.moveHorizontally(3);
    q.moveVertically(4); } }
```

```
@Test
public void testCallByVal() {
  Util u = new Util();
  int i = 10;
  ~~assertTrue(i == 10);~~
  u.reassignInt(i);
  ~~assertTrue(i == 10);~~
}
```

11?



# LECTURE 12

MONDAY OCTOBER 21

# Method Call: Callee vs. Caller

```
class A {
  ...
  void m(T param) {
    /* use of param */
  }
}
```

parameter

```
class B {
  ...
  void n(...){
    A co = new A();
    co.m(arg);
  }
}
```

argument

int i = 10;
m(i);

1. Primitive
int bool double char

2. Reference

copy value of arg.

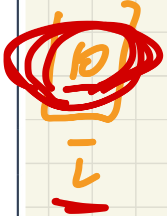
# Call by Value: Re-Assigning Primitive Parameter

```
public class Util {
  void reassignInt(int j) {
    j = j + 1; }
  void reassignRef(Point q) {
    Point np = new Point(6, 8);
    q = np; }
  void changeViaRef(Point q) {
    q.moveHorizontally(3);
    q.moveVertically(4); } }
```

param

```
1  @Test
2  public void testCallByVal() {
3    Util u = new Util();
4    int i = 10;
5    assertTrue(i == 10);
6    u.reassignInt(i);
7    assertTrue(i == 10);
8  }
```

C.O.

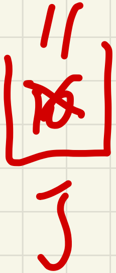


After L6, will i's value be incremented?

j = i ;

parameter

argument



# Call by Value: Re-Assigning Reference Parameter

```
1  public class Util {
2    void reassignInt(int j) {
3      j = j + 1; }
4    void reassignRef(Point q) {
5      Point np = new Point(6, 8);
6      q = np; }
7    void changeViaRef(Point q) {
8      q.moveHorizontally(3);
9      q.moveVertically(4); } }
```

par.

```
1  @Test
2  public void testCallByRef_1() {
3    Util u = new Util();
4    Point p = new Point(3, 4);
5    Point refOfPBefore = p;
6    u.reassignRef(p);
7    assertTrue(p==refOfPBefore);
8    assertTrue(p.x==3 && p.y==4);
9  }
```

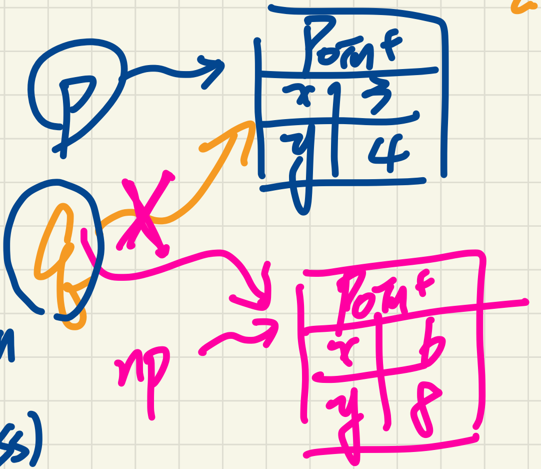
arg

After L6, is p going to point to the same object?

```
class Point {
  int x;
  int y;
  Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  void moveVertically(int y){
    this.y += y;
  }
  void moveHorizontally(int x){
    this.x += x;
  }
}
```

q = p

When the param is of ref. type, do not try to re-assign it (∵ it's useless)



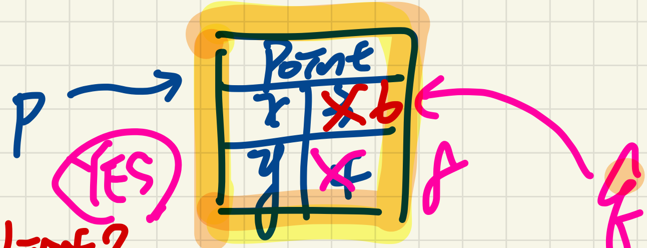
# Call by Value: Calling Mutator on Reference Parameter

```java
public class Util {
  void reassignInt(int j) {
    j = j + 1; }
  void reassignRef(Point q) {
    Point np = new Point(6, 8);
    q = np; }
  void changeViaRef(Point q) {
    q.moveHorizontally(3);
    q.moveVertically(4); } }
```

q=p

```java
1  @Test
2  public void testCallByRef_2() {
3    Util u = new Util();
4    Point p = new Point(3, 4);
5    Point refOfPBefore = p;
6    u.changeViaRef(p);
7    assertTrue(p==refOfPBefore);
8    assertTrue(p.x==6 && p.y==8);
9  }
```

```java
class Point {
  int x;
  int y;
  Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  void moveVertically(int y){
    this.y += y;
  }
  void moveHorizontally(int x){
    this.x += x;
  }
}
```

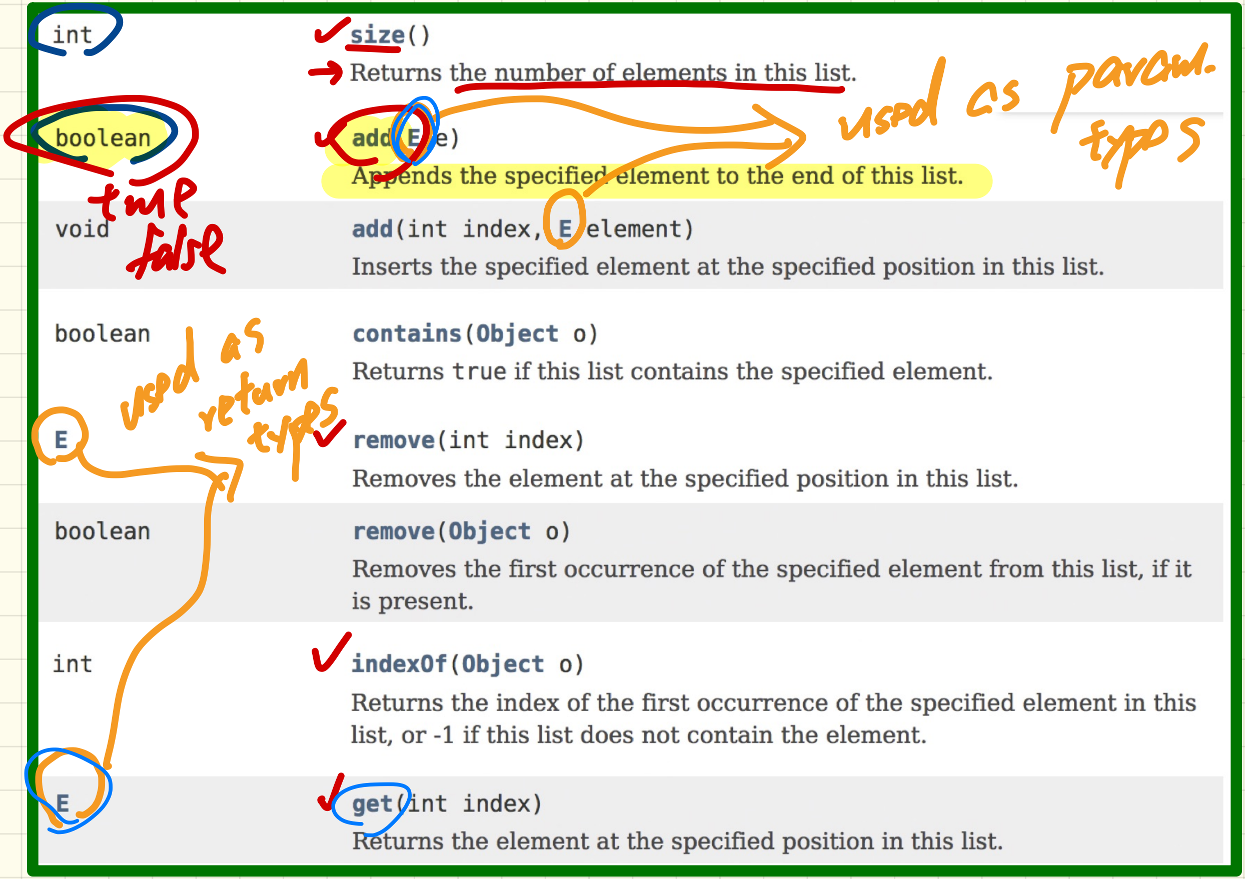


After L6:

(a) Is p pointing to the same object? YES

(b) Is the object pointed to by p initially modified? YES

# API: ArrayList



| | |
|---|---|
| int | size()<br>Returns the number of elements in this list. |
| boolean | add(E e)<br>Appends the specified element to the end of this list. |
| void | add(int index, E element)<br>Inserts the specified element at the specified position in this list. |
| boolean | contains(Object o)<br>Returns true if this list contains the specified element. |
| E | remove(int index)<br>Removes the element at the specified position in this list. |
| boolean | remove(Object o)<br>Removes the first occurrence of the specified element from this list, if it is present. |
| int | indexOf(Object o)<br>Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element. |
| E | get(int index)<br>Returns the element at the specified position in this list. |

used as param. types

true
false

used as return types

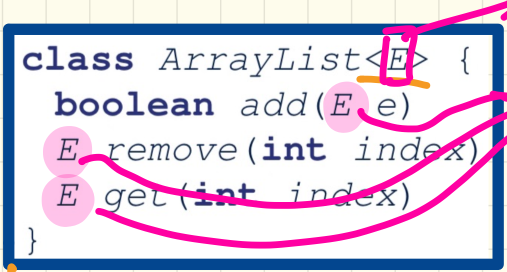
# Generic Parameters: ArrayList

declaring a g.p.

<E>

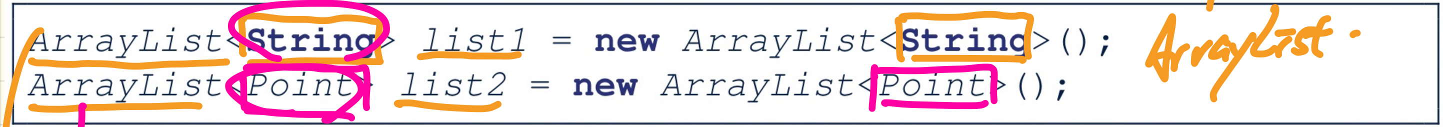
usages of E.

generic parameter for some type that will be instantiated by users of ArrayList.



```
class ArrayList<E> {
  boolean add(E e)
  E remove(int index)
  E get(int index)
}
```

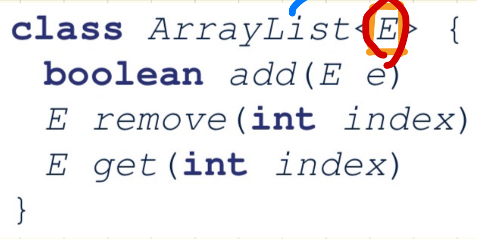
# Caller of ArrayList



```
ArrayList<String> list1 = new ArrayList<String>();
ArrayList<Point> list2 = new ArrayList<Point>();
```

user1

user2



```
class ArrayList<E> {
  boolean add(E e)
  E remove(int index)
  E get(int index)
}
```

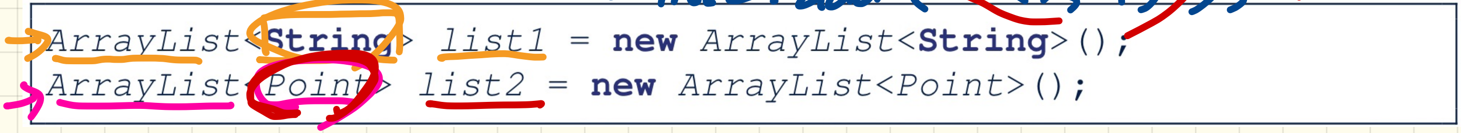
ArrayList<Object> list3 =
new - - ;

① list1.add(new Point(3,4)); ✗

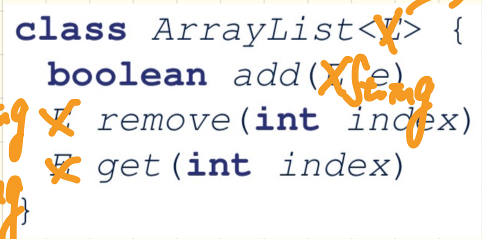
② list1.add("(3,4)");

③ list2.add(new Point(3,4));

④ list2.add("(3,4)"); ✗



```
ArrayList<String> list1 = new ArrayList<String>();
ArrayList<Point> list2 = new ArrayList<Point>();
```



```
class ArrayList<E> {
  boolean add(E e)
  E remove(int index)
  E get(int index)
}
```

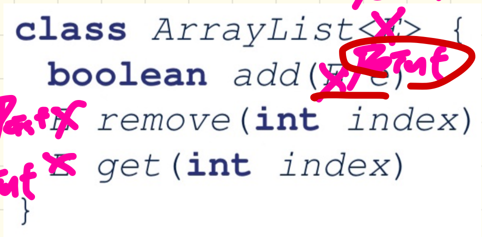
String

class ArrayList<~~E~~ String>

boolean add(~~E~~ String e)

String ~~E~~ remove(int index)

String ~~E~~ get(int index)



```
class ArrayList<E> {
  boolean add(E e)
  E remove(int index)
  E get(int index)
}
```

Point

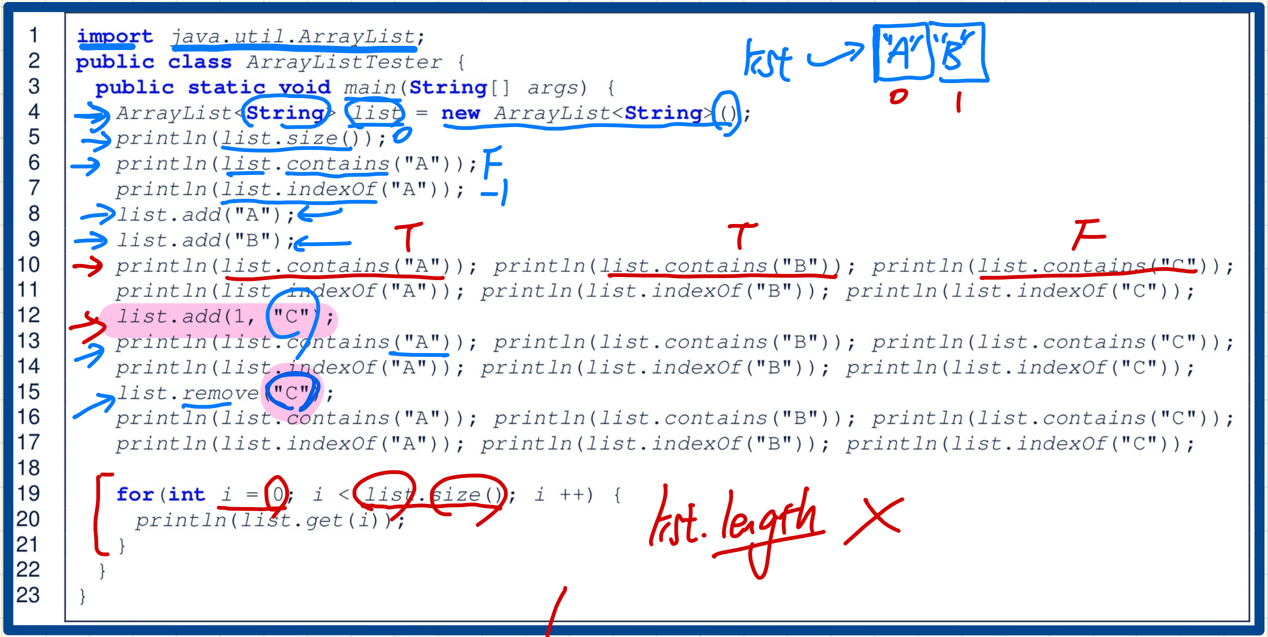
class ArrayList<~~E~~ Point>

boolean add(~~E~~ Point e)

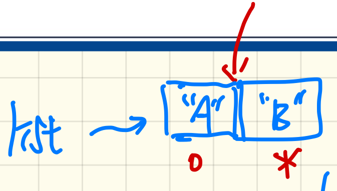
Point ~~E~~ remove(int index)

Point ~~E~~ get(int index)

# Use of ArrayList



```
import java.util.ArrayList;
public class ArrayListTester {
 public static void main(String[] args) {
  ArrayList<String> list = new ArrayList<String>();
  println(list.size());
  println(list.contains("A"));
  println(list.indexOf("A"));
  list.add("A");
  list.add("B");
  println(list.contains("A")); println(list.contains("B")); println(list.contains("C"));
  println(list.indexOf("A")); println(list.indexOf("B")); println(list.indexOf("C"));
  list.add(1, "C");
  println(list.contains("A")); println(list.contains("B")); println(list.contains("C"));
  println(list.indexOf("A")); println(list.indexOf("B")); println(list.indexOf("C"));
  list.remove("C");
  println(list.contains("A")); println(list.contains("B")); println(list.contains("C"));
  println(list.indexOf("A")); println(list.indexOf("B")); println(list.indexOf("C"));

  for(int i = 0; i < list.size(); i ++) {
   println(list.get(i));
  }
 }
}
```



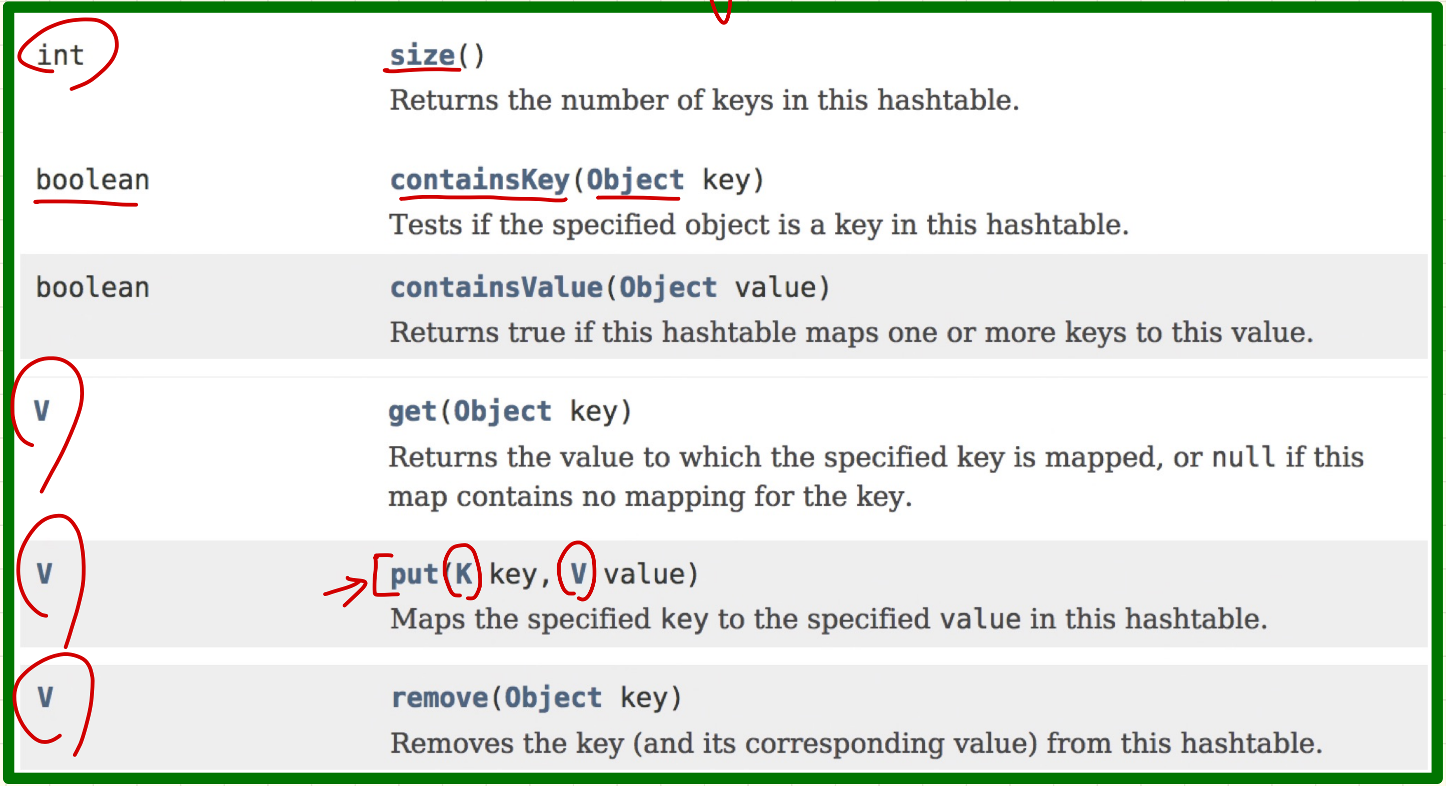
# Hash Table

- 2-column table
- keys contain no duplicates
- Values may contain duplicates
- A key is used to identify a unique row

| keys | values |
|---|---|
| "Alan" | "A" |
| "Mark" | "B+" |
| "Tom" | "A" |
| ~~"Mark"~~ | |

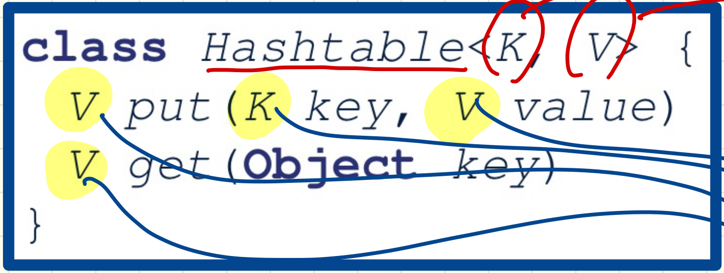
# API: HashTable

two generic param:
K & V



| | |
|---|---|
| int | size()<br>Returns the number of keys in this hashtable. |
| boolean | containsKey(Object key)<br>Tests if the specified object is a key in this hashtable. |
| boolean | containsValue(Object value)<br>Returns true if this hashtable maps one or more keys to this value. |
| V | get(Object key)<br>Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key. |
| V | put(K key, V value)<br>Maps the specified key to the specified value in this hashtable. |
| V | remove(Object key)<br>Removes the key (and its corresponding value) from this hashtable. |

# Generic Parameters: Hashtable



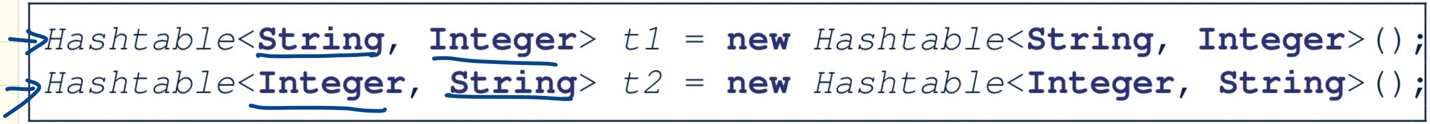
```
class Hashtable<K, V> {
  V put(K key, V value)
  V get(Object key)
}
```

generic parameters

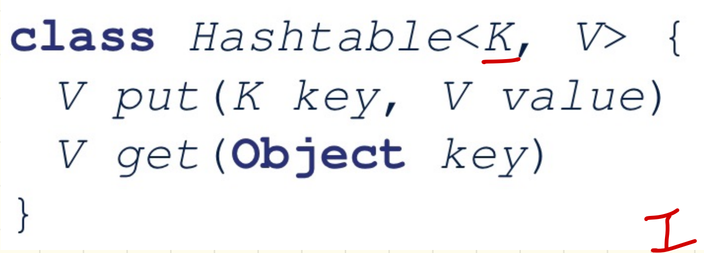
< . - - - >

usages of g.p.

# Caller of Hashtable

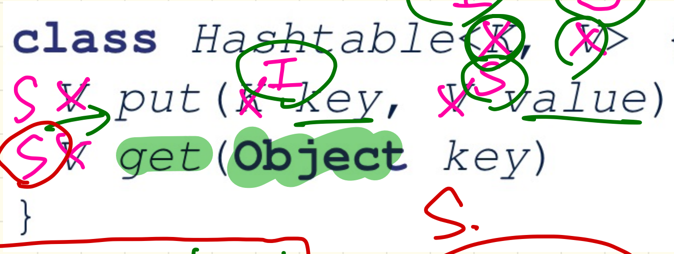


```
Hashtable<String, Integer> t1 = new Hashtable<String, Integer>();
Hashtable<Integer, String> t2 = new Hashtable<Integer, String>();
```



```
class Hashtable<K, V> {
  V put(K key, V value)
  V get(Object key)
}
```

I



```
class Hashtable<K, V> {
  V put(K key, V value)
  V get(Object key)
}
```

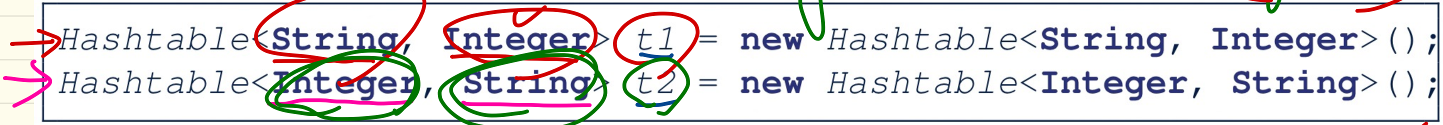
I S

S ~~V~~ put(~~K~~ I key, ~~V~~ S value)

S ~~V~~ get(Object key)

S.

t1.get("alan") vs. t2.get(34)



```
Hashtable<String, Integer> t1 = new Hashtable<String, Integer>();
Hashtable<Integer, String> t2 = new Hashtable<Integer, String>();
```

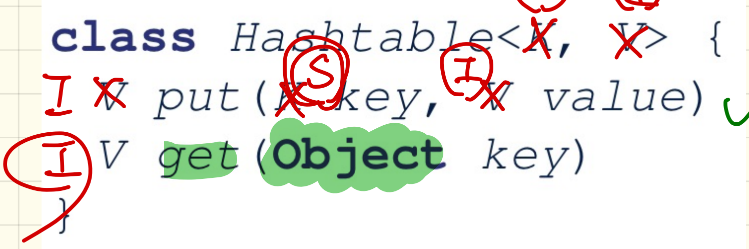
① t1.put("alan", 34); ✓

② t1.put(34, "alan"); ✗

③ t2.put("alan", 34); ✗

④ t2.put(34, "alan"); ✓

S I

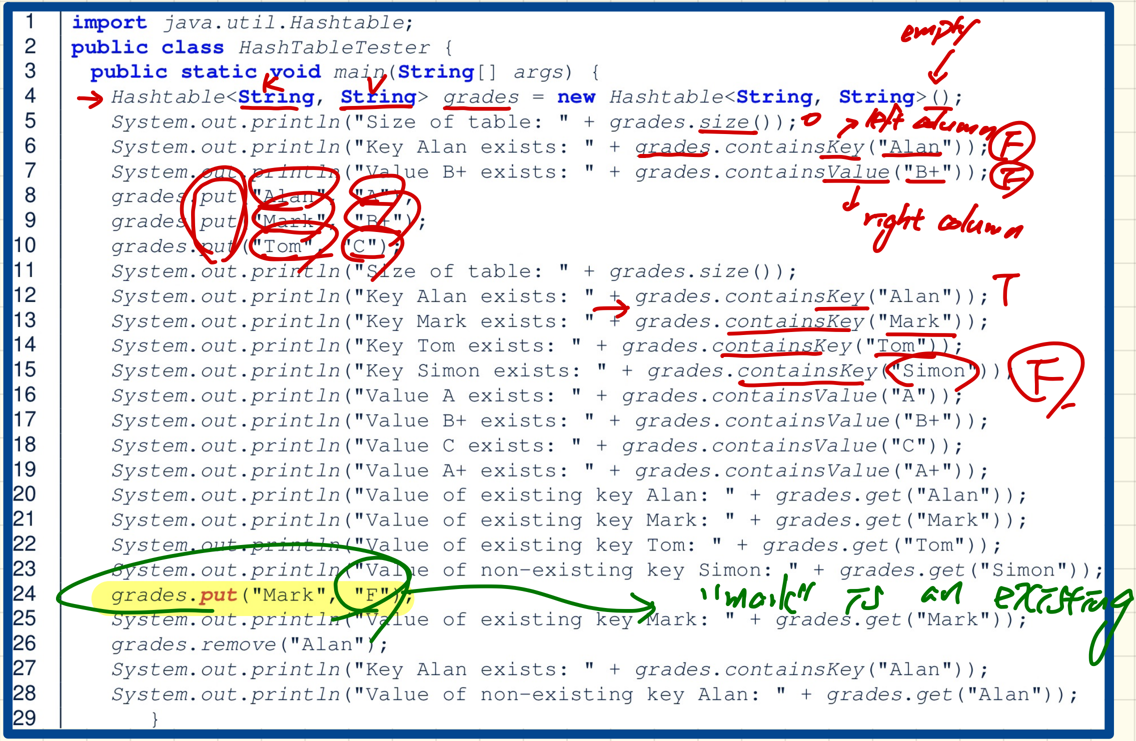


```
class Hashtable<K, V> {
  V put(K key, V value)
  V get(Object key)
}
```

I ~~V~~ put(~~K~~ S key, I ~~V~~ value)

I V get(Object key)

# Use of HashTable



```java
import java.util.Hashtable;
public class HashTableTester {
  public static void main(String[] args) {
    Hashtable<String, String> grades = new Hashtable<String, String>();
    System.out.println("Size of table: " + grades.size());
    System.out.println("Key Alan exists: " + grades.containsKey("Alan"));
    System.out.println("Value B+ exists: " + grades.containsValue("B+"));
    grades.put("Alan", "A");
    grades.put("Mark", "B+");
    grades.put("Tom", "C");
    System.out.println("Size of table: " + grades.size());
    System.out.println("Key Alan exists: " + grades.containsKey("Alan"));
    System.out.println("Key Mark exists: " + grades.containsKey("Mark"));
    System.out.println("Key Tom exists: " + grades.containsKey("Tom"));
    System.out.println("Key Simon exists: " + grades.containsKey("Simon"));
    System.out.println("Value A exists: " + grades.containsValue("A"));
    System.out.println("Value B+ exists: " + grades.containsValue("B+"));
    System.out.println("Value C exists: " + grades.containsValue("C"));
    System.out.println("Value A+ exists: " + grades.containsValue("A+"));
    System.out.println("Value of existing key Alan: " + grades.get("Alan"));
    System.out.println("Value of existing key Mark: " + grades.get("Mark"));
    System.out.println("Value of existing key Tom: " + grades.get("Tom"));
    System.out.println("Value of non-existing key Simon: " + grades.get("Simon"));
    grades.put("Mark", "F");
    System.out.println("Value of existing key Mark: " + grades.get("Mark"));
    grades.remove("Alan");
    System.out.println("Key Alan exists: " + grades.containsKey("Alan"));
    System.out.println("Value of non-existing key Alan: " + grades.get("Alan"));
      }
```

empty

0 → left column

F

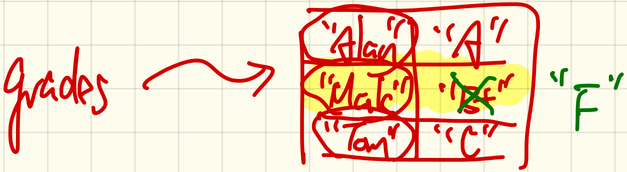
F

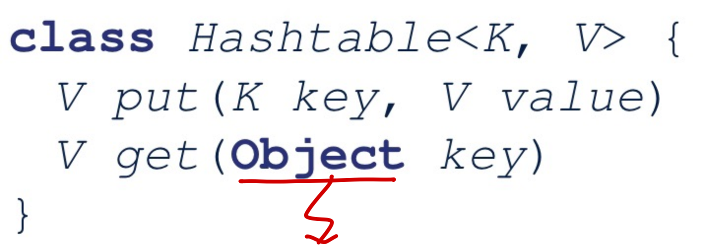
right column

T

F

"Mark" is an existing key, overwrite its value.





```
class Hashtable<K, V> {
  V put(K key, V value)
  V get(Object key)
}
```

K

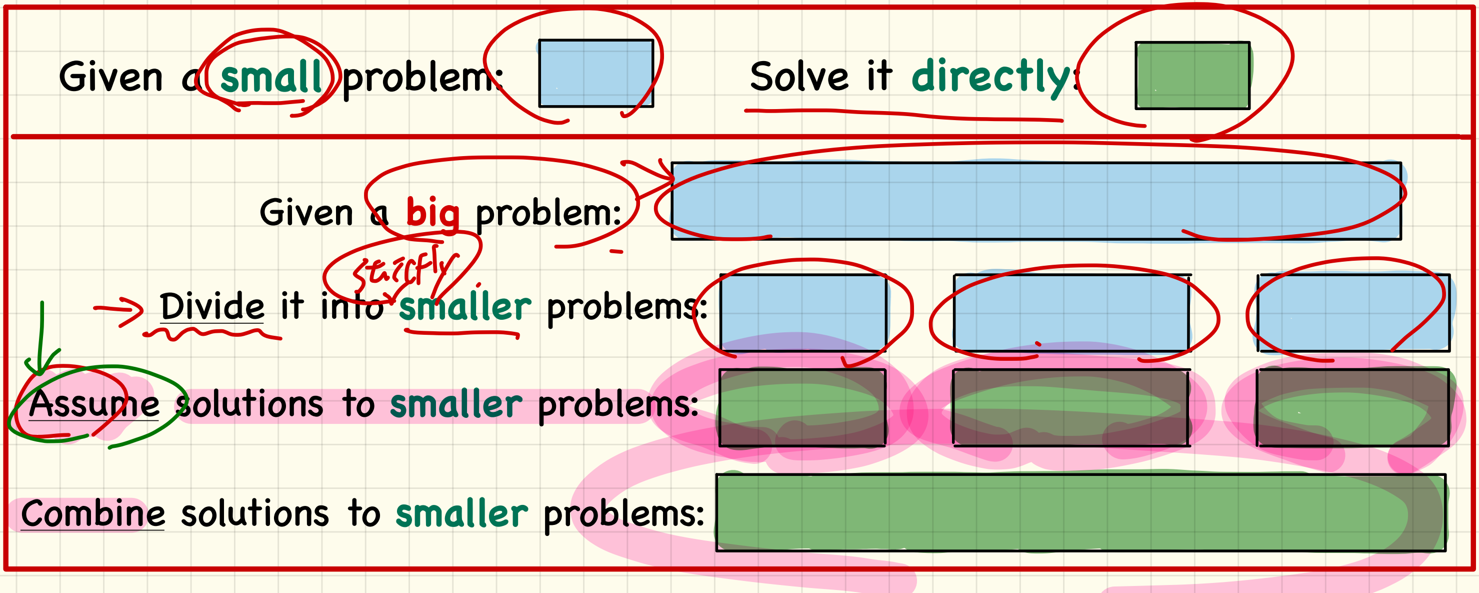
do this if you're defining your own.

gen. p. -

# Solving a Problem **Recursively**

vs. ~~iteratively~~ -

divide-and-conquer.



Given a **small** problem:

Solve it **directly**:

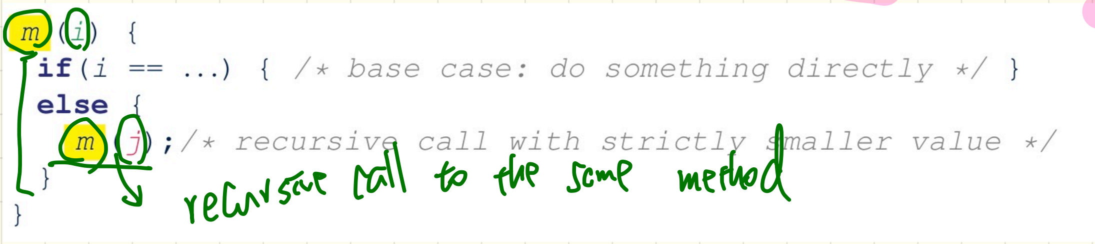
Given a **big** problem:

strictly

Divide it into **smaller** problems:

Assume solutions to **smaller** problems:

Combine solutions to **smaller** problems:



```
m(i) {
  if(i == ...) { /* base case: do something directly */ }
  else {
    m(j);/* recursive call with strictly smaller value */
  }
}
```

recursive call to the same method

# Fibonacci number .

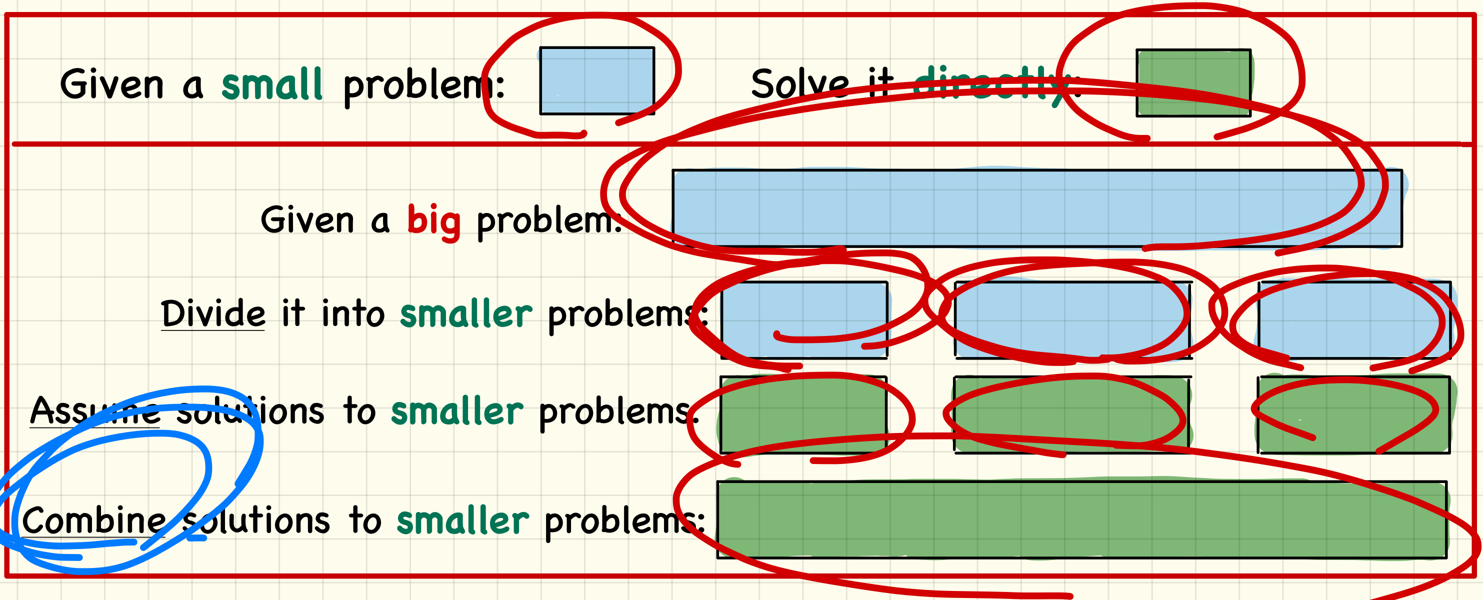
$$\underbrace{1 \quad 1} \quad \underbrace{2 \quad 3} \quad 5 \quad . \quad - \quad -$$

$$\left\{ \begin{array}{ll} \text{factorial} & n! \\ \text{Fibonacci} & \text{number} \end{array} \right\} \rightarrow \begin{array}{l} \text{① loop implement?} \\ \text{② recursive methods ?} \end{array}$$

LECTURE 13

WEDNESDAY OCTOBER 23

# Solving a Problem **Recursively**



Given a **small** problem:

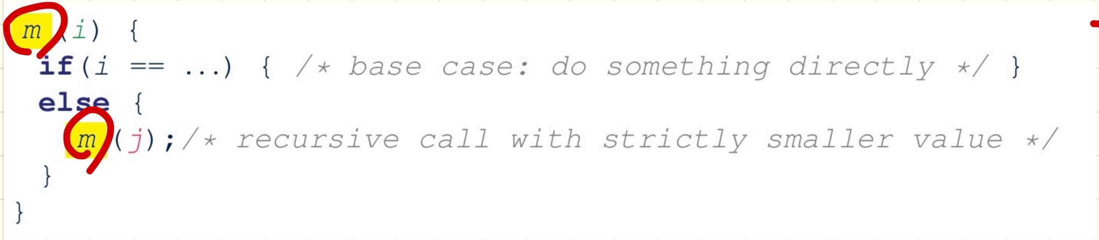
Solve it **directly**:

Given a **big** problem:

Divide it into **smaller** problems:

Assume solutions to **smaller** problems:

Combine solutions to **smaller** problems:



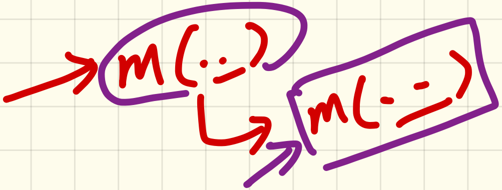
```
m(i) {
  if(i == ...) { /* base case: do something directly */ }
  else {
    m(j);/* recursive call with strictly smaller value */
  }
}
```

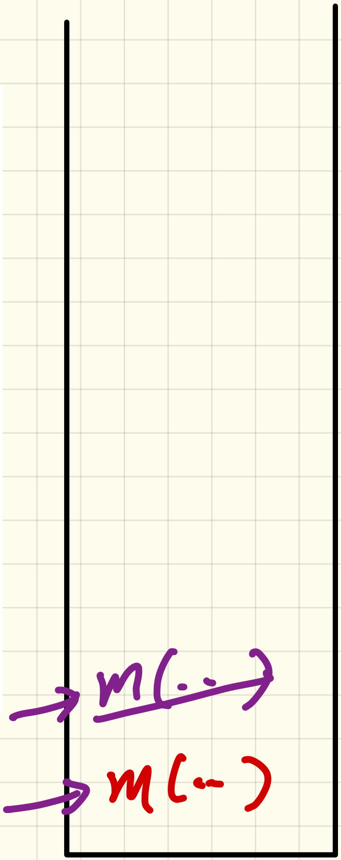
# Tracing Recursion via a Stack

- When a method is called, it is ***activated*** (and becomes *active*) and *pushed* onto the stack.
- When the body of a method makes a (helper) method call, that (helper) method is ***activated*** (and becomes *active*) and *pushed* onto the stack.

  ⇒ The stack contains activation records of all *active* methods.
  - *Top* of stack denotes the current point of execution.
  - Remaining parts of stack are (temporarily) ***suspended***.
- When entire body of a method is executed, stack is *popped*.

  ⇒ The current point of execution is returned to the new *top* of stack (which was ***suspended*** and just became ***active***).
- Execution terminates when the stack becomes *empty*.





Runtime Stack

# Problem

$$n! = \begin{cases} n = 1 & 0 \\ n > 1 & n * (n-1)! \end{cases}$$

size of original prob.

size of a strictly smaller prob.

$$\boxed{4}! = 4 \times 3!$$

size

$$\boxed{4} \times \boxed{3 \times 2 \times 1}$$

size of problem

$3!$

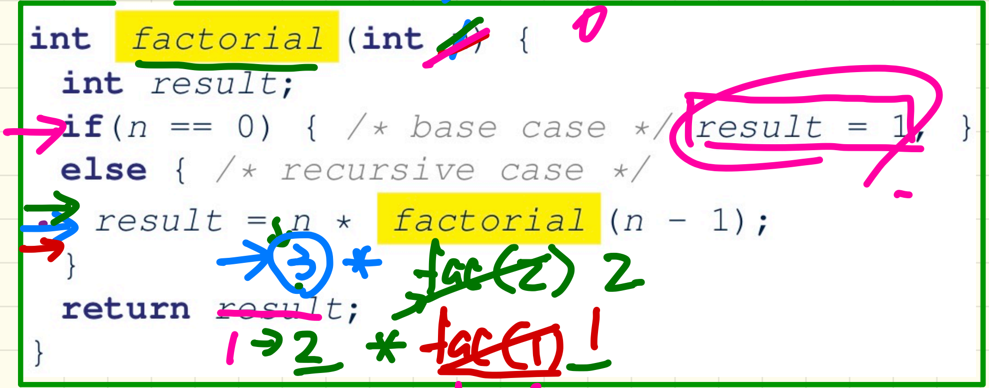
solution to a strictly smaller problem.

# Recursive Solution: factorial

0! = 1

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{if } n \geq 1 \end{cases}$$

3 2 1 0

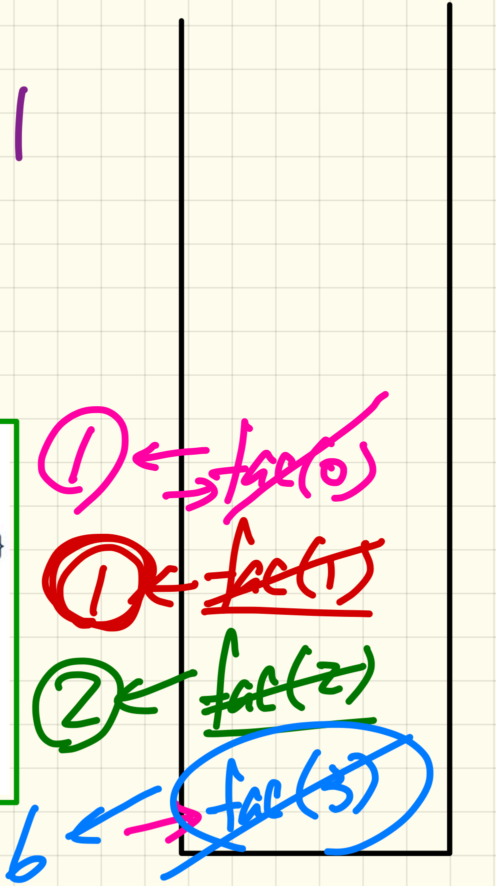


```
int  factorial (int n) {
  int  result;
  if(n == 0) { /* base case */ result = 1; }
  else { /* recursive case */
    result = n * factorial (n - 1);
  }
  return result;
}
```

3 * ~~fac(2)~~ 2

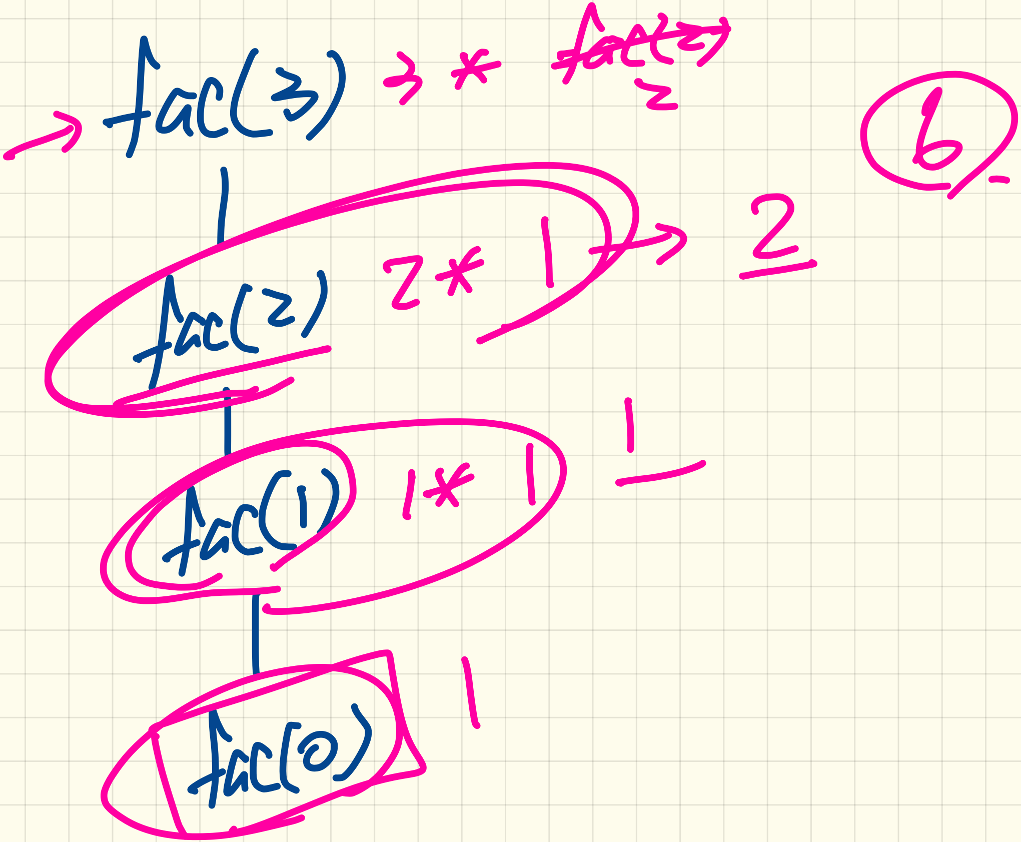
1→2 * ~~fac(1)~~ 1

1 * ~~fac(0)~~ 1

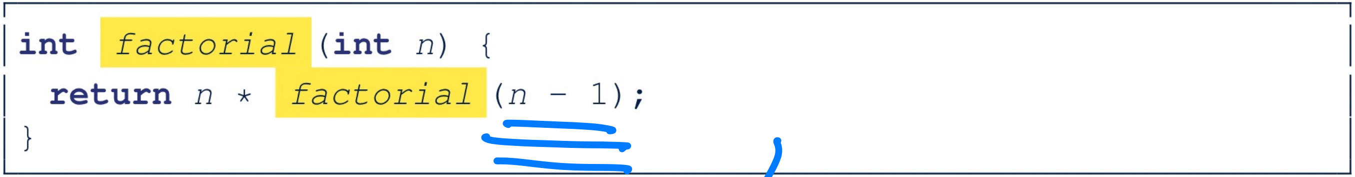


Runtime Stack

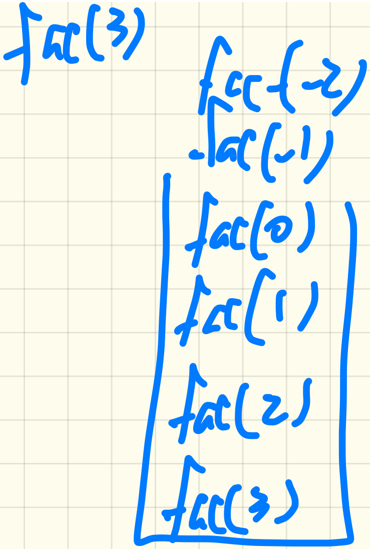
**Example:** factorial(3)


→ fac(3) 3 * ~~fac(2)~~ 2
6.
fac(2) 2 * 1 → 2
fac(1) 1 * 1
1
fac(0)
1

# Common Errors of Recursion (1)



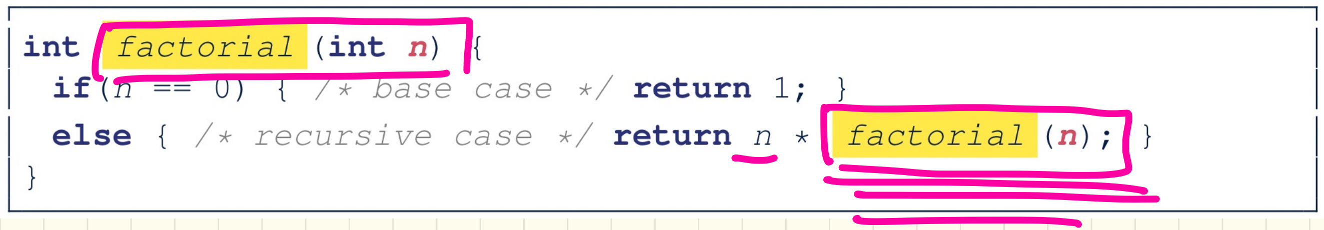
```
int  factorial (int n)  {
  return n *  factorial (n - 1);
}
```



missing bases

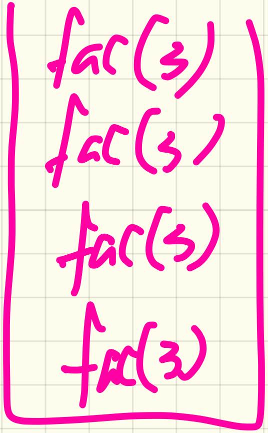
→ no termination

# Common Errors of Recursion (2)



```
int factorial (int n) {
  if(n == 0) { /* base case */ return 1; }
  else { /* recursive case */ return n * factorial (n); }
}
```

fac(3)



|  |  |  |  |  |
|---|---|---|---|---|
|  |  | ✓ | ✓ |  |
| 1. | 1 | 2 | 3 | 5 |
| fib(1) | fib(2) | fib(3) | fib(4) | fib(5) |
|  |  |  | \|\| |  |

fib(3) + fib(2)

# Recursive Solution: Fibonacci Number

$$F_n = \begin{cases} 1 & \text{if } n = 1 \\ 1 & \text{if } n = 2 \\ \boxed{F_{n-1}} + F_{n-2} & \text{if } n > 2 \end{cases}$$

combine

original problem

solution to a S.S. problem

solution to another S.S. problem

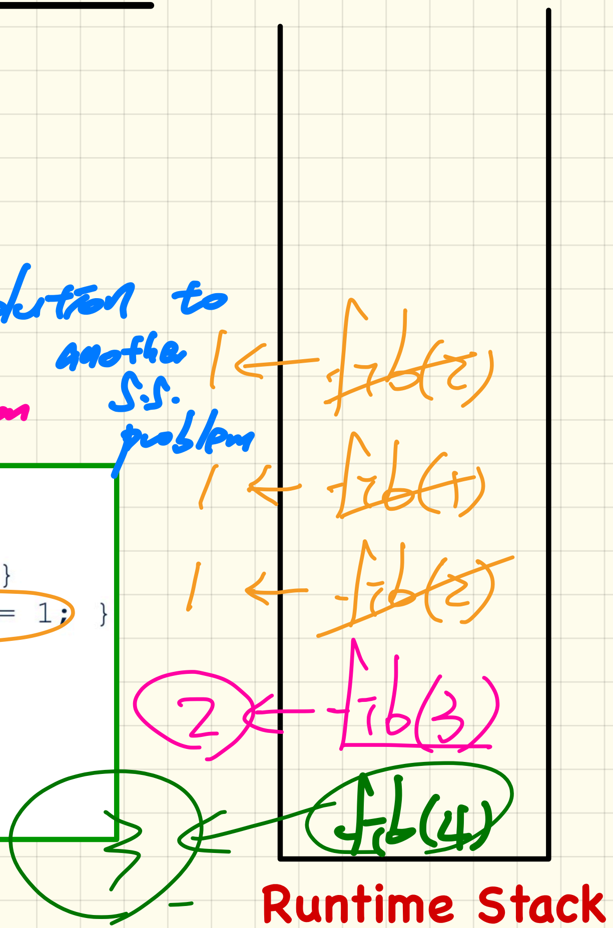
```
int  fib (int n) {
  int result;
  if(n == 1) { /* base case */ result = 1; }
  else if(n == 2) { /* base case */ result = 1; }
  else { /* recursive case */
    result = fib (n - 1) + fib (n - 2);
  }
  return result;
}
```

2 fib(3) + fib(2) 1

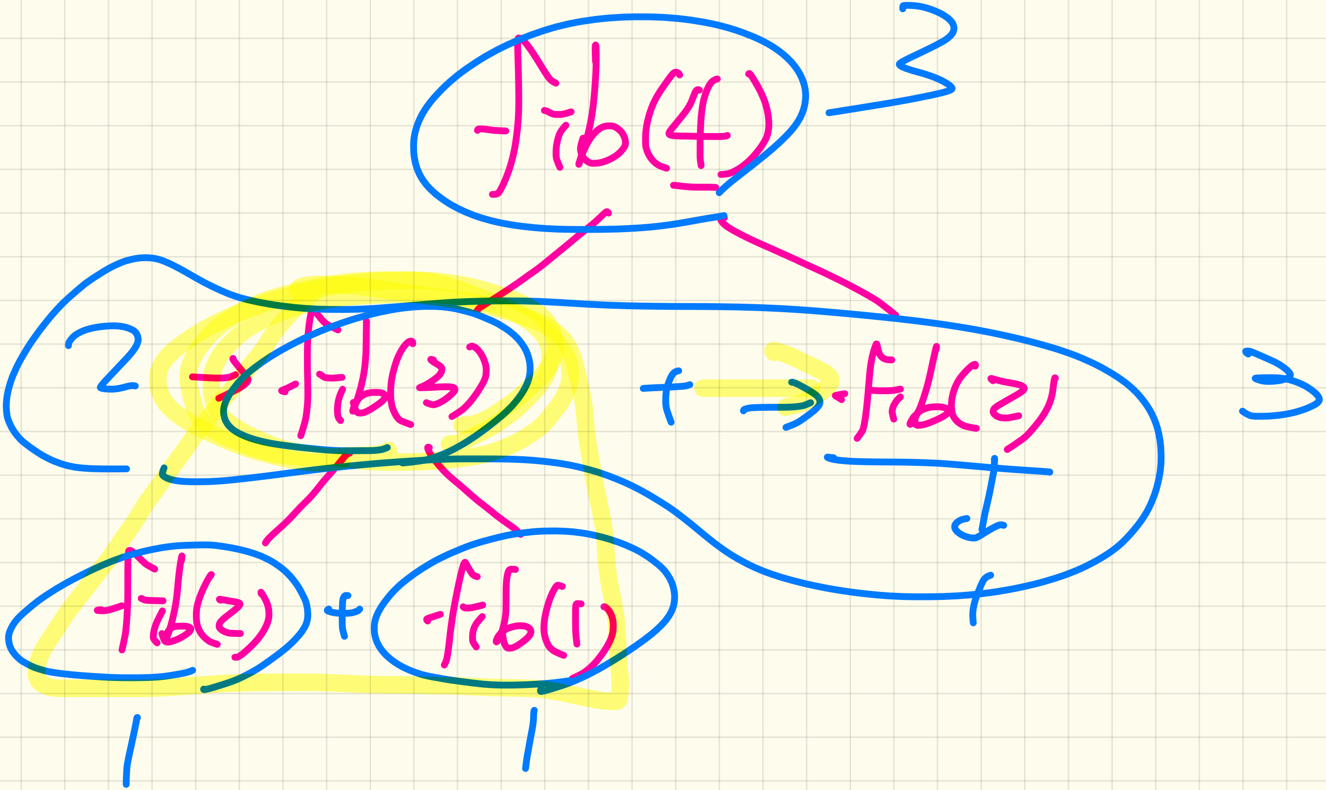
fib(2) + fib(1)

1 1 2

Example: fib(4)



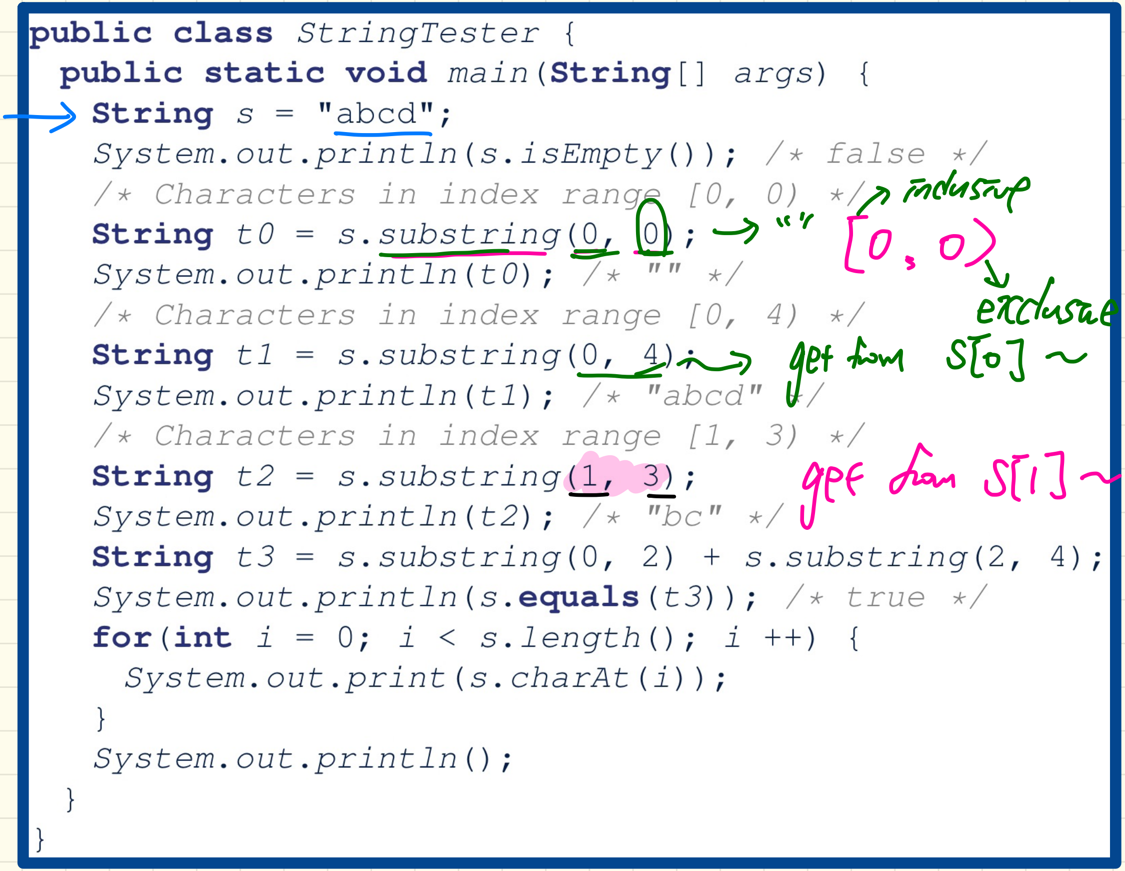
Runtime Stack


fib(4)
3
2
fib(3)
+
fib(2)
3
1
fib(2)
+
fib(1)
1
1
1

# Use of String

0, -1

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 'a' | 'b' | 'c' | 'd' |



```java
public class StringTester {
  public static void main(String[] args) {
    String s = "abcd";
    System.out.println(s.isEmpty()); /* false */
    /* Characters in index range [0, 0) */
    String t0 = s.substring(0, 0);
    System.out.println(t0); /* "" */
    /* Characters in index range [0, 4) */
    String t1 = s.substring(0, 4);
    System.out.println(t1); /* "abcd" */
    /* Characters in index range [1, 3) */
    String t2 = s.substring(1, 3);
    System.out.println(t2); /* "bc" */
    String t3 = s.substring(0, 2) + s.substring(2, 4);
    System.out.println(s.equals(t3)); /* true */
    for(int i = 0; i < s.length(); i ++) {
      System.out.print(s.charAt(i));
    }
    System.out.println();
  }
}
```

→ ""

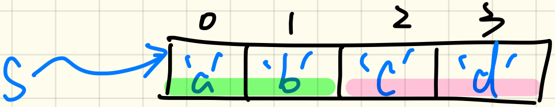
inclusive

[0, 0)

exclusive

get from S[0] ~ S[3]

get from S[1] ~ S[2]



String s

s.substring(0, 2)

+

s.substring(2, $\underset{4}{\text{s.length()}}$)

i is a valid index

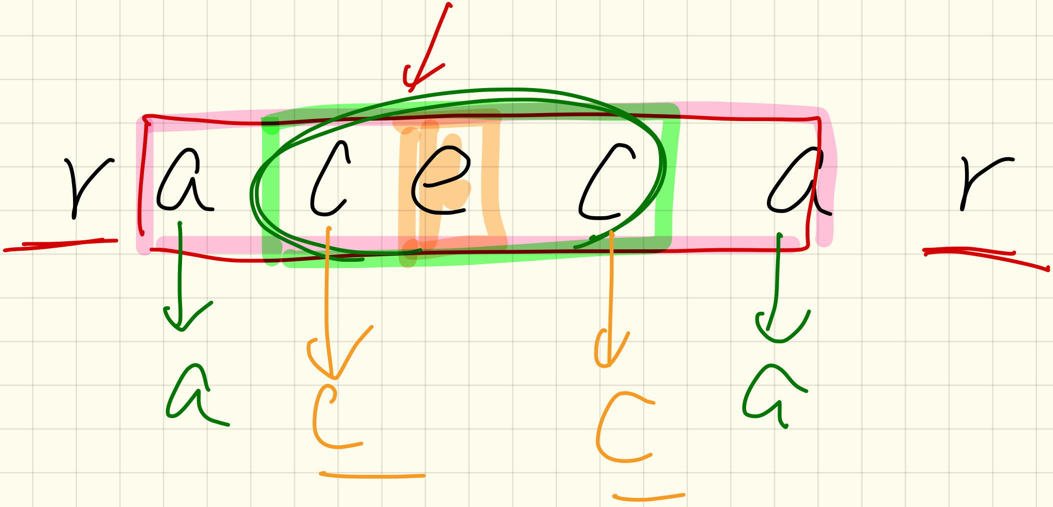
s.substring(0, i)

+

s.substring(i, s.length())

||

s

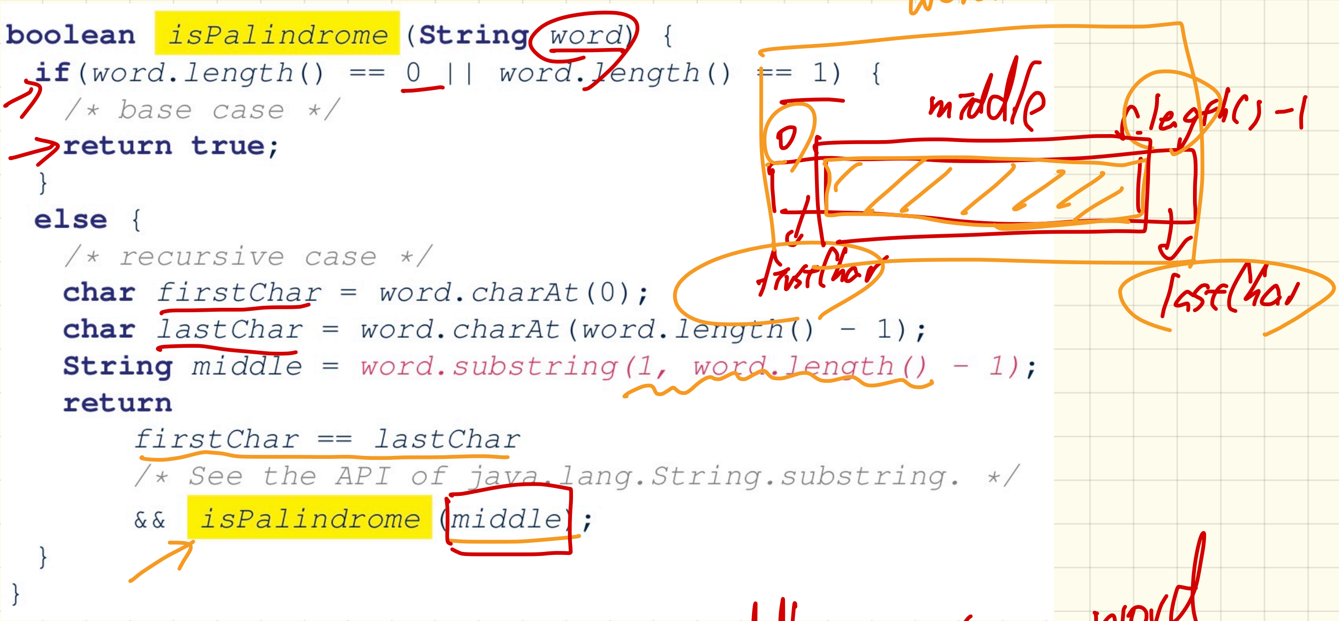

r a c e c a r
a
c
c
a

# Problem: Palindrome

""

" "

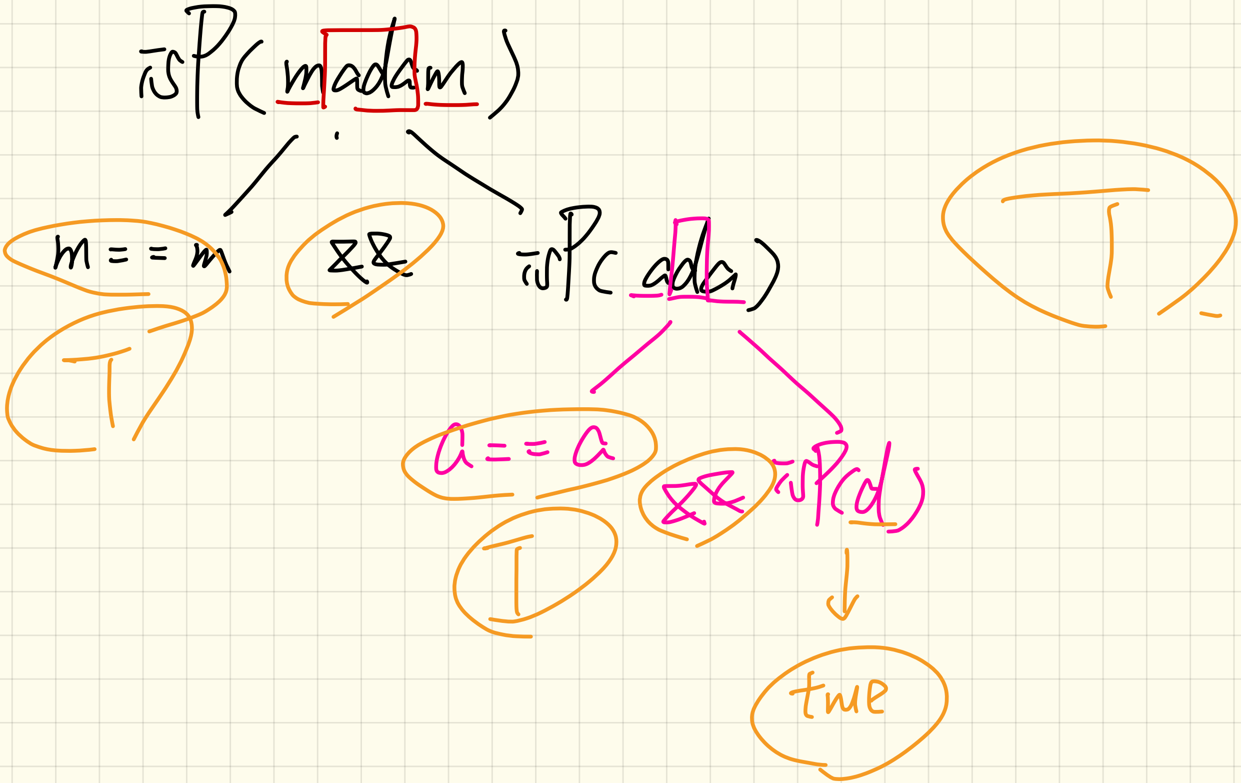
word

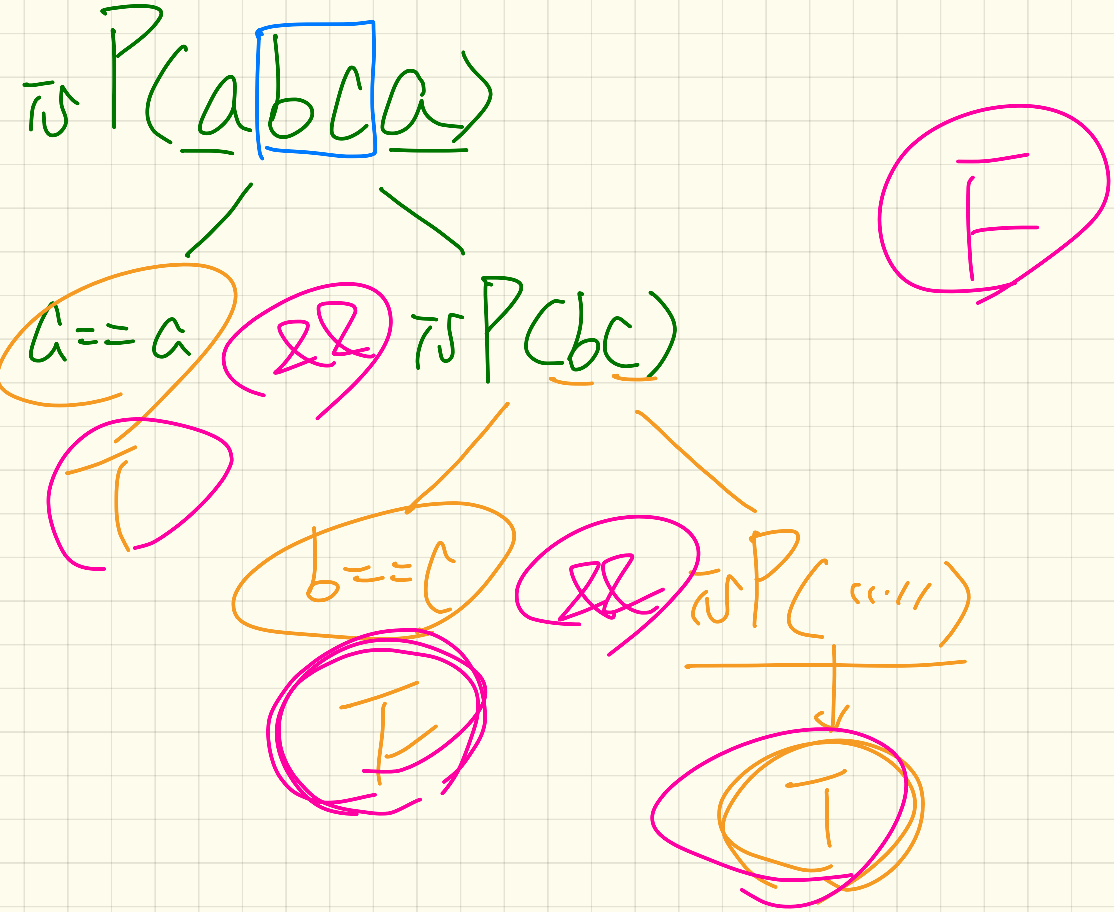


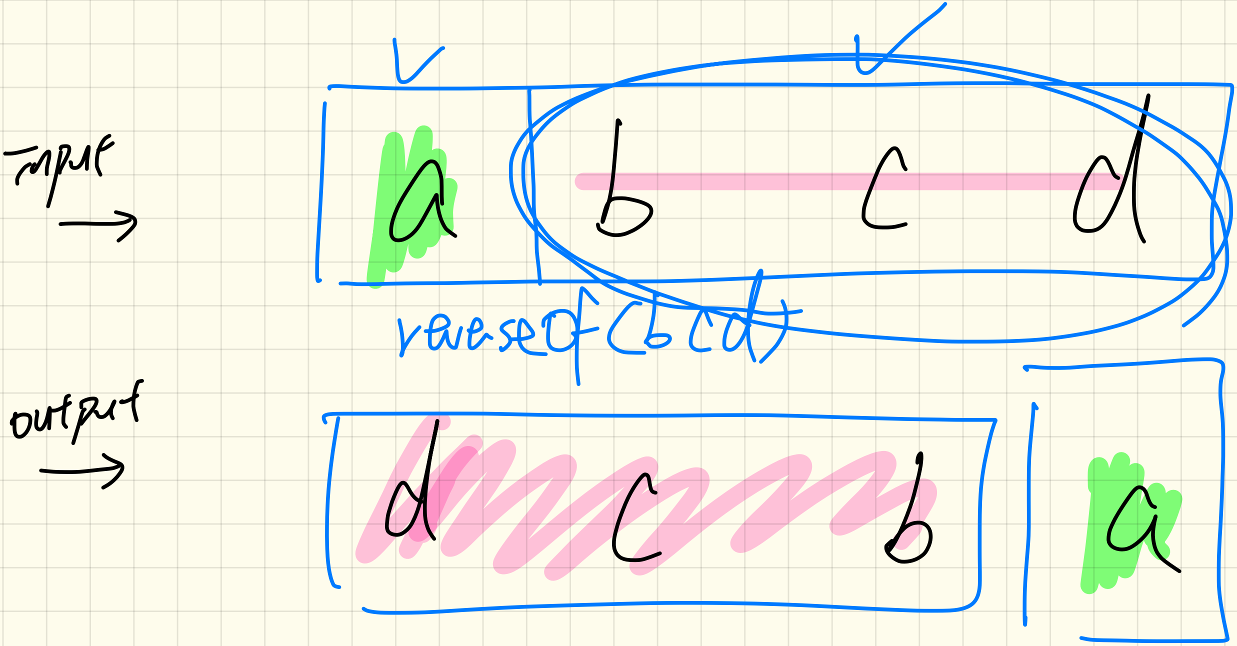
```
boolean isPalindrome (String word) {
  if(word.length() == 0 || word.length() == 1) {
    /* base case */
    return true;
  }
  else {
    /* recursive case */
    char firstChar = word.charAt(0);
    char lastChar = word.charAt(word.length() - 1);
    String middle = word.substring(1, word.length() - 1);
    return
        firstChar == lastChar
        /* See the API of java.lang.String.substring. */
        && isPalindrome (middle);
  }
}
```

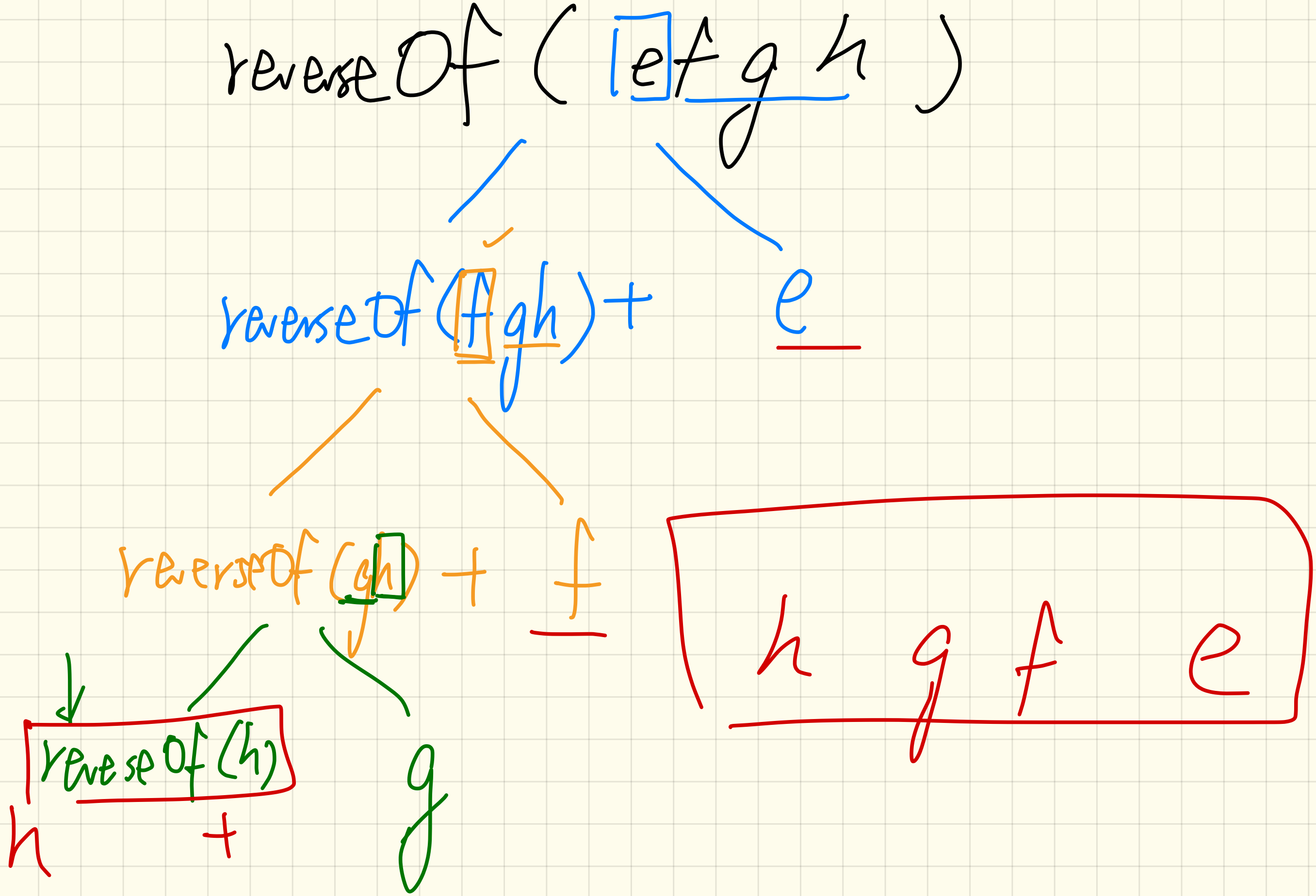
middle vs. word

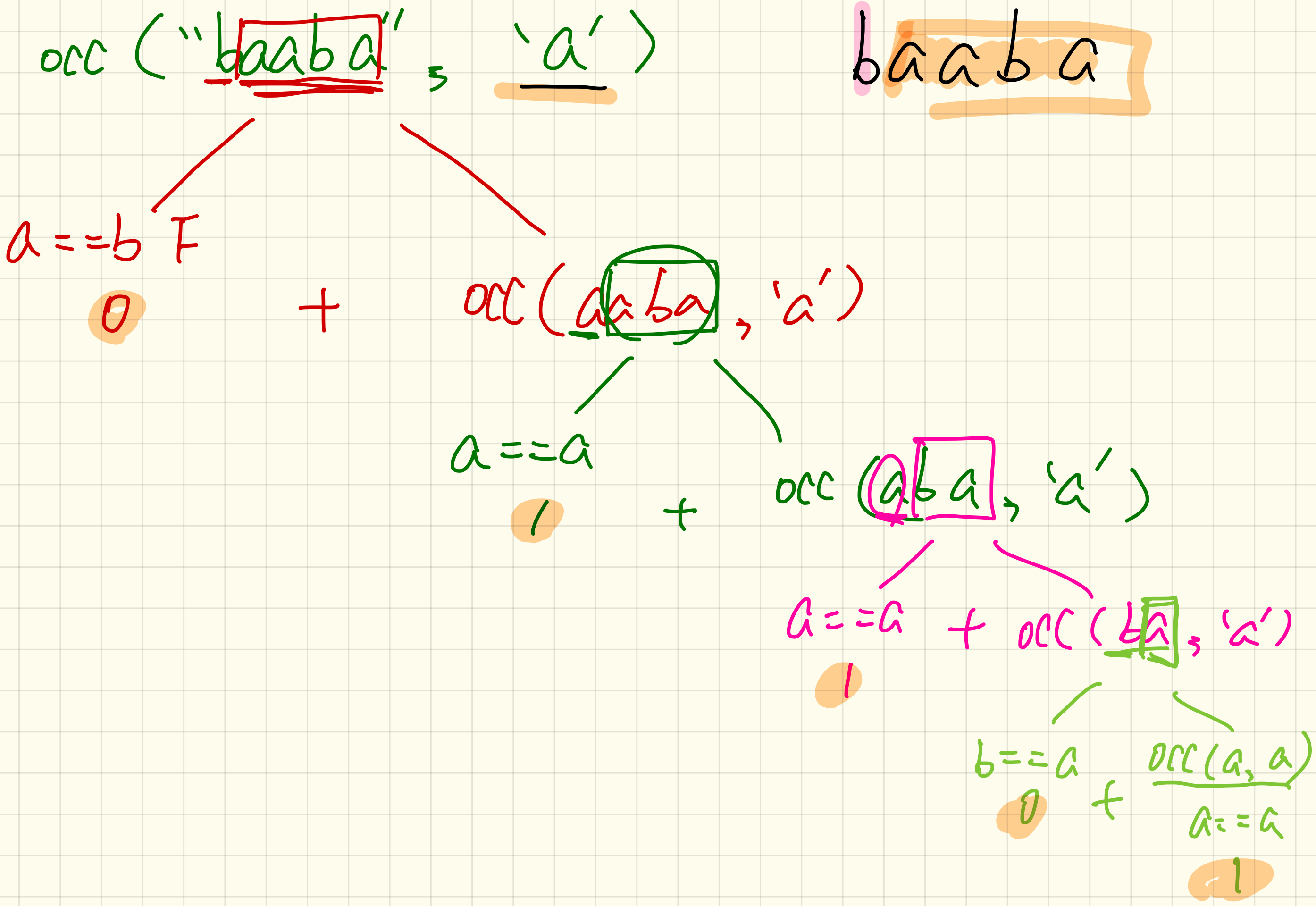
<


isP(madam)
m == m
&&
isP(ada)
T
a == a
&&
isP(d)
T
true
T

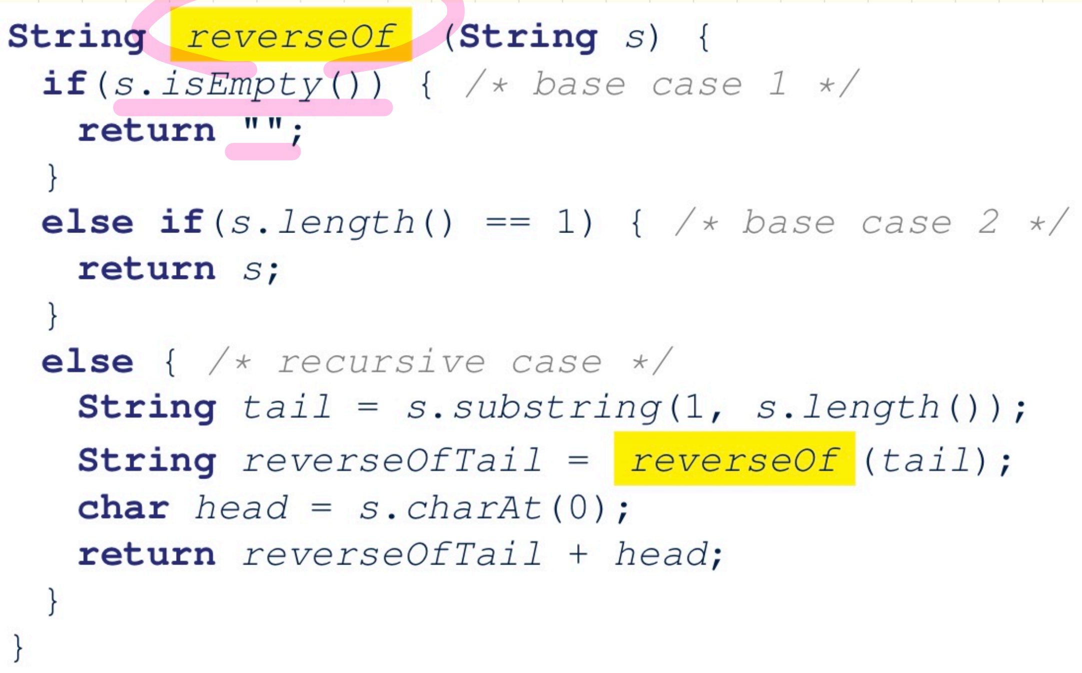

is P(abca)
a==a
&&
is P(bc)
T
b==c
&&
is P(...)
F
T
F


input →
a
b
c
d
reverseOf(bcd)
output →
d
c
b
a


reverseOf(efgh)
reverseOf(fgh) +
e
reverseOf(gh) +
f
reverseOf(h)
g
h
+
h g f e

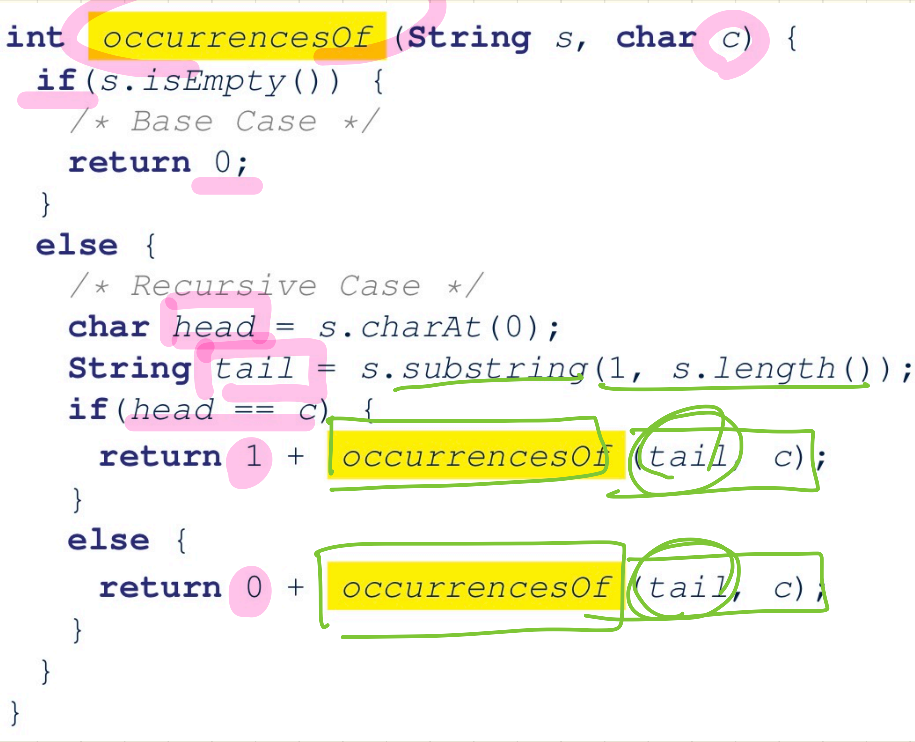

occ("baaba", 'a')
baaba
a==b F
0
+
occ(aaba, 'a')
a==a
1
+
occ(aba, 'a')
a==a
1
+ occ(ba, 'a')
b==a
0
+
occ(a, a)
a==a
1

# Problem: Reverse of a String

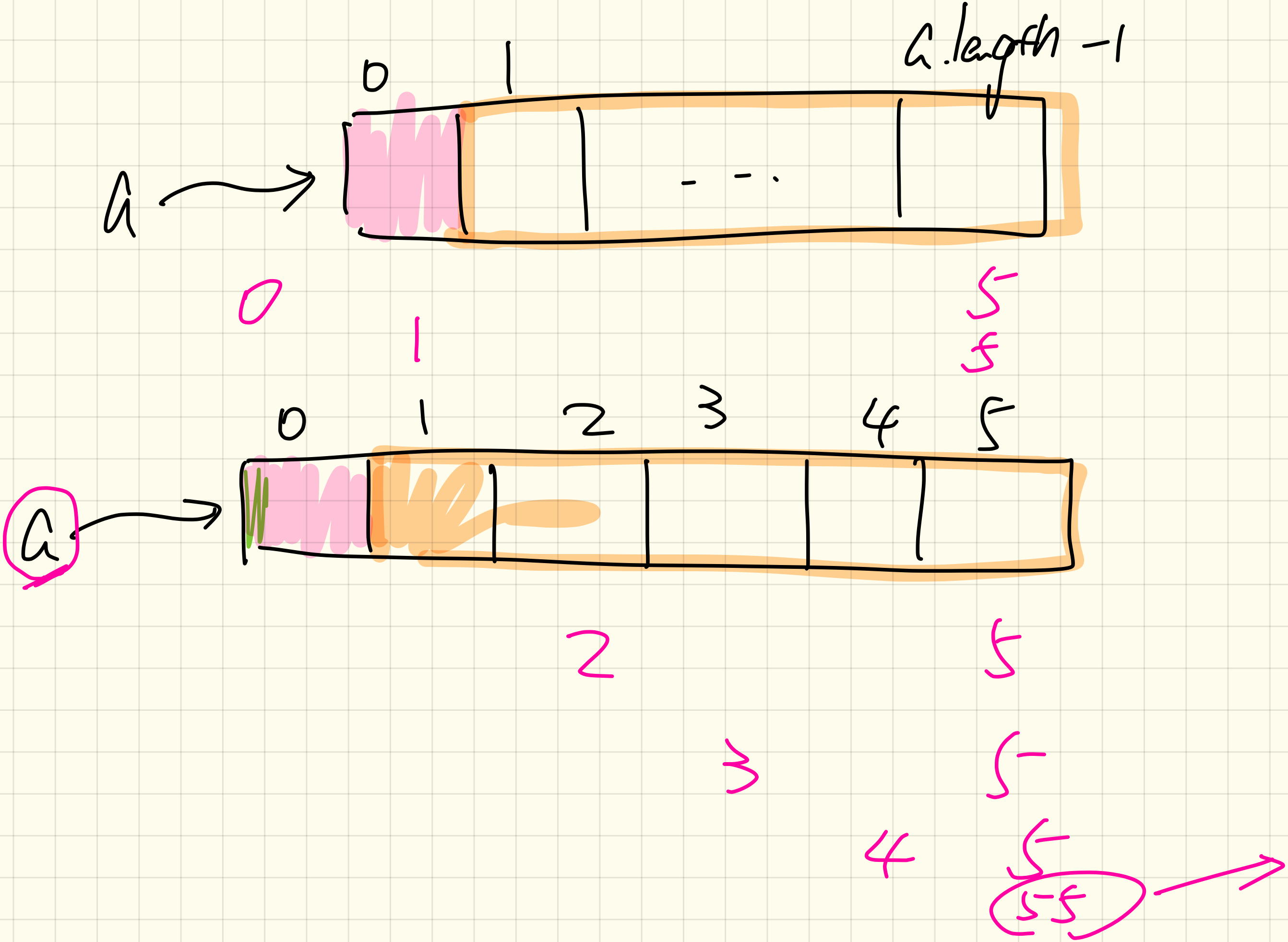


