Tutorial on

Object-Oriented Programming in Java

Assumptions:

- Educational Github account
- Private repository for EECS1021 workspace
- Cloned to your computer desktop

✓Workspace (eecs2021-lab-workspace)

↳ ✓Java project

  ↳ ✓package (≈ folder)

    ↳ ✓Java class (≈ file)

    ~~ .java

You can only execute a Java class with the "main" method.

- Compilation Stage (writing)
- Execution Stage (runtime)

main method : sequential composition of programming statements

- Print statement

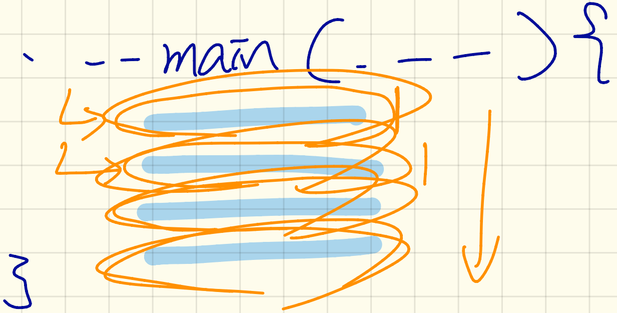
✓- Assignment

✓- If-statements

✓- Loops

At compile time, statements are separated by semicolons ;

At runtime, from top to bottom, one line is executed after another.



3 * (7 / 3) + (7 % 3) = 7

quo.

Math /

b

5 / 2 +

2 −

5.2

5.0 / 2 2.5 ×

5 / 2.0 2.5 /

5 / 2.5 2.0 quotient

5.2 / 2.0

2.6 remainder

number literals:

2

3

4.6

7

Java

+

−

*

/

%

modulo

int / int

&

int. / int.

precise result.

7 / 3 2

Integer / Integer

7 % 3 1

$$5 + \underline{\boxed{2 / 2}} = 6$$

$$\boxed{\underbrace{(5 + 2)}_{\boxed{7}}}\boxed{/}\boxed{2} \qquad \cancel{\underline{3.5}}\ ? \qquad 3$$

$$X\ (5+2).0\ /\ 2$$

$$\underbrace{(5+2)}_{7} / \underline{2.0} \qquad \boxed{3.5}.$$

Concatenation

println ("(5 + 2) / 2 is" + ((5+2) /2))

String literal

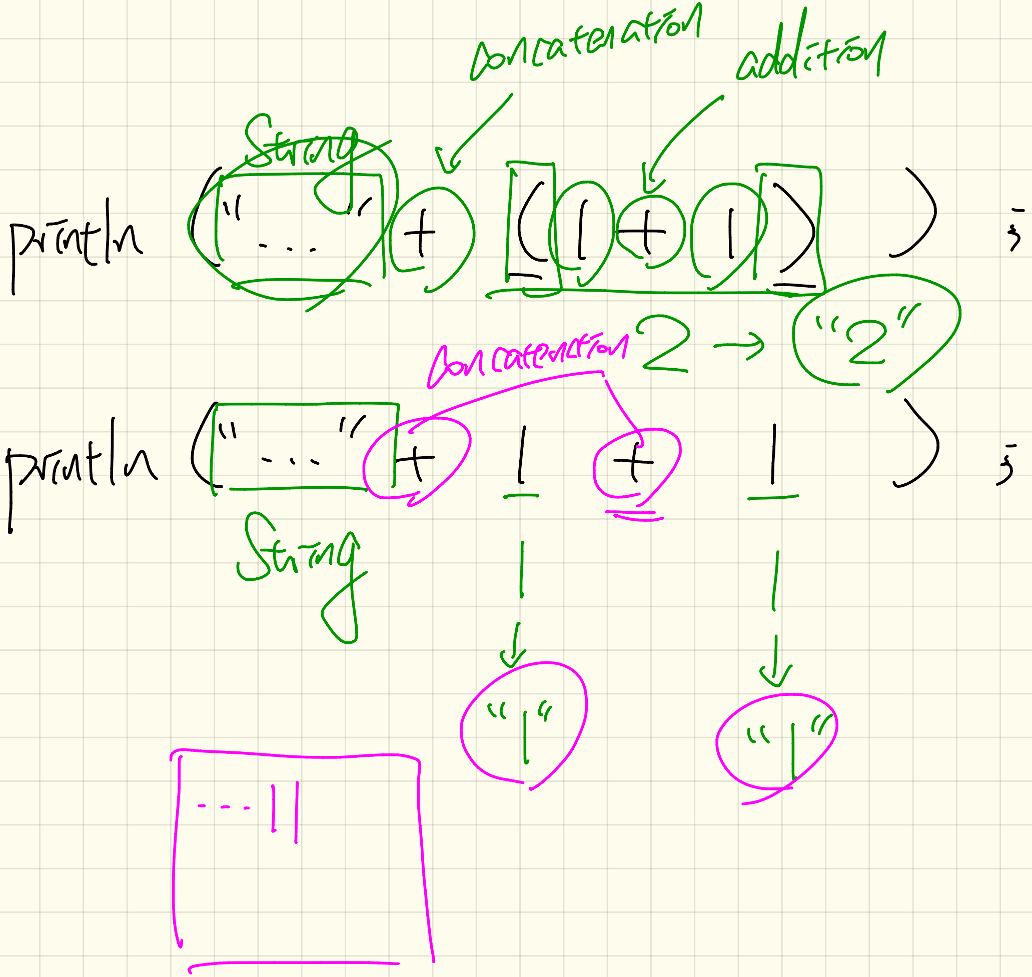
3

"3"

Number literal

(5+2) /2 is 3

Lassonde School of Eng.

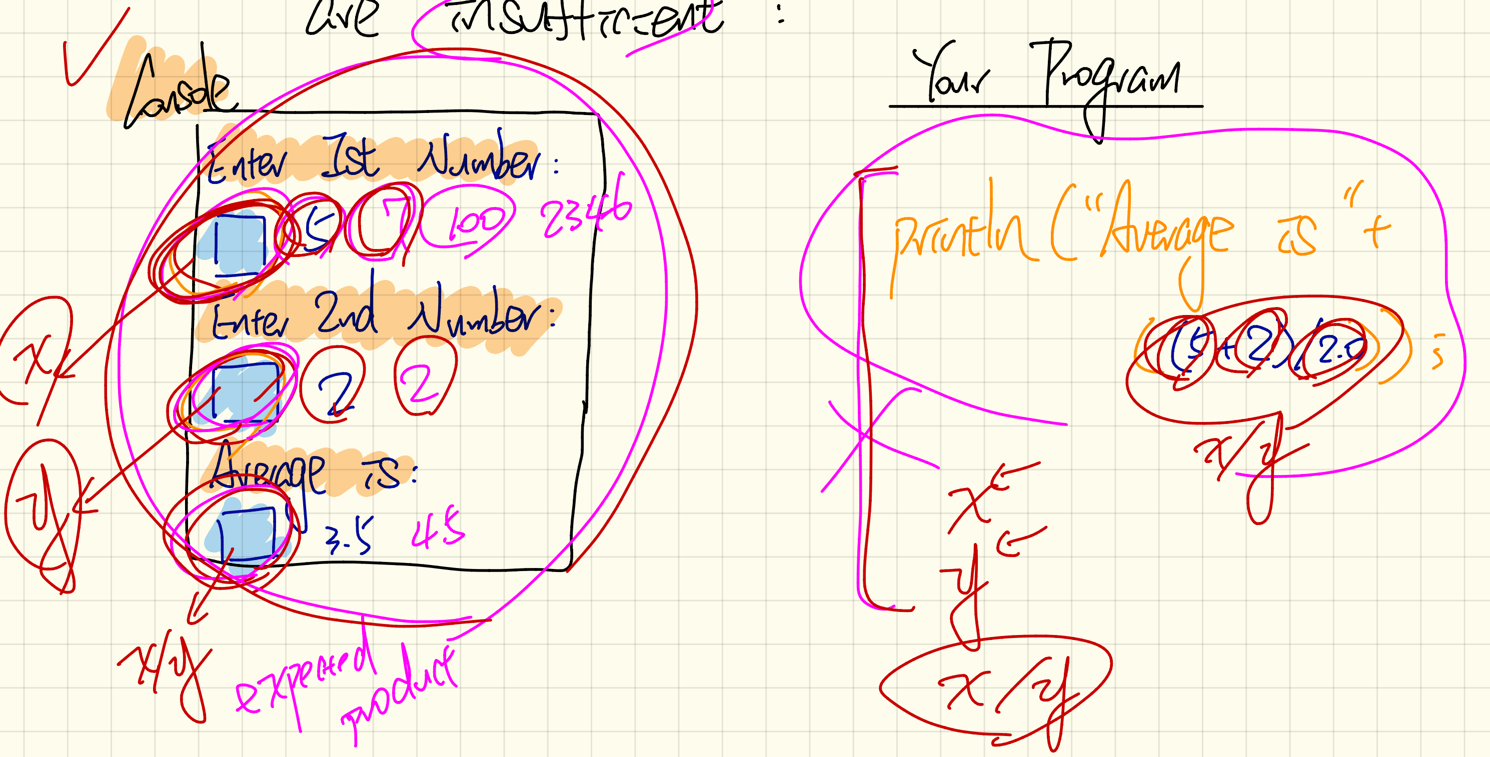


```
println("..." + (1 + 1));
```

```
println("..." + 1 + 1);
```

"a" 2.6 5

Scenarios where String and Number literals are insufficient :

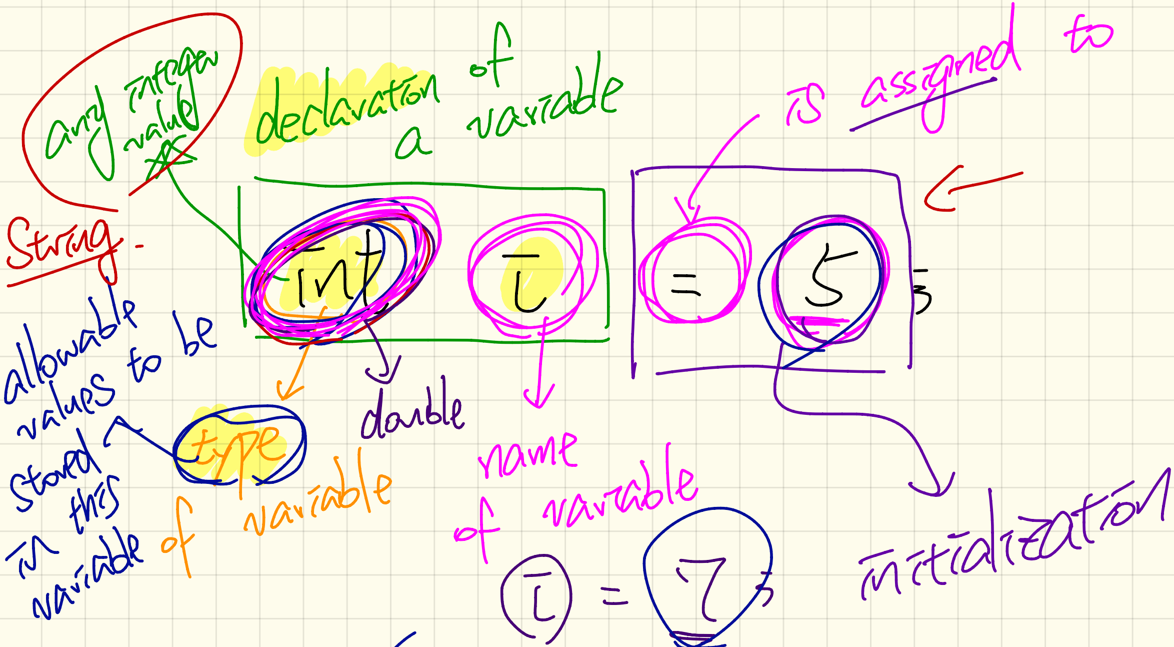


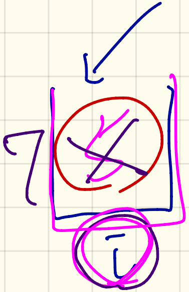
Console

```
Enter 1st Number:
5 7 100 2346
Enter 2nd Number:
2 2
Average is:
3.5 4.5
```

Your Program

```
println("Average is " +
    (5+2)/2.0);
```





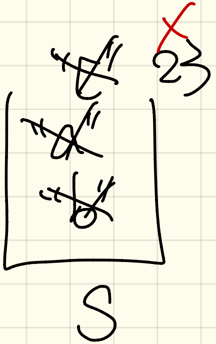
**

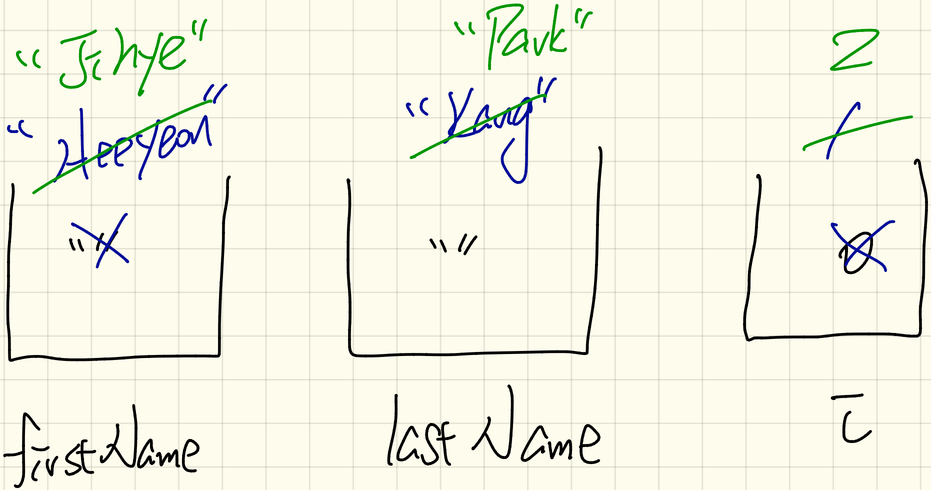
- You can only declare the type of a variable once. ✓
- You may change the stored vavlue in a variable.

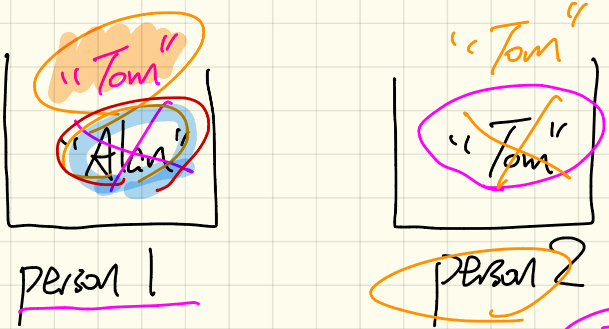
initialization (after declaration)

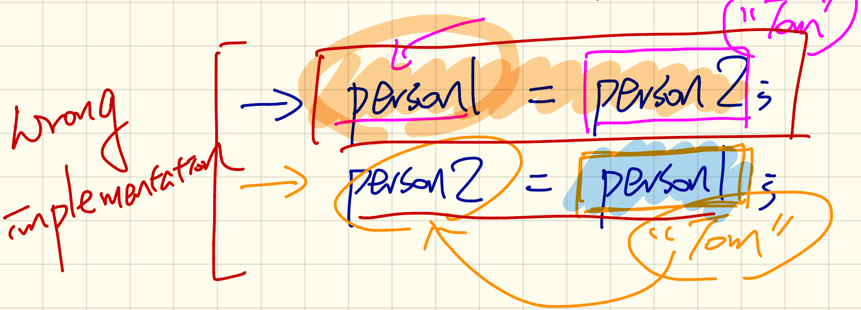
```
String s = "a";
s = "b";
```

re-assignment



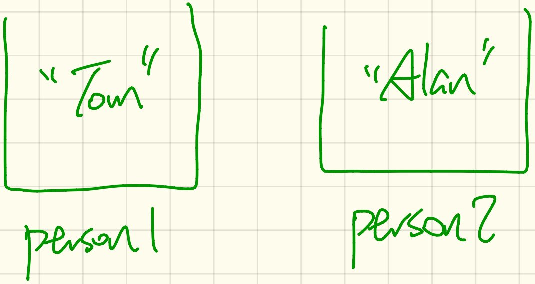

"Jihye"
~~"Heeyeon"~~
~~""~~
firstName
"Park"
~~"Kang"~~
""
lastName
2
~~1~~
~~0~~
i

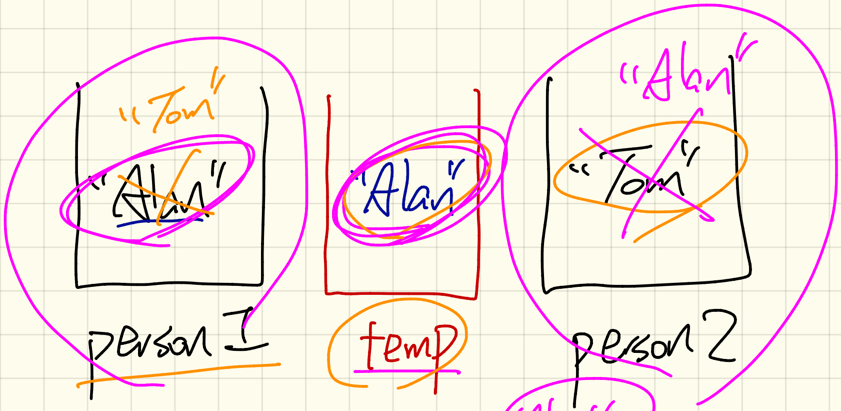




```
person1 = person2;
person2 = person1;
```

Want to achieve:

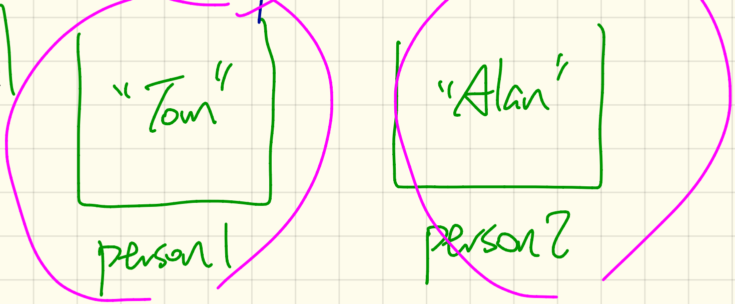




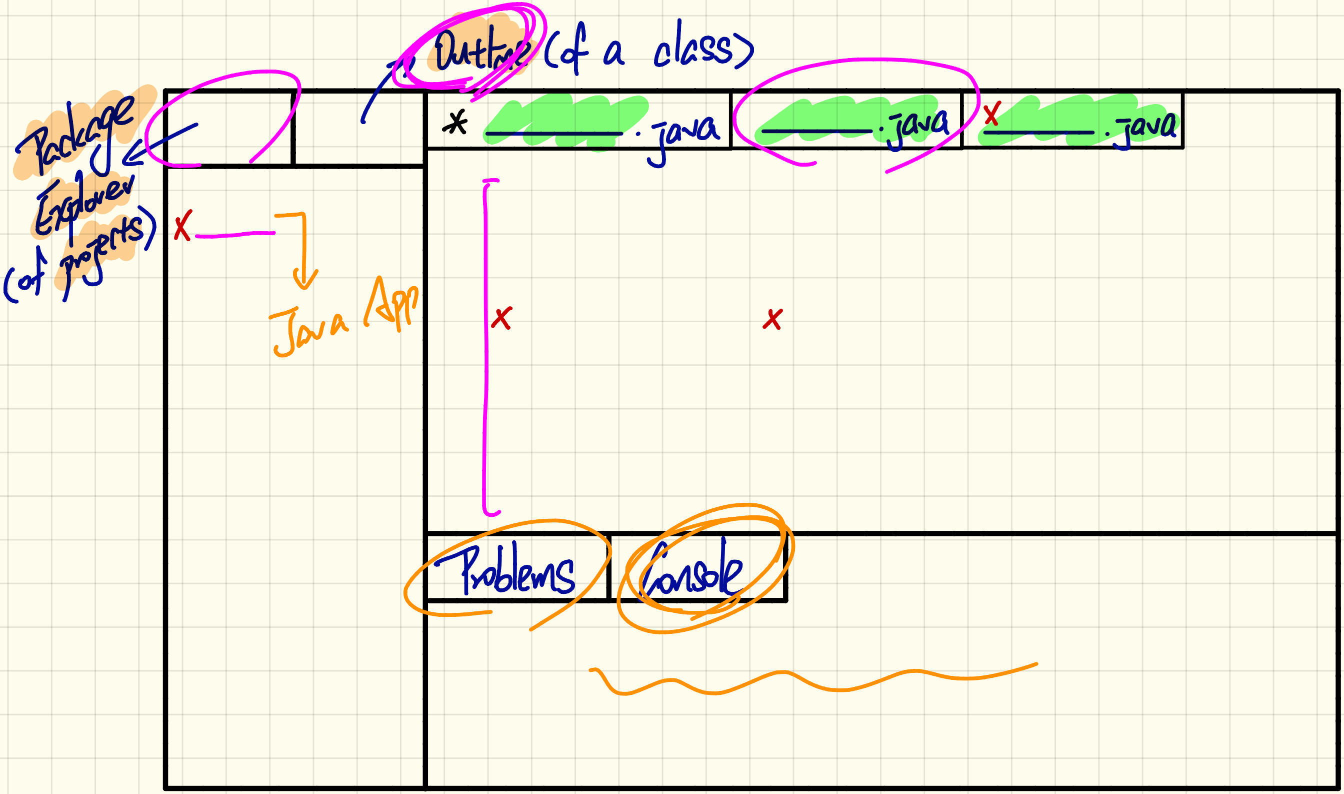
Correct Implementation

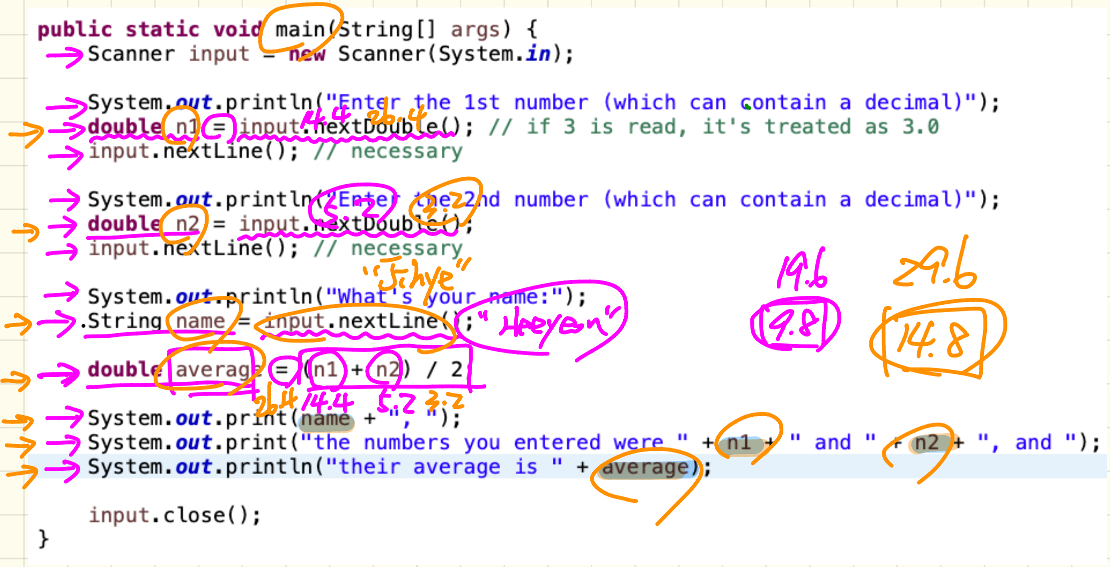
```
temp = person1; "Alan"
person1 = person2; "Tom"
person2 = temp; "Alan"
```

Want to achieve:

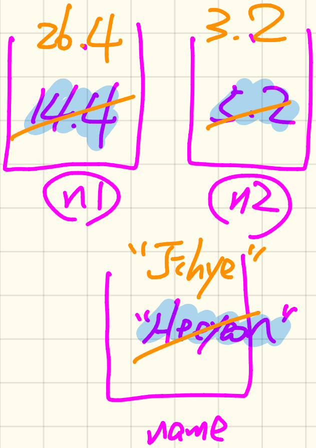


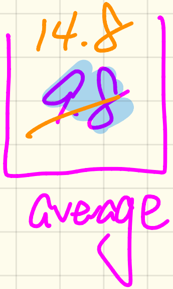
Java Perspective : Developing Code



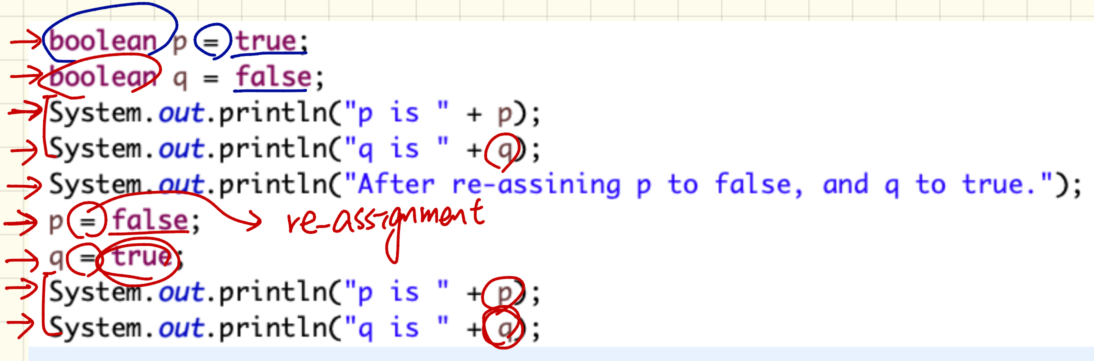


```
public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    System.out.println("Enter the 1st number (which can contain a decimal)");
    double n1 = input.nextDouble(); // if 3 is read, it's treated as 3.0
    input.nextLine(); // necessary

    System.out.println("Enter the 2nd number (which can contain a decimal)");
    double n2 = input.nextDouble();
    input.nextLine(); // necessary

    System.out.println("What's your name:");
    String name = input.nextLine();

    double average = (n1 + n2) / 2;

    System.out.print(name + ", ");
    System.out.print("the numbers you entered were " + n1 + " and " + n2 + ", and ");
    System.out.println("their average is " + average);

    input.close();
}
```



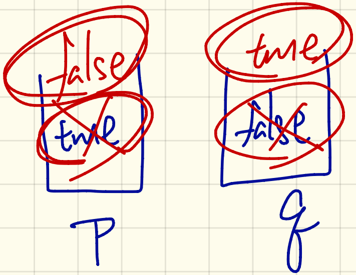


```
Enter 1st num :
14.4 ↲        26.4 ↲
Enter 2nd num :
5.2 ↲         3.2 ↲
What's your name :
Heeyeon ↲     Jihye ↲
              Heeyeon  14.4  5.2  9.8
              Jihye    26.4  3.2  14.8
```



```
boolean p = true;
boolean q = false;
System.out.println("p is " + p);
System.out.println("q is " + q);
System.out.println("After re-assining p to false, and q to true.");
p = false;
q = true;
System.out.println("p is " + p);
System.out.println("q is " + q);
```

re-assignment



p is true
q is false
After . - -
p is false
q is true

Truth Tables of Logical Operators

unary operator

Negation (not)

P	!P
true	false
false	true

1 true,
0 false

0 0 →	0
0 1 →	1
1 0 →	2
1 1 →	3

binary operator

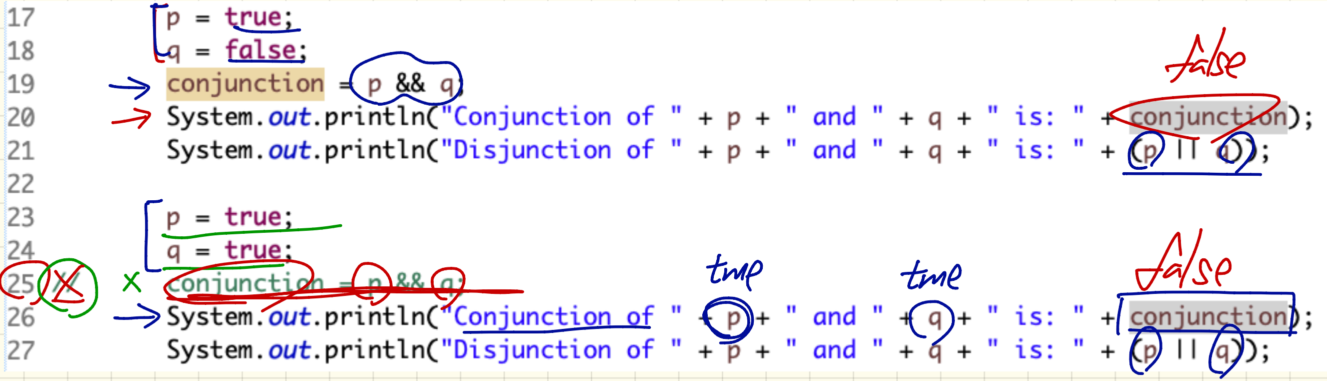
Conjunction (and)

true, false

P	q	P && q
false	false	false
false	true	false
true	false	false
true	true	true

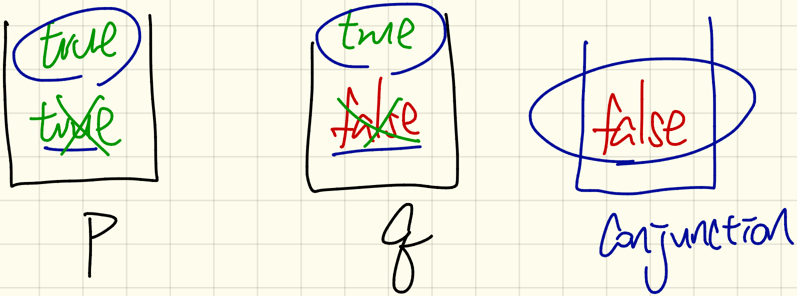
Disjunction (or)

P	q	P || q
false	false	false
false	ture	true
ture	false	true
ture	ture	true



```
        p = true;
        q = false;
        conjunction = p && q;
        System.out.println("Conjunction of " + p + " and " + q + " is: " + conjunction);
        System.out.println("Disjunction of " + p + " and " + q + " is: " + (p || q));

        p = true;
        q = true;
        ~~conjunction = p && q;~~
        System.out.println("Conjunction of " + p + " and " + q + " is: " + conjunction);
        System.out.println("Disjunction of " + p + " and " + q + " is: " + (p || q));
```

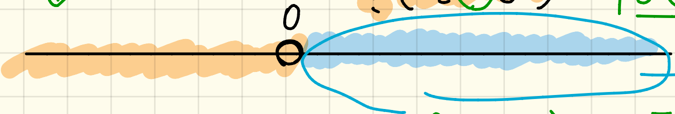
logical error



$i > 0$

$!(i > 0)$

$\boxed{i \leq 0}$ lessOrEqualToZero



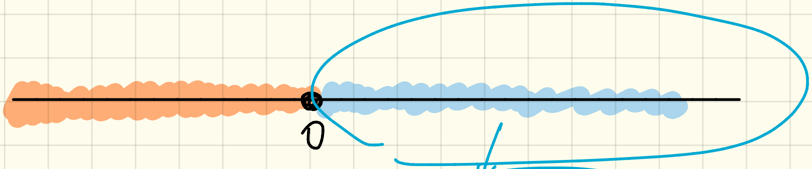
$!(i < 0)$

$i \geq 0$ → $i > 0$

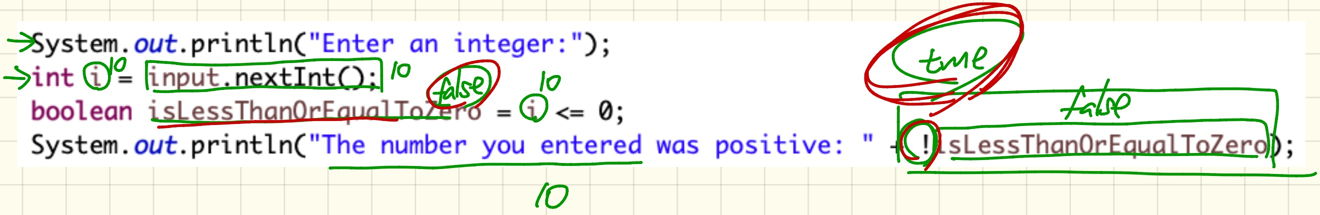
negation

$\boxed{i > 0}$ → $!(i > 0)$ equ. $\boxed{i \leq 0}$

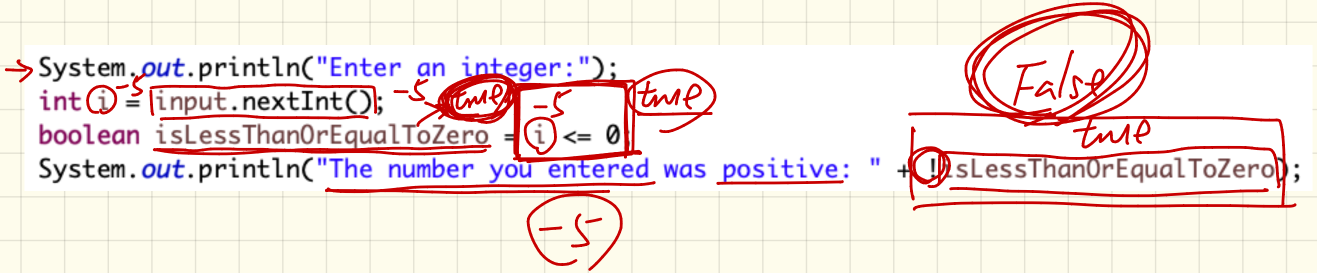
$\boxed{!(i \leq 0)}$ negation



$!(i \leq 0)$

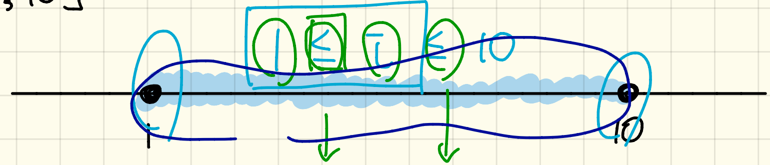


```
System.out.println("Enter an integer:");
int i = input.nextInt();
boolean isLessThanOrEqualToZero = i <= 0;
System.out.println("The number you entered was positive: " + !isLessThanOrEqualToZero);
```



```
System.out.println("Enter an integer:");
int i = input.nextInt();
boolean isLessThanOrEqualToZero = i <= 0;
System.out.println("The number you entered was positive: " + !isLessThanOrEqualToZero);
```

[1, 10] i



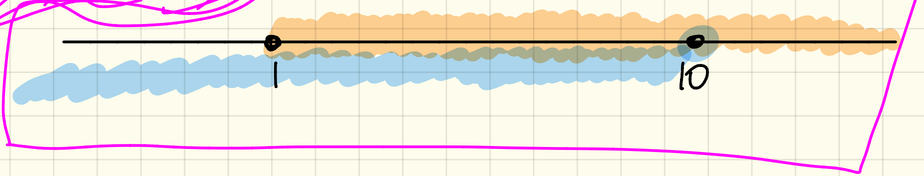
1 <= i && i <= 10   <=   <=

1 <= i <= 10   X

1 <= i && i <= 10

true

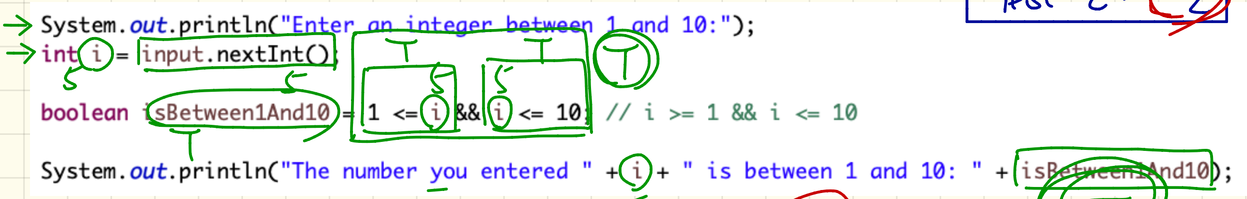
1 <= i || i <= 10



Correct Version

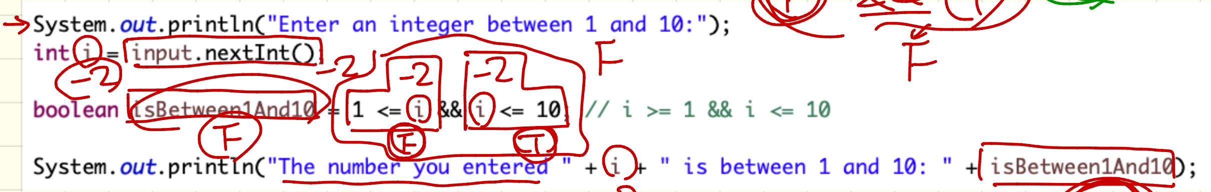
T && T

Test 1: 5
Test 2: -2



```
System.out.println("Enter an integer between 1 and 10:");
int i = input.nextInt();

boolean isBetween1And10 = 1 <= i && i <= 10; // i >= 1 && i <= 10

System.out.println("The number you entered " + i + " is between 1 and 10: " + isBetween1And10);
```

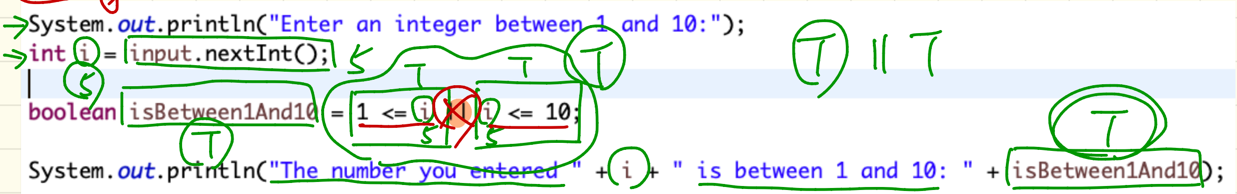
F && T
F



```
System.out.println("Enter an integer between 1 and 10:");
int i = input.nextInt();

boolean isBetween1And10 = 1 <= i && i <= 10; // i >= 1 && i <= 10

System.out.println("The number you entered " + i + " is between 1 and 10: " + isBetween1And10);
```

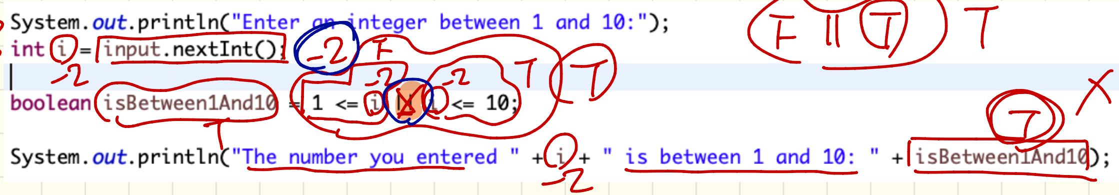
Wrong Version

T || T

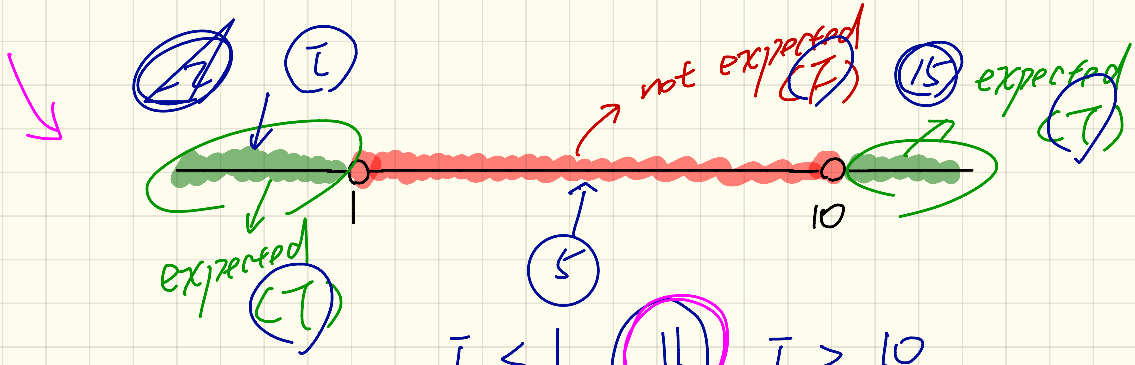


```
System.out.println("Enter an integer between 1 and 10:");
int i = input.nextInt();

boolean isBetween1And10 = 1 <= i || i <= 10;

System.out.println("The number you entered " + i + " is between 1 and 10: " + isBetween1And10);
```

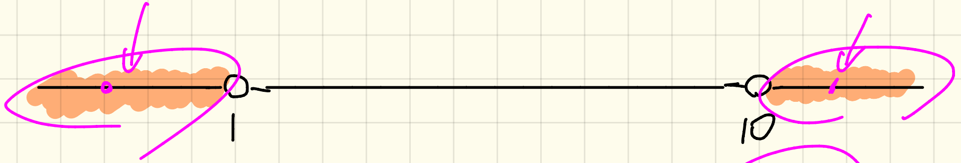
F || T
T



```
System.out.println("Enter an integer between 1 and 10:");
int i = input.nextInt();

boolean isBetween1And10 = 1 <= i || i <= 10;

System.out.println("The number you entered " + i + " is between 1 and 10: " + isBetween1And10);
```



