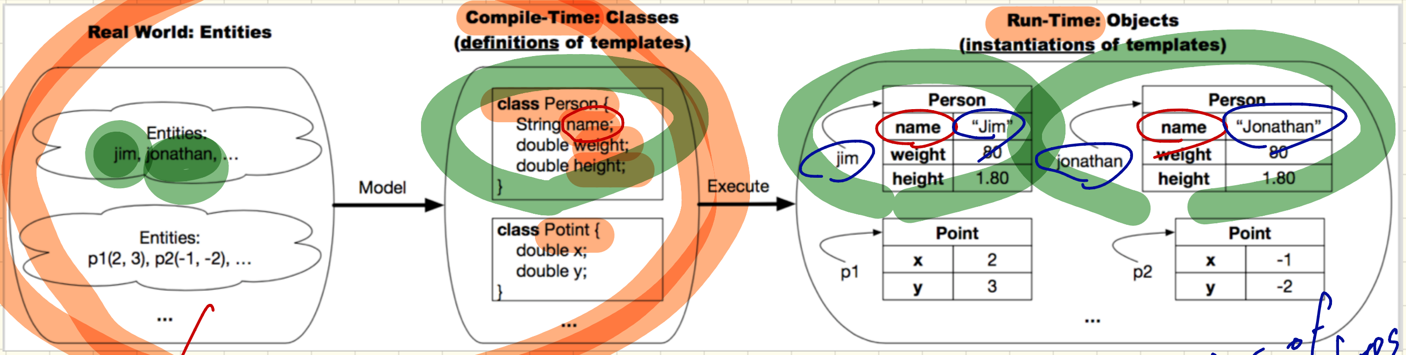
EECS 2030 B Fall 2018

# Advanced Object-Oriented Programming

Wednesday Sep. 5

Lecture I

# The Observe-Model-Execute Process



**Real World: Entities**

Entities:
jim, jonathan, …

Entities:
p1(2, 3), p2(-1, -2), …

…

Model

**Compile-Time: Classes**
**(definitions of templates)**

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

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

…

Execute

**Run-Time: Objects**
**(instantiations of templates)**

jim

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

jonathan

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

p1

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

p2

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

…

critical

nouns → class attribute

verbs. → method

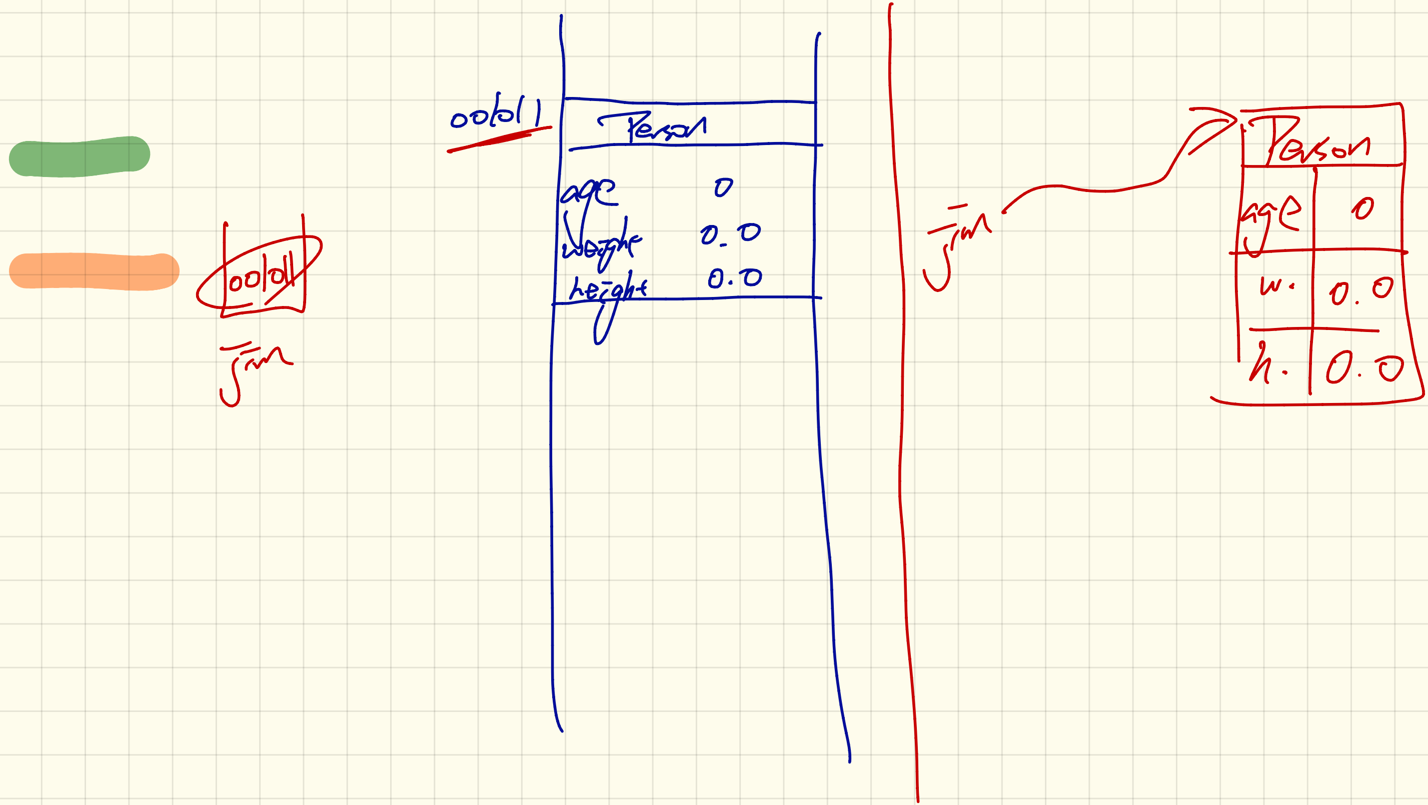
↳ accessor → return some info.

mutator → change the values of attributes of some context object

mutate

constructor → construct a new object

```
Person jim = new Person();
```



Monday Sep. 10

Lecture 2

Lab 0 & 1.1

Lab session today

Study Advice : check moodle announcement.

# Visualizing Person objects at Runtime

```
class Person {
  int age;
  String nationality;
  double weight;
  double height;

  Person(int a, String n, double w, double h) {
    this.age = a;
    this.nationality = n;
    this.weight = w;
    this.height = h;
  }

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

  void gainWeightBy(double units) {
    this.weight = this.weight + units;
  }
}
```

2nd parameter

50

"British"

80

1.8

local var.

jim alan

jim alan

jim alan ↳ class-level variable

bmi

Person alan = new Person(45, "UK", 75, 1.7);

double bmi2 = alan.getBMI();

jim

jim

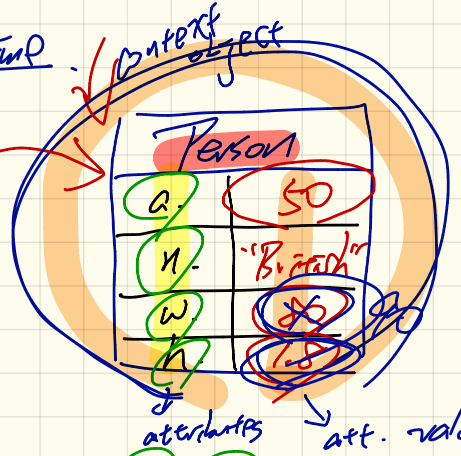
jim.getBMI()

c.o.

jim

## Runtime

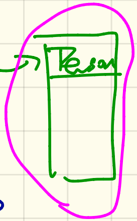
context object



attributes

att. values

alan



look up the mem. location according to the address stored in jim.

## Tester Code

2nd argument value

```
Person jim = new Person(50, "British", 80, 1.8);
double bmi = jim.getBMI();
jim.gainWeightBy(10);
bmi = jim.getBMI();
```

Context object

```
class Person {
  double weight;
  void setWeight(double weight){
      weight = weight;
  }
```

shadow

```
this. weight = weight;
```

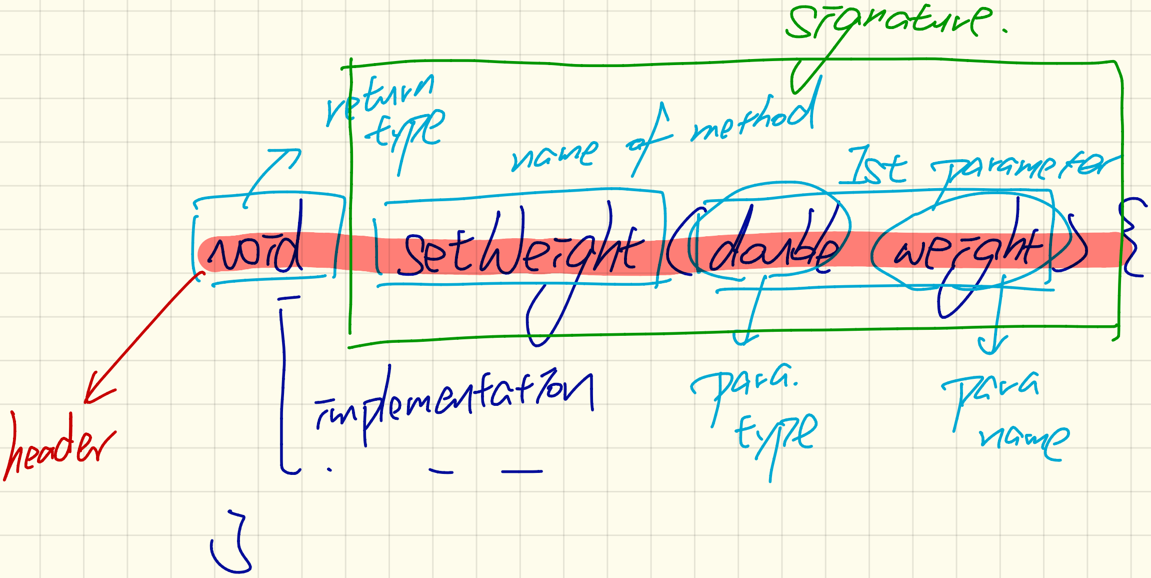
# OOP

## class (compile-time)

- ↳ attributes
- ↳ methods

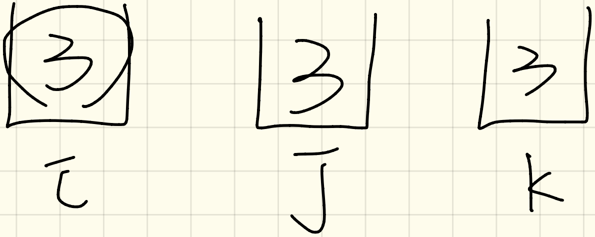
## objects (runtime)

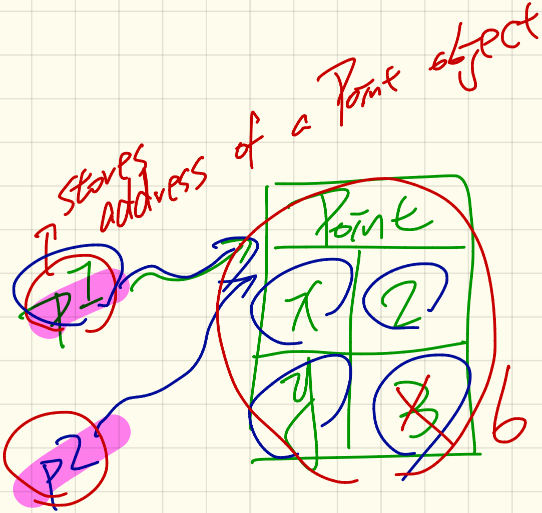
- ↳ objects
- ↳ context object
- ↳ method call



```
void setWeight(double weight) {
    implementation
    . _ _
}
```

header vs. Signature



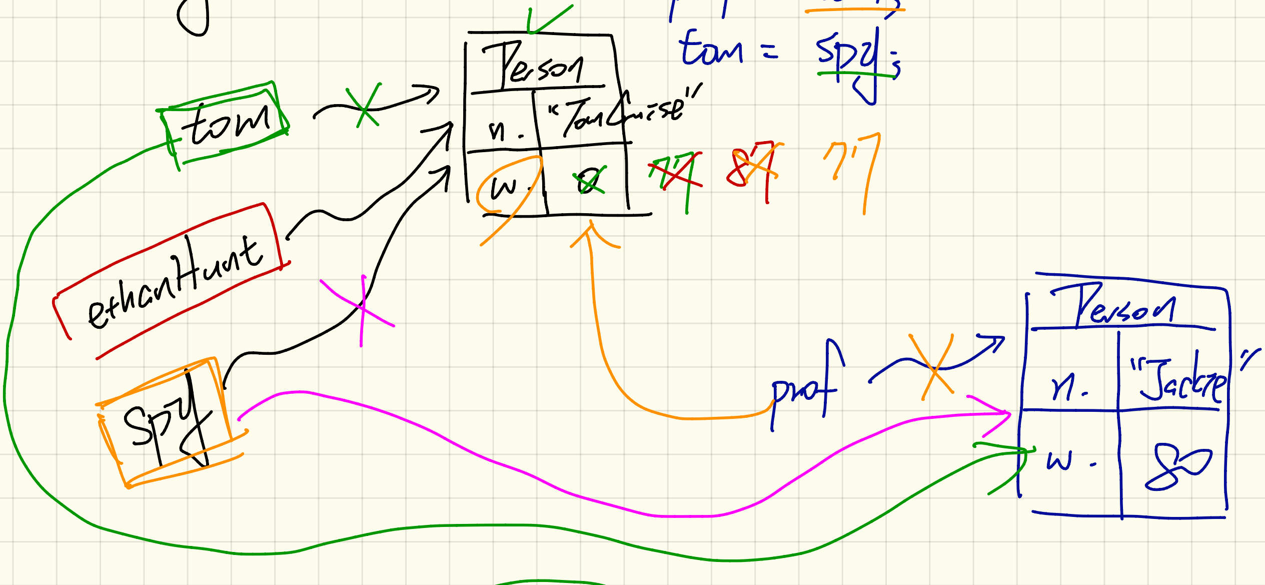


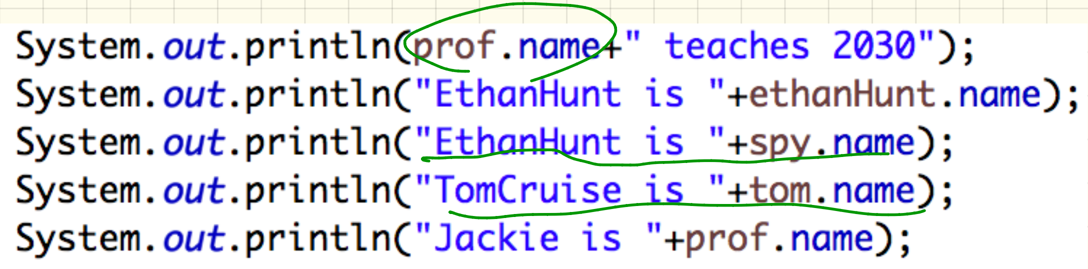
```
Point p2 = p1;
println(p1.x + " " + p1.y);
                 2   3
p2.moveUp(3);
println(p1.x + " " + p1.y);
```

aliasing: as if p1.moveUp(3)

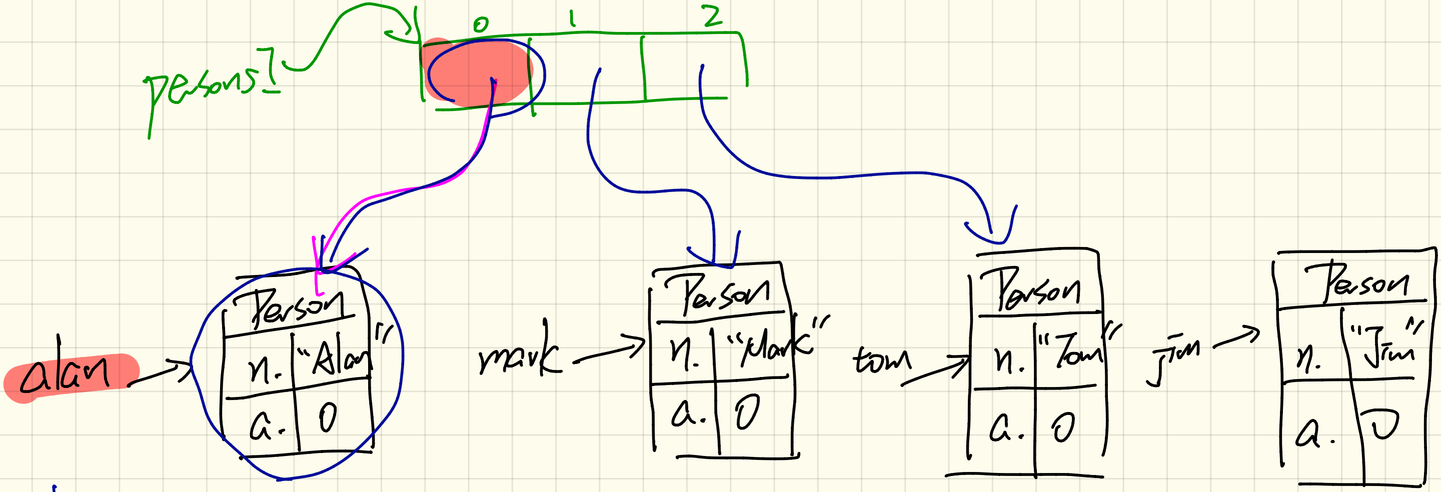
# Aliasing

```
spy = prof;
prof = tom;
tom = spy;
```



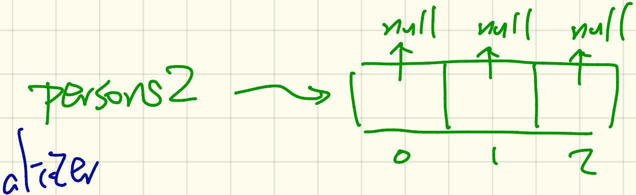


```
System.out.println(prof.name+" teaches 2030");
System.out.println("EthanHunt is "+ethanHunt.name);
System.out.println("EthanHunt is "+spy.name);
System.out.println("TomCruise is "+tom.name);
System.out.println("Jackie is "+prof.name);
```



aliasing

alan == persons1[0]



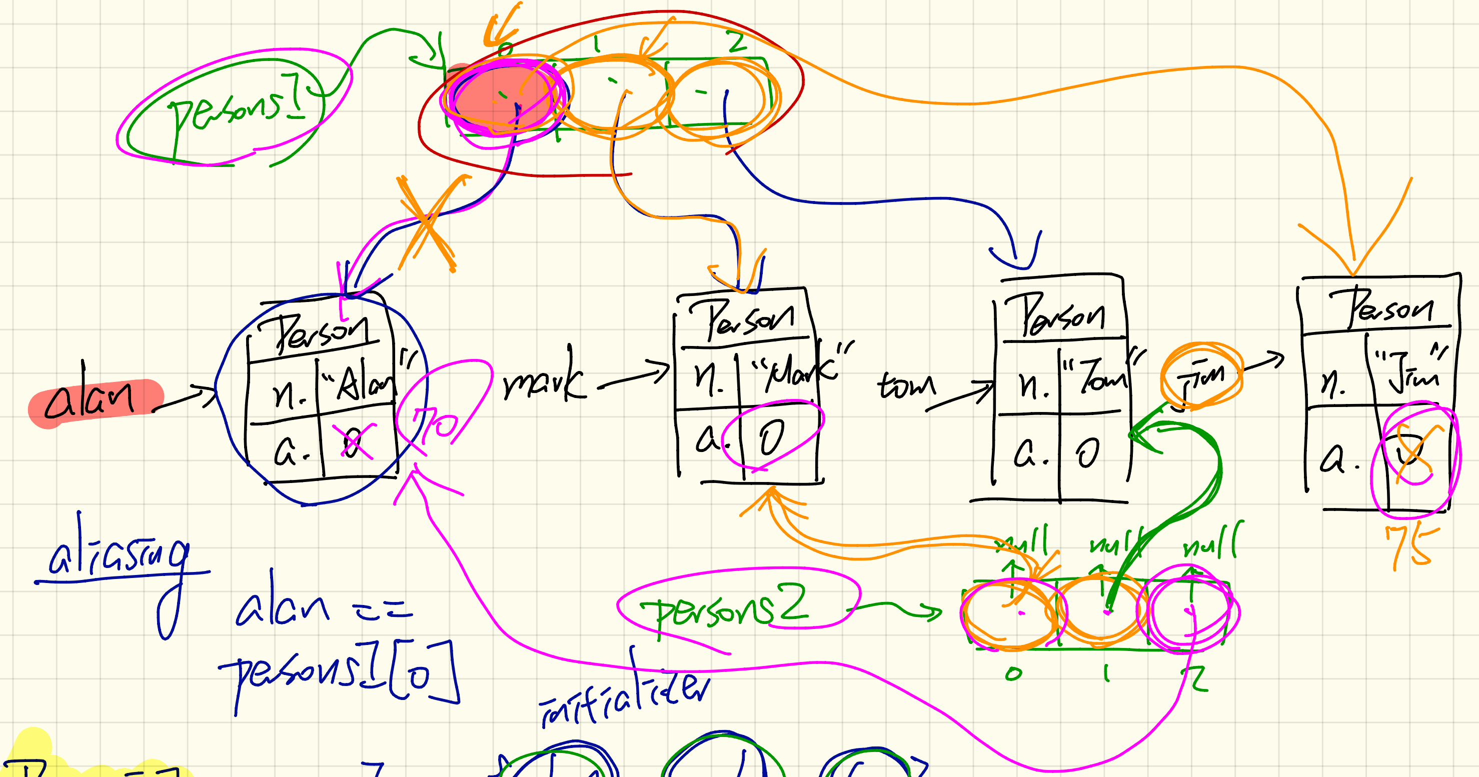
initializer

```
Person[] persons1 = {alan, mark, tom};
↳ Person[] persons1 = new Person[3];
  persons1[0] = alan;   ____[1] = mark;   ____[2] = tom;
```

```
for( int  i = 0 ;  i < persons1.length ;  i++ ){

    persons2[i] =

}
```

Wednesday Sep. 12

Lecture 3



aliasing

alan == persons1[0]

```
Person[] persons1 = {alan, mark, tom};
↳ Person[] persons1 = new Person[3];
  persons1[0] = alan;  ____[1] = mark;  ____[2] = tom;
```

$187 / 3.0$

$x / (double) y$

```
for ( int i = 0 ; i < persons1.length ; i++ ){
    persons2[i] = persons1[(i+1) % persons1.length];
}
```

3

| i | (i+1) % 3 |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | 0 |

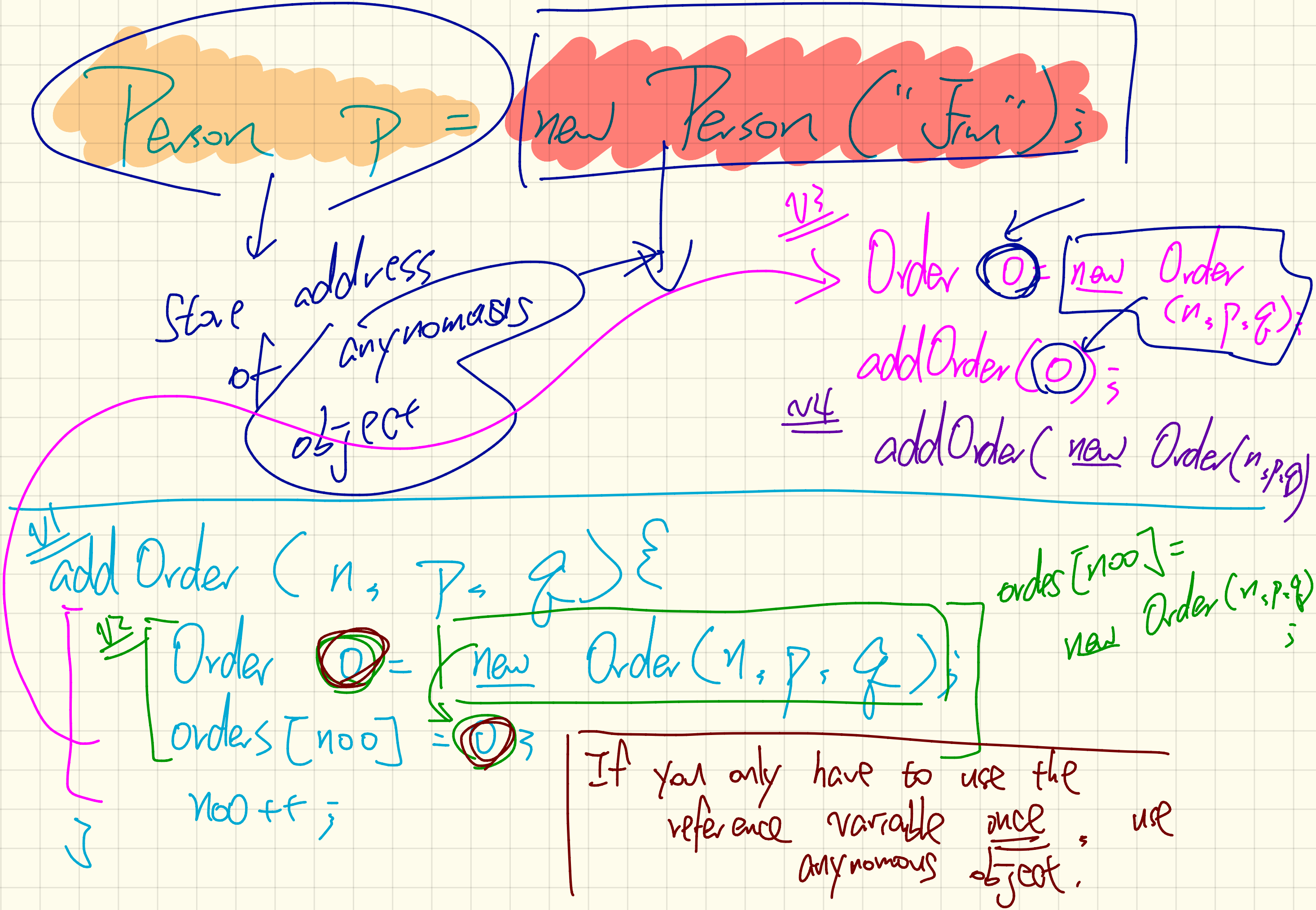
$187 / 3$ ] 62

$187 \% 3$ ] 1

x y integers

$$\underbrace{(x/y)}_{q} * y + (x\%y) = x$$

$$\underbrace{(187/3)}_{q} * 3 + \underbrace{(187 \% 3)}_{r} = 187$$



```
Person p = new Person("Jim");
```

store address of anynomaus object

v3

```
Order o = new Order(n, p, q);
addOrder(o);
```

v4

```
addOrder( new Order(n, p, q))
```

v1

```
addOrder (n, p, q) {
    Order o = new Order(n, p, q);
    orders[noo] = o;
    noo++;
}
```

v2

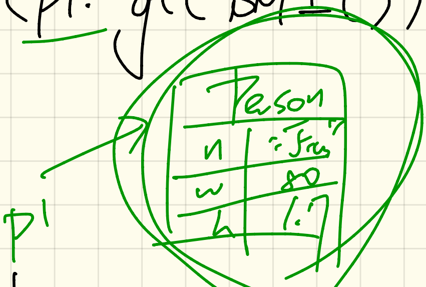
```
orders[noo] = new Order(n, p, q);
```

If you only have to use the reference variable once, use anynomous object.

m1(){ L

```
Person p1 = new Person("Jim", 80, 1.7);

print(p1.getBMI());
print(p1.getBMI());
⋮
```

3

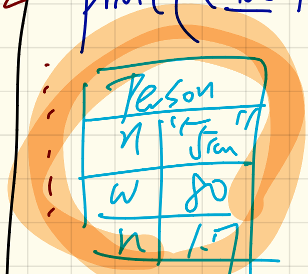


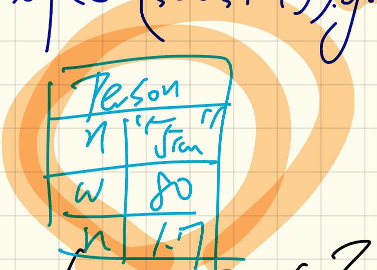
o.m1(); R

p1?

on the fly

```
1 print((new Person("Jim", 80, 1.7)).getBMI());
2 print((new Person("Jim", 80, 1.7)).getBMI());
```





Q1. L and R produce same console output?

Q2. L and R have the same meaning?
(visualization)

int p.t.

Integer r.t.

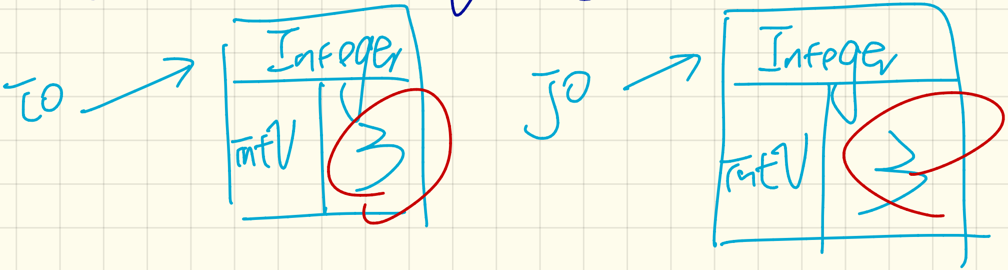
Wrapper

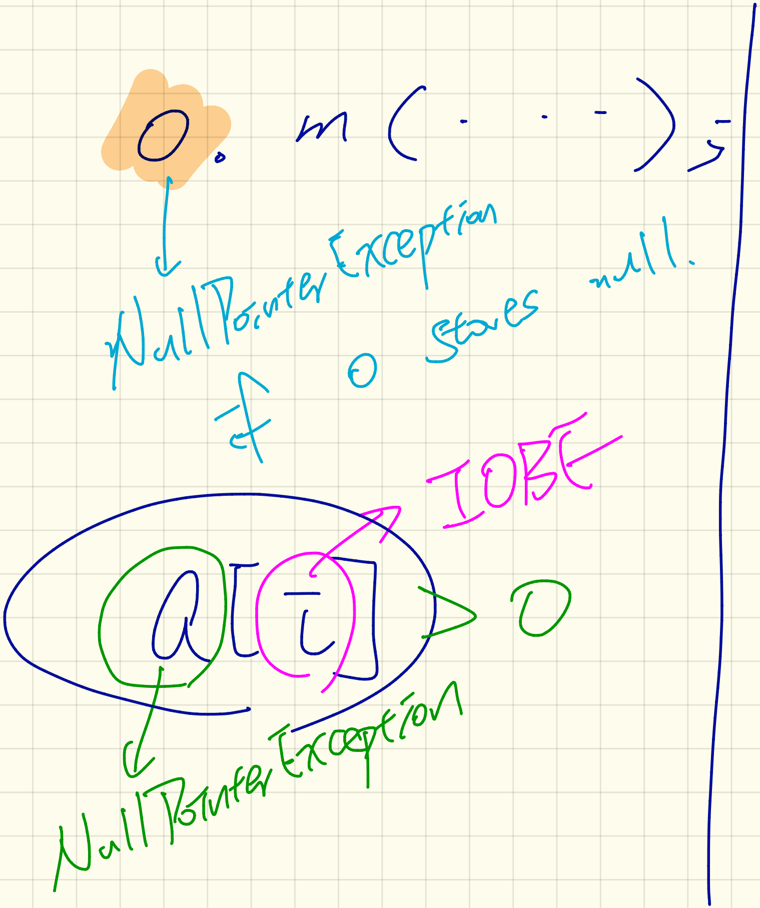
i == j T

i0 == j0 F

i0.equals(j0) T

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

Integer i0 = new Integer(i);
Integer j0 = new Integer(j);
```




o. m( . . . );
NullPointerException
if o stores null.
IOBE
a[i]
> 0
NullPointerException

① a[i] >0

② (0<=i && i<a.length)

③ (a!=null)

Q1. ③ && ② && ①

F

Q2. ② && ① && ③

a null

if a is null

NullPointerException when evaluating a.length

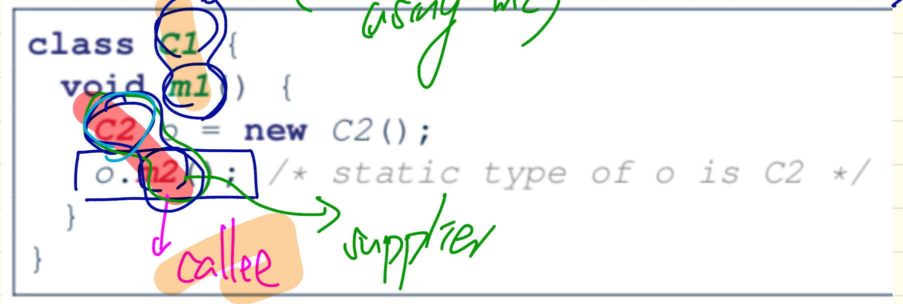
# Caller vs. Callee

caller

client

supplier

caller (client using m2)



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

callee

supplier

Q. Can a method be a caller and a callee simultaneously?

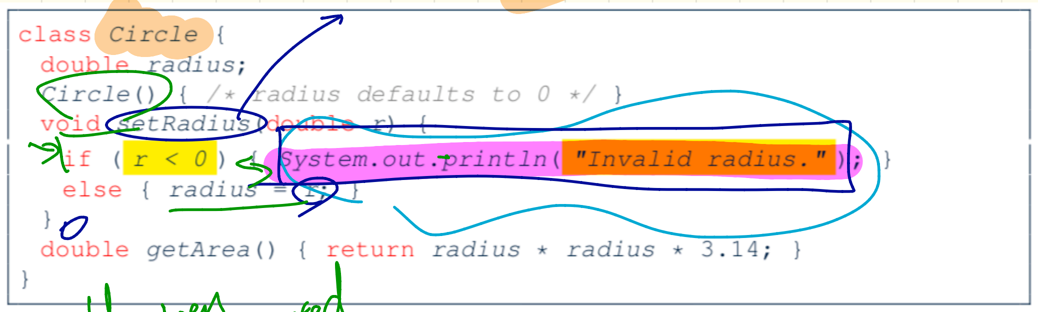
caller

```
class C2 {
  void m2() {
    C3 o = new C3();
    o.m3();
  }
}
```

callee

# Error Handling with Console Messages : Circles

caller or callee?

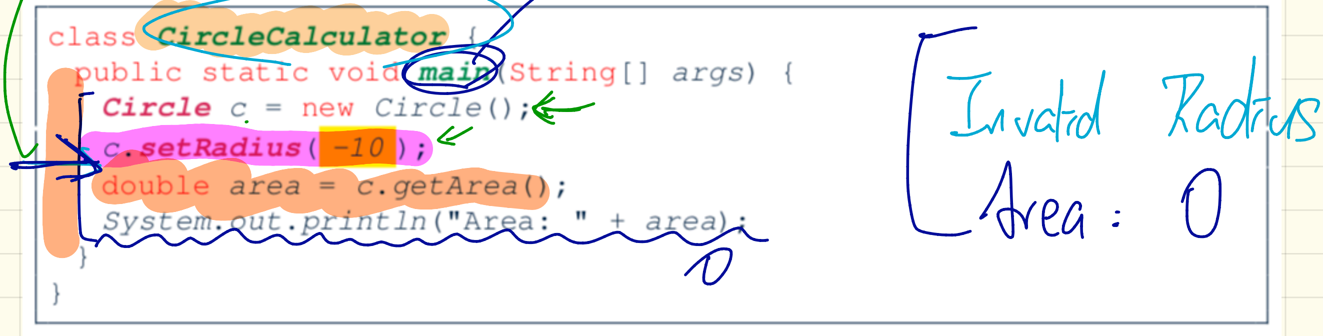


```
class Circle {
  double radius;
  Circle() { /* radius defaults to 0 */ }
  void setRadius(double r) {
   if (r < 0) { System.out.println("Invalid radius."); }
   else { radius = r; }
  }
  double getArea() { return radius * radius * 3.14; }
}
```

0

should have been interrupted.

caller or callee?



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

0

Invalid Radius.
Area: 0

Monday September 17

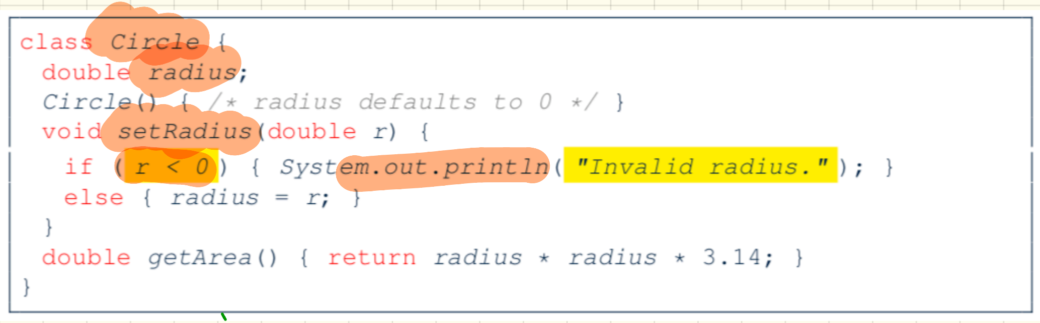
Lecture 4

Lab I part 2

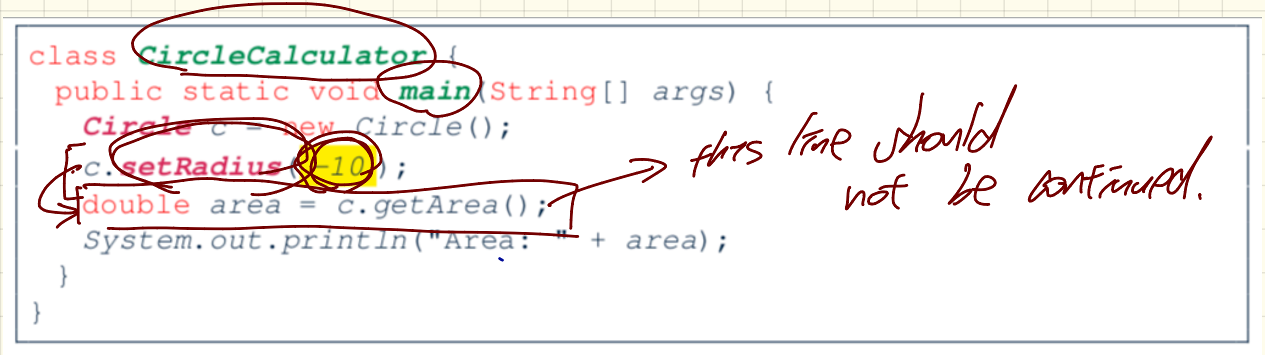
2D arrays

nested loops

# Error Handling with Console Messages: Circles



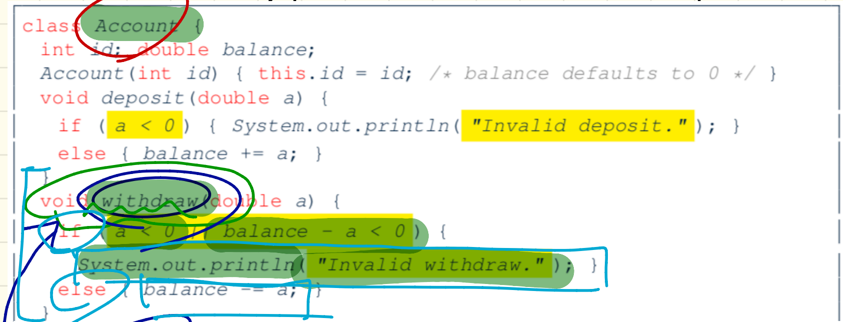
```
class Circle {
  double radius;
  Circle() { /* radius defaults to 0 */ }
  void setRadius(double r) {
    if (r < 0) { System.out.println("Invalid radius."); }
    else { radius = r; }
  }
  double getArea() { return radius * radius * 3.14; }
}
```



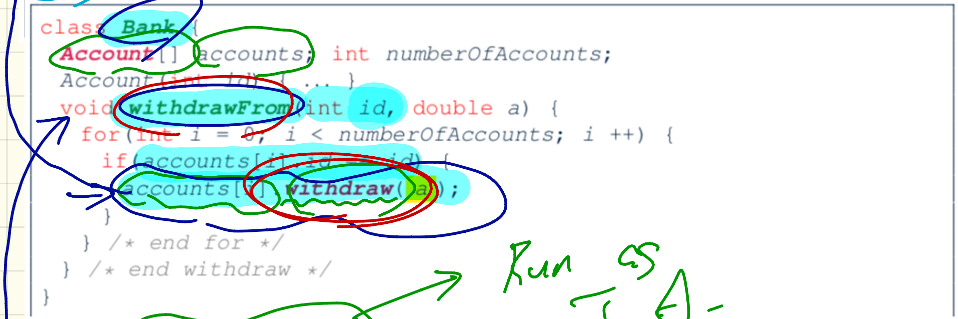
```
class CircleCalculator {
  public static void main(String[] args) {
    Circle c = new Circle();
    c.setRadius(-10);
    double area = c.getArea();
    System.out.println("Area: " + area);
  }
}
```

this line should not be continued.

# Error Handling with Console Messages: Call Chain ✓

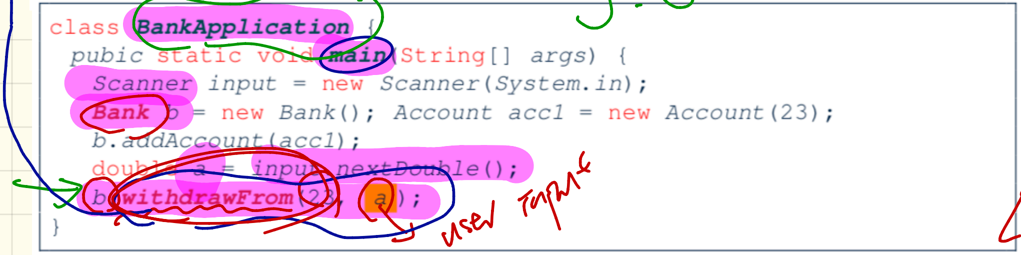


```
class Account {
  int id; double balance;
  Account(int id) { this.id = id; /* balance defaults to 0 */ }
  void deposit(double a) {
    if ( a < 0 ) { System.out.println("Invalid deposit."); }
    else { balance += a; }
  }
  void withdraw(double a) {
    if ( a < 0 || balance - a < 0 ) {
     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



```
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);
      }
    } /* end for */
  } /* end withdraw */
}
```

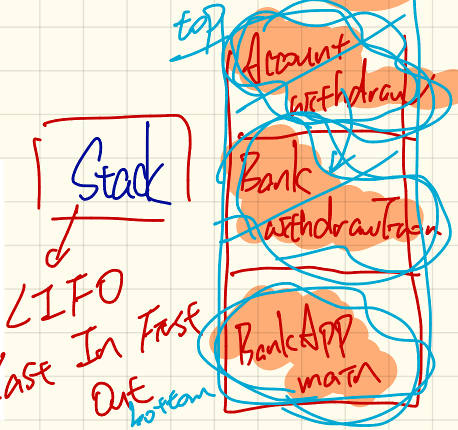
Run as J.A.



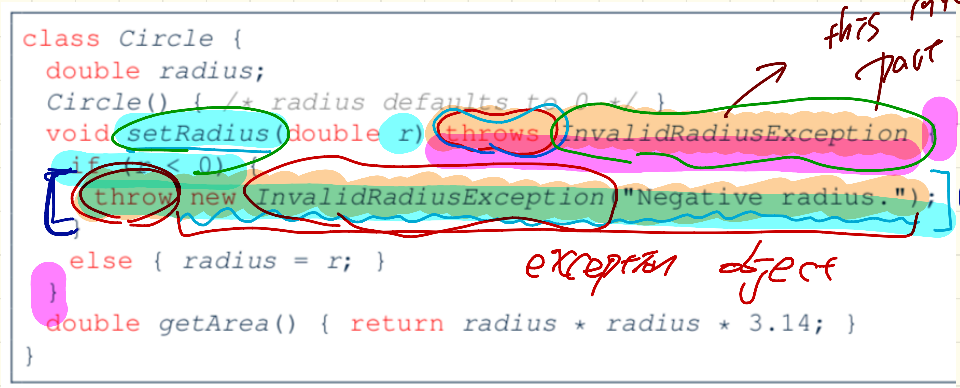
```
class BankApplication {
  pubic static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    Bank b = new Bank(); Account acc1 = new Account(23);
    b.addAccount(acc1);
    double a = input.nextDouble();
    b.withdrawFrom(23, a);
}
```

user input

| context class | caller | callee |
|---|---|---|
| Account | Account withdraw | |
| Bank | Bank withdrawFrom | Account withdraw |
| BankApp | main | Bank withdrawFrom |



# Circle Class with Exceptions (Example 1)

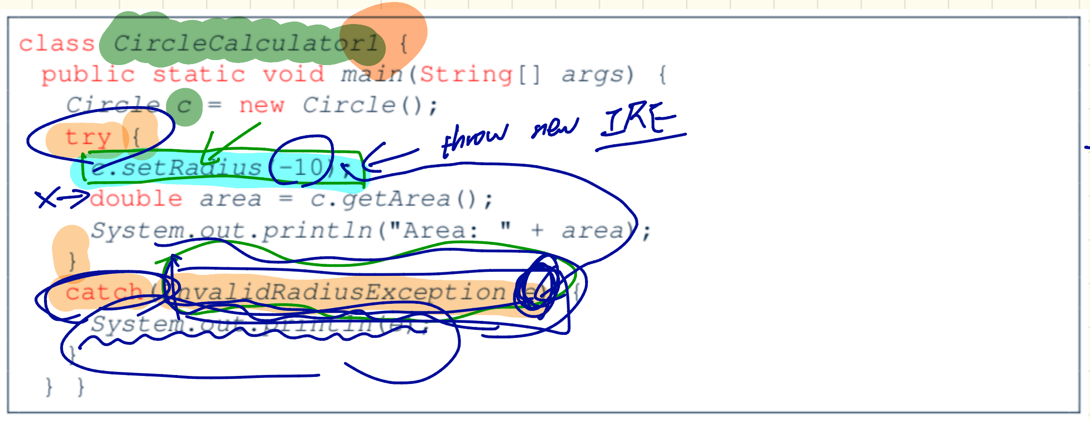


```
class Circle {
  double radius;
  Circle() { /* radius defaults to 0 */ }
  void setRadius(double r) throws InvalidRadiusException {
   if (r < 0) {
    throw new InvalidRadiusException("Negative radius.");
   }
   else { radius = r; }
  }
  double getArea() { return radius * radius * 3.14; }
}
```

this info becomes part of the API to inform the potential caller of setRadius

IRE e = new IRE("...");
throw e;

exception object



```
class CircleCalculator1 {
  public static void main(String[] args) {
    Circle c = new Circle();
    try {
      c.setRadius(-10);
      double area = c.getArea();
      System.out.println("Area: " + area);
    }
    catch(InvalidRadiusException e) {
      System.out.println(e);
    }
  } }
```

throw new IRE

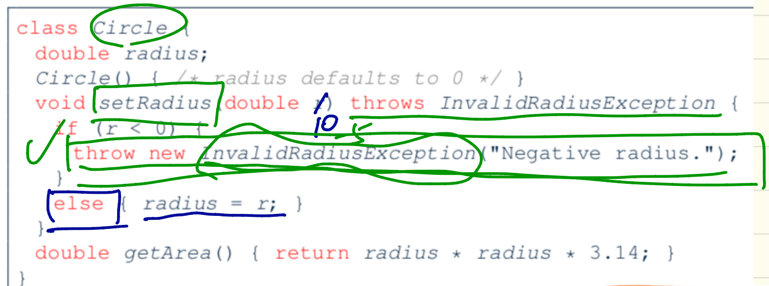
X →

accessor
return
normal

method
throw
abnormal

```
Enter radius:
-10
Invalid radius, try again:
-2
In ________ , t a. :
10
314
```

# Circle Class with Exceptions (Example 2)



```
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; }
}
```

catch (IRE e) {

e

Syso

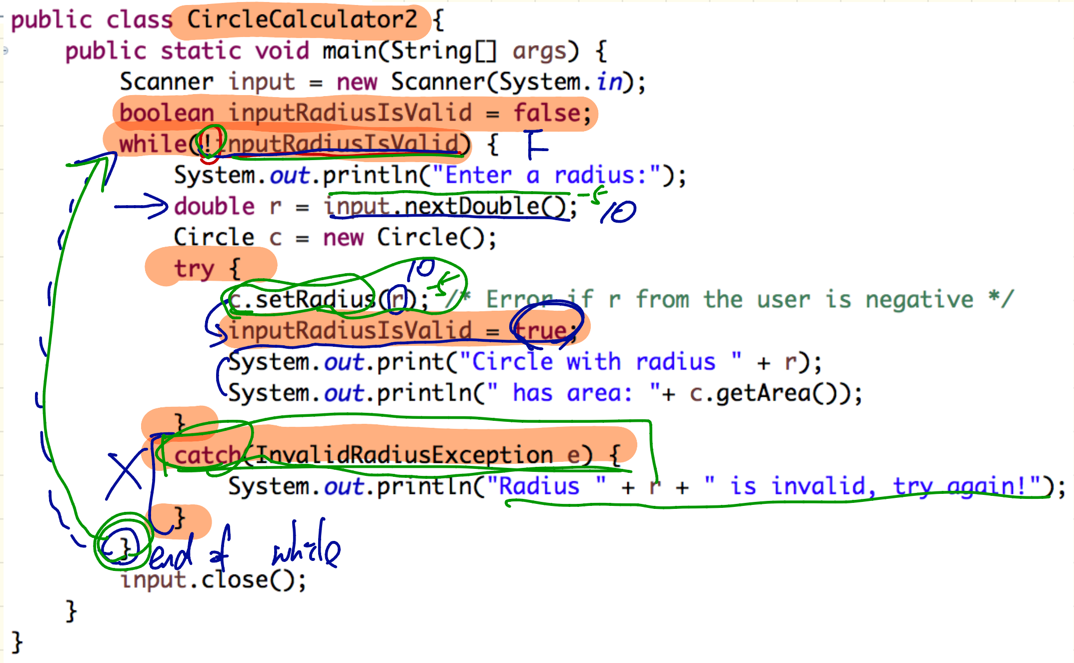
}

Case 1

User enters 10

Case 2

User enters -5

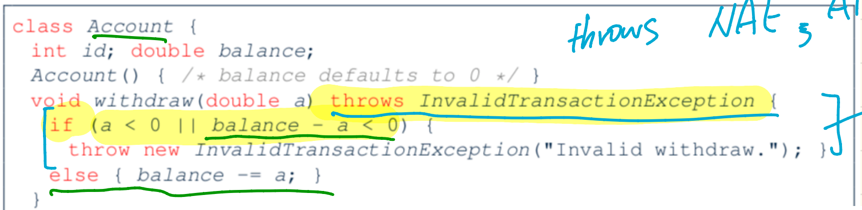


```
public class CircleCalculator2 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        boolean inputRadiusIsValid = false;
        while(!inputRadiusIsValid) {
            System.out.println("Enter a radius:");
            double r = input.nextDouble();
            Circle c = new Circle();
            try {
                c.setRadius(r); /* Error if r from the user is negative */
                inputRadiusIsValid = true;
                System.out.print("Circle with radius " + r);
                System.out.println(" has area: "+ c.getArea());
            }
            catch(InvalidRadiusException e) {
                System.out.println("Radius " + r + " is invalid, try again!");
            }
        }
        input.close();
    }
}
```

# Bank Example with Exceptions

NAE: NegativeAmountException

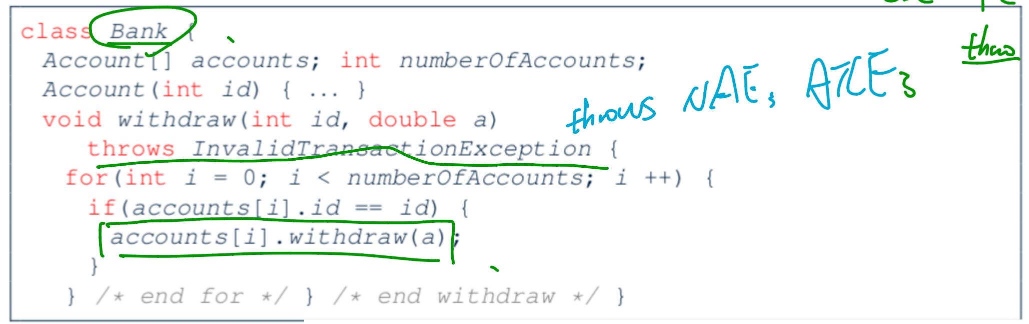
ATLE: AmountTooLargeException



```
class Account {
  int id; double balance;
  Account() { /* balance defaults to 0 */ }
  void withdraw(double a) throws InvalidTransactionException {
   if (a < 0 || balance - a < 0) {
    throw new InvalidTransactionException("Invalid withdraw."); }
   else { balance -= a; }
  }
}
```

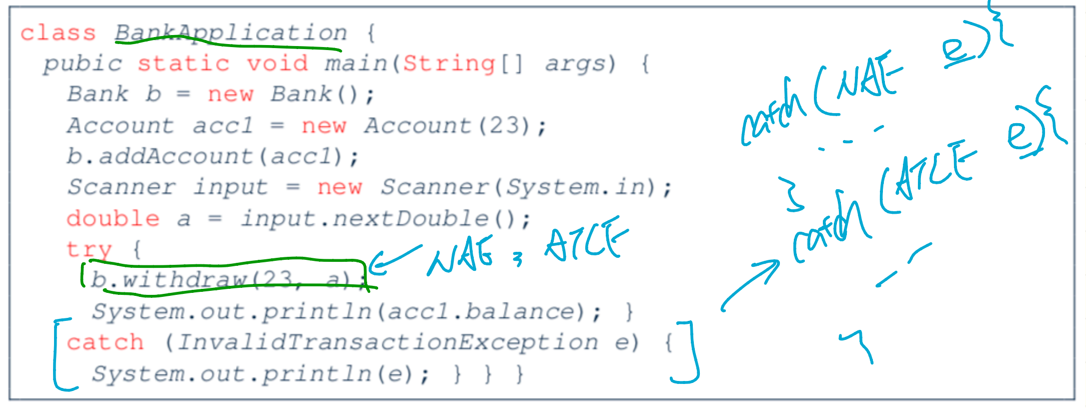
throws NAE, ATLE

```
if (a < 0) {
  throw new NAE("neg. a.");
}
else if (balance - a < 0) {
  throw new ATLE("too lar.");
}
```



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

throws NAE, ATLE

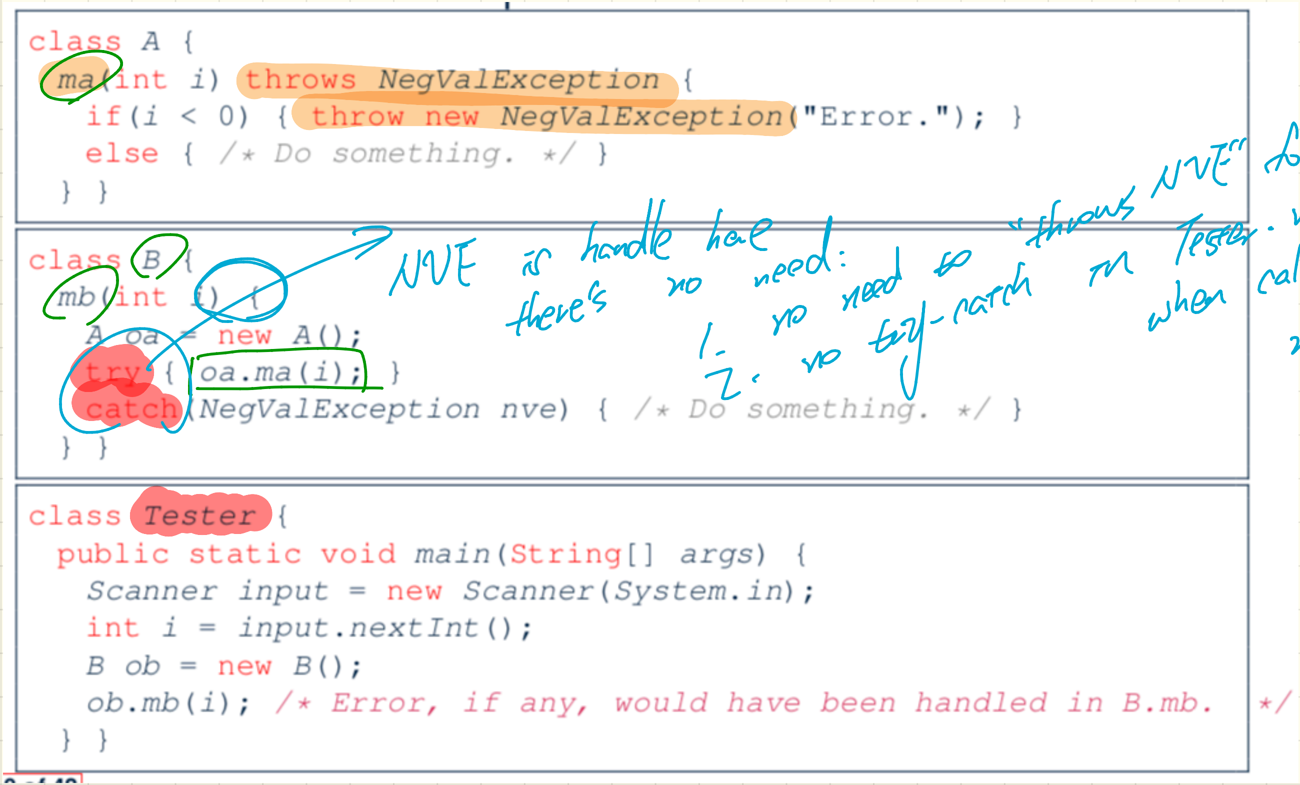


```
class BankApplication {
  pubic static void main(String[] args) {
   Bank b = new Bank();
   Account acc1 = new Account(23);
   b.addAccount(acc1);
   Scanner input = new Scanner(System.in);
   double a = input.nextDouble();
   try {
    b.withdraw(23, a);
    System.out.println(acc1.balance); }
   catch (InvalidTransactionException e) {
    System.out.println(e); } } }
```

NAE, ATLE

```
catch (NAE e) {
  ...
}
catch (ATLE e) {
  ...
}
```

# To Handle or Not To Handle : VI



```
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. */
  } }
```

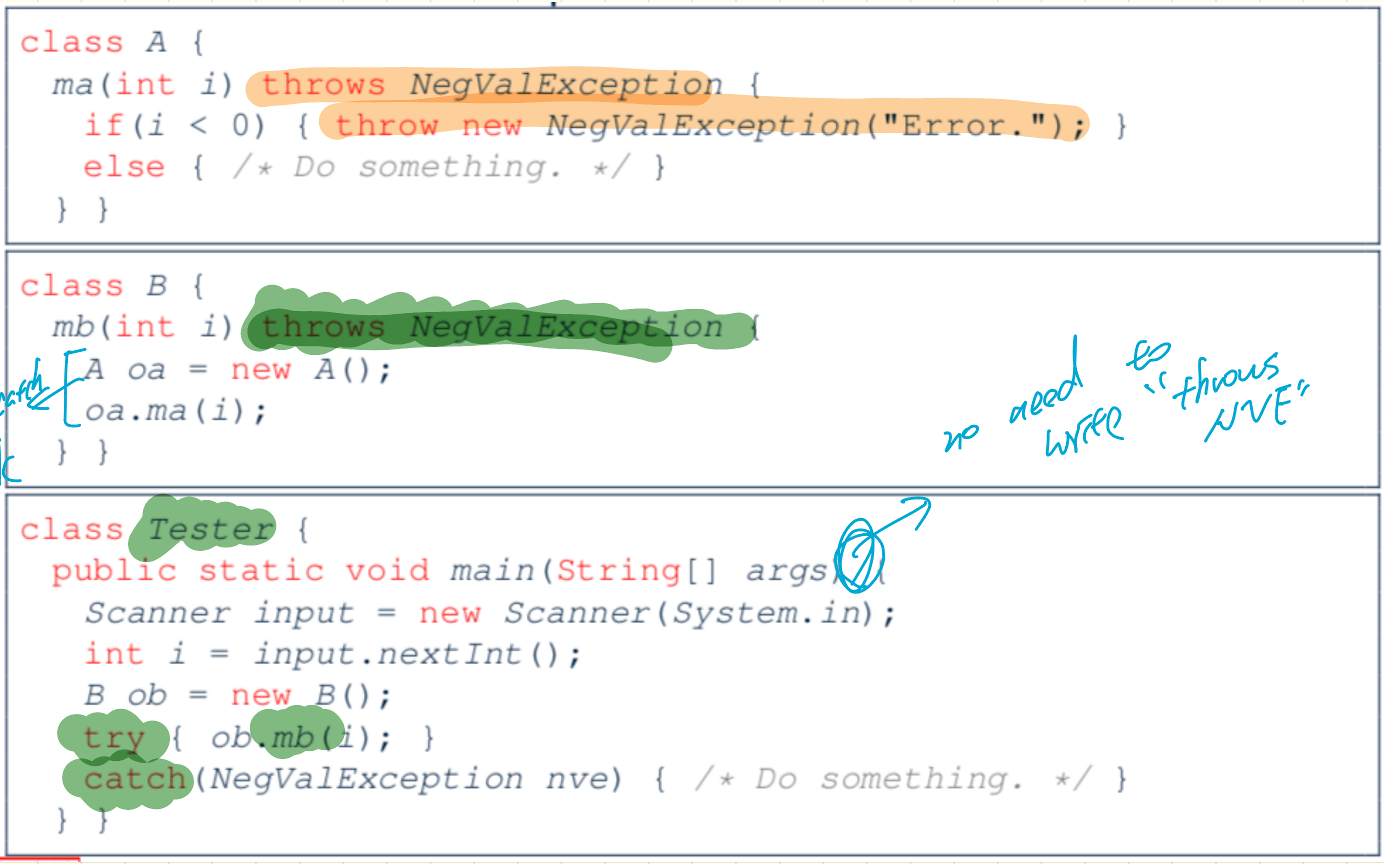
NVE is handle here

there's no need:

1. no need to "throws NVE" for mb

2. no try-catch in Tester.main when calling mb

# To Handle or Not To Handle : V2



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

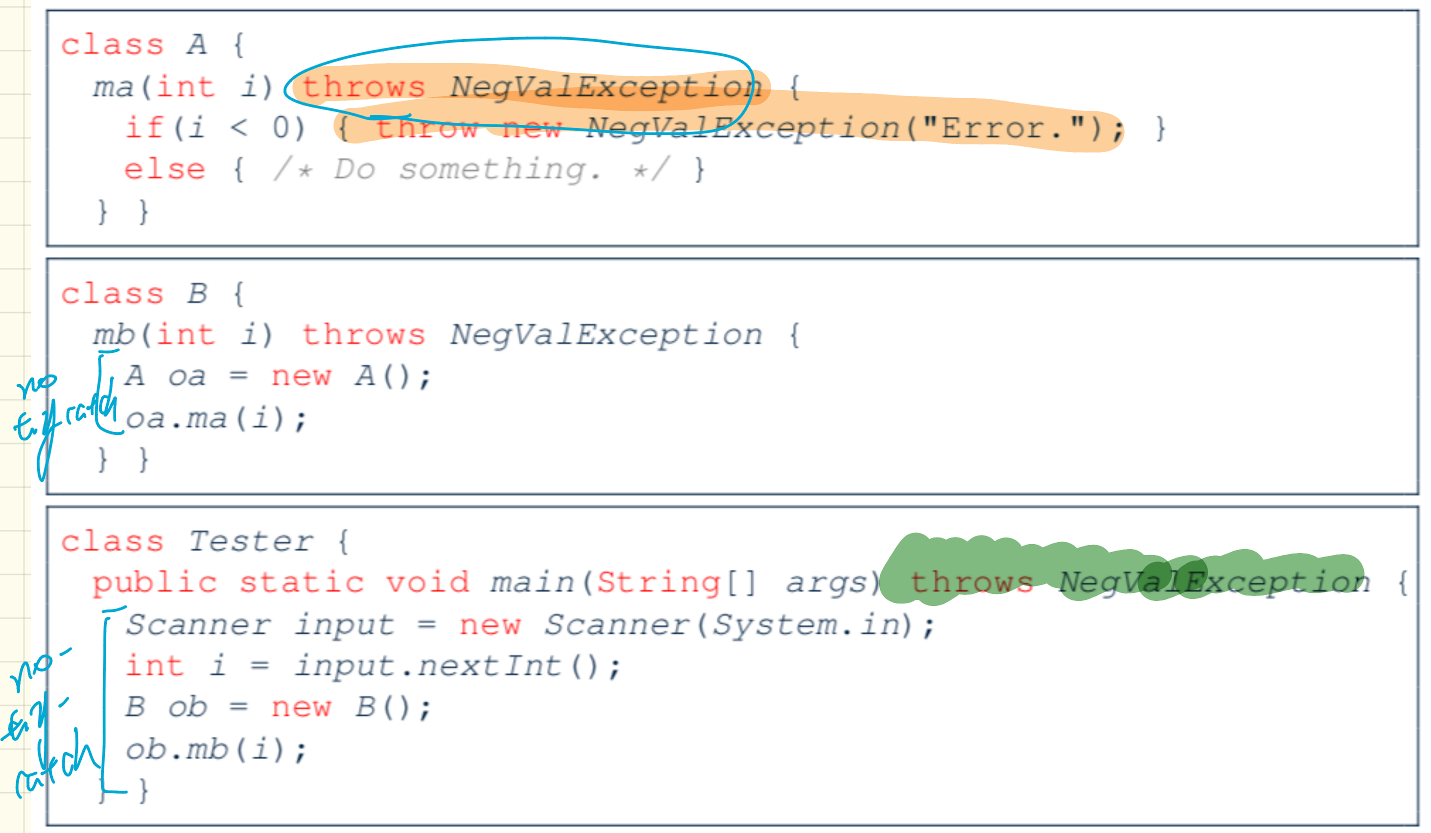
```
class B {
  mb(int i) throws NegValException {
    A oa = new A();
    oa.ma(i);
  } }
```

no try-catch block

no need to write "throws NVE"

```
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. */ }
  } }
```

# To Handle or Not To Handle : V3



```
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);
  } }
```

no try-catch

```
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);
  } }
```

no-try-catch

Integer.parseInt("256");

↳ 256

Integer.parseInt(" two");

↳ NFE

Wednesday Sep. 19

Lecture 5

Lab Test I : Oct. 7

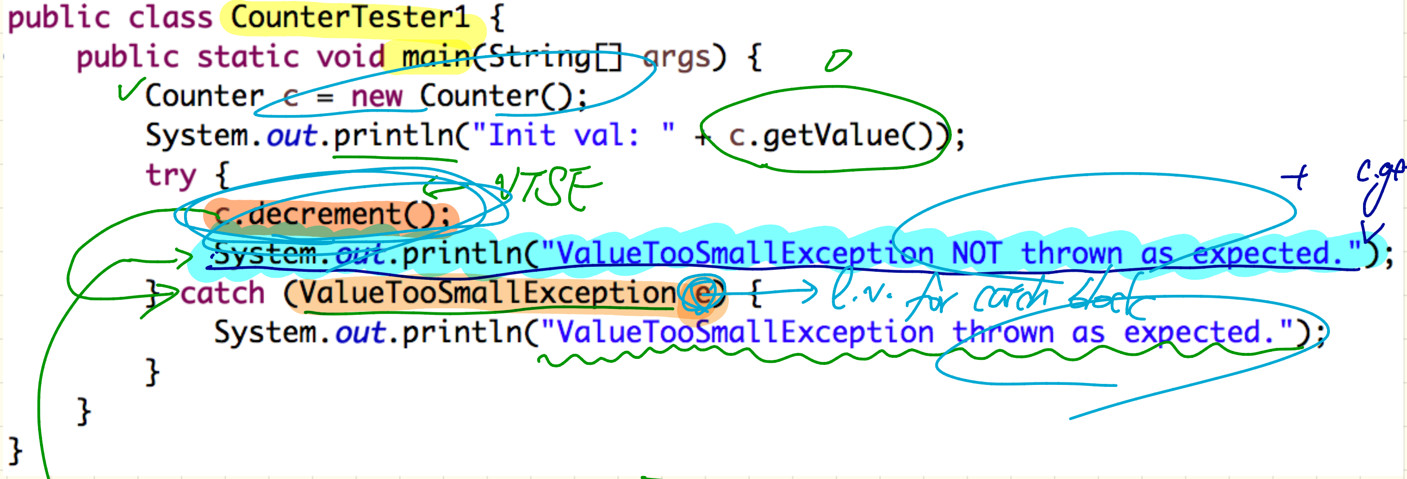
Slides:

- Classes & Objects
- Exceptions
- JUnit
- Monday Sep. 24

Programming:

- Lab 1 (2D arrays, nested loops)
- Practice Problem

# Testing from Console (V1) : Test 1



```
public class CounterTester1 {
    public static void main(String[] args) {
        Counter c = new Counter();
        System.out.println("Init val: " + c.getValue());
        try {
            c.decrement();
            System.out.println("ValueTooSmallException NOT thrown as expected.");
        } catch (ValueTooSmallException e) {
            System.out.println("ValueTooSmallException thrown as expected.");
        }
    }
}
```

0

VTSE

+ c.getV()

l.v. for catch block

VTSE did not occur

c 0

inc 1

inc 2

inc 3

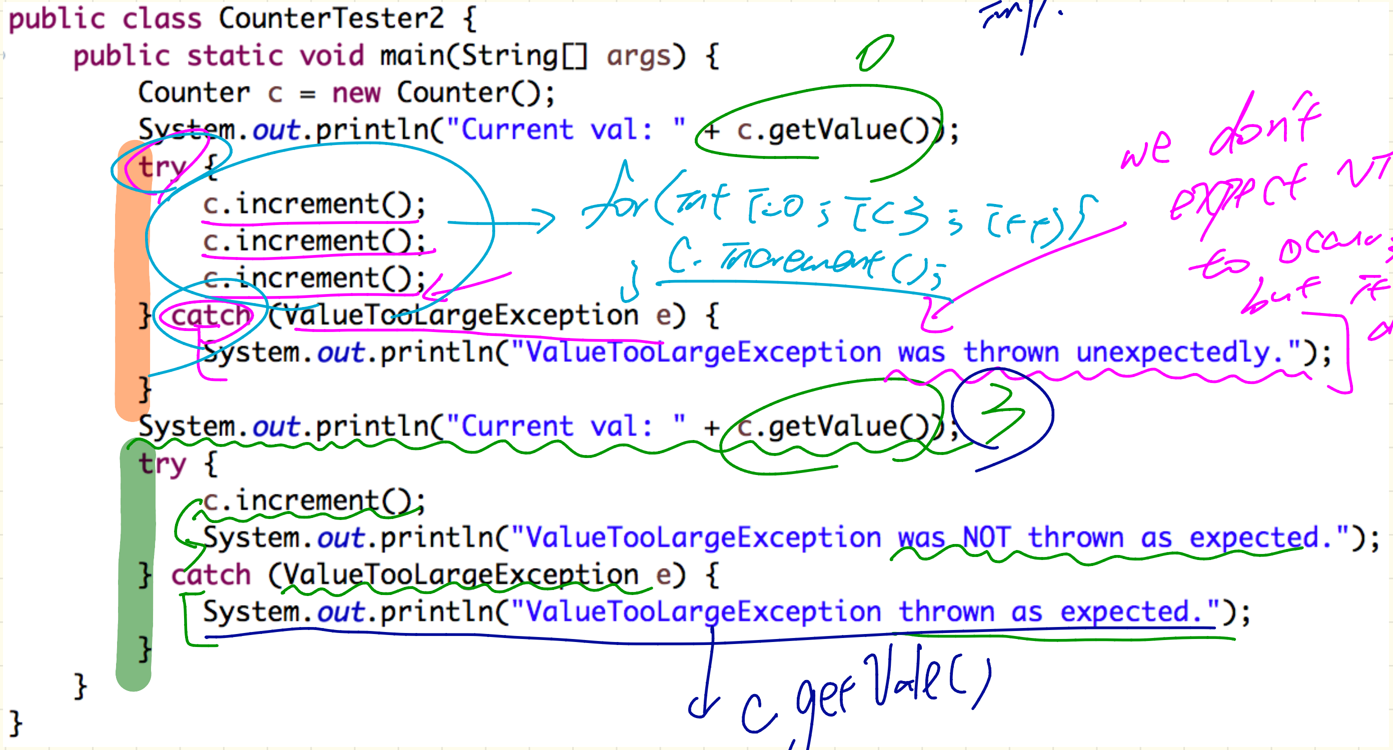
inc

no VTCE expected

VTCE expected

# Testing from Console (V1): Test 2

assume: correct imp.



```
public class CounterTester2 {
    public static void main(String[] args) {
        Counter c = new Counter();
        System.out.println("Current val: " + c.getValue());
        try {
            c.increment();
            c.increment();
            c.increment();
        } catch (ValueTooLargeException e) {
            System.out.println("ValueTooLargeException was thrown unexpectedly.");
        }
        System.out.println("Current val: " + c.getValue());
        try {
            c.increment();
            System.out.println("ValueTooLargeException was NOT thrown as expected.");
        } catch (ValueTooLargeException e) {
            System.out.println("ValueTooLargeException thrown as expected.");
        }
    }
}
```

0

for(int i=0; i<3; i++){
c.increment();
}

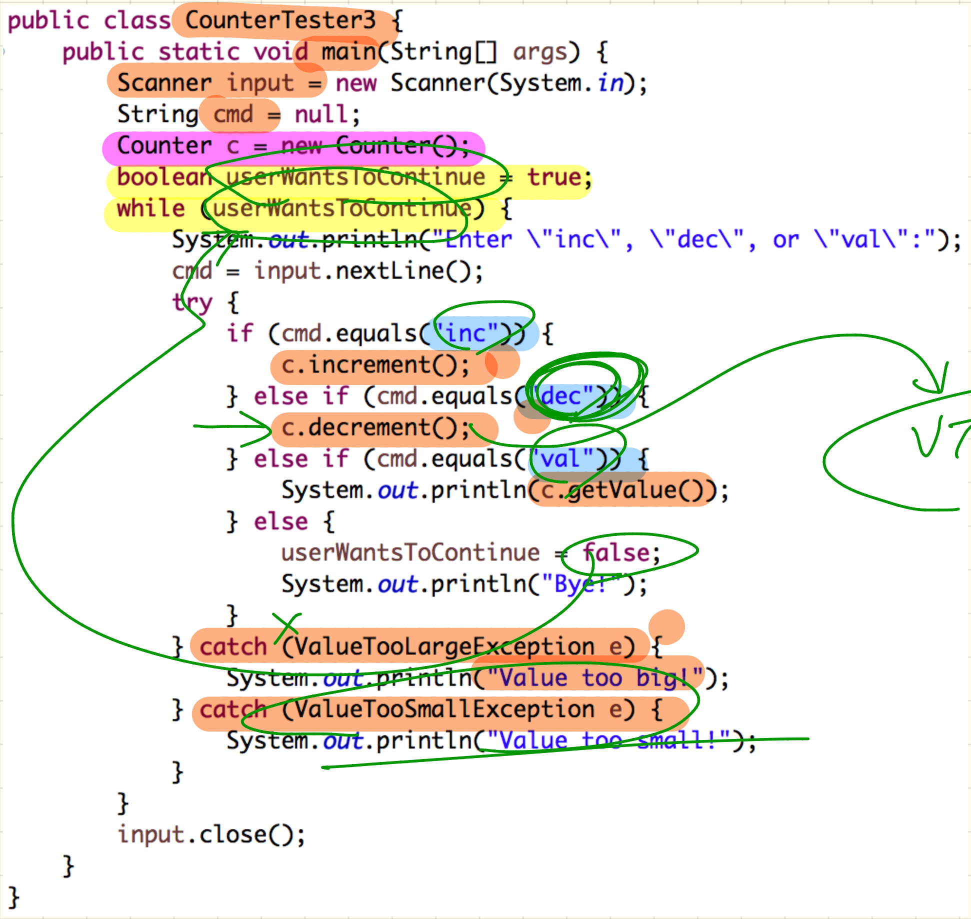
we don't expect VTLE to occur, but it did.