```
String reverseOf (String s) {
  if(s.isEmpty()) { /* base case 1 */
    return "";
  }
  else if(s.length() == 1) { /* base case 2 */
    return s;
  }
  else { /* recursive case */
    String tail = s.substring(1, s.length());
    String reverseOfTail = reverseOf (tail);
    char head = s.charAt(0);
    return reverseOfTail + head;
  }
}
```

# Problem: Number of Occurrences



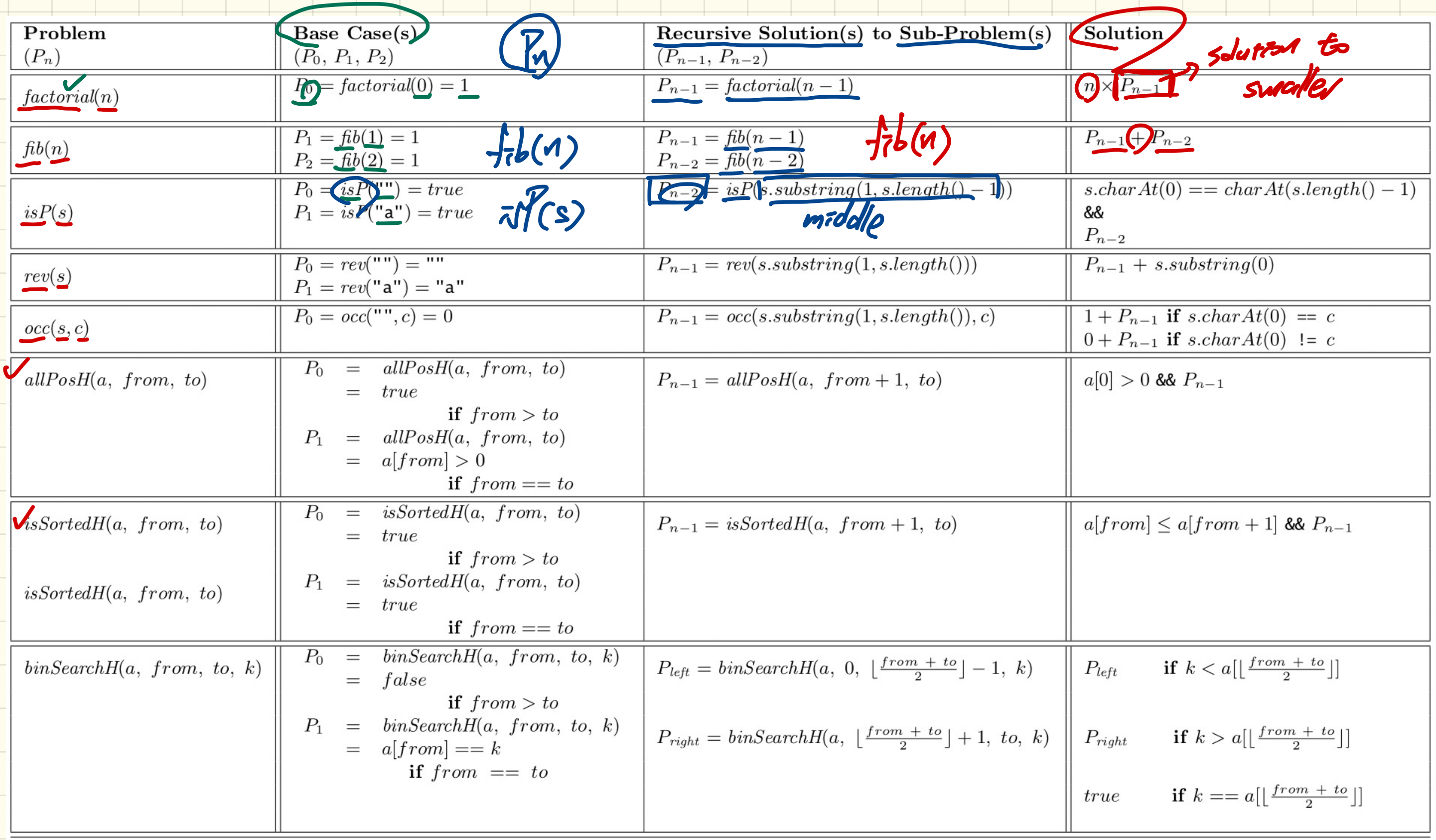
```
int occurrencesOf (String s, char c) {
  if(s.isEmpty()) {
    /* Base Case */
    return 0;
  }
  else {
    /* Recursive Case */
    char head = s.charAt(0);
    String tail = s.substring(1, s.length());
    if(head == c) {
      return 1 + occurrencesOf (tail, c);
    }
    else {
      return 0 + occurrencesOf (tail, c);
    }
  }
}
```


a.length -1
0
1
a
. . .
0
1
5
5
0
1
2
3
4
5
a
2
5
3
5
4
5
5 5

# LECTURE 14

MONDAY OCTOBER 28

# Solving Problems Recursively



| **Problem**<br>$(P_n)$ | **Base Case(s)**<br>$(P_0,\ P_1,\ P_2)$ $P_n$ | **Recursive Solution(s) to Sub-Problem(s)**<br>$(P_{n-1},\ P_{n-2})$ | **Solution** |
|---|---|---|---|
| ✓ $factorial(n)$ | $P_0 = factorial(0) = 1$ | $P_{n-1} = factorial(n-1)$ | $n \times \boxed{P_{n-1}}$<br>solution to smaller |
| $fib(n)$ | $P_1 = fib(1) = 1$<br>$P_2 = fib(2) = 1$<br>fib(n) | $P_{n-1} = fib(n-1)$<br>$P_{n-2} = fib(n-2)$<br>fib(n) | $P_{n-1} + P_{n-2}$ |
| $isP(s)$ | $P_0 = isP(\texttt{""}) = true$<br>$P_1 = isP(\texttt{"a"}) = true$<br>isP(s) | $\boxed{P_{n-2}} = isP(\underbrace{s.substring(1, s.length() - 1)}_{\text{middle}})$ | $s.charAt(0) == charAt(s.length() - 1)$<br>&&<br>$P_{n-2}$ |
| $rev(s)$ | $P_0 = rev(\texttt{""}) = \texttt{""}$<br>$P_1 = rev(\texttt{"a"}) = \texttt{"a"}$ | $P_{n-1} = rev(s.substring(1, s.length()))$ | $P_{n-1} + s.substring(0)$ |
| $occ(s, c)$ | $P_0 = occ(\texttt{""}, c) = 0$ | $P_{n-1} = occ(s.substring(1, s.length()), c)$ | $1 + P_{n-1}$ **if** $s.charAt(0)\ ==\ c$<br>$0 + P_{n-1}$ **if** $s.charAt(0)\ !=\ c$ |
| ✓ $allPosH(a,\ from,\ to)$ | $P_0 = allPosH(a,\ from,\ to)$<br>$= true$<br>**if** $from > to$<br>$P_1 = allPosH(a,\ from,\ to)$<br>$= a[from] > 0$<br>**if** $from == to$ | $P_{n-1} = allPosH(a,\ from + 1,\ to)$ | $a[0] > 0$ && $P_{n-1}$ |
| ✓ $isSortedH(a,\ from,\ to)$<br><br>$isSortedH(a,\ from,\ to)$ | $P_0 = isSortedH(a,\ from,\ to)$<br>$= true$<br>**if** $from > to$<br>$P_1 = isSortedH(a,\ from,\ to)$<br>$= true$<br>**if** $from == to$ | $P_{n-1} = isSortedH(a,\ from + 1,\ to)$ | $a[from] \leq a[from + 1]$ && $P_{n-1}$ |
| $binSearchH(a,\ from,\ to,\ k)$ | $P_0 = binSearchH(a,\ from,\ to,\ k)$<br>$= false$<br>**if** $from > to$<br>$P_1 = binSearchH(a,\ from,\ to,\ k)$<br>$= a[from] == k$<br>**if** $from\ ==\ to$ | $P_{left} = binSearchH(a,\ 0,\ \lfloor \frac{from + to}{2} \rfloor - 1,\ k)$<br><br>$P_{right} = binSearchH(a,\ \lfloor \frac{from + to}{2} \rfloor + 1,\ to,\ k)$ | $P_{left}$ **if** $k < a[\lfloor \frac{from + to}{2} \rfloor]$<br><br>$P_{right}$ **if** $k > a[\lfloor \frac{from + to}{2} \rfloor]$<br><br>$true$ **if** $k == a[\lfloor \frac{from + to}{2} \rfloor]$ |

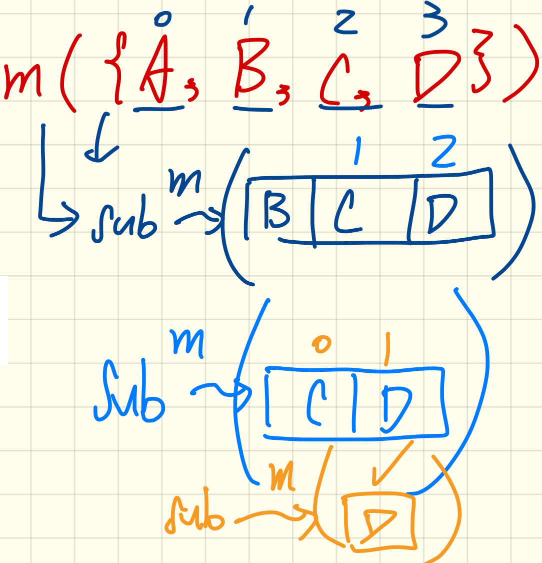
# Recursion on an Array: Passing new Sub-Arrays

```
void m(int[] a) {
  if(a.length == 0) { /* base case */ }
  else if(a.length == 1) { /* base case */ }
  else {
    int[] sub = new int[a.length - 1];
    for(int i = 1; i < a.length; i ++) { sub[0] = a[i - 1]; }
    m(sub) } }
```

Say a1 = {}, consider m(a1)

Say a2 = {A}, consider m(a2)

Say a3 = {A, B, C, D}, consider m(a3)



# Recursion on an Array: Passing Same Array Reference

```
void m(int[] a, int from, int to) {
  if(from > to) { /* base case */ }
  else if(from == to) { /* base case */ }
  else { m(a, from + 1, to) } }
```

✓

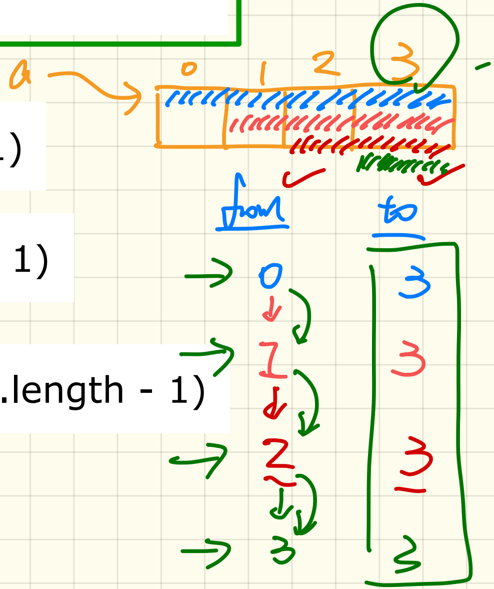
Say a1 = {}, consider m(a1, 0, a1.length - 1)

✓

Say a2 = {A}, consider m(a2, 0, a2.length - 1)

✓

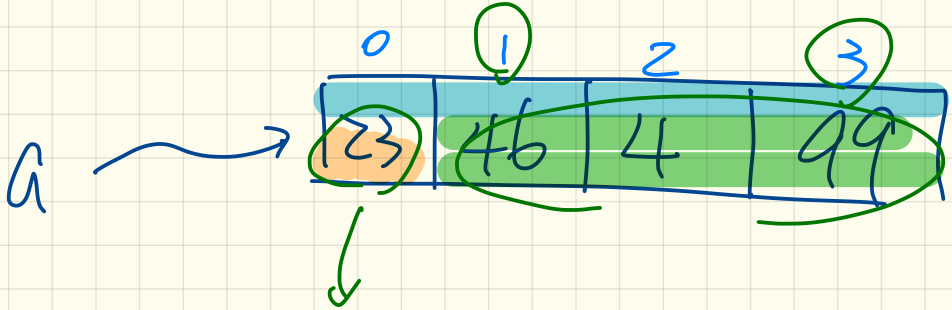
Say a3 = {A, B, C, D}, consider m(a3, 0, a3.length - 1)



$$\left(\forall \underline{x} \mid \underbrace{\boxed{\text{false}}}_{\text{range}} \bullet \underline{P(x)}\right) \equiv \text{True}$$

Are all numbers in an <u>empty array</u> a positive?

↳ Is it possible to find a <u>witness</u> in a s.t. it is <u>not</u> positive?

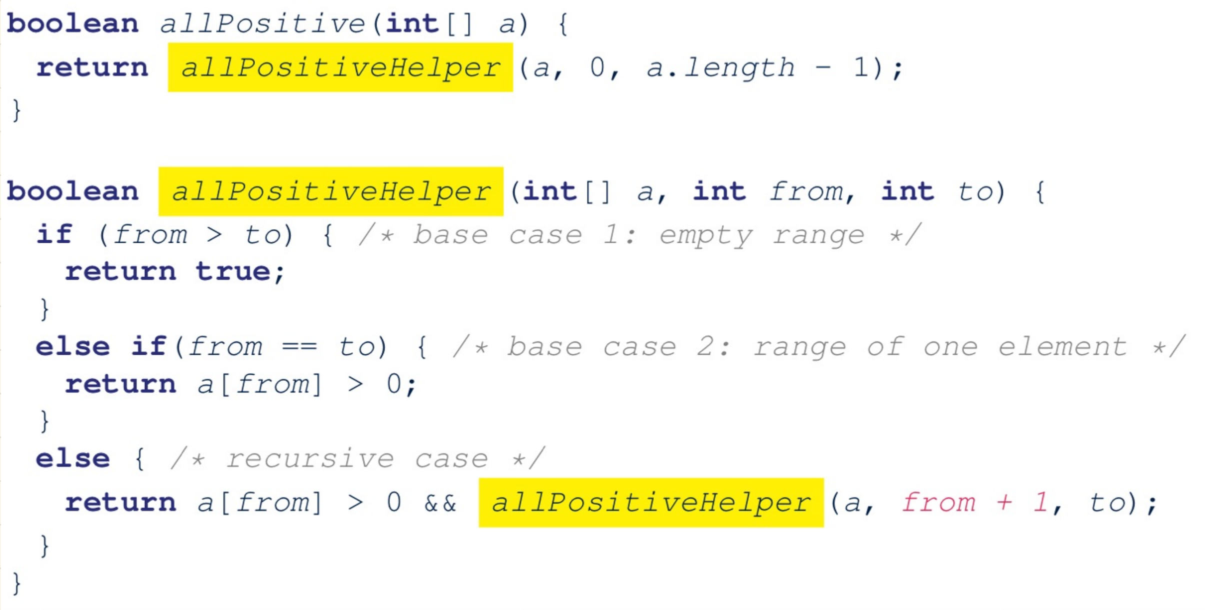


allP(a) true a[0] > 0

&&

true [allP( sub array from index 1 to index 3 )

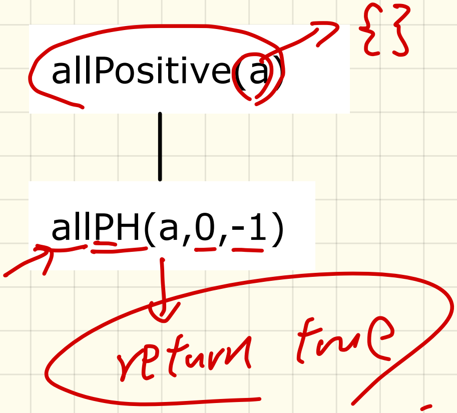
# Problem: Are All Numbers Positive?

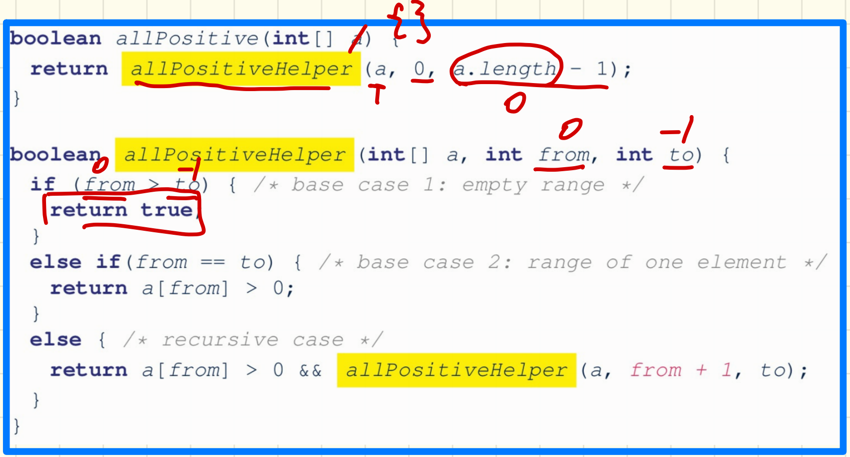


```java
boolean allPositive(int[] a) {
  return allPositiveHelper(a, 0, a.length - 1);
}

boolean allPositiveHelper(int[] a, int from, int to) {
  if (from > to) { /* base case 1: empty range */
    return true;
  }
  else if(from == to) { /* base case 2: range of one element */
    return a[from] > 0;
  }
  else { /* recursive case */
    return a[from] > 0 && allPositiveHelper(a, from + 1, to);
  }
}
```

# Tracing Recursion:

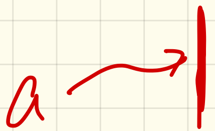
## allPositive





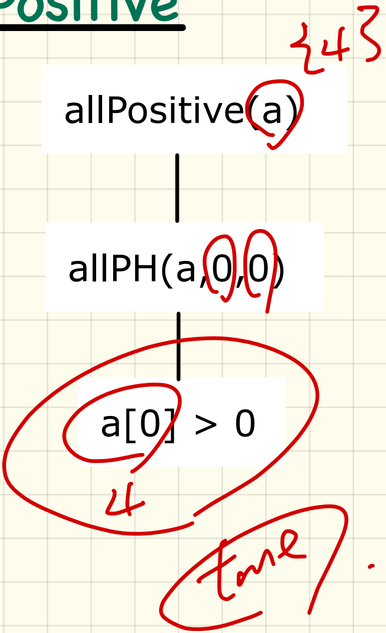
```
boolean allPositive(int[] a) {
  return allPositiveHelper (a, 0, a.length - 1);
}

boolean allPositiveHelper (int[] a, int from, int to) {
  if (from > to) { /* base case 1: empty range */
    return true;
  }
  else if(from == to) { /* base case 2: range of one element */
    return a[from] > 0;
  }
  else { /* recursive case */
    return a[from] > 0 && allPositiveHelper (a, from + 1, to);
  }
}
```

Say a = {}



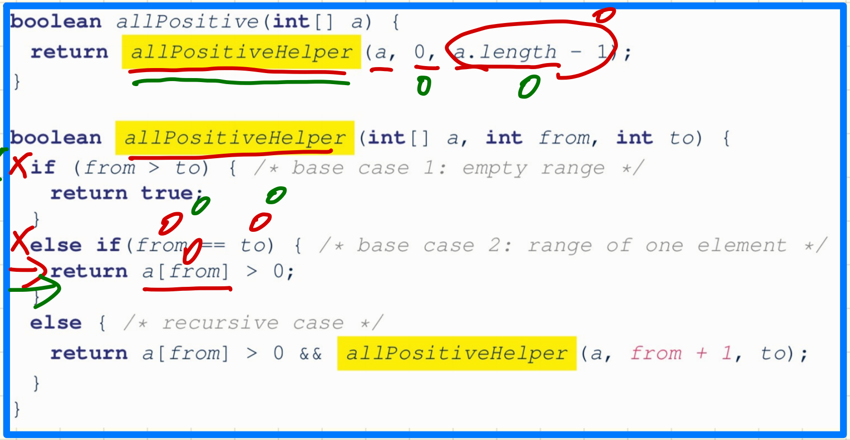
# Tracing Recursion:

## allPositive


allPositive(a)

allPH(a,0,0)

a[0] > 0


```
boolean allPositive(int[] a) {
  return allPositiveHelper (a, 0, a.length - 1);
}

boolean allPositiveHelper (int[] a, int from, int to) {
  if (from > to) { /* base case 1: empty range */
    return true;
  }
  else if(from == to) { /* base case 2: range of one element */
    return a[from] > 0;
  }
  else { /* recursive case */
    return a[from] > 0 && allPositiveHelper (a, from + 1, to);
  }
}
```

a[0] > 0

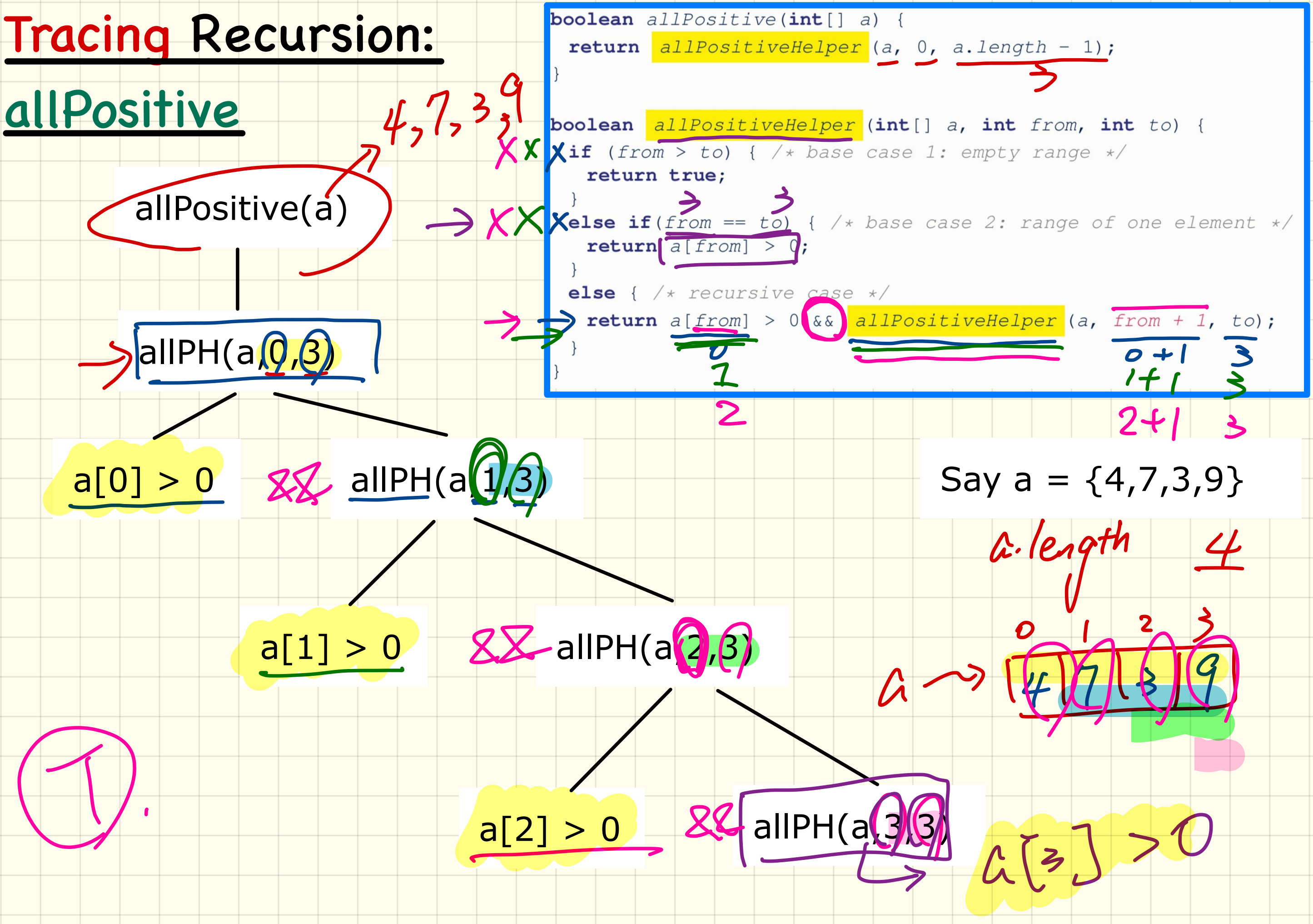
false

Say a = {4}

Say a = {-10}

# Tracing Recursion:

## allPositive

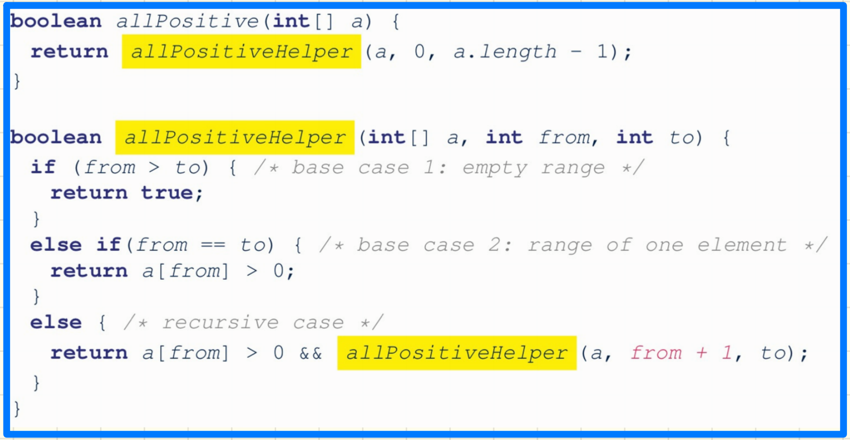


```
boolean allPositive(int[] a) {
  return allPositiveHelper (a, 0, a.length - 1);
}

boolean allPositiveHelper (int[] a, int from, int to) {
  if (from > to) { /* base case 1: empty range */
    return true;
  }
  else if(from == to) { /* base case 2: range of one element */
    return a[from] > 0;
  }
  else { /* recursive case */
    return a[from] > 0 && allPositiveHelper (a, from + 1, to);
  }
}
```

Say a = {4,7,3,9}

# Tracing Recursion:

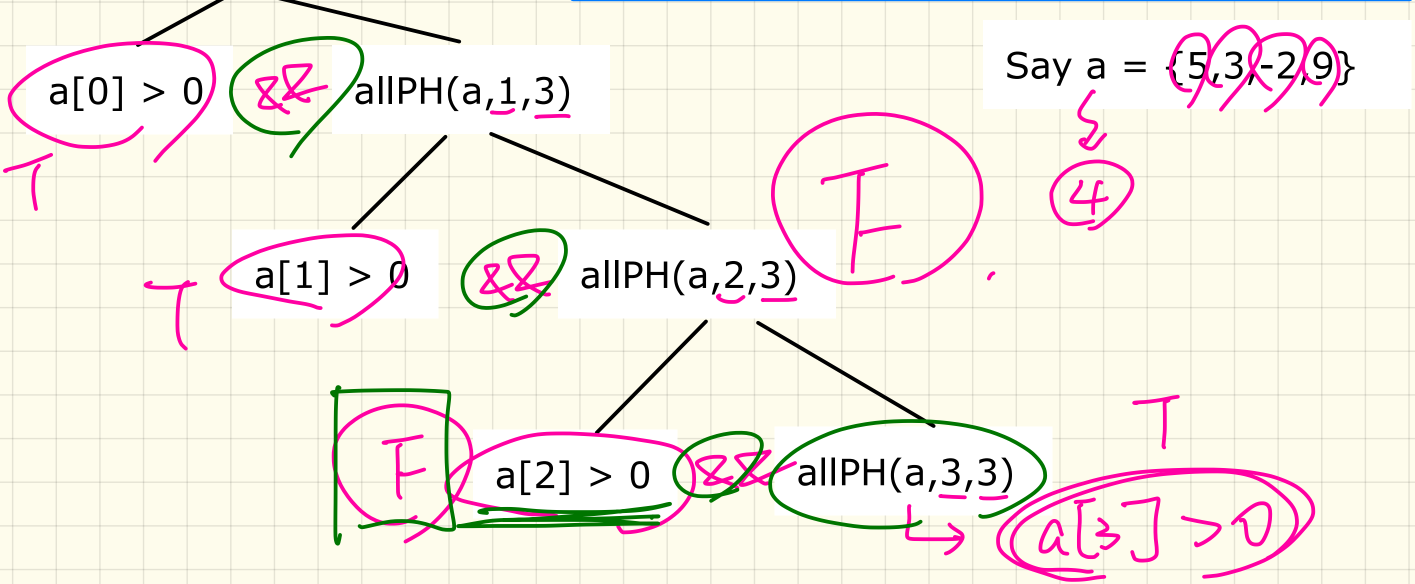
## allPositive



```
boolean allPositive(int[] a) {
  return allPositiveHelper(a, 0, a.length - 1);
}

boolean allPositiveHelper(int[] a, int from, int to) {
  if (from > to) { /* base case 1: empty range */
    return true;
  }
  else if(from == to) { /* base case 2: range of one element */
    return a[from] > 0;
  }
  else { /* recursive case */
    return a[from] > 0 && allPositiveHelper(a, from + 1, to);
  }
}
```

allPositive(a)

allPH(a,0,3)



An array is sorted in:

(a) ascending order ✓ $\{1, 3, 4, 5\}$

$\{1, \boxed{3, 3}, 4, 5\}$

$3 < 3$ ✗

(b) non-ascending order

$!(a[0] < a[1])$

$=$

$a[0] \geq a[1]$

(c) ✓ descending order

(d) non-descending order

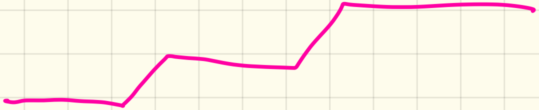
$!(a[0] > a[1])$

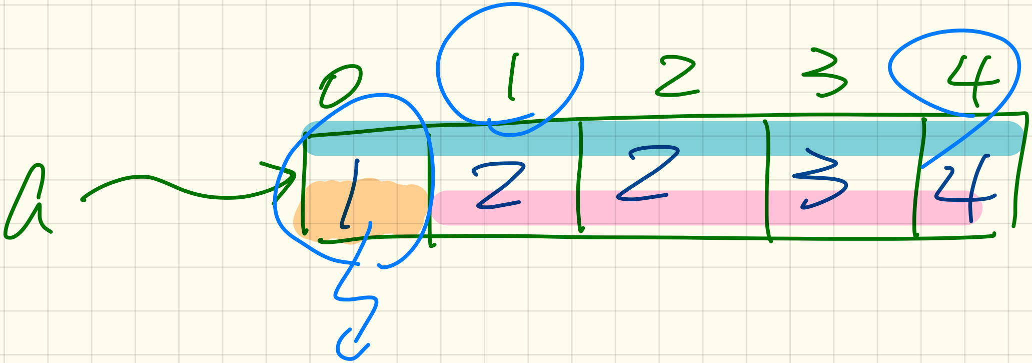
$= a[0] \leq a[1]$

non-descending

$$!(a[0] > a[1])$$

$$\equiv \quad a[0] \leq a[1]$$

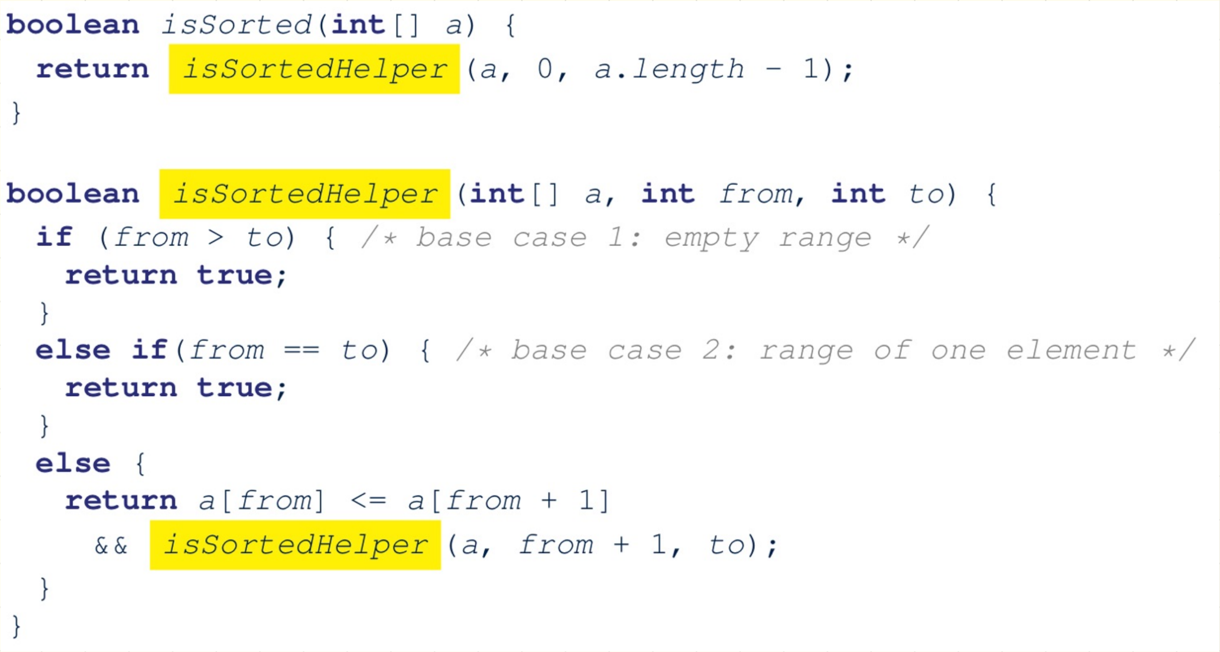




isSorted(a)

= a[0] ≤ a[1] && isSorted(
sub array from
indices 1 to 4)

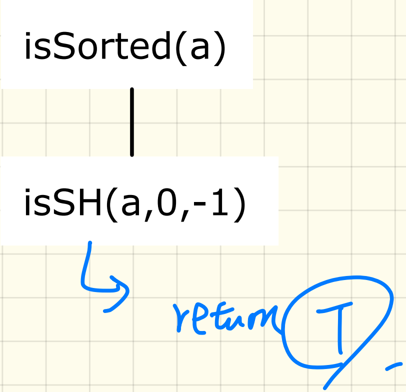
# Problem: Are Numbers Sorted?



```
boolean isSorted(int[] a) {
  return isSortedHelper (a, 0, a.length - 1);
}

boolean isSortedHelper (int[] a, int from, int to) {
  if (from > to) { /* base case 1: empty range */
    return true;
  }
  else if(from == to) { /* base case 2: range of one element */
    return true;
  }
  else {
    return a[from] <= a[from + 1]
      && isSortedHelper (a, from + 1, to);
  }
}
```

# Tracing Recursion:

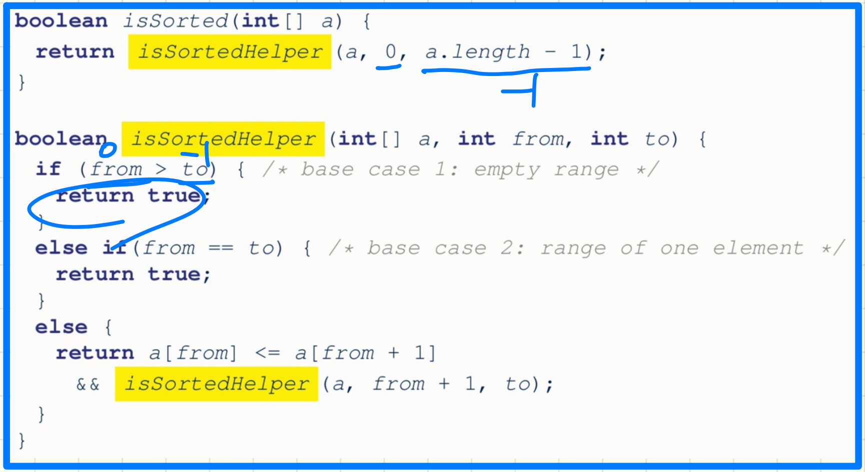
## isSorted



isSorted(a)

isSH(a,0,-1)

return T

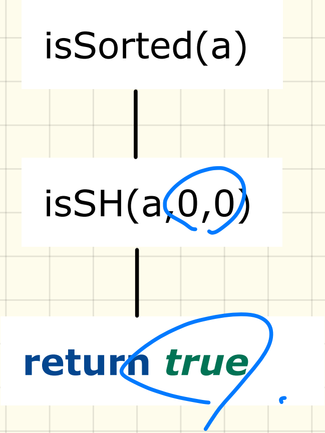


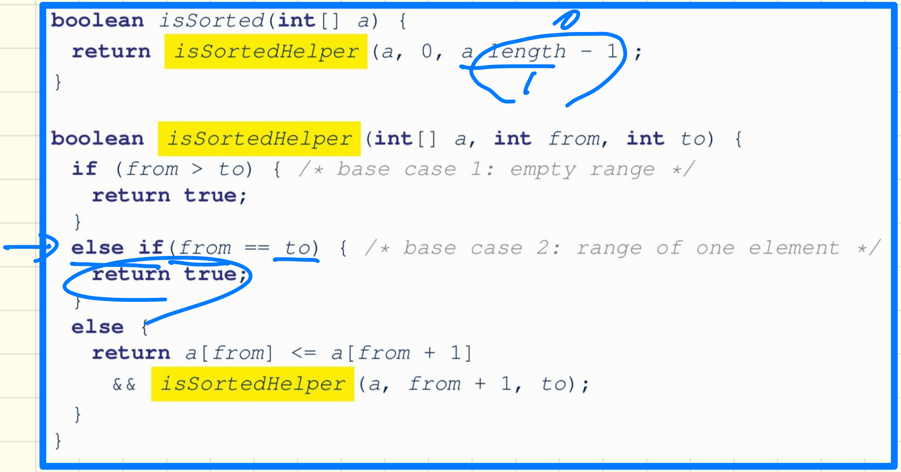
```
boolean isSorted(int[] a) {
  return isSortedHelper(a, 0, a.length - 1);
}

boolean isSortedHelper(int[] a, int from, int to) {
  if (from > to) { /* base case 1: empty range */
    return true;
  }
  else if(from == to) { /* base case 2: range of one element */
    return true;
  }
  else {
    return a[from] <= a[from + 1]
      && isSortedHelper(a, from + 1, to);
  }
}
```

Say a = {}

# Tracing Recursion:

## isSorted





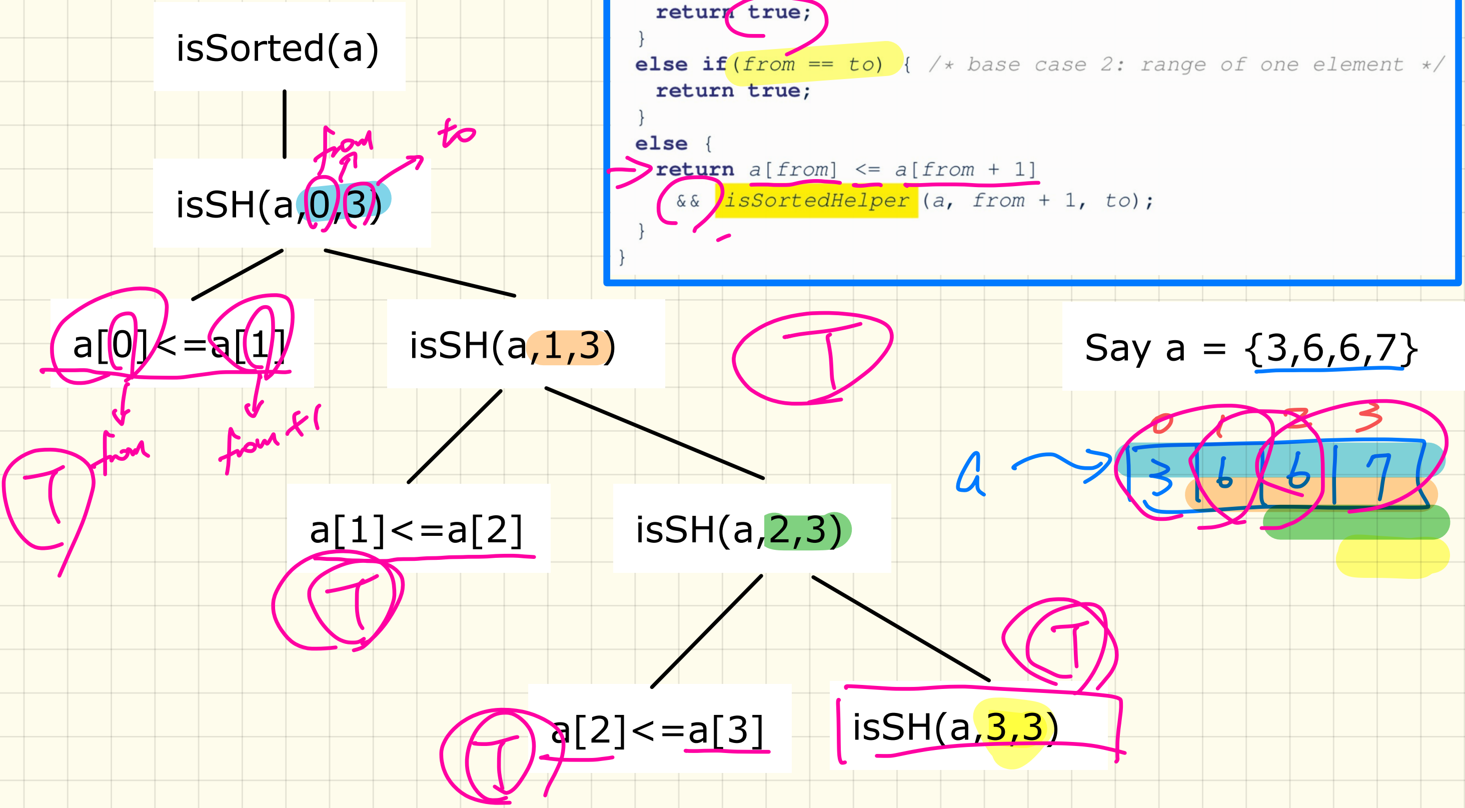
```
boolean isSorted(int[] a) {
  return isSortedHelper(a, 0, a.length - 1);
}

boolean isSortedHelper(int[] a, int from, int to) {
  if (from > to) { /* base case 1: empty range */
    return true;
  }
  else if(from == to) { /* base case 2: range of one element */
    return true;
  }
  else {
    return a[from] <= a[from + 1]
      && isSortedHelper(a, from + 1, to);
  }
}
```

Say a = {4}

# Tracing Recursion:

## isSorted

```
boolean isSorted(int[] a) {
  return isSortedHelper (a, 0, a.length - 1);
}

boolean isSortedHelper (int[] a, int from, int to) {
  if (from > to) { /* base case 1: empty range */
    return true;
  }
  else if (from == to) { /* base case 2: range of one element */
    return true;
  }
  else {
    return a[from] <= a[from + 1]
      && isSortedHelper (a, from + 1, to);
  }
}
```

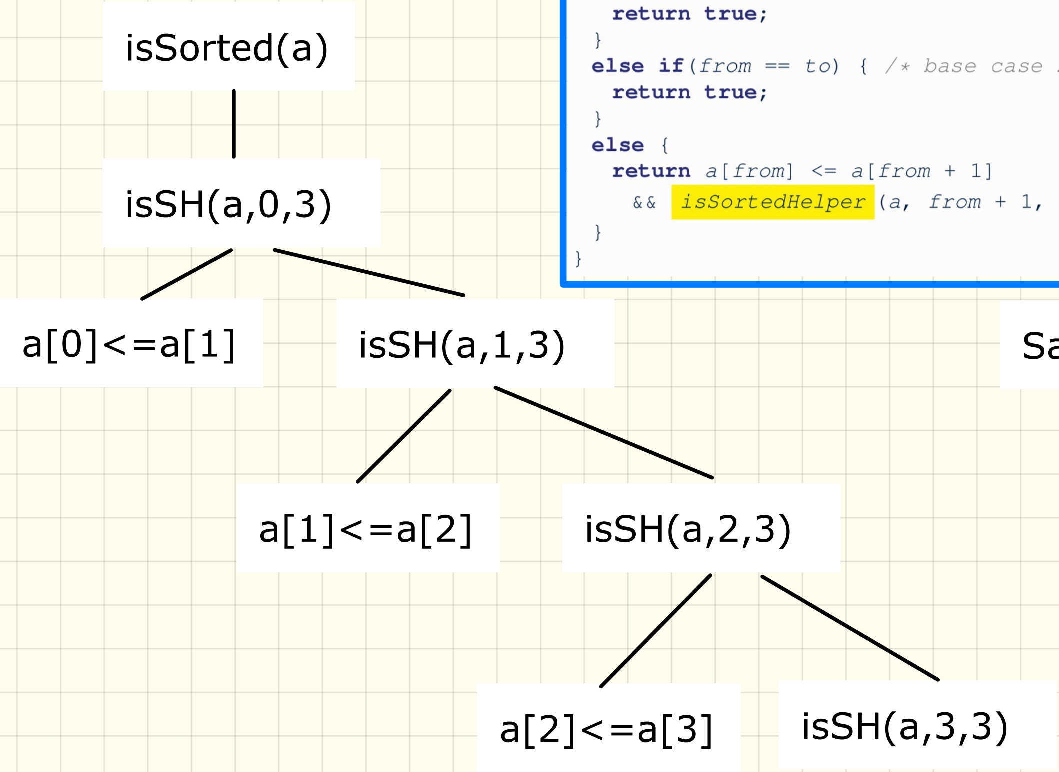


Say a = {3,6,6,7}

# Tracing Recursion:

## isSorted

```
boolean isSorted(int[] a) {
  return isSortedHelper (a, 0, a.length - 1);
}

boolean isSortedHelper (int[] a, int from, int to) {
  if (from > to) { /* base case 1: empty range */
    return true;
  }
  else if(from == to) { /* base case 2: range of one element */
    return true;
  }
  else {
    return a[from] <= a[from + 1]
      && isSortedHelper (a, from + 1, to);
  }
}
```



isSorted(a)

isSH(a,0,3)

a[0]<=a[1]

isSH(a,1,3)

a[1]<=a[2]

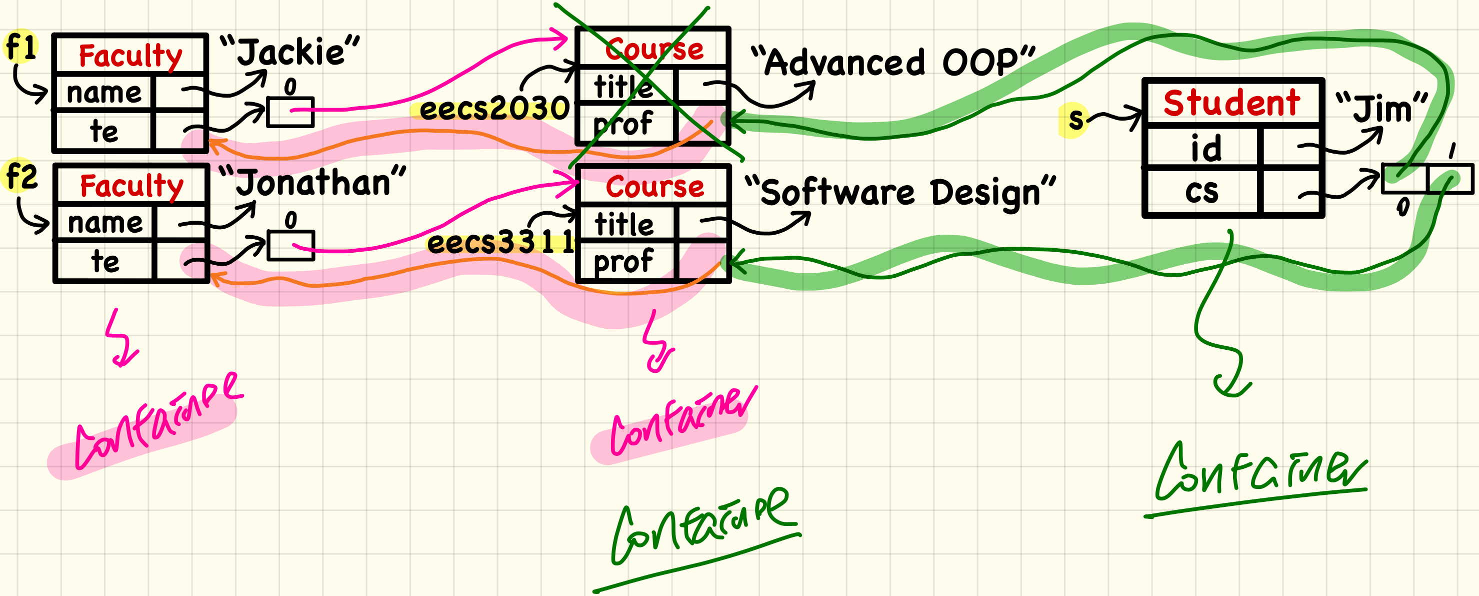
isSH(a,2,3)

a[2]<=a[3]

isSH(a,3,3)

Say a = {3,6,5,7}

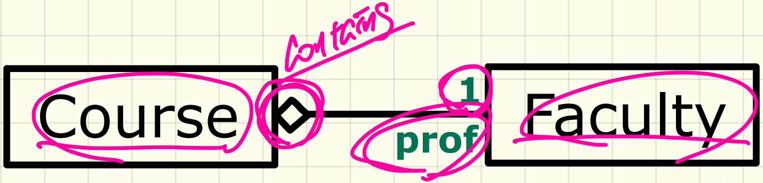
# Container vs. Containee



What if a course is deleted?

# Aggregation: Design

## Design 1: Single Containee



Each Course object contains 1 Faculty object as its prof.

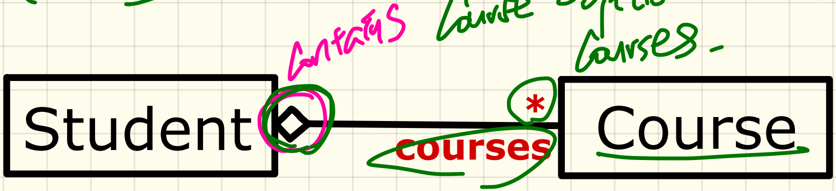
# Java Implementation

```
class Course {
    Faculty prof;
    ...
}
```

```
class Faculty {
    ...
}
```

## Design 2: Multiple Containees

Each Student object contains a collection of Course objects as their courses.



```
class Student {
    Course[] courses;
    ...
}
```

```
class Course {
    ...
}
```

# LECTURE 15

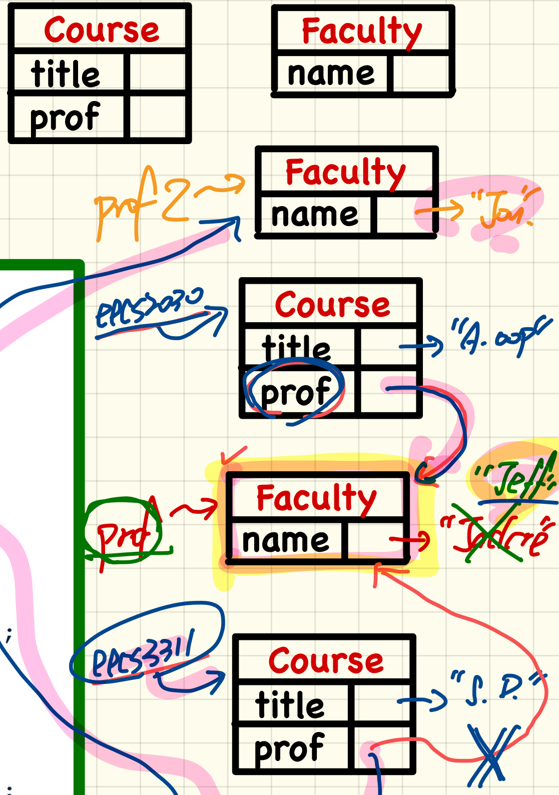
WEDNESDAY OCTOBER 30

# Aggregation (1)

```
class Course {
  String title;
  Faculty prof;
  Course(String title) {
    this.title = title;
  }
  void setProf(Faculty ~~prof~~ prof) {
    this.prof = prof;
  }
  Faculty getProf() {
    return this.prof;
  }
}
```

```
class Faculty {
  String name;
  Faculty(String name) {
    this.name = name;
  }
  void setName(String name) {
    this.name = name;
  }
  String getName() {
    return this.name;
  }
}
```

```
@Test
public void testAggregation1() {
  Course eecs2030 = new Course("Advanced OOP");
  Course eecs3311 = new Course("Software Design");
  Faculty prof = new Faculty("Jackie");
  eecs2030.setProf(prof);
  eecs3311.setProf(prof);
  assertTrue(eecs2030.getProf() == eecs3311.getProf());
  /* aliasing */
  prof.setName("Jeff");
  assertTrue(eecs2030.getProf() == eecs3311.getProf());
  assertTrue(eecs2030.getProf().getName().equals("Jeff"));

  Faculty prof2 = new Faculty("Jonathan");
  eecs3311.setProf(prof2);
  assertTrue(eecs2030.getProf() != eecs3311.getProf());
  assertTrue(eecs2030.getProf().getName().equals("Jeff"));
  assertTrue(eecs3311.getProf().getName().equals("Jonathan"));
}
```



# Aggregation (2)

elements in list are of type Course

```
class Student {
  String id; ArrayList<Course> cs; /* courses */
  Student(String id) { this.id = id; cs = new ArrayList<>(); }
  void addCourse(Course c) { cs.add(c); }
  ArrayList<Course> getCS() { return cs; }
}
```

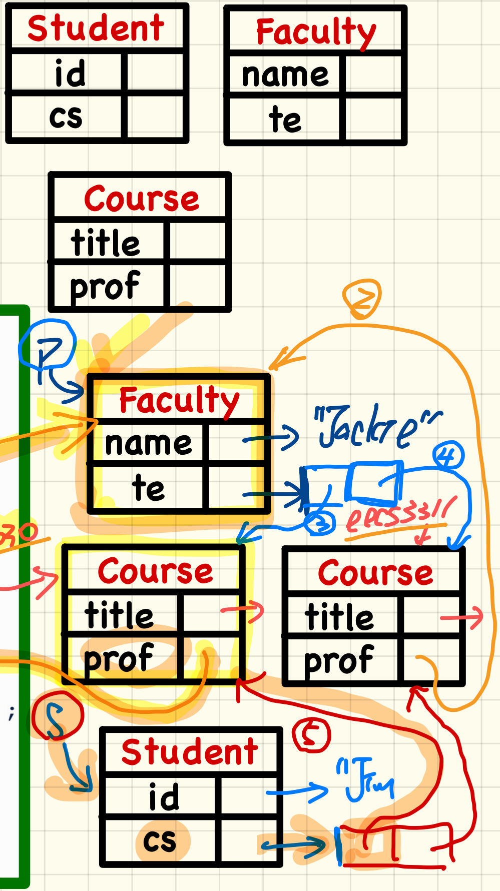
```
class Course { String title; Faculty prof; }
```

```
class Faculty {
  String name; ArrayList<Course> te; /* teaching */
  Faculty(String name) { this.name = name; te = new ArrayList<>(); }
  void addTeaching(Course c) { te.add(c); }
  ArrayList<Course> getTE() { return te; }
}
```

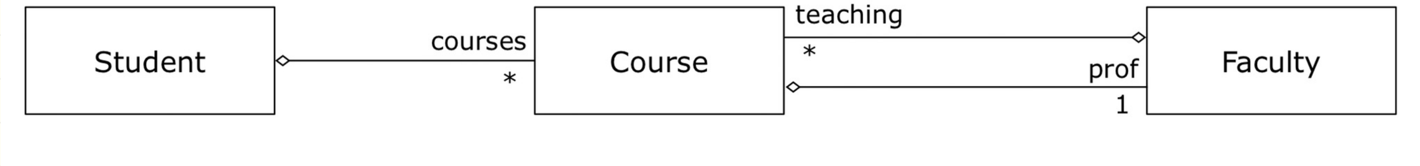
```
@Test
public void testAggregation2() {
  Faculty p = new Faculty("Jackie");
  Student s = new Student("Jim");
  Course eecs2030 = new Course("Advanced OOP");
  Course eecs3311 = new Course("Software Design");
  eecs2030.setProf(p); 1
  eecs3311.setProf(p); 2
  p.addTeaching(eecs2030); 3
  p.addTeaching(eecs3311); 4
  s.addCourse(eecs2030); 5
  s.addCourse(eecs3311); 6

  assertTrue(eecs2030.getProf() == s.getCS().get(0).getProf());
  assertTrue(s.getCS().get(0).getProf()
             == s.getCS().get(1).getProf());
  assertTrue(eecs3311 == s.getCS().get(1));
  assertTrue(s.getCS().get(1) == p.getTE().get(1));
}
```

A.L<Course>

Course



# Dot Notation for Navigating Aggregations (1)



```
class Student {
  String id;
  ArrayList<Course> cs;
}
```

```
class Course {
  String title;
  Faculty prof;
}
```

```
class Faculty {
  String name;
  ArrayList<Course> te;
}
```

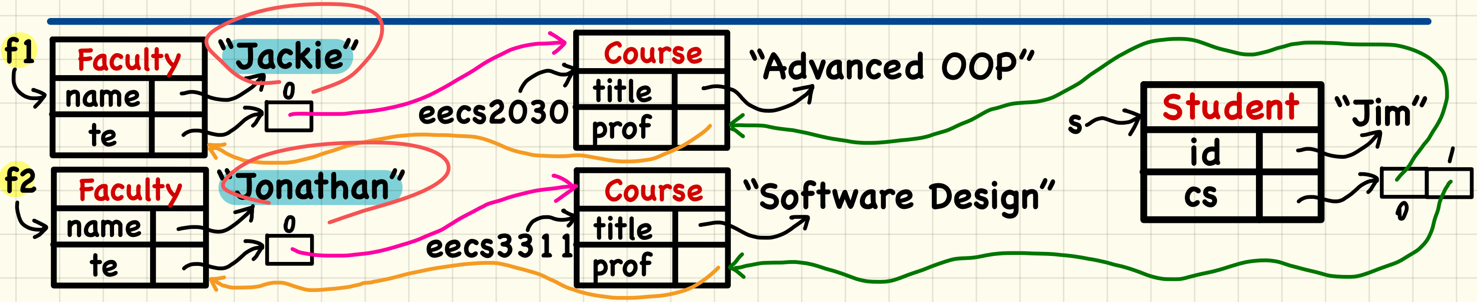
## Examples

f1.getName()

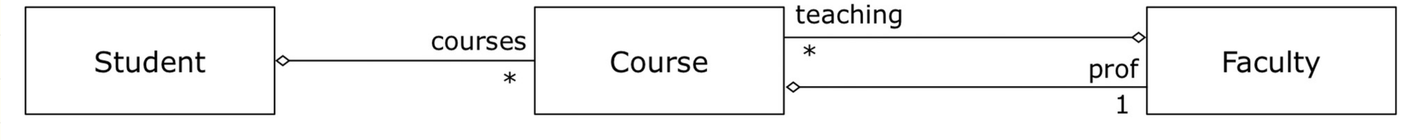
f2.getName()

```
/* Name of this faculty
 */
String getName()
```

return name;



# Dot Notation for Navigating Aggregations (2)



```
class Student {
  String id;
  ArrayList<Course> cs;
}
```

```
class Course {
  String title;
  Faculty prof;
}
```

```
class Faculty {
  String name;
  ArrayList<Course> te;
}
```

return ~~this~~.faculty. . .

.prof.name

```
/* Instructor's name
 * for this course
 */
String getName()
```

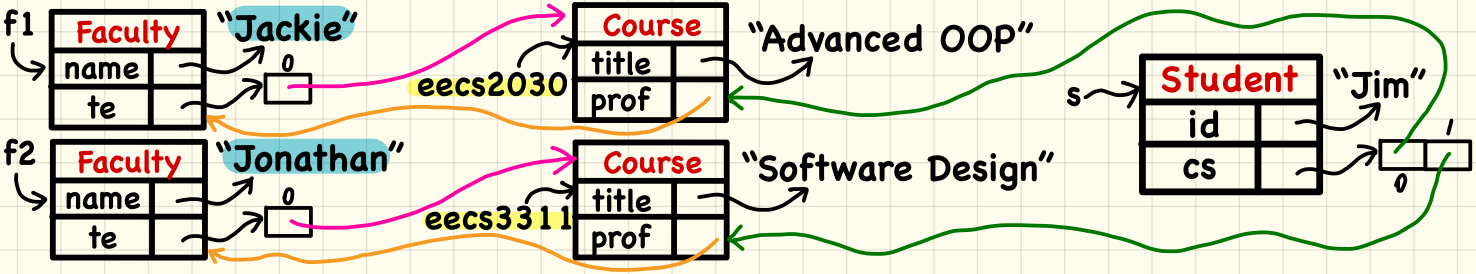
~~eecs2030~~

return ~~this~~.prof.name

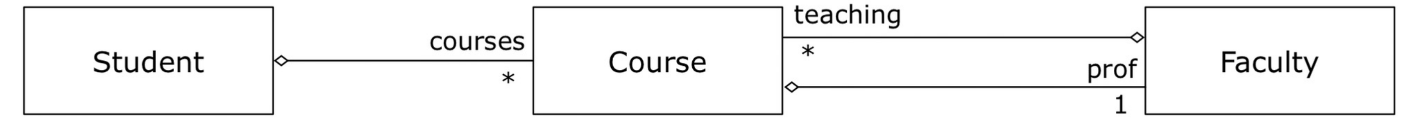
## Examples

eecs2030.getName()

eecs3311.getName()



# Dot Notation for Navigating Aggregations (3)



```
class Student {
  String id;
  ArrayList<Course> cs;
}
```

```
class Course {
  String title;
  Faculty prof;
}
```

```
class Faculty {
  String name;
  ArrayList<Course> te;
}
```

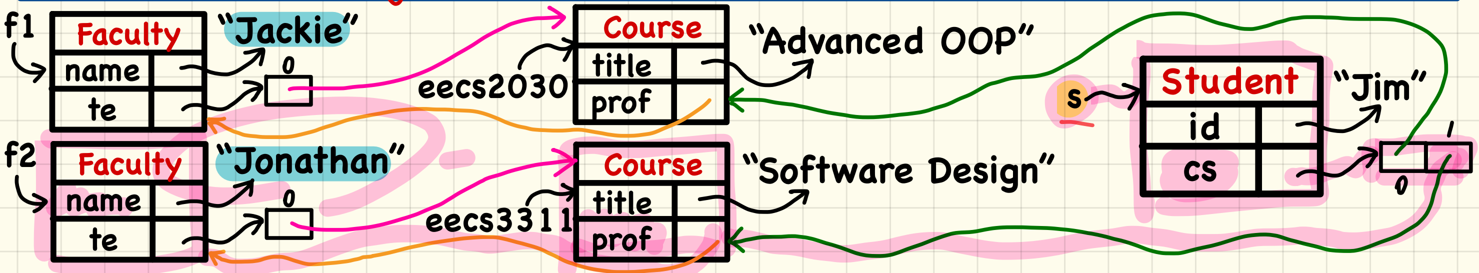
```
/* Instructor's name for
 * course stored at index i
 */
String getName(int i)
```

return ~~this~~ s. cs.get(~~i~~ i).prf.name

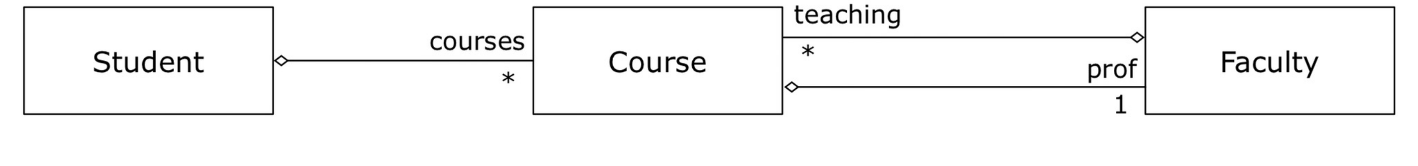
## Examples

s.getName(0)

s.getName(1)



# Dot Notation for Navigating Aggregations: Exercise



Student

courses

*

Course

teaching

*

prof

1

Faculty

```
class Student {
  String id;
  ArrayList<Course> cs;
}
```

```
class Course {
  String title;
  Faculty prof;
}
```

```
class Faculty {
  String name;
  ArrayList<Course> te;
}
```

```
/* Title of the ith teaching course
 * of the instructor for this course
 */
String getTitle(int i)
```

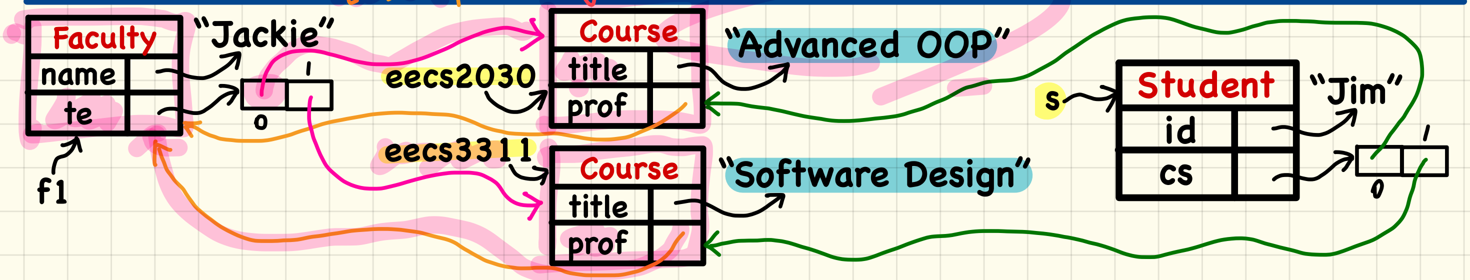
return ~~this~~.prof.te.get(~~i~~ 0).title

eecs3311

## Examples

eecs2030.getTitle(1)

eecs3311.getTitle(0)



# Composition: No Sharing

```
class Directory {
  String name;
  File[] files;
  int nof; /* num of files */
  Directory(String name) {
    this.name = name;
    files = new File[100];
  }
void addFile(String fileName) {
    files[nof] = new File(fileName);
    nof ++;
  }
}
```

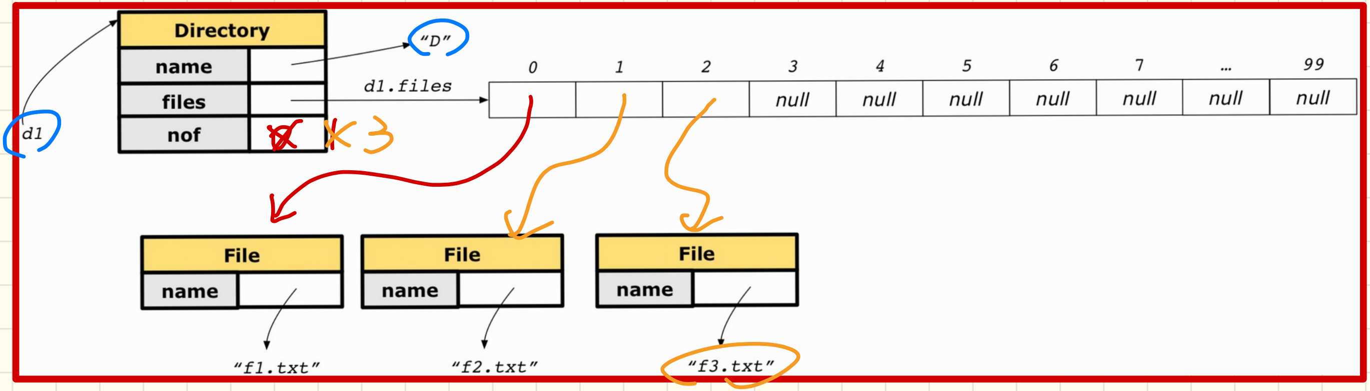
"f1.txt"

"f1.txt"

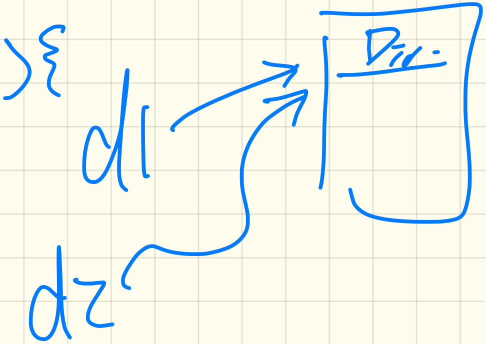
```
class File {
  String name;
  File(String name) {
    this.name = name;
  }
}
```

"f1.txt"

