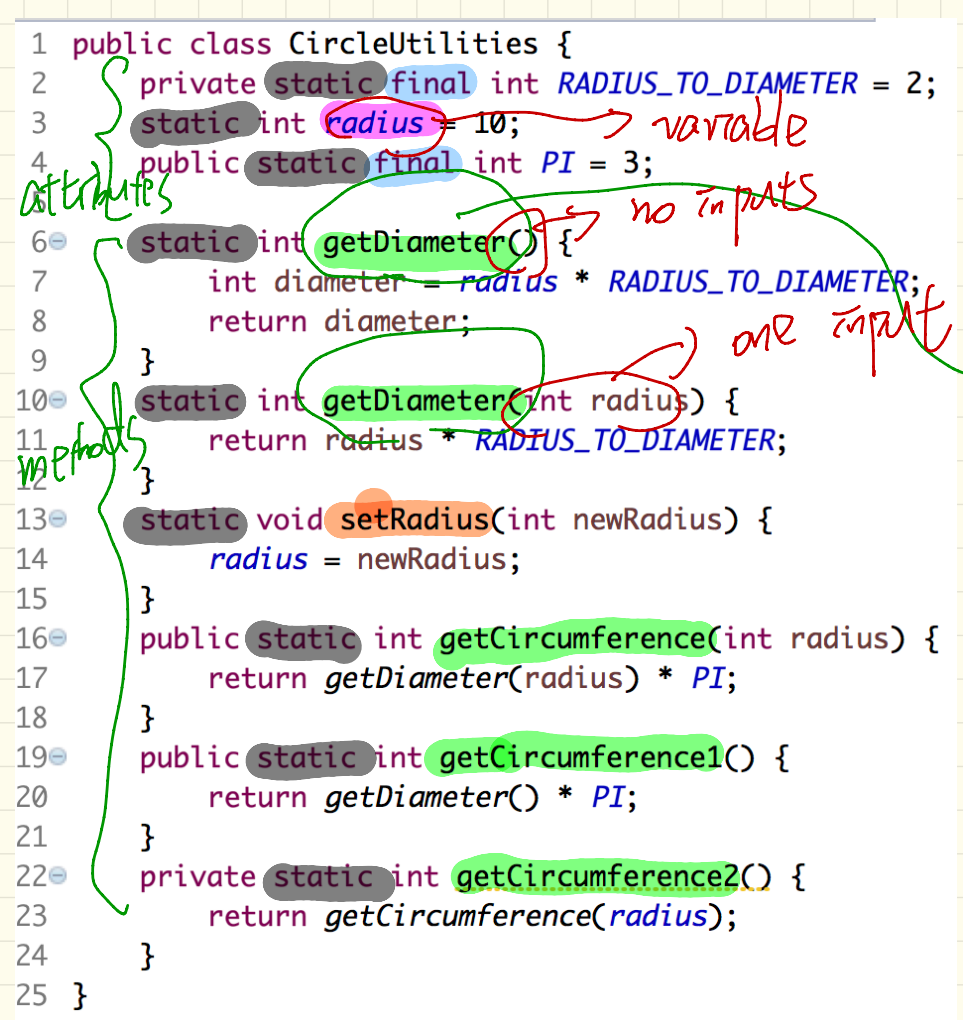
EECS 2030 A Fall 2017

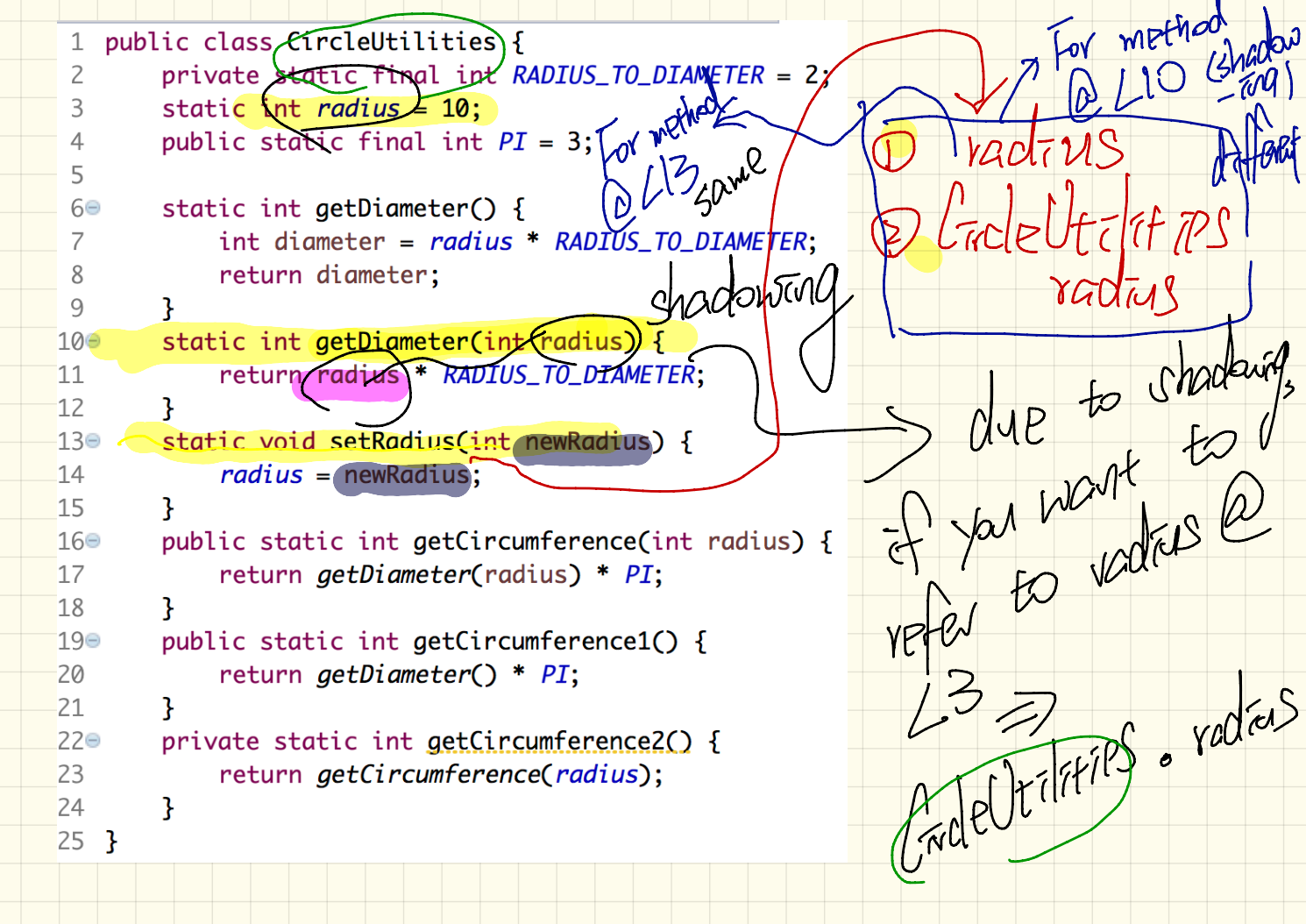
Advanced Object Oriented Programming

Tuesday September 12

Lecture 2


public class CircleUtilities {
private static final int RADIUS_TO_DIAMETER = 2;
static int radius = 10;
public static final int PI = 3;
static int getDiameter() {
int diameter = radius * RADIUS_TO_DIAMETER;
return diameter;
}
static int getDiameter(int radius) {
return radius * RADIUS_TO_DIAMETER;
}
static void setRadius(int newRadius) {
radius = newRadius;
}
public static int getCircumference(int radius) {
return getDiameter(radius) * PI;
}
public static int getCircumference1() {
return getDiameter() * PI;
}
private static int getCircumference2() {
return getCircumference(radius);
}
}
attributes
methods
variable
no inputs
one input
methods :
accessor : return non-void
mutator : return void
We overload the method getDiame. with different parameter lists
input

```java
public class CircleUtilities {
    private static final int RADIUS_TO_DIAMETER = 2;
    static int radius = 10;
    public static final int PI = 3;

    static int getDiameter() {
        int diameter = radius * RADIUS_TO_DIAMETER;
        return diameter;
    }
    static int getDiameter(int radius) {
        return radius * RADIUS_TO_DIAMETER;
    }
    static void setRadius(int newRadius) {
        radius = newRadius;
    }
    public static int getCircumference(int radius) {
        return getDiameter(radius) * PI;
    }
    public static int getCircumference1() {
        return getDiameter() * PI;
    }
    private static int getCircumference2() {
        return getCircumference(radius);
    }
}
```



```java
public class CircleUtilities {
    private static final int RADIUS_TO_DIAMETER = 2;
    static int radius = 10;
    public static final int PI = 3;

    static int getDiameter() {
        int diameter = radius * RADIUS_TO_DIAMETER;
        return diameter;
    }
    static int getDiameter(int radius) {
        return radius * RADIUS_TO_DIAMETER;
    }
    static void setRadius(int newRadius) {
        radius = newRadius;
    }
    public static int getCircumference(int radius) {
        return getDiameter(radius) * PI;
    }
    public static int getCircumference1() {
        return getDiameter() * PI;
    }
    private static int getCircumference2() {
        return getCircumference(radius);
    }
}
```

CU. getCircumference(5)

shadowing

helper methods

A method is a block of code which can be reused by referring to its name.

```java
public class CircleUtilities {
    private static final int RADIUS_TO_DIAMETER = 2;
    static int radius = 10;
    public static final int PI = 3;

    static int getDiameter() {
        int diameter = radius * RADIUS_TO_DIAMETER;
        return diameter;
    }
    static int getDiameter(int radius) {
        return radius * RADIUS_TO_DIAMETER;
    }
    static void setRadius(int newRadius) {
        radius = newRadius;
    }
    public static int getCircumference(int radius) {
        return getDiameter(radius) * PI;
    }
    public static int getCircumference1() {
        return getDiameter() * PI;
    }
    private static int getCircumference2() {
        return getCircumference(radius);
    }
}
```

CU.getCircumference(5)
= CU.getParameter(5)*3
= 5*2*3
= 30.

argument value for replacing parameter radius

parameter

fac(x) = x*(x-1)*

fac(5) fac(4) ...

modifiers

1. Visibility { private / package / class / public

2. Constant/Variable { double / final double

variable

private

double myPI = 3.14;

myPI = 6.28;

final double PI = 3.14;

PI = 6.28;

PI's Constant

final static int FOO = 2

FOO = 4;

you must access this using PI's class name.

```
class MyMath {
    public static int Foo = 3;
}
```

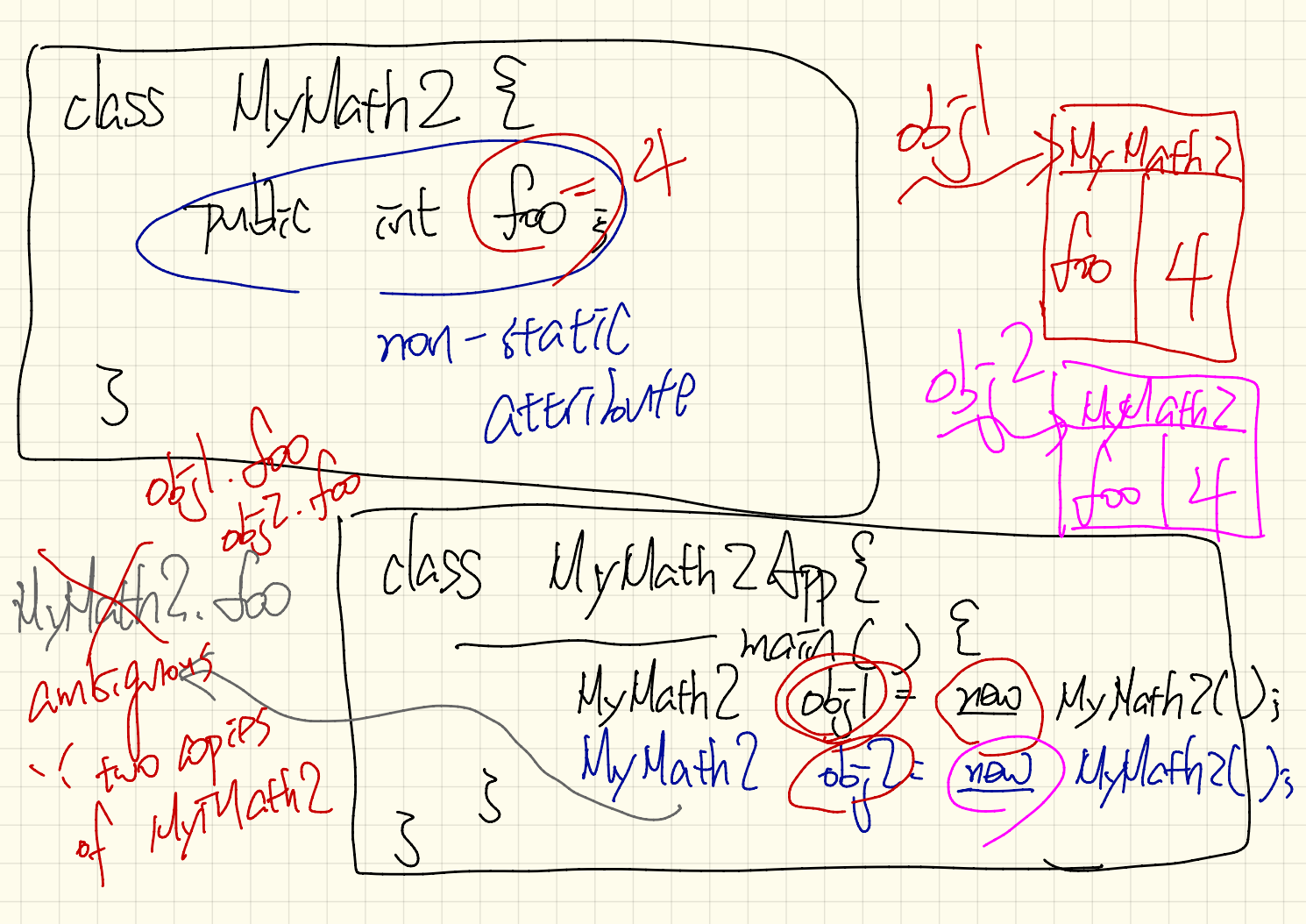
It's guaranteed only one copy of Foo will exist at runtime

We can only use the class name to access this attribute if it's static

```
class MyMathUser {
    ____ main(____) {
        MyMath.Foo = 4;
    }
}
```

MyMath.Foo

MyMath	
Foo	3


class MyMath2 {
public int foo = 4
non-static attribute
}
obj1
MyMath2
foo
4
obj2
MyMath2
foo
4
obj1.foo
obj2.foo
MyMath2.foo
ambiguous
:(two copies of MyMath2
class MyMath2App {
main() {
MyMath2 obj1 = new MyMath2();
MyMath2 obj2 = new MyMath2();
}
}

Thursday Sept. 14

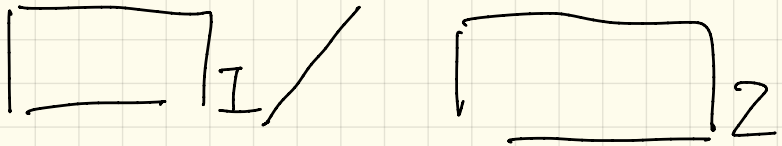
Lecture 3

office hours

13:30 – 15:30

M, Tue Th.

Number division in Java



If both  and 

are integers, the result.

is quotient.

Otherwise, result with precision.

print(8/2) 4

print(8.0/2) 4.0

print(9/2) 4

print(9/2.0) 4.5

(2+3)*4

---

int i = 9;

int j = 2;

print(i/j) 4

① print((double)i/j) 4.5

② print(((double)i)/j) 4.5

Step1: (double)i → 9.0

Step2: 9.0/2 = 4.5

2 / 3 = 0

2.0 / 3

---

int i / int j — floating

(double) (i / j)

i.0 / j

(double) i / j

((double) i) / j

```java
public class CircileUtilitesApplication {
    public static void main(String[] args) {
        System.out.println("Initial radius of CU: " + CircleUtilities.radius);
        int d1 = CircleUtilities.getDiameter();
        System.out.println("d1 is: " + d1);
        System.out.println("c1 is: " + CircleUtilities.getCircumference1());
        System.out.println("======");
        System.out.println("d2 is: " + CircleUtilities.getDiameter(20));
        System.out.println("c2 is: " + CircleUtilities.getCircumference(20));
        System.out.println("======");
        System.out.println("Change the radius of CU to 30...");
        CircleUtilities.setRadius(30);
        System.out.println("======");
        d1 = CircleUtilities.getDiameter();
        System.out.println("d1 is: " + d1);
        System.out.println("c1 is: " + CircleUtilities.getCircumference1());
        System.out.println("======");
        System.out.println("d2 is: " + CircleUtilities.getDiameter(20));
        System.out.println("c2 is: " + CircleUtilities.getCircumference(20));
    }
}

```

```java
public class CircleUtilities {
    private static final int RADIUS_TO_DIAMETER = 2;
    static int radius = 10;
    public static final int PI = 3;

    static int getDiameter() {
        int diameter = radius * RADIUS_TO_DIAMETER;
        return diameter;
    }
    static int getDiameter(int radius) {
        return radius * RADIUS_TO_DIAMETER;
    }
    static void setRadius(int newRadius) {
        radius = newRadius;
    }
    public static int getCircumference(int radius) {
        return getDiameter(radius) * PI;
    }
    public static int getCircumference1() {
        return getDiameter() * PI;
    }
    private static int getCircumference2() {
        return getCircumference(radius);
    }
}
```

print(CU.radius) 10

int d1 = CU.getDia();

println(d1); 20

println(CU.getCir1()) 60

println(CU.getDia(20)) 40

20 * 2

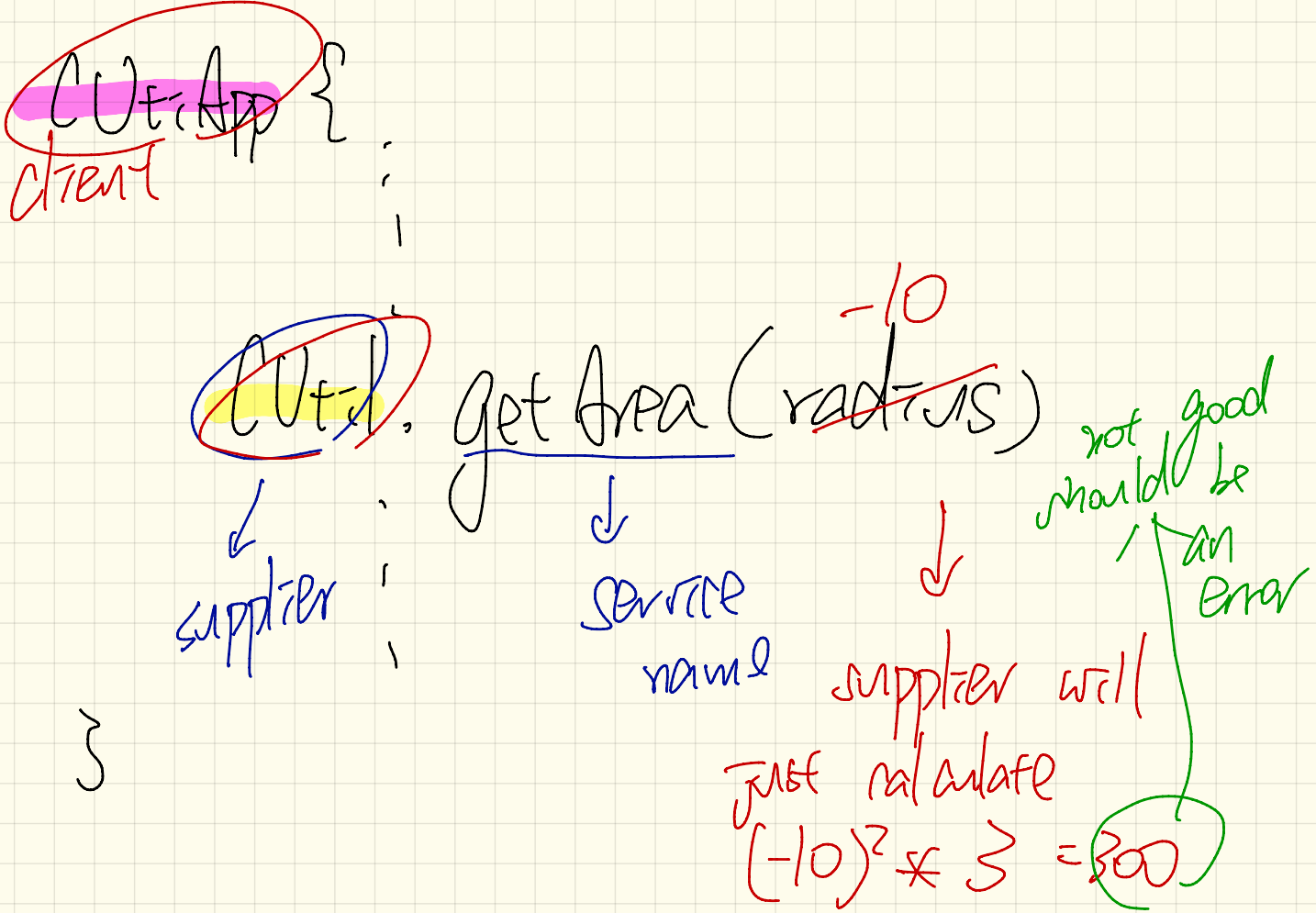
println(CU.getCir(20)) 120

20 * 2 * 3

CU	
R_I_D	2
PI	3
radius	10
⋮	

20

d1 : int


CUtilApp {
client
CUtil.
getArea(rad-JS)
-10
supplier
service name
supplier will just calculate
(-10)² * 3 = 300
not good should be an error
}

```
class MyUtil {
    static void m(int x) {
        ...
    }
}

class MyUtilApp {
    MyUtil.m(23.4);
}
```

(int) 23.4

compile

Counter:

$0 \leq value \leq ?$ Req.

```
void increment() {
    value++;
}
```

Req: When should we throw a IllegalArgumentException?

① value > MAX/3

inappropriate if value = MAX/3

Section B

Guest Lecture I

MAX = 3 MIN = 0

void increment() {

if (counter > MAX) {

throw _____

}

else {

counter ++;

}

Assume counter is currently MAX

reaching this point means:

!(counter > MAX)

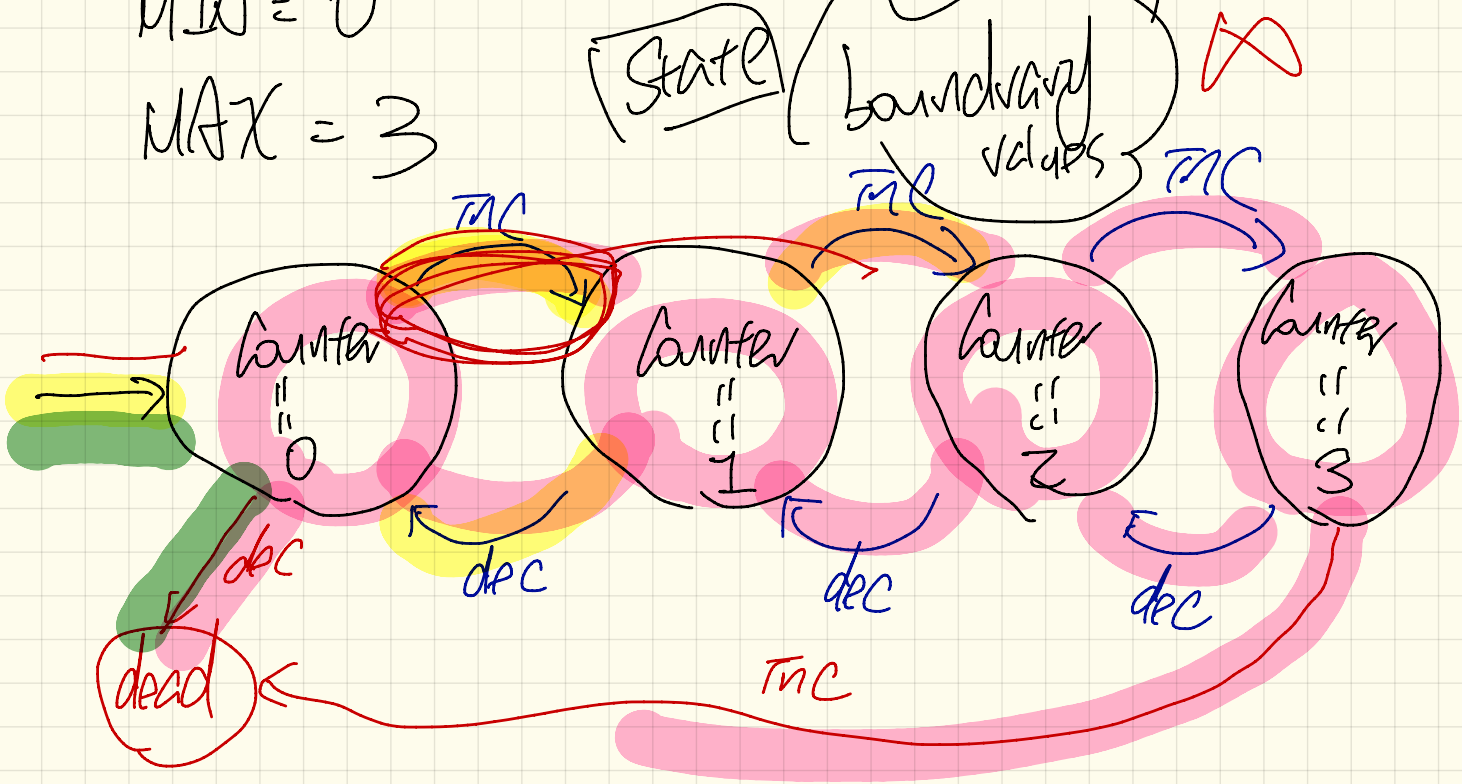
≡ counter ≤ MAX

Lifetime of a counter

Q. How many tests (paths)?

MIN = 0

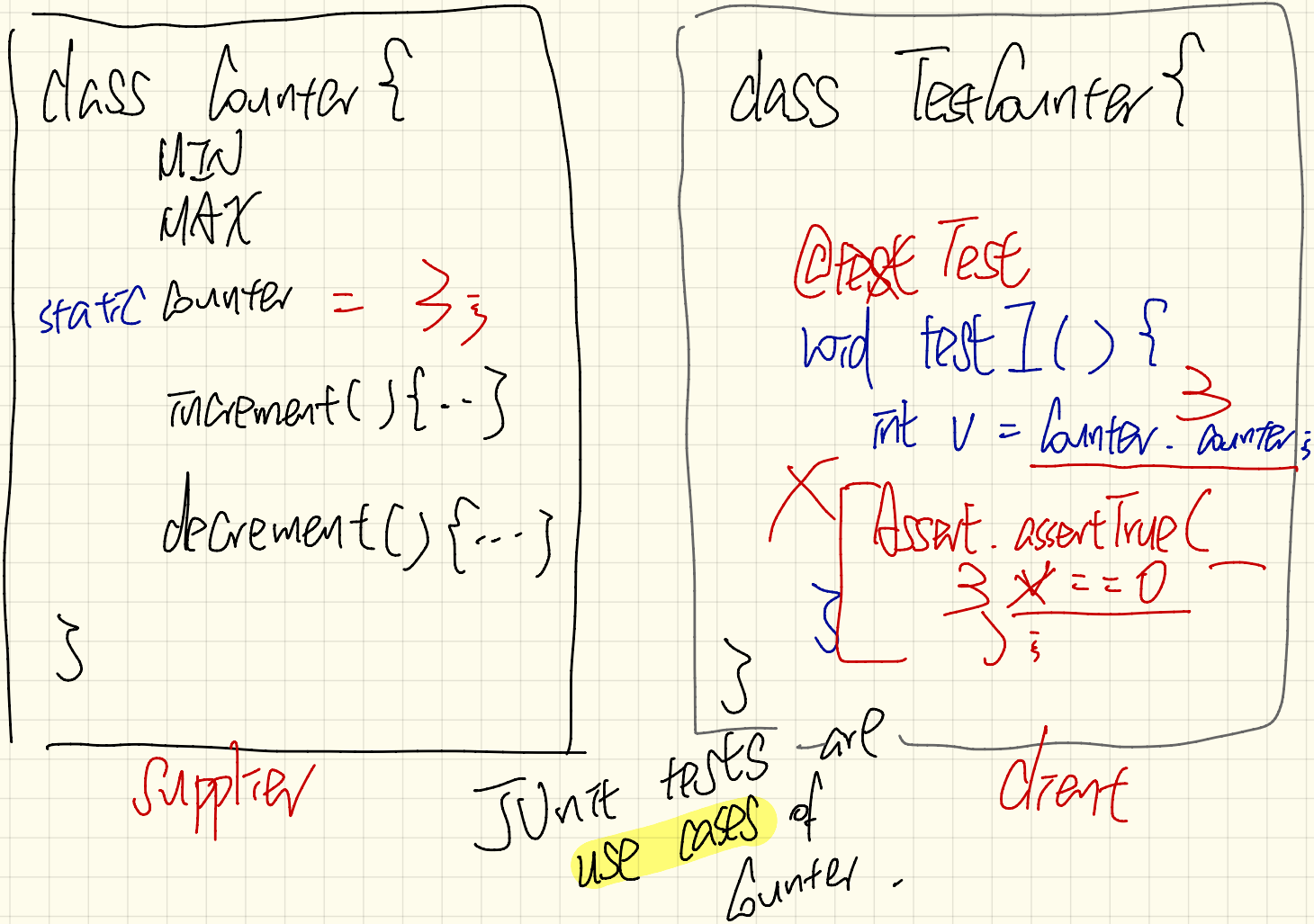
MAX = 3



Correctness of your code

efficiency

my code is "correct"
if, to my best knowledge,
all scenarios have been
tested and passed.


class Counter {
MIN
MAX
static Counter = 3;
increment() {...}
decrement() {...}
}
class TestCounter {
@Test Test
void test1() {
int v = Counter.3counter;
Assert.assertTrue(
3x == 0
;
}
}
Supplier
JUnit tests are use cases of Counter.
client

September 19

Tuesday

Lecture 4

② ✗ e.g. counter == 1 . 1 <= 3 → T

↳ exception is thrown.

```
increment(int value) {
    if ( ① counter > MAX
         ② counter <= MAX
         ③ counter >= MAX ) {
        throw
    } else {
        # counter ++;
    }
}
```

else : counter ≤ MAX

counter < MAX

③ ✓

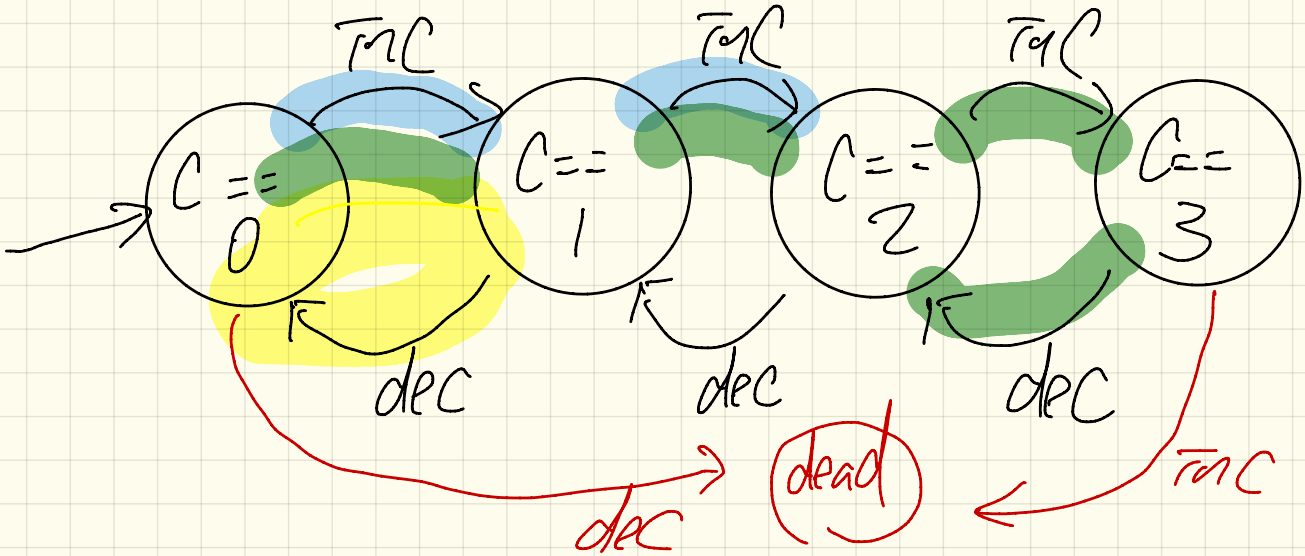
① ✗ : if counter == MAX
MAX > MAX → False

State Machine

How many test cases?
How many paths?

Transitions (mutator method calls)

[state] values of attributes



Criteria for judging your program

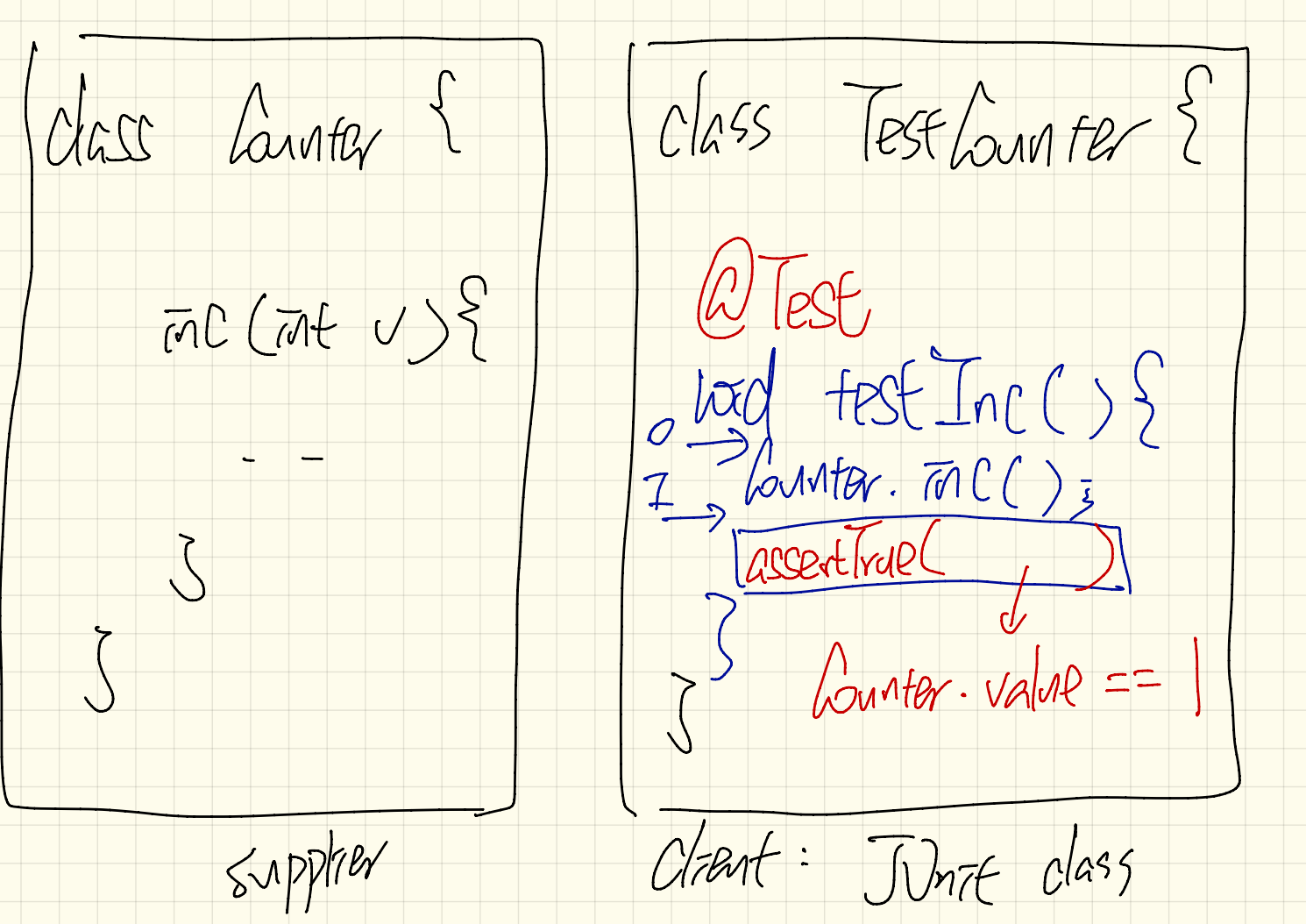
① Correctness

② Efficiency.

Opt A. ① ∧ ¬②

Opt B. ¬① ∧ ②

define a reasonably complete set of tests.

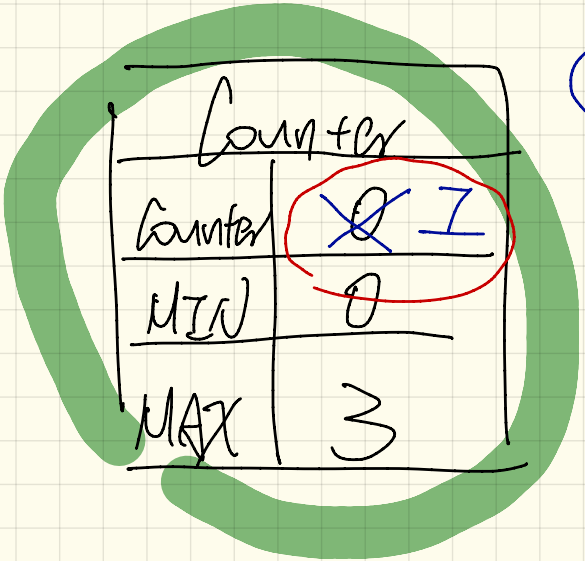

class Counter {
inc(int v){
}
}
supplier
class TestCounter {
@Test
void testInc(){
Counter.inc();
assertTrue(
Counter.value == 1
}
}
0
1
client: JUnit class

Now we want to test an abnormal use of the class, for which we expect a precondition violation to occur.

```
@Test
void testDecFromZero() {
    try {
        counter.value = MIN;
        // init val is 0
        counter.dec(); // expect exception
        // We do not expect to reach this line
        fail("no precondition violation occurred");
    } catch (IAE e) {
        // precondition violation occurred as expected
    }
}
```

breakpoints and debugger

↳ slow down your program and execute line by line-



① testOneIncrement()

② testDecFromZero()

When starting each test method we should reset the attributes to their initial values to make sure it's a fresh start.

Wednesday Sept. 20

Guest Lecture 2

When does a test method succeed?

@Test

- No exceptions thrown
- No assertions failure

} Success

Otherwise (exception or assertion failure) } failure

For utility classes, where everything is static, the order in which you run the test methods matters.

Visualizing a OC

testIncrementTwice

increment 0 → 1

1 → 2

testIncrementOnce ✓

↳ starting value of counter is no longer 0 → 2

Counter	
MIN	0
MAX	3
value	0 ~~1~~ 2

Test for abnormal scenario

```
@Test
void testDecFromZero() {
    try {
        counter.value = MIN;
        counter.decrement();
        fail("IAE not thrown");
    } catch (IAE e) {
        /* IAE thrown as expected, do nothing */
    }
}
```

expect IAE thrown →

IAE not thrown → fail

breakpoints

debugger.

↳ put program execution on slow motion so you can examine variables at intermediate steps.

class vs. object

template instance

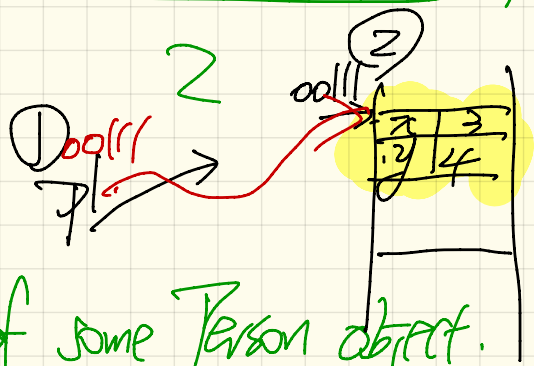
```
class Point{
};
```

```
class PointApp{
    ——— main(...){
        Point p1 = new Point(2,3);
        Point p2 = new Point(3,4);
    }
}
```

template:
every Point instance must have x and y.

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

1 3 2



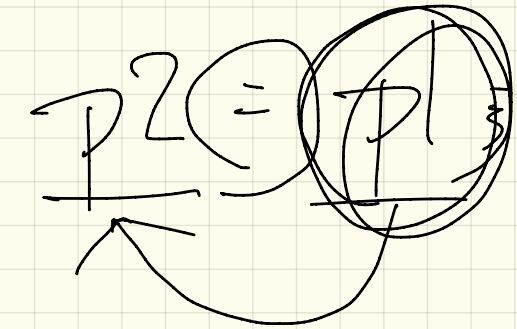
(1) Declare a variable *p1.
p1 can store the address of some Person object.

(2) Allocate space in memory and create a Point with x == 3, y == 4.

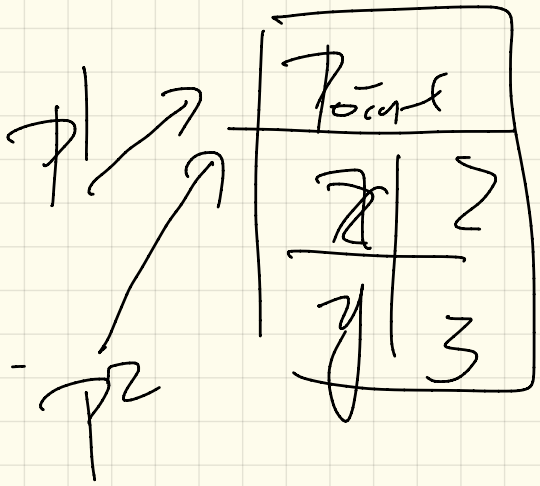
(3) Store the address of the new Point object in p1.

Point p1 = new Point(2, 3);

Point p2 = p1;



aliasing



Thursday Sept. 21

Lecture 5

Why OO

template

Observe

p1 . (x1, y1)
2 3

p2 . (x2, y2)
2 -3

Model

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

Execute

p1 → Point: x | 2, y | 3

p2 → Point: x | 2, y | -3

p1 p2

context object

p1.getY()

method

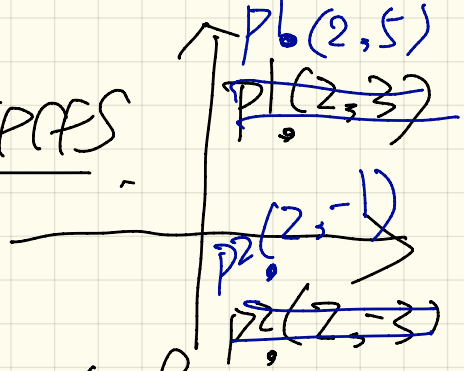
dot notation

p1.moveUp(2)

p2.getY()

p2.moveUp(2)

point objects



3

calling the same method on different context objects may give you different results.

-3

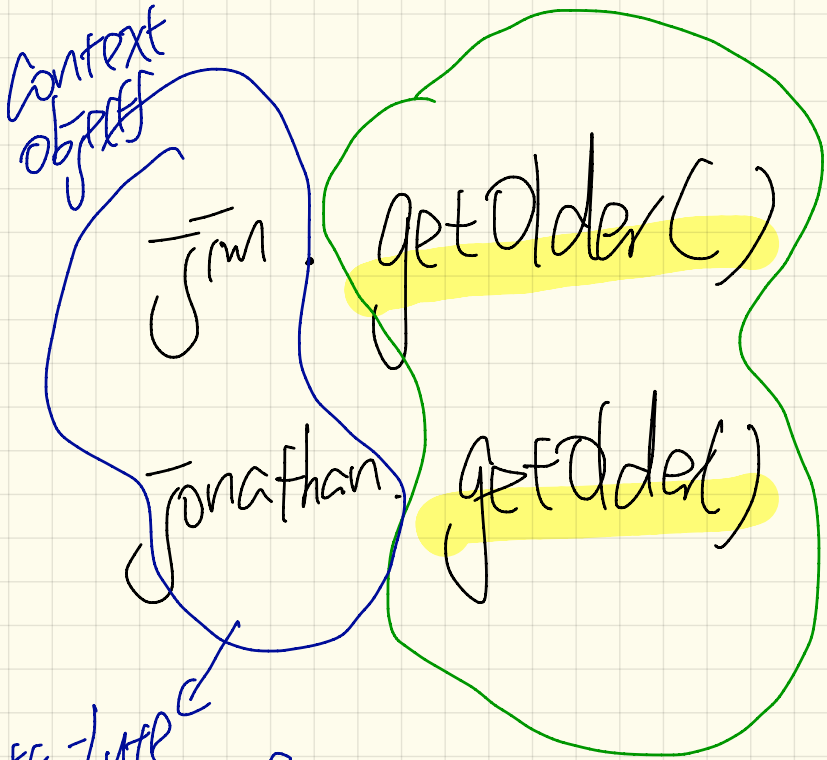
Given some natural language requirements,

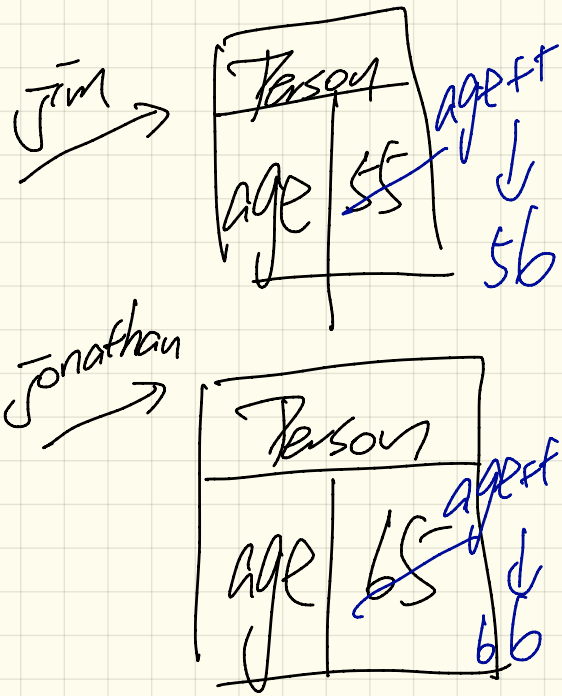
we can:

1. Identify nouns as either classes or attributes

2. Identify verbs as methods.

- change attributes
- ask something about attributes



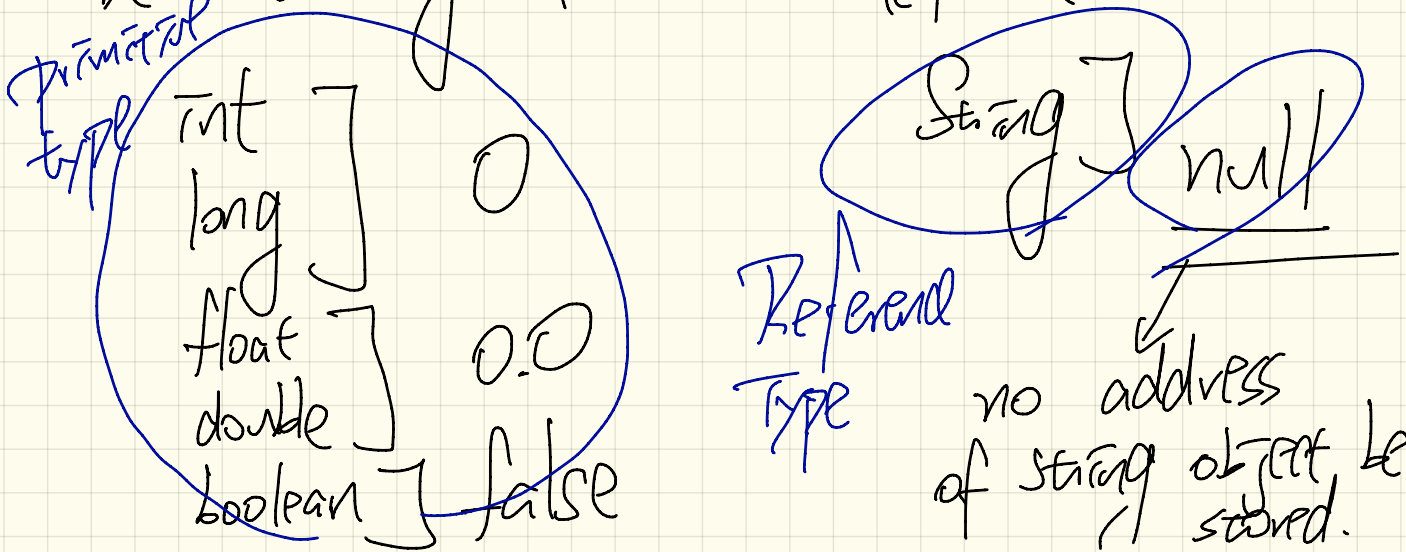


These two method calls share the same definition:

```
age++;
```

default values

When uninitialized, variables can be assigned their default values



```
class Point {
    double x;
    double y;
    Point() {
        // default values
    }
}
```

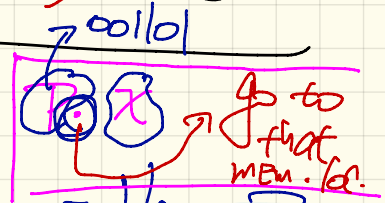
③ Store 00110l into P.