3

c.getValue()

3

# Testing from Console (V2)



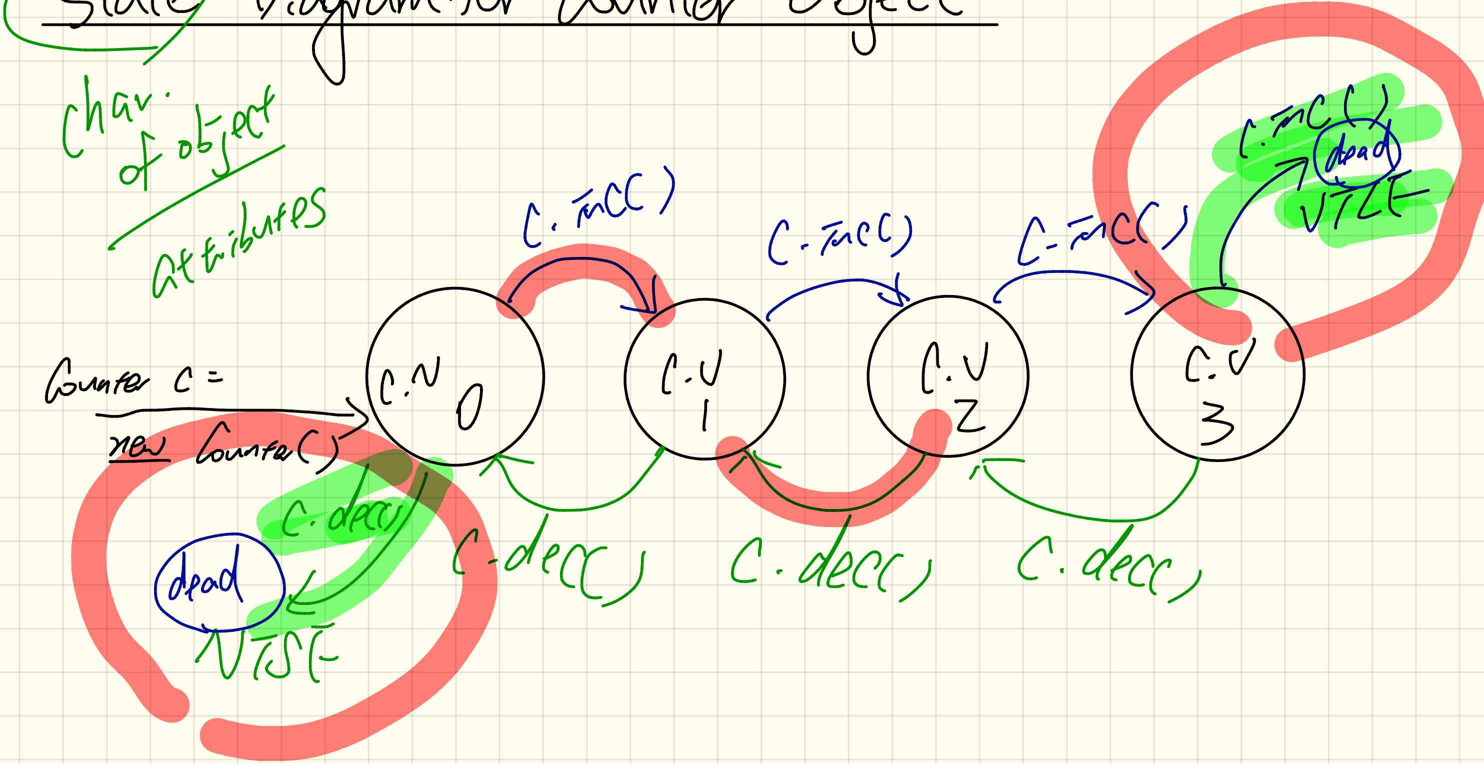
```java
public class CounterTester3 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        String cmd = null;
        Counter c = new Counter();
        boolean userWantsToContinue = true;
        while (userWantsToContinue) {
            System.out.println("Enter \"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")) {
                    System.out.println(c.getValue());
                } else {
                    userWantsToContinue = false;
                    System.out.println("Bye!");
                }
            } catch (ValueTooLargeException e) {
                System.out.println("Value too big!");
            } catch (ValueTooSmallException e) {
                System.out.println("Value too small!");
            }
        }
        input.close();
    }
}
```

VTSE

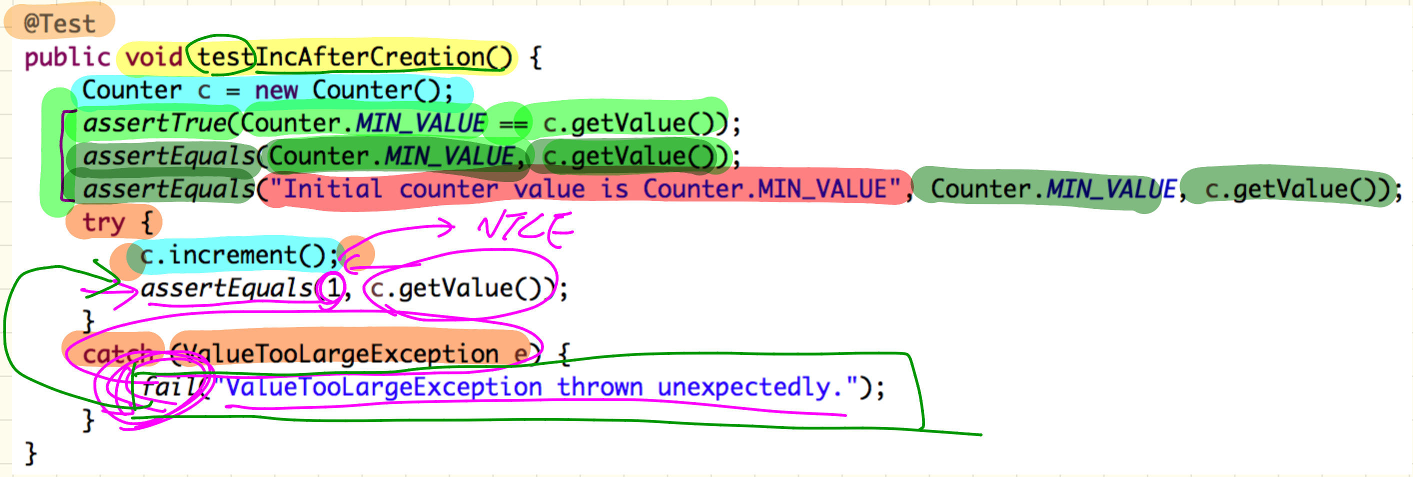
# State Diagram for a Counter Object

char. of object

attributes



# JUnit Test Case I



```
@Test
public void testIncAfterCreation() {
  Counter c = new Counter();
  assertTrue(Counter.MIN_VALUE == c.getValue());
  assertEquals(Counter.MIN_VALUE, c.getValue());
  assertEquals("Initial counter value is Counter.MIN_VALUE", Counter.MIN_VALUE, c.getValue());
  try {
    c.increment();
    assertEquals(1, c.getValue());
  }
  catch (ValueTooLargeException e) {
    fail("ValueTooLargeException thrown unexpectedly.");
  }
}
```

NICE

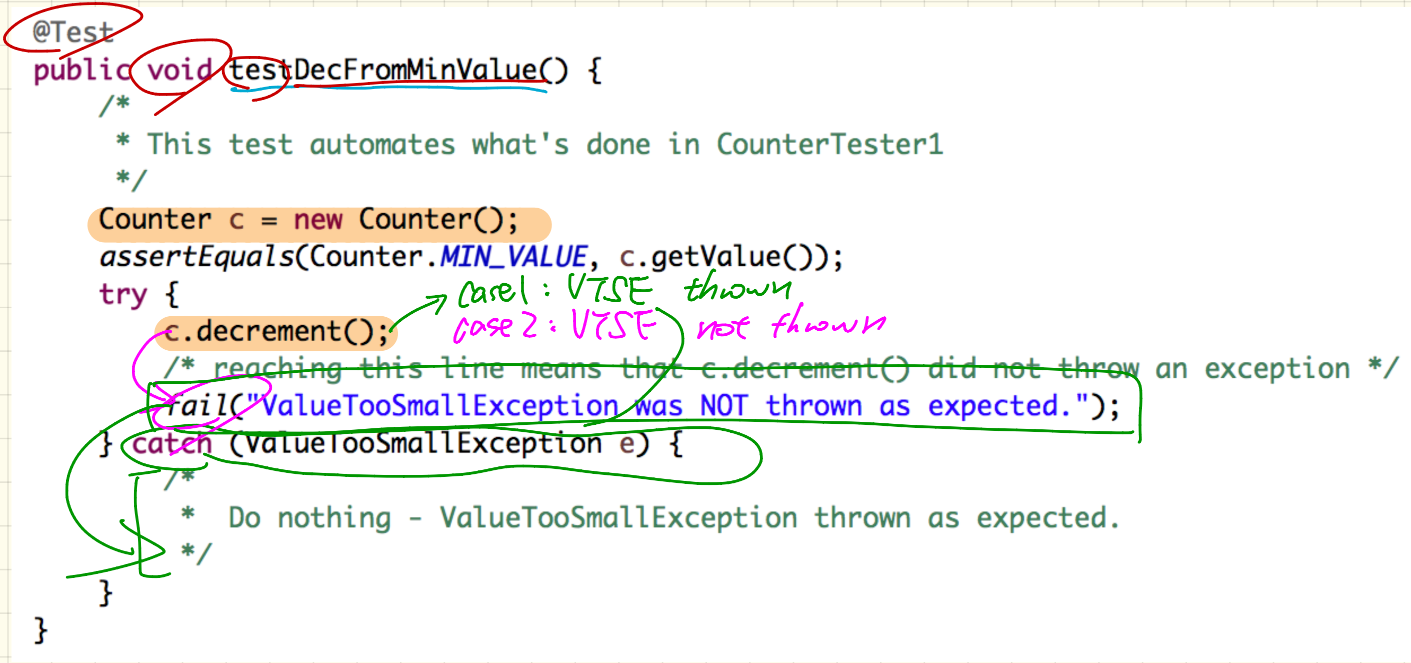
assertEquals( ______ , ______ )

expect

actual

mutator

# JUnit Test Case 2

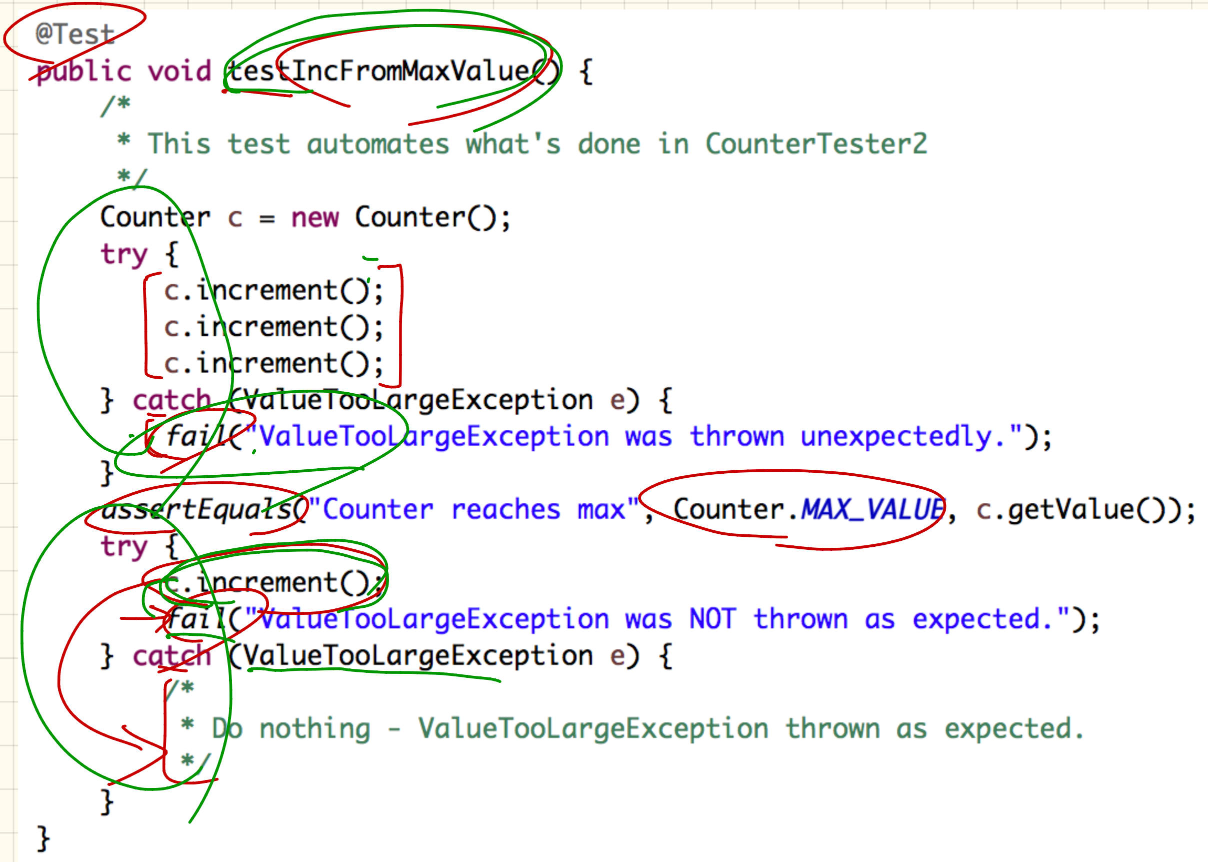


```
@Test
public void testDecFromMinValue() {
    /*
     * This test automates what's done in CounterTester1
     */
    Counter c = new Counter();
    assertEquals(Counter.MIN_VALUE, c.getValue());
    try {
        c.decrement();
        /* reaching this line means that c.decrement() did not throw an exception */
        fail("ValueTooSmallException was NOT thrown as expected.");
    } catch (ValueTooSmallException e) {
        /*
         *   Do nothing - ValueTooSmallException thrown as expected.
         */
    }
}
```

case1 : VTSE thrown

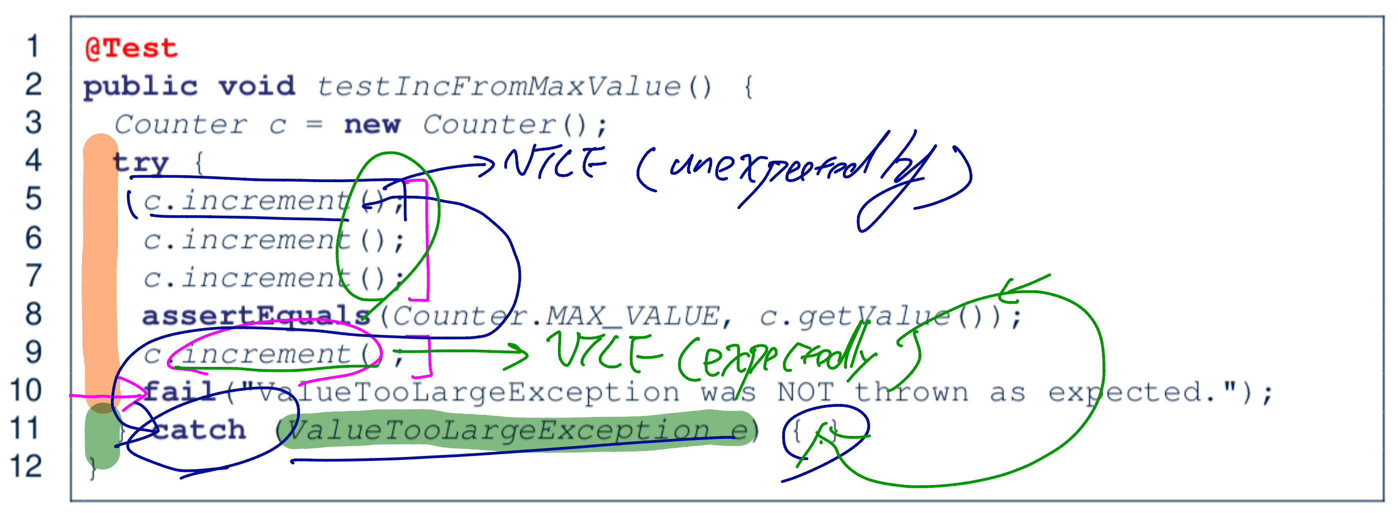
case 2 : VTSE not thrown

# JUnit Test Case 3



```java
@Test
public void testIncFromMaxValue() {
    /*
     * This test automates what's done in CounterTester2
     */
    Counter c = new Counter();
    try {
        c.increment();
        c.increment();
        c.increment();
    } catch (ValueTooLargeException e) {
        fail("ValueTooLargeException was thrown unexpectedly.");
    }
    assertEquals("Counter reaches max", Counter.MAX_VALUE, c.getValue());
    try {
        c.increment();
        fail("ValueTooLargeException was NOT thrown as expected.");
    } catch (ValueTooLargeException e) {
        /*
         * Do nothing - ValueTooLargeException thrown as expected.
         */
    }
}
```

# Question: Is this alternative version appropriate?



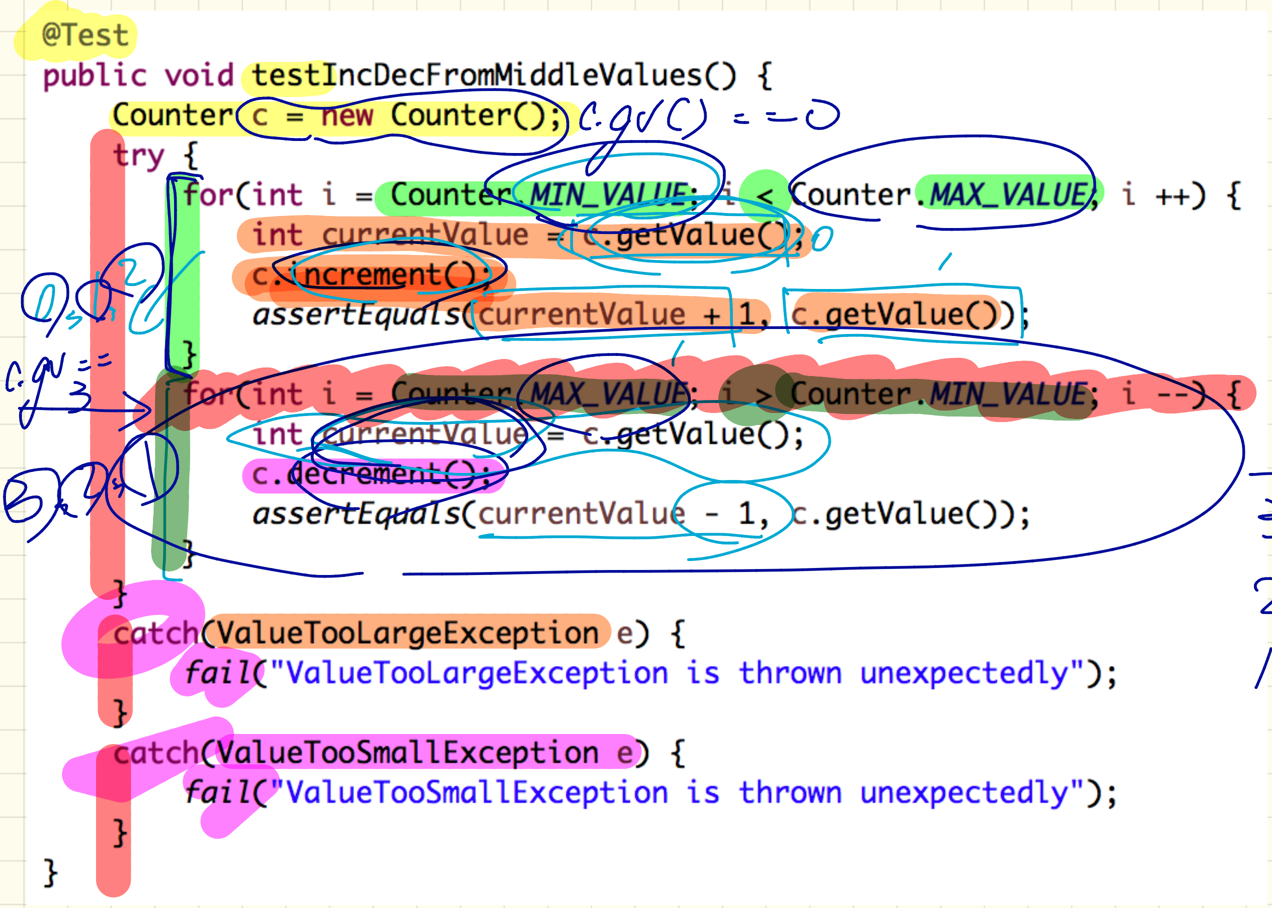
```
@Test
public void testIncFromMaxValue() {
  Counter c = new Counter();
  try {
    c.increment();
    c.increment();
    c.increment();
    assertEquals(Counter.MAX_VALUE, c.getValue());
    c.increment();
    fail("ValueTooLargeException was NOT thrown as expected.");
  } catch (ValueTooLargeException e) { }
}
```

Monday Sep. 24

Lecture 6

- Mandatory Lab Session today
  (submission within 20 minutes)
- Lab Test I Guide
  - ~ Birthday Book
  - ~ Encapsulation
  - ~ Expectation & Strategy
  - ~ equals method

# JUnit Test Case 4



```java
@Test
public void testIncDecFromMiddleValues() {
    Counter c = new Counter();
    try {
        for(int i = Counter.MIN_VALUE; i < Counter.MAX_VALUE; i ++) {
            int currentValue = c.getValue();
            c.increment();
            assertEquals(currentValue + 1, c.getValue());
        }
        for(int i = Counter.MAX_VALUE; i > Counter.MIN_VALUE; i --) {
            int currentValue = c.getValue();
            c.decrement();
            assertEquals(currentValue - 1, c.getValue());
        }
    }
    catch(ValueTooLargeException e) {
        fail("ValueTooLargeException is thrown unexpectedly");
    }
    catch(ValueTooSmallException e) {
        fail("ValueTooSmallException is thrown unexpectedly");
    }
}
```

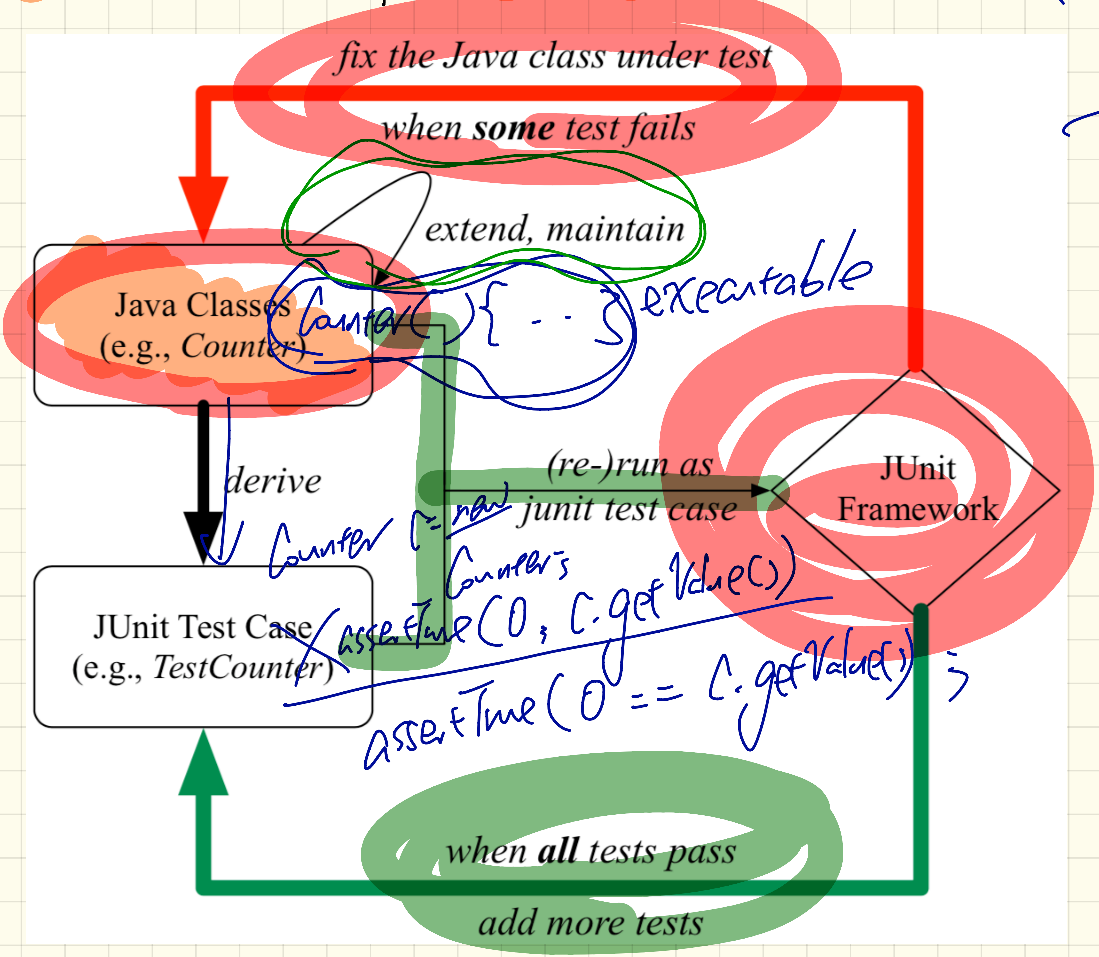
| 0 | 1 |
|---|---|
| 1 | 2 |
| 2 | 3 |

| i | c.getValue | |
|---|---|---|
| 3 | 3 | 2 |
| 2 | 2 | 1 |
| 1 | 1 | 0 |

# Test-Driven Development (TDD)

PointV1

PointV2



fix the Java class under test

when **some** test fails

extend, maintain

Java Classes (e.g., *Counter*)

derive

JUnit Test Case (e.g., *TestCounter*)

(re-)run as junit test case

JUnit Framework

when **all** tests pass

add more tests

Counter() { . . }
executable
Counter c = new Counter();
~~assertTrue(0, c.getValue())~~
assertTrue(0 == c.getValue());

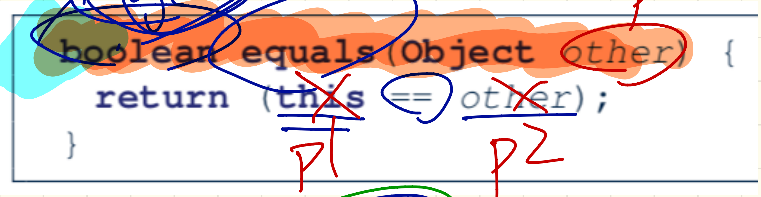
```
int i = 1;
int j = 3;
assertTrue ( i == j ); X

assertEquals( i , j );
Person p1, p2;
assertEquals (p1, p2);
```

① p1 == p2;

② p1.equals(p2)

# equals method in Object class

## Case 1: equals not overridden



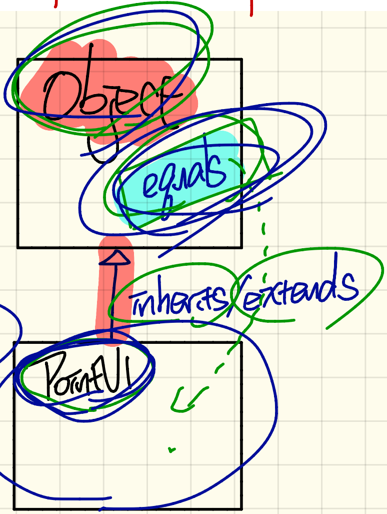
```
boolean equals(Object other) {
  return (this == other);
}
```

PointV1 p1 = ...

PointV1 p2 = ...

p1.equals(p2)

~~p1 == p2~~

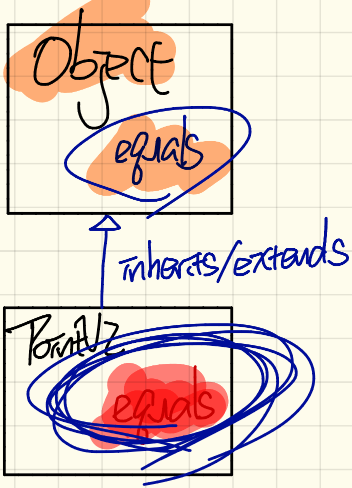


## Case 2: equals overridden

PointV2 p3 = ...

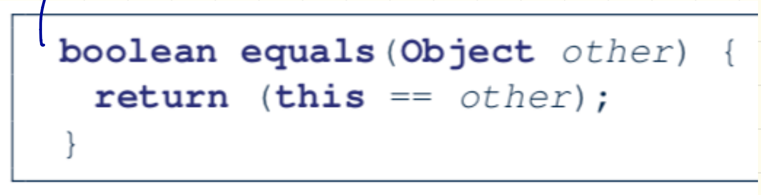
PointV2 p4 = ...

p3.equals(p4)

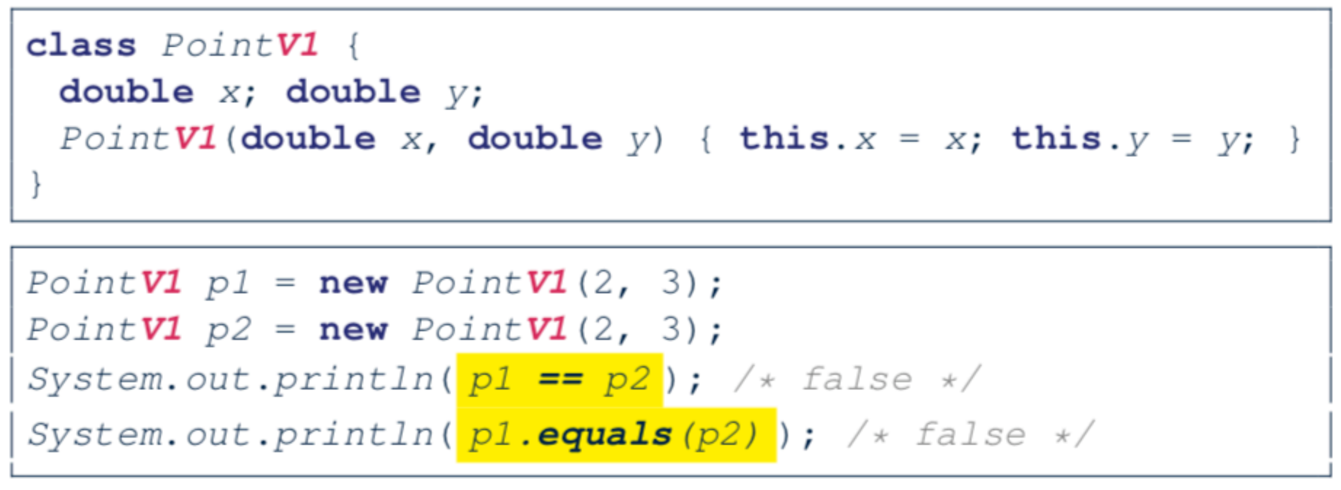


# equals method case I : calling default version

from Object class



```
boolean equals(Object other) {
  return (this == other);
}
```

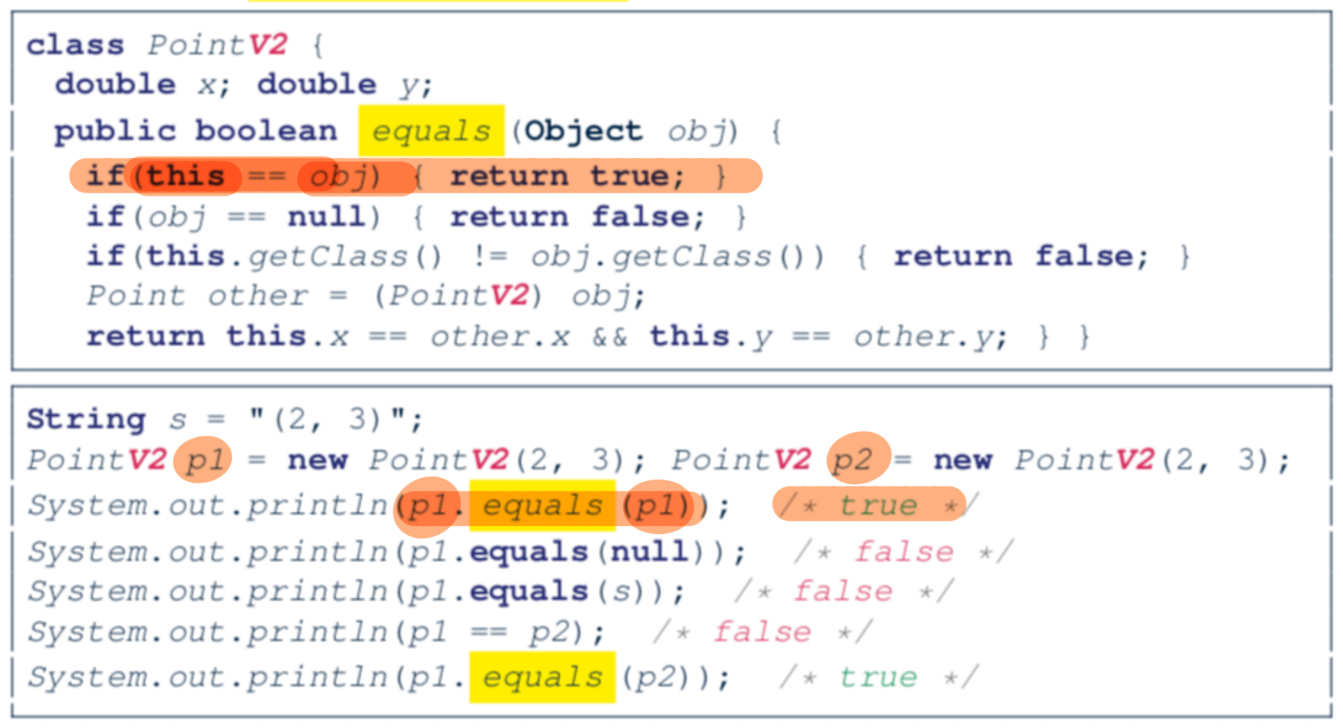


```
class PointV1 {
  double x; double y;
  PointV1(double x, double y) { this.x = x; this.y = y; }
}
```

```
PointV1 p1 = new PointV1(2, 3);
PointV1 p2 = new PointV1(2, 3);
System.out.println( p1 == p2 ); /* false */
System.out.println( p1.equals(p2) ); /* false */
```

# equals method case 2: overridding default version

Step 1: x.equals(x) == True



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

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

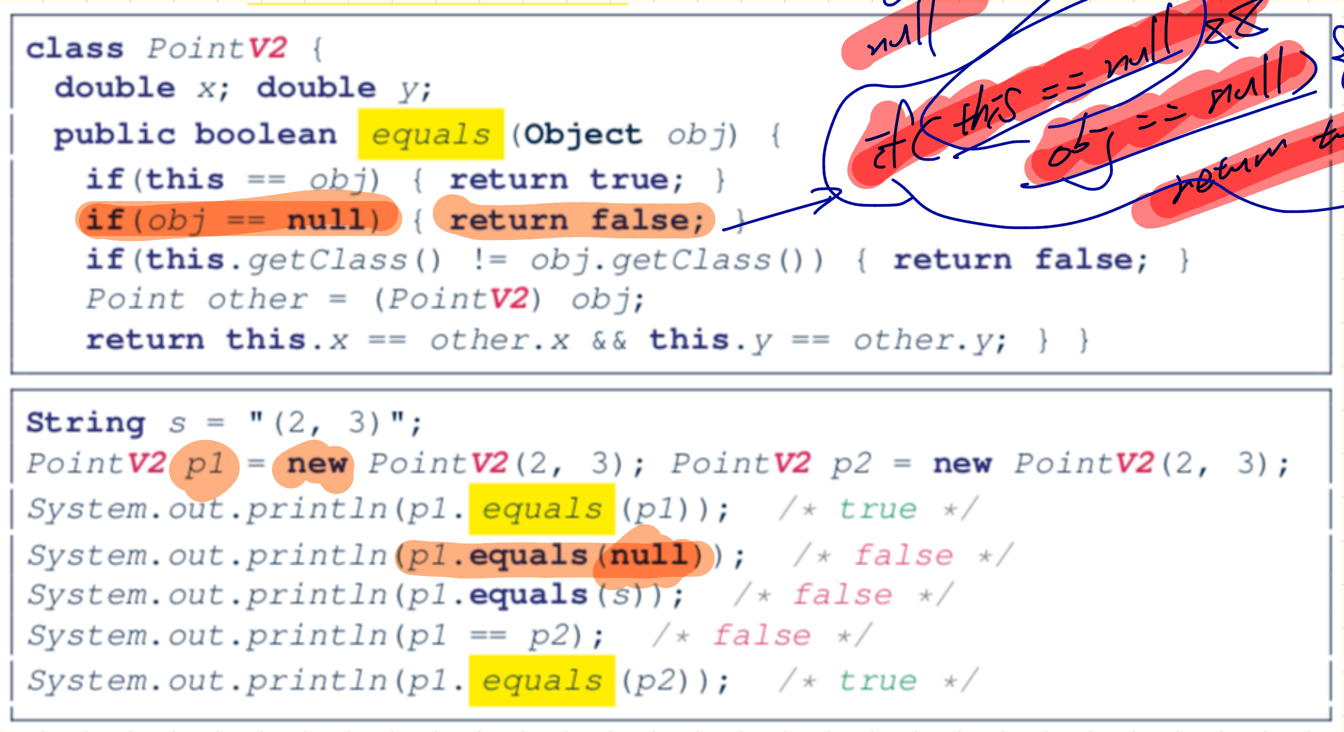
# equals method case 2: overriding default version

Step 2: x. equals(null) == False

x. equals(null);

null

if (this == null && obj == null) {
return true;



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

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

# equals method case 2: overridding default version

PointV2

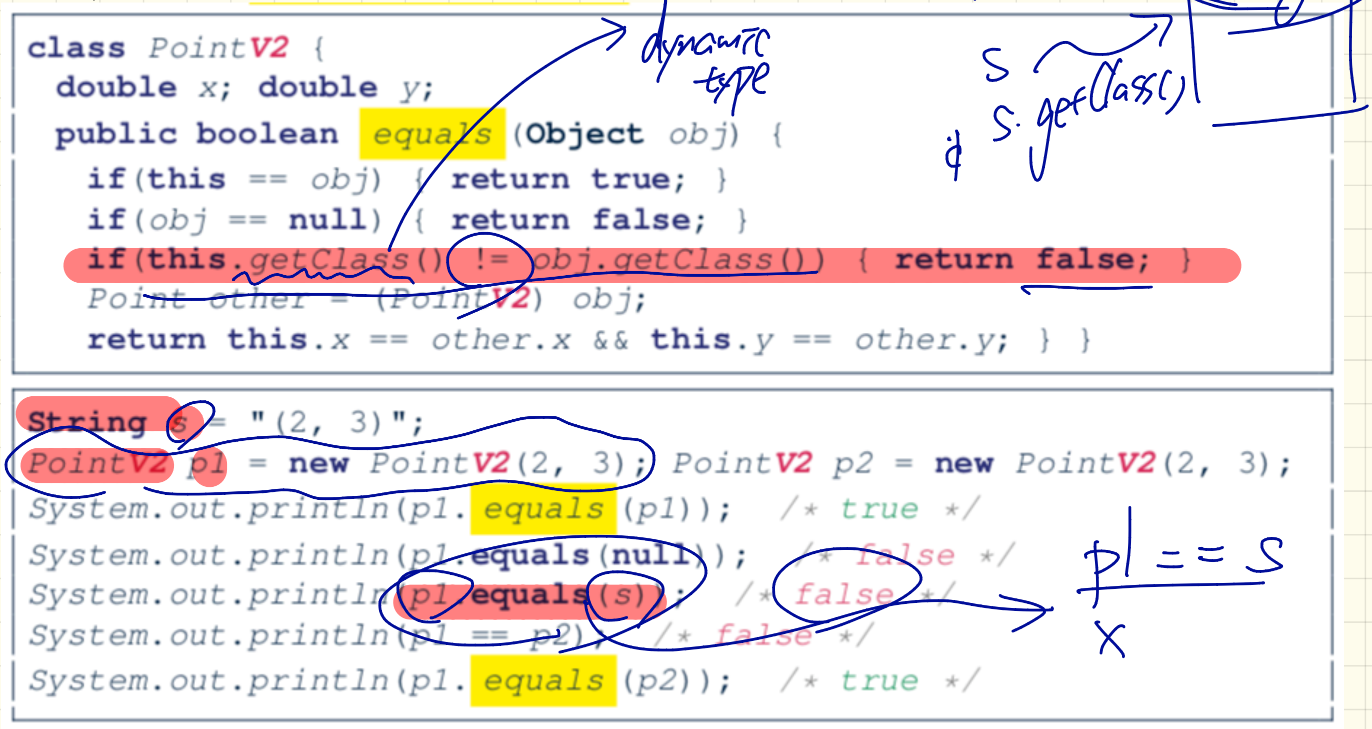
Step 3: apple. equals( banana ) == False

p1
p1.getClass()

String

dynamic type

s
≠ s.getClass()



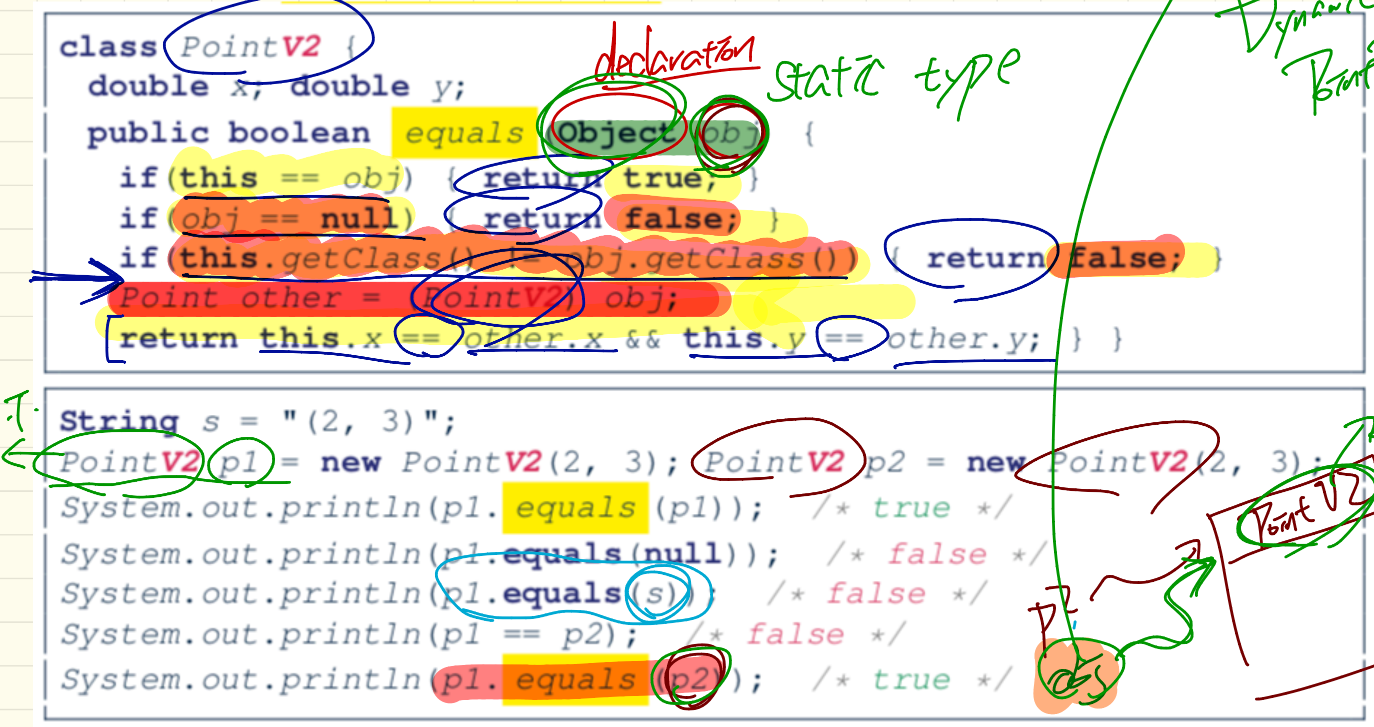
```
class PointV2 {
  double x; double y;
  public boolean equals (Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false; }
    Point other = (PointV2) obj;
    return this.x == other.x && this.y == other.y; }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

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

p1 == s
X

# equals method case 2: overriding default version

**Step 4: apple.equals(apple) depends on your def.**



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

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

declaration

static type

Static Type Object

Dynamic Type PointV2

S.T.

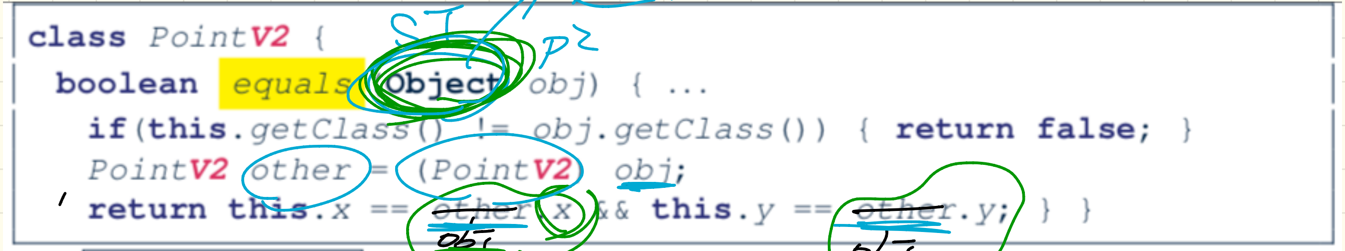
D.T. Dynamic

p2

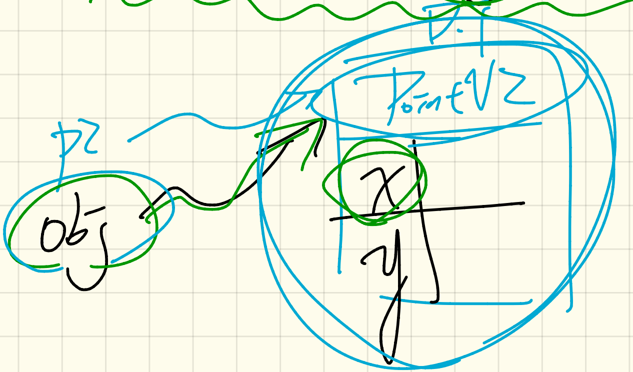
obj

PointV2

# Type Casting in Step 4 of Case 2



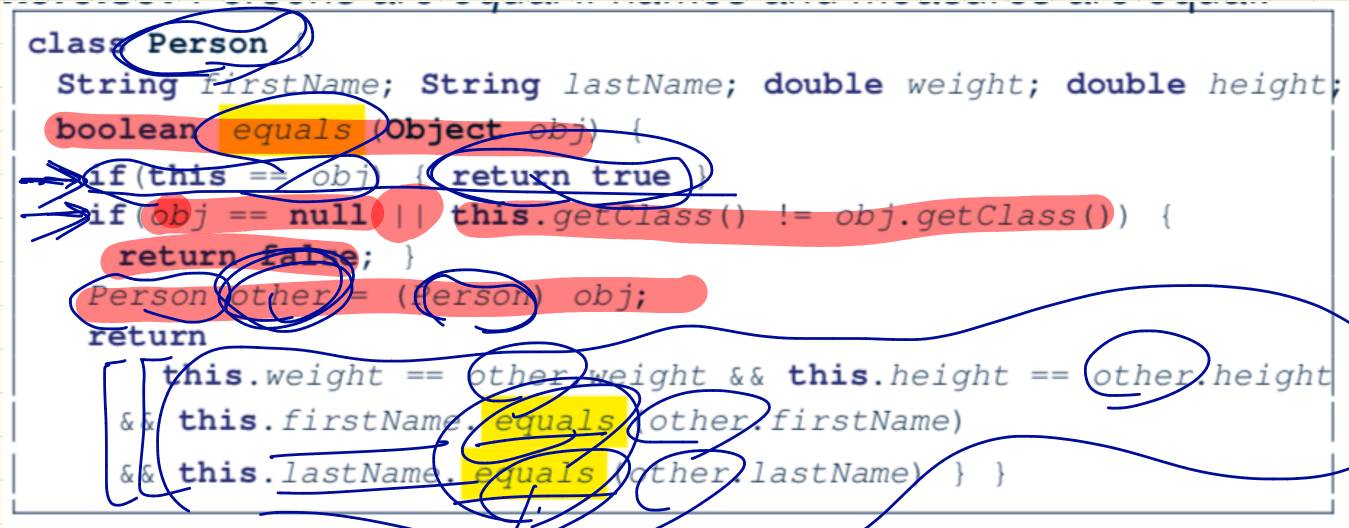
```
class PointV2 {
  boolean equals(Object obj) { ...
    if(this.getClass() != obj.getClass()) { return false; }
    PointV2 other = (PointV2) obj;
    return this.x == other.x && this.y == other.y; } }
```



will not compile

obj has ST Object which means it cannot call att/met defined in its DT (PointV2)

# Equality on Person

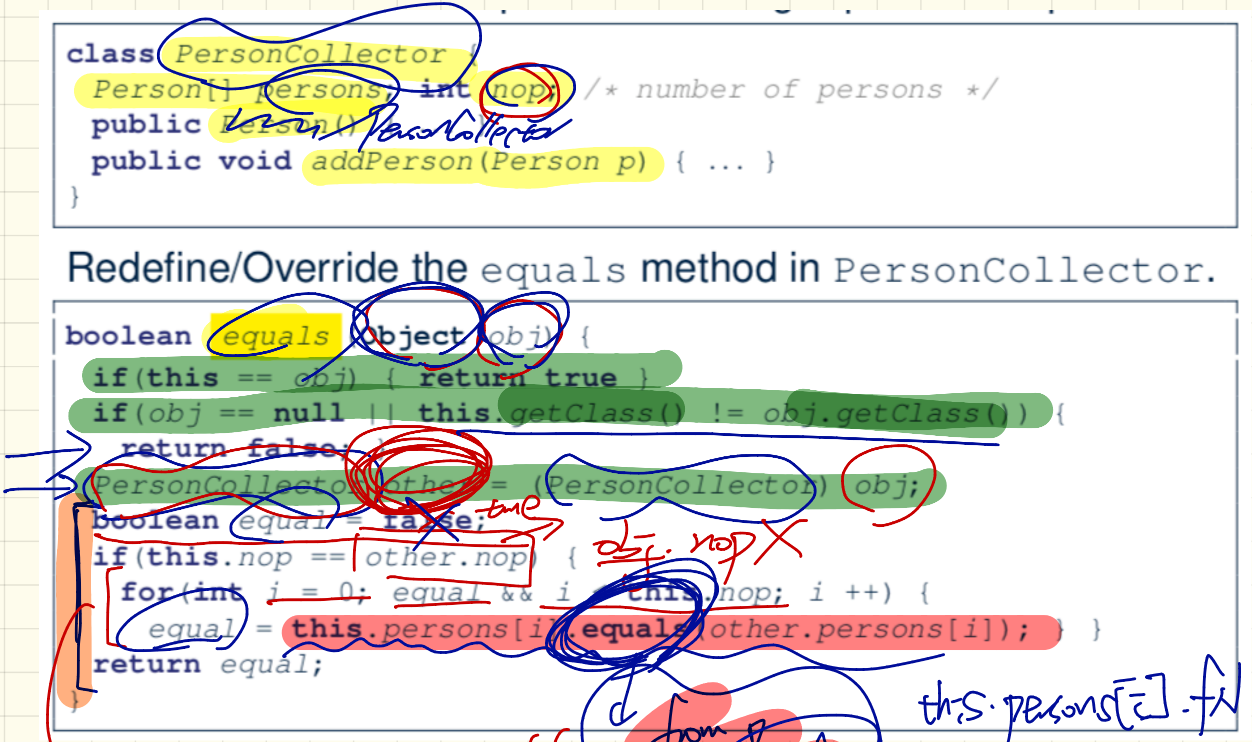


```
class Person {
  String firstName; String lastName; double weight; double height;
  boolean equals(Object obj) {
  if(this == obj) { return true }
  if(obj == null || this.getClass() != obj.getClass()) {
    return false; }
   Person other = (Person) obj;
   return
     this.weight == other.weight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

from String

# Equality on PersonCollector

pc1.equals(pc2)



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

PersonCollector

Redefine/Override the equals method in 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) {
    for(int i = 0; equal && i < this.nop; i ++) {
      equal = this.persons[i].equals(other.persons[i]); } }
  return equal;
}
```

true

obj.nop X

from Person

String

this.persons[i].fN.equals(other.persons[i].fN)

&&

do not mention obj !!

Wednesday Sep. 26

Lecture 7

- Today:

  ① More equals method examples
  ( will be covered in Lab Test I )

  ② Comparable and compareTo
  ( will not be covered in Lab Test I )

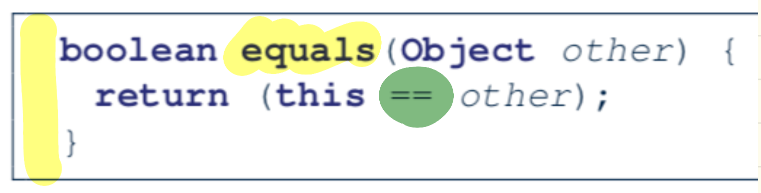
- int hashCode()

  ~ integer accessor (≈ getBMI()) based on attribute values and a formula

  ~ complete story next Monday!

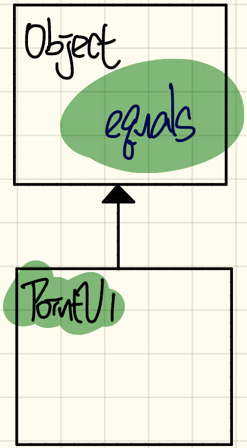
# equals method in Object class

## Case 1: equals not overridden



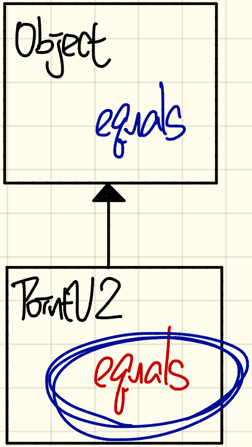
```
boolean equals(Object other) {
  return (this == other);
}
```

```
PointV1 p1 = ...
PointV1 p2 = ...
p1.equals(p2);
```

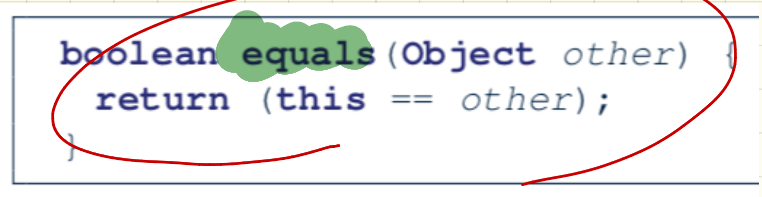


## Case 2: equals overridden

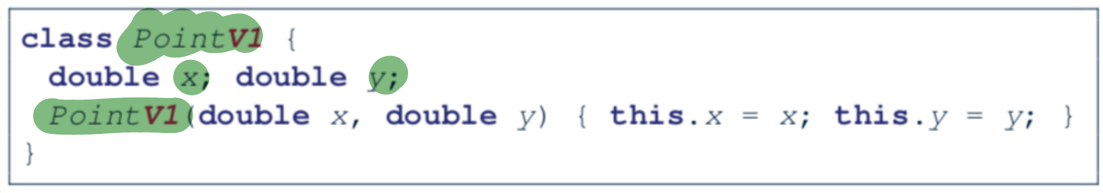
```
PointV2 p3 = ...
PointV2 p4 = ...
p3.equals(p4);
```



(Case 1)

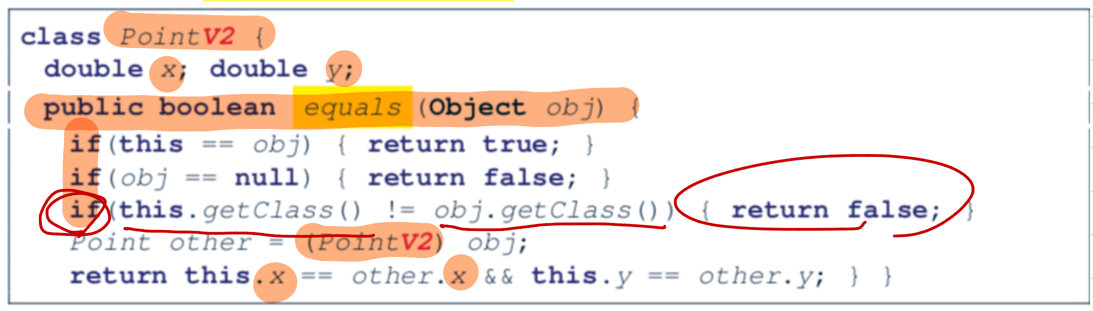


```
boolean equals(Object other) {
  return (this == other);
}
```



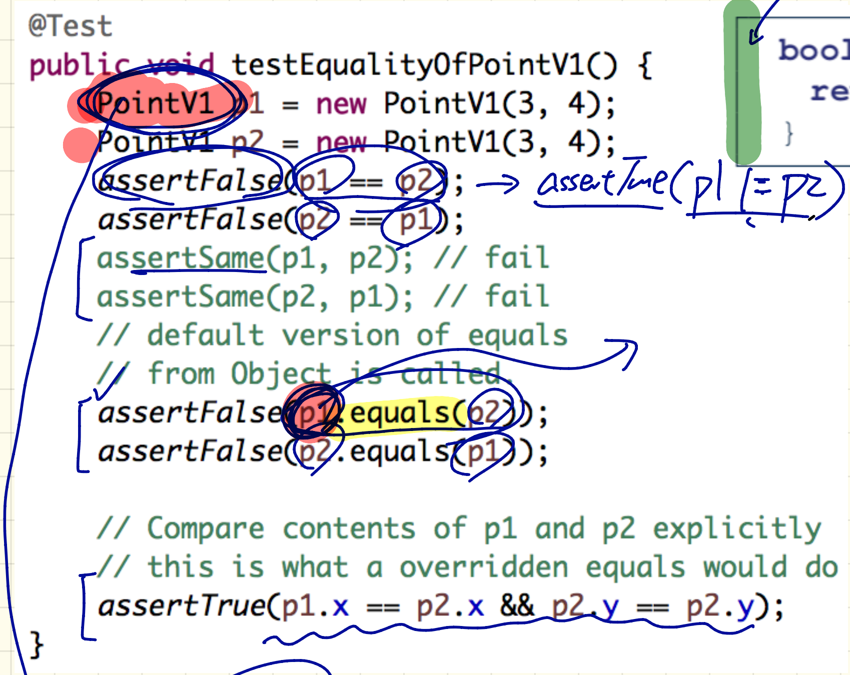
```
class PointV1 {
  double x; double y;
  PointV1(double x, double y) { this.x = x; this.y = y; }
}
```

(Case 2)



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

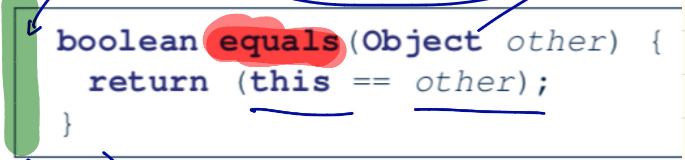
# assertSame vs. assertEquals (1)



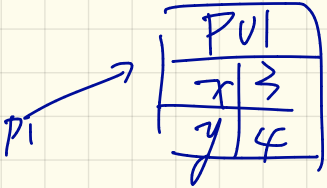
```
@Test
public void testEqualityOfPointV1() {
    PointV1 p1 = new PointV1(3, 4);
    PointV1 p2 = new PointV1(3, 4);
    assertFalse(p1 == p2);
    assertFalse(p2 == p1);
    assertSame(p1, p2); // fail
    assertSame(p2, p1); // fail
    // default version of equals
    // from Object is called.
    assertFalse(p1.equals(p2));
    assertFalse(p2.equals(p1));

    // Compare contents of p1 and p2 explicitly
    // this is what a overridden equals would do
    assertTrue(p1.x == p2.x && p2.y == p2.y);
}
```

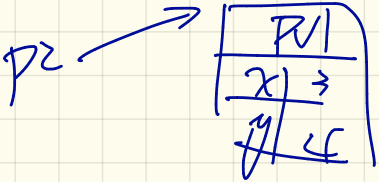
→ assertTrue(p1 != p2)

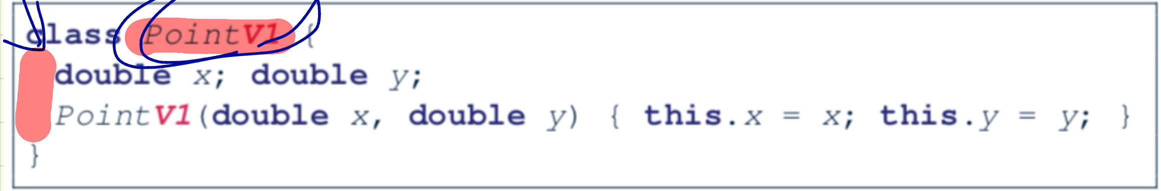
equals from Object class



```
boolean equals(Object other) {
  return (this == other);
}
```

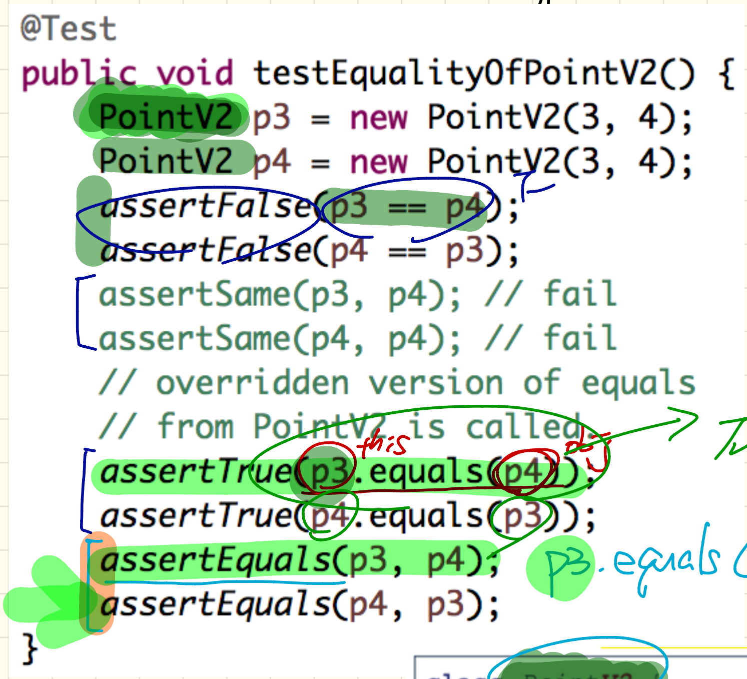




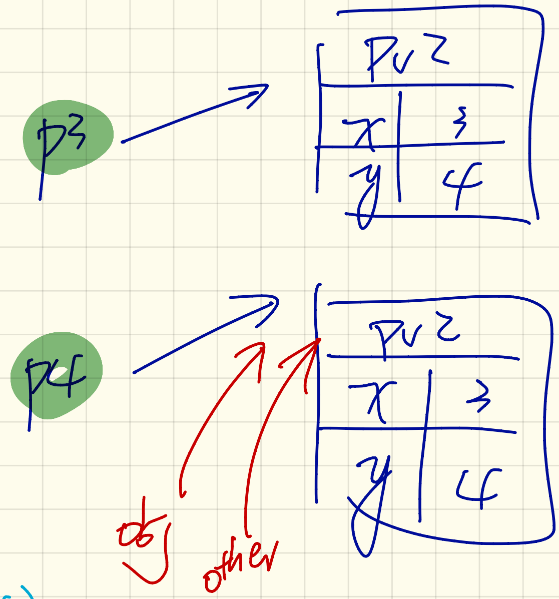


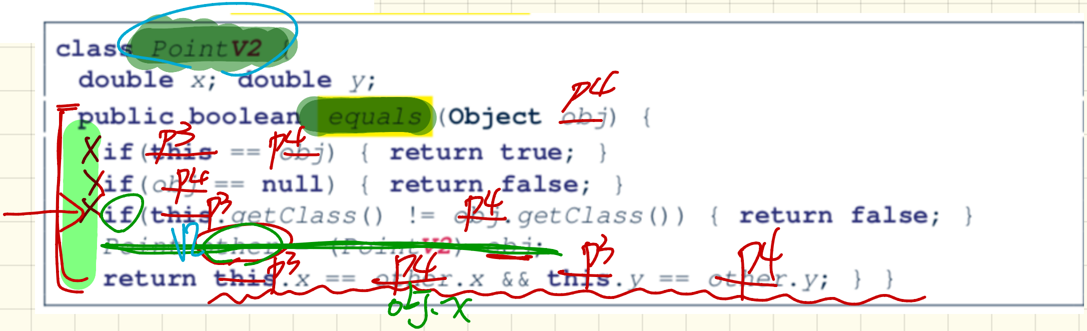
```
class PointV1 {
double x; double y;
PointV1(double x, double y) { this.x = x; this.y = y; }
}
```

# assertSame vs. assertEquals (2)



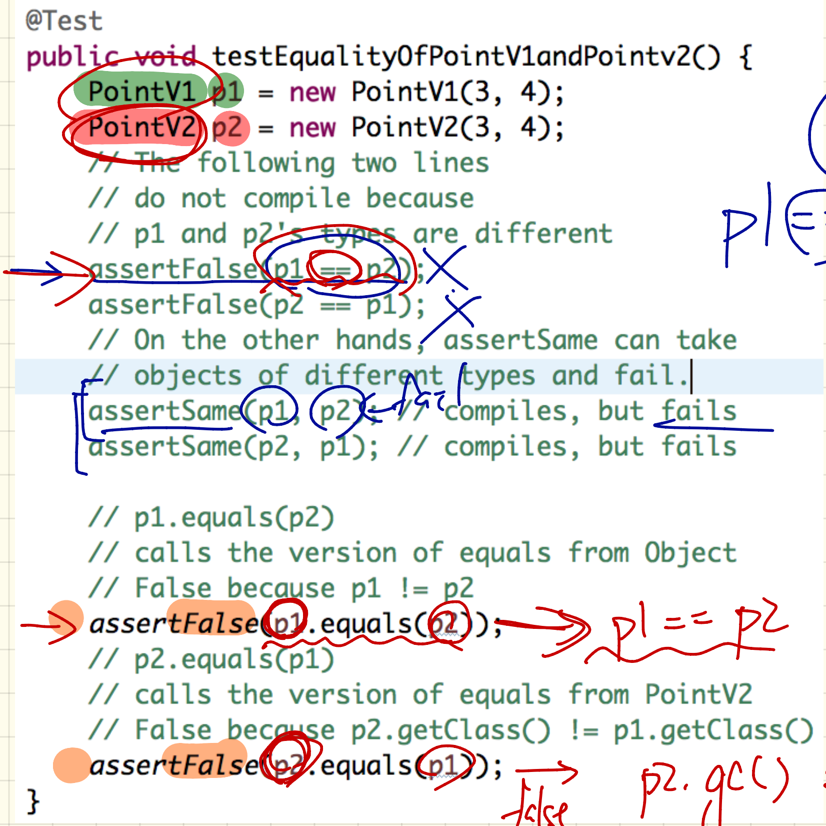
```
@Test
public void testEqualityOfPointV2() {
    PointV2 p3 = new PointV2(3, 4);
    PointV2 p4 = new PointV2(3, 4);
    assertFalse(p3 == p4);
    assertFalse(p4 == p3);
    assertSame(p3, p4); // fail
    assertSame(p4, p4); // fail
    // overridden version of equals
    // from PointV2 is called
    assertTrue(p3.equals(p4));
    assertTrue(p4.equals(p3));
    assertEquals(p3, p4);
    assertEquals(p4, p3);
}
```