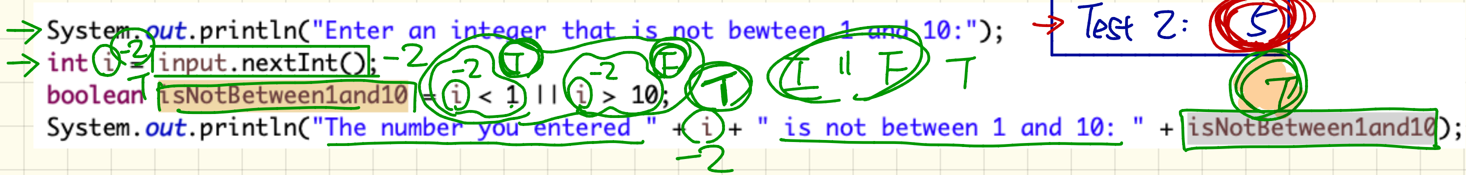
$i < 1 \;||\; i > 10$



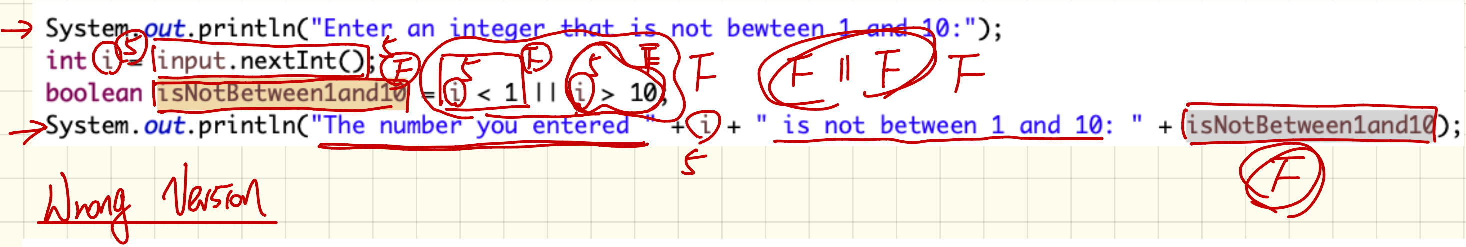
$\boxed{i < 1 \;\&\&\; i > 10}$ false

Correct Version

Test 1: -2
Test 2: 5

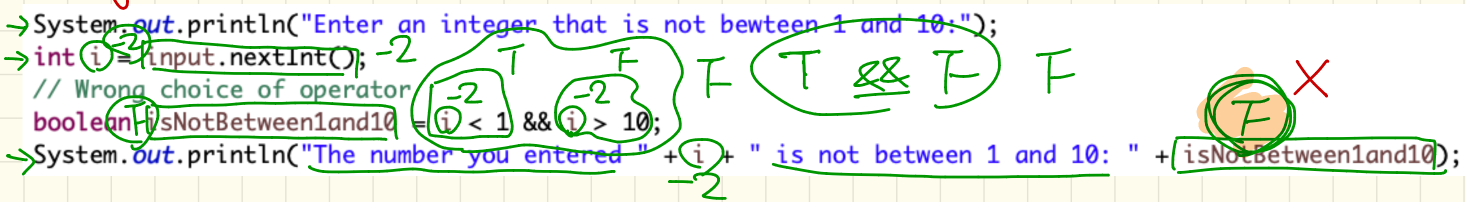


```java
System.out.println("Enter an integer that is not bewteen 1 and 10:");
int i = input.nextInt();
boolean isNotBetween1and10 = i < 1 || i > 10;
System.out.println("The number you entered " + i + " is not between 1 and 10: " + isNotBetween1and10);
```

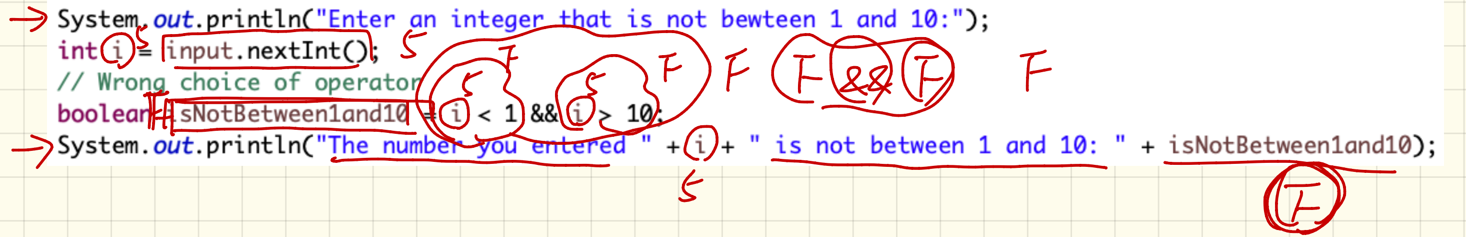


```java
System.out.println("Enter an integer that is not bewteen 1 and 10:");
int i = input.nextInt();
boolean isNotBetween1and10 = i < 1 || i > 10;
System.out.println("The number you entered " + i + " is not between 1 and 10: " + isNotBetween1and10);
```

Wrong Version



```java
System.out.println("Enter an integer that is not bewteen 1 and 10:");
int i = input.nextInt();
// Wrong choice of operator
boolean isNotBetween1and10 = i < 1 && i > 10;
System.out.println("The number you entered " + i + " is not between 1 and 10: " + isNotBetween1and10);
```

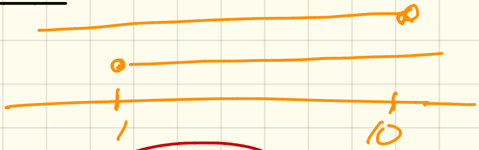


```java
System.out.println("Enter an integer that is not bewteen 1 and 10:");
int i = input.nextInt();
// Wrong choice of operator
boolean isNotBetween1and10 = i < 1 && i > 10;
System.out.println("The number you entered " + i + " is not between 1 and 10: " + isNotBetween1and10);
```

(i)

Problem: Expect Input within an interval

1 <= i && i <= 10

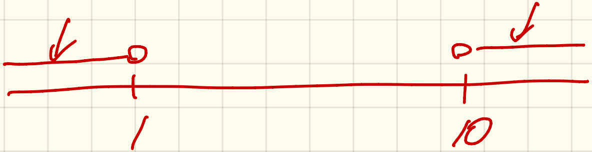


1 <= i || i <= 10 → true

Problem: Expect Input outside an interval

i < 1 || i > 10

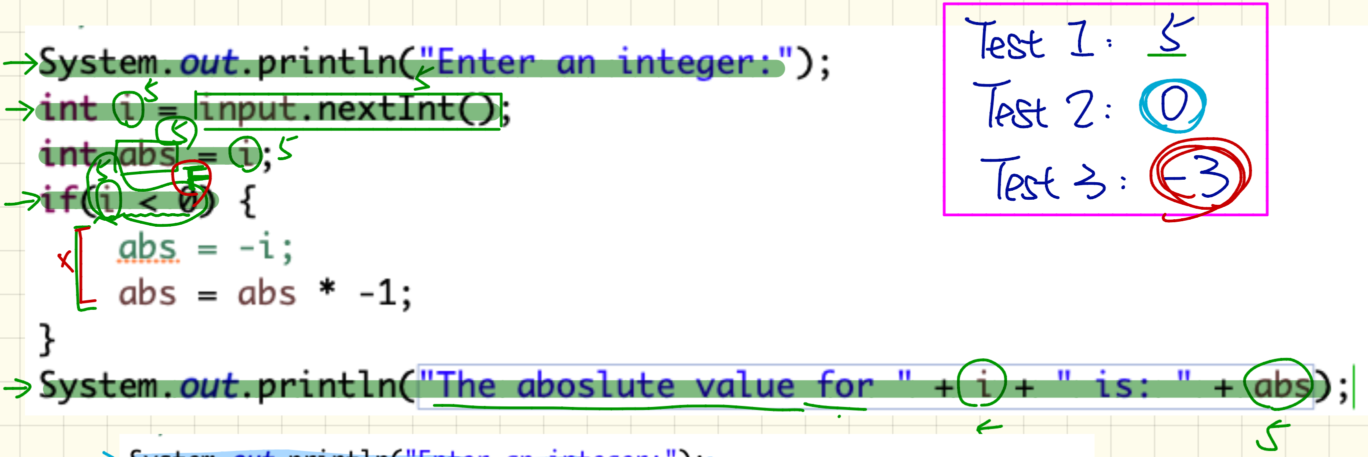
i < 1 && i > 10 → false



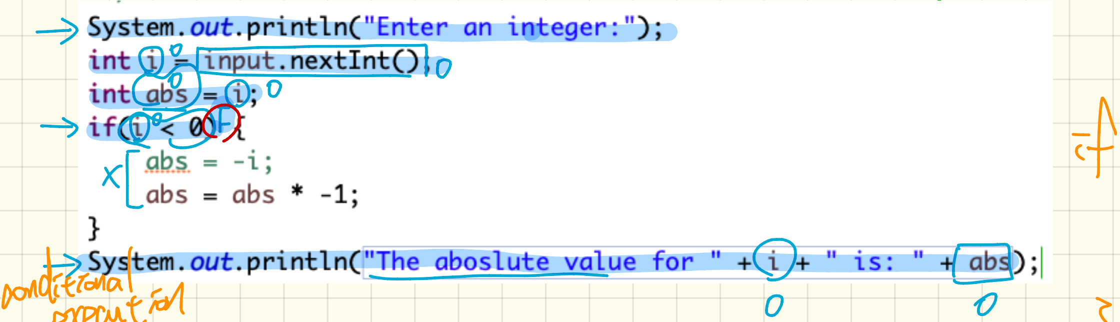
Test 1: 5

Test 2: 0

Test 3: -3

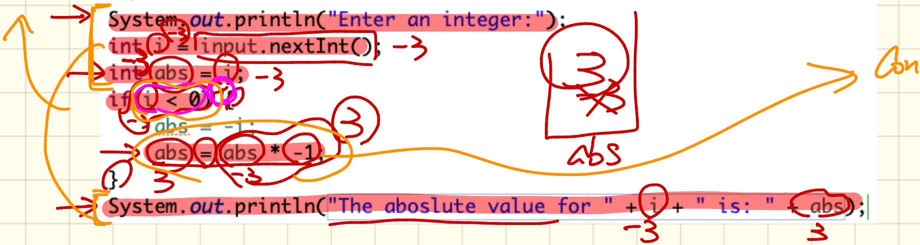


```
System.out.println("Enter an integer:");
int i = input.nextInt();
int abs = i;
if(i < 0) {
    abs = -i;
    abs = abs * -1;
}
System.out.println("The aboslute value for " + i + " is: " + abs);
```



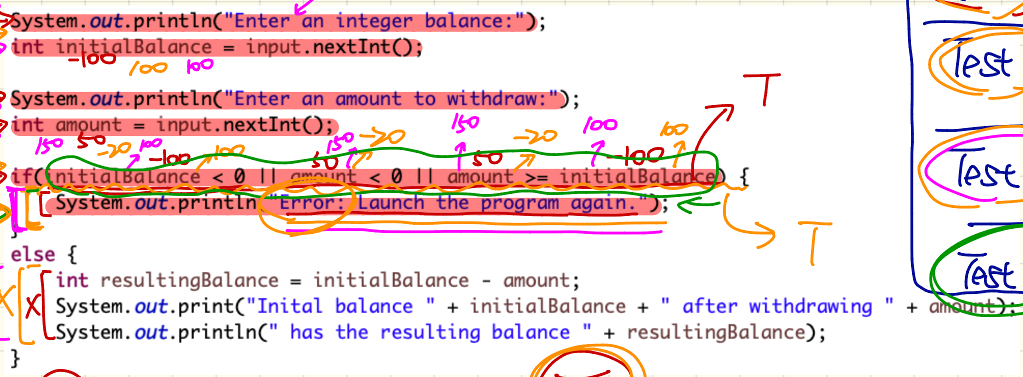
```
System.out.println("Enter an integer:");
int i = input.nextInt();
int abs = i;
if(i < 0) {
    abs = -i;
    abs = abs * -1;
}
System.out.println("The aboslute value for " + i + " is: " + abs);
```

unconditional execution



```
System.out.println("Enter an integer:");
int i = input.nextInt();
int abs = i;
if(i < 0) {
    abs = -i;
    abs = abs * -1;
}
System.out.println("The aboslute value for " + i + " is: " + abs);
```

conditional execution



```
System.out.println("Enter an integer balance:");
int initialBalance = input.nextInt();

System.out.println("Enter an amount to withdraw:");
int amount = input.nextInt();

if(initialBalance < 0 || amount < 0 || amount >= initialBalance) {
  System.out.println("Error: Launch the program again.");
}
else {
  int resultingBalance = initialBalance - amount;
  System.out.print("Inital balance " + initialBalance + " after withdrawing " + amount);
  System.out.println(" has the resulting balance " + resultingBalance);
}
```

-100 100 100

150 50 -20 100 -100 100 50 150 -20 150 50 -20 100 100 100 T

T

Test 1: balance == -100
amount == 50

Test 2: balance == 100
amount == -20

Test 3: balance == 100
amount == 150

Test 4: balance == 100
amount == 44

T || F || T = T

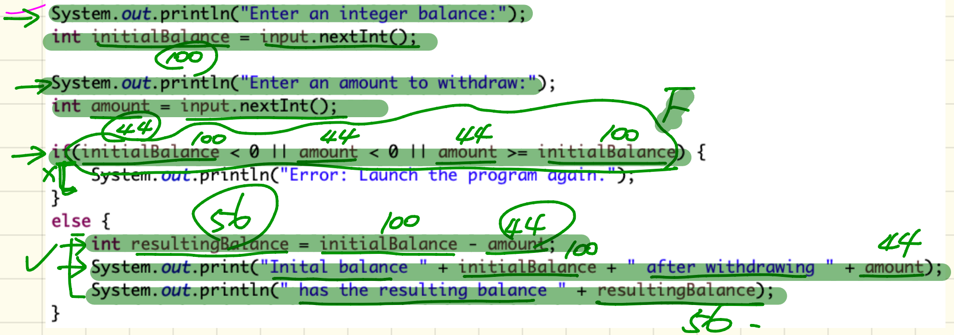
$\frac{100 < 0}{F}$ || $\frac{-20 < 0}{T}$ || $\frac{-20 >= 100}{F}$ T

$\frac{100 < 0}{F}$ || $\frac{150 < 0}{F}$ || $\frac{150 >= 100}{T}$ T

F

am. bal

$\frac{100 < 0}{F}$ || $\frac{44 < 0}{F}$ || $\frac{44 >= 100}{F}$



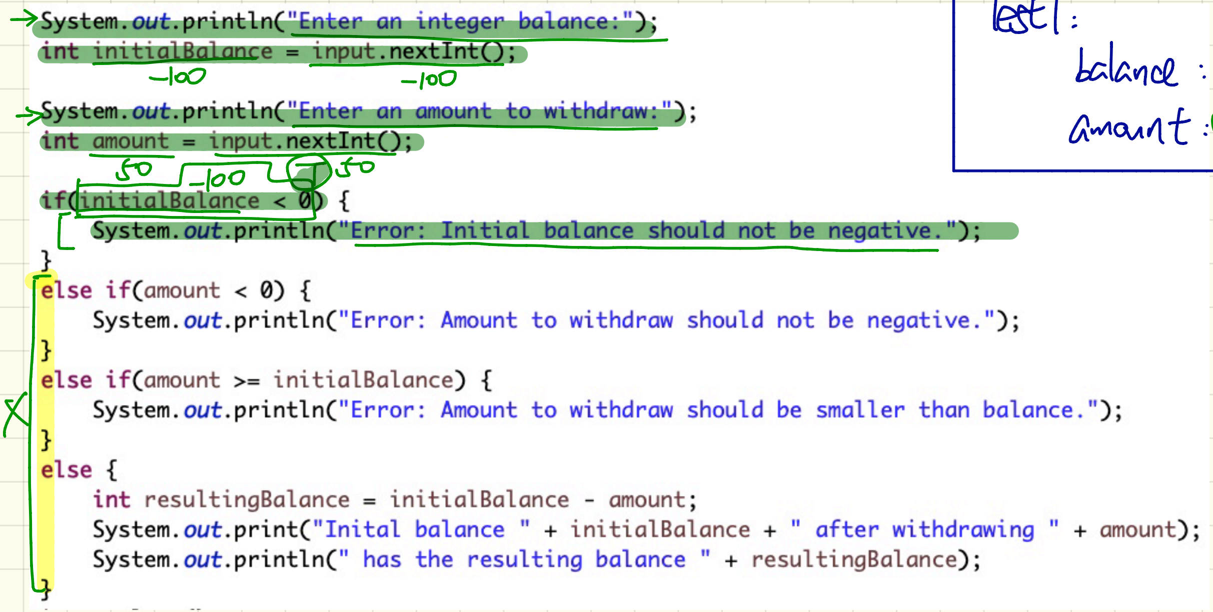
```
System.out.println("Enter an integer balance:");
int initialBalance = input.nextInt();

System.out.println("Enter an amount to withdraw:");
int amount = input.nextInt();

if(initialBalance < 0 || amount < 0 || amount >= initialBalance) {
  System.out.println("Error: Launch the program again.");
}
else {
  int resultingBalance = initialBalance - amount;
  System.out.print("Inital balance " + initialBalance + " after withdrawing " + amount);
  System.out.println(" has the resulting balance " + resultingBalance);
}
```

100

44 100 44 44 100 F

56 100 44 100 44

56



```
System.out.println("Enter an integer balance:");
int initialBalance = input.nextInt();

System.out.println("Enter an amount to withdraw:");
int amount = input.nextInt();

if(initialBalance < 0) {
    System.out.println("Error: Initial balance should not be negative.");
}
else if(amount < 0) {
    System.out.println("Error: Amount to withdraw should not be negative.");
}
else if(amount >= initialBalance) {
    System.out.println("Error: Amount to withdraw should be smaller than balance.");
}
else {
    int resultingBalance = initialBalance - amount;
    System.out.print("Inital balance " + initialBalance + " after withdrawing " + amount);
    System.out.println(" has the resulting balance " + resultingBalance);
}
```

-100 -100

50 -100 T 50

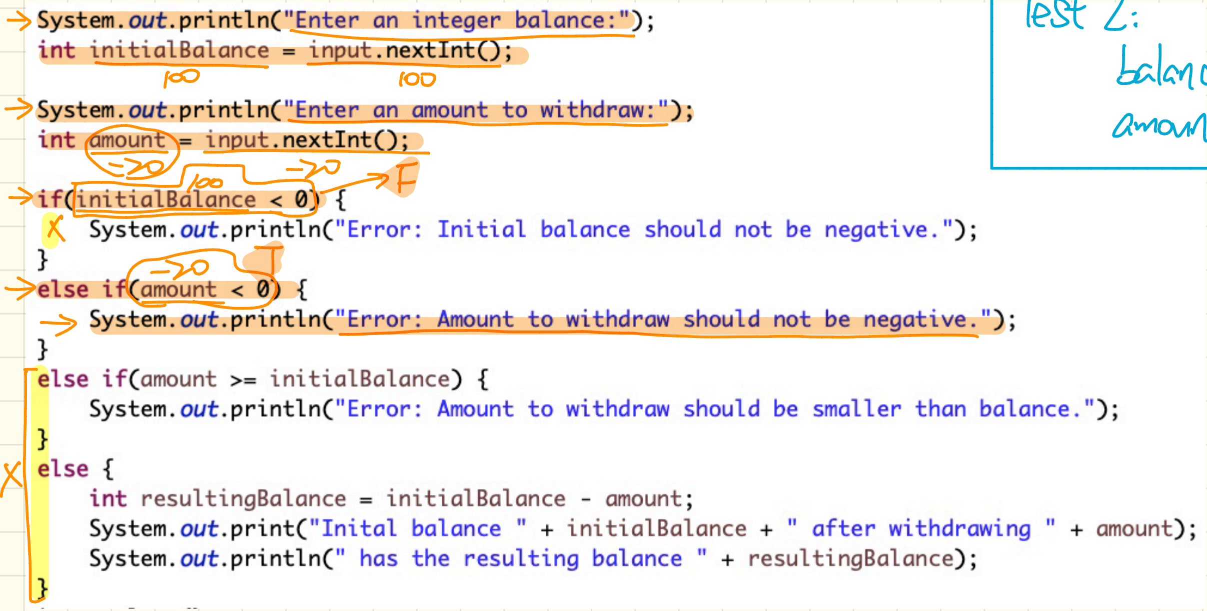
X

Test1:

balance : -100

amount : 50

$-100 < 0$ T



```
System.out.println("Enter an integer balance:");
int initialBalance = input.nextInt();

System.out.println("Enter an amount to withdraw:");
int amount = input.nextInt();

if(initialBalance < 0) {
    System.out.println("Error: Initial balance should not be negative.");
}
else if(amount < 0) {
    System.out.println("Error: Amount to withdraw should not be negative.");
}
else if(amount >= initialBalance) {
    System.out.println("Error: Amount to withdraw should be smaller than balance.");
}
else {
    int resultingBalance = initialBalance - amount;
    System.out.print("Inital balance " + initialBalance + " after withdrawing " + amount);
    System.out.println(" has the resulting balance " + resultingBalance);
}
```