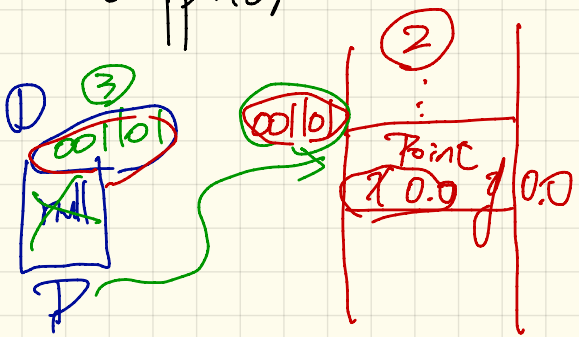
Supplier

```
class PointApp {
    ——— main(--) {
        Point p = new Point();
    }
}
```

① ③ ②

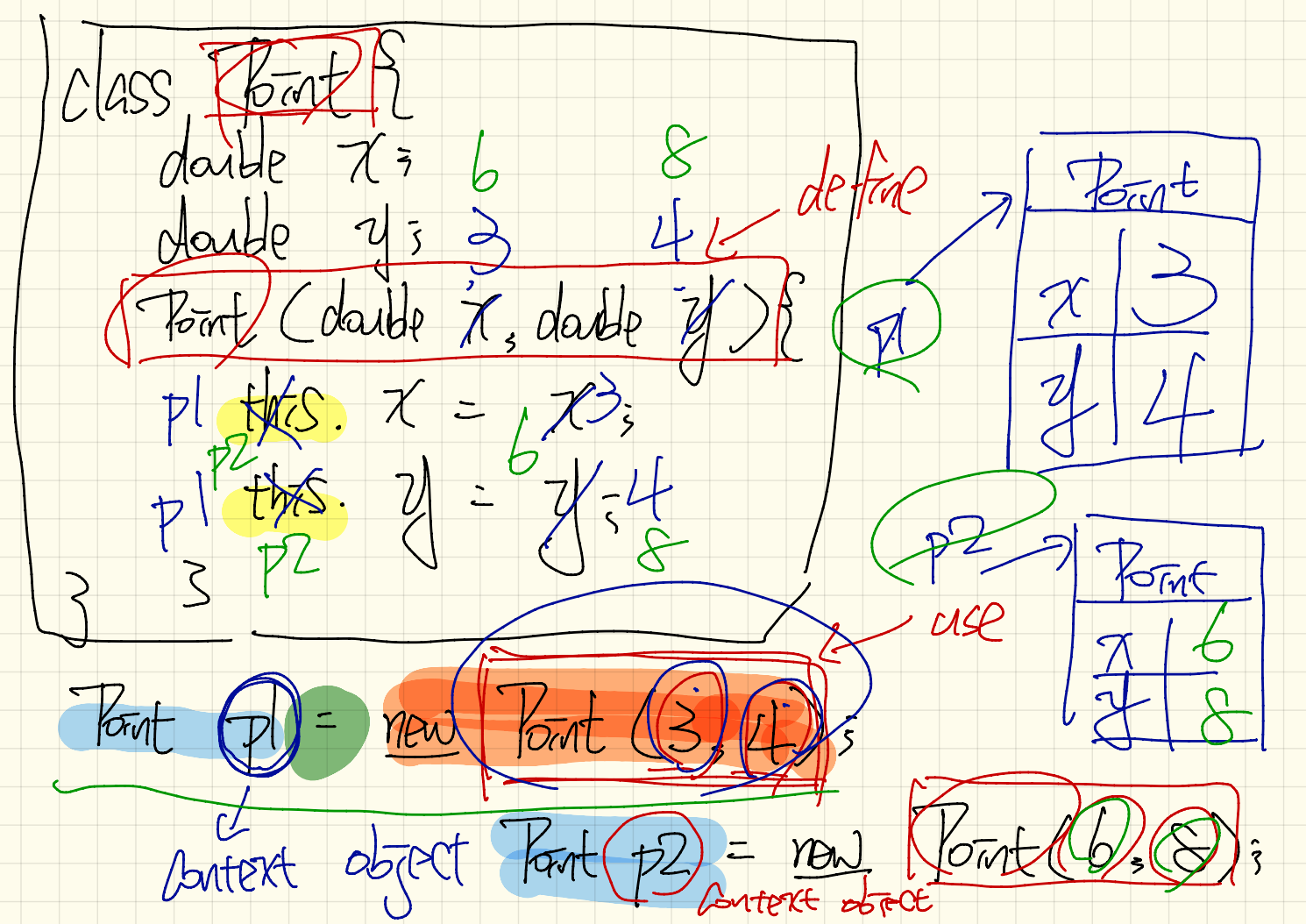
User/Client





① We declare a variable p. p can only store addresses of a memory portion that store Point information. Point info.

② Allocate a memory portion for storing


class Point {
double x;
double y;
Point (double x, double y) {
p1 this. x = x;
p2 this. y = y;
}
define
p1
Point
x 3
y 4
p2
Point
x 6
y 8
use
Point p1 = new Point (3, 4);
Point p2 = new Point (6, 8);
context object
context object

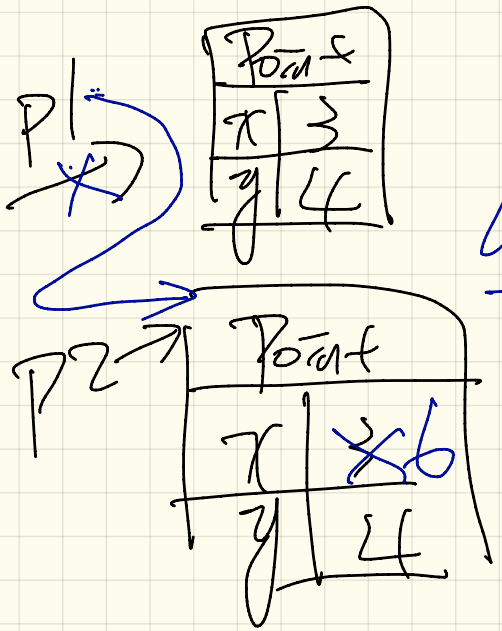
Point p1 = new . . . .

Point p2 = new . . .

p1 = p2 ;

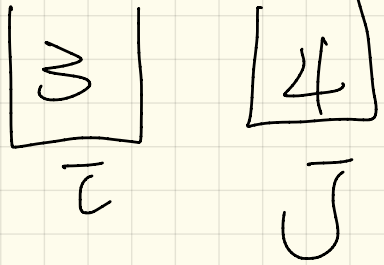
p1.x = 6

p2.x ?



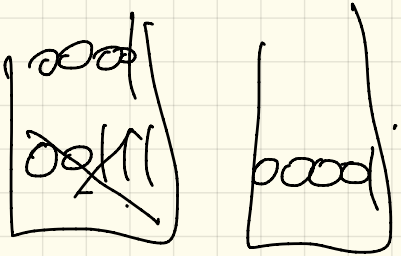
aliasing

int i = 3;
int j = 4;



i == j F

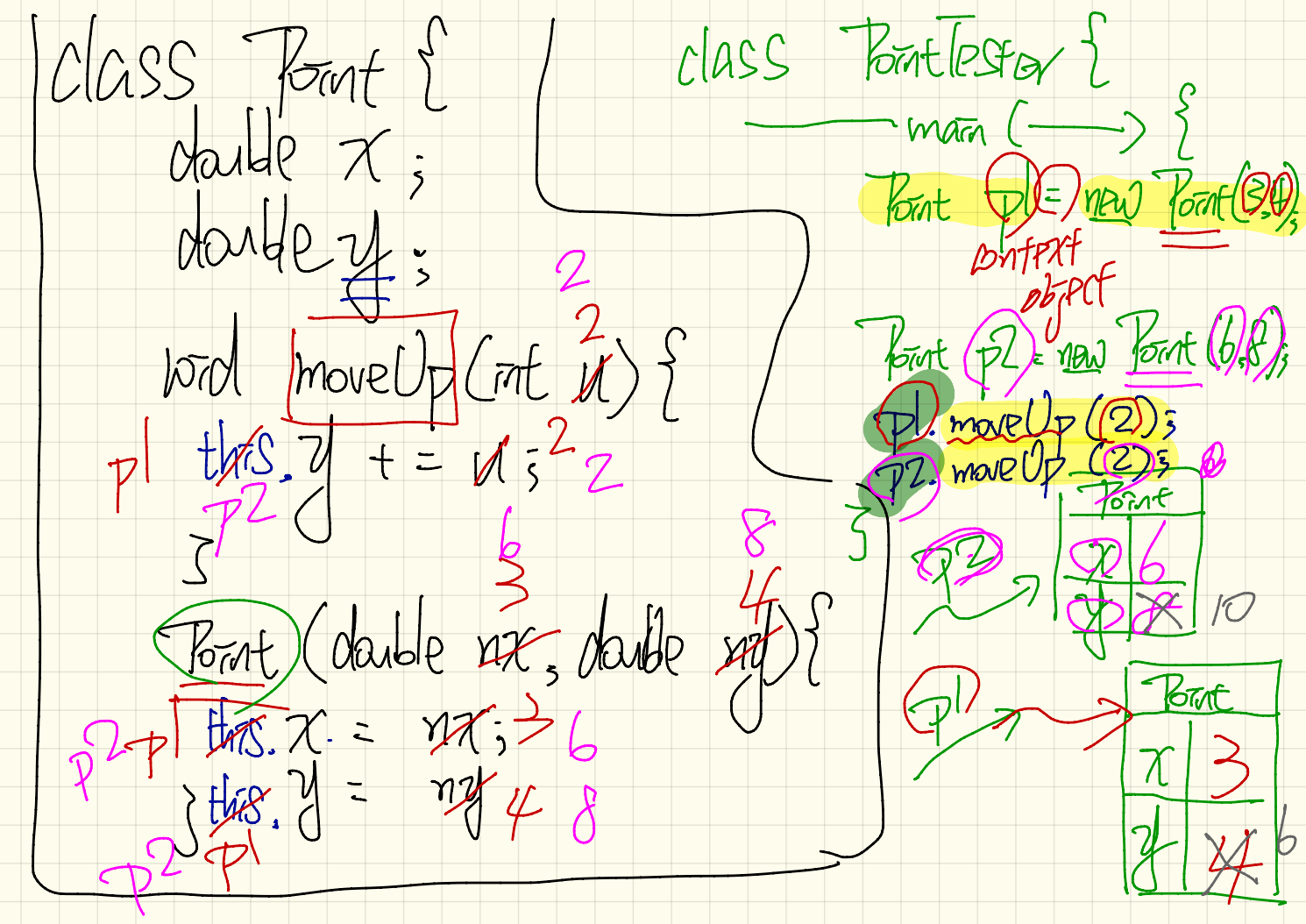
Point p1 = new Point(3,4);
Point p2 = new Point(3,4);

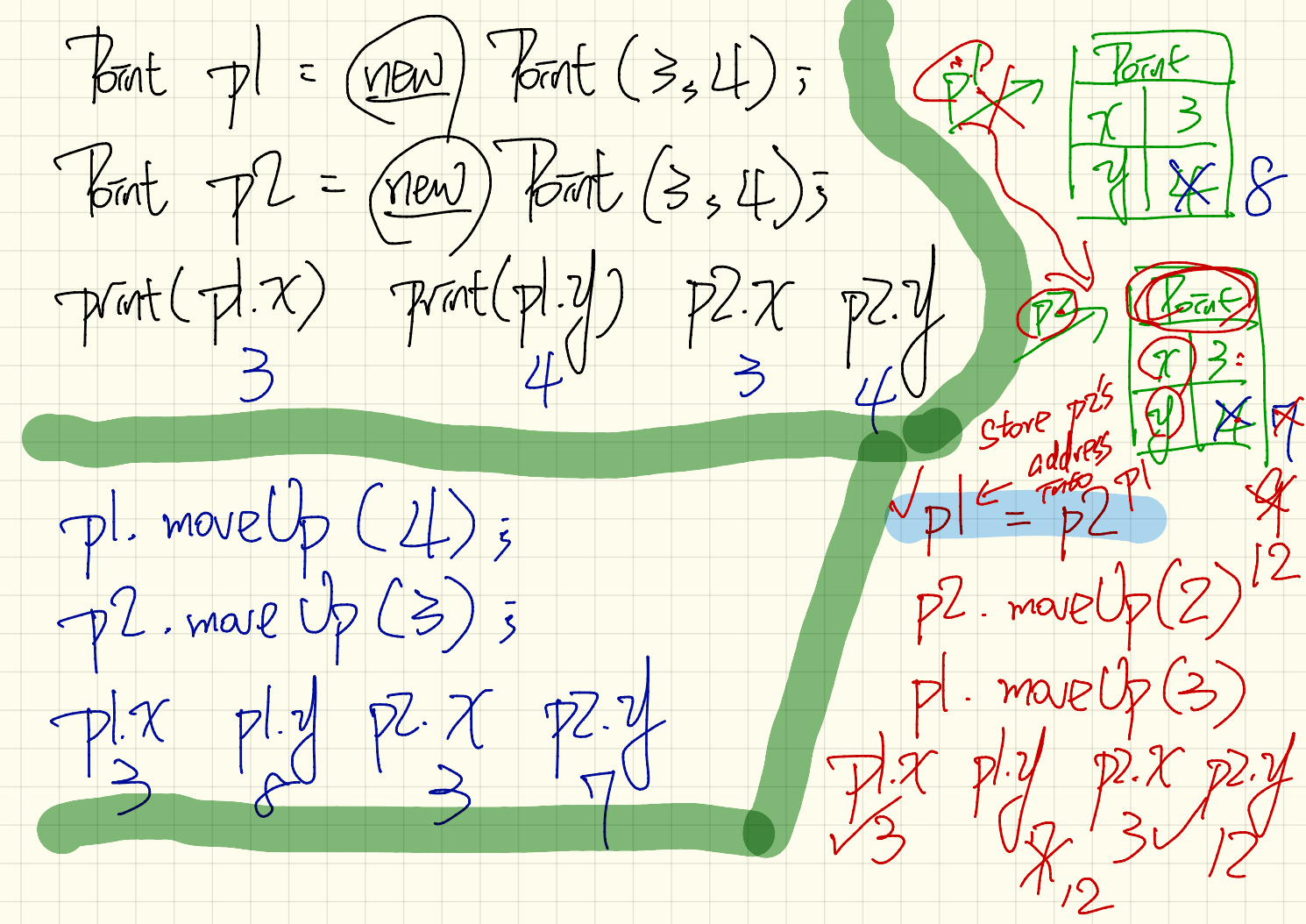


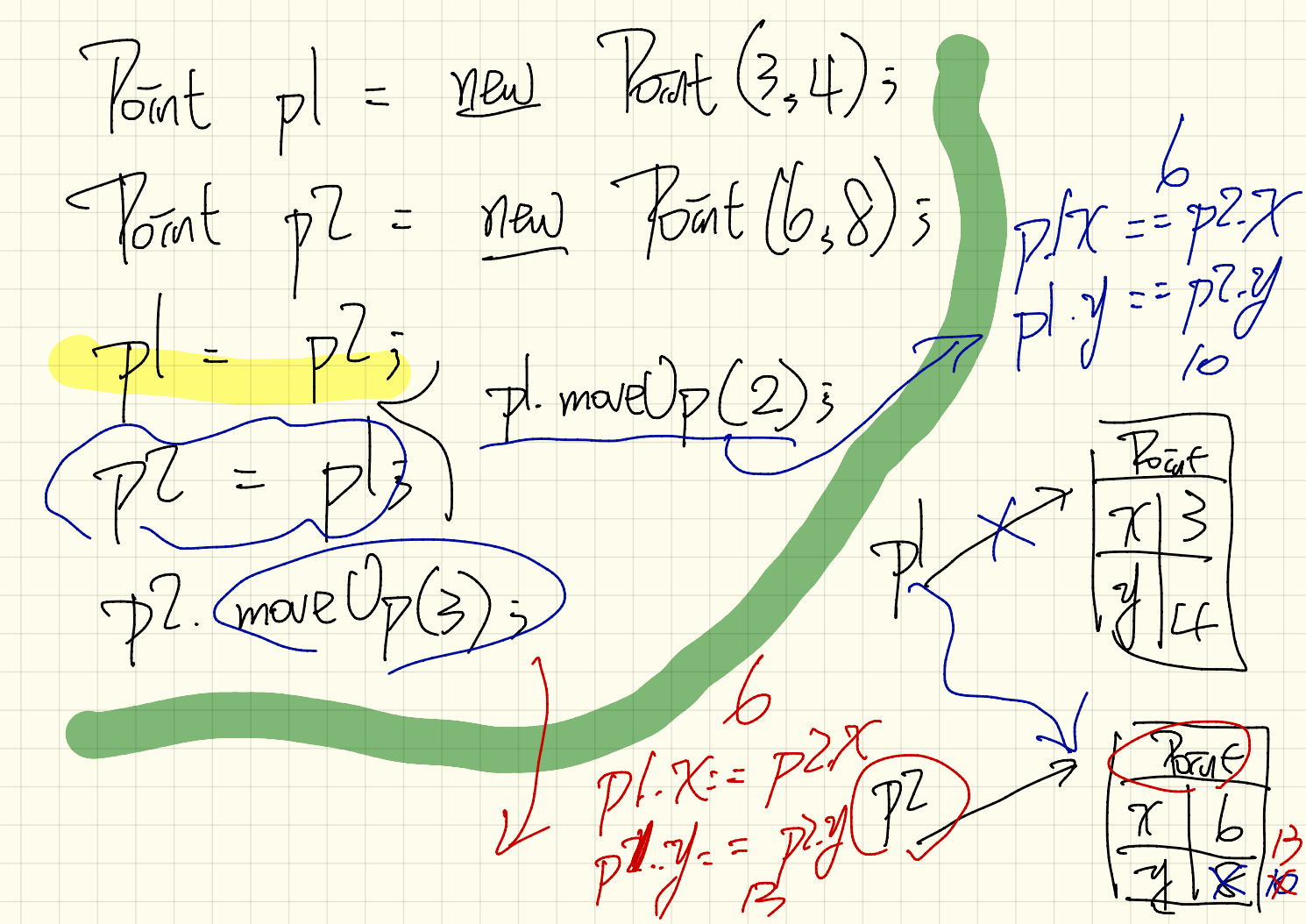
p1 = p2, p1 == p2

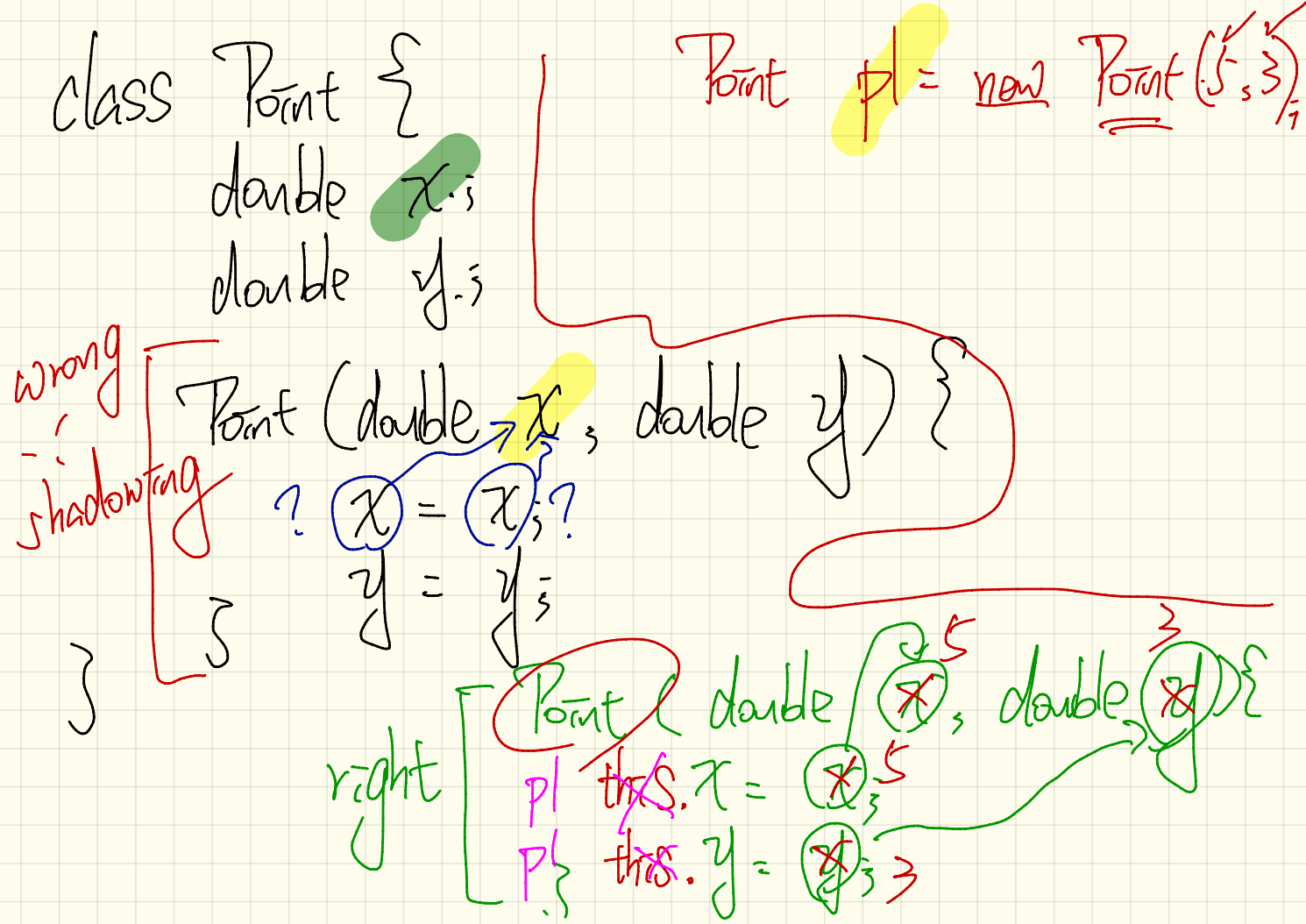
Lecture 6

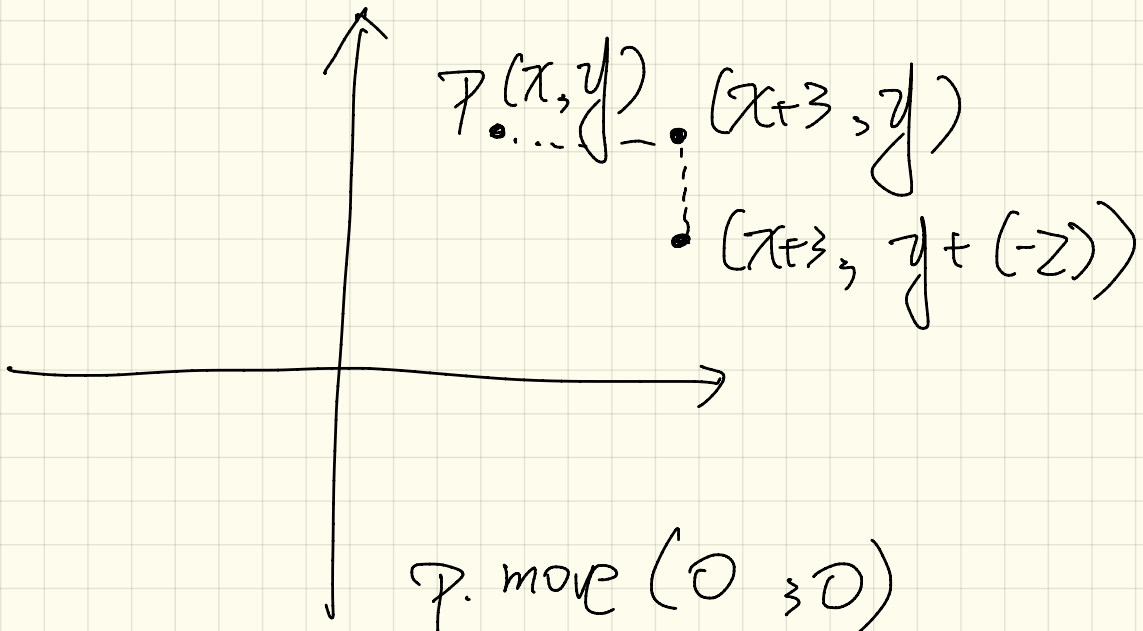
Tuesday Sept. 26


class Point {
double x;
double y;
void moveUp(int u) {
this.y += u;
}
Point(double nx, double ny) {
this.x = nx;
this.y = ny
}
class PointTester {
main (-->) {
Point p1 = new Point(3,4);
Point p2 = new Point(6,8);
p1.moveUp(2);
p2.moveUp(2);
}
Point
x 6
y 10
Point
x 3
y 6


Point p1 = new Point(3,4);
Point p2 = new Point(3,4);
print(p1.x) print(p1.y) p2.x p2.y
3 4 3 4
p1.moveUp(4);
p2.moveUp(3);
p1.x p1.y p2.x p2.y
3 8 3 7
p1
Point
x 3
y 8
p2
Point
x 3
y 7
store p2's address into p1
p1 = p2
p2.moveUp(2)
p1.moveUp(3)
p1.x p1.y p2.x p2.y
3 12 3 12


Point p1 = new Point(3,4);
Point p2 = new Point(6,8);
p1 = p2;
p2 = p1;
p2.moveUp(3);
p1.moveUp(2);
p1.x == p2.x
p1.y == p2.y
6
10
p1
Point
x 3
y 4
p1.x = p2.x
p1.y = p2.y
p2
Point
x 6
y 8
13
10

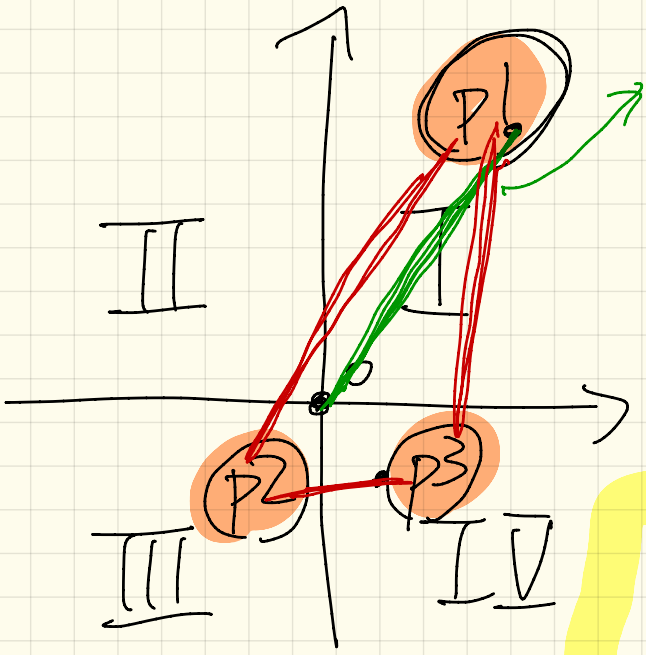

class Point {
double x;
double y;
Point p1 = new Point(5, 3);
wrong
shadowing
Point(double x, double y) {
? x = x; ?
y = y;
}
}
right
Point(double x, double y){
p1.x = x;
p1.y = y;
this
this



P. move(0, 0)

P. move(3 , −2)

horizontal | vertical



p1.getQ(p2)

p1.isAtFirstQuadrant()

T

p2.isAtFQ()

F

p1.getQ() return 1

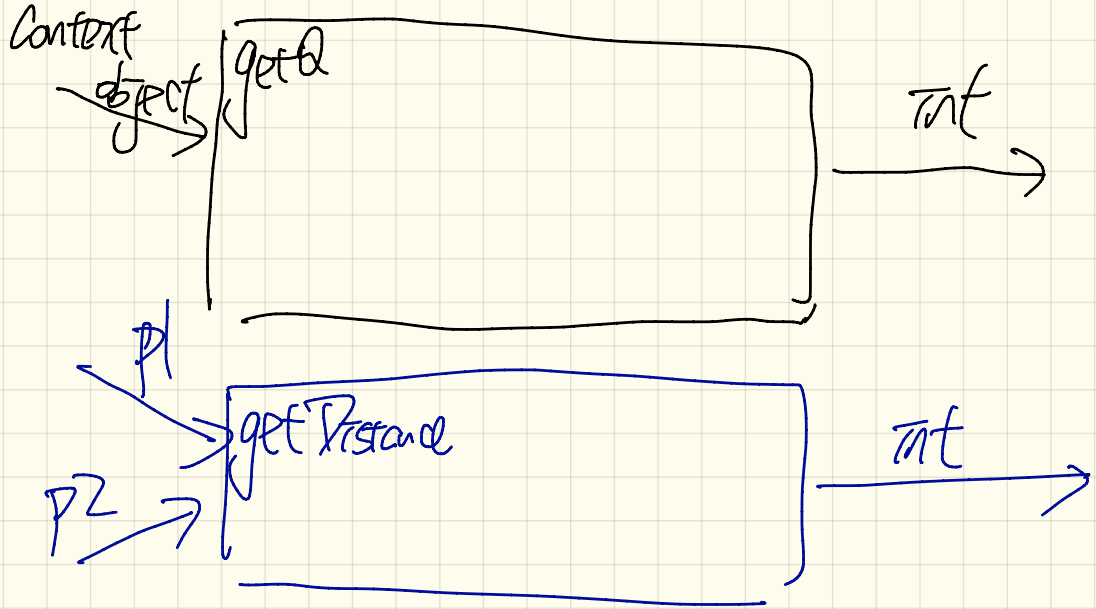
O.getQ() . 0

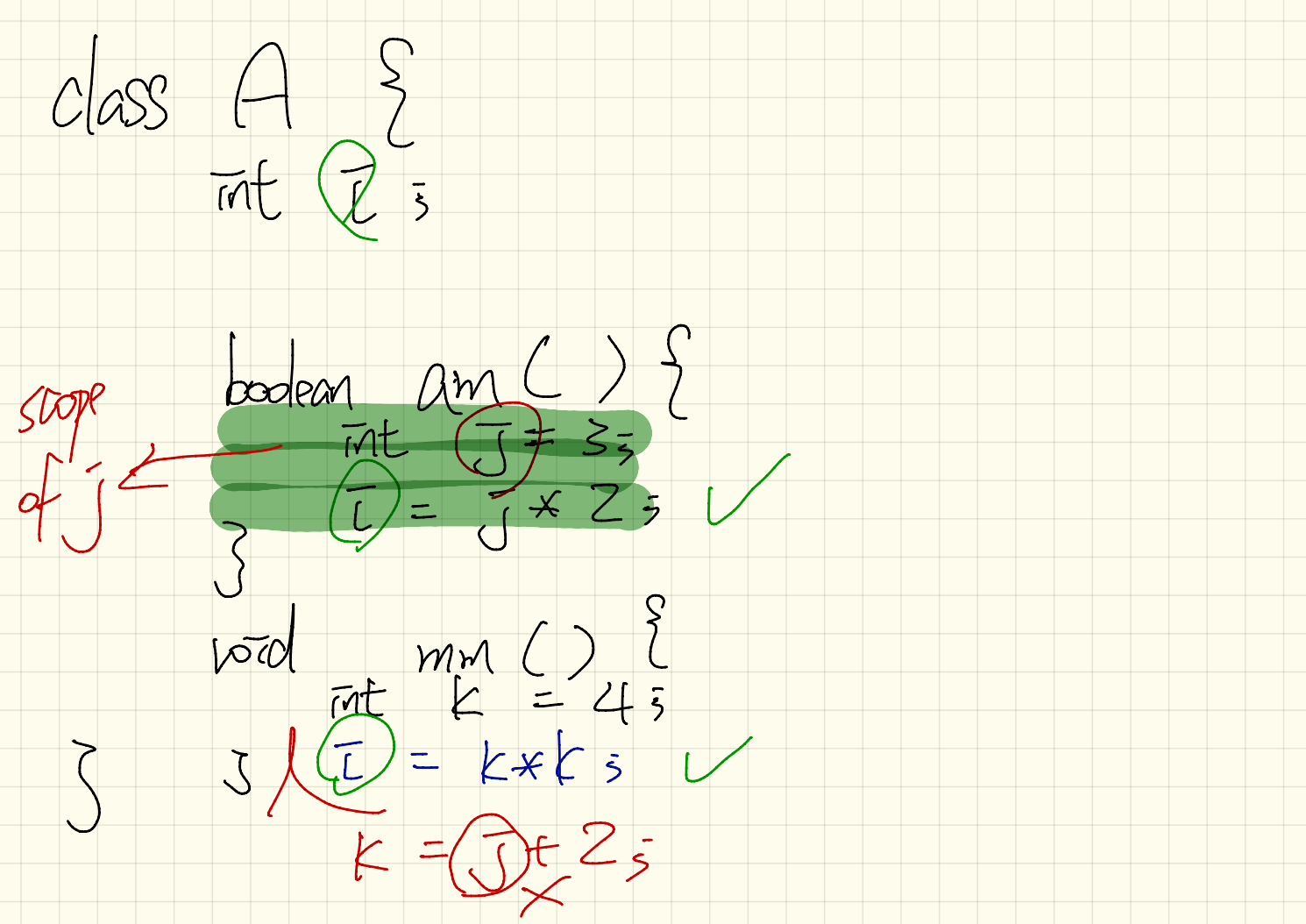
p3.getQ() 4

p1.getDistance(p2)

p1.getDistance(p2, p3)

Method - a block of code
- blackbox

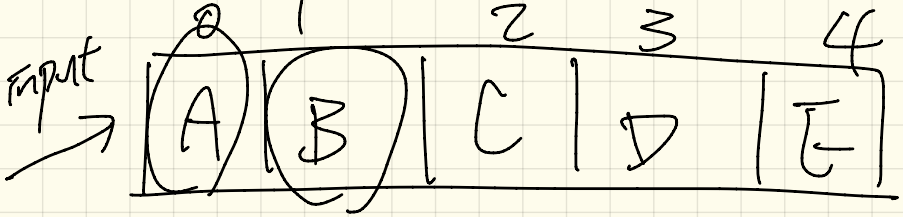



class A {
int i;
boolean am() {
int j = 3;
i = j * 2;
}
scope of j
void mm() {
int k = 4;
i = k * k;
j = k + 2;
}

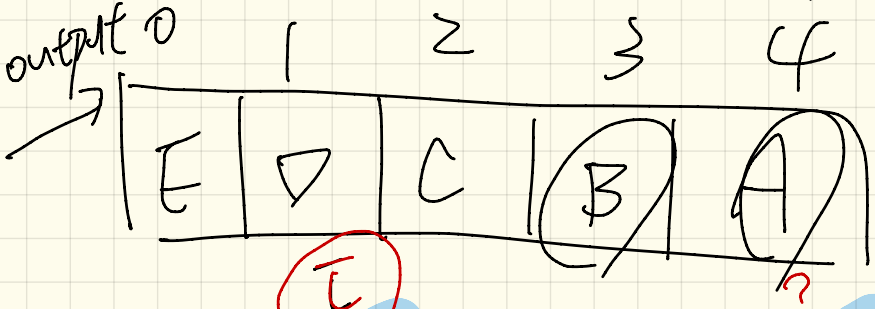
Tutorial Lab Test 1

Exercises

Tuesday Sept. 26



input.length == 5



i + ? = length - 1

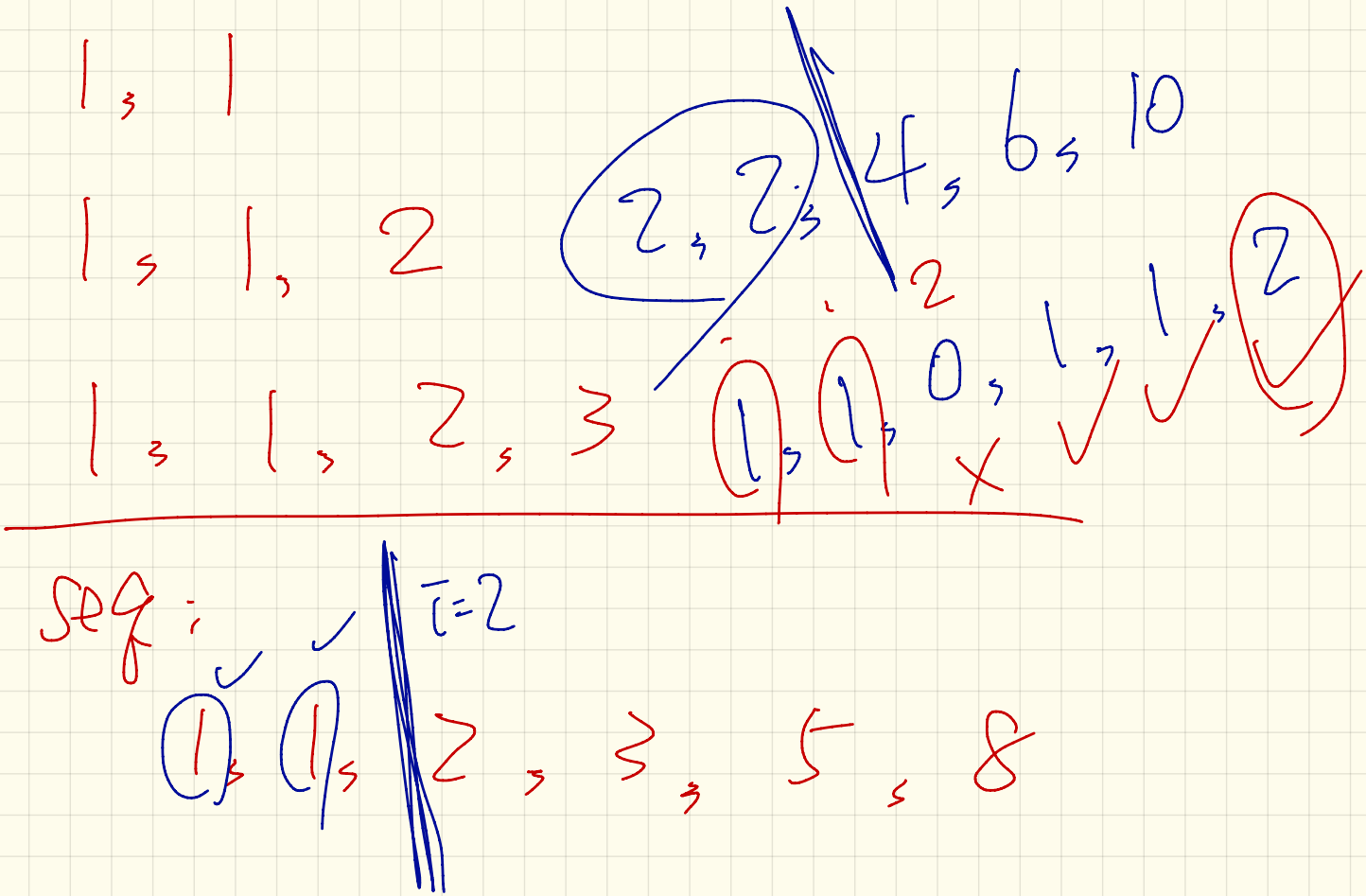
? = length - i - 1

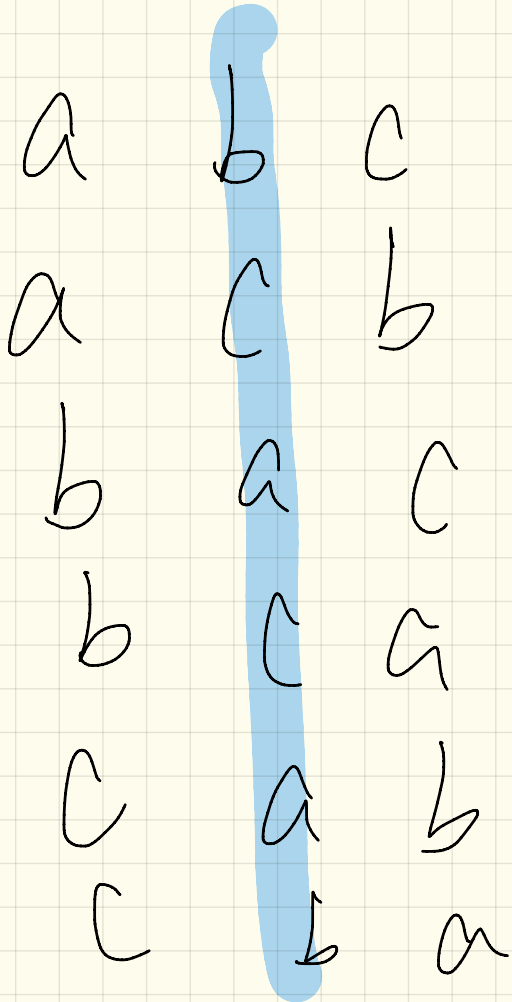
input[0] == output[4]

input[1] == output[3]

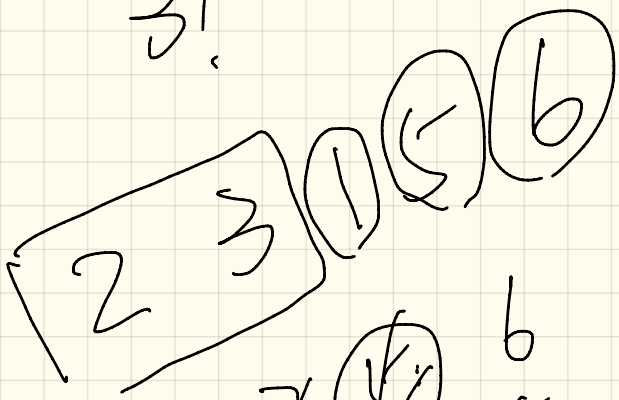
input[2] output[2]

input[3] output[1]


1, 1
1, 1, 2
1, 1, 2, 3
2, 2, 4, 6, 10
2
1, 1, 0, 1, 1, 2
seq:
1, 1, 2, 3, 5, 8
i=2



3!



max

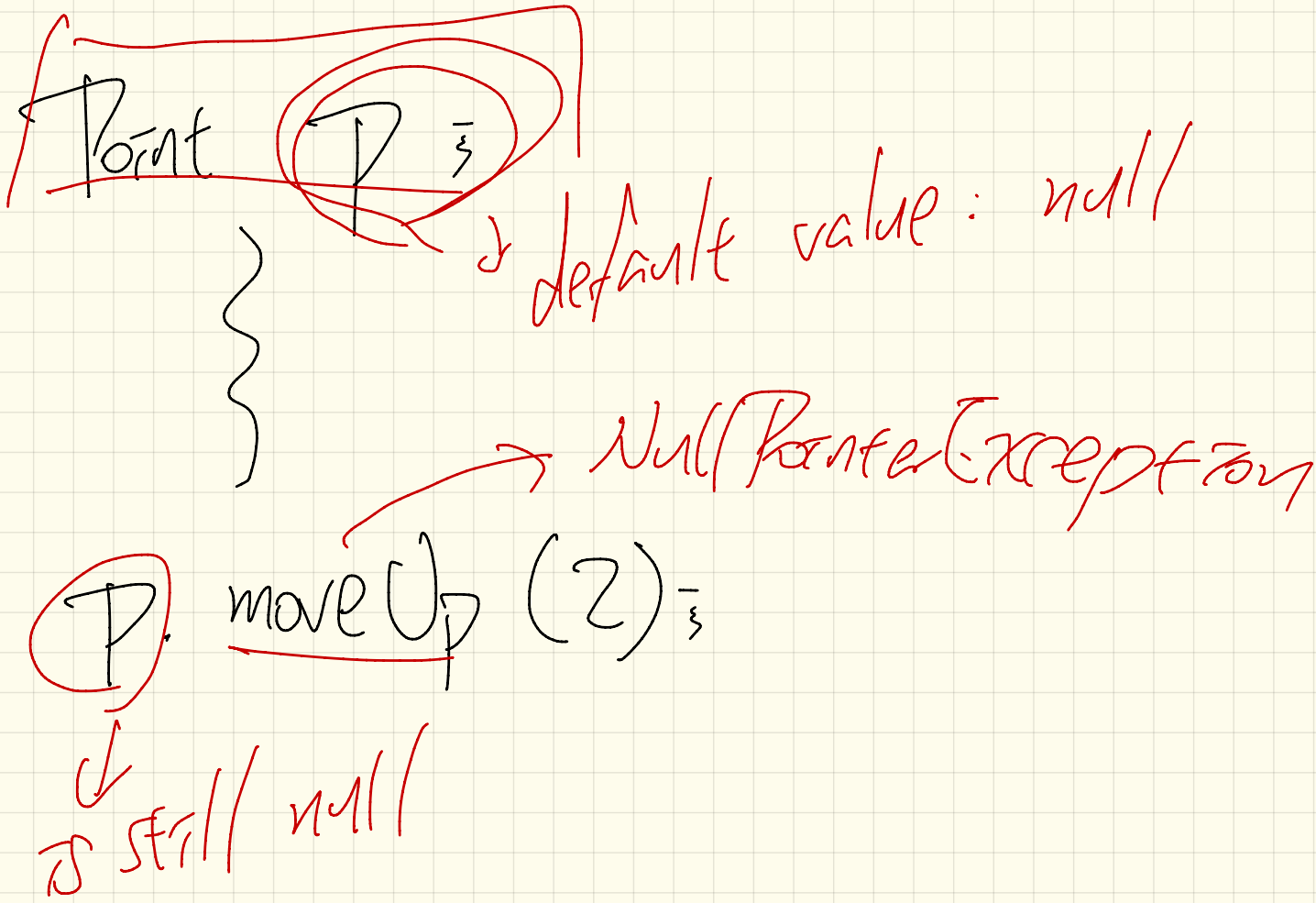
max2

max

max2

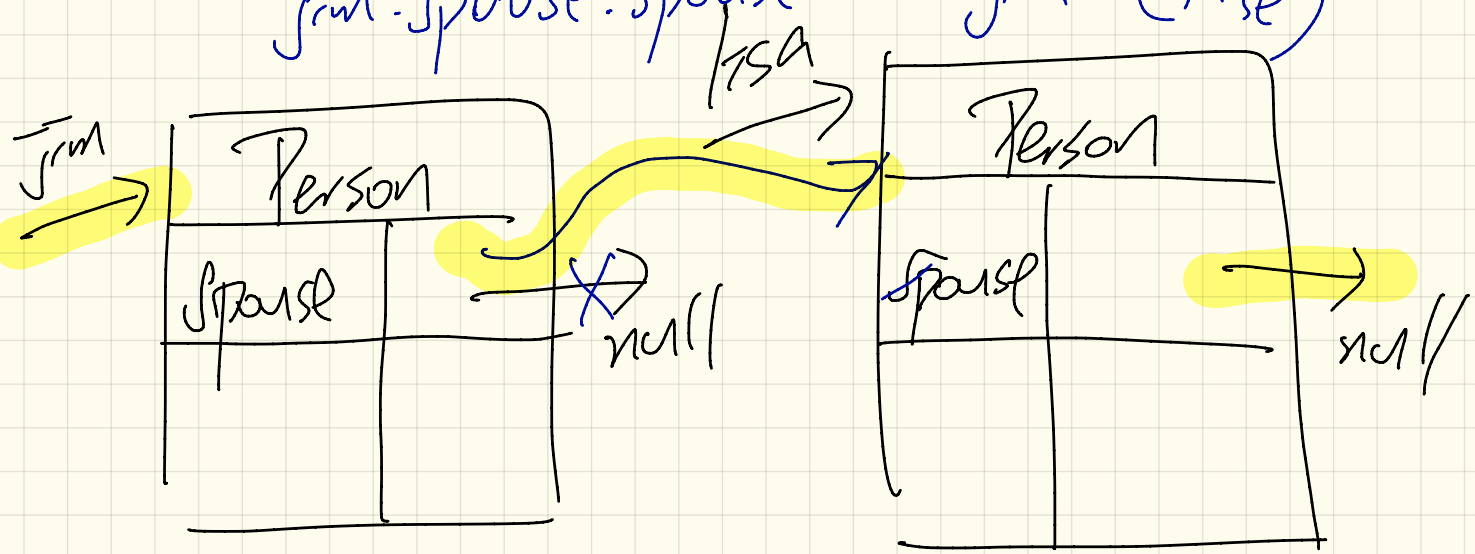
Lecture 7

Thursday Sept. 28


Point p;
default value: null
p.moveUp(2);
→ NullPointerException
still null

jim.spouse == lisa (true)

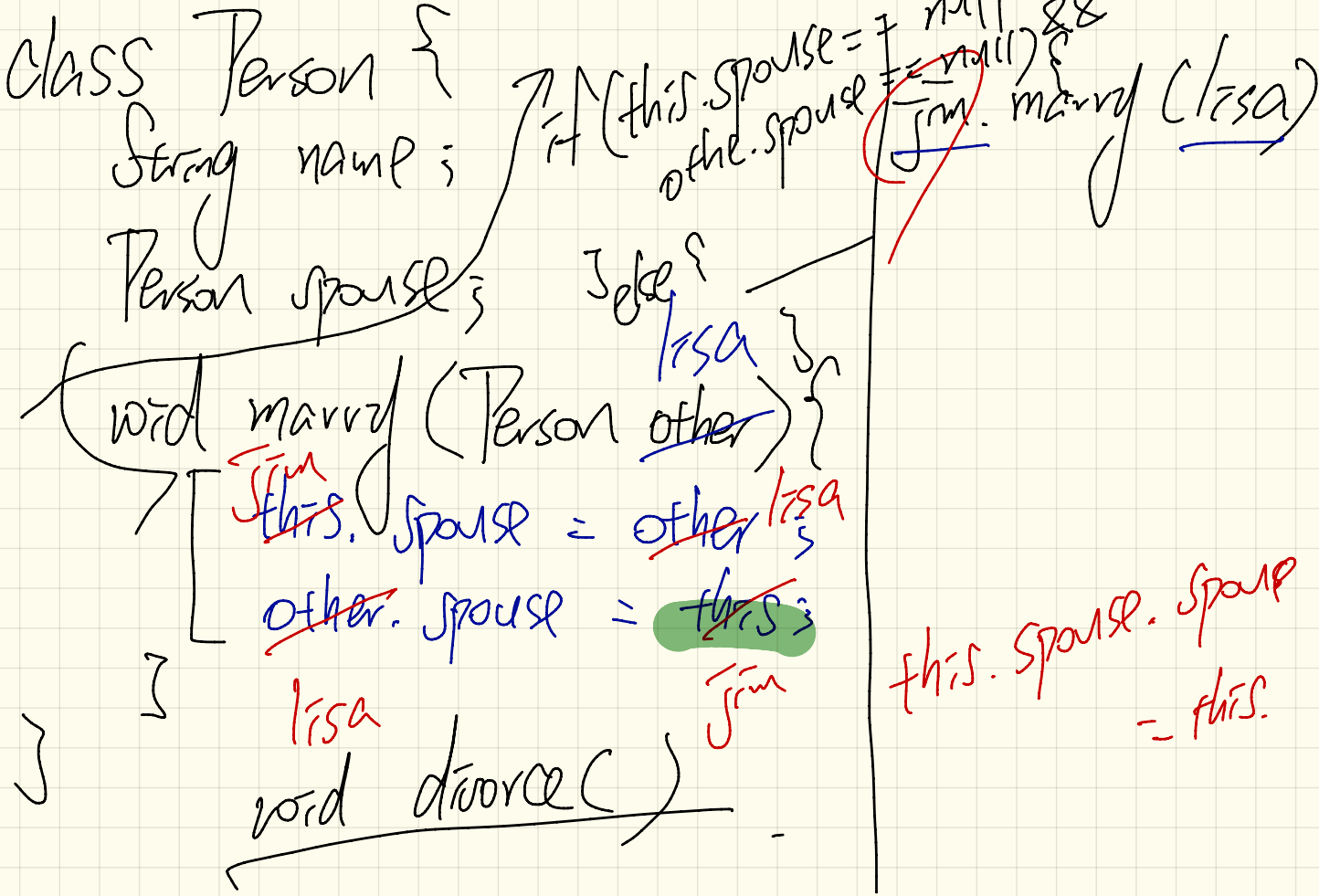
jim.spouse.spouse == jim (false)



jim.spouse == null

lisa.spouse == null.

jim.marry(lisa)


class Person {
String name;
Person spouse;
void marry(Person other) {
this.spouse = other;
other.spouse = this;
}
void divorce()
if (this.spouse != null &&
other.spouse != null) {
jim.marry(lisa)
} else {
lisa
}
this.spouse.spouse
= this.
lisa
jim

```
class Point {
    void moveUpBy(int u) {
        this.y += u;
    }
    Point movedUpBy(int u) {
        Point other = new Point(this.x, this.y);
        other.moveUpBy(u);
        return other;
    }
}
```