```
class PointV2 {
  double x; double y;
  public 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; } }
```

# assertSame vs. assertEquals (3)



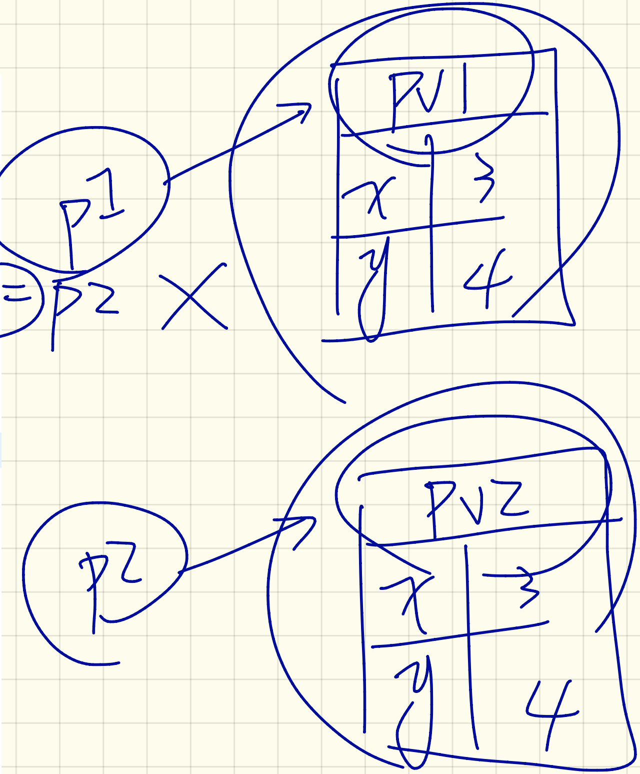
```
@Test
public void testEqualityOfPointV1andPointv2() {
    PointV1 p1 = new PointV1(3, 4);
    PointV2 p2 = new PointV2(3, 4);
    // The following two lines
    // do not compile because
    // p1 and p2's types are different
    assertFalse(p1 == p2);
    assertFalse(p2 == p1);
    // On the other hands, assertSame can take
    // objects of different types and fail.
    assertSame(p1, p2); // compiles, but fails
    assertSame(p2, p1); // compiles, but fails

    // p1.equals(p2)
    // calls the version of equals from Object
    // False because p1 != p2
    assertFalse(p1.equals(p2));
    // p2.equals(p1)
    // calls the version of equals from PointV2
    // False because p2.getClass() != p1.getClass()
    assertFalse(p2.equals(p1));
}
```

fail

→ p1 == p2

→ false p2.gc() != p1.gc()

p1 == p2 X



# Overriding & Reusing equals method

```java
class Person {
    String firstName;
    String lastName;
    double weight;
    double height;

    public Person(String firstName, String lastName, double weight, double height) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.weight = weight;
        this.height = height;
    }

    public 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);
    }
```

context objects

redefined version in String

```java
class PersonCollector {
    Person[] persons;
    int nop; /* number of persons */

    public PersonCollector() {
        persons = new Person[10];
    }

    public void addPerson(Person p) {
        persons[nop] = p;
        nop ++;
    }

    public 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;
    }
}
```

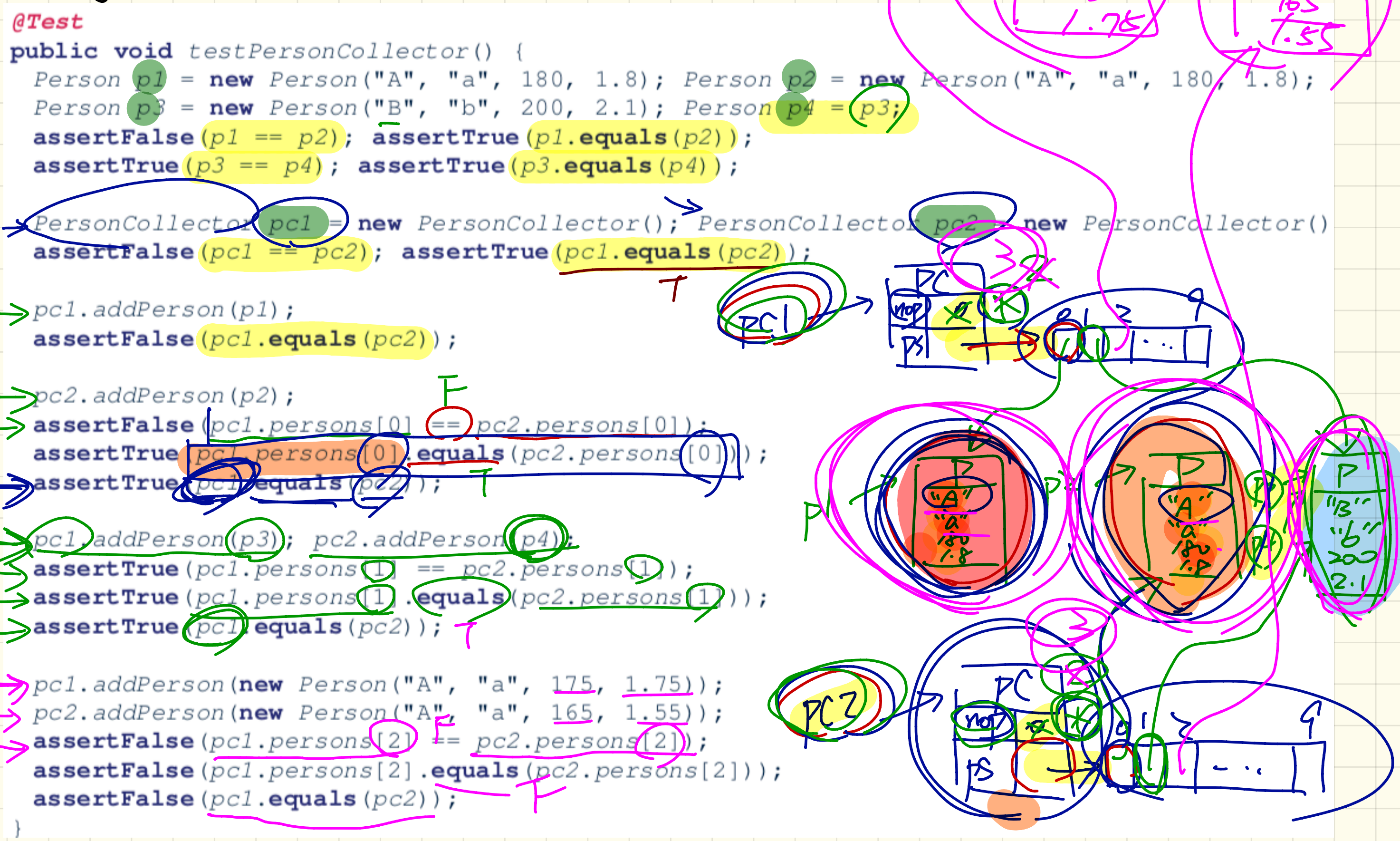
PC

PC2

0

Person

# Testing Person and PersonCollector



```java
@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));

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

  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));

  pc1.addPerson(new Person("A", "a", 175, 1.75));
  pc2.addPerson(new Person("A", "a", 165, 1.55));
  assertFalse(pc1.persons[2] == pc2.persons[2]);
  assertFalse(pc1.persons[2].equals(pc2.persons[2]));
  assertFalse(pc1.equals(pc2));
}
```

Employees:

| name | id | salary |
|---|---|---|
| alan | 2 | 4500.34 |
| mark | 3 | 3450.67 |
| tom | 1 | 3450.67 |

Sorting based on id's:

tom alan mark

emp smaller if id smaller

larger comes first

Sorting based on salaries and id's.

Smaller comes first

alan tom mark

Monday Oct. 7

Lecture 8

Employees:

| name | id | salary |
|---|---|---|
| - alan | 2 | 4500.34 |
| - mark | 3 | 3450.67 |
| - tom | 1 | 3450.67 |

List 1: tom alan mark

List 2: alan tom mark

highest salary

Alternatively "smallest" object in the list

mark alan tom

3 2 1

Sorting based on id's

emp smaller if id smaller

List 1 → tom alan mark

→ Sorting based on salaries and id's

larger comes first

smaller comes first

List 2 → alan tom mark

# Comparable Employee: Version I

generic parameter

alan.compareTo(mark);

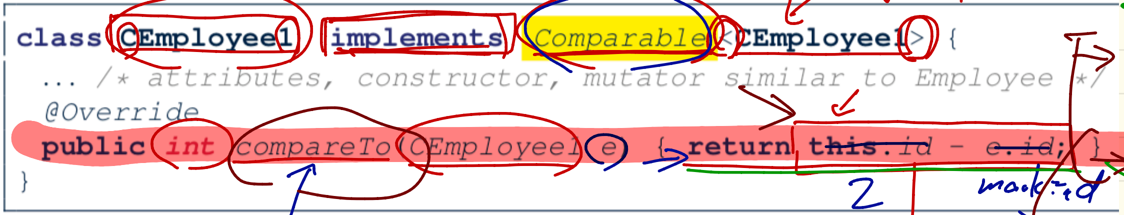
2 3

"alan > mark"

tom.compareTo(alan)

"tom > alan"

3 2 1



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

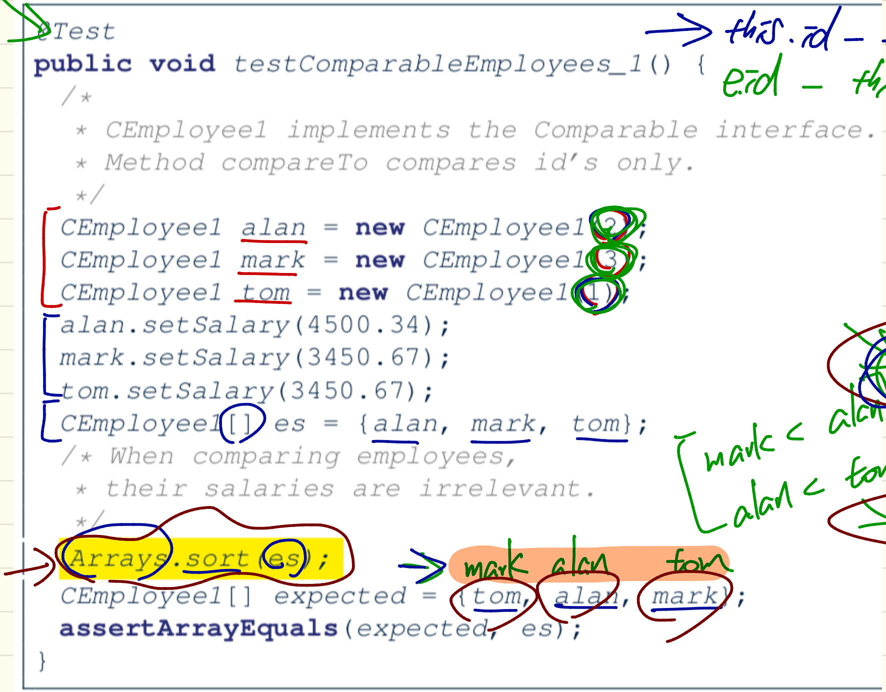
2

mark.id

return e.id - this.id;

2 1

1



```
@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);
}
```

this.id - e.id

e.id - this.id

< 0

> 0

== 0

tom > alan > mark

mark alan tom

mark < alan

alan < tom

alan.compareTo(mark); -1

"alan < mark"

tom.compareTo(alan); -1

1 2

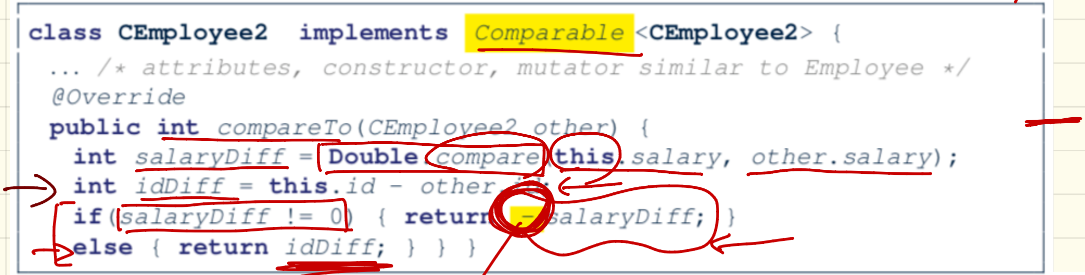
"tom < alan"

<-> tom < alan < mark

# Comparable Employee: Version 2.1

Double.compare(alan.salary, mark.salary);

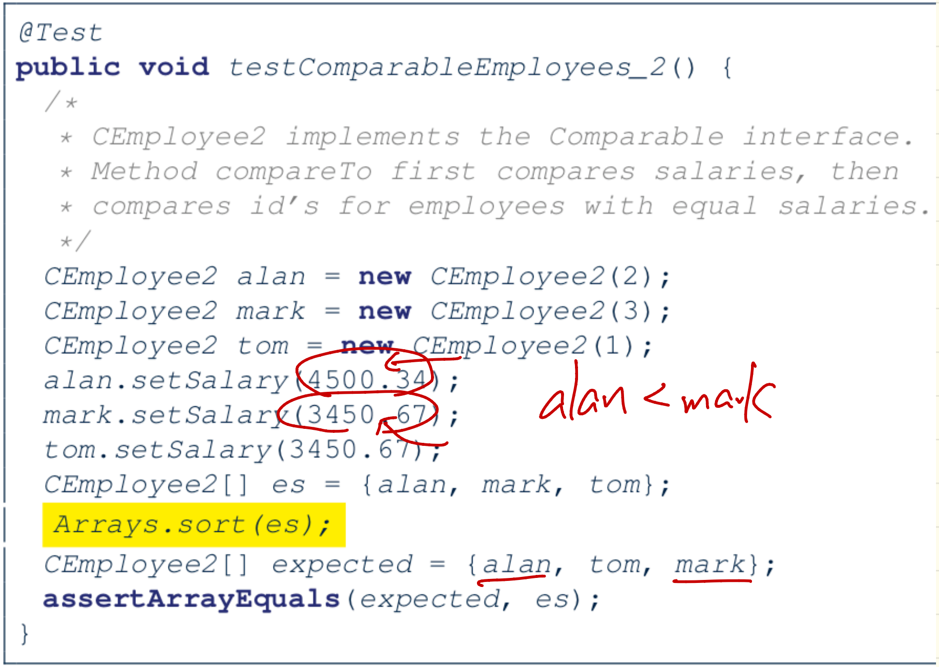
- +



```java
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; } } }
```

Double

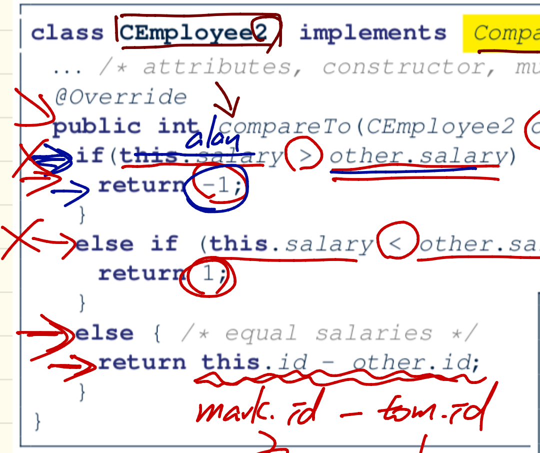
without this, alan will appear later than mark in the list.



```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);
}
```

alan < mark

# Comparable Employee: Version 2.2



```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;
    }
}
```

alan

mark.id - tom.id

alan.compareTo(mark); -1

"alan < mark"

alan.compareTo(tom); -1

"alan < tom"

mark.compareTo(tom); 2

this other

"mark > tom"

r > s

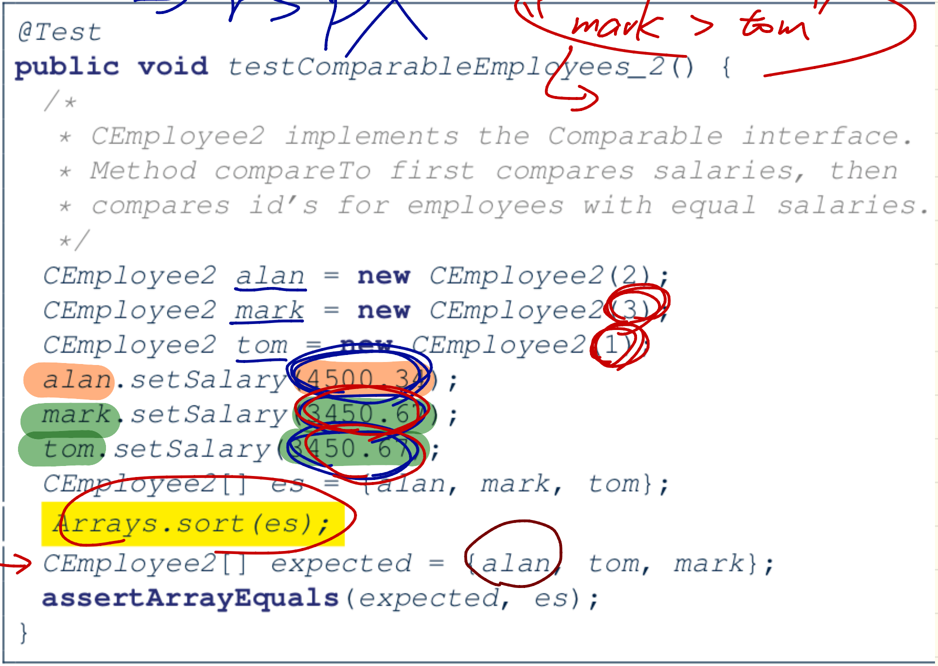
s > p

⇒ r > p

larger salary

↳ occur earlier in the sorted list

↳ considered as "smaller"



```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);
}
```

```
String[] names = {"alan", "mark", "mark"};
```

map → entries ✓

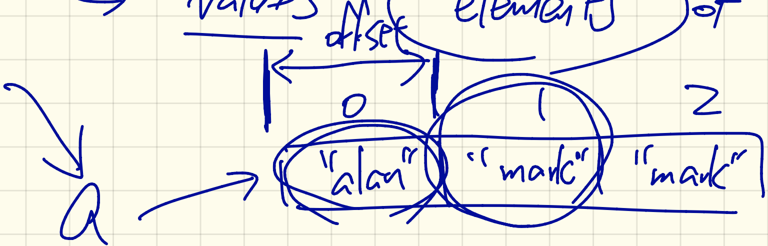
↳ keys

↳ values

indices 0, 1, 2

elements of array

2031



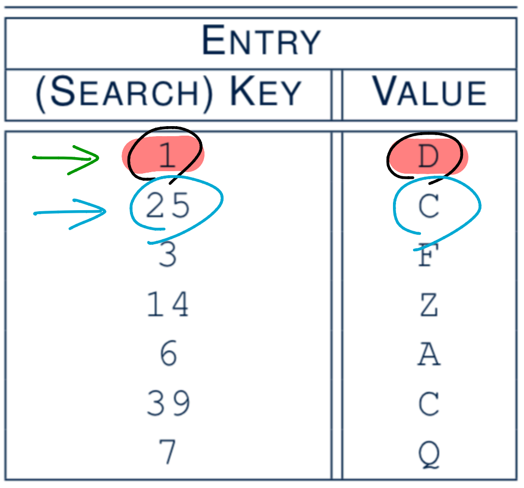
a → beginning address of array

offset

a[0] → go directly

a[1] → go to address with 1 unit of offset.

# Implementing a Map using an Array



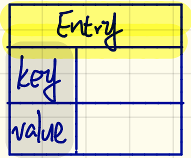
| ENTRY | |
|---|---|
| (SEARCH) KEY | VALUE |
| 1 | D |
| 25 | C |
| 3 | F |
| 14 | Z |
| 6 | A |
| 39 | C |
| 7 | Q |

Worst Case:

$y = x$

# of iterations

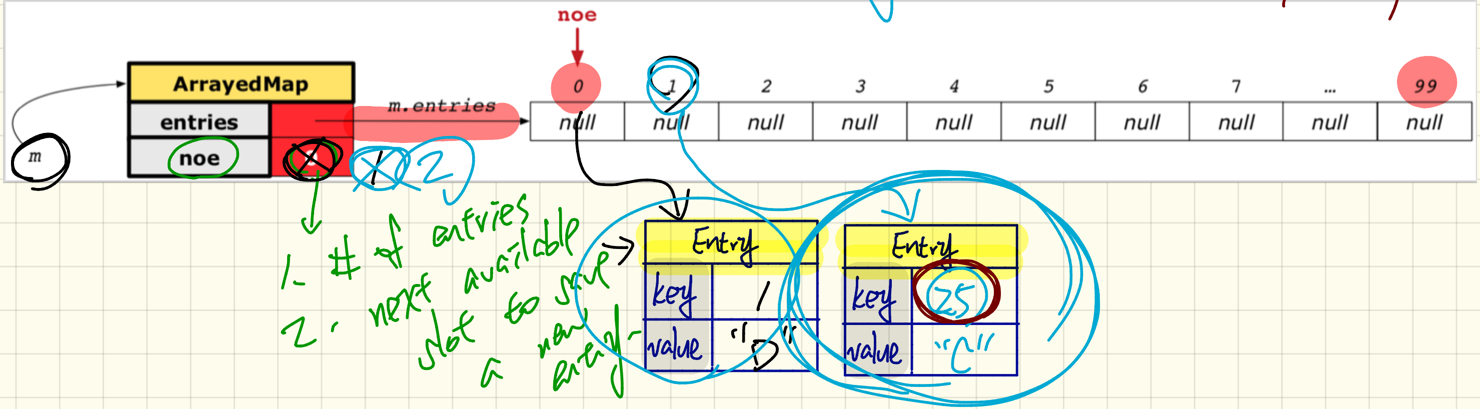
# of stored entries



m.entries[ 25 ] → a key but not the correct index to look up.

m.entries[0] =

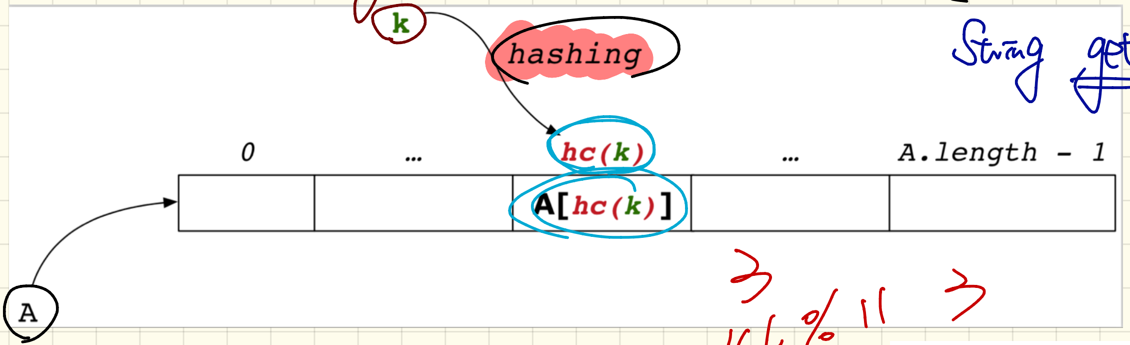
m.get(25)



1. # of entries
2. next available slot to store a new entry

Hashing
→ m.get(1)
→ m.get(3)
key 25 3

A[ 0 ] → efficient.
1
2



3
1

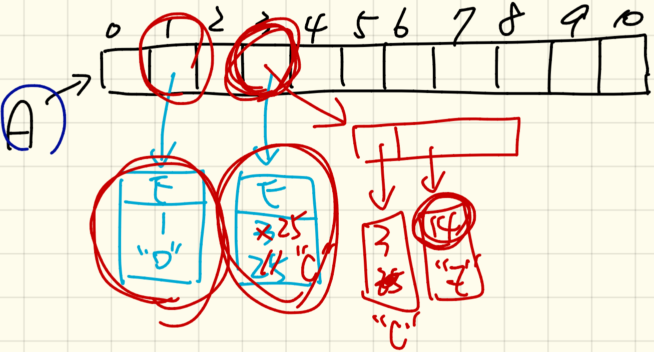
```
String get(int ~~key~~){
    return A[key % 11];
```

3

1 % 11 1
3 % 11 3

3
14 % 11 3

Say: A.length is 11 and
hc(k) = k % 11

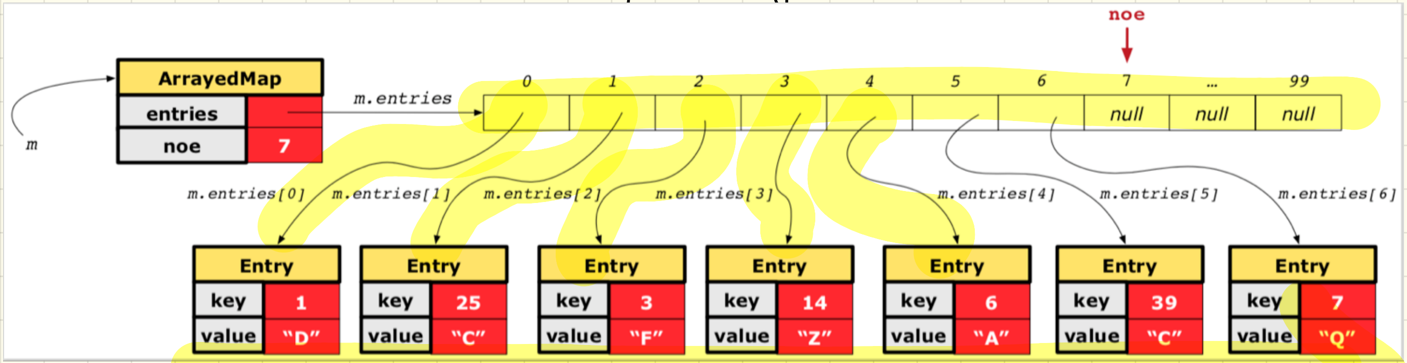


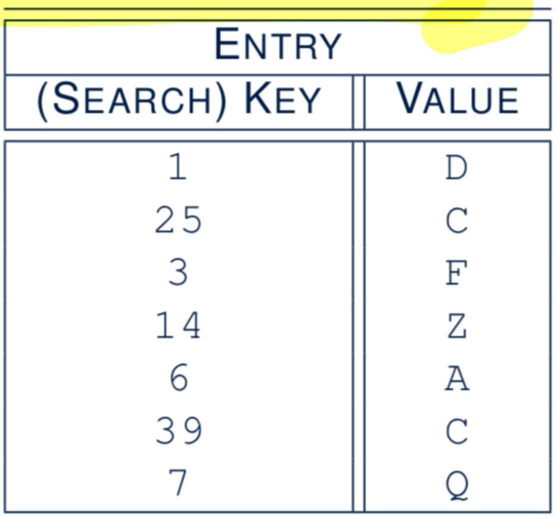
| hc(k) | Entry | |
|---|---|---|
| | (Search) Key | Value |
| 1 | 1 | D |
| 3 | 25 | C |
| | 3 | F |
| | 14 | Z |
| | 6 | A |
| | 39 | C |
| | 7 | Q |

Wednesday Oct. 3

Lecture 9

# Inefficient Implementation of Map: Array



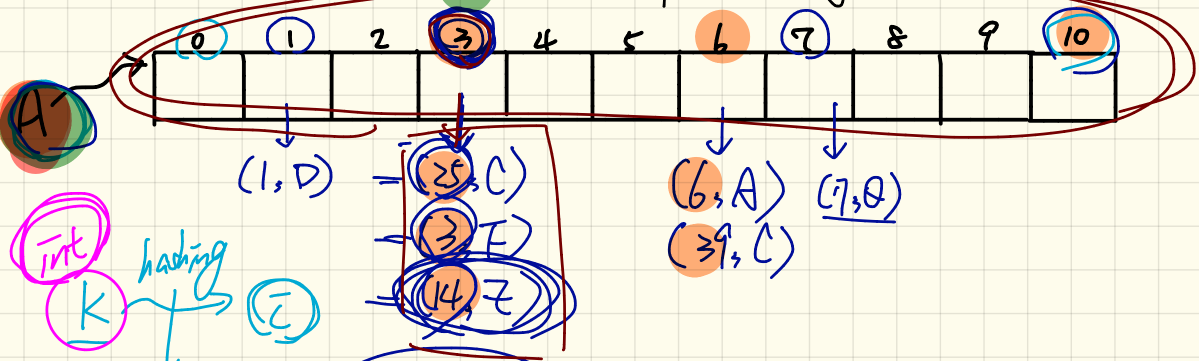


| ENTRY | |
|---|---|
| (SEARCH) KEY | VALUE |
| 1 | D |
| 25 | C |
| 3 | F |
| 14 | Z |
| 6 | A |
| 39 | C |
| 7 | Q |

## Running Time of Searching

≈ # of iterations for search

# Efficient Implementation of Map: Hashing



$hc(k) = K \% 11$

m. get(7)

m. get(14) → 14 % 11 == 3

|  | ENTRY |  |
|---|---|---|
| hc(k) | (SEARCH) KEY | VALUE |
| 1 | 1 | D |
| 3 | 25 | C |
| 3 | 3 | F |
| 3 | 14 | Z |
| 6 | 6 | A |
| 6 | 39 | C |
| 7 | 7 | Q |

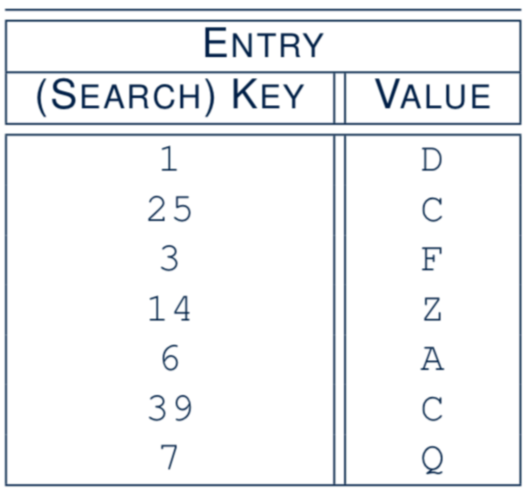
## Running Time of Searching: 25

25

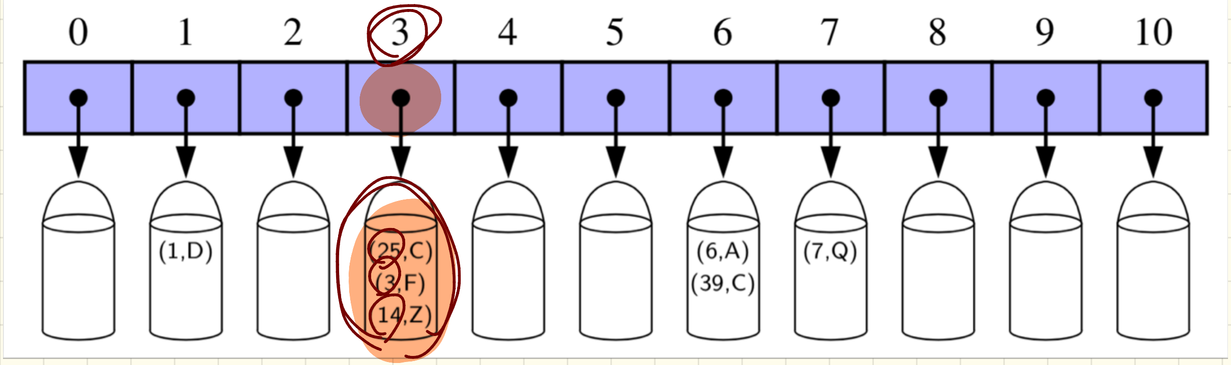
1. calculate hc(~~k~~)
2. indexing A[~~hc(k)~~ 3]

3

# Bucket Array



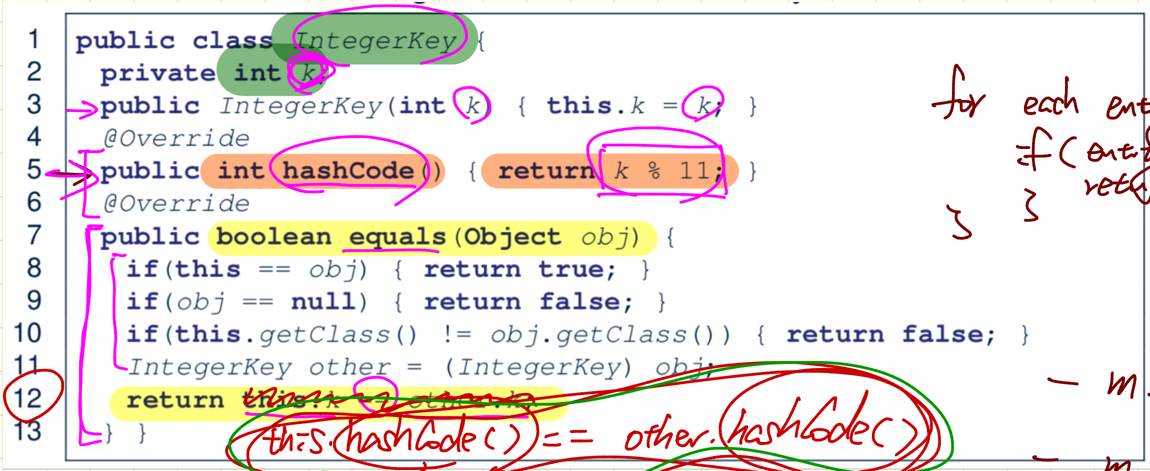
| Entry | |
|---|---|
| (Search) Key | Value |
| 1 | D |
| 25 | C |
| 3 | F |
| 14 | Z |
| 6 | A |
| 39 | C |
| 7 | Q |



How do you search through a bucket where entry keys have same hash code?

# Implementing hashCode() for IntegerKey

IK ik_25 = new IK(25);



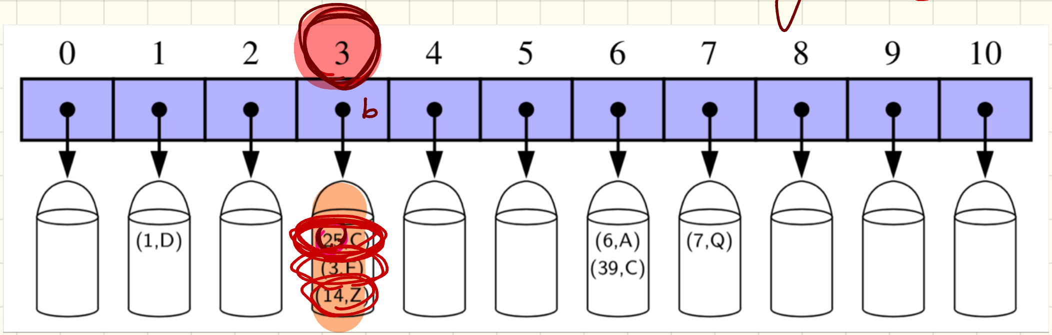
```
1   public class IntegerKey {
2     private int k;
3     public IntegerKey(int k) { this.k = k; }
4     @Override
5     public int hashCode() { return k % 11; }
6     @Override
7     public boolean equals(Object obj) {
8       if(this == obj) { return true; }
9       if(obj == null) { return false;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

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

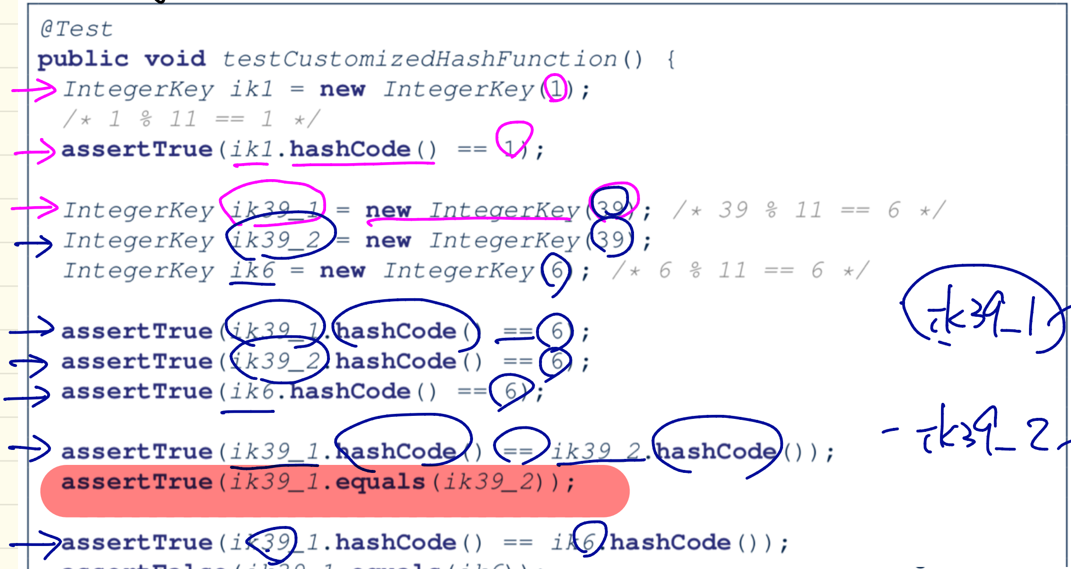
```
for each entry in bucket 'b' {
    if (entry.key.equals(25)) {
        return entry;
    }
}
```

- m.get(25)
- m.get(3)

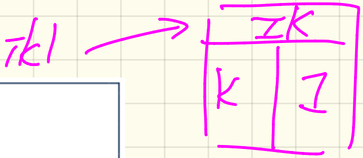
Q. Change L12 to this.hashCode() == other.hashCode() ?



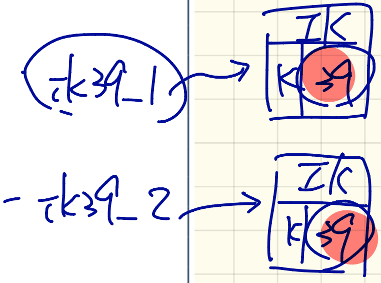
# Testing Overridden Hash Function

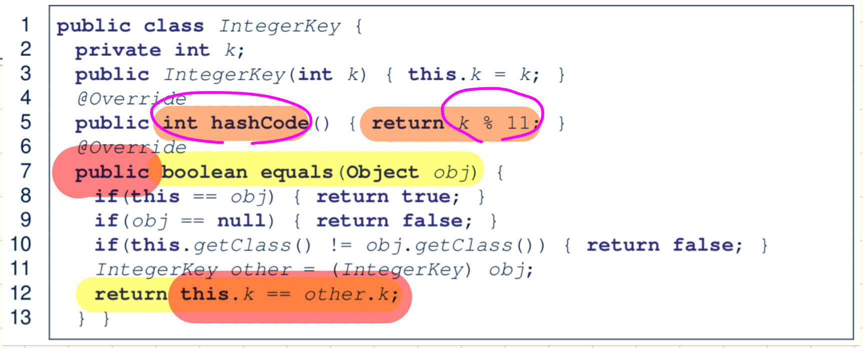


```
@Test
public void testCustomizedHashFunction() {
  IntegerKey ik1 = new IntegerKey(1);
  /* 1 % 11 == 1 */
  assertTrue(ik1.hashCode() == 1);

  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));
}
```



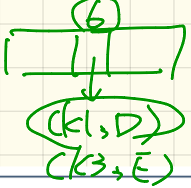
ik1.k % 11

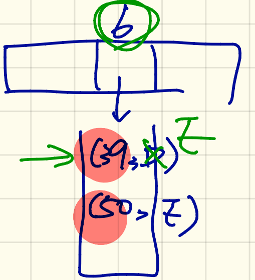


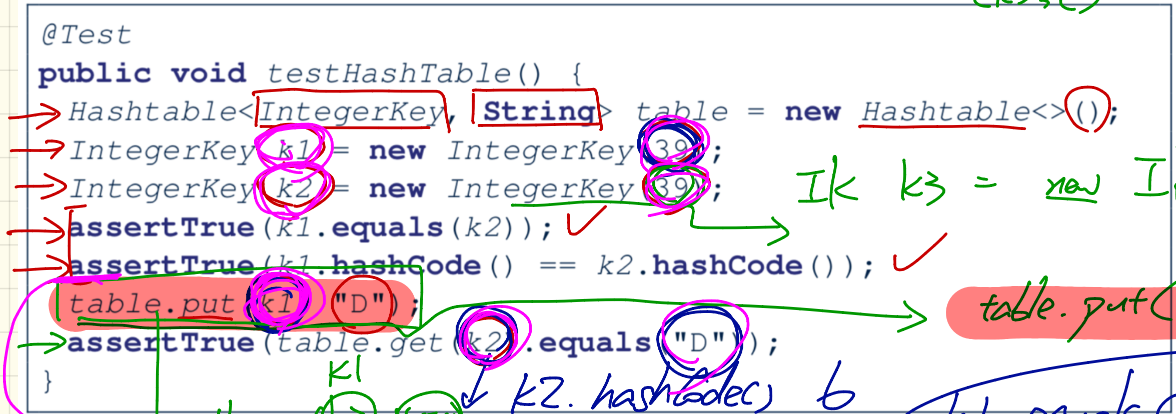


```
public class IntegerKey {
  private int k;
  public IntegerKey(int k) { this.k = k; }
  @Override
  public int hashCode() { return k % 11; }
  @Override
  public boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass())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

# Using hashCode() for HashTable







```
@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"));
}
```

Ik k3 = new Ik(50); hc 6

table.put(k3, "E");

k1

k2.hashCode() 6

k1.equals(k3) T

k1.hc() == k3.hc() T

table.put(k2, "E");

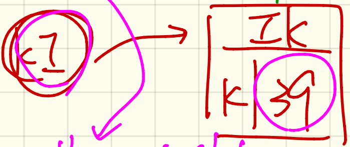
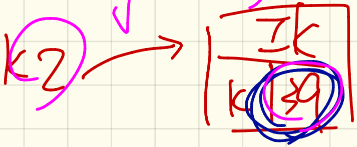
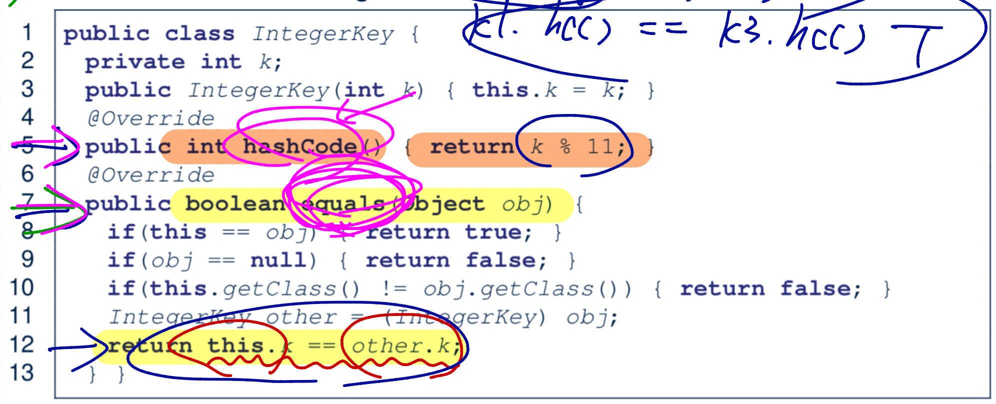


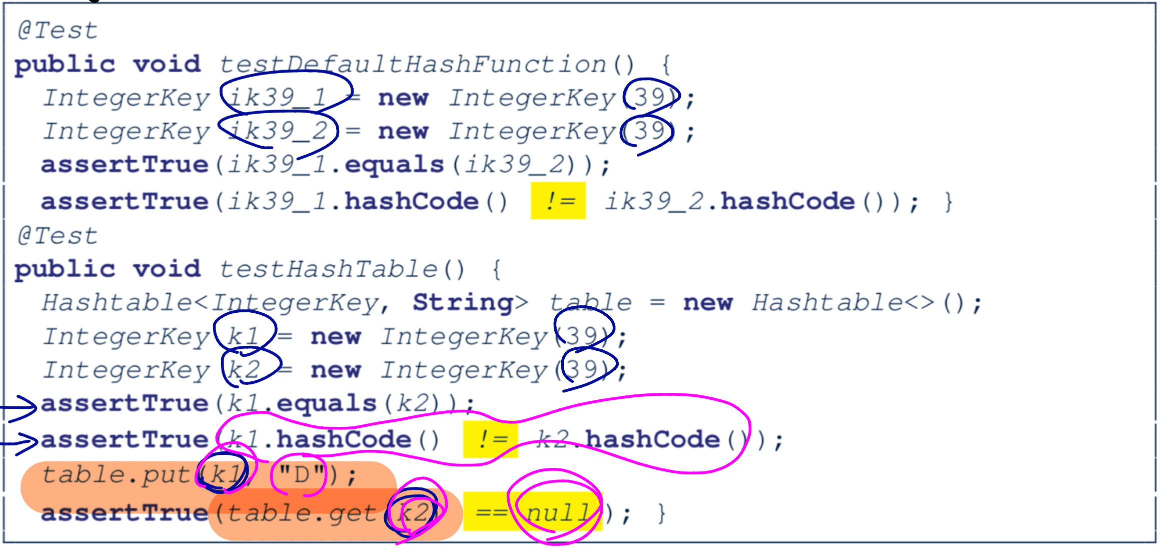
table.get(k1)





```
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;
  } }
```

# Using Default Hash Function for HashTable



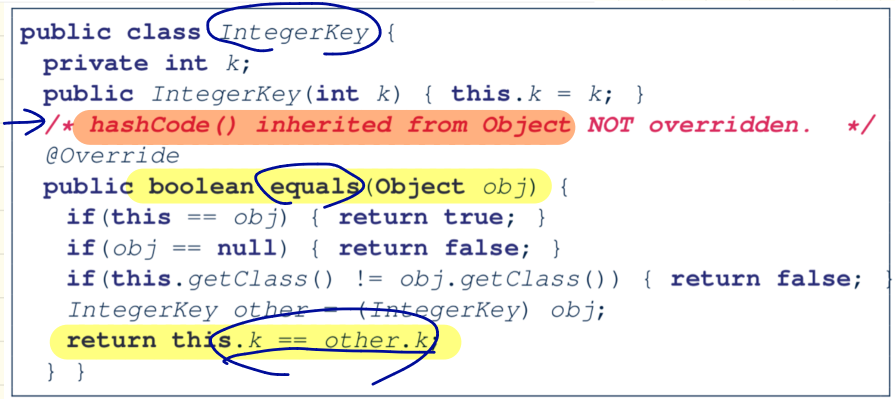
```
@Test
public void testDefaultHashFunction() {
  IntegerKey ik39_1 = new IntegerKey(39);
  IntegerKey ik39_2 = new IntegerKey(39);
  assertTrue(ik39_1.equals(ik39_2));
  assertTrue(ik39_1.hashCode() != ik39_2.hashCode()); }
@Test
public void testHashTable() {
  Hashtable<IntegerKey, String> table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

Contract for hashing:

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

Say: k1 39

k2 39



```
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;
} }
```

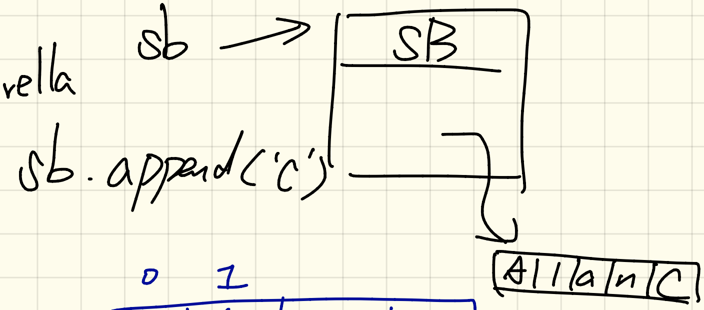
It rains   I use umberella

$$-\left(P \Rightarrow q\right)$$

Contrapositive

$$-\left(\neg q \Rightarrow \neg P\right)$$

I don't use umberella   it doesn't rain



Input:

| 0 | 1 | | |
|---|---|---|---|
| 3 | 1 | 2 | 4 |

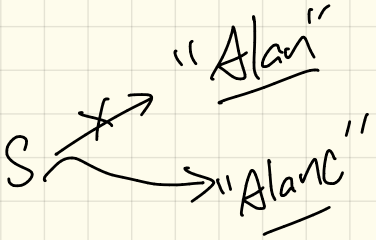
Output:

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |

$$S \mathrel{+}= C$$

$$S = \boxed{S + C}$$

'c'



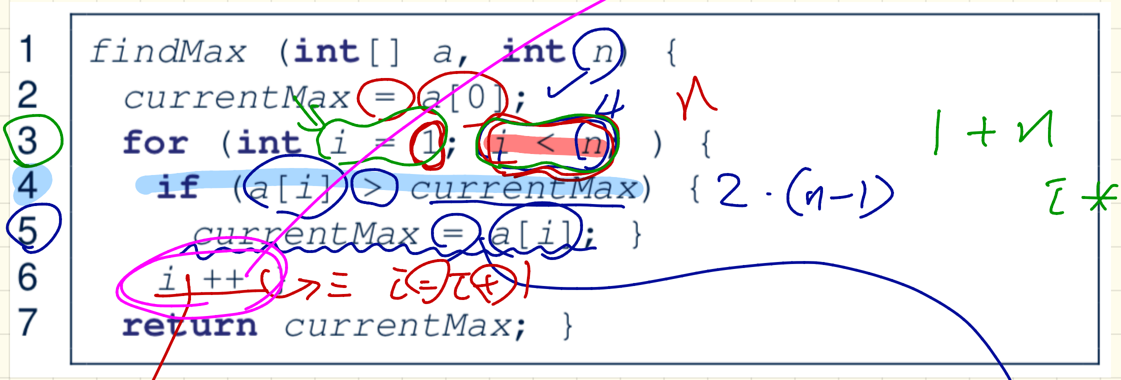
Monday Oct. 15

Lecture 10

- Lab Test I marks by Friday
- Lab 3

  Tutorial on Java Collections.

# Counting # of Primitive Operations



```
findMax (int[] a, int n) {
  currentMax = a[0];
  for (int i = 1; i < n ) {
   if (a[i] > currentMax) {
     currentMax = a[i]; }
   i ++
  return currentMax; }
```

4

n

1 + n

{ 2 · (n − 1)

≡ i = i + 1

4

i *= a[ i % 2 ]

|||

i = i * a[ i % 2 ]

i *= a[ i % a[i] ]

5

6 · (n − 1)

findMax ( { 2 , 1 , 4 , 5 } , 4 )

size 4

6

currentMax = ( a[i] * a[i] ) % a[i]

i *= a[i]

|||

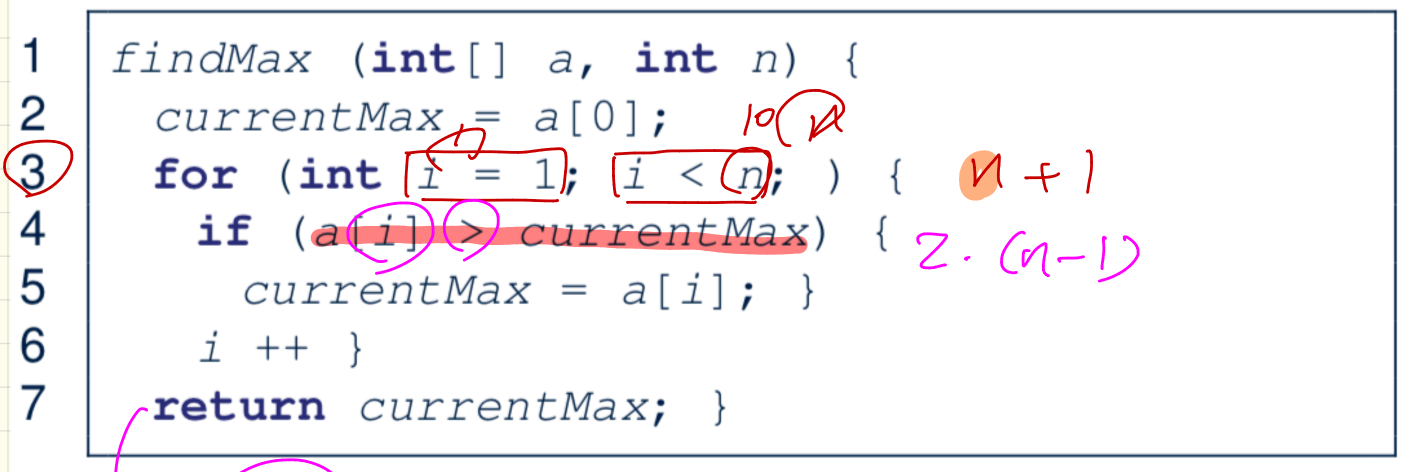
i = i * a[i]

| i | i < 4 |
|---|---|
| ~~0~~ | |
| 1 | T |
| 2 | T |
| 3 | T |
| 4 | F |

4 < 4

5

4



```
findMax (int[] a, int n) {
  currentMax = a[0];
  for (int i = 1; i < n; ) {
    if (a[i] > currentMax) {
      currentMax = a[i]; }
    i ++ }
  return currentMax; }
```

10 $\cancel{n}$

$n + 1$

$2 \cdot (n-1)$

return a[i]

$2 \cdot$

| i | i<10 |
|---|---|
| 1 | T |
| ⋮ | ⋮ |
| 9 | T |
| 10 | F |

10

$(7\overset{10}{\cancel{n}} - 2) * 2ms$

$= (68 * 2) ms$

Method 1

$7n - 2$

Method 2

$8n + 9$

Input size : 100

time for 70 : 2ms

ms

absolute RT

$\cancel{7}\, n - 2$

$n^2 - 2$

vs.

$\cancel{8}\, n + 9$

$\cancel{\log n}$

$8 \cdot n^0$

Asymptotically ; same

$\cancel{\pi}\underline{n} \; \cancel{-2}$

$\textcircled{n}$

# Big-O

→ RT of your algo.

$f(n) \in O(g(n))$ if there are:

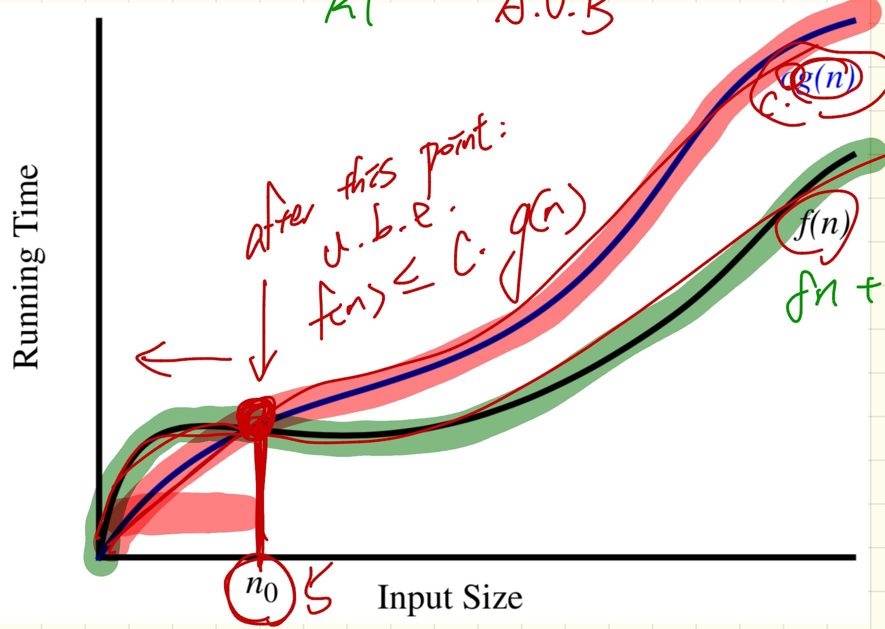
- A real constant $c > 0$
- An integer constant $n_0 \geq 1$

such that:

$$f(n) \leq c \cdot g(n) \quad \text{for } n \geq n_0$$

→ upper bound effect

RT A.U.B



after this point: u.b.e. $f(n) \leq c \cdot g(n)$

$9 \cdot n$

$8n + 5$

5

Example: $f(n) = 8n + 5$

$g(n) = n$

Prove: $f(n) \in O(g(n))$

Choose: $c = 9$

$n_0$ ?

$O(n)$

$\to RT_1(n) = 7n - 2$

$RT_2(n) = 6n^2 - 100$

$O(n^2)$

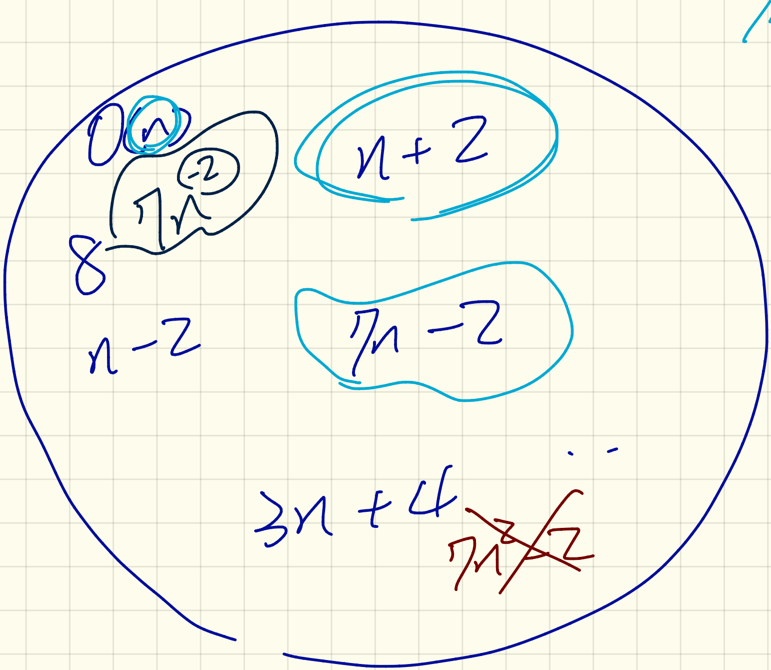
$O(n)$ a set of functions

$7n-2$ is $O(n)$

←

$a \cdot x + b$

$b$



$7n^{2}-2$ is $O(n^{1})$

1M

$$f(n) = \underline{a_0}\,\underline{n^0} + \underline{a_1}\,\underline{n^1} + \cdots + \underline{a_d}\,\underline{n^d}$$

Prove : $f(n)$ is $O(n^d)$

② Is $f(n) \le c \cdot n^d$ ?