Test 2:
balance : 100
amount : -20

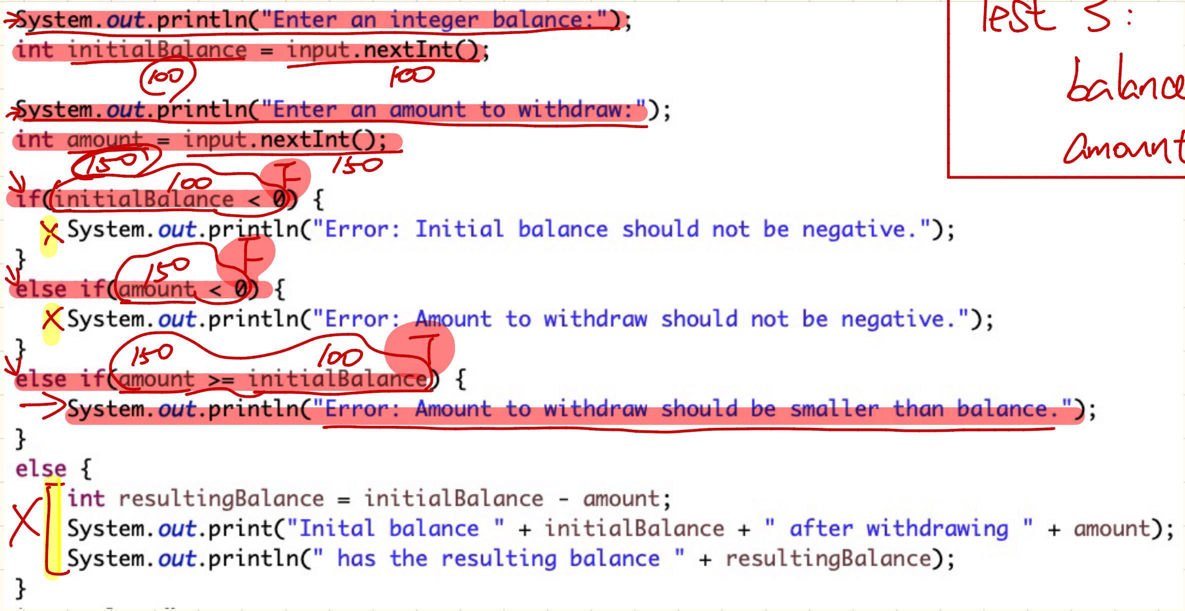
100 < 0 F

-20 < 0 T

Test 3:

balance : 100

amount 150



```
System.out.println("Enter an integer balance:");
int initialBalance = input.nextInt();

System.out.println("Enter an amount to withdraw:");
int amount = input.nextInt();

if(initialBalance < 0) {
   System.out.println("Error: Initial balance should not be negative.");
}
else if(amount < 0) {
   System.out.println("Error: Amount to withdraw should not be negative.");
}
else if(amount >= initialBalance) {
   System.out.println("Error: Amount to withdraw should be smaller than balance.");
}
else {
   int resultingBalance = initialBalance - amount;
   System.out.print("Inital balance " + initialBalance + " after withdrawing " + amount);
   System.out.println(" has the resulting balance " + resultingBalance);
}
```

100 < 0 F

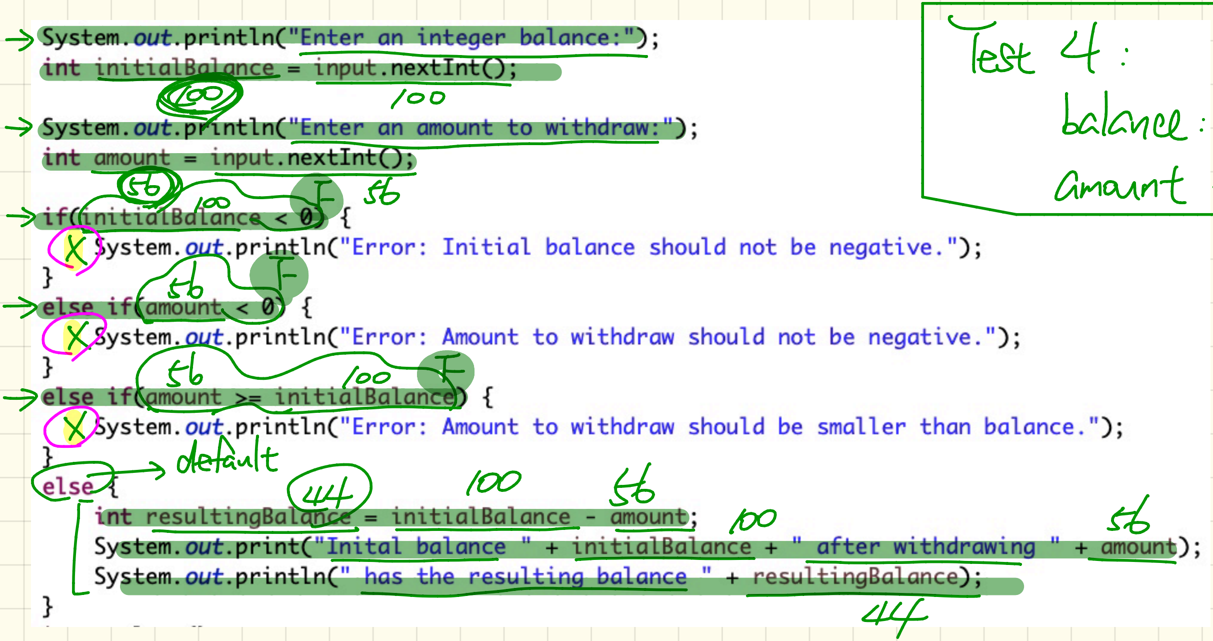
150 < 0 F

150 >= 100 T

Test 4 :

balance : 100

amount : 56



```
System.out.println("Enter an integer balance:");
int initialBalance = input.nextInt();

System.out.println("Enter an amount to withdraw:");
int amount = input.nextInt();

if(initialBalance < 0) {
  System.out.println("Error: Initial balance should not be negative.");
}
else if(amount < 0) {
  System.out.println("Error: Amount to withdraw should not be negative.");
}
else if(amount >= initialBalance) {
  System.out.println("Error: Amount to withdraw should be smaller than balance.");
}
else {
  int resultingBalance = initialBalance - amount;
  System.out.print("Inital balance " + initialBalance + " after withdrawing " + amount);
  System.out.println(" has the resulting balance " + resultingBalance);
}
```

- 100 < 0 F

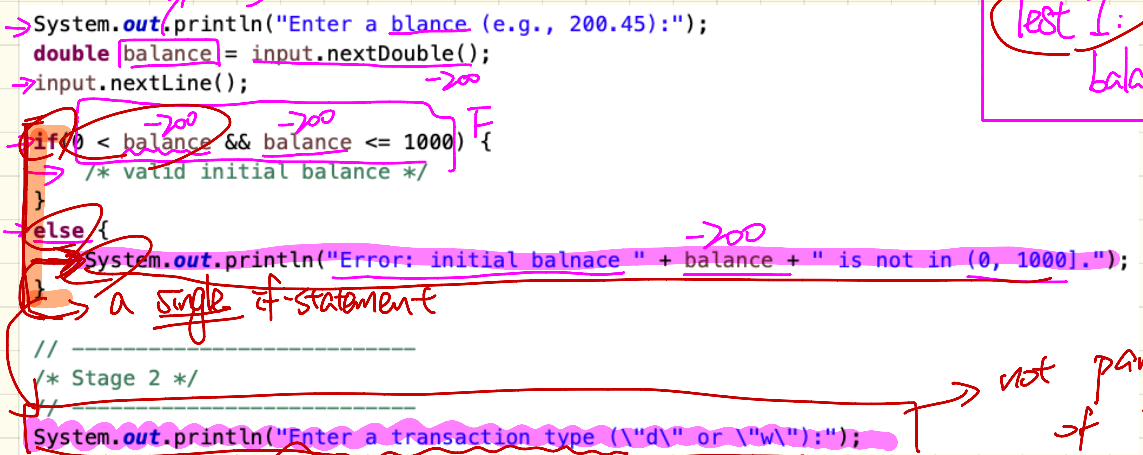
- 56 < 0 F

- 56 >= 100 F

```
if ( . . . ){
  S1
}
else if ( . . ) {
  S2
}
  ⋮
else { . . . }
```

Test 1:

balance : -200



```
System.out.println("Enter a blance (e.g., 200.45):");
double balance = input.nextDouble();
input.nextLine();

if(0 < balance && balance <= 1000) {
    /* valid initial balance */
}
else {
    System.out.println("Error: initial balnace " + balance + " is not in (0, 1000].");
}

// ---------------------------
/* Stage 2 */
// ---------------------------
System.out.println("Enter a transaction type (\"d\" or \"w\"):");
```

-200

-200

-200 -200 F

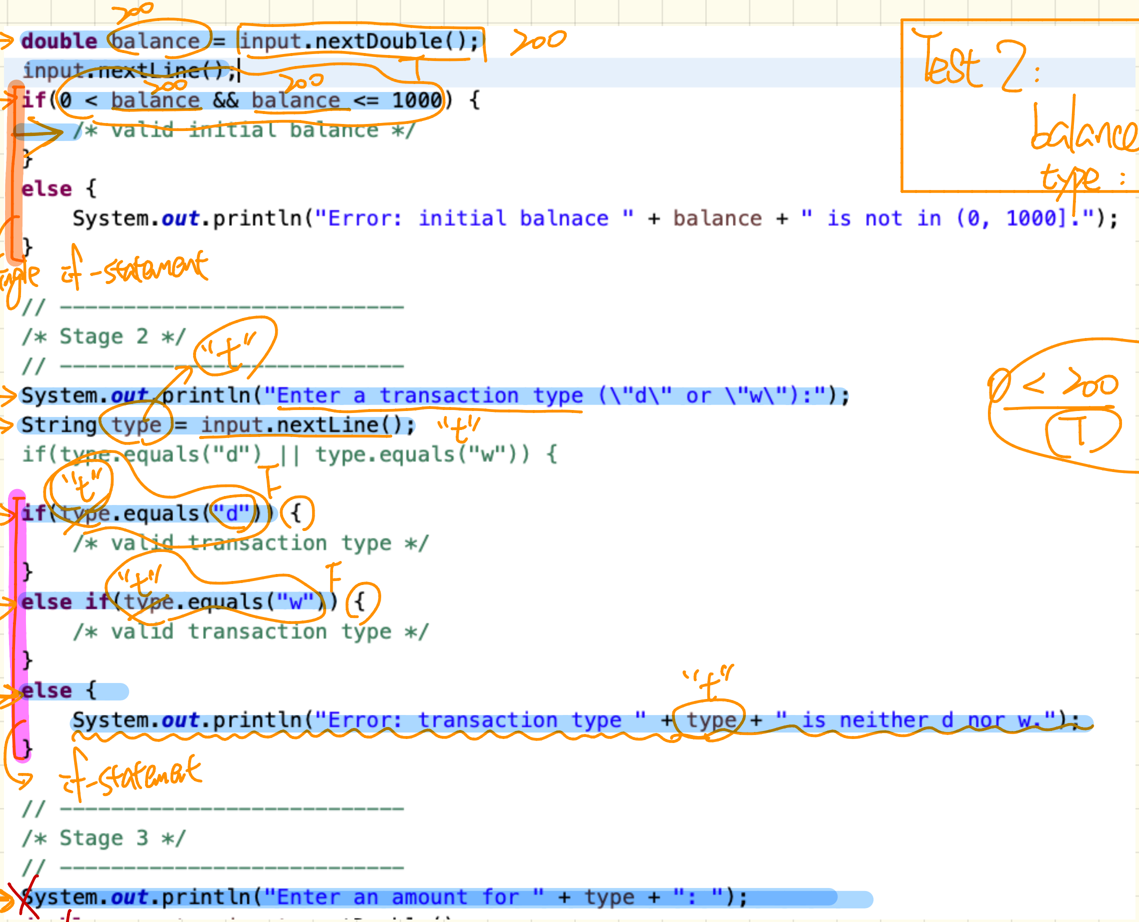
-200

a single if-statement

not part of the above single if-statement.

$\underbrace{0 < -200}_{F}$ && $\underbrace{-200 <= 1000}_{T}$

F



```
double balance = input.nextDouble();
input.nextLine();
if(0 < balance && balance <= 1000) {
    /* valid initial balance */
}
else {
    System.out.println("Error: initial balnace " + balance + " is not in (0, 1000].");
}
// ---------------------------
/* Stage 2 */
// ---------------------------
System.out.println("Enter a transaction type (\"d\" or \"w\"):");
String type = input.nextLine();
if(type.equals("d") || type.equals("w")) {

if(type.equals("d")) {
    /* valid transaction type */
}
else if(type.equals("w")) {
    /* valid transaction type */
}
else {
    System.out.println("Error: transaction type " + type + " is neither d nor w.");
}
// ---------------------------
/* Stage 3 */
// ---------------------------
System.out.println("Enter an amount for " + type + ": ");
```

200

single if-statement

"t"

F

F

"t"

if-statement

logical error

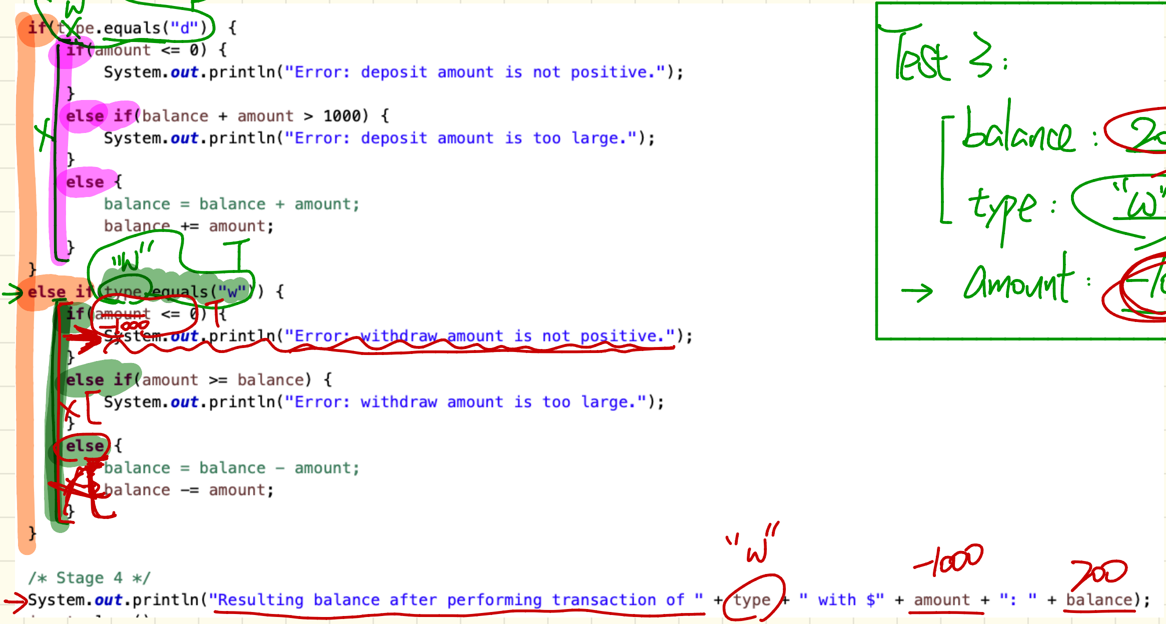
Test 2:

balance: 200 ✓

type: "t"

$\underbrace{0 < 200}_{T}$ && $\underbrace{200 <= 1000}_{T}$

T



```
if(type.equals("d")) {
    if(amount <= 0) {
        System.out.println("Error: deposit amount is not positive.");
    }
    else if(balance + amount > 1000) {
        System.out.println("Error: deposit amount is too large.");
    }
    else {
        balance = balance + amount;
        balance += amount;
    }
}
else if(type.equals("w")) {
    if(amount <= 0) {
        System.out.println("Error: withdraw amount is not positive.");
    }
    else if(amount >= balance) {
        System.out.println("Error: withdraw amount is too large.");
    }
    else {
        balance = balance - amount;
        balance -= amount;
    }
}

/* Stage 4 */
System.out.println("Resulting balance after performing transaction of " + type + " with $" + amount + ": " + balance);
```

Test 3:

balance : 200

type : "w"

→ amount : -1000

"w" "d" F   -1000 <= 0   T

"w" "w" T

Version I

```
prompt user for balance
if ( valid balance ) {

}
else {
    print error
}
prompt user for transaction type
```

Version 2

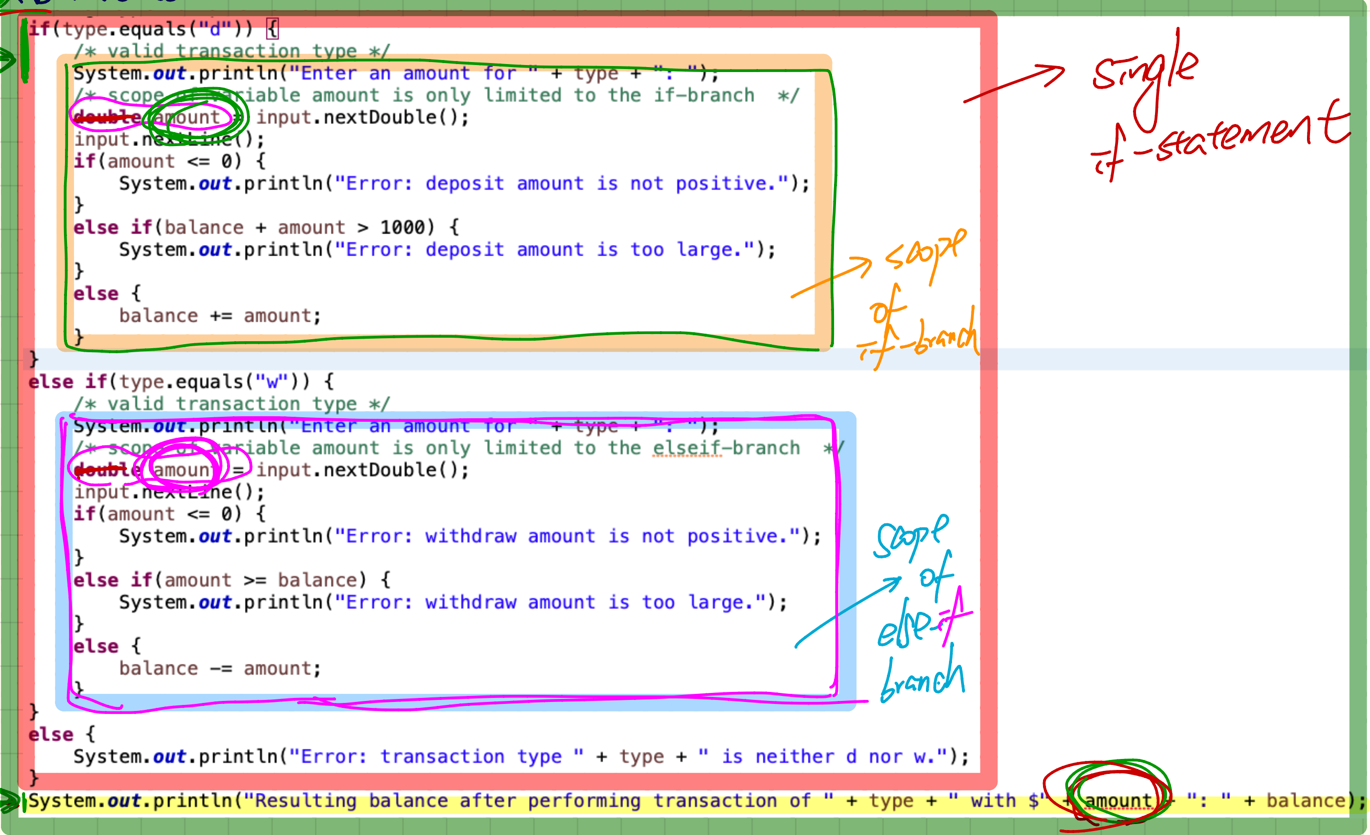
```
prompt user for balance
if ( valid balance ) {
    prompt user for transaction type
}
else {
    print error
}
```