Point p2 = p.movedUpBy(2);

return address of other

p2 (2,5)

p (2,3)

other

p2

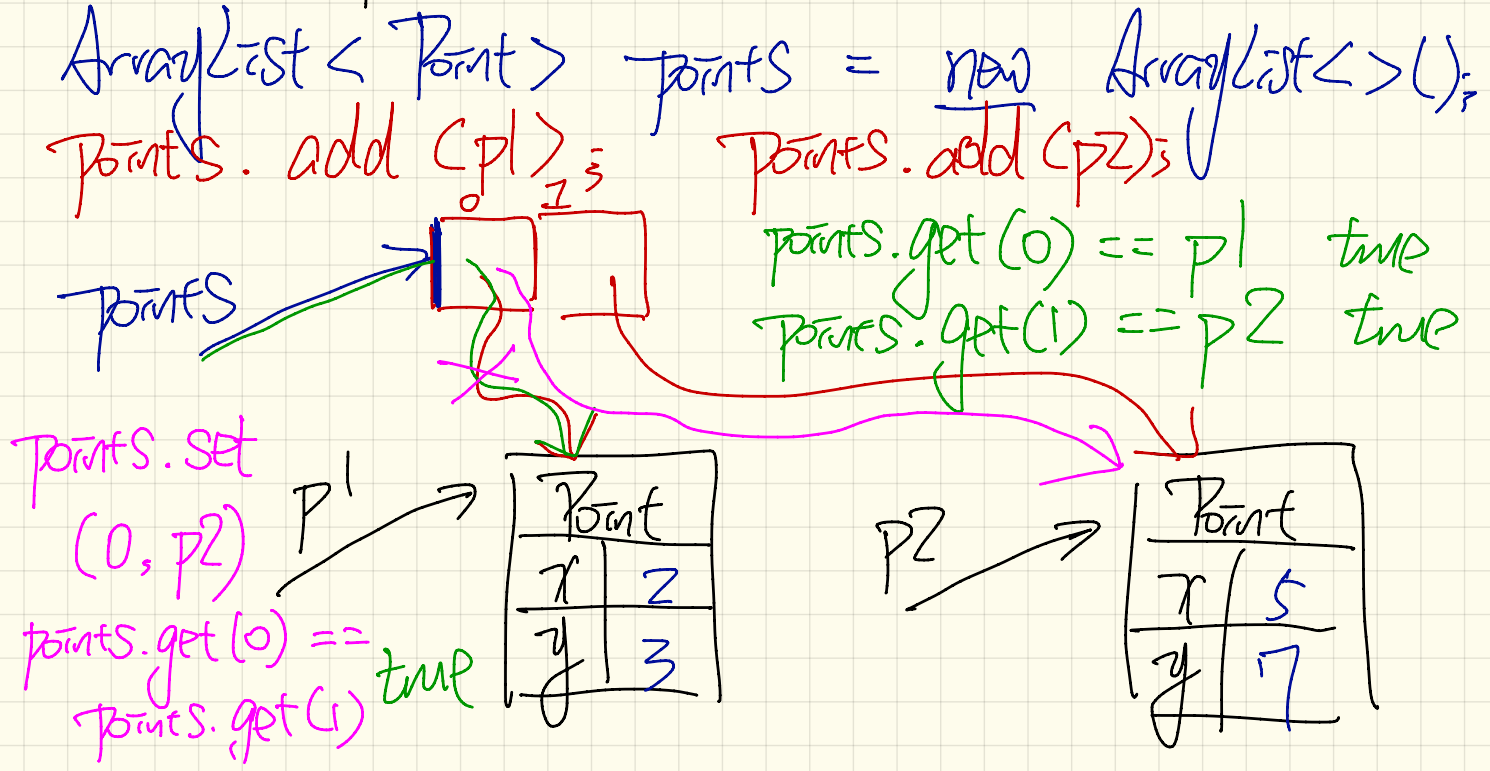
Point	
x	2
y	5

return the address of other

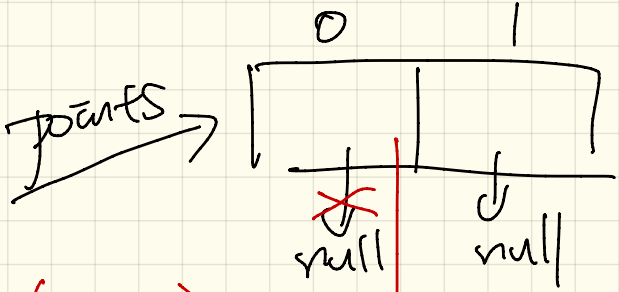
ArrayList<Point> points;

At runtime, `points` stores the address of an ArrayList object. Every element of the array list stores the address of some Point object.

```
Point p1 = new Point(2, 3);
Point p2 = new Point(5, 7);
ArrayList<Point> points = new ArrayList<>();
points.add(p1);
points.add(p2);
```



```
Point[] points;

Point p = new Point(3,4);

points = new Point[2];
```




```
points[0] = new Point(3,4);
```

Point p1 = new Point(2,3)

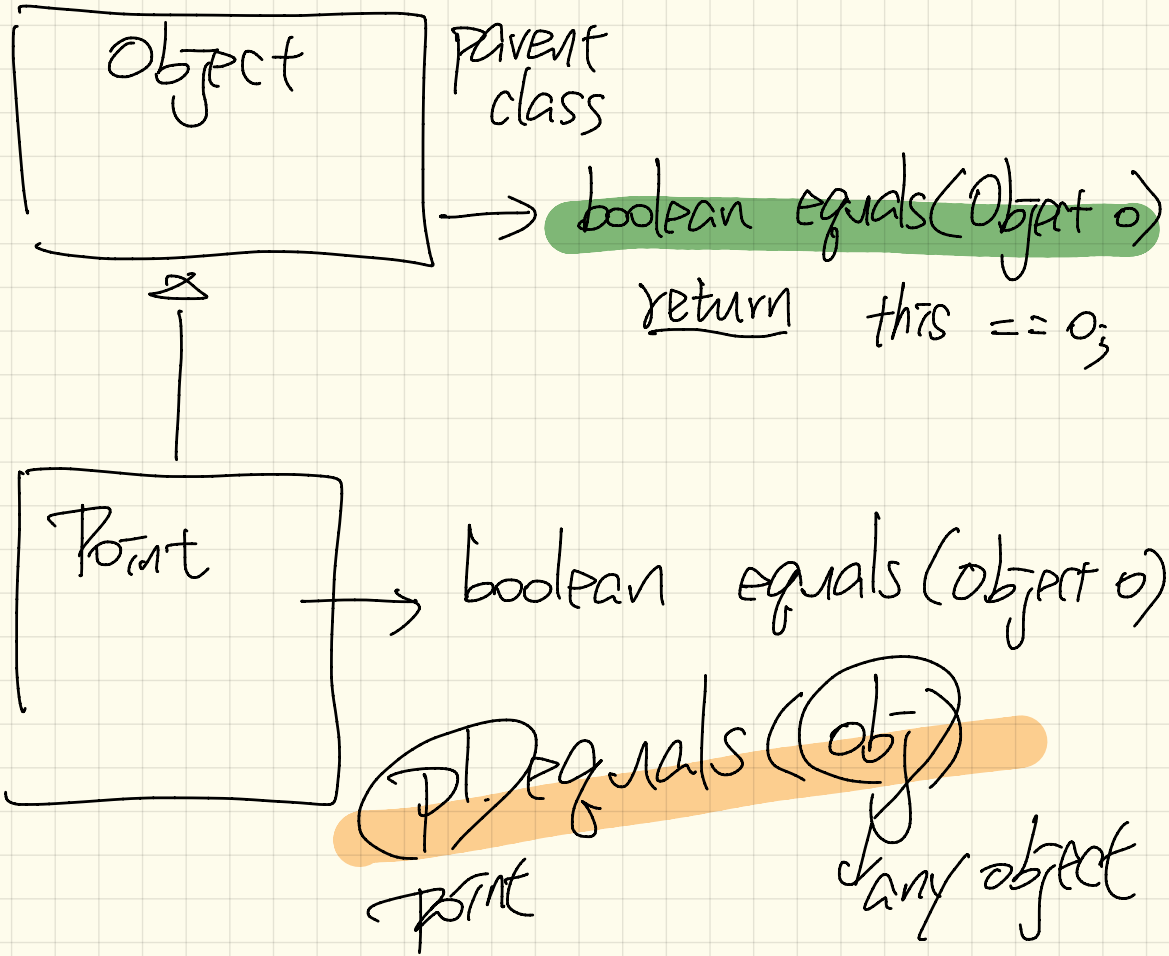
Point p2 = new Point(2,3)

① p1 == p2 false

② [p1.x == p2.x && p1.y == p2.y] true

define a helper method "equals"

p1.equals(p2)


Object
parent class
boolean equals(Object o)
return this == o;
Point
boolean equals (Object o)
p1.equals(obj)
point
any object

Point p1 = new Point(2,3);

Point p2 = p1;

Point p3 = new Point(2,3);

p1.equals(p2) ✓

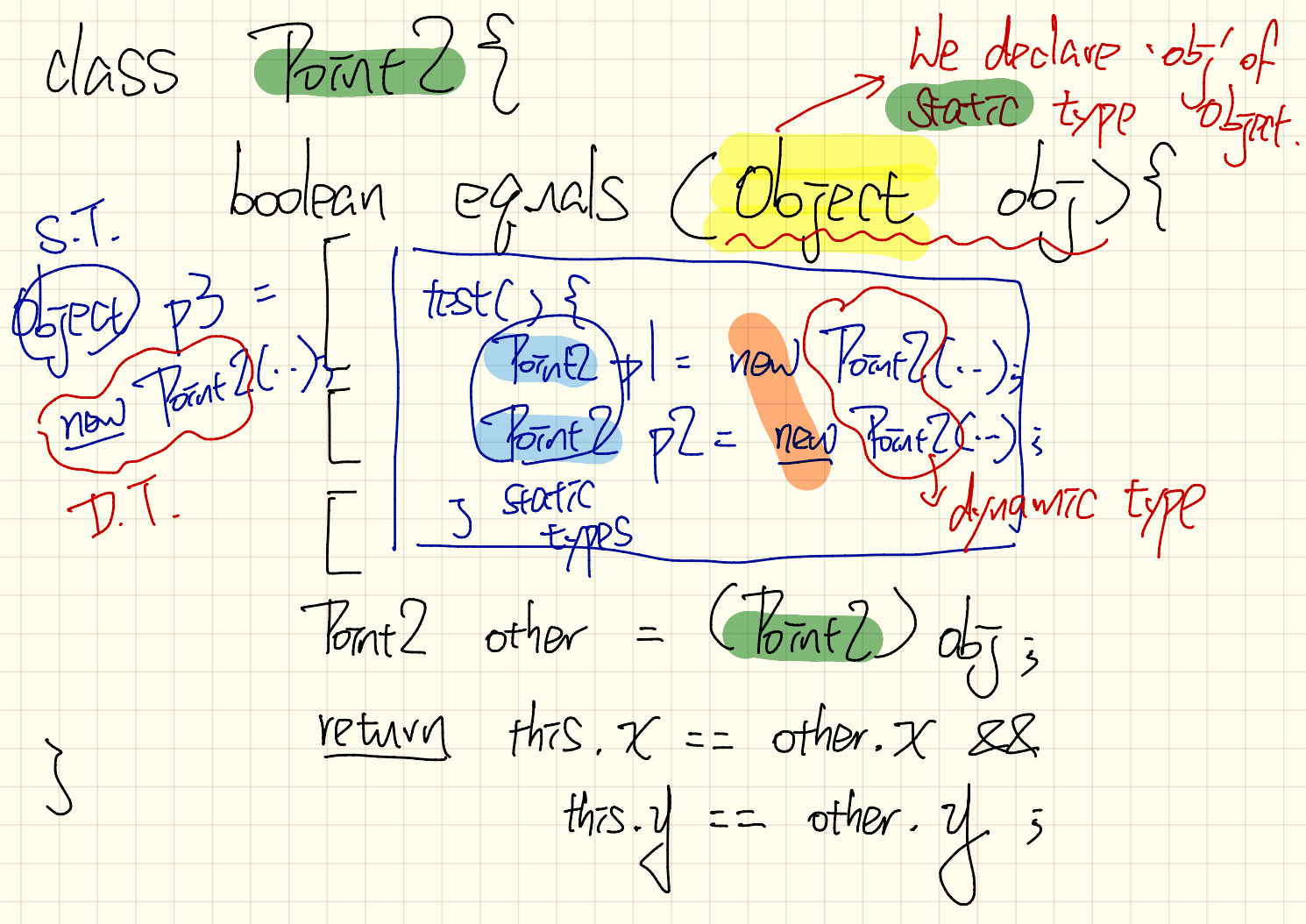
p1.equals(null) ✗

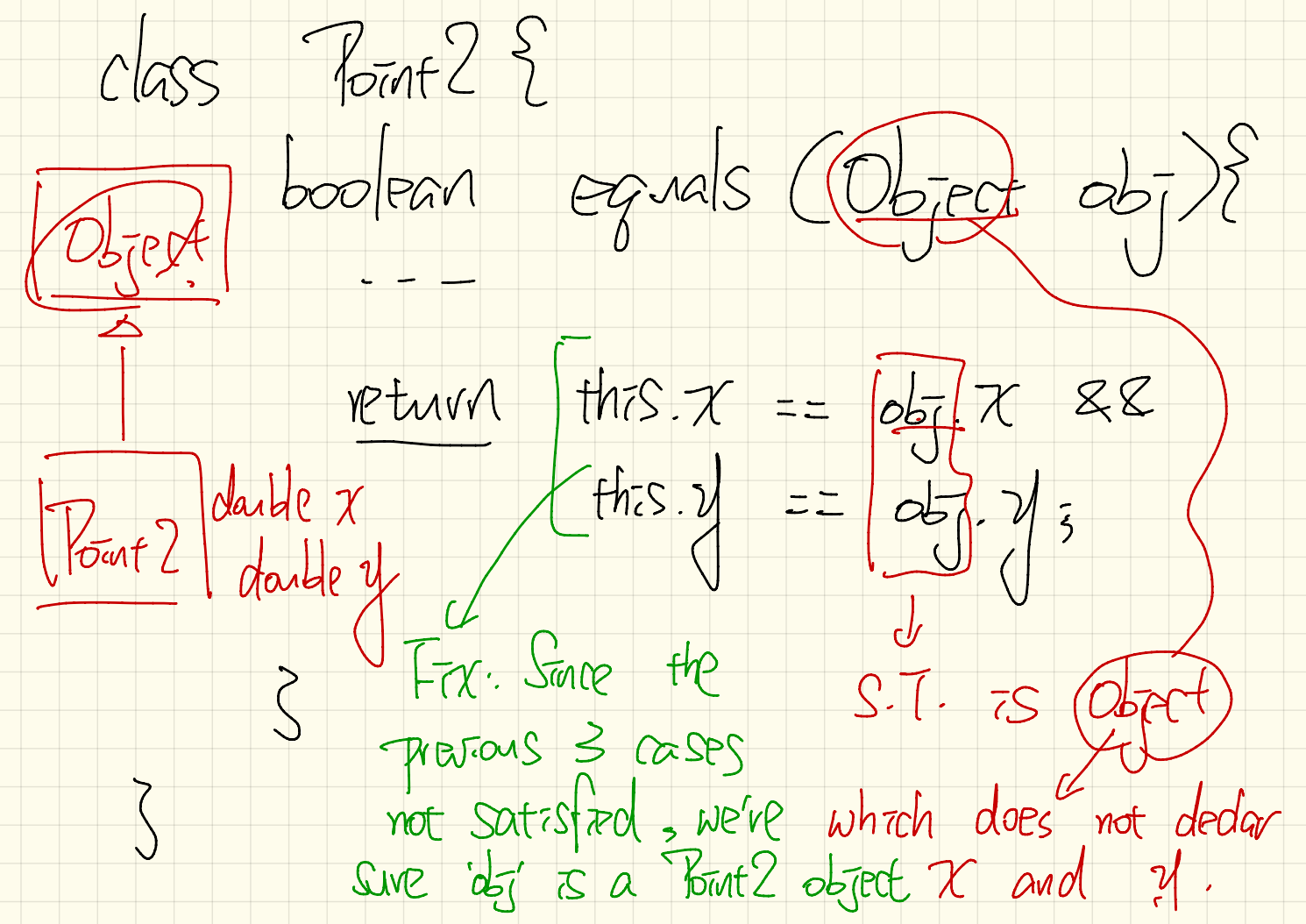
String p3 = "(2,3)"; p1.equals(p3) ✓ expected

p1.equals(p3) ✗

Lecture 8

Tuesday Oct. 3


class Point2 {
We declare 'obj' of static type Object.
boolean equals (Object obj) {
S.T.
Object p3 = new Point2(..);
D.T.
test() {
Point2 p1 = new Point2(..);
Point2 p2 = new Point2(..);
static types
dynamic type
Point2 other = (Point2) obj;
return this.x == other.x &&
this.y == other.y;
}


class Point2 {
boolean equals (Object obj){
...
return this.x == obj.x &&
this.y == obj.y;
}
}
Object
Point2
double x
double y
Fix: Since the previous 3 cases not satisfied, we're sure 'obj' is a Point2 object x and y.
S.T. is Object which does not declar

class Point2 {

boolean equals (Object obj) {

- - -

Point2 other = (Point2) obj;

temporarily change the S.T. of obj from Object to Point2

return this.x == other.x && this.y == other.y ;

alternatively:

return this.x == ((Point2) obj).x && this.y == ((Point2) obj).y ;

}

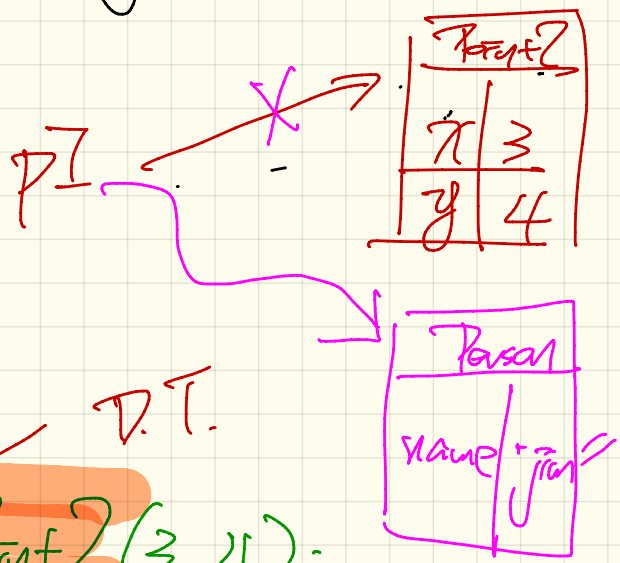
}

once declared, can never be changed

Static type (type during declaration)

Point2 p1

Person jim



Dynamic type

D.T.

(1) Point2 p1 = new Point2(3,4);

S.T.

(2) Object p1 = new Point2(3,4);

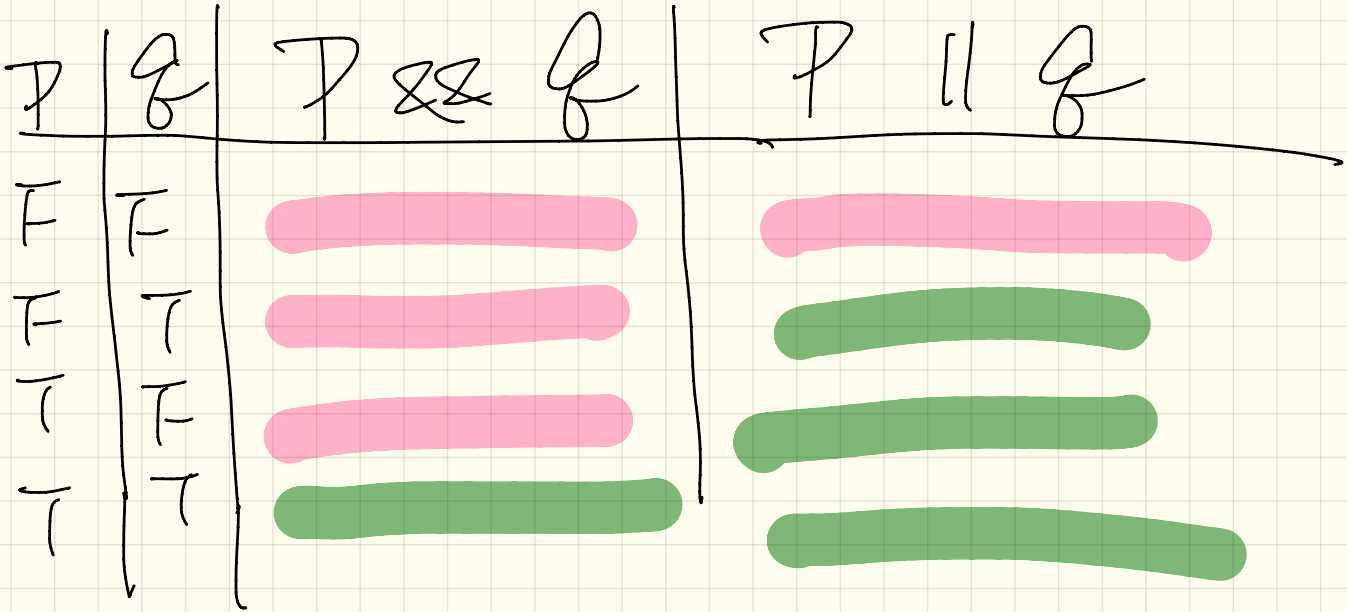
p1 = new Person("jim"); ← D.T. changes to Person.

Q. Swap the order: this.getClass() != obj.getClass() || obj == null. ✗ When obj is null → NullPointerException.

```
else if (obj == null || this.getClass() != obj.getClass()) {
    return false
}
```

Q. what if obj is null.

A. No. :(SCE.



P is F,
no need to evaluate Q

P is T,
no need to evaluate Q.

int i = read from user.

P Q R

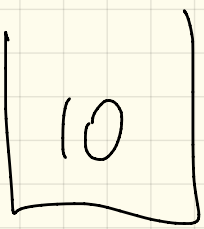
if (0 <= i && i < a.length && a[i] > y)

guard to ensure i is a valid index

if (R && P && Q) X when i < 0 — R will throw IndexOutOfBound

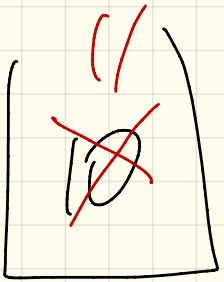
if (P && R && Q) X when i >= a.length

int i = 10



i

argument



j

parameter

Point p = new Point(3, 4);

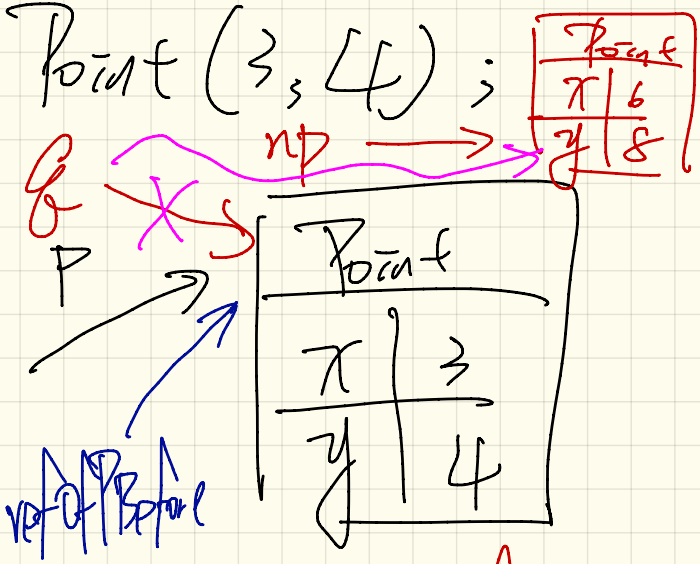
Point refOfPBefore = p;

u.reassignRef(p);

assertTrue(p == refOfPBefore);

void reassignRef(Point q) {
Point np = new Point(6, 8);
q = np;
}

call by ref



Lecture 9

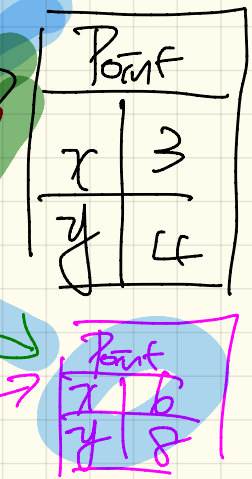
Thursday Oct. 5

```
class Util {
  void reassignRef(Point q) {
    Point np = new Point(6, 8);
    q = np;
  }
}
```

expecting an address of some Point object

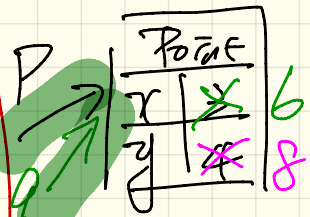
a copy of p will replace every occurrence of q

```
testCallByRef_1(){
  Util u = new Util();
  Point p = new Point(3, 4);
  u.reassignRef(p);
}
```



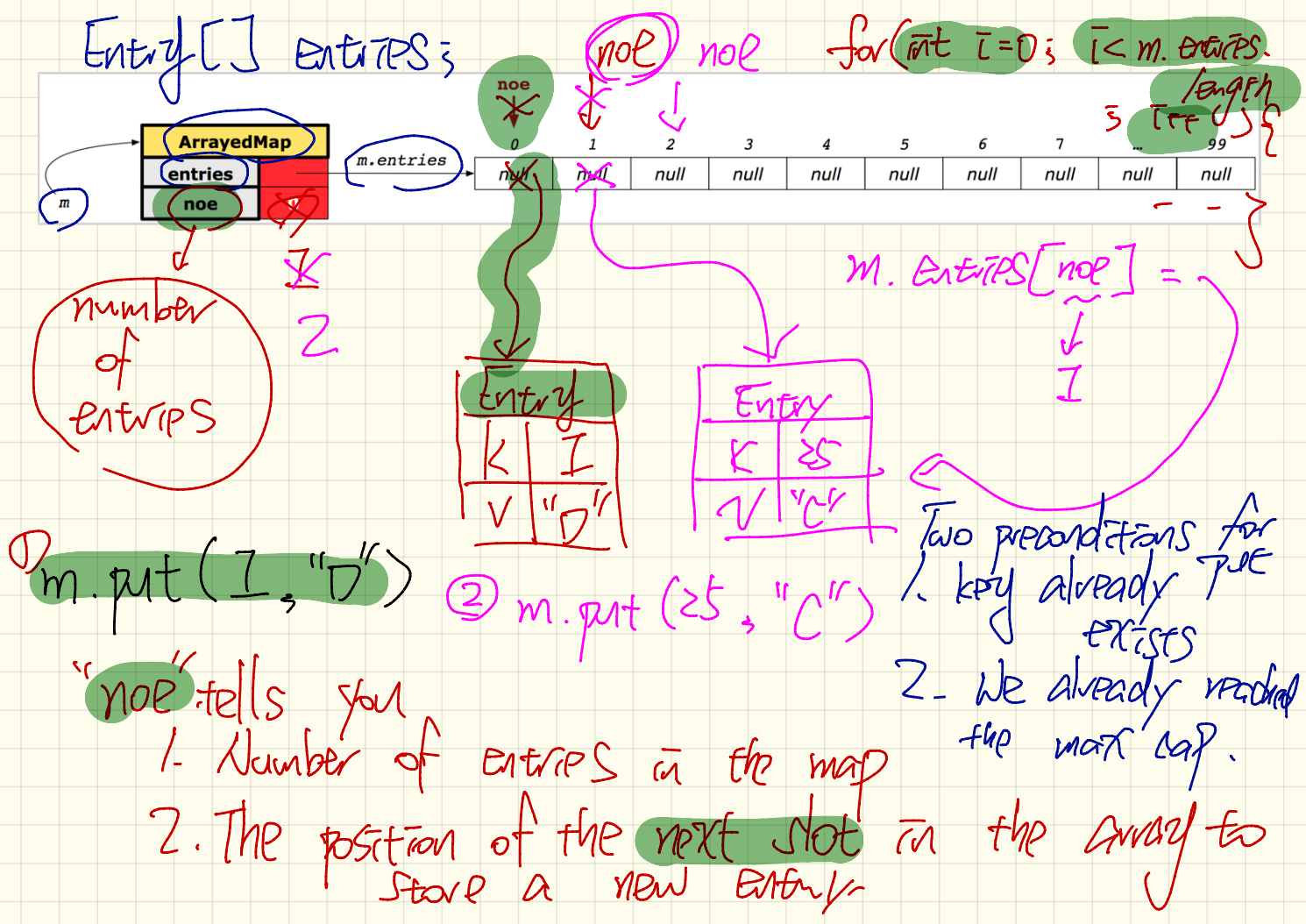
```
class Ut {
  void changeViaRef(Point q) {
    q.moveH(3);
    q.moveV(4);
  }
}
```

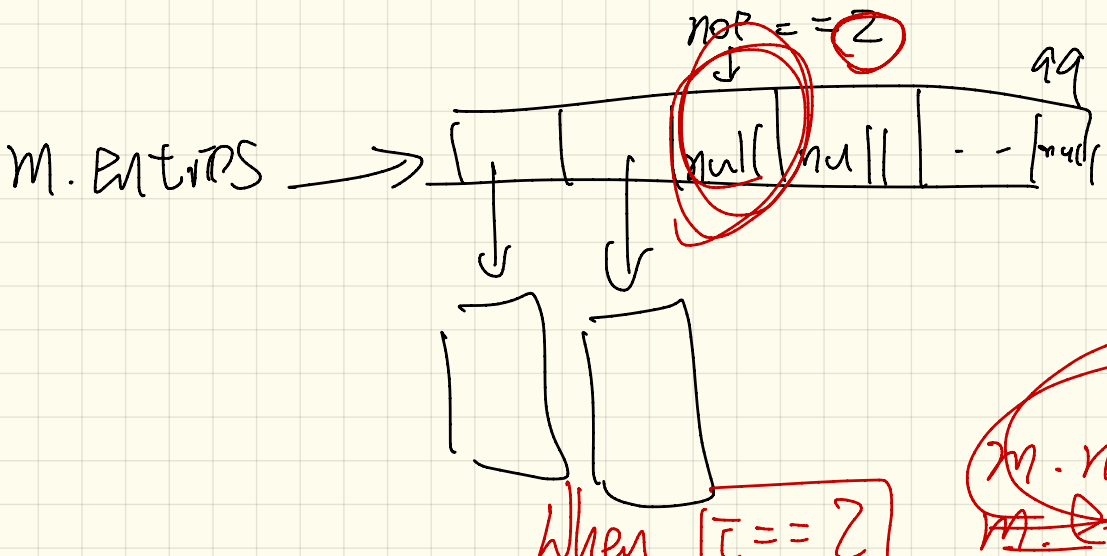
q is replaced by a copy of p.




```
testCallByRef_2() {
  Point p = new Point(3,4);
  u.changeViaRef(p);
}
```

p.x == 6 &&
q.y == 8.
assertTrue(
p.x == q.x &&
p.y == q.y &&


Entry[] entries;
noe
noe
for(int i=0; i< m.entries.length; i++){
ArrayedMap
entries
noe
m
m.entries
0 1 2 3 4 5 6 7 ... 99
null null null null null null null null null null
number of entries
2
Entry
K I
V "D"
Entry
K 25
V "C"
m.entries[noe] =
noe
1
① m.put(1, "D")
② m.put(25, "C")
"noe" tells you
1. Number of entries in the map
2. The position of the next slot in the array to store a new entry.
Two preconditions for put
1. key already put exists
2. We already reached the max cap.




```
for( int i=0; i < m.entries.length ; i++) {
    println( m.entries[i].getKey() );
}
```

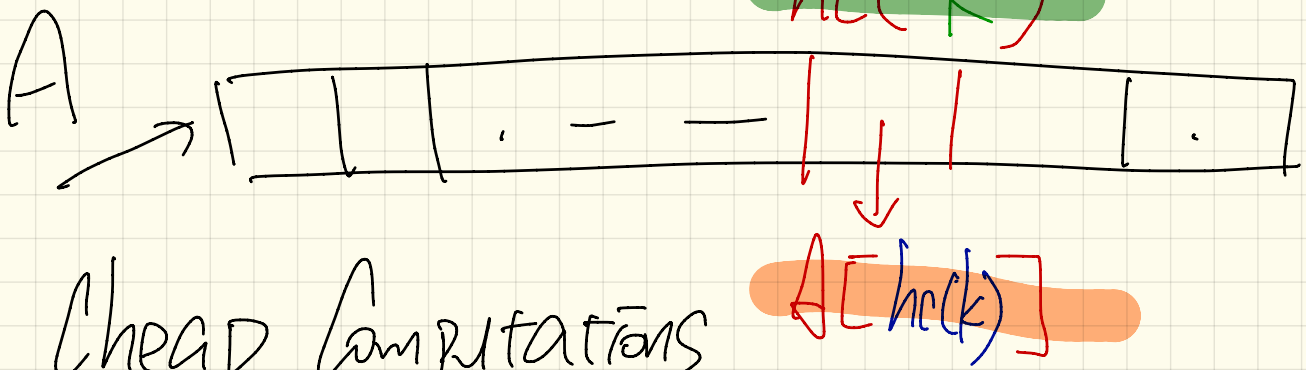
null.

function map vs. relation

(1, "D") (7, "D")

Hashing

K → hashing → hc(k)

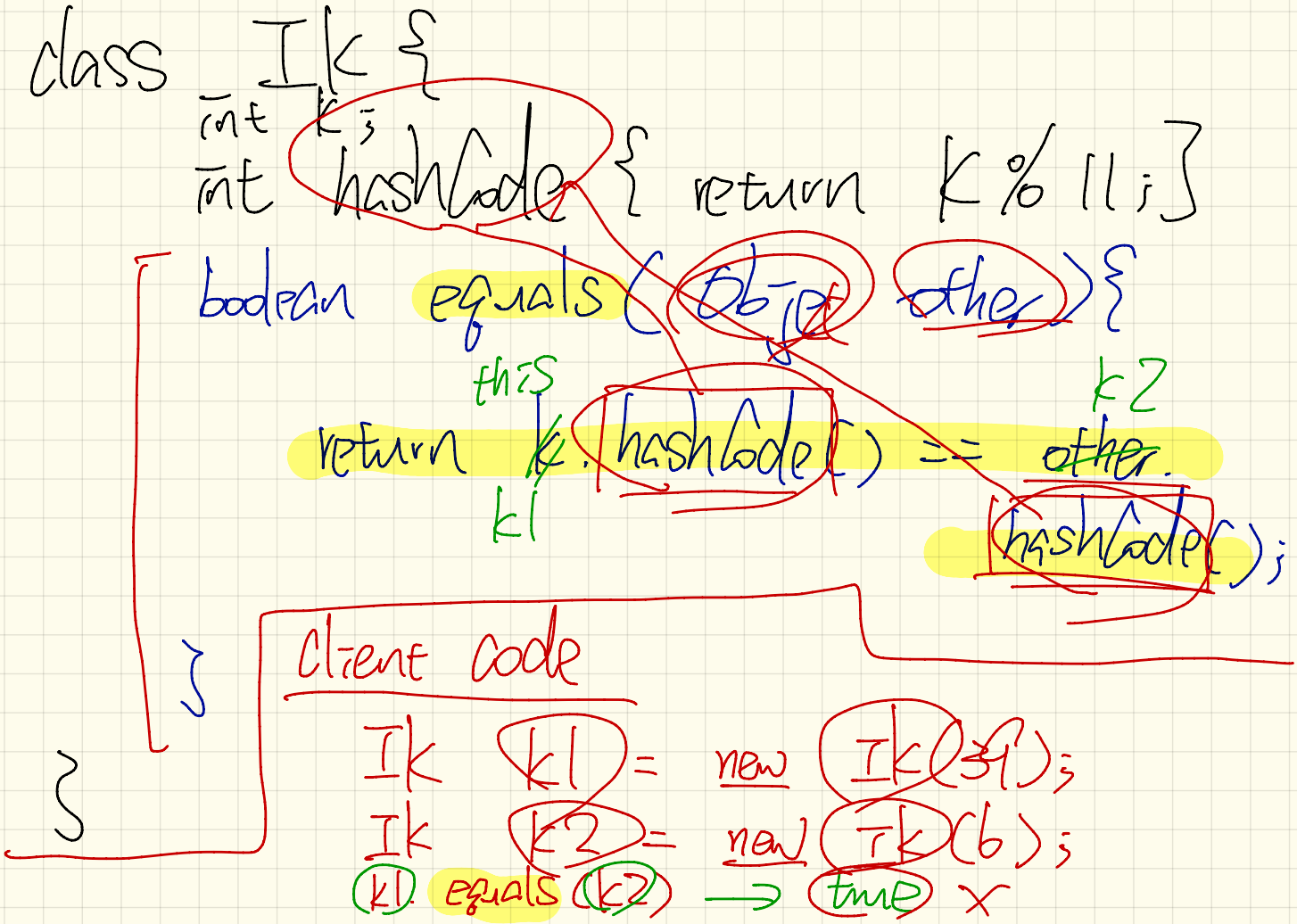


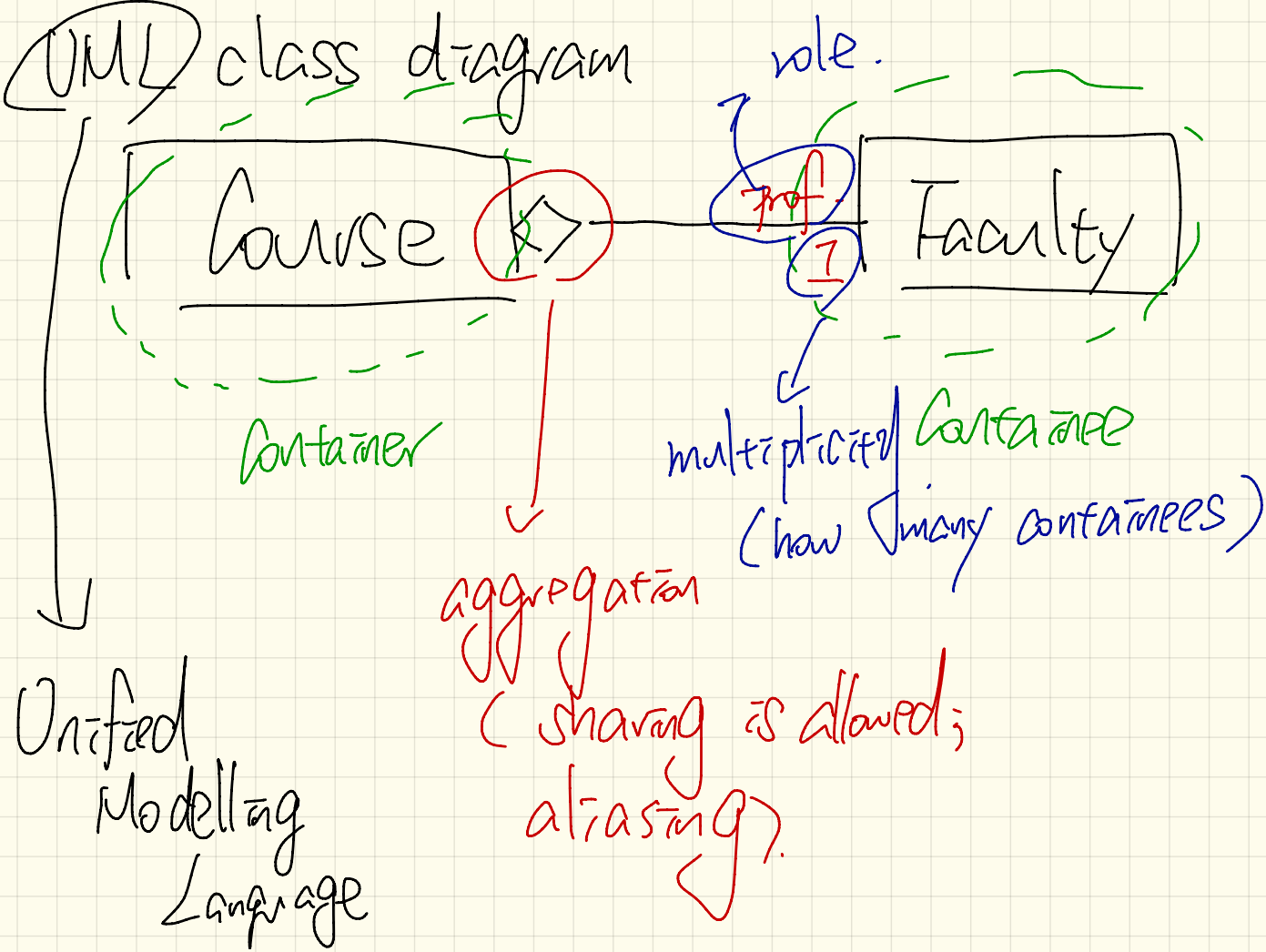
Cheap Computations

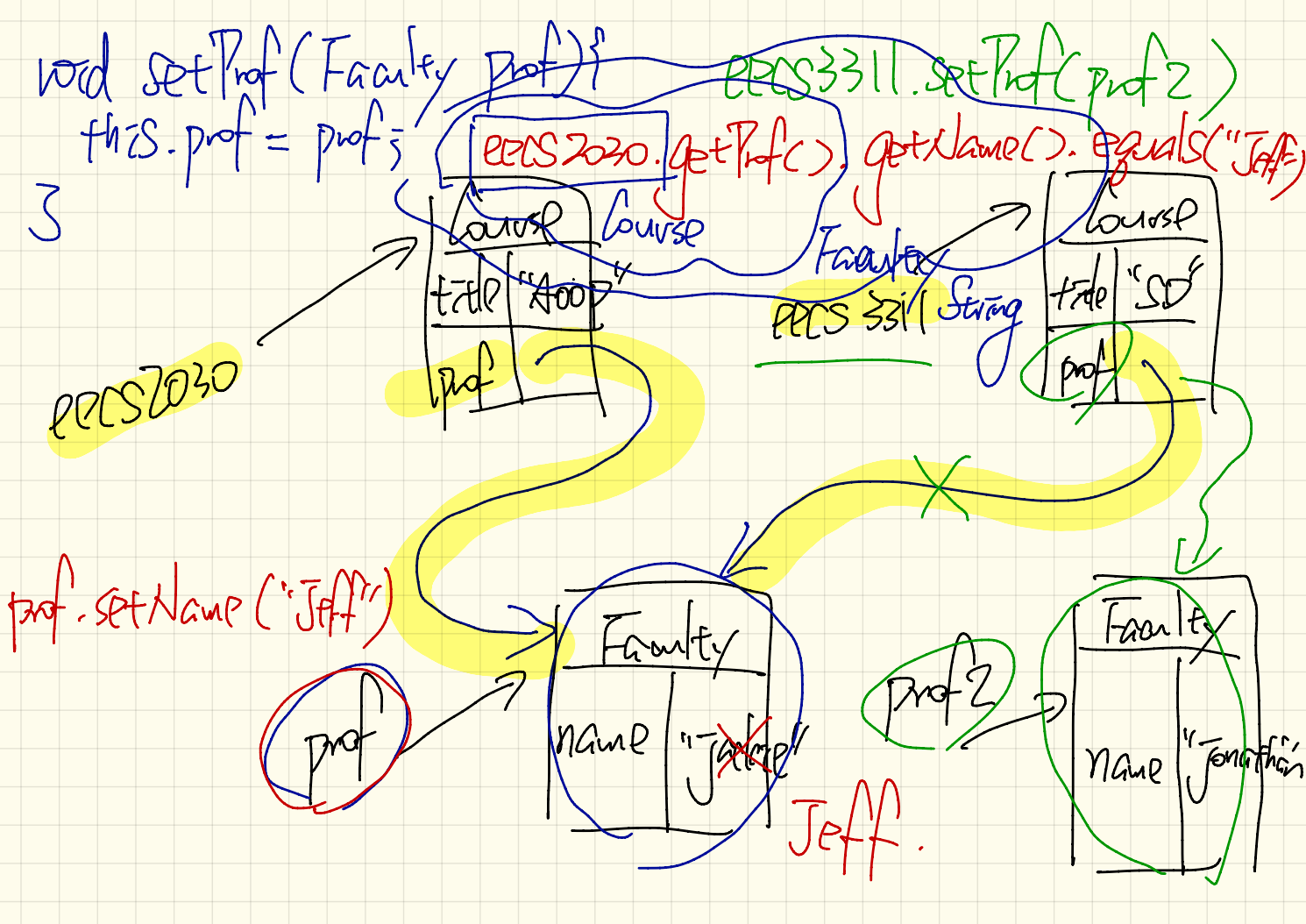
1. Arithmetic (K % 11)
2. Array indexing (A[i])

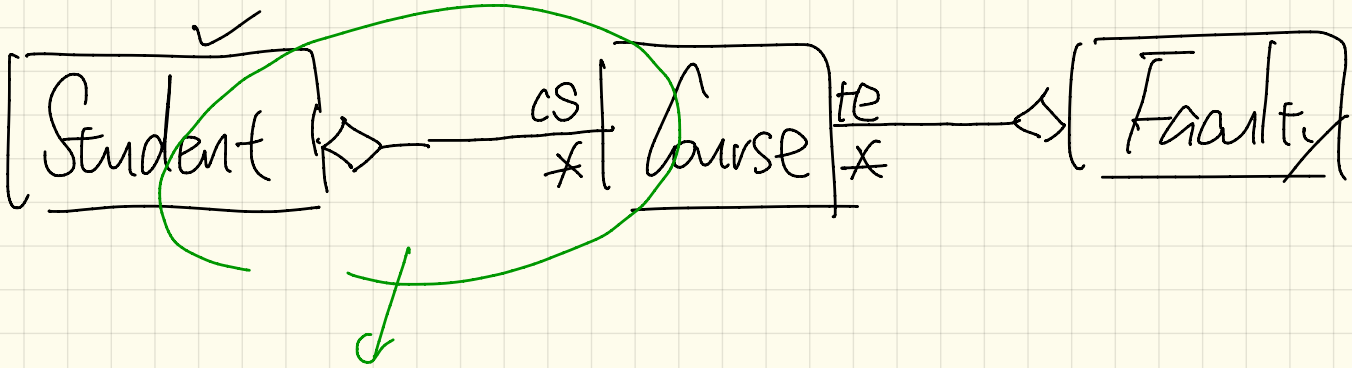
Lecture 10

Tuesday Oct. 10

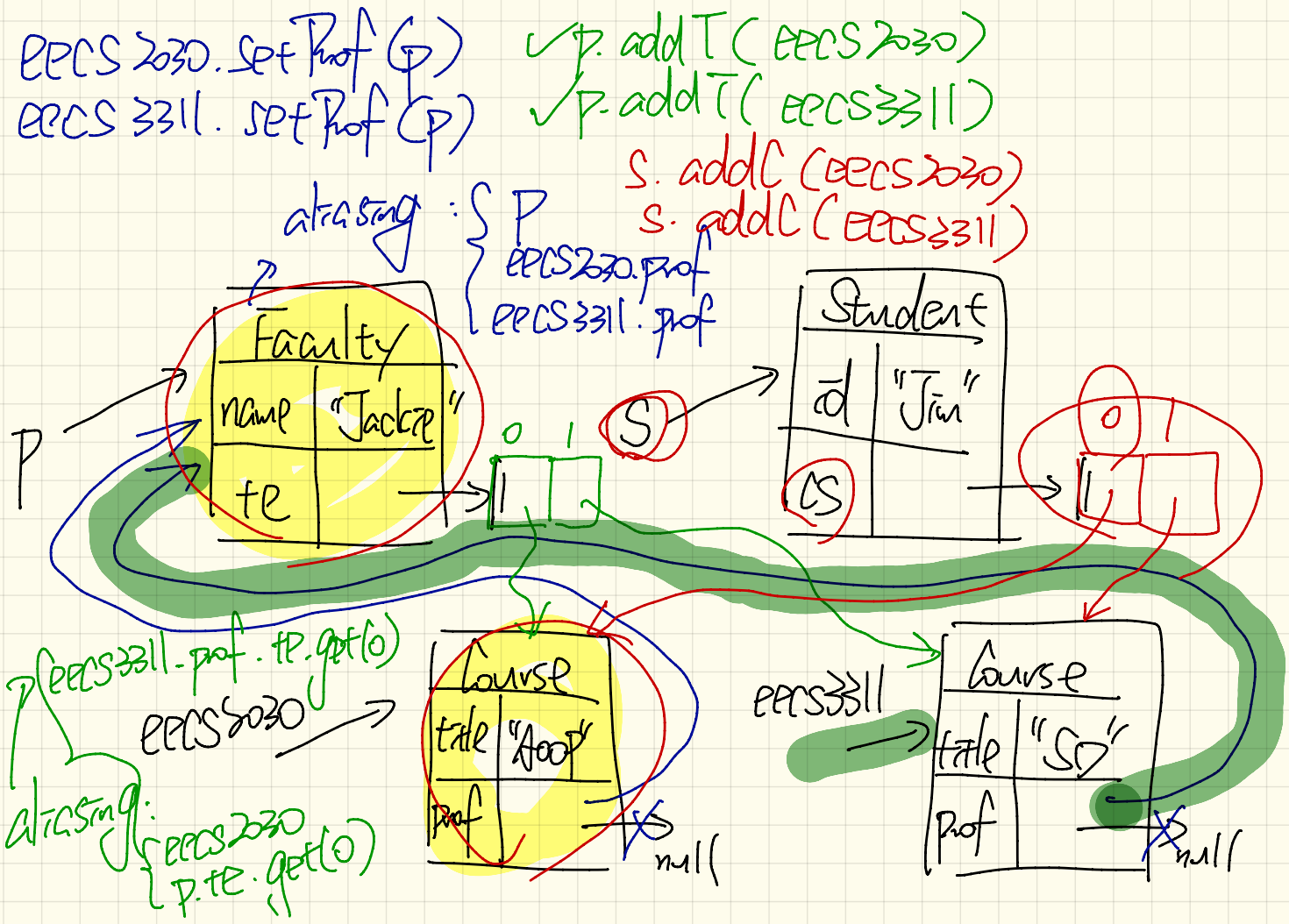

class Ik {
int k;
int hashCode { return k % 11; }
boolean equals (Object other){
this
k1
k2
return k.hashCode() == other.
hashCode();
}
Client code
Ik k1 = new Ik(3);
Ik k2 = new Ik(6);
k1 equals (k2) → true X
}


UML class diagram
role.
Course
<>
Prof.
1
Faculty
container
multiplicity
container
(how many containees)
aggregation
(sharing is allowed;
aliasing)
Unified
Modelling
Language


void setProf(Faculty prof){
this.prof = prof;
}
EECS2030.getProf().getName().equals("Jeff")
EECS3311.setProf(prof2)
Course
title "OOP"
prof
Course
title "SD"
prof
EECS2030
Faculty
EECS3311
String
prof.setName("Jeff")
prof
Faculty
name "Jeffrey"
Jeff.
prof2
Faculty
name "Jonathan"