Choose

→ $c = |a_0| + |a_1| + \cdots + |a_d|$

$n_0 = 1$

① Is $f(1) \le c \cdot 1^d$ ?

$$a_0 \underline{1^0} + a_1 1^1 + \cdots + a_d 1^d \le |a_0| \cdot \underline{1^d} + |a_1| \cdot 1^d + \cdots + |a_d| 1^d$$

$\le$

$f(n) = 3 \log n + 2$ is $O(\log n)$

$c = 5$

| $n_0$ | $3 \log n + 2$ | $5 \cdot \log n$ | |
|---|---|---|---|
| 1 | 2 ✗ | 0 | |
| 2 | 5 | 5 | ✓ |

Wednesday Oct. 17

Lecture 11

- Lab Test 2 : October 29

  Study Guide availabe next Monday

$O(100n)$ vs. $O(2n)$

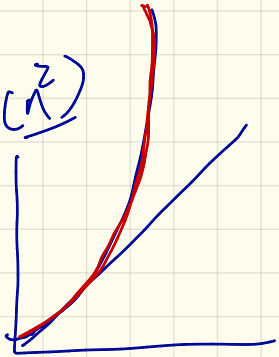
$O(n^0) \subset O(n^1) \subset O(n^2) \subset \cdots$

$O(2^n)$

$2n$ is $O(n)$ is $O(n^2)$ is $O(n^3)$

$2n + 100 \cdot$ is $\log n$

$O(n^2)$ is $2n$

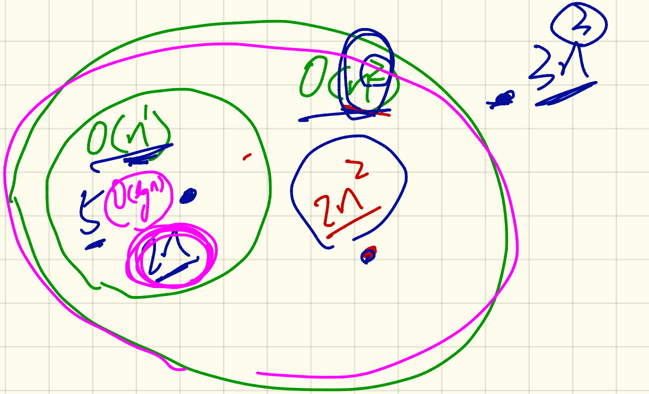


$O(n)$ $O(n^2)$ is ~~$O(n)$~~

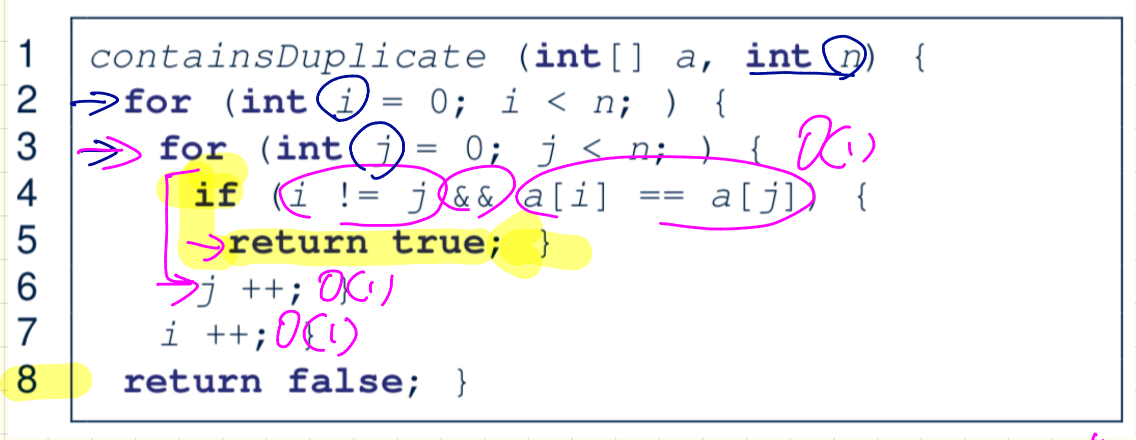
$c = 2 + 100 = 102$

| $n_0$ | $2n + 100 \cdot \log n$ | $102 \cdot n$ |
|---|---|---|
| 1 | $2 + 0 = 2$ | $\leq 102$ |

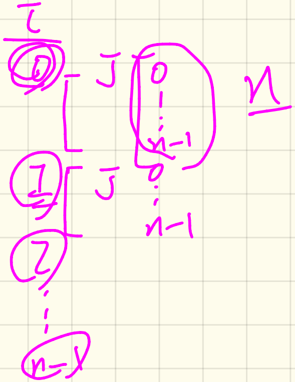
$O(n^1)$ vs. $O(n^2)$



# Determining Asymptotic Upper Bound (1)

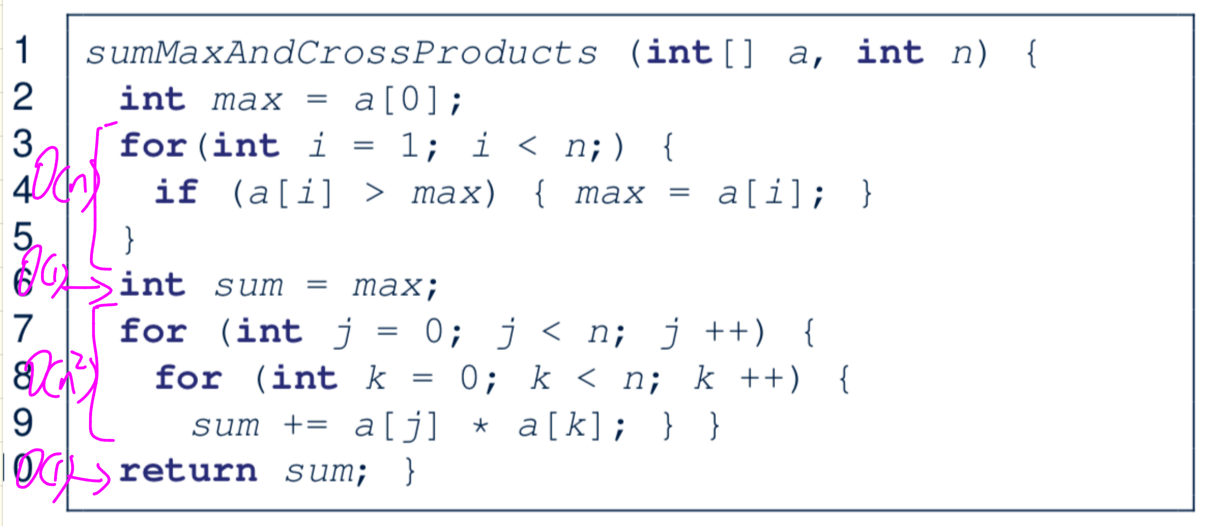


```
containsDuplicate (int[] a, int n) {
  for (int i = 0; i < n; ) {
    for (int j = 0; j < n; ) {   O(1)
      if (i != j && a[i] == a[j]) {
        return true; }
      j ++; O(1) }
    i ++; O(1) }
  return false; }
```



$$O(\underbrace{1}_{\text{body of loop}} \times \underbrace{n}_{\text{possible values of } j \text{ for each } i} * \underbrace{n}_{\text{possible values for } i}) = O(n^2)$$

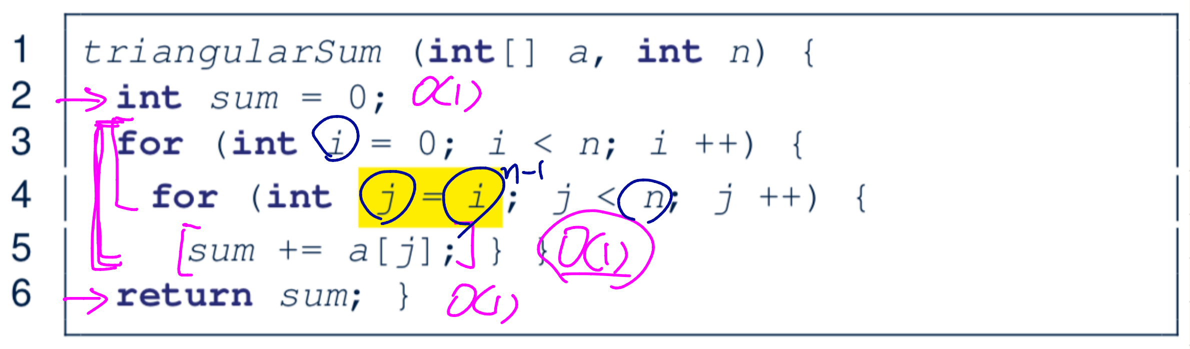
# Determining Asymptotic Upper Bound (2)



```
sumMaxAndCrossProducts (int[] a, int n) {
  int max = a[0];
  for(int i = 1; i < n;) {
    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; }
```

$$O(\boxed{n} \underline{+} n^2) = O(n^2)$$

# Determining Asymptotic Upper Bound (3)



```
triangularSum (int[] a, int n) {
  int sum = 0;   O(1)
  for (int i = 0; i < n; i ++) {
    for (int j = i; j < n; j ++) {
      sum += a[j]; } }   O(1)
  return sum; }   O(1)
```

n-1

I

0 [ J 0 ... n-1 ] n

1 [ J I ... n-1 ] n-1

2 ...

n-1 [ J=n-1 ] 1

$$n + (n-1) + \cdots + (1) = O(n^2)$$

```
m( int[] a, int n) {
       i=0   i<n²   i++
    for( ;  - . . ;  ) {
           O(1)
    }
}
```

$O(n^2)$

$O(n)$ ? ✗

7

|   |   |   |   |   |
|---|---|---|---|---|
| 2 | 3 | 1 | 4 | 6 |

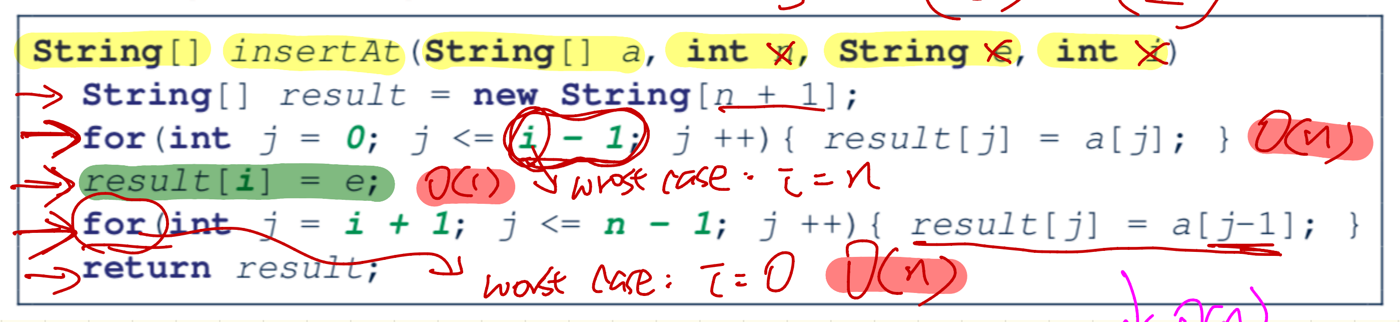
|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 2 | 3 | 1 | 4 | 6 | 7 |

# Inserting into an array

Input → | 3 | 1 | 4 | 2 |

output → | 1 | 2 | 3 | 4 |

3 "Jim" 1



```
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(n)

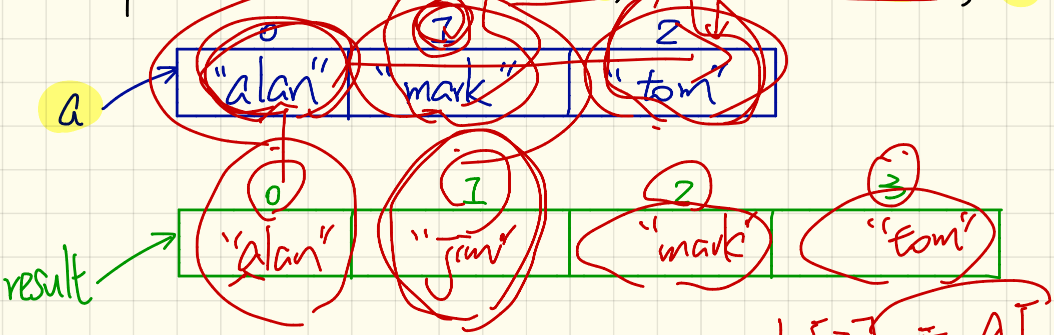
O(1)

worst case: i = n

worst case: i = 0 O(n)

O(n)

Example: insertAt({"alan", "mark", "tom"}, 3, "jim", 1)



RT ?

result[2] = a[1]

[3] = a[2]

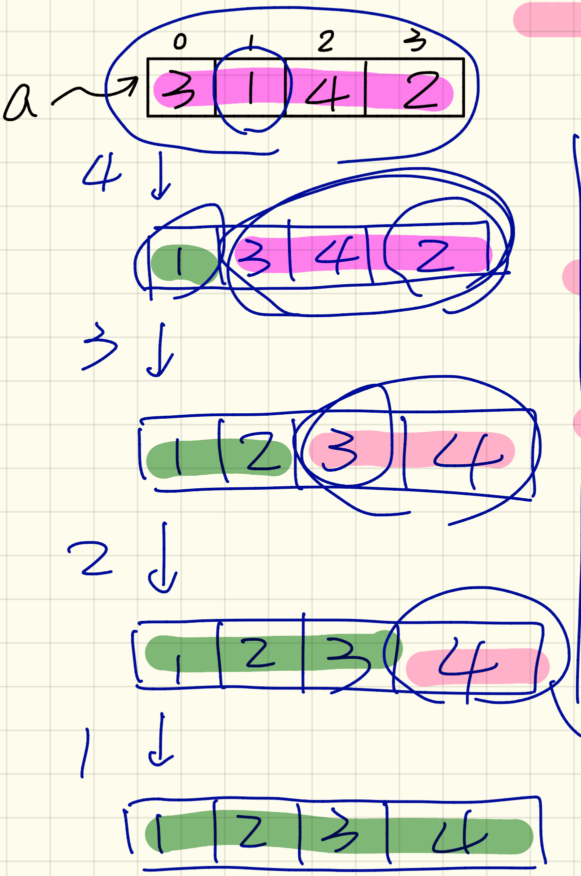
# Selection Sort : Idea

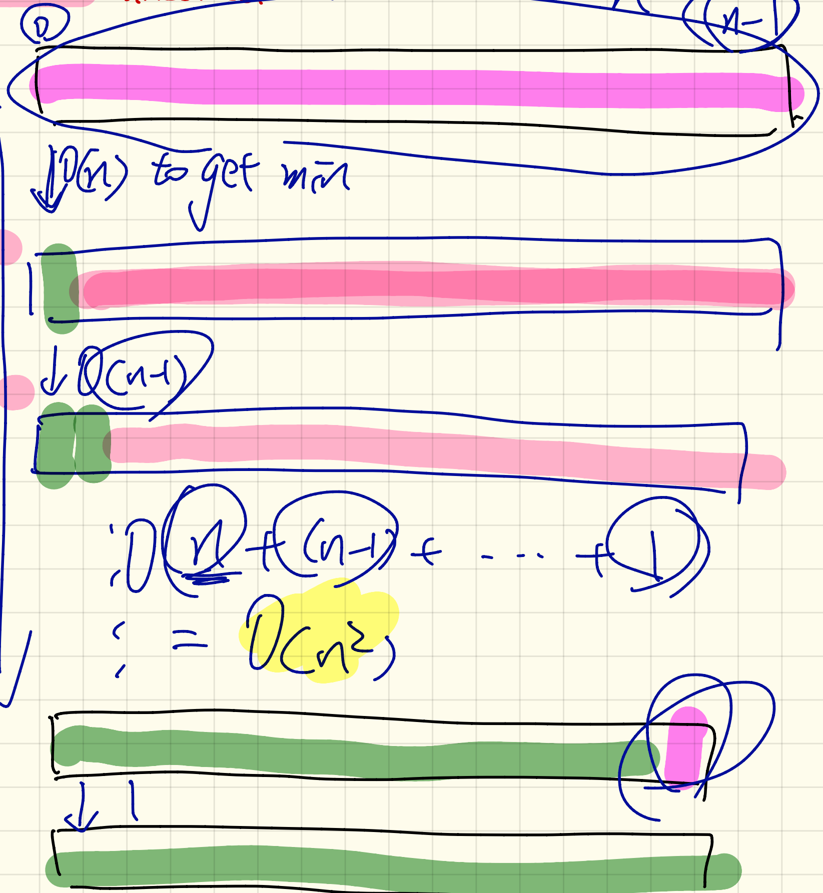
🟩 Sorted ✓

🟥 unsorted ✓

How many selections?

n





$$O(n + (n-1) + \cdots + 1)$$

$$= O(n^2)$$

# Insertion Sort : Idea

🟩 sorted

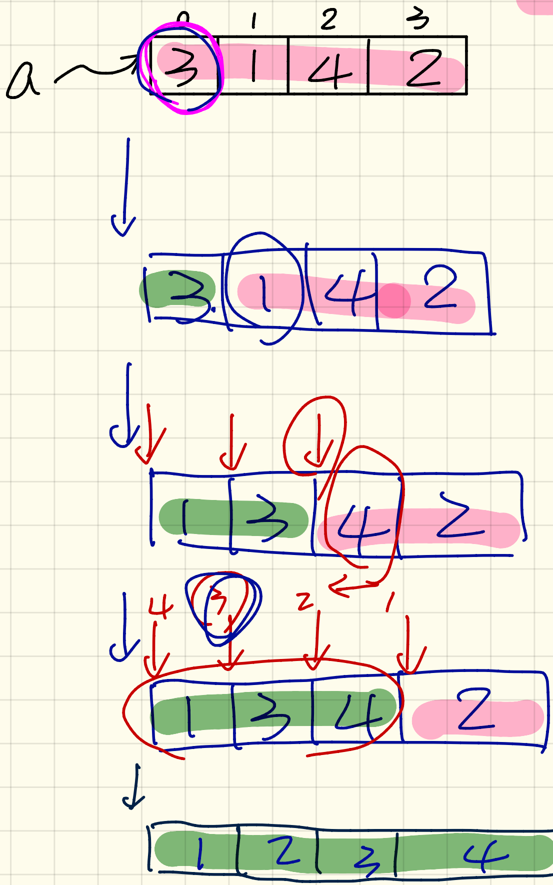
🟥 unsorted

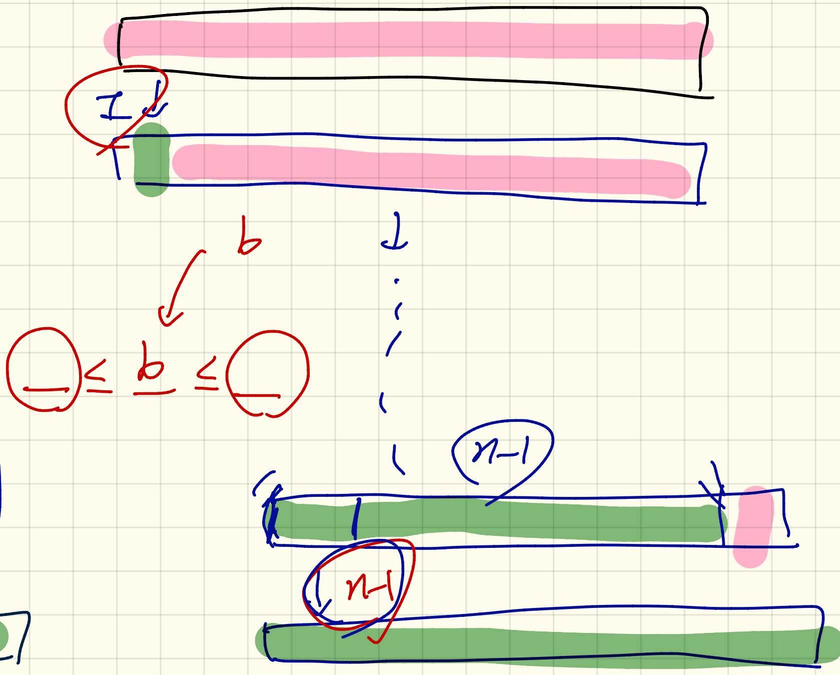
$$O(1+2+\cdots+(n-1))$$

$$= O(n^2) \quad \boxed{1000} \quad \underline{1M}$$

pick the left-most element of 🟥

insert it to the correct spot in 🟩





Monday Oct. 22

Lecture 12

100 marks? 18.98%

A/A+? 33.47%

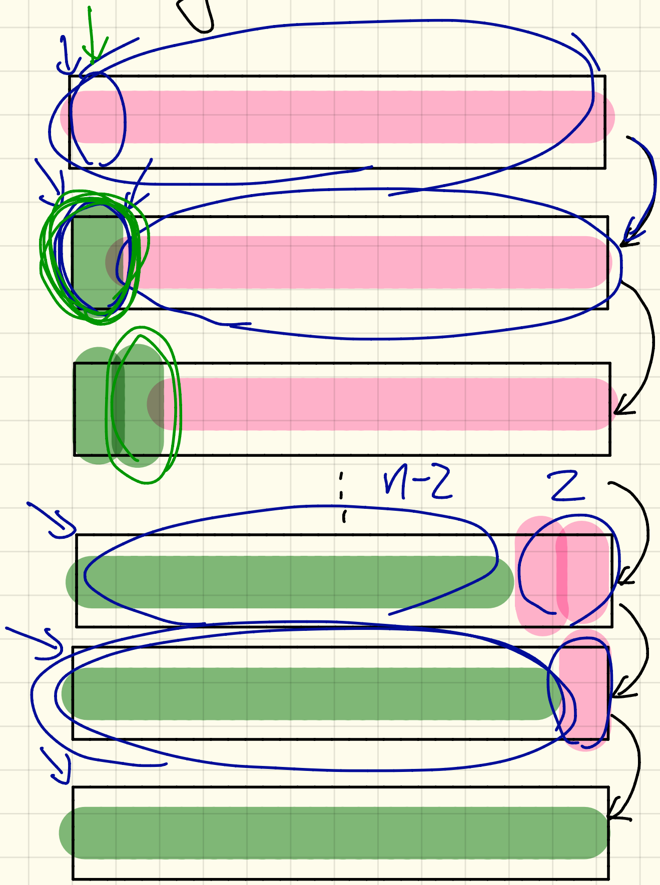
E/F 33.58%

feedback

- Lab Test 1 marks (programming)

- Lab Test 2 postponed: Monday Nov. 5

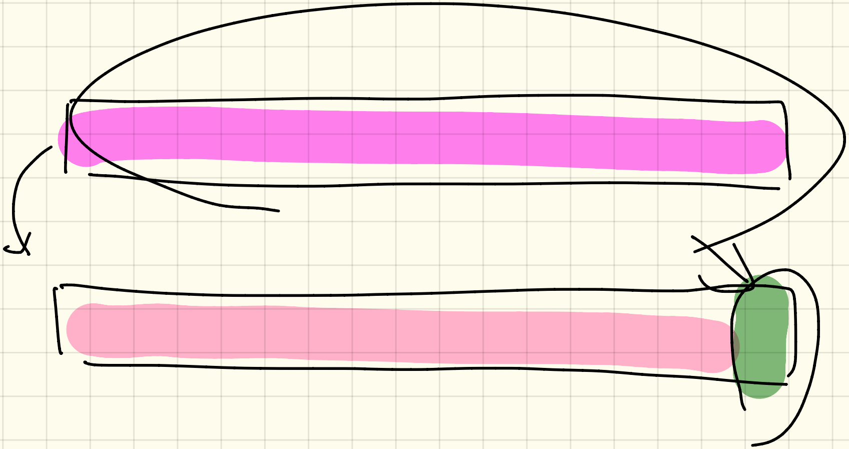
# Sorting



| Selection Sort | Insertion Sort |
|---|---|
| n | 1 |
| n-1 | 2 |
| ⋮ | ⋮ |
| 2 | n-2 |
| 1 | n-1 |

$\frac{(n+1)*n}{2}$ is $O(n^2)$

$\frac{(1+(n-1))(n-1)}{2}$ is $O(n^2)$



SS IS $O(n^2)$

$n = 1000 \rightarrow 1M$ PO.

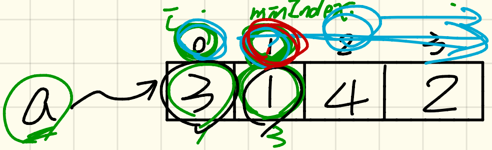
$n = 1M \rightarrow (1M)^2$ PO

Merge Sort $O(n \cdot \log n)$

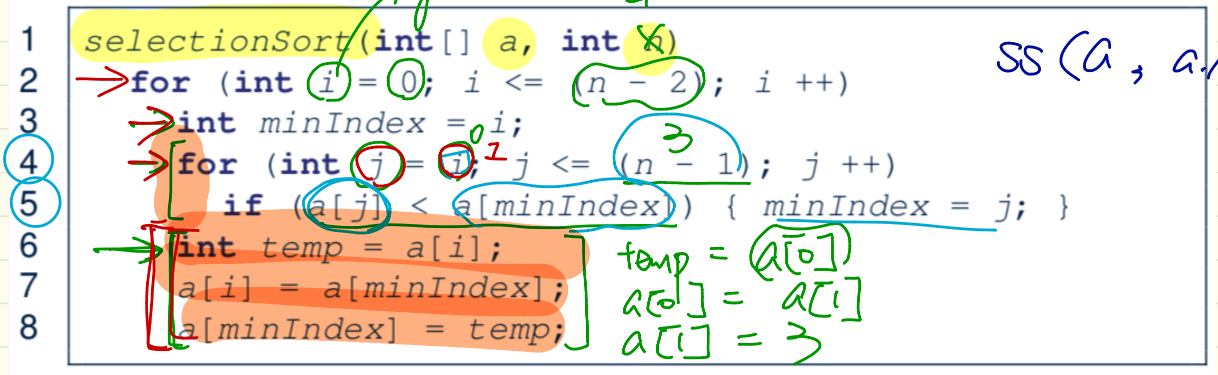
$n = 1000 \rightarrow 1000 \cdot \log_2 1000 = 10k$

$n = 1M \rightarrow 1M \cdot \log_2 1000 = 20k$

# Selection Sort : Code



green area



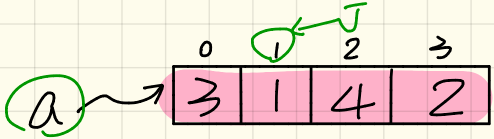
```
1 selectionSort(int[] a, int n)
2   for (int i = 0; i <= (n - 2); i ++)
3     int minIndex = i;
4     for (int j = i; j <= (n - 1); j ++)
5       if (a[j] < a[minIndex]) { minIndex = j; }
6     int temp = a[i];
7     a[i] = a[minIndex];
8     a[minIndex] = temp;
```

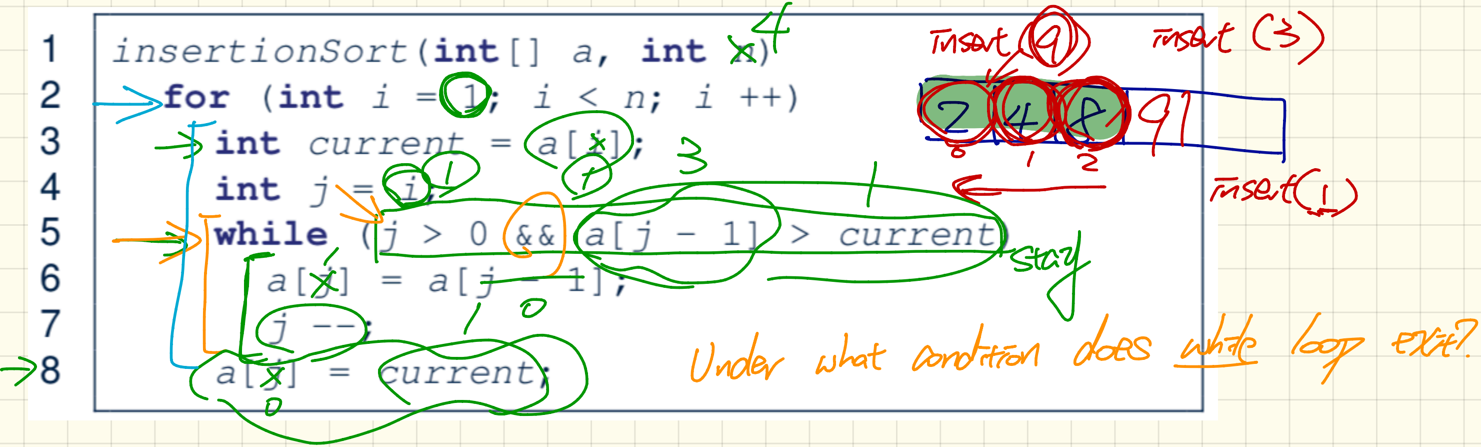
SS (a, a.legth)

temp = a[0]
a[0] = a[1]
a[1] = 3

| i | Inner loop j fom ? to ? | minIndex at L6 | after L6~L8, a becomes? |
|---|---|---|---|
| 0 | 0 1 2 3 | 1 a[1] 1 | 1 \| 3 \| 4 \| 2<br>i minIndex |
| 1 | 1 2 3 | 3 a[3] 2 | i<br>1 \| 2 \| 4 \| 3 |

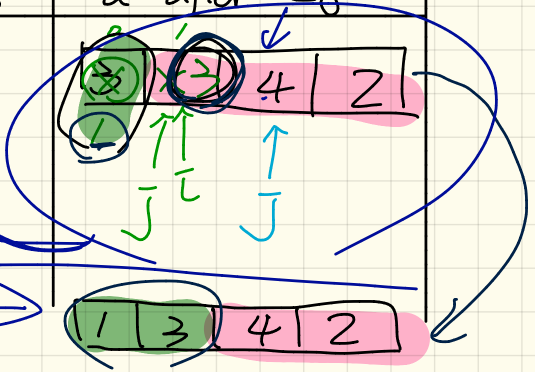
# Insertion Sort : Code





```
1 insertionSort(int[] a, int n)
2   for (int i = 1; i < n; i ++)
3     int current = a[i];
4     int j = i;
5     while (j > 0 && a[j - 1] > current)
6       a[j] = a[j - 1];
7       j --;
8     a[j] = current;
```

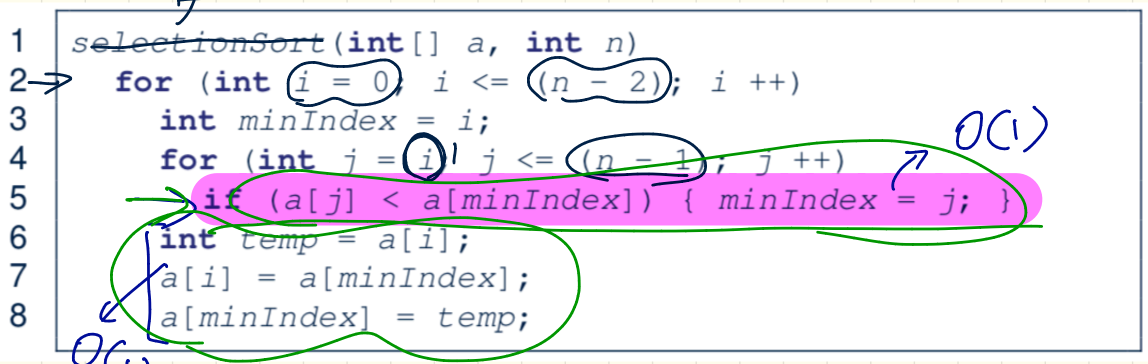
Under what condition does while loop exit?

| i | current | j at L8 | a at L8 | a after L8 |
|---|---|---|---|---|
| 1 | 1 | 0 | |  |
| 2 | | | | |

```
while (j > 0 && a[j-1] > current)

↳ exit: !(j>0 && a[j-1] > current)

        |||

        j ≤ 0 || a[j-1] ≤ current
```

# Asymptotic Upper Bounds

f



```
~~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)

i

0 [j 0 … n-1] n

1 [j 1 … n-1] n-1

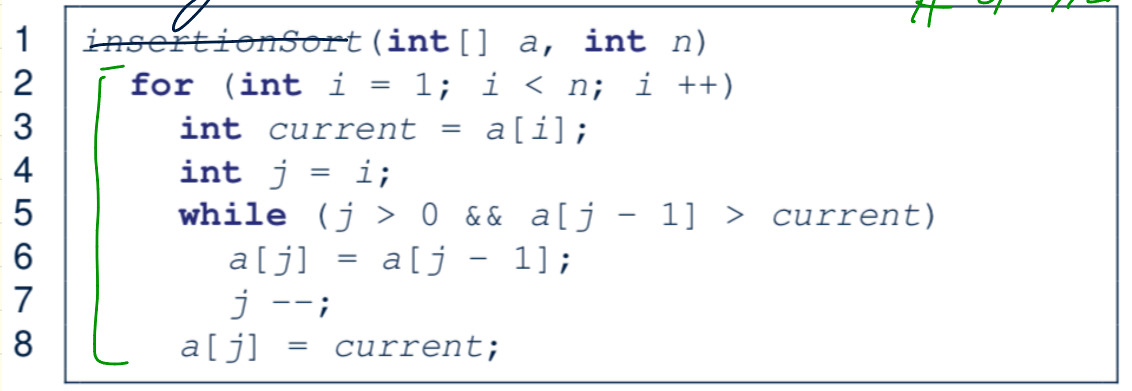
2

⋮

n-2 [j n-2 … n-1] 2

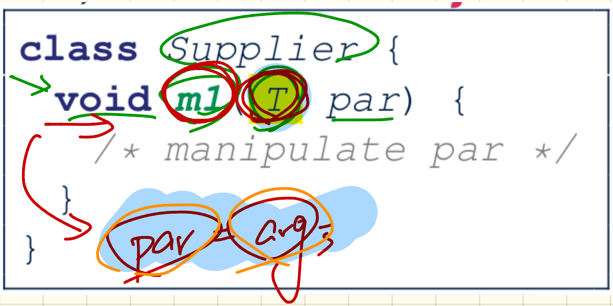
$$O\big(\underbrace{(n + (n-1) + \cdots + 2)}_{\#\text{ of iterations}} \cdot \underbrace{1}_{\text{RT for each iteration}}\big) = O(n^2)$$

g



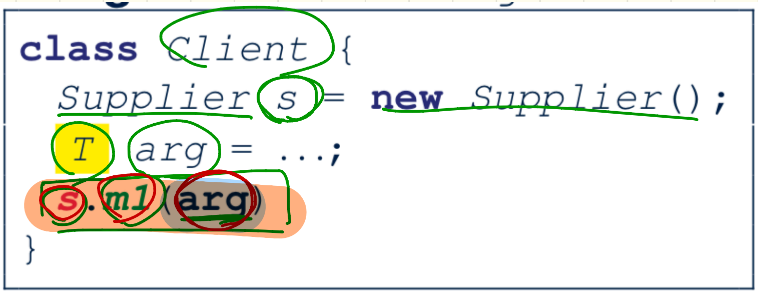
```
~~insertionSort~~(int[] a, int n)
  for (int i = 1; i < n; i ++)
    int current = a[i];
    int j = i;
    while (j > 0 && a[j - 1] > current)
      a[j] = a[j - 1];
      j --;
    a[j] = current;
```

# Call by Value (1)



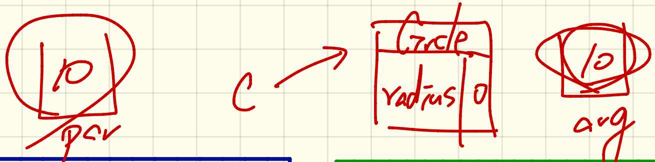
```
class Supplier {
  void m1(T par) {
    /* manipulate par */
  }
}
```

par = arg;



```
class Client {
  Supplier s = new Supplier();
  T arg = ...;
  s.m1(arg)
}
```

## T being Primitive



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

par = arg;

10

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

# Call by Value (2)

Is pink obj going to be changed?

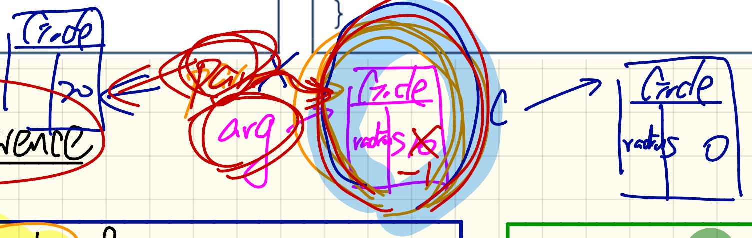
① par = new Circle(20) No

② par.setRadius(-1); YES

```
class Supplier {
  void m1( T  par) {
    /* manipulate par */
  }
}
```

```
class Client {
  Supplier s = new Supplier();
  T  arg = ...;
  s.m1(arg)
}
```

T being Reference

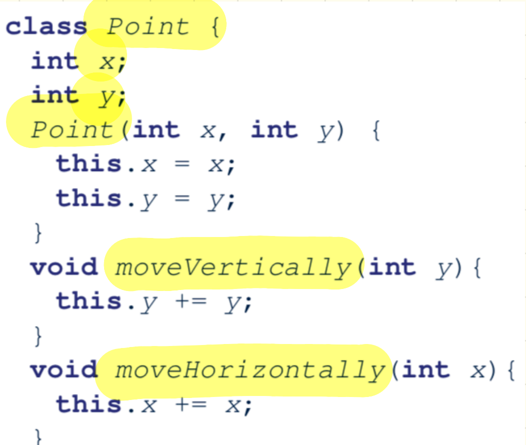


```
class Circle {
  int radius;
  Circle (int radius) { this.radius = radius; }
  void setRadius ( Circle par ) {
    par = new Circle(20);
    this.radius = par.radius;
  }
}
```

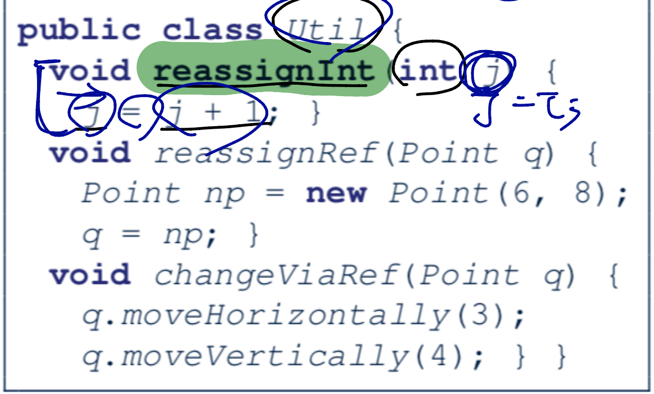
par = arg

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

# Call by Value : Primitive Type

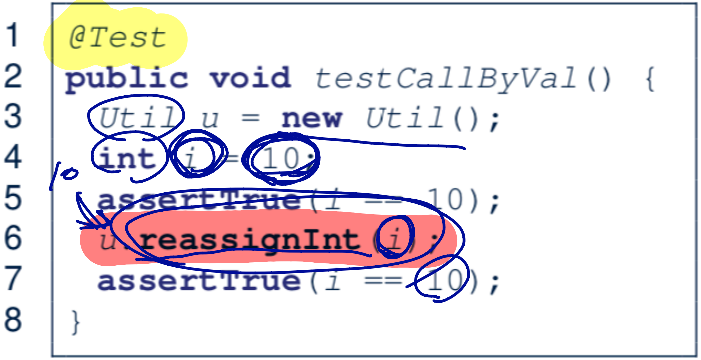


```java
class Point {
  int x;
  int y;
  Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  void moveVertically(int y){
    this.y += y;
  }
  void moveHorizontally(int x){
    this.x += x;
  }
}
```



```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); } }
```

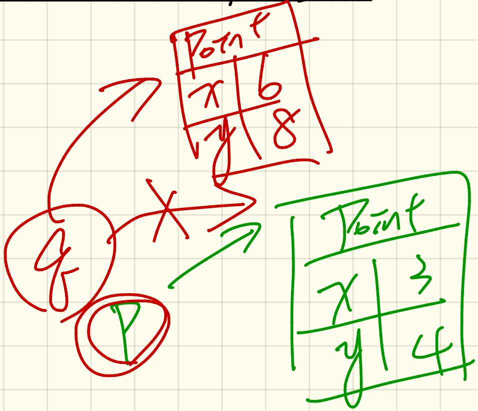
j = i ;

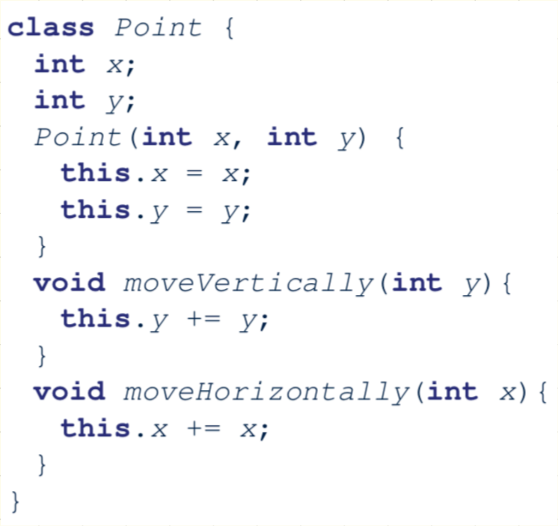


```java
@Test
public void testCallByVal() {
  Util u = new Util();
  int i = 10;
  assertTrue(i == 10);
  u.reassignInt(i);
  assertTrue(i == 10);
}
```

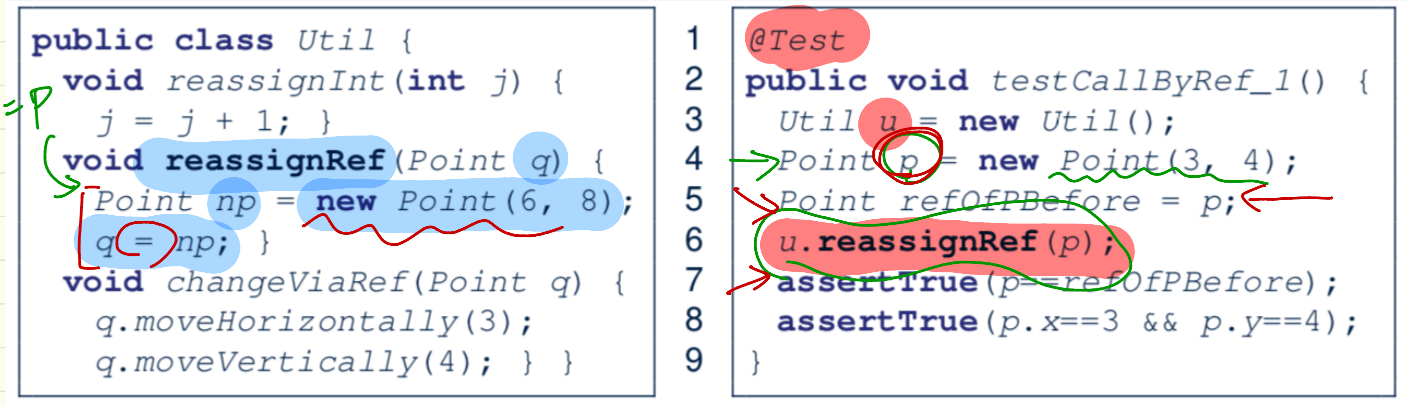
10

# Call by Value : Reference Type (1)





```
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

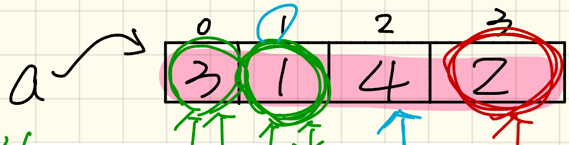
```
public class Util {
  void reassignInt(int j) {
    j = j + 1; }
  void reassignRef(Point q) {
    Point np = new Point(6, 8);
    q = np; }
  void changeViaRef(Point q) {
    q.moveHorizontally(3);
    q.moveVertically(4); } }
```

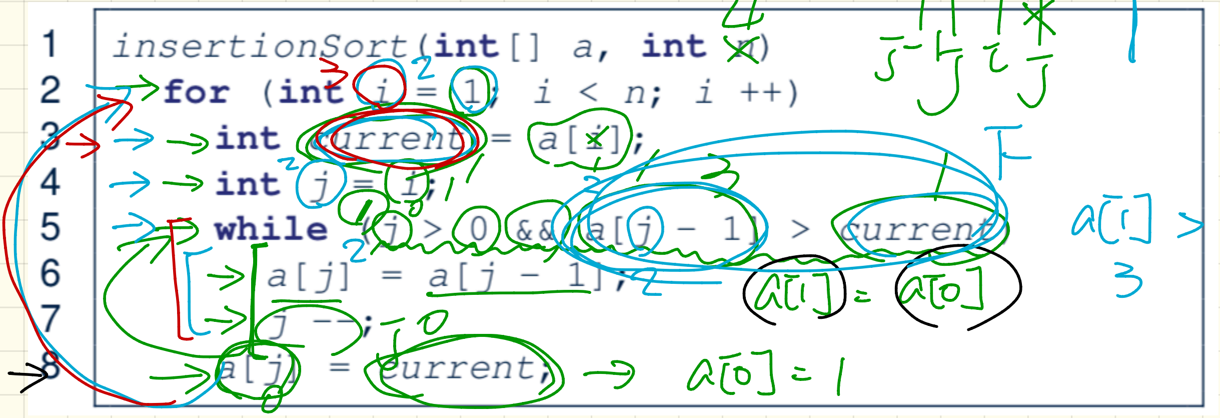
```
@Test
public void testCallByRef_1() {
  Util u = new Util();
  Point p = new Point(3, 4);
  Point refOfPBefore = p;
  u.reassignRef(p);
  assertTrue(p==refOfPBefore);
  assertTrue(p.x==3 && p.y==4);
}
```

Wednesday Oct. 24

Lecture 13

# Insertion Sort : Code





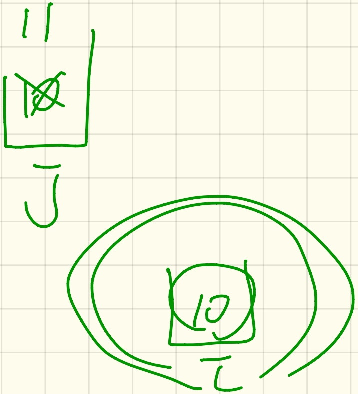
```
1  insertionSort(int[] a, int n)
2    for (int i = 1; i < n; i ++)
3      int current = a[i];
4      int j = i;
5      while (j > 0 && a[j - 1] > current)
6        a[j] = a[j - 1];
7        j --;
8      a[j] = current;
```

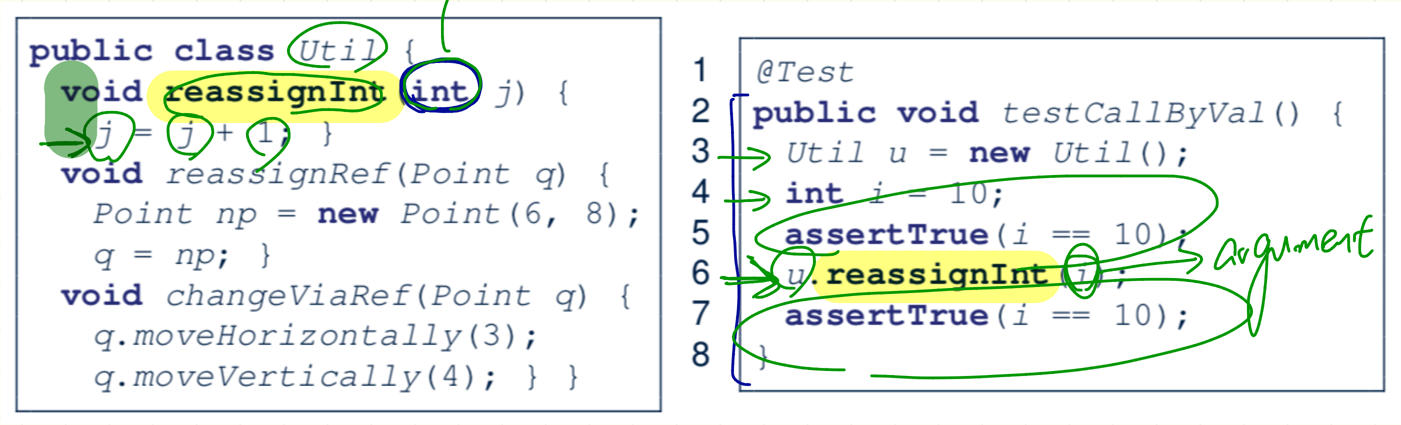
| i | current | j at L8 | a at L8 | a after L8 |
|---|---|---|---|---|
| 1 | a[1] 1 | | 3 ~~1~~ 3 4 2 | 1 3 4 2 |
| 2 | a[2] 4 | | | i |
| 3 | a[3] 2 | | 1 ~~3~~ 2 ~~4~~ 3 ~~2~~ 4 | 1 3 4 2 |

# Call by Value : Primitive Type

Scope of j

Implicitly: j = i



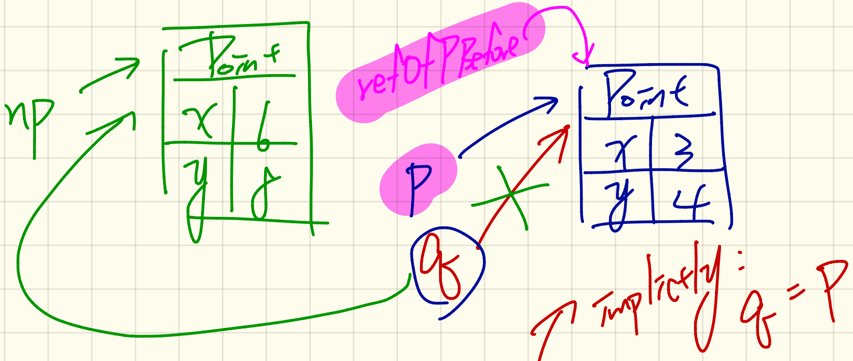


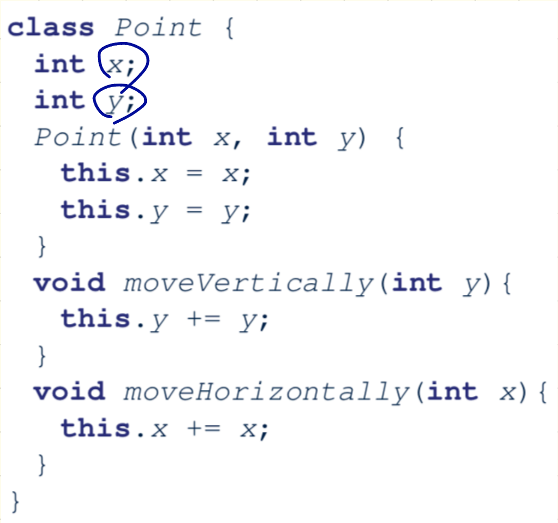
```
public class Util {
  void reassignInt(int j) {
    j = j + 1; }
  void reassignRef(Point q) {
    Point np = new Point(6, 8);
    q = np; }
  void changeViaRef(Point q) {
    q.moveHorizontally(3);
    q.moveVertically(4); } }
```

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

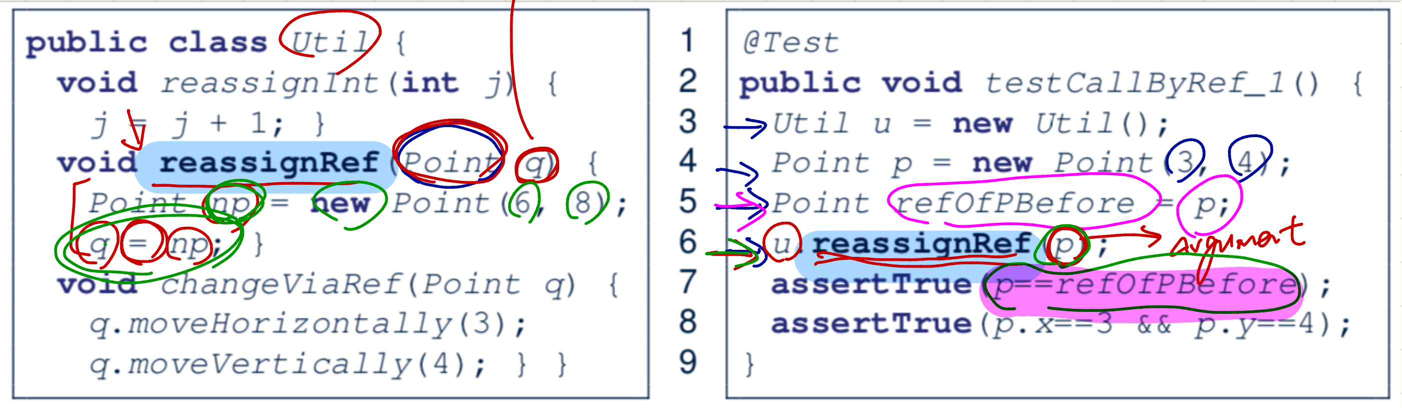
argument

# Call by Value : Reference Type (1)





```
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;
  }
}
```



```
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); } }
```

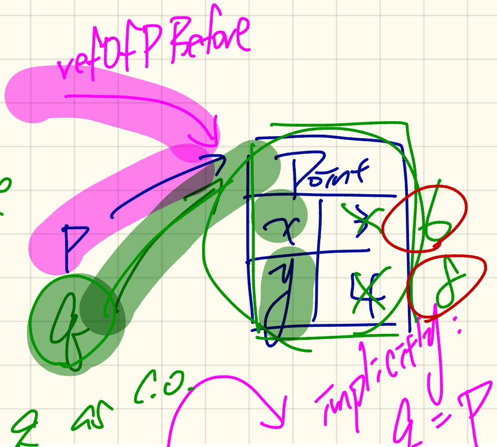
```
@Test
public void testCallByRef_1() {
  Util u = new Util();
  Point p = new Point(3, 4);
  Point refOfPBefore = p;
  u.reassignRef(p);
  assertTrue(p==refOfPBefore);
  assertTrue(p.x==3 && p.y==4);
}
```

argument

# Call by Value : Reference Type (2)

1. p and q are aliases of the same object.

2. To modify that object, you can use p or q as C.O.

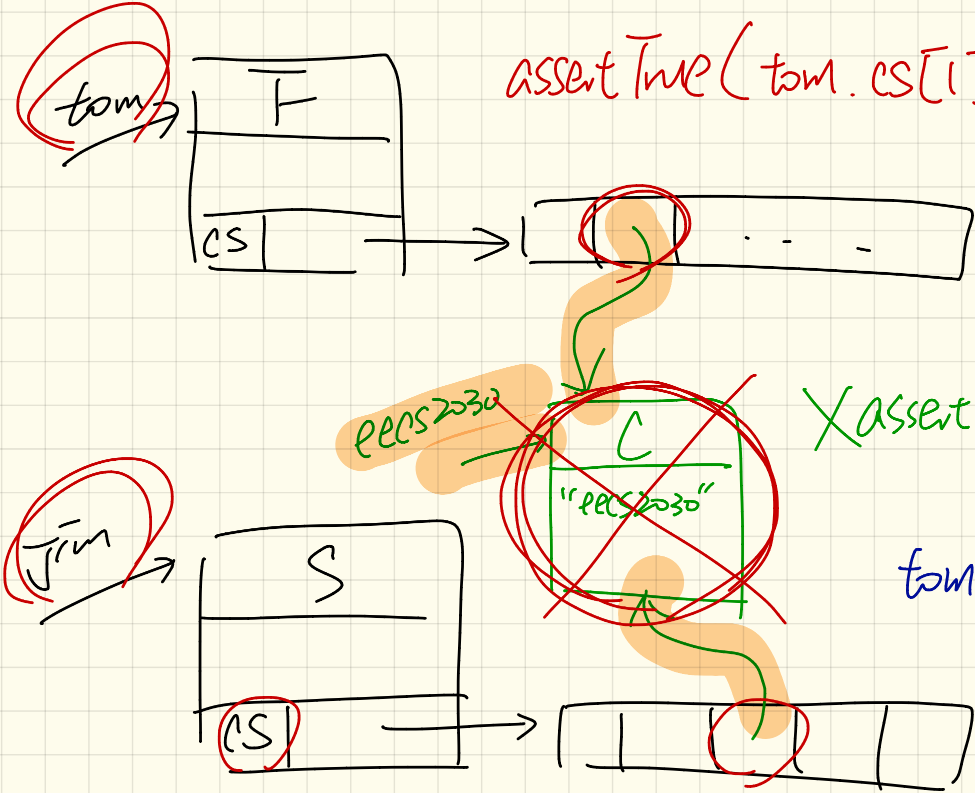


```java
class Point {
  int x;
  int y;
  Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  void moveVertically(int y){
    this.y += y;
  }
  void moveHorizontally(int x){
    this.x += x;
  }
}
```

```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); } }
```

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

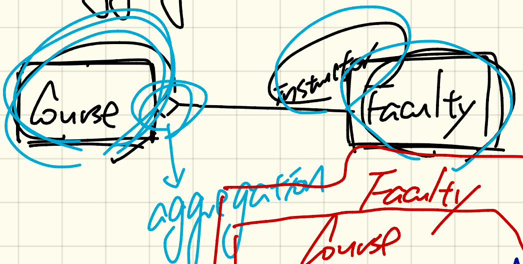
assertTrue(tom.cs[1] == jim.cs[2])



X assertEquals( , ) .equals

tom.cs[1].setName("eecs2040")

# Aggregation (1)



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

eecs2030

eecs3311

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

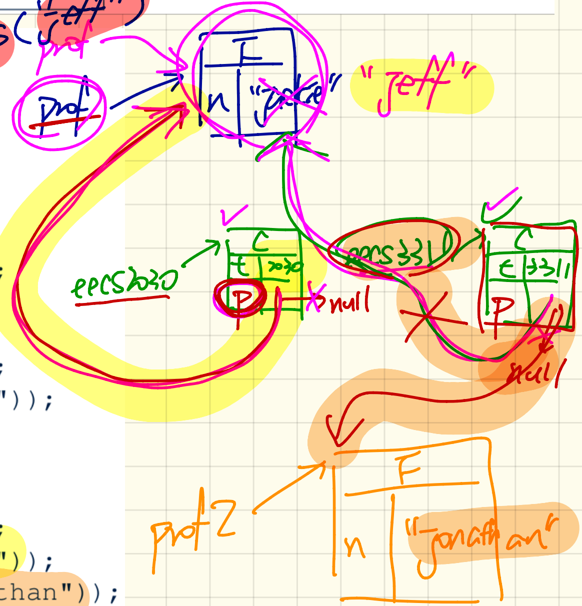
Faculty

Course

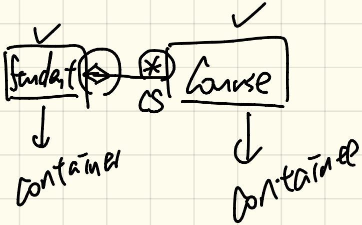
eecs2030.getProf().getName().equals("Jeff")

String

```
@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)



```
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; }
}
```

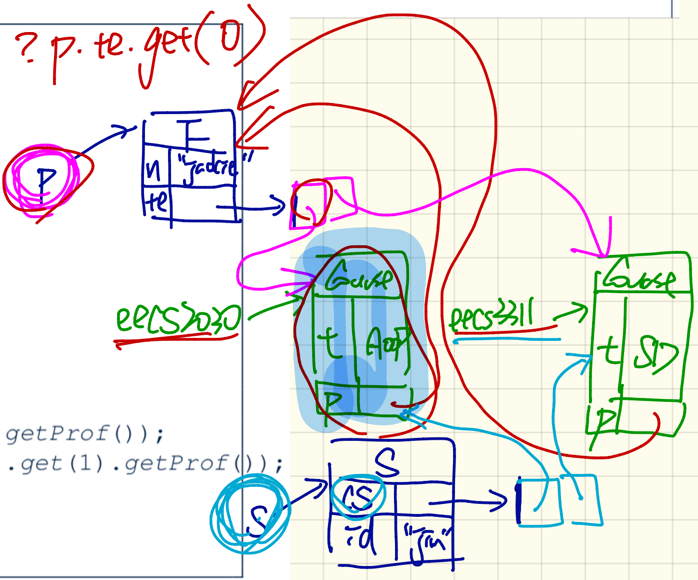
```
class Course { String title; ~~}~~ prof
```

```
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);
  eecs3311.setProf(p);
  p.addTeaching(eecs2030);
  p.addTeaching(eecs3311);
  s.addCourse(eecs2030);
  s.addCourse(eecs3311);

  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));
}
```

eecs2030 == ? p.te.get(0)

==
eecs2030
s.cs.get(0)
p.te.get(0)



Monday Oct. 29

Lecture 14

~ Lab Test 1 Marks (Written)

~ Lab Test 2 Guide

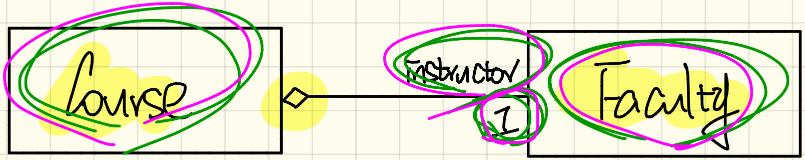
↳ Programming 40%

↳ Written 60%

# Review: Aggregation

# Java Implementation

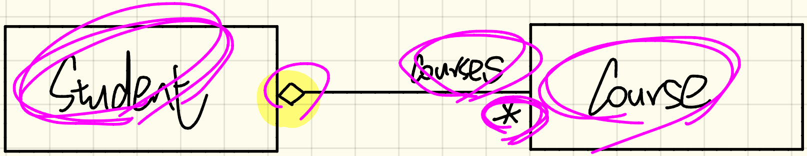
## Single Containee



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

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

## Multiple Containees



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

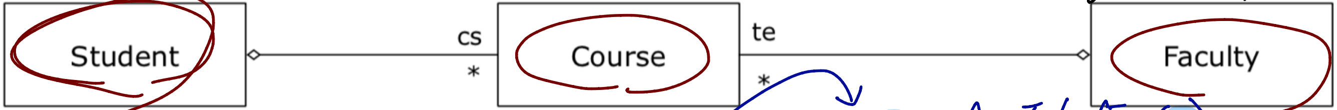
```
class Course {
    ...
}
```

# Dot Notation for Navigation Aggregations

f1 → Course

c2.prof → Course

s.cs.get(1).prof



c2.getCourseTitleOfIns(0) → "2001"

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

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

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

String getInstructorName(int i)

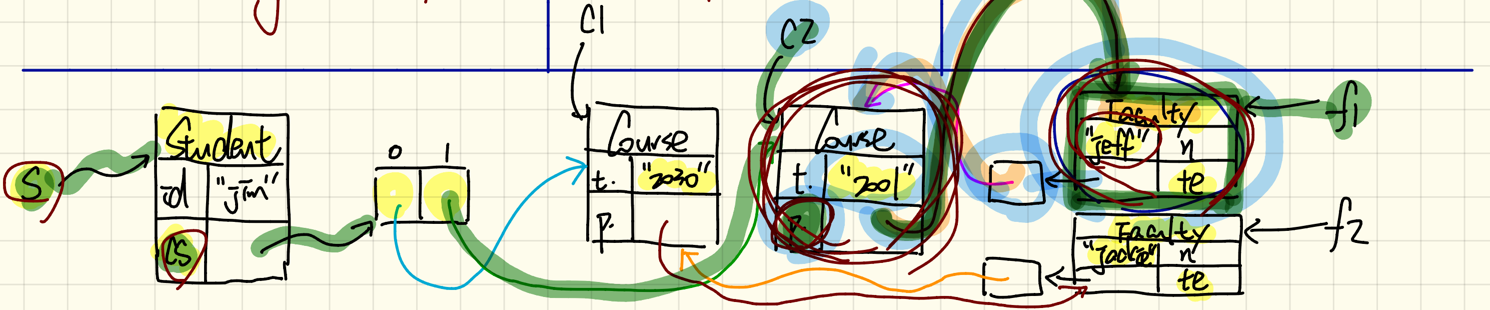
this.cs.get(i).prof.name

String getInstructorName()

this.prof.name

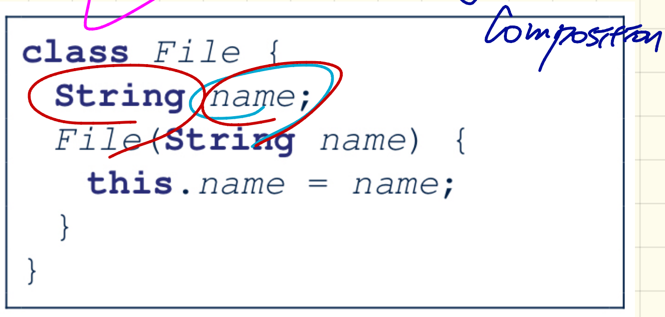
String getName()

this.name

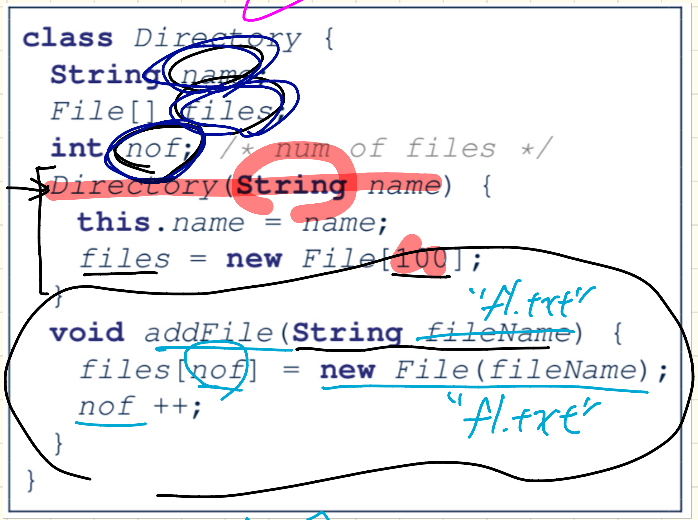


# Composition : No Sharing

Composition.



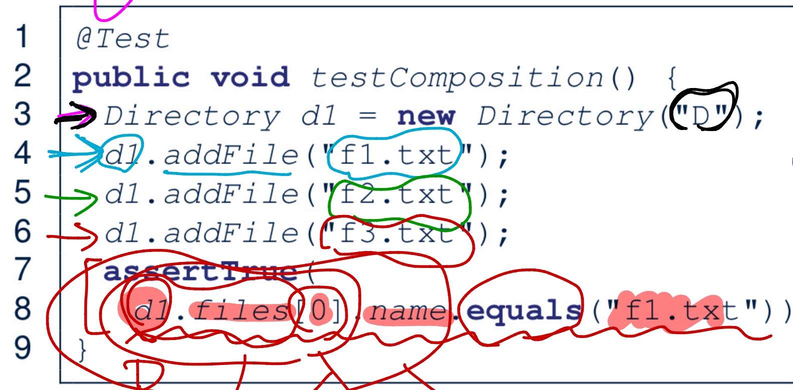
```java
class File {
  String name;
  File(String name) {
    this.name = name;
  }
}
```



```java
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"



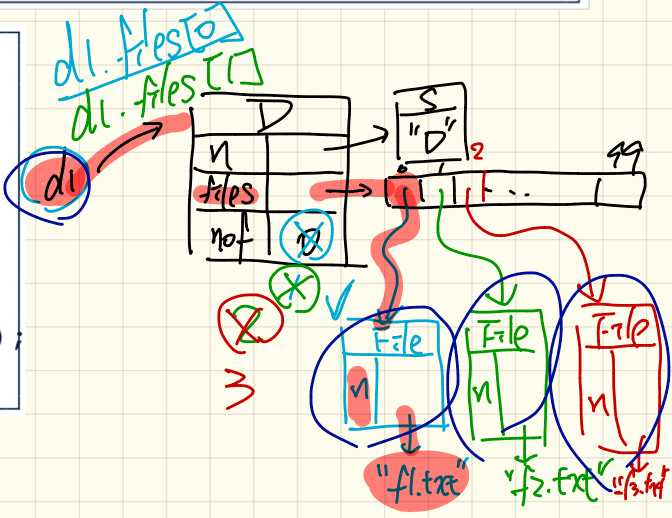
```java
@Test
public void testComposition() {
  Directory d1 = new Directory("D");
  d1.addFile("f1.txt");
  d1.addFile("f2.txt");
  d1.addFile("f3.txt");
  assertTrue(
    d1.files[0].name.equals("f1.txt"));
}
```

D

File[]

File

String



# Copy Constructor

```
class Directory {
    Directory (Directory other) {

    }
}
```

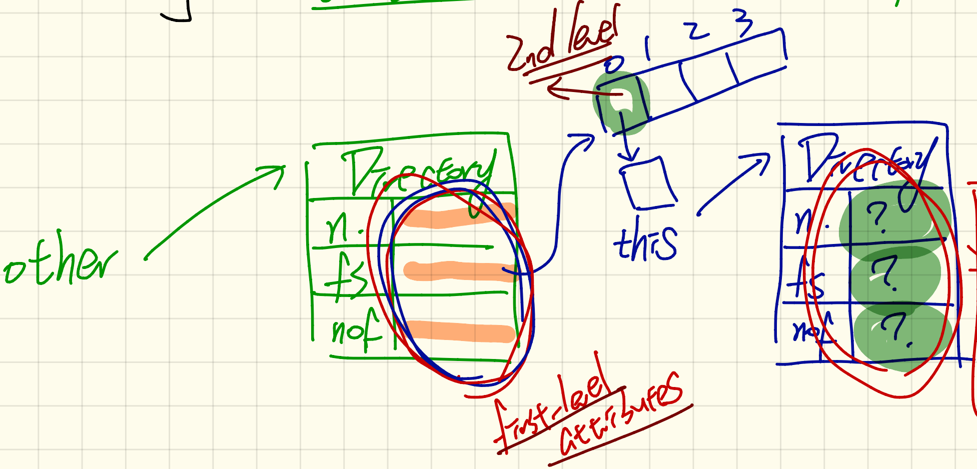
copy constructor

constructor

attributes : name, files, nof

Shallow copy :

copy attribute values for 1st level



this . n = other . n

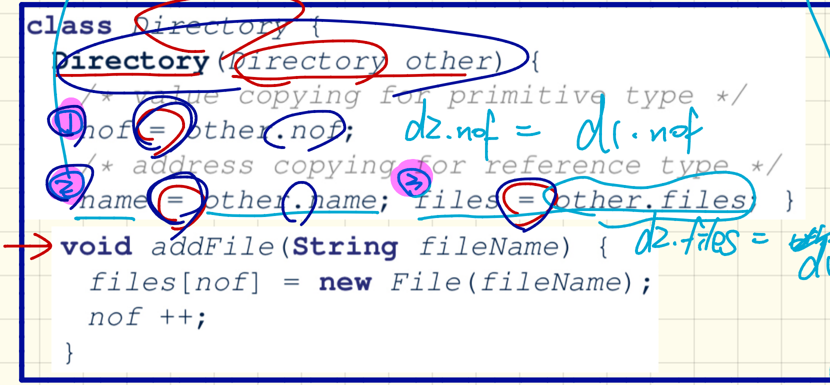
this . fs = other . fs

O(1)

this . nof = nof

# Composition : Copy Constructor ( Shallow Copy )

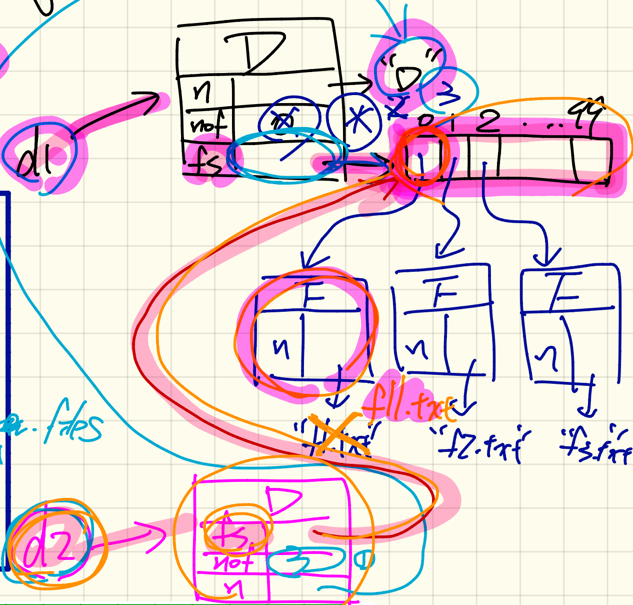
d2. name = d1. name



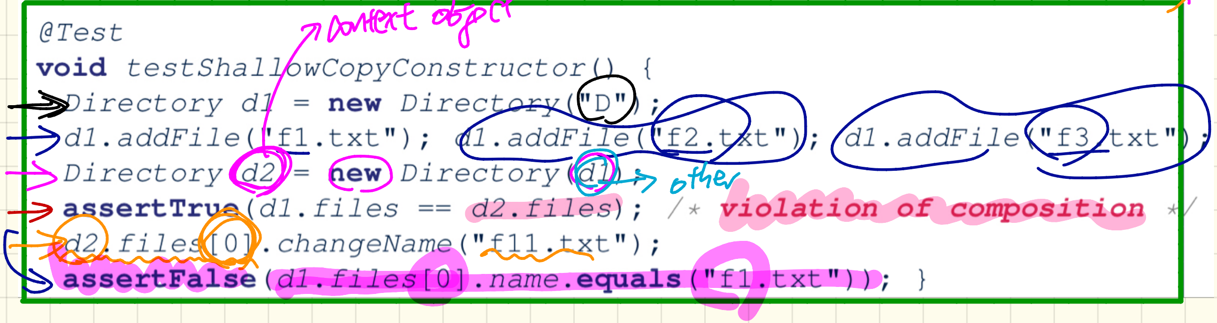
```
class Directory {
  Directory(Directory other) {
    /* value copying for primitive type */
    nof = other.nof;
    /* address copying for reference type */
    name = other.name; files = other.files; }
  void addFile(String fileName) {
    files[nof] = new File(fileName);
    nof ++;
  }
```

d2.nof = d1.nof

d2.files = ~~other~~ d1.files



context object



```
@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")); }
```

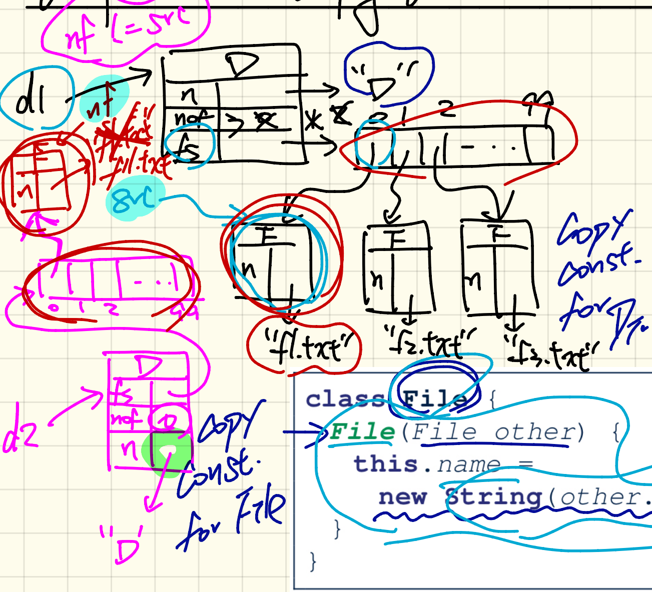
other

# Composition : Copy Constructor ( Deep Copy )

d2.files == d1.files F

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

calling a constructor in the current class



Copy const. for File

Copy const. for Directory

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

deep copy

```
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) { ... } }
```

d1

**

```
@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")); }
```

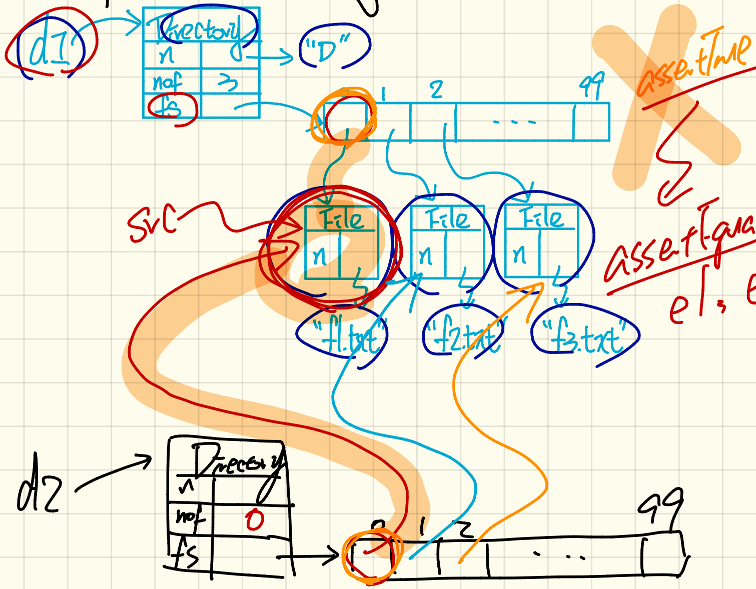
Wednesday Oct. 31

Lecture 15

# Composition : Copy Constructor

Q: Composition?

assertTrue(d1.files[0] != d2.files[0]);



```
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++) {
                 d1
      File src = other.files[i];
d2    ~~File nf = new File(src);~~
      this.addFile(nf src); } }
  void addFile(File f) { ... } }
```

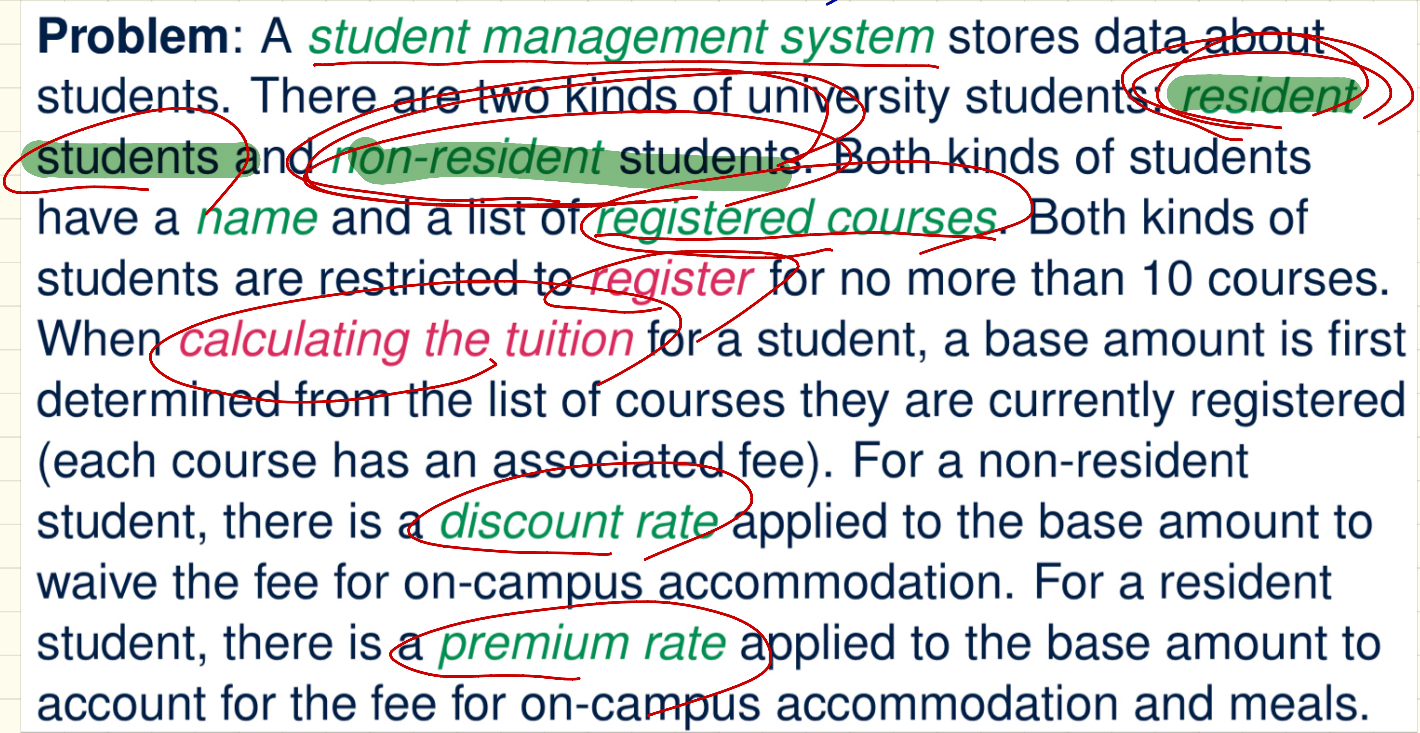
```
files[nof 0] = f;
nof ++;
```