Version 2

```
prompt user for balance
if ( valid balance ) {
    prompt user for transaction type
    if ( valid transaction type ) {
        prompt user for amount
        if ( amount is valid ) { ... }
        else { print error }
    }
    else {
        print error
    }
}
else {
    print error
}
```

main(. —) {

if (balance is valid) {



```
if(type.equals("d")) {
    /* valid transaction type */
    System.out.println("Enter an amount for " + type + ": ");
    /* scope of variable amount is only limited to the if-branch */
    ~~double~~ amount = input.nextDouble();
    input.nextLine();
    if(amount <= 0) {
        System.out.println("Error: deposit amount is not positive.");
    }
    else if(balance + amount > 1000) {
        System.out.println("Error: deposit amount is too large.");
    }
    else {
        balance += amount;
    }
}
else if(type.equals("w")) {
    /* valid transaction type */
    System.out.println("Enter an amount for " + type + ": ");
    /* scope of variable amount is only limited to the elseif-branch */
    ~~double~~ amount = input.nextDouble();
    input.nextLine();
    if(amount <= 0) {
        System.out.println("Error: withdraw amount is not positive.");
    }
    else if(amount >= balance) {
        System.out.println("Error: withdraw amount is too large.");
    }
    else {
        balance -= amount;
    }
}
else {
    System.out.println("Error: transaction type " + type + " is neither d nor w.");
}
System.out.println("Resulting balance after performing transaction of " + type + " with $" + amount + ": " + balance);
```

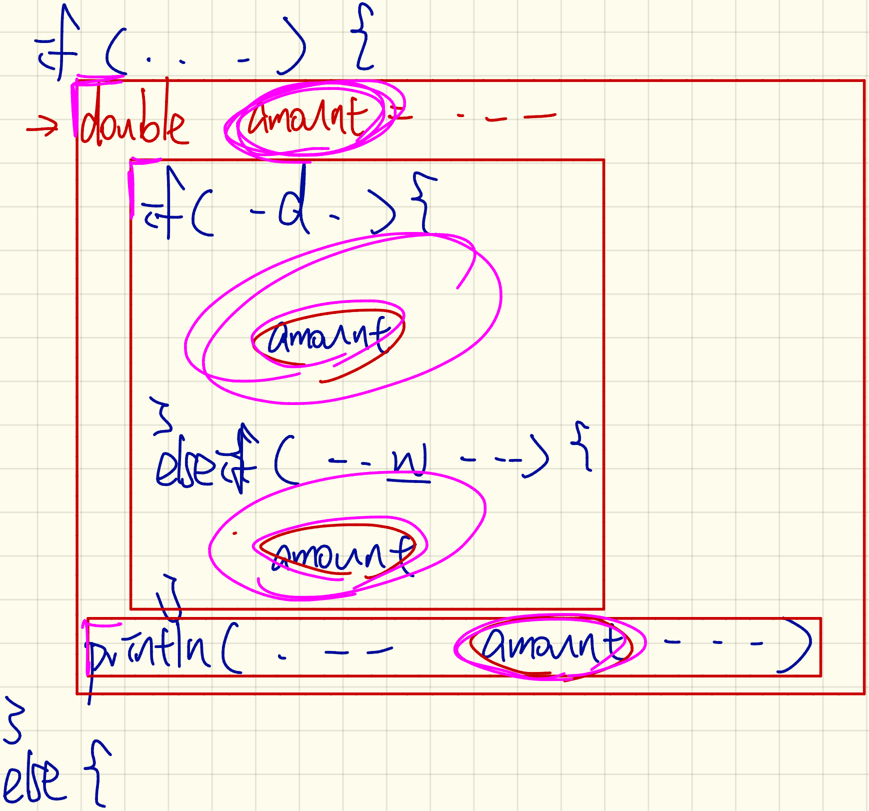
amount

single if-statement

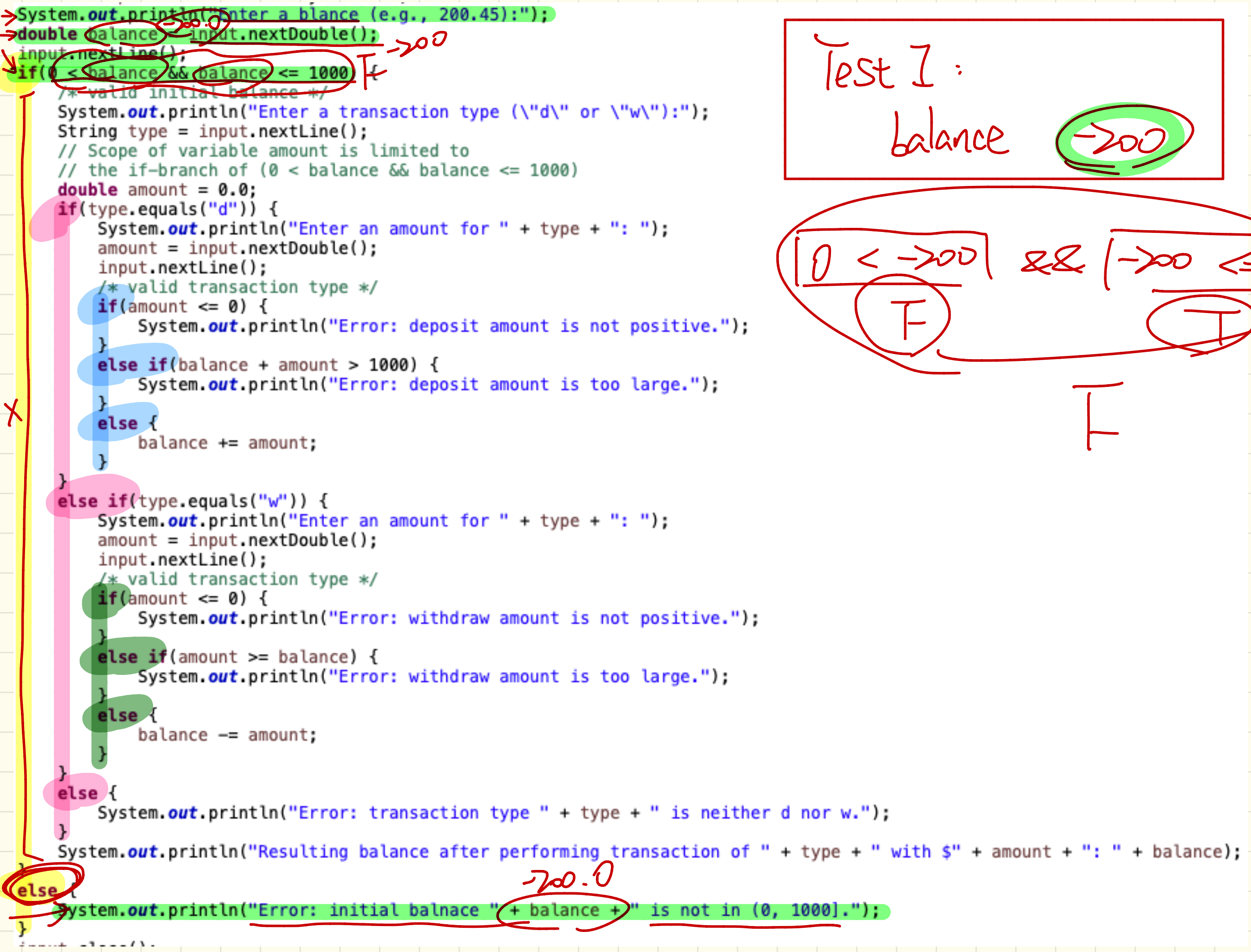
scope of if-branch

scope of else-if branch

}
else { print error }
input.close();
}



```
if ( . . . ) {
→ double amount = . . . .
    if( . d . ) {
        amount
    }
    else if ( . . w . . . ) {
        amount
    }
    println( . . . amount . . . )
}
else {

}
```



```
System.out.println("Enter a blance (e.g., 200.45):");
double balance = input.nextDouble();
input.nextLine();
if(0 < balance && balance <= 1000) {
  /* valid initial balance */
  System.out.println("Enter a transaction type (\"d\" or \"w\"):");
  String type = input.nextLine();
  // Scope of variable amount is limited to
  // the if-branch of (0 < balance && balance <= 1000)
  double amount = 0.0;
  if(type.equals("d")) {
    System.out.println("Enter an amount for " + type + ": ");
    amount = input.nextDouble();
    input.nextLine();
    /* valid transaction type */
    if(amount <= 0) {
      System.out.println("Error: deposit amount is not positive.");
    }
    else if(balance + amount > 1000) {
      System.out.println("Error: deposit amount is too large.");
    }
    else {
      balance += amount;
    }
  }
  else if(type.equals("w")) {
    System.out.println("Enter an amount for " + type + ": ");
    amount = input.nextDouble();
    input.nextLine();
    /* valid transaction type */
    if(amount <= 0) {
      System.out.println("Error: withdraw amount is not positive.");
    }
    else if(amount >= balance) {
      System.out.println("Error: withdraw amount is too large.");
    }
    else {
      balance -= amount;
    }
  }
  else {
    System.out.println("Error: transaction type " + type + " is neither d nor w.");
  }
  System.out.println("Resulting balance after performing transaction of " + type + " with $" + amount + ": " + balance);
}
else {
  System.out.println("Error: initial balnace " + balance + " is not in (0, 1000].");
}
input.close();
```

-200

F -> -200

X

-200.0

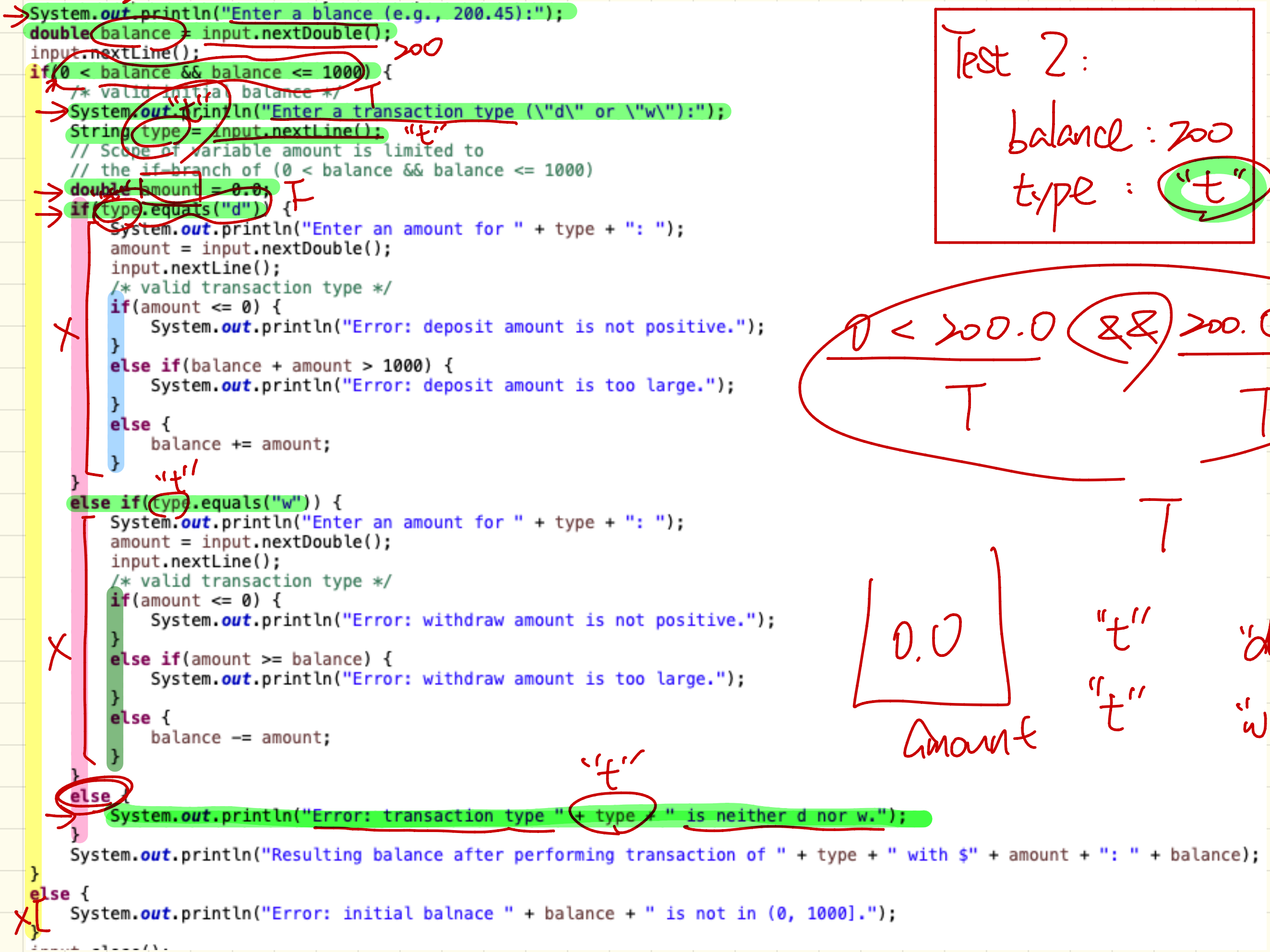
Test 1:

balance -200

0 < -200 && -200 <= 1000

F T

F



```
System.out.println("Enter a blance (e.g., 200.45):");
double balance = input.nextDouble();
input.nextLine();
if(0 < balance && balance <= 1000) {
    /* valid initial balance */
    System.out.println("Enter a transaction type (\"d\" or \"w\"):");
    String type = input.nextLine();
    // Scope of variable amount is limited to
    // the if-branch of (0 < balance && balance <= 1000)
    double amount = 0.0;
    if(type.equals("d")) {
        System.out.println("Enter an amount for " + type + ": ");
        amount = input.nextDouble();
        input.nextLine();
        /* valid transaction type */
        if(amount <= 0) {
            System.out.println("Error: deposit amount is not positive.");
        }
        else if(balance + amount > 1000) {
            System.out.println("Error: deposit amount is too large.");
        }
        else {
            balance += amount;
        }
    }
    else if(type.equals("w")) {
        System.out.println("Enter an amount for " + type + ": ");
        amount = input.nextDouble();
        input.nextLine();
        /* valid transaction type */
        if(amount <= 0) {
            System.out.println("Error: withdraw amount is not positive.");
        }
        else if(amount >= balance) {
            System.out.println("Error: withdraw amount is too large.");
        }
        else {
            balance -= amount;
        }
    }
    else {
        System.out.println("Error: transaction type " + type + " is neither d nor w.");
    }
    System.out.println("Resulting balance after performing transaction of " + type + " with $" + amount + ": " + balance);
}
else {
    System.out.println("Error: initial balnace " + balance + " is not in (0, 1000].");
}
input.close();
```

Test 2:

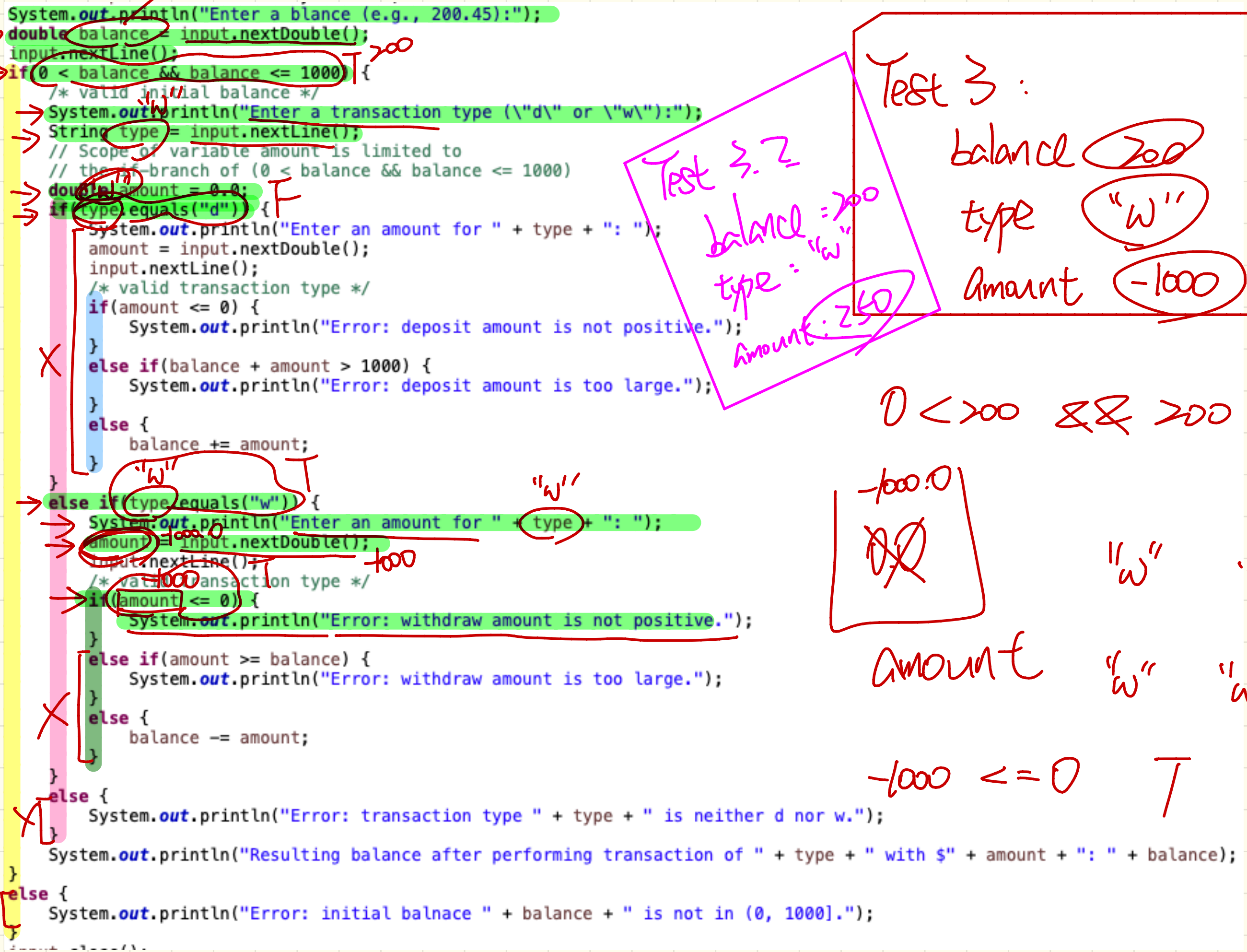
balance : 200

type : "t"

$$\underbrace{0 < 200.0}_{T} \; \&\& \; \underbrace{200.0 <= 1000}_{T}$$

T

Amount			
0.0	"t"	"d"	F
	"t"	"w"	F



```
System.out.println("Enter a blance (e.g., 200.45):");
double balance = input.nextDouble();
input.nextLine();
if(0 < balance && balance <= 1000) {
    /* valid initial balance */
    System.out.println("Enter a transaction type (\"d\" or \"w\"):");
    String type = input.nextLine();
    // Scope of variable amount is limited to
    // the if-branch of (0 < balance && balance <= 1000)
    double amount = 0.0;
    if(type.equals("d")) {
        System.out.println("Enter an amount for " + type + ": ");
        amount = input.nextDouble();
        input.nextLine();
        /* valid transaction type */
        if(amount <= 0) {
            System.out.println("Error: deposit amount is not positive.");
        }
        else if(balance + amount > 1000) {
            System.out.println("Error: deposit amount is too large.");
        }
        else {
            balance += amount;
        }
    }
    else if(type.equals("w")) {
        System.out.println("Enter an amount for " + type + ": ");
        amount = input.nextDouble();
        input.nextLine();
        /* valid transaction type */
        if(amount <= 0) {
            System.out.println("Error: withdraw amount is not positive.");
        }
        else if(amount >= balance) {
            System.out.println("Error: withdraw amount is too large.");
        }
        else {
            balance -= amount;
        }
    }
    else {
        System.out.println("Error: transaction type " + type + " is neither d nor w.");
    }
    System.out.println("Resulting balance after performing transaction of " + type + " with $" + amount + ": " + balance);
}
else {
    System.out.println("Error: initial balnace " + balance + " is not in (0, 1000].");
}
input.close();
```

200.0

200 T

"w"

"w" F

X

"w" T

"w"