```
@Test
public void testComposition() {
  Directory d1 = new Directory("D");
  d1.addFile("f1.txt");
  d1.addFile("f2.txt");
  d1.addFile("f3.txt");
  assertTrue(
    d1.files[0].name.equals("f1.txt"))
}
```



```
class Directory{
    Directory(Directory other){
        this = other;
    }
}

    Directory d1 = new Directory("D1");
    Directory d2 = new Directory(d1);
```



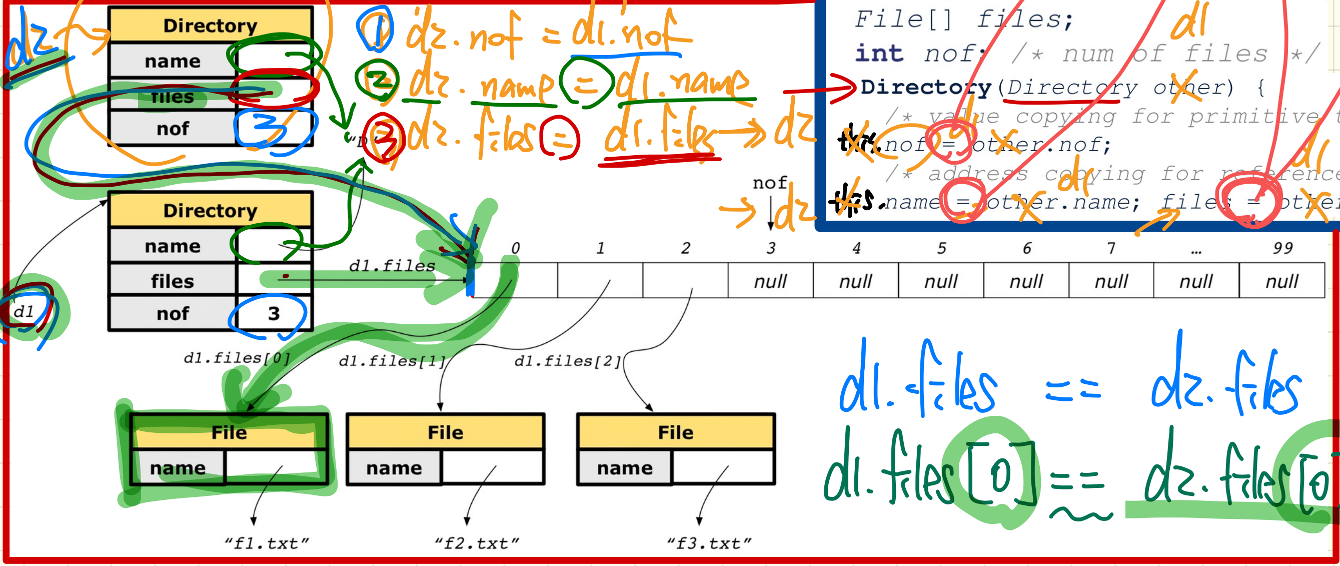
# Composition: Copy Constructor (Shallow Copy)

```
@Test
void testShallowCopyConstructor() {
  Directory d1 = new Directory("D");
  d1.addFile("f1.txt"); d1.addFile("f2.txt"); d1.addFile("f3.txt"
  Directory d2 = new Directory(d1);
  assertTrue(d1.files == d2.files); /* violation of composition *
  d2.files[0].changeName("f11.txt");
  assertFalse(d1.files[0].name.equals("f1.txt")); }
```

shallow copy

```
class Directory {
  String name;
  File[] files;
  int nof; /* num of files */
  Directory(Directory other) {
    /* value copying for primitive type */
    this.nof = other.nof;
    /* address copying for reference type */
    this.name = other.name; files = other.files; }
```

aliasing



① d2.nof = d1.nof

② d2.name = d1.name

③ d2.files = d1.files → d2

d1.files == d2.files (T)

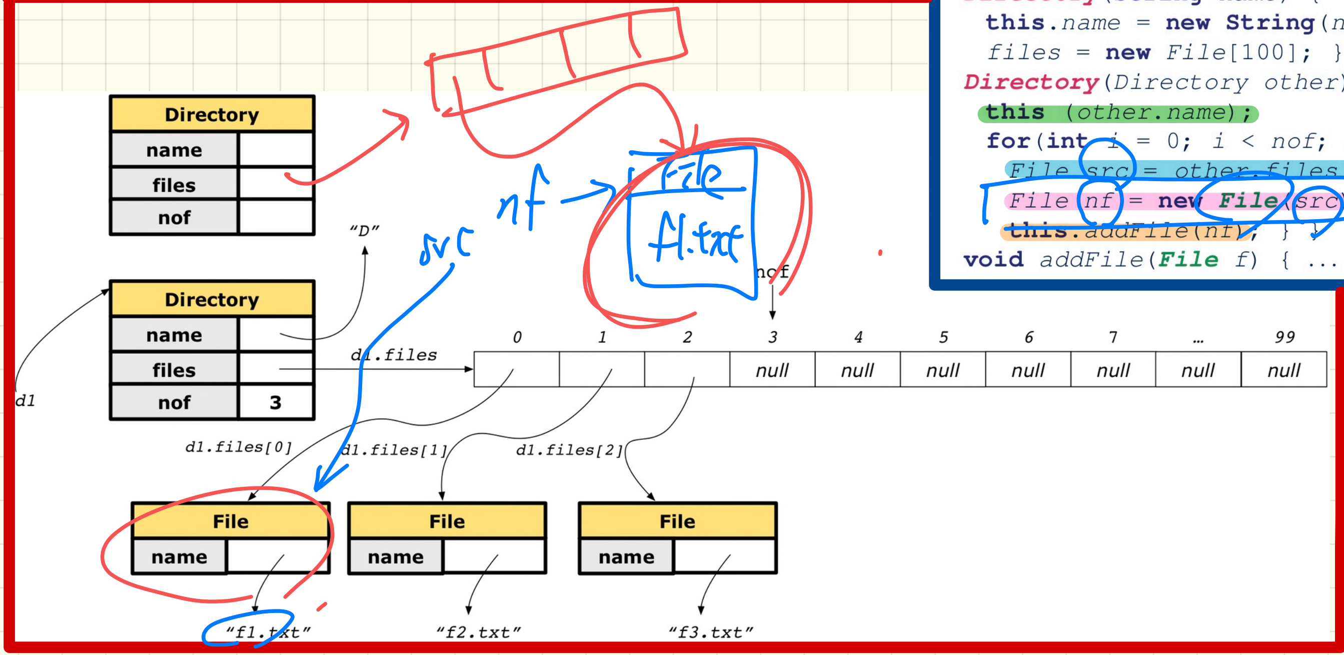
d1.files[0] == d2.files[0]

# Composition: Copy Constructor (Deep Copy)

```
@Test
void testDeepCopyConstructor() {
  Directory d1 = new Directory("D");
  d1.addFile("f1.txt"); d1.addFile("f2.txt"); d1.addFile("f3.txt"
  Directory d2 = new Directory(d1);
  assertTrue(d1.files != d2.files); /* composition preserved */
  d2.files[0].changeName("f11.txt");
  assertTrue(d1.files[0].name.equals("f1.txt")); }
```

```
class File {
  File(File other) {
    this.name =
      new String(other.name);
  }
}
```

```
class Directory {
  Directory(String name) {
    this.name = new String(name);
    files = new File[100]; }
  Directory(Directory other) {
   this (other.name);
   for(int i = 0; i < nof; i ++) {
    File src = other.files[i];
    File nf = new File(src);
    this.addFile(nf); } }
  void addFile(File f) { ... } }
```



# Copy Constructor (Composition?)

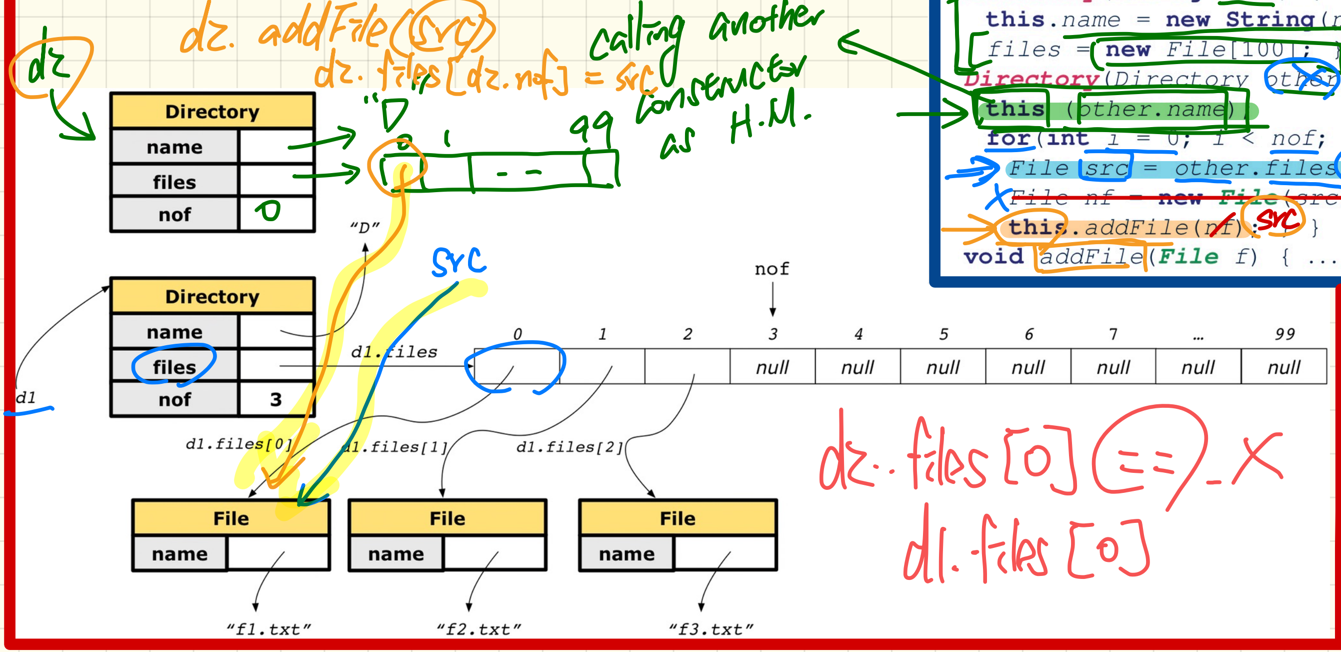
File src = d1.files[0];

```
@Test
void testDeepCopyConstructor() {
  Directory d1 = new Directory("D");
  d1.addFile("f1.txt"); d1.addFile("f2.txt"); d1.addFile("f3.txt"
  Directory d2 = new Directory(d1);
  assertTrue(d1.files != d2.files); /* composition preserved */
  d2.files[0].changeName("f11.txt");
  assertTrue(d1.files[0].name.equals("f1.txt")); }
```

```
class File {
  File(File other) {
    this.name =
      new String(other.name);
  }
}
```

```
class Directory {
  Directory(String name) {
    this.name = new String(name);
    files = new File[100]; }
  Directory(Directory other) { d1
    this (other.name);
    for(int i = 0; i < nof; i ++) {
      File src = other.files[i];
      ~~File nf = new File(src);~~
      this.addFile(~~nf~~ src); } }
  void addFile(File f) { ... } }
```

calling another constructor as H.M.



d2.files[0] == X
d1.files[0]

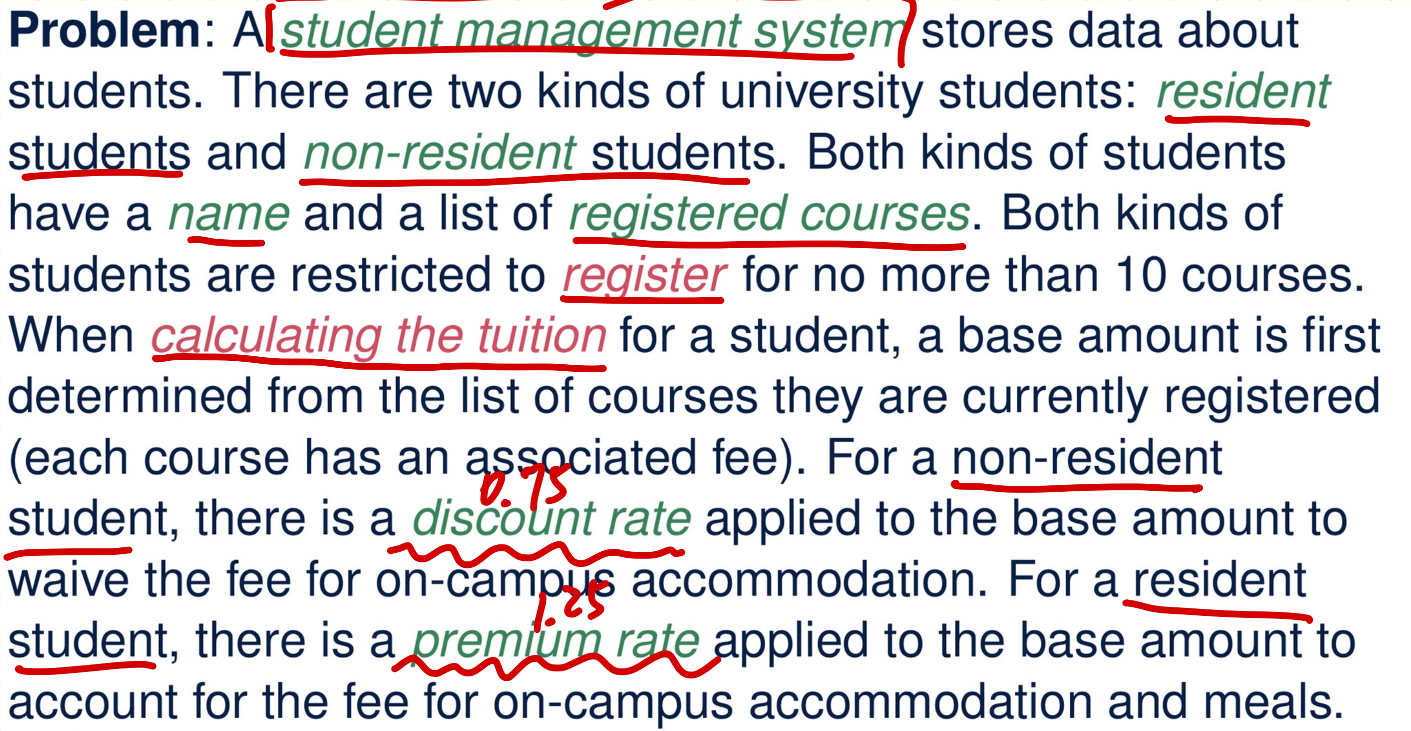
# LECTURE 16

MONDAY NOVEMBER 4

# Inheritance: Motivating Problem

Nouns -> classes, attributes, accessors

Verbs -> mutators



**Problem**: A *student management system* stores data about students. There are two kinds of university students: *resident* students and *non-resident* students. Both kinds of students have a *name* and a list of *registered courses*. Both kinds of students are restricted to *register* for no more than 10 courses. When *calculating the tuition* for a student, a base amount is first determined from the list of courses they are currently registered (each course has an associated fee). For a non-resident student, there is a 0.75 *discount rate* applied to the base amount to waive the fee for on-campus accommodation. For a resident student, there is a 1.25 *premium rate* applied to the base amount to account for the fee for on-campus accommodation and meals.

# First Design Attempt

```
class Student {
   Course[] courses;
   int noc;
   int kind;
   double premiumRate;
   double discountRate;
   Student (int kind){
     this.kind = kind;
   }
   ...
}
```

```
double getTuition(){
   double tuition = 0;
   for(int i = 0; i < noc; i++){
     tuition += courses[i].fee;
   }
   if (this.kind == 1) {
     return tuition * premiumRate;
   }
   else if (this.kind == 2) {
     return tuition * discountRate;
   }
}
```

```
double register(Course c){
   int MAX;
   if (this.kind == 1) { MAX = 6; }
   else if (this.kind == 2) { MAX = 4; }
   if (noc == MAX) { /* Error */ }
   else {
     courses[noc] = c;
     noc++;
   }
}
```

repetition

# First Design Attempt

```
class Student {
  Course[] courses;
  int noc;
  int kind;
  double premiumRate;
  double discountRate;
  Student (int kind){
    this.kind = kind;
  }
  ...
}
```

related to different purposes:

RS

NRS

```
double getTuition(){
  double tuition = 0;
  for(int i = 0; i < noc; i++){
    tuition += courses[i].fee;
  }
  if (this.kind == 1) {
    return tuition * premiumRate;
  }
  else if (this.kind == 2) {
    return tuition * discountRate;
  }
}
```

```
double register(Course c){
  int MAX;
  if (this.kind == 1) { MAX = 6; }
  else if (this.kind == 2) { MAX = 4; }
  if (noc == MAX) { /* Error */ }
  else {
    courses[noc] = c;
    noc++;
  }
}
```

## Good design?

Judge by **Cohesion**

all methods or attr. in a single class must be related to a single purpose.

# First Design Attempt

```
class Student {
   Course[] courses;
   int noc;        1 : RS
   int kind;       2 : NRS
   double premiumRate;
   double discountRate;
                  1 2 3
   Student (int kind){
      this.kind = kind;
   }
   ...
}
```

3: IS

```
double getTuition(){
   double tuition = 0;
   for(int i = 0; i < noc; i++){
      tuition += courses[i].fee;
   }
   if (this.kind == 1) {
      return tuition * premiumRate;
   }
   else if (this.kind == 2) {
      return tuition * discountRate;
   }
   else if ( this.kind = 3) { . . . }
}
```

```
double register(Course c){
   int MAX;
   if (this.kind == 1) { MAX = 6; }
   else if (this.kind == 2) { MAX = 4; }
   else if (this.kind = 3) { . . . }
   if (noc == MAX) { /* Error */ }
   else {
      courses[noc] = c;
      noc++;
   }
}
```

## Good design?

Judge by Single Choice Principle

- Repeated if-conditions
- A new kind is introduced?
- An existing kind is obeselete?

change should be done in a single place

V1

Student

The kind

pr

dr

issue of cohesion

V2

ResidentStudent

pr

NonResidentStuden

dr

cohesion resolved.

# Testing Student Classes (without inheritance)

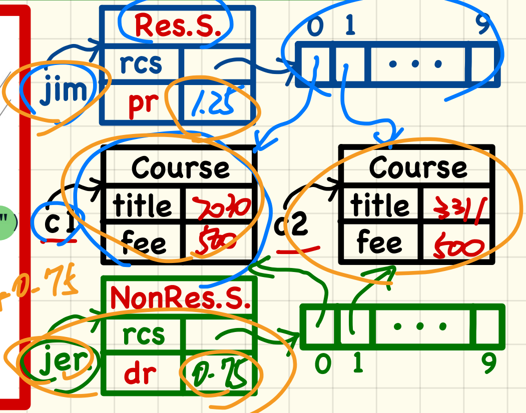
```
class ResidentStudent {
  String name;
  Course[] registeredCourses;
  int numberOfCourses;
  double premiumRate; /* there's a mutator me
  ResidentStudent (String name) {
   this.name = name;
     registeredCourses = new Course[10];
  }
  void register(Course c) {
   registeredCourses[numberOfCourses] = c;
   numberOfCourses ++;
  }
  double getTuition() {
   double tuition = 0;
   for(int i = 0; i < numberOfCourses; i ++) {
    tuition += registeredCourses[i].fee;
   }
   return tuition * premiumRate;
  }
```

```
class NonResidentStudent {
  String name;
  Course[] registeredCourses;
  int numberOfCourses;
  double discountRate; /* there's a mutator me
  NonResidentStudent (String name) {
   this.name = name;
     registeredCourses = new Course[10];
  }
  void register(Course c) {
   registeredCourses[numberOfCourses] = c;
   numberOfCourses ++;
  }
  double getTuition() {
   double tuition = 0;
   for(int i = 0; i < numberOfCourses; i ++) {
    tuition += registeredCourses[i].fee;
   }
   return tuition * discountRate;
  }
```

```
class StudentTester {
  static void main(String[] args) {
   Course c1 = new Course("EECS2030", 500.00); /* title and fee */
   Course c2 = new Course("EECS3311", 500.00); /* title and fee */
   ResidentStudent jim = new ResidentStudent("J. Davis");
   jim.setPremiumRate(1.25);
   jim.register(c1); jim.register(c2);
   NonResidentStudent jeremy = new NonResidentStudent("J. Gibbons");
   jeremy.setDiscountRate(0.75);
   jeremy.register(c1); jeremy.register(c2);
   System.out.println("Jim pays " + jim.getTuition());
   System.out.println("Jeremy pays " + jeremy.getTuition());
  }
}
```

1000 * 1.25

1000 * 0.75



# Student Classes (without inheritance): Maintenance (1)

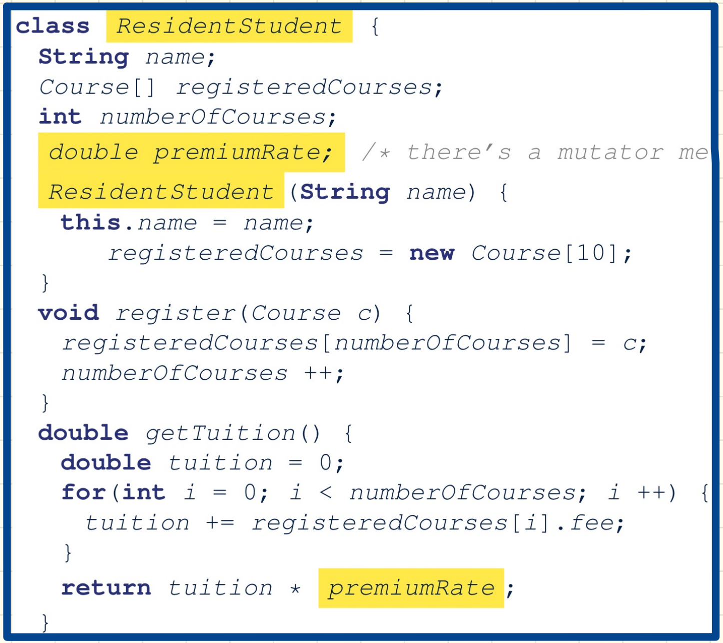
```
class ResidentStudent {
  String name;
  Course[] registeredCourses;
  int numberOfCourses;
  double premiumRate; /* there's a mutator me
  ResidentStudent (String name) {
    this.name = name;
      registeredCourses = new Course[10];
  }
  void register(Course c) {
    registeredCourses[numberOfCourses] = c;
    numberOfCourses ++;
  }
  double getTuition() {
    double tuition = 0;
    for(int i = 0; i < numberOfCourses; i ++) {
      tuition += registeredCourses[i].fee;
    }
    return tuition * premiumRate;
  }
```

```
class NonResidentStudent {
  String name;
  Course[] registeredCourses;
  int numberOfCourses;
  double discountRate; /* there's a mutator me
  NonResidentStudent (String name) {
    this.name = name;
      registeredCourses = new Course[10];
  }
  void register(Course c) {
    registeredCourses[numberOfCourses] = c;
    numberOfCourses ++;
  }
  double getTuition() {
    double tuition = 0;
    for(int i = 0; i < numberOfCourses; i ++) {
      tuition += registeredCourses[i].fee;
    }
    return tuition * discountRate;
  }
```

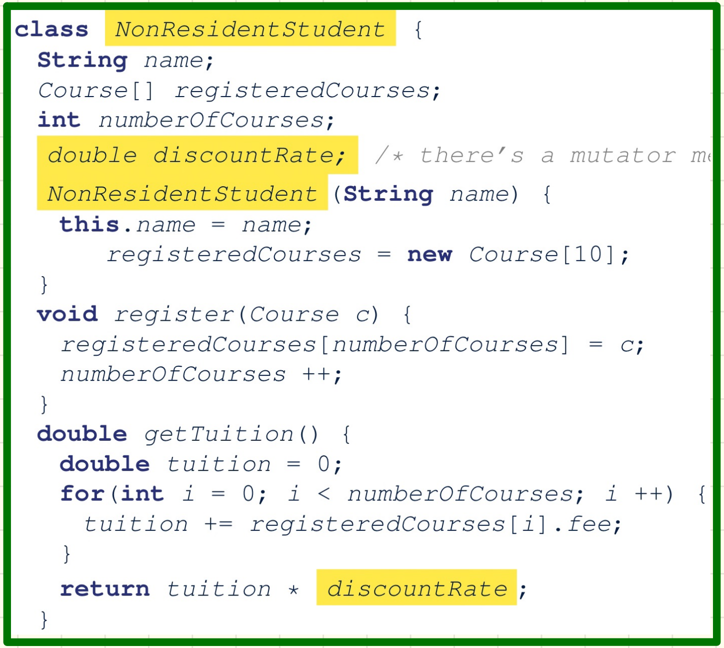
## Maintenance: e.g., a new registration constraint

```
if(numberOfCourses >= MAX_ALLOWANCE) {
  throw new IllegalArgumentException("Too Many Courses");
}
else { … }
```

# Student Classes (without inheritance): Maintenance (2)



```
class ResidentStudent {
  String name;
  Course[] registeredCourses;
  int numberOfCourses;
  double premiumRate; /* there's a mutator me
  ResidentStudent (String name) {
    this.name = name;
       registeredCourses = new Course[10];
  }
  void register(Course c) {
    registeredCourses[numberOfCourses] = c;
    numberOfCourses ++;
  }
  double getTuition() {
    double tuition = 0;
    for(int i = 0; i < numberOfCourses; i ++) {
      tuition += registeredCourses[i].fee;
    }
    return tuition * premiumRate;
  }
```

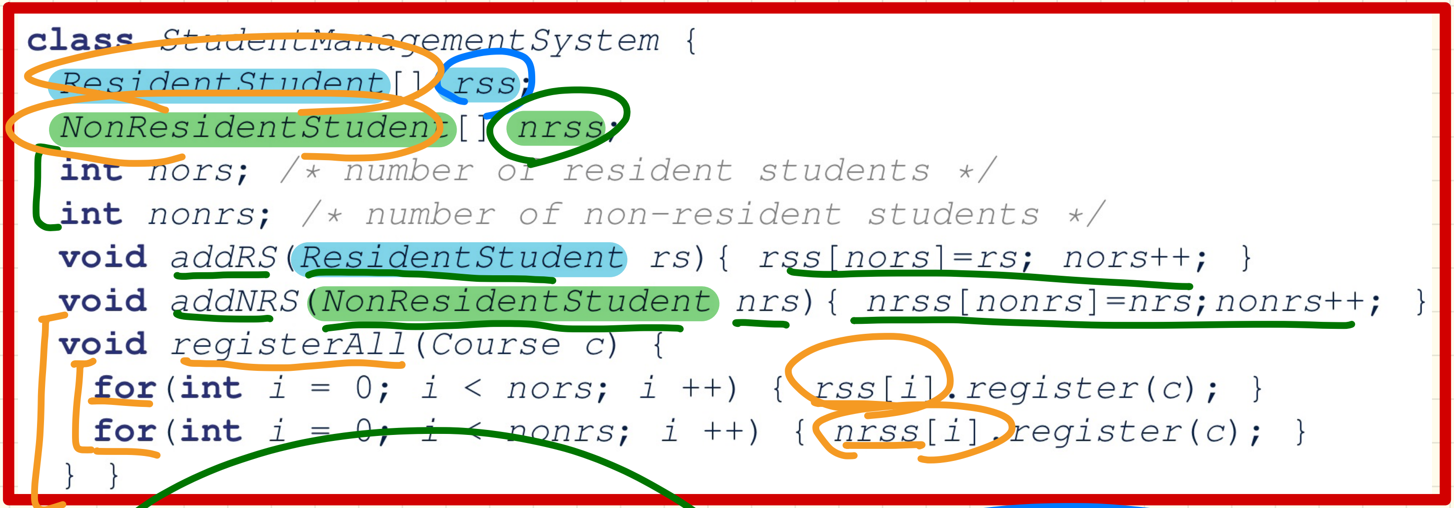


```
class NonResidentStudent {
  String name;
  Course[] registeredCourses;
  int numberOfCourses;
  double discountRate; /* there's a mutator me
  NonResidentStudent (String name) {
    this.name = name;
       registeredCourses = new Course[10];
  }
  void register(Course c) {
    registeredCourses[numberOfCourses] = c;
    numberOfCourses ++;
  }
  double getTuition() {
    double tuition = 0;
    for(int i = 0; i < numberOfCourses; i ++) {
      tuition += registeredCourses[i].fee;
    }
    return tuition * discountRate;
  }
```

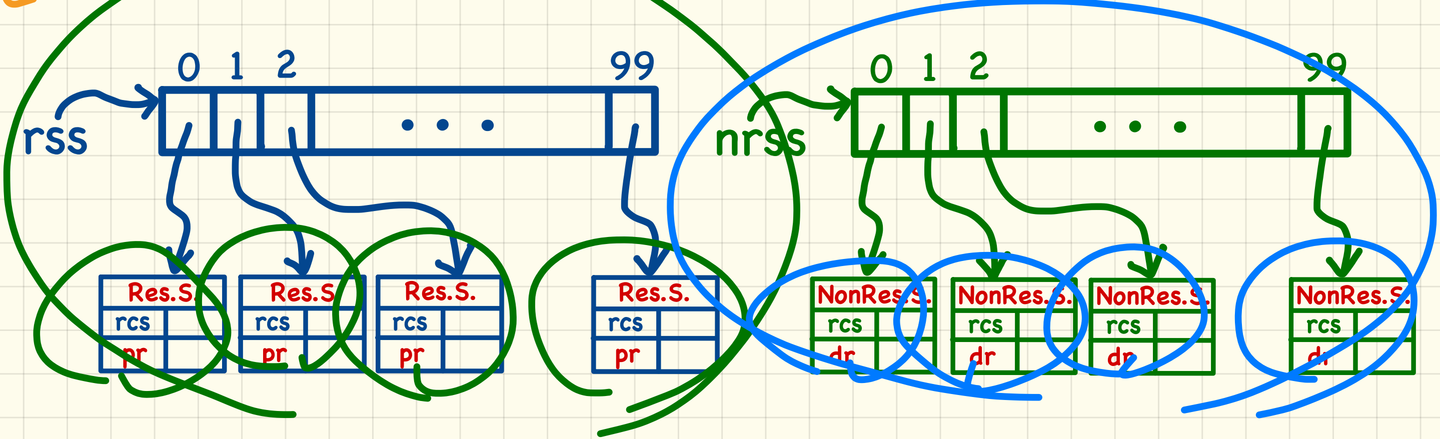
**Maintenance**: e.g., a new formula for tuition

```
/* ... can be premiumRate or discountRate */
...
return tuition * inflationRate * ...;
```

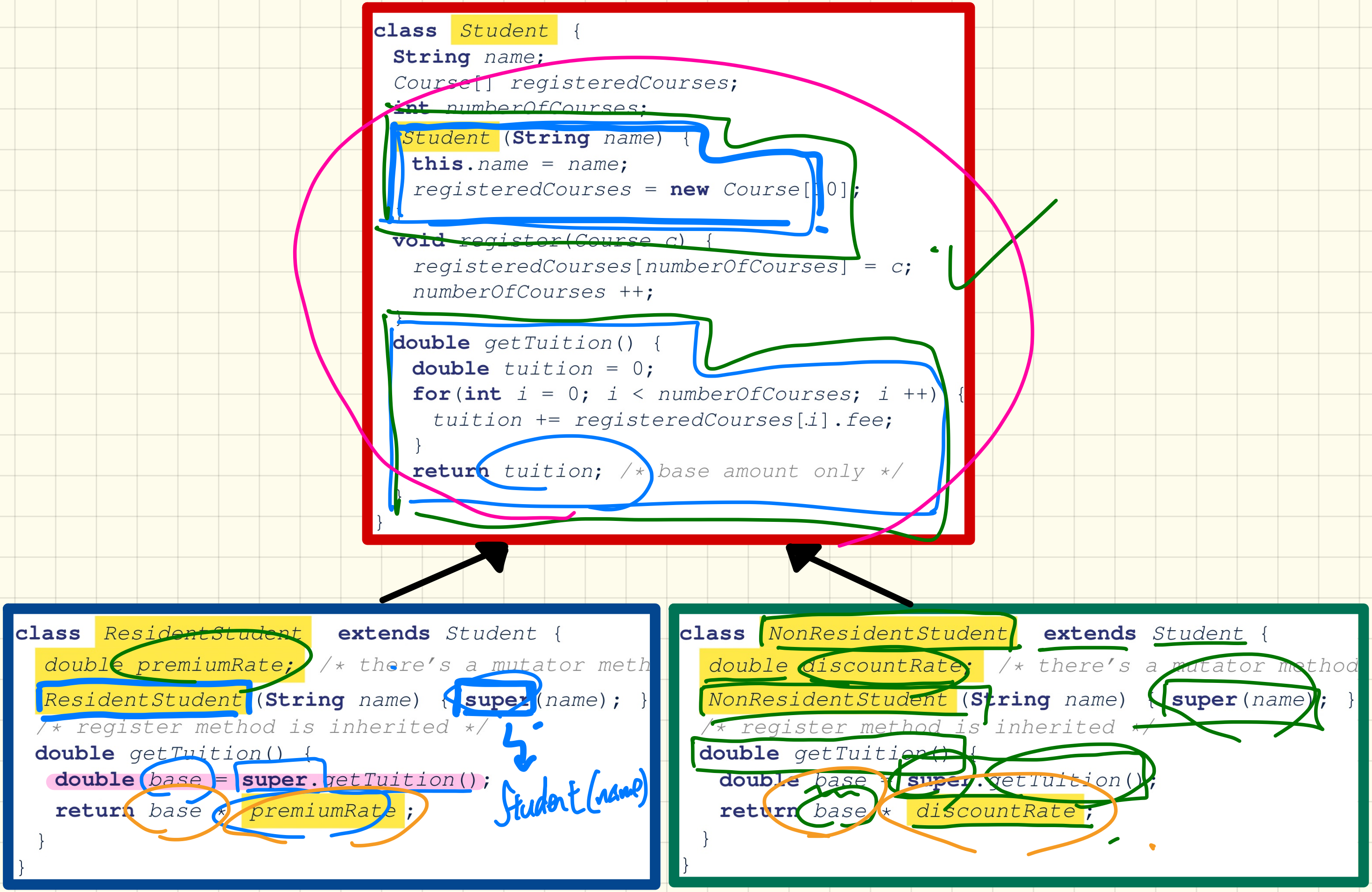
# A Collection of Students (without inheritance)



```
class StudentManagementSystem {
  ResidentStudent[] rss;
  NonResidentStudent[] nrss;
  int nors; /* number of resident students */
  int nonrs; /* number of non-resident students */
  void addRS(ResidentStudent rs){ rss[nors]=rs; nors++; }
  void addNRS(NonResidentStudent nrs){ nrss[nonrs]=nrs;nonrs++; }
  void registerAll(Course c) {
    for(int i = 0; i < nors; i ++) { rss[i].register(c); }
    for(int i = 0; i < nonrs; i ++) { nrss[i].register(c); }
} }
```



# Student Classes (with inheritance)

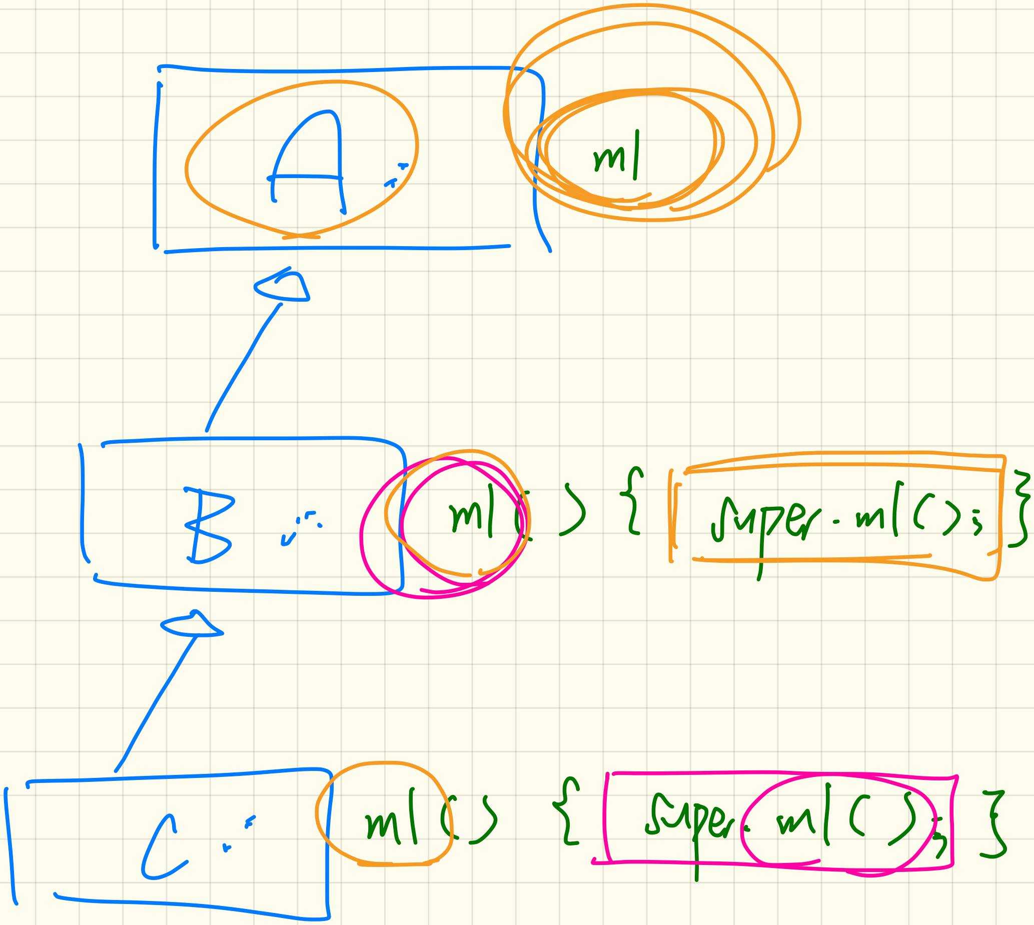


```
class Student {
  String name;
  Course[] registeredCourses;
  int numberOfCourses;
  Student (String name) {
    this.name = name;
    registeredCourses = new Course[10];
  }
  void register(Course c) {
    registeredCourses[numberOfCourses] = c;
    numberOfCourses ++;
  }
  double getTuition() {
    double tuition = 0;
    for(int i = 0; i < numberOfCourses; i ++) {
      tuition += registeredCourses[i].fee;
    }
    return tuition; /* base amount only */
  }
}
```

```
class ResidentStudent extends Student {
  double premiumRate; /* there's a mutator meth
  ResidentStudent (String name) { super(name); }
  /* register method is inherited */
  double getTuition() {
    double base = super.getTuition();
    return base * premiumRate;
  }
}
```

Student(name)

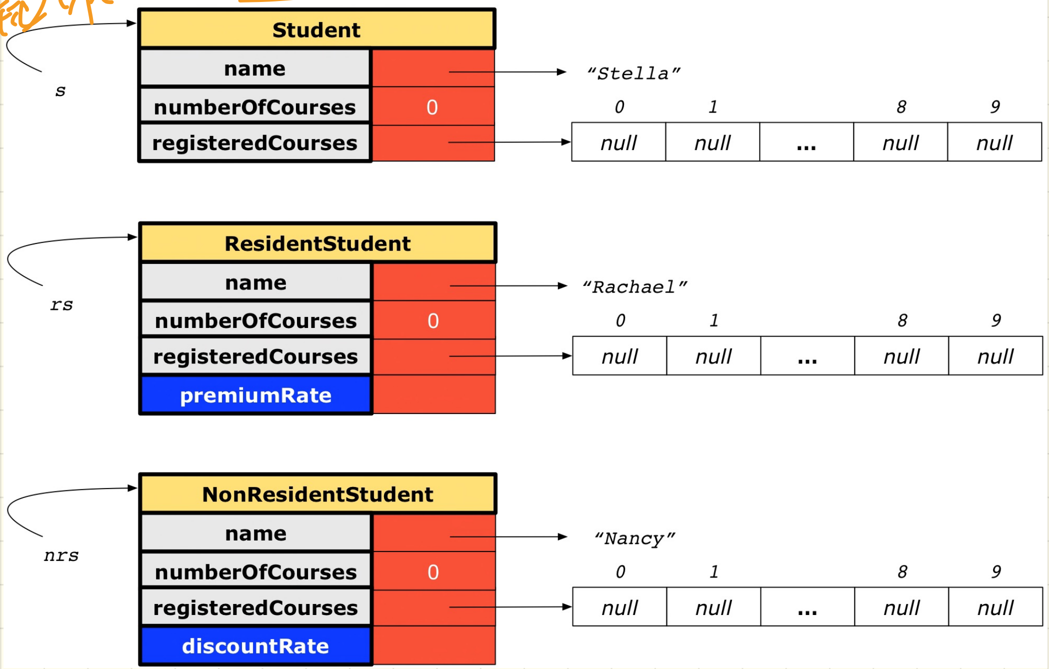
```
class NonResidentStudent extends Student {
  double discountRate; /* there's a mutator method
  NonResidentStudent (String name) { super(name); }
  /* register method is inherited */
  double getTuition() {
    double base = super.getTuition();
    return base * discountRate;
  }
}
```


A :
m1
B :
m1() { super.m1(); }
C :
m1() { super.m1(); }

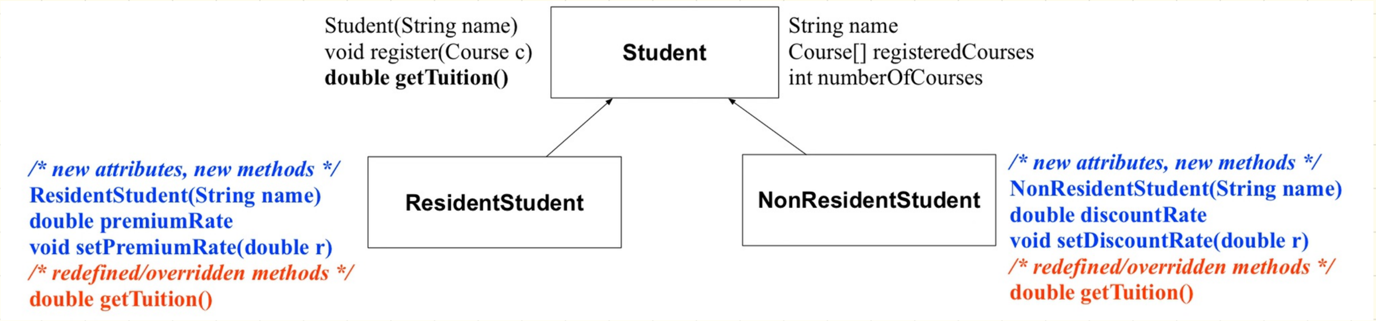
# Visualizing Parent and Child Objects

```
Student s = new Student("Stella");
ResidentStudent rs = new ResidentStudent("Rachael");
NonResidentStudent nrs = new NonResidentStudent("Nancy");
```

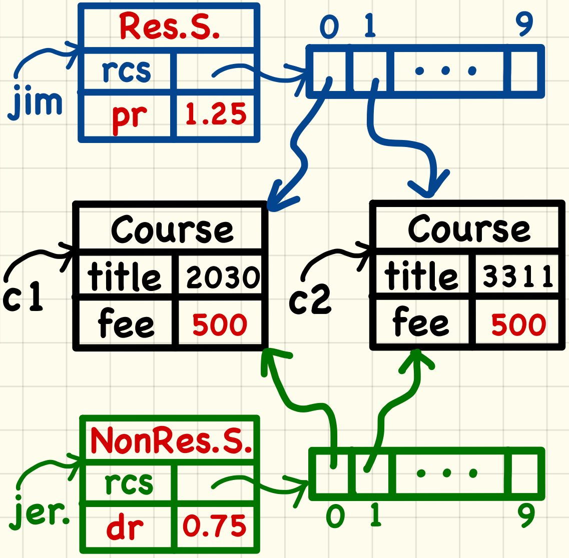
static type



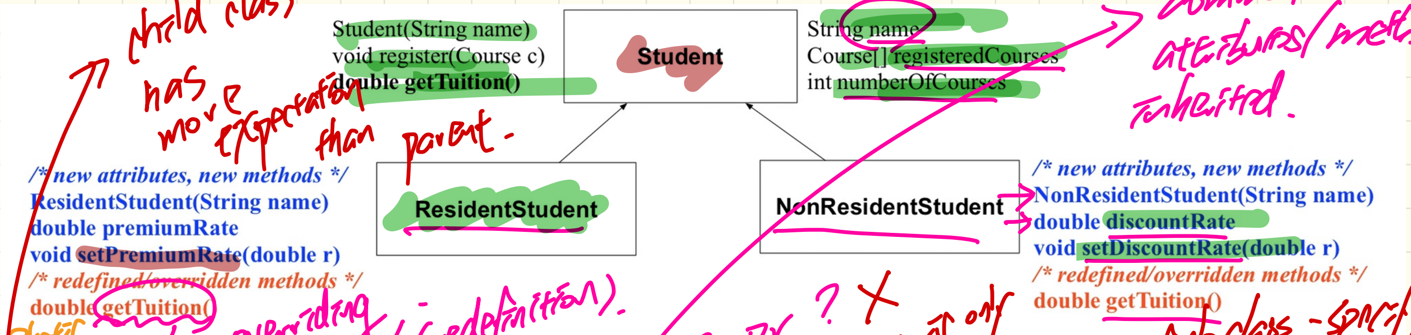
# Testing Student Classes (with inheritance)


Student(String name)
void register(Course c)
double getTuition()
Student
String name
Course[] registeredCourses
int numberOfCourses
/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()
ResidentStudent
NonResidentStudent
/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()

```
class StudentTester {
  static void main(String[] args) {
    Course c1 = new Course("EECS2030", 500.00); /* title and fee */
    Course c2 = new Course("EECS3311", 500.00); /* title and fee */
    ResidentStudent jim = new ResidentStudent("J. Davis");
    jim.setPremiumRate(1.25);
    jim.register(c1); jim.register(c2);
    NonResidentStudent jeremy = new NonResidentStudent("J. Gibbons")
    jeremy.setDiscountRate(0.75);
    jeremy.register(c1); jeremy.register(c2);
    System.out.println("Jim pays " + jim.getTuition());
    System.out.println("Jeremy pays " + jeremy.getTuition());
  }
}
```



# Student Classes (with inheritance): Expectations



Student(String name)
void register(Course c)
**double getTuition()**

Student

String name
Course[] registeredCourses
int numberOfCourses

*/* new attributes, new methods */*
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
*/* redefined/overridden methods */*
double getTuition()

ResidentStudent

NonResidentStudent

*/* new attributes, new methods */*
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
*/* redefined/overridden methods */*
double getTuition()

child class has more expectation than parent.

Common attributes/meth. inherited.

overriding / (redefinition).

static types

s.pr ? ✗ ∵ pr is only in RS

subclass-specific attributes

```
Student s = new Student("Stella");
ResidentStudent rs = new ResidentStudent("Rachael");
NonResidentStudent nrs = new NonResidentStudent("Nancy");
```

| | name | rcs | noc | reg | getT | pr | setPR | dr | setDR |
|---|---|---|---|---|---|---|---|---|---|
| s. | 🟩 | 🟩 | 🟩 | 🟩 | 🟩 | 🟥 | 🟥 | 🟥 | 🟥 |
| rs. | 🟩 | 🟩 | 🟩 | 🟩 | 🟩 | 🟩 | 🟩 | 🟥 | 🟥 |
| nrs. | 🟩 | 🟩 | 🟩 | 🟩 | 🟩 | 🟥 | 🟥 | 🟩 | 🟩 |

T

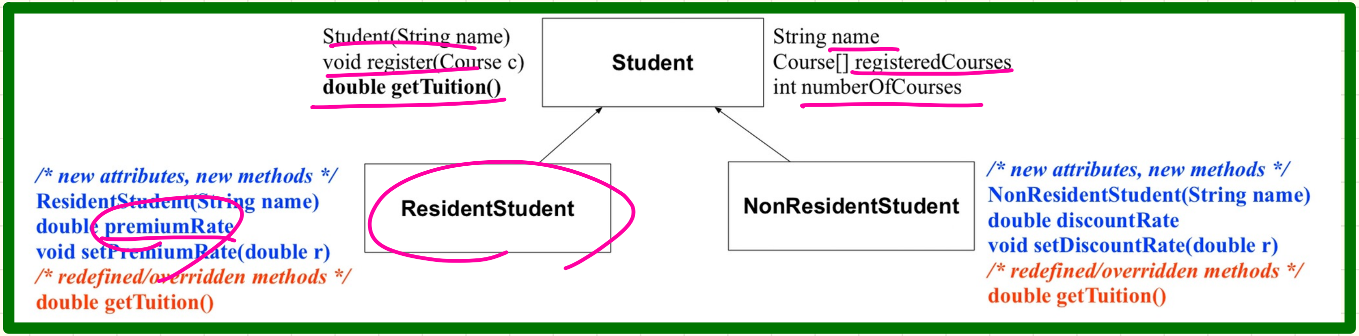
$v$ ; · — —

??

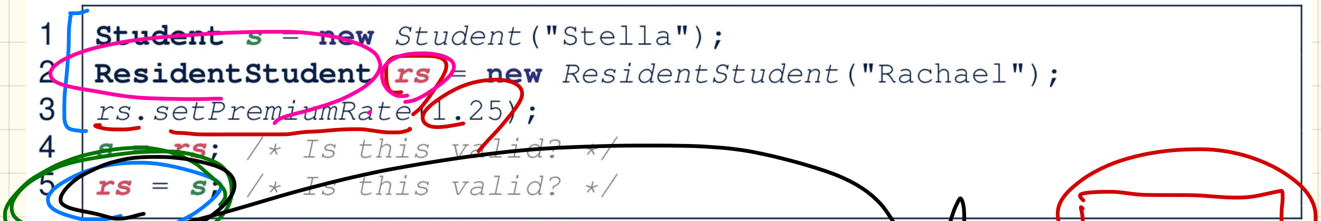
$v \cdot \boxed{\qquad\qquad\qquad\qquad}$

expectation

# Intuition: Polymorphism



Student(String name)
void register(Course c)
double getTuition()
Student
String name
Course[] registeredCourses
int numberOfCourses
/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()
ResidentStudent
NonResidentStudent
/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()



```
Student s = new Student("Stella");
ResidentStudent rs = new ResidentStudent("Rachael");
rs.setPremiumRate(1.25);
s = rs; /* Is this valid? */
rs = s; /* Is this valid? */
```

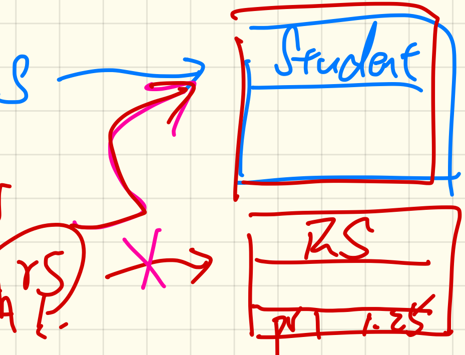
Expectation on rs?

rs. pr

ST: RS

executing this AFTER!

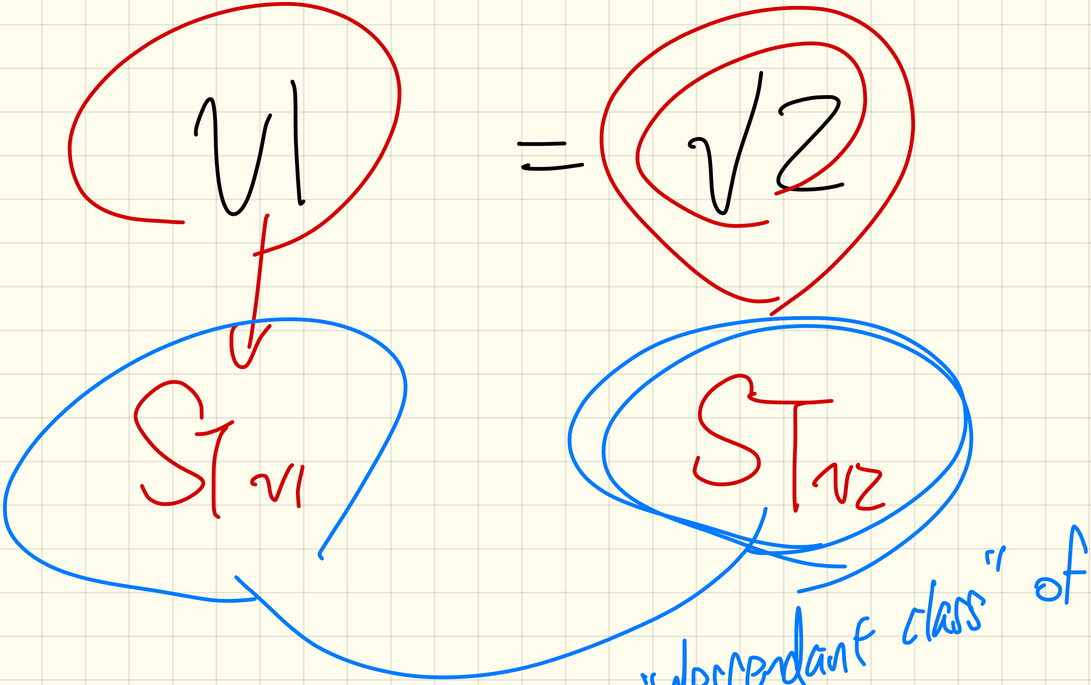
crash!



Assume rs = s compiled.

Executing rs = s

re-direct rs.

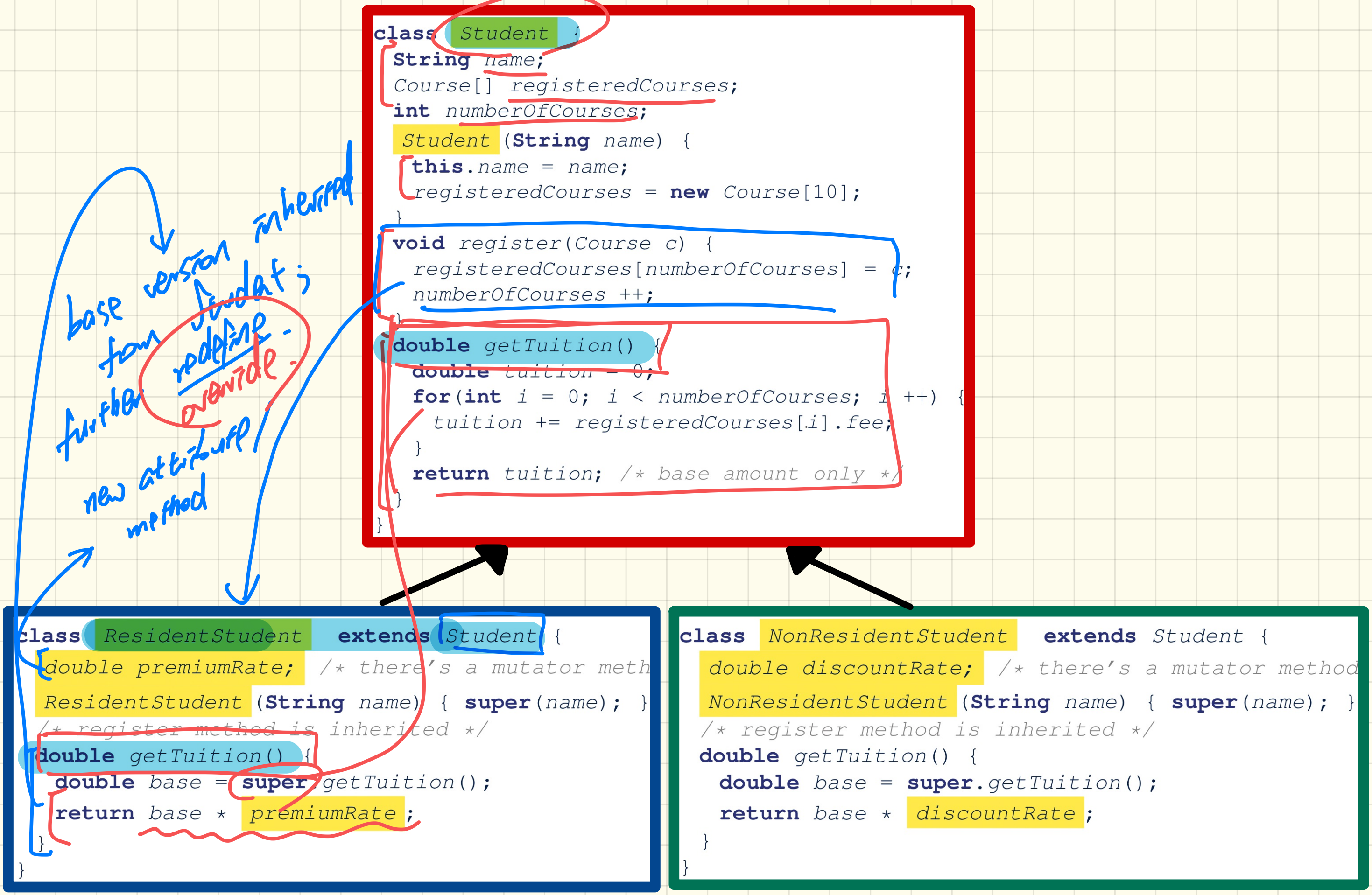


is a "descendant class" of $ST_{v1}$.

# LECTURE 17

# WEDNESDAY NOVEMBER 6

# Review: Student Classes (with inheritance)

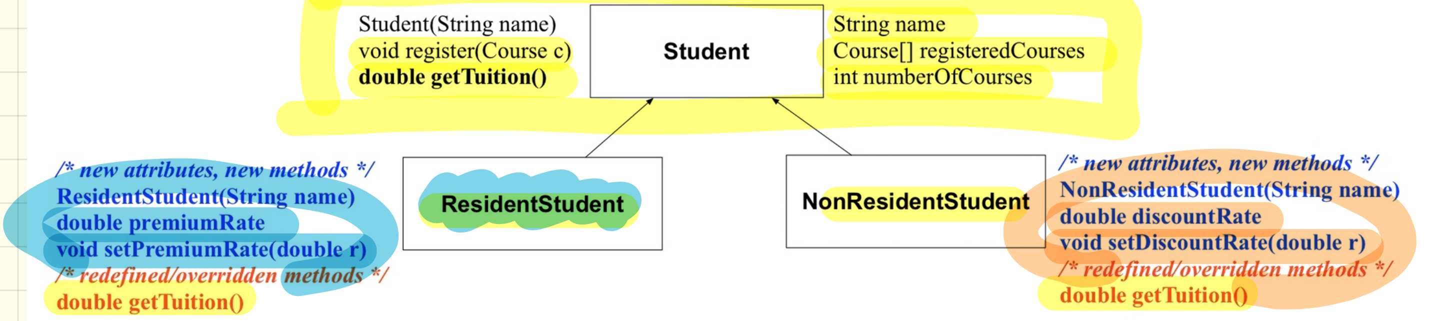


```
class Student {
  String name;
  Course[] registeredCourses;
  int numberOfCourses;
  Student (String name) {
    this.name = name;
    registeredCourses = new Course[10];
  }
  void register(Course c) {
    registeredCourses[numberOfCourses] = c;
    numberOfCourses ++;
  }
  double getTuition() {
    double tuition = 0;
    for(int i = 0; i < numberOfCourses; i ++) {
      tuition += registeredCourses[i].fee;
    }
    return tuition; /* base amount only */
  }
}
```

```
class ResidentStudent extends Student {
  double premiumRate; /* there's a mutator meth
  ResidentStudent (String name) { super(name); }
  /* register method is inherited */
  double getTuition() {
    double base = super.getTuition();
    return base * premiumRate;
  }
}
```

```
class NonResidentStudent extends Student {
  double discountRate; /* there's a mutator method
  NonResidentStudent (String name) { super(name); }
  /* register method is inherited */
  double getTuition() {
    double base = super.getTuition();
    return base * discountRate;
  }
}
```

# Review: Static Types and Expectations



Student(String name)
void register(Course c)
**double getTuition()**

**Student**

String name
Course[] registeredCourses
int numberOfCourses

*/* new attributes, new methods */*
**ResidentStudent(String name)**
**double premiumRate**
**void setPremiumRate(double r)**
***/* redefined/overridden methods */***
**double getTuition()**

**ResidentStudent**

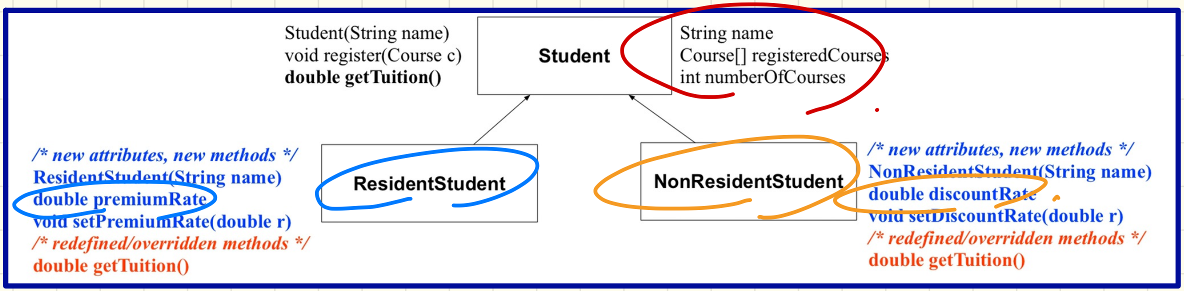
**NonResidentStudent**

*/* new attributes, new methods */*
**NonResidentStudent(String name)**
**double discountRate**
**void setDiscountRate(double r)**
***/* redefined/overridden methods */***
**double getTuition()**

```
Student s = new Student("Stella");
ResidentStudent rs = new ResidentStudent("Rachael");
NonResidentStudent nrs = new NonResidentStudent("Nancy");
```

| | name | rcs | noc | reg | getT | pr | setPR | dr | setDR |
|---|---|---|---|---|---|---|---|---|---|
| s. | 🟩 | 🟩 | 🟩 | 🟩 | 🟩 | 🟥 | 🟥 | 🟥 | 🟥 |
| rs. | 🟩 | 🟩 | 🟩 | 🟩 | 🟩 | 🟩 | 🟩 | 🟥 | 🟥 |
| nrs. | 🟩 | 🟩 | 🟩 | 🟩 | 🟩 | 🟥 | 🟥 | 🟩 | 🟩 |

# Review: Visualizing Parent and Child Objects


Student(String name)
void register(Course c)
double getTuition()
Student
String name
Course[] registeredCourses
int numberOfCourses
/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()
ResidentStudent
NonResidentStudent
/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()

## Inheritance Hirarchy

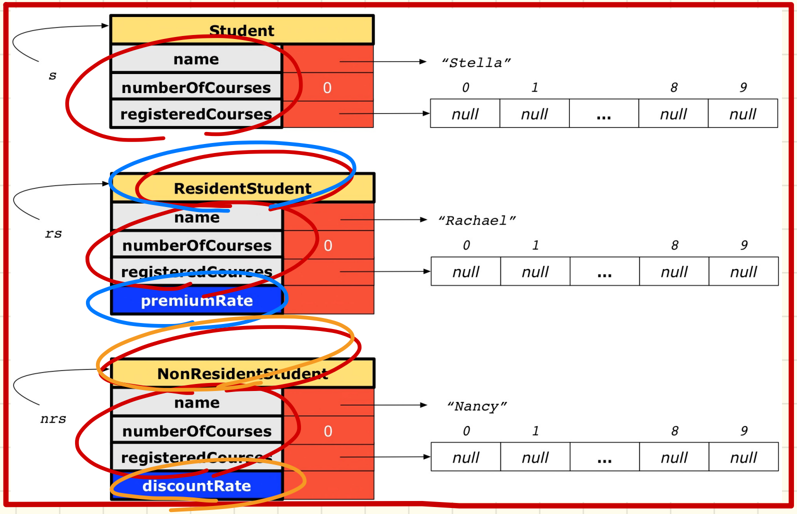
```
Student s = new Student("Stella");
ResidentStudent rs = new ResidentStudent("Rachael");
NonResidentStudent nrs = new NonResidentStudent("Nancy");
```

## Declaring Static Types

static types

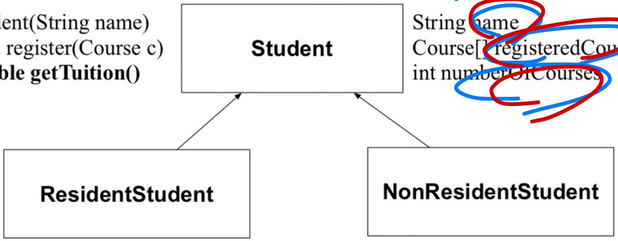
dynamic types

## Runtime Object Structure



# Intuition: Polymorphism

→ assignments in context of inheritance



Student(String name)
void register(Course c)
**double getTuition()**

String name
Course[] registeredCourses
int numberOfCourses

*/* new attributes, new methods */*
**ResidentStudent(String name)**
**double premiumRate**
**void setPremiumRate(double r)**
*/* redefined/overridden methods */*
**double getTuition()**

*/* new attributes, new methods */*
**NonResidentStudent(String name)**
**double discountRate**
**void setDiscountRate(double r)**
*/* redefined/overridden methods */*
**double getTuition()**

ST

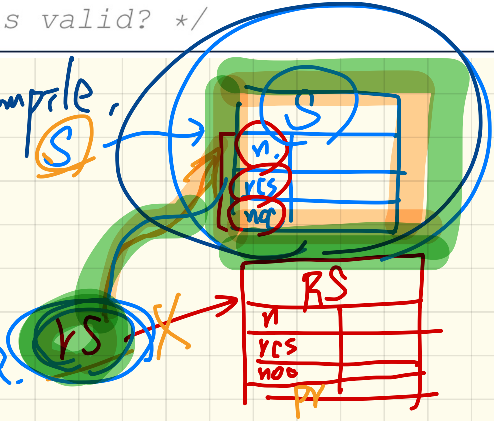
```
1  Student s = new Student("Stella");
2  ResidentStudent rs = new ResidentStudent("Rachael");
3  rs.setPremiumRate(1.25);
4  s = rs; /* Is this valid? */
5  rs = s; /* Is this valid? */
```

✓ ✓ X

Assume L5 should compile.

Execute rs = s;

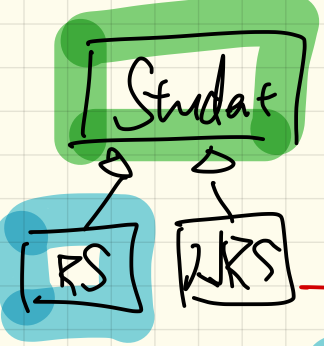
Crash! L5 should not compile.



Expectations on rs.

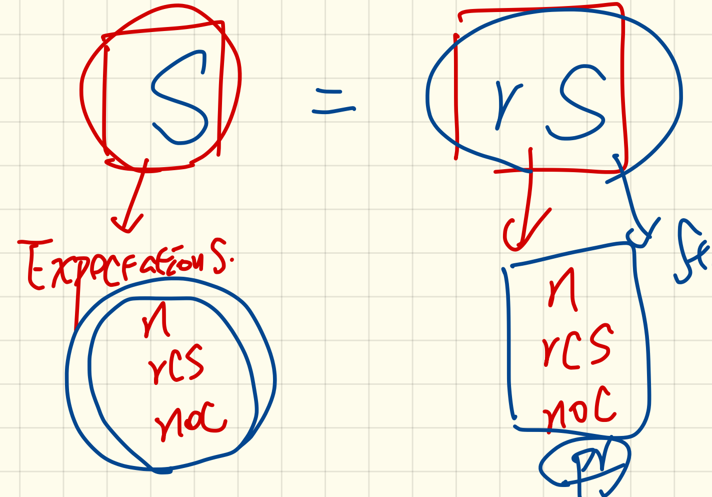
rs. n
rcs
noc
pr.

rs. pr.



Student s = . . _

ResidentStudent rS = . . _

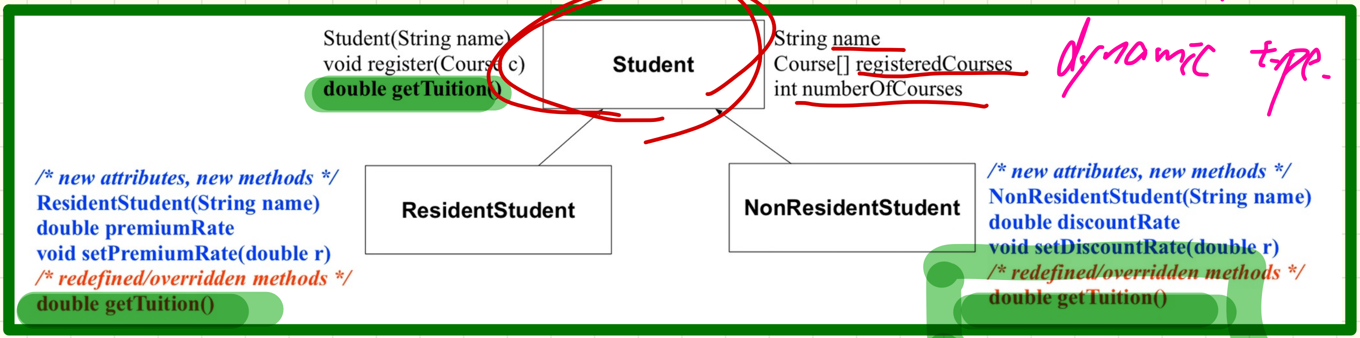


valid

staticType of rS (RS) is a child class of ST of s (Student)

# Intuition: Dynamic Binding

→ runtime behaviour depends on dynamic type.



Student(String name)
void register(Course c)
double getTuition()

Student

String name
Course[] registeredCourses
int numberOfCourses

/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()

ResidentStudent

NonResidentStudent

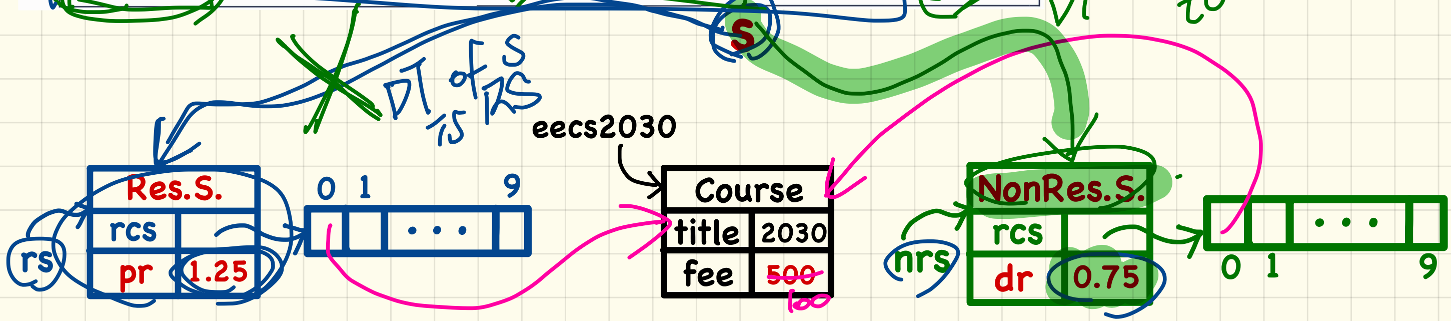
/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()

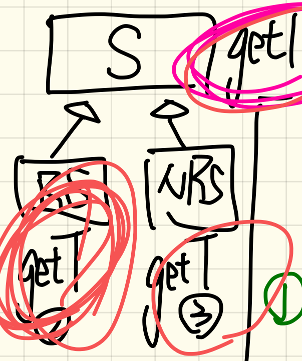
```
Course eecs2030 = new Course("EECS2030", 100.0);
Student s;
ResidentStudent rs = new ResidentStudent("Rachael");
NonResidentStudent nrs = new NonResidentStudent("Nancy");
rs.setPremiumRate(1.25); rs.register(eecs2030);
nrs.setDiscountRate(0.75); nrs.register(eecs2030);
s = rs; System.out.println(s.getTuition()); /* output: 125.0 */
s = nrs; System.out.println(s.getTuition()); /* output: 75.0 */
```

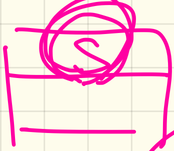
S.pr X

ST: Student

DT of S to NRS





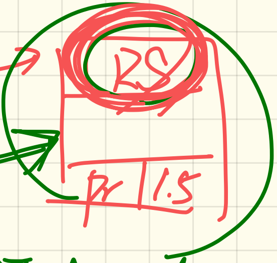


Student s = new Student();

s. getTuition(); ①

RS rs = new RS();

rs.setPr(1.5);



Compilation

s.pr X

rs.pr ✓

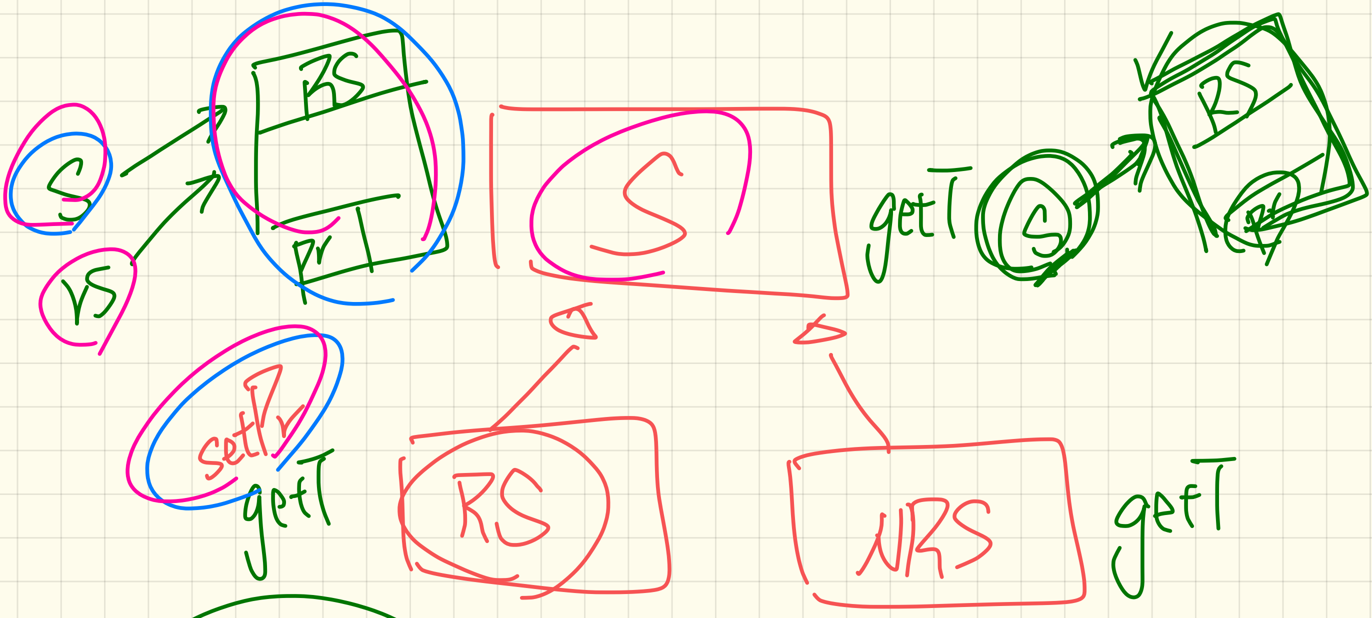
② s = rs;

③.1 s.pr X ∵ ST of s doesn't support

③.2 s.getTuition();

ⓐ It compiles ∵ ST of S (Stud) supports that.

ⓑ Which version of getT is called? RS ∵ DT of S is RS ∴ call RS version.



Student s;

RS rs = new RS(..);

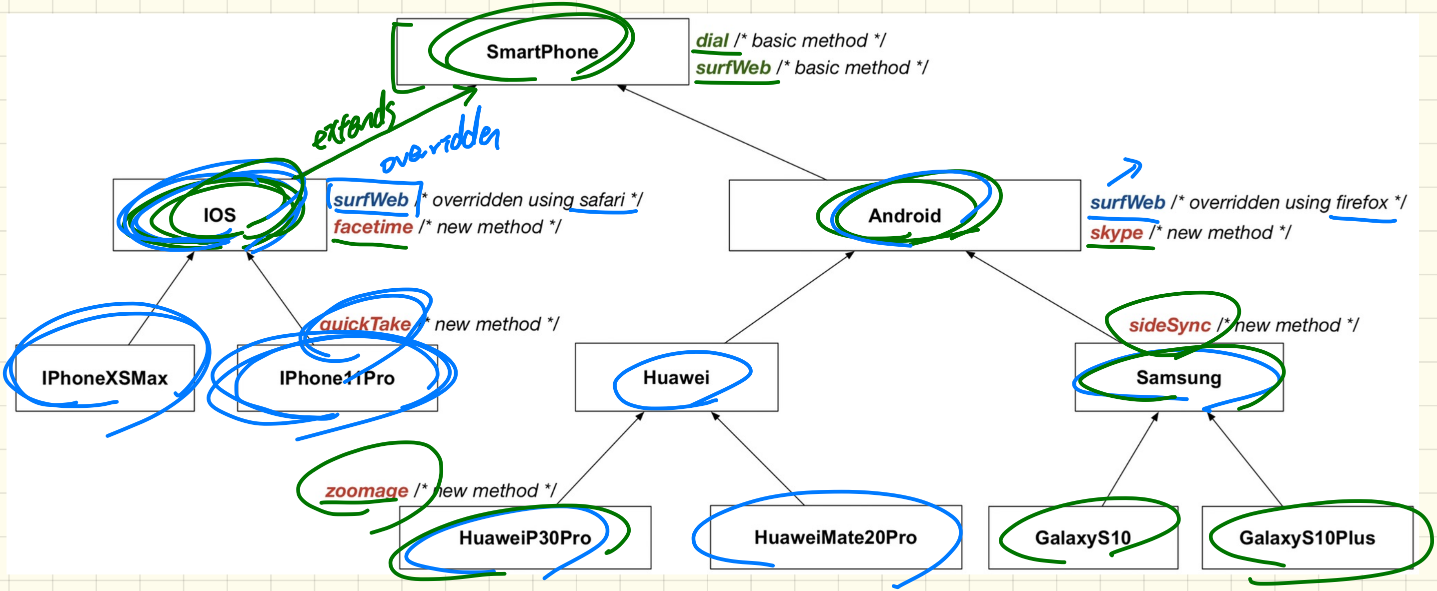
rs. setPr(..);

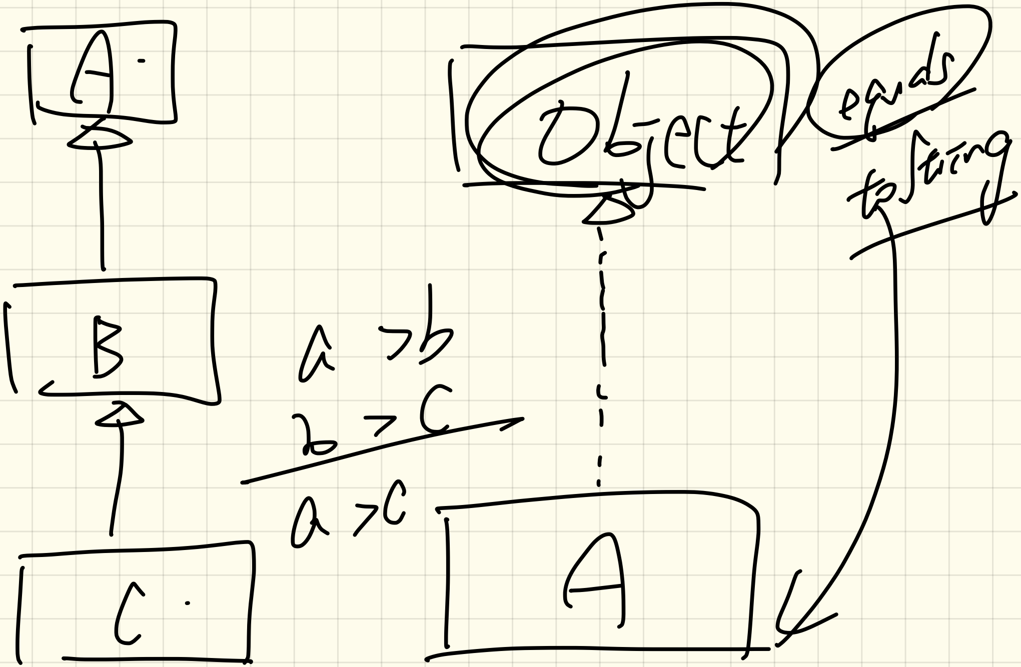
s = rs;

s. setPr(1.5); ? X

rs. setPr(2.7);

# Multi-Level Inheritance Hierarchy of Smartphones



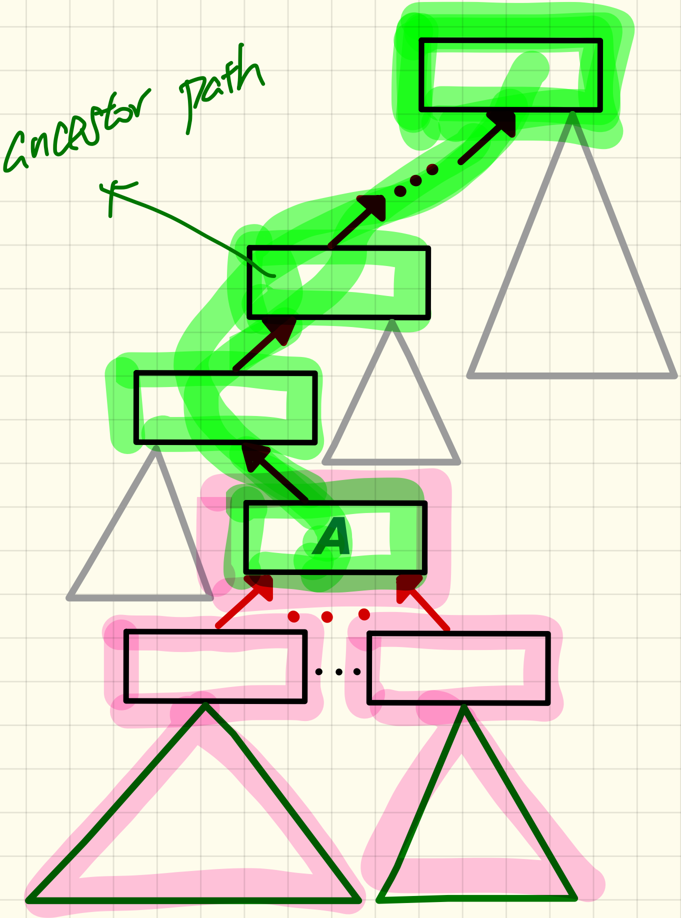

A
B
C
a >b
b > c
a >c
Object
equals
toString
A

# Inheritance Forms a Type Hierarchy

ancestors of A

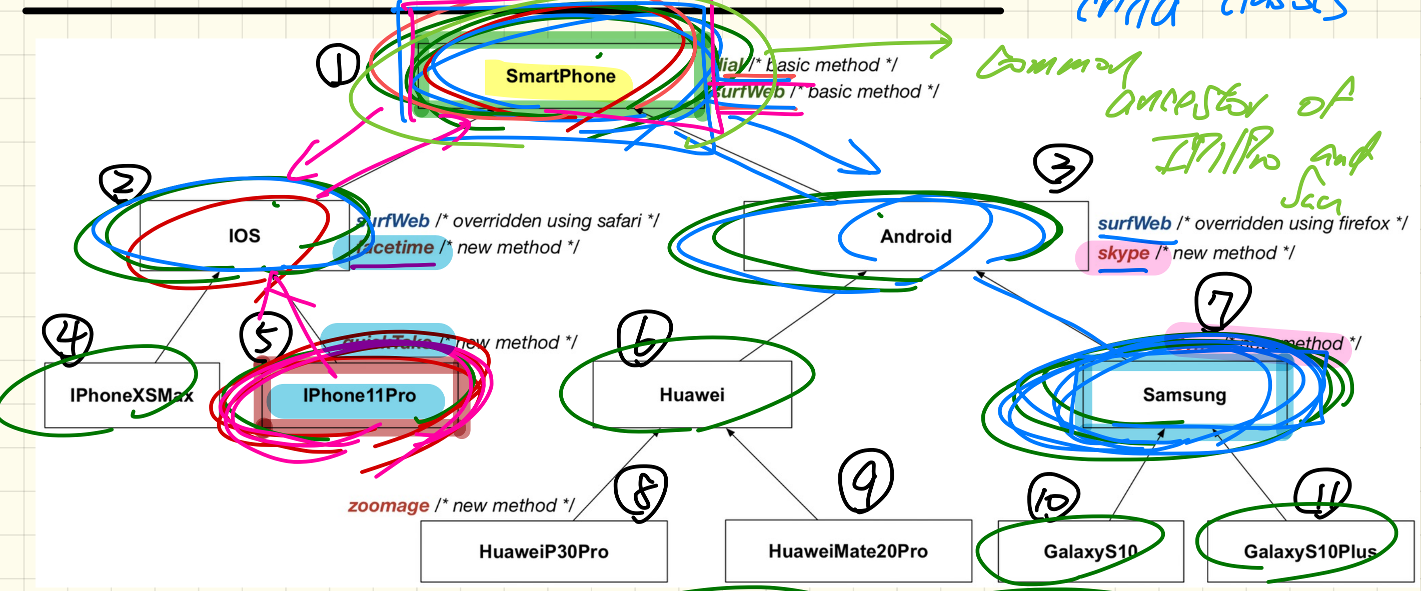
descendants of A

ancestor path



# Inheritance Accumulates Code for Reuse

child classes



SmartPhone

dial /* basic method */

surfWeb /* basic method */

IOS

surfWeb /* overridden using safari */

facetime /* new method */

Android

surfWeb /* overridden using firefox */

skype /* new method */

IPhoneXSMax

IPhone11Pro

quickTake /* new method */

Huawei

Samsung

/* new method */

zoomage /* new method */

HuaweiP30Pro

HuaweiMate20Pro

GalaxyS10

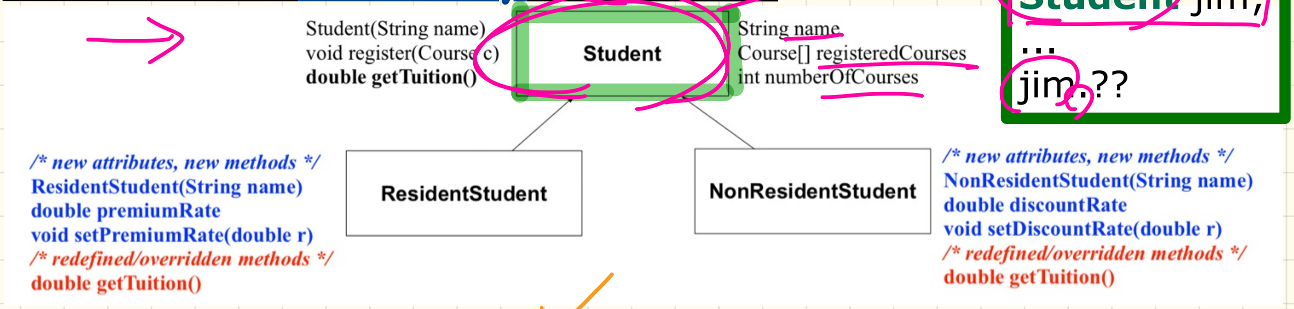
GalaxyS10Plus

Common ancestor of IP11Pro and Sam

| | ancestors | expectations | descendants |
|---|---|---|---|
| 🟩 | SmartPhone | dial, surfWeb | ① ~ ⑪. |
| 🟦 | SanSung, Android, SP | dial, surfWeb, skype, side sync | ⑦ ⑩ ⑪ |
| 🟥 | IP11Pro, IOS, SP | dial, surfWeb, faceTime, quickTake | ⑤. |

# Static Types determine Expectations

## Inheritance Herarchy: Students



Student(String name)
void register(Course c)
**double getTuition()**

**Student**

String name
Course[] registeredCourses
int numberOfCourses

***/* new attributes, new methods */***
**ResidentStudent(String name)**
**double premiumRate**
**void setPremiumRate(double r)**
***/* redefined/overridden methods */***
**double getTuition()**

**ResidentStudent**

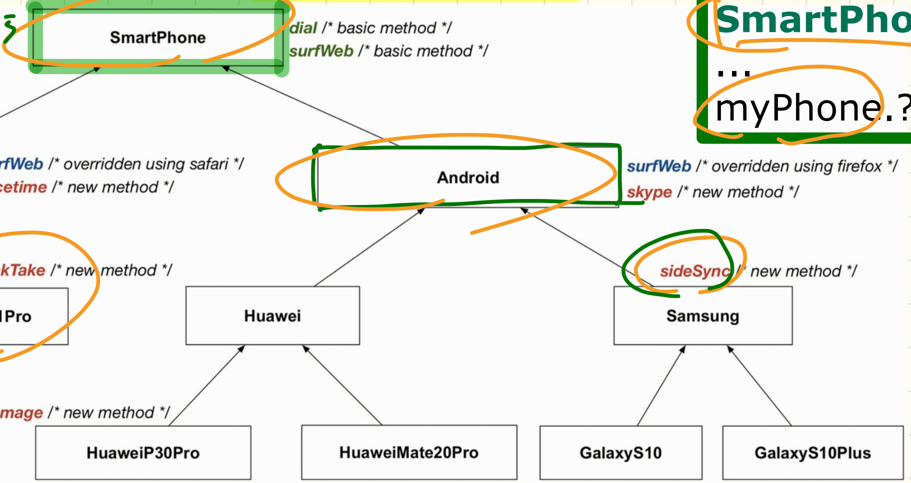
**NonResidentStudent**

***/* new attributes, new methods */***
**NonResidentStudent(String name)**
**double discountRate**
**void setDiscountRate(double r)**
***/* redefined/overridden methods */***
**double getTuition()**

**Declare:**
**Student** jim;
...
jim.??

## Inheritance Herarchy: Smart Phones

Android p1 = ...;
p1 . sideSync X
p1 . quickTake



**SmartPhone**

*dial* /* basic method */
*surfWeb* /* basic method */

**IOS**

*surfWeb* /* overridden using safari */
*facetime* /* new method */

**Android**

*surfWeb* /* overridden using firefox */
*skype* /* new method */

**IPhoneXSMax**

*quickTake* /* new method */

**IPhone11Pro**

**Huawei**

*sideSync* /* new method */

**Samsung**

*zoomage* /* new method */

**HuaweiP30Pro**

**HuaweiMate20Pro**

**GalaxyS10**

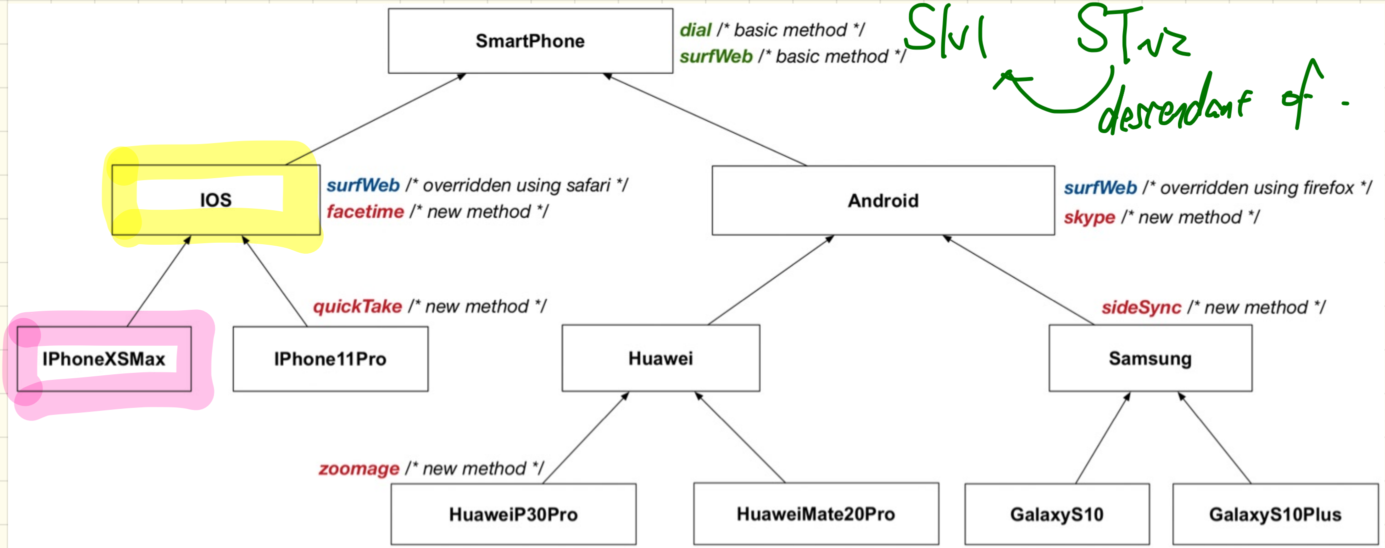
**GalaxyS10Plus**

**Declare:**
**SmartPhone** myPhone;
...
myPhone.??

# Rules of Substitutions (1)

$$\frac{v1}{ST_{v1}} = \frac{v2}{ST_{v2}}$$

descendant of.


SmartPhone
dial /* basic method */
surfWeb /* basic method */
IOS
surfWeb /* overridden using safari */
facetime /* new method */
Android
surfWeb /* overridden using firefox */
skype /* new method */
quickTake /* new method */
sideSync /* new method */
IPhoneXSMax
IPhone11Pro
Huawei
Samsung
zoomage /* new method */
HuaweiP30Pro
HuaweiMate20Pro
GalaxyS10
GalaxyS10Plus

**Declarations:**

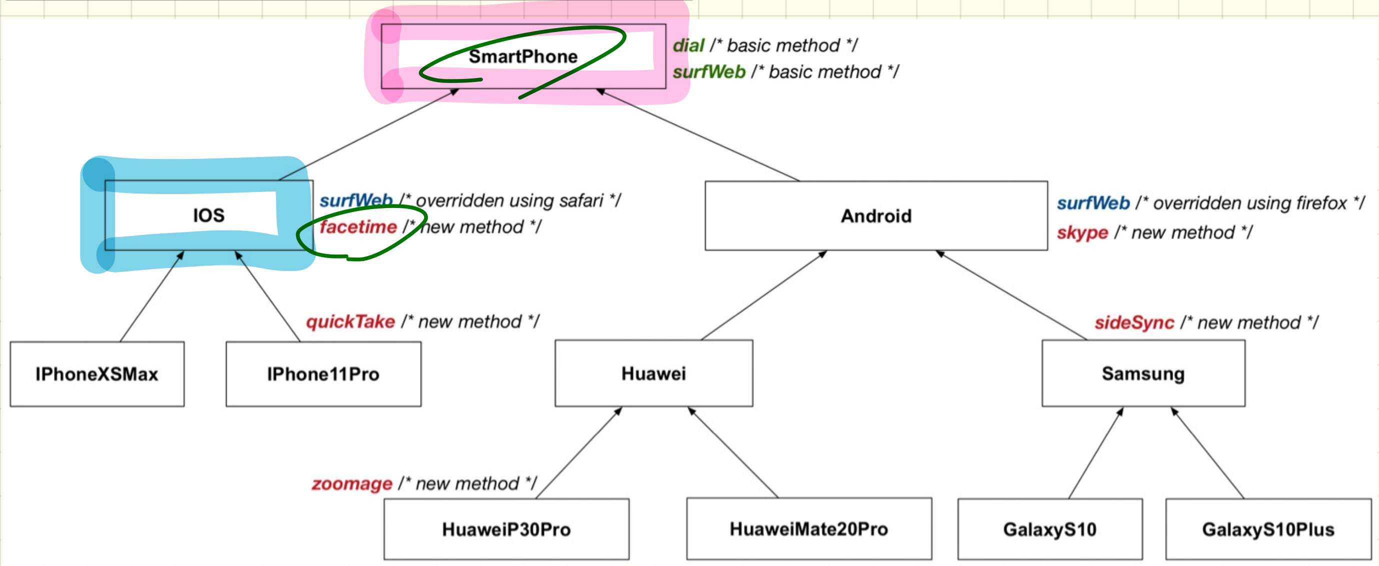
```
IOS sp1;
IPhoneXSMax sp2;
IPhonePro11 sp3;
```

**Substitutions:**