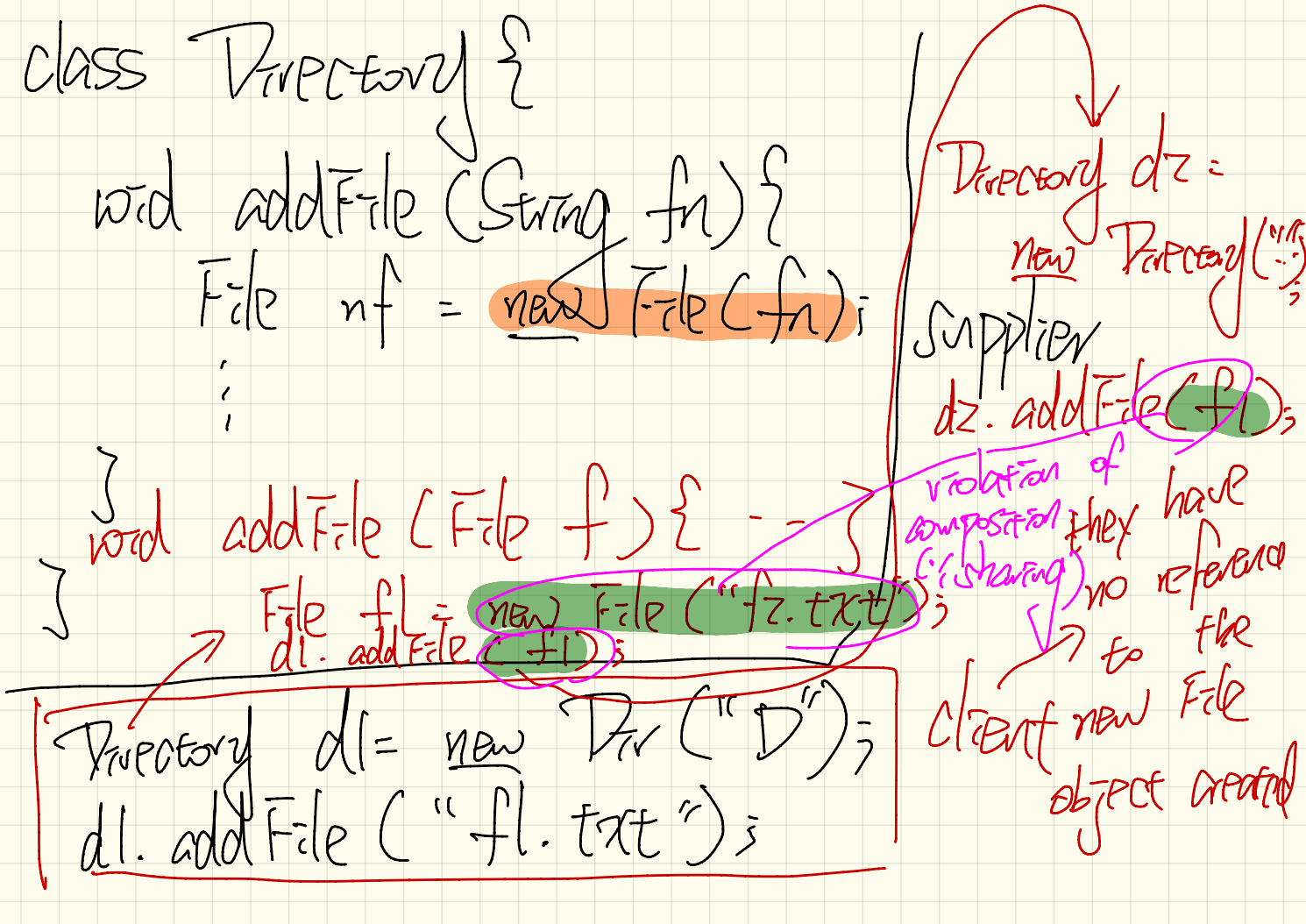


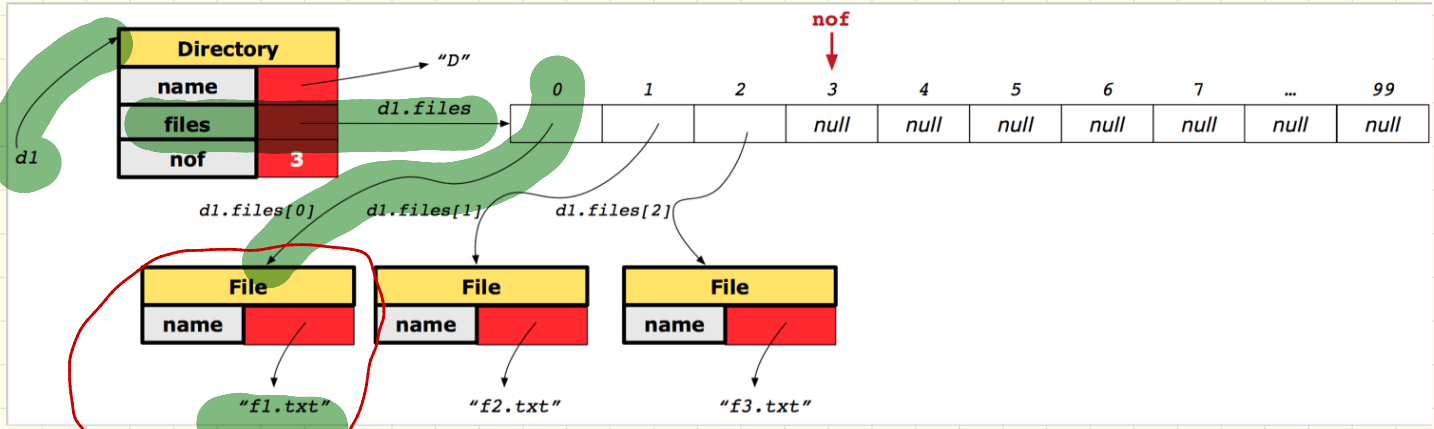
Each student has a collection of courses.


eecs2030.setProf(p)
eecs3311.setProf(p)
p.addT(eecs2030)
p.addT(eecs3311)
s.addC(eecs2030)
s.addC(eecs3311)
aliasing : { p
eecs2030.prof
eecs3311.prof
Faculty
name "Jackie"
te
p
0 1
S
Student
id "Jim"
cs
0
1
Course
title "Foo"
prof
null
eecs3311
Course
title "SD"
prof
null
p(eecs3311.prof.te.get(0)
eecs2030
aliasing: { eecs2030
p.te.get(0)

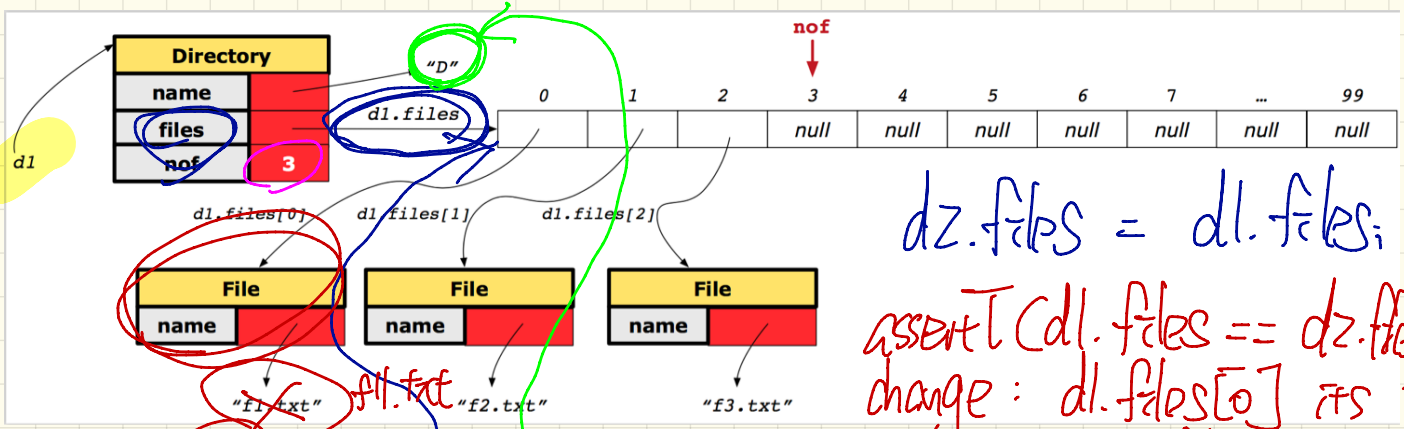
Lecture 11

Thursday Oct. 12


class Directory {
void addFile(String fn) {
File nf = new File(fn);
}
void addFile(File f) {
File fl = new File("fz.txt");
dl.addFile(fl);
}
}
Directory dl = new Dir("D");
dl.addFile("fl.txt");
Directory dz = new Directory("...");
supplier
dz.addFile(fl);
violation of composition (sharing)
they have no reference to the client new File object created



this file object is only referenced from d1.
(no sharing via aliasing).



dz.files = d1.files;

assert(d1.files == dz.files)
change: d1.files[0] its name to "f1.txt"

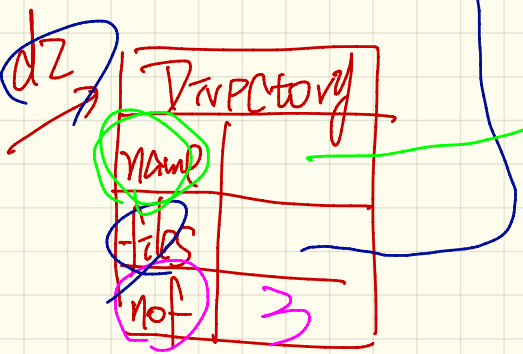
Directory dz = new Directory(d1);

call by ref.
other 是 a copy of d1.

class Directory {
  Directory(Directory other){
    this.nof = other.nof;
    this.files = other.files;
    this.name = other.name;
  }
}

primitive 2
ref 3 4

this: dz
other: d1



class CEmployee1 implements Comparable<...>{

int compareTo(CEmployee1 other){

return this.id - other.id;

}

}

alan

other

After Arrays.sort:

tom, alan, mark

mark.compareTo(alan) |

CEmployee objects:

	ids
alan	2
mark	3
tom	1

java

alan.compareTo(mark) -1

tom.compareTo(alan) -1

math

alan < mark

tom < alan

tom < mark

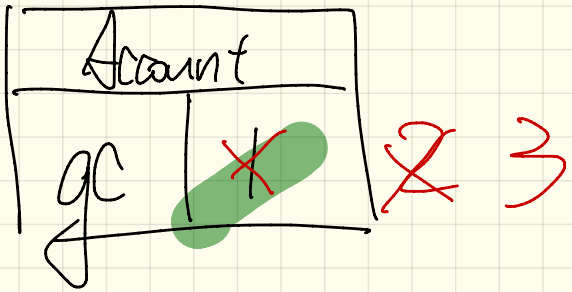
alan 5000

mark 3000

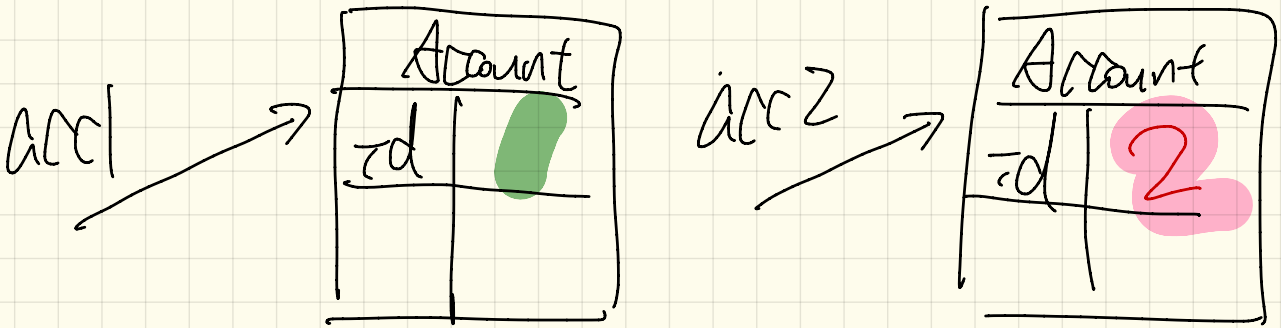
Double.compare(alan.salary, mark.salary)

+ returning this as the result for compareTo in CEmployee2 is not right

static int globalCounter



acc1 = new Account("Jim");



```
class Account {
    String branchName;
    static int c;
    static void m() {
        branchName
    }
}
```

```
Account acc = new Account("B1");
acc.branchName;
```

```
Account.m()
```

Lecture 12

Tuesday Oct. 17

RT = Summation of RTs of all primitive operations

$$= \sum_{i=1}^{N} t_i(PM_i)$$

$$= C \cdot \sum_{i=1}^{N} PM_i = C \cdot N \approx N$$

2 ops.

op1. t(op1)

op2. t(op2)

Prog A more efficient than Prog B

⇒ # P.O. of A ≤ # P.O. of B

Example: Counting # of primitive operations.

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

i	i < n
1	T
2	T
⋮	
n-1	T
n	F

e.g. findMax({2,3,4}, 3)

input size (a.length).

$7 \cdot n^1 + 2 \cdot n^0$

$3n - 2$

n == 10

i	i < 10
1	T
2	T
⋮	
9	T
10	F

Example: Bounding Function.

n	8n + 5	9n
1	13	9
2	21	18
3	29	27
4	37	36
5	45	45
6	53	54
...		

f(n) (code)

input size

$f(n) = 8n + 5$ (your code)

$g(n) = n$

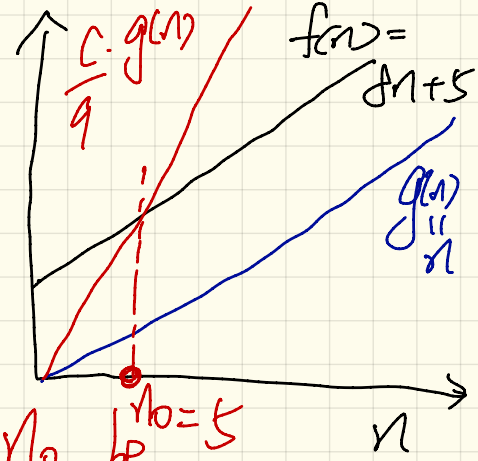
Show that f(n) can be bounded by g(n):

choose $c = 9$.

$9 \cdot \underset{\sim}{n} \rightarrow g(n)$

$\underset{\sim}{c}$

RT



What should n_0 be?

Starting from which

$$f(n) \leq \underset{\sim}{c} \cdot g(n)$$

before $n_0 = 5$, no upper bound effect!

$f(n) = 3n + 5$

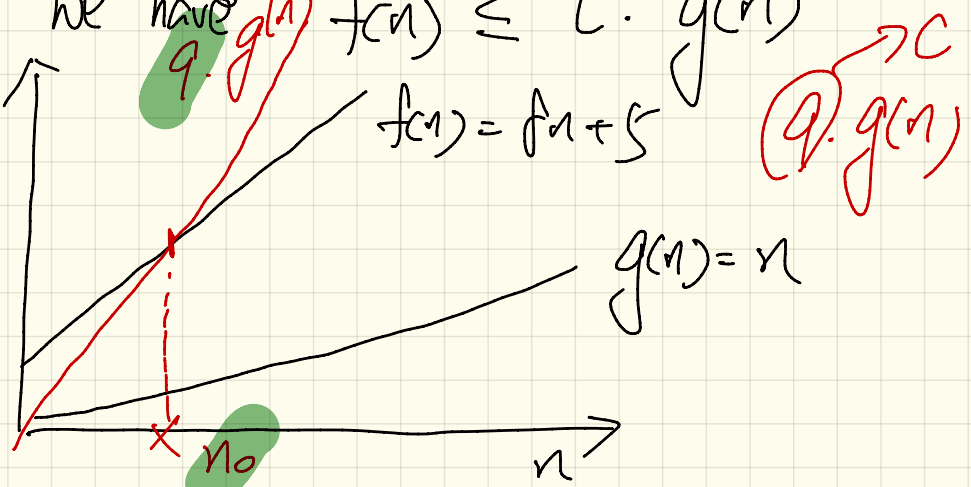
Show: $f(n)$ is $O(n)$

$g(n)$

Show that starting from n_0,

we have $f(n) \leq c \cdot g(n)$

RT



$$f(n) = a_0 \cdot n^0 + a_1 \cdot n^1 + \cdots + a_d \cdot n^d$$

$$\leq a_0 \cdot n^d + a_1 \cdot n^d + \cdots + a_d \cdot n^d$$

$$\leq (|a_0| + |a_1| + \cdots + |a_d|) \cdot n^d$$

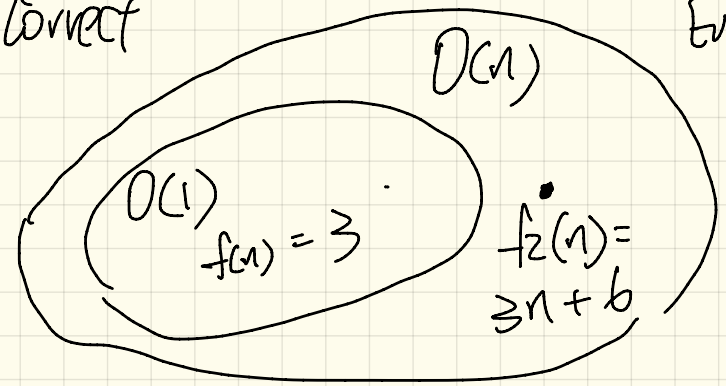
n is input size

$\Rightarrow n > 0$

$\Rightarrow n^0 < n^1 < n^2 < n^3 \cdots$ ∎

3 is $O(1)$ correct

is $O(n)$



Every function that is $O(1)$ is also $O(n)$

Not vice versa.

Example: Determining Asymptotic Running Time

```
1  maxOf (int x, int y) {
2    int max = x;
3    if (y > x) {
4      max = y;
5    }
6    return max;
7  }
```

Lecture 13

Thursday Oct. 19

Example: Determining Asymptotic Running Time

```
1  containsDuplicate (int[] a, int n) {
2    for (int i = 0; i < n; ) {
3      for (int j = 0; j < n; ) {
4        if (i != j && a[i] == a[j]) {
5          return true; }
6        j ++; }
7      i ++; }
8    return false; }
```

O(1)

$\bar{i} = n-1$

$\bar{j} = 0 \ldots n-1$ (n)

Body of loop takes O(1) $\begin{cases} \bar{i} = 0 \\ \bar{j} = 0 \ldots n-1 \end{cases}$ (n)

How many times? n^2 $\begin{cases} \bar{i} = 1 \\ \bar{j} = 0 \ldots n-1 \end{cases}$ (n)

RT: $O(1) * n^2 = O(n^2)$

Example: Determining Asymptotic Running Time

```
1  sumMaxAndCrossProducts (int[] a, int n) {
2    int max = a[0];                                  O(1)
3    for(int i = 1; i < n;) {                         O(n)
4      if (a[i] > max) { max = a[i]; }
5    }
6    int sum = max;                                   O(1)
7    for (int j = 0; j < n; j ++) {                   O(n^2)
8      for (int k = 0; k < n; k ++) {
9        sum += a[j] * a[k]; } }
10   return sum; }                                    O(1)
```

RT: $O(n + 1 + n^2 + 1) = O(n^2)$.

n^2 dominates the RT

Example: Determining Asymptotic Running Time

```
1  triangularSum (int[] a, int n) {
2    int sum = 0;
3    for (int i = 0; i < n; i ++) {
4      for (int j = i; j < n; j ++) {
5        sum += a[j]; } }
6    return sum; }
```

$\bar{I} = n-1$

$\bar{J} = n-1 \ldots n-1$ (1)

$O(1)$

$\bar{J} = \bar{I}$

$\bar{I} = 0$

$\bar{J} = \frac{\bar{I}}{0} \ldots n-1$ (n)

How many iterations?

$n + (n-1) + (n-2) + \ldots + 1 = \frac{n \cdot (n+1)}{2}$

$O(n^2) = n^2$

$\bar{I} = 1$

$\bar{J} = 1 \ldots (n-1)$ (n-1)

Running Time of Inserting into an Array [a, b]

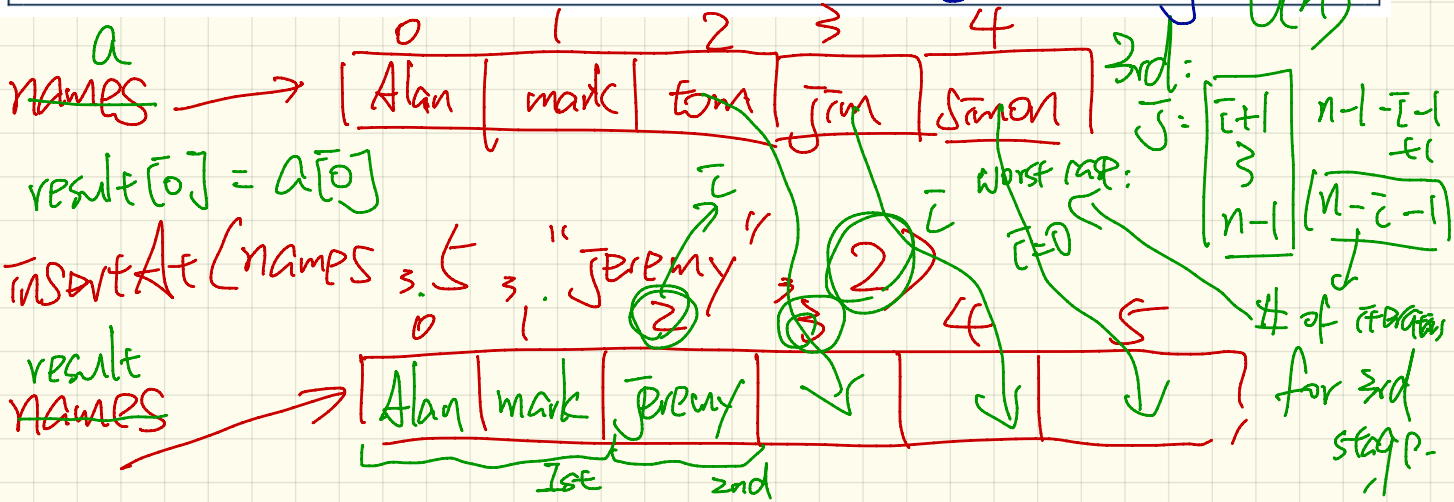
```
insertAt(String[] a, int n, String e, int i)
  String[] result = new String[n + 1];
  for(int j = 0; j < i; j ++){ result[j] = a[j]; }
  result[i] = e;
  for(int j = i + 1; j < n; j ++){ result[j] = a[j-1]; }
  return result;
```

a.length

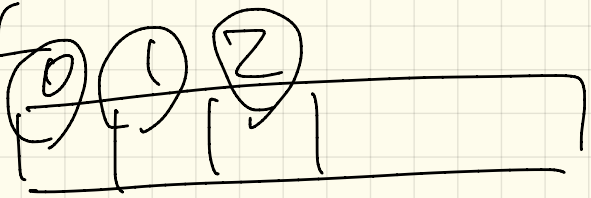
1st: O(1)

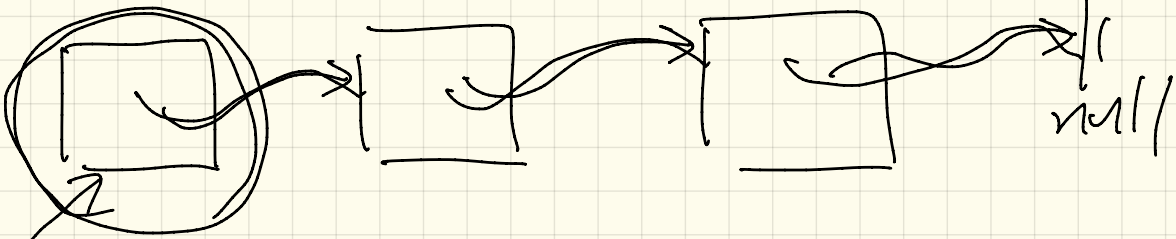
2nd: O(1) — depends on i

3rd: O(n)



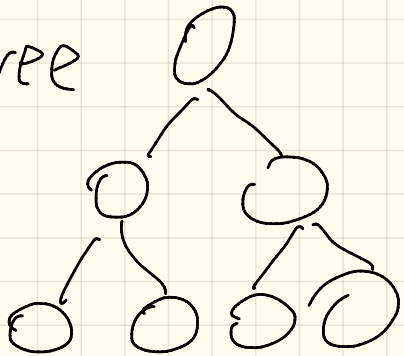
linear



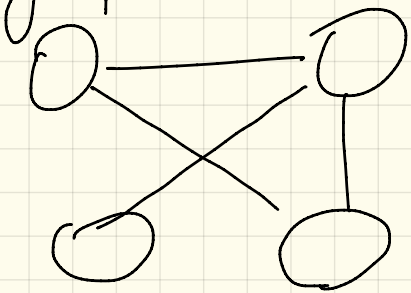


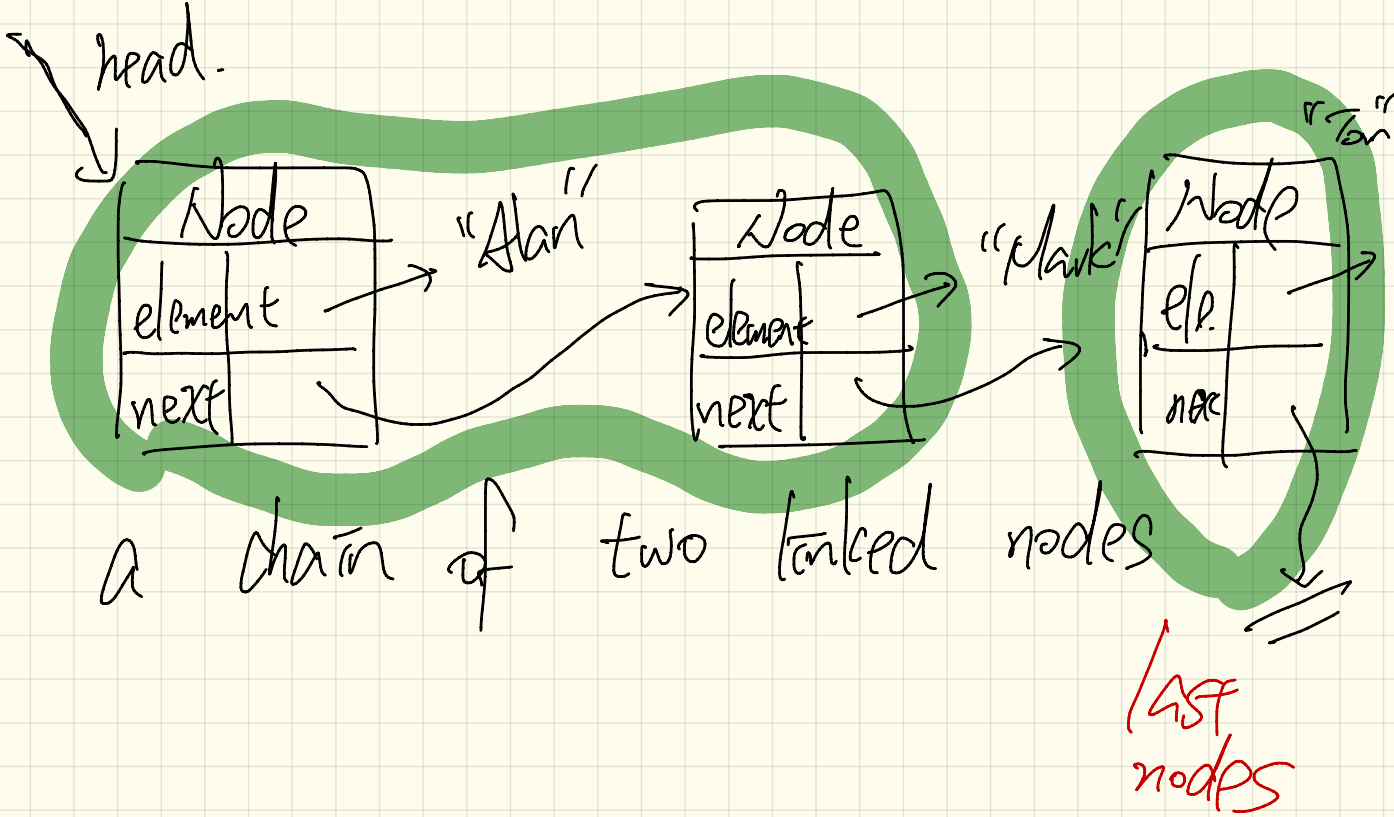
non-linear

tree

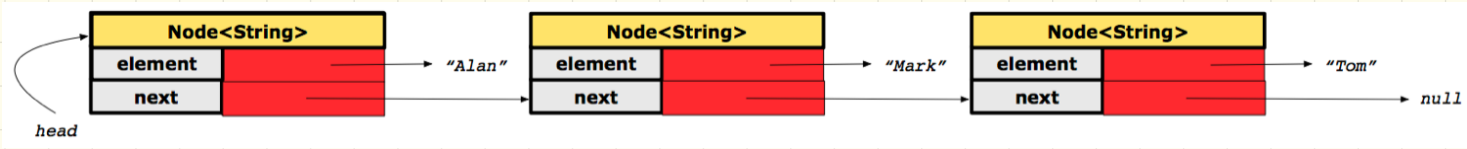


graph





a chain of two linked nodes




```
public class Node {
   private String element;
   private Node next;
   public Node(String e, Node n) { element = e; next = n; }
   public String getElement() { return element; }
   public Node getNext() { return next; }
   public void setNext(Node n) { next = n; }
}
```

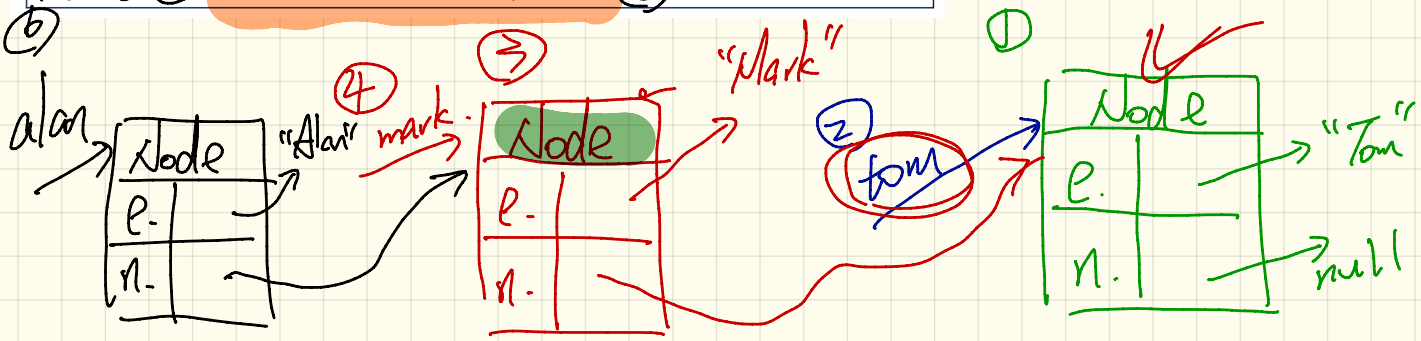
mark

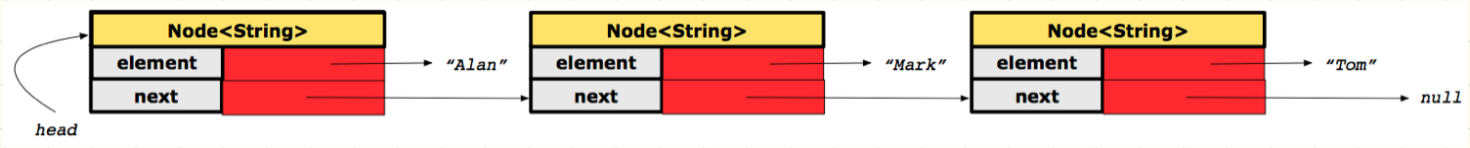
alan.next == mark

Node alan = new Node ("Alan", mark)

Approach 1

```
Node tom = new Node("Tom", null);
Node mark = new Node("Mark", tom);
Node alan = new Node("Alan", mark);
```






```
public class Node {
  private String element;
  private Node next;
  public Node(String e, Node n) { element = e; next = n; }
  public String getElement() { return element; }
  public Node getNext() { return next; }
  public void setNext(Node n) { next = n; }
}
```

Approach 2

```
Node alan = new Node("Alan", null);
Node mark = new Node("Mark", null);
Node tom = new Node("Tom", null);
alan.setNext(mark);
mark.setNext(tom);
```

Lecture 14

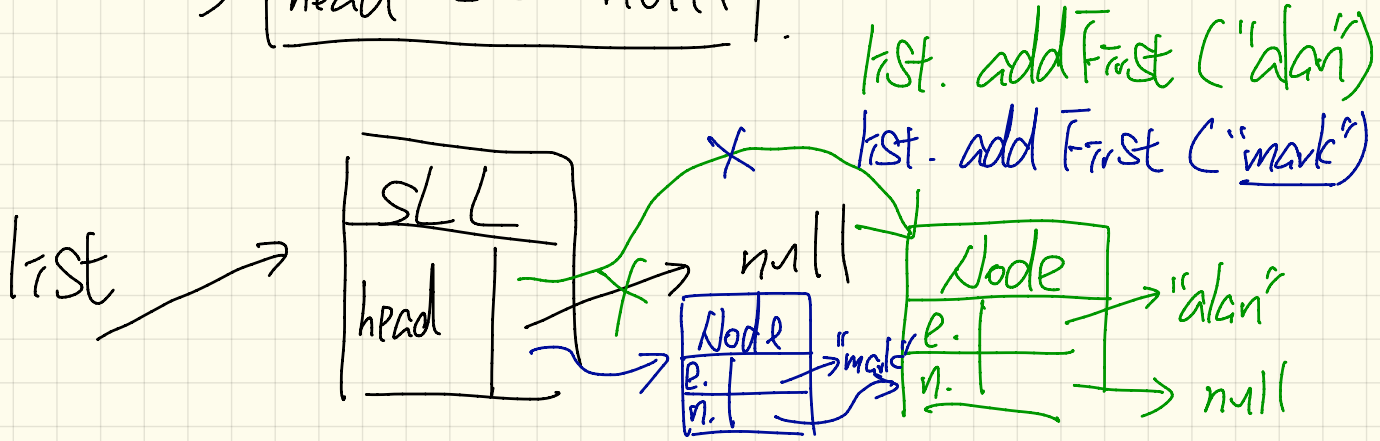
Tuesday Oct. 24

SLL empty

```
SLL list = new SLL();
```

↳ create an empty list

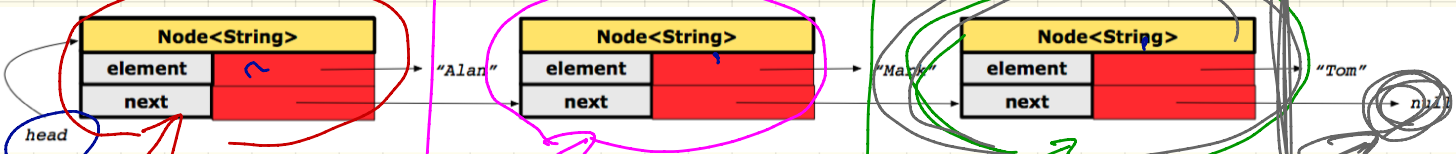
↳ `head == null`.



```
1  int getSize() {
2    int size = 0;
3    Node current = head;
4    while (current != null) {
5      /* exit when current == null */
6      current = current.getNext();
7      size ++;
8    }
9    return size;
10 }
```

① As soon as current becomes null, exit from the loop.

list.getSize() returns 3



current ①

current size becomes 1

current size becomes 2

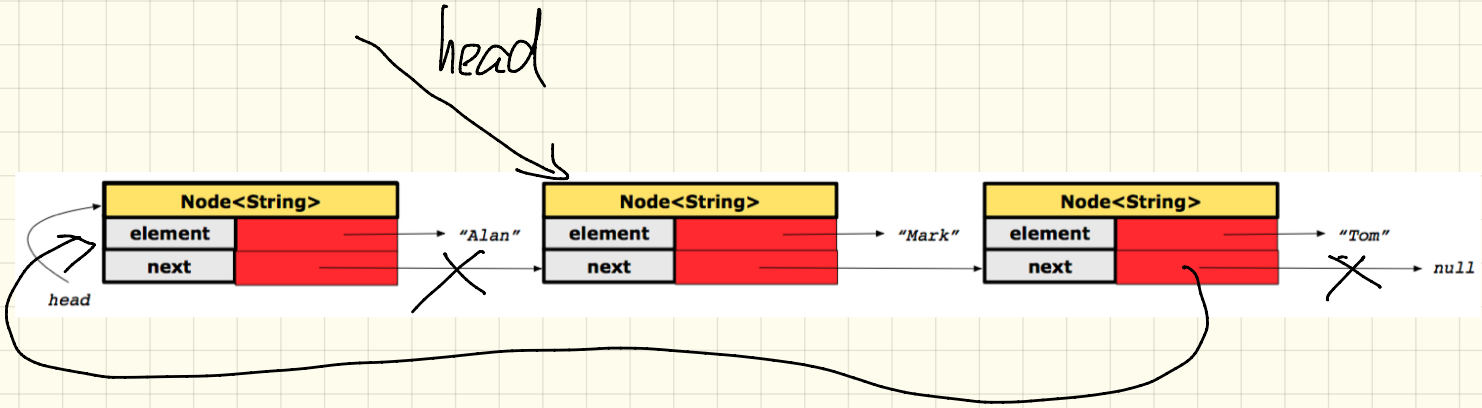
size becomes 3

current = current.getNext() ②

current = current.getNext() ③

current = current.getN()

null

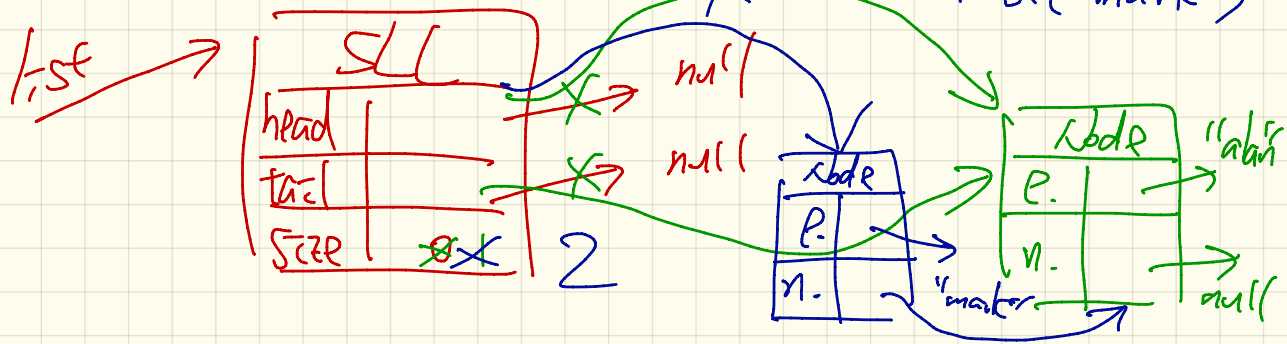


list.shiftToLeft()

```
1  Node getTail(Node head) {
2    Node current = head;
3    Node tail = null;
4    while (current != null) {
5      /* exit when current == null */
6      tail = current;
7      current = current.getNext();
8    }
9    return tail;
10 }
```

```
class SLL {
    Node head;
    Node tail;
    int size = 0;
    SLL() { head = null; tail = null; size = 0; }
}
SLL list = new SLL();
```

list.addFirst("alan")
list.addFirst("mark")



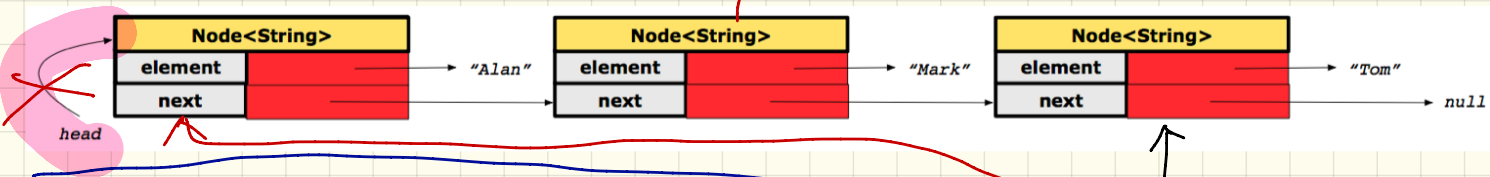
```
1  addFirst (String e) {
2    head = new Node(e, head);
3    if (size == 0) {
4      tail = head;
5    }
6    size ++;
7  }
```

Node nn = new Node(e, null);
nn.setNext(head);
head = nn;

list.addFirst("Jim")

special case for adding the very first node!

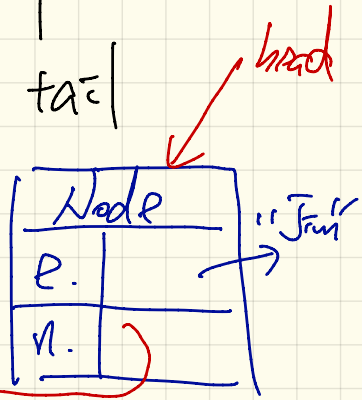
size == 0



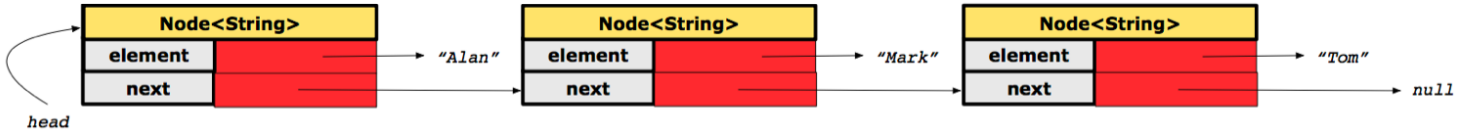

```
class Node {
    Node(String e, Node next) {
        this.element = e; this.next = next;
    }
}
```

head = new Node("Jim", head)

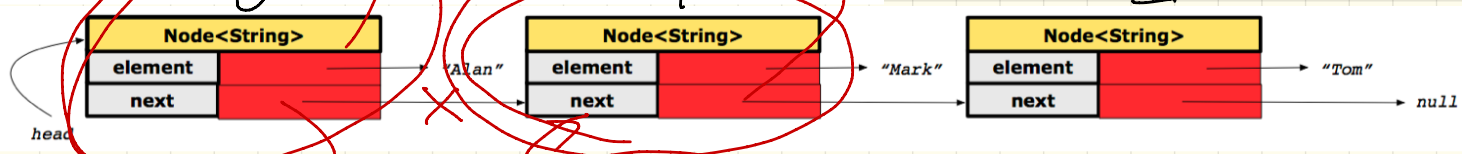
tail

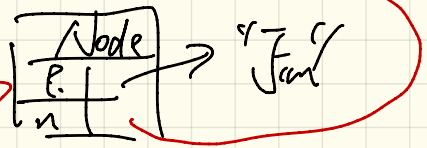


```
1  Node getNodeAt (int i) {
2    if (i < 0 || i >= size) {
3      throw IllegalArgumentException("Invalid Index");
4    }
5    else {
6      int index = 0;
7      Node current = head;
8      while (index < i) {  /* exit when index == i */
9        index ++;
10       /* current is set to node at index i
11        * last iteration: index incremented from i - 1 to i
12        */
13       current = current.getNext();
14     }
15     return current;
16   }
17 }
```



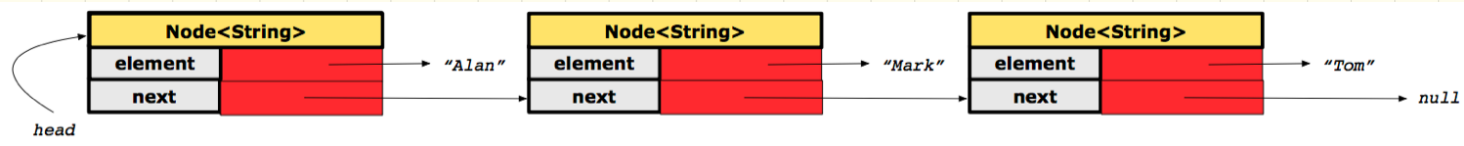
```
1  addAt (int i, String e) {
2    if (i < 0 || i >= size) {
3      throw IllegalArgumentException("Invalid Index.");
4    }
5    else {
6      if (i == 0) {
7        addFirst(e);
8      }
9      else {
10       Node nodeBefore = getNodeAt(i - 1);
11       newNode = new Node(e, nodeBefore.getNext());
12       nodeBefore.setNext(newNode);
13       size ++;
14     }
15   }
16 }
```





list.addAt(1, "Jean")

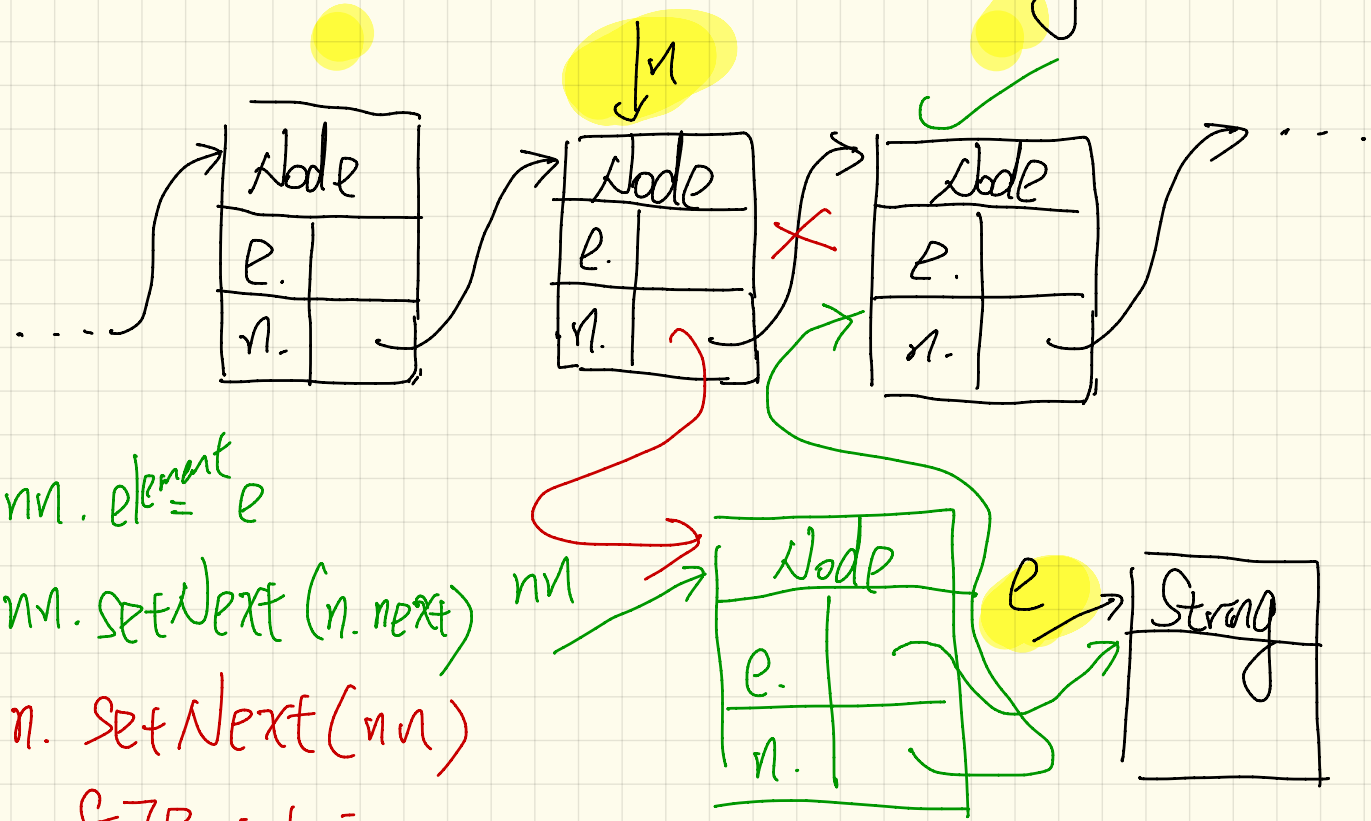
```
1  removeLast () {
2    if (size == 0) {
3      System.err.println("Empty List.");
4    }
5    else if (size == 1) {
6      removeFirst();
7    }
8    else {
9      Node secondLastNode = getNodeAt(size - 2);
10     secondLastNode.setNext(null);
11     tail = secondLastNode;
12     size --;
13   }
14 }
```



Lecture 15

Tuesday Oct. 31

void insertAfter(Node n, String e)



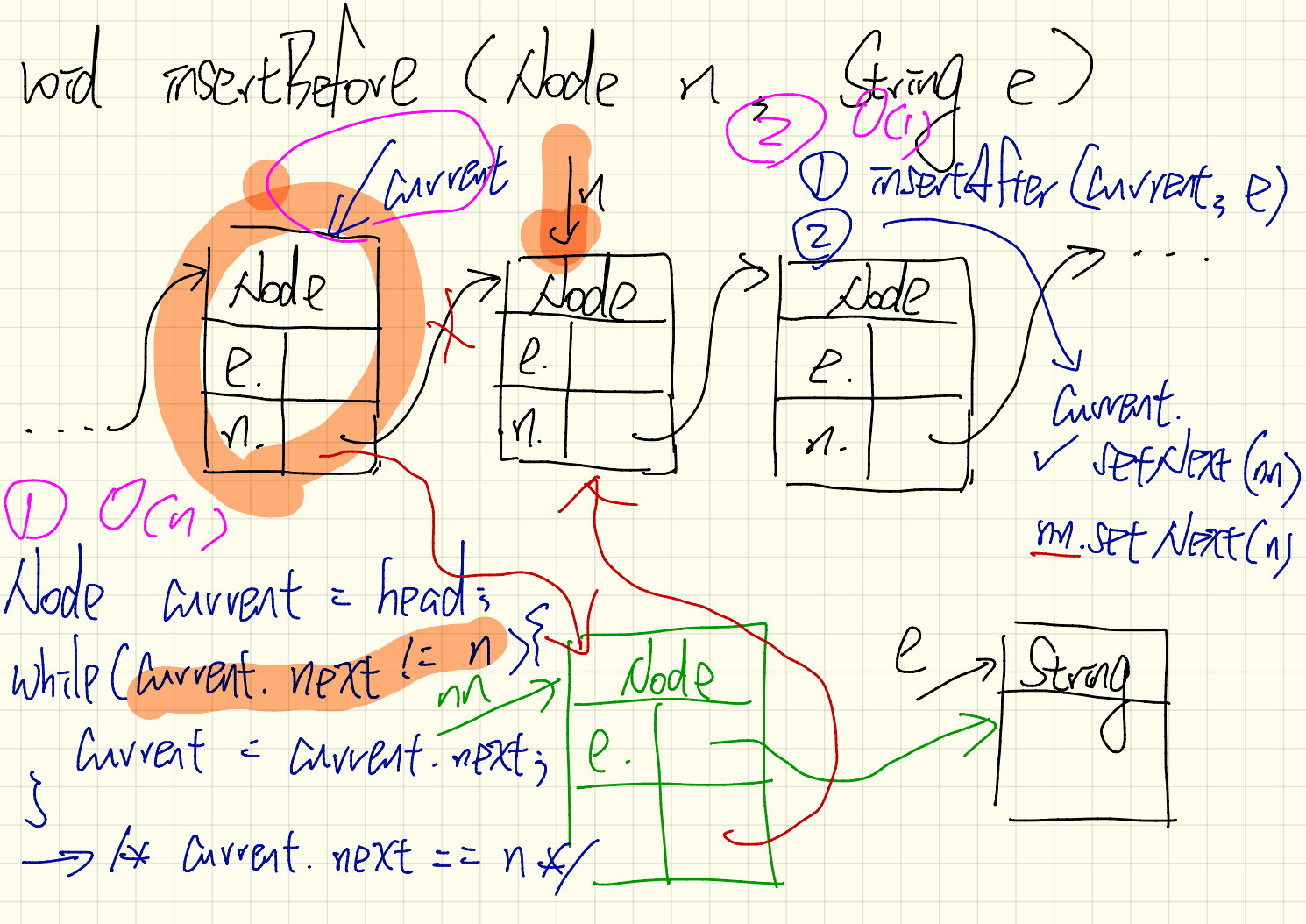
nn.element = e

nn.setNext(n.next)

n.setNext(nn)

size++;

void insertBefore (Node n, String e)



① O(n)

② O(1)

① insertAfter (current, e)

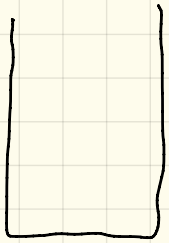
② current.
✓ setNext (nn)
nn.setNext(n)

```
Node current = head;
while (current.next != n){
    current = current.next;
}
→ /* current.next == n */
```

Stack Operations

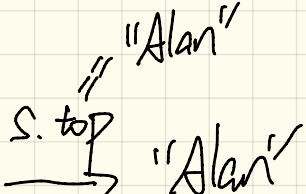
push, pop, top, size

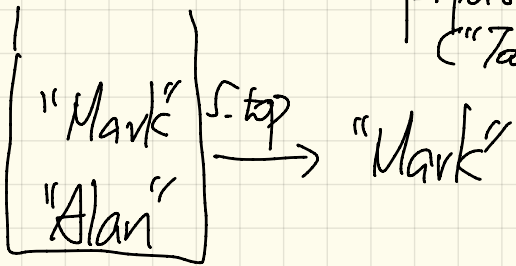
↳ ① return the current top
② remove the top

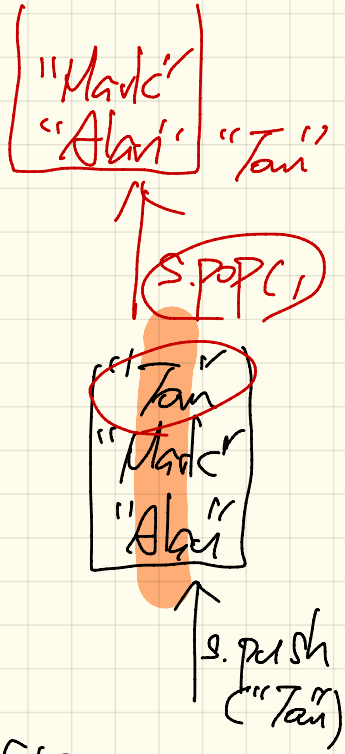


↓ S.push("Alan")

S.top → "Alan"







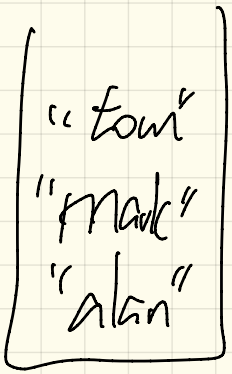
empty

Stack S

S.push("alan");

S.push("mark")

S.push("tom")



S.pop()

S.pop()

S.pop()

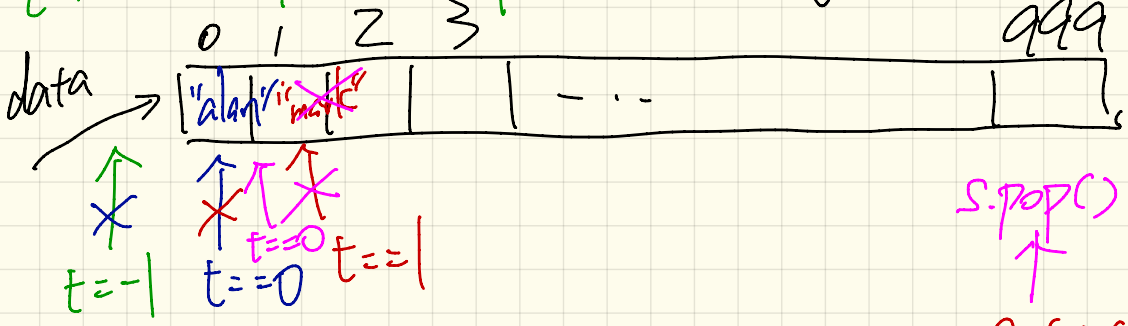
S.size() 0

"tom" "mark" "alan"

reverse order for pushing.