```
@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")); }
```

# 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 *discount rate* applied to the base amount to waive the fee for on-campus accommodation. For a resident student, there is a *premium rate* applied to the base amount to account for the fee for on-campus accommodation and meals.

s0.

| Student | |
|---|---|
| kind | "R" "NR" |
| pr | — |
| dr | — |

$2x + 2y$

$2(x+y)$

```
if(s.kind.equals("R")){
    ...
}
else if (s.kind.equals("NR"))
    ...
}
```

# Testing Student Classes (without inheritance)

```
class ResidentStudent {
  String name;
  Course[] registeredCourses;
  int numberOfCourses;
  double premiumRate; /* there's a mutator met
  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;
  }
```

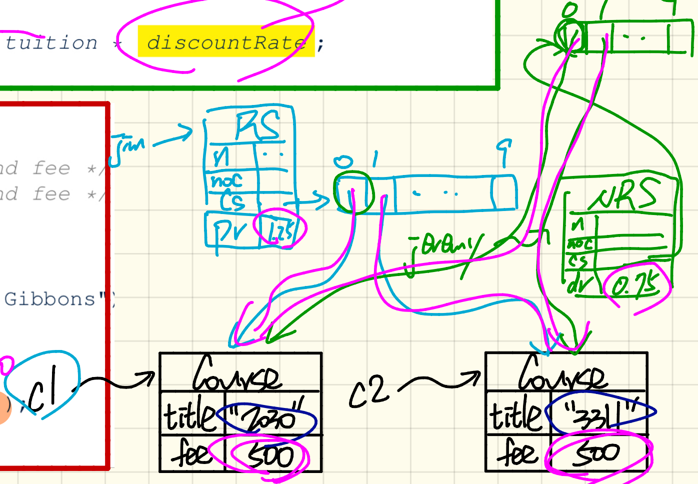
```
class NonResidentStudent {
  String name;
  Course[] registeredCourses;
  int numberOfCourses;
  double discountRate; /* there's a mutator me
  NonResidentStudent (String name) {
    this.name = name;
    registeredCourses = new Course[10];
  }
  void register(Course c) {
    registeredCourses[numberOfCourses] = 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());
  }
}
```

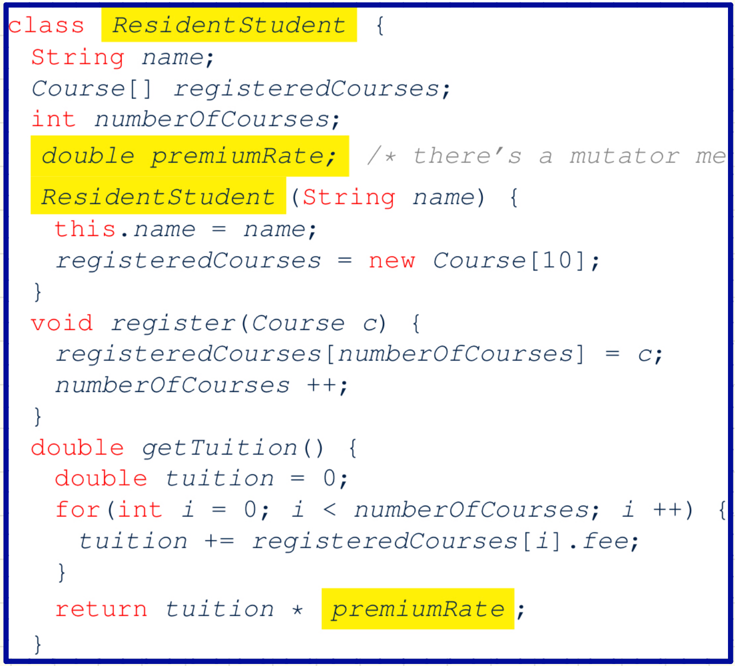
1250

750

① ②



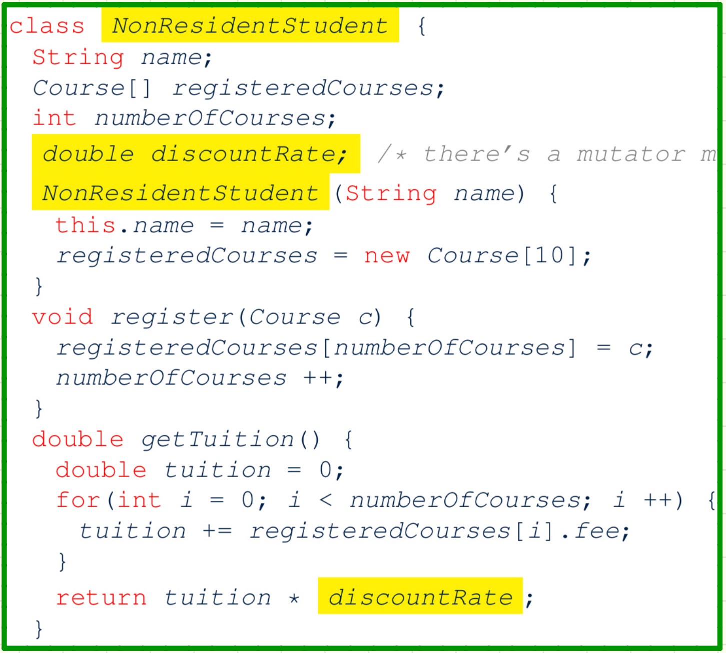
# Student Classes (without inheritance): Maintenance Problem



```
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;
  }
```

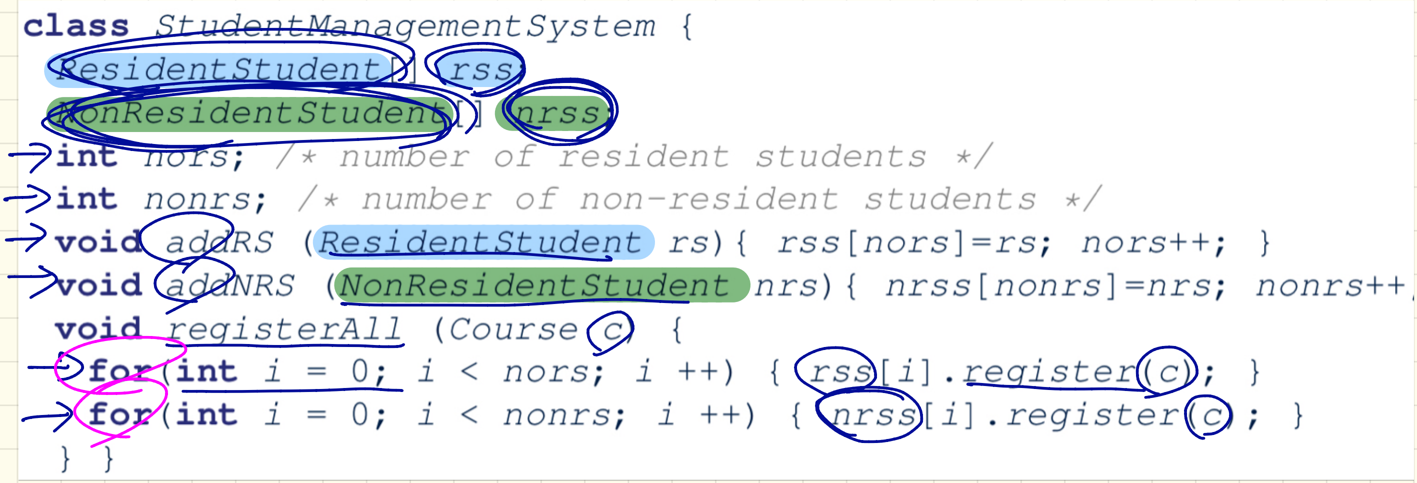
Maintenance:

1. Change on registration policy.
2. Change on tuition calculation.

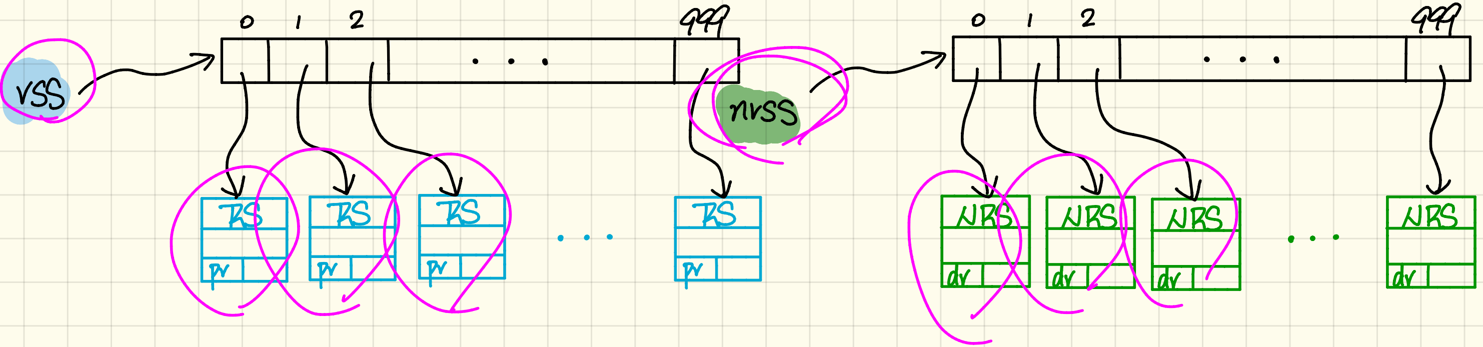


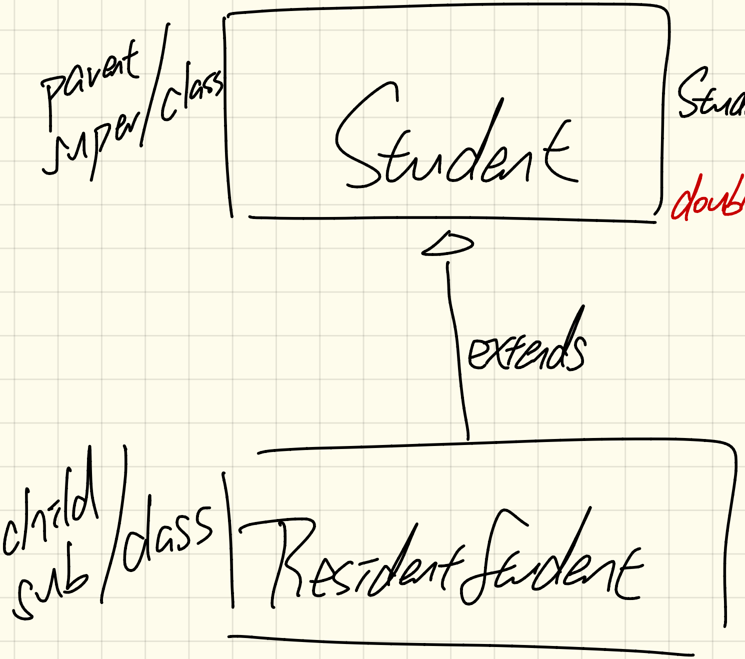
```
class NonResidentStudent {
  String name;
  Course[] registeredCourses;
  int numberOfCourses;
  double discountRate; /* there's a mutator m
  NonResidentStudent (String name) {
    this.name = name;
    registeredCourses = new Course[10];
  }
  void register(Course c) {
    registeredCourses[numberOfCourses]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

# A Collection of Students (without inheritance)

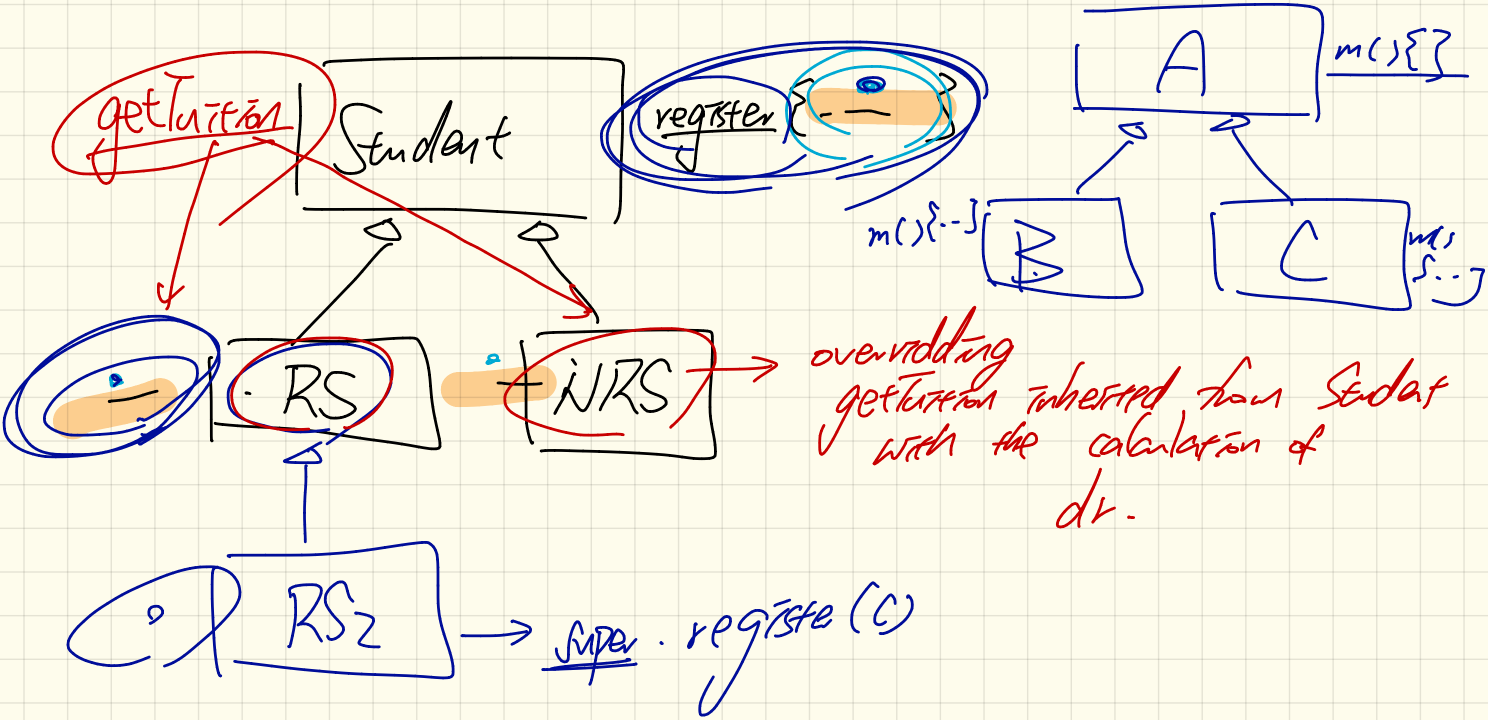


```
class StudentManagementSystem {
  ResidentStudent[] rss;
  NonResidentStudent[] nrss;
  int nors; /* number of resident students */
  int nonrs; /* number of non-resident students */
  void addRS (ResidentStudent rs){ rss[nors]=rs; nors++; }
  void addNRS (NonResidentStudent nrs){ nrss[nonrs]=nrs; 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 ( String name ){ - - - }
double getTuition ( ) { }

ResidentStudent(String name){
    super(name);
}
double getTuition(){
    return super.getTuition() * pr;
}
```



overridding getTuition inherited from Student with the calculation of dr.

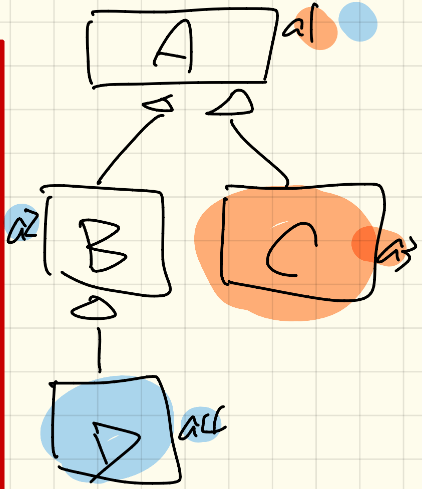
super.register(c)

# Student Classes (with inheritance)

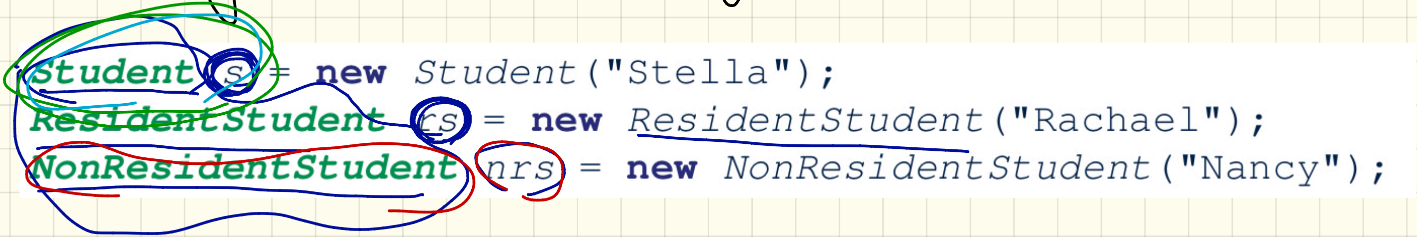
```java
class Student {
  String name;
  Course[] registeredCourses;
  int numberOfCourses;
  Student (String name) {
    this.name = name;
    registeredCourses = new Course[10];
  }
  void register(Course c) {
    registeredCourses[numberOfCourses] = c;
    numberOfCourses ++;
  }
  double getTuition()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

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

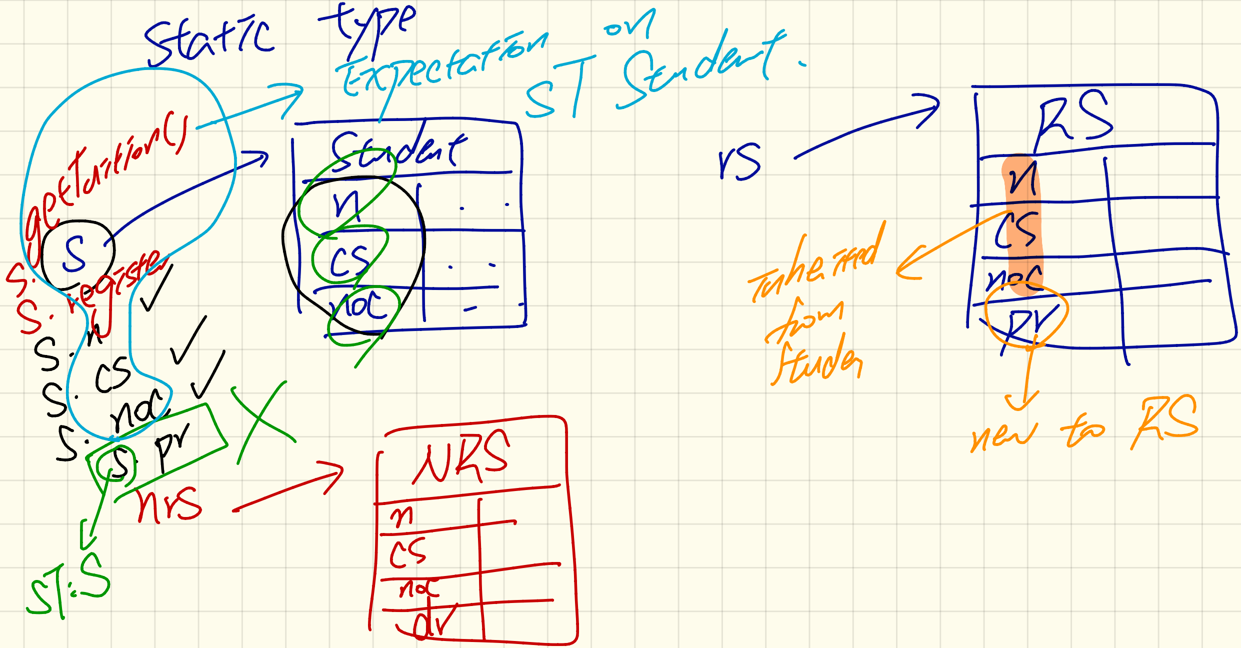
```java
class NonResidentStudent extends Student {
  double discountRate; /* there's a mutator method
  NonResidentStudent (String name) { super(name); }
  /* register method is inherited */
  double getTuition() {
    double base = super.getTuition();
    return base * discountRate;
  }
}
```



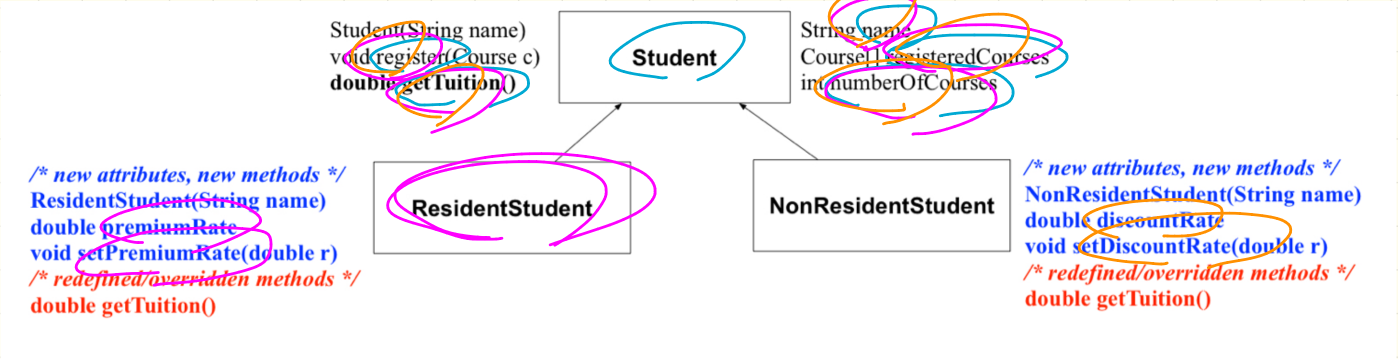
# Visualizing Parent and Child Objects



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



# Student classes (with inheritance): Expectations



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

**Student**

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()**

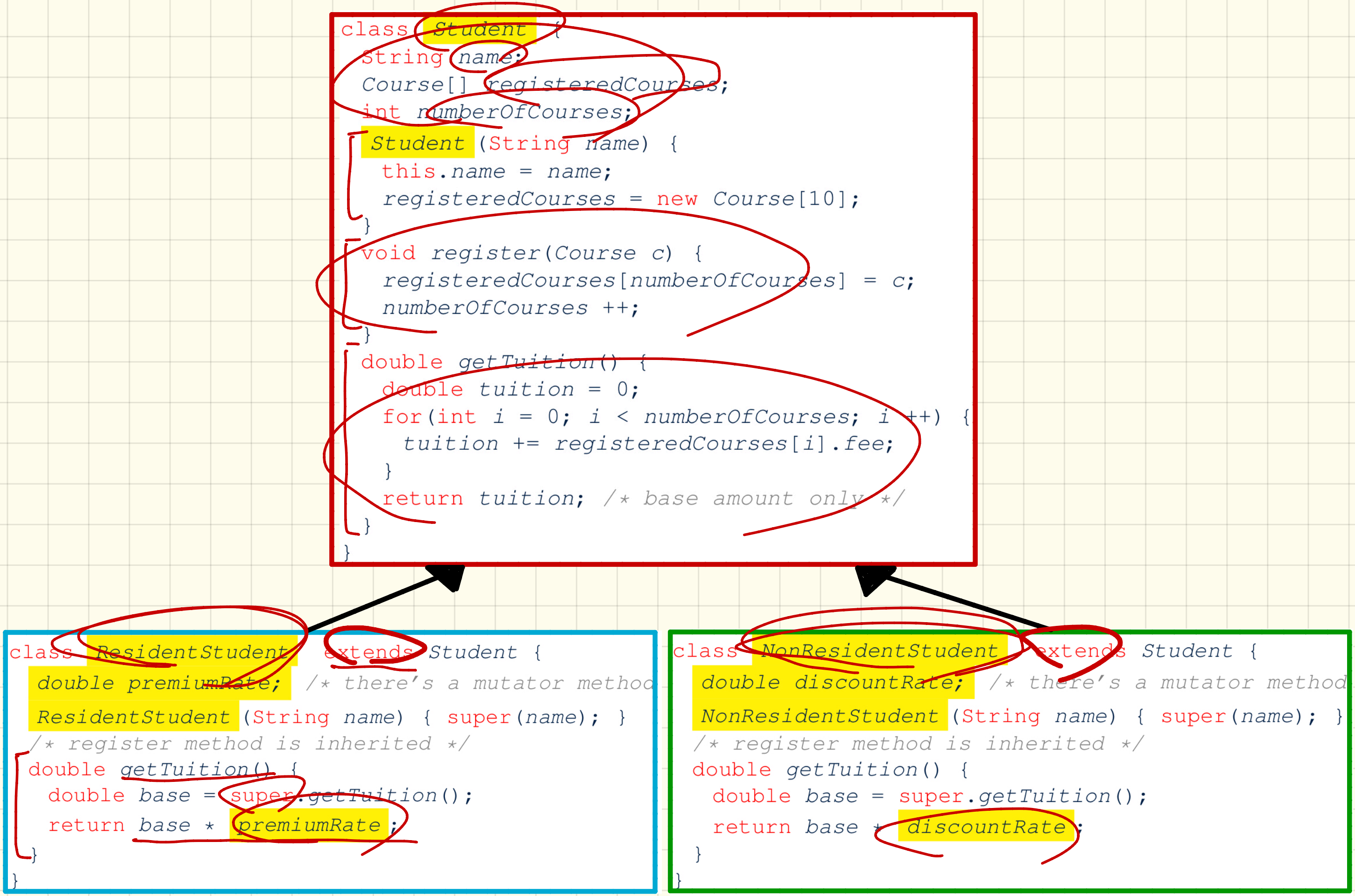
```
Student s = new Student("Stella");
ResidentStudent rs = new ResidentStudent("Rachael");
NonResidentStudent nrs = new NonResidentStudent("Nancy");
```

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

Monday Nov. 5

Lecture 16

# Review: Student Classes (with inheritance)

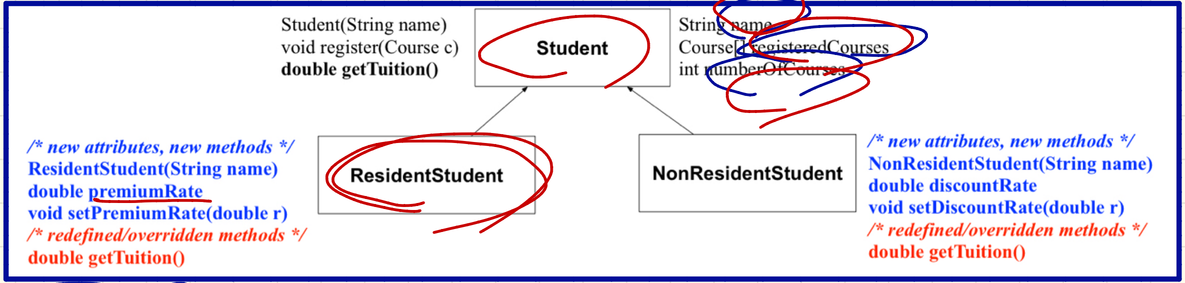


```
class Student {
  String name;
  Course[] registeredCourses;
  int numberOfCourses;
  Student (String name) {
    this.name = name;
    registeredCourses = new Course[10];
  }
  void register(Course c) {
    registeredCourses[numberOfCourses] = c;
    numberOfCourses ++;
  }
  double getTuition() {
    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 method
  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: Visualizing Parent and Child Objects



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

Student

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()**

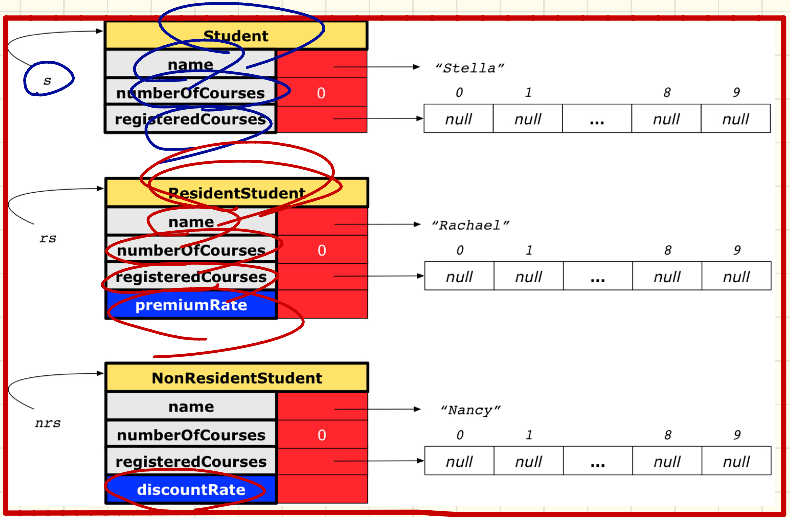
Inheritance Hierarchy

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

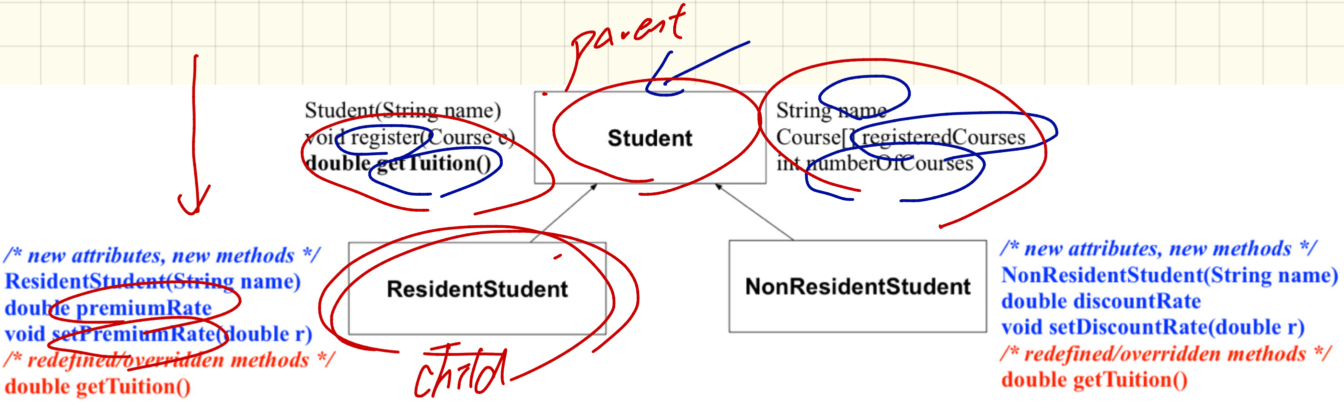
Declaring Variable

static type

Runtime Object Structure



# Review: Static Types and Expectations

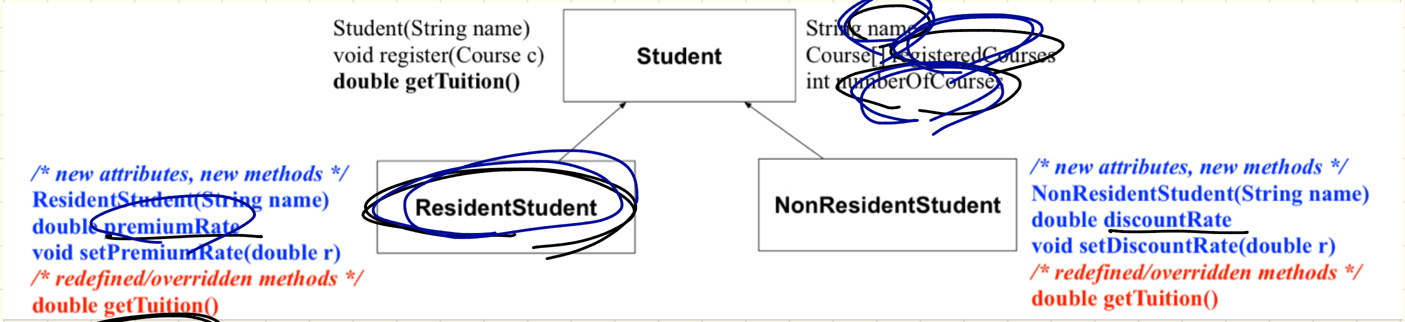

parent
Student(String name)
void register(Course c)
double getTuition()
Student
String name
Course[] registeredCourses
int numberOfCourses
/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()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

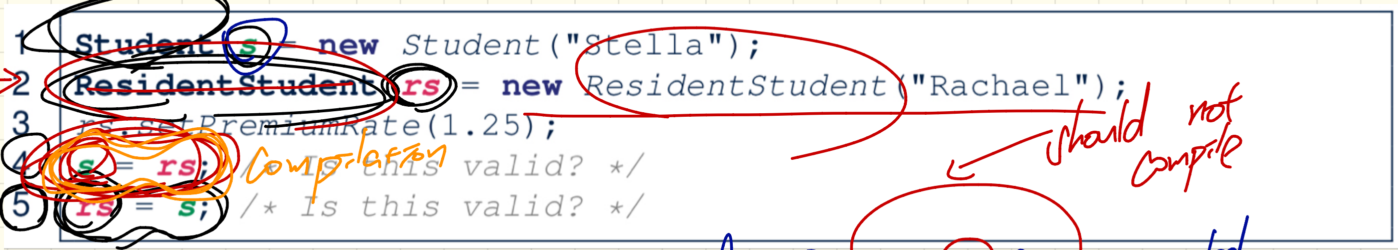
ST

```
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. | ✓ | | | | | × | | ✓ | |

# 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? */
```

Compilation

should not compile

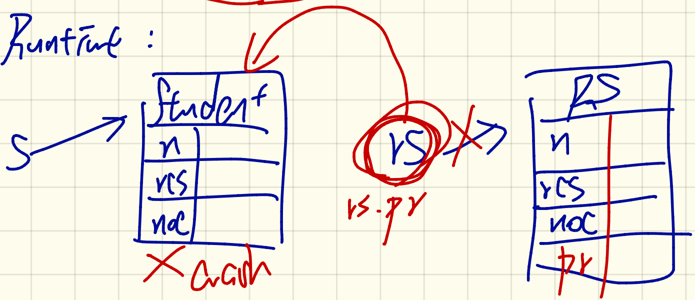
## Expectations

s.name
s.rcs
s.noc
s.pr
s.dr X

rs.name
rs.rcs
rs.noc
rs.pr
rs.dr X

Assume rs = s compiled

Runtime :

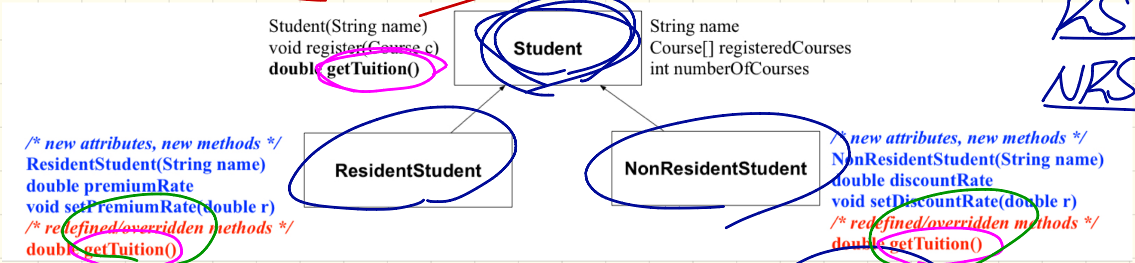


# Intuition: Dynamic Binding

Student s

RS s

NRS s

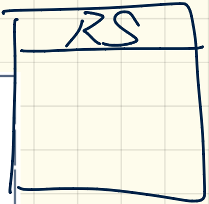


Student(String name)
void register(Course c)
**double getTuition()**

String name
Course[] registeredCourses
int numberOfCourses

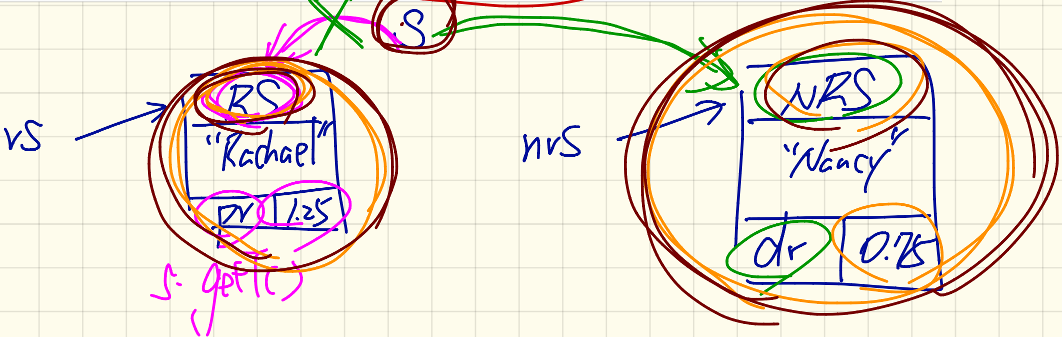
*/* new attributes, new methods */*
**ResidentStudent(String name)**
**double premiumRate**
**void setPremiumRate(double r)**
*/* redefined/overridden methods */*
**double getTuition()**

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



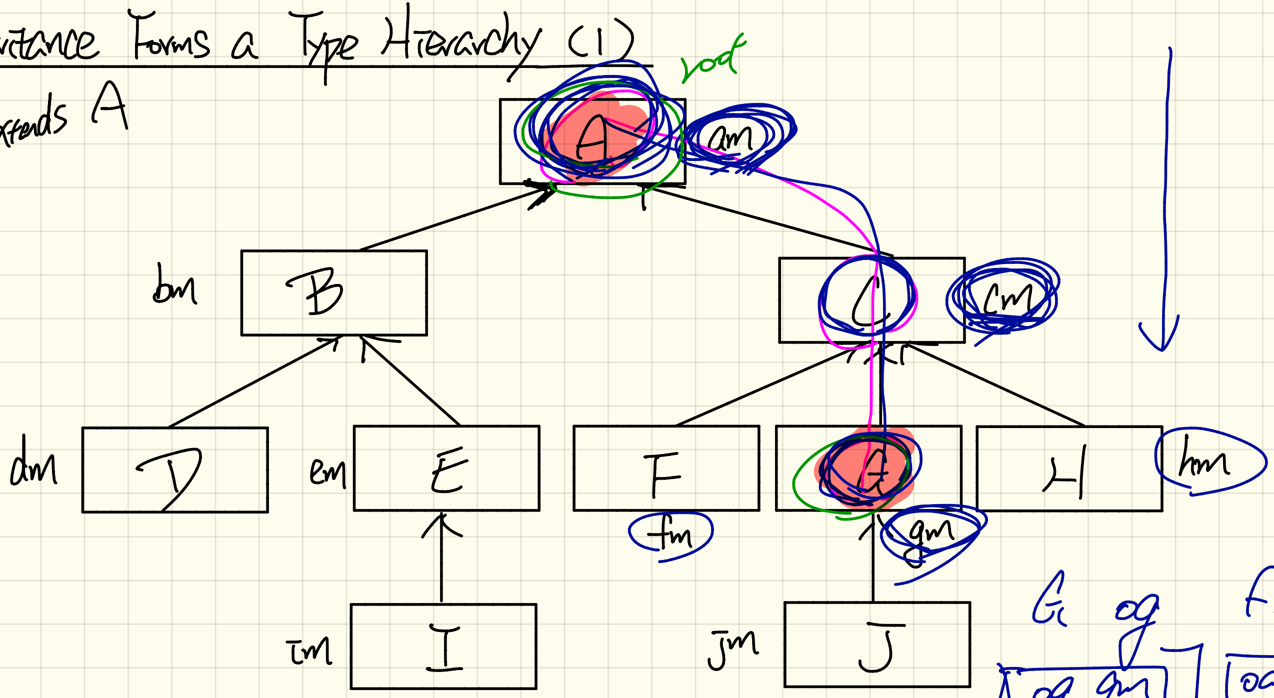
S.T.

```
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 */
```



# Inheritance Forms a Type Hierarchy (1)

B extends A



| | ancestors | expectations | descendants |
|---|---|---|---|
| A | A | am | all classes |
| C | C, A | cm, am | C, F, G, H, J |
| G | G, C, A | gm, cm, am | J, G |

G og

og.gm

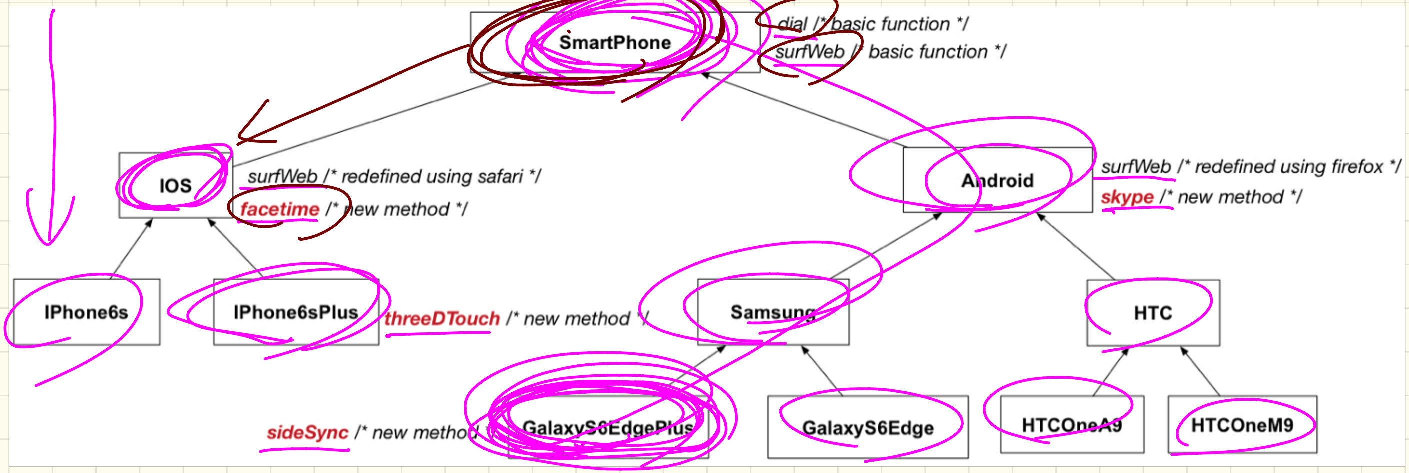
og.cm

og.am

A oa

oa.am

# Inheritance Forms a Type Hierarchy (2)



SmartPhone

dial /* basic function */

surfWeb /* basic function */

IOS

surfWeb /* redefined using safari */

facetime /* new method */

Android

surfWeb /* redefined using firefox */

skype /* new method */

IPhone6s

IPhone6sPlus

threeDTouch /* new method */

Samsung

HTC

sideSync /* new method */

GalaxyS6EdgePlus

GalaxyS6Edge

HTCOneA9

HTCOneM9

| | ancestors | expectations | descendants |
|---|---|---|---|
| SmartPhone | SP | | |
| Android | A, SP | | |
| GS6EP | GS6EP, S, A, SP | | |

# Substitutions ≈ Re-assignments

When considering compilation, only look at static types.

Rule: the ST of ~~RHS~~ RS is a descendant class of the ST of ~~LHS~~ Stud.

## Declarations

Student s;

ResidentStudent rs;

NonResidentStudent nrs;

## Substitutions

ST: Student

s = rs;

ST: RS

valid

s = nrs;

S

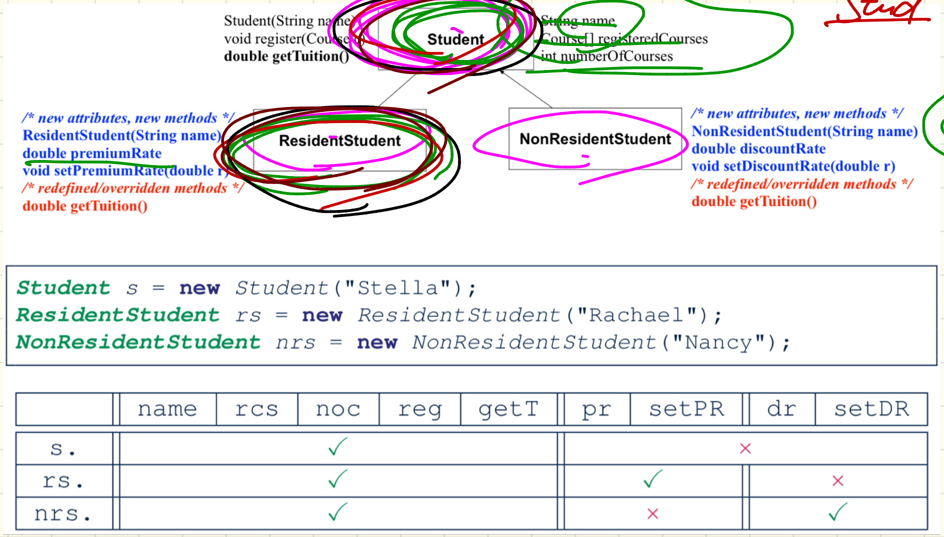
invalid

rs = s;

ST: S

RS

rs = nrs;



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. | ✓ | | | | | × | | ✓ | |

Student S = [ - —— ] ;

S. name ✓

S. pr ✗ ∵ ST of s (Student) doesn't declar pr.

a descendant class of

ResidentStuden s2 = [ ]

ST

S2. name

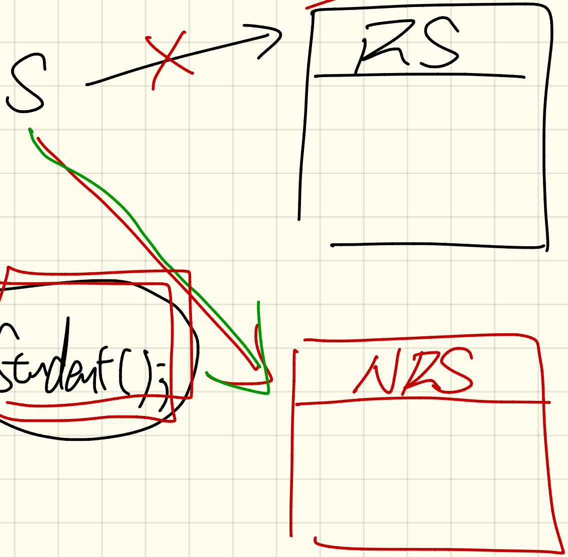
S. pr ✓

Student s = new ResidentStudent(- -);

→ ST: Student DT: RS

s = new NonResidentStudent(- -);

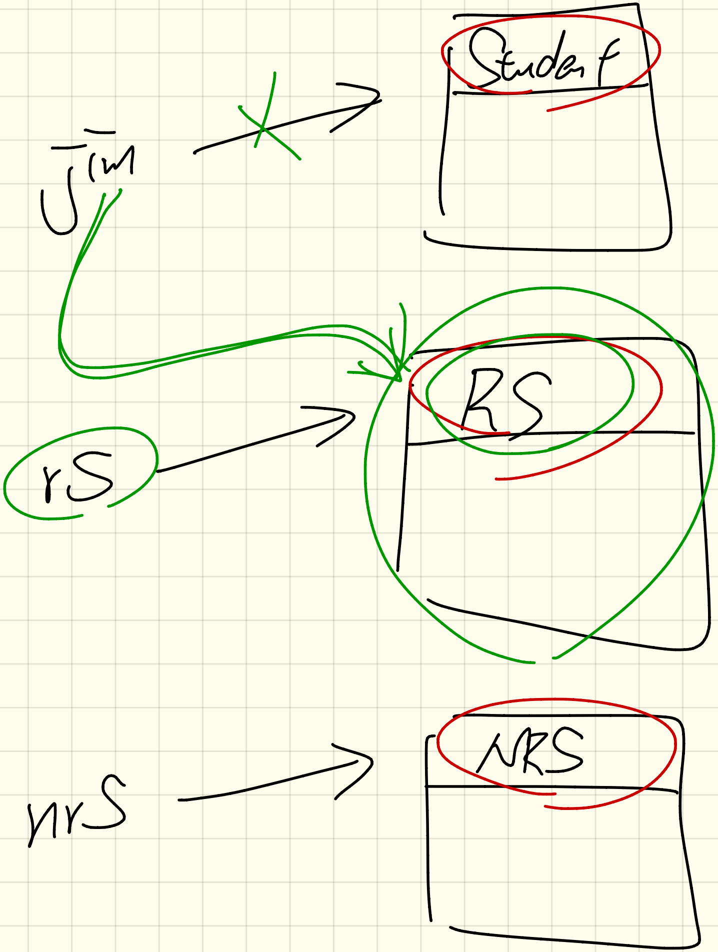
→ ST: Student DT: NRS



ResidentStudent s2 = ~~new~~ Student();

s2.pr

# DT:



ST: Student ST: RS

jim = rS; ✓

ST of jim : Student

DT of jim : RS

Wednesday Nov. 7

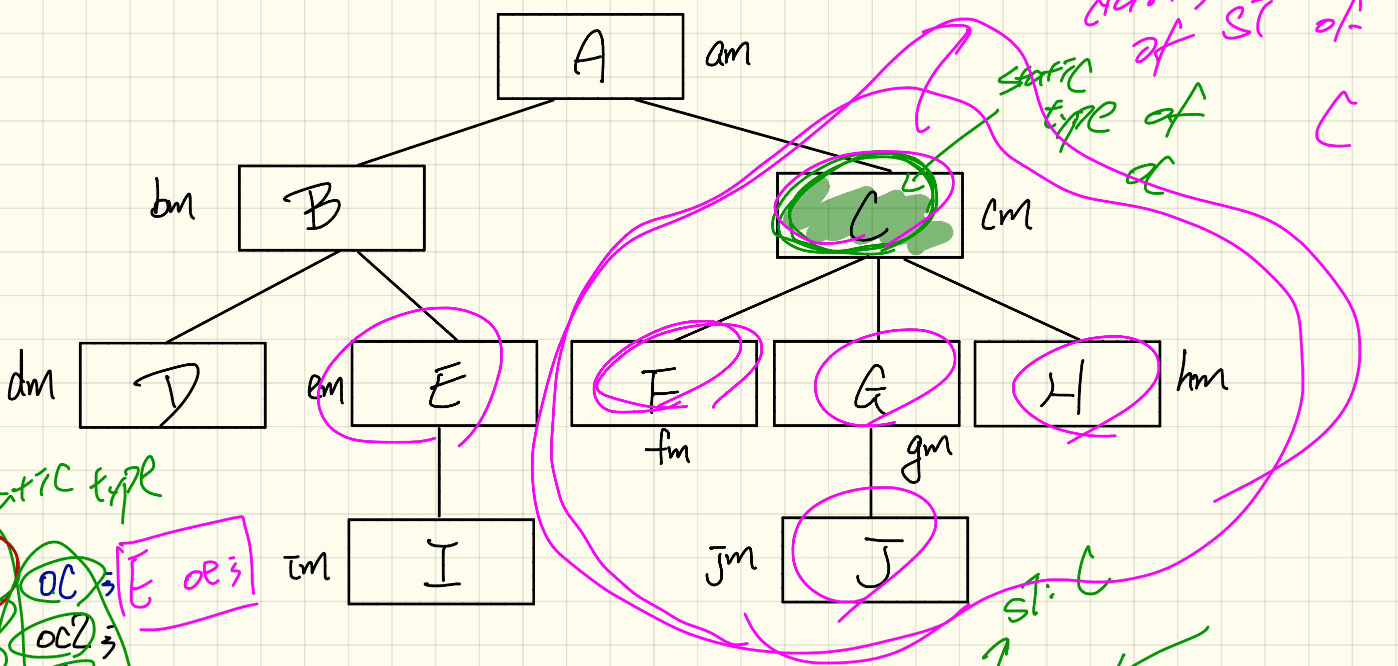
Lecture 17

# Static Type vs. Dynamic Type

- Does the code compile? static type

- How does the compilable code behave at runtime? dynamic type

# Rules of Substitutions



static type

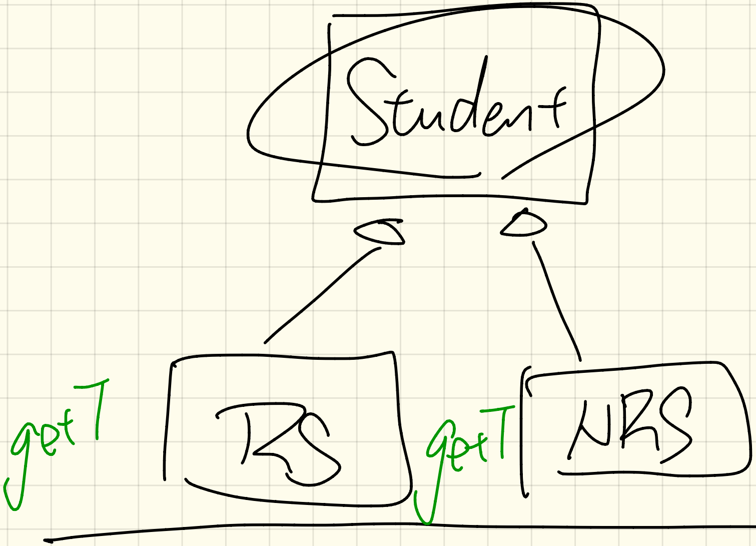
C oc; E oe;
C oc2;
A oa;
F of;
G og;
H oh;
J oj;

Safe to substitute oc with : oc = ?

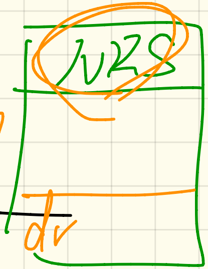
oc2
of

Unsafe to substitute oc with : oc = ?

C oc = oe
E X



```
Student s;
RS rs;
NRS nrs;
```



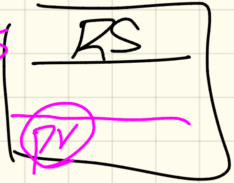
## Polymorphism

```
s = new RS(); ✓
s.getT();
s = new NRS();
s.getT();
```

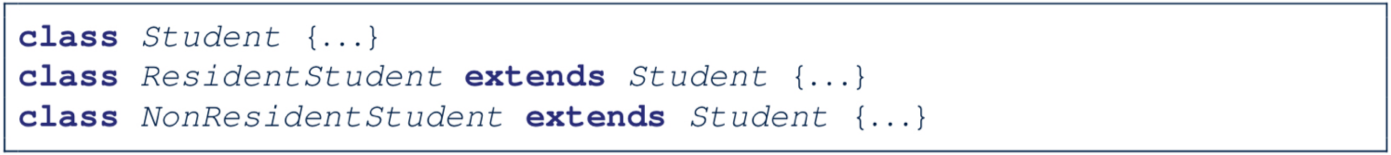
DT: RS

DT: NRS

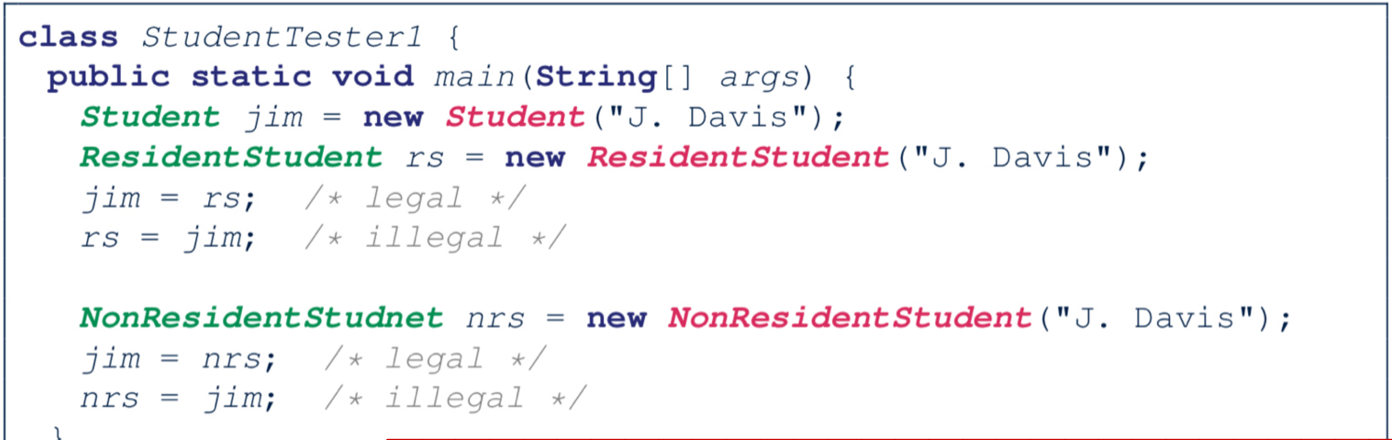
s



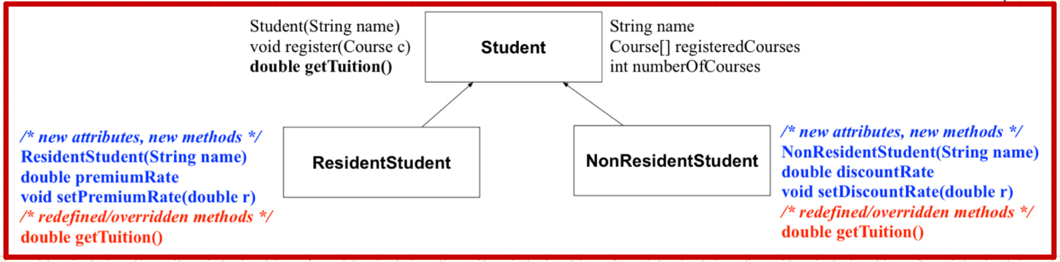
# Polymorphism (1)



```
class Student {...}
class ResidentStudent extends Student {...}
class NonResidentStudent extends Student {...}
```

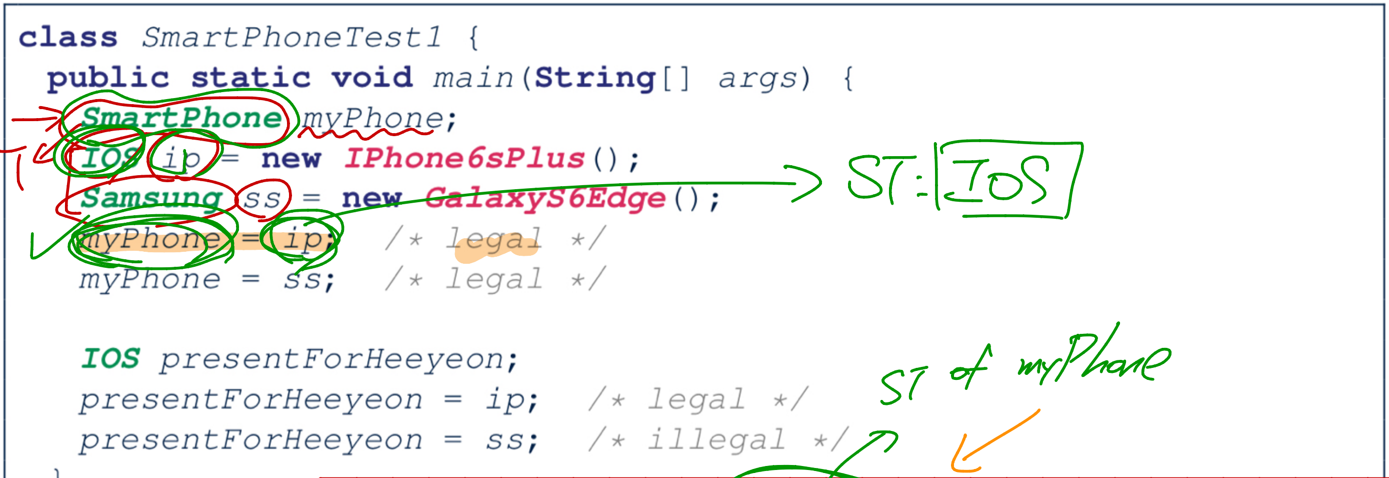


```
class StudentTester1 {
  public static void main(String[] args) {
    Student jim = new Student("J. Davis");
    ResidentStudent rs = new ResidentStudent("J. Davis");
    jim = rs;   /* legal */
    rs = jim;   /* illegal */

    NonResidentStudnet nrs = new NonResidentStudent("J. Davis");
    jim = nrs;   /* legal */
    nrs = jim;   /* illegal */
  }
}
```



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()

# Polymorphism (2)

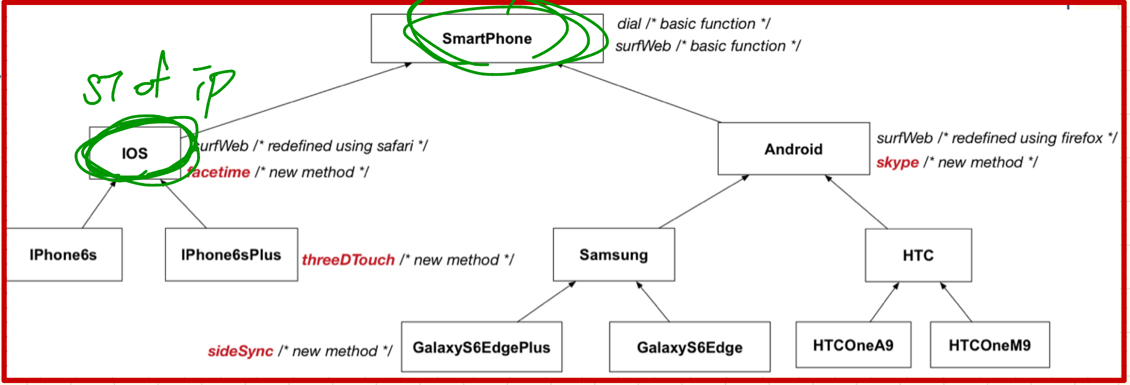


```
class SmartPhoneTest1 {
  public static void main(String[] args) {
    SmartPhone myPhone;
    IOS ip = new IPhone6sPlus();
    Samsung ss = new GalaxyS6Edge();
    myPhone = ip;   /* legal */
    myPhone = ss;   /* legal */

    IOS presentForHeeyeon;
    presentForHeeyeon = ip;   /* legal */
    presentForHeeyeon = ss;   /* illegal */
  }
}
```

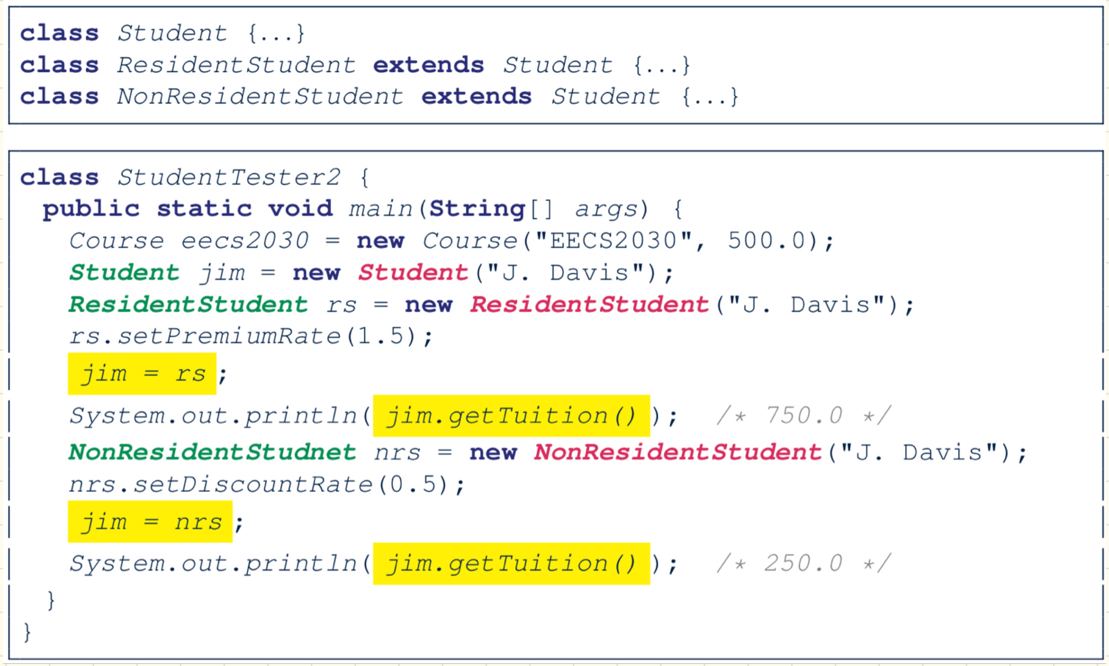
ST

ST: IOS

ST of myPhone

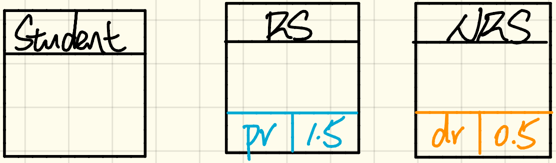

SmartPhone
dial /* basic function */
surfWeb /* basic function */
ST of ip
IOS
surfWeb /* redefined using safari */
facetime /* new method */
Android
surfWeb /* redefined using firefox */
skype /* new method */
IPhone6s
IPhone6sPlus
threeDTouch /* new method */
Samsung
HTC
sideSync /* new method */
GalaxyS6EdgePlus
GalaxyS6Edge
HTCOneA9
HTCOneM9

# Dynamic Binding (1)

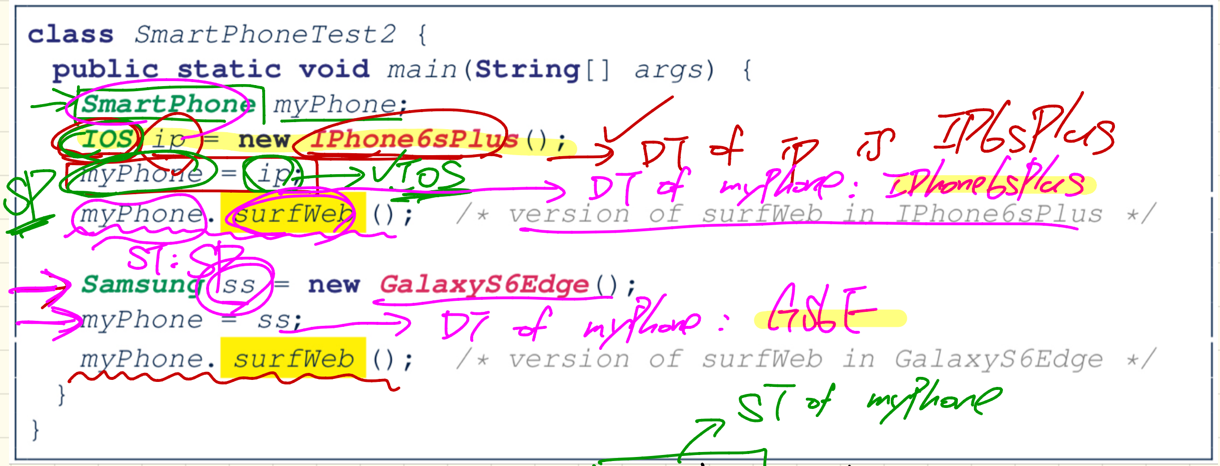


```
class Student {...}
class ResidentStudent extends Student {...}
class NonResidentStudent extends Student {...}
```

```
class StudentTester2 {
  public static void main(String[] args) {
    Course eecs2030 = new Course("EECS2030", 500.0);
    Student jim = new Student("J. Davis");
    ResidentStudent rs = new ResidentStudent("J. Davis");
    rs.setPremiumRate(1.5);
    jim = rs;
    System.out.println( jim.getTuition() );   /* 750.0 */
    NonResidentStudnet nrs = new NonResidentStudent("J. Davis");
    nrs.setDiscountRate(0.5);
    jim = nrs;
    System.out.println( jim.getTuition() );   /* 250.0 */
  }
}
```



# Dynamic Binding (2)



```
class SmartPhoneTest2 {
  public static void main(String[] args) {
    SmartPhone myPhone;
    IOS ip = new IPhone6sPlus();
    myPhone = ip;
    myPhone. surfWeb(); /* version of surfWeb in IPhone6sPlus */

    Samsung ss = new GalaxyS6Edge();
    myPhone = ss;
    myPhone. surfWeb(); /* version of surfWeb in GalaxyS6Edge */
  }
}
```

SP

✓ DT of ip is IP6sPlus

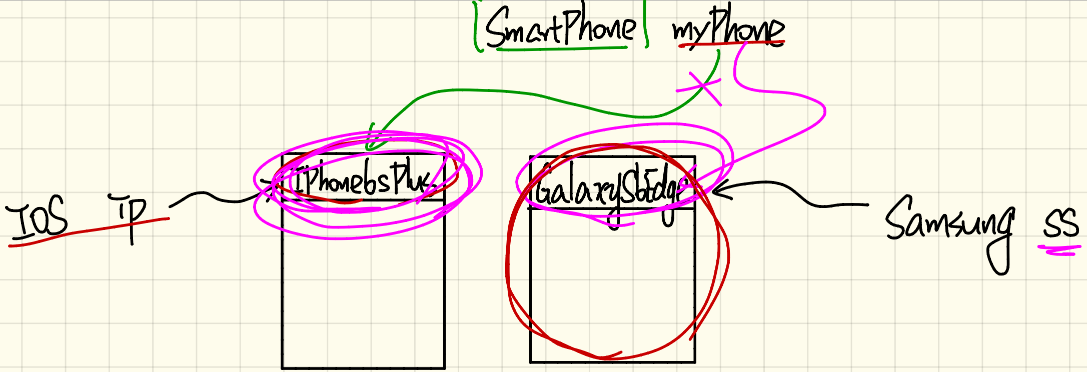
✓IOS

DT of myPhone: IPhone6sPlus

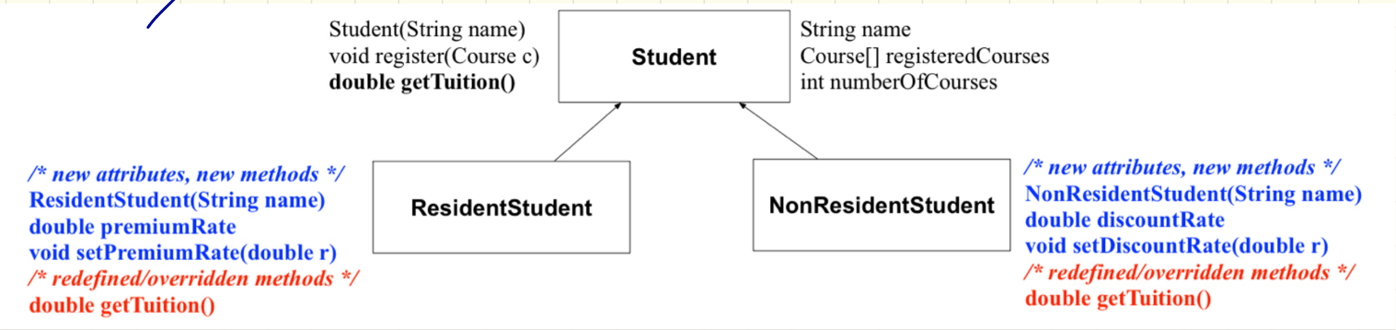
ST: SP

DT of myPhone: G6E

ST of myPhone



# Type Cast: Motivation



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

**Student**

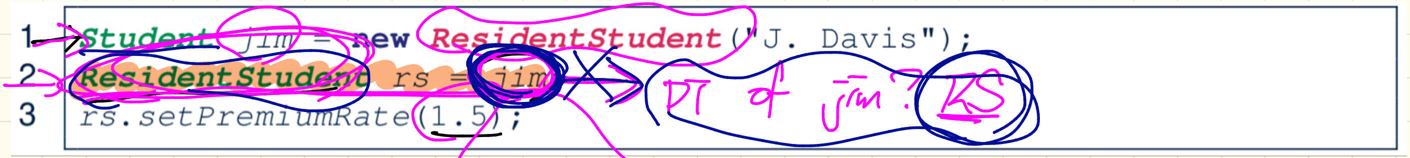
String name
Course[] registeredCourses
int numberOfCourses

**ResidentStudent**

**NonResidentStudent**

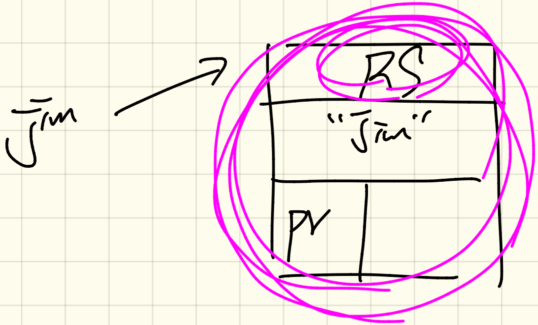
***/* new attributes, new methods */***
**ResidentStudent(String name)**
**double premiumRate**
**void setPremiumRate(double r)**
***/* redefined/overridden methods */***
**double getTuition()**

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



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

DT of jim? RS



At this point, jim's DT is really a RS, but Java compiler would not allow us to assign "jim" to a RS.

```
Student s = new RS(..);
RS rs = s; ✗
```

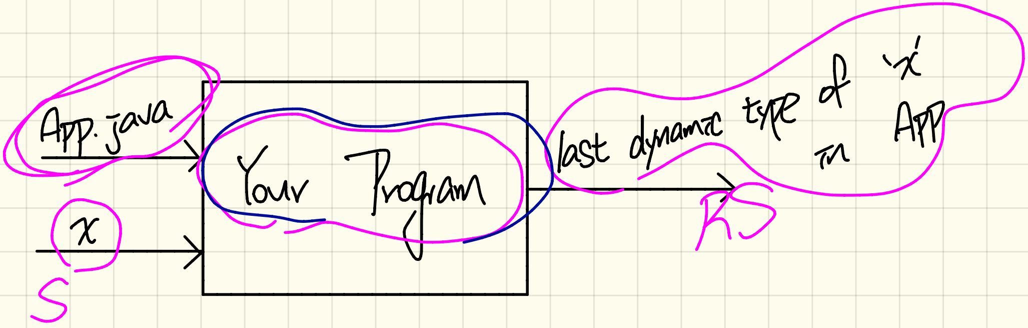
```
RS rs = (RS) s;
```

ST: Student

temporarily change the ST to RS

✓

# Keeping Track of Dynamic Types: Undecidable



e.g.