```
sp1 = sp2;
sp1 = sp3;
```

can the ST of sp2 fullfill expectations of ST of sp1

# Rules of Substitutions (2)


SmartPhone
dial /* basic method */
surfWeb /* basic method */
IOS
surfWeb /* overridden using safari */
facetime /* new method */
Android
surfWeb /* overridden using firefox */
skype /* new method */
quickTake /* new method */
sideSync /* new method */
IPhoneXSMax
IPhone11Pro
Huawei
Samsung
zoomage /* new method */
HuaweiP30Pro
HuaweiMate20Pro
GalaxyS10
GalaxyS10Plus

**Declarations:**

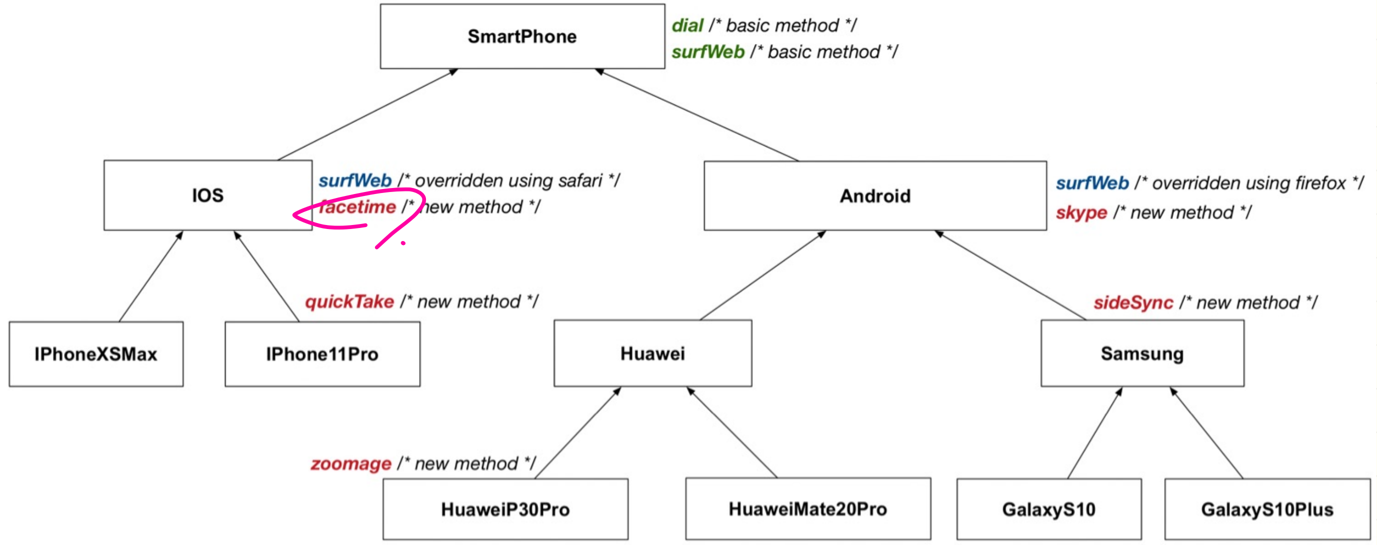
```
IOS sp1;
SmartPhone sp2;
```

**Substitutions:**

```
sp1 = sp2;
```

ST: IOS  ST: SP

# Rules of Substitutions (3)



SmartPhone
dial /* basic method */
surfWeb /* basic method */

IOS
surfWeb /* overridden using safari */
facetime /* new method */

Android
surfWeb /* overridden using firefox */
skype /* new method */

IPhoneXSMax

IPhone11Pro
quickTake /* new method */

Huawei

Samsung
sideSync /* new method */

HuaweiP30Pro
zoomage /* new method */

HuaweiMate20Pro

GalaxyS10

GalaxyS10Plus

?

**Declarations:**

```
IOS sp1;
HuaweiP30Pro sp2;
```

**Substitutions:**

```
sp1 = sp2;
```

ST: IOS

ST: HuaweiP30Pro

e.g. facetime cannot be fulfilled by

# Visualization: Static Type vs. Dynamic Type

Declaration:

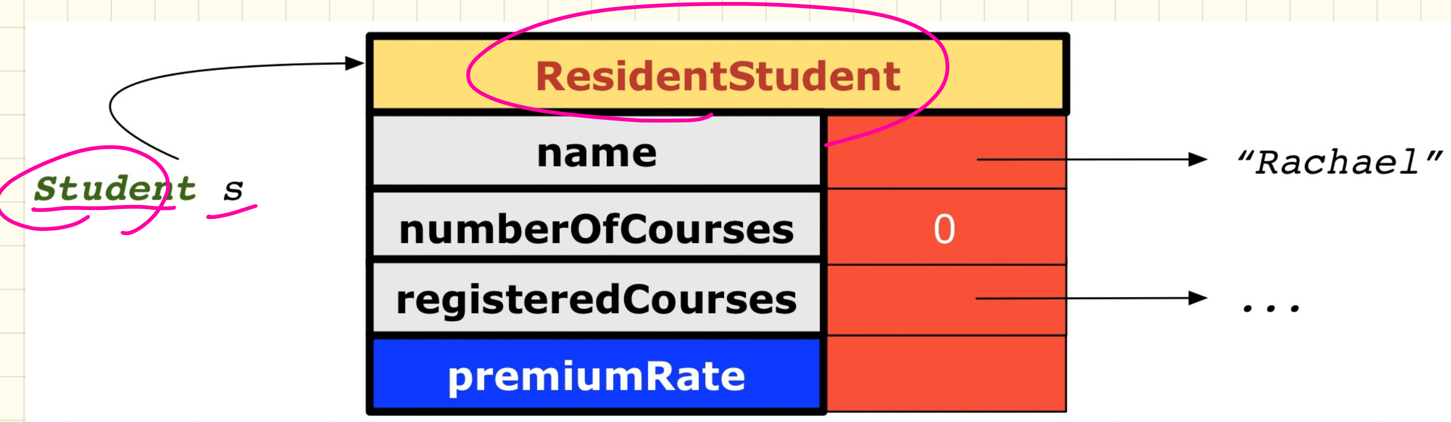
```
Student s;
```

ST

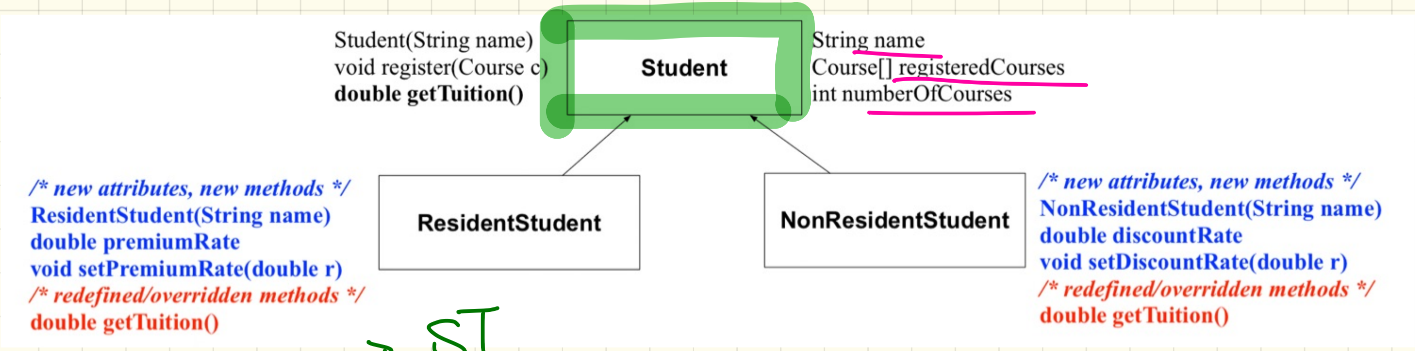
Substitution:

```
s = new ResidentStudent("Rachael");
```

DT



# Change of Dynamic Type (1.1)


Student(String name)
void register(Course c)
double getTuition()
Student
String name
Course[] registeredCourses
int numberOfCourses
/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()
ResidentStudent
NonResidentStudent
/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()

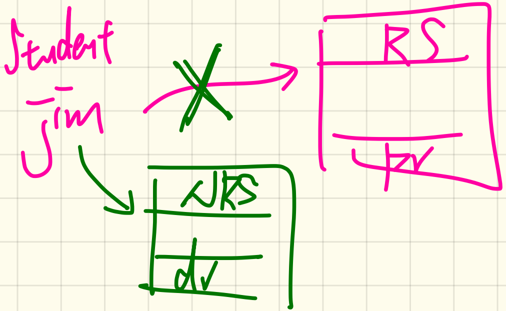
ST

**Example 1:**

```
Student jim = new ResidentStudent(...);
jim = new NonResidentStudent(...);
```

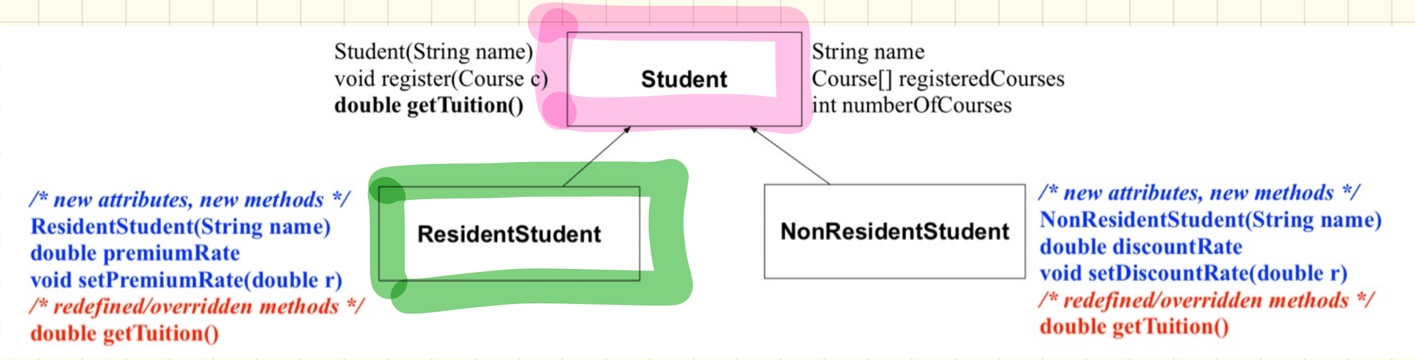
→ DT of jim: RS

→ DT of jim NRS



can RS (the attempted DT) of jim. fullfil the expectation (defined by ST Student)

# Change of Dynamic Type (1.2)



Student(String name)
void register(Course c)
**double getTuition()**

String name
Course[] registeredCourses
int numberOfCourses

/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()

/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()

**Example 2:**

```
ResidentStudent jeremy = new Student(...); ✗
```

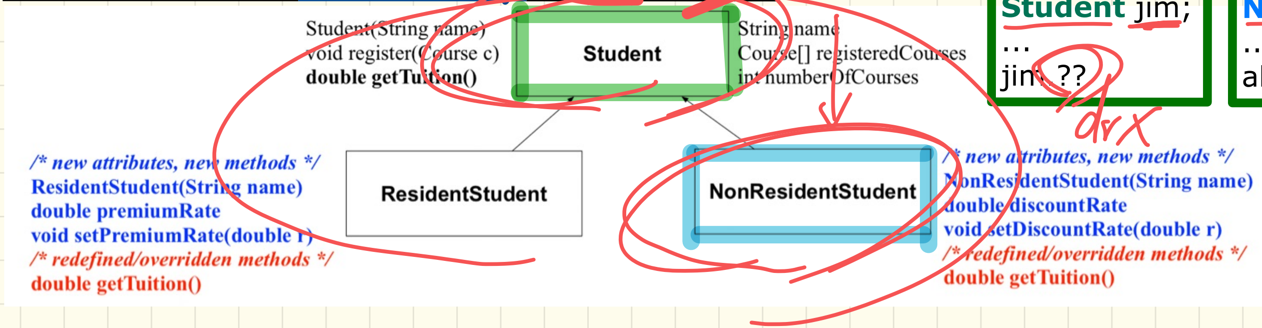
∵ Student cannot fulfill expectations of RS

# LECTURE 18

MONDAY NOVEMBER 11

# Static Types determine Expectations

## Inheritance Herarchy: Students



Student(String name)
void register(Course c)
double getTuition()

Student

String name
Course[] registeredCourses
int numberOfCourses

/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()

ResidentStudent

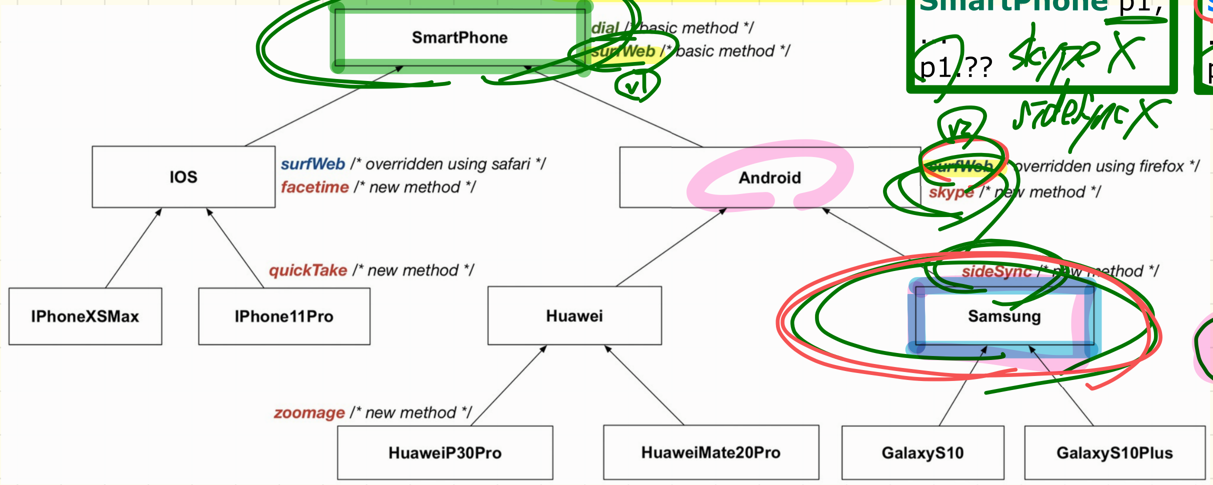
NonResidentStudent

/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()

**Declare**:
```
Student jim;
...
jim.??
```
dr X

**Declare**:
```
NRS alan;
...
alan.??
```
dr

## Inheritance Herarchy: Smart Phones



SmartPhone

dial /* basic method */
surfWeb /* basic method */

v1

IOS

surfWeb /* overridden using safari */
facetime /* new method */

Android

surfWeb /* overridden using firefox */
skype /* new method */

v2

IPhoneXSMax

quickTake /* new method */

IPhone11Pro

Huawei

sideSync /* new method */

Samsung

zoomage /* new method */

HuaweiP30Pro

HuaweiMate20Pro

GalaxyS10

GalaxyS10Plus

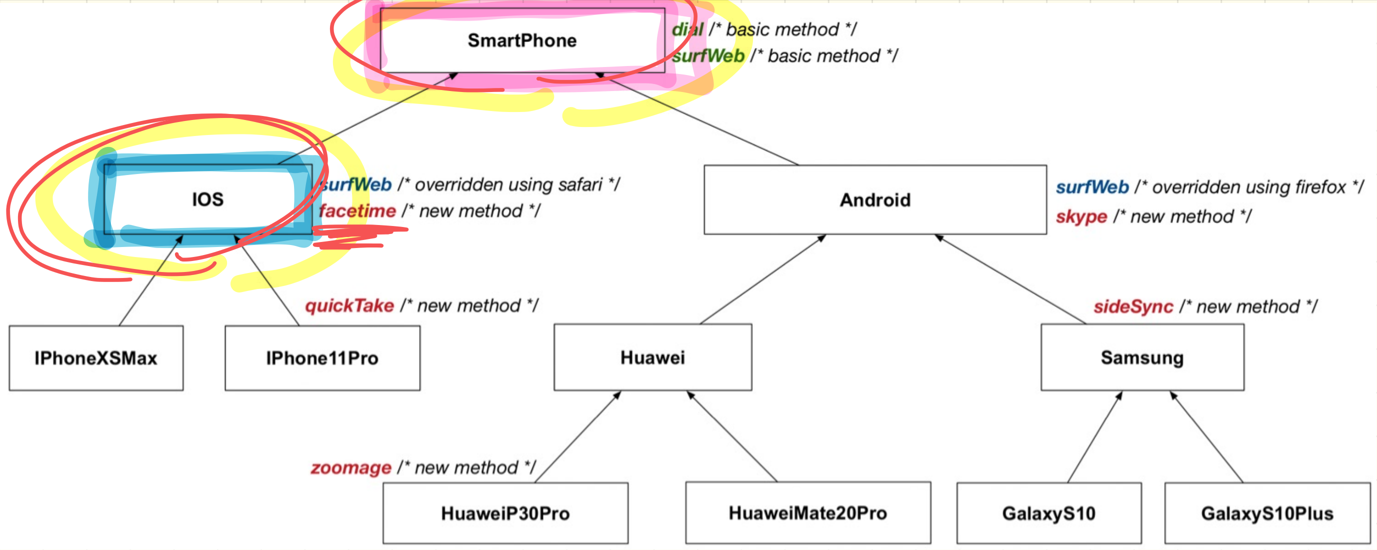
**Declare**:
```
SmartPhone p1;
...
p1.??
```
skype X
sideSync X

**Declare**:
```
Samsung p2;
...
p2.??
```
p2 = new Samsung();

skype
sideSync

p2. surfWeb ✓

# Rules of Substitutions



SmartPhone
dial /* basic method */
surfWeb /* basic method */
IOS
surfWeb /* overridden using safari */
facetime /* new method */
Android
surfWeb /* overridden using firefox */
skype /* new method */
IPhoneXSMax
IPhone11Pro
quickTake /* new method */
Huawei
Samsung
sideSync /* new method */
zoomage /* new method */
HuaweiP30Pro
HuaweiMate20Pro
GalaxyS10
GalaxyS10Plus

**Declarations:**

```
IOS sp1;
SmartPhone sp2;
```

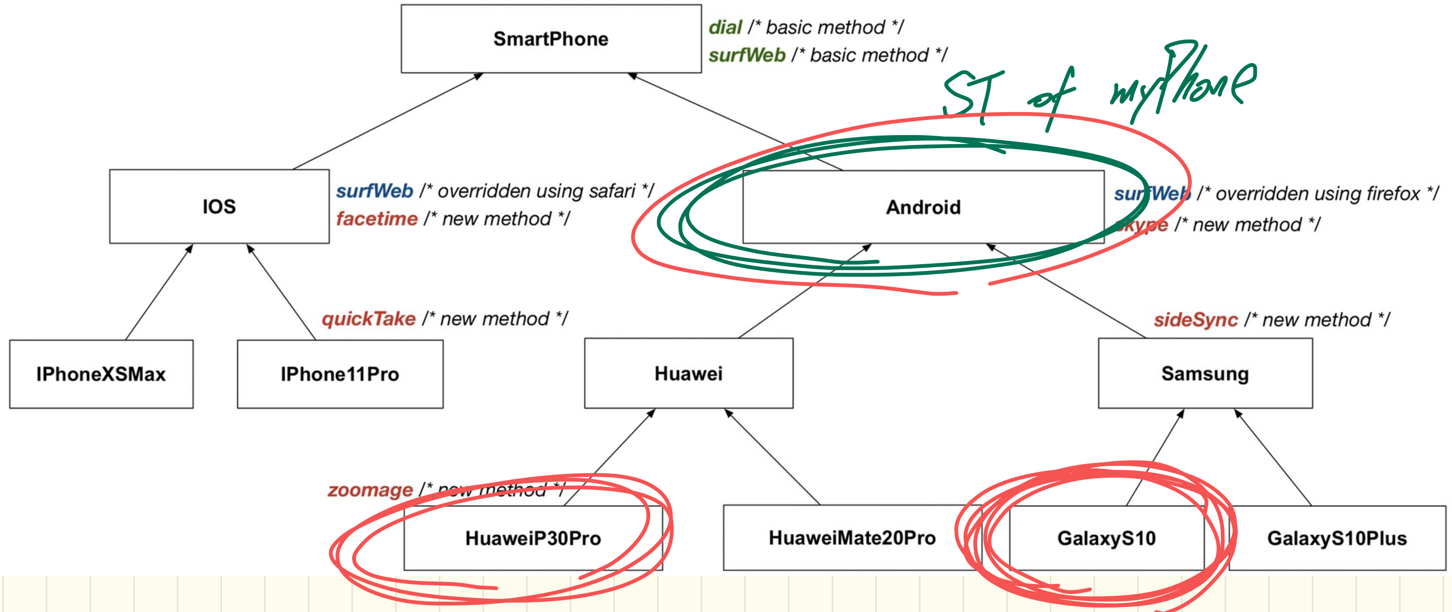
**Substitutions:**

```
sp1 = sp2;
```

Can ST of sp2 fulfill what's expected on ST of sp1?

NO ∵ facetime not support on sp2.

# Change of Dynamic Type: Exercise (1)

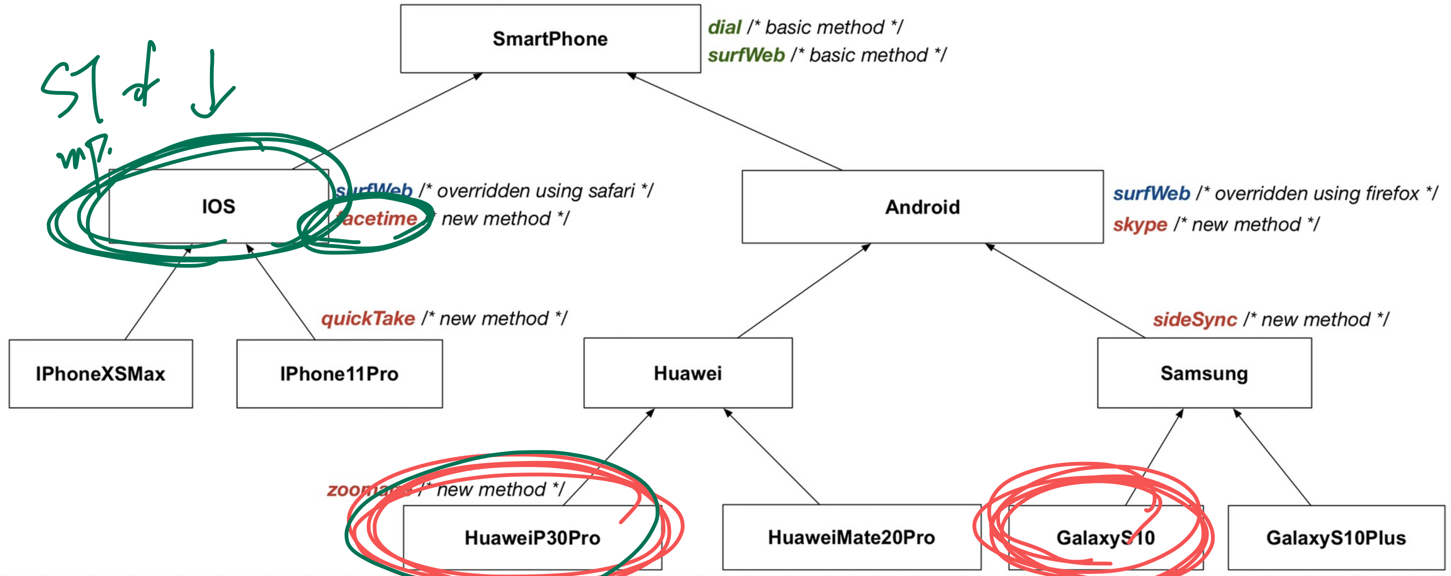


## Exercise 1:

```
Android myPhone = new HuaweiP30Pro(...);
myPhone = new GalaxyS10(...);
```

✓

# Change of Dynamic Type: Exercise (2)



## Exercise 2:

```
IOS myPhone = new HuaweiP30Pro(...); ①
myPhone = new GalaxyS10(...); ②
```

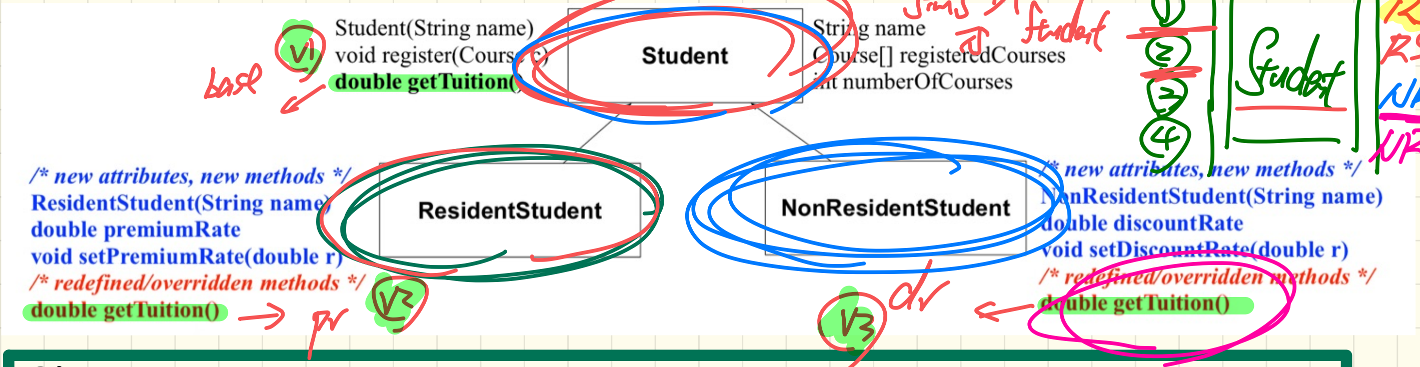
X ∵ e.g. facetime not supported

X ∵ e.g. f.t. not supported.

# Change of Dynamic Type (2.1)

before ①: jim's DT ⇓ Student

| after | ST of jim | DT of jim |
|---|---|---|
| ① | Student | RS |
| ② | | RS |
| ③ | | NRS |
| ④ | | NRS |



Student(String name)
void register(Course c)
double getTuition()

String name
Course[] registeredCourses
int numberOfCourses

/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()

/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()

## Given:

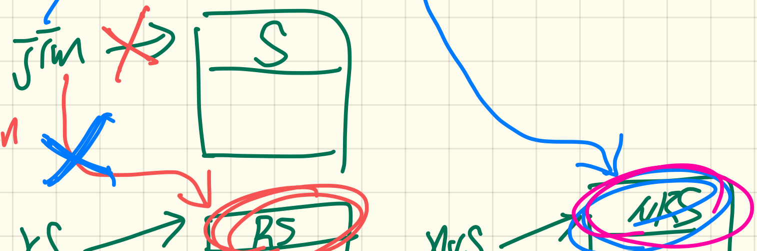
```
Student jim = new Student(...);
ResidentStudent rs = new ResidentStudent(...);
NonResidentStudent nrs = new NonResidentStudent(...);
```

## Example 1:

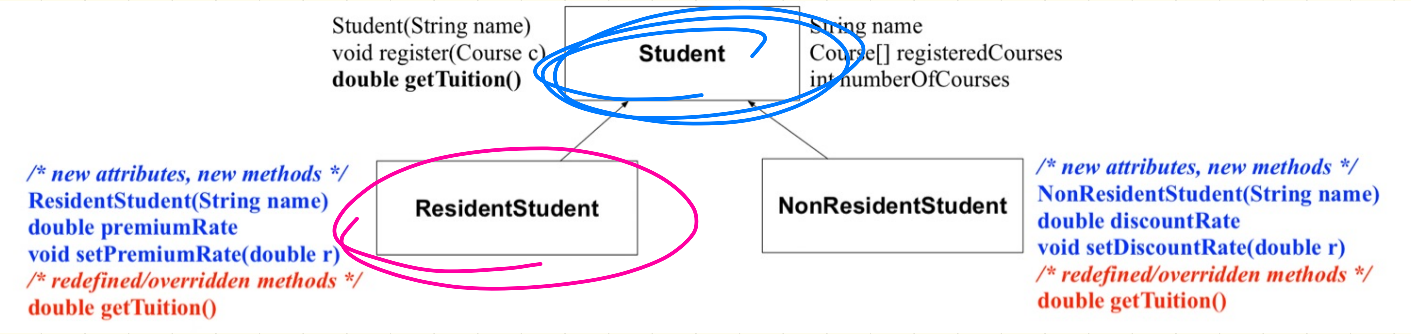
```
jim = rs;                      ①
println(jim.getTuition());     ②
jim = nrs;                     ③
println(jim.getTuition());     ④
```

DT of jim ⇓ RS call that version

DT of jim ⇓ NRS ∴ call that version



# Change of Dynamic Type (2.2)


Student(String name)
void register(Course c)
double getTuition()
Student
String name
Course[] registeredCourses
int numberOfCourses
/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()
ResidentStudent
NonResidentStudent
/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()

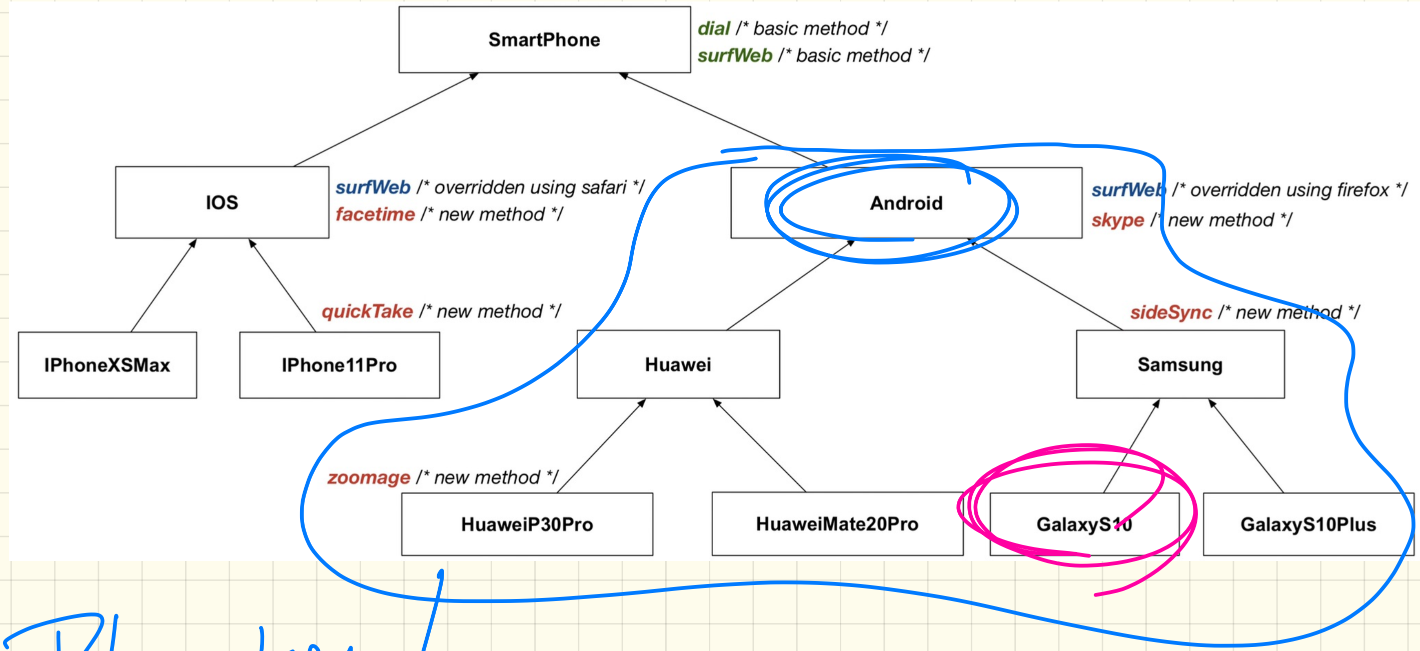
**Given:**

```
Student jim = new Student(...);
ResidentStudent rs = new ResidentStudent(...);
NonResidentStudent nrs = new NonResidentStudent(...);
```

**Example 2:**

```
rs = jim;  X
println(rs.getTuition());
nrs = jim;
println(nrs.getTuition());
```

cannot be executed ∵ the previous line does not compile.


SmartPhone
dial /* basic method */
surfWeb /* basic method */
IOS
surfWeb /* overridden using safari */
facetime /* new method */
Android
surfWeb /* overridden using firefox */
skype /* new method */
IPhoneXSMax
IPhone11Pro
quickTake /* new method */
Huawei
Samsung
sideSync /* new method */
zoomage /* new method */
HuaweiP30Pro
HuaweiMate20Pro
GalaxyS10
GalaxyS10Plus

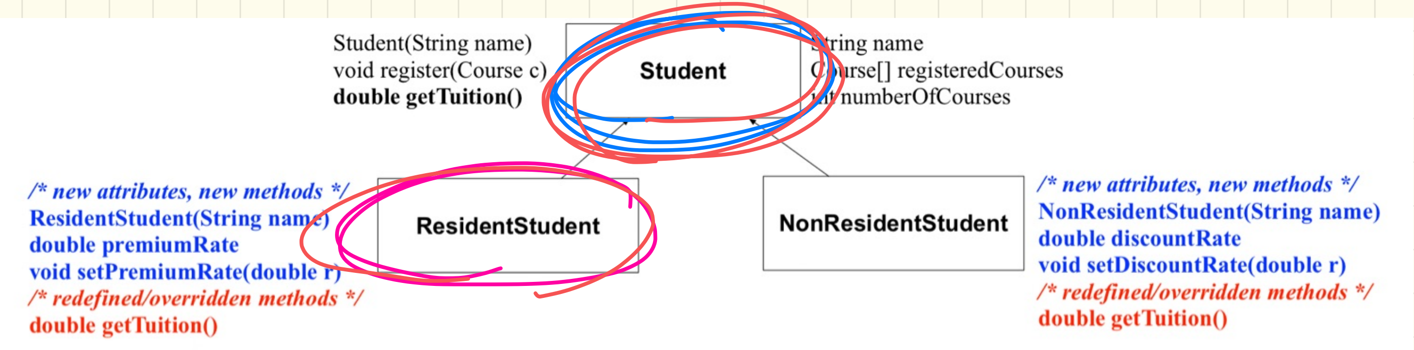
Polymorphism

Android myPhone = ? ;

new GS10();

all valid types of objects for substituting Android.

# Type Cast: Motivation

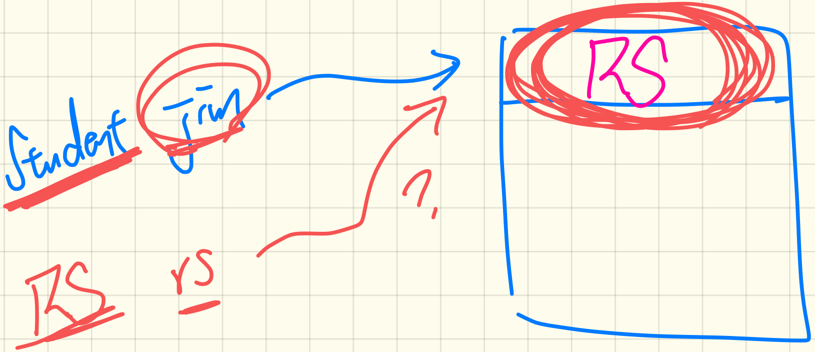


Student(String name)
void register(Course c)
double getTuition()
Student
String name
Course[] registeredCourses
int numberOfCourses
/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()
ResidentStudent
NonResidentStudent
/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()

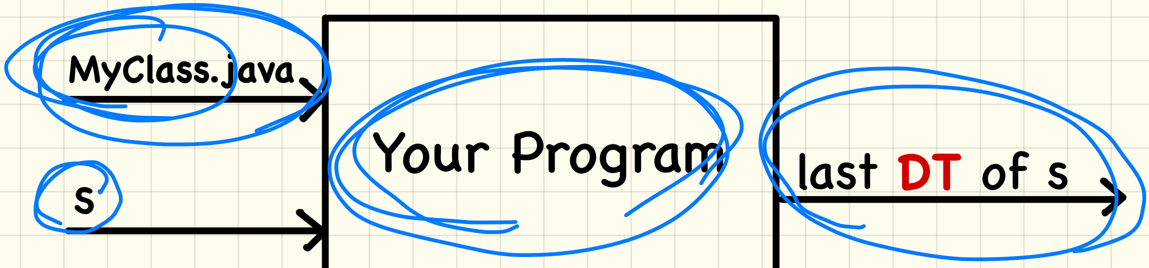
```
Student jim = new ResidentStudent("J. Davis");
ResidentStudent rs = jim;
rs.setPremiumRate(1.5);
```

?

dynamically points to RS



# An A+ Challenge: Inferring the DT of a Variable



RS

undecidable

```
class MyClass {
main (...)
   Student s = ...;
   ...
   s = new ResidentStudent(...);
 }
}
```

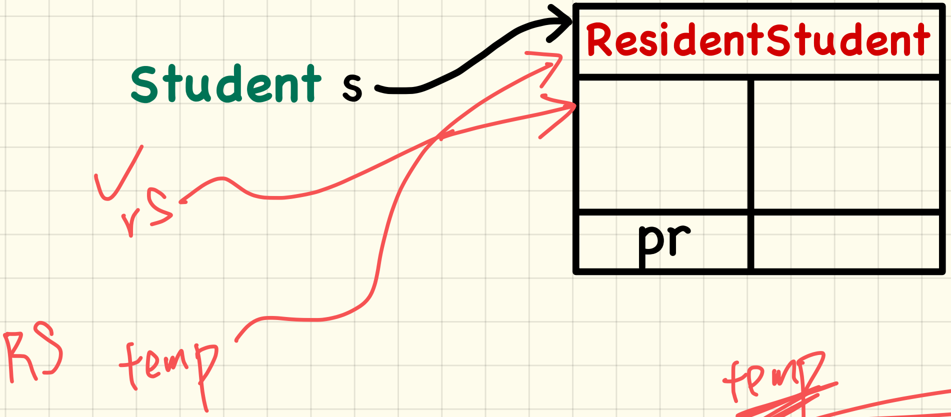
B

while ( ~~true~~ ) {
 ...
}

i =

# Anatomy of a Type Cast (1)

```
Student s = new ResidentStudent("Jim");
```

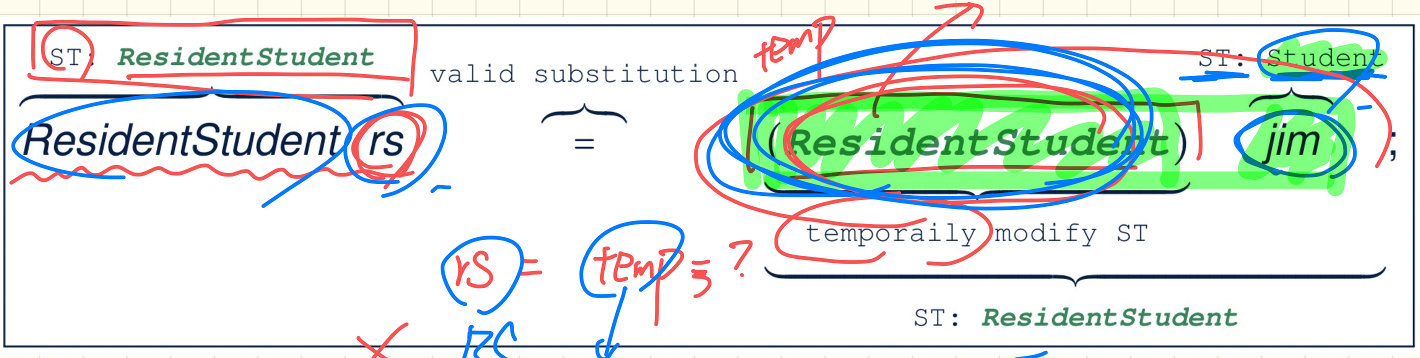


temp

```
ResidentStudent rs = (ResidentStudent) s;
```

# Anatomy of a Type Cast (2)

( ——— )
cast



ST: ResidentStudent

valid substitution

ST: Student

```
ResidentStudent rs = (ResidentStudent) jim;
```

temporaily modify ST

ST: ResidentStudent

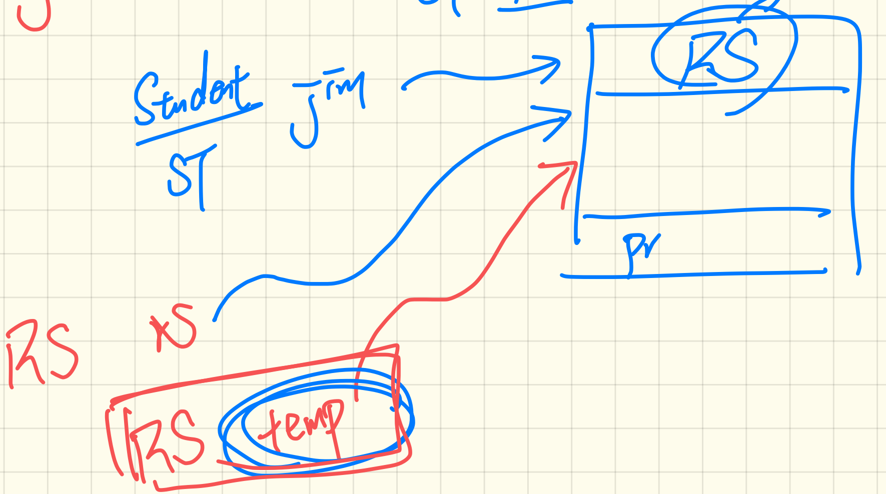
temp

rs = temp = ?

rs = jim; ✗ RS

ST: RS

DT



During a cast:

1. No new object created
2. ST of jim not modified
3. a temp. alias of ST RS is created

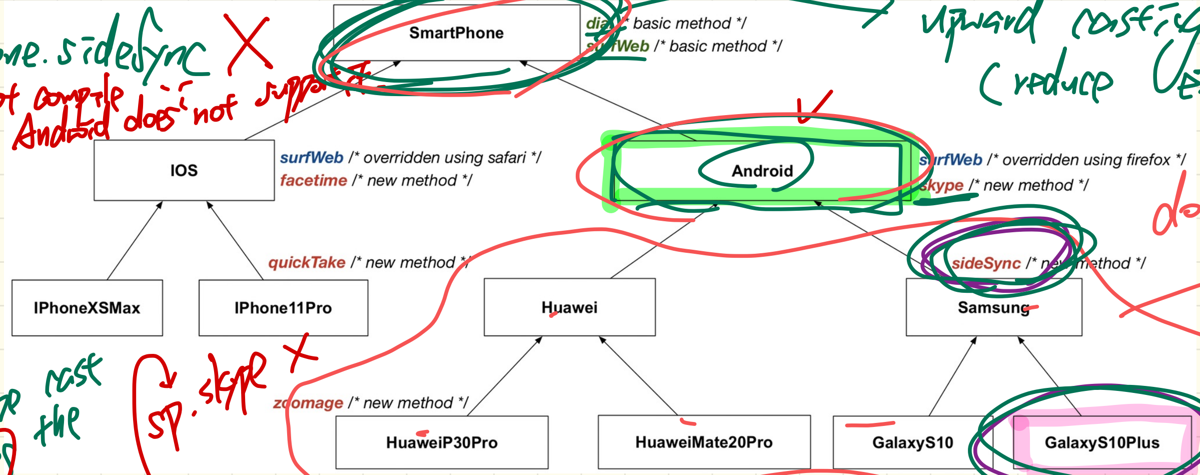
# Type Cast

↳ change the expectation

1. does the cast compile?

2. if the cast compiles, does it cause an Exception (ClassCastExc.)

# Compilable Type Casts: Upwards vs. Downwards



SmartPhone

dial /* basic method */
surfWeb /* basic method */

IOS

surfWeb /* overridden using safari */
facetime /* new method */

Android

surfWeb /* overridden using firefox */
skype /* new method */

quickTake /* new method */

sideSync /* new method */

IPhoneXSMax

IPhone11Pro

Huawei

Samsung

zoomage /* new method */

HuaweiP30Pro

HuaweiMate20Pro

GalaxyS10

GalaxyS10Plus

myPhone.sideSync ✗
not compile
ST Android does not support

upward casting.
(reduce expectations)

downward casting
(widen expect.)

sp.skype ✗

A type cast changes the compiler's expectations

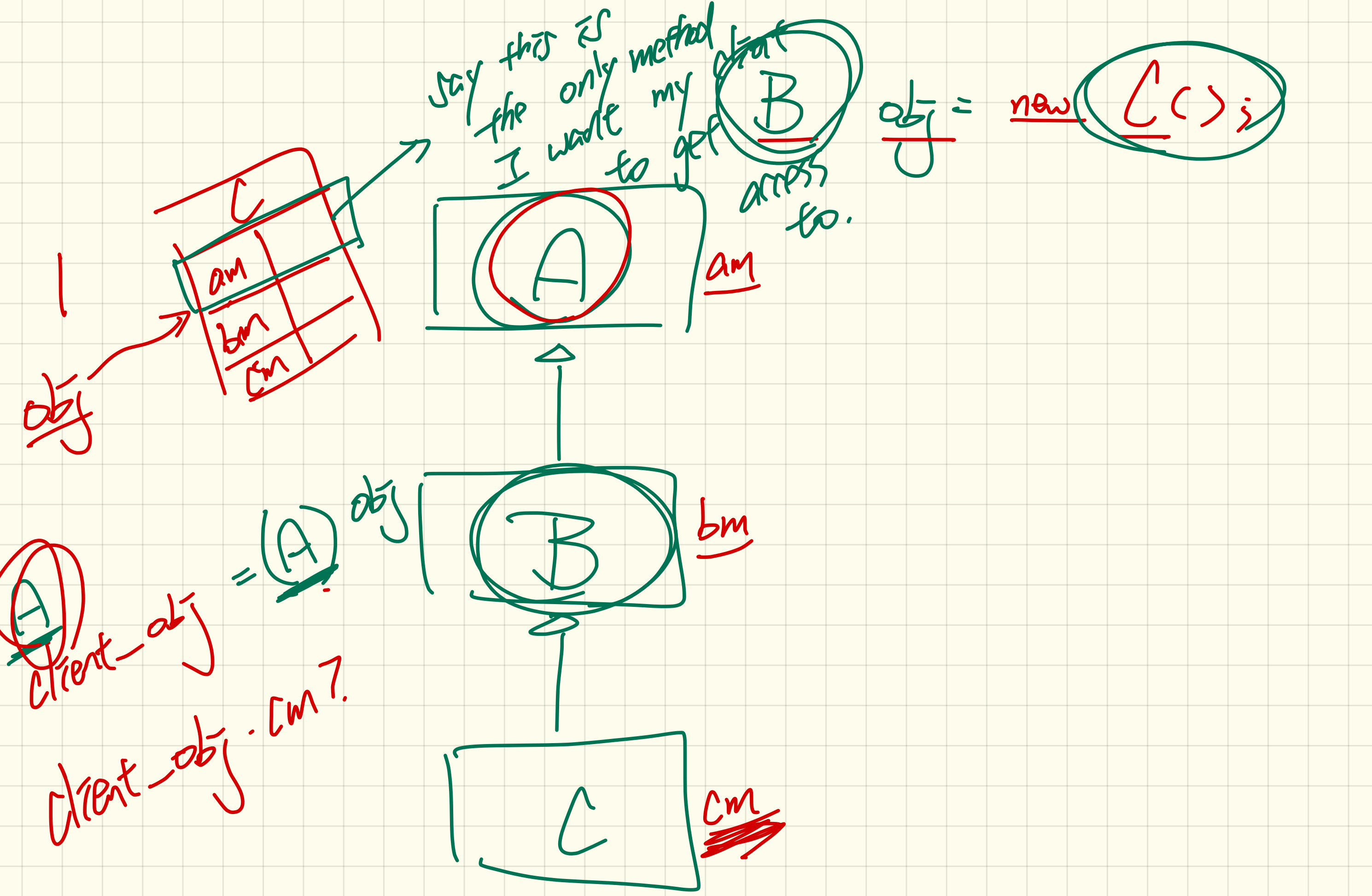
expectations?
(ST).

## Expectations

```
Android myPhone = new GalaxyS10Plus();
SmartPhone sp = (SmartPhone) myPhone;
GalaxyS10Plus ga = (GalaxyS10Plus) myPhone;
```

upward cast

| | sp | myPhone | ga |
|---|---|---|---|
| dial | ✓ | ✓ | ✓ |
| surfWeb | ✓ | ✓ | ✓ |
| skype | ✗ | ✓ | ✓ |
| sideSync | ✗ | ✗ | ✓ |
| facetime | ✗ | ✗ | ✗ |
| quickTake | ✗ | ✗ | ✗ |
| zoomage | ✗ | ✗ | ✗ |



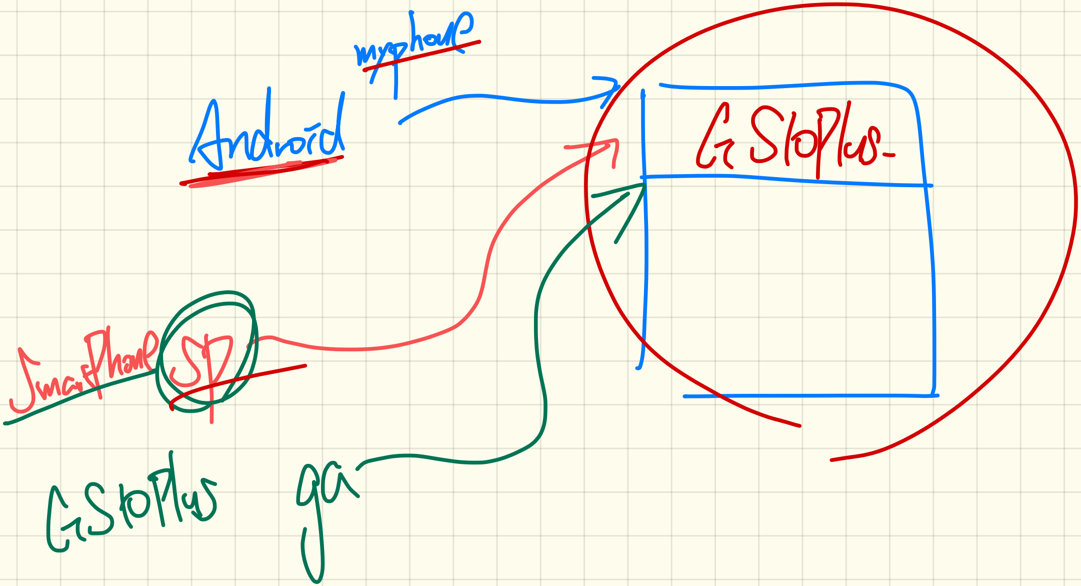
say this is the only method I want my B to get access to.

obj = new C();

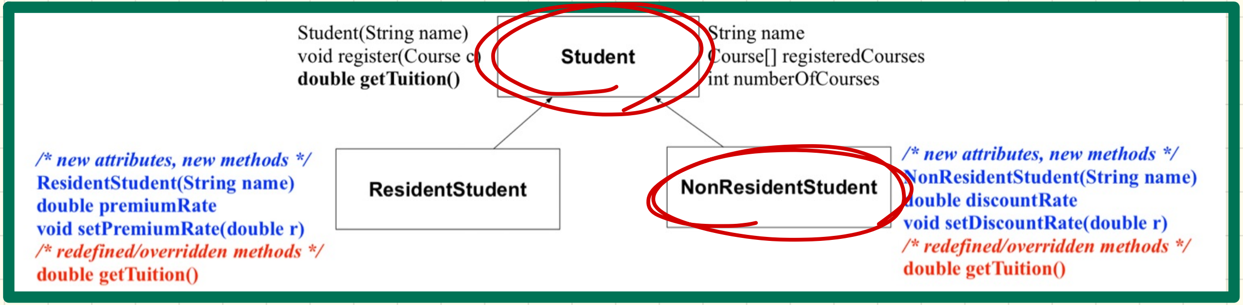
client_obj = (A) obj

client_obj.cm?

```java
Android myPhone = new GalaxyS10Plus();

SmartPhone sp = (SmartPhone) myPhone;

GalaxyS10Plus ga = (GalaxyS10Plus) myPhone;
```



# Compilable Type Cast May Fail at Runtime (1)


Student(String name)
void register(Course c)
double getTuition()
Student
String name
Course[] registeredCourses
int numberOfCourses
/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()
ResidentStudent
NonResidentStudent
/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()

```
Student jim = new NonResidentStudent("J. Davis");
ResidentStudent rs = (ResidentStudent) jim;
rs.setPremiumRate(1.5);
```

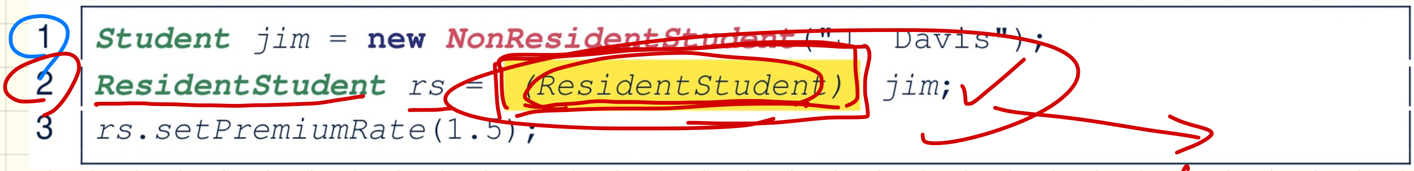
cast version of jim of ST: RS

ResidentStudent rs = jim; X

valid downward casting!

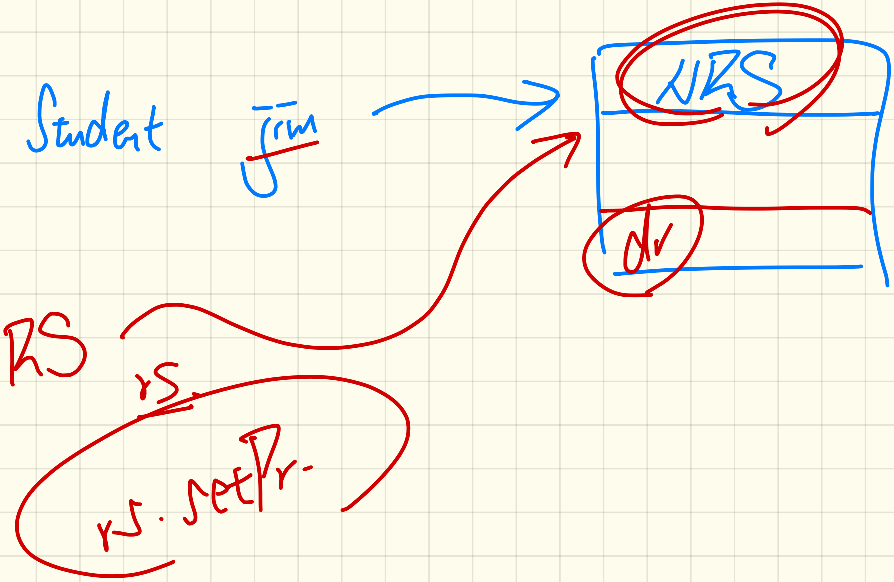
ST: Student

rs = cast version ✓

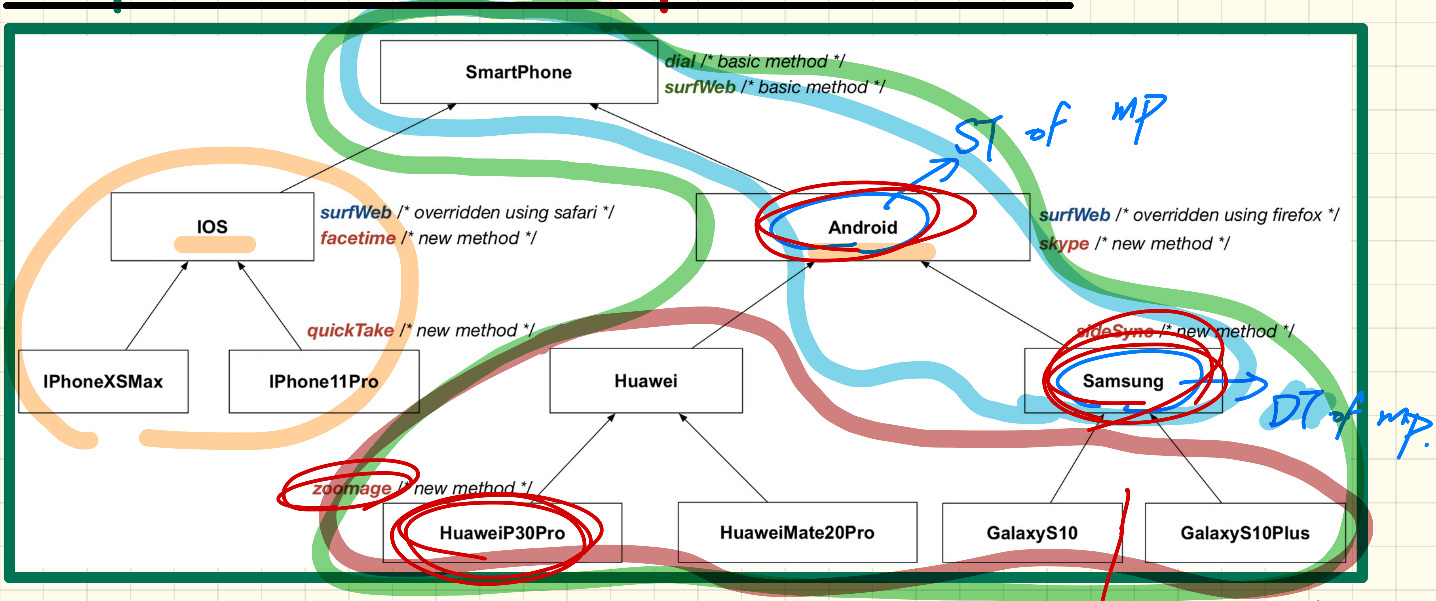


```
Student jim = new NonResidentStudent("J. Davis");
ResidentStudent rs = (ResidentStudent) jim;
rs.setPremiumRate(1.5);
```

class cast exception.



# Compilable Cast vs. Exception-Free Cast



```
Android myPhone = new Samsung();
```

downward casting

but DT of SP cannot support it.

Compilable Casts 🟩

Exception-Free Casts 🟦

Non-Compilable Casts 🟧

ClassCastException 🟥

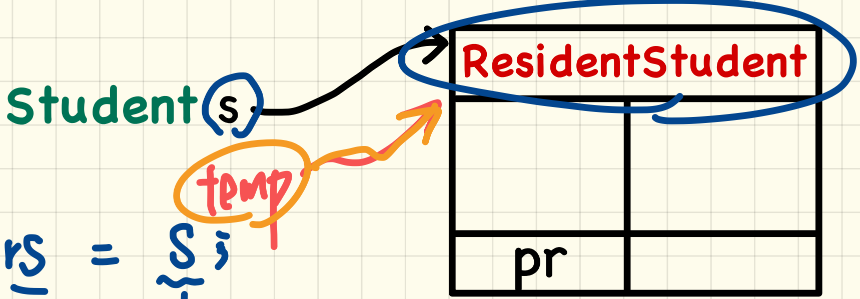
# LECTURE 19

## WEDNESDAY NOVEMBER 13

# Anatomy of a Type Cast

```
Student s = new ResidentStudent("Jim");
```

ST DT



cast version of 's'

temp

RS ≤ ST

rs = s;

ST: STU.

is this valid?

temp with ST RS

ST: Stu.

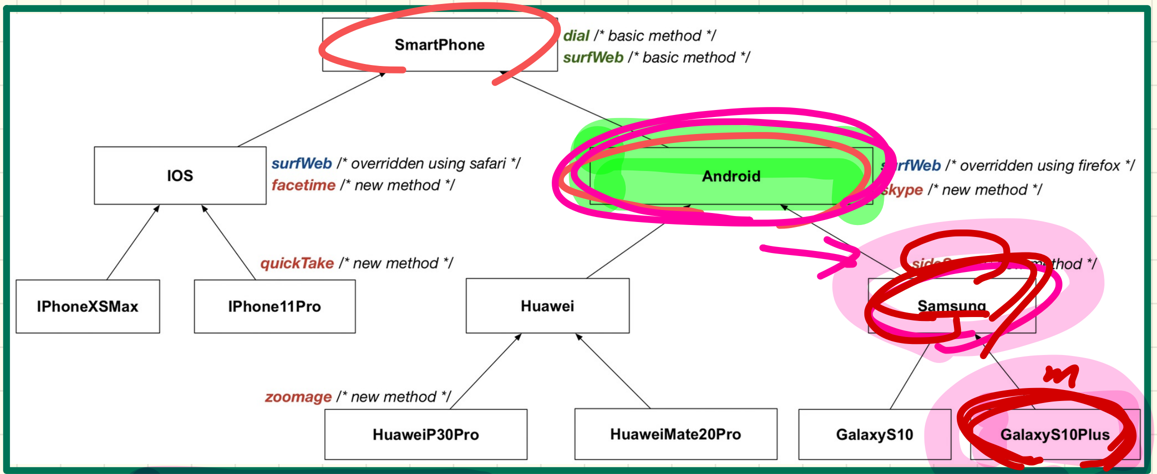
```
ResidentStudent rs = (ResidentStudent) s;
```

RS is a descendant of RS. ✓

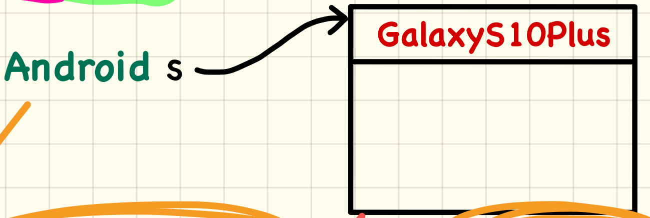
ST: RS

is a descendant of Stu.

down casting.


SmartPhone
dial /* basic method */
surfWeb /* basic method */
IOS
surfWeb /* overridden using safari */
facetime /* new method */
Android
surfWeb /* overridden using firefox */
skype /* new method */
IPhoneXSMax
IPhone11Pro
quickTake /* new method */
Huawei
Samsung
sideS... /* new method */
zoomage /* new method */
HuaweiP30Pro
HuaweiMate20Pro
GalaxyS10
GalaxyS10Plus
m

```
Android s = new GalaxyS10Plus();
```



## Substitutions of Type Cast

①

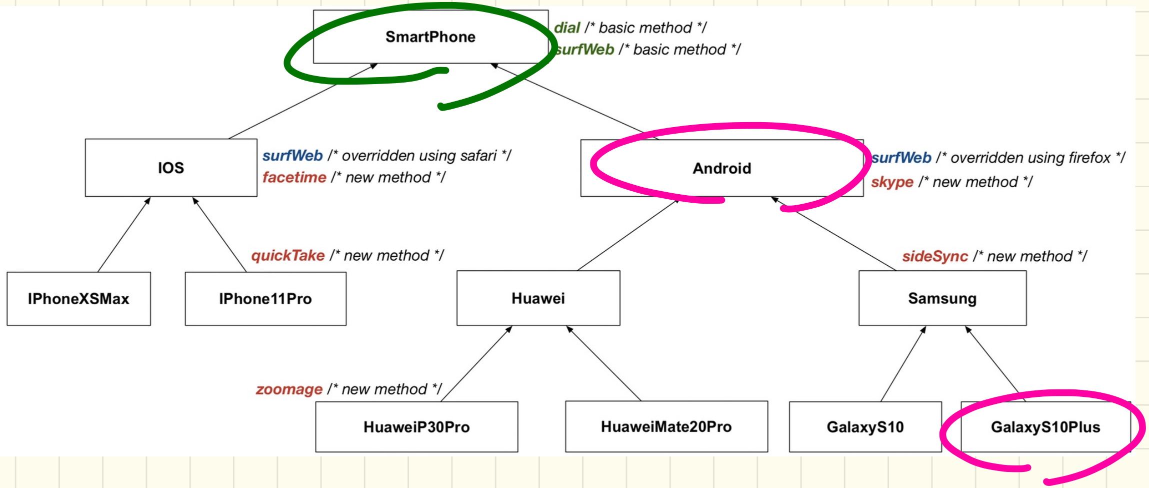
```
SmartPhone sp1 = (Samsung) s;
SmartPhone sp2 = (SmartPhone) s;
GalaxyS10Plus sp3 = (Samsung) s;
```

Down cast

ST: Samsung - ∵ Samsung ✓

ST: SmartPhone

ST: Samsung. ✗ ST not a desc. of GS10P


SmartPhone
dial /* basic method */
surfWeb /* basic method */
IOS
surfWeb /* overridden using safari */
facetime /* new method */
Android
surfWeb /* overridden using firefox */
skype /* new method */
IPhoneXSMax
IPhone11Pro
quickTake /* new method */
Huawei
Samsung
sideSync /* new method */
zoomage /* new method */
HuaweiP30Pro
HuaweiMate20Pro
GalaxyS10
GalaxyS10Plus

Android s = new Samsung(); ok: down

✓ Android sp4 = (Samsung) s;

SmartPhone sp5 = (GalaxyS10Plus) s;

DT GS10P.

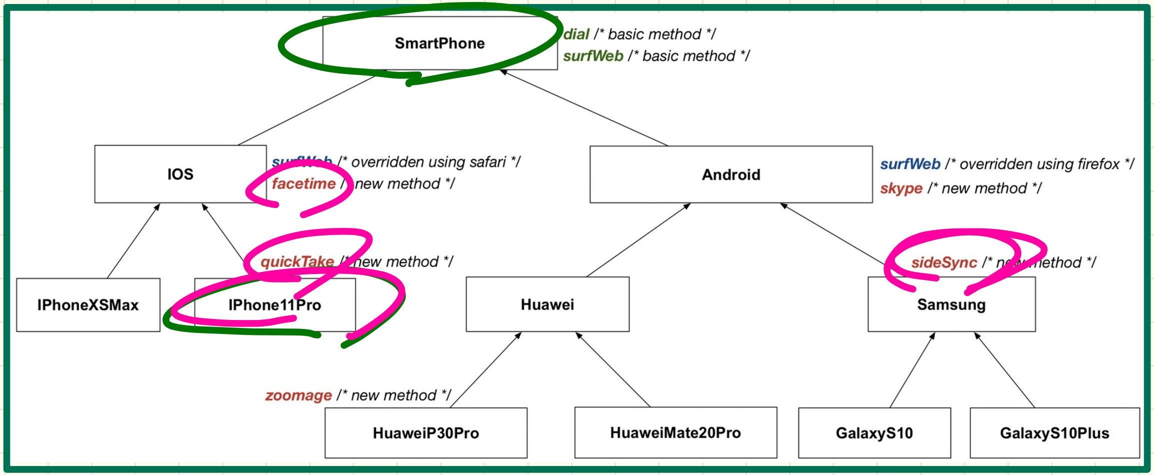
ST: SP.

✗ Samsung sp6 = sp5;

# Type Cast

# Named vs.

# Anonymous


SmartPhone
dial /* basic method */
surfWeb /* basic method */
IOS
surfWeb /* overridden using safari */
facetime /* new method */
Android
surfWeb /* overridden using firefox */
skype /* new method */
IPhoneXSMax
quickTake /* new method */
IPhone11Pro
Huawei
sideSync /* new method */
Samsung
zoomage /* new method */
HuaweiP30Pro
HuaweiMate20Pro
GalaxyS10
GalaxyS10Plus

**Named Cast**: Use intermediate variable to store the cast result.