-1000.0

-1000

-1000

X

X

X

Test 3.2

balance : 200

type : "w"

amount : 250

Test 3 :

balance 200

type "w"

amount -1000

0 < 200 && 200 <= 1000

-1000.0

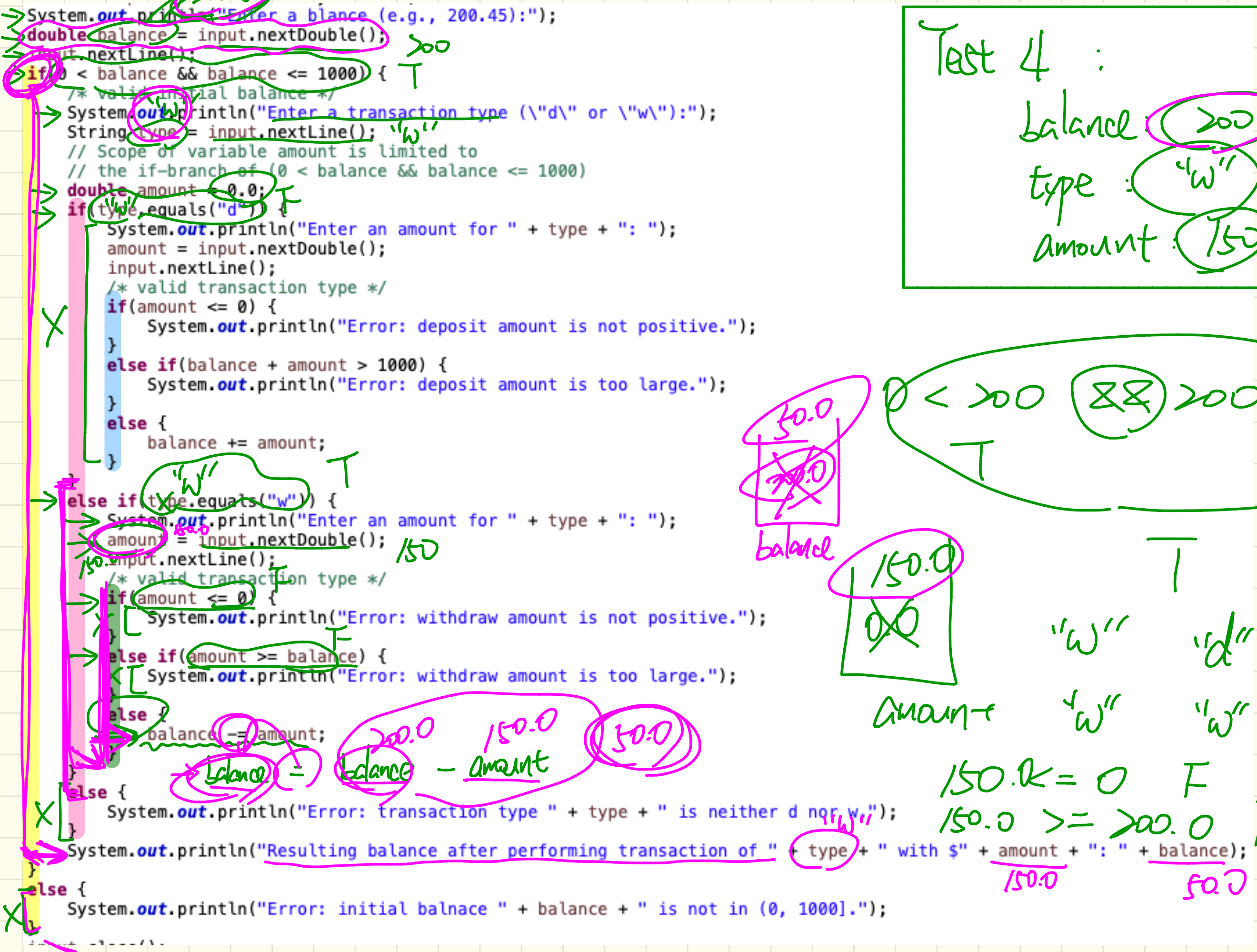
~~0.0~~

"w" "d" F

amount

"w" "w" T

-1000 <= 0 T



```
System.out.println("Enter a blance (e.g., 200.45):");
double balance = input.nextDouble();
input.nextLine();
if(0 < balance && balance <= 1000) {
    /* valid initial balance */
    System.out.println("Enter a transaction type (\"d\" or \"w\"):");
    String type = input.nextLine();
    // Scope of variable amount is limited to
    // the if-branch of (0 < balance && balance <= 1000)
    double amount = 0.0;
    if(type.equals("d")) {
        System.out.println("Enter an amount for " + type + ": ");
        amount = input.nextDouble();
        input.nextLine();
        /* valid transaction type */
        if(amount <= 0) {
            System.out.println("Error: deposit amount is not positive.");
        }
        else if(balance + amount > 1000) {
            System.out.println("Error: deposit amount is too large.");
        }
        else {
            balance += amount;
        }
    }
    else if(type.equals("w")) {
        System.out.println("Enter an amount for " + type + ": ");
        amount = input.nextDouble();
        input.nextLine();
        /* valid transaction type */
        if(amount <= 0) {
            System.out.println("Error: withdraw amount is not positive.");
        }
        else if(amount >= balance) {
            System.out.println("Error: withdraw amount is too large.");
        }
        else {
            balance -= amount;
        }
    }
    else {
        System.out.println("Error: transaction type " + type + " is neither d nor w.");
    }
    System.out.println("Resulting balance after performing transaction of " + type + " with $" + amount + ": " + balance);
}
else {
    System.out.println("Error: initial balnace " + balance + " is not in (0, 1000].");
}
input.close();
```

Test 4 :

balance : 200

type : "w"

amount : 150

200.0

200

"w"

0 < 200 && 200 <= 1000
T T
T

balance: ~~200.0~~ 50.0

amount: ~~0.0~~ 150.0

"w" "d" F

"w" "w" T

150.0 <= 0 F

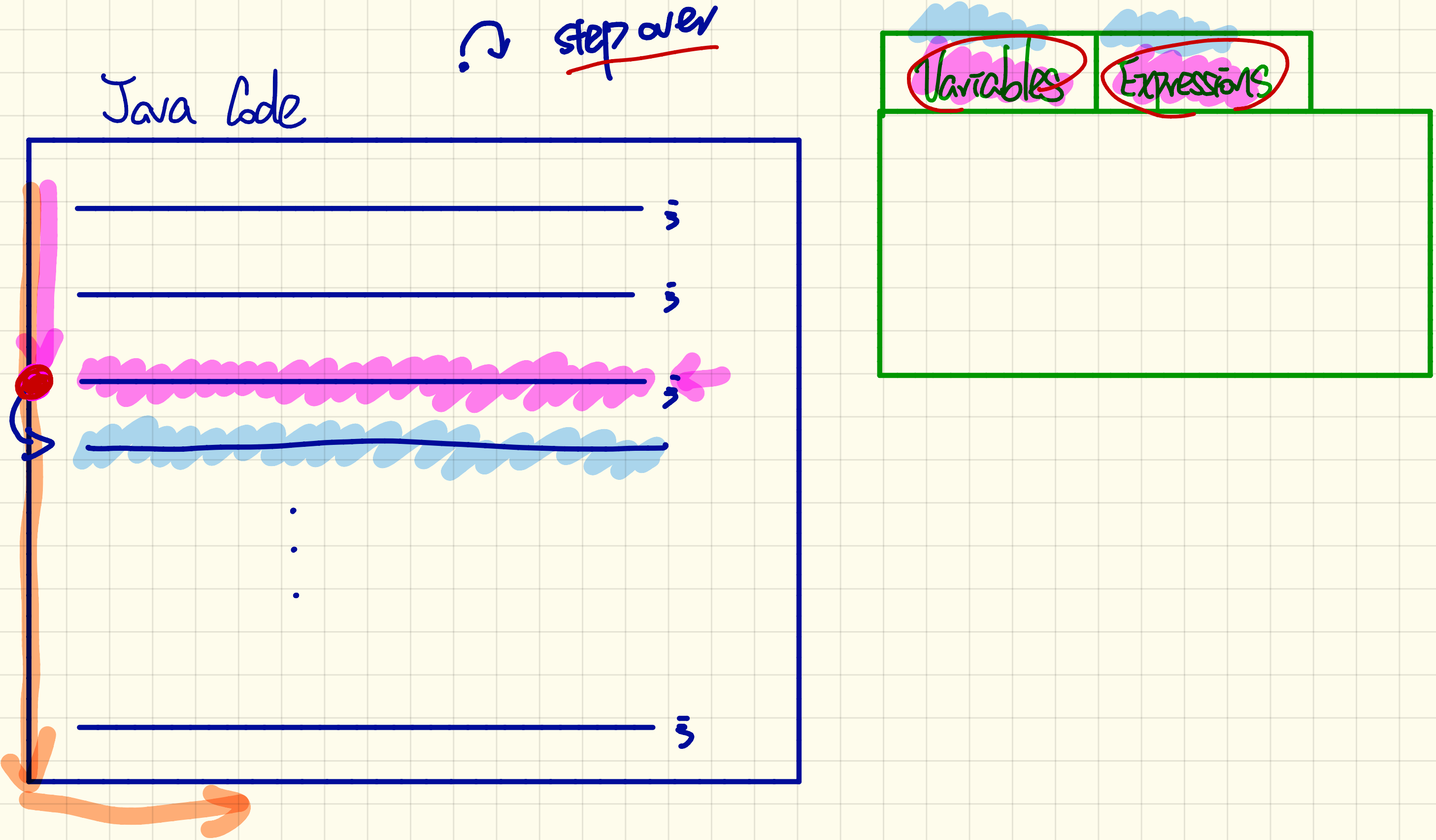
150.0 >= 200.0 F

balance = balance - amount

200.0 150.0 50.0

"w" 150.0 50.0

Ideas of Breakpoints & Debugger

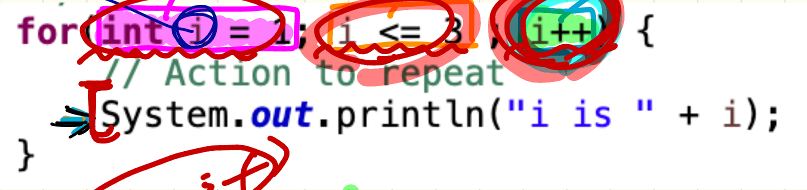


loop counter

initialization (once only)

stay condition

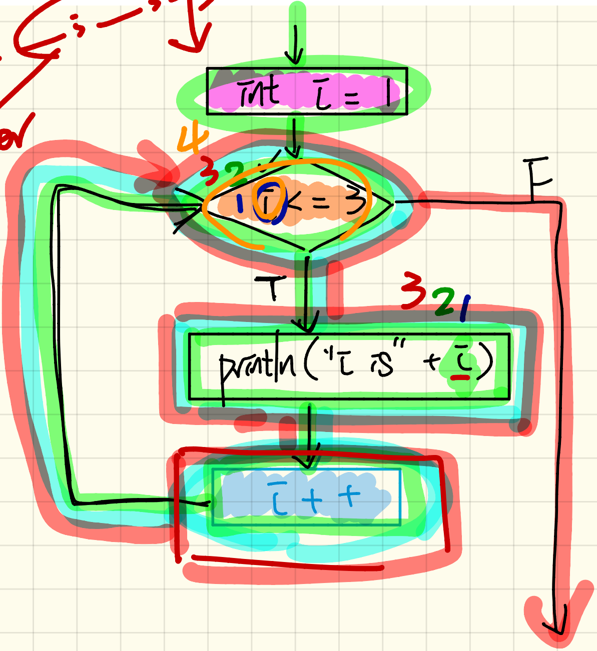
progress of loop (executed at the end of each iteration)



```
for(int i = 1; i <= 3; i++) {
    // Action to repeat
    System.out.println("i is " + i);
}
```

for(; — ;)

header

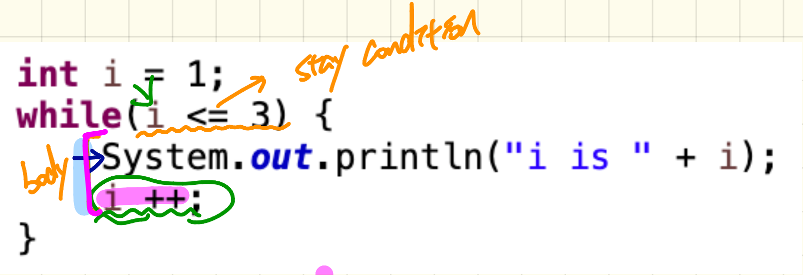


stay condition

i	i<=3	
1	1<=3	T
2	2<=3	T
3	3<=3	T
4	4<=3	F

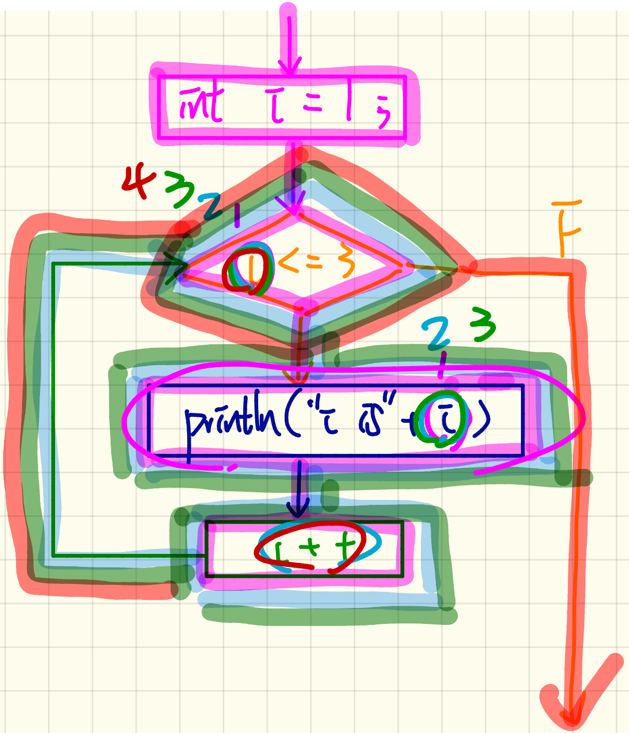
execute the body of the loop

exit from loop

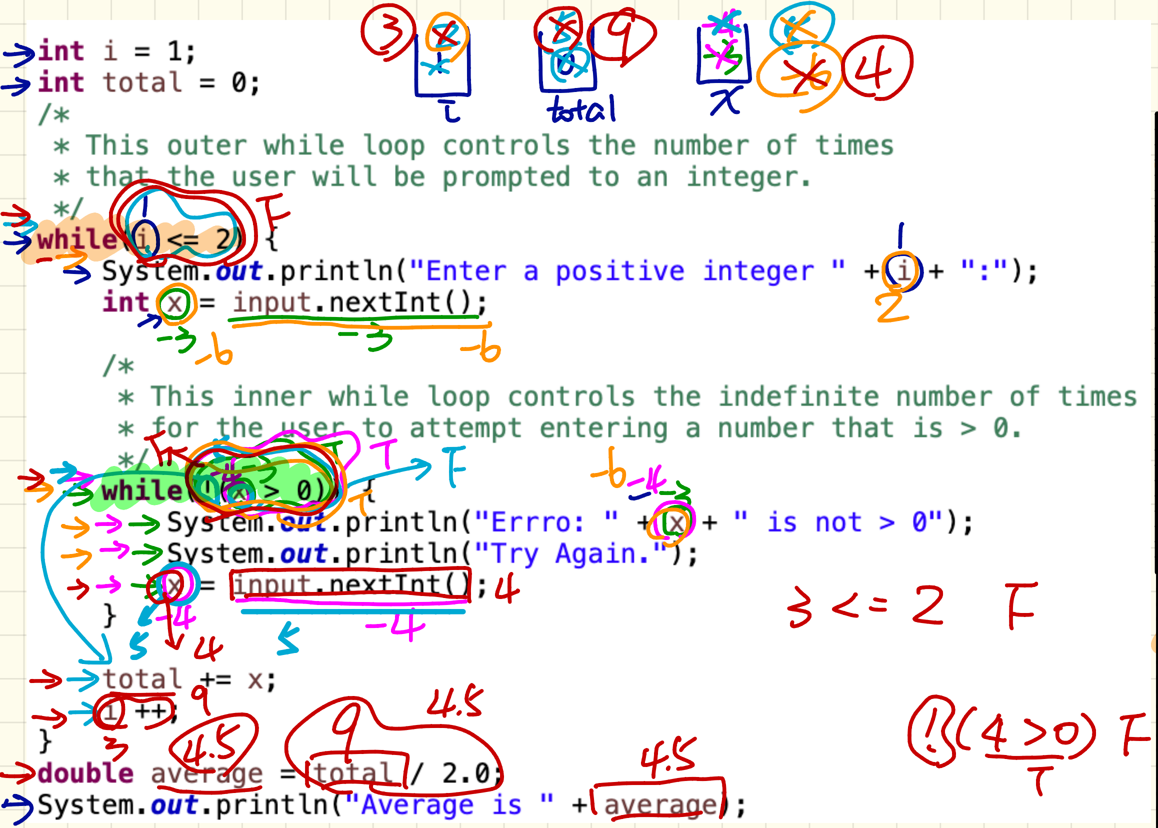


```
int i = 1;
while(i <= 3) {
  System.out.println("i is " + i);
  i ++;
}
```

- i is 1
- i is 2
- i is 3



i	i<=3	
1	1 <= 3	T
2	2<=3	T
3	3 <= 3	T
4	4<=3	F



```
int i = 1;
int total = 0;
/*
 * This outer while loop controls the number of times
 * that the user will be prompted to an integer.
 */
while(i <= 2) {
    System.out.println("Enter a positive integer " + i + ":");
    int x = input.nextInt();
    /*
     * This inner while loop controls the indefinite number of times
     * for the user to attempt entering a number that is > 0.
     */
    while(!(x > 0)) {
        System.out.println("Erro: " + x + " is not > 0");
        System.out.println("Try Again.");
        x = input.nextInt();
    }
    total += x;
    i ++;
}
double average = total / 2.0;
System.out.println("Average is " + average);
```

$1 <= 2$ T

$!(\underbrace{-3 > 0}_{F})$ T

$!(\underbrace{-4 > 0}_{F})$ T

$!(\underbrace{5 > 0}_{T})$ F

$2 <= 2$ T

$!(\underbrace{-6 > 0}_{F})$ T

$!(\underbrace{4 > 0}_{T})$ F

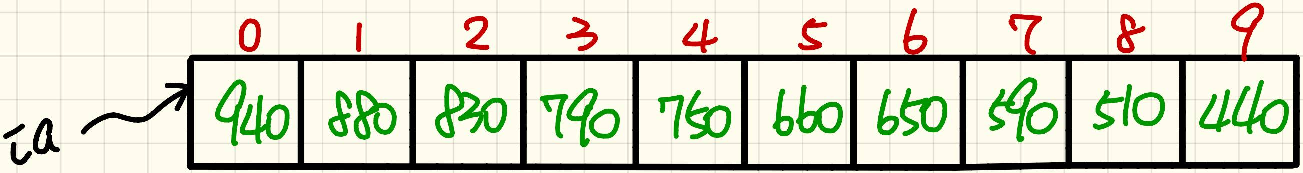
$3 <= 2$ F

Console

```
Enter integer 1:
-3
Error: -3 is not >0
Try again
-4
Error: -4 is not >0
Try again
5
Enter integer 2:
-6
Error: -6 is not >0
Try again
4
Average is 4.5
```

Array of Integers (1)

No pattern on stored values



ia

0	1	2	3	4	5	6	7	8	9
940	880	830	790	750	660	650	590	510	440

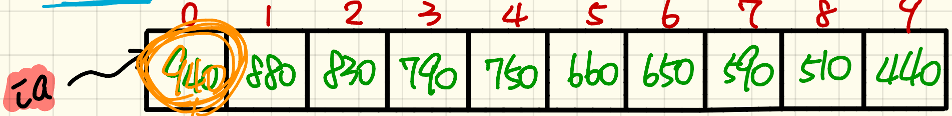
Declaration and Initialization : Approach 1 (Initializer)

Declaration and Initialization : Approach 2 (Assignments)

Indexing of an Array

int[] ia = { ... };

ia[0] ia[1] Nth element ia[N-1]



int 4 bytes

2nd element:

ia[1]