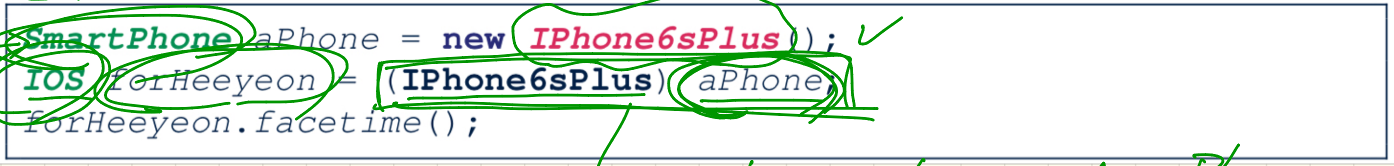
App.java

```
Student s;
   ⋮
s = new RS(...);
```

```
Student s;
while (true) {

}
s = new RS(..);
```

# Type Cast: Named or Anonymous

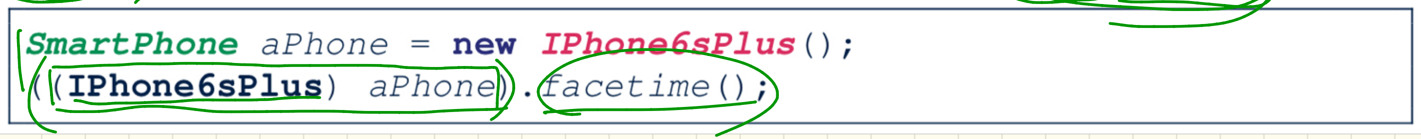
## Named Cast



```
SmartPhone aPhone = new IPhone6sPlus(); ✓
IOS forHeeyeon = (IPhone6sPlus) aPhone;
forHeeyeon.facetime();
```

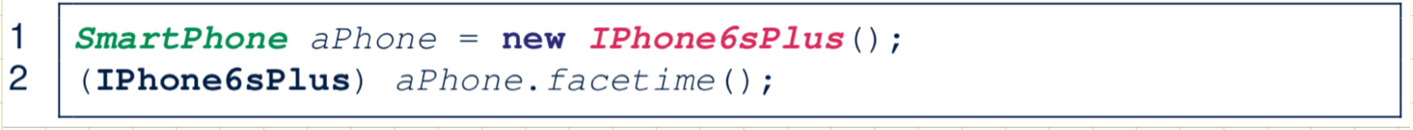
change the ST of aPhone to IP6sPlus

## Anonymous Cast



```
SmartPhone aPhone = new IPhone6sPlus();
((IPhone6sPlus) aPhone).facetime();
```

## Problem?

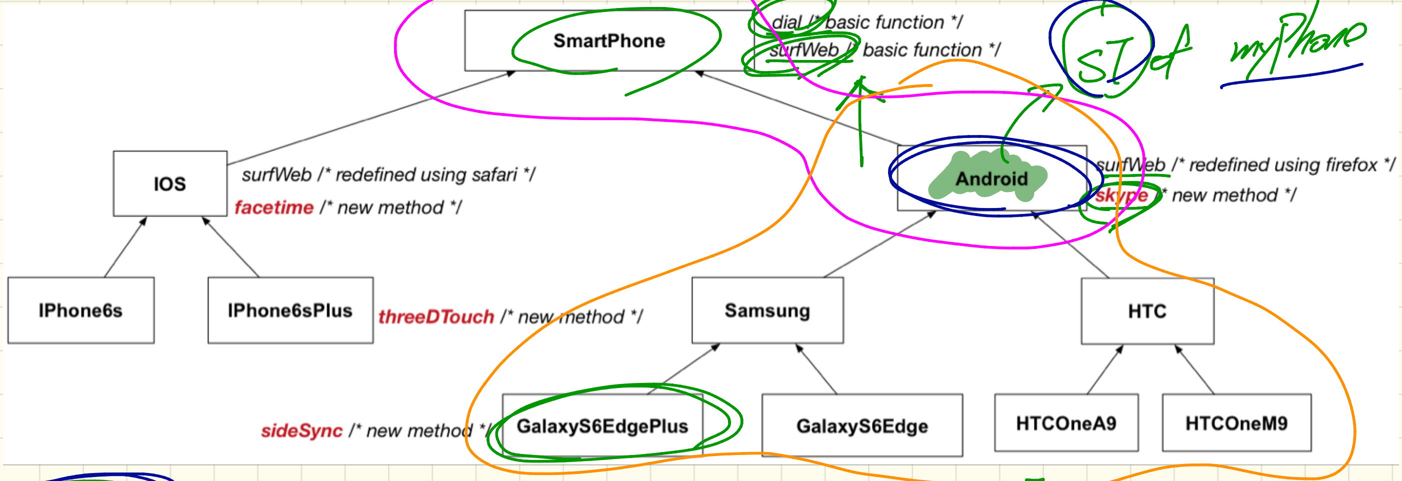


```
SmartPhone aPhone = new IPhone6sPlus();
(IPhone6sPlus) aPhone.facetime();
```

# Compilable Cast: Upward vs. Downward

expectations

ST of myPhone


SmartPhone
dial /* basic function */
surfWeb /* basic function */
IOS
surfWeb /* redefined using safari */
facetime /* new method */
Android
surfWeb /* redefined using firefox */
skype /* new method */
IPhone6s
IPhone6sPlus
threeDTouch /* new method */
Samsung
HTC
sideSync /* new method */
GalaxyS6EdgePlus
GalaxyS6Edge
HTCOneA9
HTCOneM9

```
Android myPhone = new GalaxyS6EdgePlus();
SmartPhone sp = (SmartPhone) myPhone;
GalaxyS6EdgePlus ga = (GalaxyS6EdgePlus) myPhone;
```

EXPECTATIONS

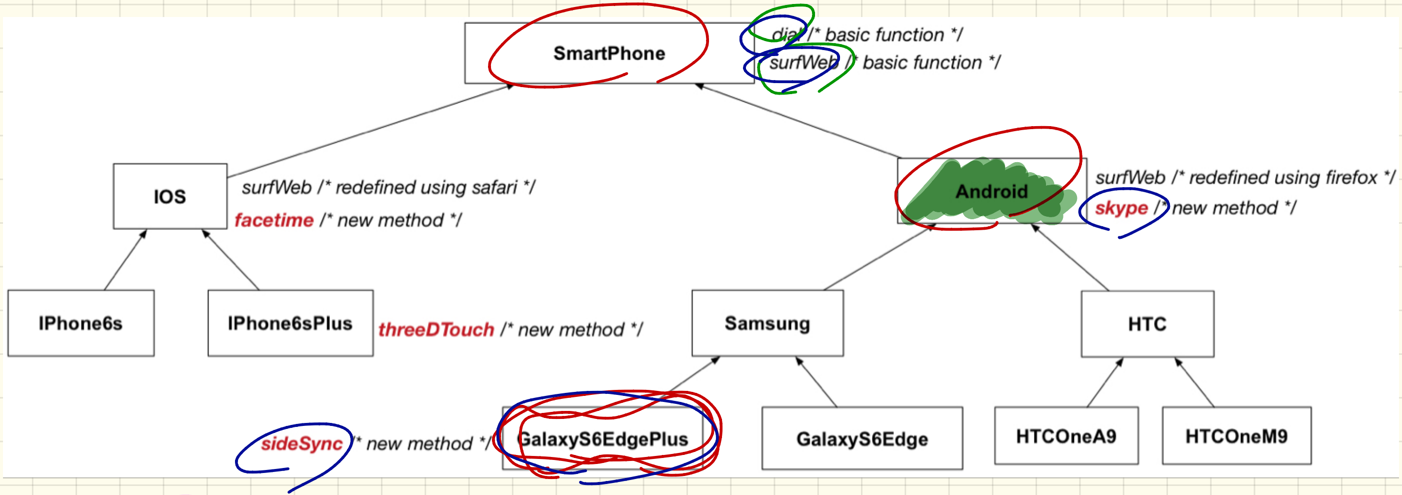
myPhone : [ skype<br>
surfweb
dial

sp : dial
surfWeb

ga : dial, surfweb, skype, sideSy

Upward casting

Downward casting



SmartPhone

dial /* basic function */

surfWeb /* basic function */

IOS

surfWeb /* redefined using safari */

facetime /* new method */

Android

surfWeb /* redefined using firefox */

skype /* new method */

IPhone6s

IPhone6sPlus

threeDTouch /* new method */

Samsung

HTC

sideSync /* new method */

GalaxyS6EdgePlus

GalaxyS6Edge

HTCOneA9

HTCOneM9

Android myPhone = . . ;

myPhone . dial

surfweb

skype

Upward casting

((SmartPhone) myPhone) . dial

surfweb

SmartPhone

Downward Casting

((GS6EP) myPhone) . dial

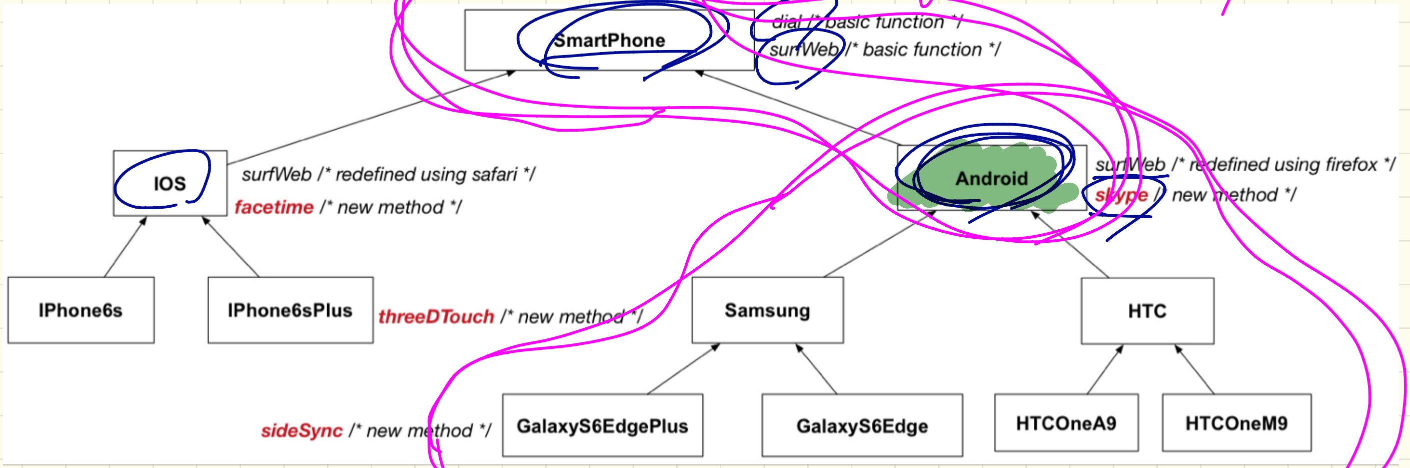
surfweb

skype

sidesync

ST: GS6EP

upward casting (narrow expectation)


SmartPhone
dial /* basic function */
surfWeb /* basic function */
IOS
surfWeb /* redefined using safari */
facetime /* new method */
Android
surfWeb /* redefined using firefox */
skype /* new method */
IPhone6s
IPhone6sPlus
threeDTouch /* new method */
Samsung
HTC
sideSync /* new method */
GalaxyS6EdgePlus
GalaxyS6Edge
HTCOneA9
HTCOneM9

downward casting (widen expectation)

$\boxed{\text{Android}}$ p =

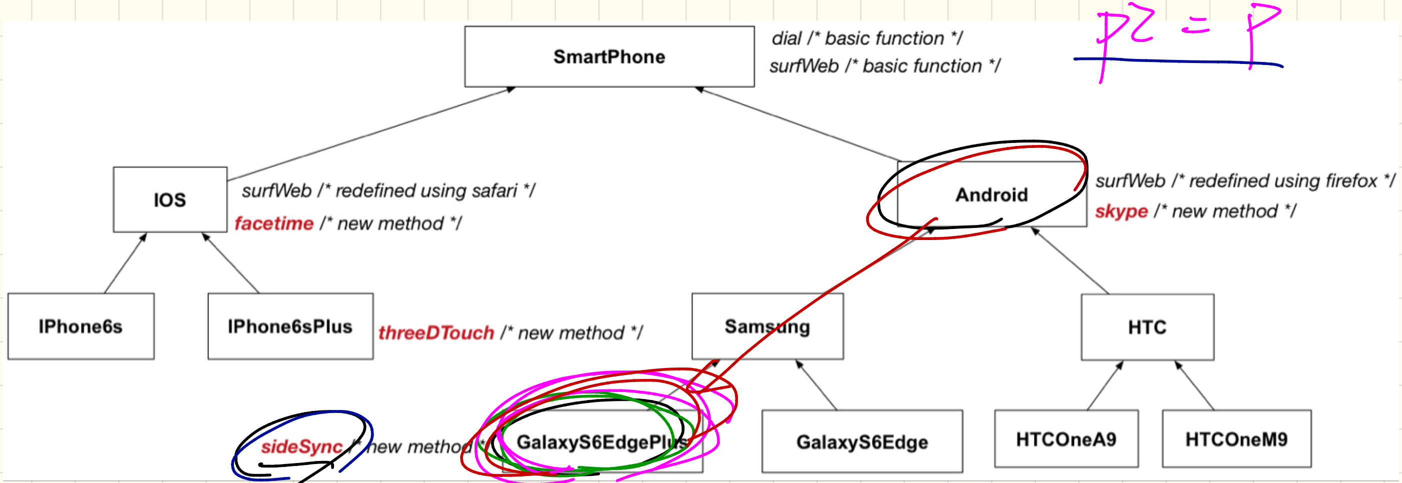
ST

p.skype
p.surfweb
p.dial

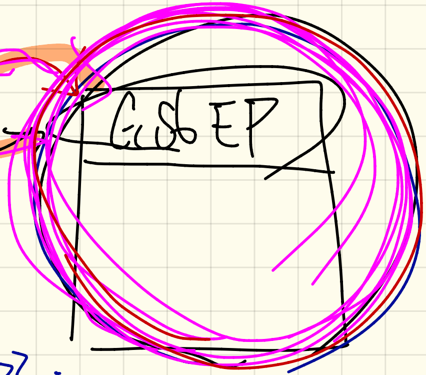
What kind of cast will compile ?

↓ temporarily changes the ST.

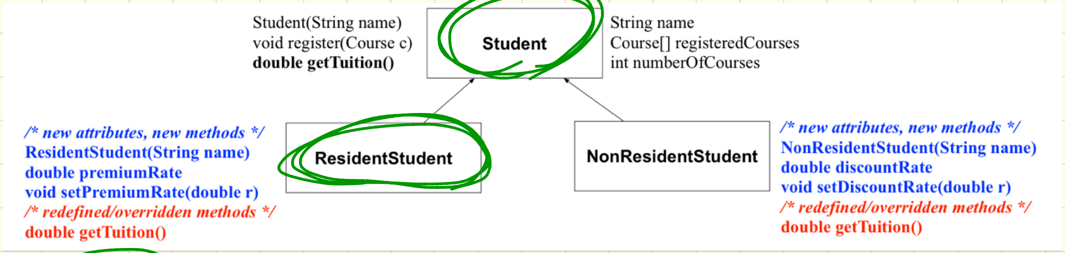
p2 = p


SmartPhone
dial /* basic function */
surfWeb /* basic function */
IOS
surfWeb /* redefined using safari */
facetime /* new method */
Android
surfWeb /* redefined using firefox */
skype /* new method */
IPhone6s
IPhone6sPlus
threeDTouch /* new method */
Samsung
HTC
sideSync /* new method */
GalaxyS6EdgePlus
GalaxyS6Edge
HTCOneA9
HTCOneM9

```
Android p = new GS6EP(..);
→ p.skype
  p.surfweb
  p.dial
  p.sideSync ✗
  p.sideSync

GS6EP p2 =
(GS6EP) p2;
p2.sideSync

GS6EP p2
Android p

Android p3 = (Android) p2;
```



# Compilable Cast May Fail at Runtime (1)

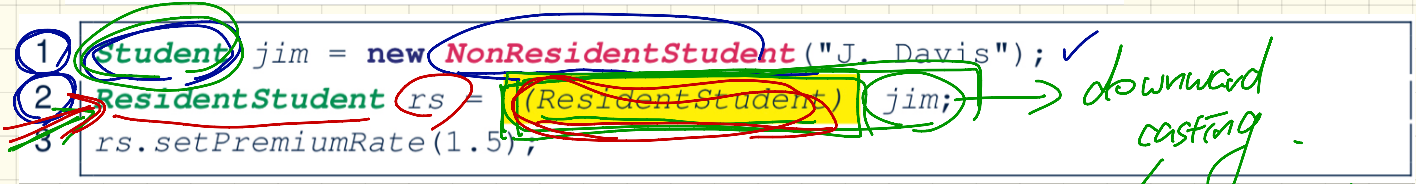


Student(String name)
void register(Course c)
**double getTuition()**

String name
Course[] registeredCourses
int numberOfCourses

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

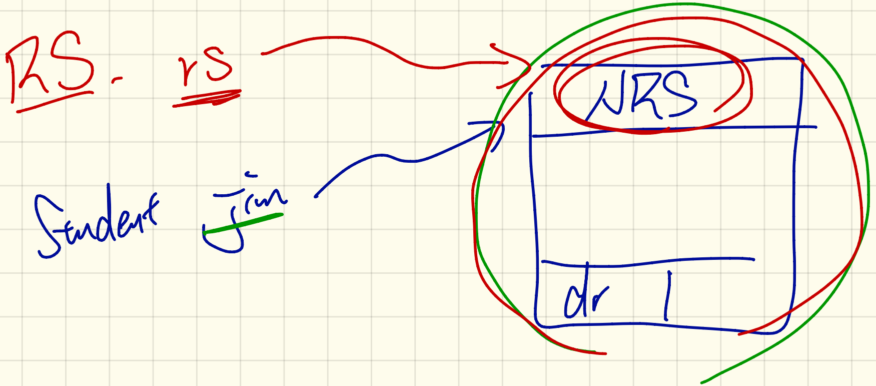
*/* new attributes, new methods */*
**NonResidentStudent(String name)**
**double discountRate**
**void setDiscountRate(double r)**
*/* redefined/overridden methods */*
**double getTuition()**



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

✓

downward casting.
↳ compile!



rs.pr

ClassCastException.

Monday Nov. 12

Lecture 18

- Lab Test 3: Nov. 19
- Guide & Exercises

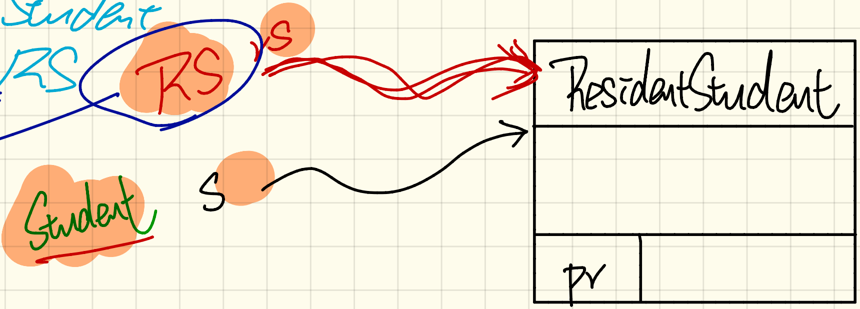
# Anatomy of a Type Cast

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

→ ST : Student

PT : RS

descendant



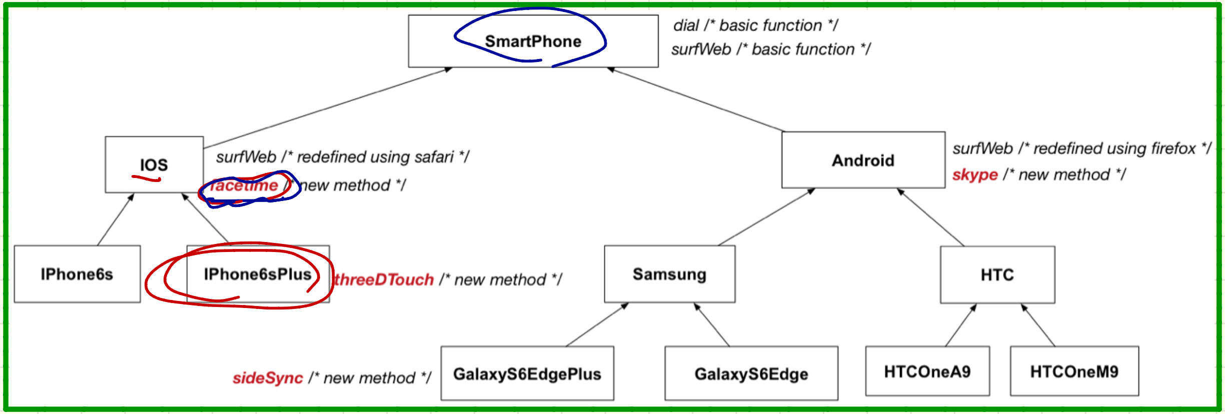
```
ResidentStudent rs = (ResidentStudent) s;
```

ST

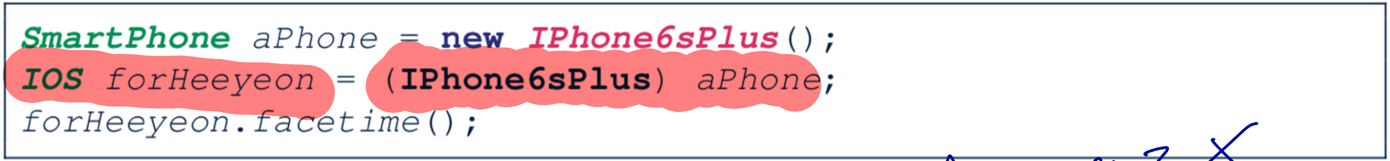
ST: Student

a new ref of ST RS

# Type Casts


SmartPhone
dial /* basic function */
surfWeb /* basic function */
IOS
surfWeb /* redefined using safari */
facetime /* new method */
Android
surfWeb /* redefined using firefox */
skype /* new method */
IPhone6s
IPhone6sPlus
threeDTouch /* new method */
Samsung
HTC
sideSync /* new method */
GalaxyS6EdgePlus
GalaxyS6Edge
HTCOneA9
HTCOneM9

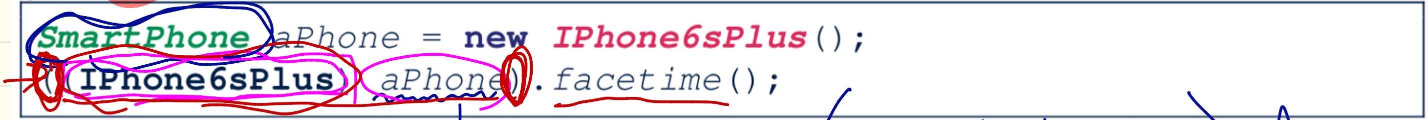
## Named Cast



```
SmartPhone aPhone = new IPhone6sPlus();
IOS forHeeyeon = (IPhone6sPlus) aPhone;
forHeeyeon.facetime();
```

aPhone.facetime() ? ✗

## Anonymous Cast



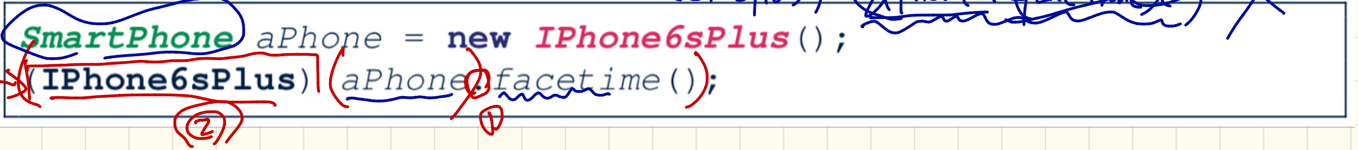
```
SmartPhone aPhone = new IPhone6sPlus();
((IPhone6sPlus) aPhone).facetime();
```

↓ SP

((IP6sPlus) aPhone).facetime() ✓

## Problem?

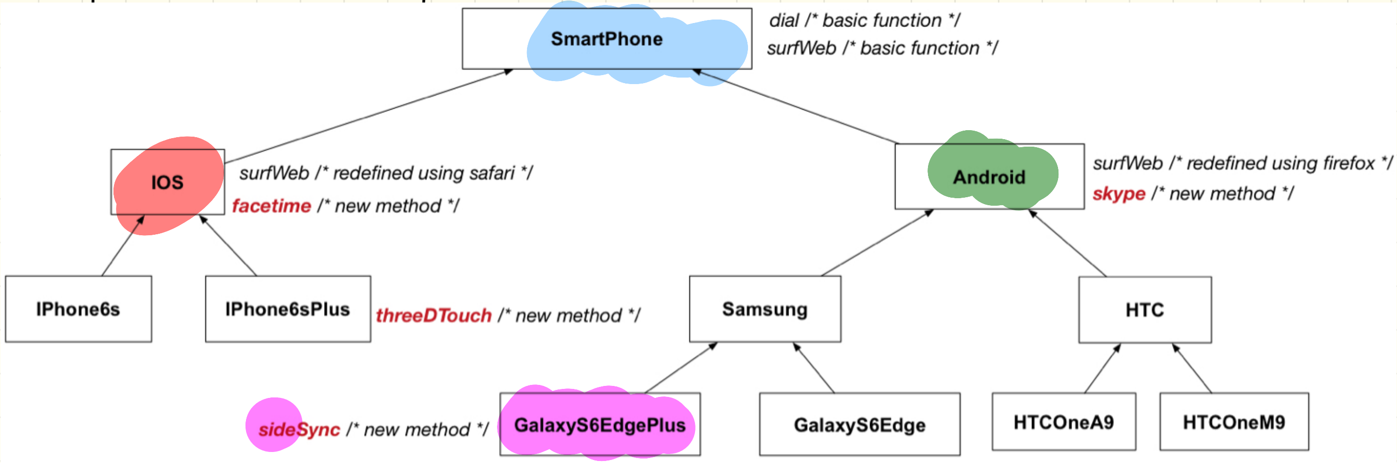
→ (IP6sPlus) (aPhone.facetime()) ✗



```
SmartPhone aPhone = new IPhone6sPlus();
(IPhone6sPlus) aPhone.facetime();
```

① ②

# Compilable Cast : Upward vs. Downward


SmartPhone
dial /* basic function */
surfWeb /* basic function */
IOS
surfWeb /* redefined using safari */
facetime /* new method */
Android
surfWeb /* redefined using firefox */
skype /* new method */
IPhone6s
IPhone6sPlus
threeDTouch /* new method */
Samsung
HTC
sideSync /* new method */
GalaxyS6EdgePlus
GalaxyS6Edge
HTCOneA9
HTCOneM9

```
Android myPhone = new GalaxyS6EdgePlus();
SmartPhone sp = (SmartPhone) myPhone;
GalaxyS6EdgePlus ga = (GalaxyS6EdgePlus) myPhone;
```

EXPECTATIONS

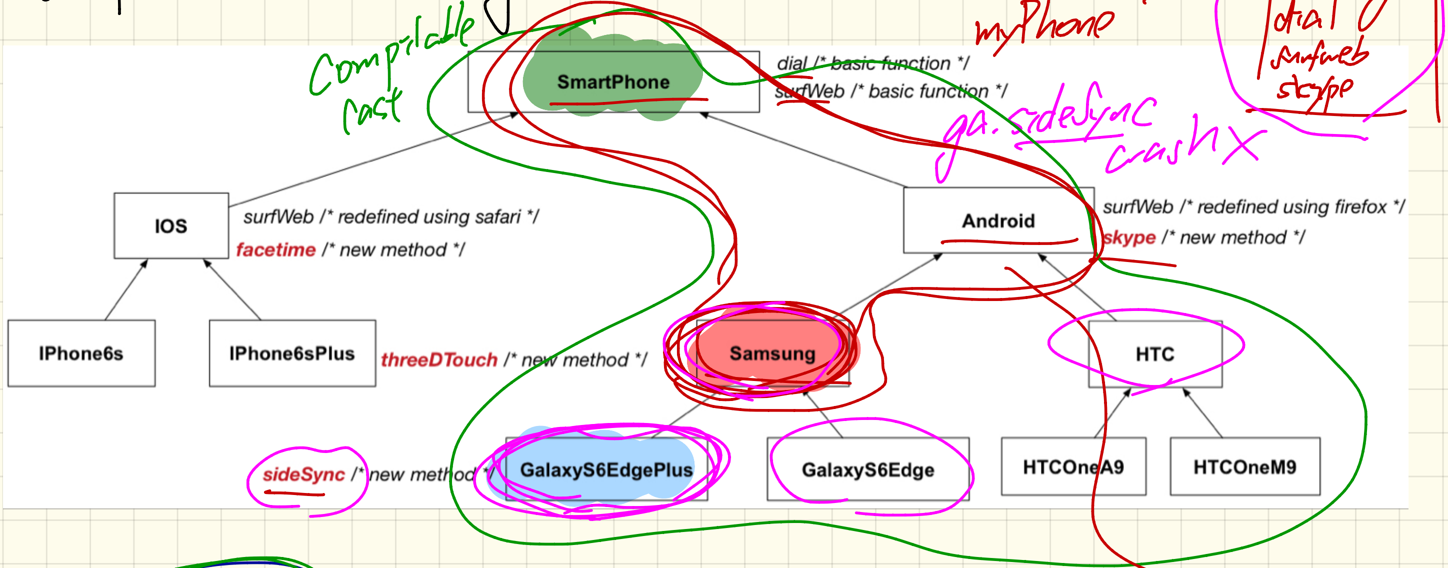
| | myPhone | sp | ga |
|---|---|---|---|
| dial | 🟩 | 🟩 | 🟩 |
| surfWeb | 🟩 | 🟩 | 🟩 |
| facetime | 🟥 | 🟥 | 🟥 |
| threeDTouch | 🟥 | 🟥 | 🟥 |
| skype | 🟩 | 🟥 | 🟩 |
| sideSync | 🟥 | 🟥 | 🟩 |

# Compilable Cast May Fail at Runtime (2)

GSLEP ga

myPhone

| Samsung |
|---|
| dial |
| surfWeb |
| skype |



SmartPhone
dial /* basic function */
surfWeb /* basic function */
IOS
surfWeb /* redefined using safari */
facetime /* new method */
Android
surfWeb /* redefined using firefox */
skype /* new method */
IPhone6s
IPhone6sPlus
threeDTouch /* new method */
Samsung
HTC
sideSync /* new method */
GalaxyS6EdgePlus
GalaxyS6Edge
HTCOneA9
HTCOneM9
Compilable cast
ga.sideSync
crash X

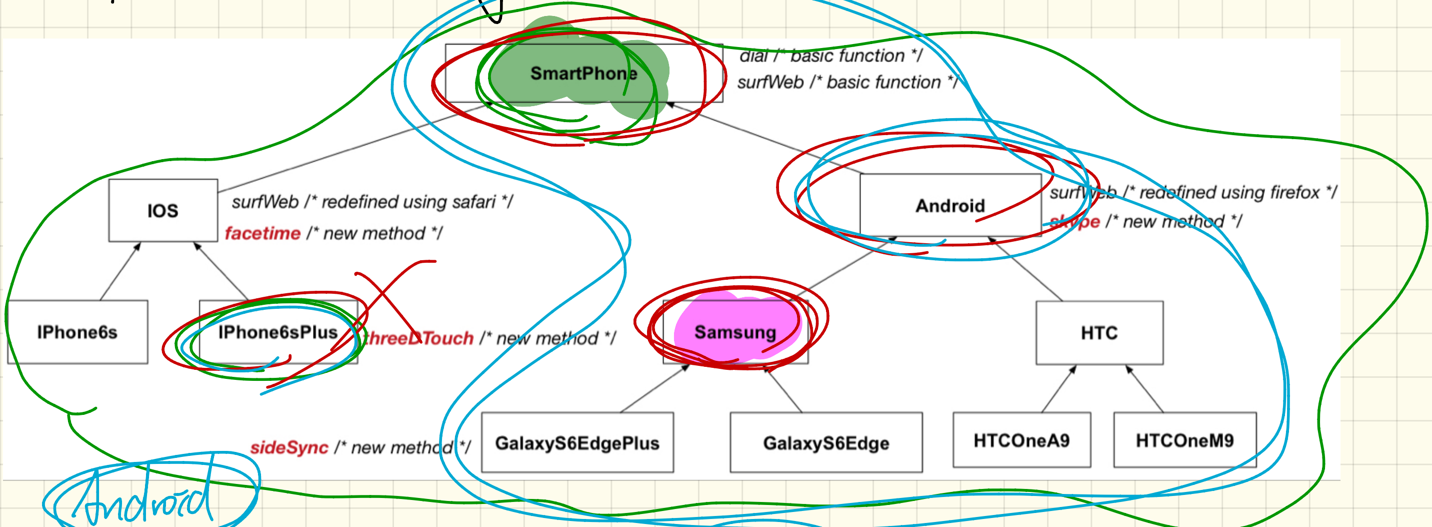
```
SmartPhone myPhone = new Samsung();
GalaxyS6EdgePlus ga = (GalaxyS6EdgePlus) myPhone;
```

compilable and no CCE

ST: SP

→ assume no ClassCastException
↳ invalid if there's a CCE.

# Compilable Cast May Fail at Runtime (3)

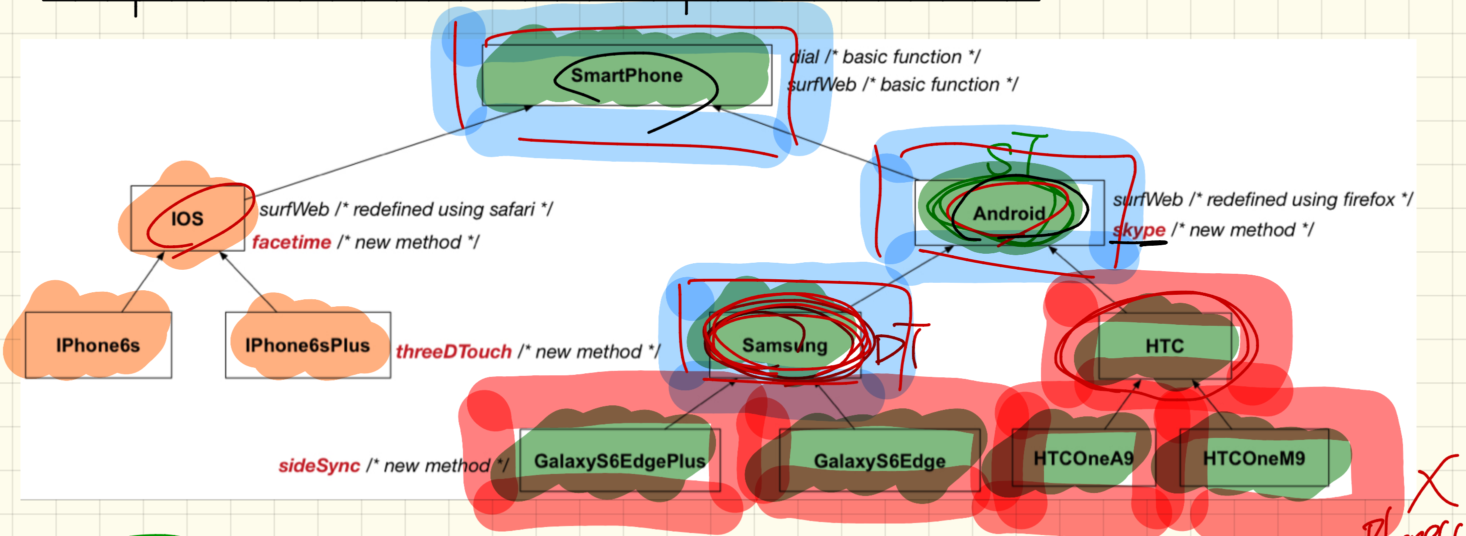

SmartPhone
dial /* basic function */
surfWeb /* basic function */
IOS
surfWeb /* redefined using safari */
facetime /* new method */
Android
surfWeb /* redefined using firefox */
skype /* new method */
IPhone6s
IPhone6sPlus
threeDTouch /* new method */
Samsung
HTC
sideSync /* new method */
GalaxyS6EdgePlus
GalaxyS6Edge
HTCOneA9
HTCOneM9

```
Android
~~SmartPhone~~ myPhone = new Samsung();
IPhone6sPlus ip = (IPhone6sPlus) myPhone;
```

compiles but

CCE

# Compilable Cast vs. Exception-Free Cast


SmartPhone
dial /* basic function */
surfWeb /* basic function */
IOS
surfWeb /* redefined using safari */
facetime /* new method */
Android
ST
surfWeb /* redefined using firefox */
skype /* new method */
IPhone6s
IPhone6sPlus
threeDTouch /* new method */
Samsung
DT
HTC
sideSync /* new method */
GalaxyS6EdgePlus
GalaxyS6Edge
HTCOneA9
HTCOneM9

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

```
IOS ios = (IOS) myPhone;
```

X

Compilable Cast 🟩

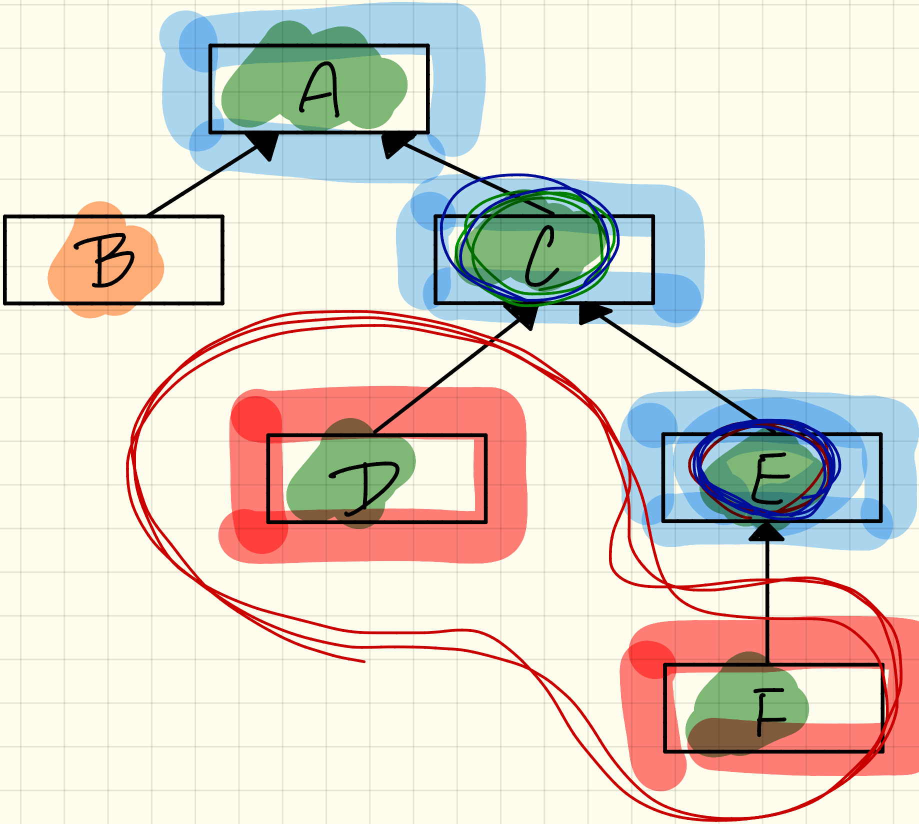
Non-Compilable Cast 🟧

Exception-Free Cast 🟦

ClassCastException Cast 🟥

# Type Casts

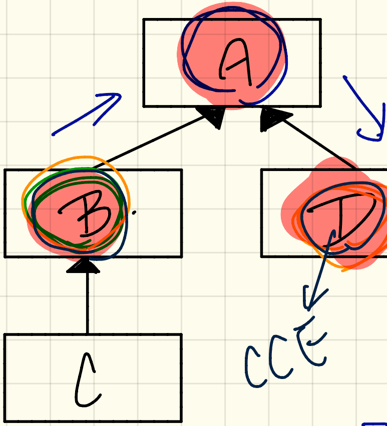
```
C   oc  =  new E();

A   oa  =  (A) oc;
E   oe  =  (E) oc;
F   of  =  (F) oc;
D   od  =  (D) oc;
B   ob  =  (B) oc;
```



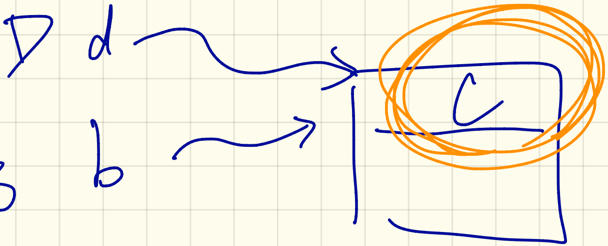
# Compilable Cast vs. Exception-Free Cast : Exercise

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

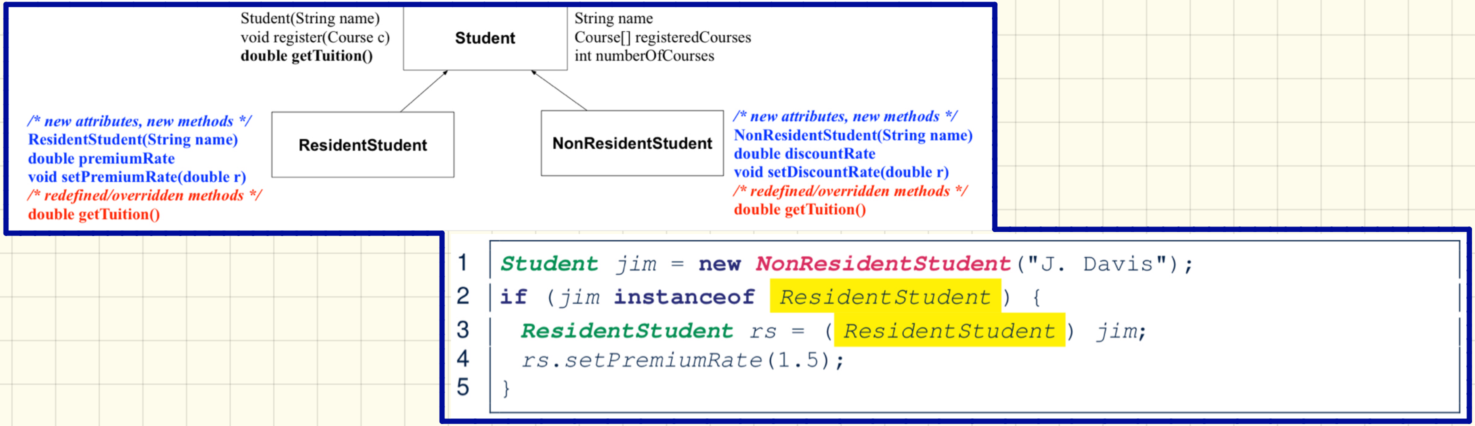
```
① B b = new C();
② D d = (D) b;    ST: B   ×              ST: D
```



$$d = (D)\,(\underbrace{\overbrace{(A)\ b}^{ST:\ A}}_{ST:\ B})$$ ✓



# Checking Dynamic Types at Runtime



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

Student

String name
Course[] registeredCourses
int numberOfCourses

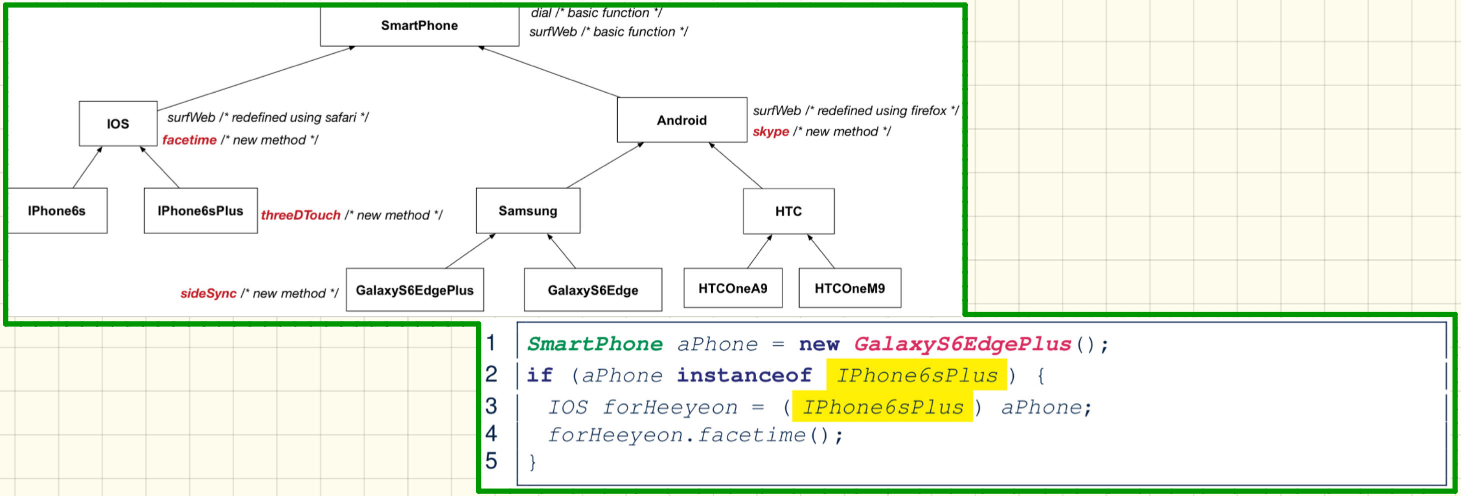
*/* new attributes, new methods */*
**ResidentStudent(String name)**
**double premiumRate**
**void setPremiumRate(double r)**
***/* redefined/overridden methods */***
**double getTuition()**

ResidentStudent

NonResidentStudent

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

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



SmartPhone

dial /* basic function */
surfWeb /* basic function */

IOS

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

Android

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

IPhone6s

IPhone6sPlus

**threeDTouch** /* new method */

Samsung

HTC

**sideSync** /* new method */

GalaxyS6EdgePlus

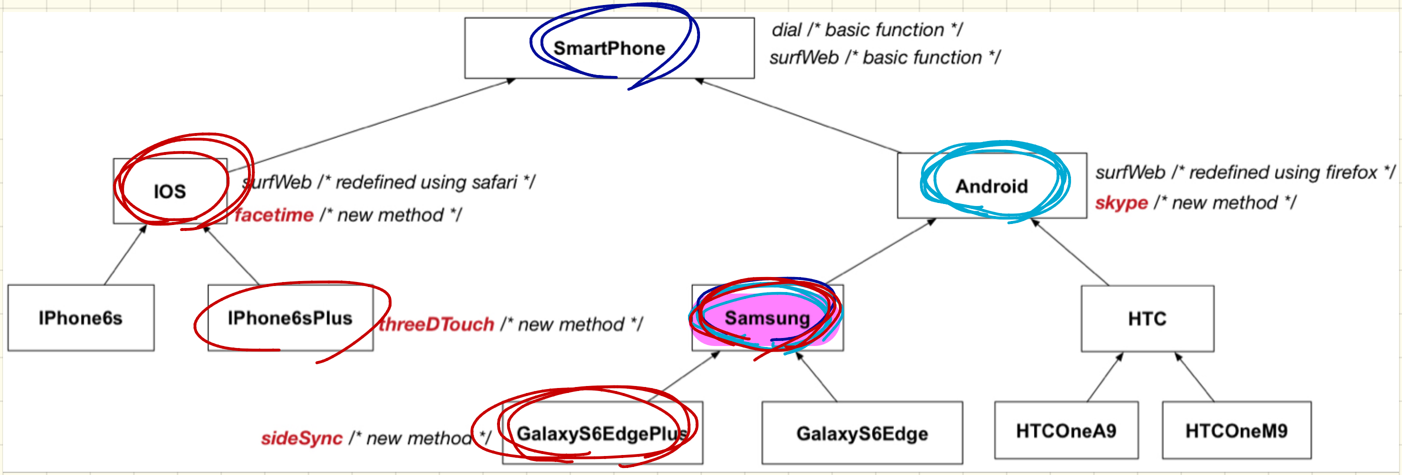
GalaxyS6Edge

HTCOneA9

HTCOneM9

```java
SmartPhone aPhone = new GalaxyS6EdgePlus();
if (aPhone instanceof IPhone6sPlus ) {
  IOS forHeeyeon = ( IPhone6sPlus ) aPhone;
  forHeeyeon.facetime();
}
```

# Use of the instanceof Operator



SmartPhone

dial /* basic function */
surfWeb /* basic function */

IOS

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

Android

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

IPhone6s

IPhone6sPlus

threeDTouch /* new method */

Samsung

HTC

sideSync /* new method */

GalaxyS6EdgePlus

GalaxyS6Edge

HTCOneA9

HTCOneM9

```
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 GalaxyS6Edge);
/* false ∵ Samsung is not a descendant of GalaxyS6Edge */
println(myPhone instanceof IOS);
/* false ∵ Samsung is not a descendant of IOS */
println(myPhone instanceof IPhone6sPlus);
/* false ∵ Samsung is not a descendant of IPhone6sPlus */
```

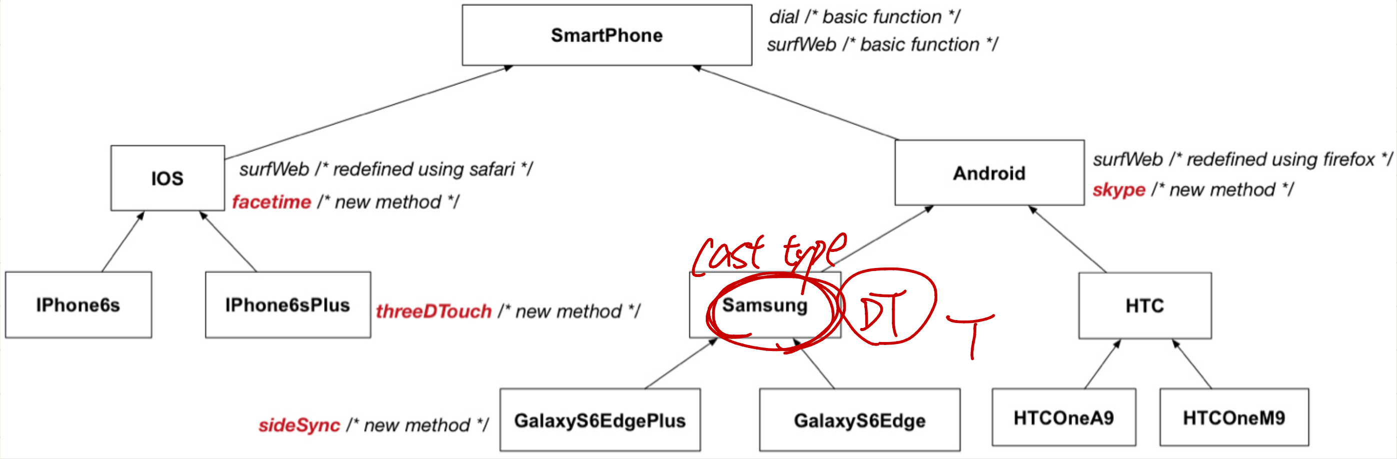
myPhone instanceof SmartPhone

True

⇒ GS6Edge o = (GS6Edge) myPhone;

CCE

# Safe Cast via Use of instanceof



SmartPhone

dial /* basic function */
surfWeb /* basic function */

IOS

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

Android

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

IPhone6s

IPhone6sPlus

**threeDTouch** /* new method */

Samsung

HTC

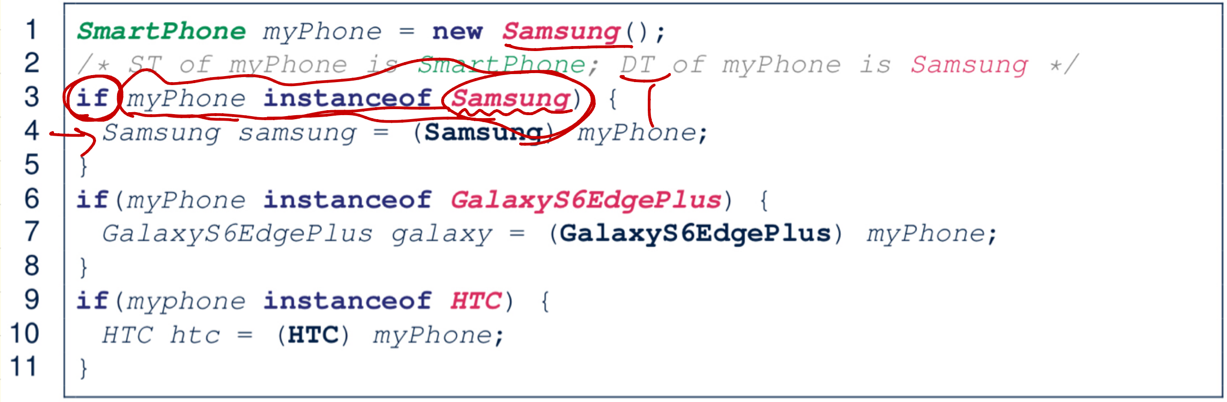
**sideSync** /* new method */

GalaxyS6EdgePlus

GalaxyS6Edge

HTCOneA9

HTCOneM9

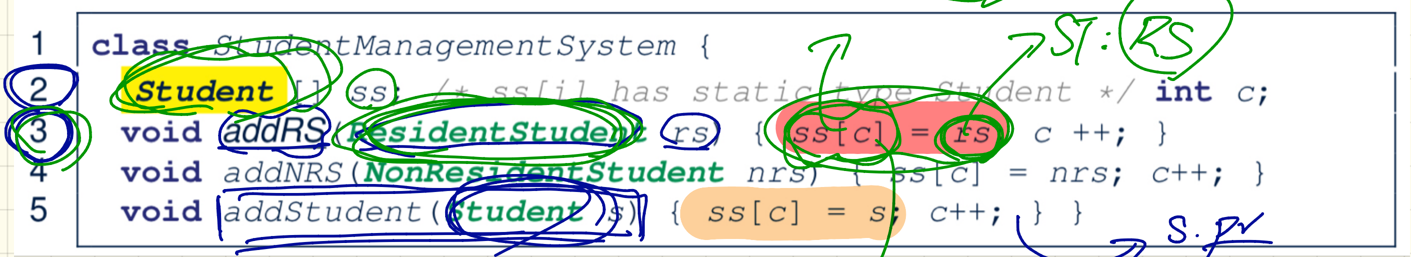


```
SmartPhone myPhone = new Samsung();
/* ST of myPhone is SmartPhone; DT of myPhone is Samsung */
if(myPhone instanceof Samsung) {
  Samsung samsung = (Samsung) myPhone;
}
if(myPhone instanceof GalaxyS6EdgePlus) {
  GalaxyS6EdgePlus galaxy = (GalaxyS6EdgePlus) myPhone;
}
if(myphone instanceof HTC) {
  HTC htc = (HTC) myPhone;
}
```

# Polymorphic Arguments (1)

ST: 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++; } }
```

s.pr

Q. Static type of ss[0], ss[1], ..., ss[ss.length -1] ?

Q. In addRS : does ss[C] = rs compile ?

addRS(RS rs){
  rs.pr
  rs.set_pr
}

ss[0].name

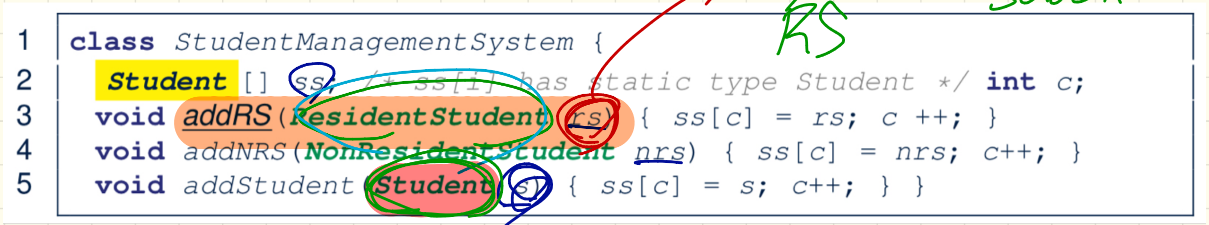
ss[0].pr X

ST: Student

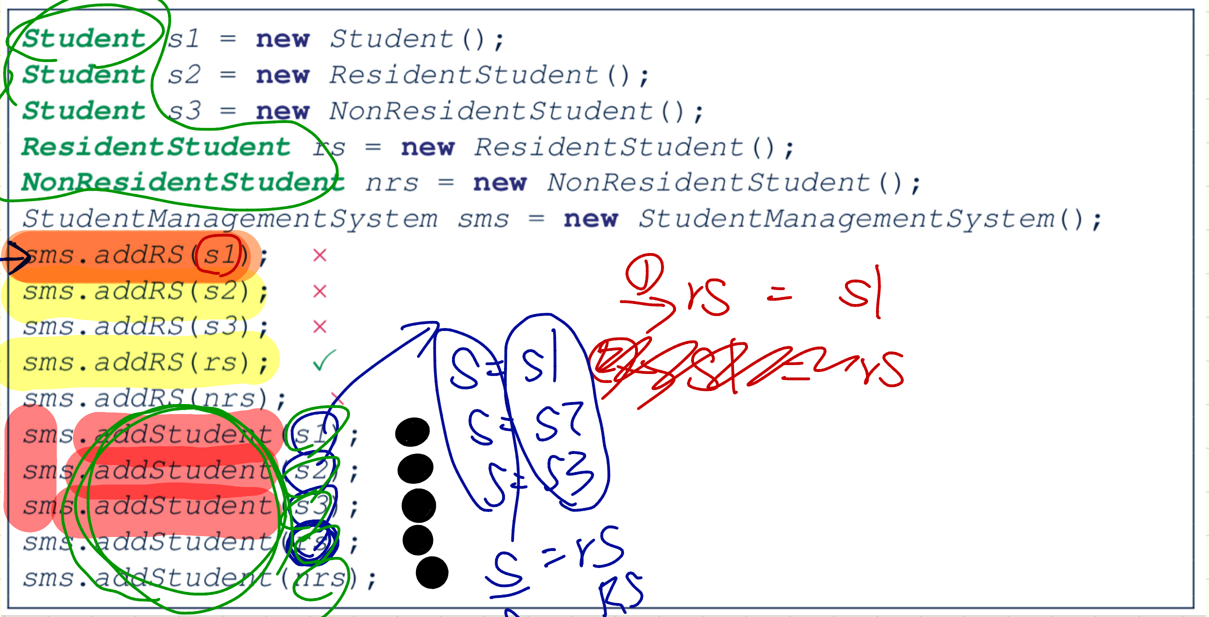
Compile ∵ The ST of rs (RS) is a descendant of the ST of ss[C]

# Polymorphic Arguments (2)

rs = s1
Student
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++; } }
```



```
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);    ●
```

① rs = s1

② ~~s1 = rs~~

s = s1
s = s2
s = s3

s = rs
S    RS

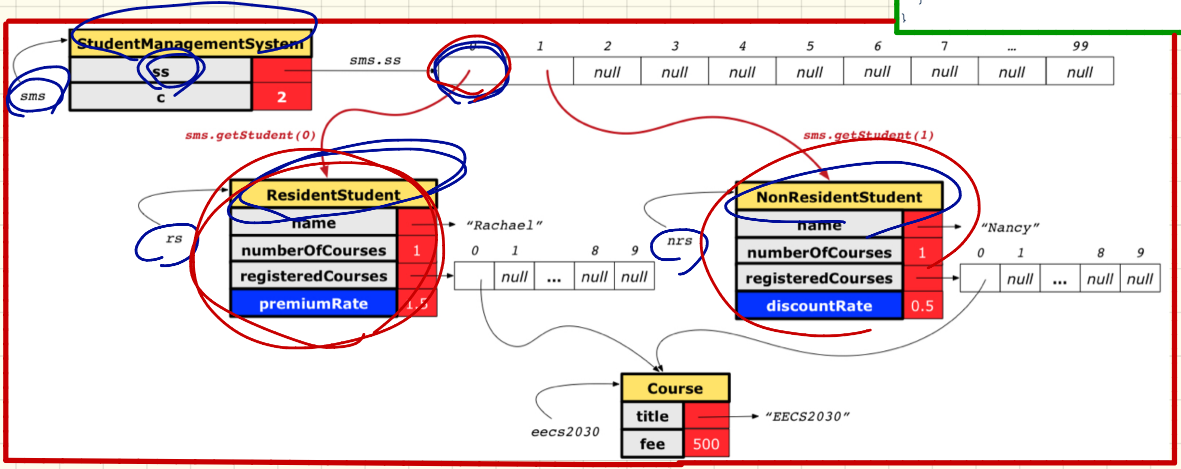
# A Polymorphic Collection of Students

ST:S
DT:RS
sms.ss[0].getTuition()

DT:NRS
ST:S
sms.ss[1].getTuition()

```
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());
}
```

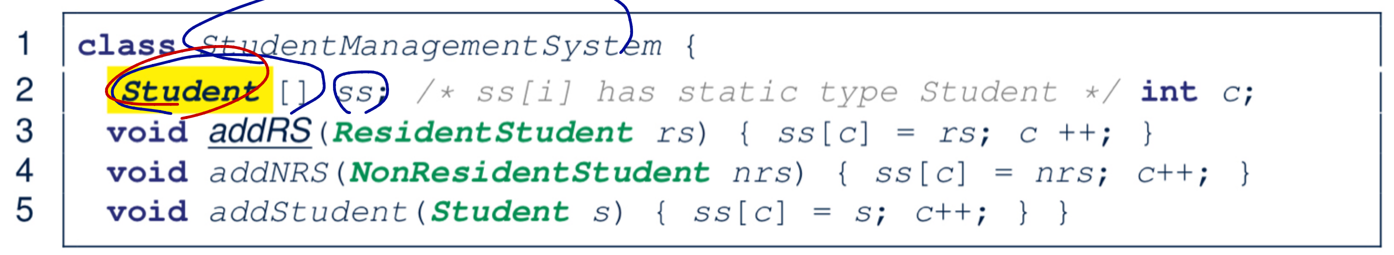
```
class StudentManagementSystem {
  Student[] students;
  int numOfStudents;

  void addStudent(Student s) {
    students[numOfStudents] = s;
    numOfStudents ++;
  }

  void registerAll (Course c) {
    for(int i = 0; i < numberOfStudents; i ++) {
      students[i].register(c)
    }
  }
}
```



Wednesday Nov. 14

Lecture 19

# Polymorphic Arguments (1)

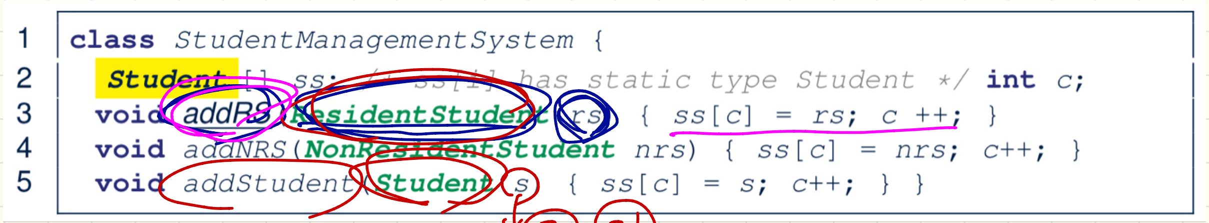


```
class StudentManagementSystem {
  Student [] ss; /* ss[i] has static type Student */ int c;
  void addRS(ResidentStudent rs) { ss[c] = rs; c ++; }
  void addNRS(NonResidentStudent nrs) { ss[c] = nrs; c++; }
  void addStudent(Student s) { ss[c] = s; c++; } }
```

Q. Static type of ss[0], ss[1], ..., ss[ss.length - 1] ?

Q. In addRS: does ss[c] = rs compile?

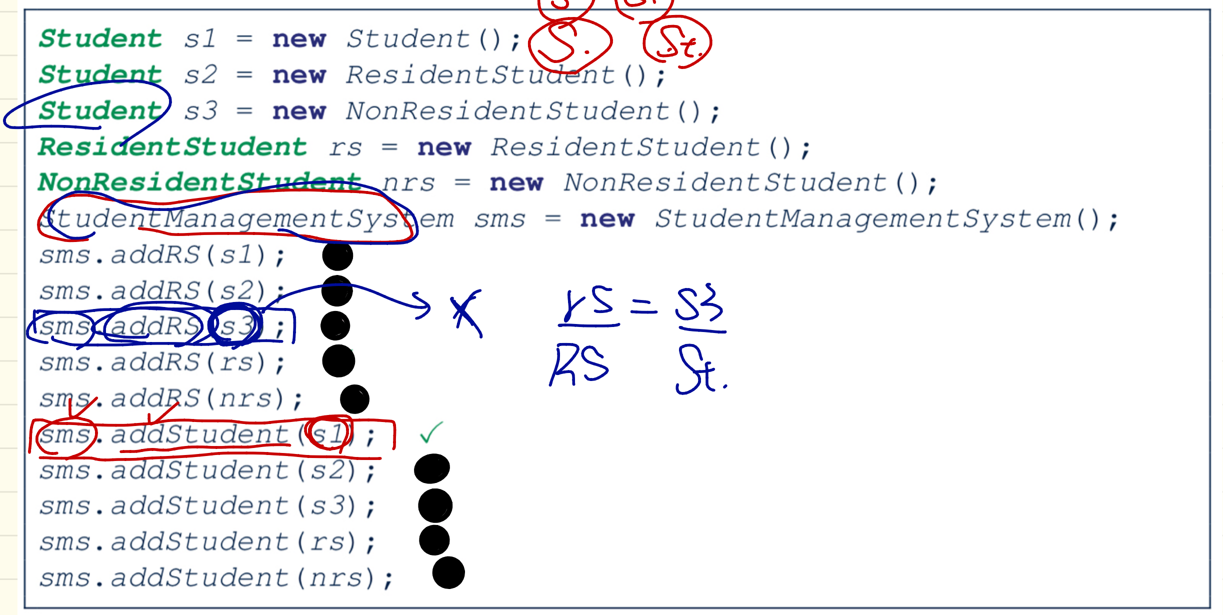
# Polymorphic Arguments (2)



```
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++; } }
```

s = s1

S.   St.



```
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);          ●   X
sms.addRS(rs);          ●
sms.addRS(nrs);          ●
sms.addStudent(s1);        ✓
sms.addStudent(s2);        ●
sms.addStudent(s3);        ●
sms.addStudent(rs);        ●
sms.addStudent(nrs);        ●
```

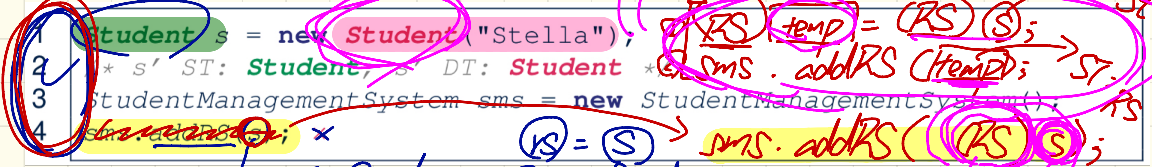
rs = s3

RS   St.