Implement a Stack using Array

t : index of the top.



s.pop() → data[t];
String ot = top();
data[t] = null;
t--;
return ot;

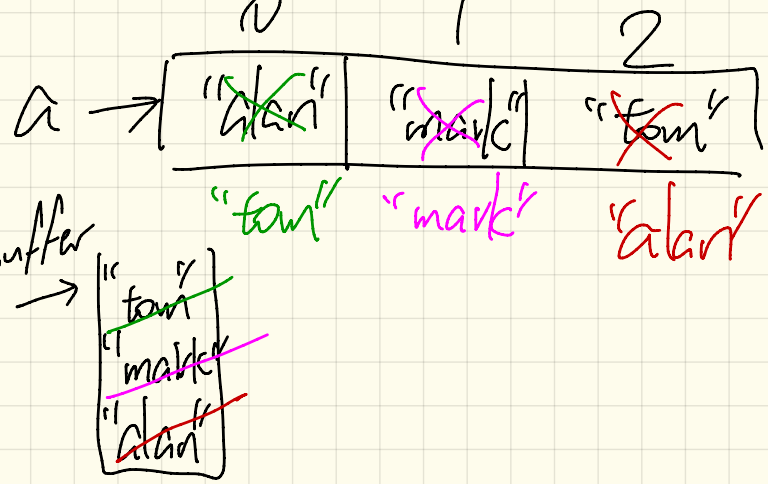
s.top() → data[t]

s.push("alan") → s.top() → s.push("mark")

↳ t++ ; data[t] = "alan"

↳ data[t]

↳ t++ ; data[t] = "mark"




```
int i=0;
while(!buffer.isEmpty()){
    a[i] = buffer.pop();
    i++;
}
```

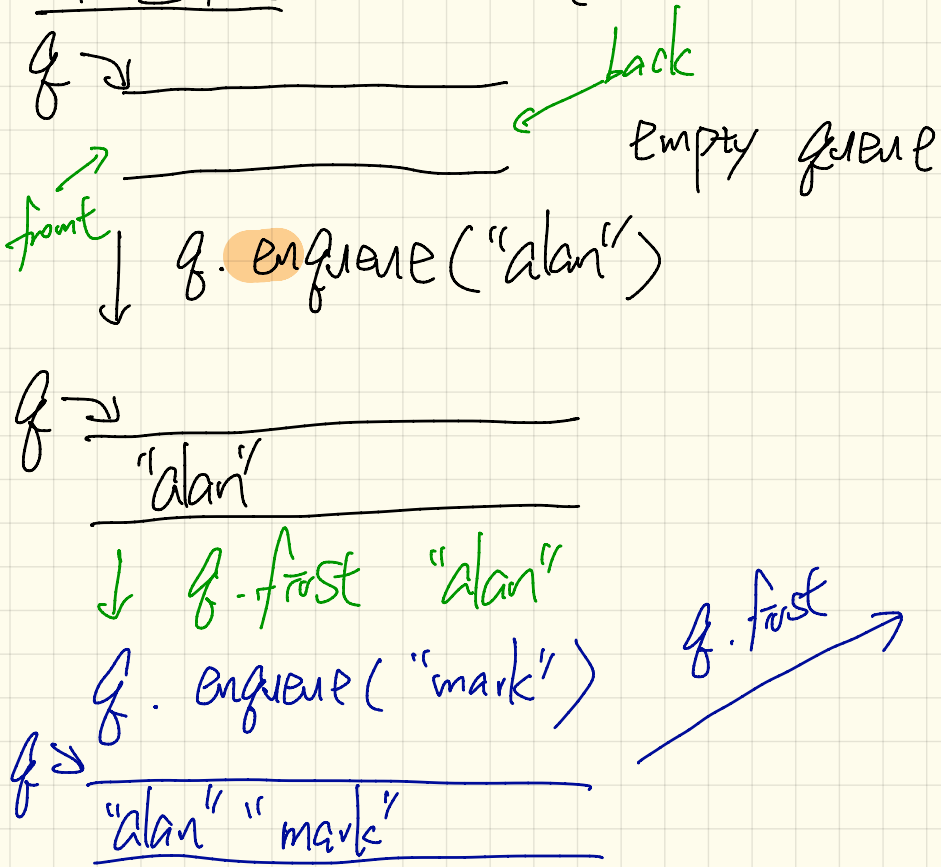
```
for(int i=0; i<a.length; i++){
    a[i] = buffer.pop();
}
```

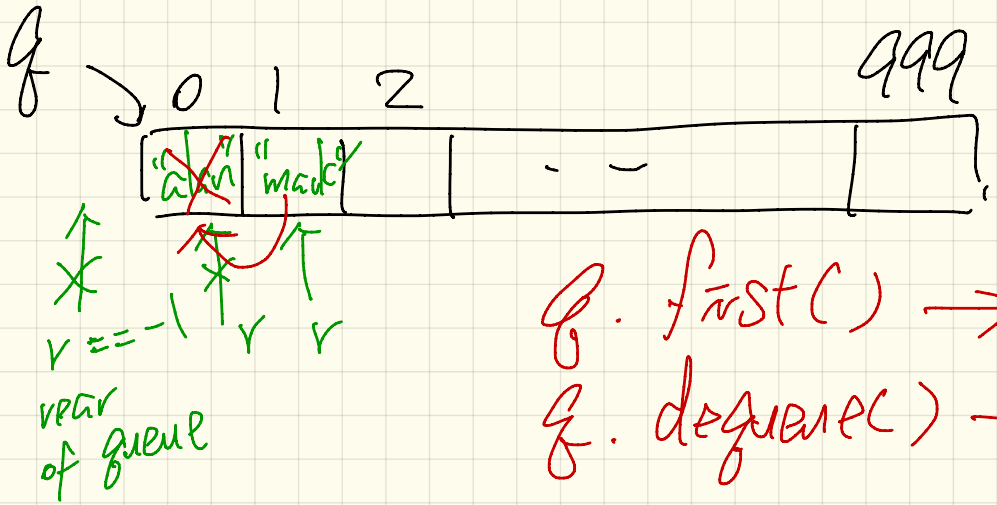
① i=0, "tom"

② i=1, "mark"

③ i=2, "alan"

FIFO Queue


q
back
empty queue
front
q.enqueue("alan")
q
"alan"
q.first "alan"
q.enqueue("mark")
q
"alan" "mark"
q.first
"alan"



q.first() → q[0]

q.dequeue() → O(n)

shift

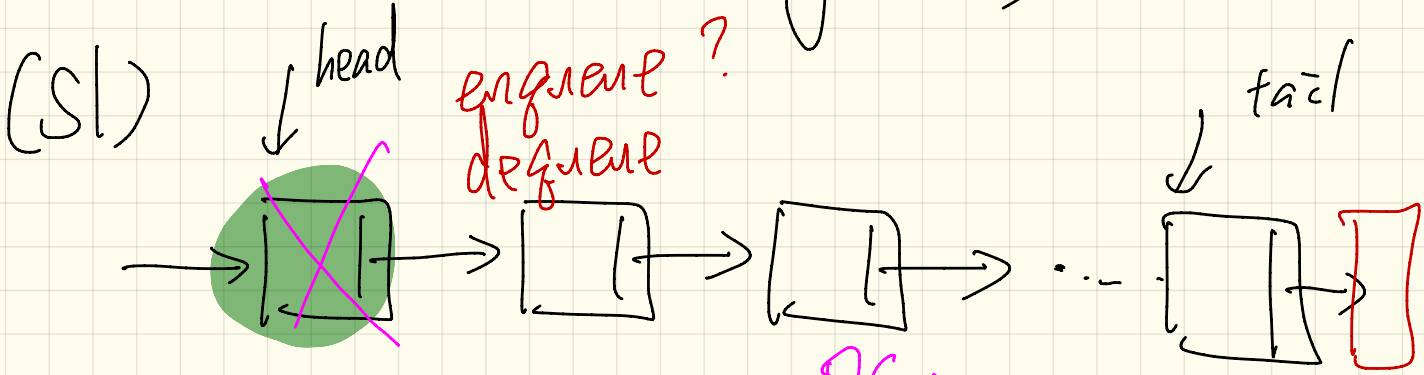
q.enqueue("alan") → q.enqueue("mavk")

↳ r++;
q[r] = "alan"

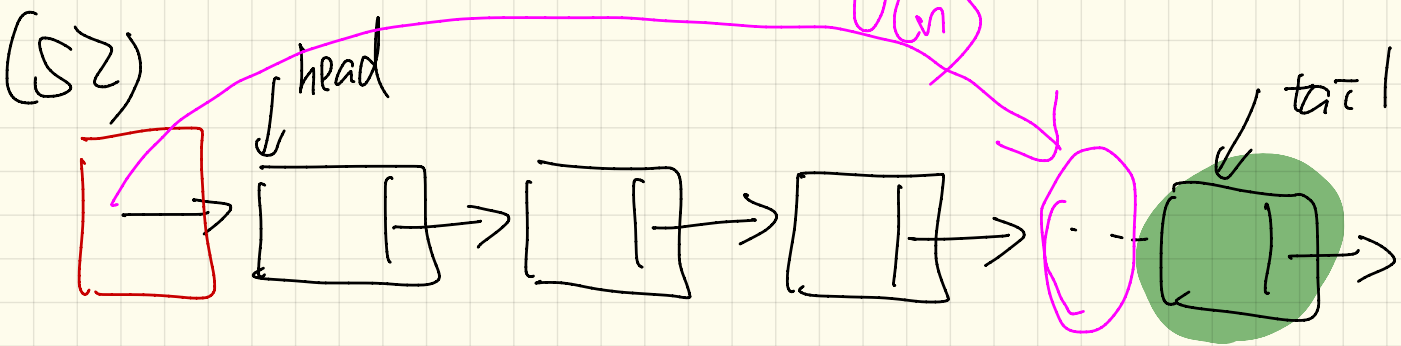
↳ r++;
q[r] = "mavk"

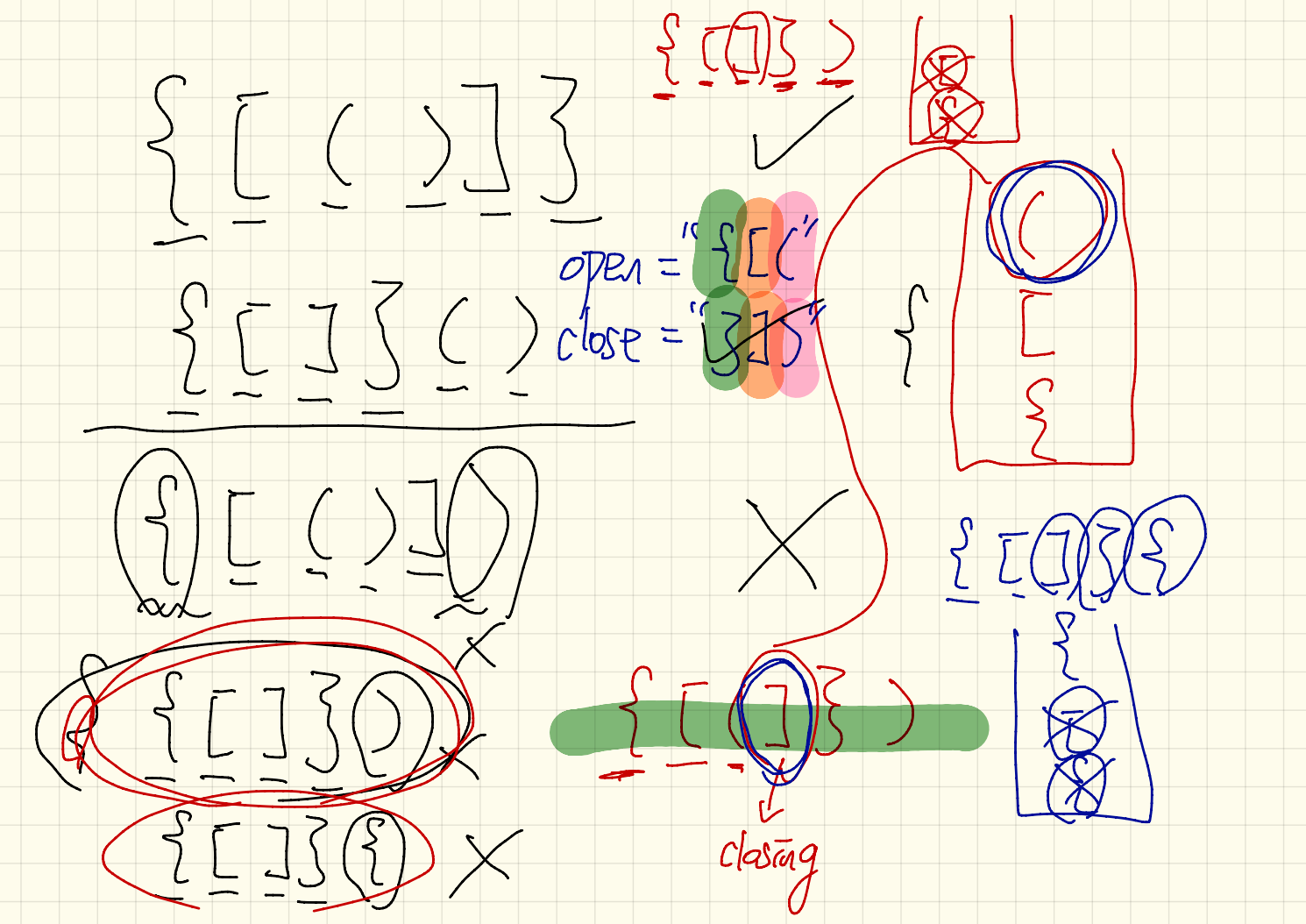
Implement the Q using SLL

(S1)



(S2)




open = "{[("
close = ")]}"
closing

Lecture 16

Thursday Nov. 2

$$P(n) = \frac{(1+n)n}{2}$$

$P(n)$ = sum of the first n integers from 1

Base Cases

$P(1)$ $P(2)$

Recursive / Inductive Cases

Assume: $P(n-1) = \frac{(1+(n-1))(n-1)}{2}$

Prove: $P(n)$ (derive)

Problem : $n!$

$5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$

$4!$

$7! = 7 * 6!$

size of problem

$$n! = \begin{cases} 1 & n = 0 \\ n \cdot [(n-1) \cdot (n-2) \cdots 1] & n > 0 \end{cases}$$

$(n-1)!$

$n! = n * (n-1)!$

① Base Cases

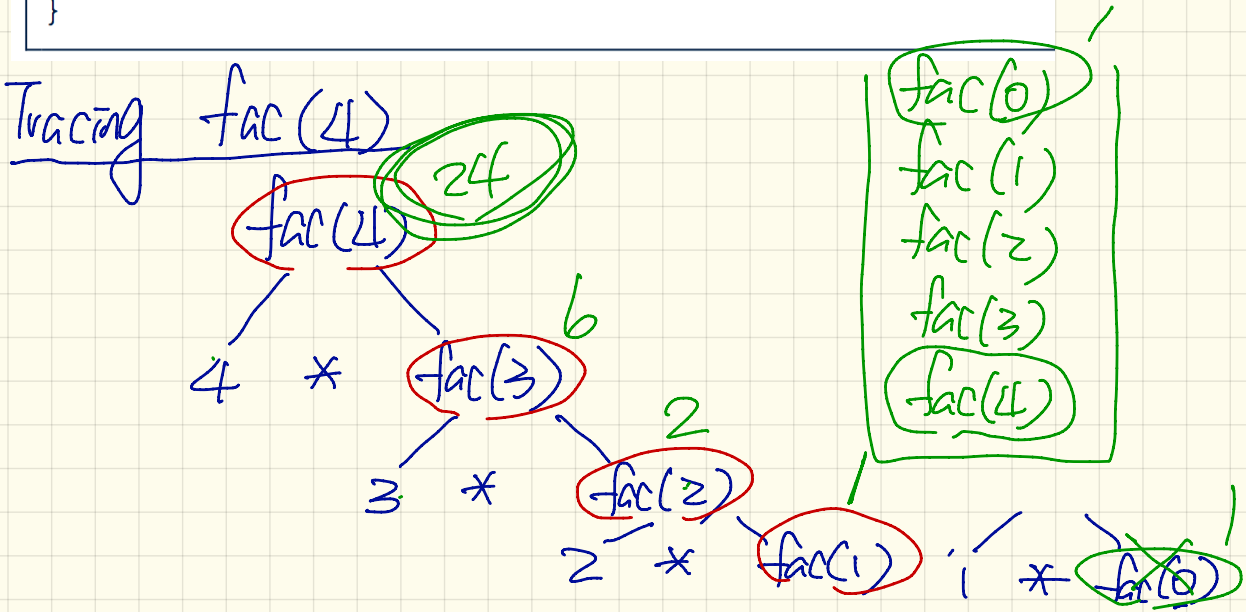
$0! = 1$ $\quad$ $1! = 1$

② Recursive Cases

Assume we have $(n-1)!$ — Solution to a strictly smaller problem.

Derive: $n! = (n-1)! * n$ — $(n-1)!$: sub-problem; $n!$: original problem

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



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

Tracing fac(3)

1. Activate fac(3), push (fac(3))
2. Execute fac(3)
   ↳ 3 * fac(2)
3. Activate fac(2), push (fac(2))
4. Execute fac(2)
   ↳ 2 * fac(1)
5. Activate fac(1)
   ↳ push (fac(1))
6. Execute fac(1)
   ↳ 1 * fac(0)
7. Activate fac(0)
   push (fac(0))
8. fac(0) returns 1
   pop()
   → fac(1) = 1 * 1
   = 1
9. fac(1) returns 1
   pop()
10. fac(2) returns 2

11. fac(3) returns 6
    pop()

Fib. Seq.

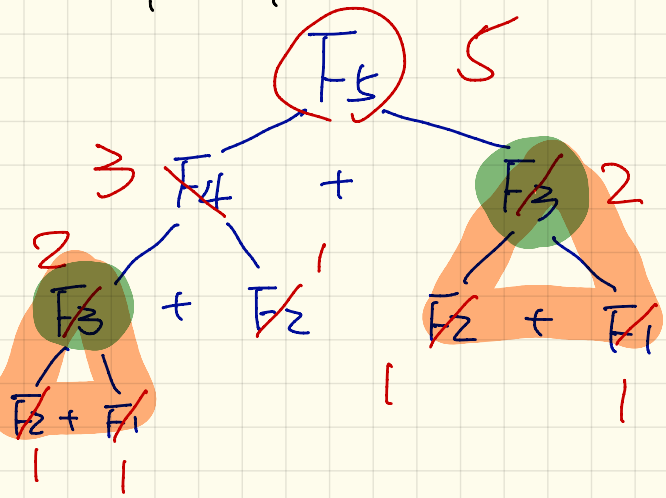
1 1 2 3 5 . . .

F_1 F_2 $F_1 + F_2$

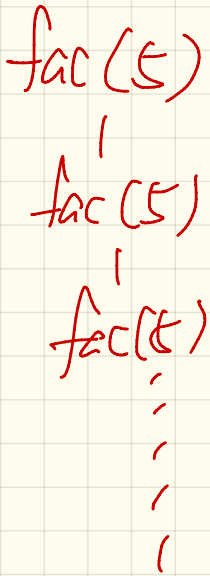
$$F_n = \begin{cases} 1 & n = 1 \\ 1 & n = 2 \\ F_{n-1} + F_{n-2} & n > 2 \end{cases}$$

F_{n-1}: solution to a strictly smaller problem

F_{n-2}: solution to smaller problem



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



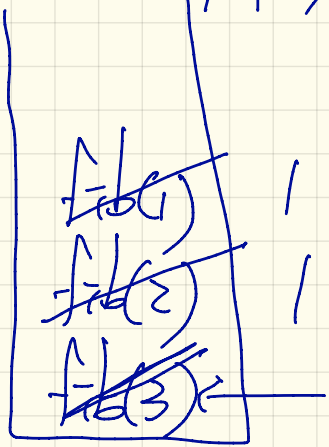
```
int fib(int n) {
  int result;
  if (n == 1) { /* base case */ result = 1; }
  else if (n == 2) { /* base case */ result = 1; }
  else { /* recursive case */
    result = fib(n - 1) + fib(n - 2);
  }
  return result;
}
```

3 2 1

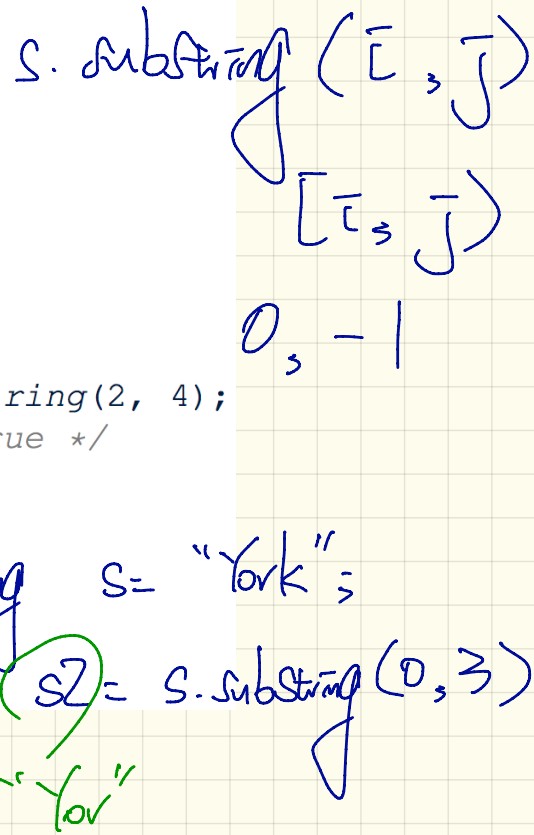
2 1

fib(3)

1. Activate fib(3)
   ↳ push(fib(3))
2. Execute top of stack fib(3)
   ↳ fib(2) + fib(1)
3. Activate fib(2)
   push
4. Execute fib(2)
   ↳ return 1
   ↳ pop
5. Activate fib(1)
   ↳ push
6. Execute fib(1)
   return 1
   pop
7. Execute fib(3)
   return
   1+1 = 2
   pop



```
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();
  }
}
```



String S = "..."

$(0 \le i < S.length() - 1)$

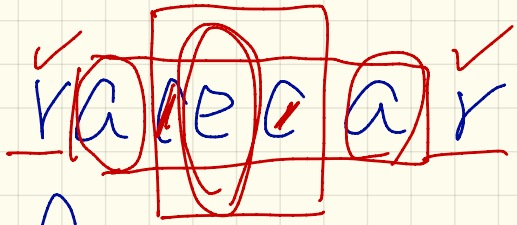
S.substring(0, i) + S.substring(i, S.length() ~~-1~~)

S = "York" S.length() == 4

S.substring(0, 2) + S.substring(2, 4)

0 1 2 3

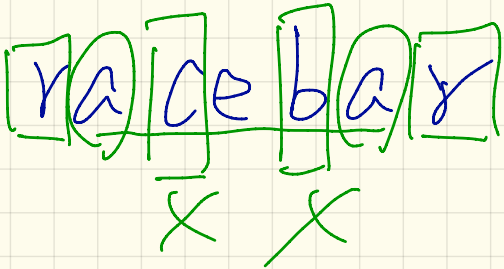
"Yo" + "rk"



Base Cases:

"" is P

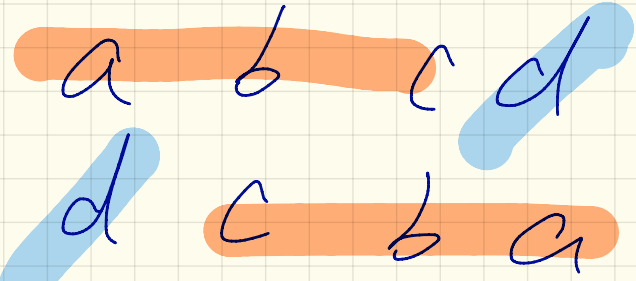
"_" is P



Recursive Cases:

reverse

input: a b c d



reverse(b c d) + a

output: d c b a

reverse ("") ""

reverse ("_") "_"

Lecture 17

Tuesday Nov. 7

Problem	Sub-Problems	Solution
fac(n)	fac(n-1)	n * fac(n-1)
fib(n)	fib(n-1) fib(n-2)	fib(n-1) + fib(n-2)
isP(c1,s,c2)	isP(s)	c1 == c2 && isP(s)
reverse(c1,s)	reverse(s)	reverse(s) + c1
occOf(c1,s,c2)	occOf(s, c2)	occOf("aaab", 'a') = 1 + occOf("aab", 'a', 2) occOf("baab", 'a') = 0 + occOf("aab", 'a', 2)

if c1 == c2
1 + occOf(s, 'a')
else
0 +

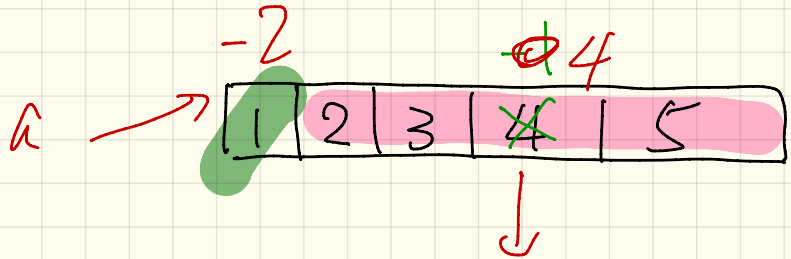
$\forall x : \text{int} \mid x \in \{1, 2, 3\} \cdot x > 2$ F

$\exists x : \text{int} \mid x \in \{1, 2, 3\} \cdot x > 2$ T

(1) $\forall x \mid x \in \emptyset \cdot P(x)$ T ⁓ no counter-ex. e s.t. $e \in \emptyset$ $\hat{\neg P}(e)$

(2) $\exists x \mid x \in \emptyset \cdot P(x)$ F ⁓ no witness e s.t. $e \in \emptyset$ $\hat{P}(e)$

isAllPos(int[] a)



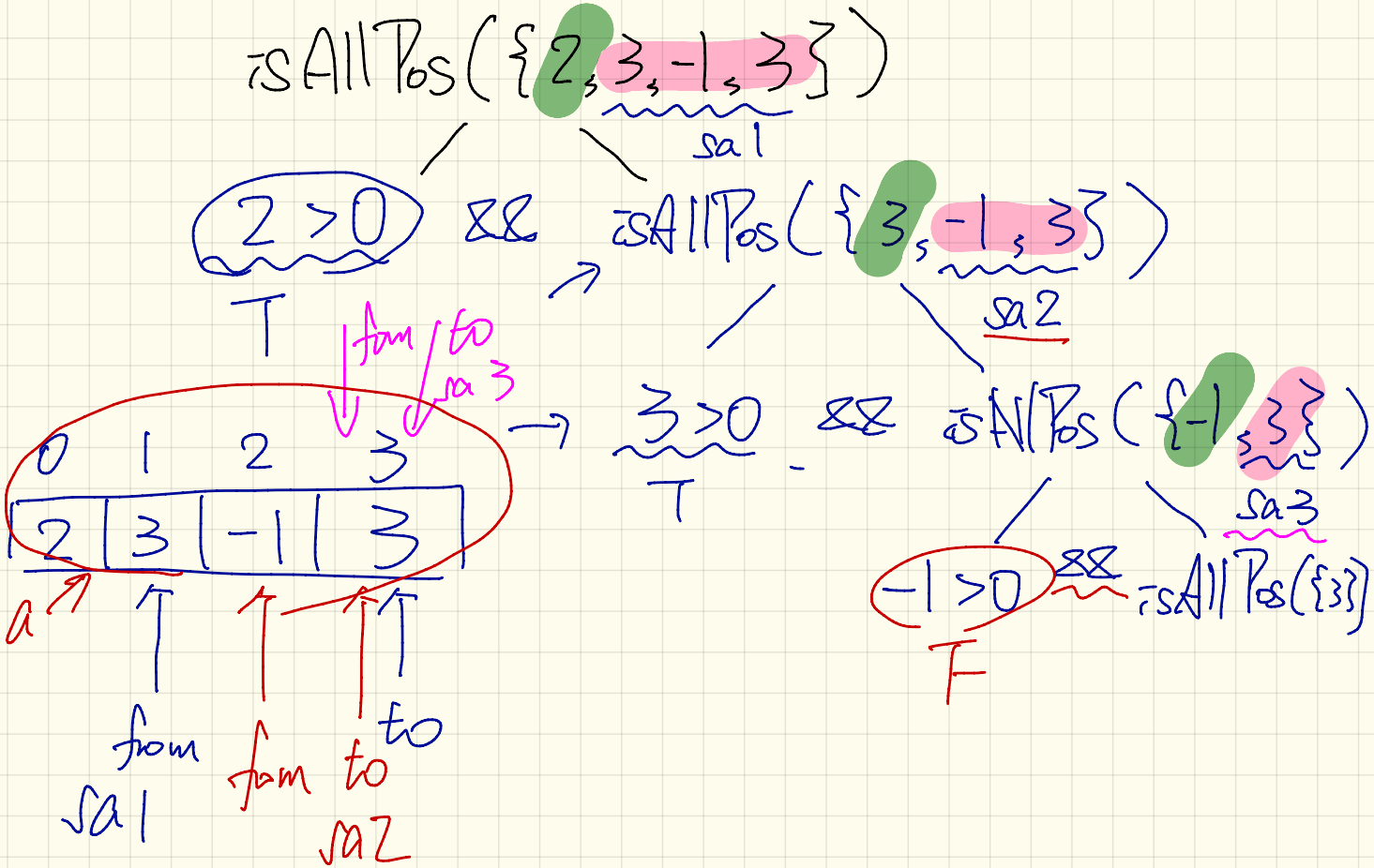
isAllPositive({2, 3, 4, 5})

F

a[0]>0

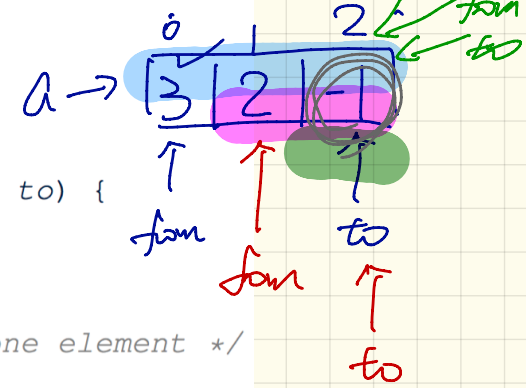
&&

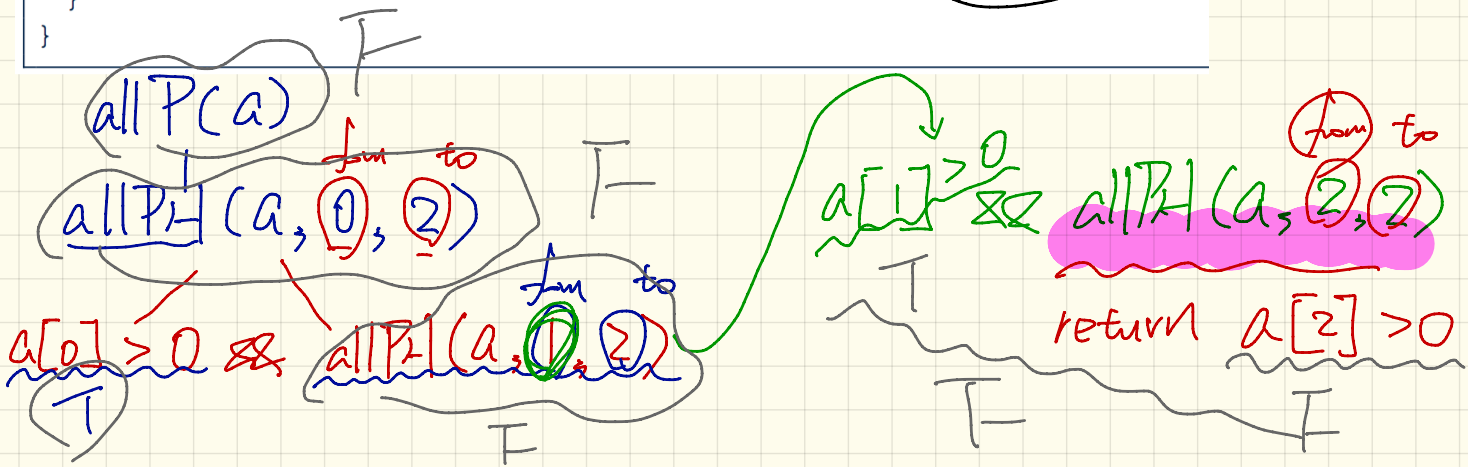
isAllPos({2, 3, 4, 5})

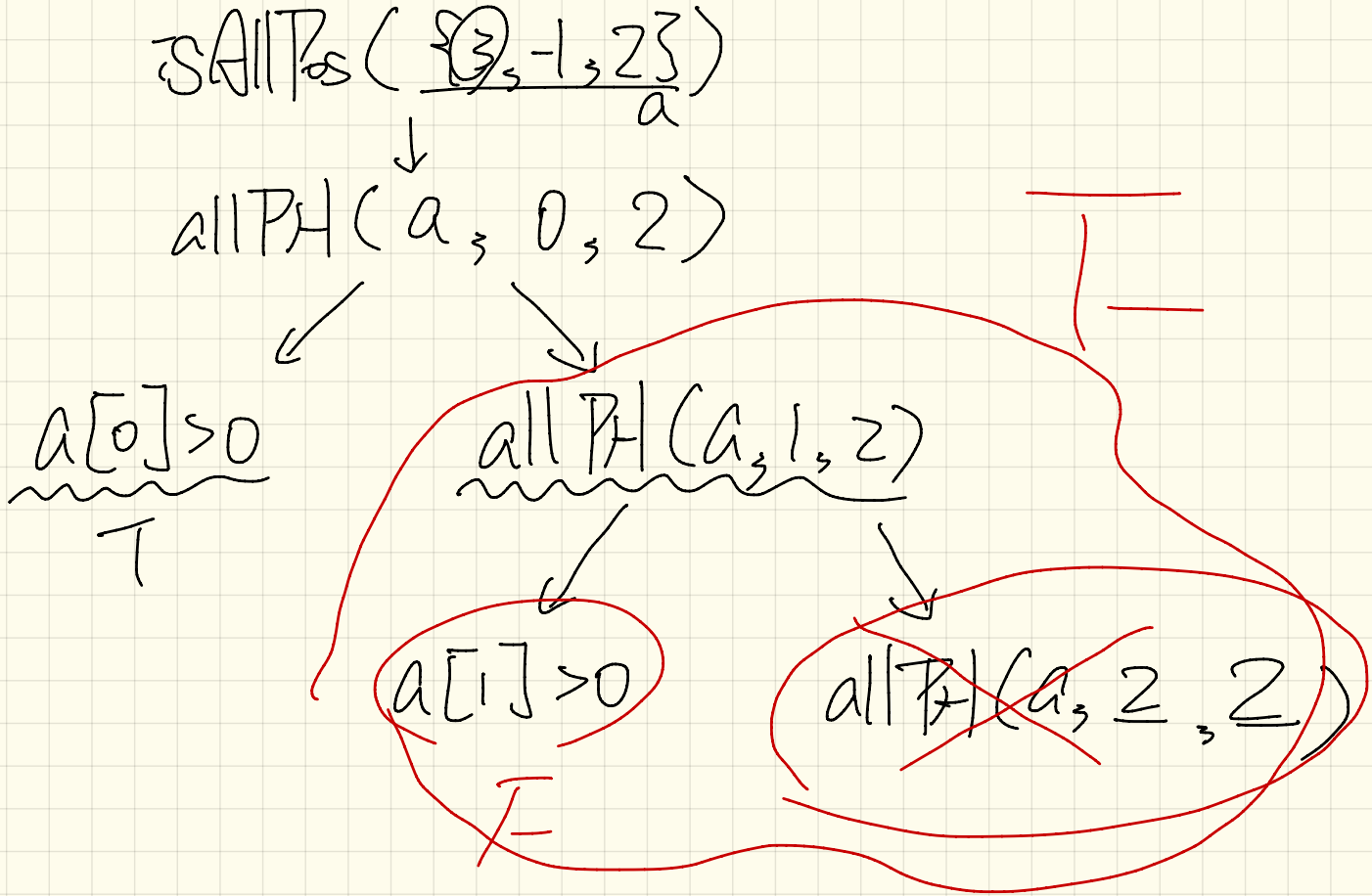

isAllPos({2,3,-1,3})
sa1
2 > 0 && isAllPos({3,-1,3})
T
sa2
from to
sa3
0 1 2 3
2 3 -1 3
3 > 0 && isNIPos({-1,3})
T
sa3
-1 > 0 && isAllPos({3})
F
a
from
sa1
from to
sa2
to

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

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

empty array.






allPos({3,-1,2}, a)
allPH(a, 0, 2)
a[0]>0
T
allPH(a, 1, 2)
a[1]>0
allPH(a, 2, 2)

int[] @ = new int[0];

allPositive(@)

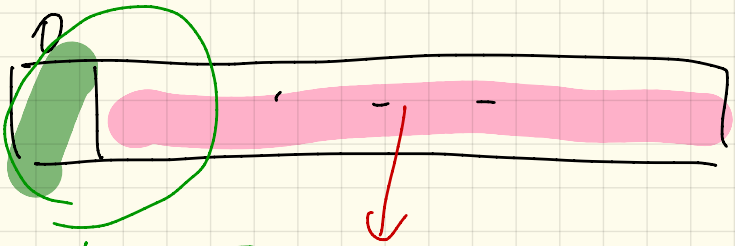
allPH(a, 0, -1)

from > to

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

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

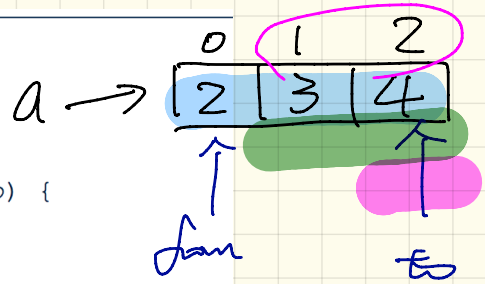
boolean isSorted(int[] a)

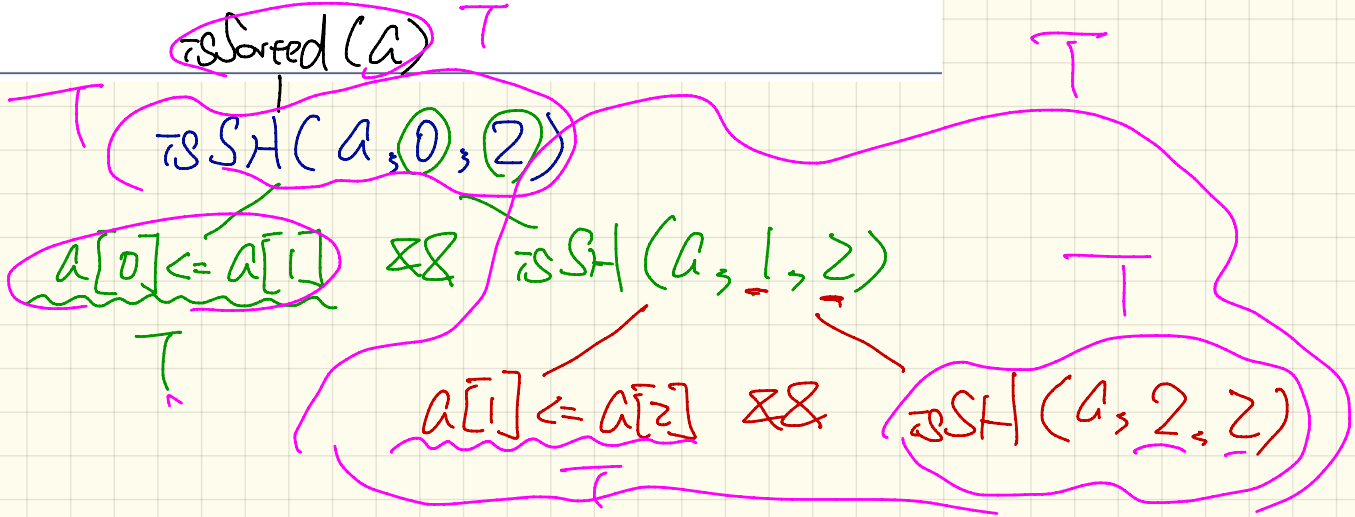
{} → T



a[0] ≤ a[i] && isSorted( )

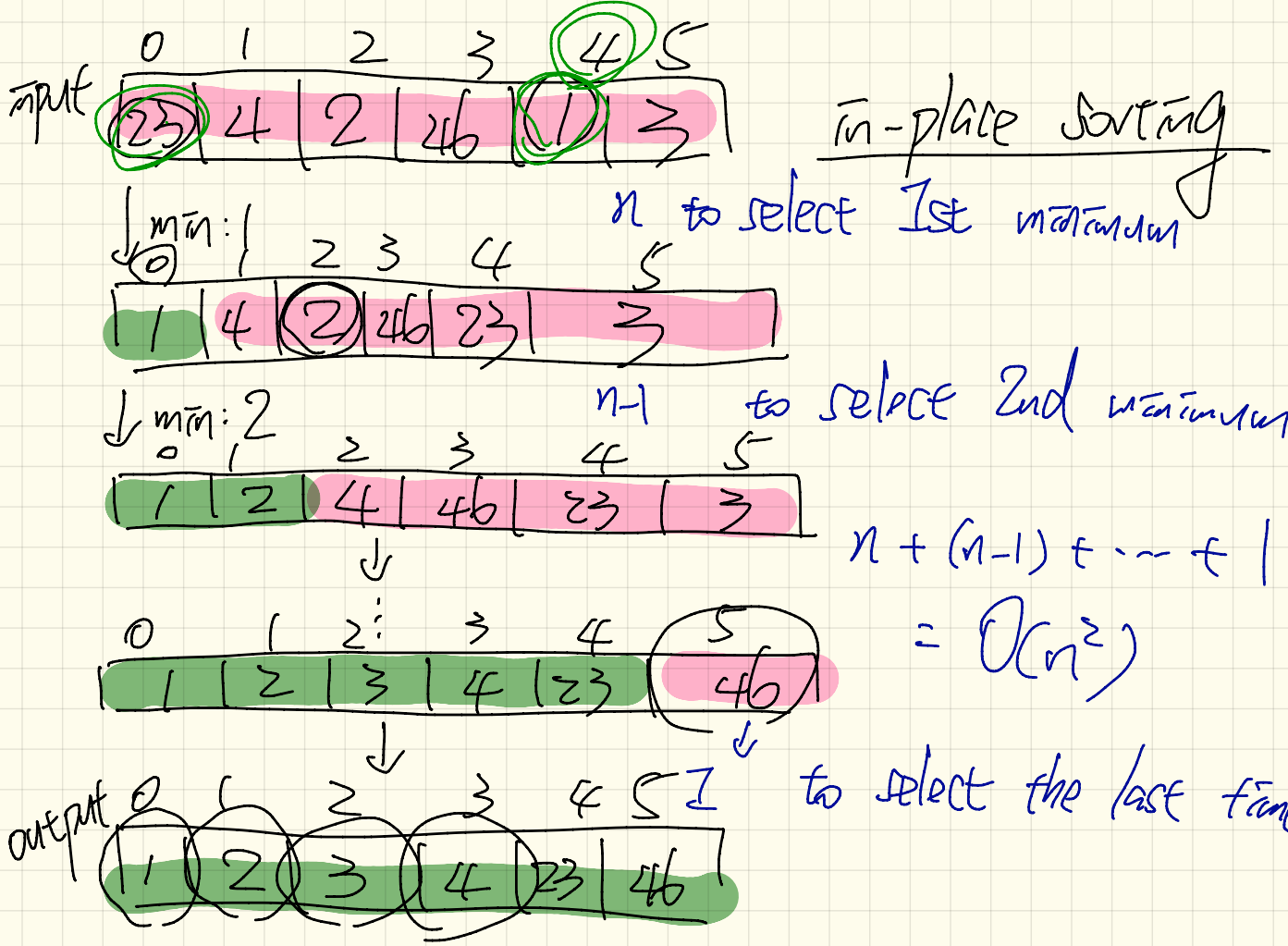
```
boolean isSorted(int[] a) {
  return isSortedHelper(a, 0, a.length - 1);
}

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





Lecture 18

Thursday Nov. 9


0 1 2 3 4 5
input
23 4 2 46 1 3
in-place sorting
n to select 1st minimum
min: 1
0 1 2 3 4 5
1 4 2 46 23 3
min: 2
n-1 to select 2nd minimum
0 1 2 3 4 5
1 2 4 46 23 3
n + (n-1) + ... + 1
= O(n²)
0 1 2 3 4 5
1 2 3 4 23 46
1 to select the last times
output
0 1 2 3 4 5
1 2 3 4 23 46

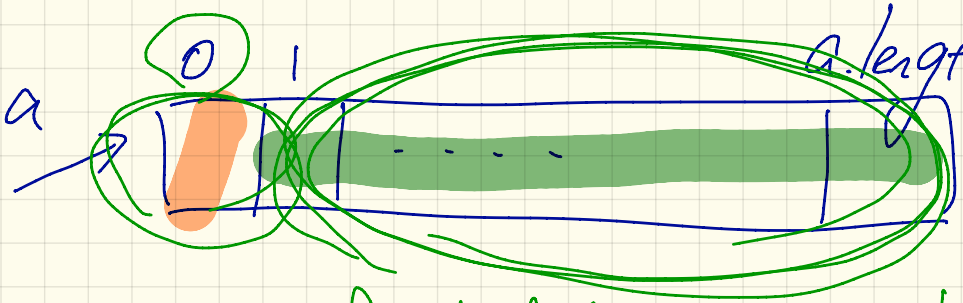
Problem: Recursively find min from an array

(assume a.length >= 1)