↔ 0x7c1 + 4*1 = 0x7c5

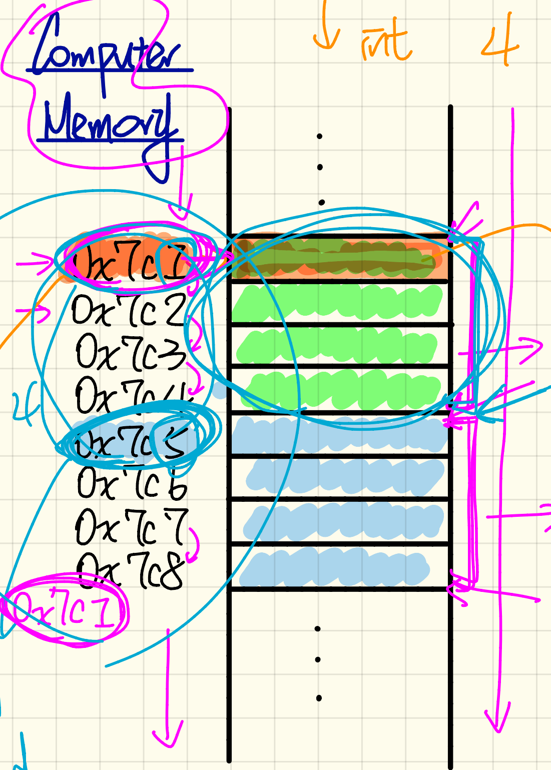
start of storing contents of ia

1st element:

ia[0]

starting address of array

# of units of offsets



store 1st element (4 bytes int)

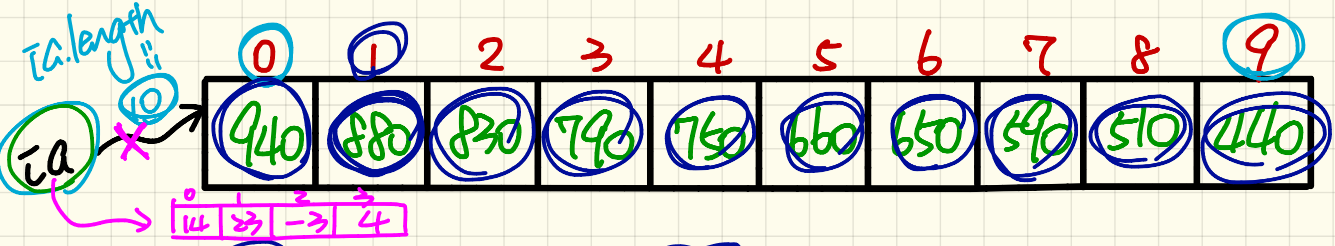
store 2nd element (4 bytes int)

unit of storage is 4 bytes

starting address of 1st element

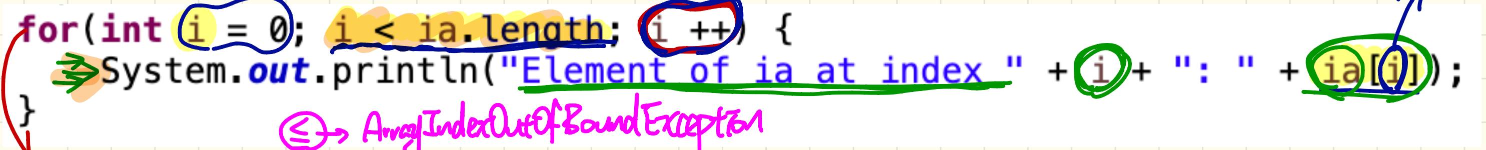
0x7c1 + 4*0 = 0x7c1

starting address of 2nd element



0	1	2	3	4	5	6	7	8	9
940	880	830	790	750	660	650	590	510	440

0	1	2	3
14	23	-3	4



```
for(int i = 0; i < ia.length; i ++) {
  System.out.println("Element of ia at index " + i + ": " + ia[i]);
}
```

≤ → ArrayIndexOutOfBoundException

i	i < ia.length (10)		ia[i]		iteration
0	0 < 10	T	ia[0]	940	1
1	1 < 10	T	ia[1]	880	2
2	2 < 10	T	ia [2]	830	3
3	3 < 10	T	ia [3]	790	4
4	4 < 10	T	ia [4]	750	5
5	5 < 10	T	ia [5]	660	6
6	6 < 10	T	ia [6]	650	7
7	7 < 10	T	ia [7]	590	8
8	8 < 10	T	ia [8]	510	9
9	9 < 10	T	ia [9]	440	10
10	10 < 10	F			

last iteration

Array of Integers (2)

there is pattern on stored values

ia →

0	1	2	3	4
7	10	13	16	19

Declaration and Initialization : Approach 3 (Loops)

(3.1) Loop counter i denotes each stored value

a

↓
value = 7 + term * 3

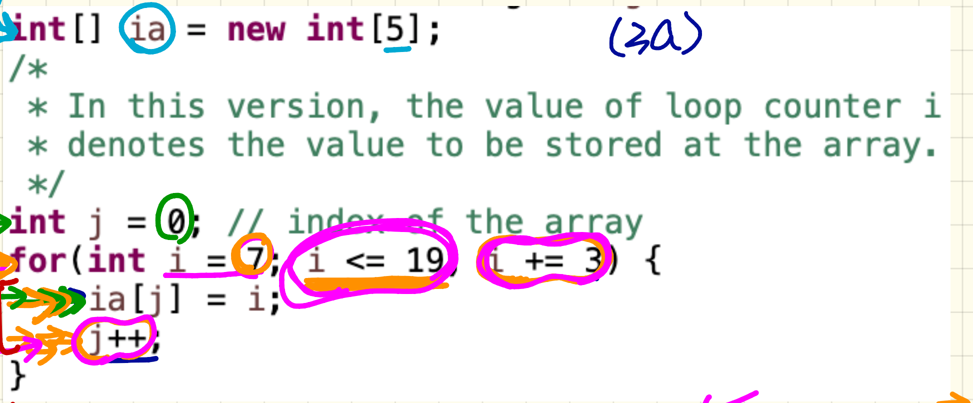
(3.2) Loop counter denotes ith term in the arithmetic seq.

b, c

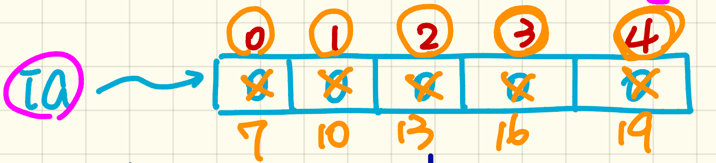
	→ term	formula	value
0	1	7 + 0 * 3	7
1	2	7 + 1 * 3	10
2	3	7 + 2 * 3	13
3	4	7 + 3 * 3	16
4	5	7 + 4 * 3	19

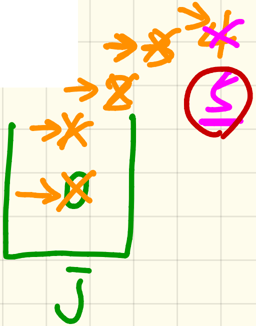
↓
value = 7 + (term - 1) * 3

(3a)

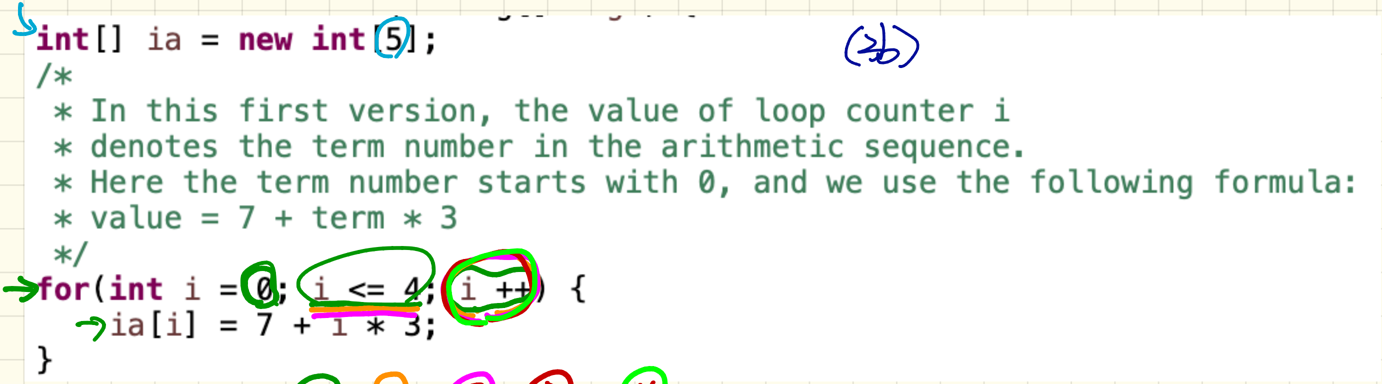


```
int[] ia = new int[5];
/*
 * In this version, the value of loop counter i
 * denotes the value to be stored at the array.
 */
int j = 0; // index of the array
for(int i = 7; i <= 19; i += 3) {
  ia[j] = i;
  j++;
}
```





i	i <= 19	ia[j] = i
7	7 <= 19 T	ia[0] = 7
10	10 <= 19 T	ia[1] = 10
13	13 <= 19 T	ia[2] = 13
16	16 <= 19 T	ia[3] = 16
19	19 <= 19 T	ia[4] = 19
22	22 <= 19 F	



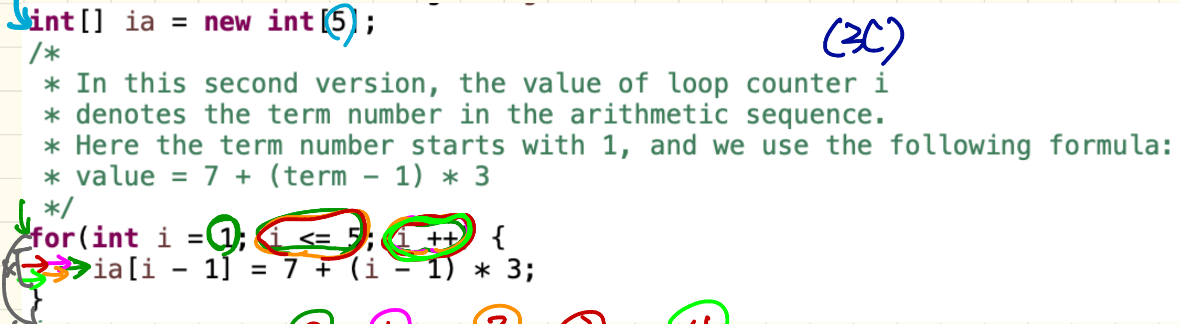
(2b)

```
int[] ia = new int[5];
/*
 * In this first version, the value of loop counter i
 * denotes the term number in the arithmetic sequence.
 * Here the term number starts with 0, and we use the following formula:
 * value = 7 + term * 3
 */
for(int i = 0; i <= 4; i ++) {
   ia[i] = 7 + i * 3;
}
```

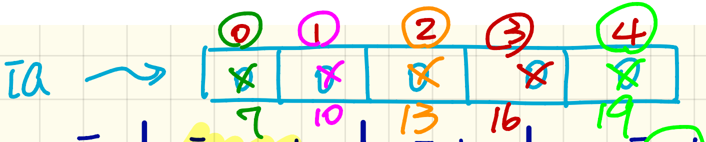
ia	0	1	2	3	4
	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~
	7	10	13	16	19

i	i <= 4		ia[i] = 7 + i * 3
0	0 <= 4	T	ia[0] = 7 + 0 * 3 = 7
1	1 <= 4	T	ia[1] = 7 + 1 * 3 = 10
2	2 <= 4	T	ia[2] = 7 + 2 * 3 = 13
3	3 <= 4	T	ia[3] = 7 + 3 * 3 = 16
4	4 <= 4	T	ia[4] = 7 + 4 * 3 = 19
5	5 <= 4	F	

(3C)



```
int[] ia = new int[5];
/*
 * In this second version, the value of loop counter i
 * denotes the term number in the arithmetic sequence.
 * Here the term number starts with 1, and we use the following formula:
 * value = 7 + (term - 1) * 3
 */
for(int i = 1; i <= 5; i ++) {
  ia[i - 1] = 7 + (i - 1) * 3;
}
```



i	i <= 5	i-1	ia[i-1] = 7 + (i-1) * 3	
1	1 <= 5 T	0	ia[0] = 7 + 0 * 3	7
2	2 <= 5 T	1	ia[1] = 7 + 1 * 3	10
3	3 <= 5 T	2	ia[2] = 7 + 2 * 3	13
4	4 <= 5 T	3	ia[3] = 7 + 3 * 3	16
5	5 <= 5 T	4	ia[4] = 7 + 4 * 3	19
6	6 <= 5 F	—	—	

Computational Problems:

Given an array, determine whether or not:

1. all elements satisfy a property

   allPos true

   &&

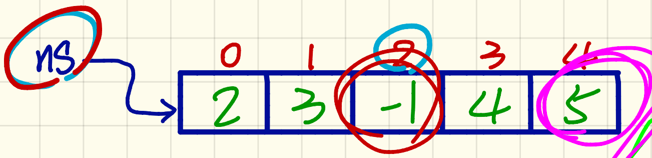
   empty array

2. some element satisfies a property

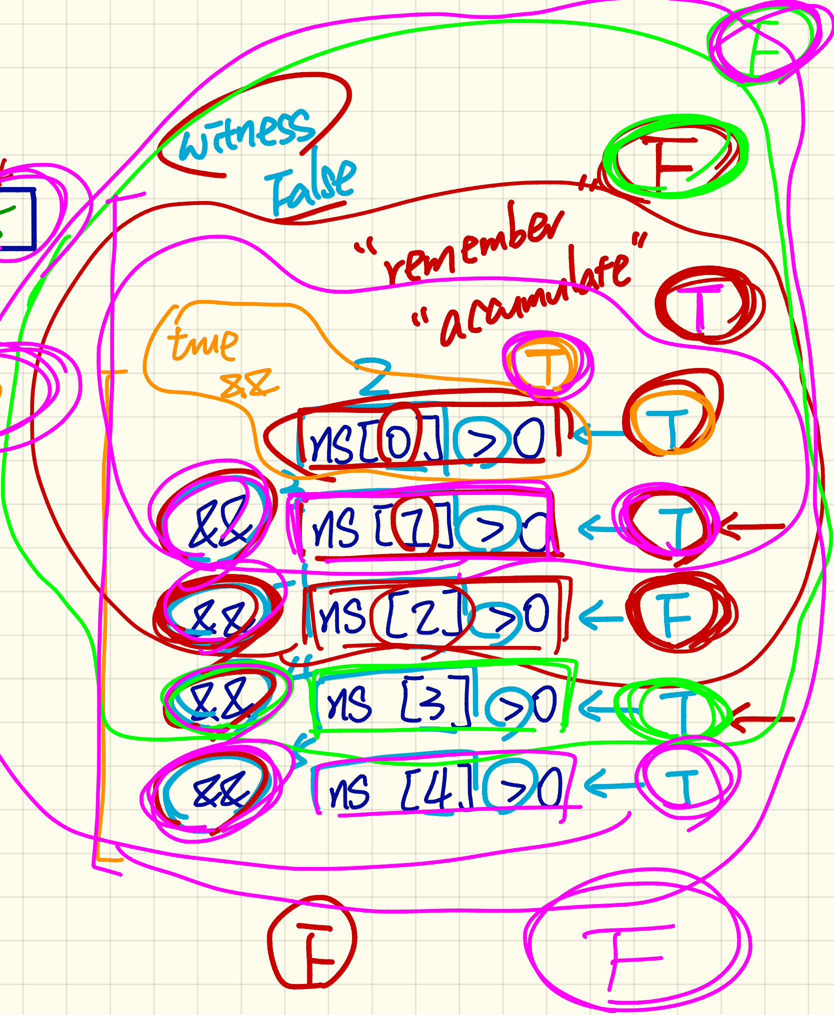
   somePos false

   ||

   empty array

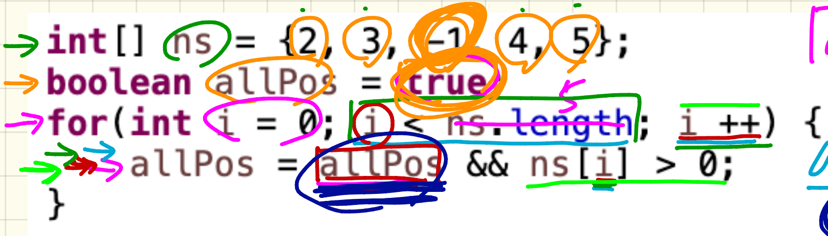


boolean allPositive = true



Computational Problem: Are all numbers positive?

Version 1



```
int[] ns = {2, 3, -1, 4, 5};
boolean allPos = true;
for(int i = 0; i < ns.length; i ++) {
  allPos = allPos && ns[i] > 0;
}
```

ns.length 5

i	i < ns.length		ns[i] > 0
0	0 < 5	T	t
1	1 < 5	T	t
2	2 < 5	T	F
3	3 < 5	T	T
4	4 < 5	T	T
5	5 < 5	F	

allPos = true && ns[0] > 0

2

t.

t

allPos = true && ns[1] > 0

3

t

allPos = true && ns[2] > 0

-1

F

F

allPos = false && ns[3] > 0

T

F

F

ns

0	1	2	3	4
2	3	-1	4	5

~~true~~

~~true~~

~~true~~

~~false~~

false

false

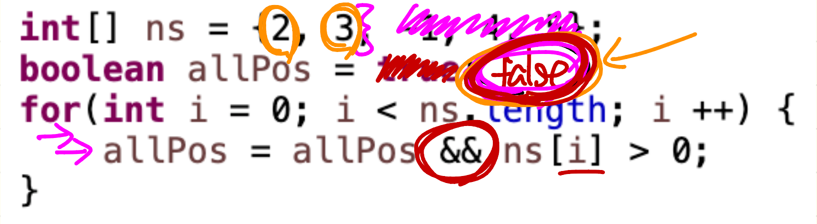
false

allPos

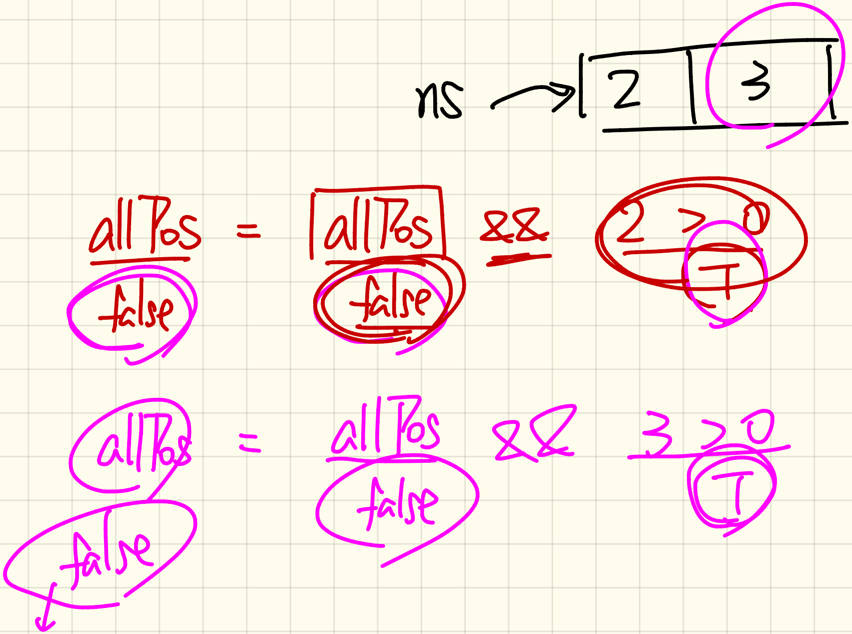
allPos = allPos && ns[4] > 0

F

T



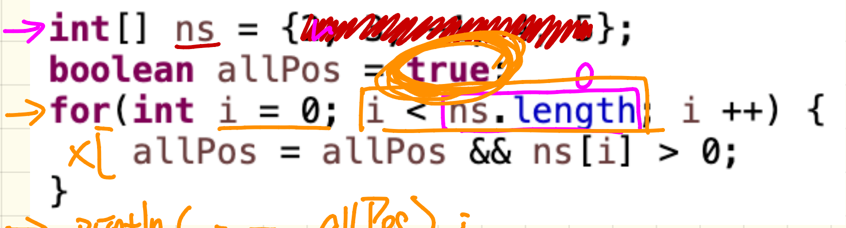
```
int[] ns = {2, 3} ~~-1, 4~~};
boolean allPos = ~~true~~ false;
for(int i = 0; i < ns.length; i ++) {
  allPos = allPos && ns[i] > 0;
}
```



Computational Problem : Are all numbers positive?

Version I

= new int[0];



```
int[] ns = {~~[illegible]~~};
boolean allPos = true;
for(int i = 0; i < ns.length; i ++) {
  allPos = allPos && ns[i] > 0;
}
```

0

println(- - allPos);

true.