# Casting Arguments

sms. addRS ( (ResidentStudent) s ) ;

Compiles ?



```
Student s = new Student("Stella");
/* s' ST: Student; s' DT: Student */
StudentManagementSystem sms = new StudentManagementSystem();
~~sms.addRS(s);~~ ×
```

ST: S.

RS temp = (RS) s;

sms . addRS (temp);

ST.

rs = s

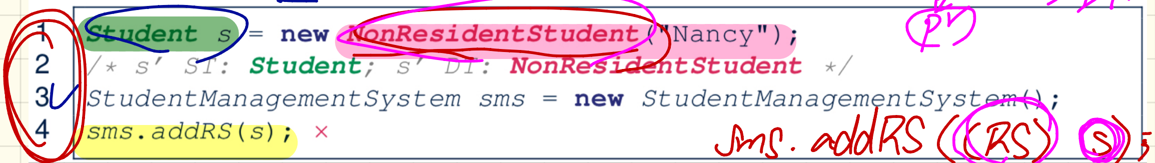
ST: Student   RS   Stud.

sms. addRS ( (RS) s );

ClassCastException ?

PV

DT: Student → CCE .



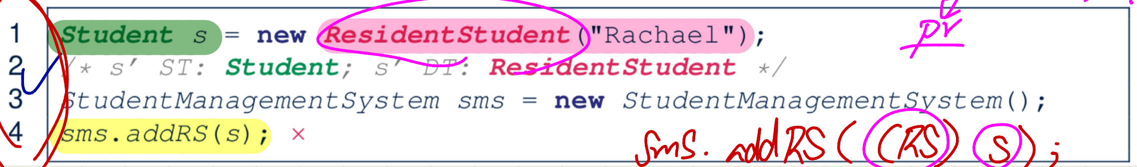
```
Student s = new NonResidentStudent("Nancy");
/* s' ST: Student; s' DT: NonResidentStudent */
StudentManagementSystem sms = new StudentManagementSystem();
sms.addRS(s); ×
```

ClassCastException ?

sms. addRS ((RS) s);

DT: NRS → CCE

PV

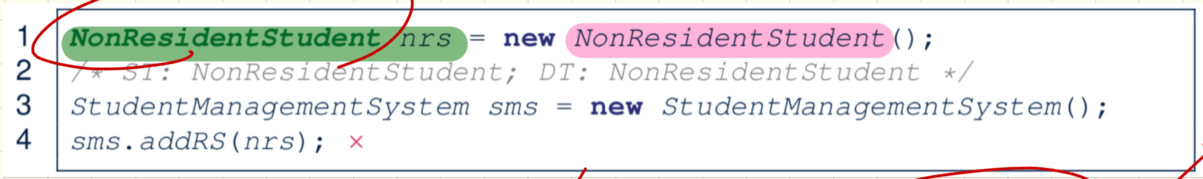


```
Student s = new ResidentStudent("Rachael");
/* s' ST: Student; s' DT: ResidentStudent */
StudentManagementSystem sms = new StudentManagementSystem();
sms.addRS(s); ×
```

ClassCastException ?

sms. addRS ( (RS) s ) ;

DT: RS → No CCE



```
NonResidentStudent nrs = new NonResidentStudent();
/* ST: NonResidentStudent; DT: NonResidentStudent */
StudentManagementSystem sms = new StudentManagementSystem();
sms.addRS(nrs); ×
```

ST: NRS

✓ sms. addRS ( (ResidentStudent) nrs ) ;

No Compiles ?

# A Polymorphic Collection of Students

at runtime : T

if (sms.ss[0] instanceof NRS) {

NRS nrs = (NRS) sms.ss[0];

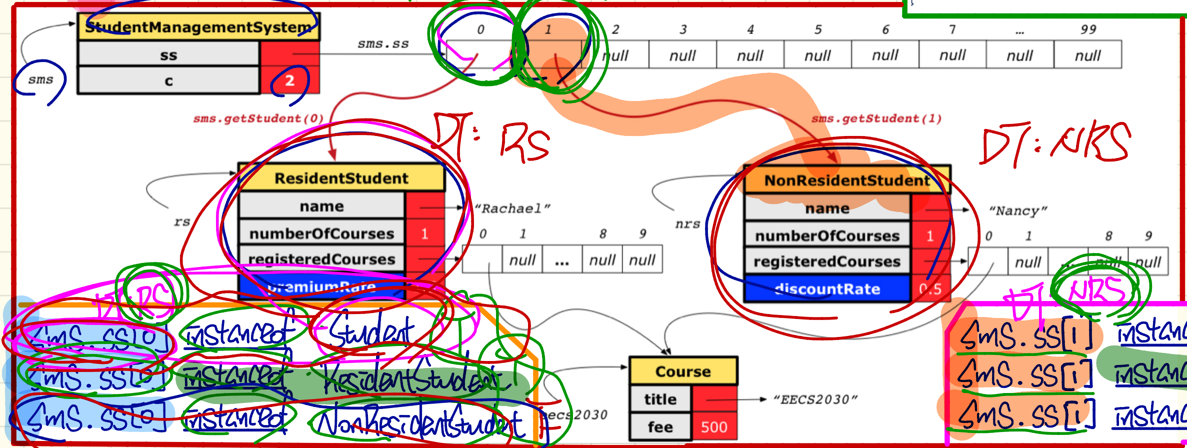
}

```
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() );
}
```

```
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)
  }
 }
}
```

ST: Student

ST: Student



DT: RS

sms.ss[0] instanceof Student

sms.ss[0] instanceof ResidentStudent

sms.ss[0] instanceof NonResidentStudent

DT: NRS

sms.ss[1] instanceof Student

sms.ss[i] instanceof ResidentStudent

sms.ss[i] instanceof NonResidentStudent

# Polymorphic Return Values

return type (static)

```
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;
  } }
```

0 1

DT: NRS (s) = ss[i]; → DT: NRS

s = ss[0];

DT: RS

DT: RS

```
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 of s?

RS

ST: Student

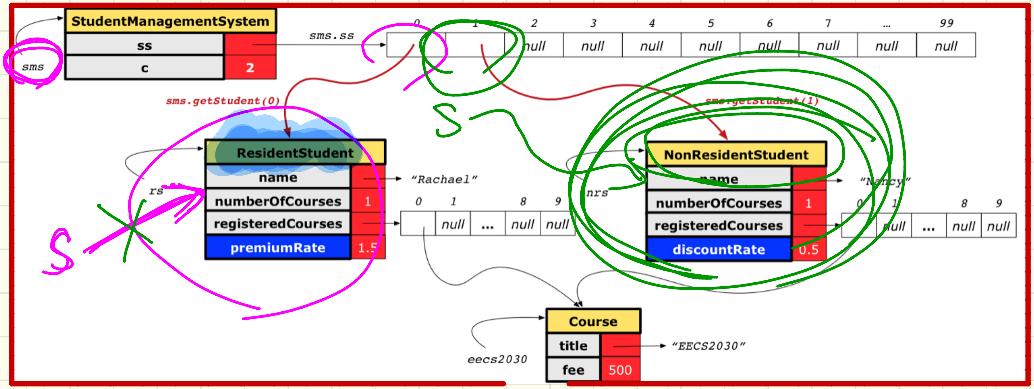
T

F

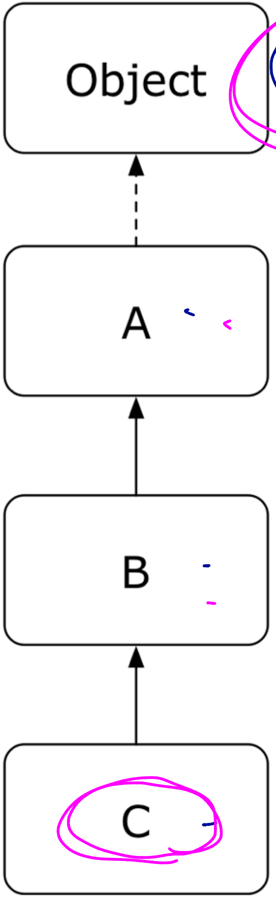
DT: RS

DT: NRS

1



# Overridden Method & 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));
```

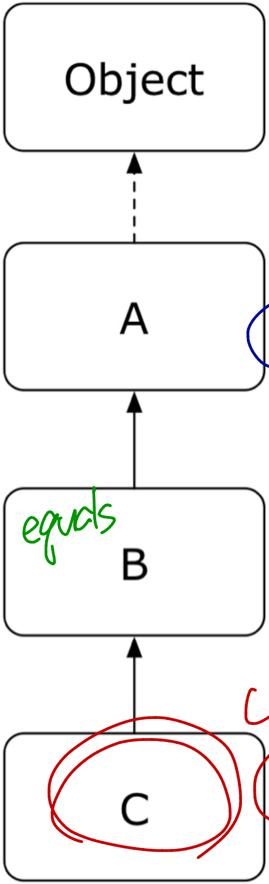
**L3** calls which version of equals? [Object]

Q1: compile? ✓

Q2: version?

# Overridden Method & Dynamic Binding (2)

A



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

```
boolean equals (Object obj) {
  /* overridden version */
}
```

Point p1 = - - ;

p1. equals (c1) ;

if ( this. getClass() != othe. getC..){

} F

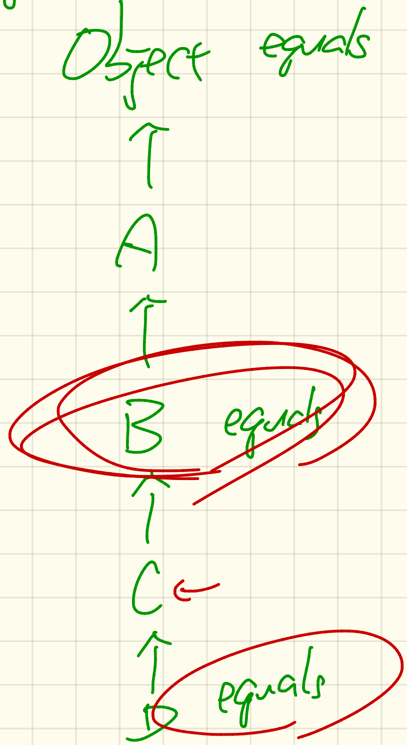
```
class A {
  /*equals not overridden*/
}
class B extends A {
  /*equals not overridden*/
}
class C extends B {
  boolean equals(Object obj) {
    /* overridden version */
  }
}
```

```
1  Object c1 = new C();
2  Object c2 = new C();
3  println(c1.equals(c2));
```

**L3** calls which version of equals? [ C ]

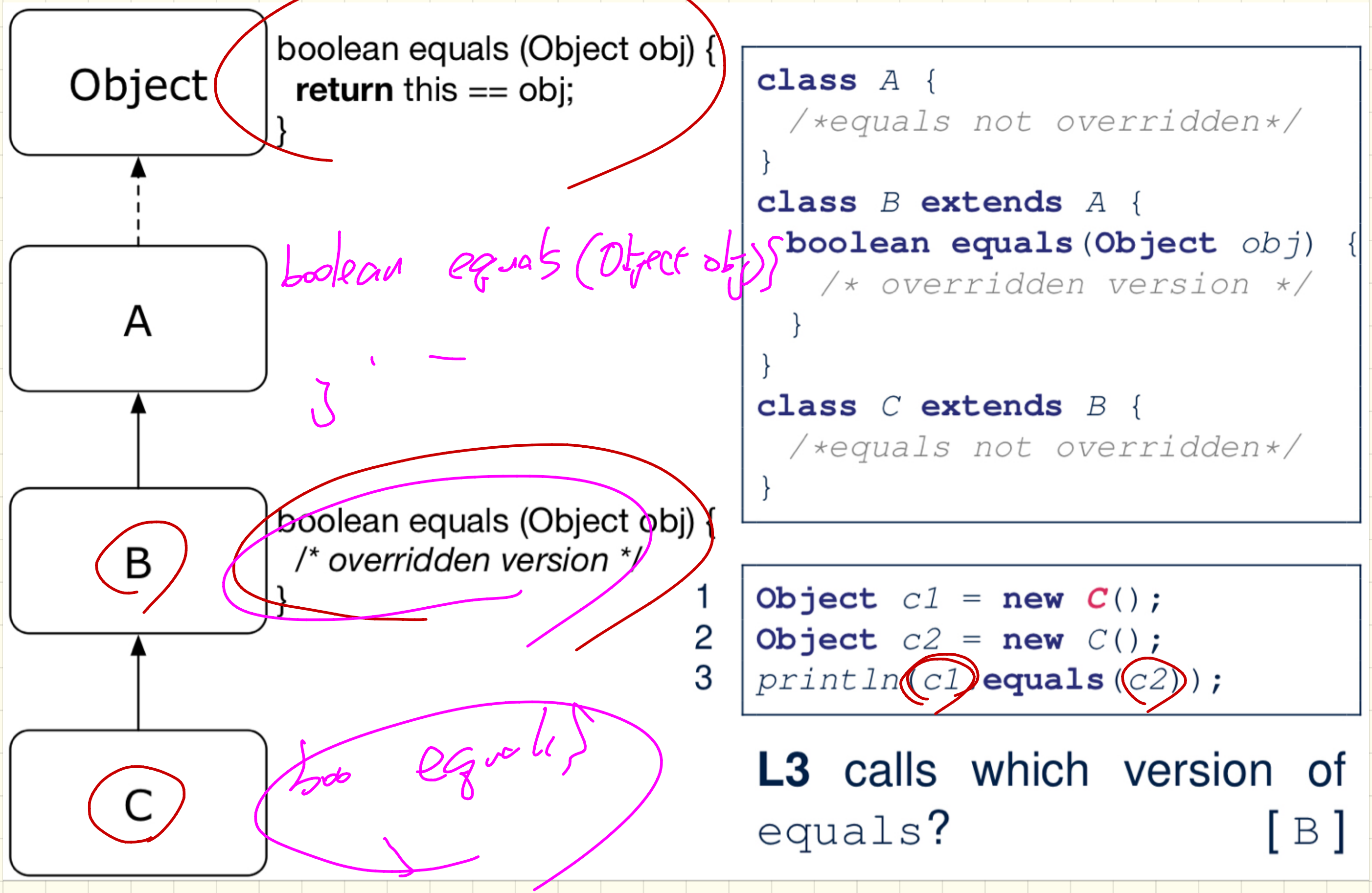
```
Point p1 = new Point (3, 4);

println (p1);

println (p1.toString());
```

```
① o2.equals(o3);     DT: C
② o3.equals(o2);     DT: D
```



```
Object o1 = new B();
Object o2 = new C();
Object o3 = new D();
  o1.equals(o2);          o2.equals(o1);
DT:B                      DT: C
```

# Overridden Method & Dynamic Binding (3)



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

boolean equals (Object obj) {

}

```
boolean equals (Object obj) {
  /* overridden version */
}
```

boo equals }

```
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 ]

Monday Nov. 19

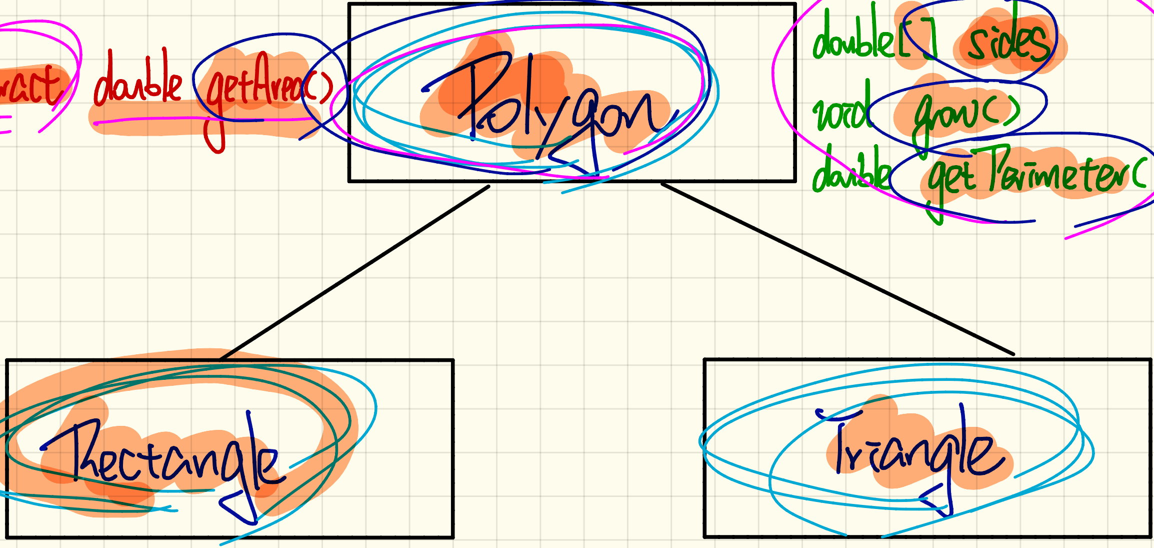
Lecture 20

# Abstract vs. Concrete Implementations

Polygon p = ~~new~~ Polygon();

p.getArea();

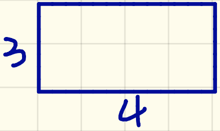
abstract double getArea()



double[] sides
void grow()
double getPerimeter()

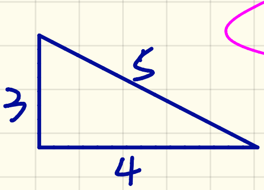
double getArea()

w * l



double getArea()

$\sqrt{s(s-a)(s-b)(s-c)}$



$\sqrt{6 \cdot 1 \cdot 3 \cdot 2} = 6$

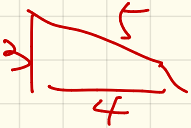
# Abstract Class and descendants

Polygon p;
p = ~~new~~ Polygon(..);
p = new Triangle(..);
p = new Rectangle(..);

```
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();
}
```

super

anynomous object



extends

extends

| 2 | 4 | 3 | 4 |
|---|---|---|---|

abstract

| 3 | 4 | 5 |
|---|---|---|

```
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]; }
}
```

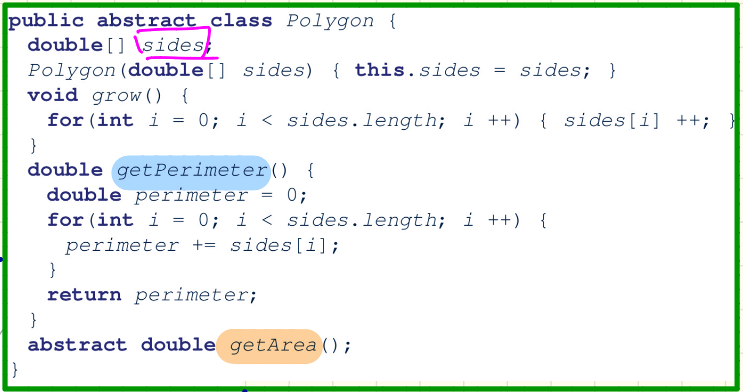
3 4

```
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;
  }
}
```

3 4 5

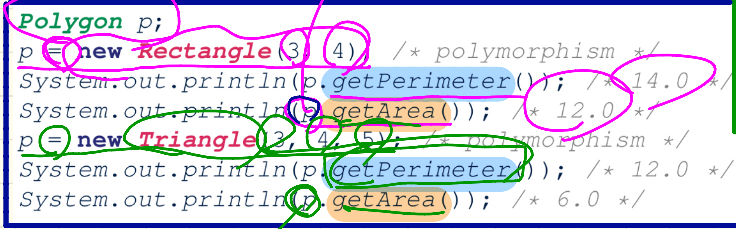
Rectangle(3, 4)

# Polymorphic Collection of Polygons



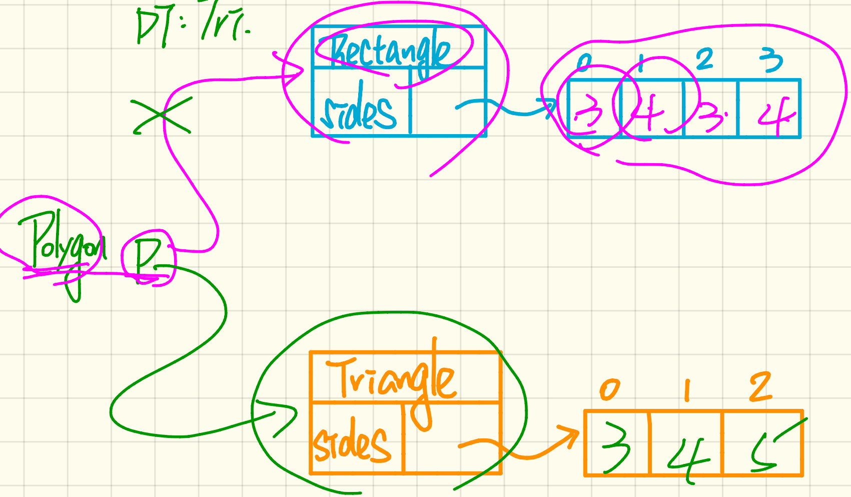
```
public abstract class Polygon {
  double[] sides;
  Polygon(double[] sides) { this.sides = sides; }
  void grow() {
    for(int i = 0; i < sides.length; i ++) { sides[i] ++; }
  }
  double getPerimeter() {
    double perimeter = 0;
    for(int i = 0; i < sides.length; i ++) {
      perimeter += sides[i];
    }
    return perimeter;
  }
  abstract double getArea();
}
```

DT: Rec.



```
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 */
```

DT: Tri.



# Polymorphic Collection of Polygons

```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 */
```

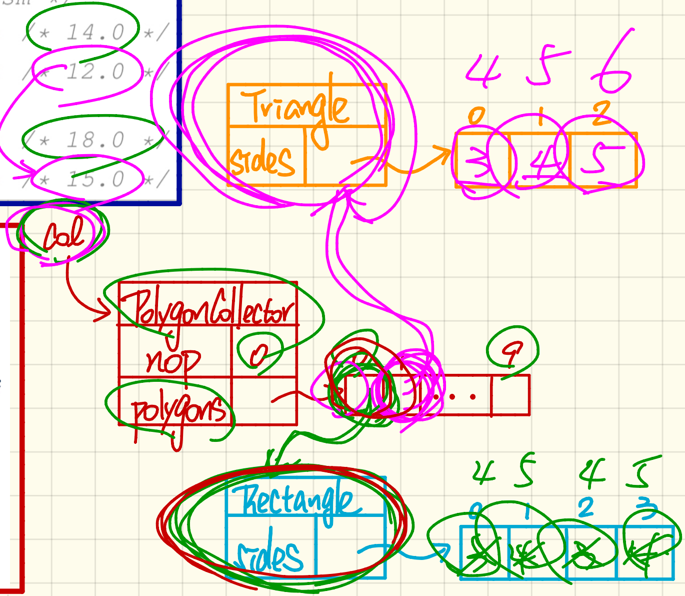
```java
public class PolygonCollector {
  Polygon[] polygons;
  int numberOfPolygons;
  PolygonCollector() { polygons = new Polygon[10]; }
  void addPolygon(Polygon p) {
    polygons[numberOfPolygons] = p; numberOfPolygons ++;
  }
  void growAll() {
    for(int i = 0; i < numberOfPolygons; i ++) {
      polygons[i].grow();
    }
  }
}
```

Tri

polygons[i].grow()

Rec.

polygons[0].grow()



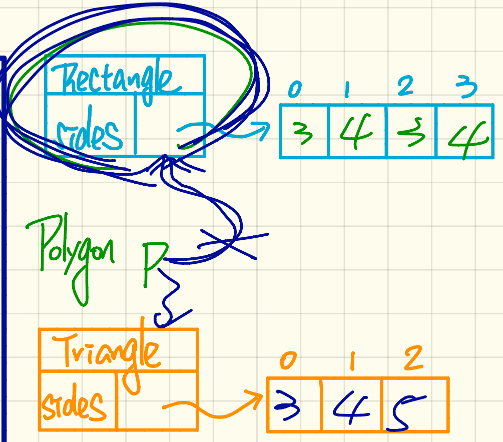
# Polymorphic Return Values of Polygons

DT: Rec.

```
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);    × X
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: Poly

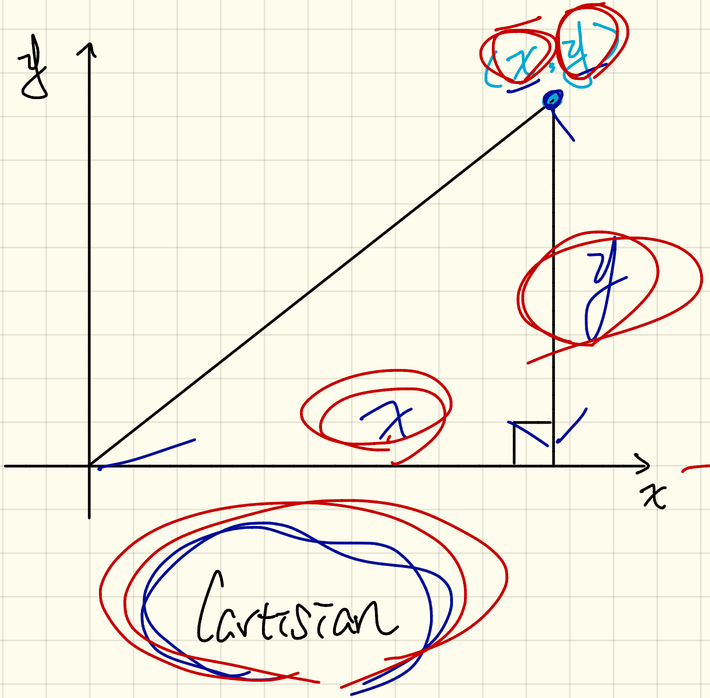
DT: Rec

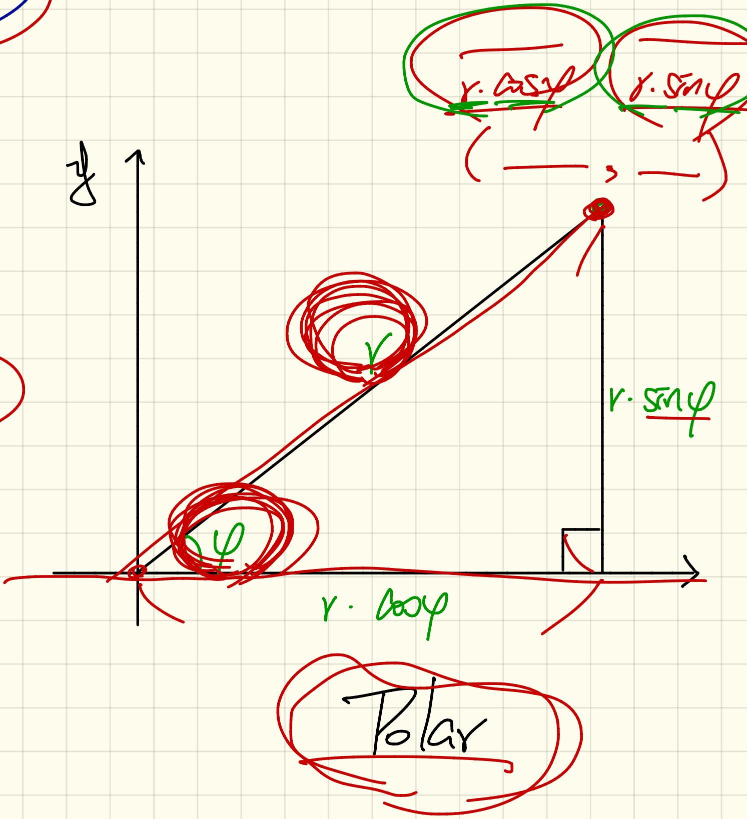


```
public abstract class Polygon {
  double[] sides;
  Polygon(double[] sides) { this.sides = sides; }
  void grow() {
    for(int i = 0; i < sides.length; i ++) { sides[i] ++; }
  }
  double getPerimeter() {
    double perimeter = 0;
    for(int i = 0; i < sides.length; i ++) {
      perimeter += sides[i];
    }
    return perimeter;
  }
  abstract double getArea();
}
```

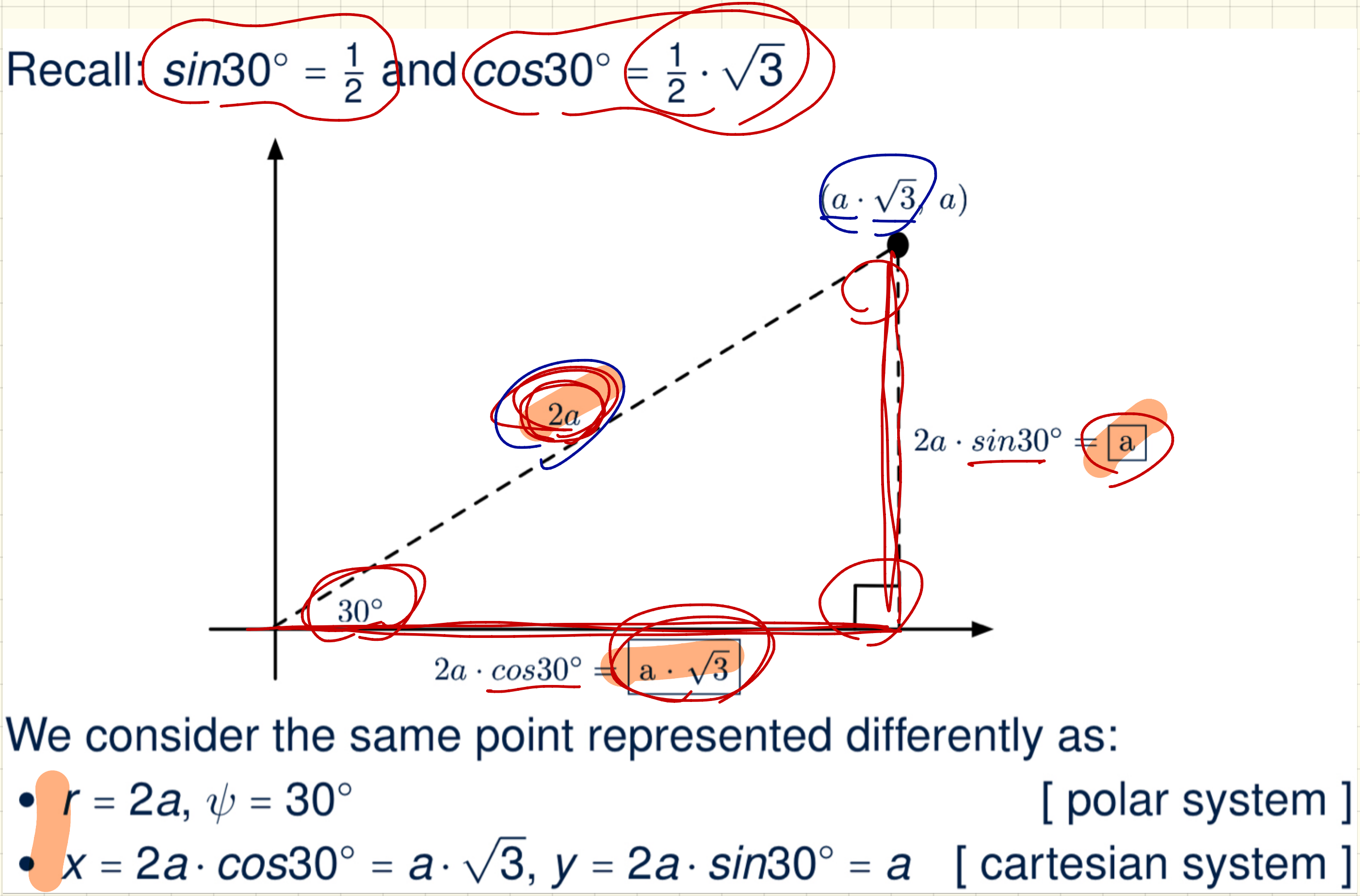
```
public class PolygonConstructor {
  Polygon getPolygon(double[] sides) {
    Polygon p = null;
    if(sides.length == 3) {
      p = new Triangle(sides[0], sides[1], sides[2]);
    }
    else if(sides.length == 4) {
      p = new Rectangle(sides[0], sides[1]);
    }
    return p;
  }
  void grow(Polygon p) { p.grow(); }
}
```

# Two Representations of a 2D Point





# Cartesian vs. Polar: Example

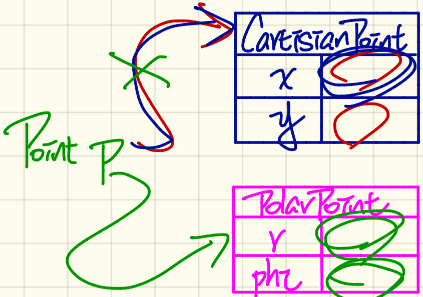


Recall: $sin30° = \frac{1}{2}$ and $cos30° = \frac{1}{2} \cdot \sqrt{3}$

$(a \cdot \sqrt{3}, a)$
$2a$
$2a \cdot sin30° = \boxed{a}$
$30°$
$2a \cdot cos30° = \boxed{a \cdot \sqrt{3}}$

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 ]



```
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; }
}
```

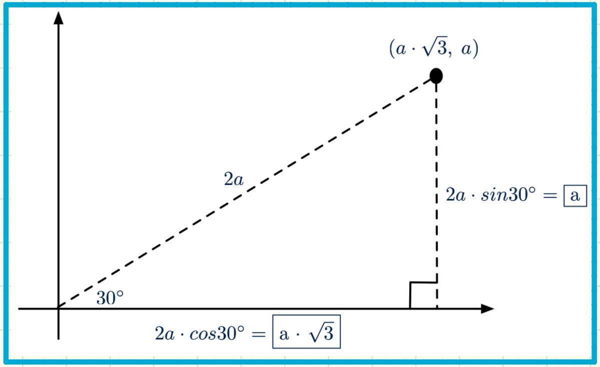
```
public class PolarPoint implements Point {
  double phi;
  double r;
  public PolarPoint(double r, double phi) {
    this.r = r;
    this.phi = phi;
  }
  public double getX() { return Math.cos(phi) * r; }
  public double getY() { return Math.sin(phi) * r; }
}
```

```
double A = 5;
double X = A * Math.sqrt(3);
double Y = A;
Point p;
p = new CartisianPoint(X, Y); /* polymorphism */
print("(" + p.getX() + ", " + p.getY() + ")");
p = new PolarPoint(2 * A, Math.toRadians(30)); /
print("(" + p.getX() + ", " + p.getY() + ")");
```

1 2 3 4

DT: CP

DT: PP

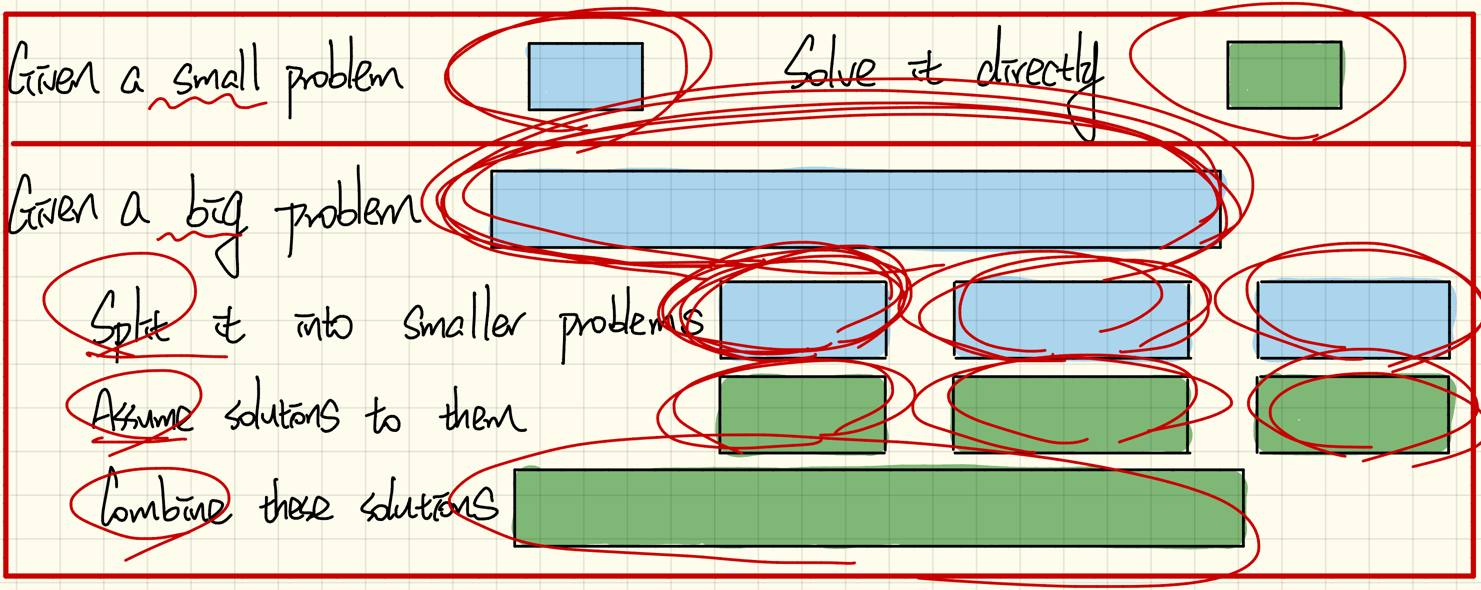


$(a \cdot \sqrt{3},\ a)$
$2a$
$2a \cdot sin30^\circ = \boxed{a}$
$30^\circ$
$2a \cdot cos30^\circ = \boxed{a \cdot \sqrt{3}}$

Wednesday Nov. 21

Lecture 21

# Solving a Problem Recursively



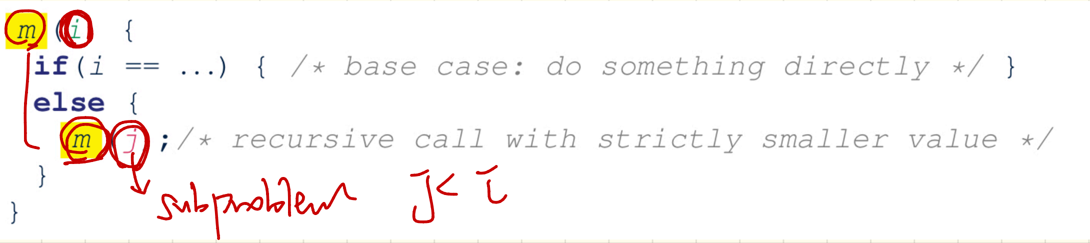
Given a small problem — Solve it directly

Given a big problem

Split it into smaller problems

Assume solutions to them

Combine these solutions



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

subproblem   $j < i$

# Factorial

recursive solution to a strictly smaller problem

small problem

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

original problem

combine

(1) 2

0 → 3

4

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

1 * fac(0)

2 * fac(1) 1

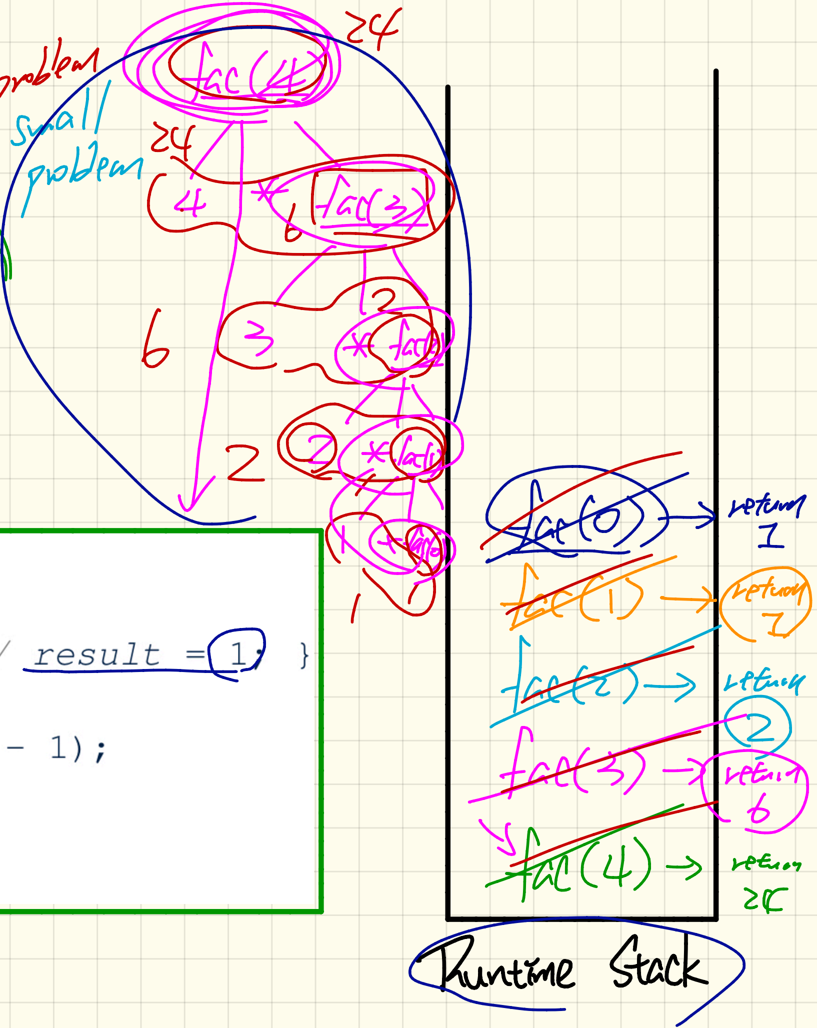
3 * fac(2) 2

4 * fac(3) 6 24

1 2

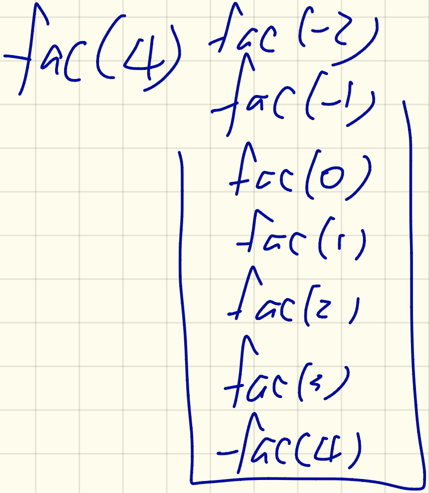
24

factorial (4)



V1

```
int fac(int n){
    int result;
    result = n * fac(n-1);
    return result;
}
```

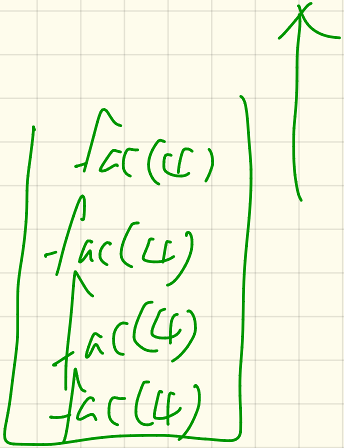


lack of base case

V2

```
int fac ( int n) {
    int result;
    if(n == 0){ result = 1;}
} else { n * fac (n) ; }
```

fac(4)



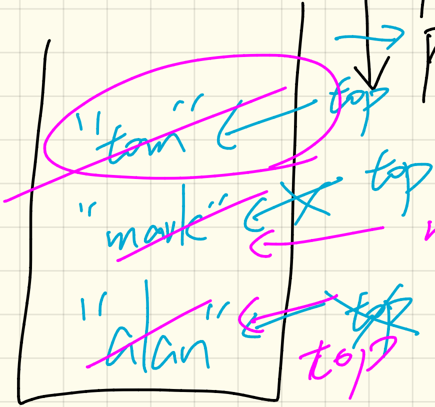
# stack

last-in-first-out

add -> push

remove -> pop

push("alan")
push("mark")
push("tom")



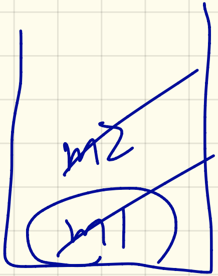
mark

pop()  tom

pop()  mark

pop  alan

m1() {
~~m2();~~
}



$F_{\underline{n}}$

$$\begin{array}{ccccccccc} 1 & 1 & 2 & 3 & 5 & 8 & 13 & 21 & 34 \; \ldots \\ F_1 & F_2 & F_3 & F_4 & F_5 & & & & \end{array}$$

$F_n$

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

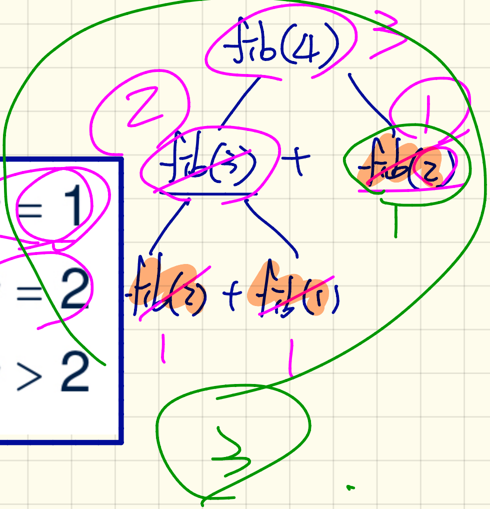
$F_{n-1}$ $\quad$ $F_{n/2}$

$F_{n-2}$

# Fibonacci Number

fib(3)

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



2 3 4 1

```
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;
}
```

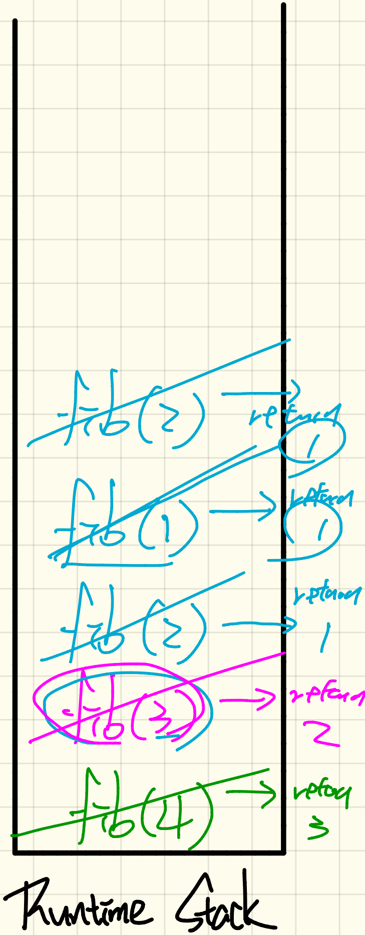
2

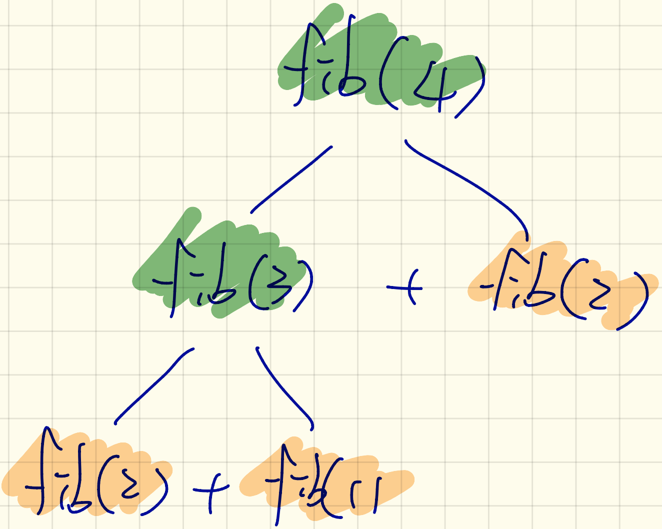
fib(2)

fib(3)

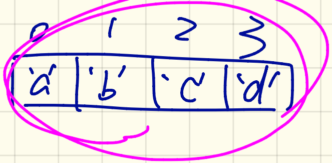
fib(2) + fib(1) 2

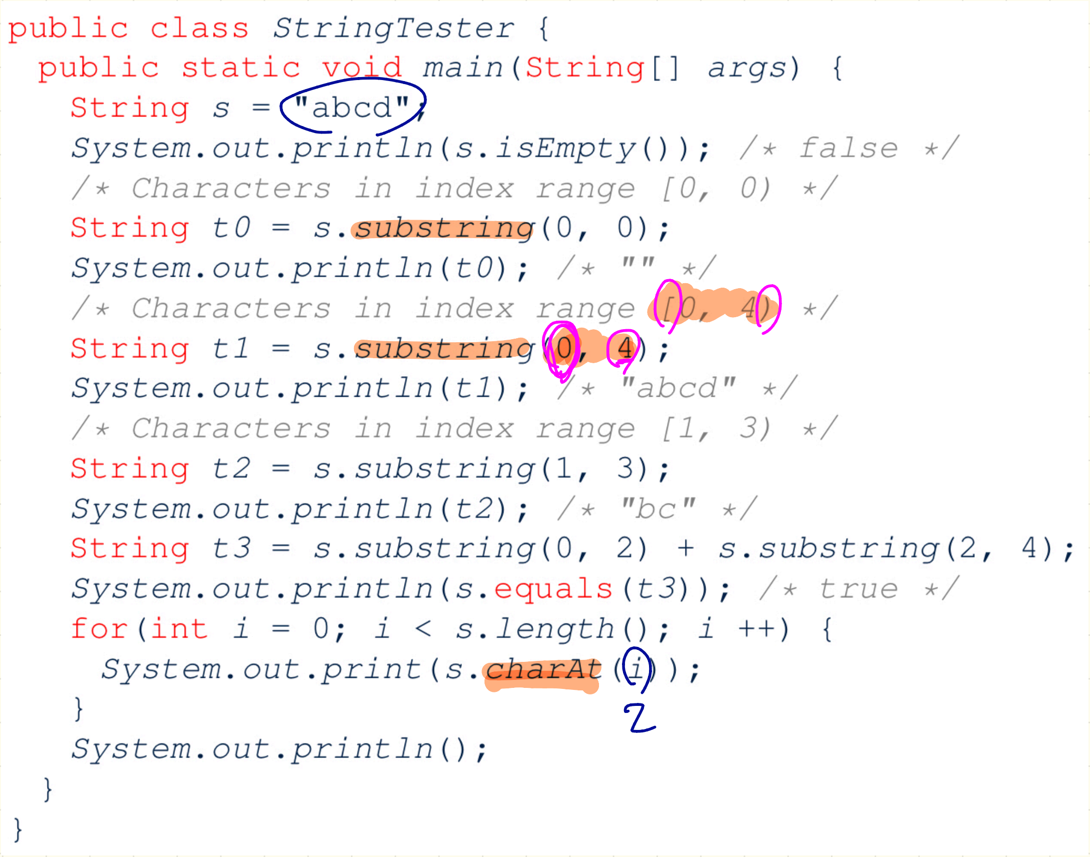
fib(4)




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

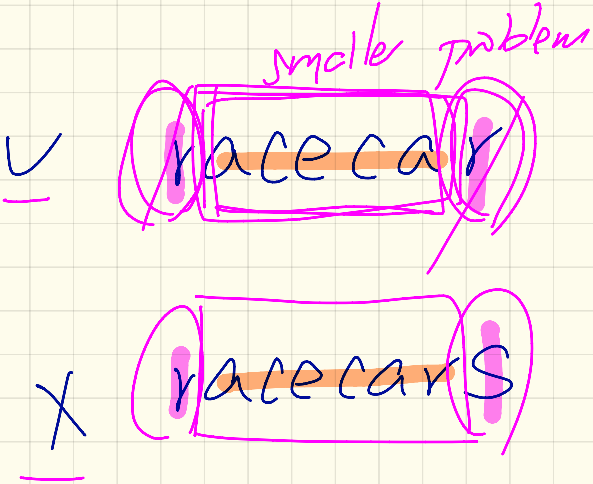
# Use of String





```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();
  }
}
```

2



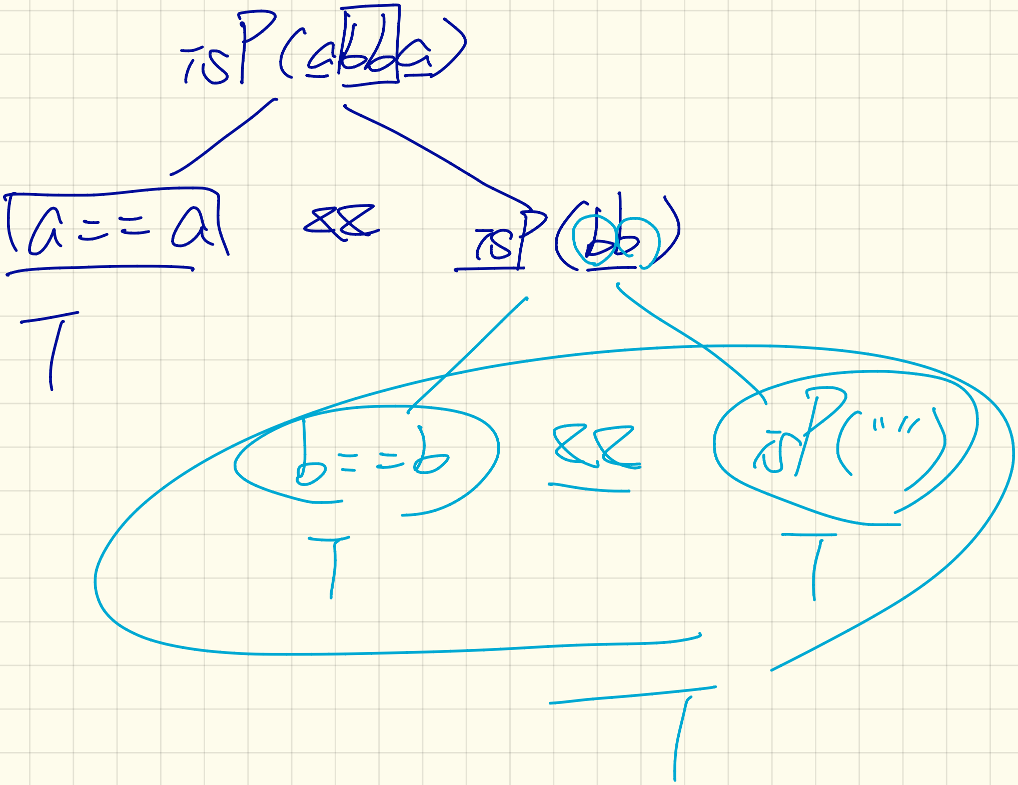
S

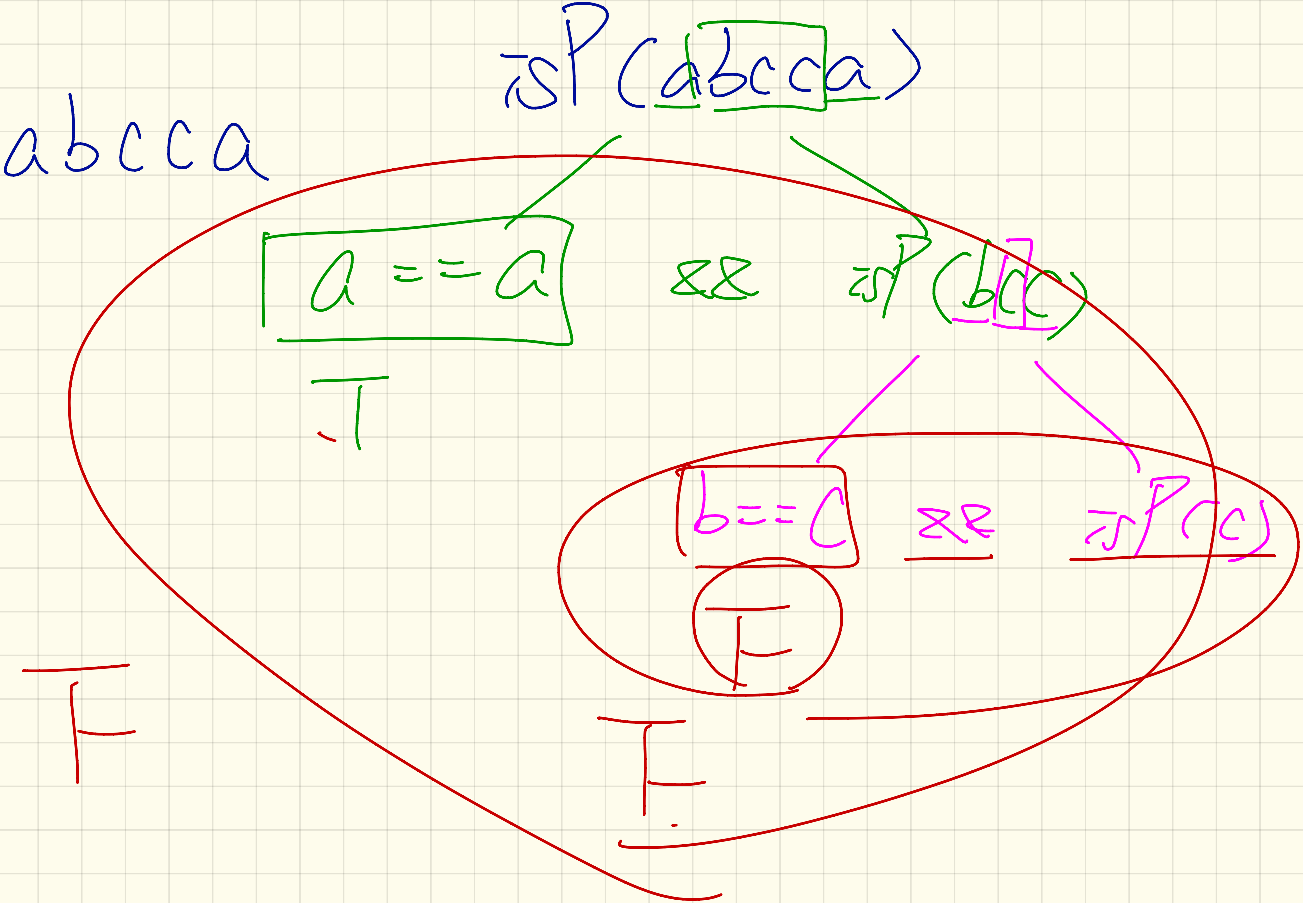
fc(S) == lc(S)

&&

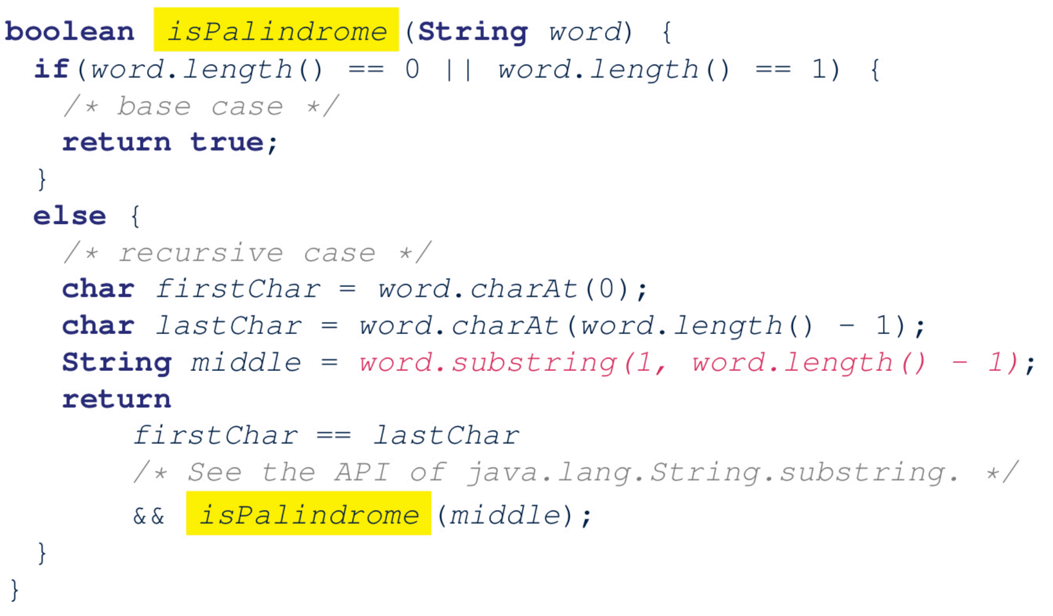
isP( substring( ——— ))

abba




abcca
isP(abcca)
a == a
&&
isP(bcc)
T
b == c
&&
isP(c)
F
F
F

# Palindrome

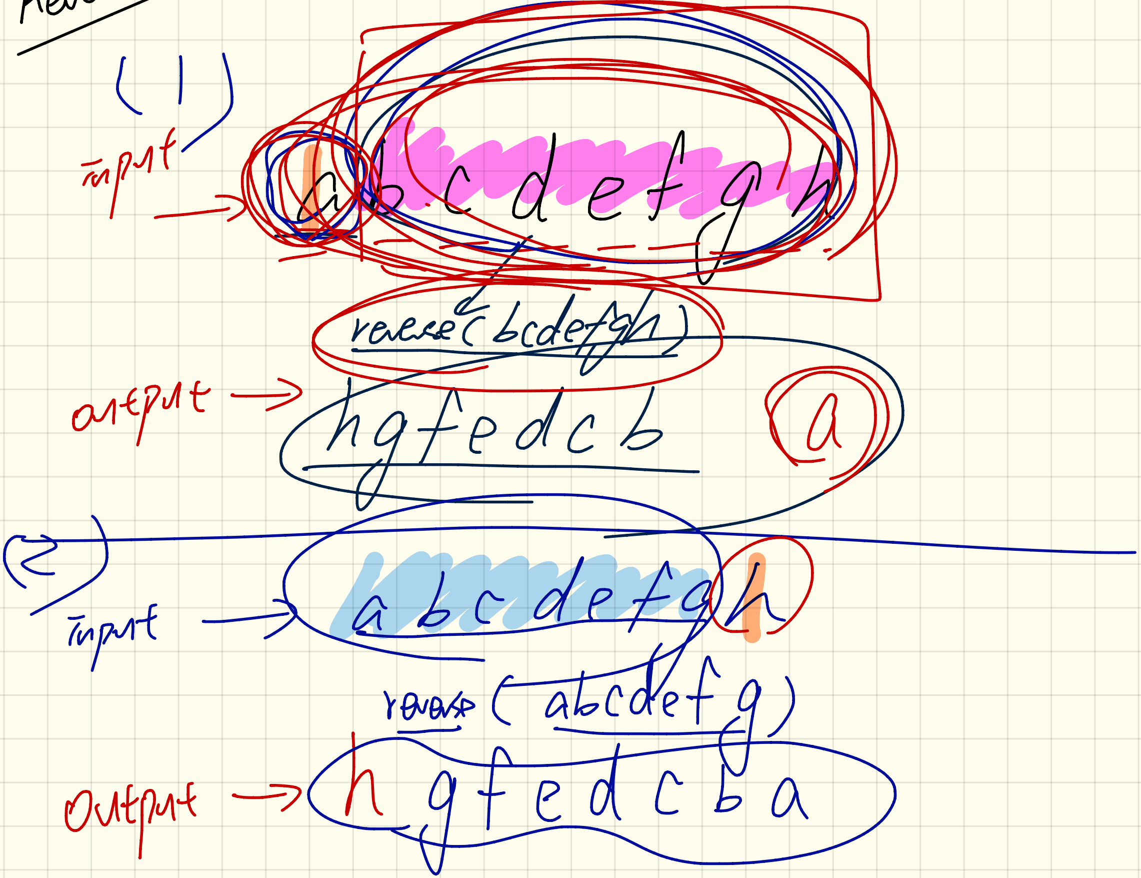


```
boolean isPalindrome (String word) {
  if(word.length() == 0 || word.length() == 1) {
    /* base case */
    return true;
  }
  else {
    /* recursive case */
    char firstChar = word.charAt(0);
    char lastChar = word.charAt(word.length() - 1);
    String middle = word.substring(1, word.length()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

Monday Nov. 26

Lecture 22

# ReverseOf.


(1)
tail
input →
abcdefgh
reverse(bcdefgh)
output →
hgfedcb
a
(2)
Input →
abcdefgh
reverse(abcdefg)
Output →
hgfedcba

# Reverse of a String

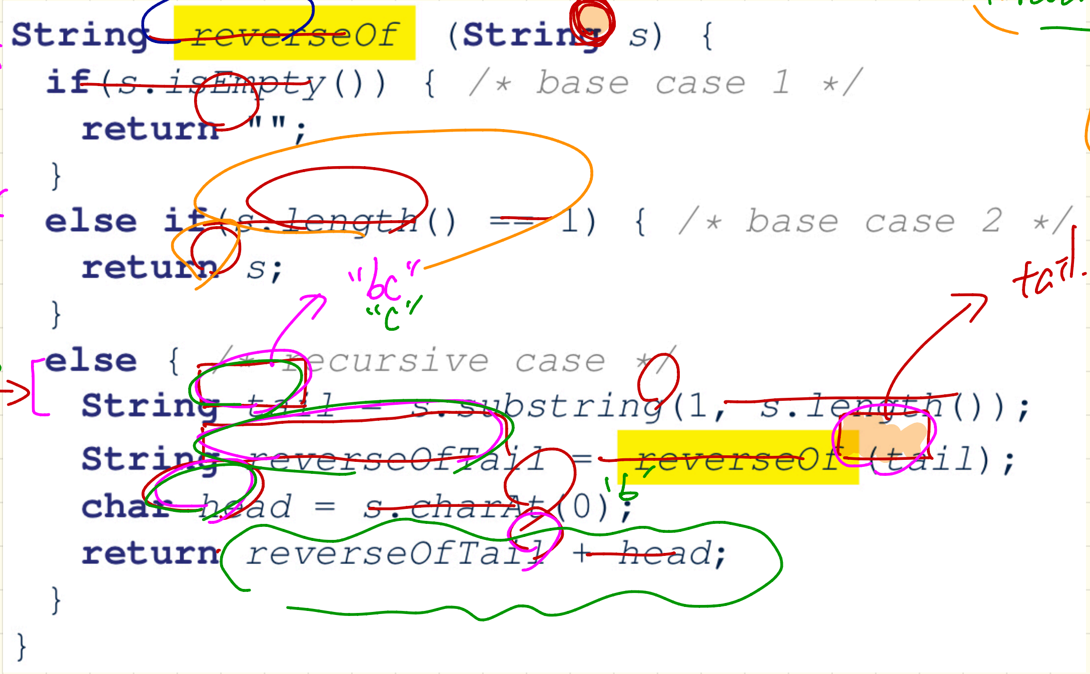
reverse("abc") cba

cb reverse("bc") + a

reverse("c") + b

c cb

"c"
"bc"
"abc"



```
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;
  }
}
```

"bc"
"c"

'b'

tail.length() < s.length()

# Number of Occurrences

abca

occ(abca)

2

'a'

'd'

0

'c'

"abca"

a == a    occ("bca", 'a')

1    +    1    =    2

d == a    +    occ("bca", 'd')

0    +    0    =    0

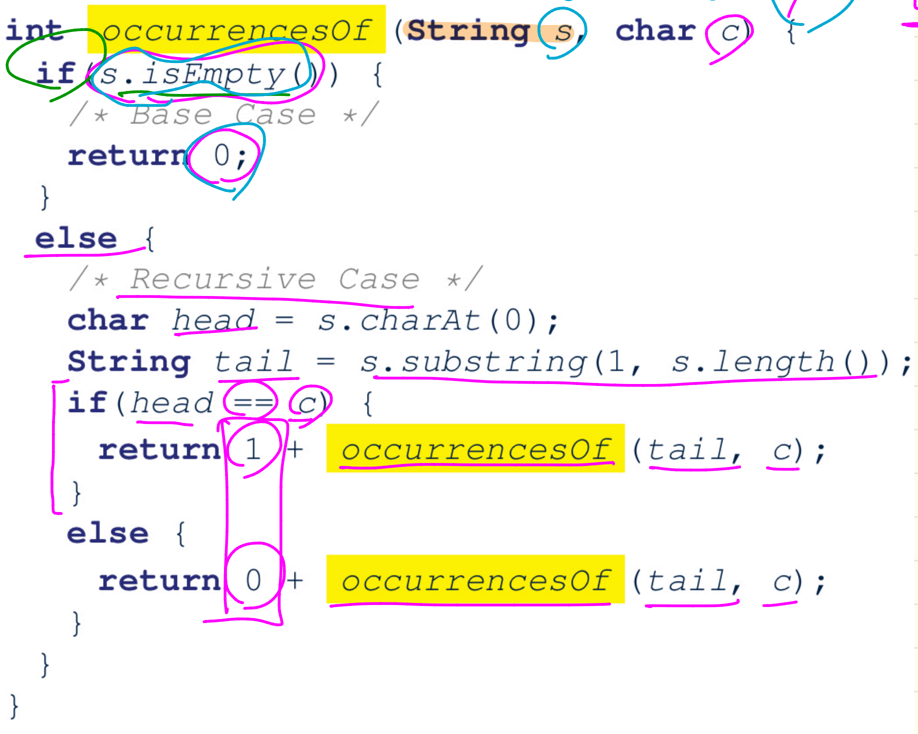
a == c    +    occ("bca", 'c')

0    +    1    =    1

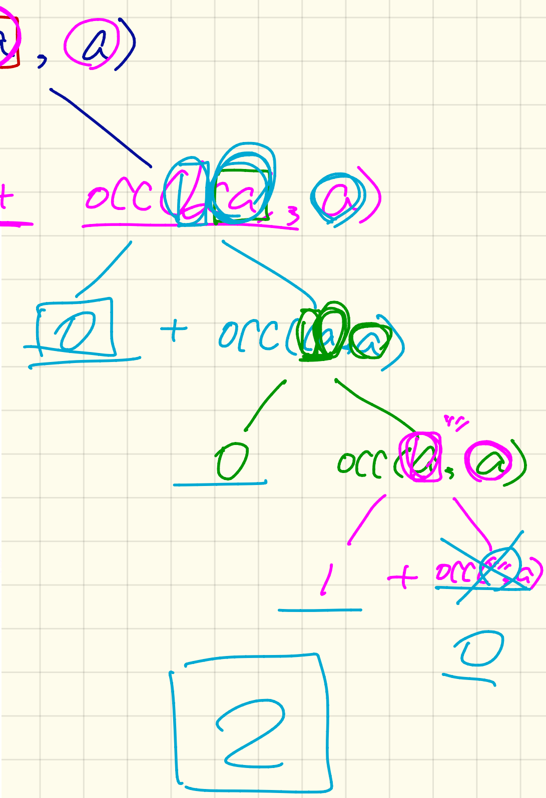
# Number of Occurrences

occ(abca, a)

bca a 1 +