```
SmartPhone aPhone = new IPhone11Pro();
IOS forHeeyeon = (IPhone11Pro) aPhone;
forHeeyeon.facetime();
```

down

ST: IPhone11Pro

**Anonymous Cast**: Use the cast result directly.

```
SmartPhone aPhone = new IPhone11Pro();
((IPhone11Pro) aPhone).facetime();
```

- sideSync X
- facetime
- quickTake ✓

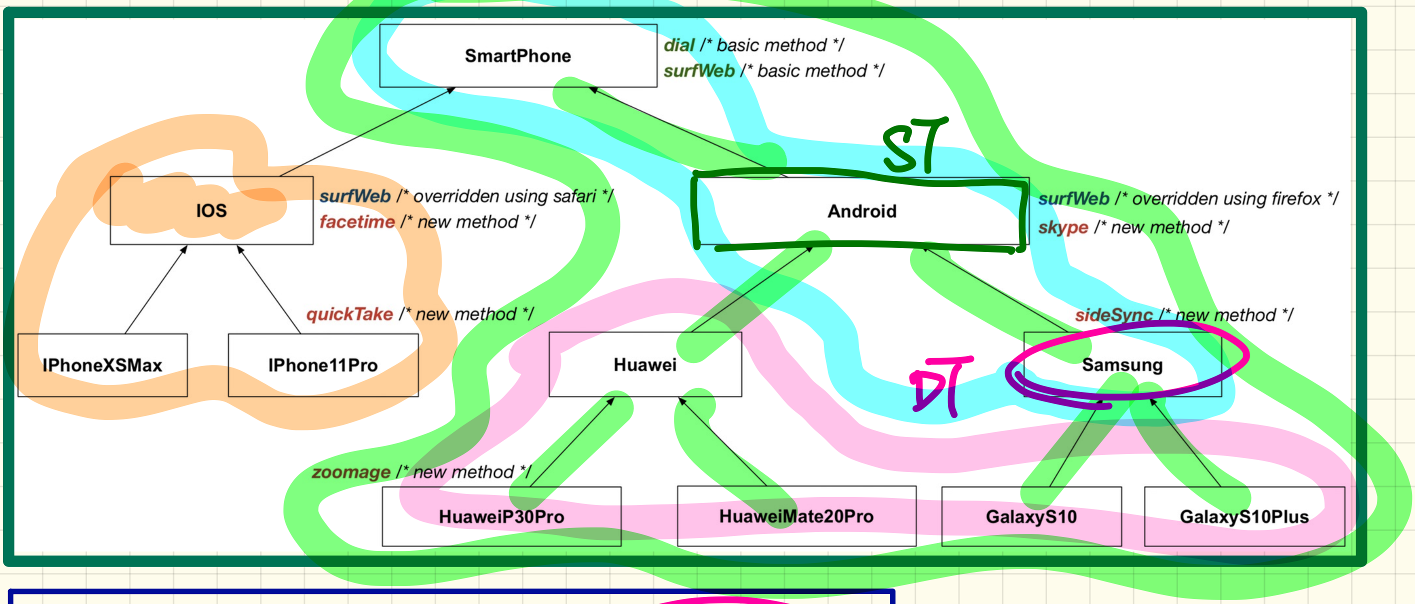
ST: IP11Pro

((IP11Pro) aPhone).facet

```
1 SmartPhone aPhone = new IPhone11Pro();
2 (IPhone11Pro) aPhone.facetime(); X
```

ST

# Compilable Cast vs. Exception-Free Cast



SmartPhone — dial /* basic method */, surfWeb /* basic method */

IOS — surfWeb /* overridden using safari */, facetime /* new method */

IPhoneXSMax

IPhone11Pro — quickTake /* new method */

Android — surfWeb /* overridden using firefox */, skype /* new method */

Huawei

HuaweiP30Pro — zoomage /* new method */

HuaweiMate20Pro

Samsung — sideSync /* new method */

GalaxyS10

GalaxyS10Plus

```
Android myPhone = new Samsung();
```

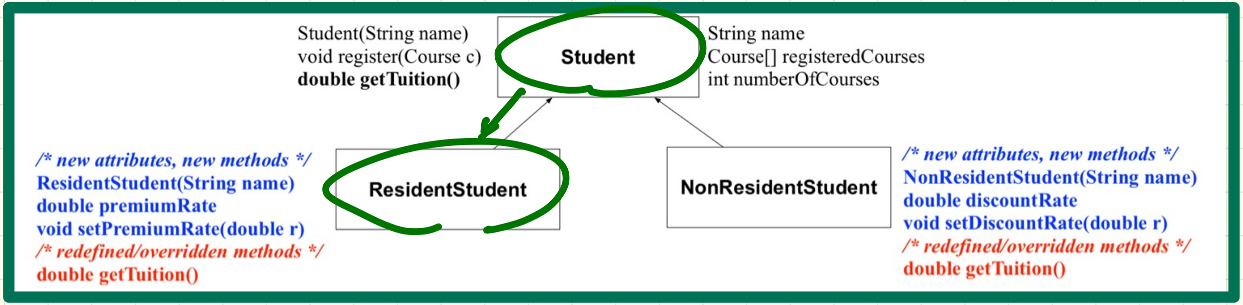
Compilable Casts 🟩

Non-Compilable Casts 🟧

Exception-Free Casts 🟦

ClassCastException 🟥

# Compilable Type Cast May Fail at Runtime (1)

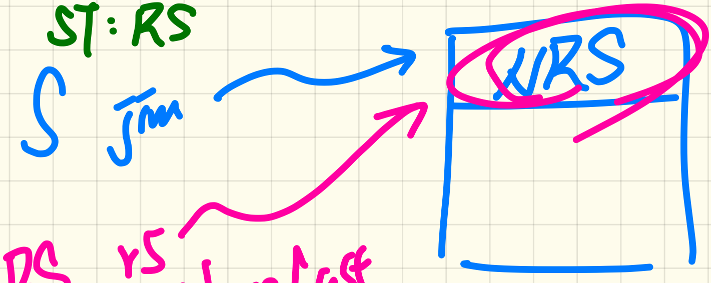


Student(String name)
void register(Course c)
double getTuition()
Student
String name
Course[] registeredCourses
int numberOfCourses
/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()
ResidentStudent
NonResidentStudent
/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()

```
Student jim = new NonResidentStudent("J. Davis");
ResidentStudent rs = (ResidentStudent) jim;
rs.setPremiumRate(1.5);
```

compiles; down

ST: RS



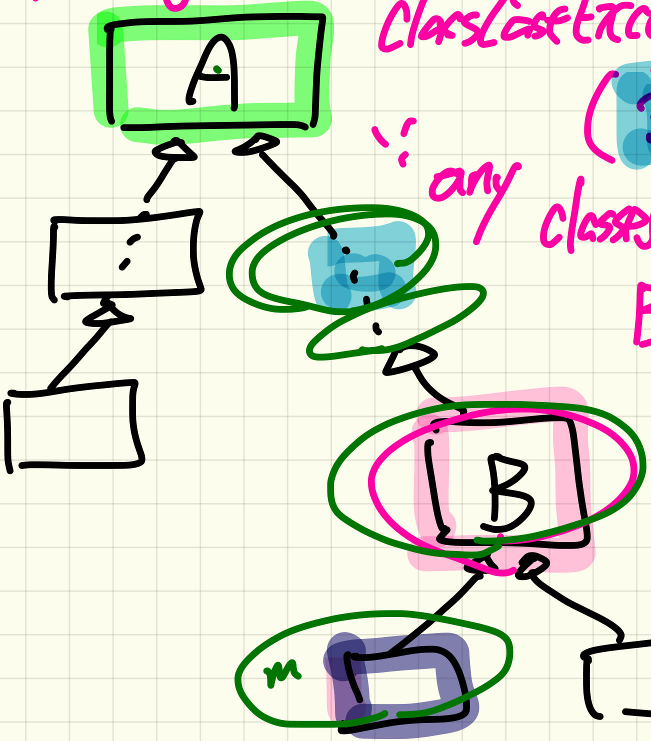
ClassCast Excep.

Executing this at runtime, it will be found that the DT of rs (NRS) cannot fulfill the expectations on ST of rs (RS)

- Down cast always compiles.
- Down cast beyond the DT of obj will cause ClassCastException

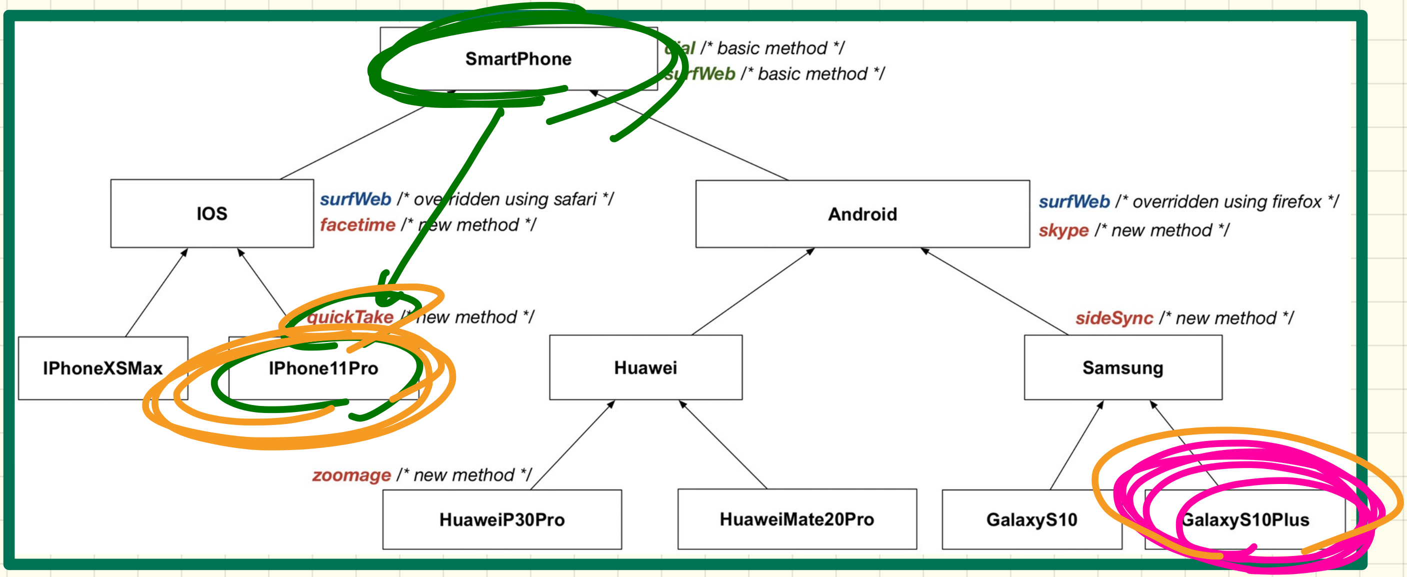
A obj = new B();

(??) obj



∵ any classes strictly lower than B have expectations that B can't support.

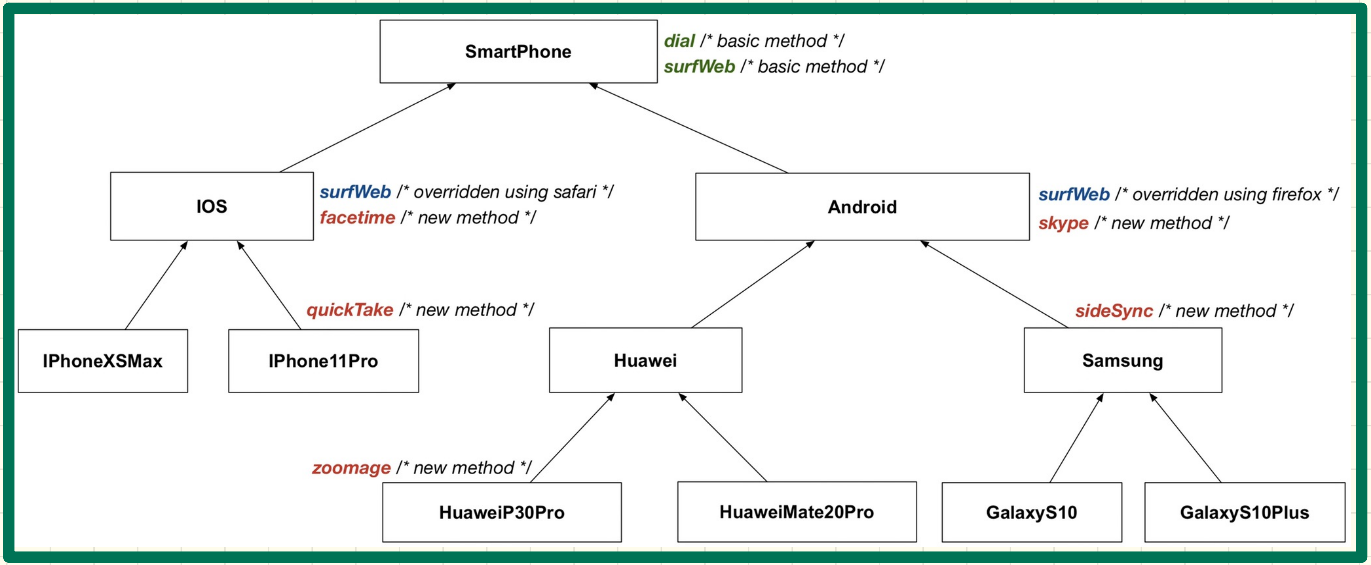
# Compilable Type Cast May Fail at Runtime (2)


SmartPhone
dial /* basic method */
surfWeb /* basic method */
IOS
surfWeb /* overridden using safari */
facetime /* new method */
Android
surfWeb /* overridden using firefox */
skype /* new method */
IPhoneXSMax
IPhone11Pro
quickTake /* new method */
Huawei
Samsung
sideSync /* new method */
zoomage /* new method */
HuaweiP30Pro
HuaweiMate20Pro
GalaxyS10
GalaxyS10Plus

```
SmartPhone aPhone = new GalaxyS10Plus();
IPhone11Pro forHeeyeon = (IPhone11Pro) aPhone;
forHeeyeon.quickTake();
```

✓ ✓ valid

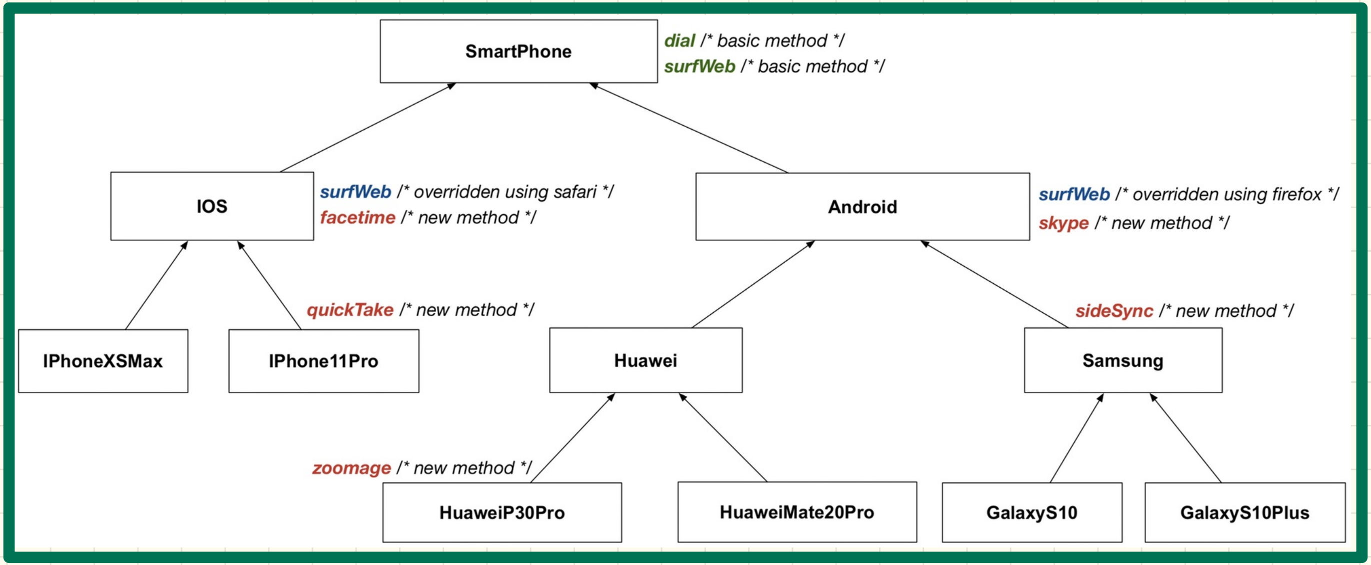
# Exercise: Compilable Type Cast? Fail at Runtime? (1)


SmartPhone
dial /* basic method */
surfWeb /* basic method */
IOS
surfWeb /* overridden using safari */
facetime /* new method */
Android
surfWeb /* overridden using firefox */
skype /* new method */
quickTake /* new method */
sideSync /* new method */
IPhoneXSMax
IPhone11Pro
Huawei
Samsung
zoomage /* new method */
HuaweiP30Pro
HuaweiMate20Pro
GalaxyS10
GalaxyS10Plus

```
SmartPhone myPhone = new Samsung();
/* ST of myPhone is SmartPhone; DT of myPhone is Samsung */
GalaxyS10Plus ga = (GalaxyS10Plus) myPhone;
```

**Compilable?** **ClassCastException at runtime?**

# Exercise: Compilable Type Cast? Fail at Runtime? (2)


SmartPhone
dial /* basic method */
surfWeb /* basic method */
IOS
surfWeb /* overridden using safari */
facetime /* new method */
Android
surfWeb /* overridden using firefox */
skype /* new method */
IPhoneXSMax
IPhone11Pro
quickTake /* new method */
Huawei
Samsung
sideSync /* new method */
zoomage /* new method */
HuaweiP30Pro
HuaweiMate20Pro
GalaxyS10
GalaxyS10Plus

```
SmartPhone myPhone = new Samsung();
/* ST of myPhone is SmartPhone; DT of myPhone is Samsung */
IPhone11Pro ip = (IPhone11Pro) myPhone;
```

**Compilable?** **ClassCastException at runtime?**

# Compilable Cast vs. Exception-Free Cast: Exercise

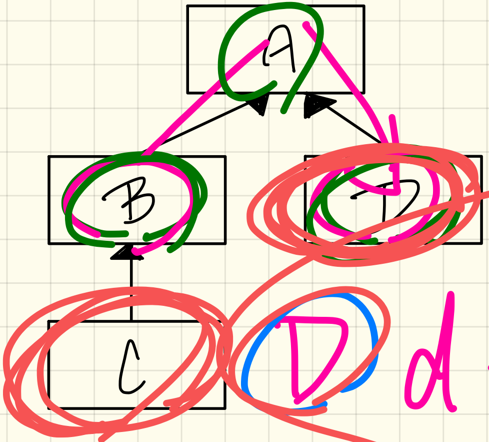
```
class A { }
class B extends A { }
class C extends B { }
class D extends A { }
```

```
B b = new C(); ✓ → ST: B  DT: C
D d = (D) b;  ×
```

✓

ClassCastException

∵ DT of b is C, which cannot fulfill expectations of D;

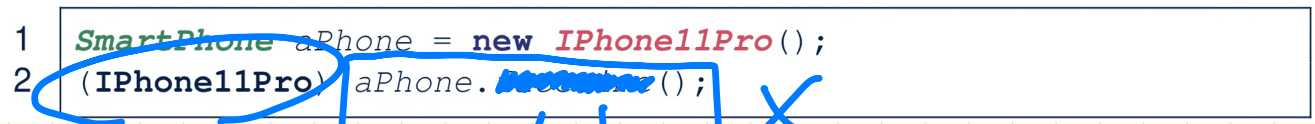


B b ⟶ C

$$D\ d = \underbrace{(D)\ \underbrace{((A)\ b)}_{\text{Up}}}_{\text{Down}}$$

ST: A

D



```
SmartPhone aPhone = new IPhone11Pro();
(IPhone11Pro) aPhone.[redacted]();
```

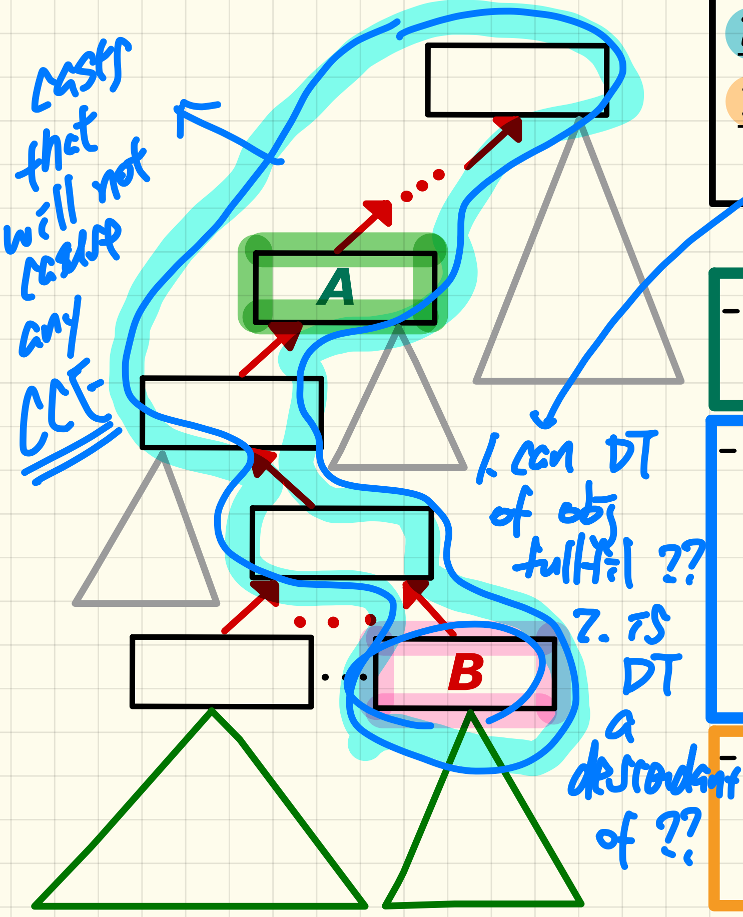
dial.

X

void dial

# The instanceof Operator

```
1  A obj = new B();
2  if (obj instanceof ??) {
3    ?? obj2 = (??) obj;
   }
```

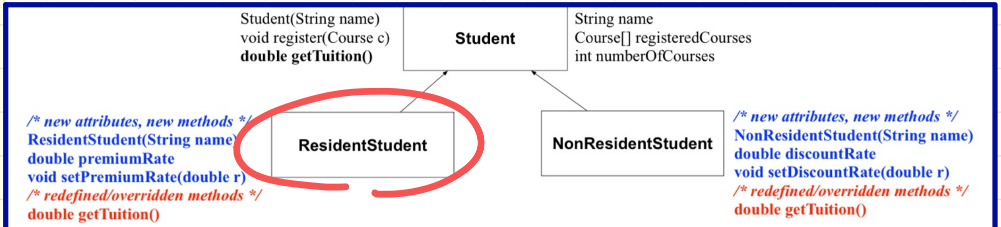

A
B
casts that will not cause any CCE
1. can DT of obj fulfill ??
2. Is DT a descendant of ??

- L1 compiles if B can fulfill expectations of A.

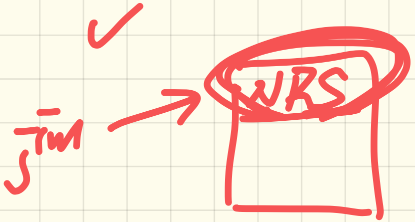
- L3:
  - Compiles if Up or Down cast w.r.t. A.
  - ClassCastException if B cannot fulfill expectations on ??.

- L2:
  - Evaluates to true if B can fulfill expectations on ??.

# Checking Dynamic Types at Runtime

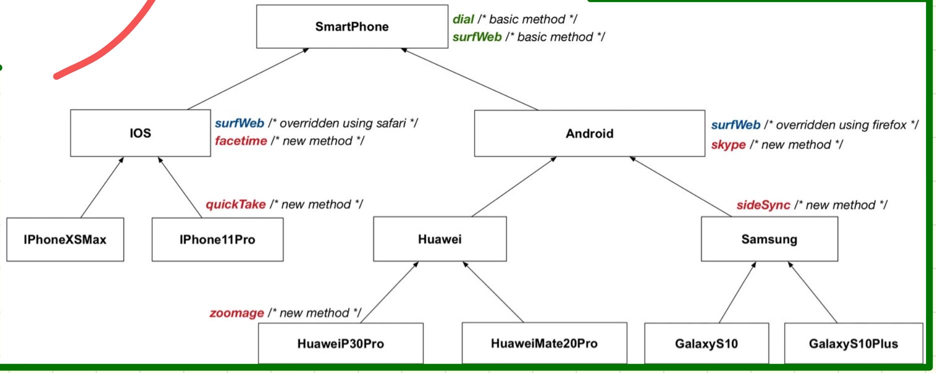


Student(String name)
void register(Course c)
double getTuition()
Student
String name
Course[] registeredCourses
int numberOfCourses
/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()
ResidentStudent
NonResidentStudent
/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()



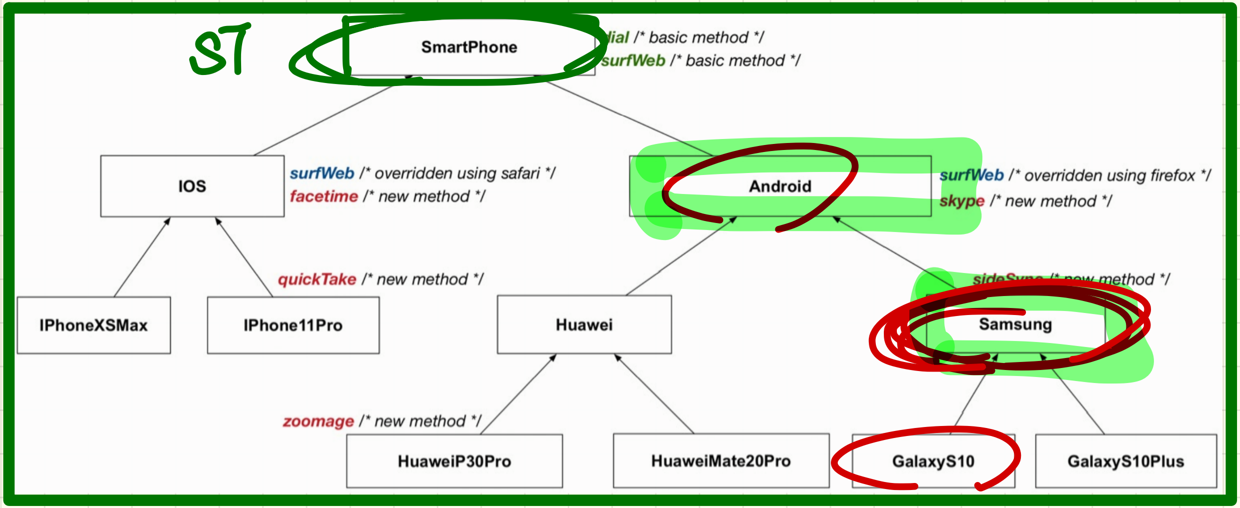
```
Student jim = new NonResidentStudent("J. Davis");
if (jim instanceof ResidentStudent ) {
  ResidentStudent rs = ( ResidentStudent ) jim;
  rs.setPremiumRate(1.5);
}
```

```
SmartPhone aPhone = new GalaxyS10Plus();
if (aPhone instanceof IPhone11Pro) {
  IOS forHeeyeon = (IPhone11Pro) aPhone;
  forHeeyeon.facetime();
}
```



SmartPhone
dial /* basic method */
surfWeb /* basic method */
IOS
surfWeb /* overridden using safari */
facetime /* new method */
Android
surfWeb /* overridden using firefox */
skype /* new method */
quickTake /* new method */
sideSync /* new method */
IPhoneXSMax
IPhone11Pro
Huawei
Samsung
zoomage /* new method */
HuaweiP30Pro
HuaweiMate20Pro
GalaxyS10
GalaxyS10Plus

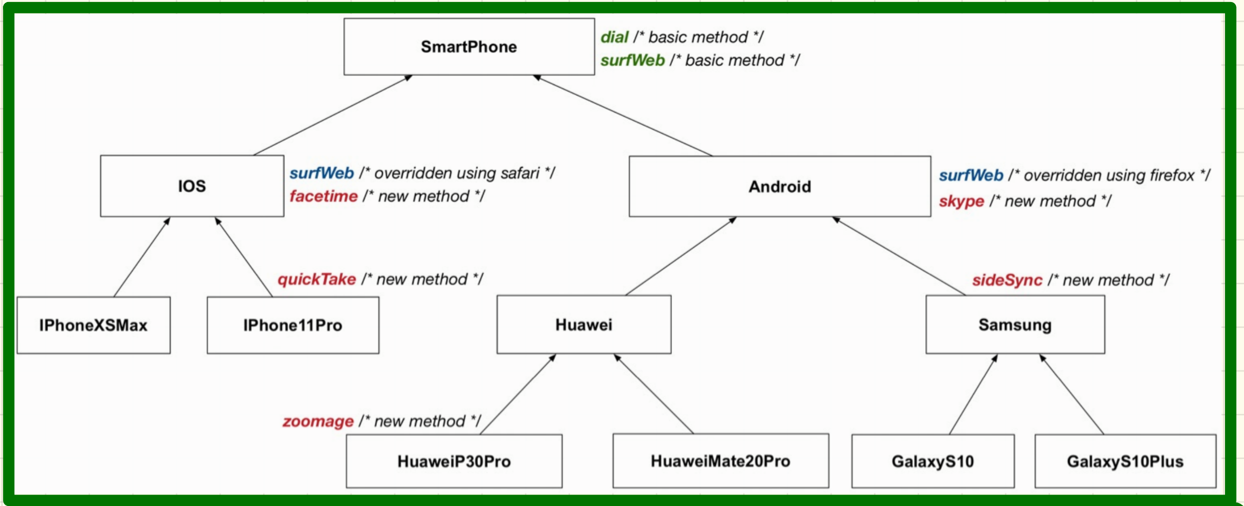
# Use of the instanceof Operator


ST
SmartPhone
dial /* basic method */
surfWeb /* basic method */
IOS
surfWeb /* overridden using safari */
facetime /* new method */
Android
surfWeb /* overridden using firefox */
skype /* new method */
quickTake /* new method */
sideSync /* new method */
IPhoneXSMax
IPhone11Pro
Huawei
Samsung
zoomage /* new method */
HuaweiP30Pro
HuaweiMate20Pro
GalaxyS10
GalaxyS10Plus

```
SmartPhone myPhone = new Samsung();
println(myPhone instanceof Android);
/* true ∵ Samsung is a descendant of Android */}
println(myPhone instanceof Samsung);
/* true ∵ Samsung is a descendant of Samsung */}
println(myPhone instanceof GalaxyS10);
/* false ∵ Samsung is not a descendant of GalaxyS10 */
println(myPhone instanceof IOS);
/* false ∵ Samsung is not a descendant of IOS */
println(myPhone instanceof IPhone11Pro);
/* [redacted] Samsung is not a descendant of IPhone11Pro */
```

myPhone instanceof ??
evaluates to true if
Samsung can
fulfill expectations on ??.

# Safe Cast via Use of the instanceof Operator


SmartPhone
dial /* basic method */
surfWeb /* basic method */
IOS
surfWeb /* overridden using safari */
facetime /* new method */
Android
surfWeb /* overridden using firefox */
skype /* new method */
quickTake /* new method */
sideSync /* new method */
IPhoneXSMax
IPhone11Pro
Huawei
Samsung
zoomage /* new method */
HuaweiP30Pro
HuaweiMate20Pro
GalaxyS10
GalaxyS10Plus

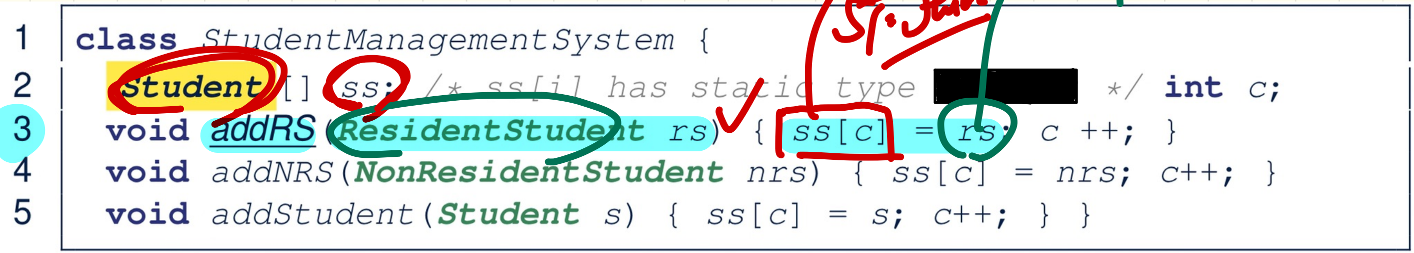
```
SmartPhone myPhone = new Samsung();
/* ST of myPhone is SmartPhone; DT of myPhone is Samsung */
if(myPhone instanceof Samsung) {
  Samsung samsung = (Samsung) myPhone;
}
if(myPhone instanceof GalaxyS10Plus) {
  GalaxyS10Plus galaxy = (GalaxyS10Plus) myPhone;
}
if(myphone instanceof HTC) {
  HTC htc = (HTC) myPhone;
}
```

before I can cast myPhone into Samsung,

run a check to see if the DT of myP. can fullfil all expe. of Samsung

myPhone instanceof ?? evaluates to true if Samsung can fulfill expectations on ??.

# Polymorphic Arguments (1)



```
class StudentManagementSystem {
  Student[] ss; /* ss[i] has static type [redacted] */ int c;
  void addRS(ResidentStudent rs) { ss[c] = rs; c ++; }
  void addNRS(NonResidentStudent nrs) { ss[c] = nrs; c++; }
  void addStudent(Student s) { ss[c] = s; c++; } }
```

ST: Stude

ST: RS

✓

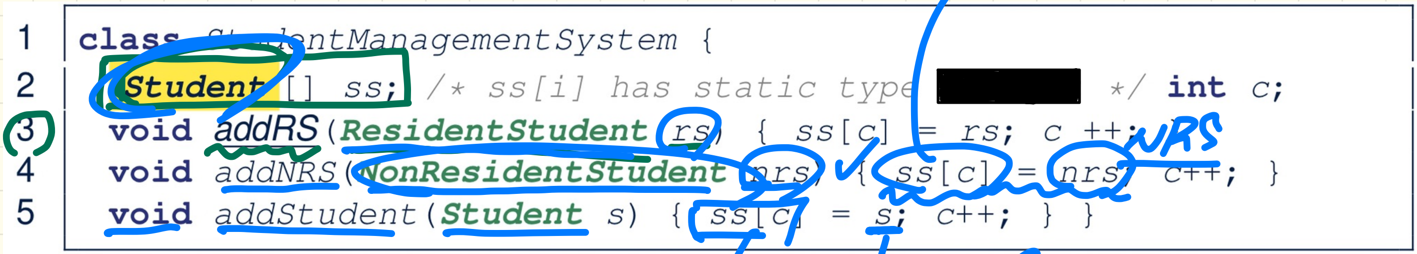
Q. Static type of ss[0], ss[1], ..., ss[ss.length - 1]?

Q. In method addRS, does ss[c] = rs compile?

# LECTURE 20

FRIDAY NOVEMBER 15

# Polymorphic Arguments (1)



```
class StudentManagementSystem {
  Student[] ss; /* ss[i] has static type [redacted] */ int c;
  void addRS(ResidentStudent rs) { ss[c] = rs; c ++; }
  void addNRS(NonResidentStudent nrs) { ss[c] = nrs; c++; }
  void addStudent(Student s) { ss[c] = s; c++; } }
```

Student

NRS ✓

Student ST: S.

Q. Static type of ss[0], ss[1], ..., ss[ss.length - 1]?

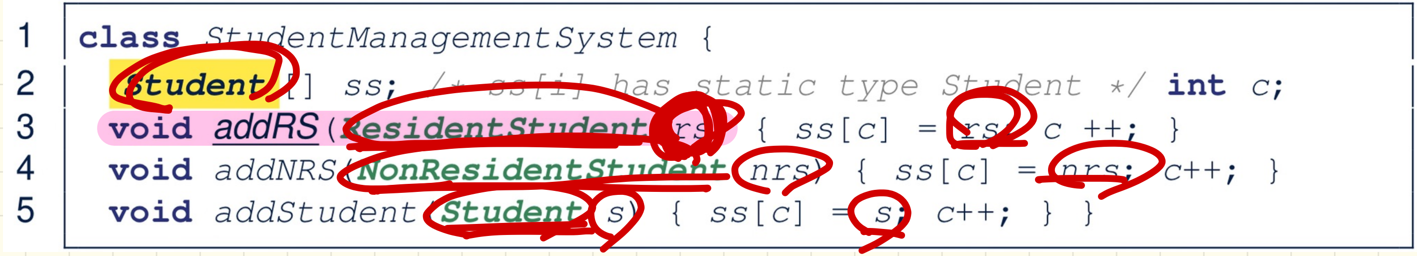
Student

Q. In method addRS, does ss[c] = rs compile?

✓

ss[c] = rs;

Student ST: RS



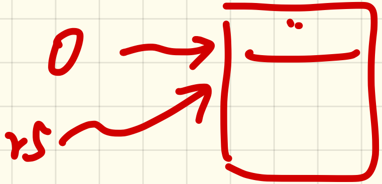
```
class StudentManagementSystem {
  Student[] ss; /* ss[i] has static type Student */ int c;
  void addRS(ResidentStudent rs) { ss[c] = rs; c ++; }
  void addNRS(NonResidentStudent nrs) { ss[c] = nrs; c++; }
  void addStudent(Student s) { ss[c] = s; c++; } }
```

SMS sms = new SMS();

:

sms.addRS(o);

rs = o;



# Polymorphic Arguments (2)

parameter: ST RS

rs = s1.

```
class StudentManagementSystem {
  Student [] ss; /* ss[i] has static type Student */ int c;
  void addRS(ResidentStudent rs) { ss[c] = rs; c ++; }
  void addNRS(NonResidentStudent nrs) { ss[c] = nrs; c++; }
  void addStudent(Student s) { ss[c] = s; c++; } }
```

```
Student s1 = new Student();
Student s2 = new ResidentStudent();
Student s3 = new NonResidentStudent();
ResidentStudent rs = new ResidentStudent();
NonResidentStudent nrs = new NonResidentStudent();
StudentManagementSystem sms = new StudentManagementSystem();
sms.addRS(s1);          ×
sms.addRS(s2);          ×
sms.addRS(s3);          ×
sms.addRS(rs);          ✓
sms.addRS(nrs);         ×
sms.addStudent(s1);     ✓
sms.addStudent(s2);     ✓
sms.addStudent(s3);     ✓
sms.addStudent(rs);     ✓
sms.addStudent(nrs);    ✓
```

argument: ST Student

not compile ∵ ST of s1 (argument) not a descendant of ST of rs (param).

ST: descendants of ST of param of addStud.

# Casting Arguments

addRS(RS rs)

sms.addRS((ResidentStudent) s) compiles?

valid down cast with ST: RS

```
Student s = new Student("Stella");
/* s' ST: Student; s' DT: Student */
StudentManagementSystem sms = new StudentManagementSystem();
sms.addRS(s);
```

① (ResidentStudent) s ✓

**ClassCastException?**

YES ∵ DT Student can't fulfill RS exp.

```
Student s = new NonResidentStudent("Nancy");
/* s' ST: Student; s' DT: NonResidentStudent */
StudentManagementSystem sms = new StudentManagementSystem();
sms.addRS(s);
```

② (RS) S ✓ can't fulfill DT

**ClassCastException?**

YES.

```
Student s = new ResidentStudent("Rachael");
/* s' ST: Student; s' DT: ResidentStudent */
StudentManagementSystem sms = new StudentManagementSystem();
sms.addRS(s);
```

③ (RS) S ✓ can fulfill

**ClassCastException?**

No

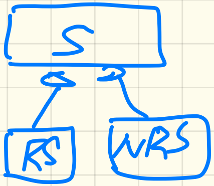
ST: NRS

(RS) nrs

sms.addRS((ResidentStudent) nrs) compiles?

NO.

```
NonResidentStudent nrs = new NonResidentStudent();
/* ST: NonResidentStudent; DT: NonResidentStudent */
StudentManagementSystem sms = new StudentManagementSystem();
sms.addRS(nrs);
```



```
class SMS {
    void addRS(RS rs) { - - }
}
```

```
Student s = new . -
```

```
sms.addRS( (RS) s )
```

valid down cast with ST RS

whether DT of s can fulfill expec. of RS.

should've done

```
if (s instanceof RS) {
    sms.addRS( (RS) s );
}
```

# A Polymorphic Collection of Students (1)

```
ResidentStudent rs = new ResidentStudent("Rachael");
rs.setPremiumRate(1.5);
NonResidentStudent nrs = new NonResidentStudent("Nancy");
nrs.setDiscountRate(0.5);
StudentManagementSystem sms = new StudentManagementSystem();
sms.addStudent(rs); /* polymorphism */
sms.addStudent(nrs); /* polymorphism */
Course eecs2030 = new Course("EECS2030", 500.0);
sms.registerAll(eecs2030);
for(int i = 0; i < sms.numberOfStudents; i ++) {
  /* Dynamic Binding:
   * Right version of getTuition will be called */
  System.out.println(sms.students[i]. getTuition());
}
```

DT: NRS

DT: RS ← sms.ss[0].getTuition();

sms.ss[1].getTuition();

call version of getT in RS

nrs

~~rs~~

void addStudent(S ~~x~~){
ss[c] = s;
c++;
}

```
class StudentManagementSystem {
  Student[] students;
  int numOfStudents;

  void addStudent(Student s) {
    students[numOfStudents] = s;
    numOfStudents ++;
  }

  void registerAll (Course c) {
    for(int i = 0; i < numberOfStudents; i ++) {
      students[i].register(c)
    }
  }
}
```

ss

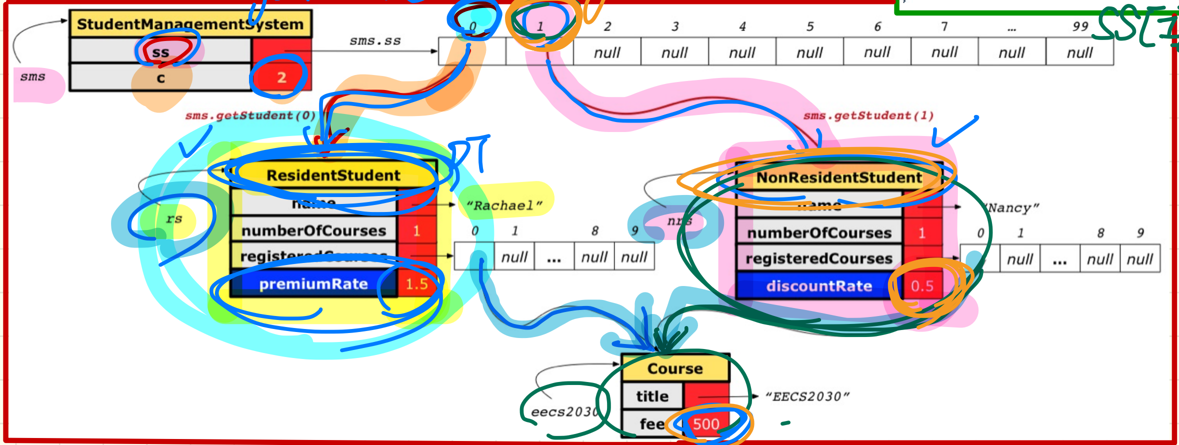
ss

ss[0].register(c);

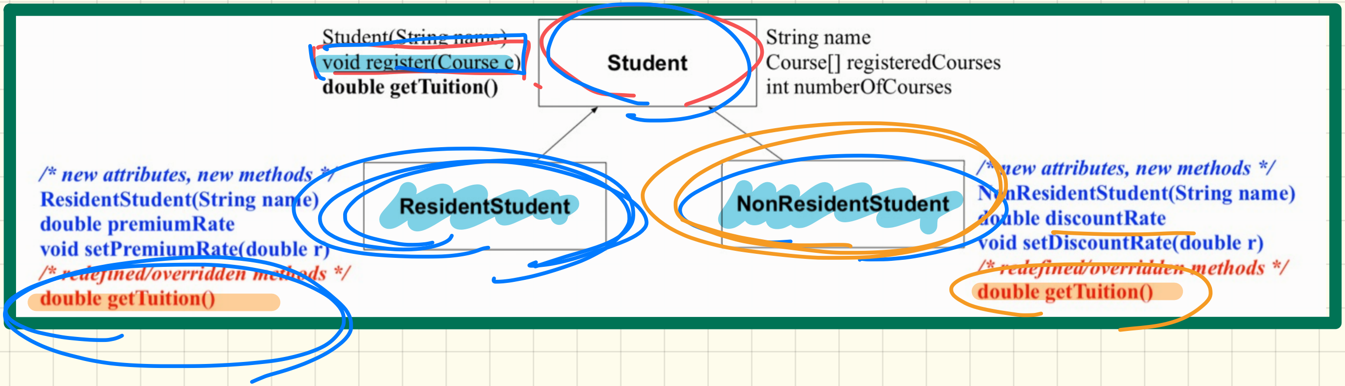
ss[1].register(c);

ss[0] = rs;

ss[1] = nrs;



# Reference: Hierarchy of Students



Student(String name)
void register(Course c)
double getTuition()
Student
String name
Course[] registeredCourses
int numberOfCourses
/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()
ResidentStudent
NonResidentStudent
/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()

# A Polymorphic Collection of Students (2)

```
ResidentStudent rs = new ResidentStudent("Rachael");
rs.setPremiumRate(1.5);
NonResidentStudent nrs = new NonResidentStudent("Nancy");
nrs.setDiscountRate(0.5);
StudentManagementSystem sms = new StudentManagementSystem();
sms.addStudent( rs ); /* polymorphism */
sms.addStudent( nrs ); /* polymorphism */
Course eecs2030 = new Course("EECS2030", 500.0);
sms.registerAll(eecs2030);
for(int i = 0; i < sms.numberOfStudents; i ++) {
  /* Dynamic Binding:
   * Right version of getTuition will be called */
  System.out.println(sms.students[i]. getTuition() );
}
```

ST: Student

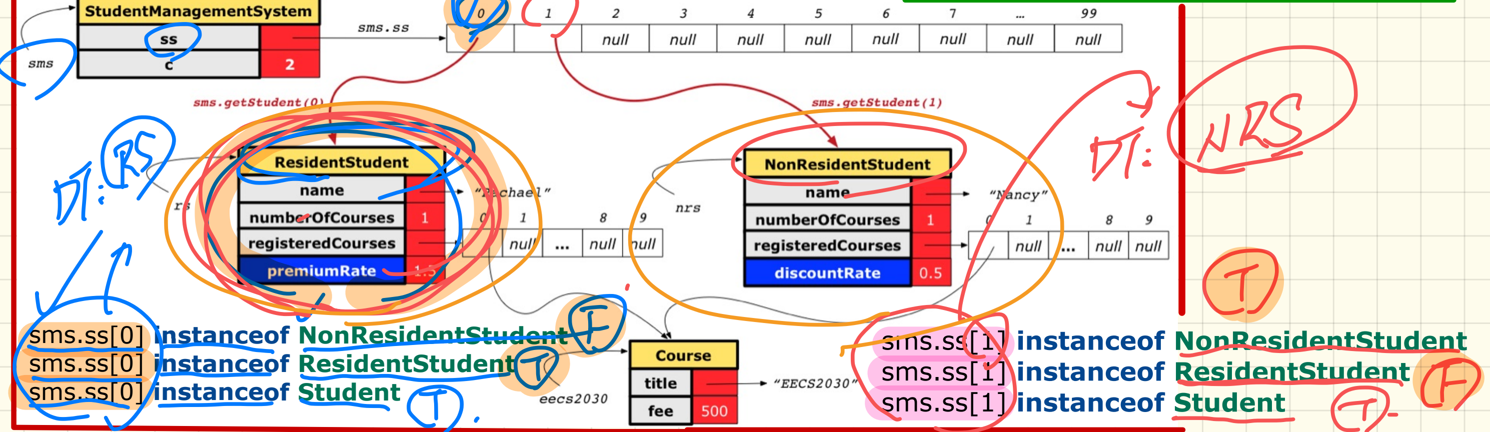
✓ sms. ss[0] . register(eecs2030);

~~sms. ss[0] . setPr(1.8)~~

↳ ST: Student

~~DT: RS~~

```
class StudentManagementSystem {
  Student[] students;
  int numOfStudents;

  void addStudent(Student s) {
    students[numOfStudents] = s;
    numOfStudents ++;
  }

  void registerAll (Course c) {
    for(int i = 0; i < numberOfStudents; i ++) {
      students[i].register(c)
    }
  }
}
```



sms.ss[0] instanceof NonResidentStudent F

sms.ss[0] instanceof ResidentStudent T

sms.ss[0] instanceof Student T

sms.ss[1] instanceof NonResidentStudent T

sms.ss[1] instanceof ResidentStudent F

sms.ss[1] instanceof Student T

```
class SMS {
  Student ss[] - -
}

SMS.ss[0].setPr(- -)
```

ST : Student

not declared (not part of expectation)

# Polymorphic Return Values

```
class StudentManagementSystem {
  Student[] ss; int c;
  void addStudent(Student s) { ss[c] = s; c++; }
  Student getStudent(int i) {
    Student s = null;
    if(i < 0 || i >= c) {
      throw new IllegalArgumentException("Invalid
    }
    else {
      s = ss[i];
    }
    return s;
  } }
```

```
Course eecs2030 = new Course("EECS2030", 500);
ResidentStudent rs = new ResidentStudent("Rachael");
rs.setPremiumRate(1.5); rs.register(eecs2030);
NonResidentStudent nrs = new NonResidentStudent("Nancy");
nrs.setDiscountRate(0.5); nrs.register(eecs2030);
StudentManagementSystem sms = new StudentManagementSystem();
sms.addStudent(rs); sms.addStudent(nrs);
Student s = sms.getStudent(0) ; /* dynamic type of s? */
                  static return type: Student
print(s instanceof Student && s instanceof ResidentStudent);/*true*/
print(s instanceof NonResidentStudent); /* false */
print(s.getTuition());/*Version in ResidentStudent called:750*/
ResidentStudent rs2 = sms.getStudent(0); ×
s =       sms.getStudent(1)   ; /* dynamic type of s? */
         static return type: Student
print(s instanceof Student && s instanceof NonResidentStudent);/*true*/
print(s instanceof ResidentStudent); /* false */
print(s.getTuition());/*Version in NonResidentStudent called:250*/
NonResidentStudent nrs2 = sms.getStudent(1); ×
```

DT: RS

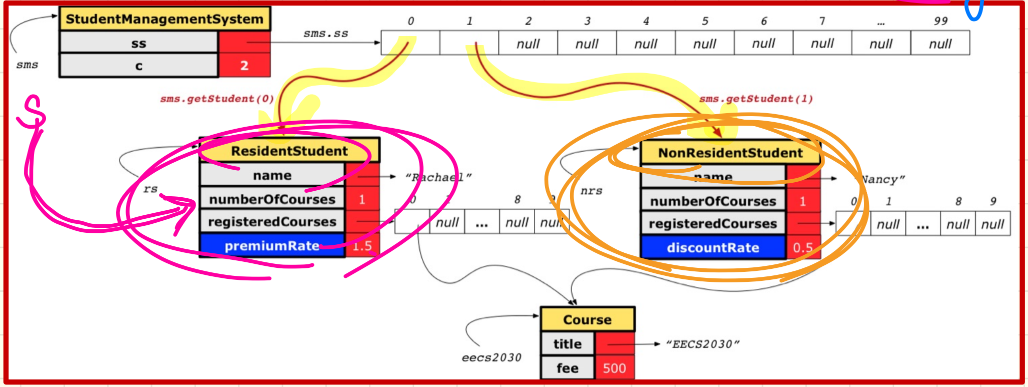
Student s = sms.getStudent(0);

ST: Student

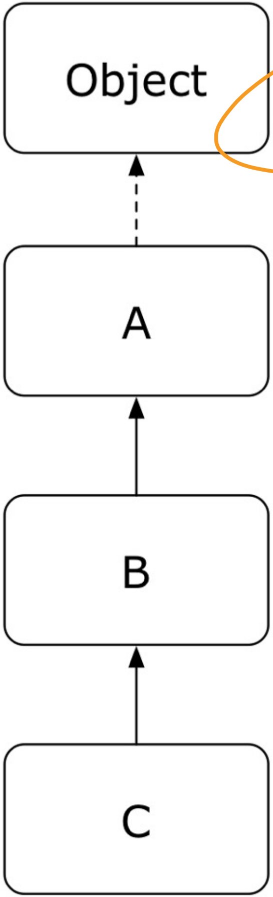
T

sms.getStudent(0) instanceof ~~ResidentStudent~~ NRS

F



# Overridden Methods and Dynamic Binding (1)



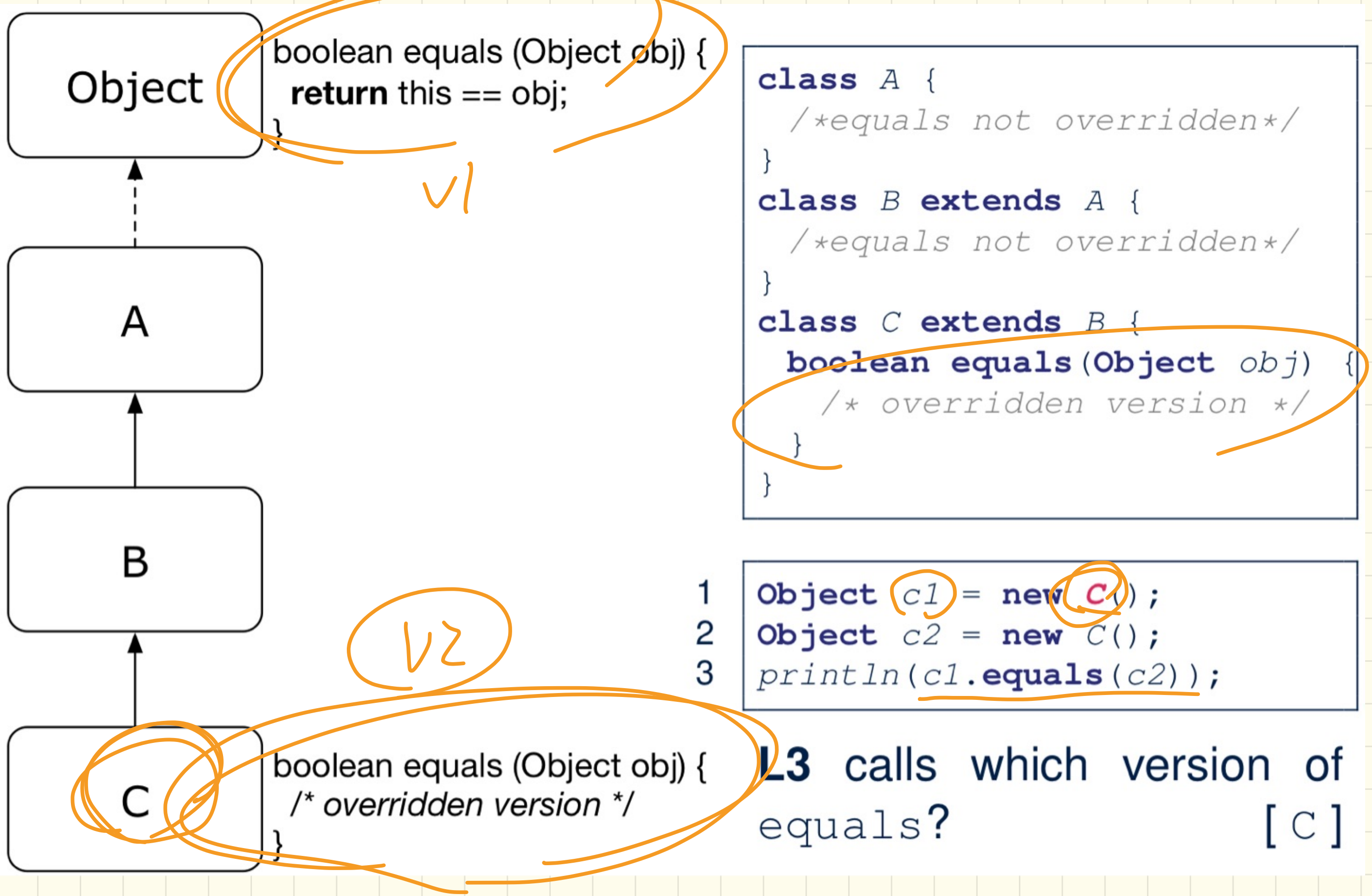
```
boolean equals (Object obj) {
  return this == obj;
}
```

```
class A {
  /*equals not overridden*/
}
class B extends A {
  /*equals not overridden*/
}
class C extends B {
  /*equals not overridden*/
}
```

```
1 Object c1 = new C();
2 Object c2 = new C();
3 println(c1.equals(c2));
```

**L3** calls which version of equals? [ Object ]

# Overridden Methods and Dynamic Binding (2)



```
boolean equals (Object obj) {
  return this == obj;
}
```

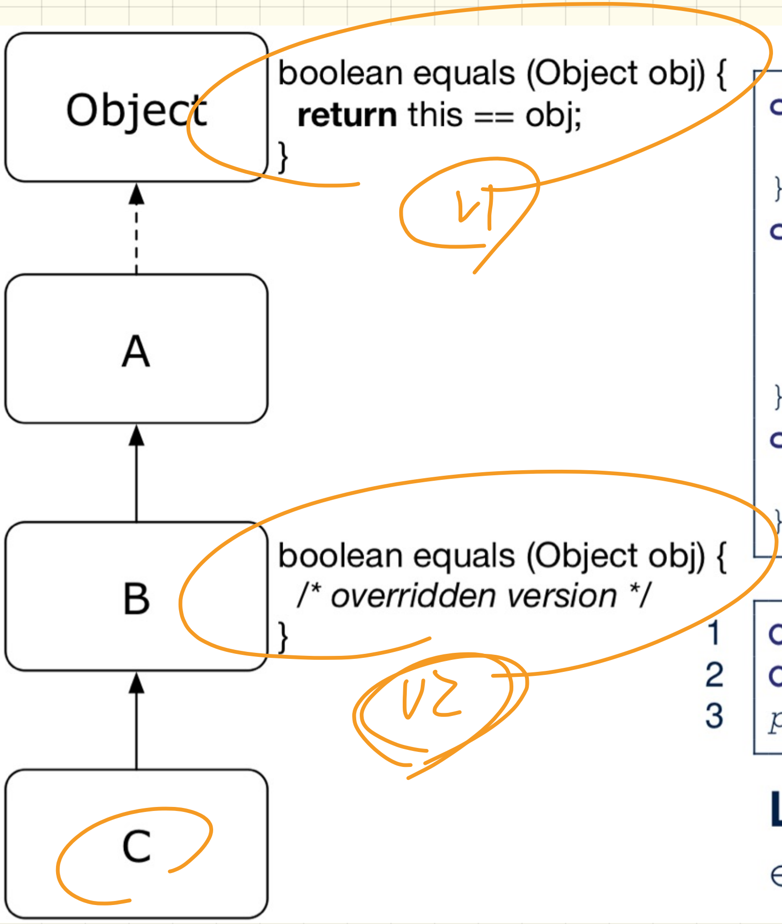
```
boolean equals (Object obj) {
  /* overridden version */
}
```

```
class A {
  /*equals not overridden*/
}
class B extends A {
  /*equals not overridden*/
}
class C extends B {
  boolean equals(Object obj) {
    /* overridden version */
  }
}
```

```
1  Object c1 = new C();
2  Object c2 = new C();
3  println(c1.equals(c2));
```

L3 calls which version of equals? [C]

# Overridden Methods and Dynamic Binding (3)



```
boolean equals (Object obj) {
  return this == obj;
}
```

```
boolean equals (Object obj) {
  /* overridden version */
}
```

```
class A {
  /*equals not overridden*/
}
class B extends A {
  boolean equals(Object obj) {
    /* overridden version */
  }
}
class C extends B {
  /*equals not overridden*/
}
```

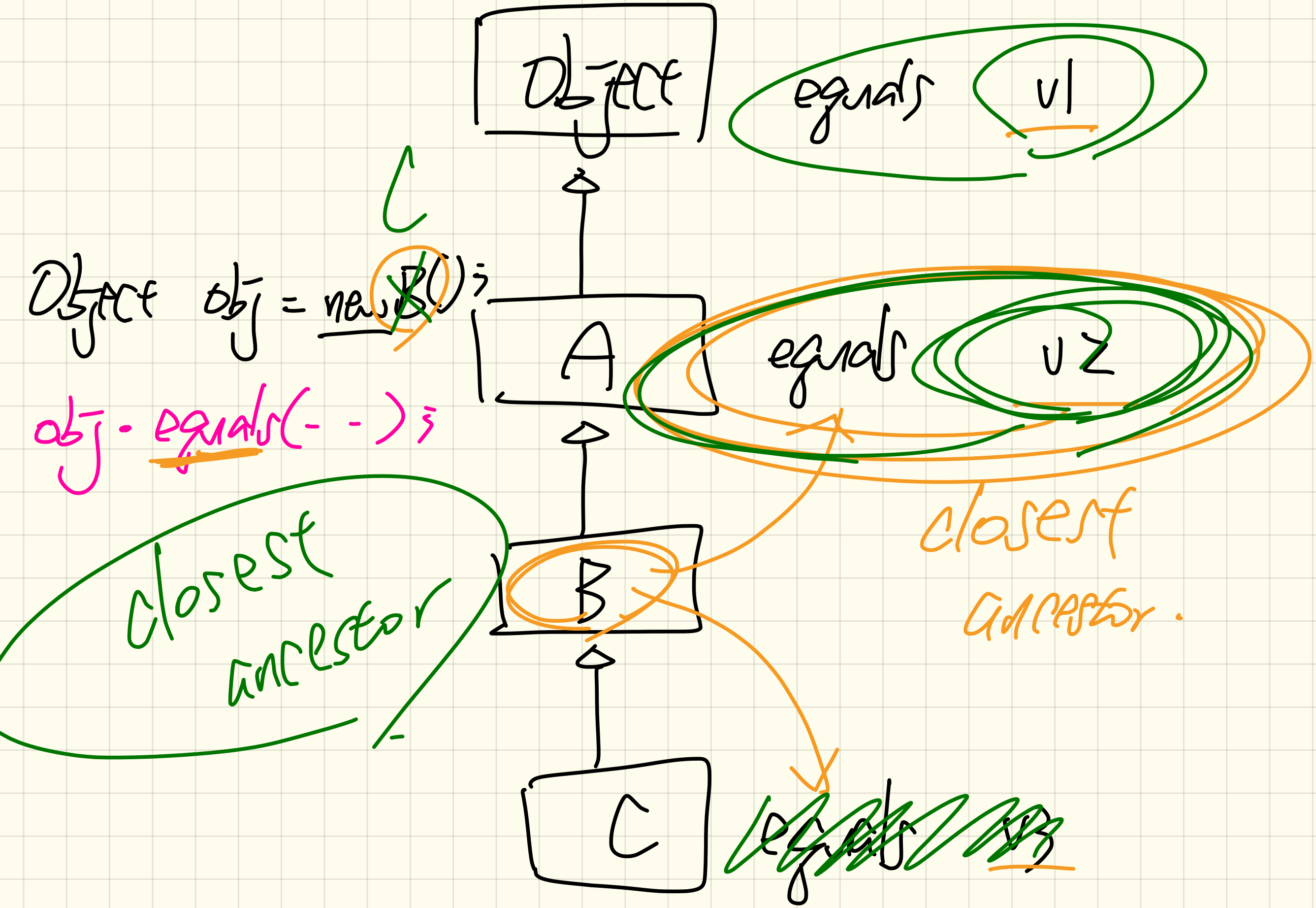
```
1  Object c1 = new C();
2  Object c2 = new C();
3  println(c1.equals(c2));
```

**L3** calls which version of `equals`? [ B ]

C

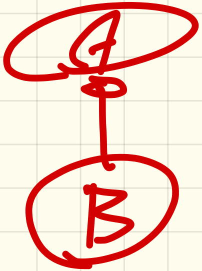
Object obj = new ~~B~~();

obj.equals(- -);



# Test 2 Review

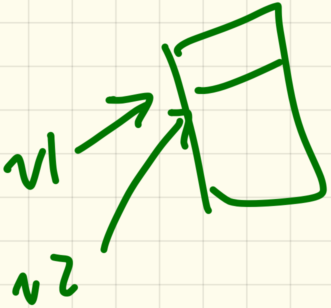
Friday November 15



```
@Test
public void test() {
    [ A obj = new B(...); ]

}
```

```
class A { - -- }
            extend A.
class B { - -- -- }
```

## aggregation.



1. allow sharing
2. aliasing possible

## composition

1. sharing not allowed
2. aliasing is absent.

```
class Person {
  int age;
}
  Person(int age){
    this.age = age;
  }
```

```
PersonCollector pc1 = new PersonCollector();
PersonCollector pc2 = new PersonCollector();
Person p1 = new Person(23);
pc1.addPerson(p1);  pc2.addPerson(p1);
```

pc1.ps[0] == pc2.ps[0]  T

```
class PersonCollector {
  Person[] ps;
  int nop;
  PersonCollector(){
    this.ps = new Person[10];
    this.nop = 0;
  }
```

→ void addPerson(Person ~~p~~ p1){

pc1 ~~this~~ pc2 .ps[nop] = p;

pc1 ~~this~~.nop ++;

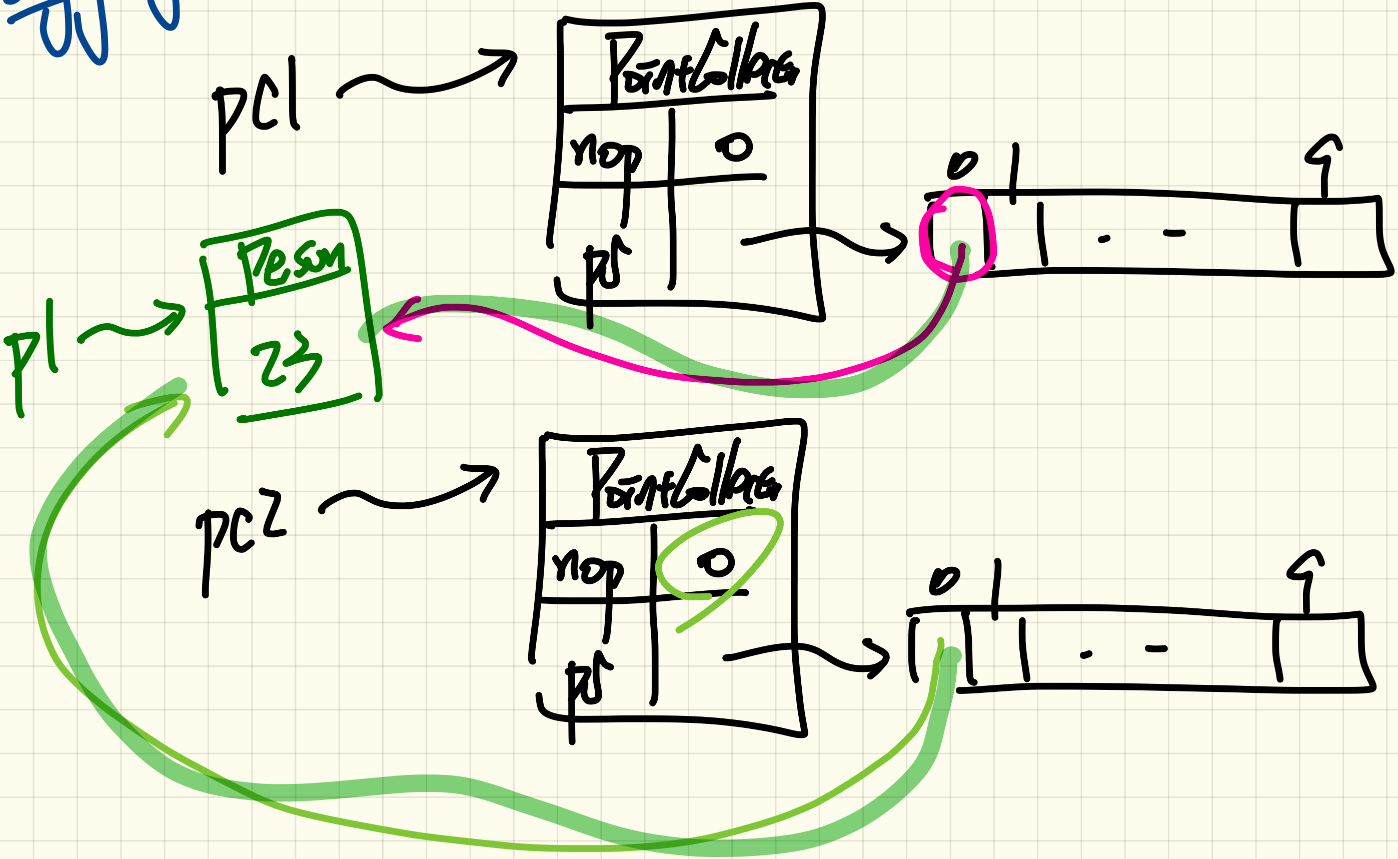
}

pc2 ✓

pc1.ps[0] = p1;

pc2.ps[0] = p1;

aggregation -



void addPerson ( Person p1 ~~p~~ ) {

→ Person newPerson = new Person ( p.age );

pc2 ~~pc1~~ this . ps [ nop ] = newPerson;

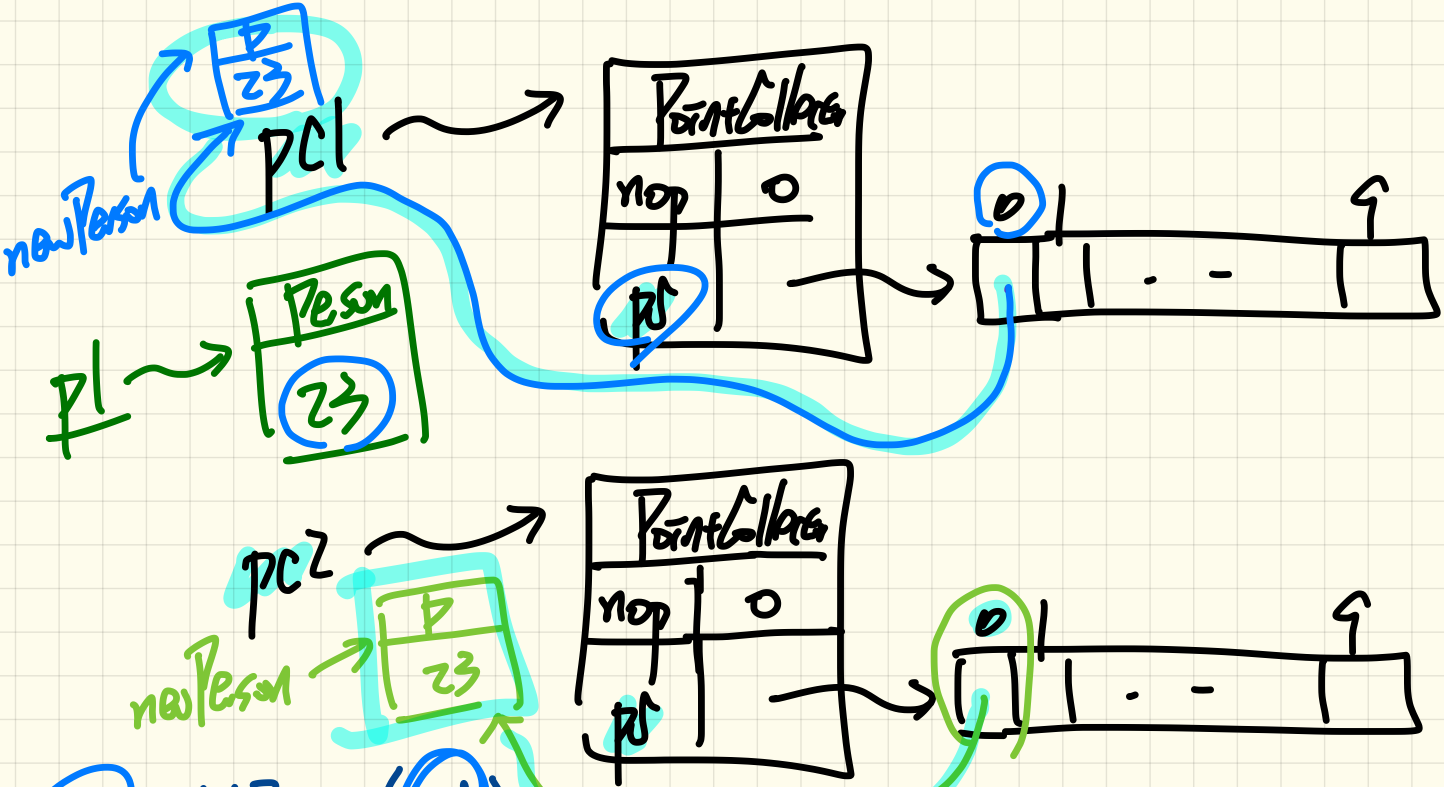
this . nop ++ ;

} ~~pc1~~

pc2

pc1 . ps[ nop ] = newPerson ;

pc2 . ps [ nop ] = newPerson;



```
pc1.addPerson(p1);
pc2.addPerson(p1);
```

pc1.ps[0] == pc2.ps[0]

F

```
class PersonCollector {
  Person[] ps;
  int      nop;

  PersonCollector(PersonCollector ~~other~~) {
                                         pc1
    pc2
    ~~this~~.nop = ~~other~~.nop;
    for(int i=0; i< this.nop; i++){
      this.ps[i] = other.ps[i];
             0             0
    }
  }
}
```

```
this.ps = new Person[~~nop~~ other.ps.length];
```

works for aggregation.

```
V2  Person np = new Person(
        other.ps[i].age);
    this.ps[i] = np;
```

v3.

```
this.ps[i] = new Person(other.ps[i].age);
```

v4.