```
int findMin(int[] a)
```

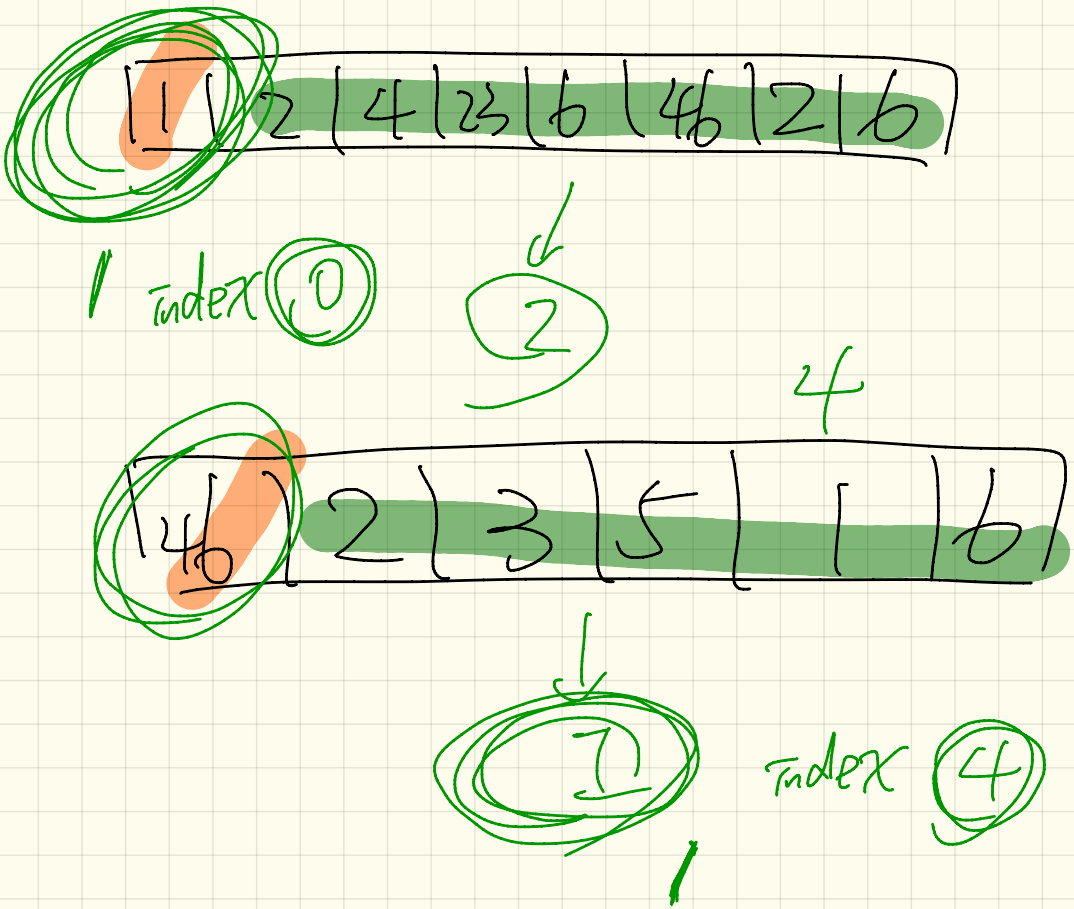
Base Case

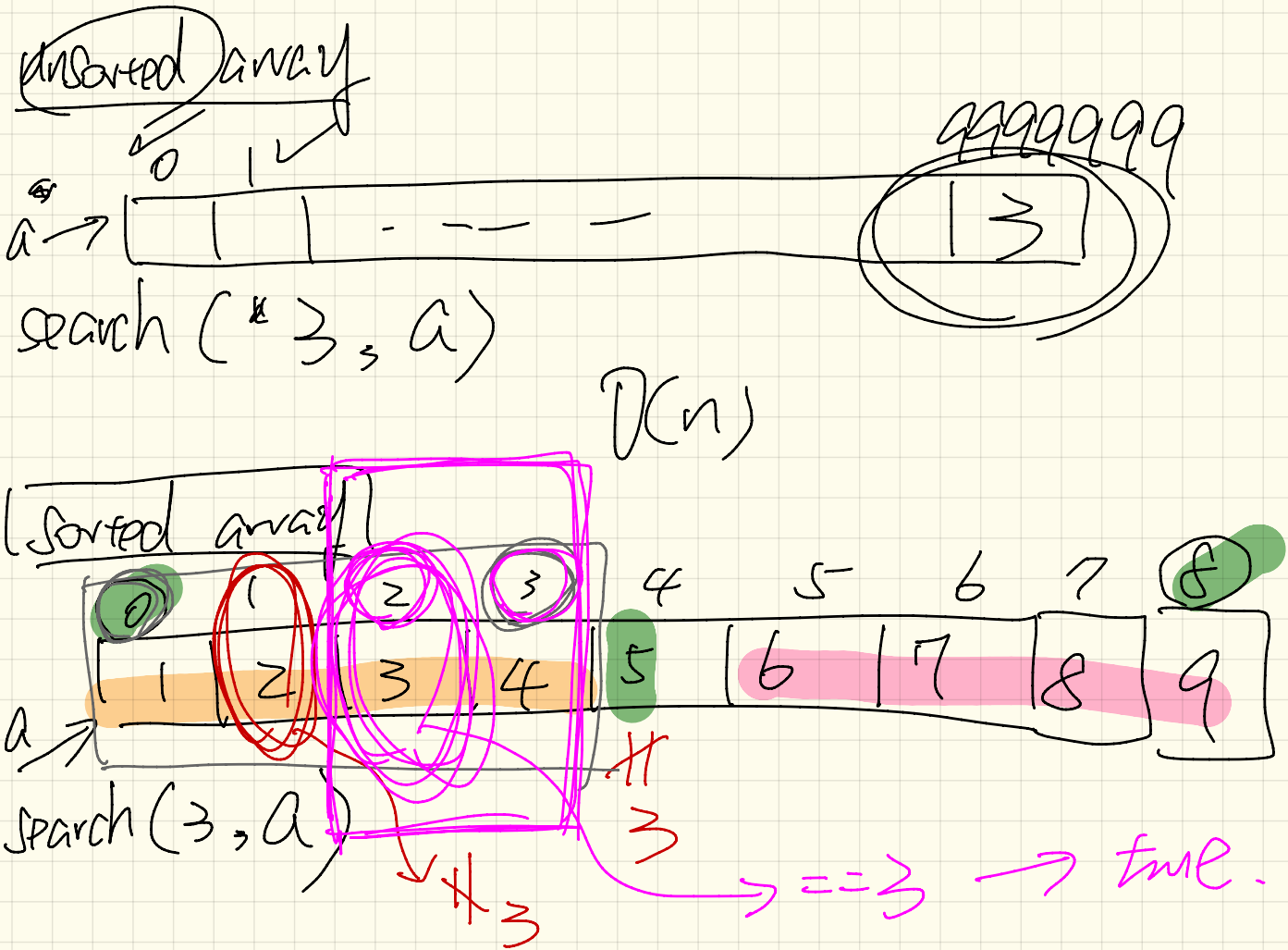
a.length == 1 → return a[0]

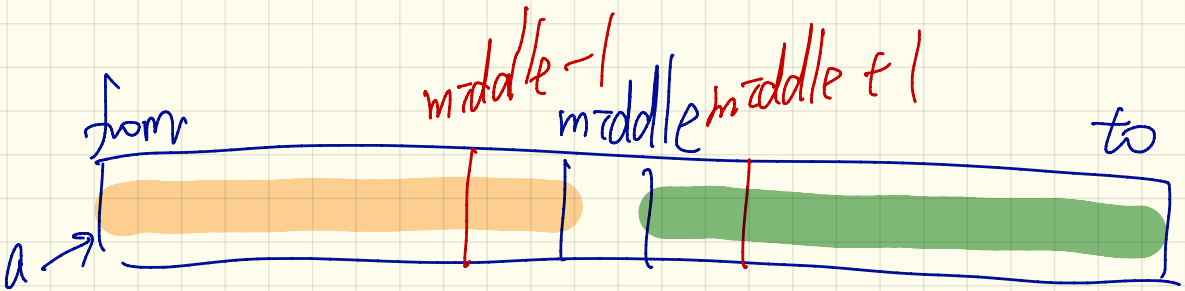


findMin(a, 1, a.length - 1)

↳ min of

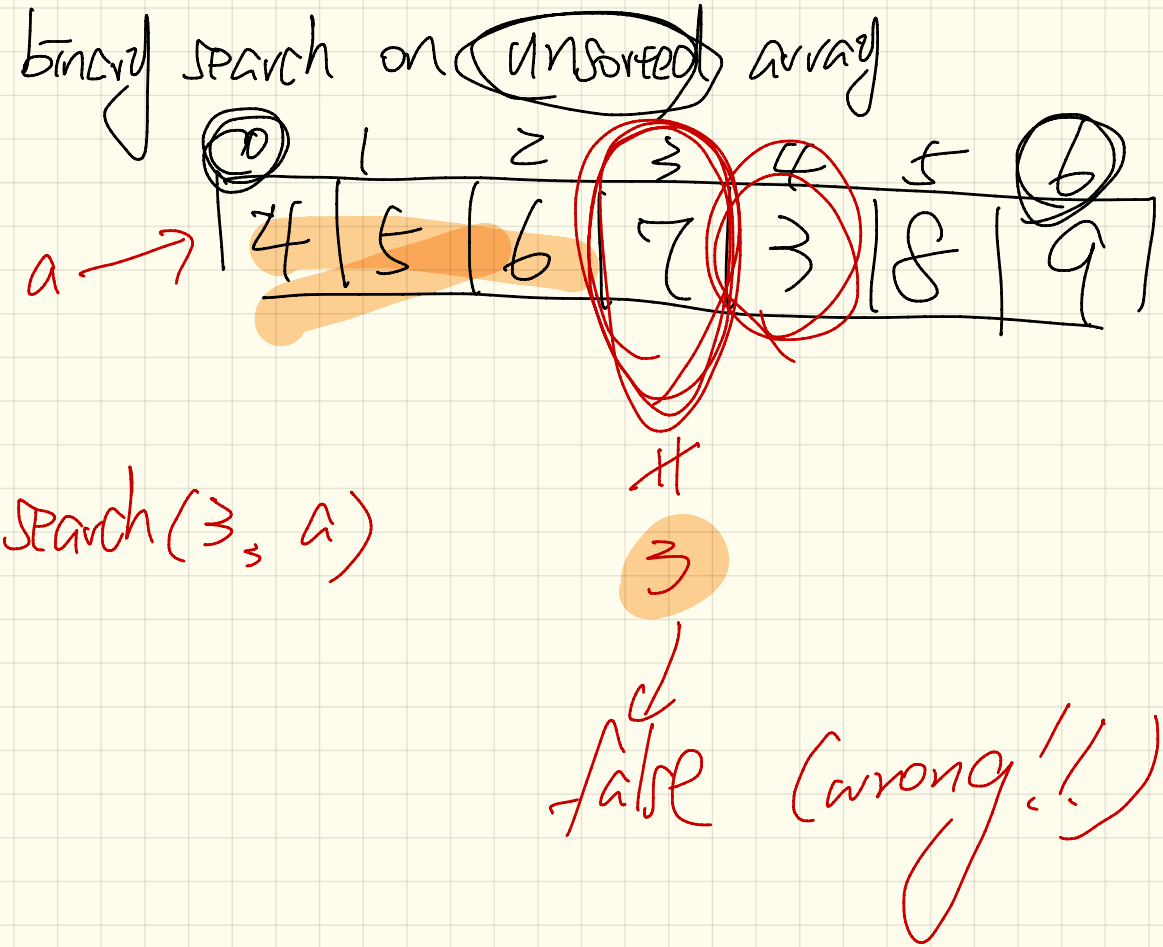

1 2 4 23 6 46 2 6
index 0
2
4
46 2 3 5 1 6
1
index 4


unsorted array
0
1
a
3
search(3, a)
O(n)
sorted array
0
1
2
3
4
5
6
7
8
9
a
search(3, a)
3
3
==3
true.

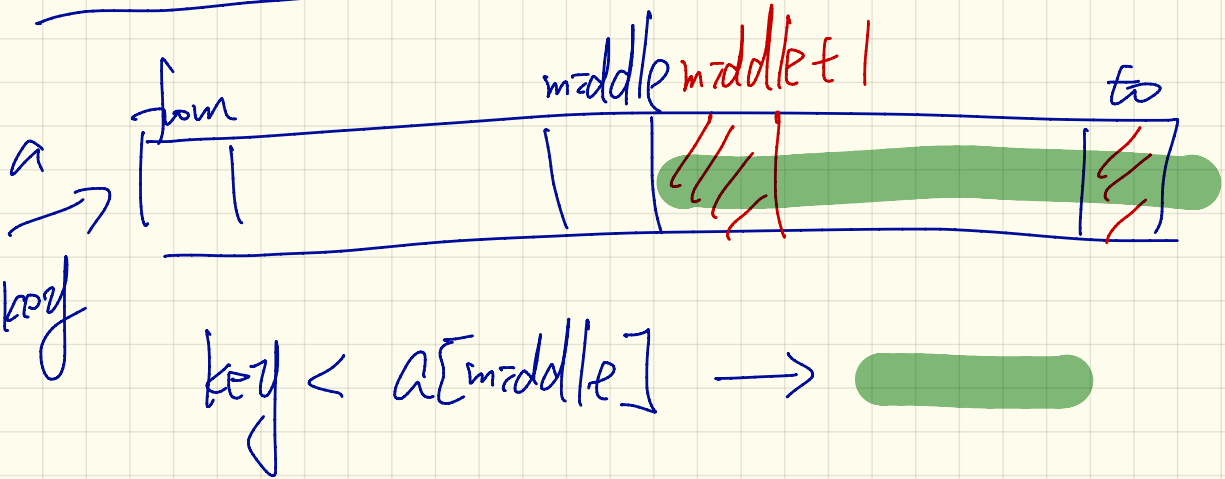


key

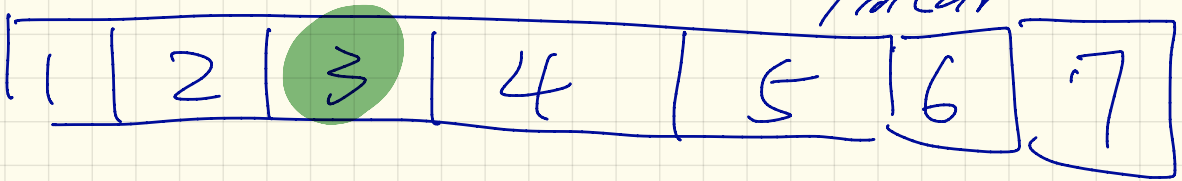
key < a[middle]


binary search on unsorted array
0
1
2
3
4
5
6
4
5
6
7
3
8
9
a
search(3, a)
3
false (wrong!!)

How to do binary search if the array is sorted in descending order?

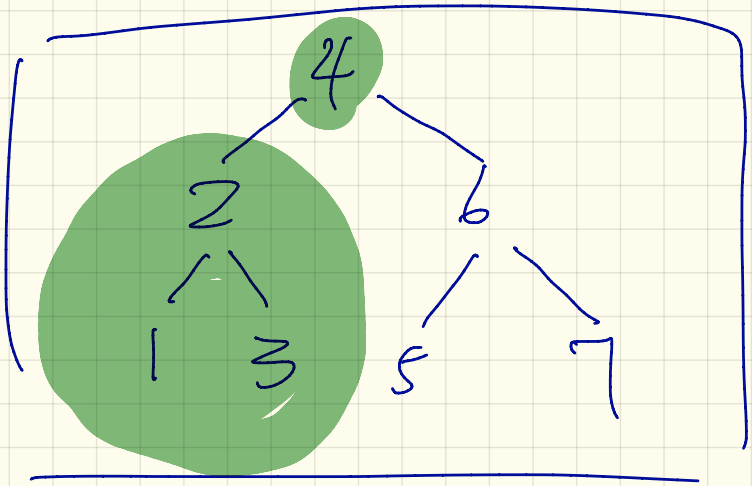


linear



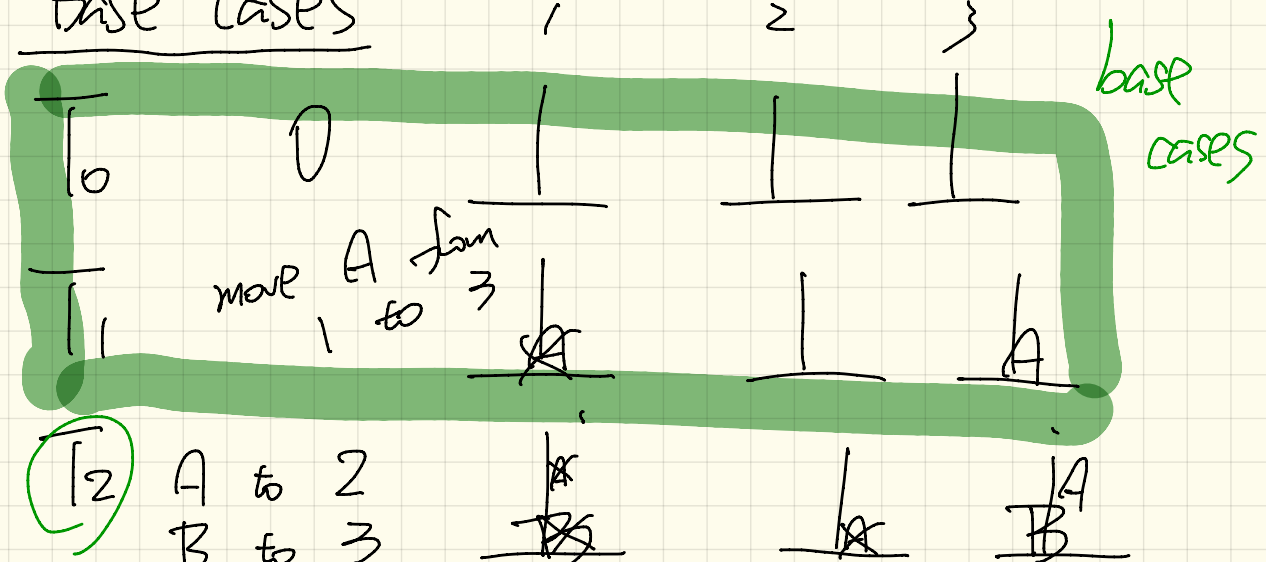
binary-search-tree

non-linear

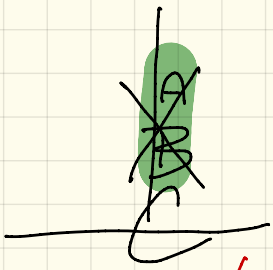


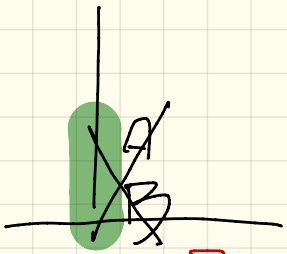
Tower of Hanoi (move from peg 1 to peg 3)

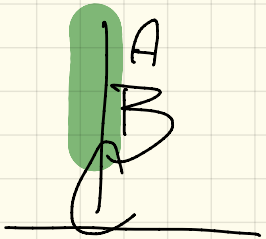
Base Cases



T_3







move A B C from 1 to 3

move $\overset{A}{B}$ from 1 to 2? 3?

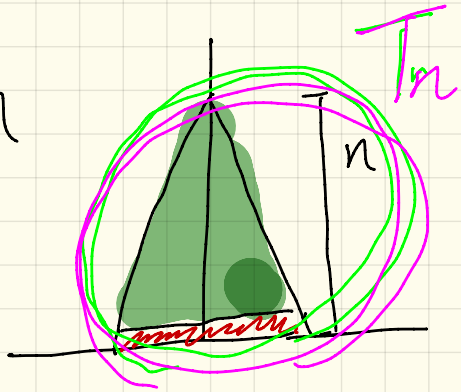
3 Steps:

Move $\overset{A}{B}$ from 1 to 2

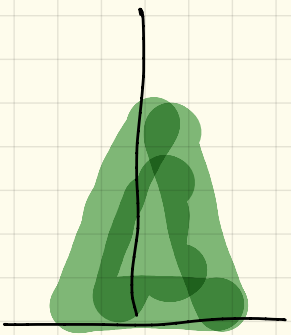
Move C from 1 to 3

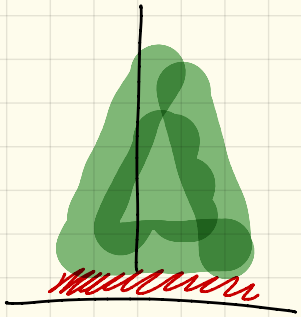
Move $\overset{A}{B}$ from 2 to 3

T_n



T_n





$T_0 = 0$

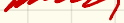
$T_1 = 1$

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

$2 \cdot T(n-1) + 1$

3 Steps

✓ - Move  from 1 to 2 T_{n-1}

- Move  from 1 to 3 1

- Move  from 2 to 3 T_{n-1}

- $T(0) = 0$
- $T(n) = 2 \cdot T(n-1) + 1$

$$T(4) = 2 \cdot T(3) + 1$$

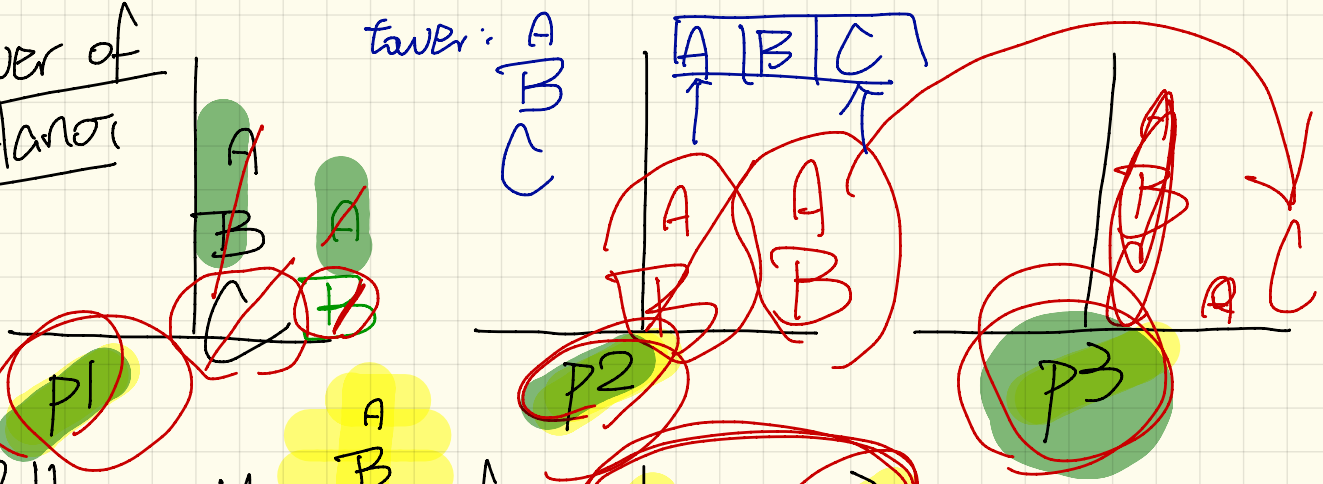
$$= 2 \cdot (2 \cdot T(2) + 1) + 1$$

$$= 2 \cdot (2 \cdot (2 \cdot T(1) + 1) + 1) + 1$$

Lecture 19

Tuesday Nov. 14

Tower of Hanoi



Problem: Move A B C from p1 to p3

Move B from p1 to p2

Move C from p1 to p3

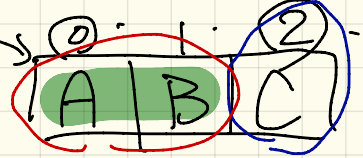
Move B from p2 to p3

Move A from p1 to p3

Move B from p1 to p2

Move A from p3 to p2

Problem: Move A, B, C from p1 to p3



---

toH(ds, 0, 2, p1, p3) =

{"A", B, C}

intermediate = 6 − 1 − 3

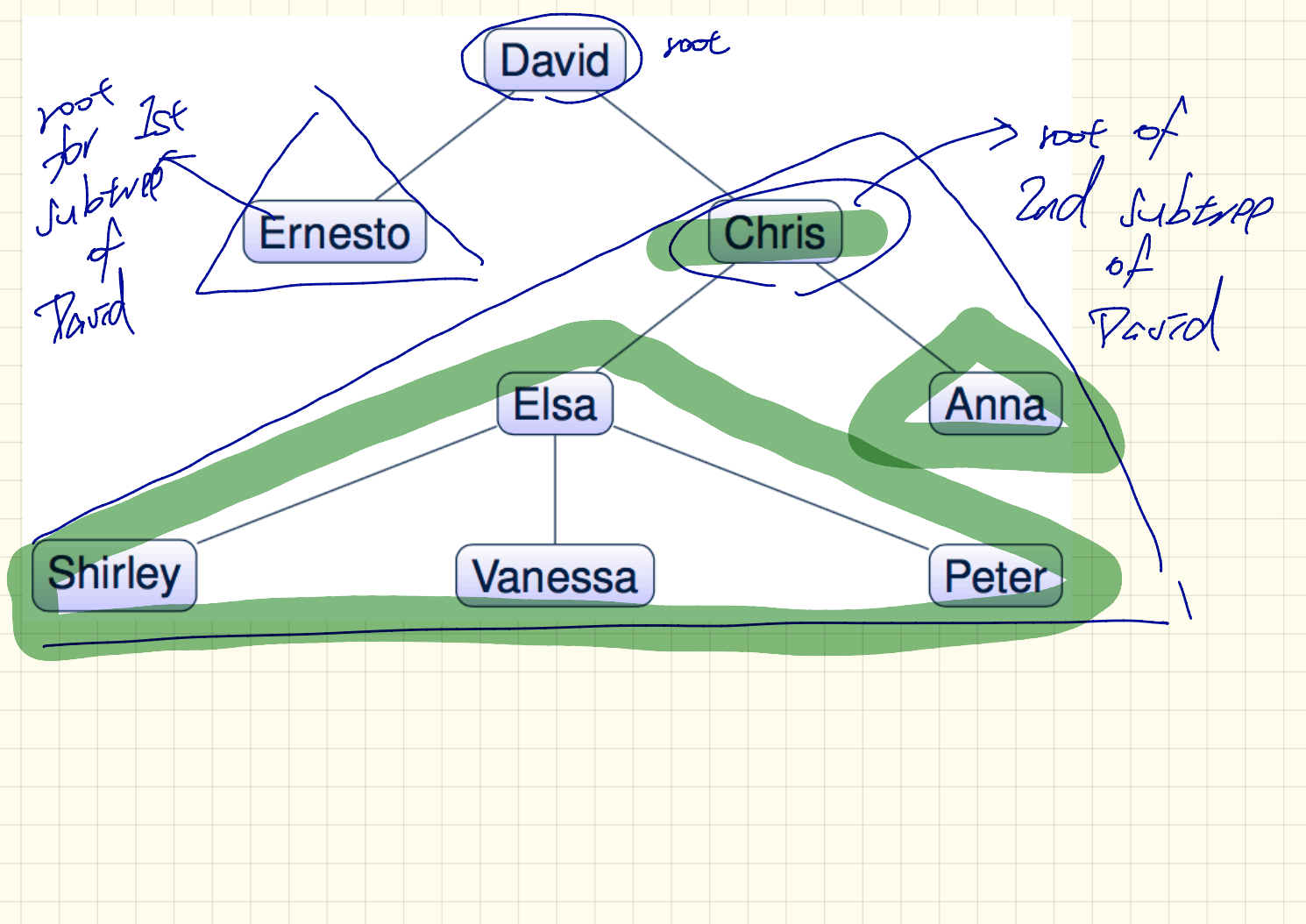
= 2

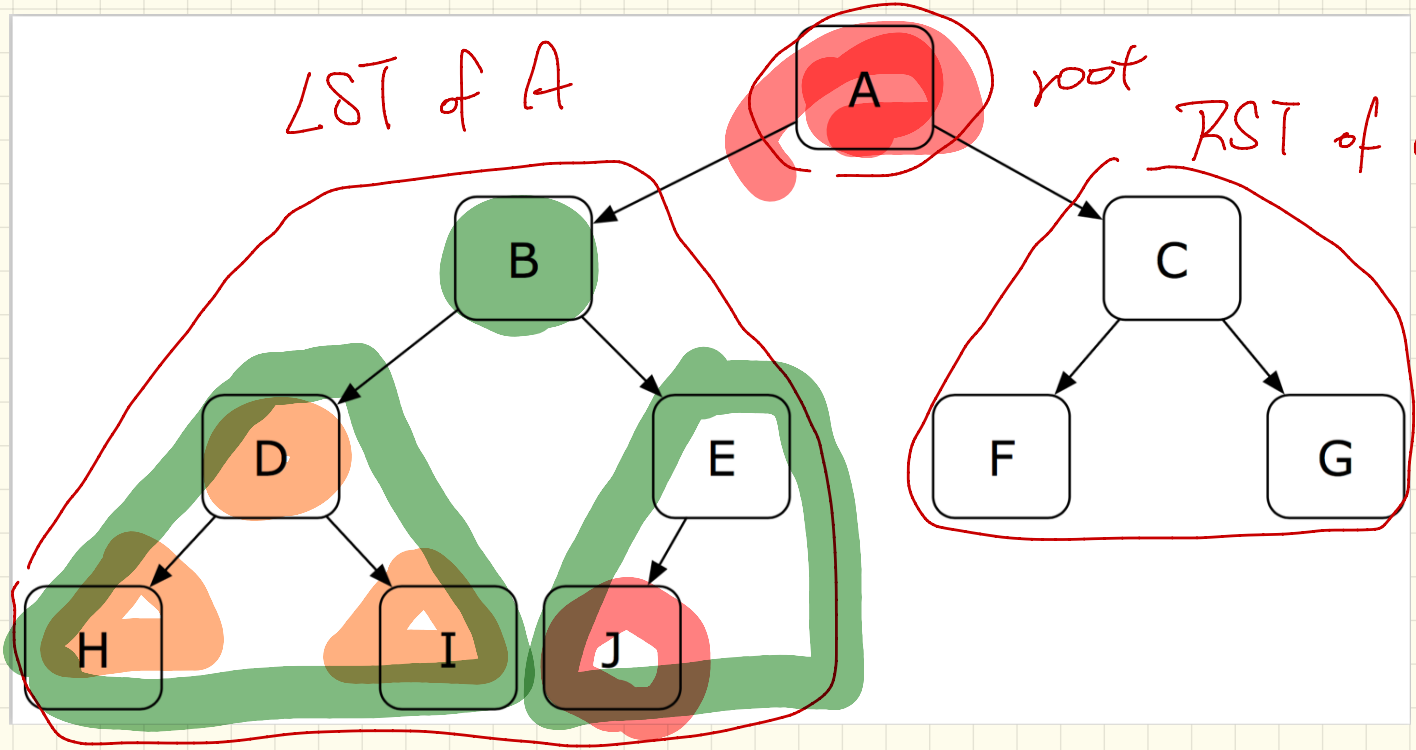
intermediate : p3

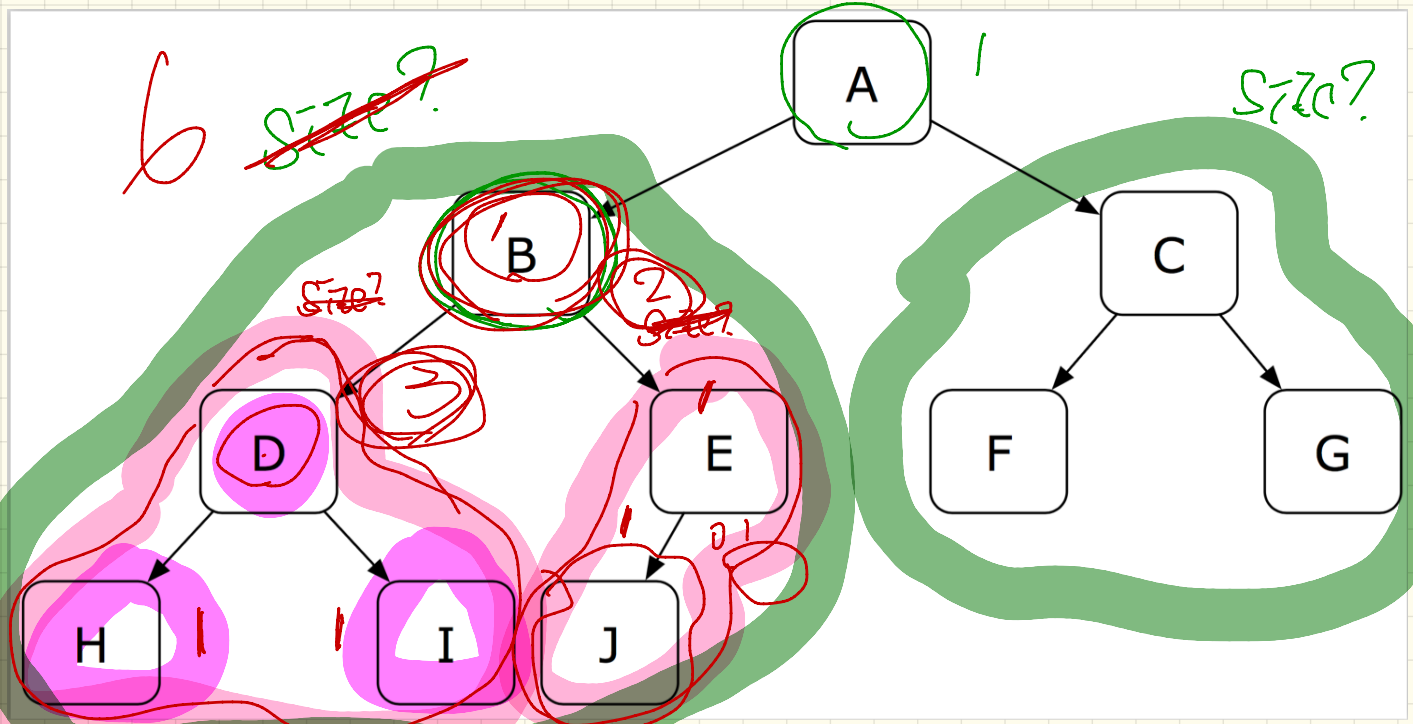
{ toH(ds, 0, 0, p1, p3)
{ move ds[0] from p1 to p2
{ toH(ds, 0, 0, p3, p2)

{ toH(ds, 0, 1, p1, intermediate: p2)
{ Move ds[1] from p1 to p3
{ toH(ds, 0, 1, intermediate, p3)

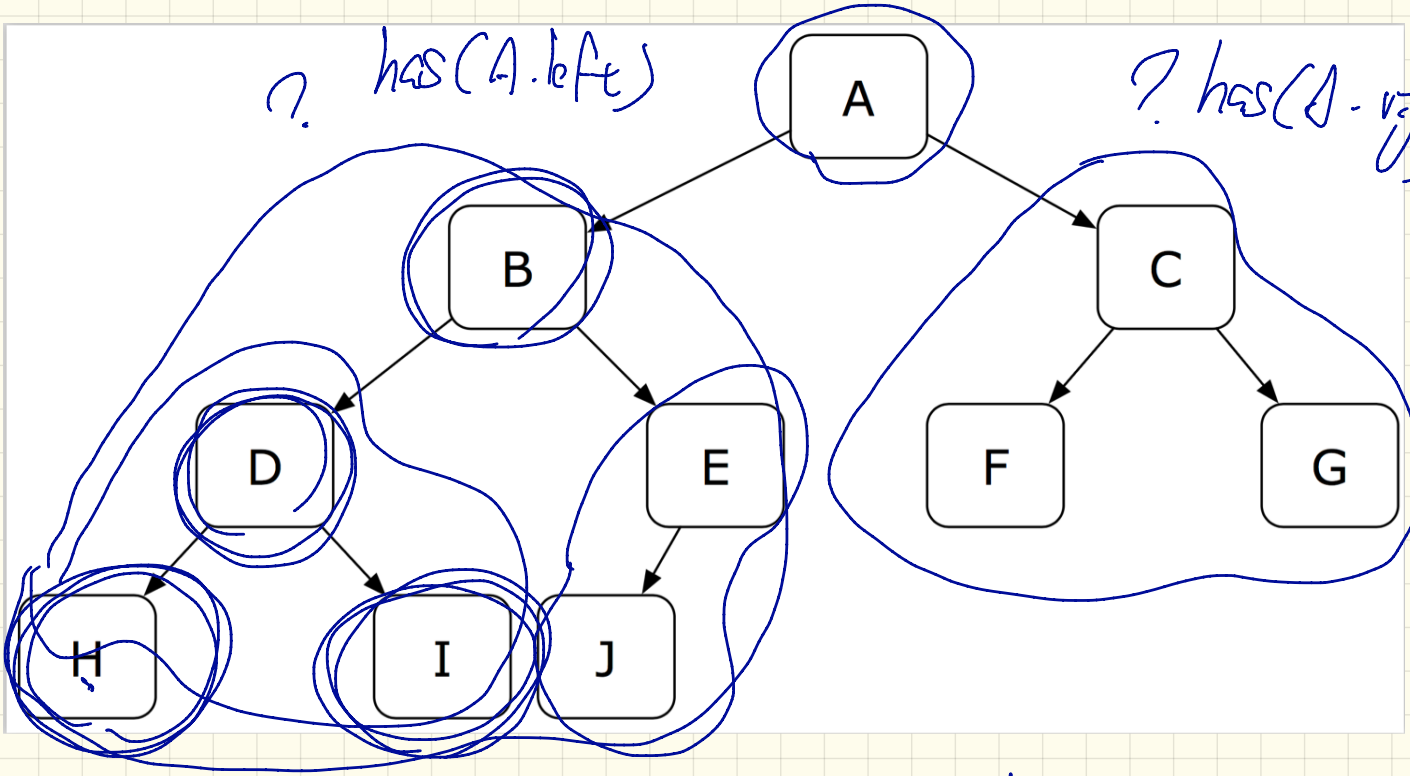
{A, B}


David
root
root for 1st subtree of David
Ernesto
Chris
root of 2nd subtree of David
Elsa
Anna
Shirley
Vanessa
Peter


LST of A
A
root
RST of A
B
C
D
E
F
G
H
I
J



Size of subtree rooted at A

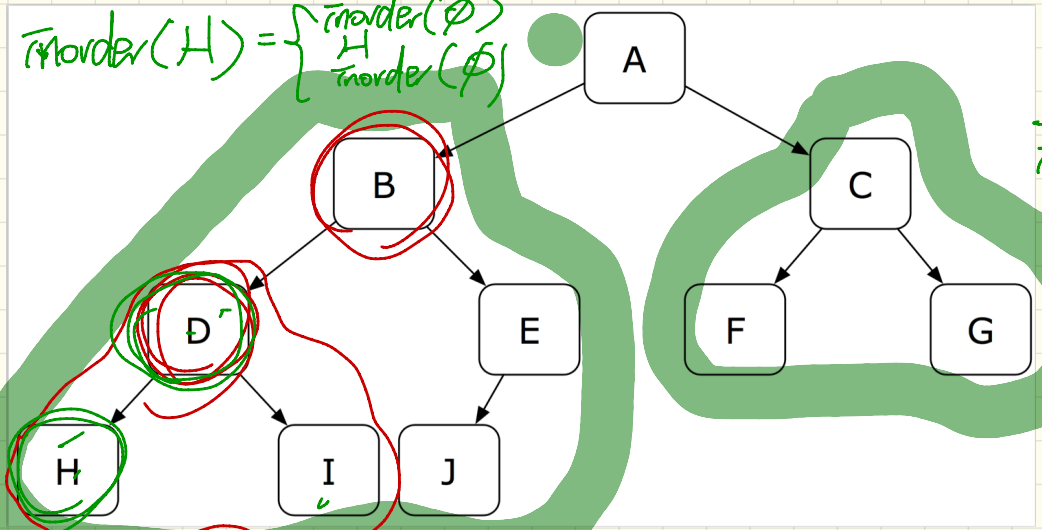


? has(A.left)

? has(A.right)

has(H)

has(A) = A == H ||
has(A.left) ||
has(A.right).



inorder(H) = { inorder(∅) / H / inorder(∅) }

H D I B J E

inorder(D) inorder(E)

inorder(A)

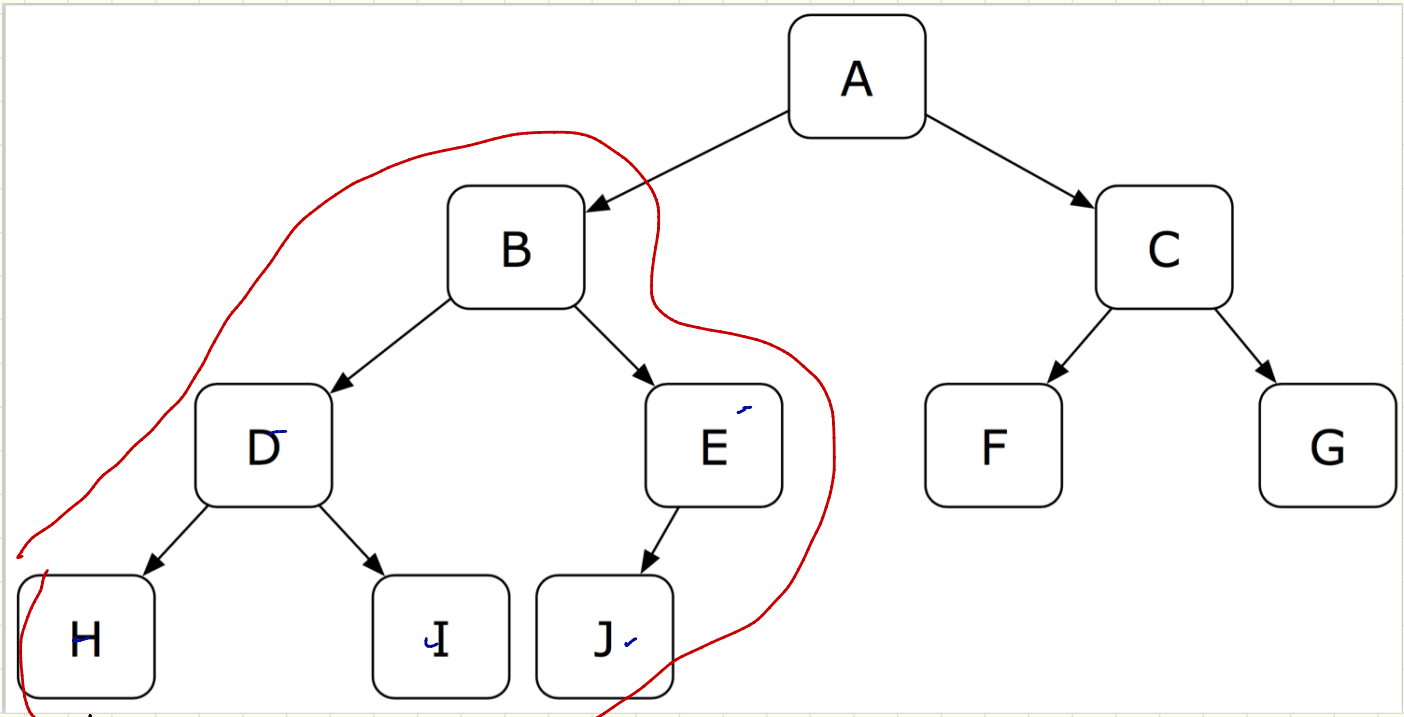
H

{ IO(H) / D / IO(I) }

inorder(A) = { inorder(A.left) / A / inorder(A.right) }

inorder(A.left) = inorder(B.left) / B / inorder(B.right)

inorder(B.left) = D



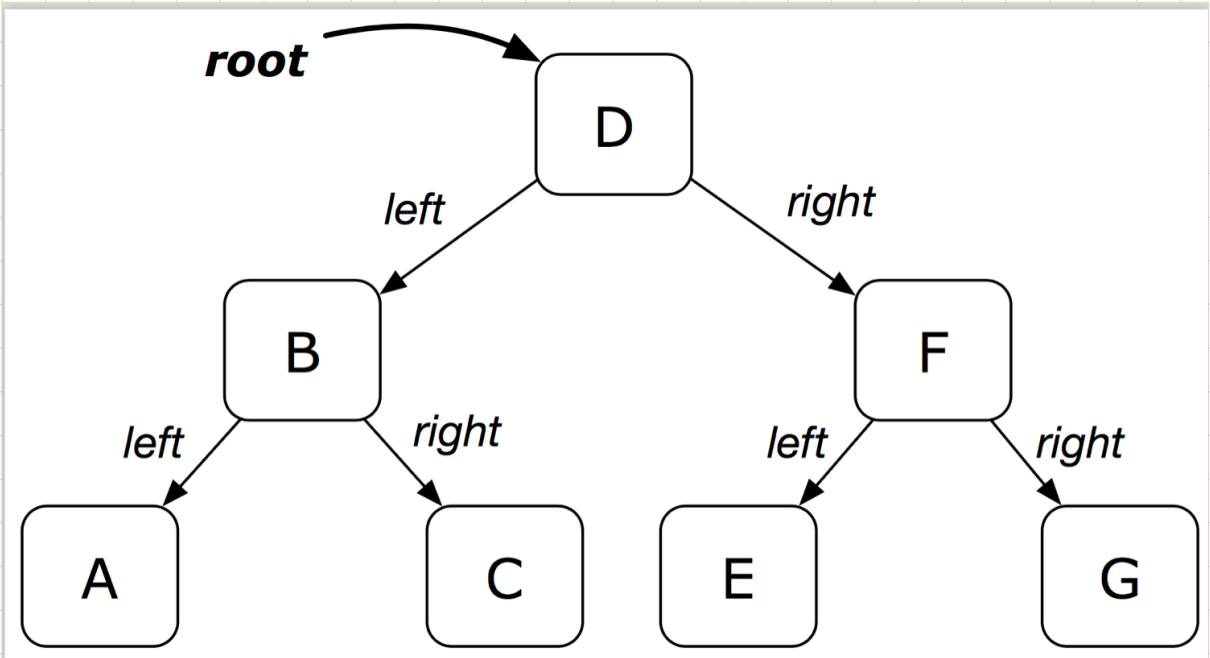
Inorder(A)

HDIBJEAFCG

TO(D) TO(E) TO(B) TO(C) TO(A)

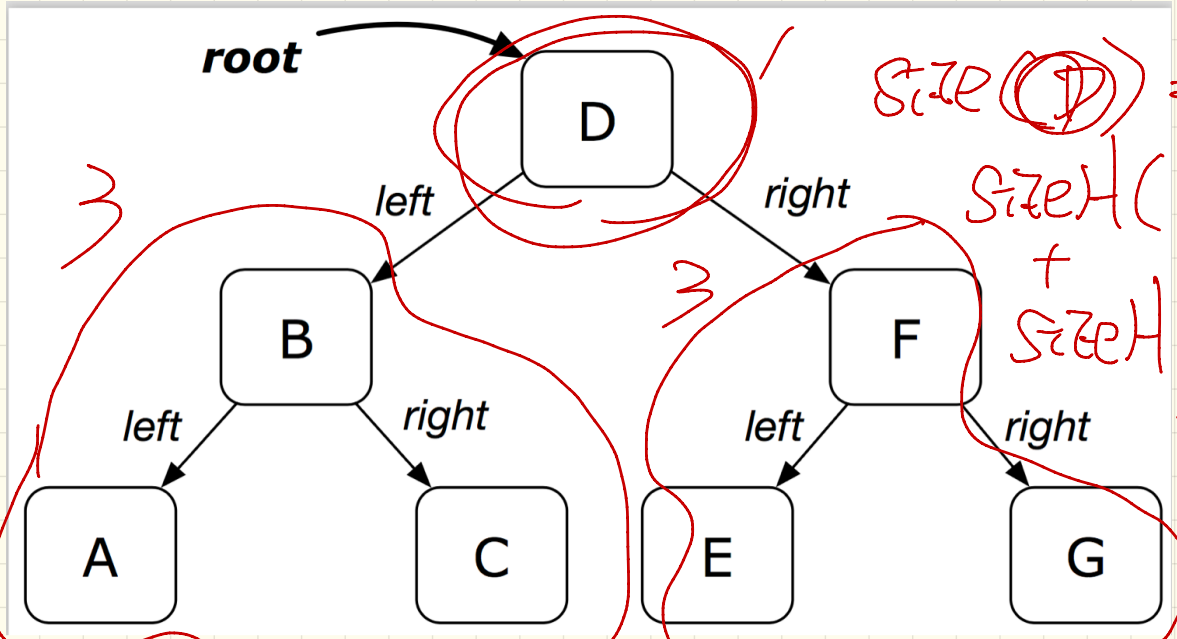
Lecture 20

Thursday Nov. 16



root
D
B
left

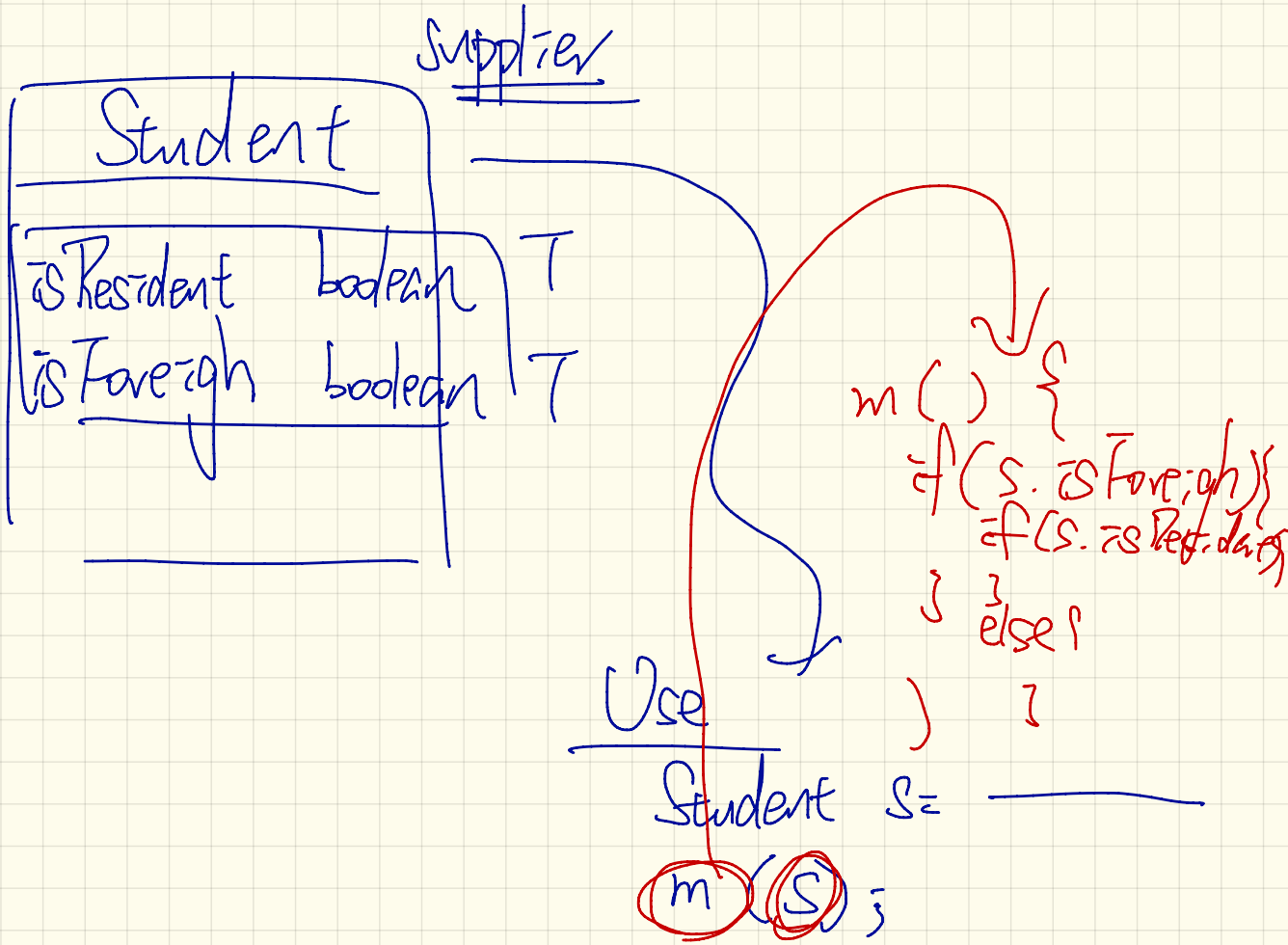
bt.addToLeft(root, "B")
↳ root.setLeft(new BTNode("B"))



size(D) =
sizeH(D.left)
+
sizeH(D.right)
+
1

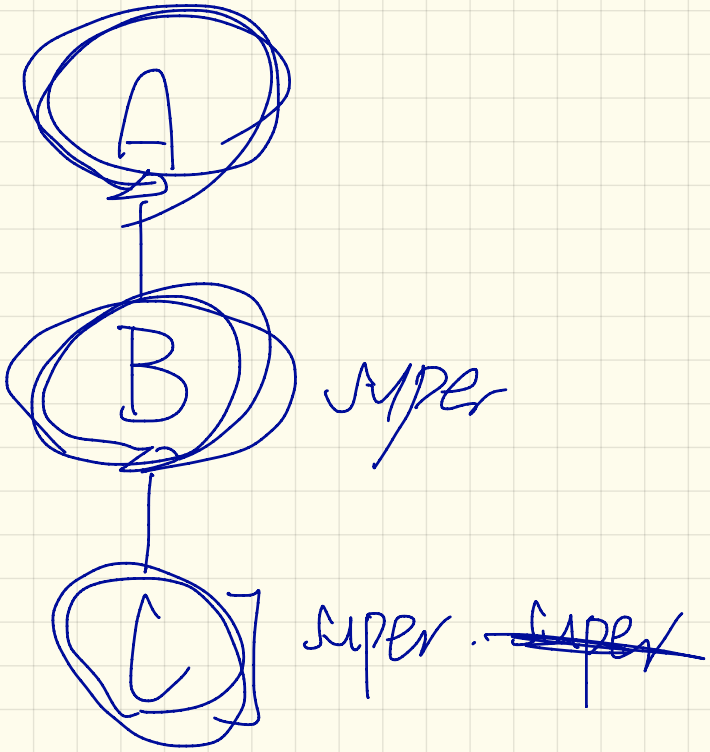
bt.inorder() → ArrayList

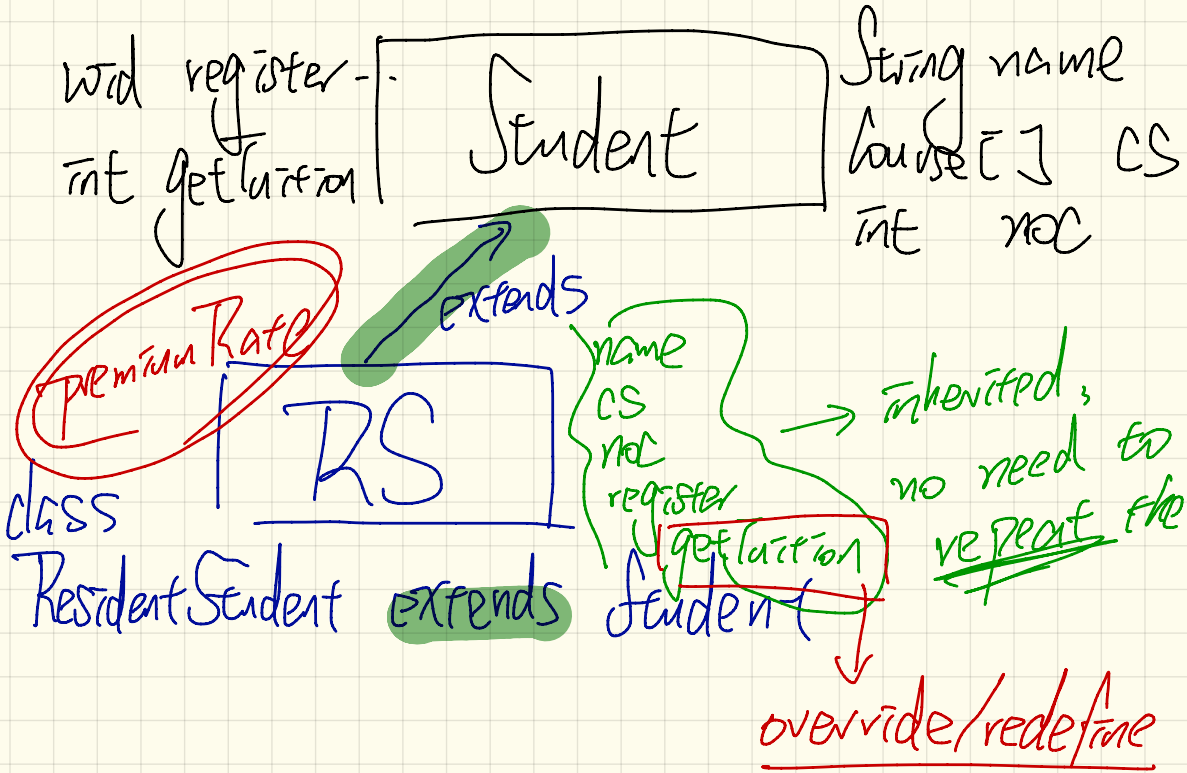
| A | B | C | D | E | F | G |

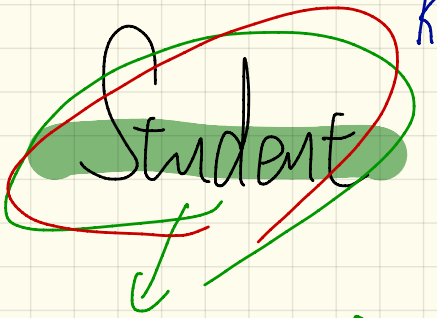

Supplier
Student
isResident boolean T
isForeign boolean T
m() {
if(s.isForeign){
if(s.isRefdang
} } else {
}
Use
Student s=
m(s);

Single Choice Principle

↳ when there's a change, there should be only one place (class, method) to apply the change.


A
B
super
C
super . super


void register()
int getTuition()
Student
String name
Course[] cs
int noC
premiumRate
extends
RS
class
ResidentStudent extends Student{
name
cs
noC
register()
getTuition()
inherited,
no need to
repeat them.
override/redefine



ResidentStudent vs;

S =

expectations of S

- S.name
- S.noC
- S.vcs
- S.register(..)
- S.getTuition(..)

inherits all expectations from Student

static type

defines the list of EXPECTATIONS on 'S'

not expectations of S.

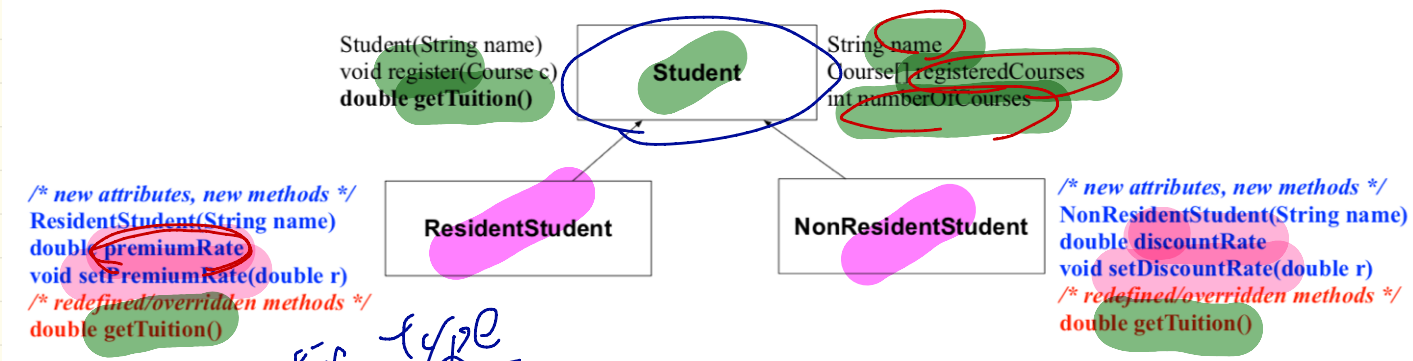
- S.pr
- S.dr

vs.pr

additional expect.

Lecture 21

Tuesday Nov. 21

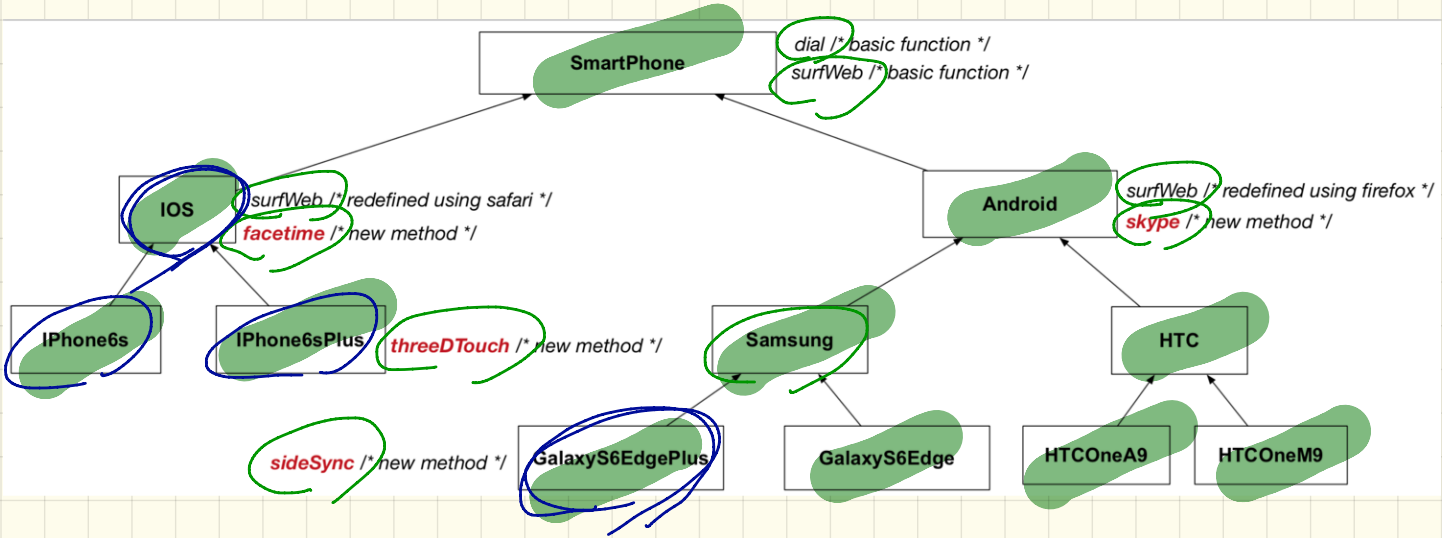


static type

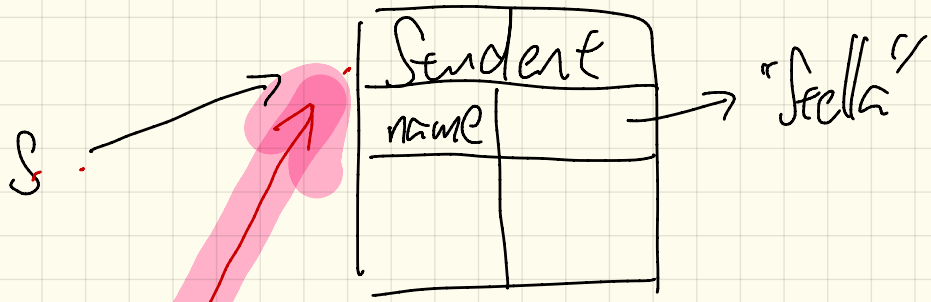
```
Student s = new Student("Stella");
ResidentStudent rs = new ResidentStudent("Rachael");
NonResidentStudent nrs = new NonResidentStudent("Nancy");
```

	name	rcs	noc	reg	getT	pr	setPR	dr	setDR
s.			✓				×		
rs.			✓				✓		×
nrs.			✓				×		✓

expectations


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

IOS myPhone ;



S = rS

① S = rS

② rS = S

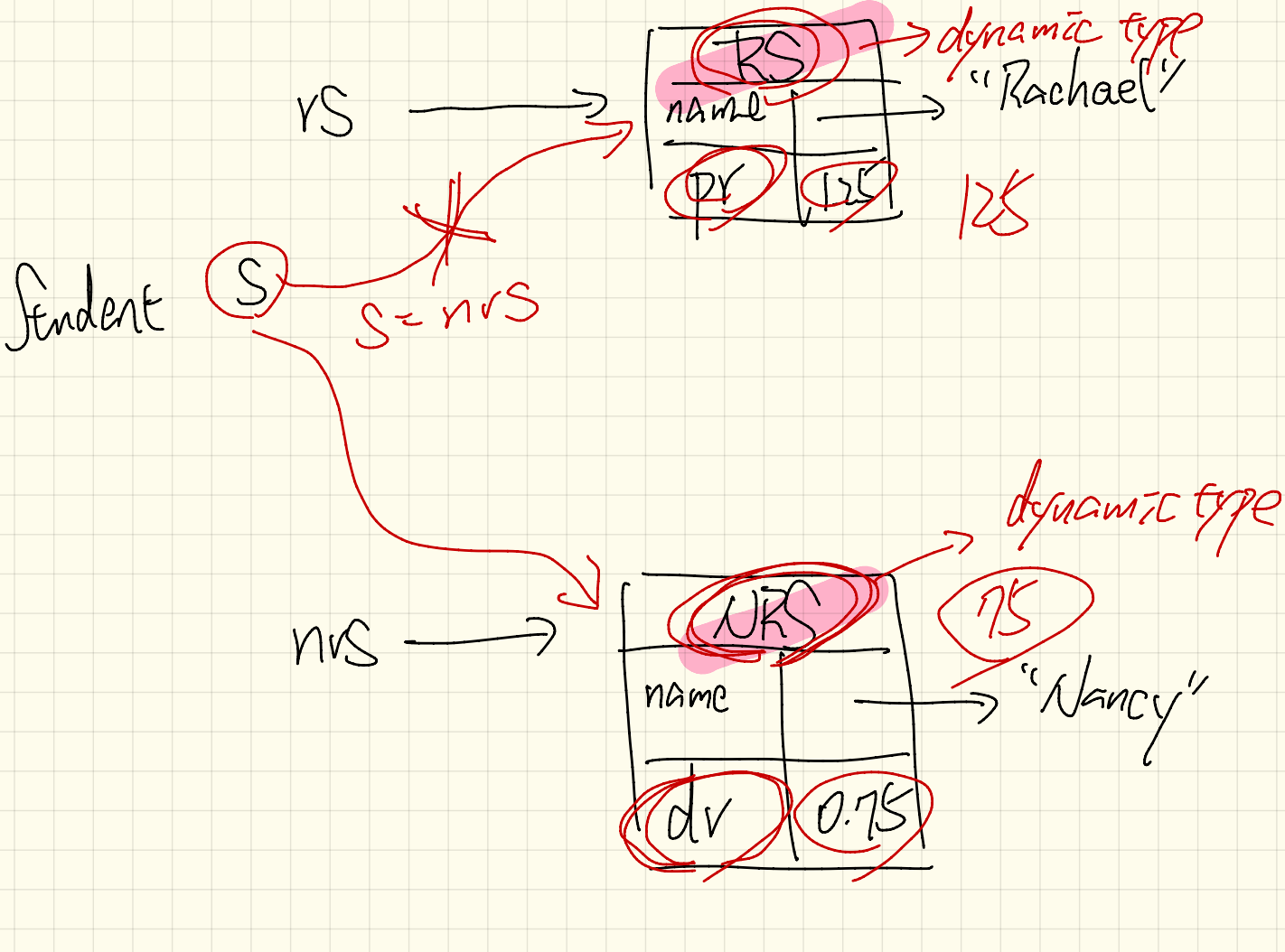
Prove by Contradiction

① Assume (rS = S) compiled

② Expectation for rS ? → Resid_Stu.

rS. premiumRate

→ crash : undefined on Stu. object.


rs
RS
name
dy
125
dynamic type
"Rachael"
125
student
s
s = nrs
nrs
NRS
name
dv
0.75
dynamic type
75
"Nancy"

Student S = new Student(...);

S → S

S.getTuition()

ResidentStudent rs = new RS(...);

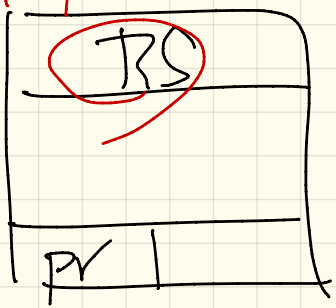
S = rs;

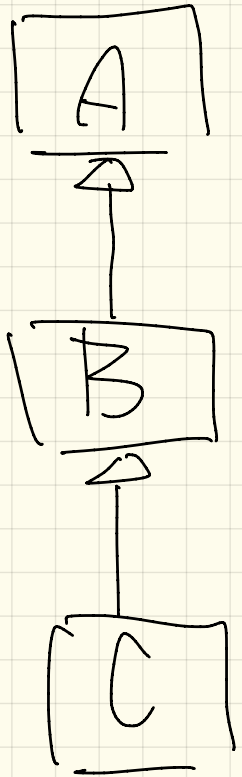
S.premiumRate ✗ ?

does not compile

Student S

ST of S (Student) does not declare premiumRate.

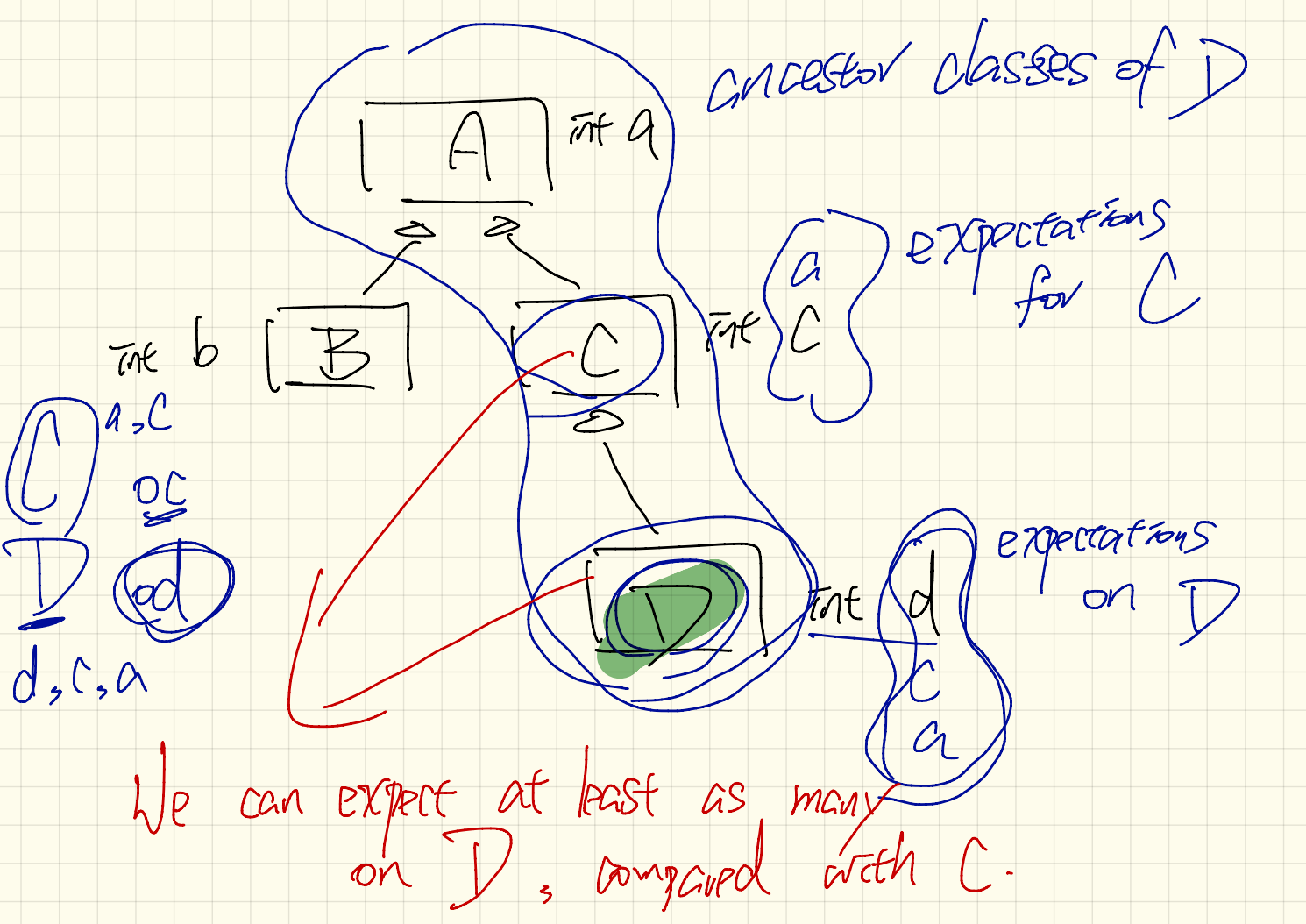




C implicitly extends A

$a > b \wedge b > C$

$a > C$

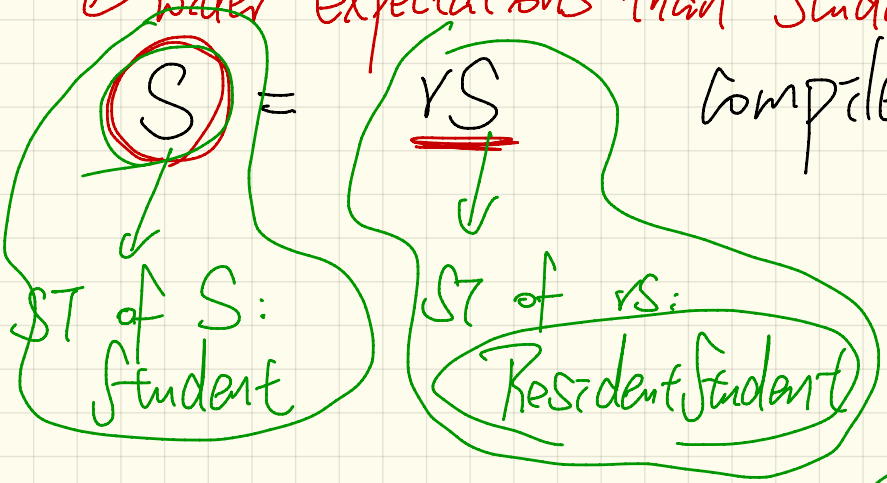

ancestor classes of D
A
int a
B
int b
C
int c
a
c
expectations for C
C
a, c
oc
D
od
d, c, a
D
int d
d
c
a
expectations on D
We can expect at least as many on D, compared with C.

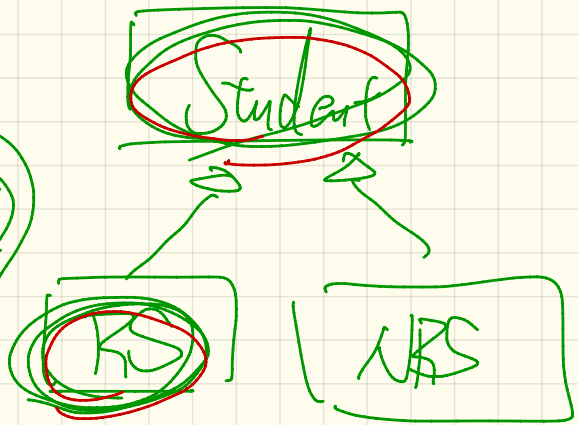
Student S = rS = S

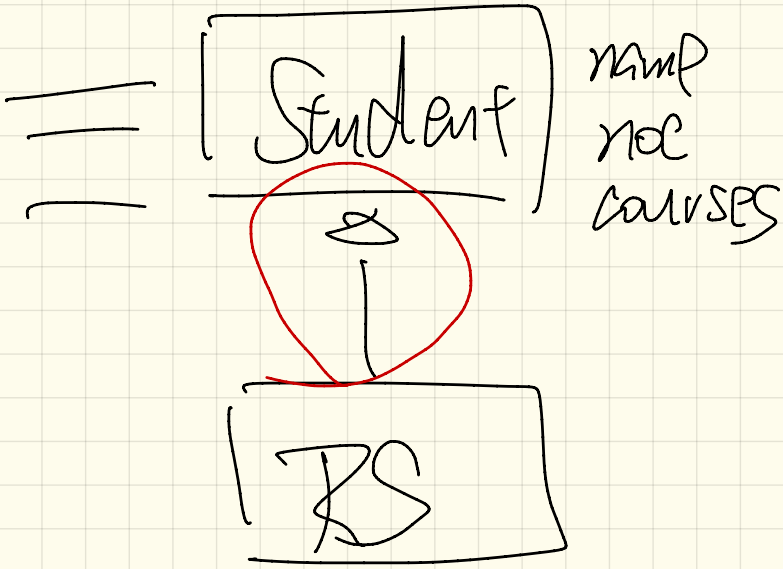
ResidentStudent rS = —

↙ wider expectations than Student (pr)

S = rS compiles ✓





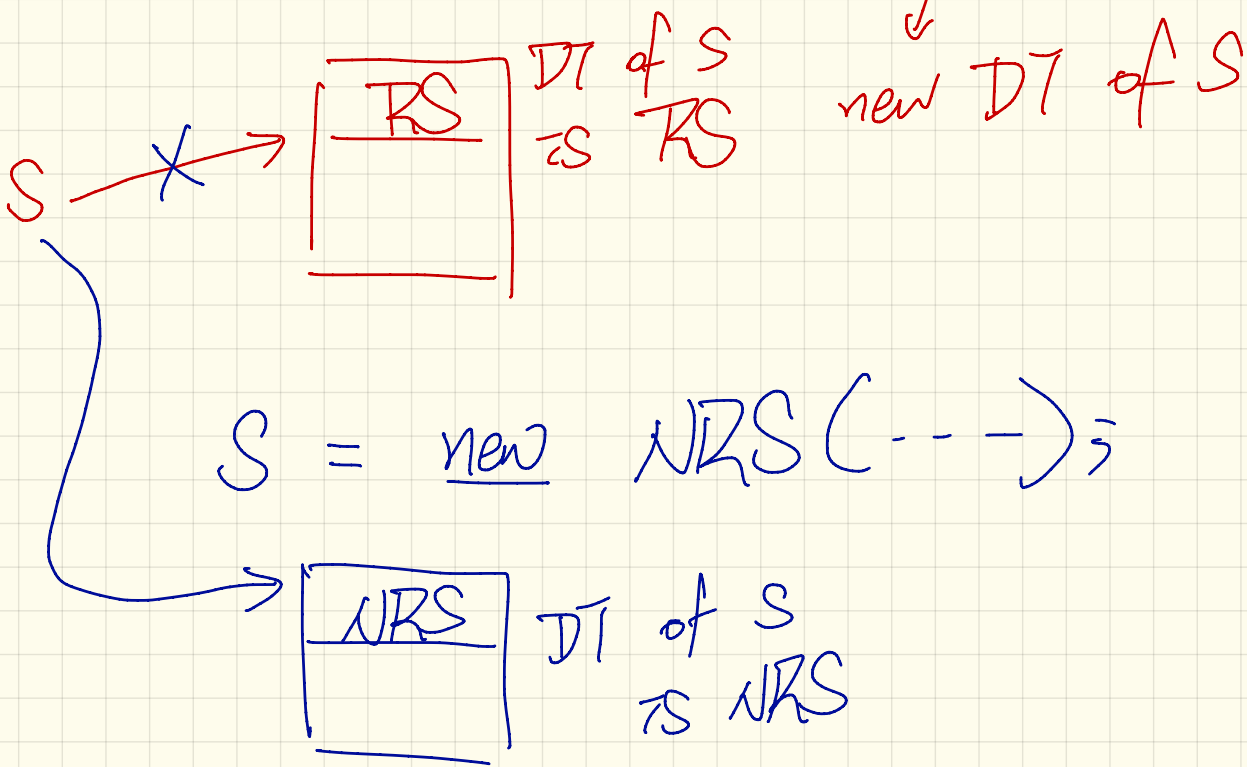


Student S = ...

ResiSet rs = ...

rs = S;

Student S = new RS(...);



① Student S = new Student();

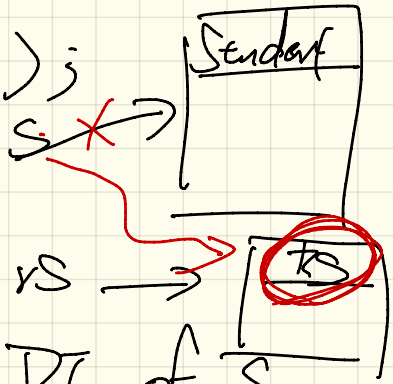
② Student rs = new RS();

③ Student nrs = new NRS();

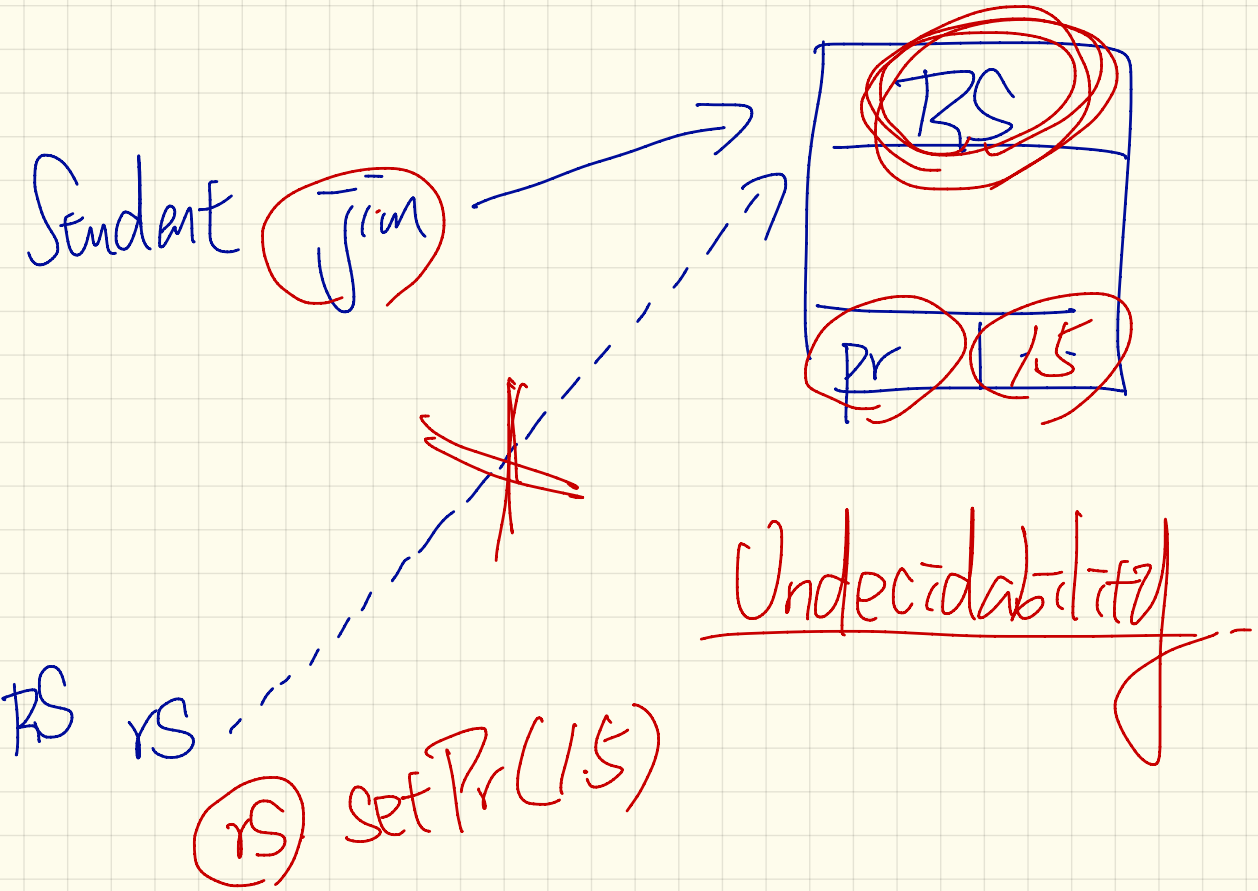
④ S = rs;

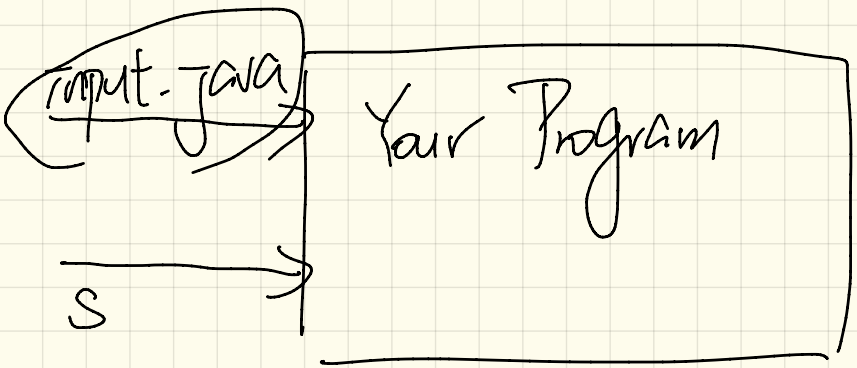
⑤ S = nrs;

ST: Student



	ST of S	DT of S	
①	Student	Student	s.getT()
②	Student	Student	
③	Student	Student	
④	Student	RS	s.getT()
⑤	Student	NRS	s.getT()


Student
Jim
RS
Pr
15
RS
rs
rs
setPr(15)
Undecidability



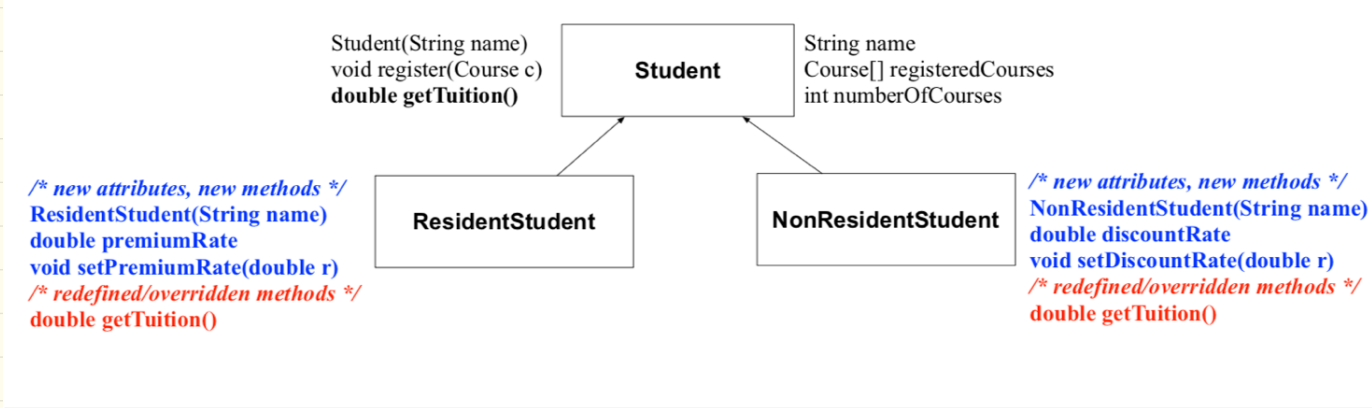
last DT of s
in
input.java.

input.java