```
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);
    }
  }
}
```



# Recursion on Array: Passing a new array

sub

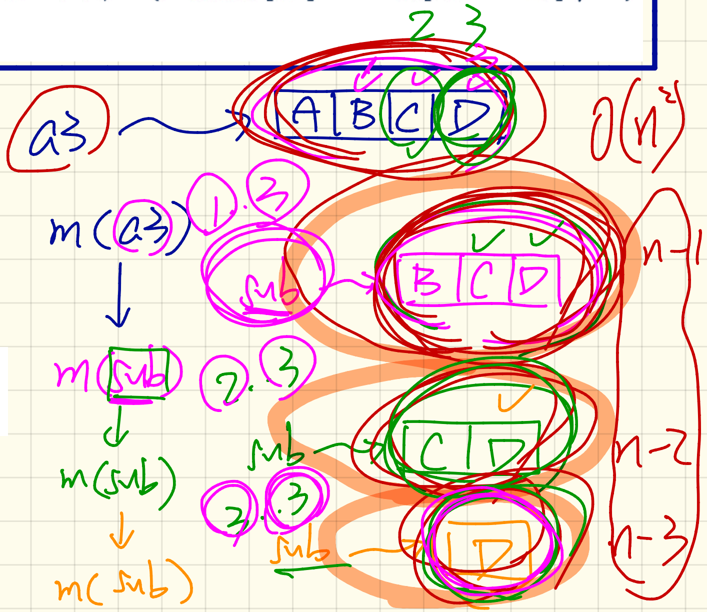
```
void m(int[] a) {
 if(a.length == 0) { /* base case */ }
 else if(a.length == 1) { /* base case */ }
 else {
  int[] sub = new int[a.length - 1];
  for(int i = 1; i < a.length; i ++) { sub[0] = a[i - 1]; }
   m(sub) } }
```

strictly smaller than a

Say a1 = {}, consider m(a1) ✓

Say a2 = {A}, consider m(a2) ✓

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



# Recursion on Array: Passing an array reference

a3 0 2 3

```
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) } }
```

m(a1, 0, a1.length-1)

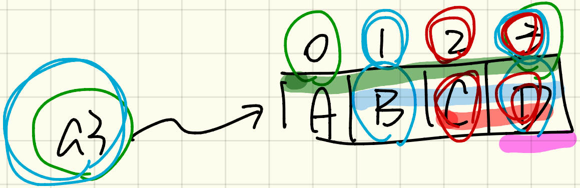
Say a1 = {}, consider ~~m(a1)~~

m(a2, 0, a2.length-1)

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

m(a3, 0, a3.length-1)

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



m(a3, 0, 3) ✓

↓

m(a3, 1, 3) ✓

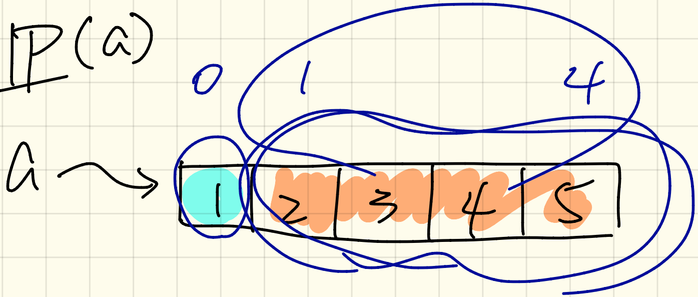
↓

m(a3, 2, 3) ✓

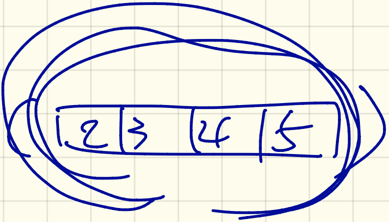
↓

m(a3, 3, 3) ✓

allP(a)



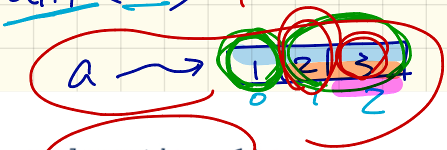
allP(a) = a[0] > 0 && allP(



T

# Are all numbers positive?

allP(a) T



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

recursive helper method

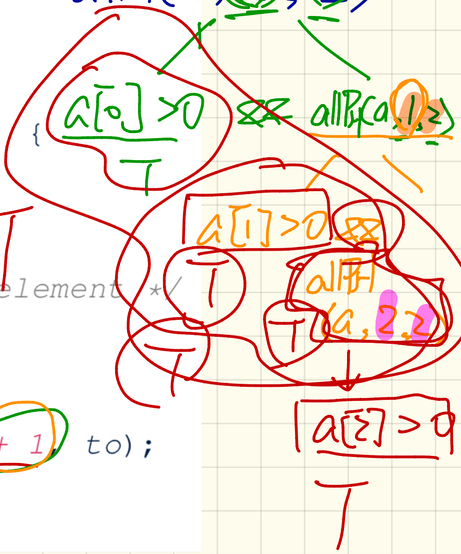
0 1    2 2

```
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);
  }
}
```

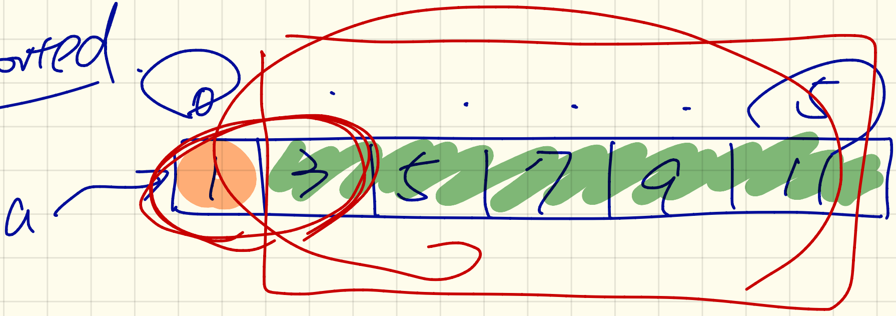
T

allP(a)

allPH(a, 0, 2)



isSorted



$$\text{isSorted}(a,\ \underline{0},\ \underline{5})$$

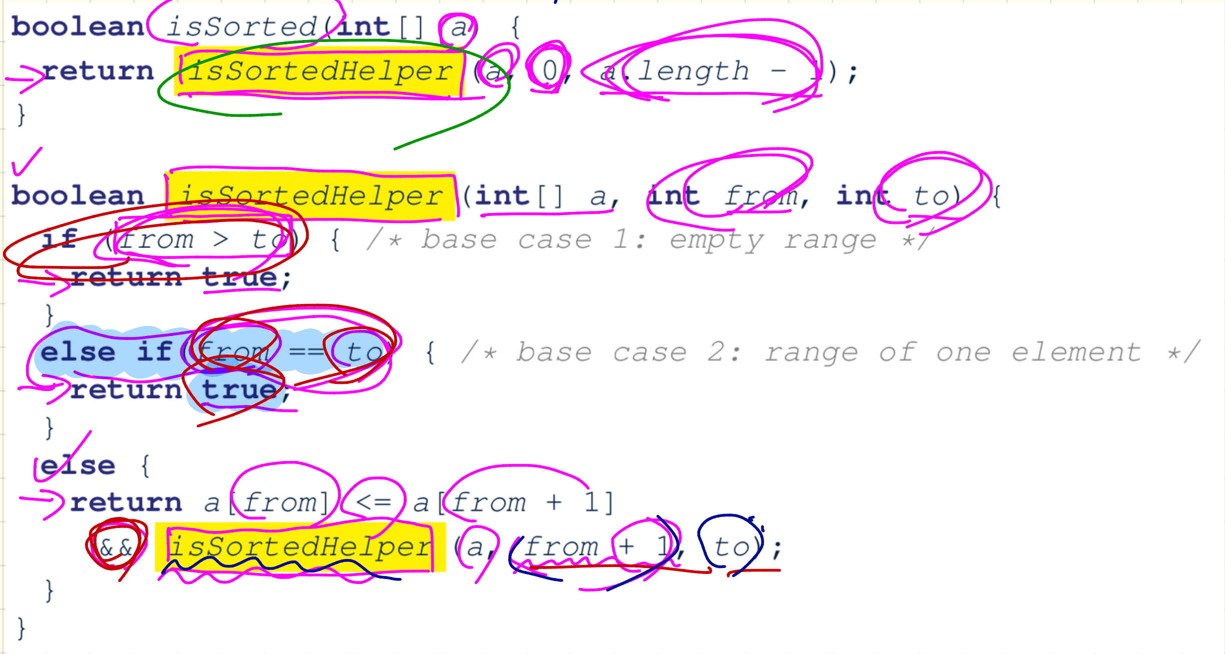
$$= \underline{a[0] \le a[0+1]} \quad \underline{\hspace{3em}} \quad \text{isSorted}(a,\ \underline{0+1},\ \underline{5})$$

Wednesday Nov. 28

Lecture 23

# Is an array sorted?

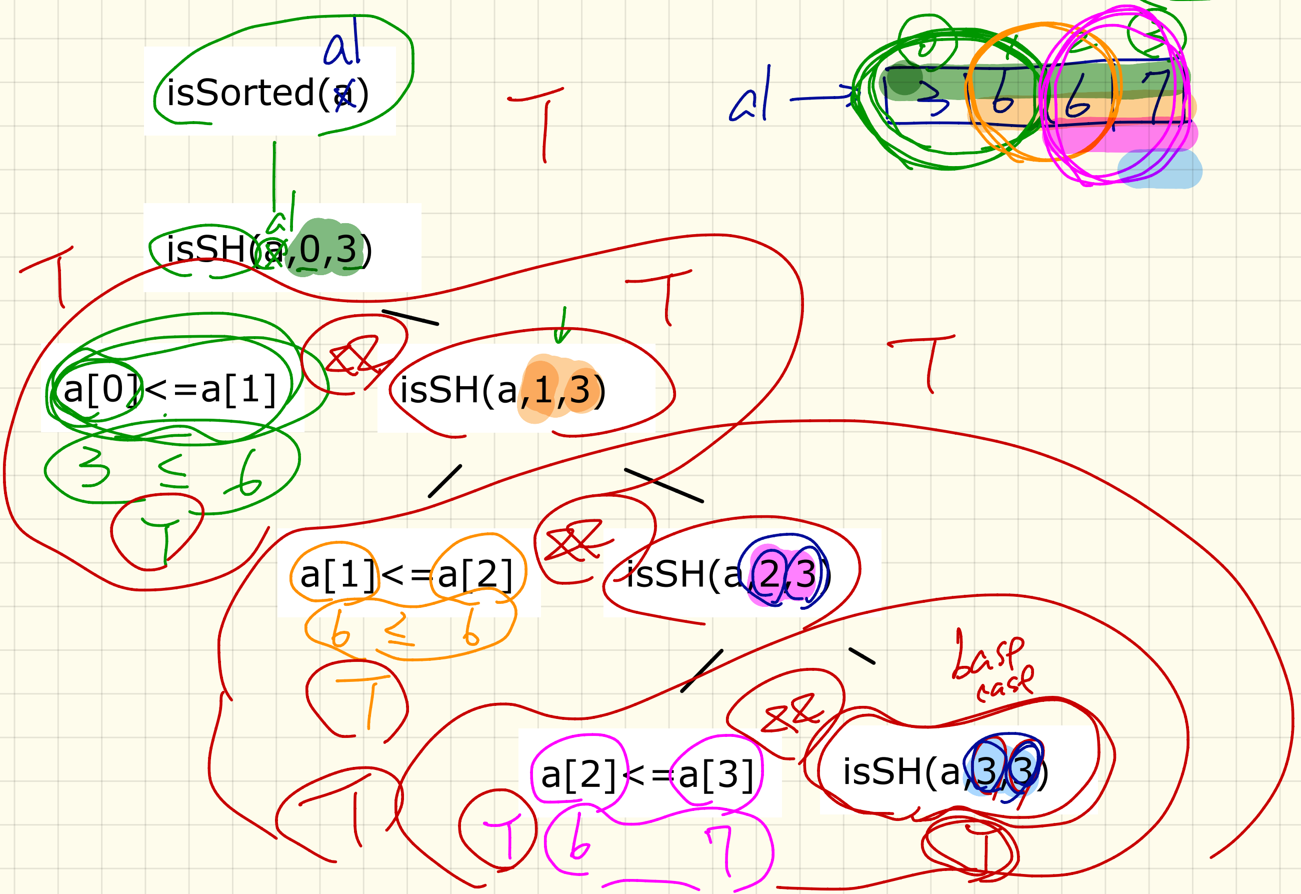
int[] a1 = {};
print( isSorted(a1))



```
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 isSorted

Say a1 = {3,6,6,7}, a2 = {3,6,5,7}



isSorted(a)

isSH(a,0,3)

a[0]<=a[1]

isSH(a,1,3)

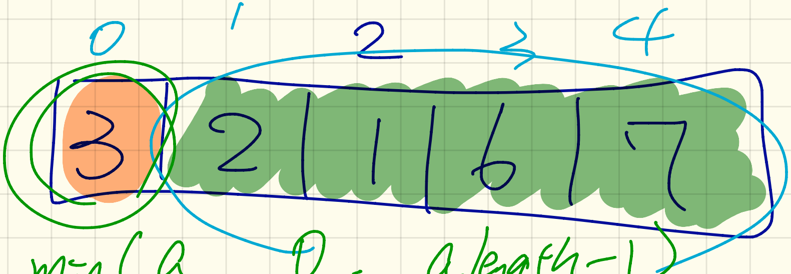
a[1]<=a[2]

isSH(a,2,3)

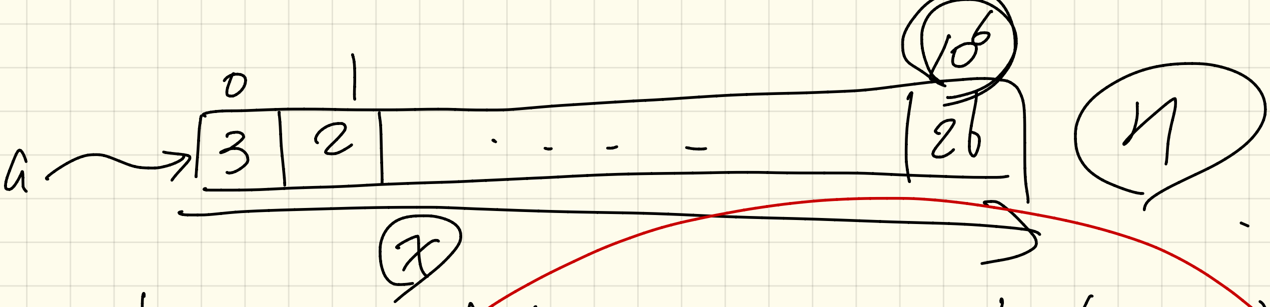
a[2]<=a[3]

isSH(a,3,3)

1



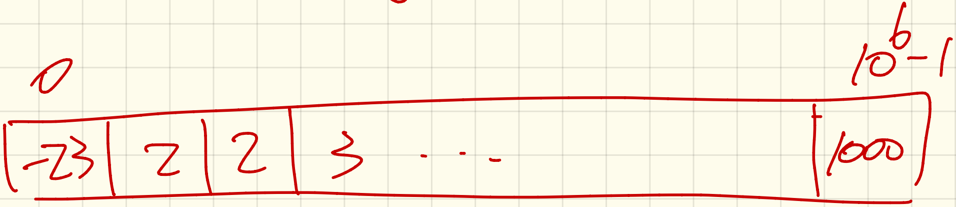
```
min(a, 0, a.length-1)
if (a is empty) { no min }
else if (a is size 1) { return a[0] }
else {
    int minOfRest = min(a, from+1, to);
    if( a[0] < minOfRest) { return a[0]; }
}-> else { return minOfRest ; }
```



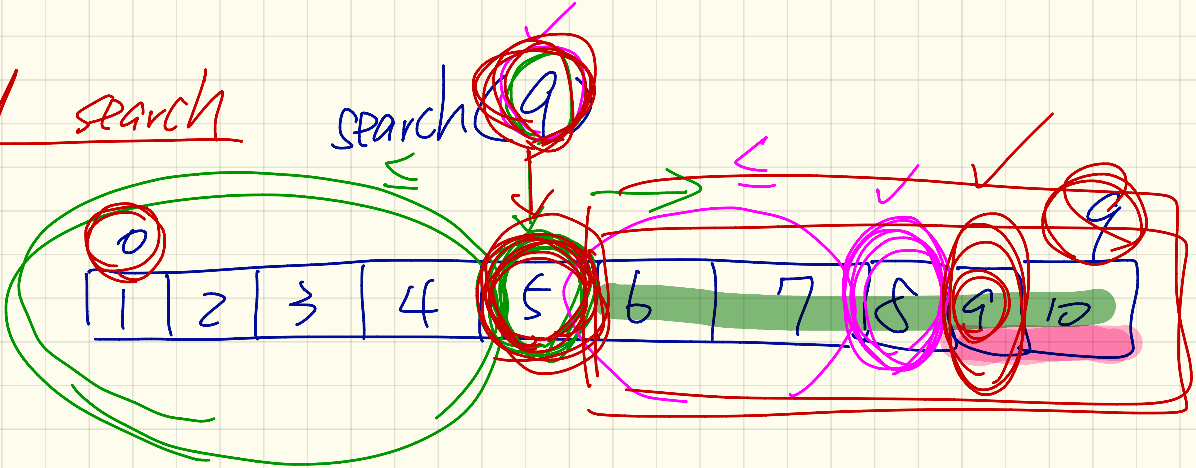
```
search(23)    for(int i=0; i<a.length; i++){
                  if(a[i] == 23){ return true}
              }
              return false;
```

$O(n)$ $10^6$

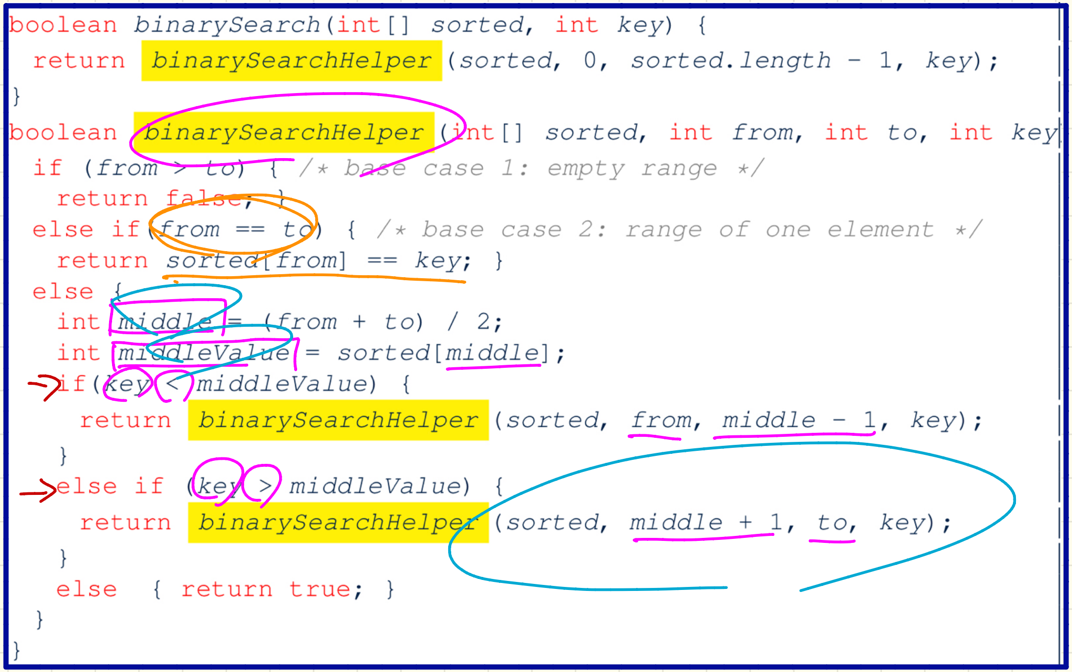
Assume input array is sorted



binary search



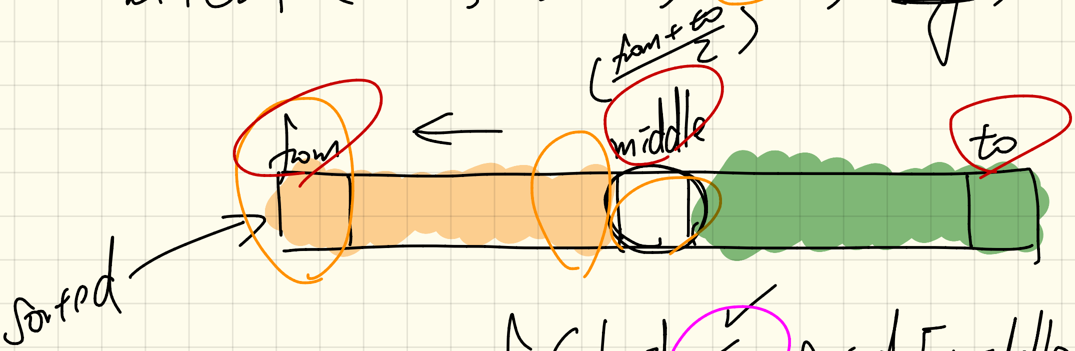
# Binary Search



```
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; }
 }
}
```

in ascending order!

```
binSH ( sorted , from , to , key )
```



```
if ( key < sorted[middle] ) {
    binSrt(sorted, from, middle-1, key);
}
else if ( key > sorted[middle] ) {
    binSrt(sorted, middle+1, to, key);
}
```

## Exercise

Modify the BinS. so that the input array is sorted in descending order.

# Binary Search: Tracing

0 1 2 3 4 5 6 7 8

Say a = {3,6,9,12,15,18,21,24,27}

search(a,18)

search(a,0,8,18)

middle = 4

middV = 15

18 > 15

mid + 1

search(a,5,8,18)

$\text{middle} = \frac{5+8}{2} = 6$

middleV = 21

18 < 21

on mid - 1

search(a,5,5,18)

Sorted [5] == 18

T

0 1 2 3 4 5 6 7 8

Say a = {3,6,9,12,15,18,21,24,27}

search(a,7)

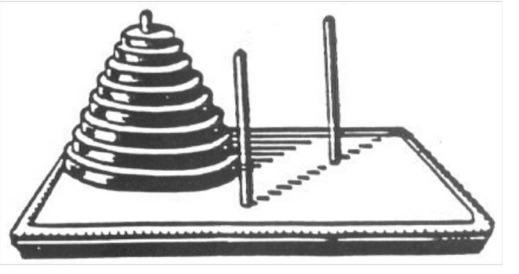
search(a,0,8,7)

search(a,0,3,7)

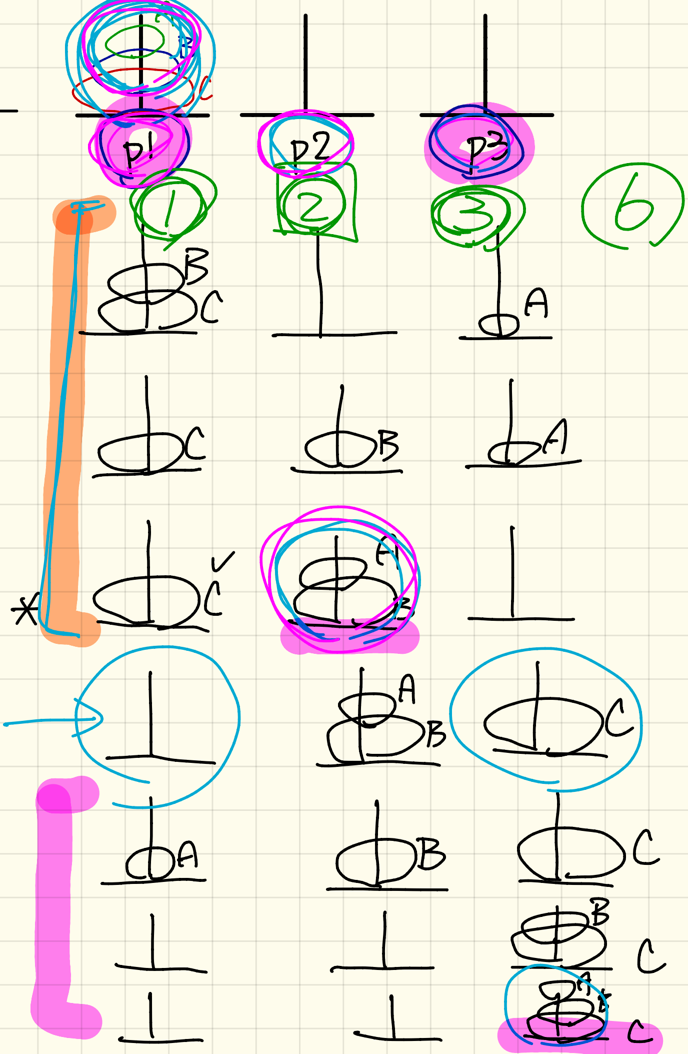
search(a,2,3,7)

search(a,2,1,7)

# Tower of Hanoi : Strategy



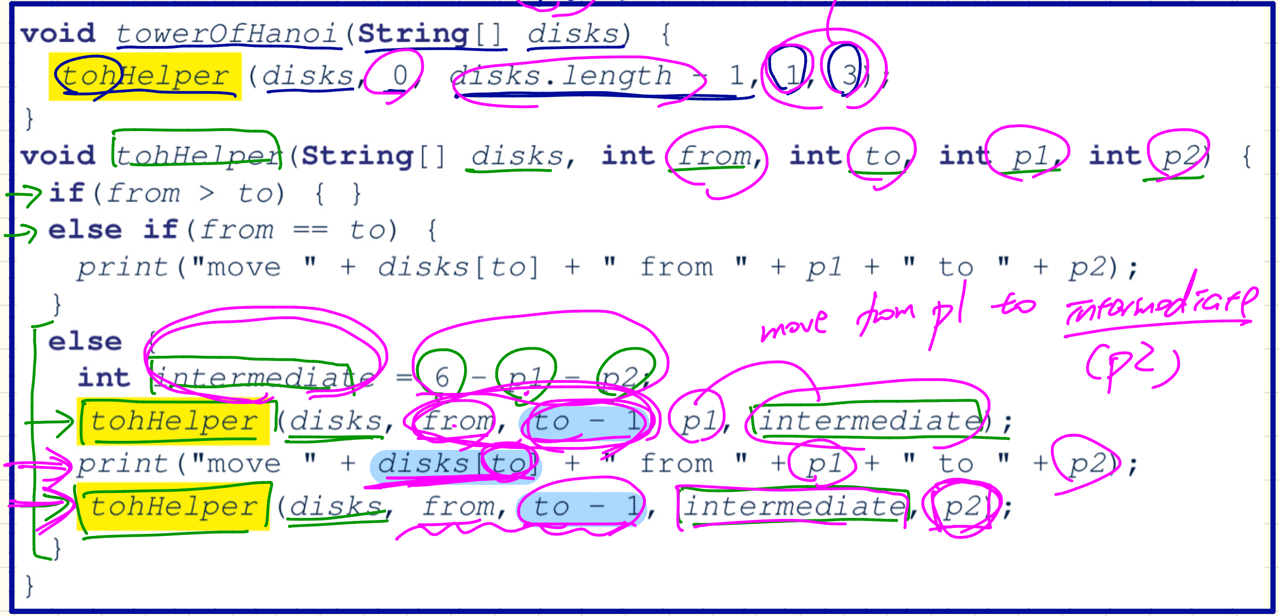
Consider 3 disks A < B < C



# Tower of Hanoi : Java

A<B<C

move from p1 to p3



```
void towerOfHanoi(String[] disks) {
  tohHelper (disks, 0, disks.length - 1, 1, 3);
}
void tohHelper(String[] disks, int from, int to, int p1, int p2) {
 if(from > to) { }
 else if(from == to) {
   print("move " + disks[to] + " from " + p1 + " to " + p2);
 }
 else {
   int intermediate = 6 - p1 - p2;
   tohHelper(disks, from, to - 1, p1, intermediate);
   print("move " + disks[to] + " from " + p1 + " to " + p2);
   tohHelper(disks, from, to - 1, intermediate, p2);
 }
}
```

move from p1 to intermediate (p2)

Say disks = {A,B,C}.
Consider towerOfHoni(disks) which calls:
tohHelper(disks, 0, disks.length - 1, 1, 3)

Monday Dec. 3

Lecture 24

# Review Sessions for Exam

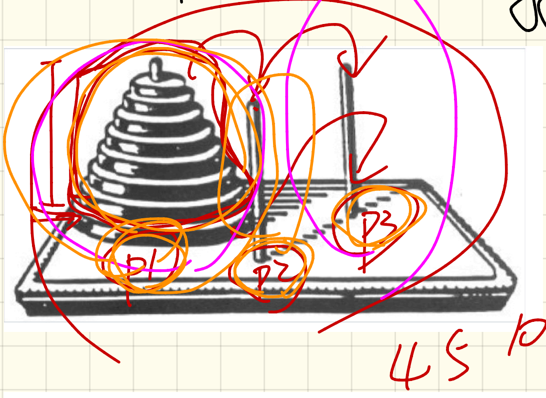
1pm ~ 3pm LAS C

Monday (Dec. 10)

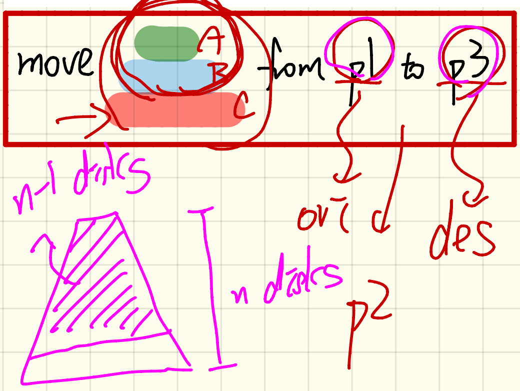
Wednesday (Dec. 12)

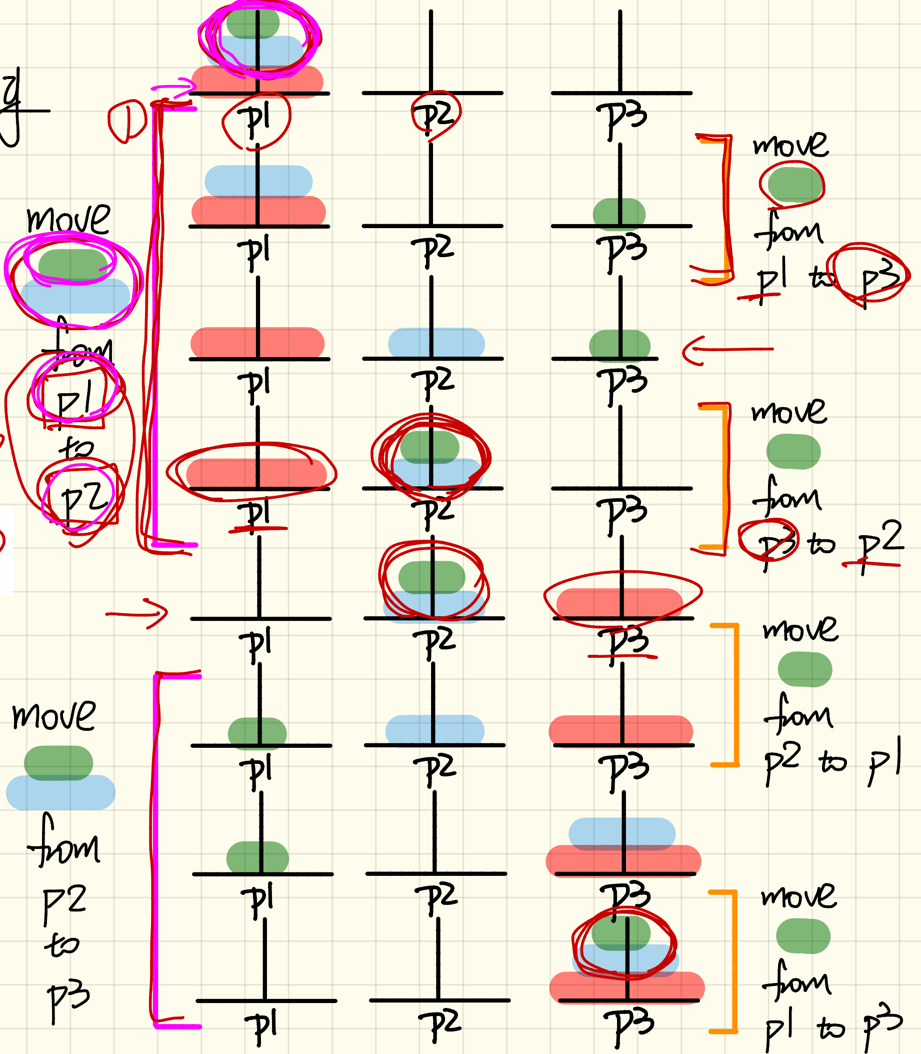
Confirm your attendance on Moodle!

# Tower of Hanoi: Strategy



Consider 3 disks A < B < C

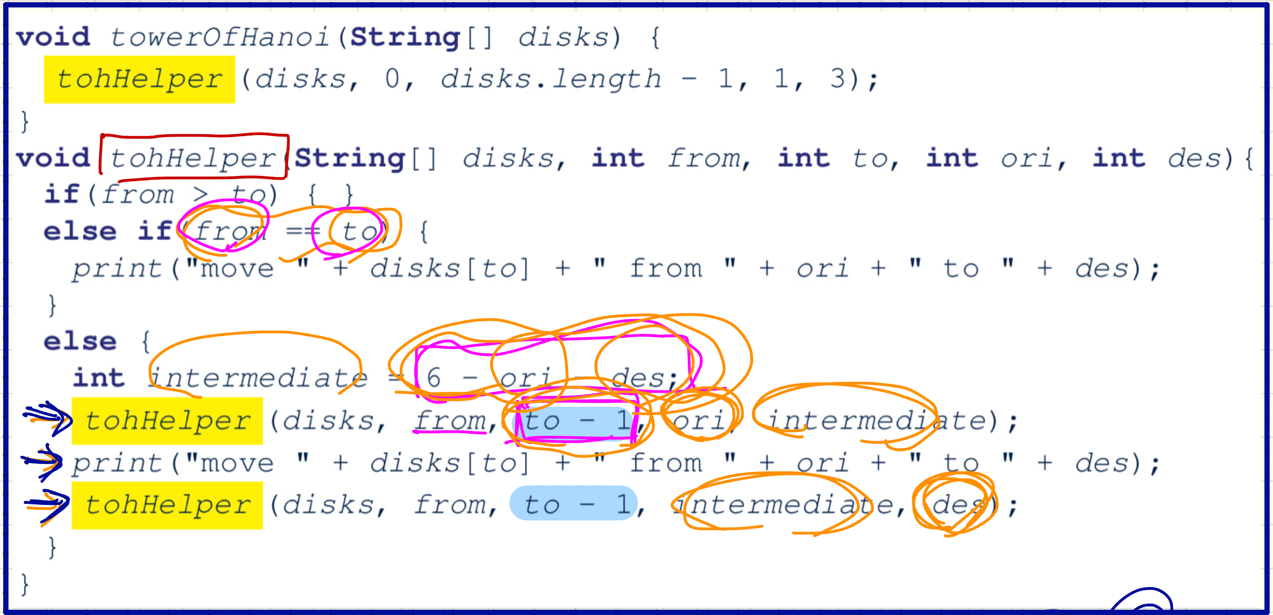




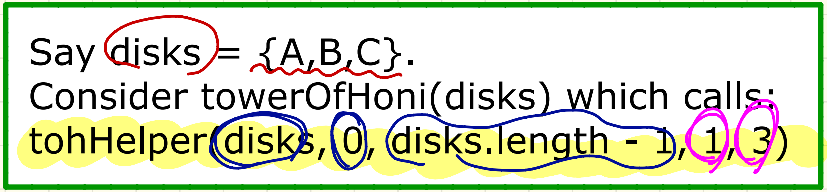
# Tower of Hanoi : Java

1 2 3

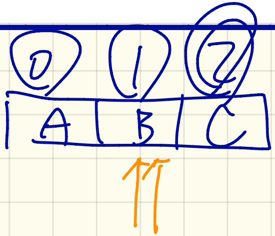
1 3 2



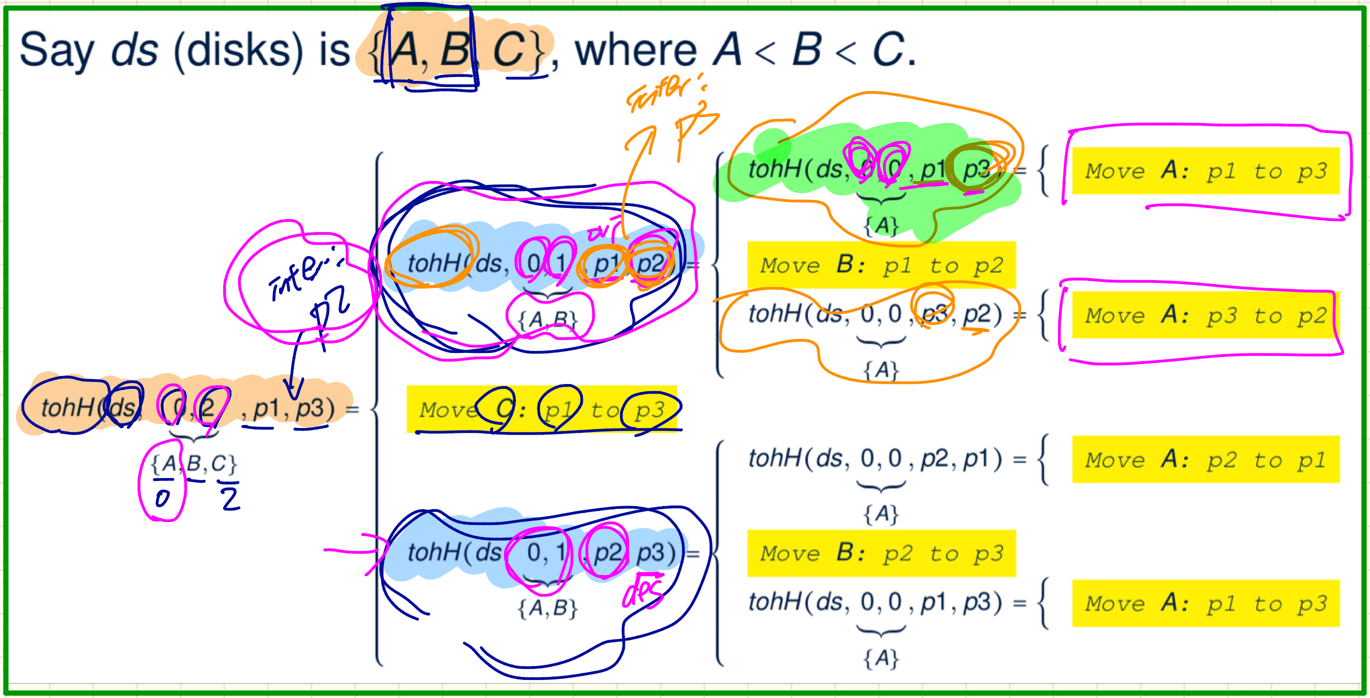
```
void towerOfHanoi(String[] disks) {
  tohHelper (disks, 0, disks.length - 1, 1, 3);
}
void tohHelper(String[] disks, int from, int to, int ori, int des){
  if(from > to) { }
  else if(from == to) {
    print("move " + disks[to] + " from " + ori + " to " + des);
  }
  else {
    int intermediate = 6 - ori - des;
    tohHelper (disks, from, to - 1, ori, intermediate);
    print("move " + disks[to] + " from " + ori + " to " + des);
    tohHelper (disks, from, to - 1, intermediate, des);
  }
}
```



Say disks = {A,B,C}.
Consider towerOfHoni(disks) which calls:
tohHelper(disks, 0, disks.length - 1, 1, 3)



# Tower of Hanoi: Tracing

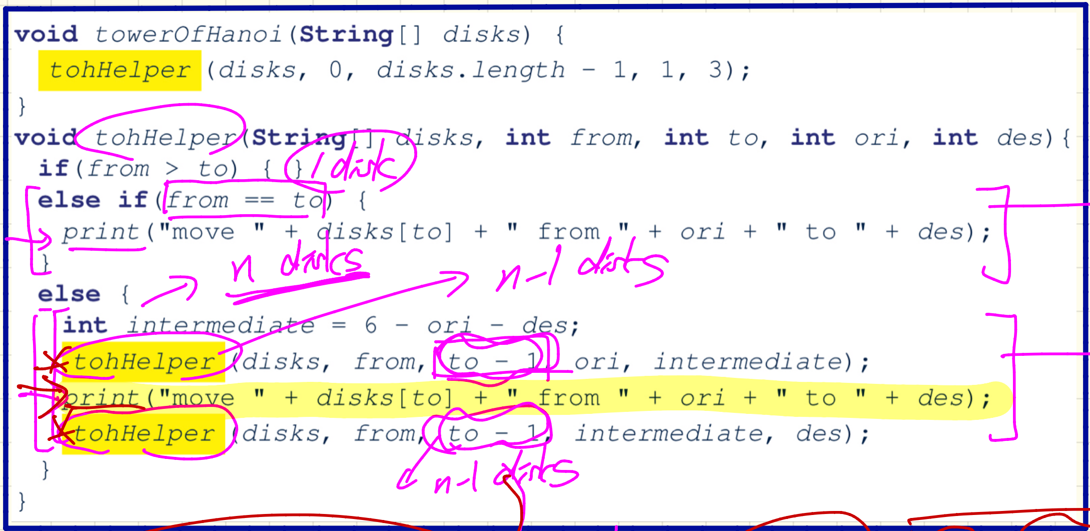


Say $ds$ (disks) is $\{A, B, C\}$, where $A < B < C$.

$$tohH(ds, \underbrace{0, 2}_{\{A,B,C\}}, p1, p3) = \begin{cases} tohH(ds, \underbrace{0, 1}_{\{A,B\}}, p1, p2) = \begin{cases} tohH(ds, \underbrace{0, 0}_{\{A\}}, p1, p3) = \{ \; \texttt{Move A: p1 to p3} \\ \texttt{Move B: p1 to p2} \\ tohH(ds, \underbrace{0, 0}_{\{A\}}, p3, p2) = \{ \; \texttt{Move A: p3 to p2} \end{cases} \\ \texttt{Move C: p1 to p3} \\ tohH(ds, \underbrace{0, 1}_{\{A,B\}}, p2, p3) = \begin{cases} tohH(ds, \underbrace{0, 0}_{\{A\}}, p2, p1) = \{ \; \texttt{Move A: p2 to p1} \\ \texttt{Move B: p2 to p3} \\ tohH(ds, \underbrace{0, 0}_{\{A\}}, p1, p3) = \{ \; \texttt{Move A: p1 to p3} \end{cases} \end{cases}$$

# Tower of Hanoi: Running Time

$T(1) \rightarrow n-(n-1)$



```
void towerOfHanoi(String[] disks) {
  tohHelper(disks, 0, disks.length - 1, 1, 3);
}
void tohHelper(String[] disks, int from, int to, int ori, int des){
  if(from > to) { }
  else if(from == to) {
    print("move " + disks[to] + " from " + ori + " to " + des);
  }
  else {
    int intermediate = 6 - ori - des;
    tohHelper(disks, from, to - 1, ori, intermediate);
    print("move " + disks[to] + " from " + ori + " to " + des);
    tohHelper(disks, from, to - 1, intermediate, des);
  }
}
```

1 disk

base case

n disks

n-1 disks

recursive

n-1 disks

formulate

$T(1) = 1$

$T(n) = 2 * T(n - 1) + 1$

$$T(n) = 2 * T(n-1) + 1$$

$$= 2*(2*T(n-2)+1)+1$$

$$= 2*(2*(2*T(n-3)+1)+1)+1$$

$$= \dots$$

$$= \overbrace{2*(2*(\dots}^{n-1} \overset{n-(n-1)}{T(1)} \overbrace{+1)+1\dots)+1}^{n-1}$$

$O(2^n) \Leftarrow 2^{n-1} + (n-1)$

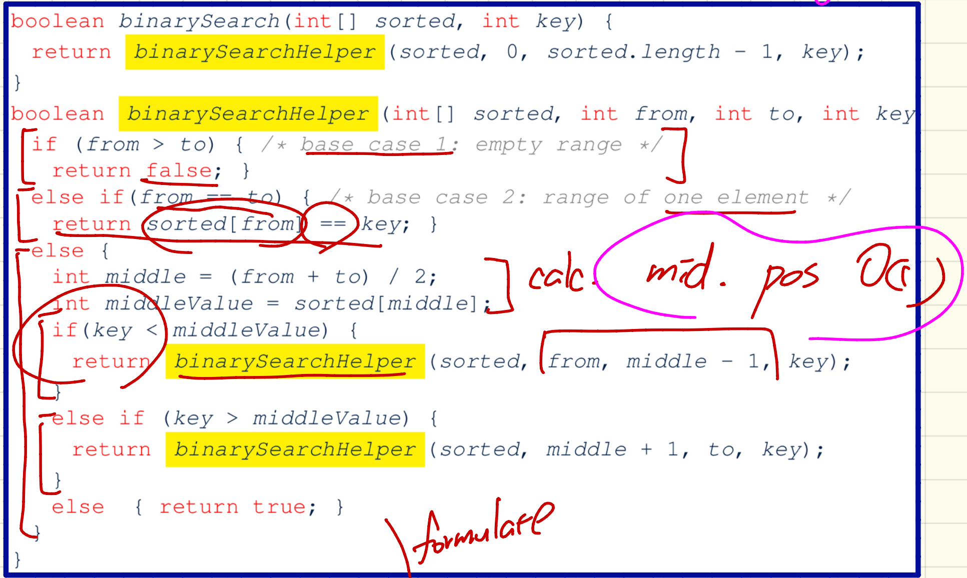
# Binary Search: Running Time

1024 8 4

assume $n = 2^i$

$\log 1024$

$1024 = 2^{10}$ $\quad n = 2^{\log n}$



```
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; }
  }
}
```

calc. mid. pos $O(1)$

formulate

$T(0) = 1$

$T(1) = 1$

$T(n) = T(n/2) + 1$

$n/4$

mid. pos.

L or R

$O(\log n)$ ←

$1 + \log n$

$T(n) = T\left(\frac{n}{2}\right) + 1$

$\frac{n}{2^1}$ $\quad = \left(T\left(\frac{n}{4}\right) + 1\right) + 1$ $\quad 2^2$

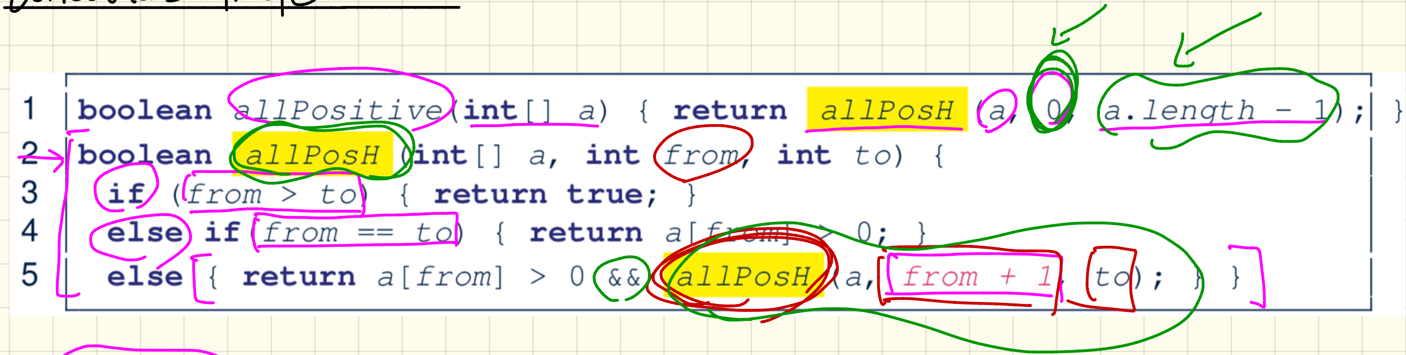
$\frac{n}{2^2}$ $\quad = \left(\left(T\left(\frac{n}{8}\right) + 1\right) + 1\right) + 1$

$= \dots$

$= T(1) + 1 + \dots + 1$

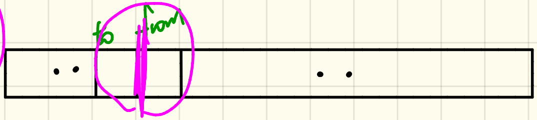
$\frac{n}{2^?}$ $\quad 1$ $\quad \log n$

# Correctness Proofs: Ideas

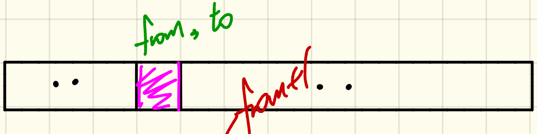


```
boolean allPositive(int[] a) { return allPosH (a, 0, a.length - 1); }
boolean allPosH (int[] a, int from, int to) {
  if (from > to) { return true; }
  else if (from == to) { return a[from] > 0; }
  else { return a[from] > 0 && allPosH (a, from + 1, to); } }
```

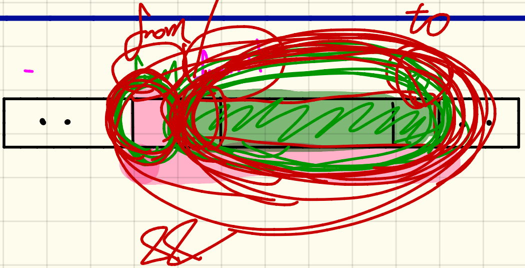
Base Case:
Empty Array



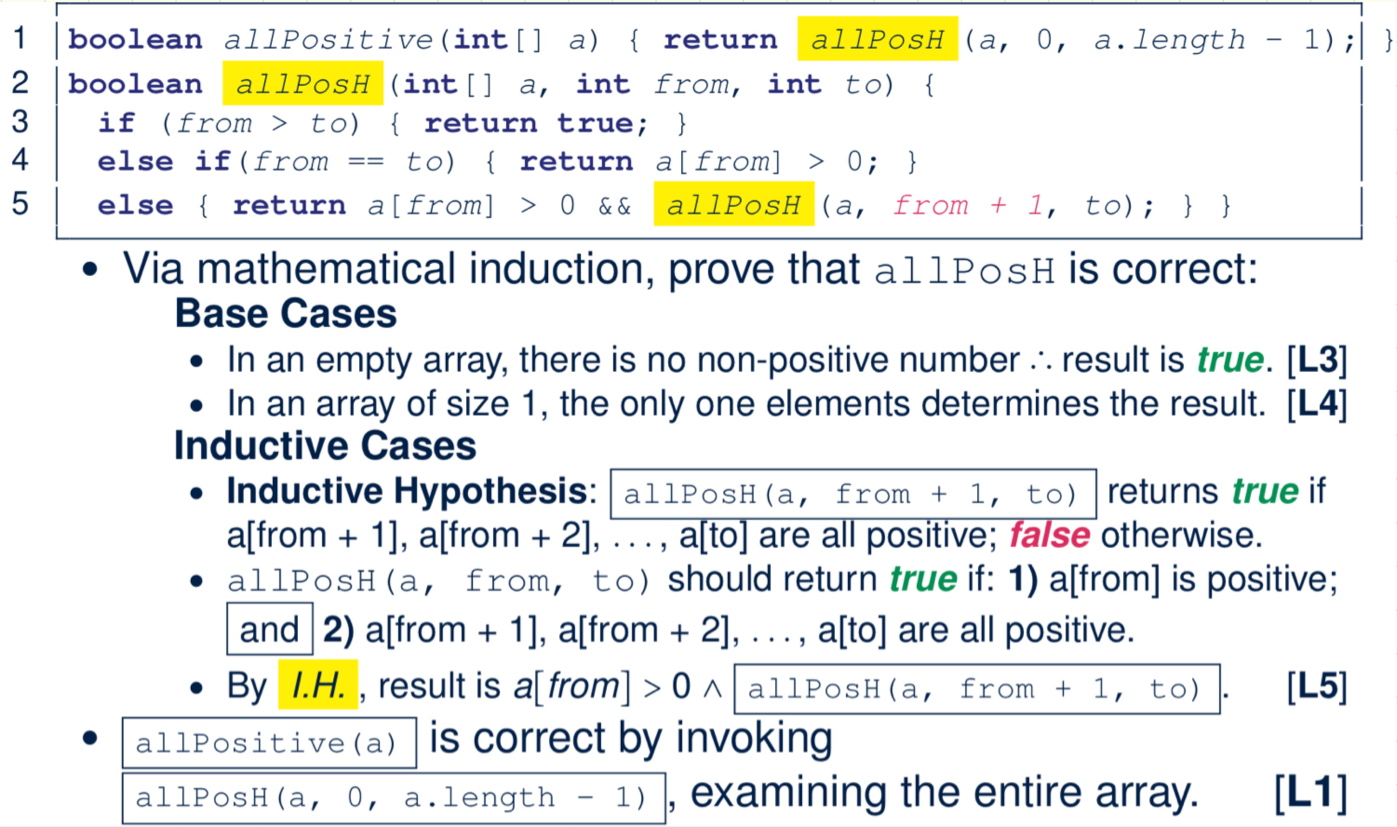
Base Case:
Array of size 1



Recursive Case:



# 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:

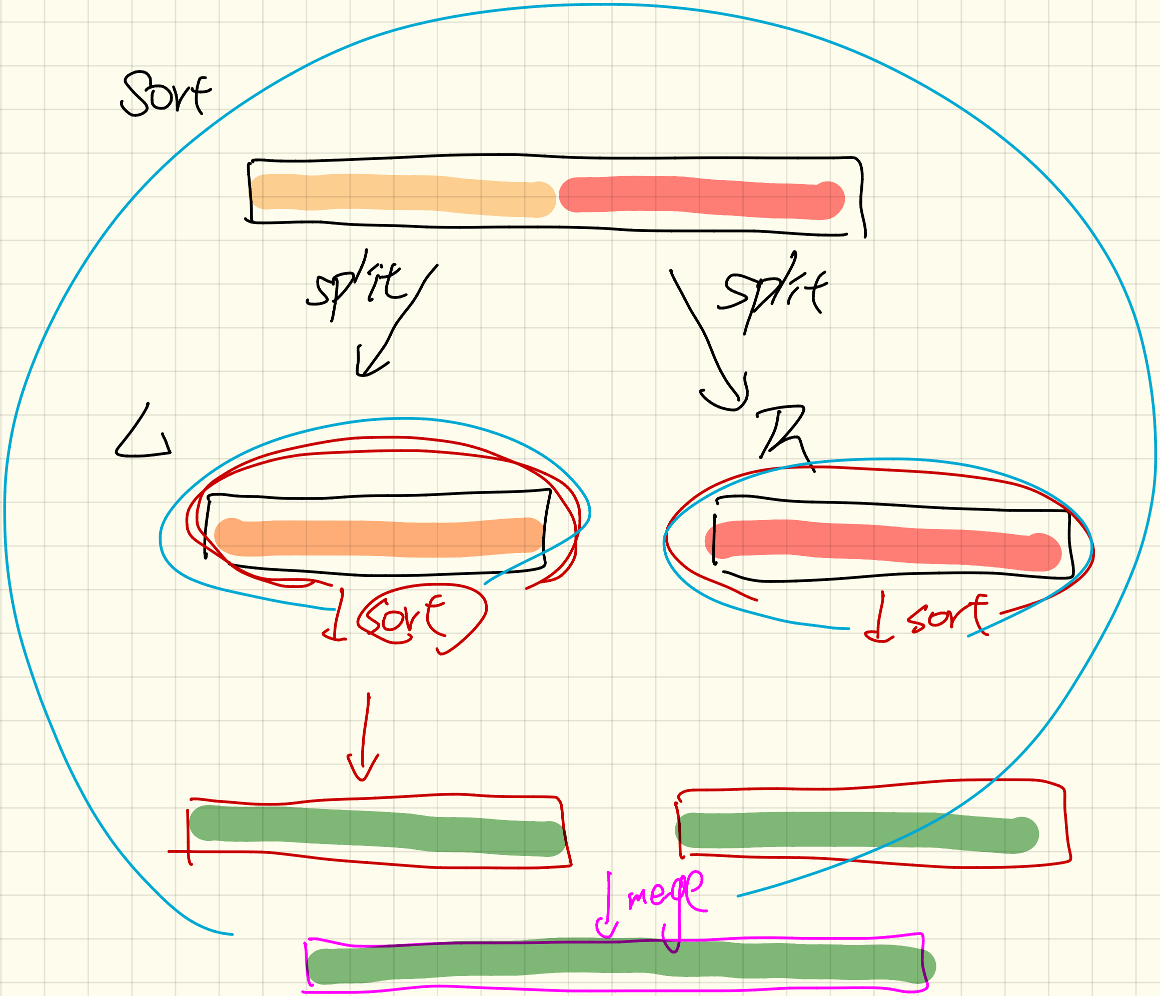
  **Base Cases**

  - 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]**

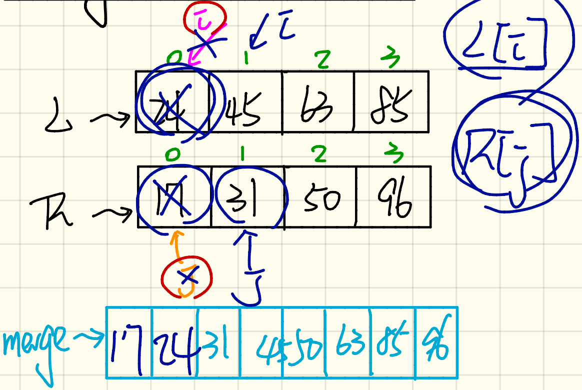
  **Inductive Cases**

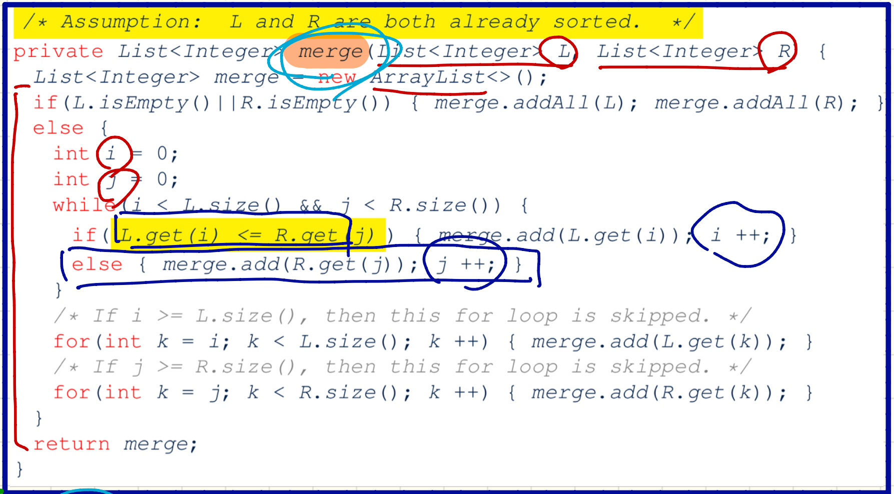
  - **Inductive Hypothesis**: `allPosH(a, from + 1, to)` returns ***true*** if a[from + 1], a[from + 2], . . . , a[to] are all positive; ***false*** otherwise.
  - `allPosH(a, from, to)` should return ***true*** if: **1)** a[from] is positive; and **2)** a[from + 1], a[from + 2], . . . , a[to] are all positive.
  - By *I.H.*, result is $a[from] > 0 \land$ `allPosH(a, from + 1, to)`. **[L5]**

- `allPositive(a)` is correct by invoking `allPosH(a, 0, a.length - 1)`, examining the entire array. **[L1]**

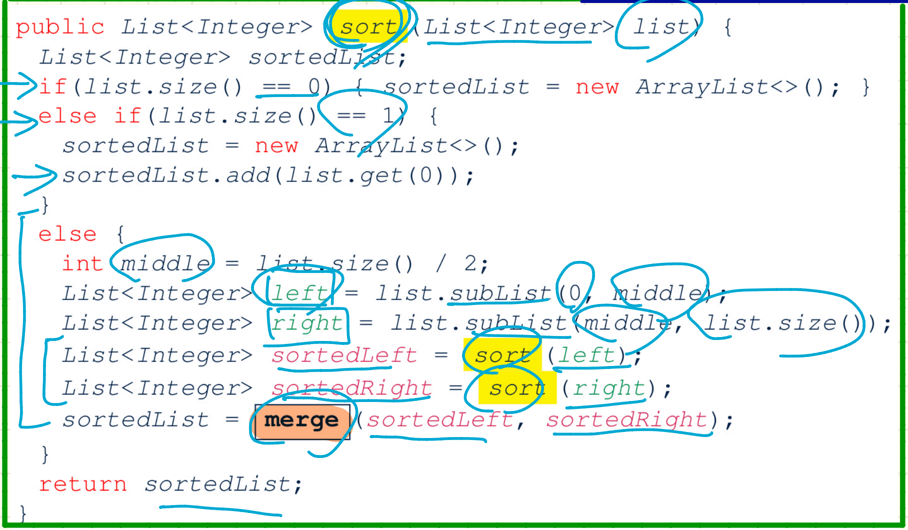

Sort
split
split
L
R
sort
sort
merge

# Merge Sort : 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;
}
```



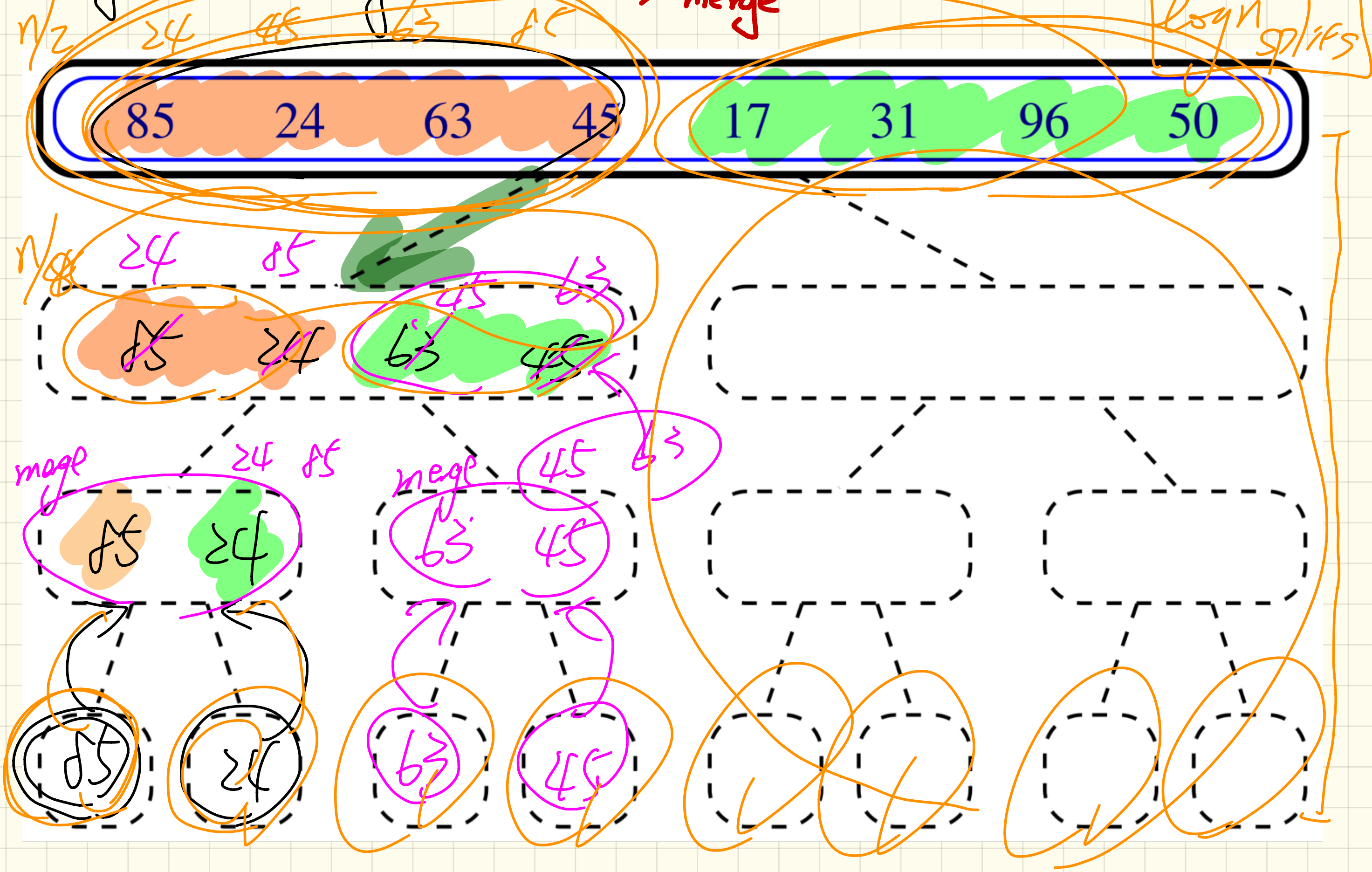
```
public List<Integer> sort(List<Integer> list) {
 List<Integer> sortedList;
 if(list.size() == 0) { sortedList = new ArrayList<>(); }
 else if(list.size() == 1) {
   sortedList = new ArrayList<>();
   sortedList.add(list.get(0));
 }
 else {
   int middle = list.size() / 2;
   List<Integer> left = 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;
}
```

# Merge Sort : Tracing

→ split

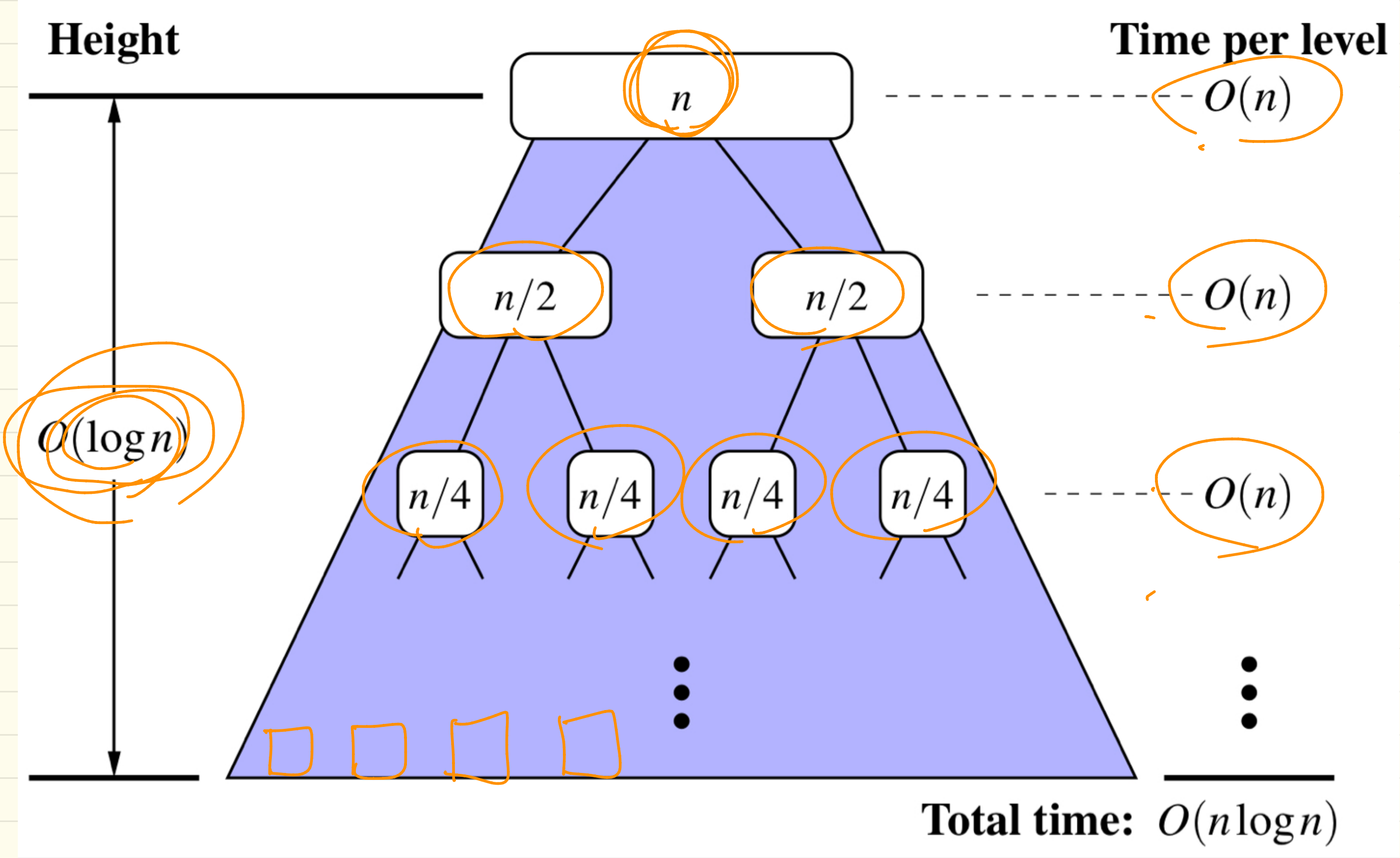
→ merge

$n \rightarrow n/2 \rightarrow n/4 \rightarrow .. \; 1$



# Merge Sort : Running Time

$n \cdot \log n$



End of Notes

All the Best!