```
✓ this.ps[i] = new Person(other.ps[i]);

class Person {
    Person(Person p) { this.age =
                       p.age; }
}
```

vs.

void addPerson(Person p){

~~this.ps[nop] = p;~~

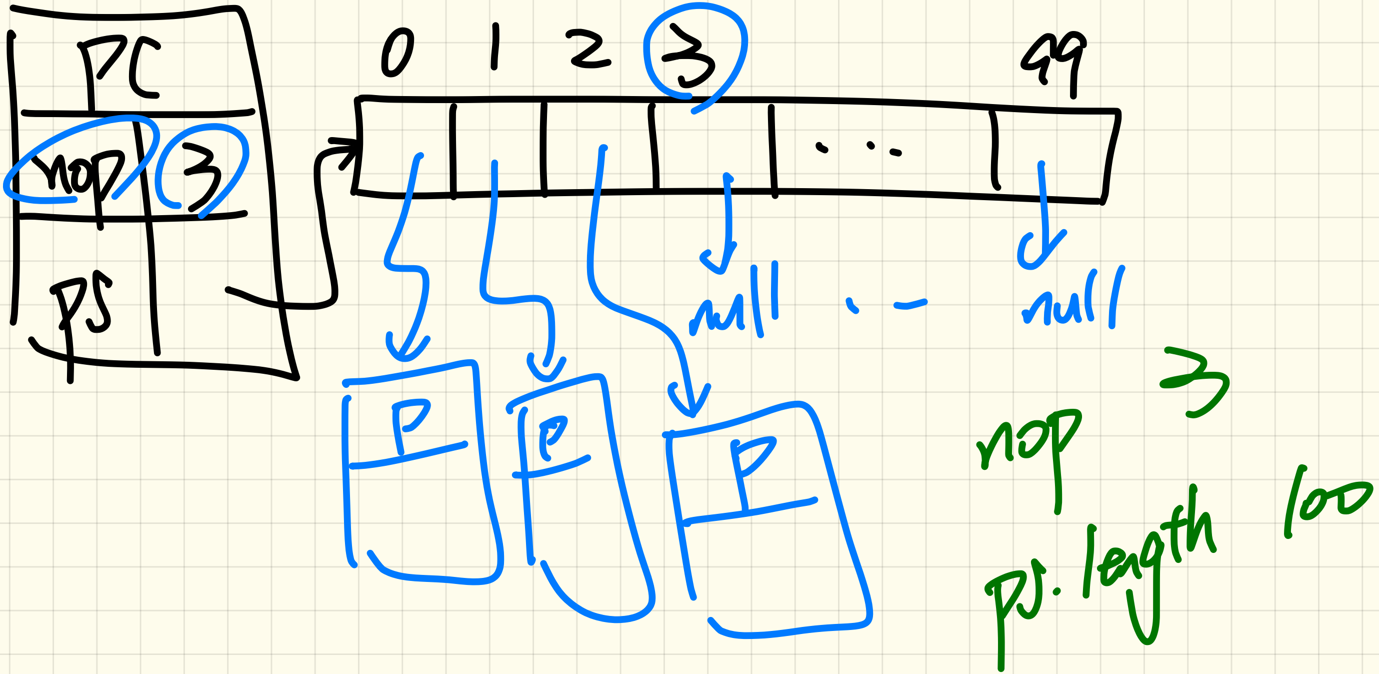
this.nop++;

}

this.ps[nop] = new Person(p);

}

```
PersonCollector(PersonCollector othe){
  this.nop = othe.nop;
  this.ps = new Person[othe.ps.len];
  for(int i = 0; i < this.nop; i++){
    this.addPerson(other.ps[i]);
  }
```


pc
nop
3
ps
0
1
2
3
99
. . .
null
. . .
null
p
p
p
nop 3
ps.length 100

PC pc1 = new PC();

pc1.ps[0] == pc2.ps[0]

pc1.ps[1] == pc2.ps[1]

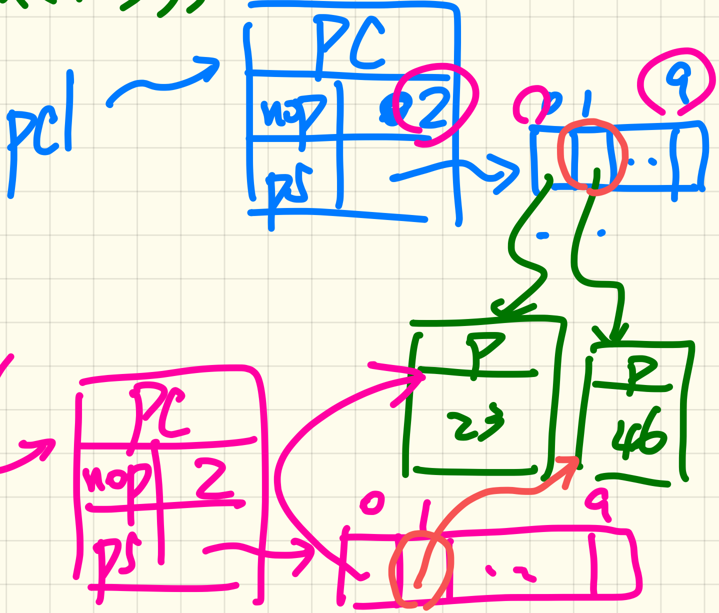
pc1.addPerson(new Person(23));

pc1.addPerson(new Person(46));

PC pc2 = new PC(pc1);

this other

$$\overset{pc2}{\cancel{\text{this}}}.ps[\underset{\substack{0\\1}}{i}] = \overset{pc1}{\cancel{\text{other}}}.ps[\underset{\substack{0\\1}}{i}]$$



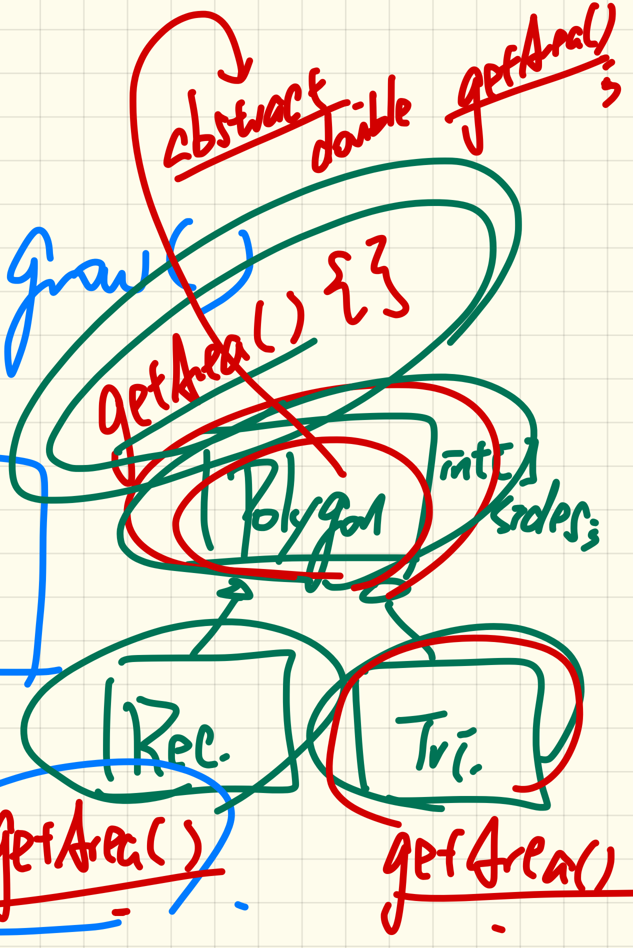
# LECTURE 21

FRIDAY NOVEMBER 22

Polygon p = new Rectangle();

println(p.getArea());

ST: Polygon



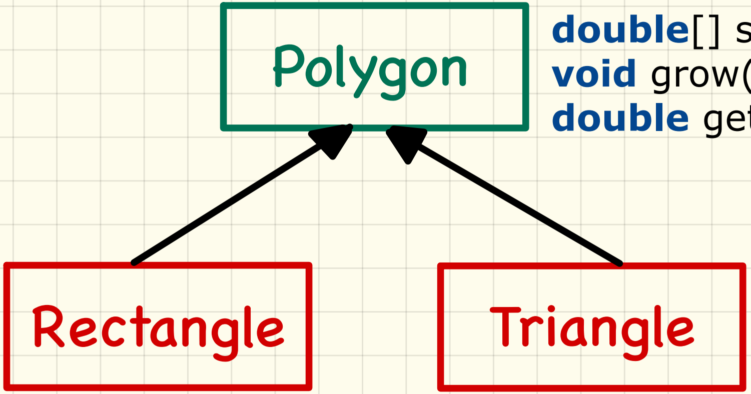
abstract double getArea();

getArea

getPerimeter.

# Abstract Implementation vs. Concrete Implementation

```
abstract double getArea();
```



Polygon

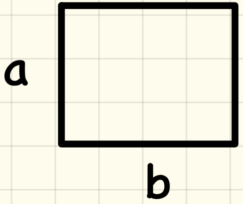
```
double[] sides;
void grow() { ... }
double getPerimeter() { ... }
```

Rectangle

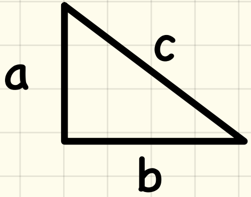
Triangle

```
double getArea() { ... }
```

```
double getArea() { ... }
```



a * b



$$\sqrt{s(s - a)(s - b)(s - c)}$$

# Abstract Class vs. Concrete Descendants

```
public abstract class Polygon {
  double[] sides;
  Polygon(double[] sides) { this.sides = sides; }
  void grow() {
   for(int i = 0; i < sides.length; i ++) { sides[i] ++; }
  }
  double getPerimeter() {
   double perimeter = 0;
   for(int i = 0; i < sides.length; i ++) {
    perimeter += sides[i];
   }
   return perimeter;
  }
  abstract double getArea();
}
```

≥ 1 method abstract

declare signature only

DT: can't be abstract class or interface

Polygon p = new Polygon( );

LHS ST: Polygon

can Polygon satisfy expectations on Polygon

p.grow

p.gP

p.getArea( )

**extends** **extends**

```
public class Rectangle extends Polygon {
 Rectangle(double length, double width) {
  super(new double[4]);
  sides[0] = length; sides[1] = width;
  sides[2] = length; sides[3] = width;
 }
 double getArea() { return sides[0] * sides[1]; }
}
```

```
public class Triangle extends Polygon {
 Triangle(double side1, double side2, double side3) {
  super(new double[3]);
  sides[0] = side1; sides[1] = side2; sides[2] = side3;
 }
 double getArea() {
  /* Heron's formula */
  double s = getPerimeter() * 0.5;
  double area = Math.sqrt(
   s * (s - sides[0]) * (s - sides[1]) * (s - sides[2]));
  return area;
 }
}
```

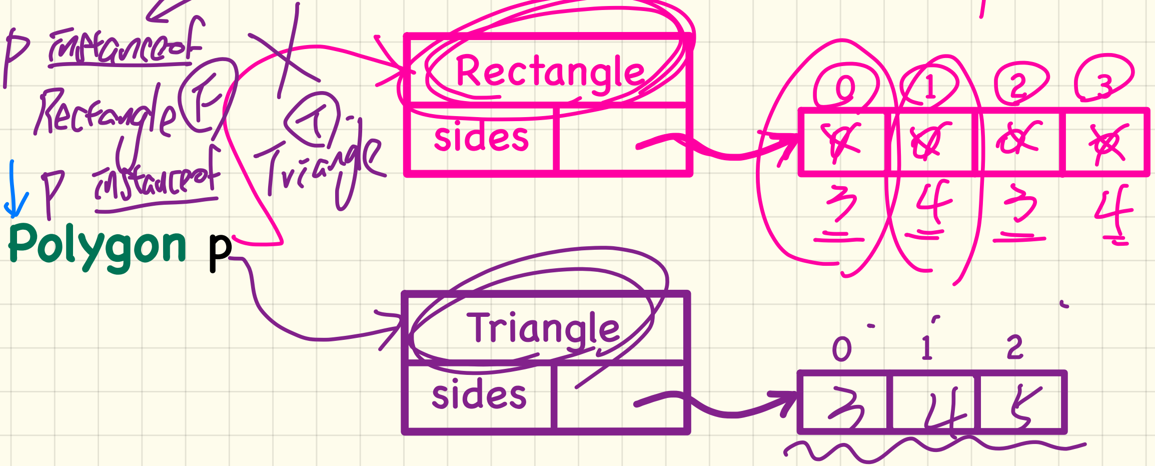
# Polymorphic Assignments of Polygons

```
public abstract class Polygon {
  double[] sides;
  Polygon(double[] sides) { this.sides = sides; }
  void grow() {
    for(int i = 0; i < sides.length; i ++) { sides[i] ++; }
  }
  double getPerimeter() {
    double perimeter = 0;
    for(int i = 0; i < sides.length; i ++) {
      perimeter += sides[i];
    }
    return perimeter;
  }
  abstract double getArea();
}
```

Polygon (new double[4]);
sides[0] = 3;
sides[1] = l;

```
Polygon p;
p = new Rectangle(3, 4); /* polymorphism */
System.out.println(p.getPerimeter()); /* 14.0 */
System.out.println(p.getArea()); /* 12.0 */
p = new Triangle(3, 4, 5); /* polymorphism */
System.out.println(p.getPerimeter()); /* 12.0 */
System.out.println(p.getArea()); /* 6.0 */
```

T.
p instanceof Rectangle

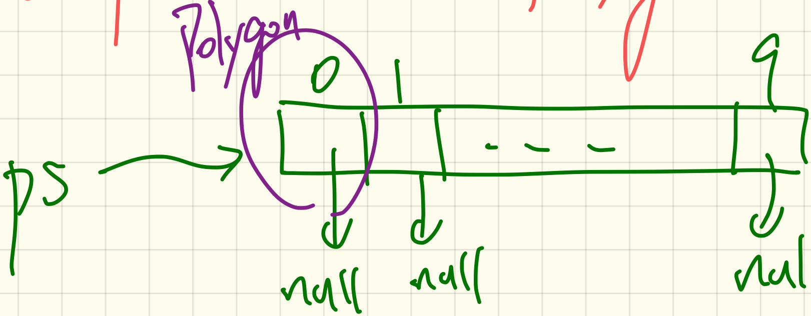


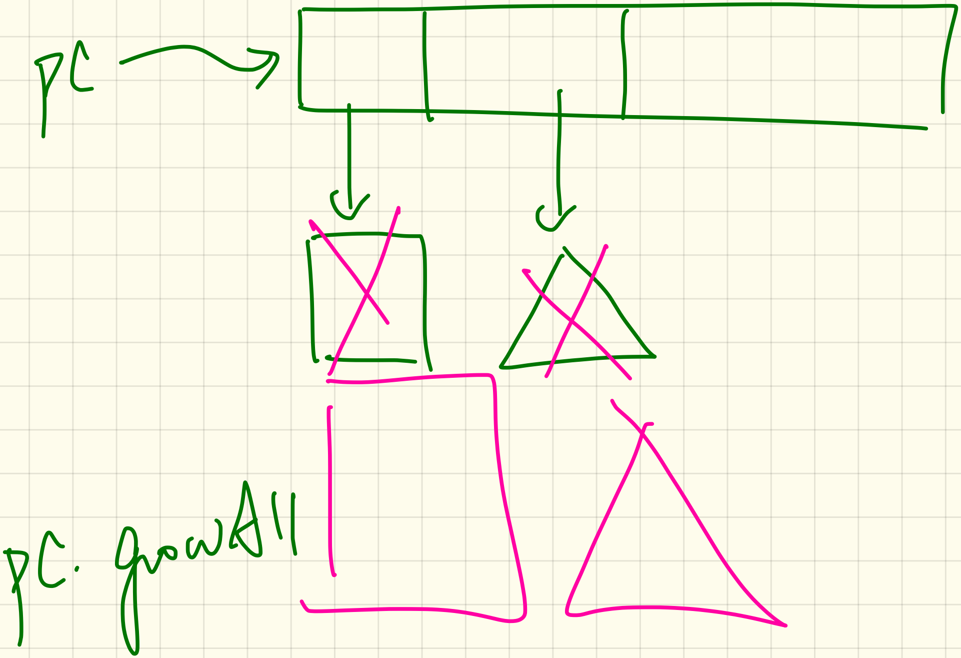
Polygon p = new Polygon(); X

abstract class

✓.

Polygon[] ps = new Polygon[10];




pc
pc.growAll()

# Polymorphic Collection of Polygons

col.polygons[1] instanceof Polygon T

col.polygons[1] instanceof Rectangle F

col.polygons[1] instanceof Triangle T

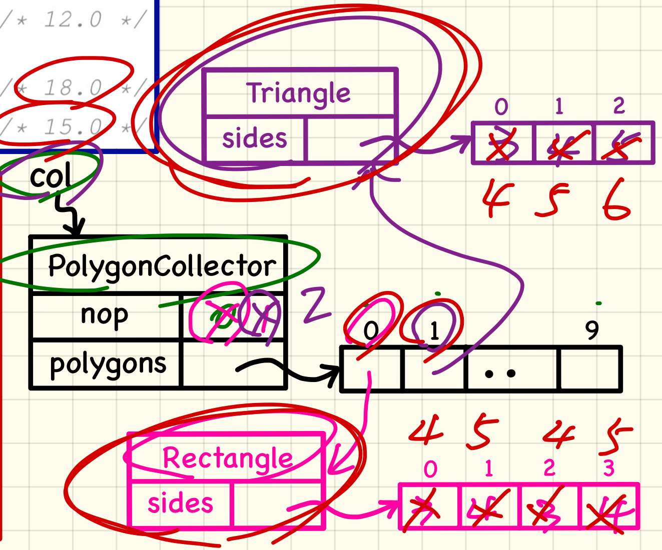
```java
public abstract class Polygon {
  double[] sides;
  Polygon(double[] sides) { this.sides = sides; }
  void grow() {
    for(int i = 0; i < sides.length; i ++) { sides[i] ++; }
  }
  double getPerimeter() {
    double perimeter = 0;
    for(int i = 0; i < sides.length; i ++) {
      perimeter += sides[i];
    }
    return perimeter;
  }
  abstract double getArea();
}
```

```java
PolygonCollector col = new PolygonCollector();
col.addPolygon(new Rectangle(3, 4)); /* polymorphism */
col.addPolygon(new Triangle(3, 4, 5)); /* polymorphism */
System.out.println(col.polygons[0].getPerimeter()); /* 14.0 */
System.out.println(col.polygons[1].getPerimeter()); /* 12.0 */
col.growAll();
System.out.println(col.polygons[0].getPerimeter()); /* 18.0 */
System.out.println(col.polygons[1].getPerimeter()); /* 15.0 */
```

```java
public class PolygonCollector {
  Polygon[] polygons;
  int numberOfPolygons;
  PolygonCollector() { polygons = new Polygon[10]; }
  void addPolygon(Polygon p) {
    polygons[numberOfPolygons] = p; numberOfPolygons ++;
  }
  void growAll() {
    for(int i = 0; i < numberOfPolygons; i ++) {
      polygons[i].grow();
    }
  }
}
```

0

1



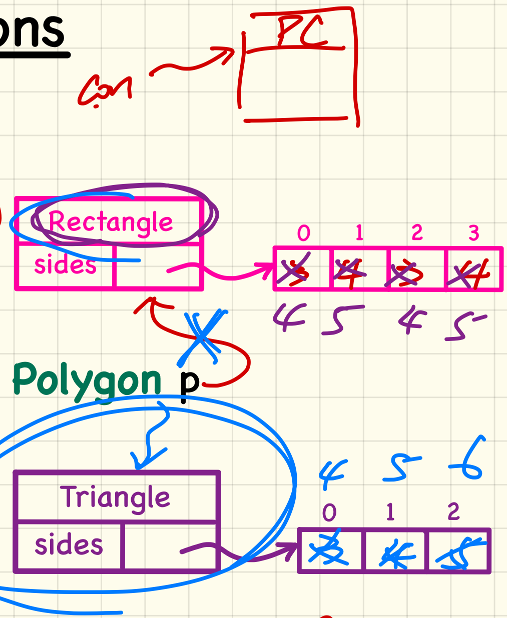
# Polymorphic Return Value of Polygons

Polygon p;

```
PolygonConstructor con = new PolygonConstructor();
double[] recSides = {3, 4, 3, 4}; p = con.getPolygon(recSides);
System.out.println(p instanceof Polygon);      ✓
System.out.println(p instanceof Rectangle);    ✓
System.out.println(p instanceof Triangle);     ✗
System.out.println(p.getPerimeter()); /* 14.0 */
System.out.println(p.getArea()); /* 12.0 */
con.grow(p);
System.out.println(p.getPerimeter()); /* 18.0 */
System.out.println(p.getArea()); /* 20.0 */
double[] triSides = {3, 4, 5}; p = con.getPolygon(triSides);
System.out.println(p instanceof Polygon);      ✓
System.out.println(p instanceof Rectangle);    ✗
System.out.println(p instanceof Triangle);     ✓
System.out.println(p.getPerimeter()); /* 12.0 */
System.out.println(p.getArea()); /* 6.0 */
con.grow(p);
System.out.println(p.getPerimeter()); /* 15.0 */
System.out.println(p.getArea()); /* 9.921 */
```

→ ST: Polygon

{3, 4, 5}



```
public abstract class Polygon {
  double[] sides;
  Polygon(double[] sides) { this.sides = sides; }
  void grow() {
    for(int i = 0; i < sides.length; i ++) { sides[i] ++; }
  }
  double getPerimeter() {
    double perimeter = 0;
    for(int i = 0; i < sides.length; i ++) {
      perimeter += sides[i];
    }
    return perimeter;
  }
  abstract double getArea();
}
```

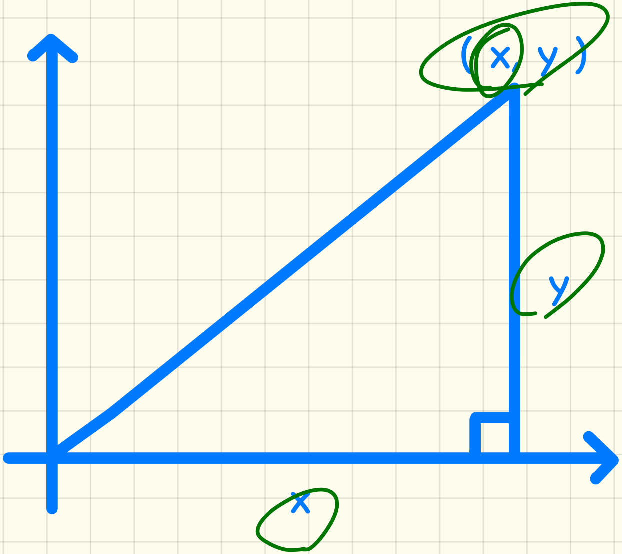
{3, 4, 3, 4}

{3, 4, 5}

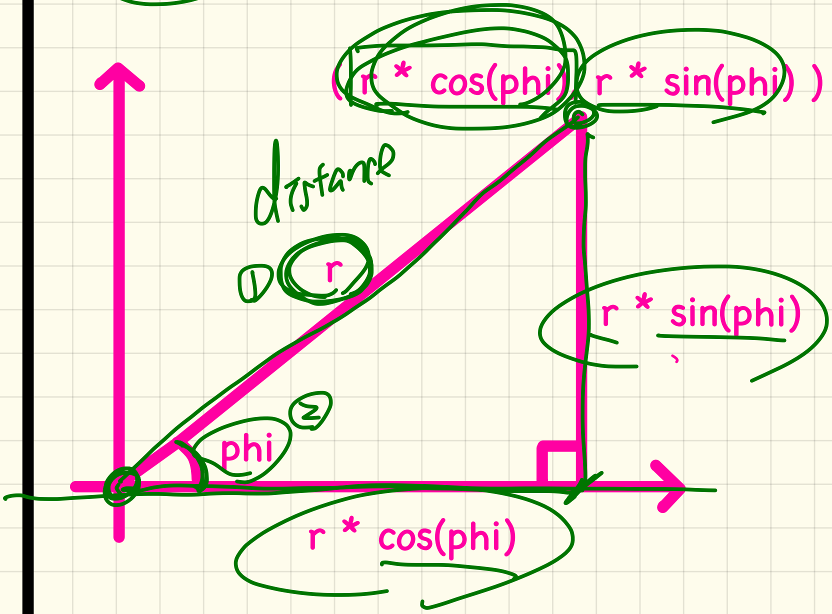
```
public class PolygonConstructor {
  Polygon getPolygon(double[] sides) {
    Polygon p = null;
    if(sides.length == 3) {
      p = new Triangle(sides[0], sides[1], sides[2]);
    }                      3 ✓       4          5
    else if(sides.length == 4) {
      p = new Rectangle(sides[0], sides[1]);
    }                      3          4
    return p;
  }
  void grow(Polygon p) { p.grow(); }
}
```

# Representations of 2-D Points: Cartesian vs. Polar

## Cartesian System



## Polar System

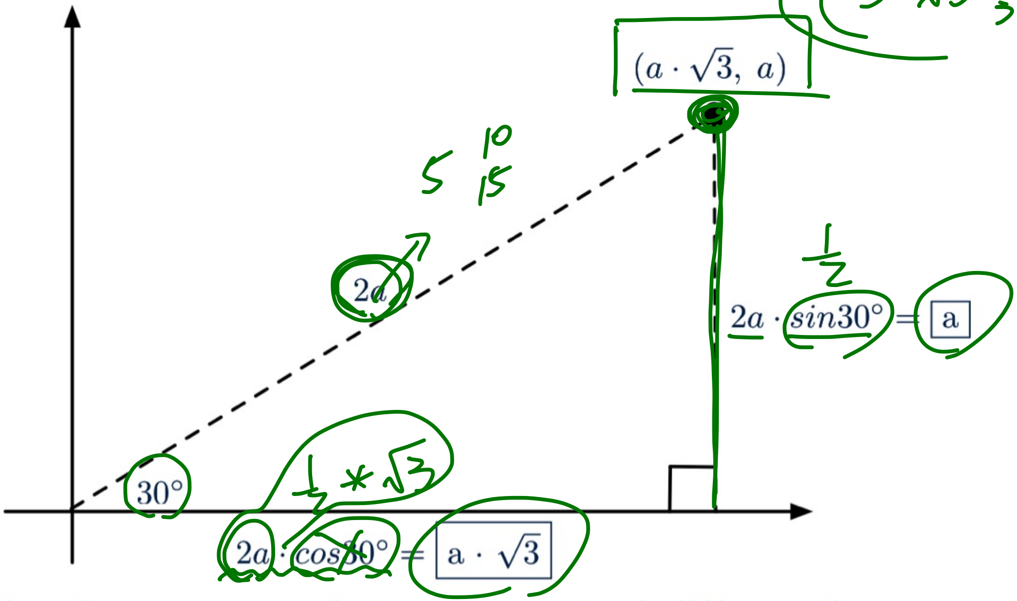


# Cartesian vs. Polar: Example

$a = 5$.

$(5 \cdot \sqrt{3}, 5)$

Recall: $sin30° = \frac{1}{2}$ and $cos30° = \frac{1}{2} \cdot \sqrt{3}$



$(a \cdot \sqrt{3},\ a)$
10
5
15
$2a$
$\frac{1}{2}$
$2a \cdot sin30° = \boxed{a}$
30°
$\frac{1}{2} * \sqrt{3}$
$2a \cdot cos30° = \boxed{a \cdot \sqrt{3}}$

We consider the same point represented differently as:

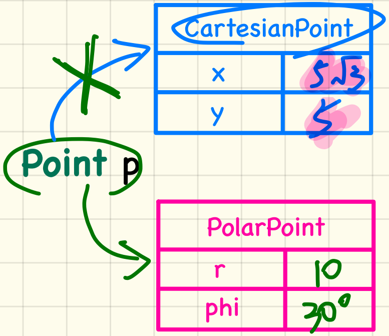
- $r = 2a$, $\psi = 30°$ [ polar system ]
- $x = 2a \cdot cos30° = a \cdot \sqrt{3}$, $y = 2a \cdot sin30° = a$ [ cartesian system ]

Point p = new ~~Point~~()

interface.

= ✓ new CartesianPoint()

✓ new PolarPoint()



```
interface Point {
        double getX();
        double getY();
}
```

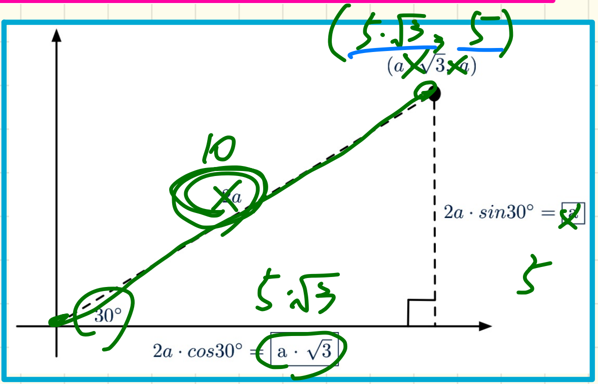
implements

implements

```
public class CartesianPoint implements Point {
  double x;
  double y;
  CartesianPoint(double x, double y) {
    this.x = x;
    this.y = y;
  }
  public double getX() { return x; }
  public double getY() { return y; }
}
```

```
public class PolarPoint implements Point {
  double phi;
  double r;
  public PolarPoint(double r, double phi) {
    this.r = r;
    this.phi = phi;
  }
  public double getX() { return Math.cos(phi) * r; }
  public double getY() { return Math.sin(phi) * r; }
}
```

```
double A = 5;
double X = A * Math.sqrt(3); 5·√3
double Y = A; 5
Point p;              5√3  5
p = new CartisianPoint(X, Y); /* polymorphism */
print("(" + p.getX() + ", " + p.getY() + ")");
                10
p = new PolarPoint(2 * A, Math.toRadians(30)); /
print("(" + p.getX() + ", " + p.getY() + ")");
```



# TEST 2 REVIEW

FRIDAY NOVEMBER 22

```
class Collector {
  C[] cs;
  void add(C obj) {...}
}
```

Collector c = new Coll.Ur();

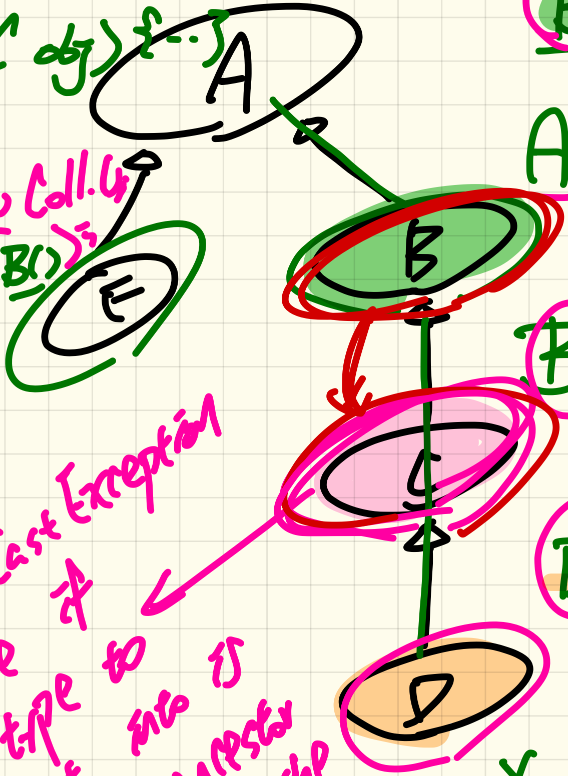
c.add((D) new B());

① valid down cast from B to D

② ClassCastExcep.

∵ DT B is not descen. of D.

ClassCastException free if the type to cast into is an ancestor of the DT.



B obj = new C(); ✓

A oa = (A) obj;

$\boxed{\text{B ob = (A) obj;}}$ A

~~D od = (D) obj;~~ down

X E oe = (E) obj;

- Inheritance
  - ↳ try eclipse demo
  - ↳ practice test
  - ↳ class Collector {

```
      void add(C oc) {...}
  }
```

∴ B cannot fulfill   C. Collector c = — —

X c. add(new B());

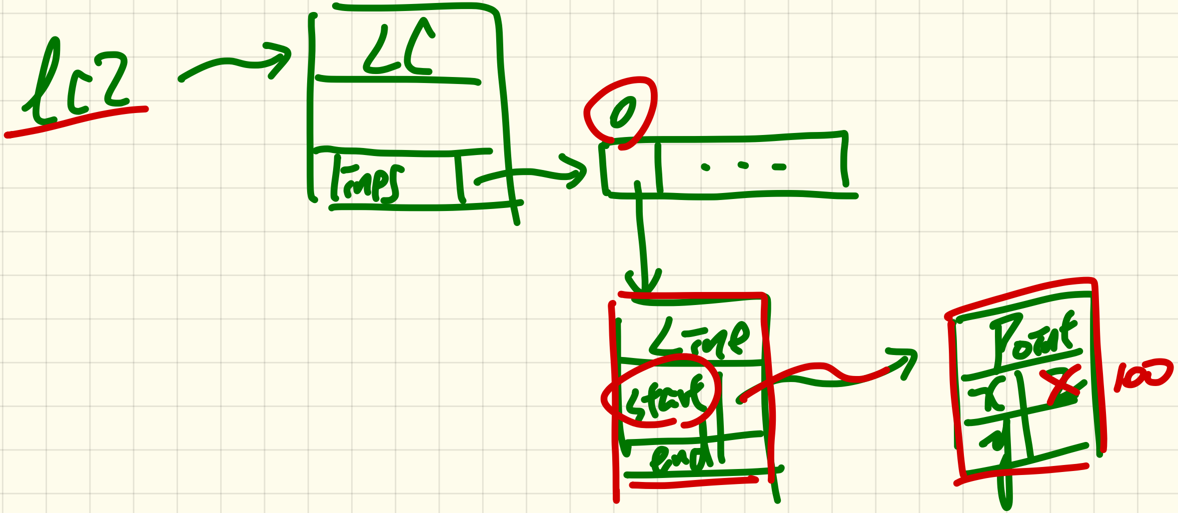
(C)

No RECURSION -

- hashCode
- equals / call by value / collection

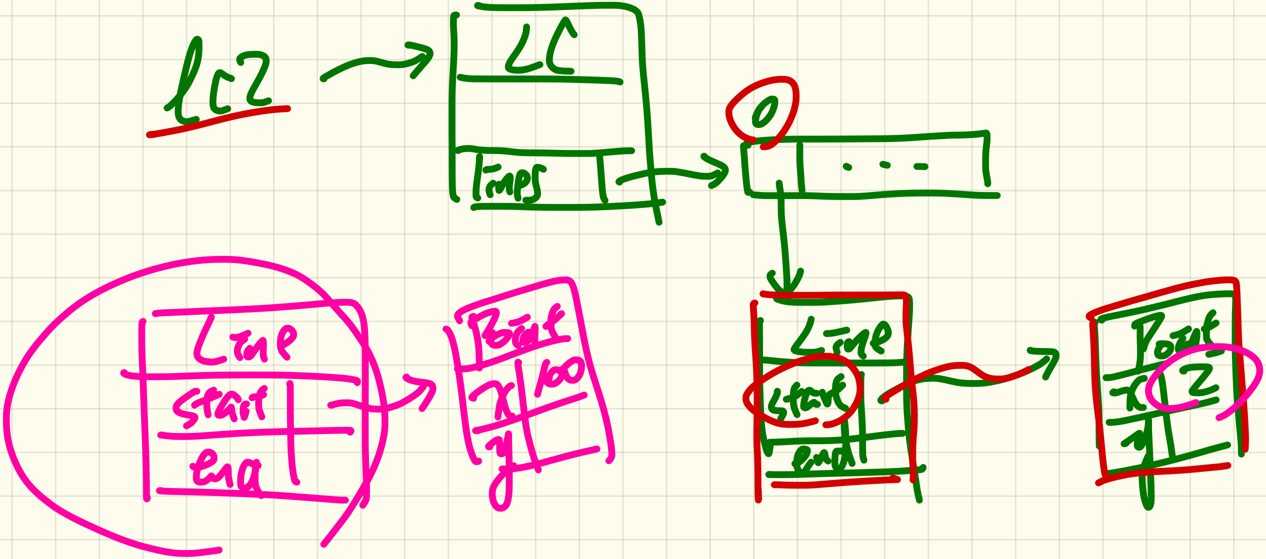
agg:

```
lc2.getLineAt(0).getStart().getX()
lc2.getLineAt(0).getStart().setX(100)
```



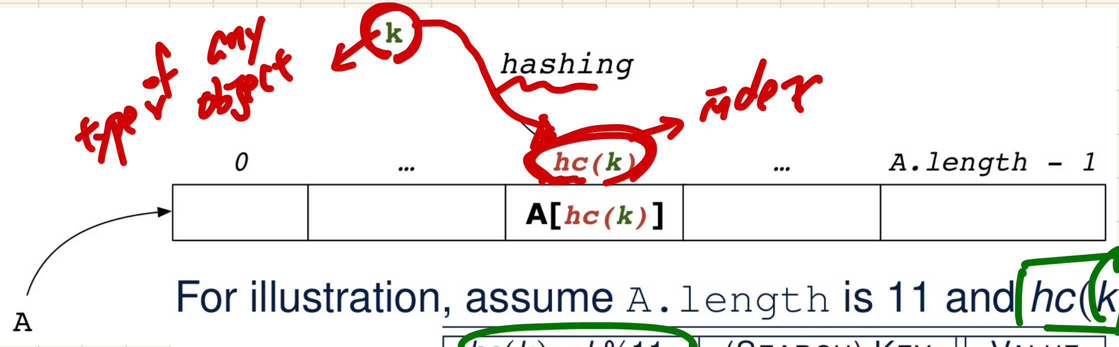
Comp: lc2.getLineAt(0).getStart().getX()

lc2.getLineAt(0).getStart().setX(100)



# Implementing a Hash Table via Hashing

$k1.equals(k2) \Rightarrow hc(k1) = hc(k2)$



- Converting k to hc(k)
- Indexing into A[hc(k)]

int

For illustration, assume A.length is 11 and $hc(k) = k\%11$.

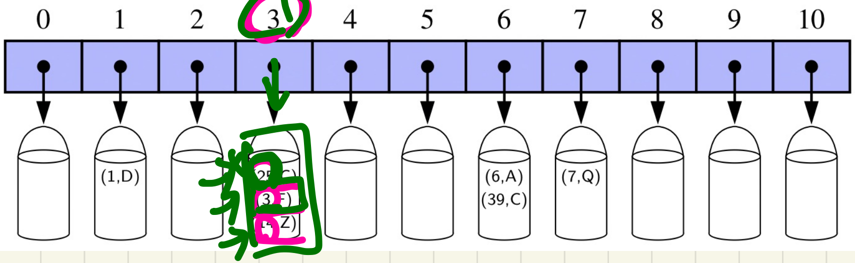
| $hc(k) = k\%11$ | (SEARCH) KEY | VALUE |
|---|---|---|
| [redacted] 1 | 1 | D |
| [redacted] 3 | 25 | C |
| [redacted] 3 | 3 | F |
| [redacted] 3 | 14 | Z |
| [redacted] 6 | 6 | A |
| [redacted] 6 | 39 | C |
| [redacted] 7 | 7 | Q |

collision: distinct keys map to same hashCode

bucket array

search(3)

hc(3) = 3



```
class A {
    B b;
    C c;

void c() {
    b = c.a.ba
}   B    A B
```

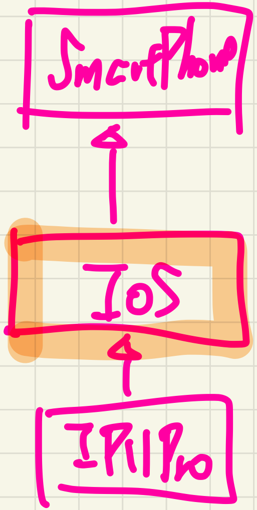
```
class B {
    A a;
    C c;
}
```

```
class C {
    A a;
    B b;
}
```

obj instanceof C

↳ TRUE if ① the dynamic type of obj can fulfill the expectations on C.

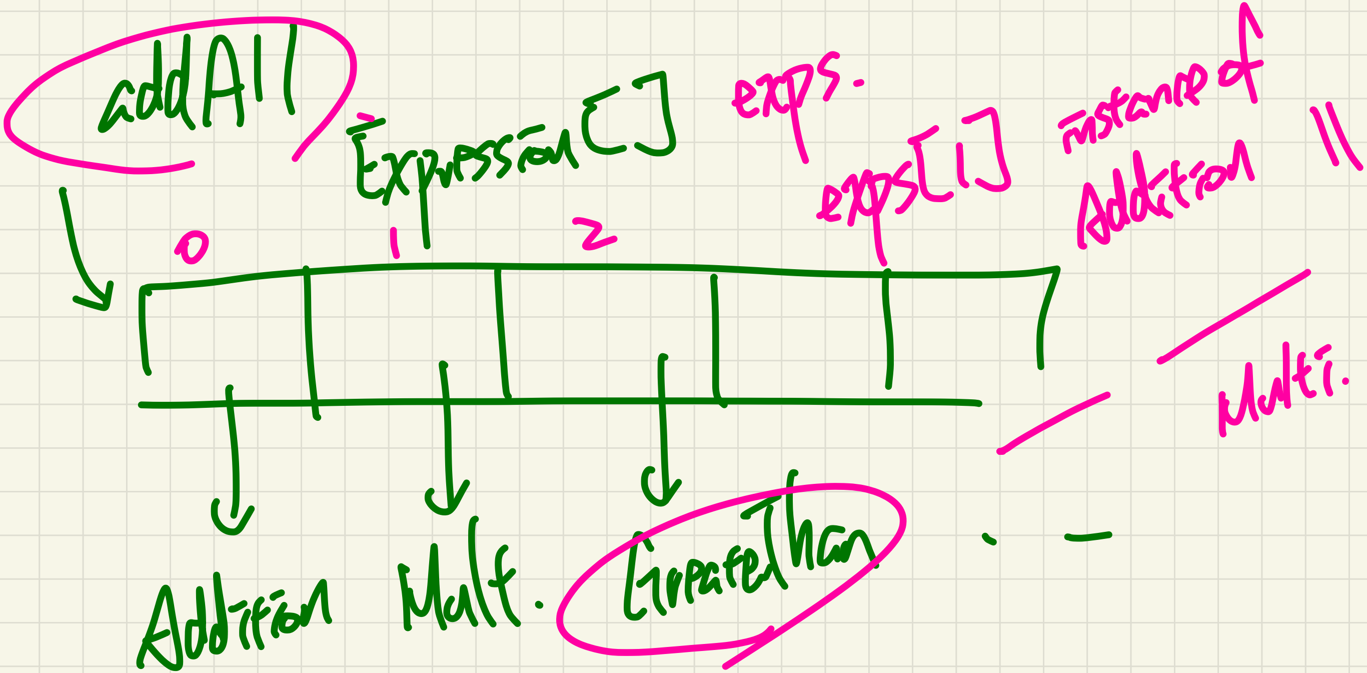
the DT of obj is a descendant of C.



```
SmartPhone p =
      new IOS();
p instanceof SmartPhone   T
p instanceof IOS   T   2
p instanceof IPhone12Pro   F
```

```
if ( obj instanceof C ) {
    ((C) obj
}
```

- Compiles if either upward or downward cast
- CCE if the DT of obj is not a descendant class of C.

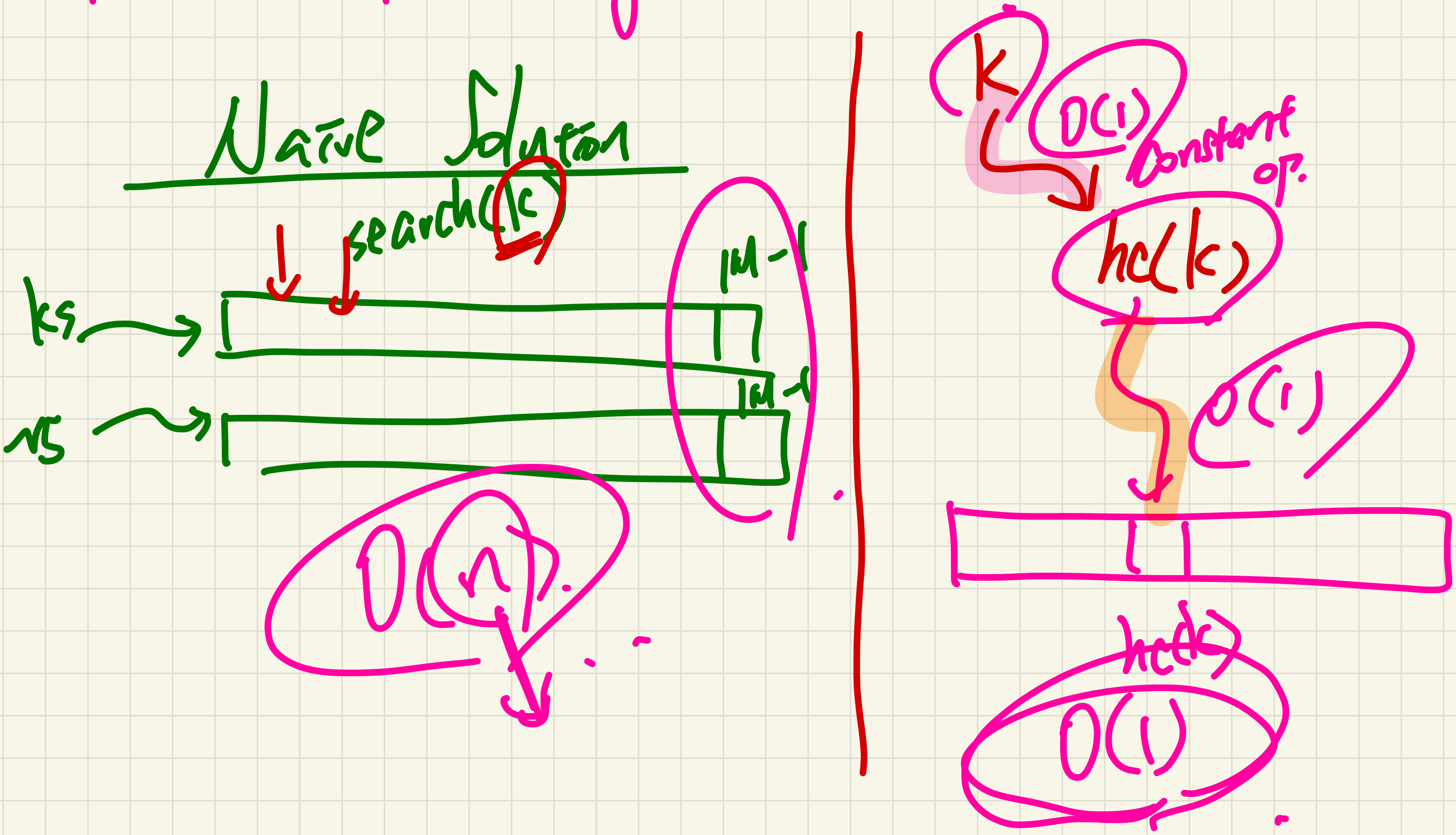

addAll
Expression[] exps.
exps[i] instanceof Addition ||
Multi.
0
1
2
Addition
Mult.
Greaterthan

$(2+3)+(3*4)+(3>4)$

Haskell

↳ functional programming.

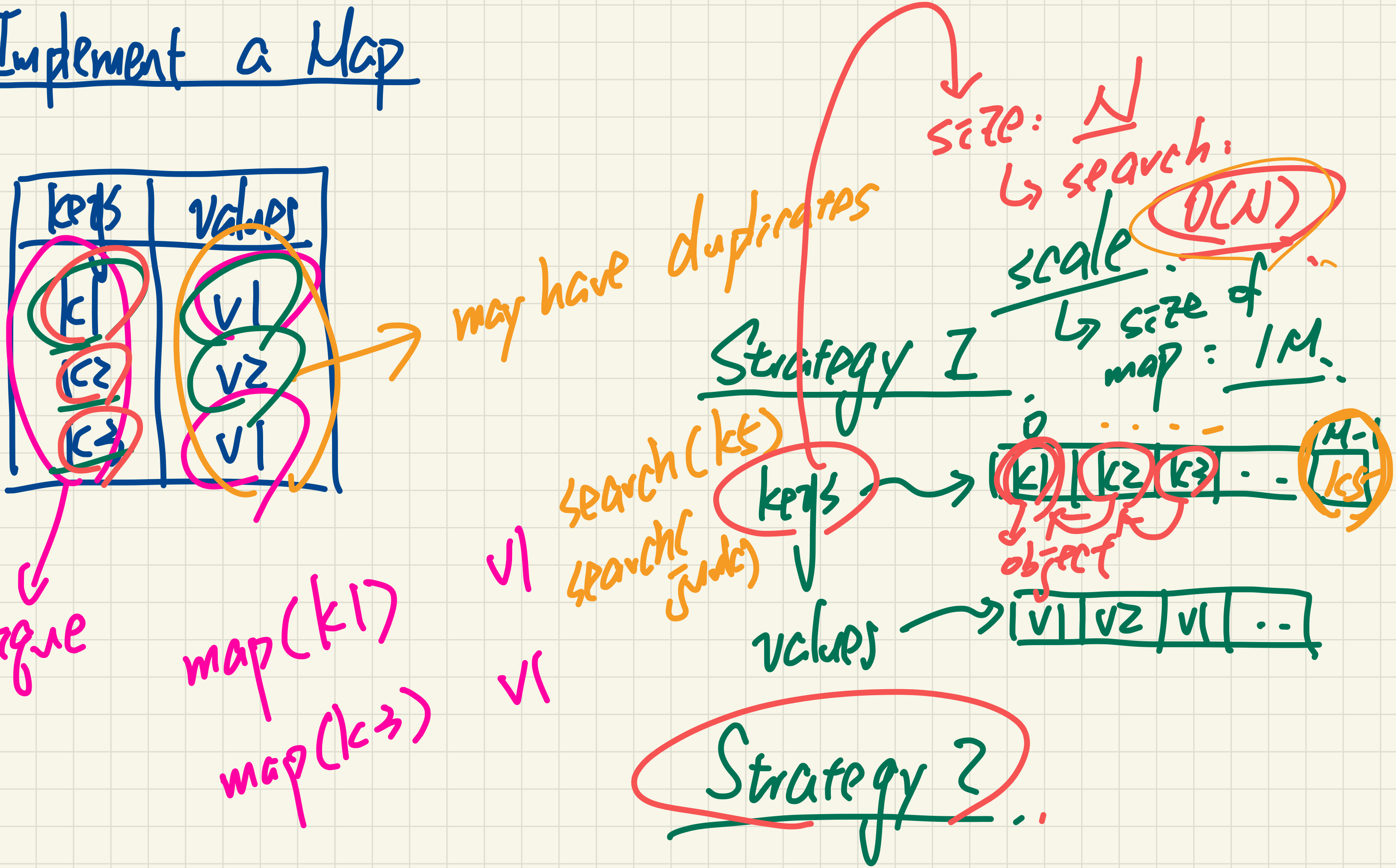
# Implement Map keys & values



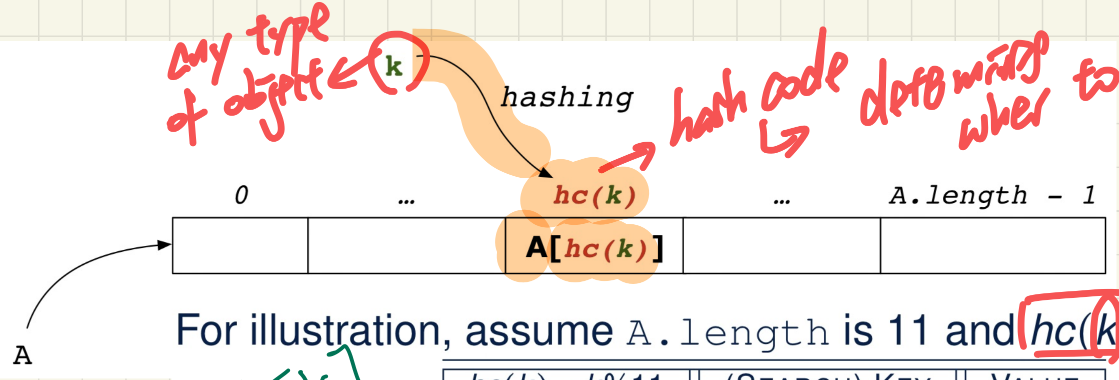
# TEST 2 REVIEW

MONDAY NOVEMBER 25

# Implement a Map



# Implementing a Hash Table via Hashing



any type of object

hash code ↳ determining where to store the key.

- Converting k to hc(k)
- Indexing into A[hc(k)]

hash code ↳ used as index

hc(k) ↳ index

A[hc(k)] ↳ value

index

For illustration, assume A.length is 11 and hc(k) = k%11

not going to be used directly as index

A[k]

| hc(k) = k%11 | (Search) Key | Value |
|---|---|---|
| [redacted] | 1 | D |
| [redacted] | 25 | C |
| [redacted] | 3 | F |
| [redacted] | 14 | Z |
| [redacted] | 6 | A |
| [redacted] | 39 | C |
| [redacted] | 7 | Q |

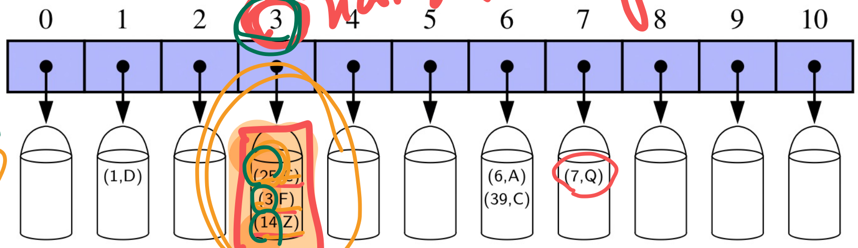
same hash code

distinct keys

collision.

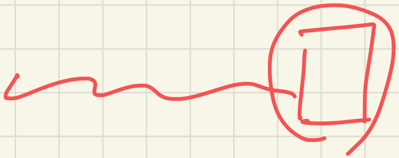
search(k)
↳ calculate hc(k) O(1)
↳ A[hc(k)] O(1)

O(1)



ArrayList<...> list

list.add(...)



HashTable<Integer, String> table =

new HashTable<>();

table.add(k, v);

↳ k.hashCode() →

↳ ba

k.hashCode()

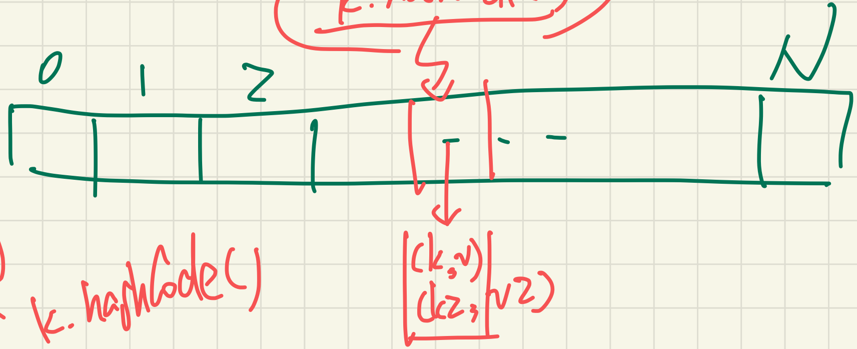


table.add(k2, v2)

↳ k2.hashCode() ~ k.hashCode()

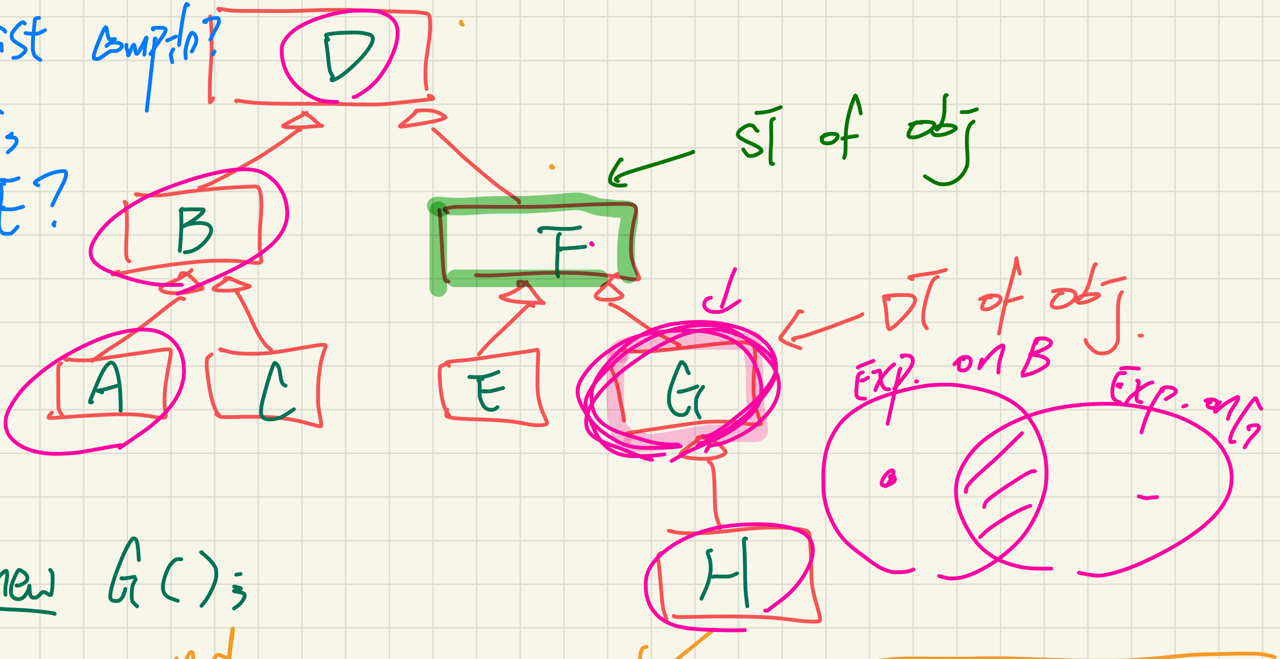
## Bucket Array

Entry

ArrayList < ArrayList <  >>

Q1. does the cast compile?

Q2. If it compiles, do we have CCE?



F obj = new G();

① (D) obj ✓ ∵ upward
✓

② (G) obj ✓ ∵ downward
✓

③ (H) obj ✓ compiles ∵ down
✗ CCE occ.

④ (B) obj

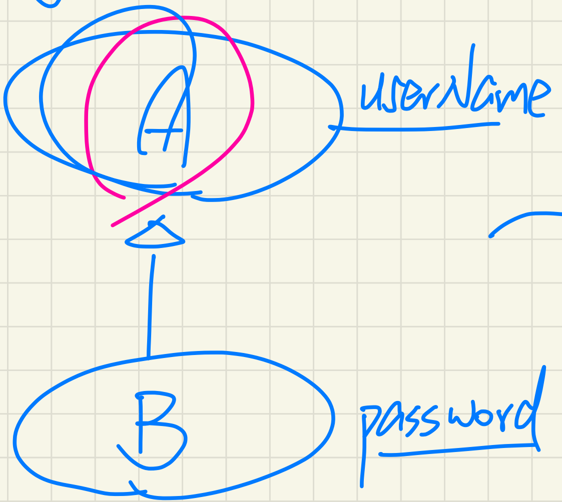
⑤ (A) obj

When do we get a CCE?

C (obj) des. of C.

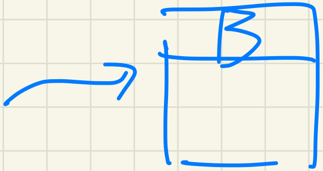
1. When DT of obj is not exp. on C.
2. When DT of obj cannot fulfill C.

# Upward Casting : Access Control



retrieveObj().un

.password



caller would only retrieve on object of static type A.

```
B obj = new B();
obj.un
obj.pw
```

```
A retrieveObj() {
  (A) obj;
}
```

# Call by Value: **Primitive** Argument

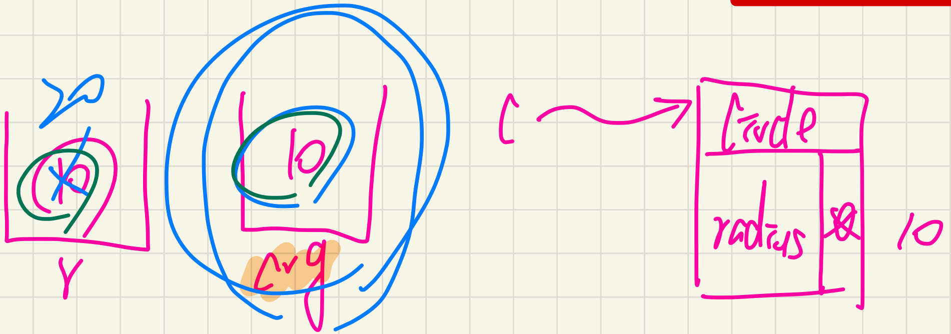
primitive parameter

r = arg

```
class Circle {
  int radius;
  void setRadius(int r) {
    this.radius = r;
  }
}
```

r = 20;

```
class CircleUser {
  ...
  Circle c = new Circle();
  int arg = 10;
  c.setRadius(arg);
  }
}
```



# Call by Value: Reference Argument

address of some circle object.

```
class Circle {
  int radius;
  Circle() {}
  Circle(int r) {
    this.radius = r;
  }
  void setRadius(Circle c) {
    this.radius = c.radius;
  }
}
```

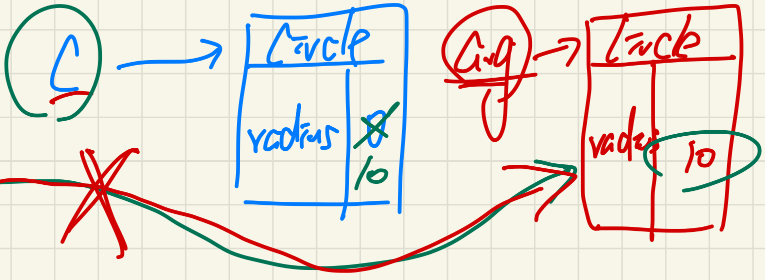
Reference Param.

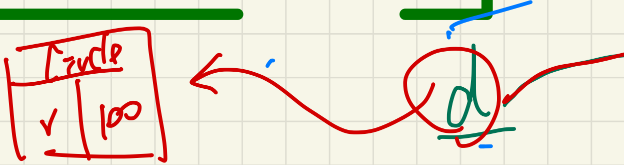
d

d = arg

① d = new Circle(100);

```
class CircleUser {
  ...
  Circle c = new Circle();
  Circle arg = new Circle(10);
  c.setRadius(arg);
}
```





```
class Circle {
  int radius;
  Circle() {}
  Circle(int r) {
    this.radius = r;
  }
  void setRadius(Circle o) {
    this.radius = o.radius;
  }
}
```

Implicitly:
d = arg;

d

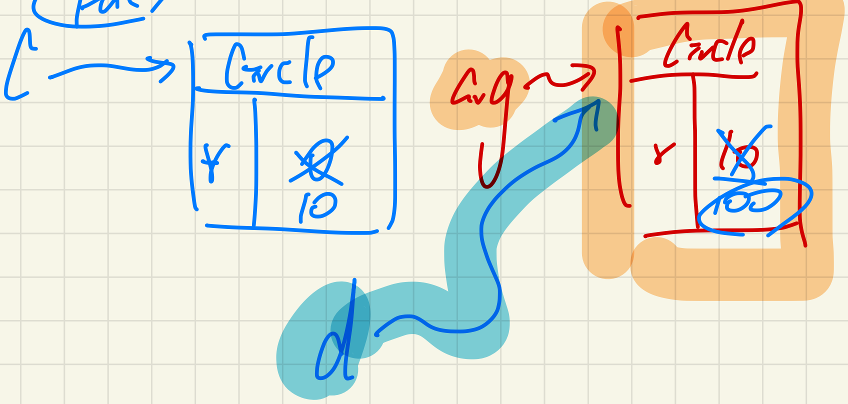
① this.radius = d.radius;

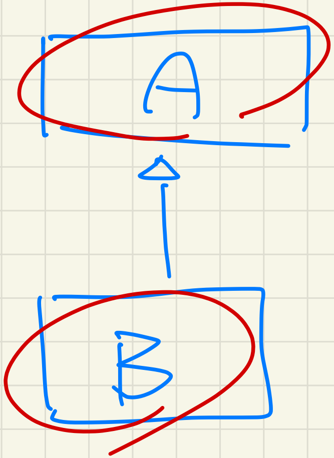
② d.setRadius(100);

```
class CircleUser {
  ...
  Circle c = new Circle();
  Circle arg = new Circle(10);
  c.setRadius(arg);
}
```

assertTrue(arg.radius == 10);

Fail.





①

```
A oa = ...;
B ob = ...;
oa = ob;
× ob = oa;
```

ST: B     ST: A

not compile

②

```
✓ A oa = new B();
  B ob = new A();
```

new object which can fulfill exp. on B.

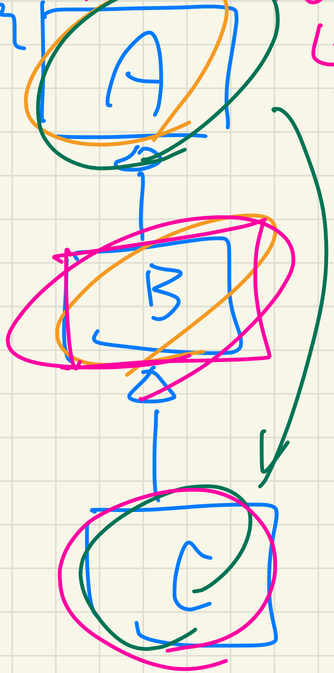
improved:

can the DT of oa fulfill expec. on C.

```
if ((oa instanceof C)) {
  (C) oa; ✓
}
(B) oa;
ob = (C) oa;
```

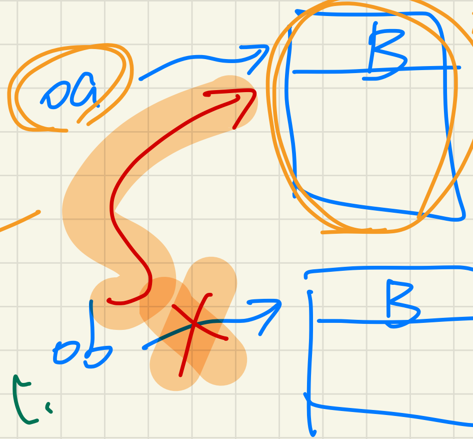
↳ compiles ∵ down cast

↳ CCE ∵ DT of oa is not desc. of C. not allowed!



```
A oa = new B();
B ob = new B();
C oc = new C();
ob = oa;    oa = oc; ✓
```

not valid ∵ ST of oa (A) is not des. of ST of ob (B)



((C) obj).

cast expression has static type C.

expectations on C.
all attributes and methods declared in ancestors of C.

# Generic Parameters: ArrayList

```
class ArrayList<E> {
  boolean add(E e)
  E remove(int index)
  E get(int index)
}
```

declare a type parameter to be used in ArrayList class

list1.add(new Point(3, 4)); X

String s = list2.remove(2); X

# Caller of ArrayList

```
ArrayList<String> list1 = new ArrayList<String>();
ArrayList<Point> list2 = new ArrayList<Point>();
```

```
class ArrayList<~~E~~ String> {
  boolean add(~~E~~ String e)
  ~~E~~ String remove(int index)
  ~~E~~ String get(int index)
}
```

```
class ArrayList<~~E~~ Point> {
  boolean add(~~E~~ Point e)
  ~~E~~ Point remove(int index)
  ~~E~~ Point get(int index)
}
```

# LECTURE 22
MONDAY NOVEMBER 25

- REVIEW SESSIONS FOR EXAM

  SURVEY ON MOODLE

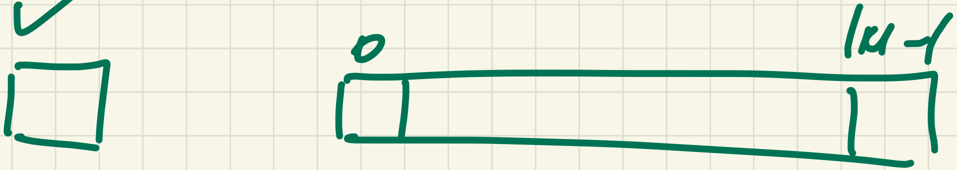
- MAKE-UP LECTURES:

  NOV. 15 } RECORDINGS
  NOV. 22

# Time Efficiency of Algo.

e.g. sort an array of integers

1. size ✓



2. Structure ✓

|   |   |   |   |
|---|---|---|---|
| 4 | 3 | 2 | 1 |

|   |   |   |   |
|---|---|---|---|
| 1 | 3 | 4 | 2 |

# Example Experiment

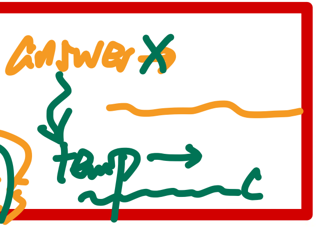
*Computational Problem*:

- **Input**: A character *c* and an integer *n*
- **Output**: A string consisting of *n* repetitions of character *c*
  e.g., Given input `'*'` and `15`, output `***************`.

*Algorithm 1* using *String* Concatenations:

```
public static String repeat1(char c, int n) {
  String answer = "";
  for (int i = 0; i < n; i ++) { answer += c; }
  return answer; }
```

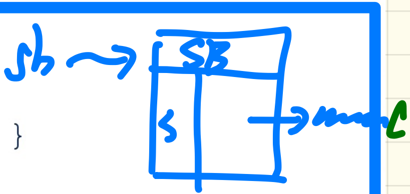
answer = answer + c;



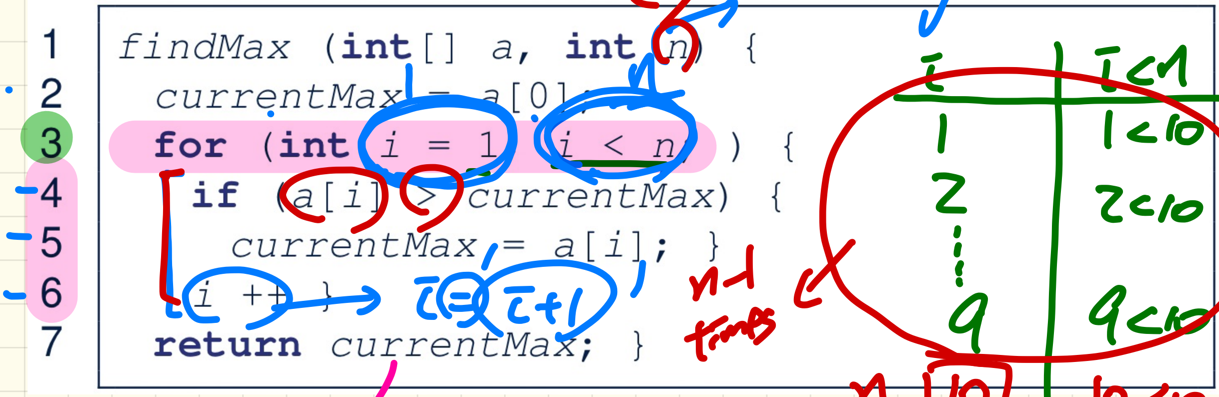
a

*Algorithm 2* using *StringBuilder* append's:

```
public static String repeat2(char c, int n) {
  StringBuilder sb = new StringBuilder();
  for (int i = 0; i < n; i ++) { sb.append(c); }
  return sb.toString(); }
```



# Counting the Number of Primitive Operations



```
1  findMax (int[] a, int n) {
2    currentMax = a[0];
3    for (int i = 1; i < n; ) {
4      if (a[i] > currentMax) {
5        currentMax = a[i]; }
6      i ++ }
7    return currentMax; }
```

10 == a.length

i ++ → i = i+1

n-1 times

| i | i<n | |
|---|---|---|
| 1 | 1<10 | T |
| 2 | 2<10 | T |
| ⋮ | | |
| 9 | 9<10 | T |
| n 10 | 10<10 | F |

**Q.** # of times **i < n** in **Line 3** is executed? → n times.

**Q.** # of times **loop body** (**Lines 4 to 6**) is executed? n-1 times

n-1 times i<n T

nth time i<n F

2·(n-1)

# Po : 7n-2