```
Student s = null;
while(true){
}
s = new RS(..);
```

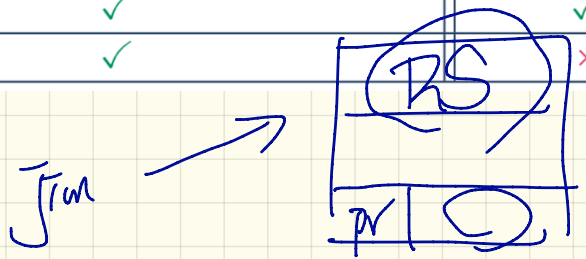
Lecture 22

Thursday Nov. 23



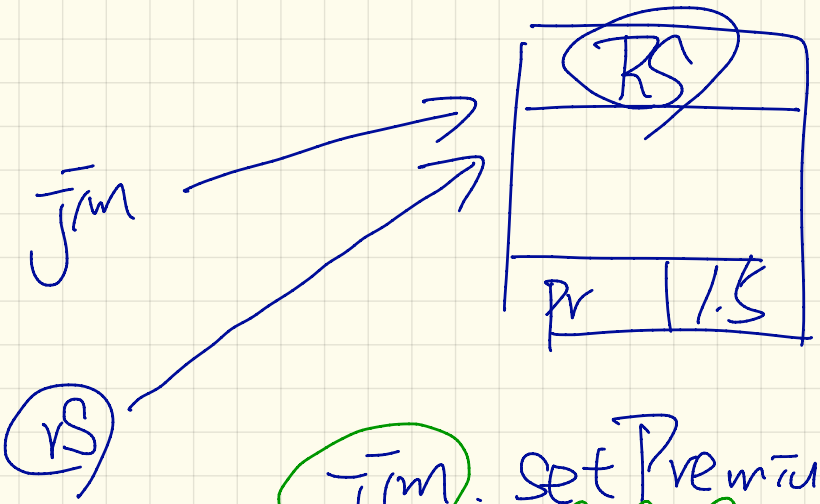