```
findMax(int[] a, int n){
      ⋮
}
```

$\cancel{6n^2 + 100}$ ·

↳ $\underline{T\,\boxed{n} - 2}$

| $n$ | # Ro<br>RT |
|---|---|
| 10 | 68 |
| 100 | 698 |
| ⋮ | |

RT.

$$\underbrace{(7n-2)}_{\text{\# of Pos}} \cdot \cancel{t} \qquad \underbrace{(10n+3)}_{\text{\# of Pos}} \cdot \cancel{t}$$

relative running time.

RT.

$$2^{n^{0}} + 4n^{\overset{\checkmark}{3}} + 2n^{\overset{\checkmark}{2}} + 3n^{1}$$

$$\boxed{\cancel{4}\,\boxed{n^{3}} + \cancel{2n^{2}} + \cancel{3n^{1}} + \cancel{1\,2^{0}}}$$

$$\boxed{n^{3}} \downarrow$$

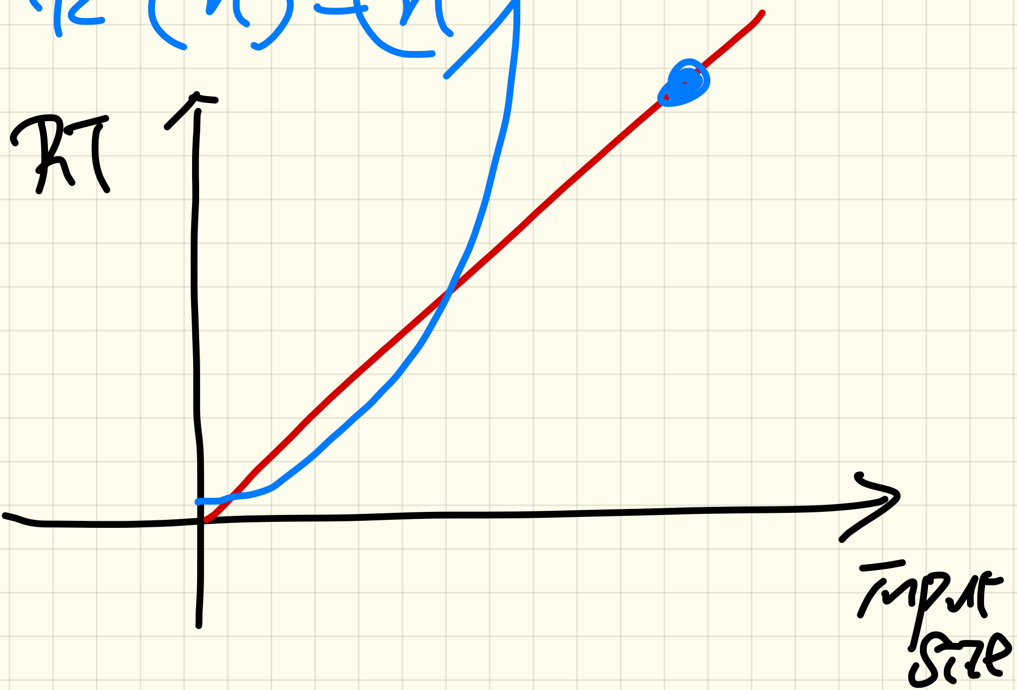
asymptotic upper bound.

multiplicative constants.

higest power (dominating over all lower terms)

$RT_1(n) = n^1$

$RT_2(n) = n^2$



$$RT_1(n) = \cancel{4}\underline{n^2} + \cancel{7n} + \cancel{16}$$

$$RT_2(n) = \cancel{100}\underline{n^2} + \cancel{5n} - \cancel{100}$$

# Asymptotic Upper Bound: Big-O

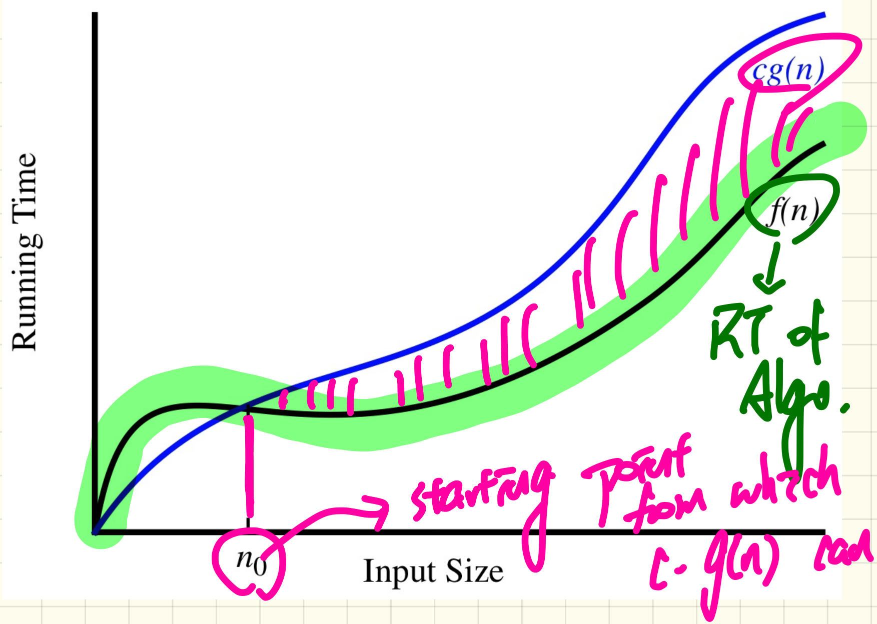
member of

$f(n) \in O(g(n))$ if there are:

- A real constant $c > 0$
- An integer constant $n_0 \geq 1$

such that:

$$f(n) \leq c \cdot g(n) \quad \text{for } n \geq n_0$$



RT of Algo.

starting point from which $c \cdot g(n)$ can upper bound $f(n)$

**Example:**

$f(n) = 8n + 5$

$g(n) = n$

**Prove:**

$f(n)$ is $O(\ g(n)\ )$

**Choose:**

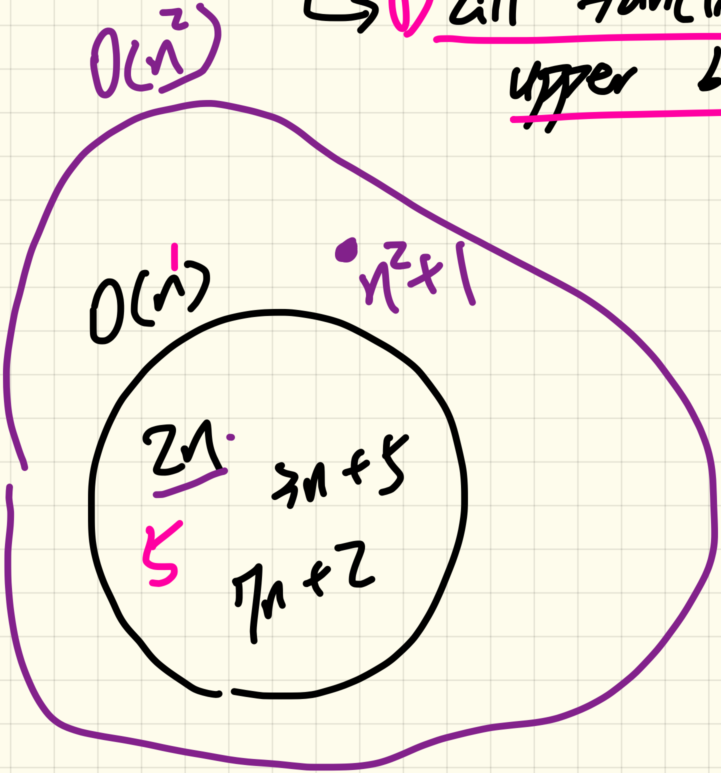
$c = 9$

What about n0?

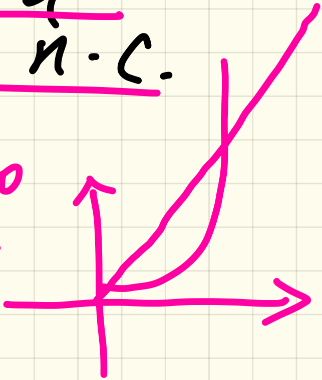
$O(n)$

↳ ① all functions that can be upper bounder by $n \cdot c$.

② all functions whose highest power $\leq 1$



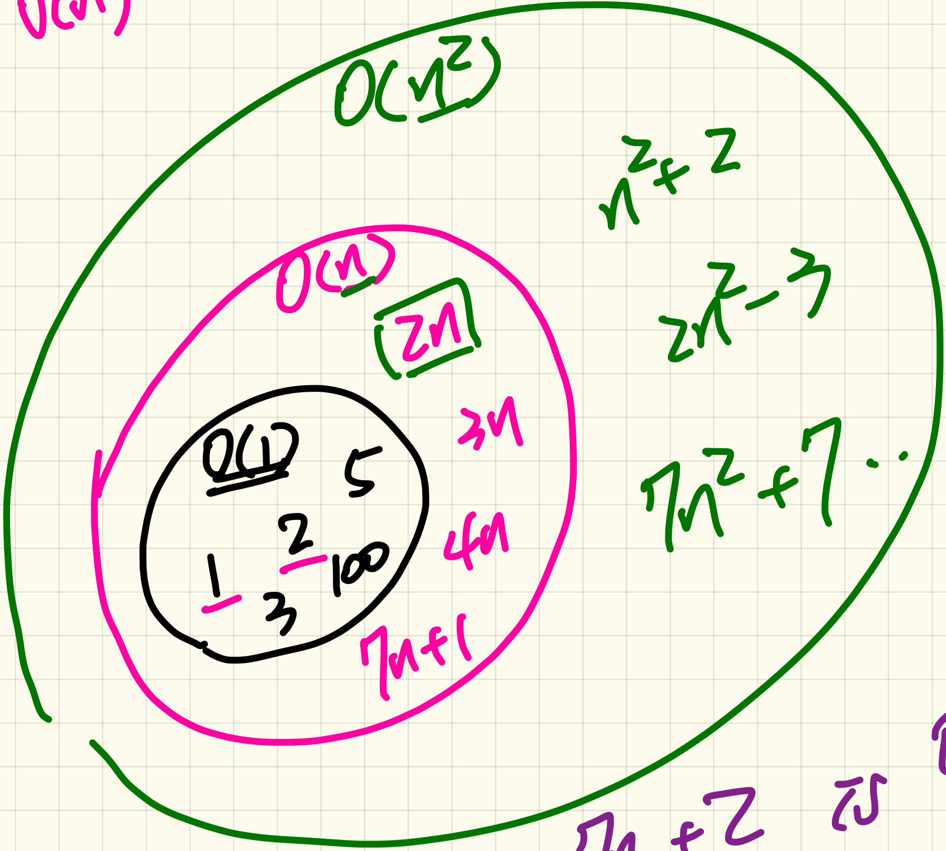
$5 \cdot n^0$



why $n^2+1 \notin O(n)$.

↳ to prove, choose $c$ s.t. $c \cdot n \geq n^2+1$

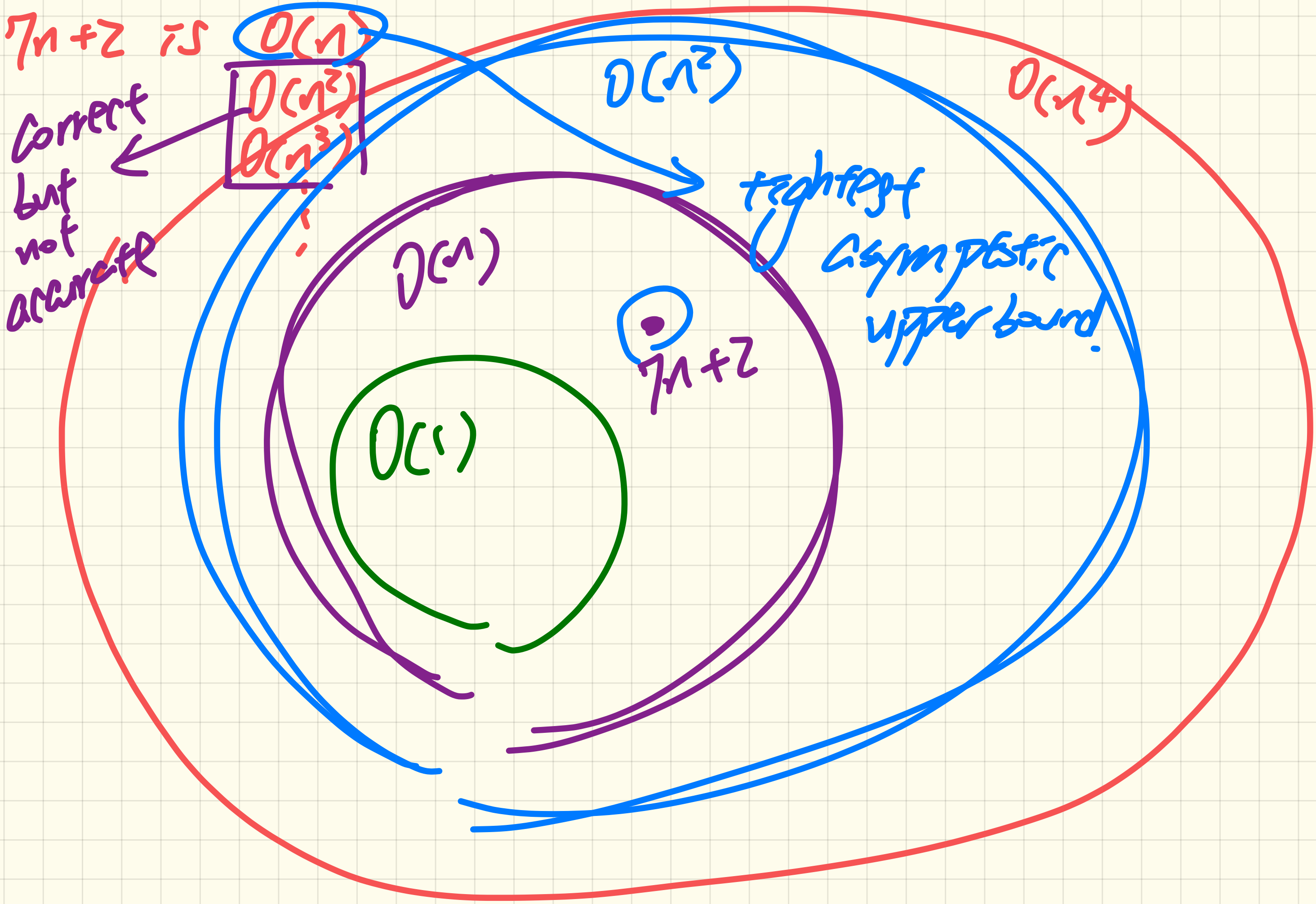
$O(1) \subset O(n)$



$7n+2$ is $O(1)$ ✗

$7n+2$ is $O(n)$ ✓

$7n+2$ is $O(n^2)$


7n+2 is $O(n)$
$O(n^2)$
$O(n^3)$
correct but not accurate
$O(n^2)$
$O(n^4)$
tightest asymptotic upper bound.
$O(n)$
7n+2
$O(1)$

$RT(n) = \boxed{8}\underline{n} + \boxed{5}$

$\in O(\underline{n})$

$13 \geq 13$.

↳ choose $\underline{c} = \cancel{7}\ 8 + 5 = 13$

s.t. $\boxed{n_0} = \cancel{7}\ \underline{1}$

↳ starting from $n \geq 1$

$13 \cdot n \geq 8n' + 5$

$$\boxed{5}n^2 + \boxed{3}n \cdot \log n + \boxed{2}n + \boxed{5}$$

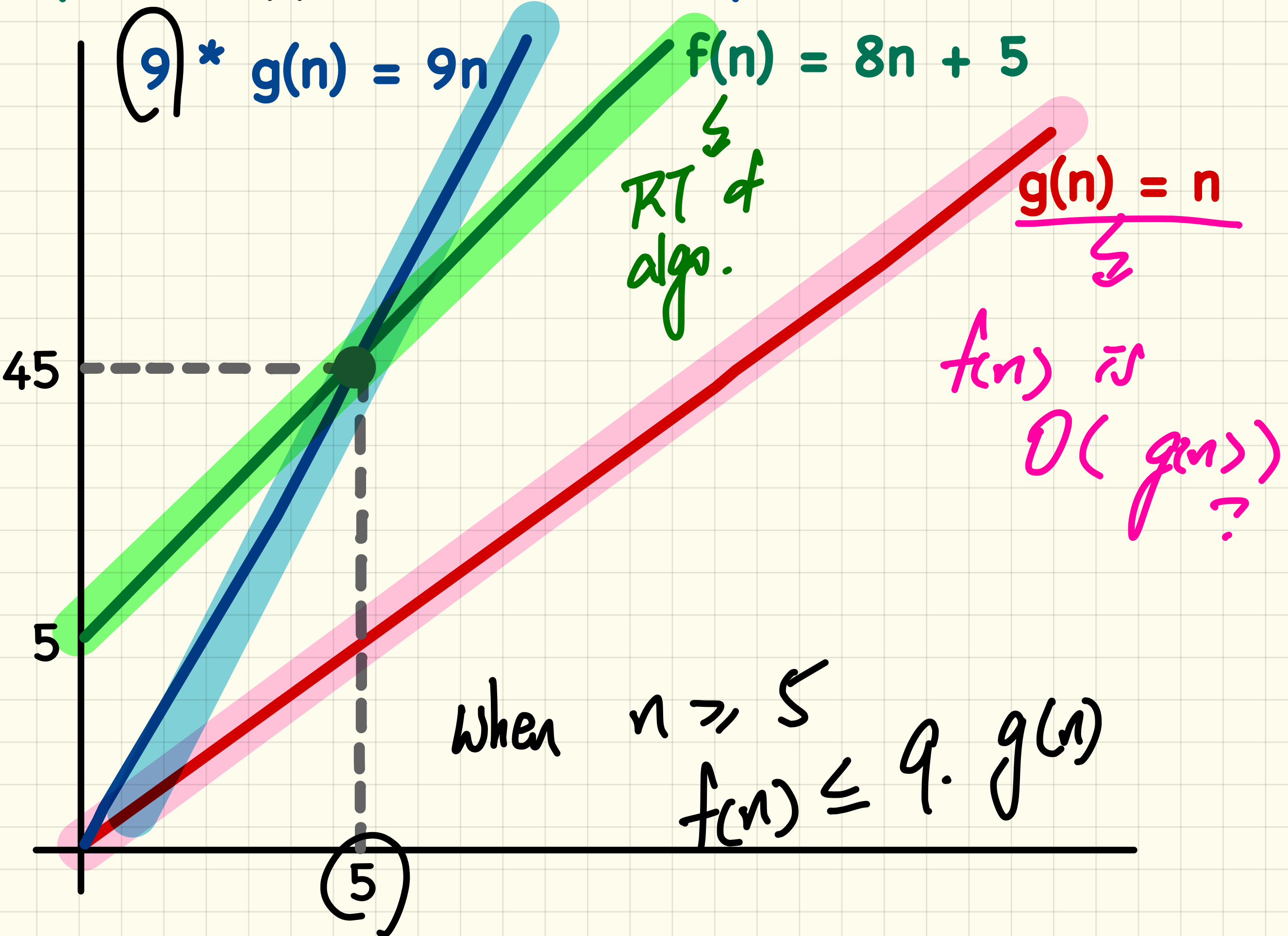
↳ $O(n^2)$

Prove. choose $c = 15$

check. starting from $n = n_0$:

$$15 \cdot n^2 \geq 5n^2 + 3n \cdot \log n + 2n + 5$$

# Asymptotic Upper Bound: Example



f(n) is O( g(n) ) ?

when $n \geq 5$

$f(n) \leq 9 \cdot g(n)$

# Lecture 23

Wednesday November 27

# Asymptotic Upper Bound: Big-O

alg. → a set of functions

$f(n) \in O(g(n))$ if there are:

- A real constant $c > 0$
- An integer constant $n_0 \geq 1$

such that:

$$f(n) \leq c \cdot g(n) \text{ for } n \geq n_0$$

upper bound effect

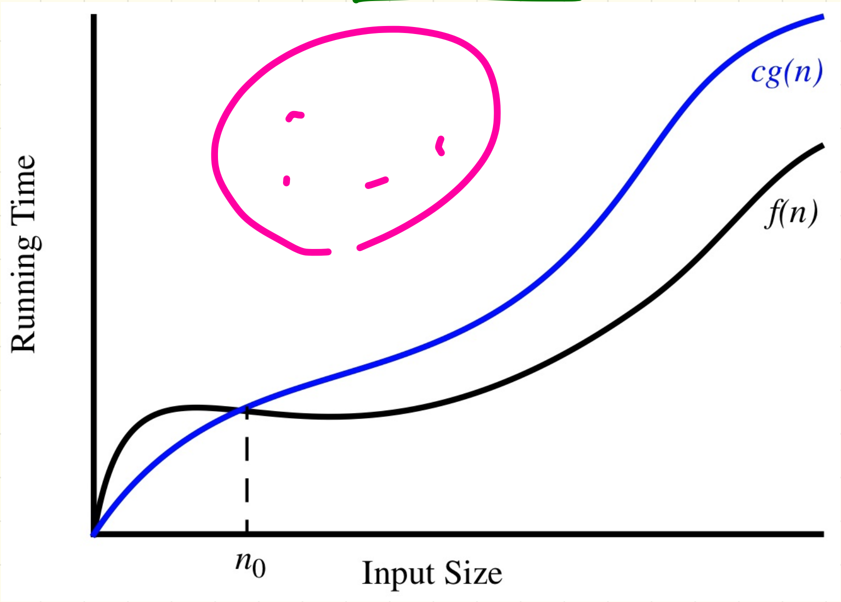
Example:

$f(n) = 8n + 5$

$g(n) = n$

Prove:

$f(n)$ is $O(g(n))$

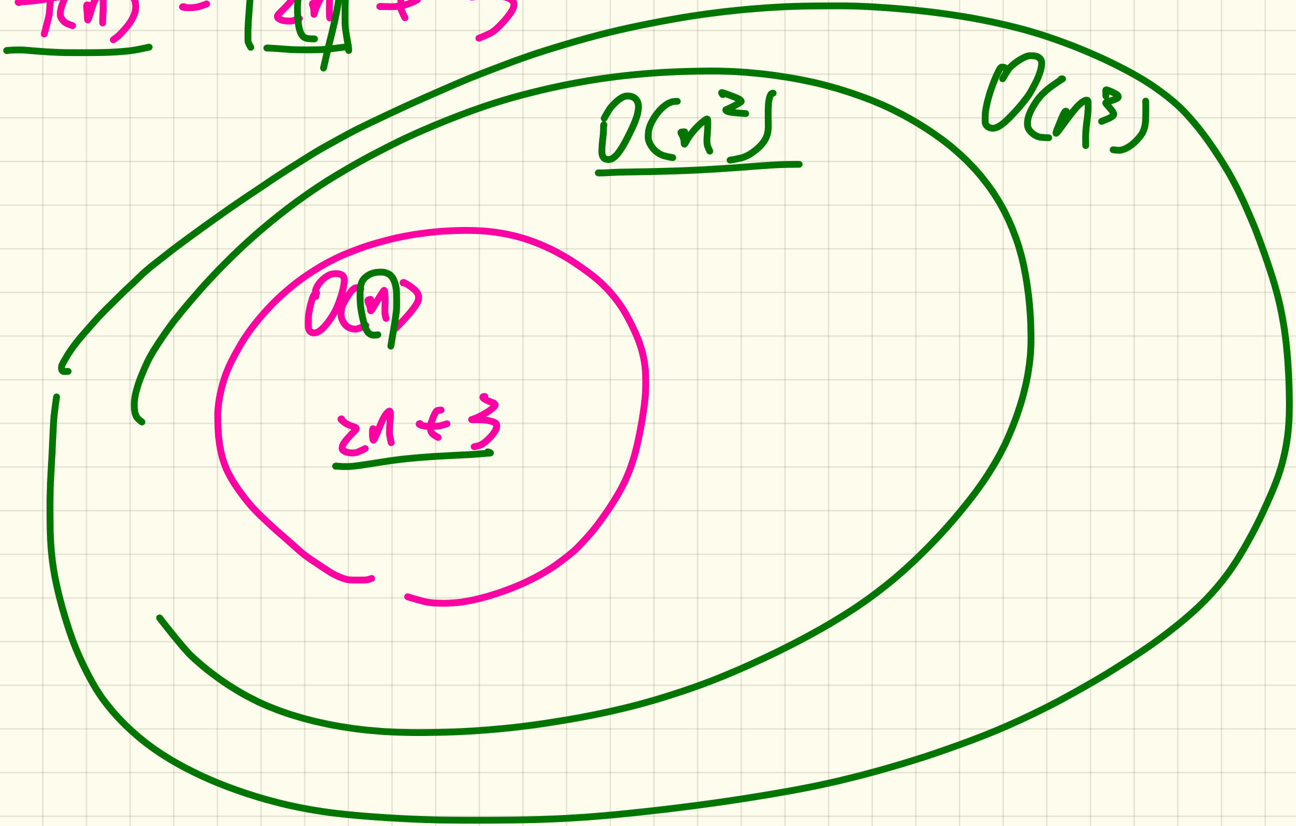


Choose:

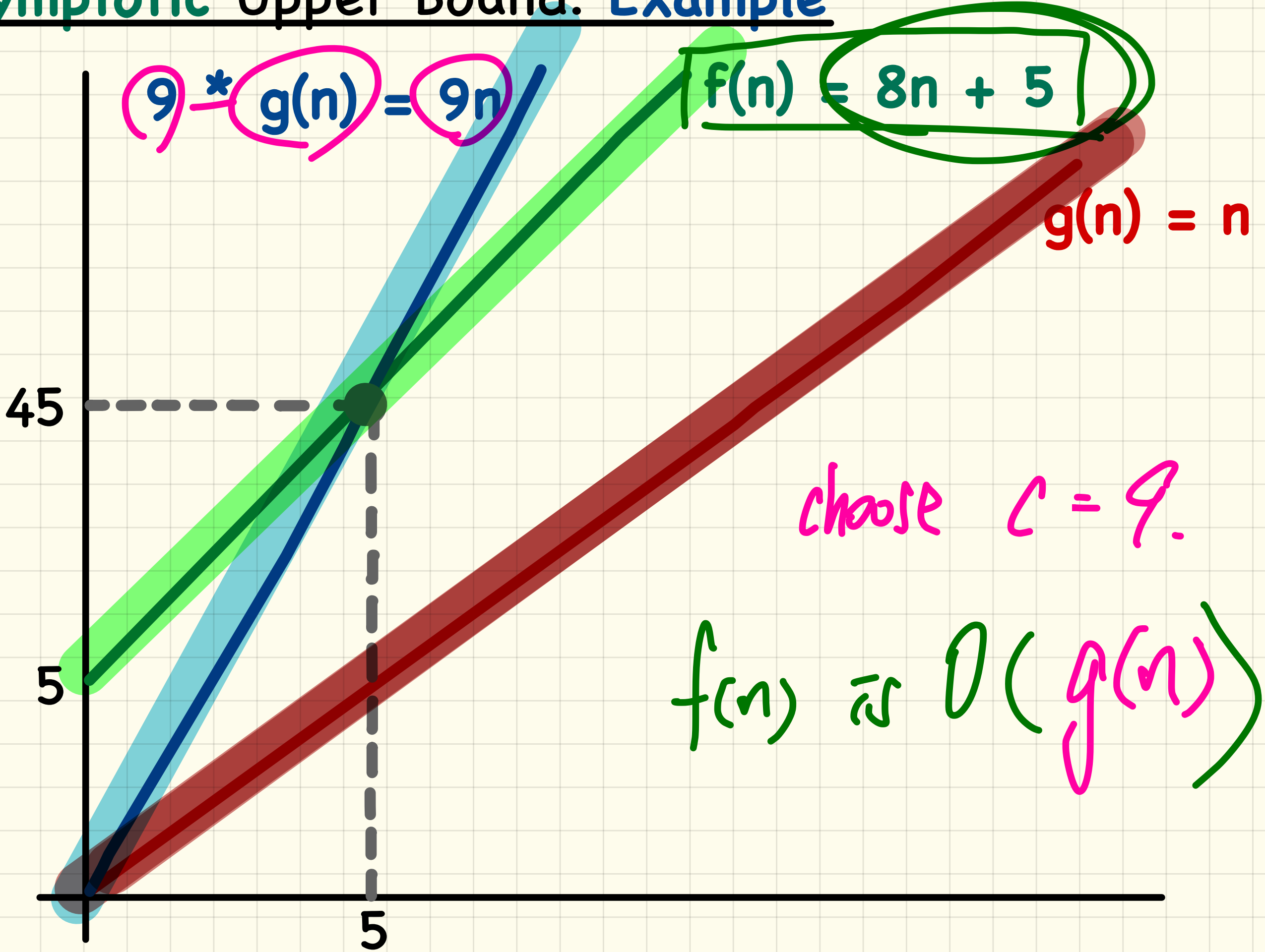
c = 9

What about n0?

$f(n) = \boxed{2n} + 3$



# Asymptotic Upper Bound: Example



choose $c = 9$.

$f(n)$ is $O(g(n))$

# Proving f(n) is O( g(n) )

$2n^2 - 3n - 7$ $\quad |2| + |-3| + |-7| = 12$

If $f(n)$ is a polynomial of degree $d$, i.e.,

$$f(n) = a_0 \cdot n^0 + a_1 \cdot n^1 + \cdots + a_d \cdot n^d$$

highest power

and $a_0, a_1, \ldots, a_d$ are integers (i.e., negative, zero, or positive), then $f(n)$ is $O(n^d)$.

~~$O(n^{d+1})$~~

$c \cdot n^d$

We prove by choosing

$$c = |a_0| + |a_1| + \cdots + |a_d|$$
$$n_0 = 1$$

Upper-bound effect starts when $n_0 = 1$? $[f(1) \le 1^d]$

$$a_0 \cdot 1^0 + a_1 \cdot 1^1 + \cdots + a_d \cdot 1^d$$
$$= a_0 + a_1 + \cdots + a_d$$

$f(1) \le |a_0| + |a_1| + \cdots + |a_d|$.

$c \cdot$

Upper-bound effect holds? $[f(n) \le n^d]$ $\quad 1^d$

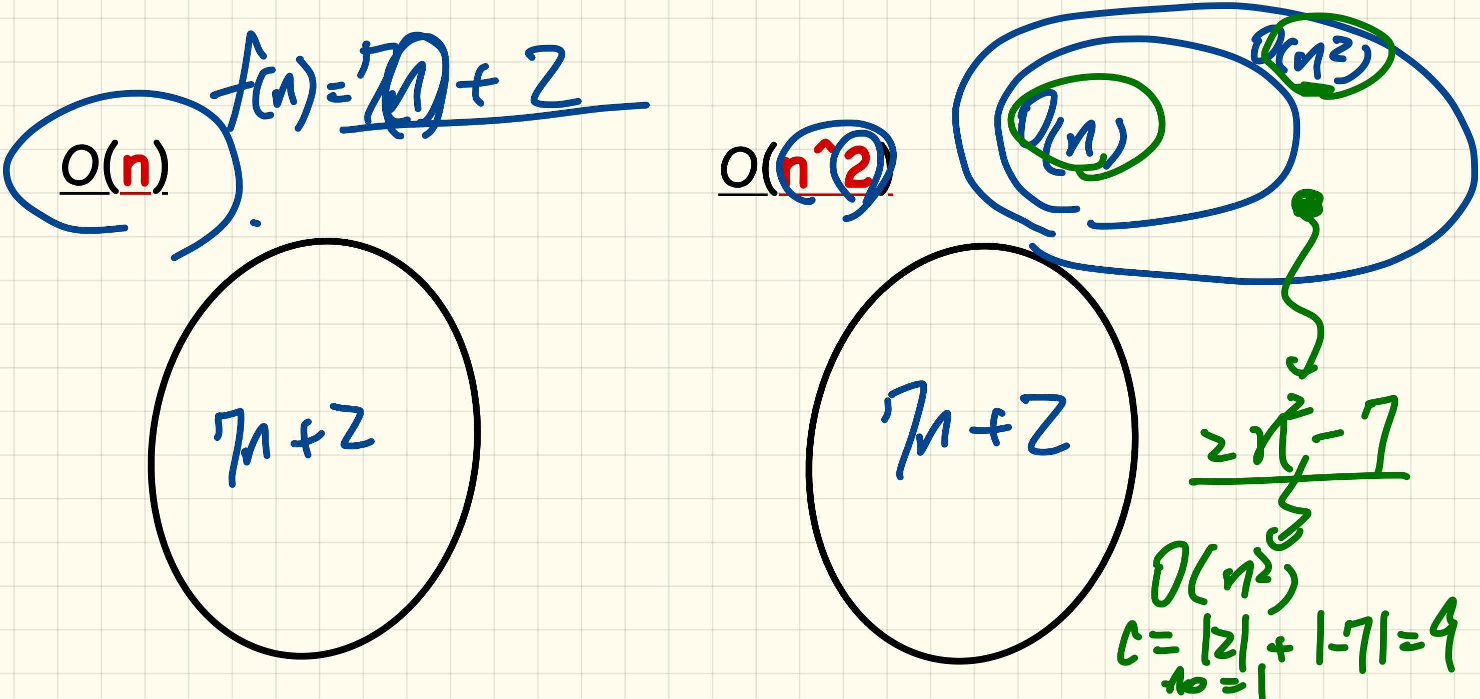
$$2 + (-3) + (-7) \overset{\checkmark}{\le} |2| + |-3| + |-7|$$

# O( g(n) ): A Set of Functions

Each member f(n) in O( g(n) ) is such that:

Higest Power of f(n) <= Highest Power of g(n)

$f(n) = 7n + 2$



$2n^2 - 7$

$O(n^2)$

$c = |2| + |-7| = 9$

$n_0 = 1$

# Asymptotic Upper Bounds: Example (1)

$$\boxed{5n^2 + 3n \cdot logn + 2n + 5 \text{ is } O(n^2)}$$

$\log_2 1 = 0$

$2^0 = 1$

**Prove.**

choose $c = ?\ 5 + 3 + 2 + 5 = 15$

$n_0 = ?\ \boxed{1}$ s.t.

$$15 \cdot \underset{1}{\cancel{n}^2} \geq 5\overset{}{\cancel{n}}^2 + 3n \cdot \log \overset{1}{\cancel{n}} + 2\overset{1}{\cancel{n}} + 5 \checkmark$$

$$15 \geq \underbrace{5 + 0 + 2 + 5}_{12}$$

# Asymptotic Upper Bounds: Example (2)

$20n^3 + 10n \cdot logn + 5$ is $O(n^3)$

Prove.

choose $c =$ ~~?~~ $35$

$n_0 =$ ~~?~~ $1$ s.t.

$$c \cdot \cancel{n^3} \geq 20n^3 + 10n \cdot \log n + 5$$

✓

$$35 \cdot 1 \geq 20 \cdot 1^3 + 0 + 5$$

# Asymptotic Upper Bounds: Example (3)

$3 \cdot logn + 2$ is $O(logn)$

Prove:

choose $c = \cancel{?}\ 5$

$n_0 = \cancel{?}\ \cancel{1}\ 2$ s.t.

$$\boxed{c \cdot \log n \geq 3 \cdot \log n + 2}$$

$$5 \cdot \underbrace{\log 1}_{0} \geq 3 \cdot \underbrace{\log 1}_{0} + 2$$

$$0 \geq 2 \quad \times$$

$$5 \cdot \underbrace{\log 2}_{1} \geq 3 \cdot \underbrace{\log 2}_{1} + 2$$

$$5 \geq 3 + 2 \quad ✓$$

# Asymptotic Upper Bounds: Example (4)

$$\boxed{2^{n+2} \text{ is } O(2^n)}$$

$$2^{n+2} = 2^2 \cdot 2^n$$

Prove

choose $c = \cancel{??} \cdot 2^2 = 4$

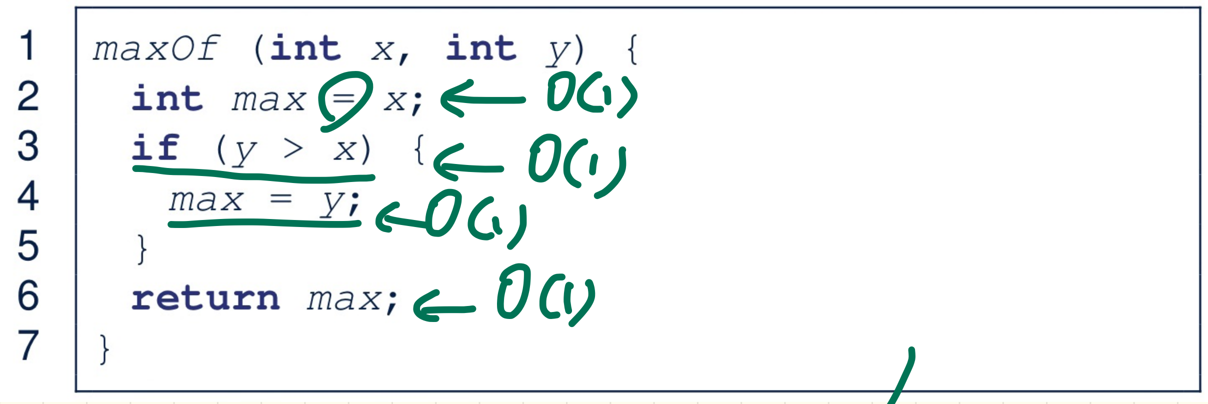
$n_0 = 1$ ? s.t.

$$c \cdot 2^{\overset{1}{\cancel{n}}} \geq 2^{\overset{1}{\cancel{n}}+2}$$

# Asymptotic Upper Bounds: Example (5)

$$\boxed{2n + 100 \cdot log n \text{ is } O(n)}$$

# Determining the Asymptotic Upper Bound (1)

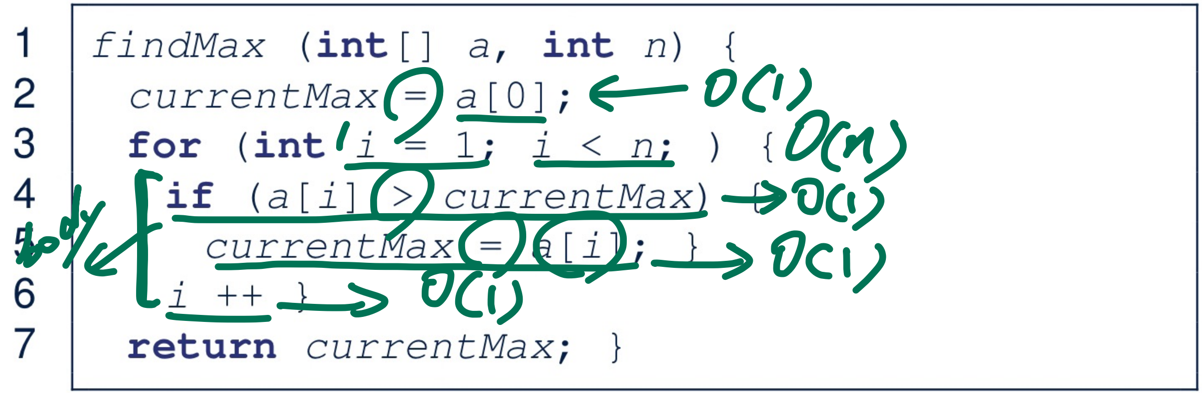


```
maxOf (int x, int y) {
  int max = x;        ← O(1)
  if (y > x) {        ← O(1)
    max = y;          ← O(1)
  }
  return max;         ← O(1)
}
```

O(1).

RT does not depend on how large x and y are.

# Determining the Asymptotic Upper Bound (2)



```
findMax (int[] a, int n) {
  currentMax = a[0];
  for (int i = 1; i < n; ) {
    if (a[i] > currentMax) {
      currentMax = a[i]; }
    i ++ }
  return currentMax; }
```

→ 1 → O(1)

→ 2 → O(1)

⋮

→ n-1 → O(1)

$O(\underbrace{1+1+\cdots+1}_{n})$

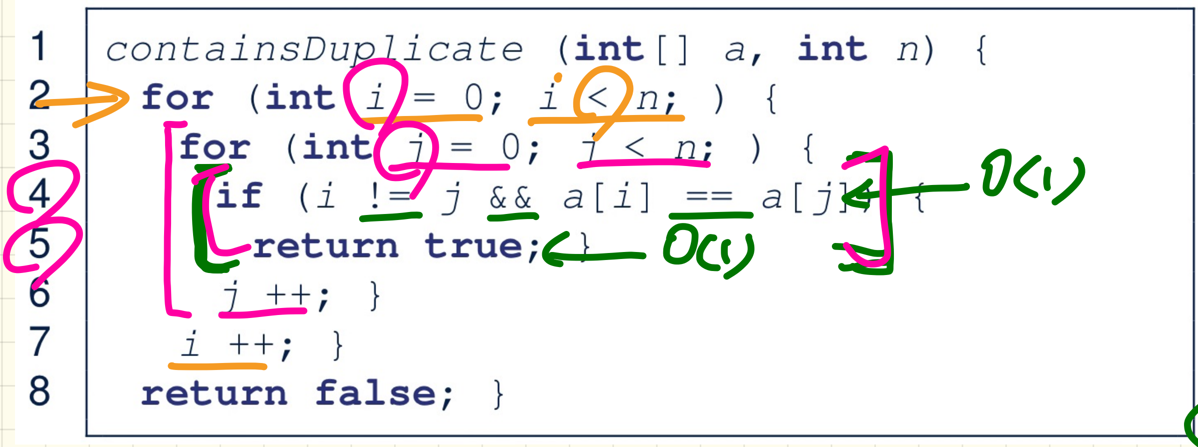
↳ $O(n)$.

Will a double-nested loop give RT = $O(n^2)$ ? No.

$O(1)$

```
O(n)  for (int i = 0 ; i < ~~n~~ 10 ; i++){
O(1)      for (int j = 5 ; j < 10 ; j++){
              print( . _ )
          }
      }
```

# Determining the Asymptotic Upper Bound (3)



```
1  containsDuplicate (int[] a, int n) {
2    for (int i = 0; i < n; ) {
3      for (int j = 0; j < n; ) {
4        if (i != j && a[i] == a[j]) {
5          return true; }
6        j ++; }
7      i ++; }
8    return false; }
```

$O(1)$

$O(1)$

$j = 0$

$\vdots$

$n-1$

$[i, j]$

$\boxed{j - i + 1}$

observe the pattern of changes on loop counter

$i = 0\ [\ j = 0 \ldots n-1\ ]\ O(n)$

$O(1)$

$1\ [\ j = 0 \ldots n-1$

$2\ [\ j = 0 \ldots n-1$

$\vdots$

$n-1\ [\ j = 0 \ldots n-1$

$n$

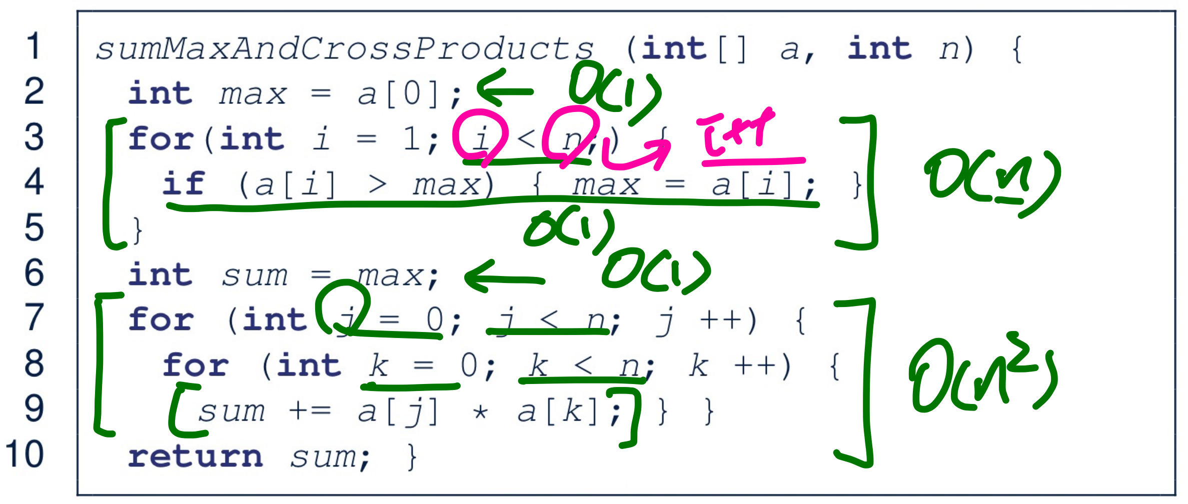
$O(n \cdot n \cdot 1) = O(n^2)$

# of iterations for j

# of iterations for i

RT for L4 L5

# Determining the Asymptotic Upper Bound (4)



```
sumMaxAndCrossProducts (int[] a, int n) {
  int max = a[0];
  for(int i = 1; i < n; i++) {
    if (a[i] > max) { max = a[i]; }
  }
  int sum = max;
  for (int j = 0; j < n; j ++) {
    for (int k = 0; k < n; k ++) {
      sum += a[j] * a[k]; } }
  return sum; }
```

$\leftarrow O(1)$

i++

$O(1)$

$O(n)$

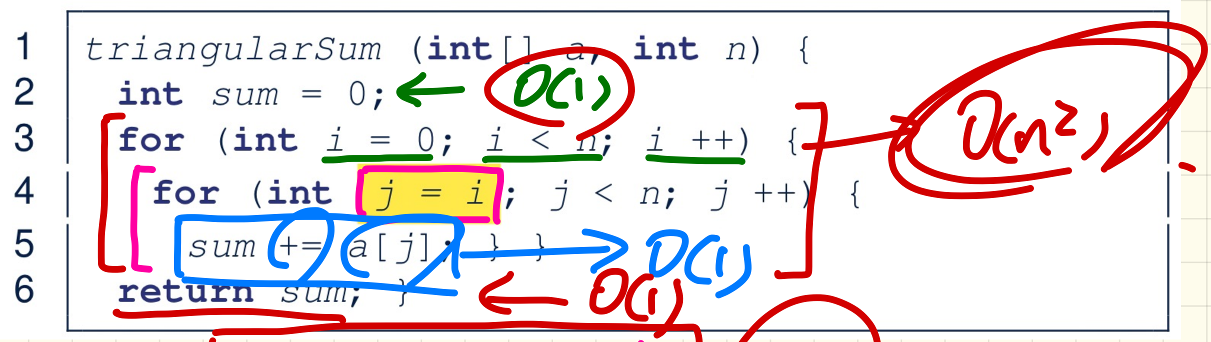
$\leftarrow O(1)$

$O(n^2)$

$$O(1 + n + 1 + n^2)$$

$$= O(n^2)$$

# Determining the Asymptotic Upper Bound (5)



```
triangularSum (int[] a, int n) {
  int sum = 0;
  for (int i = 0; i < n; i ++) {
    for (int j = i; j < n; j ++) {
      sum += a[j]; } }
  return sum; }
```

$O(1)$

$O(n^2)$

$O(1)$

$O(1)$

| | | |
|---|---|---|
| $i = 0$ | $j = 0 \,..\, n-1$ <br> $O(1)$ $O(1)$ | $n$ |
| $1$ | $j = 1 \,..\, n-1$ | $n-1$ |
| $2$ | $j = 2 \,..\, n-1$ | $n-2$ |
| $\vdots$ | | |
| $n-1$ | $j = n-1 \,..\, n-1$ | $1$ |

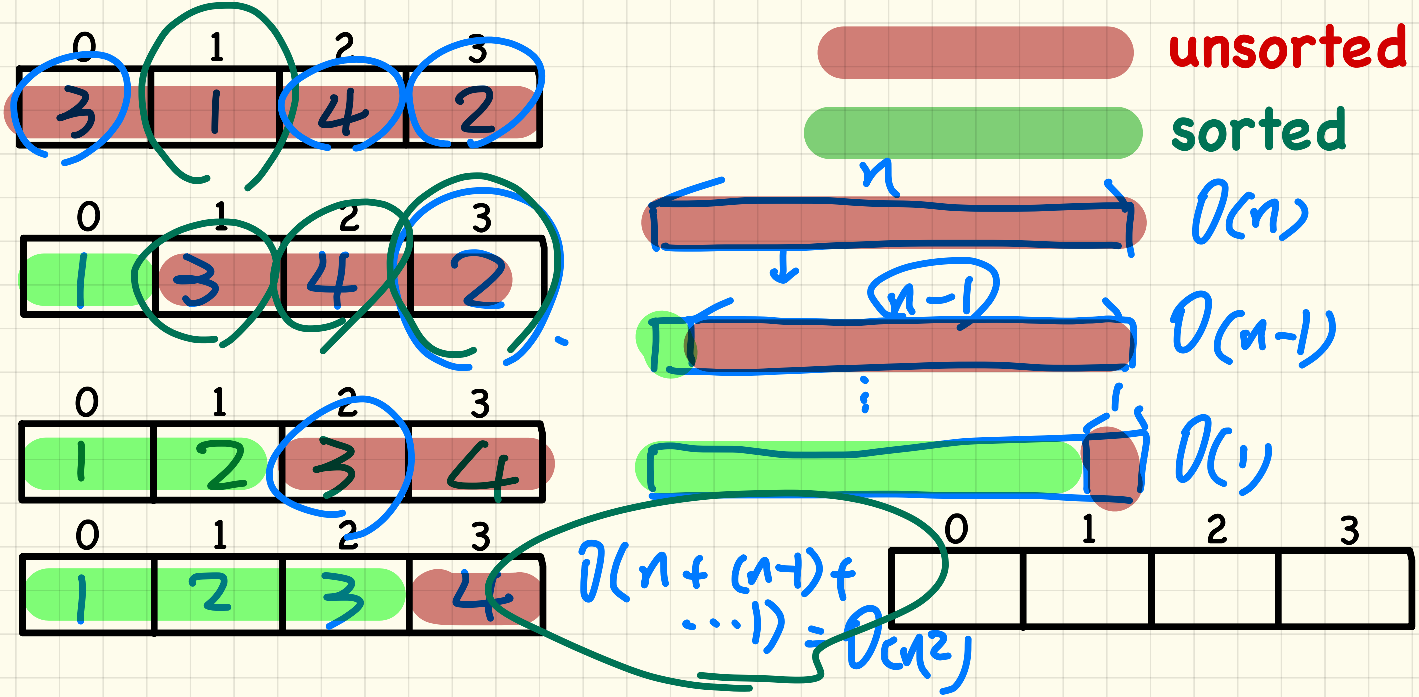
$$n + (n-1) + \dots + 1$$

$$= \frac{(n+1) * n}{2}$$

$$= O(n^2)$$

# Selection Sort

Keep **selecting** minimum from the **unsorted** portion and appending it to the end of **sorted** portion.



| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 3 | 1 | 4 | 2 |

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 3 | 4 | 2 |

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 2 | 3 | 4 |

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 2 | 3 | 4 |

| 0 | 1 | 2 | 3 |
|---|---|---|---|
|  |  |  |  |

🟥 unsorted

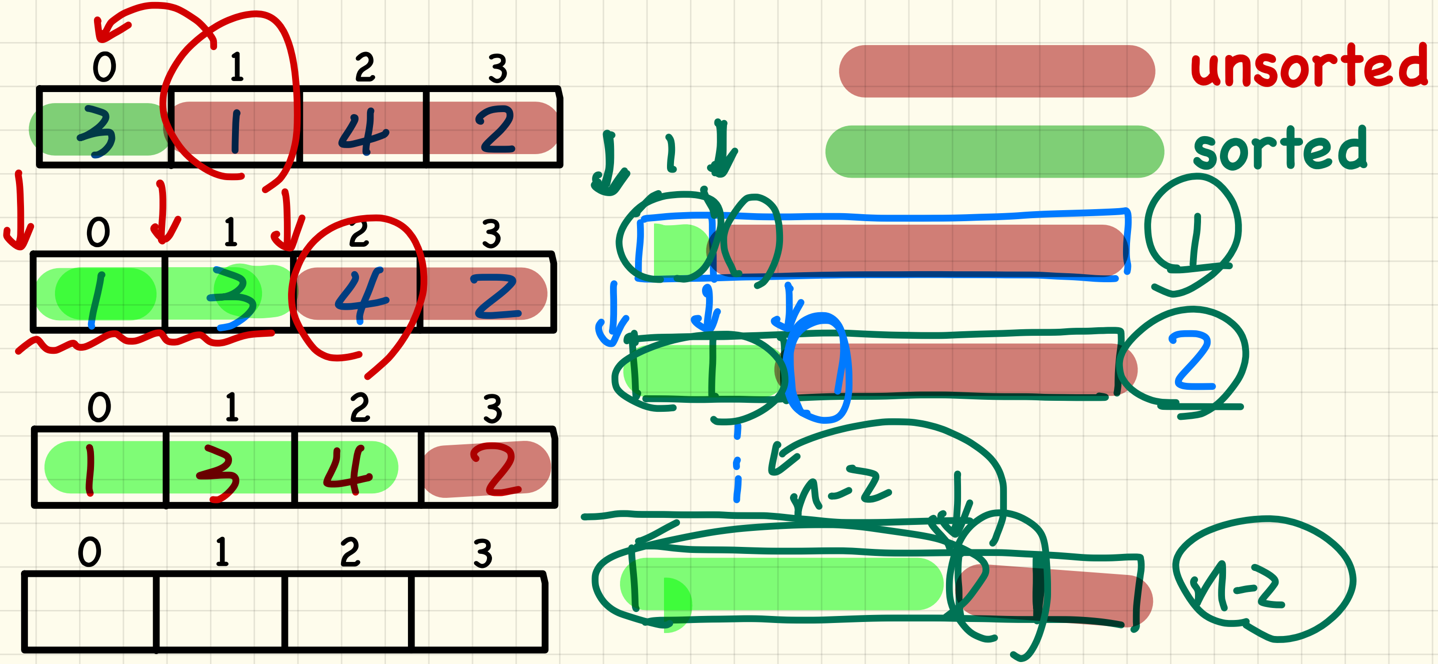
🟩 sorted

$O(n + (n-1) + \dots 1)) = O(n^2)$

# Insertion Sort

$O(1+2+\cdots(n-2)) = O(n^2)$

Keep getting 1st element from the unsorted portion and inserting it to the sorted portion.

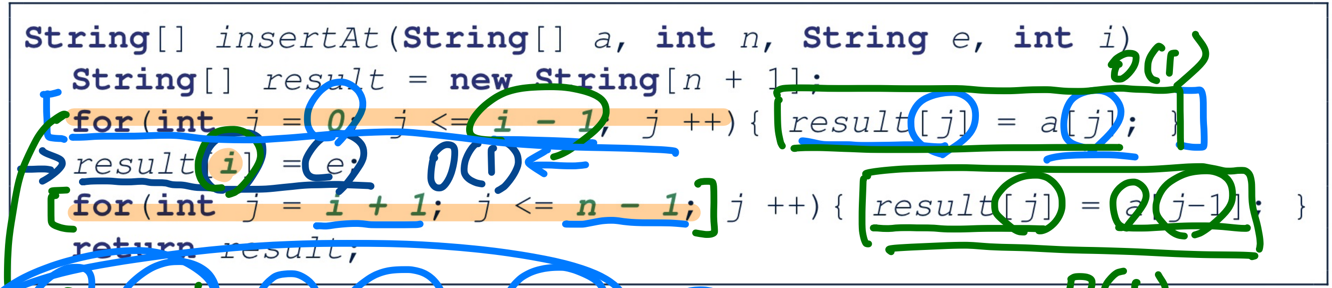


# LECTURE 24

MONDAY DECEMBER 2

# Inserting into an Array

$O(n+1) = O(n)$



```
String[] insertAt(String[] a, int n, String e, int i)
  String[] result = new String[n + 1];
  for(int j = 0; j <= i - 1; j ++){ result[j] = a[j]; }
  result[i] = e;
  for(int j = i + 1; j <= n - 1; j ++){ result[j] = a[j-1]; }
  return result;
```

$O(1)$

$O(1)$

$O(1)$

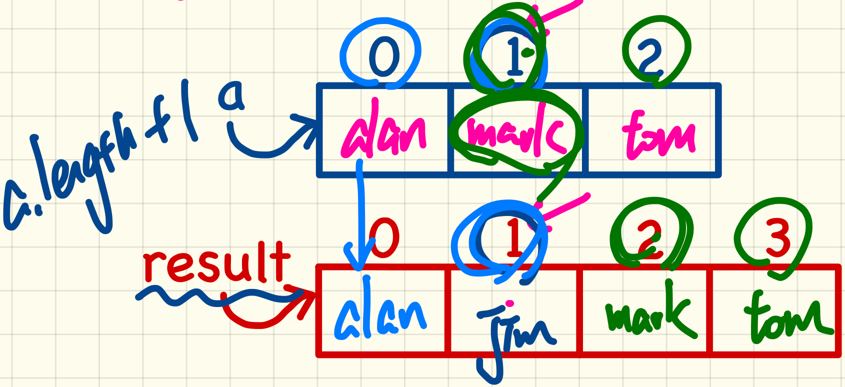
$0 .. i-1$   $i$   $i+1 .. n-1$   $n$

$((n-1)-(i+1)+1)$

$i+1$ f

**Example:**

insertAt({alan, mark, tom}, 3, jim, 1)



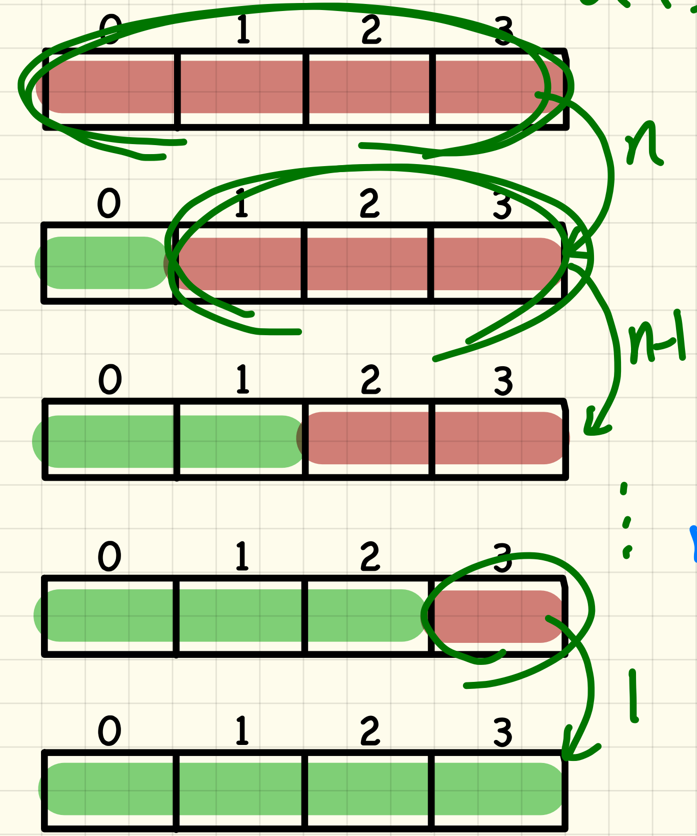
result[2] = a[1]

[3] = a[2]

$O(n + (n-1) + (n-2) + \dots + 1)$

## Selection Sort

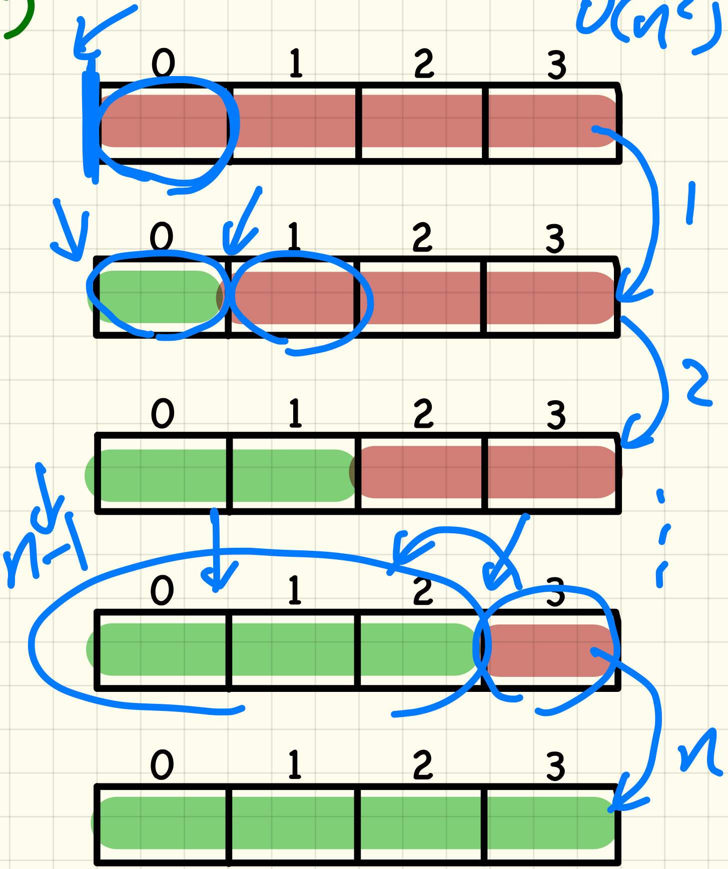
$O(n^2)$



$O(1 + 2 + 3 + \dots n)$

## Insertion Sort

$= O(n^2)$



$O(n^2)$

Input size $\boxed{1000}$

$\hookrightarrow (1000)^2 = \underline{\boxed{1M}}$ .

$O(n \cdot \log n)$

$\hookrightarrow 1000 \cdot \log 1000 = \boxed{1000}$

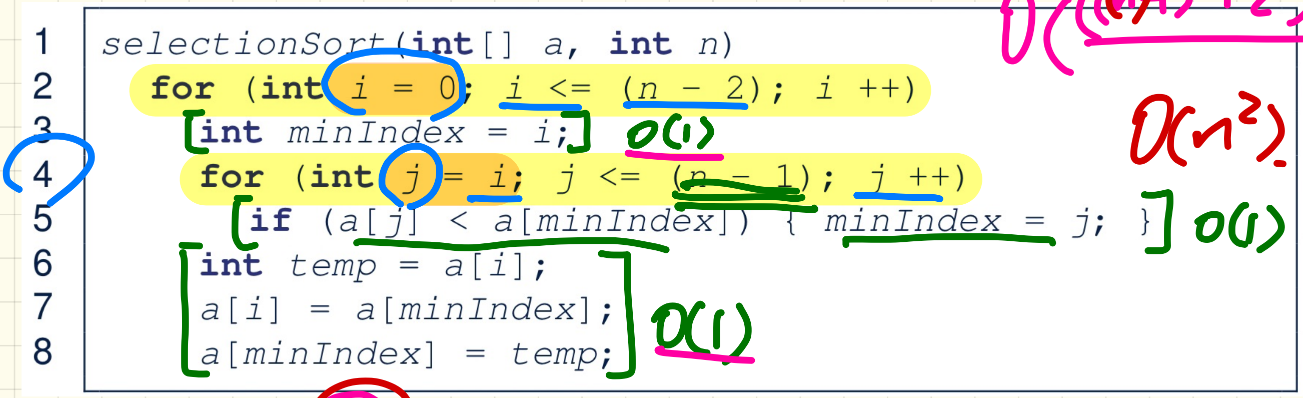
$1000 \cdot 10 =$

# Selection Sort in Java

$O((n-1) + (n-2) + \cdots + 2)$

$O\left(\frac{((n-1)+2) * (n-2)}{2}\right)$

$O(n^2)$



```
selectionSort(int[] a, int n)
  for (int i = 0; i <= (n - 2); i ++)
    int minIndex = i;
    for (int j = i; j <= (n - 1); j ++)
      if (a[j] < a[minIndex]) { minIndex = j; }
    int temp = a[i];
    a[i] = a[minIndex];
    a[minIndex] = temp;
```

$O(1)$

$O(1)$

$O(1)$

$i = 0$ $j = 0\ 1\ ..\ n-1$ ] $n-1$

$1$ $j = 1\ 2\ \cdots\ n-1$ ] $n-2$

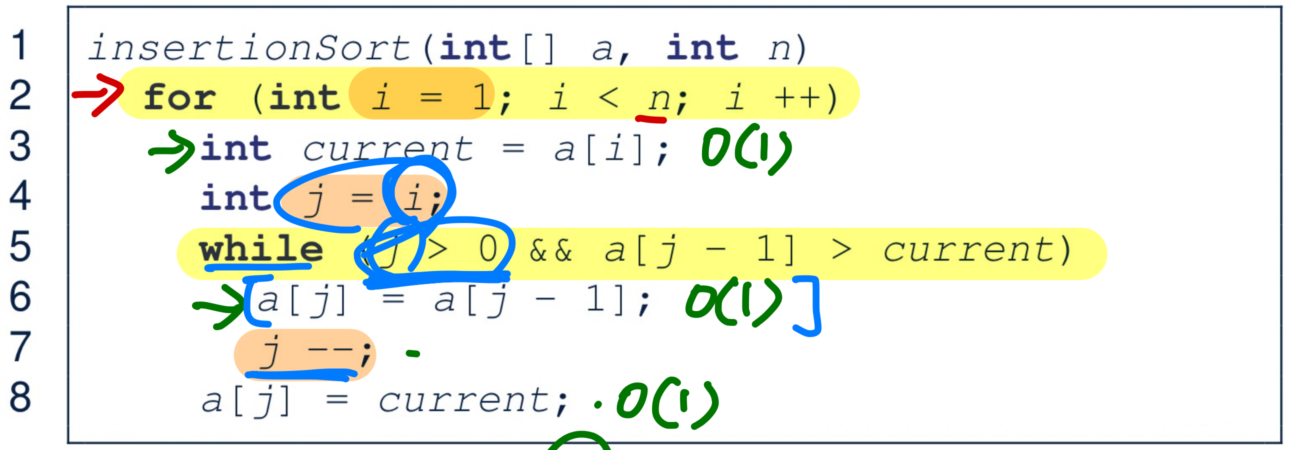
$2$ $j = 2\ 3\ \cdots\ n-1$ ] $n-3$

$\vdots$

$n-2$ $j = n-2\ ..\ n-1$ ] $2$

$n-1$

# Insertion Sort in Java



```
insertionSort(int[] a, int n)
  for (int i = 1; i < n; i ++)
    int current = a[i]; O(1)
    int j = i;
    while (j > 0 && a[j - 1] > current)
      [a[j] = a[j - 1]; O(1)]
      j --;
    a[j] = current; O(1)
```

| | | | | | |
|---|---|---|---|---|---|
| i = 1 | j = 1 | | | | |
| 2 | j = 2 | 1 | | | |
| 3 | j = 3 | 2 | 1 | | |
| n-1 | j = n-1 | n-2 | n-3 | . . . . | 1 |

$O(1 + 2 + 3 + \dots +$

$= O\left(\frac{(1 + (n-1)) \times (n-1)}{2}\right)$ (n-1))

$O(n^2)$

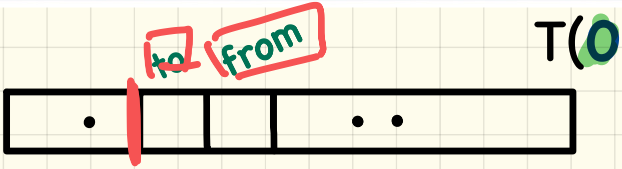
# Running Time: Ideas

running time ← T( t ) → input size.

```
boolean allPositive(int[] a) { return allPosH (a, 0, a.length - 1); }
boolean allPosH (int[] a, int from, int to) {
  if (from > to) { return true; }
  else if (from == to) { return a[from] > 0; }
  else { return a[from] > 0 && allPosH (a, from + 1, to); } }
```

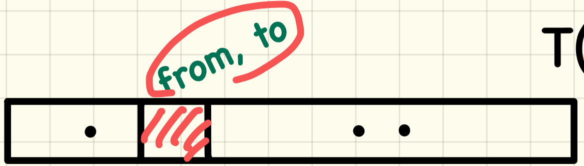
input size

**Base Case:**
**Empty Array**



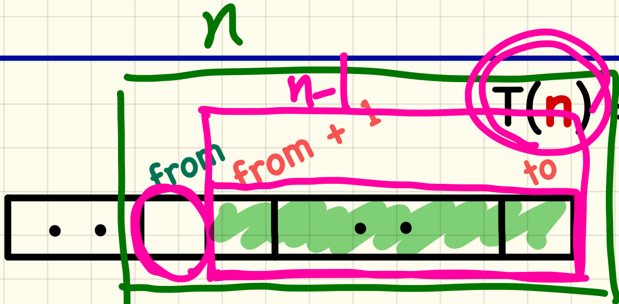
$T(0) = 1$

**Base Case:**
**Array of Size 1**



$T(1) = 1$

**Recursive Case:**
**Array of size > 1**



$T(n) = T(n - 1) + 1$

# Running Time: Unfolding Recurrence Relation

$T(0) = 1$

$T(1) = 1$

$\checkmark T(\cancel{n}) = T(\cancel{n} - 1) + 1$

$n-1 \qquad n-1$

$n-2 \qquad n-2$

$$T(n) = T(n-1) + 1$$

$$= \underbrace{(\boxed{T(n-2)} + 1)}_{T(n-1)} + 1$$

$$= (((T(n-3) + 1) + 1) + 1$$

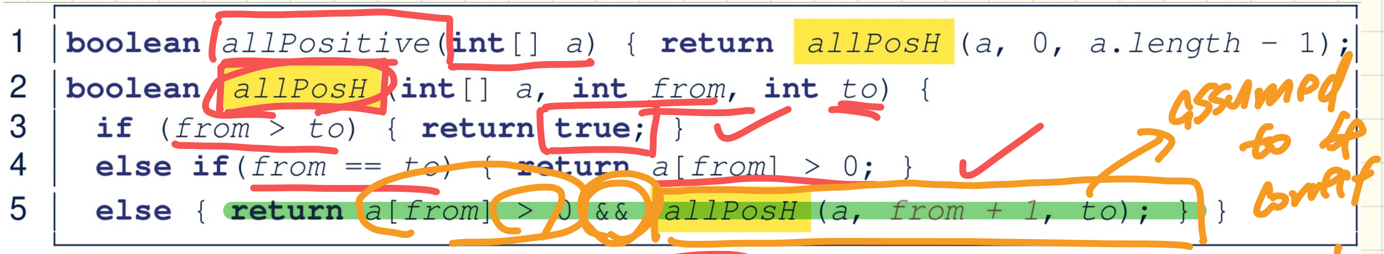
$$= \cdots$$

$$= \underbrace{(T(0) + 1)}_{T(1)} + \cdots + 1 + 1 + \underbrace{1}_{n \text{ terms}}$$

$T(n-n)$

$O(n)$.

# Correctness Proofs: Ideas



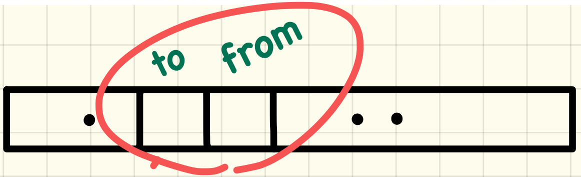
```
boolean allPositive(int[] a) { return allPosH(a, 0, a.length - 1);
boolean allPosH(int[] a, int from, int to) {
  if (from > to) { return true; }
  else if(from == to) { return a[from] > 0; }
  else { return a[from] > 0 && allPosH(a, from + 1, to); } }
```

assumed to be correct

I.H.

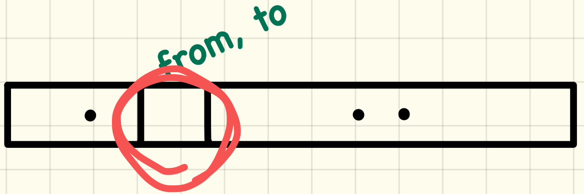
**Base Case:**

**Empty Array**



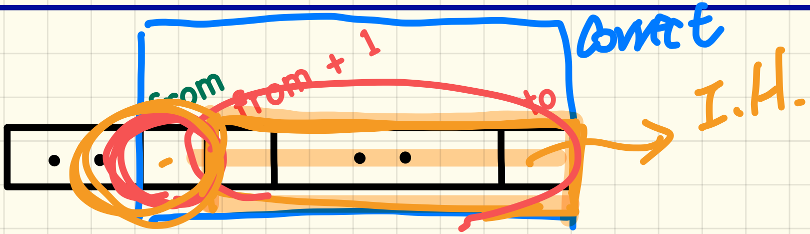
**Base Case:**

**Array of Size 1**

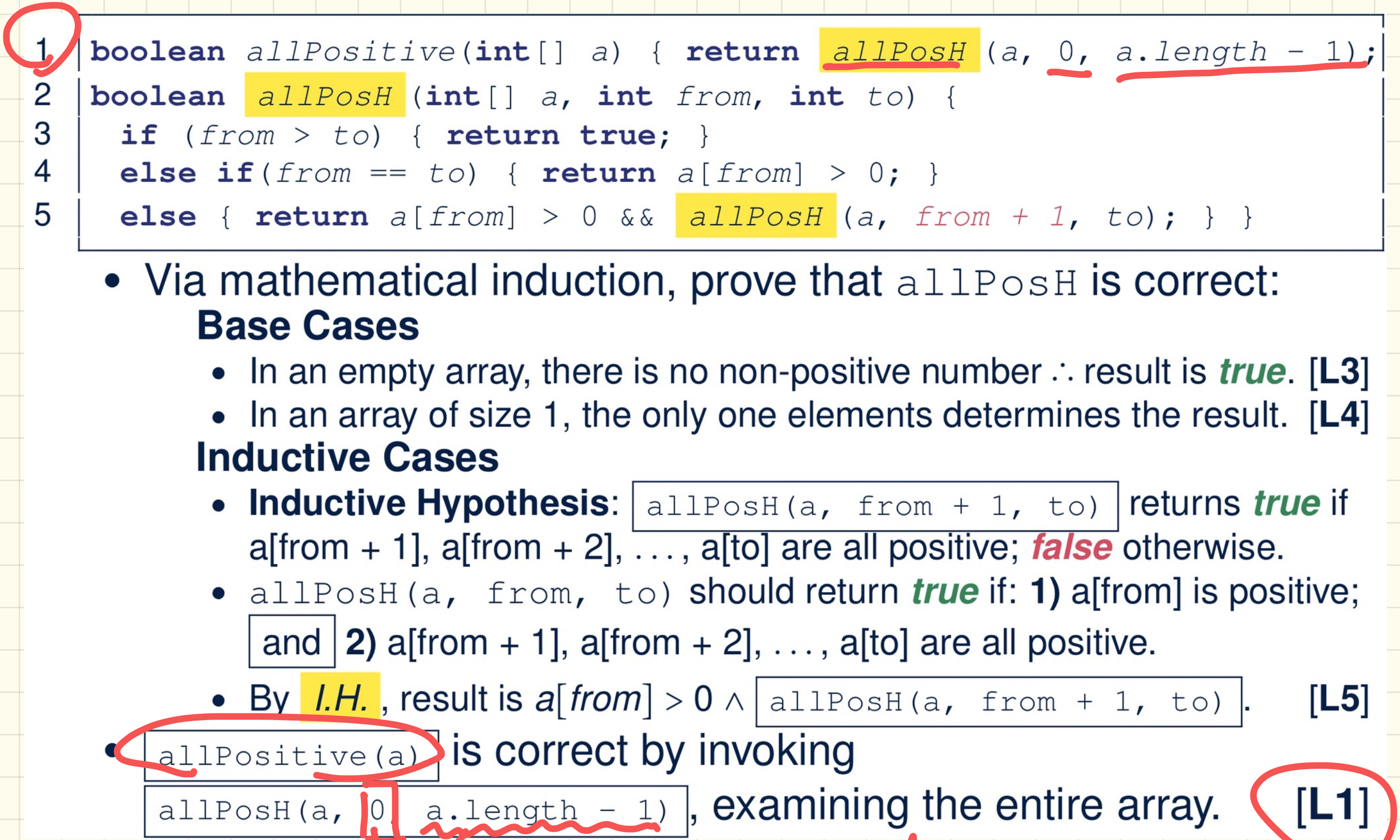


**Recursive Case:**

**Array of size > 1**



# Correctness Proofs



```
1 boolean allPositive(int[] a) { return allPosH (a, 0, a.length - 1);
2 boolean allPosH (int[] a, int from, int to) {
3   if (from > to) { return true; }
4   else if(from == to) { return a[from] > 0; }
5   else { return a[from] > 0 && allPosH (a, from + 1, to); } }
```

- Via mathematical induction, prove that `allPosH` is correct:

  **Base Cases**

  - In an empty array, there is no non-positive number ∴ result is ***true***. **[L3]**
  - In an array of size 1, the only one elements determines the result. **[L4]**

  **Inductive Cases**

  - **Inductive Hypothesis**: `allPosH(a, from + 1, to)` returns ***true*** if a[from + 1], a[from + 2], . . . , a[to] are all positive; ***false*** otherwise.
  - `allPosH(a, from, to)` should return ***true*** if: **1)** a[from] is positive; and **2)** a[from + 1], a[from + 2], . . . , a[to] are all positive.
  - By *I.H.*, result is $a[from] > 0 \wedge$ `allPosH(a, from + 1, to)`. **[L5]**
- `allPositive(a)` is correct by invoking `allPosH(a, 0, a.length - 1)`, examining the entire array. **[L1]**

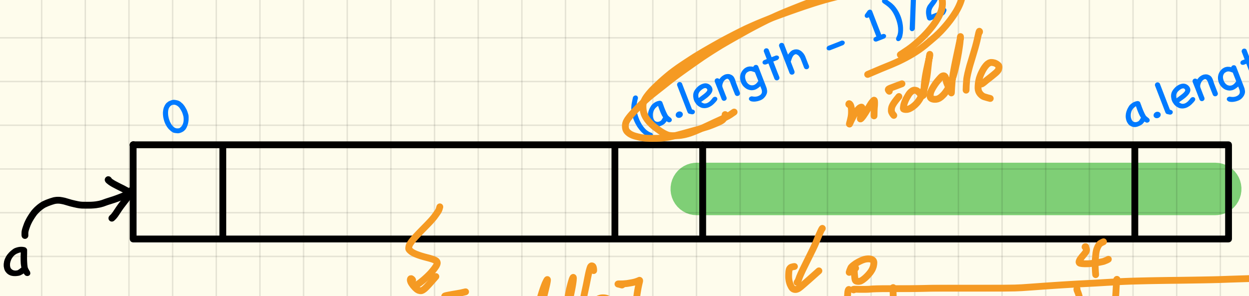
Entire range of array.

# Binary Search: Ideas

search(k)

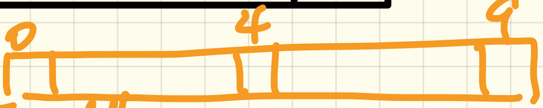
**Input**: Array sorted in **non-descending** order

1 2 4 6 7 8 - -



< a[middle]

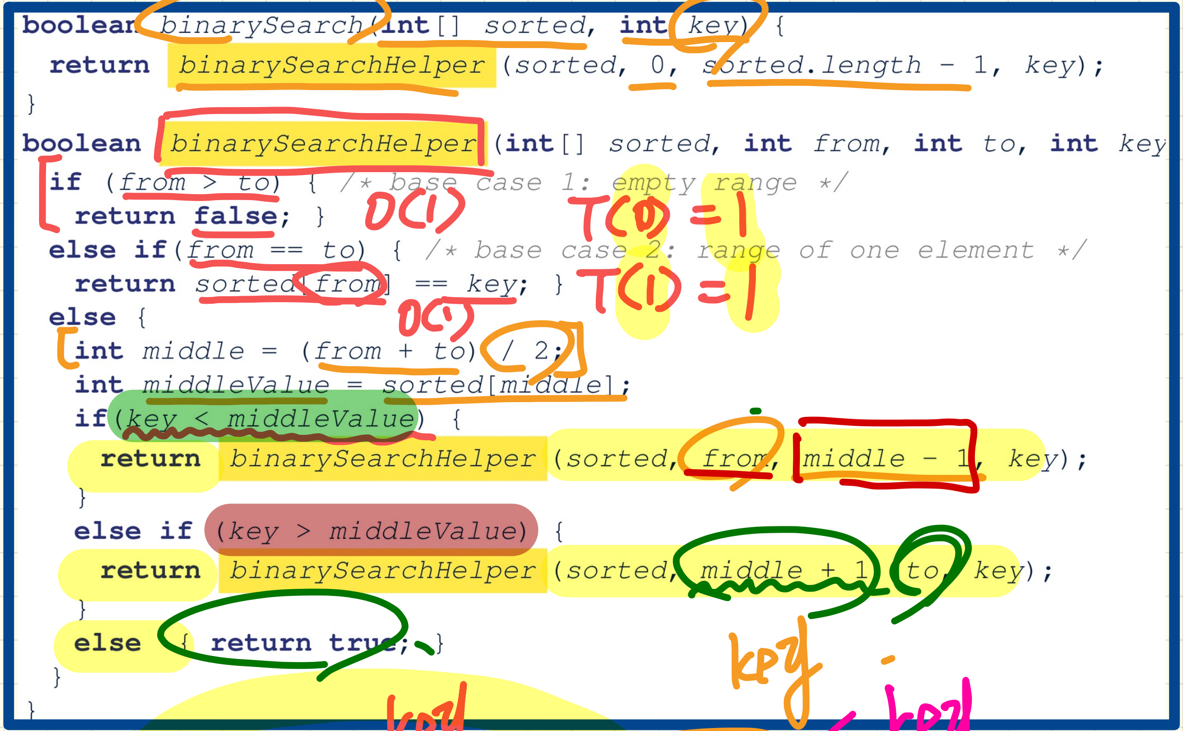
> a[middle]



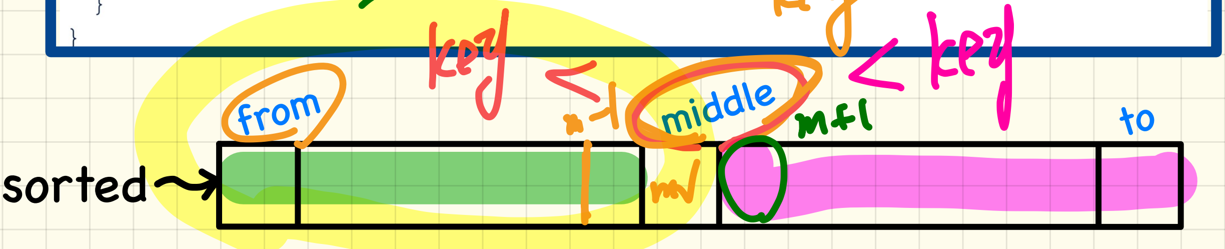
**Search**: Does key **k** exist in array **a**?

$\boxed{k > a[middle]}$

# Binary Search in Java



```
boolean binarySearch(int[] sorted, int key) {
  return binarySearchHelper(sorted, 0, sorted.length - 1, key);
}
boolean binarySearchHelper(int[] sorted, int from, int to, int key
  if (from > to) { /* base case 1: empty range */
   return false; }
  else if(from == to) { /* base case 2: range of one element */
   return sorted[from] == key; }
  else {
   int middle = (from + to) / 2;
   int middleValue = sorted[middle];
   if(key < middleValue) {
    return binarySearchHelper(sorted, from, middle - 1, key);
   }
   else if (key > middleValue) {
    return binarySearchHelper(sorted, middle + 1, to, key);
   }
   else { return true; }
  }
}
```



# Binary Search: Tracing

m

0 1 2 3 4 5 6 7 8

Say a = {3,6,9,12,15,18,21,24,27}

$18 > a[4]$

✓

search(a,18)

f t

search(a,0,8,18)

$m = (0+8)/2 = 4$

search(a,5,8,18)

$m = (5+8)/2 = 6$

$18 < a[6]$

search(a,5,5,18)

0 1 2 3 4 5 6 7 8

Say a = {3,6,9,12,15,18,21,24,27}

search(a,7)

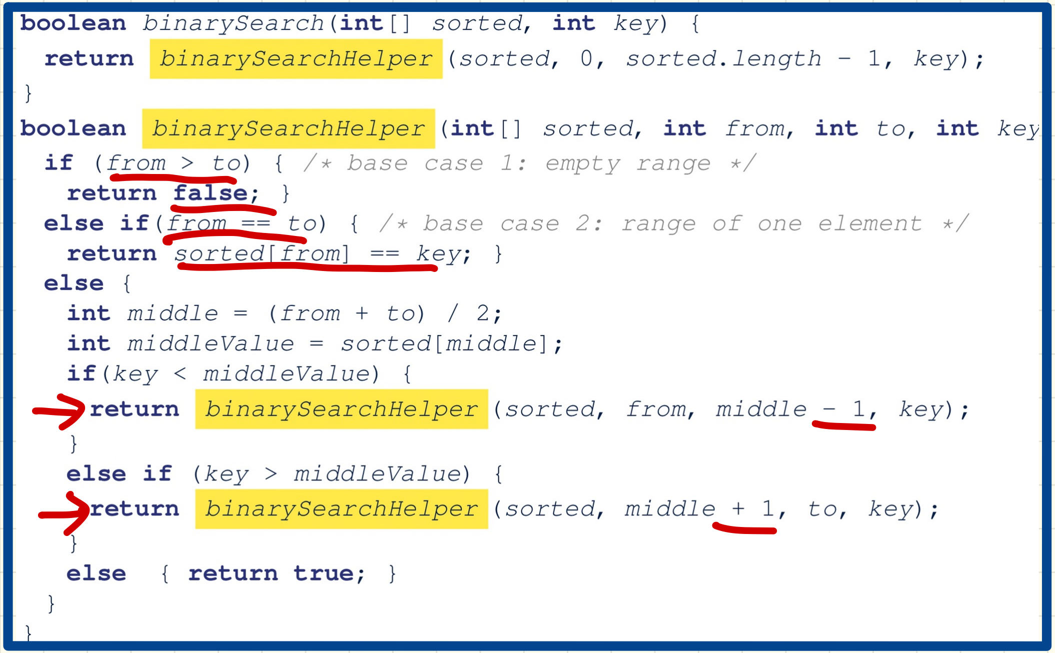
search(a,0,8,7)

search(a,0,3,7)

search(a,2,3,7)

search(a,2,1,7)

# Binary Search: Running Time



```
boolean binarySearch(int[] sorted, int key) {
  return binarySearchHelper(sorted, 0, sorted.length - 1, key);
}
boolean binarySearchHelper(int[] sorted, int from, int to, int key
  if (from > to) { /* base case 1: empty range */
    return false; }
  else if(from == to) { /* base case 2: range of one element */
    return sorted[from] == key; }
  else {
    int middle = (from + to) / 2;
    int middleValue = sorted[middle];
    if(key < middleValue) {
      return binarySearchHelper(sorted, from, middle - 1, key);
    }
    else if (key > middleValue) {
      return binarySearchHelper(sorted, middle + 1, to, key);
    }
    else { return true; }
  }
}
```

$T(0) = 1$

$T(1) = 1$

$T(n) = \underline{T(n/2)} + 1$

# Running Time: Unfolding Recurrence Relation

T(0) = 1

T(1) = 1

$T(\cancel{n}) = T(\cancel{n}/2) + 1$

$\frac{n}{2}$ $\frac{n}{2}$

$\frac{n}{4}$ $\frac{n}{4}$

$\frac{n}{2^{?}}$

$1 = \frac{n}{2^{\log n}}$

$O(\log n)$.

$$T(n) = T\left(\frac{n}{2^1}\right) + 1$$

$$= \left(T\left(\frac{n}{4}\right) + 1\right) + 1 \qquad 4 = 2^2$$

$$= \left(\left(T\left(\frac{n}{8}\right) + 1\right) + 1\right) + 1 \qquad 8 = 2^3$$

$$\vdots$$

$$1 \quad \frac{n}{2^{\log n}}$$

$$= T(1) + \underbrace{1 + \cdots + 1 + 1 + 1}_{\log n}$$

# EXAM REVIEW I

MONDAY DECEMBER 9

Selection Sort

Insertion Sort (

$O(n^2)$

Merge Sort

$O(n \cdot \log n)$

Arrays.sort .

1000 elements

$1000^2 = 1M$ ..

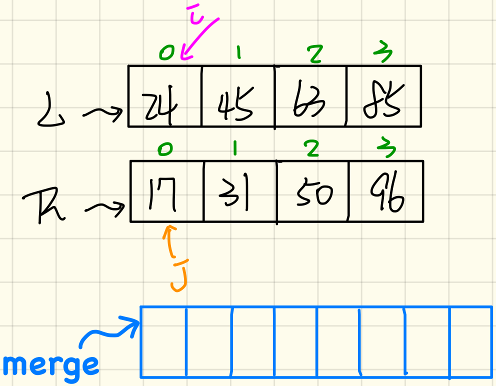
1000 elements

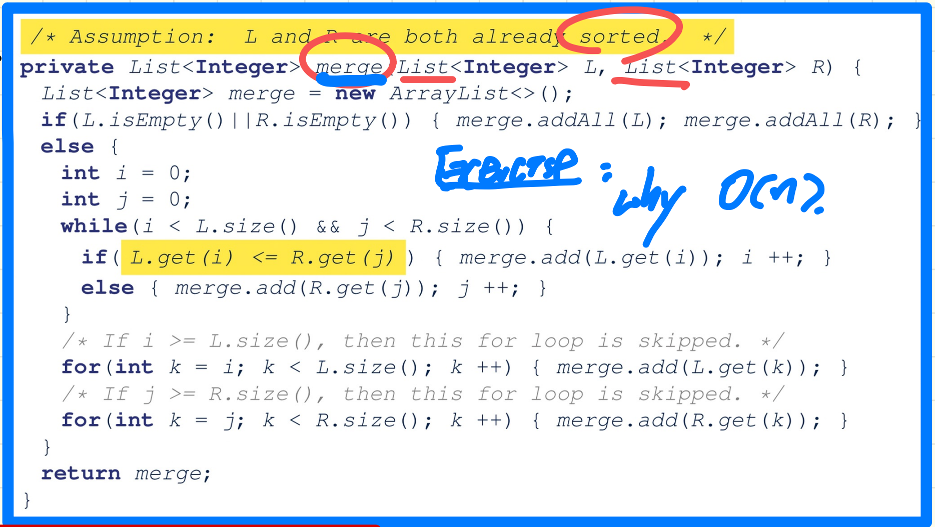
$1000 \cdot \log 1000$

10

$= 10000$

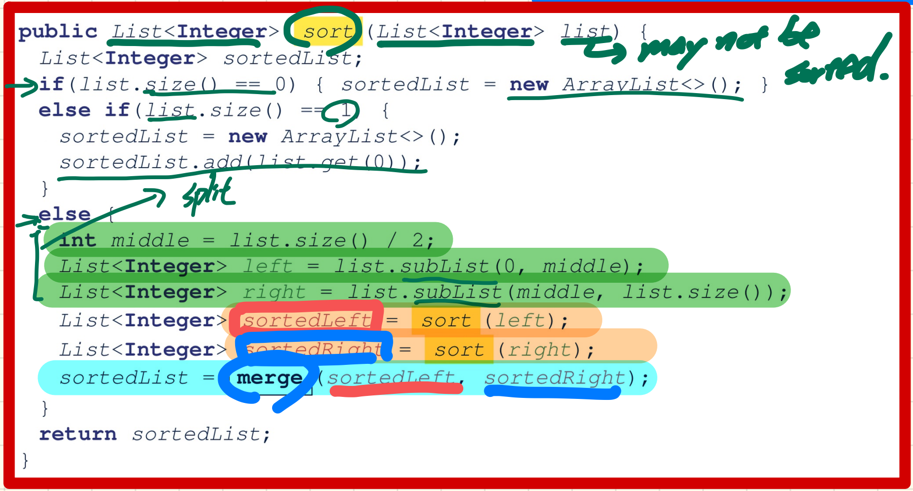
# Merge Sort in Java





```
/* Assumption:  L and R are both already sorted.  */
private List<Integer> merge(List<Integer> L, List<Integer> R) {
  List<Integer> merge = new ArrayList<>();
  if(L.isEmpty()||R.isEmpty()) { merge.addAll(L); merge.addAll(R); }
  else {
    int i = 0;
    int j = 0;
    while(i < L.size() && j < R.size()) {
      if( L.get(i)  <= R.get(j) ) { merge.add(L.get(i)); i ++; }
      else { merge.add(R.get(j)); j ++; }
    }
    /* If i >= L.size(), then this for loop is skipped. */
    for(int k = i; k < L.size(); k ++) { merge.add(L.get(k)); }
    /* If j >= R.size(), then this for loop is skipped. */
    for(int k = j; k < R.size(); k ++) { merge.add(R.get(k)); }
  }
  return merge;
}
```

Exercise: why O(n)?