```
Student s = new Student("Stella");
ResidentStudent rs = new ResidentStudent("Rachael");
NonResidentStudent nrs = new NonResidentStudent("Nancy");
```

	name	rcs	noc	reg	getT	pr	setPR	dr	setDR
s.			✓				×		
rs.			✓				✓		×
nrs.			✓				×		✓



ST: ResidentStudent

* ResidentStudent rs = (ResidentStudent) jim;



ST: Student

jim.setPremiumRate(1.5) X

ST: Student

Student jim = new RS(..);

① ResidentStudent rs = (RS) jim;
rs.setPr(1.5);

② ((RS) jim).setPr(1.5)

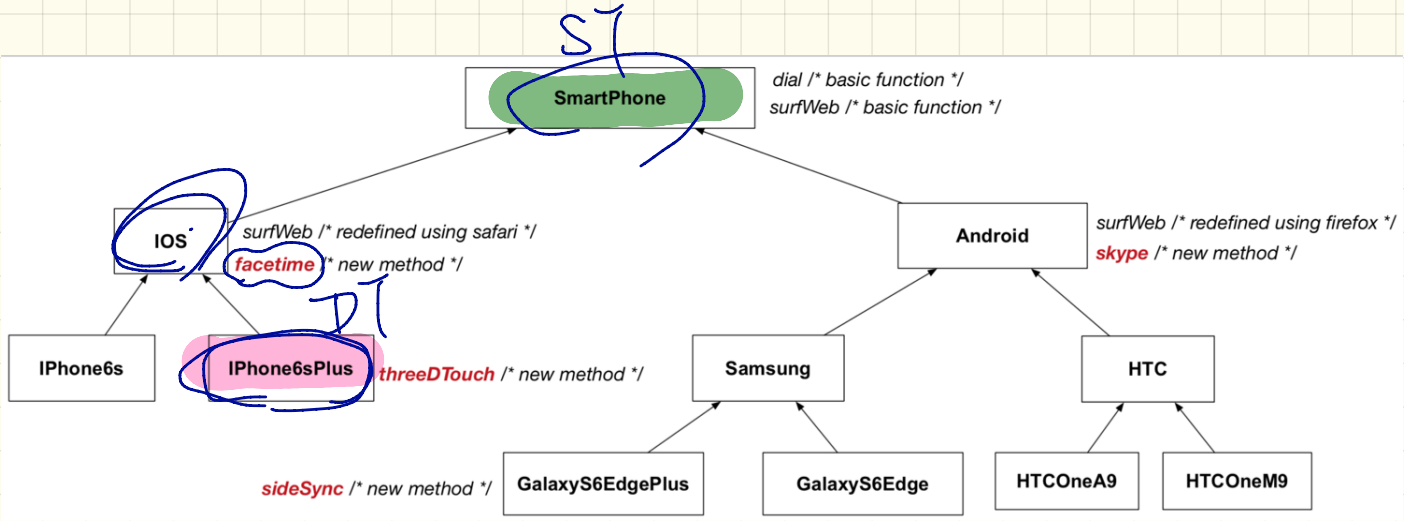
③ (RS){jim.setPr(1.5);}

Student jim = new RS(...);

Student S;

S = jim;

Student ST: Student

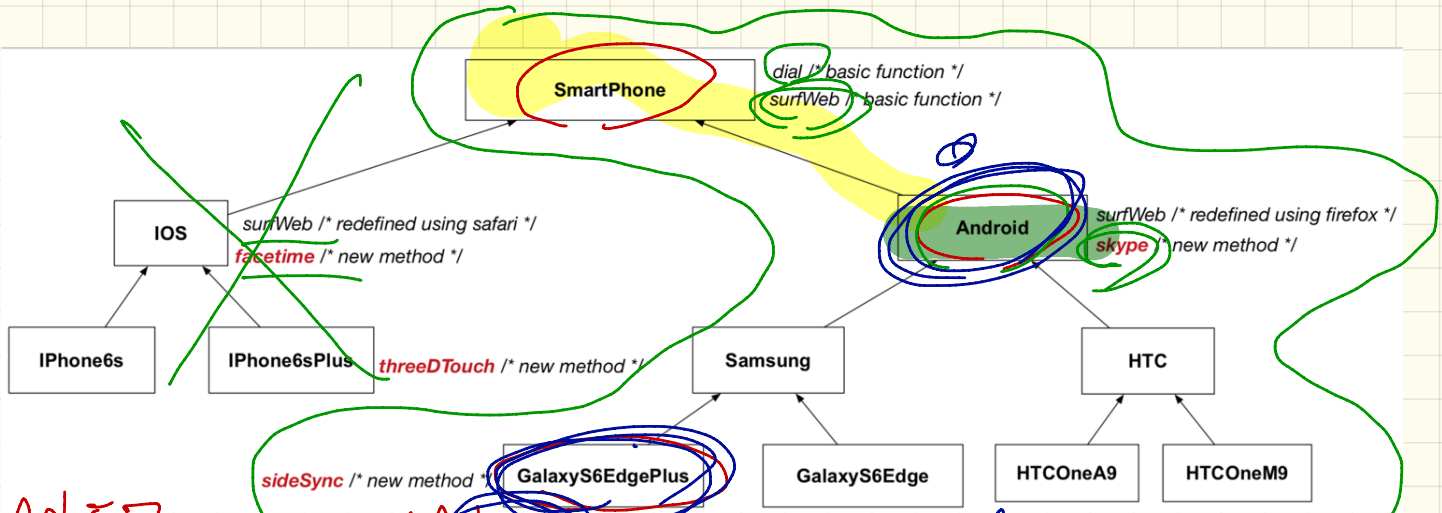


7.

IOS forHeeyeon = aPhone;

forHeeyeon → IP6sPlus

SP



GSbEP ga = (GSbEP) phone; → ST: Android

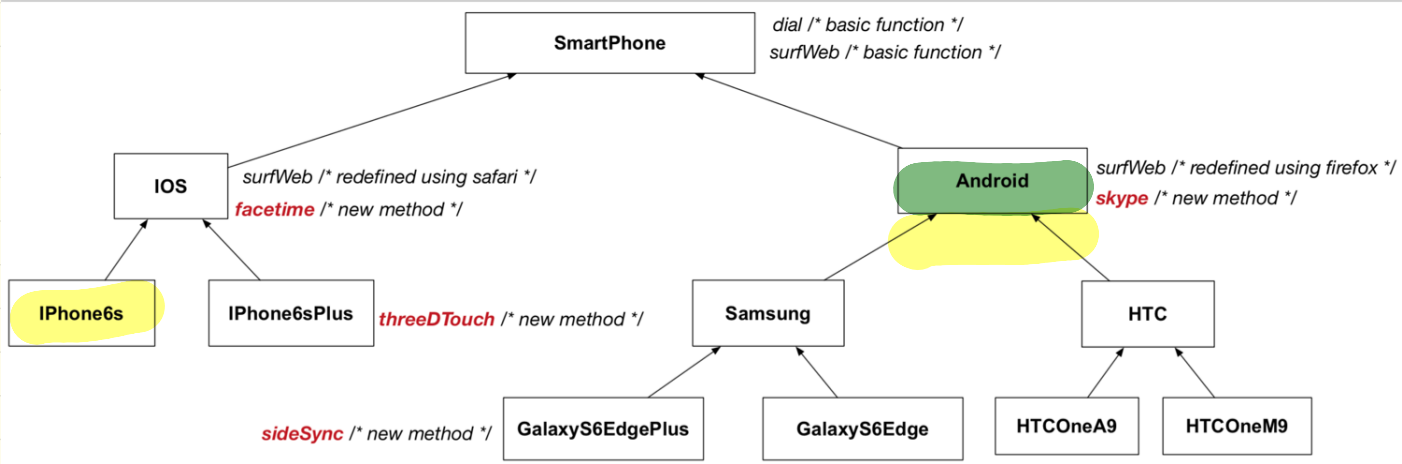
Android ga.sideSyn.

phone = | ___ ;

phone.dial surfWeb skype

SmartPhone SP = (SmartPhone) phone; → ST: Android

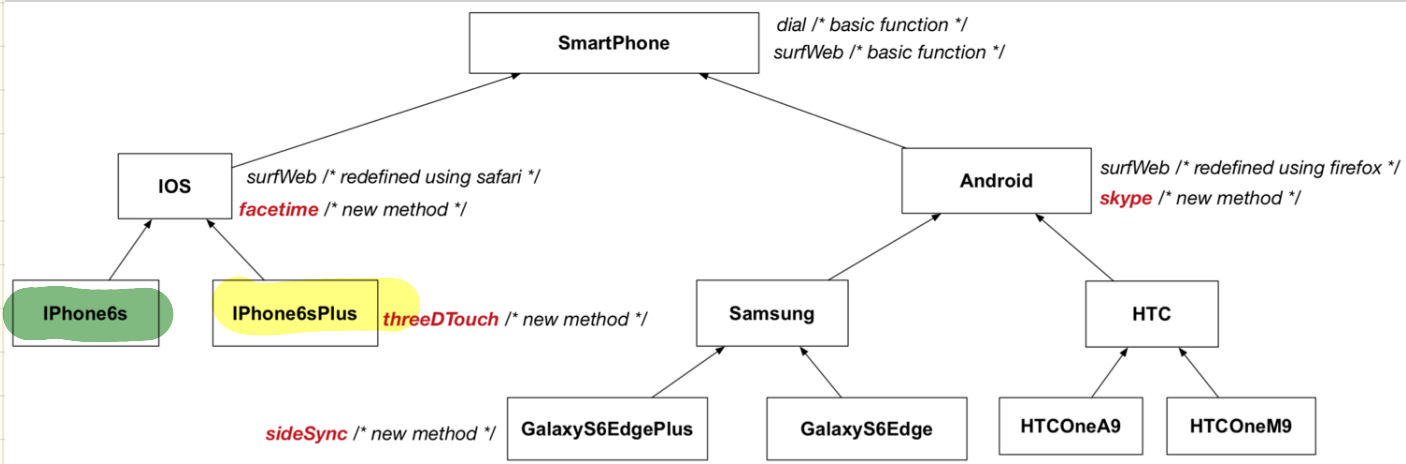
SP.dial surfWeb skype ✗



Android phone = [X]

IPhone6s ip6s = (IP6s) phone;

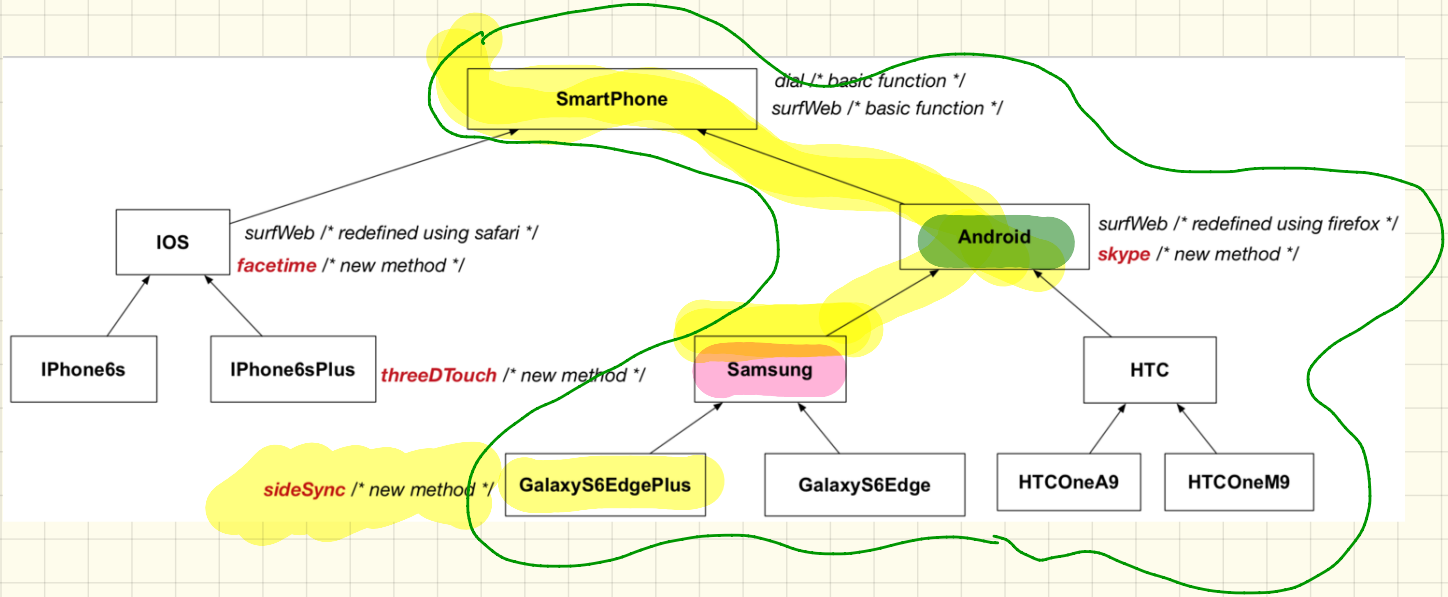
ST: Android



IP6s p = [. —]

IP6sPlus p2 = (IP6sPlus)

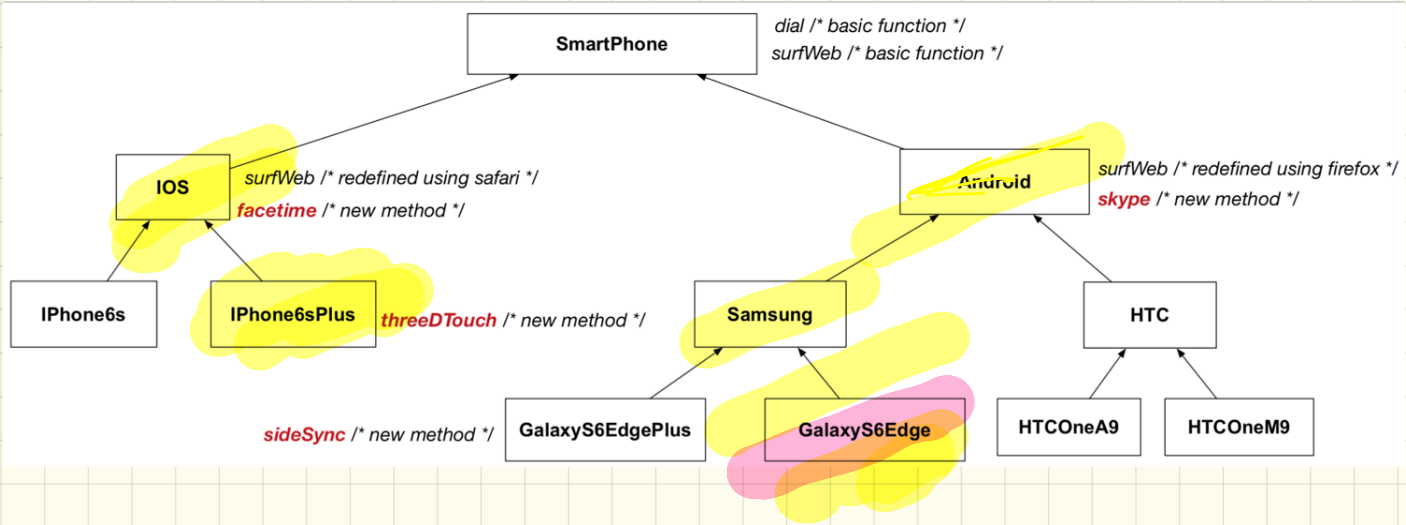
p 是 IP6s ✗

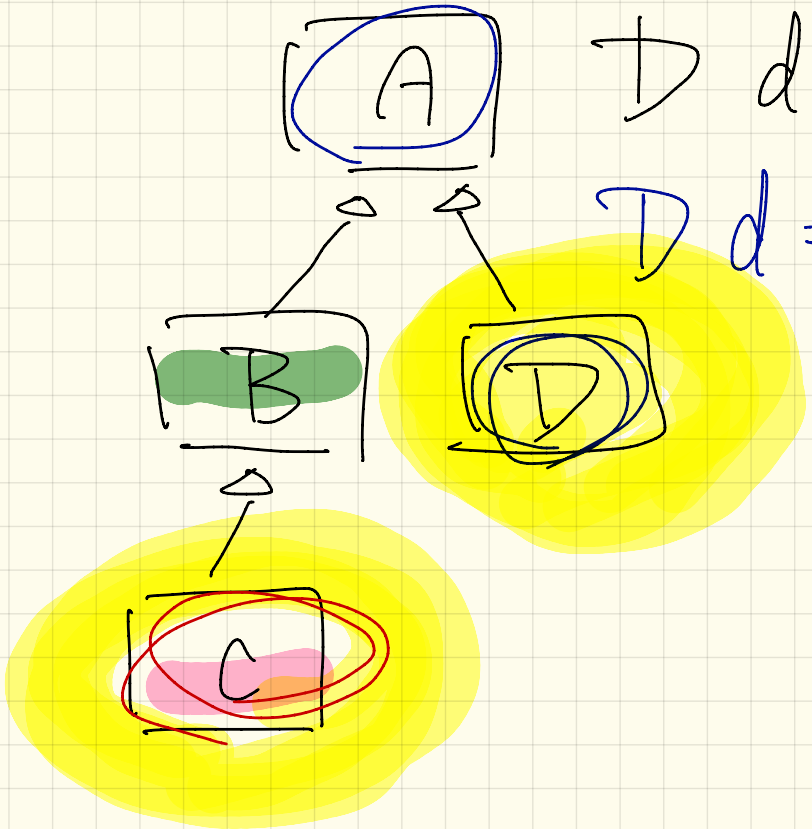


Android p = new Samsung();

① (Samsung) p

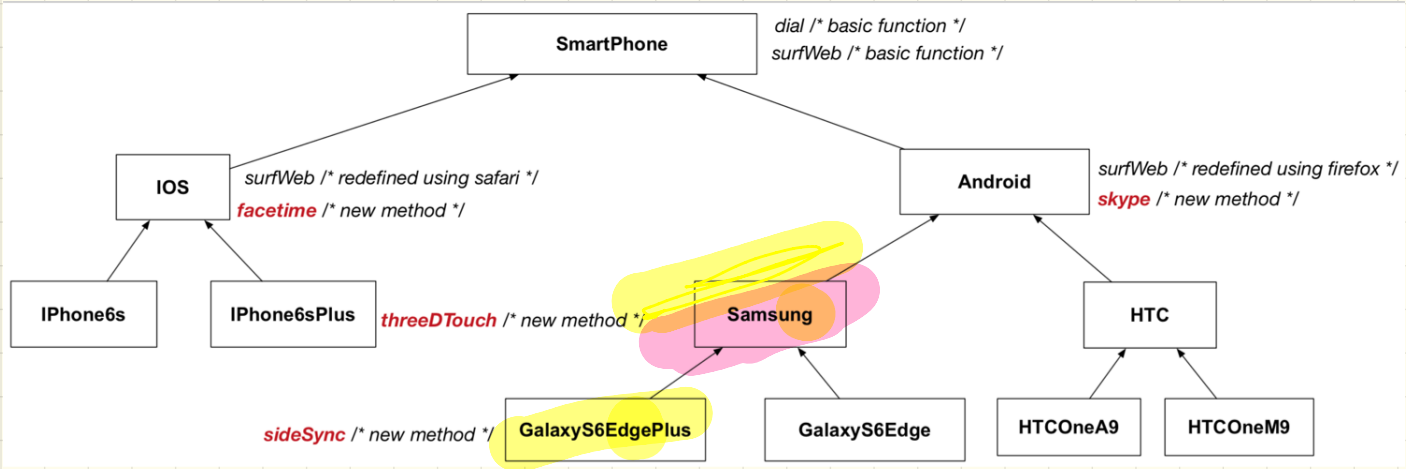
② (GalaxyS6EP) p


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



D d = (D) b;

D d = (D) ((A) b)

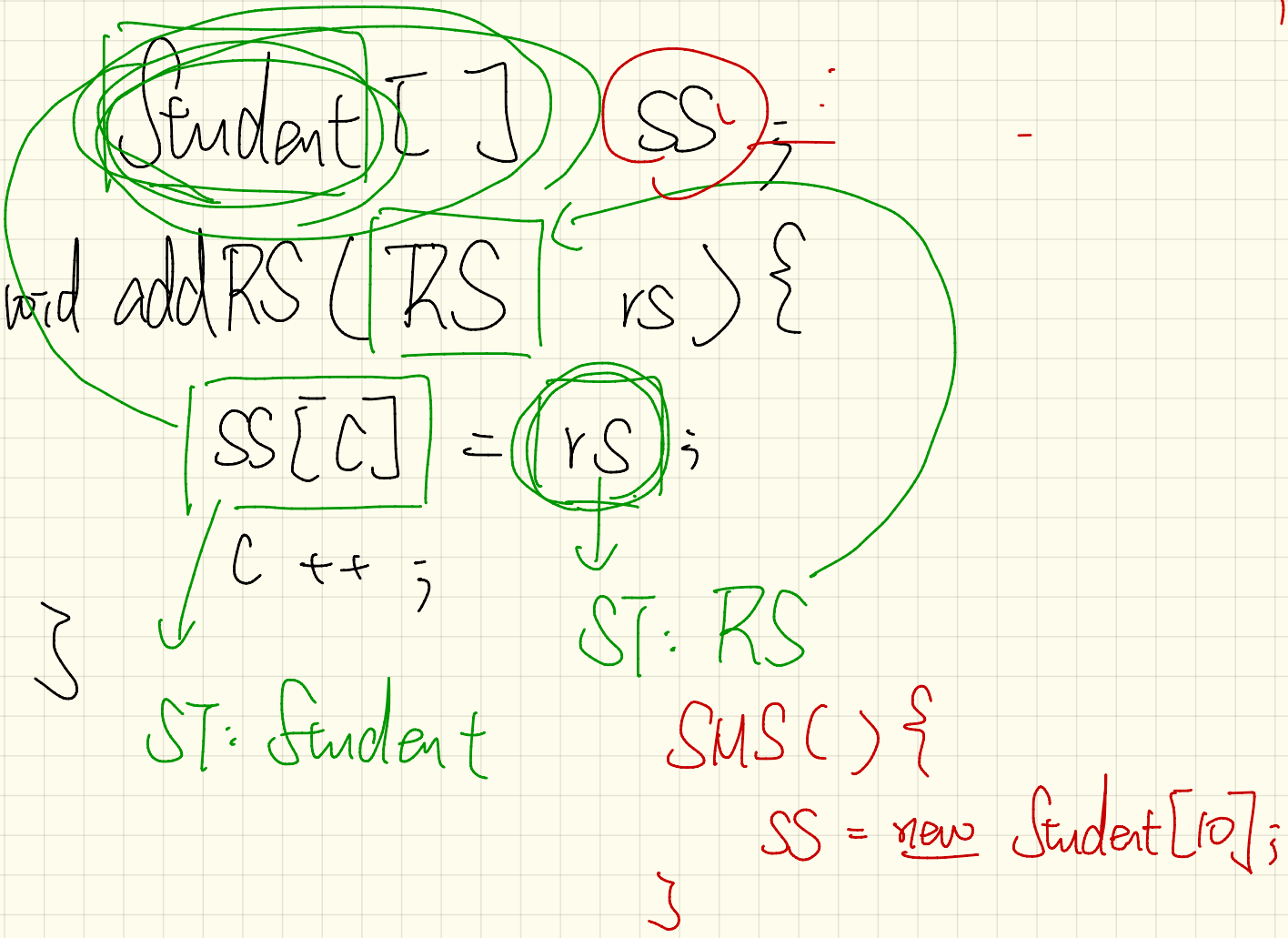

SmartPhone
dial /* basic function */
surfWeb /* basic function */
IOS
surfWeb /* redefined using safari */
facetime /* new method */
Android
surfWeb /* redefined using firefox */
skype /* new method */
IPhone6s
IPhone6sPlus
threeDTouch /* new method */
Samsung
HTC
sideSync /* new method */
GalaxyS6EdgePlus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

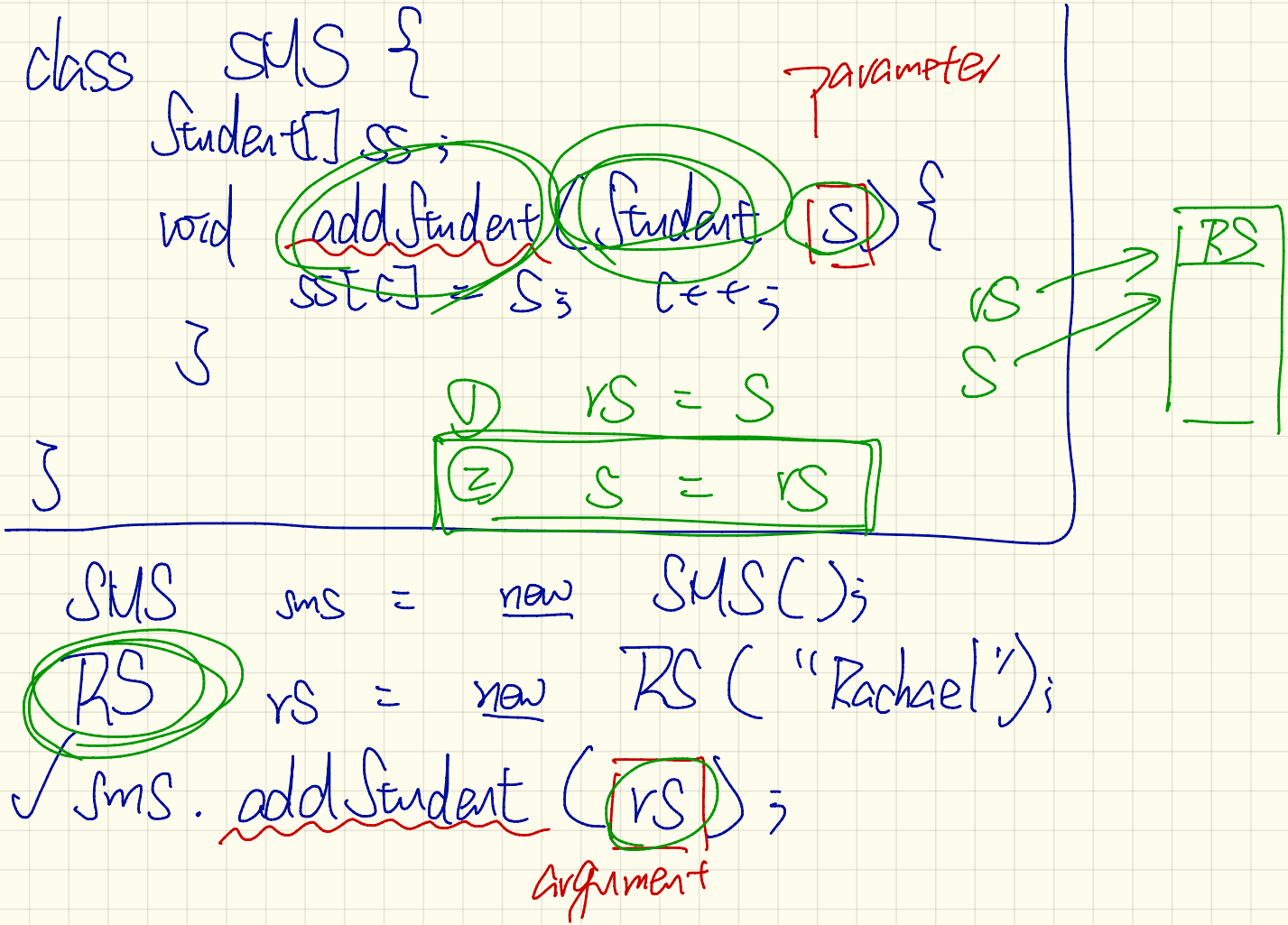
declaration → Student[] ss;

Student s;

① Static type of each item in ss is Student.

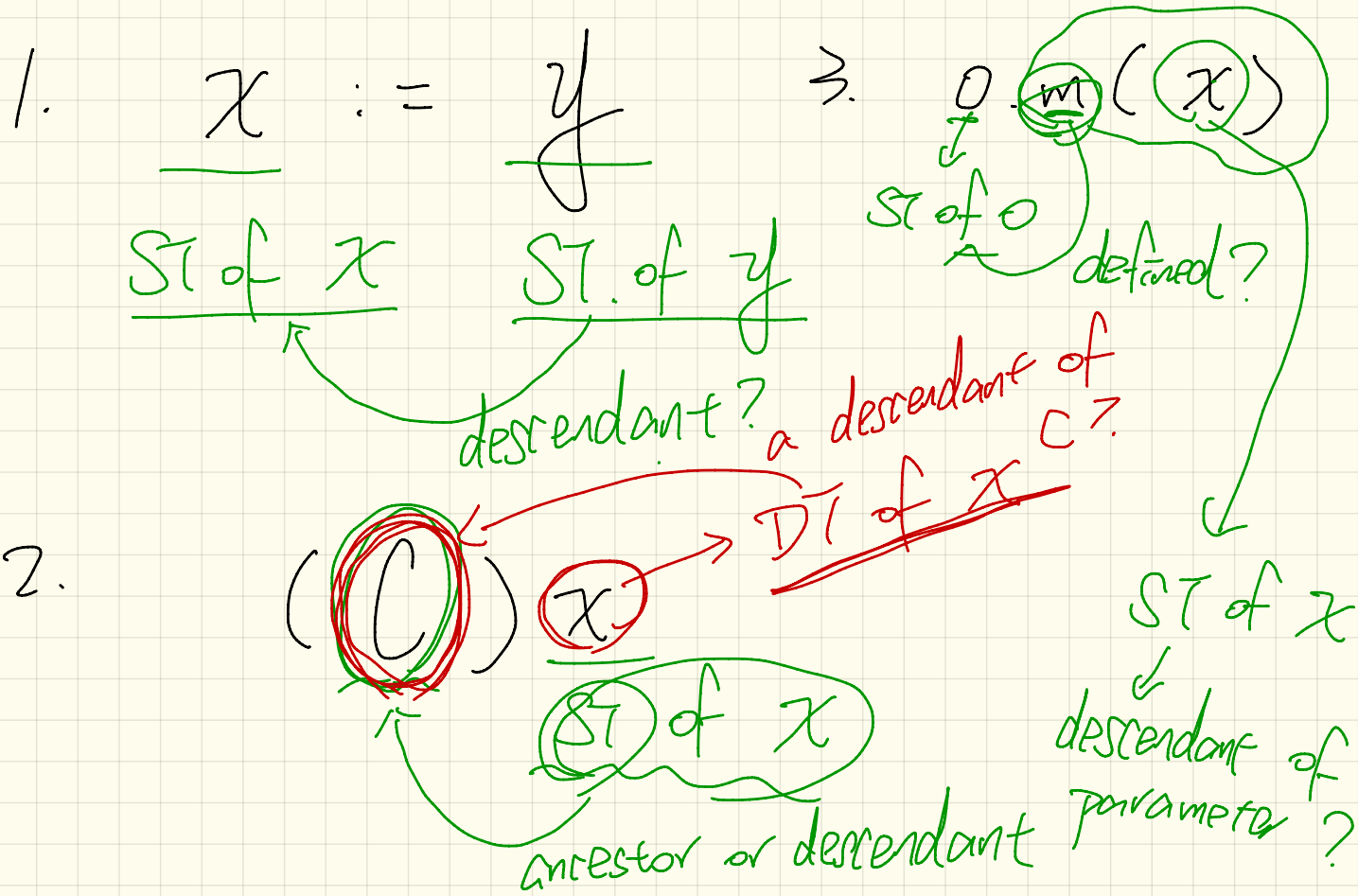
② dynamic type of each item in ss is student.

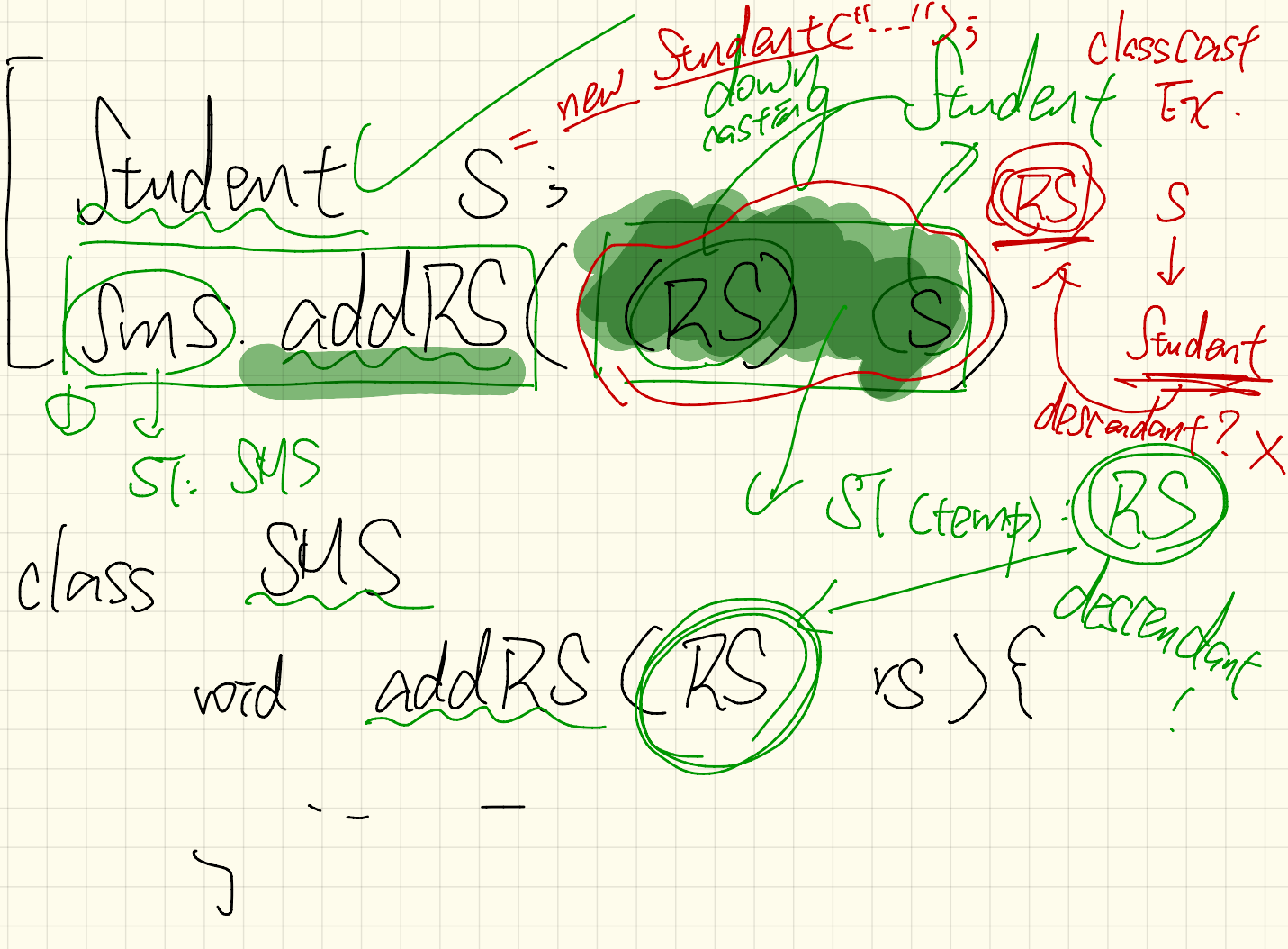

Student[]
SS;
void addRS(RS rs){
SS[c] = rS;
c++;
}
ST: Student
ST: RS
SMS(){
SS = new Student[10];
}

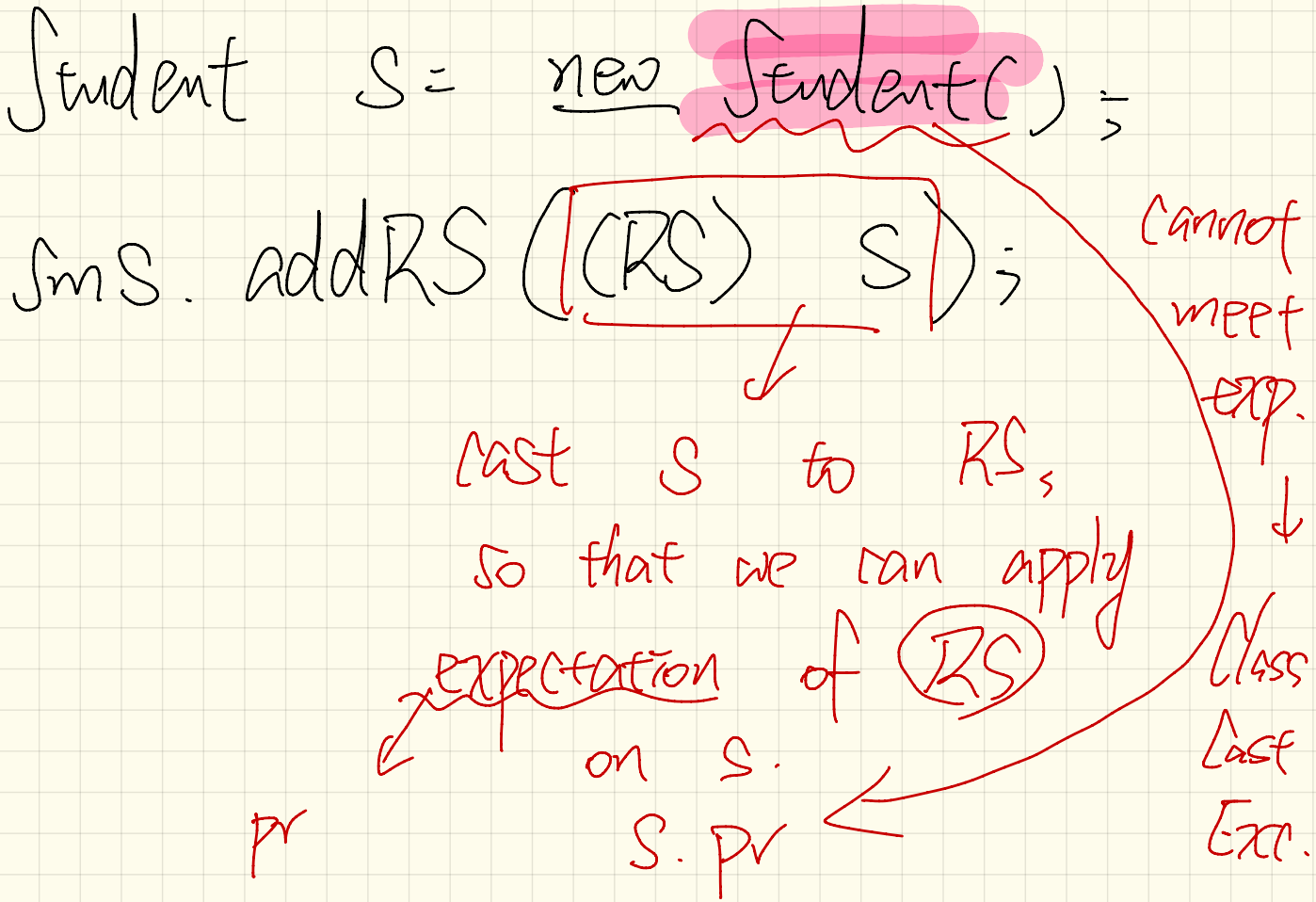

class SMS {
Student[] ss;
void addStudent (Student s) {
ss[c] = s; c++;
}
}
parameter
rs = s
s = rs
RS
rs
s
SMS sms = new SMS();
RS rs = new RS("Rachael");
sms.addStudent(rs);
argument

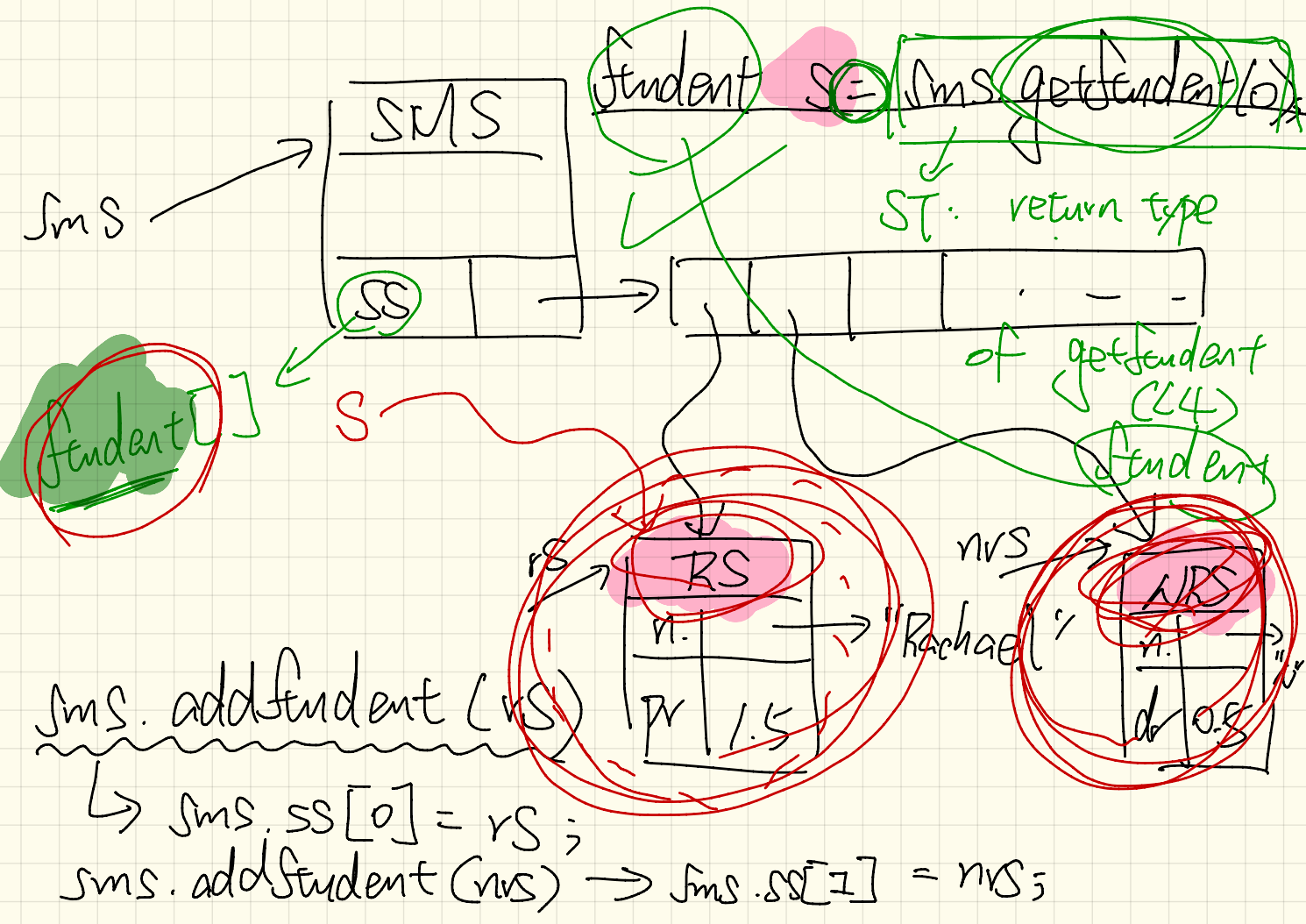
Lecture 23

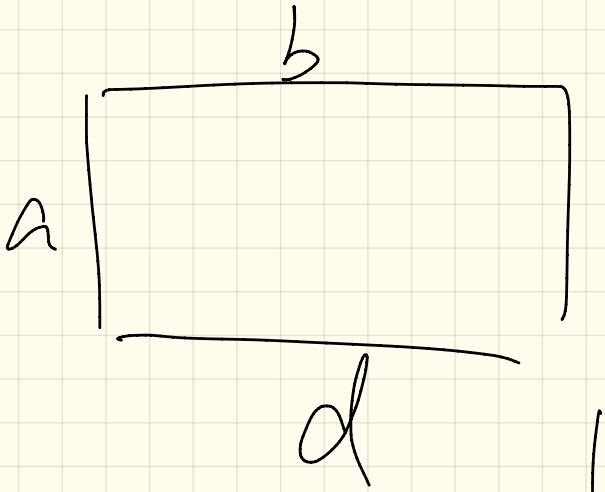
Tuesday Nov. 28


1.
x := y
ST of x
ST. of y
descendant?
2.
(C) x
DT of x
a descendant of C?
ST of x
ancestor or descendant
3.
o.m(x)
ST of o
defined?
ST of x
descendant of parameter?


Student s;
sms.addRS(RS s)
new Student("...");
down casting
Student
ClassCastEx.
RS
s
Student
descendant? X
st: SMS
st (temp): RS
class SMS
void addRS(RS rs){
descendant!


Student S = new Student();
fmS.addRS((RS) S);
cast S to RS,
so that we can apply
expectation of RS
on S:
pr
S.pr
cannot
meet
exp.
Class
Cast
Exc.

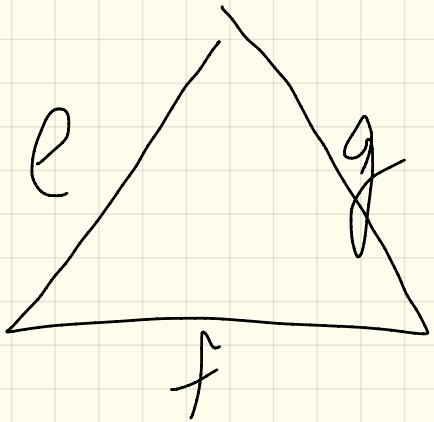

Student s = sms.getStudent(0);
ST: return type
of getStudent ()
Student
sms
SMS
ss
Student[]
s
rs
RS
n.
pr
1.5
"Rachael"
nrs
NRS
n.
dr
0.5
sms.addStudent(rs)
sms.ss[0] = rs;
sms.addStudent(nrs) → sms.ss[1] = nrs;



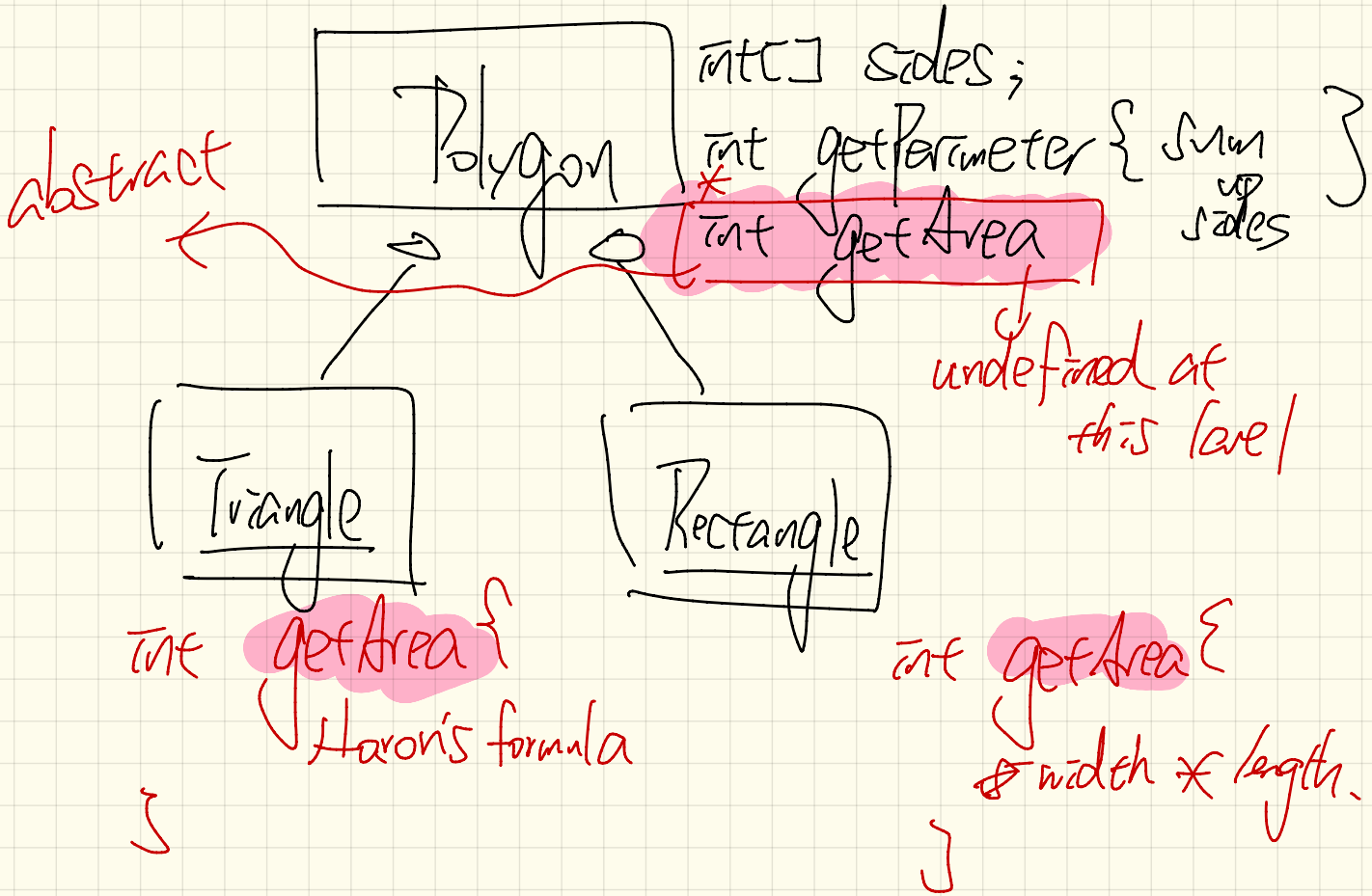
$a + b + c + d$

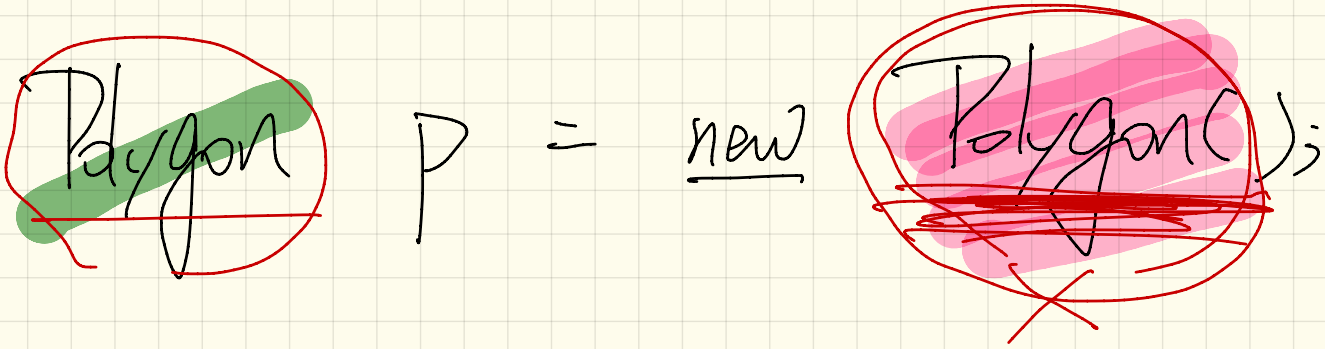
int[] sides;

perimeter



$e + g + f$


Polygon
int[] sides;
int getPerimeter { sum up sides }
abstract
int getArea
undefined at this level
Triangle
int getArea {
Haron's formula
}
Rectangle
int getArea {
width * length.
}

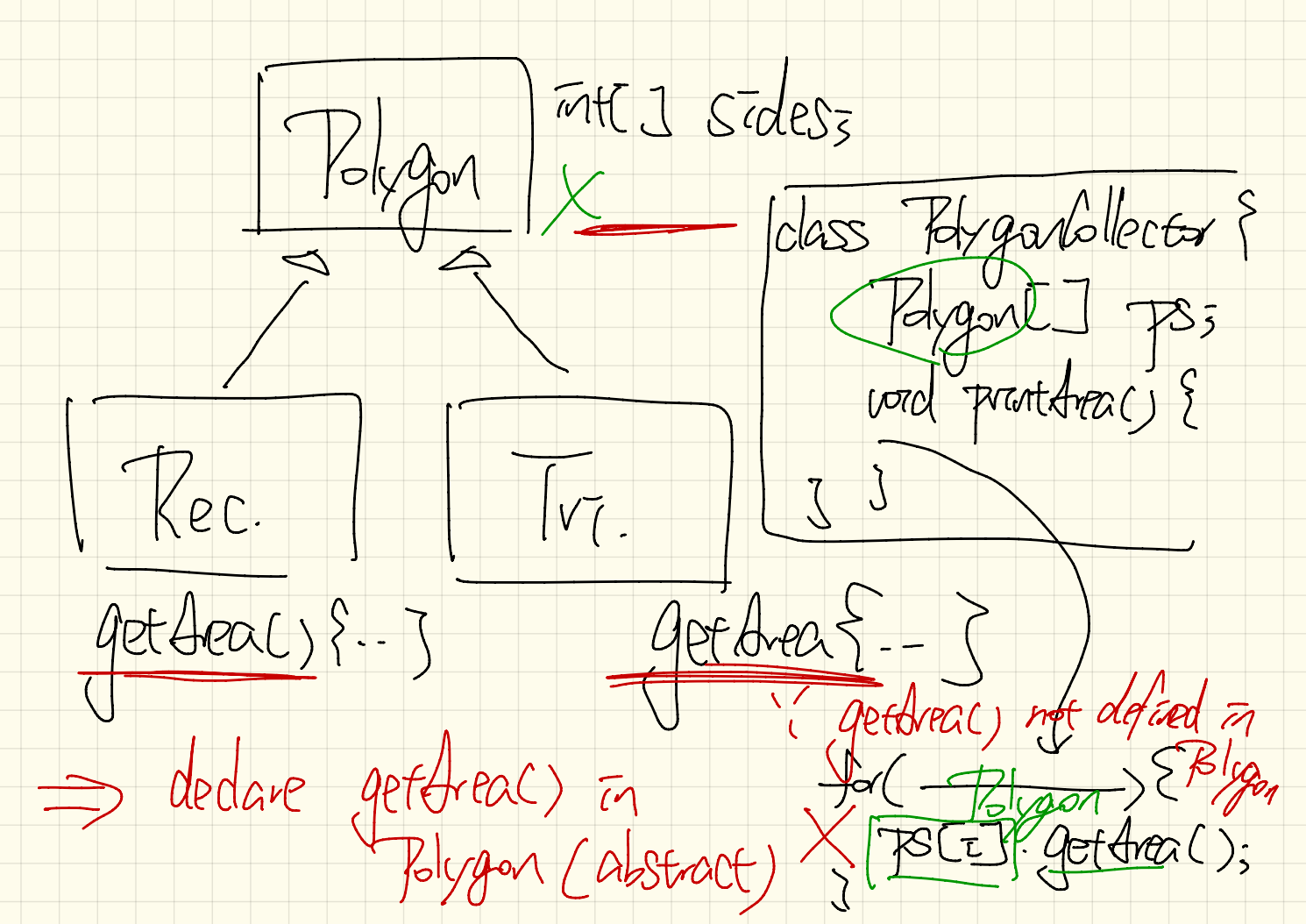


abstract class Polygon { }

can't use abstract class as DT.

p.getArea() ?

crash :(undefined!


Polygon
int[] sides;
X
class PolygonCollector {
Polygon[] PS;
void printArea() {
}}
Rec.
Tri.
getArea(){..}
getArea{..}
getArea() not defined in Polygon
for(Polygon){
PS[i].getArea();
}
⇒ declare getArea() in Polygon (abstract)
X

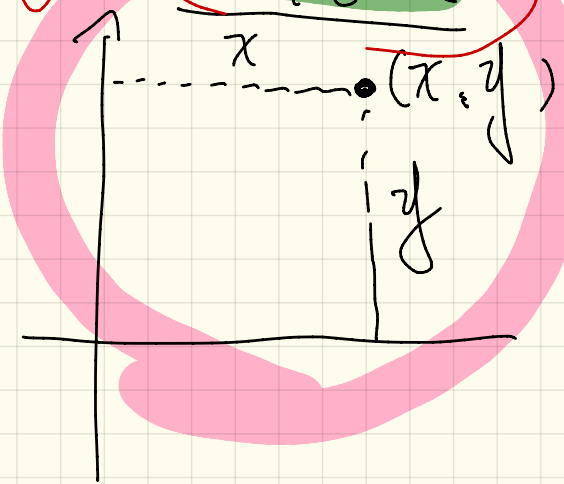
Lecture 24

Thursday Nov. 30

2-D points

getX()
getY()

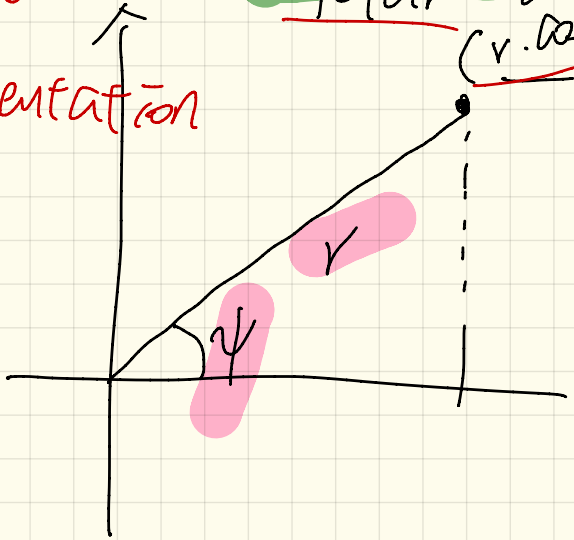
Cartesian

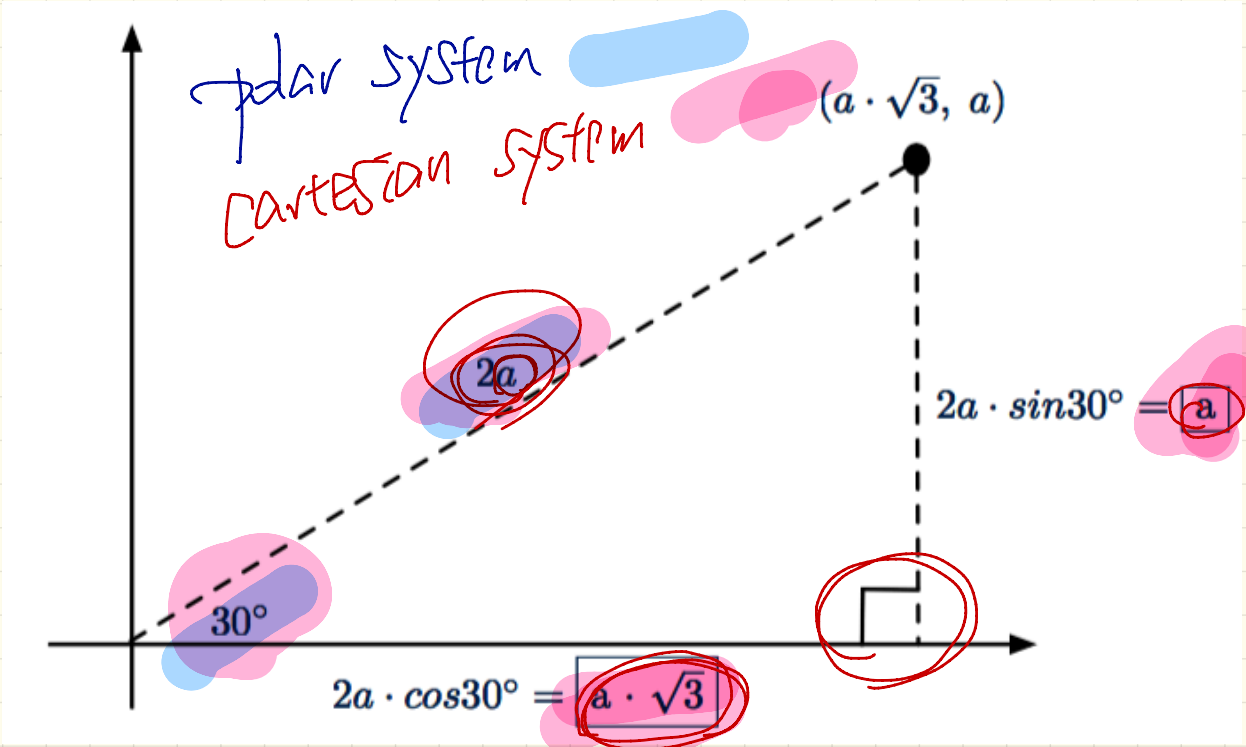


implementation

Polar

(r.cosψ, r.sinψ)



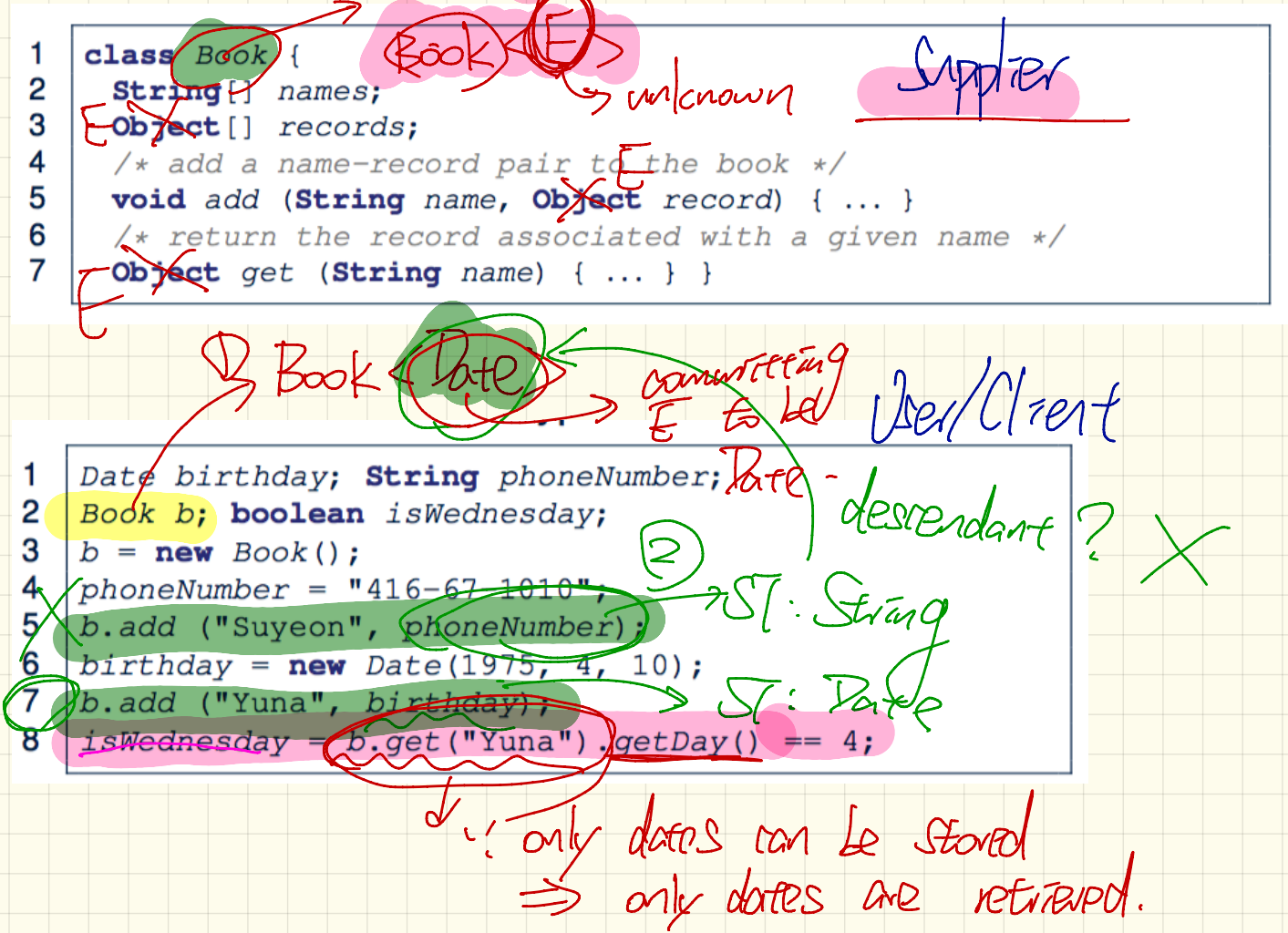

polar system
cartesian system
$(a \cdot \sqrt{3},\ a)$
$2a$
$2a \cdot sin30° = a$
$30°$
$2a \cdot cos30° = a \cdot \sqrt{3}$

```
1  class Book {
2    String[] names;
3    Object[] records;
4    /* add a name-record pair to the book */
5    void add (String name, Object record) { ... }
6    /* return the record associated with a given name */
7    Object get (String name) { ... } }
```

```
1  Date birthday; String phoneNumber;
2  Book b; boolean isWednesday;
3  b = new Book();
4  phoneNumber = "416-67-1010";
5  b.add ("Suyeon", phoneNumber);
6  birthday = new Date(1975, 4, 10);
7  b.add ("Yuna", birthday);
8  isWednesday = b.get("Yuna").getDay() == 4;
```

String

ST: object


class Book {
String[] names;
Object[] records;
/* add a name-record pair to the book */
void add (String name, Object record) { ... }
/* return the record associated with a given name */
Object get (String name) { ... } }
Book<E>
unknown
Supplier
Book<Date>
committing E to Date
User/Client
Date birthday; String phoneNumber;
Book b; boolean isWednesday;
b = new Book();
phoneNumber = "416-67-1010";
b.add ("Suyeon", phoneNumber);
birthday = new Date(1975, 4, 10);
b.add ("Yuna", birthday);
isWednesday = b.get("Yuna").getDay() == 4;
Date - descendant ? X
S: String
S: Date
only dates can be stored
⇒ only dates are retrieved.

```
1  class Book {
2    String[] names;
3    Object[] records;
4    /* add a name-record pair to the book */
5    void add (String name, Object record) { ... }
6    /* return the record associated with a given name */
7    Object get (String name) { ... } }
```

Book<String> tb;

Book<Date> bd;

bd.get(...) ST? Date

tb.get(...) ST? String

①

```
Node<String> n =
        new Node<String>("Tom", null);
```

②

```
Node<String> n =
        new Node<>(- - -);
```

END

ALL THE BEST !