```
public List<Integer> sort (List<Integer> list) {
  List<Integer> sortedList;
  if(list.size() == 0) { sortedList = new ArrayList<>(); }
  else if(list.size() == 1) {
    sortedList = new ArrayList<>();
    sortedList.add(list.get(0));
  }
  else {
    int middle = list.size() / 2;
    List<Integer> left = list.subList(0, middle);
    List<Integer> right = list.subList(middle, list.size());
    List<Integer> sortedLeft = sort (left);
    List<Integer> sortedRight = sort (right);
    sortedList = merge (sortedLeft, sortedRight);
  }
  return sortedList;
}
```

→ may not be sorted.

split

$T(0) = 1$

$T(1) = 1$

$T(n) = 2 * T(n/2) + n + 1$

split

left & right

merge

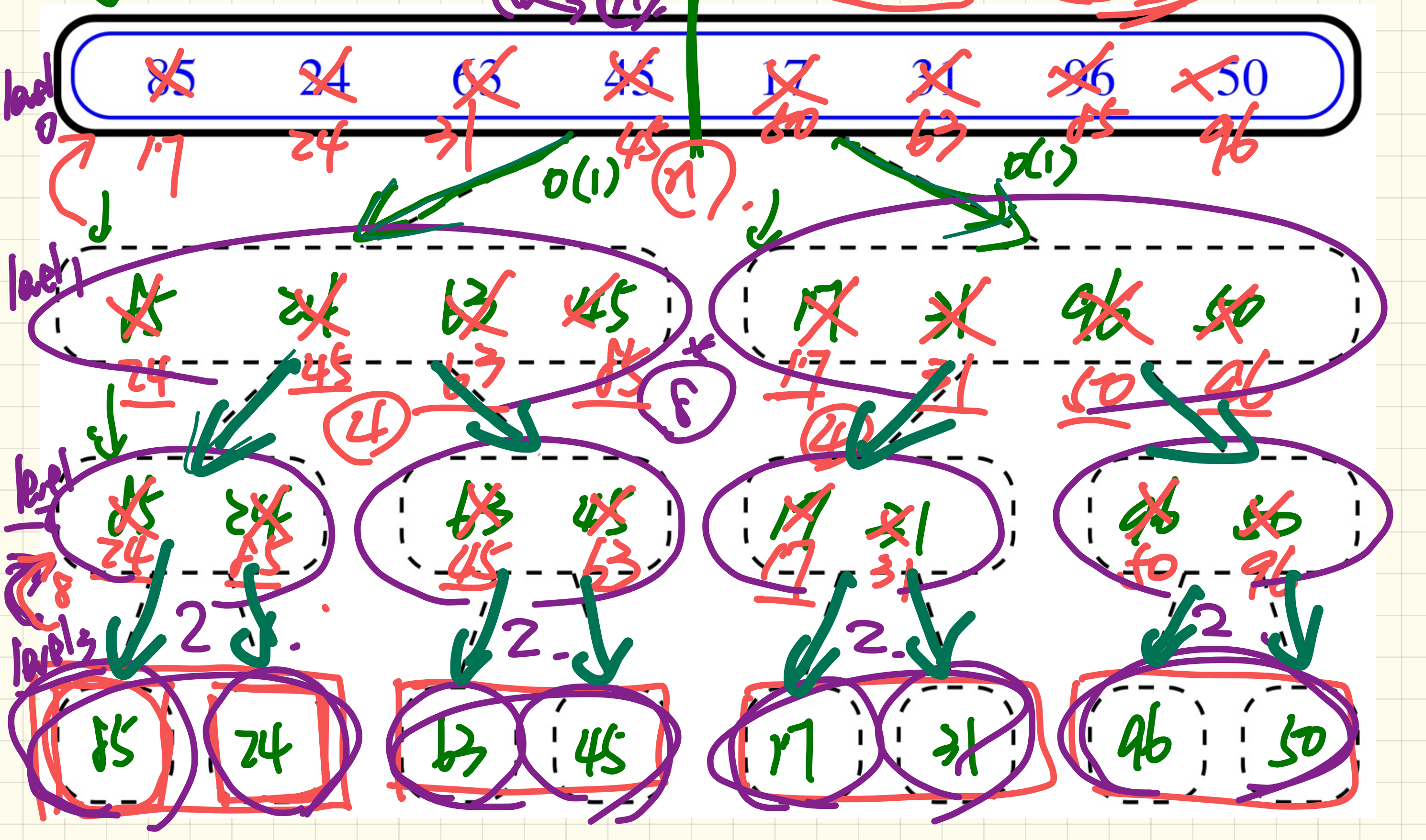
# Merge Sort: Tracing

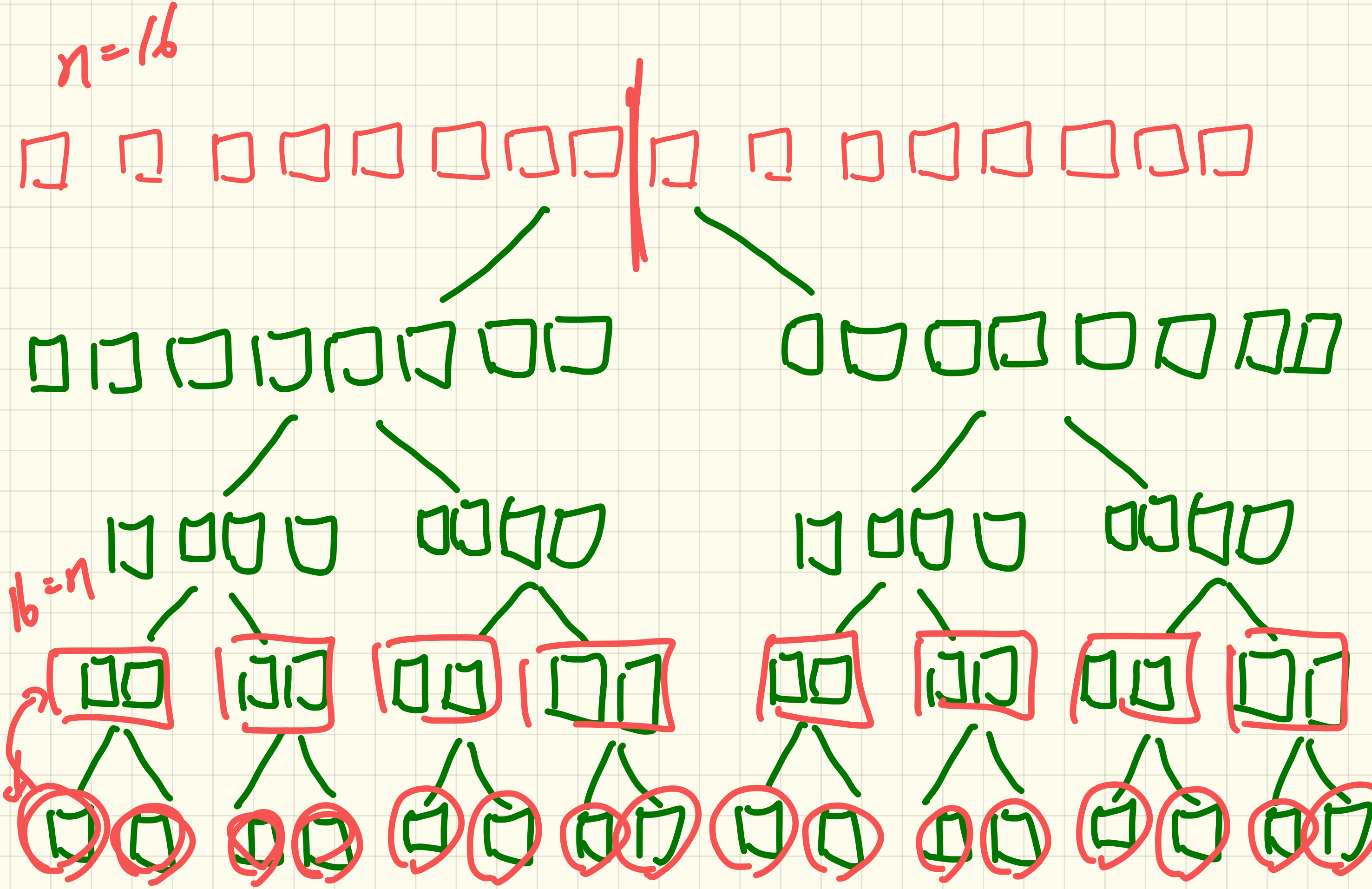
RT from L3 to L2 8 elements n

RT from L2 to L1:

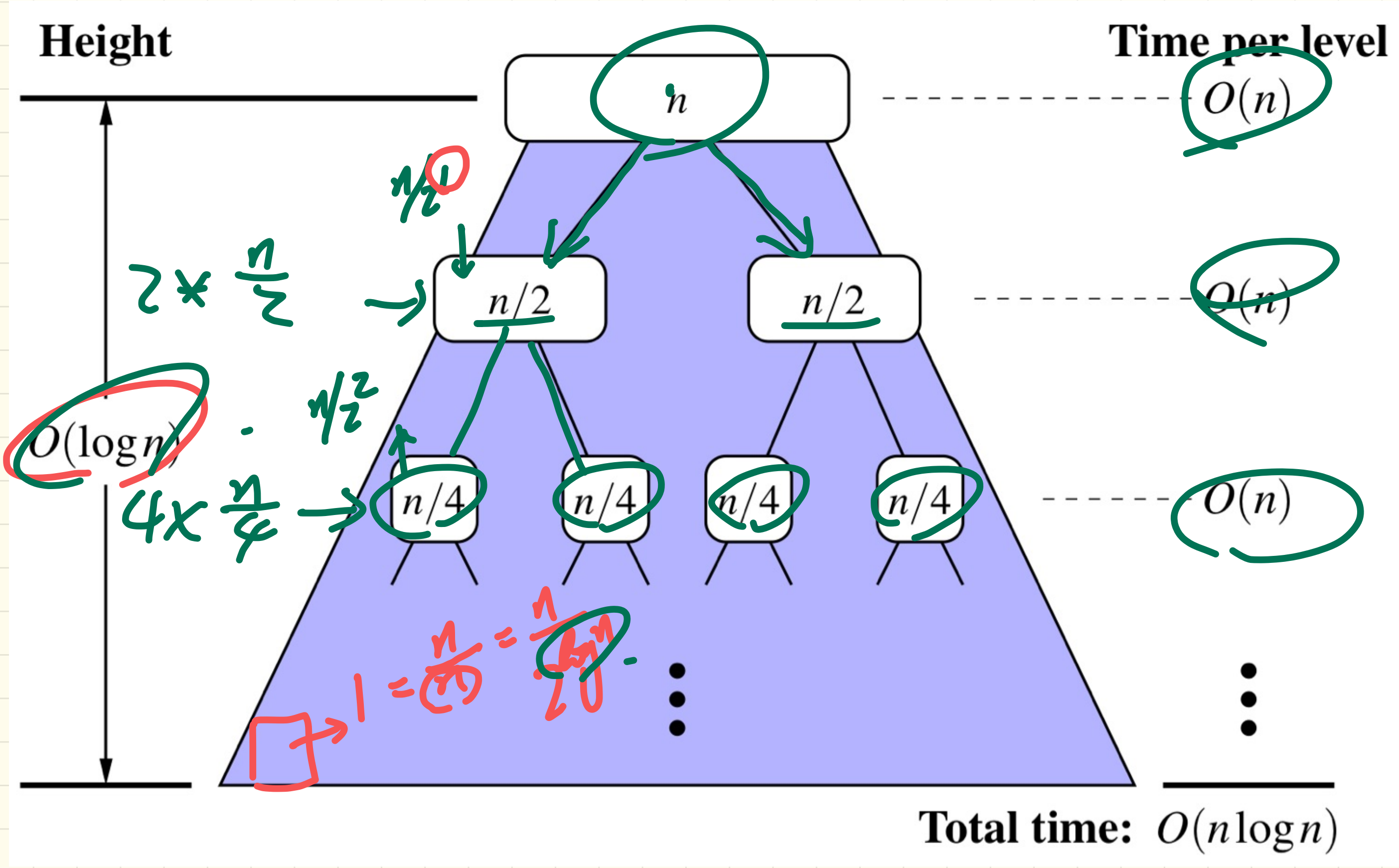
split

merge




n=16
16=n

# Merge Sort: Running Time



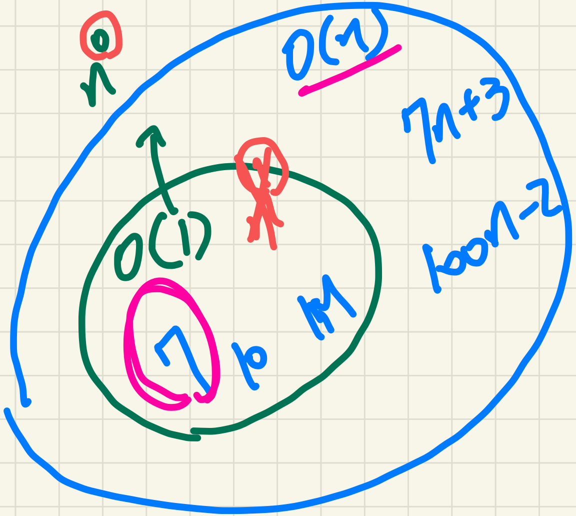
$n^k$ $k \geq 0$

$n^0 = 1$ $O(1)$

$n^1 = n$ $O(n)$

$\vdots$

O(?) a set of functions which can be upper bounded by ?

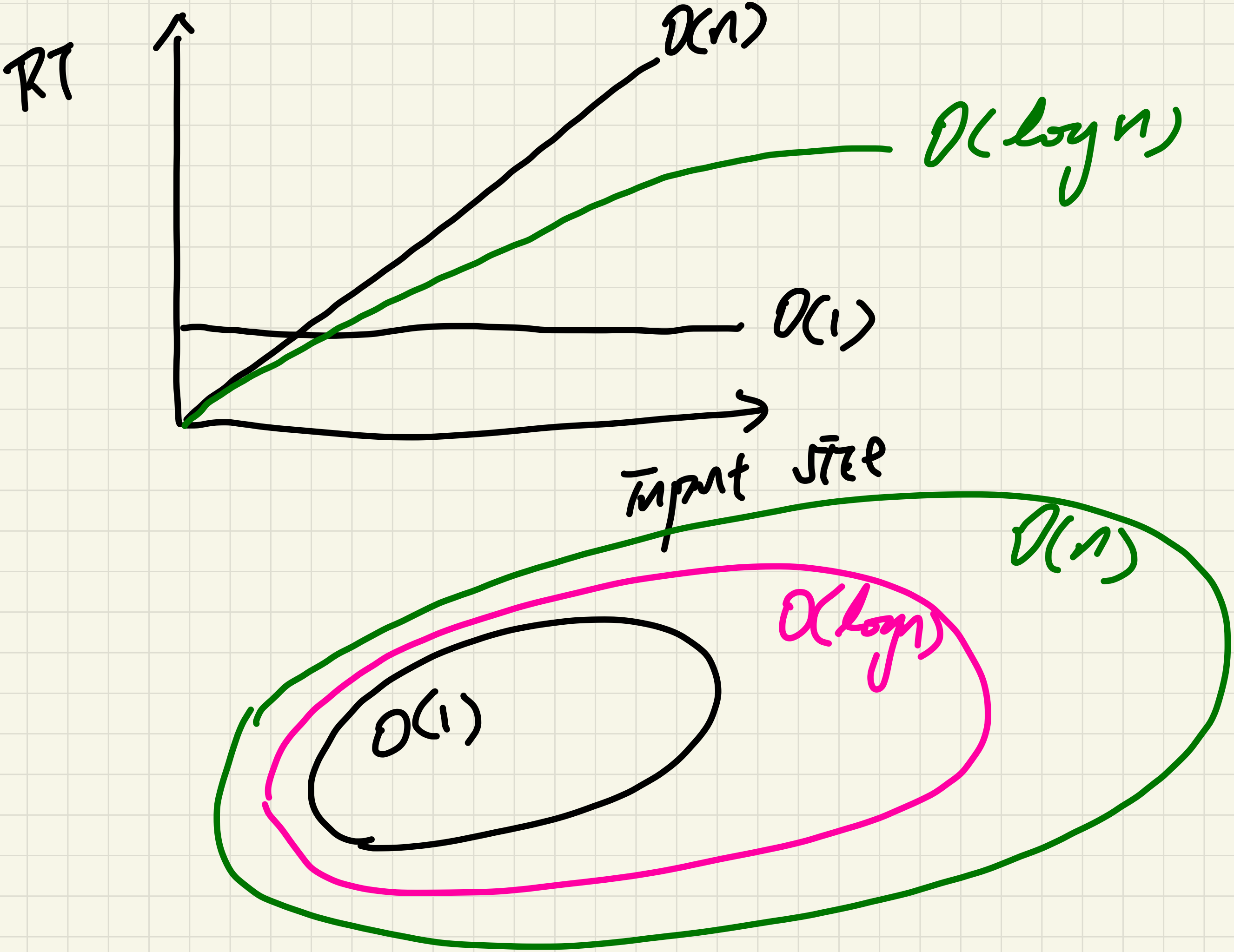


✓ $7 \in O(1)$

✓ $7 \in O(n)$

$7 \in O(1)$

$c = 7$


RT
O(n)
O(log n)
O(1)
Input size
O(n)
O(log n)
O(1)

$$f(n) = 5n^2 + 3n \cdot \log n + 2n + 5.$$

$$\leftarrow O(n^2) \quad 5 \cdot 1^2 + 3 \cdot 1 \cdot \log 1 + 2 \cdot 1 + 5$$

$$5 + 3 + 2 + 5 = \boxed{15}$$

Prove.

choose $\boxed{c} = ?$

$n_0 = ?\ 1$ s.t.

$$f(n) \le \cancel{c} \cdot n^2$$

$15$

✓

$$f(1) \le 15 \cdot 1^2$$

$12$

$f(n) = \boxed{3} \cdot \log n + \boxed{2}$

$\underline{f(n) \in O(\underline{\log n})}$ .

$$\underline{3} \cdot \log \overset{1}{\cancel{n}}_{\underline{2}} + 2 \le \overset{}{\cancel{c}}_{\underline{5}} \cdot \log \cancel{n}_{2}$$

$\cancel{3+2}$ ✓

$\underline{5} \le \underline{5}$

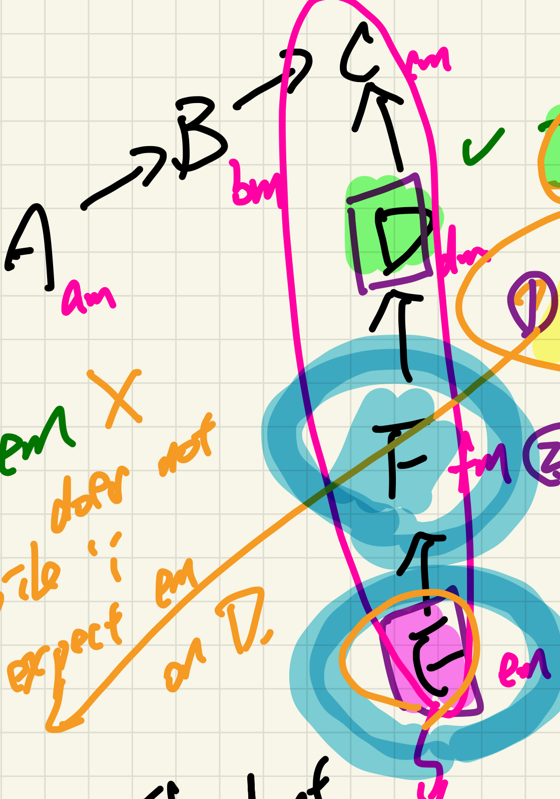
Prove.

choose $\underline{c} = \cancel{4}\ \underline{5}$

$\underline{\underline{n_0}} = \cancel{4}\ \underline{\cancel{1}}\ 2$ s.t.

$$\underline{3} \cdot \underline{\log \overset{1}{\cancel{n}}} + \cancel{2} \le \underset{5}{\cancel{c}} \cdot \log \overset{1}{\cancel{n}} \quad \text{for } n \geq n_0$$

$$2 \le 0$$



✓ D obj = new F();

① ((E) obj).em ✓ compile.

② ((E) obj).gm ✗ not compile

downcast

(E) obj. em ✗

does not compile ∵ cannot expect em on D.

CCE

↳ ∵ DT F is not a descendant of cast type E.

em
fm
dm
cm
-

((E) obj). dm ✓

downcast

merge sort:

$\boxed{O(n \cdot \log n)}$

tightest.

merge sort $\in O(n^2)$

correct but not accurate.

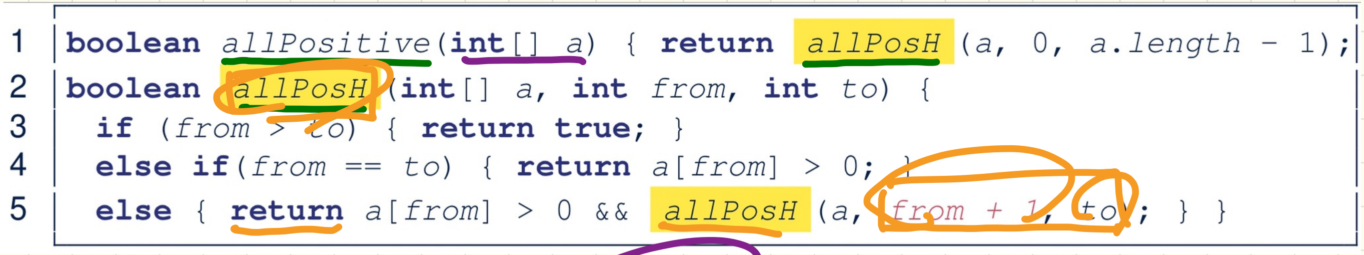
insertion sort:

$O(n^2)$

tightest.

# Correctness Proofs: Ideas

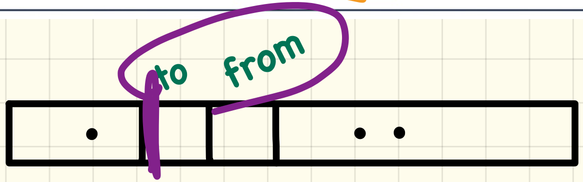
from ≤ to



```
boolean allPositive(int[] a) { return allPosH (a, 0, a.length - 1); }
boolean allPosH (int[] a, int from, int to) {
  if (from > to) { return true; }
  else if(from == to) { return a[from] > 0; }
  else { return a[from] > 0 && allPosH (a, from + 1, to); } }
```

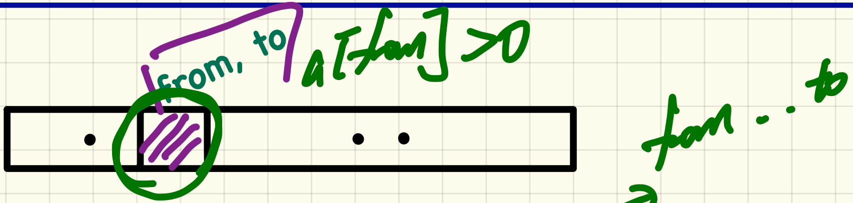
**Base Case:**

**Empty Array**



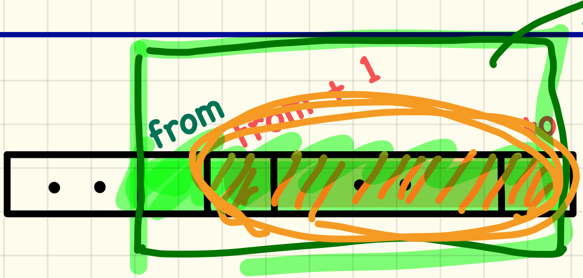
**Base Case:**

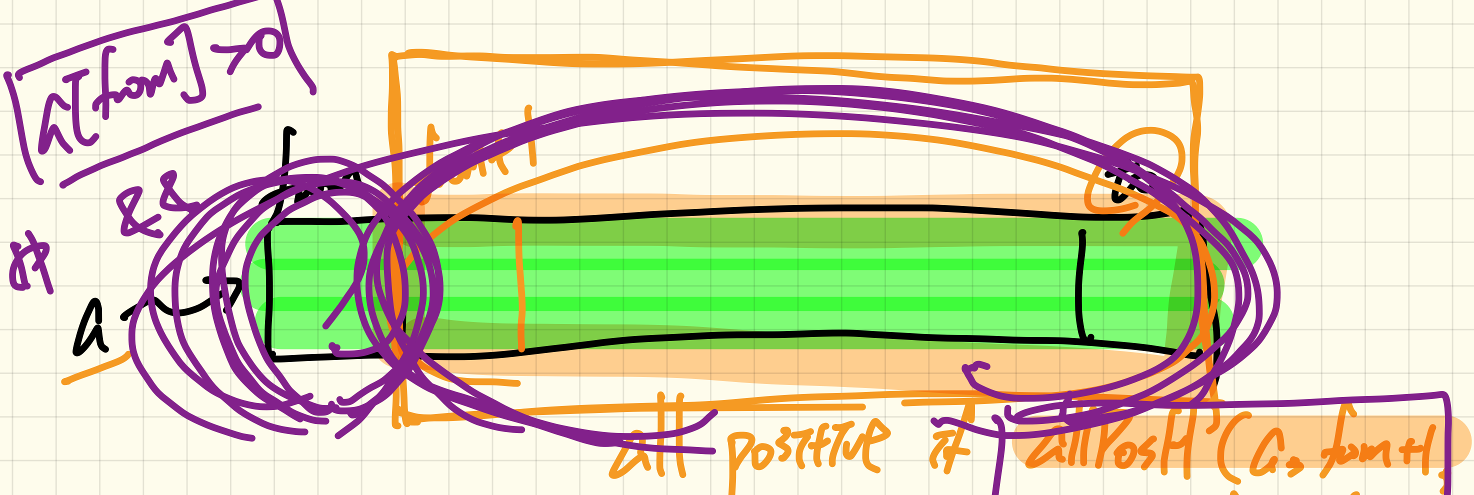
**Array of Size 1**



**Recursive Case:**

**Array of size > 1**





**Problem.** Are elements a[from], a[from+1], ..., a[to] all positive?

I.H. calling allPosH(a, from+1, to) will return T iff a[from+1], ..., a[to]

# Correctness Proofs

a

```
1  boolean allPositive(int[] a) { return allPosH (a, 0, a.length - 1); }
2  boolean allPosH (int[] a, int from, int to) {
3    if (from > to) { return true; }
4    else if(from == to) { return a[from] > 0; }
5    else { return a[from] > 0 && allPosH (a, from + 1, to); } }
```

I.H.

- Via mathematical induction, prove that allPosH is correct:

  **Base Cases**

  - In an empty array, there is no non-positive number ∴ result is ***true***. **[L3]**
  - In an array of size 1, the only one elements determines the result. **[L4]**

  **Inductive Cases**

  - **Inductive Hypothesis**: allPosH(a, from + 1, to) returns ***true*** if a[from + 1], a[from + 2], . . . , a[to] are all positive; ***false*** otherwise.
  - allPosH(a, from, to) should return ***true*** if: **1)** a[from] is positive; and **2)** a[from + 1], a[from + 2], . . . , a[to] are all positive.
  - By *I.H.*, result is $a[from] > 0 \wedge$ allPosH(a, from + 1, to). **[L5]**

- allPositive(a) is correct by invoking allPosH(a, 0, a.length - 1), examining the entire array. **[L1]**

allP[]

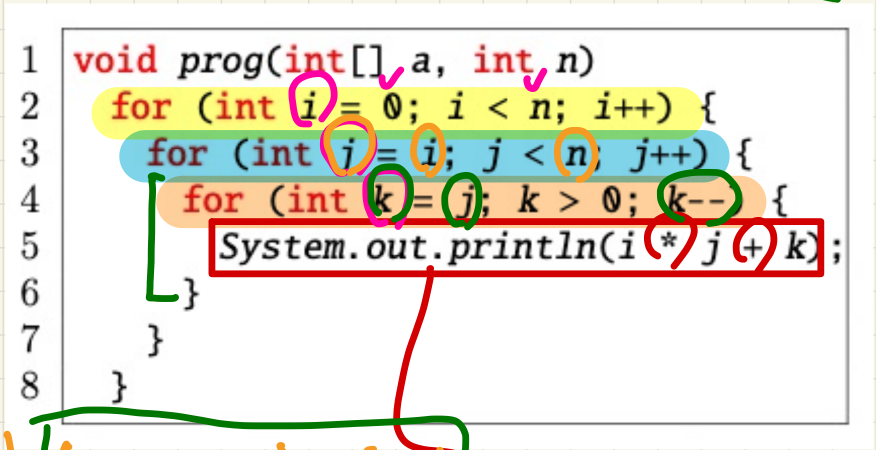
expect

allP.

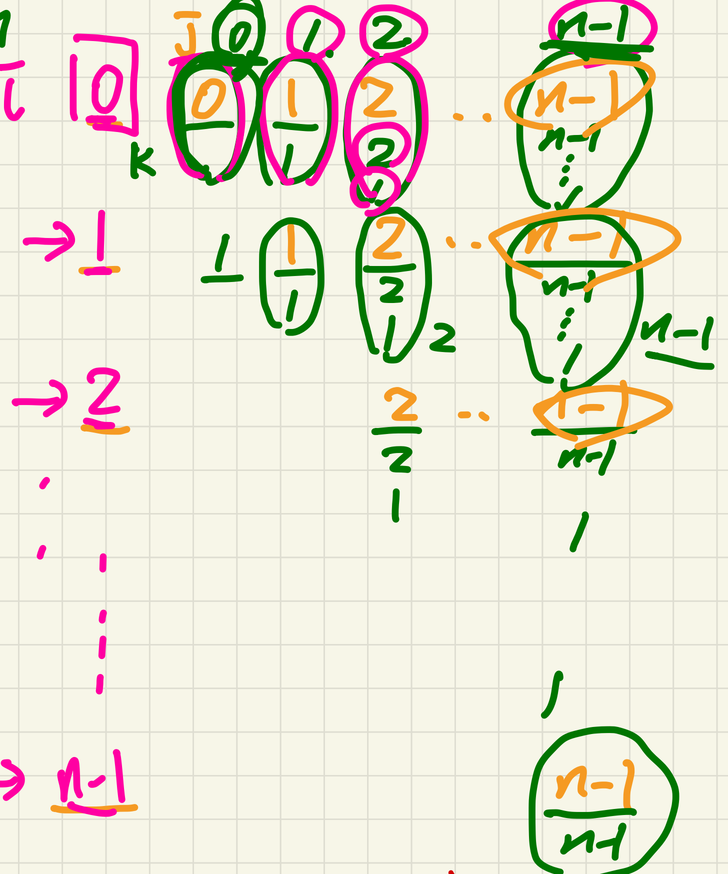
# EXAM REVIEW II

TUESDAY DECEMBER 10

$$a_1 + a_2 + \cdots a_n = \frac{(a_1 + a_n) * n}{2}$$



```
void prog(int[] a, int n)
  for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
      for (int k = j; k > 0; k--) {
        System.out.println(i * j + k);
      }
    }
  }
```



$$\sum_{i=0}^{n-1} \frac{(i + (n-1)) * (n-i)}{2}$$

body of loop

O(1)

$$i = 0 \quad \frac{(0 + (n-1)) * (n-0)}{2}$$

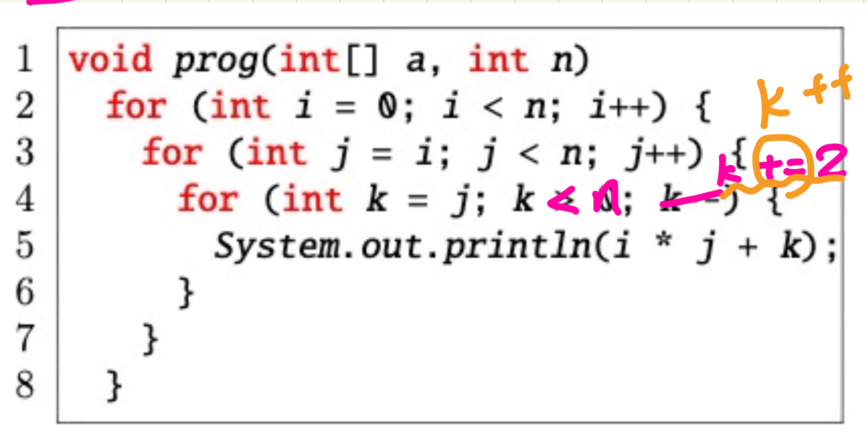
$$i = 1 \quad \frac{(1 + (n-1)) * (n-1)}{2}$$

⋮

$$i = n-1 \quad \frac{((n-1) + (n-1)) * (n-(n-1))}{2} \Big]$$

$$RT = O(\underbrace{1}_{\text{each iteration}} * \# \text{ of iterations})$$

Exercise -



```
void prog(int[] a, int n)
  for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
      for (int k = j; k > 0; k--) {
        System.out.println(i * j + k);
      }
    }
  }
```

k++

k += 2

< n

Design an algorithm for X

s.t. RT(n · log n) 50%

. RT($n^2$)

# Javadocs:

1. Read Javadocs.

@param

@return

@throws

@precondition

↳ assume

~~String~~ m( = . - ){

. . -

}

# Unit Testing.

1. Read unit tests.

2. ~~No need to write Junit tests~~

2. Given problem, come up with test cases.

## Option 1.

```
/** @return - -
 * @param x ...
 * @param y --
 * @throws X - -   */
double divide(double x, double y) throws X {
    if (y == 0) { throw new X(...); }
    return x / y;
}
```

defensive about illegal value passed by caller.

```
class MyClass {
    void m1() { divide(input, 0);
        [System.println( Y );
    }
    private ... divide(...) {
} }
```

when things under your control

## Option 2:

private

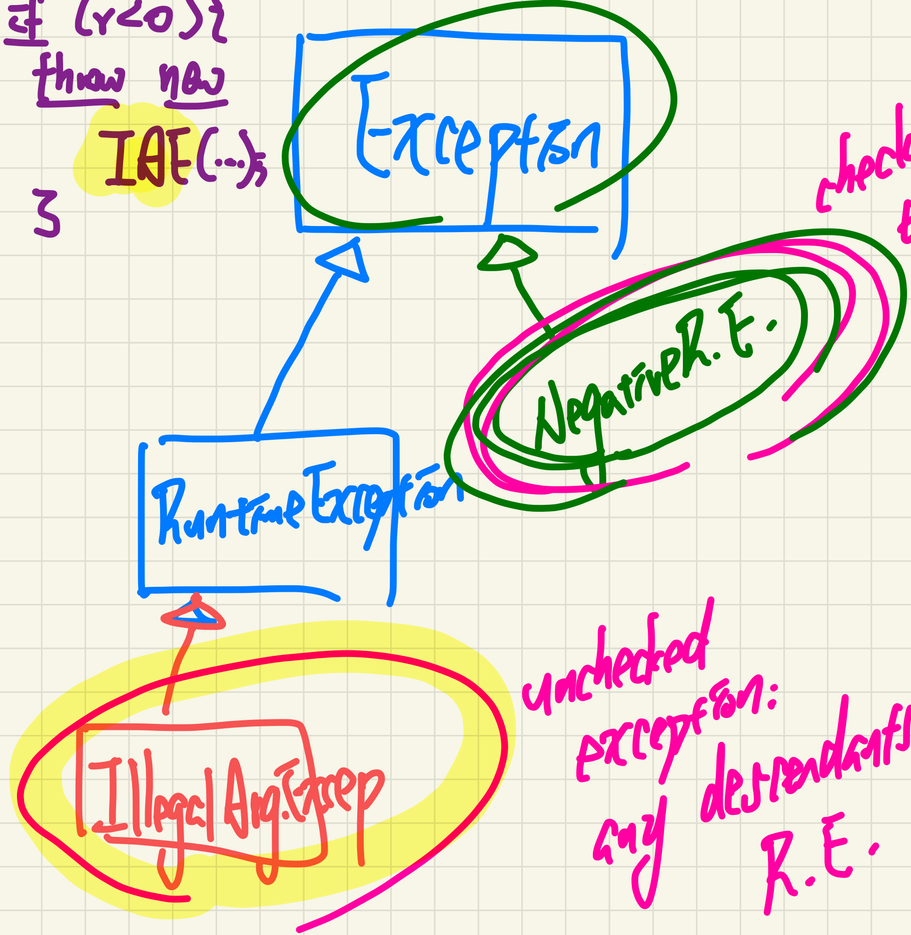
```
/** @return
 @param x
 @param y
double divide(double x, double y) {
    return x / y;
}
```

assume: y != 0.

@precondition y != 0

under what circumstance can you call divide.

```
void m(int r) {
  if (r<0){
    throw new
      IAE(...);
}
```



checked exception: any direct child class of Exception.
(subject to: catch or specify Req.

unchecked exception: any descendants of R.E.

```
void m(int r) throws NRE {
  if (r<0) {
    throw new NRE(..);
}
```

## Exercise.

two counters

c1 : min -1 ] init : 0
max 3

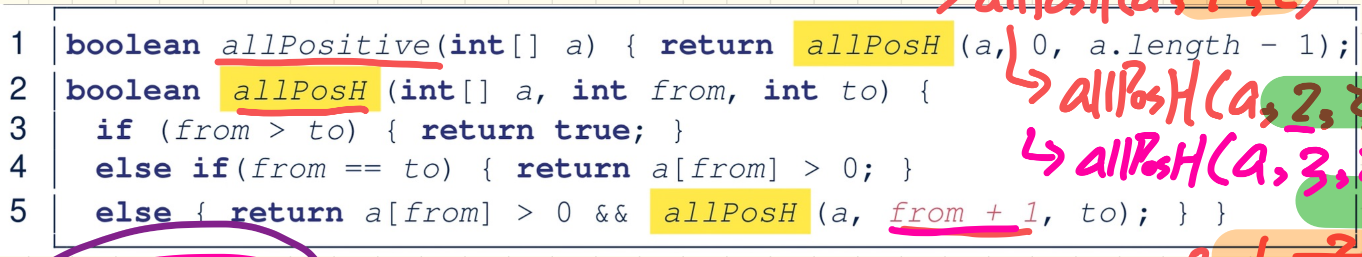
inc

c2 : min 4 ] init : 4
max 7

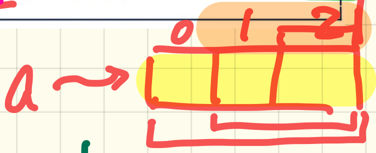
dec

# Correctness Proofs: Ideas

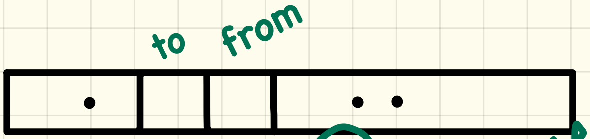
allPosH (a, 0, 2)
↳ allPosH(a, 1, 2)
↳ allPosH(a, 2, 2)
↳ allPosH(a, 3, 2)



```
boolean allPositive(int[] a) { return allPosH (a, 0, a.length - 1); }
boolean allPosH (int[] a, int from, int to) {
  if (from > to) { return true; }
  else if(from == to) { return a[from] > 0; }
  else { return a[from] > 0 && allPosH (a, from + 1, to); } }
```



Base Case:
Empty Array

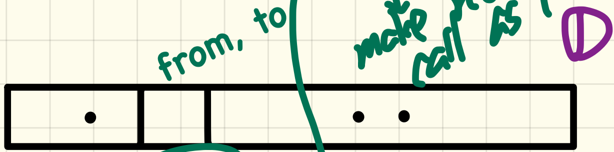


make recursive call as the I.H.

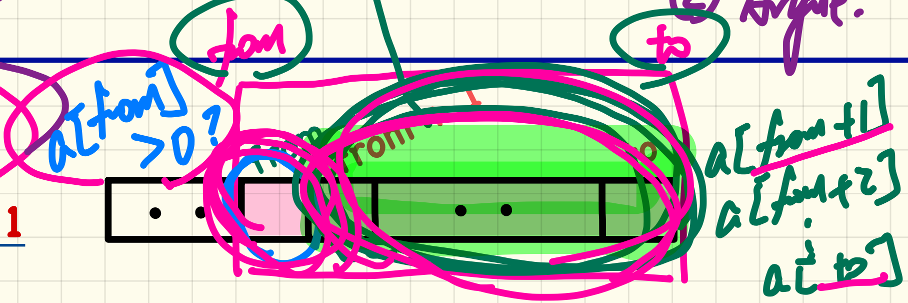
① Link to the code (line #'s)

② Argue.

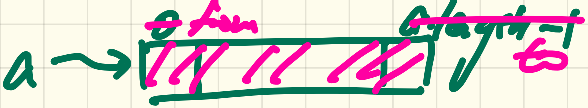
Base Case:
Array of Size 1

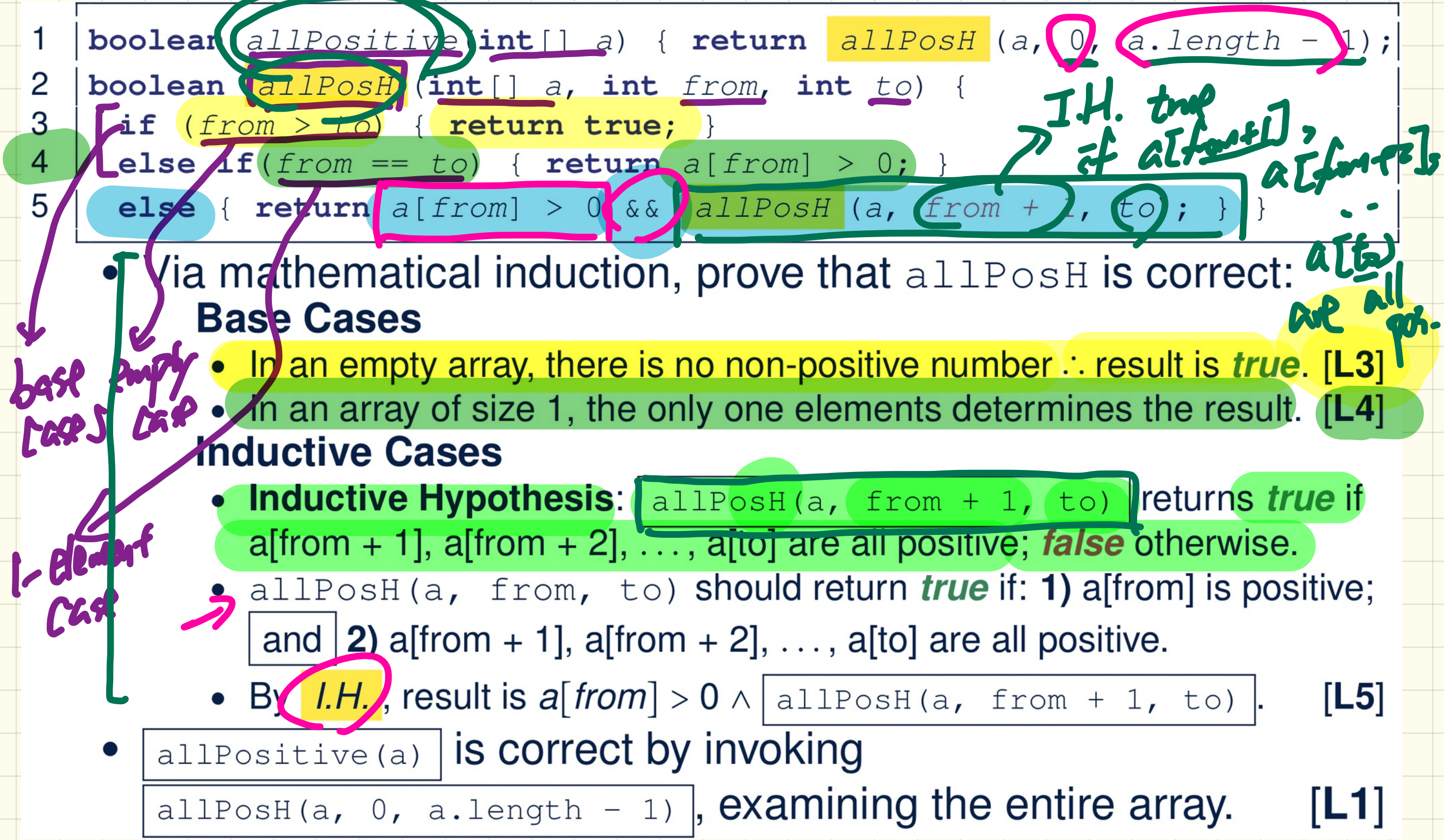


Recursive Case:
Array of size > 1



# Correctness Proofs





```
1 boolean allPositive(int[] a) { return allPosH (a, 0, a.length - 1); }
2 boolean allPosH (int[] a, int from, int to) {
3   if (from > to) { return true; }
4   else if(from == to) { return a[from] > 0; }
5   else { return a[from] > 0 && allPosH (a, from + 1, to); } }
```

I.H. true if a[from+1], a[from+2], .. a[to] are all pos.

- Via mathematical induction, prove that `allPosH` is correct:

  **Base Cases**

  - In an empty array, there is no non-positive number ∴ result is ***true***. **[L3]**
  - In an array of size 1, the only one elements determines the result. **[L4]**

  **Inductive Cases**

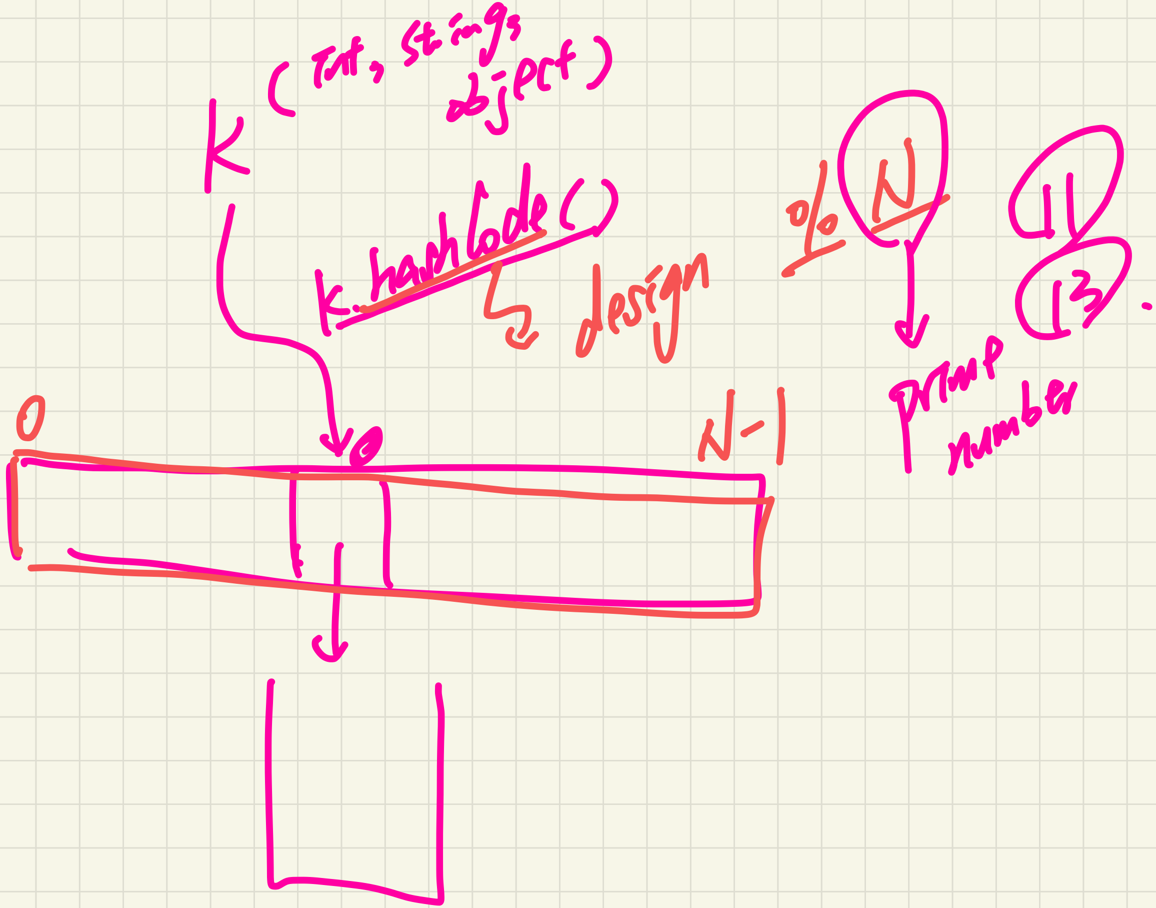
  - **Inductive Hypothesis**: `allPosH(a, from + 1, to)` returns ***true*** if a[from + 1], a[from + 2], . . . , a[to] are all positive; ***false*** otherwise.
  - `allPosH(a, from, to)` should return ***true*** if: **1)** a[from] is positive; and **2)** a[from + 1], a[from + 2], . . . , a[to] are all positive.
  - By *I.H.*, result is $a[from] > 0 \land$ `allPosH(a, from + 1, to)`. **[L5]**

- `allPositive(a)` is correct by invoking `allPosH(a, 0, a.length - 1)`, examining the entire array. **[L1]**

base cases

empty case

1-element case


K (int, strings, object)
K.hashCode()
↳ design
% N
prime number
11
13.
0
N-1

180 min.

→ 50 multiple choice

↳ <2 minutes:

1st round: complete all q's that you can answer

5 written questions.

↳ 10 min.

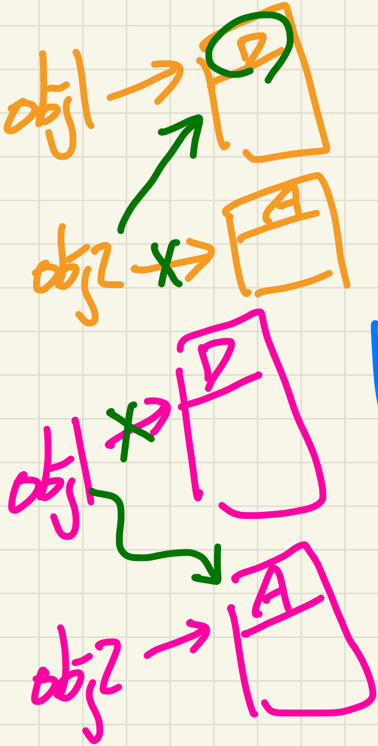
Domain

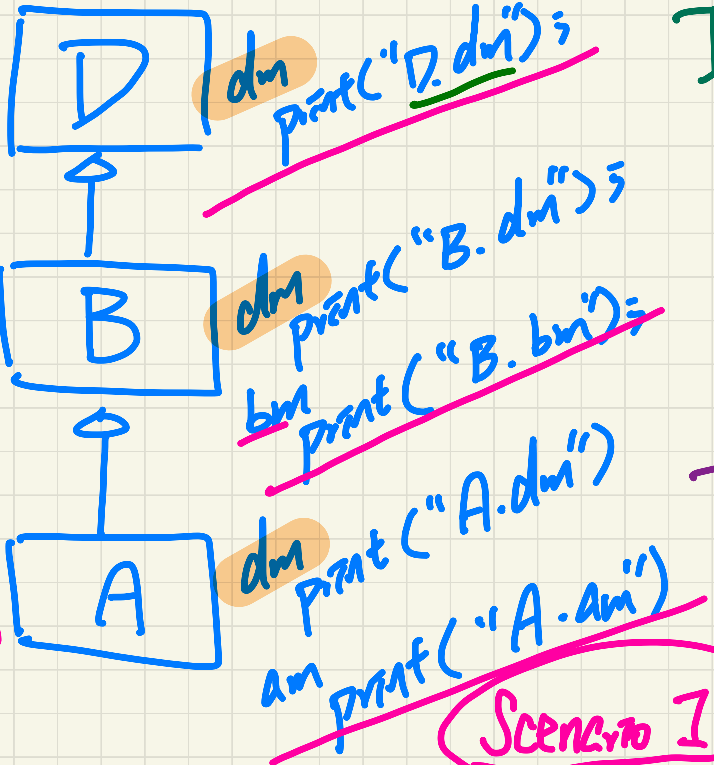
$hc(k) = k \% 11$

$hc(k1) == hc(k2)$

$\underbrace{hc(23)}_{1} == \underbrace{hc(34)}_{1}$ ✓ collision

$\underset{23 \% 11}{hc(23)} == \underset{23 \% 11}{hc(23)}$ → same result for the same input.





```
D obj1 = new D();
obj1.bm(); X
ST: D
obj1.dm(); -> "D.dm"

D obj2 = new A();
obj2.dm(); -> "A.dm"
```

Scenario 1

```
obj1 = obj2;
obj1.dm();
"A.dm"
```

Scenario 2

```
obj2 = obj1;
obj2.dm();
"D.dm"
```

```
class App {
  ... main(...) {
    C oc = new C();
    D obj1 = new A();
    oc.add(obj1, obj1);
    B obj2 = new A();
    oc.add(obj2, obj2);
```

X X

ST: B ST: B

```
oc.add(obj2, (A) obj2);
```

$\underline{b} = \underline{obj2}$
B B

$\underline{a} = \underline{obj2}$
A B

```
class C {
  B[] array;
  int noi; /* # of items */
  void add(D d) {
    a[noi] = d;
  }
  void add(B b, A a) {
    a[noi] = b;
    noi++;
  }
```

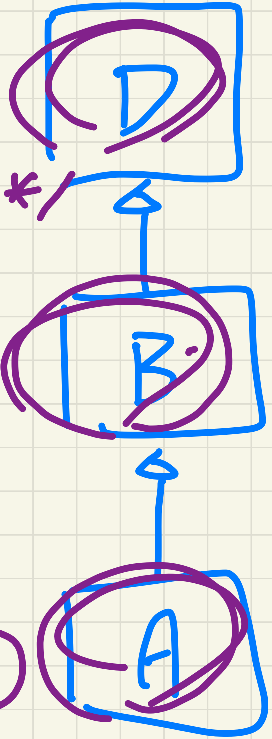
X X

ST: B ST: D

b = obj1

a[noi] = a;

ST: B ST: A



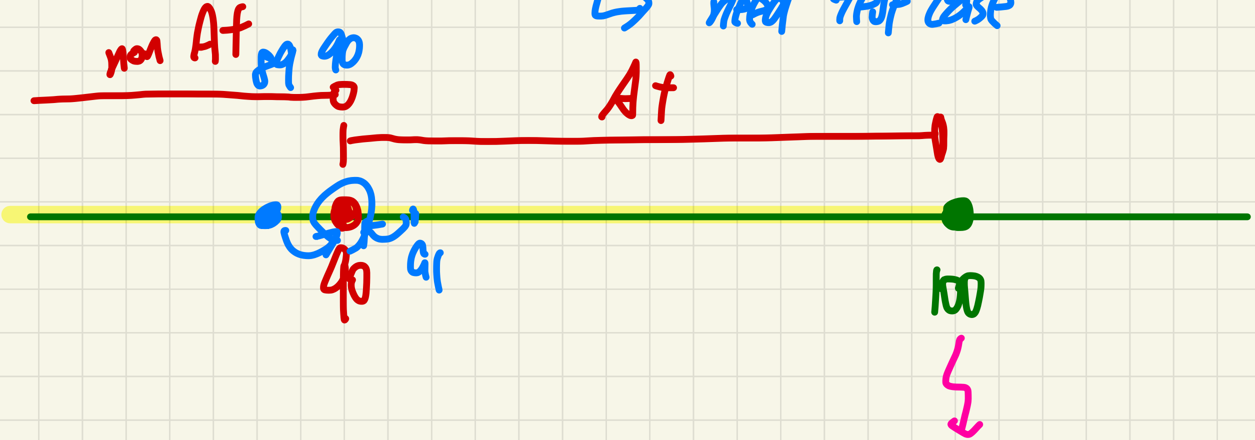
Can abstract class implement interface?

# Exam Review III

Thursday December 13

@throws IllegalArgumentException when grade > 100

↳ need test case



@pre.
assume input is never >100

```
/**
 * @param x
 * @param y
 * @pre.   1 ≤ x ≤ 5 and   -3 ≤ y ≤ 2
 */
int abs ( int x , int y ) {
  → /* imp. */  return x - y;
}
```

x > y

abs( 2 , 1 )

abs( 4 , 0 )

complete?

Q. Which of the following tests would reveal an error in the imp.?

(A) abs(2, 1)

(B) abs(4, 0)

(C) abs(2, 2)

[✓ (D) abs(1, 2)]

return 1 - 2 = -1

```
/**
 * @pre: ~~radius is non-negative~~
 */
double area( double radius) {
    /* .. */
}
```

! <0 ≥0

@throws IAE when radius is negative.

Which of the following tests would be considered as boundary cases?

(Z) 0

(A) -0.5

(B) 0.3

(C) -0.8

[✓ (D) All.]

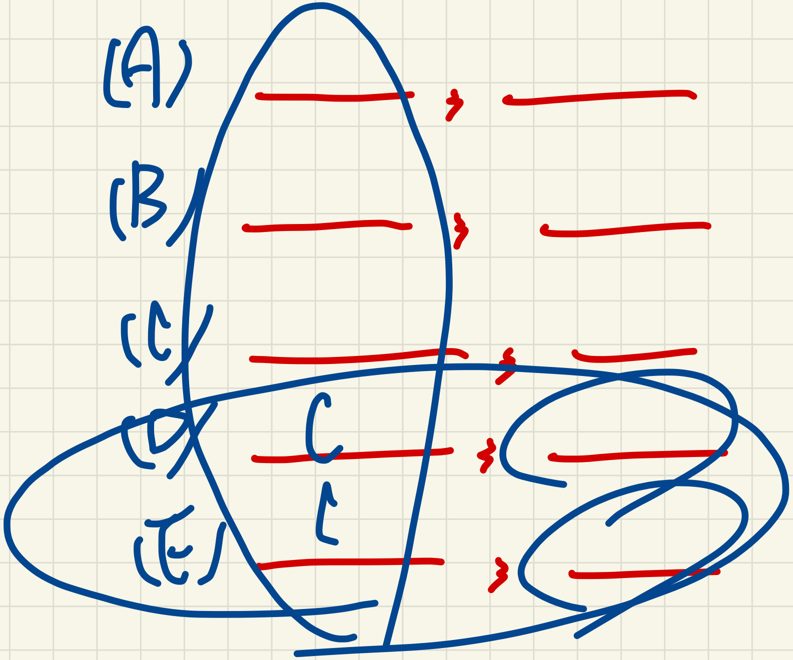
(E) None

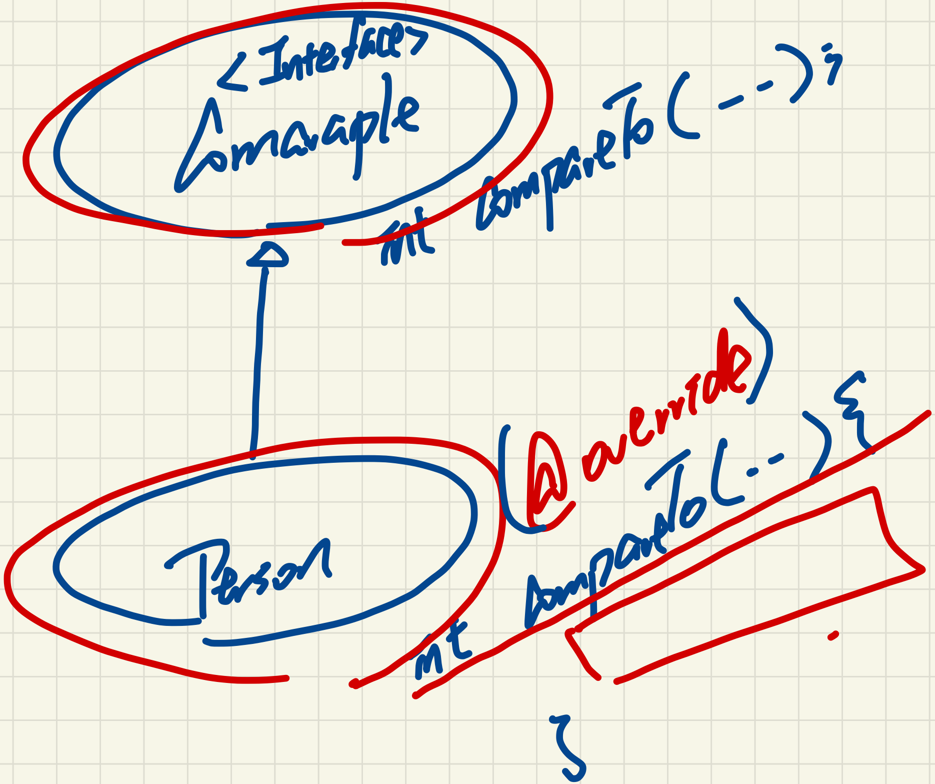
x 10.2

x.compareTo(y) must return 0 if x is equal to y.

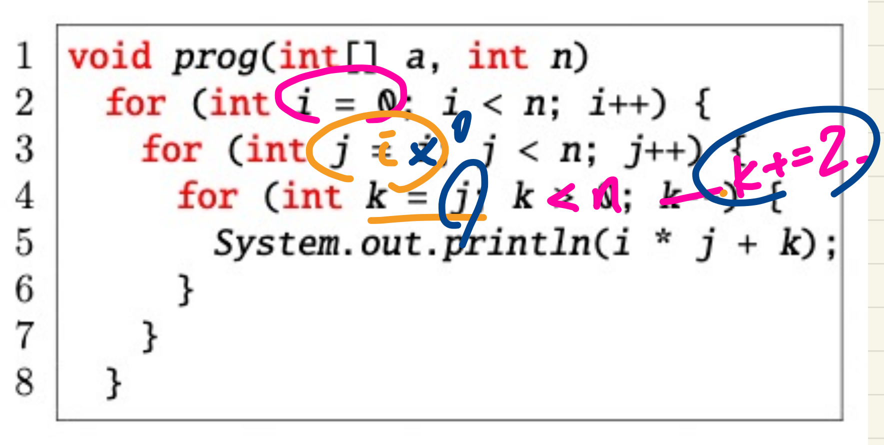
x.equals(y)

T




<Interface>
Comaraple
int compareTo(--);
Person
@override
int compareTo(--){
}

$$\begin{array}{ccc} r & > & s \\ s & > & p \\ \hline \boxed{r} & \boxed{>} & \boxed{p} \end{array}$$



```
void prog(int[] a, int n)
  for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
      for (int k = j; k <= 0; k--) {
        System.out.println(i * j + k);
      }
    }
  }
```

$i=0 \quad j=0 \; .. \sim \; n-1$

$i=10 \quad j=10 \quad 11 .. \quad n-1$

| k=10 | 11 | n-1 |
|---|---|---|
| 12 | 13 | |
| ⋮ | ⋮ | |
| n-1 | n-1 | |

n-1

For each value of j, the # of iterat. for k is $O(\frac{n}{2})$

$$\overbrace{n + n + \cdots + n}^{n \text{ terms}} = n^2$$

$$n + (n-1) + \cdots + 1 = \frac{(n+1) \times n}{2}$$

$O(n^2)$

1. Linear search

if we call binS(a, m+1, to) { for ( - - ) }

a[from], a[from+1], ..., a[n-1]

why is this call not to miss k in a[from]?

O(n)

$T(0) = 1$

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

$O(\log n)$.

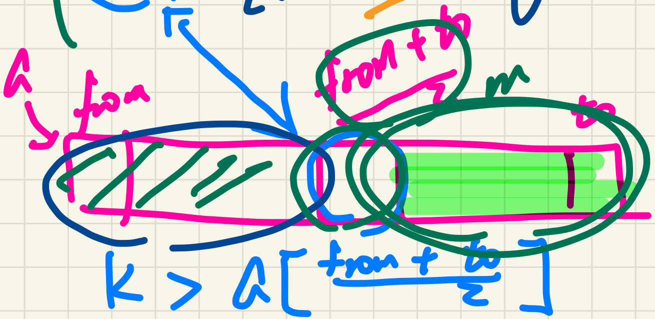
k.

2. binary search.

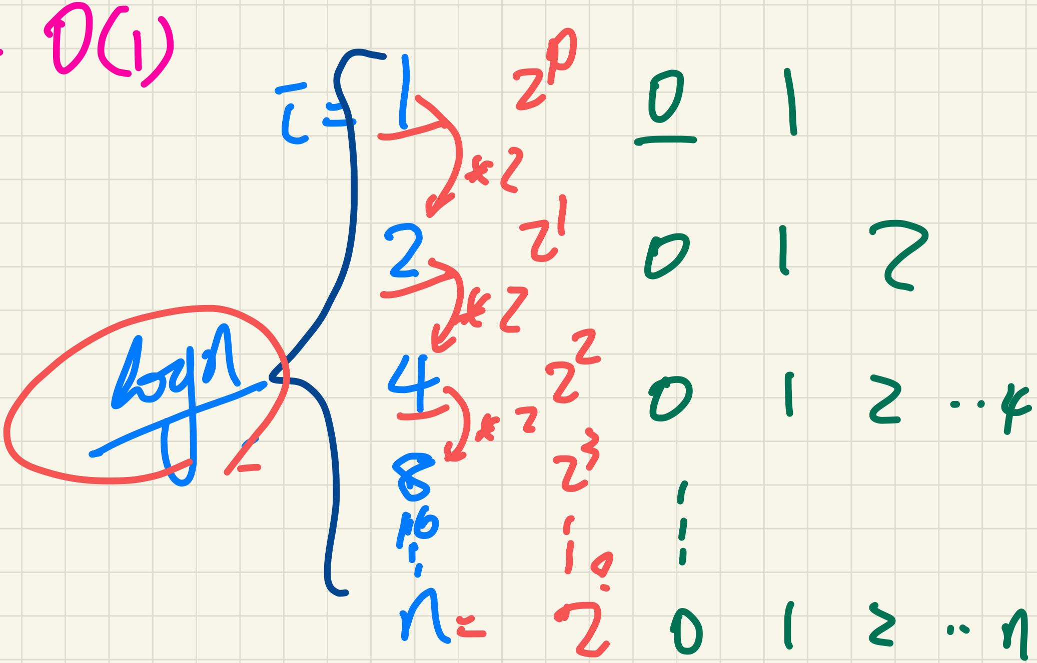
a. code.

b. running time ( formulate & solve recurr. rel.)

c. correctness



```
for( int i =1; i<=n; i = 2*i){
    for( int j=0; j<=i; j++){
        print(...);  O(1)
    }
}
```



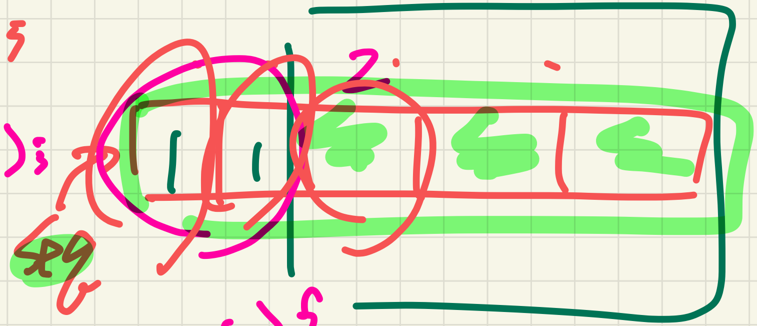
$T(0)=$ $T(1)=$ $T(2)=$

1 1 2 3 5 8 . . -

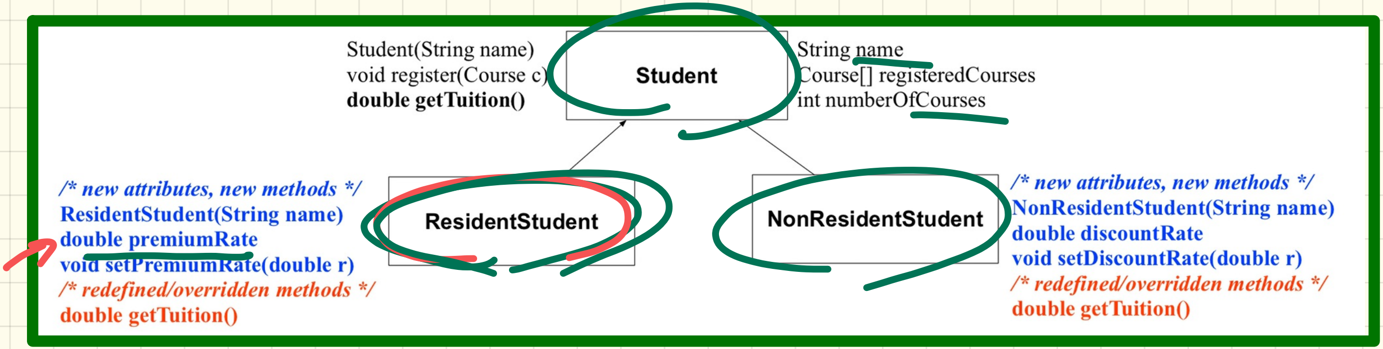
@pre. n >= 0 → assume n is not negative, no need to test n= -1, -2, -

$T(n)=$ $T(n-1)+1$

```
int[] fib(int n){
    int[] seq = new int[n];
    seq[0] =1; seq[1]= 1;
    fibH(seq, 2, seq.length-1);
    return seq;
}

void fibH(int[] seq, int from, int to){

}.
```



# Intuition: Polymorphism


Student
Student(String name)
void register(Course c)
double getTuition()
String name
Course[] registeredCourses
int numberOfCourses
ResidentStudent
/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()
NonResidentStudent
/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()

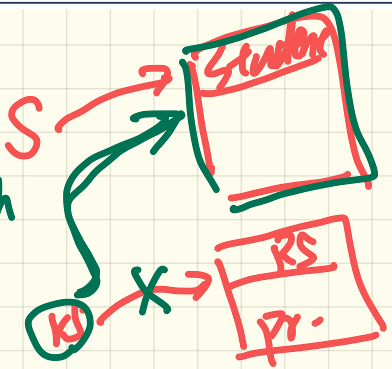
```
Student s = new Student("Stella");
ResidentStudent rs = new ResidentStudent("Rachael");
rs.setPremiumRate(1.25);
s = rs; /* Is this valid? */
rs = s; /* Is this valid? */
```

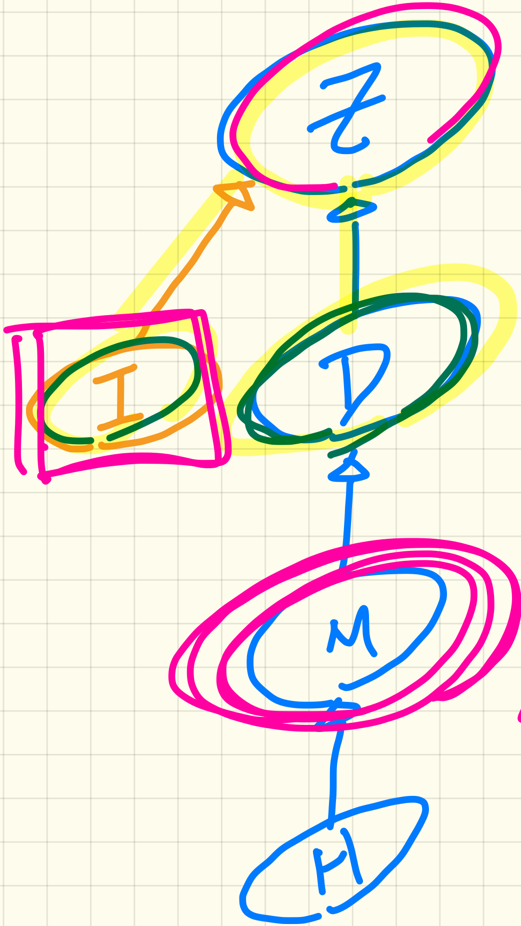
why is this not valid?

Say we allowed rs = s: X

↳ Exp. on rs:

pr rs.pr crash





D obj = new M();

I obj2 = obj; X

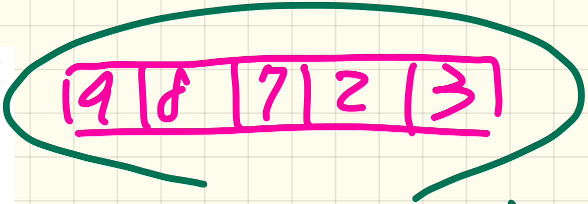
$$I\ obj2 = \underbrace{(I)}_{\text{neither upward nor doward}}\underbrace{obj}_{ST:\ D}; \quad X$$

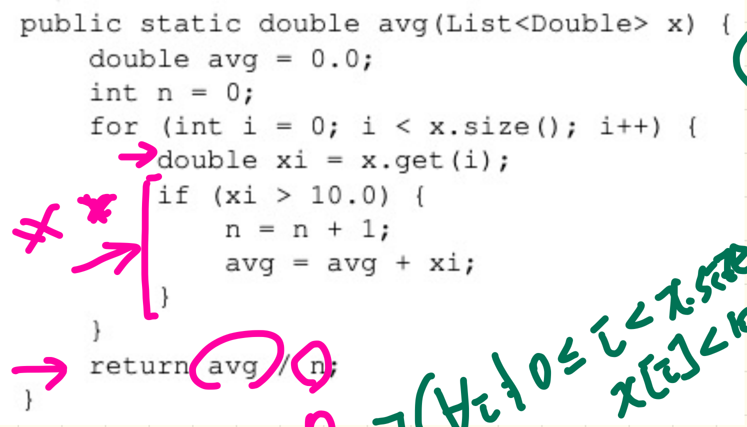
compiles but CCE.

I obj2 = (I)((Z) obj)

```
I  obj2 = (I)((Z) obj)  → CCE.
```

```
if ( obj instanceof Z &&
   (Z)obj instanceof I) {
      obj2 = (I)((Z) obj);
}
```





```
public static double avg(List<Double> x) {
    double avg = 0.0;
    int n = 0;
    for (int i = 0; i < x.size(); i++) {
        double xi = x.get(i);
        if (xi > 10.0) {
            n = n + 1;
            avg = avg + xi;
        }
    }
    return avg / n;
}
```

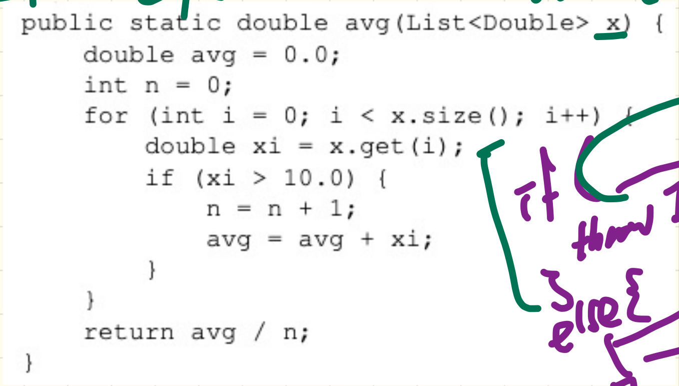
**

0 0

$\neg(\forall i \mid 0 \le i < x.size \bullet x[i] < 10)$

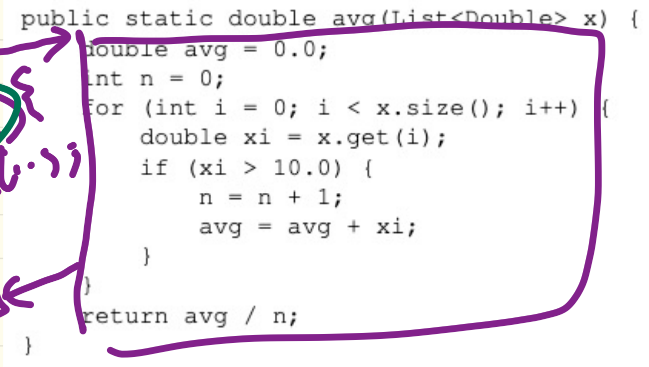
Opt1. @pre. not all numbers are < 10

Opt2. @throws IAE not all #'s are < 10

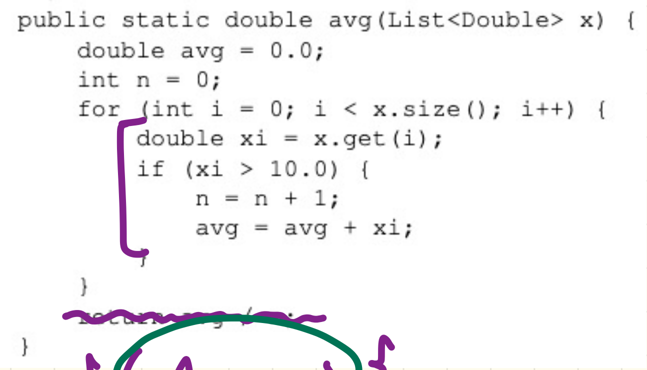


```
public static double avg(List<Double> x) {
    double avg = 0.0;
    int n = 0;
    for (int i = 0; i < x.size(); i++) {
        double xi = x.get(i);
        if (xi > 10.0) {
            n = n + 1;
            avg = avg + xi;
        }
    }
    return avg / n;
}
```

```
if (  ) {
    throw IAE(..);
} else {

}
```

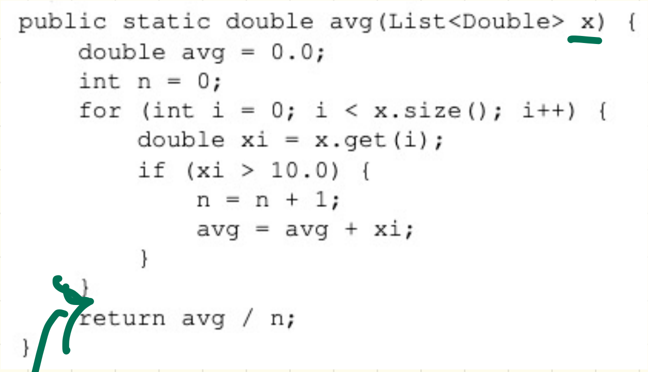


```
public static double avg(List<Double> x) {
    double avg = 0.0;
    int n = 0;
    for (int i = 0; i < x.size(); i++) {
        double xi = x.get(i);
        if (xi > 10.0) {
            n = n + 1;
            avg = avg + xi;
        }
    }
    return avg / n;
}
```



```
public static double avg(List<Double> x) {
    double avg = 0.0;
    int n = 0;
    for (int i = 0; i < x.size(); i++) {
        double xi = x.get(i);
        if (xi > 10.0) {
            n = n + 1;
            avg = avg + xi;
        }
    }
    ~~return avg / n;~~
}
```

```
if (n==0) {
    throw new IAE
} else {
    return avg/n;
}
```



```
public static double avg(List<Double> x) {
    double avg = 0.0;
    int n = 0;
    for (int i = 0; i < x.size(); i++) {
        double xi = x.get(i);
        if (xi > 10.0) {
            n = n + 1;
            avg = avg + xi;
        }
    }
    return avg / n;
}
```

```
if (n==0) {
    n = 1;
}
return avg/n;
```

$\frac{\underset{0}{avg}}{\underset{1}{n}}$

0.

assertSame(o1, o2)

↳ o1 == o2

assertEquals(o1, o2)

↳ o1.equals(o2)

END

ALL THE BEST !