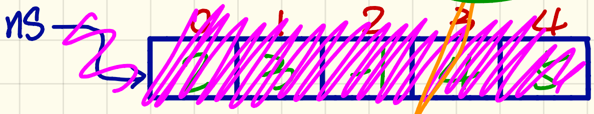
Empty Array

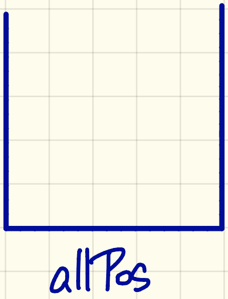
cannot find a violation

cannot find a witness

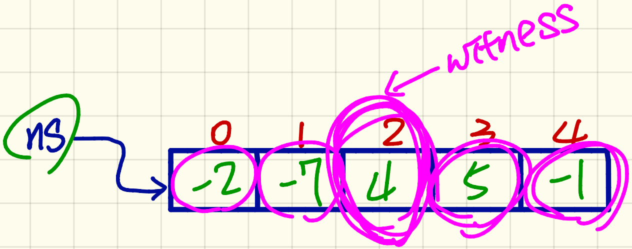
→ all elements in an empty array are positive: true

i	i < ns.length	ns[i] > 0
0	0 < 0 F	





∵ You cannot find any witness that is not positive.



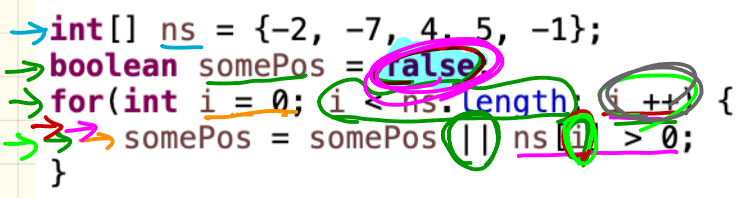
boolean atLeastOnePosNum

somePos

```
   ns[0] > 0    F
|| ns[1] > 0    F    F
|| ns[2] > 0    T    T
|| ns[3] > 0    T    T
|| ns[4] > 0    F    T
```

Computational Problem: Is there a positive number?

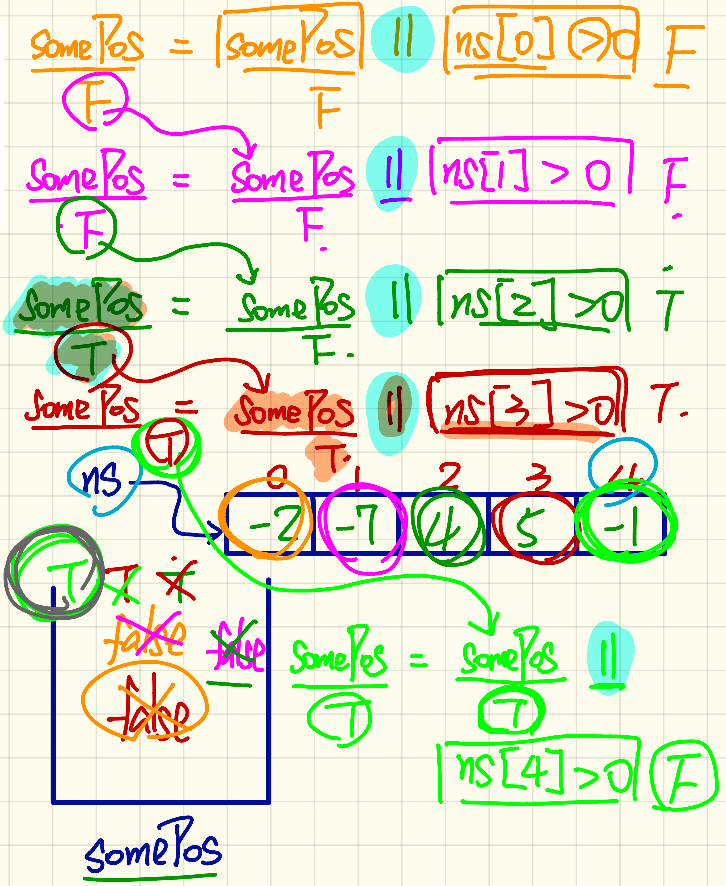
Version I



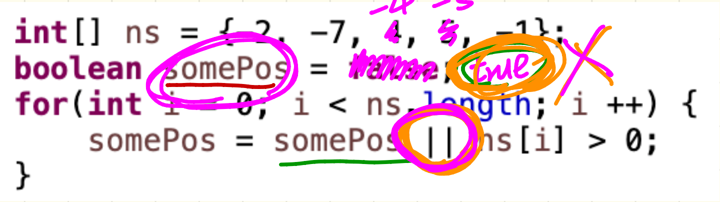
```
int[] ns = {-2, -7, 4, 5, -1};
boolean somePos = false;
for(int i = 0; i < ns.length; i ++) {
    somePos = somePos || ns[i] > 0;
}
```

ns.length 5

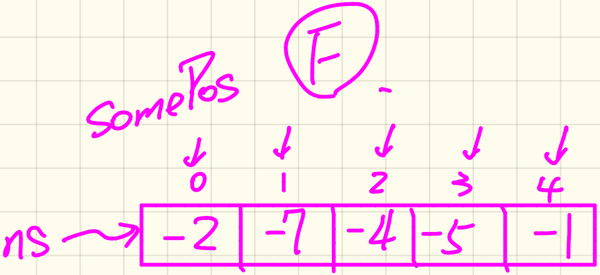
i	i < ns.length		ns[i] > 0
0	0 < 5	T	F
1	1 < 5	T	F
2	2 < 5	T	T
3	3 < 5	T	T
4	4 < 5	T	F
5	5 < 5	F	



somePos = somePos || ns[0] > 0 F
F F
somePos = somePos || ns[1] > 0 F
F F
somePos = somePos || ns[2] > 0 T
T F
somePos = somePos || ns[3] > 0 T
T T
ns
0 1 2 3 4
-2 -7 4 5 -1
T ~~T~~ ~~F~~
~~false~~ ~~false~~
~~false~~
somePos
somePos = somePos || ns[4] > 0 F
T T



```
int[] ns = {-2, -7, 4, 5, -1};
boolean somePos = false;
for(int i = 0; i < ns.length; i ++) {
    somePos = somePos || ns[i] > 0;
}
```



somePos = true || ns[0] > 0

true

somePos = true || ns[1] > 0

true

somePos = true || ns[2] > 0

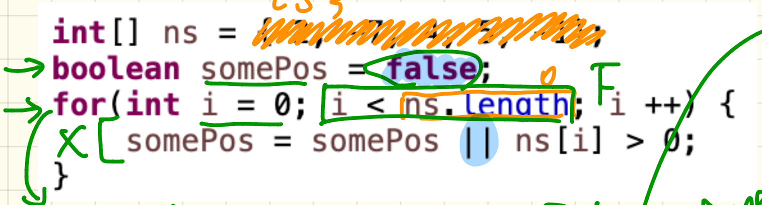
true

Computational Problem: Is there a positive number?

Version I

new int[0];

{ };



```
int[] ns = [illegible];
boolean somePos = false;
for(int i = 0; i < ns.length; i ++) {
  somePos = somePos || ns[i] > 0;
}
```

0

F

X

println(" . . . . " + somePos);

false

cannot find a satisfaction witness

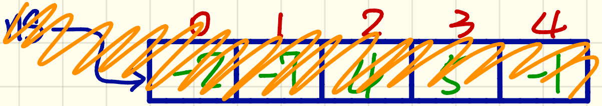
some element in an empty array is positive: false

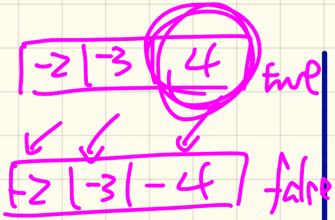
Empty Array

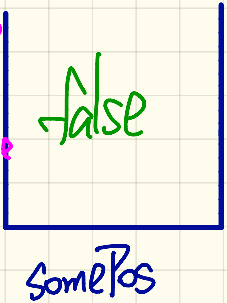
ns.length == 0

ns ~→ |

i	i < ns.length	ns[i] > 0
0	0 < 0 F	

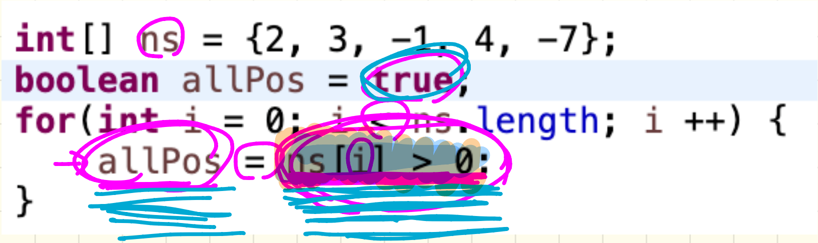




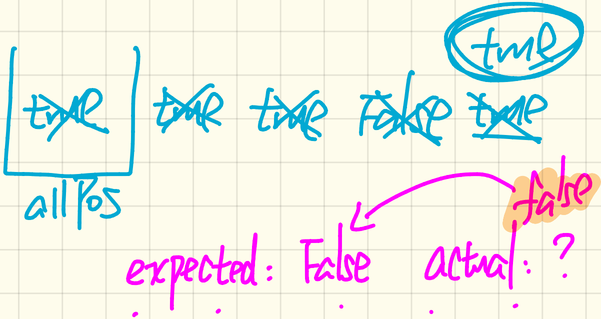


Computational Problem: Are all numbers positive?

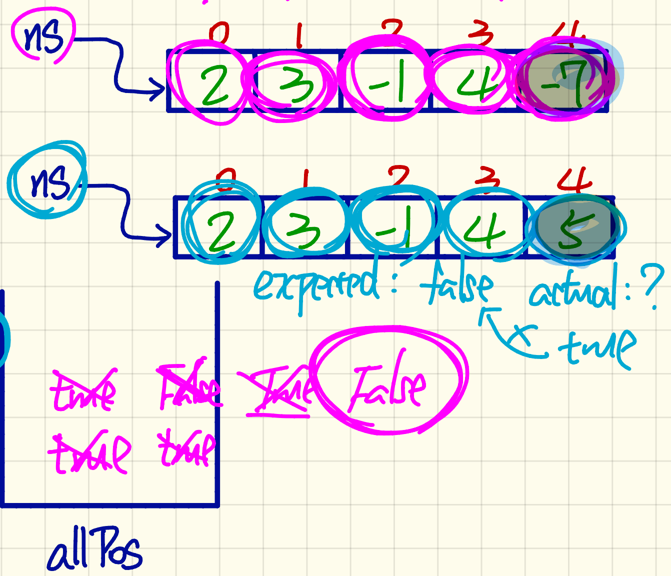
Version 2



```
int[] ns = {2, 3, -1, 4, -7};
boolean allPos = true;
for(int i = 0; i < ns.length; i ++) {
  allPos = ns[i] > 0;
}
```

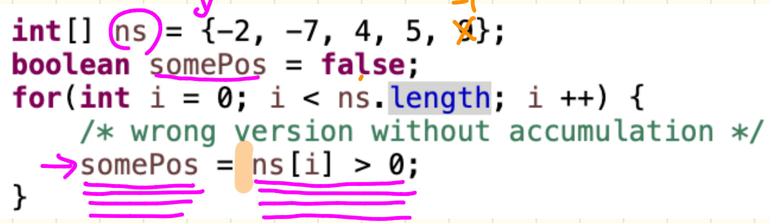


i	i < ns.length	ns[i] > 0	
0	0 < 5 T	2 > 0 T	2>0 T
1	1 < 5 T	3 > 0 T	3>0 T
2	2 < 5 T	-1 > 0 F	-1>0 F
3	3 < 5 T	4 > 0 T	4>0 T
4	4 < 5 T	-7 > 0 F	5>0 T
5	5 < 5 F		



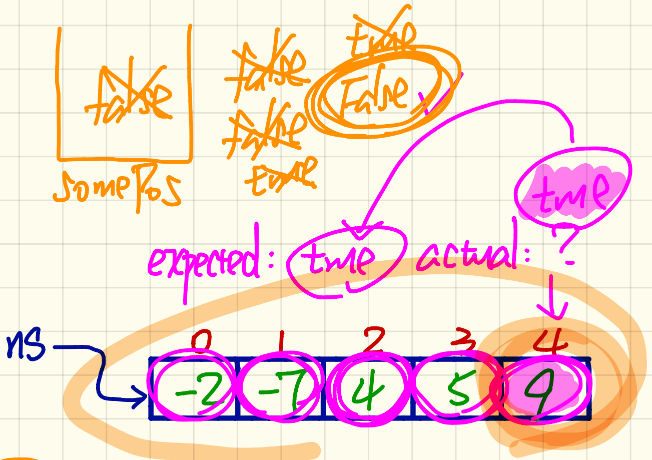
Computational Problem: Is there a positive number?

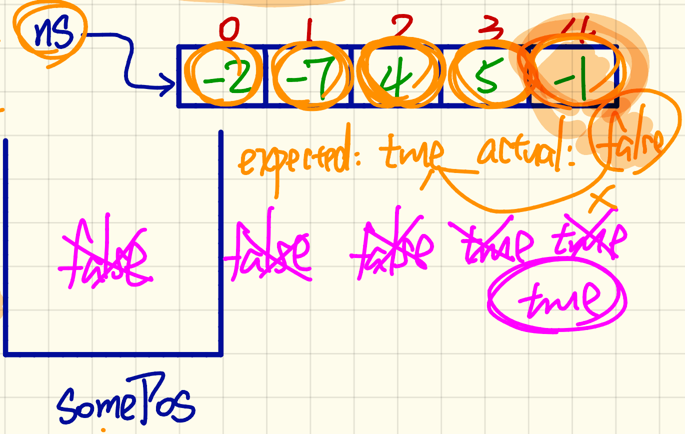
Version 2



```
int[] ns = {-2, -7, 4, 5, 9};
boolean somePos = false;
for(int i = 0; i < ns.length; i ++) {
    /* wrong version without accumulation */
  somePos = ns[i] > 0;
}
```

i	i < ns.length		ns[i] > 0			
0	0 < 5	T	-2 > 0	F	-2>0	F
1	1 < 5	T	-7 > 0	F	-7>0	F
2	2 < 5	T	4 > 0	T	4>0	T
3	3 < 5	T	5 > 0	T	5>0	T
4	4 < 5	T	9 > 0	T	-1>0	F
5	5 < 5	F				



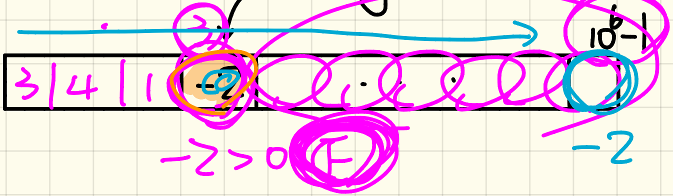


Motivation for Early Exit

all positive?

<violation witness>

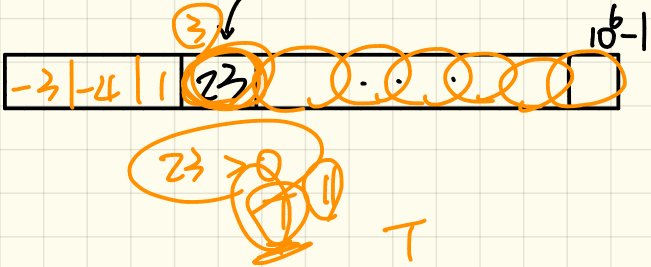
1st negative number found



Some positive?

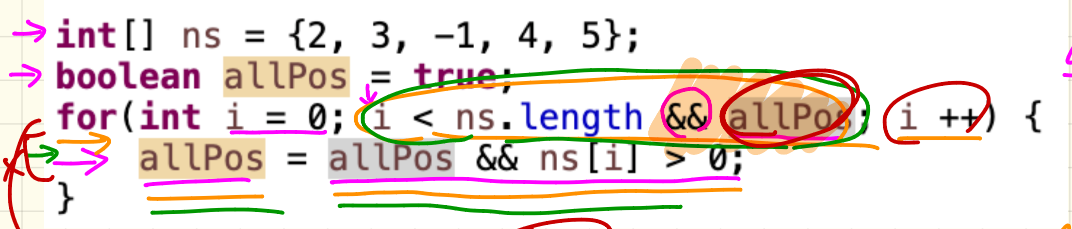
<satisfaction witness>

1st positive number found



Computational Problem : Are all numbers positive?

Version 3



```
int[] ns = {2, 3, -1, 4, 5};
boolean allPos = true;
for(int i = 0; i < ns.length && allPos; i ++) {
  allPos = allPos && ns[i] > 0;
}
```

print (allPos) → false

i	i < ns.length && allPos	ns[i] > 0
0	0 < 5 && true T T	2 > 0 T
1	1 < 5 && true T T	3 > 0 T
2	2 < 5 && true T false	-1 > 0 F
3	3 < 5 T && F F	

allPos = allPos && ns[0] > 0

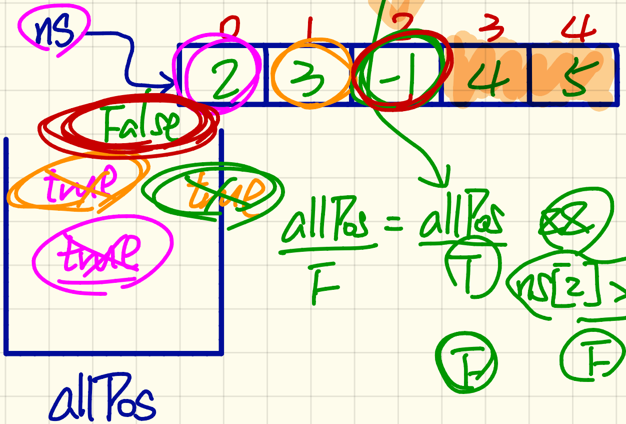
T = T && T → T

allPos = allPos && ns[1] > 0

T = T && T → T

allPos = allPos && ns[2] > 0

F = T && F → F

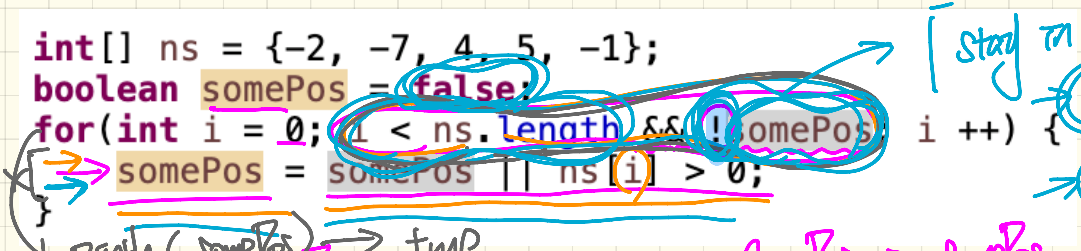


Computational Problem: Is there a positive number?

Version 3

$p \ \&\& \ q \rightarrow\ !(p \&\& q)$ exit

stay $\rightarrow\ !p \ || \ !q$



```
int[] ns = {-2, -7, 4, 5, -1};
boolean somePos = false;
for(int i = 0; i < ns.length && !somePos; i ++) {
  somePos = somePos || ns[i] > 0;
}
```

println(somePos) → true

&& !somePos

[stay in the loop if:
(1) i is a valid index
(2) it's not the case that we've encountered a pos. num.

i	i < ns.length && !somePos	ns[i] > 0
0	0<5 && !false T T T	-2 > 0 F
1	1<5 && !false T T T	-7 > 0 F
2	2<5 && !false T T T	4 > 0 T
3	3<5 && !true T F F	

somePos = somePos || ns[0] > 0
F F F

somePos = somePos || ns[1] > 0
F F. F.

somePos = somePos || ns[2] > 0
T F T

ns →

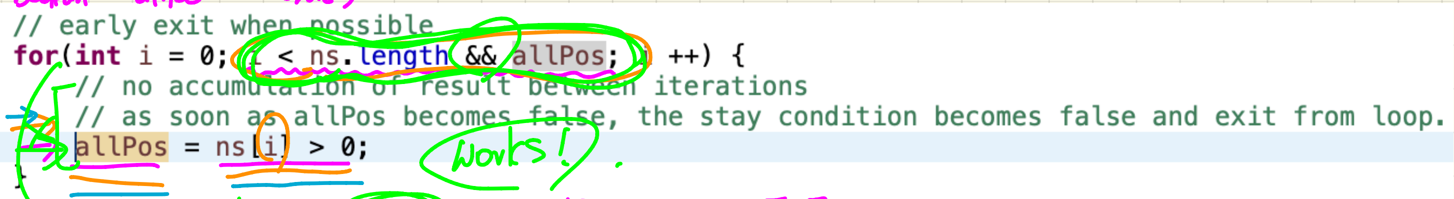
0	1	2	3	4
-2	-7	4	5	-1

somePos: ~~false~~ ~~false~~ ~~false~~ true

Computational Problem : Are all numbers positive?

Version 4

boolean allPos = true;



```
// early exit when possible
for(int i = 0; i < ns.length && allPos; i ++) {
  // no accumulation of result between iterations
  // as soon as allPos becomes false, the stay condition becomes false and exit from loop.
  allPos = ns[i] > 0;
}
```

Works!

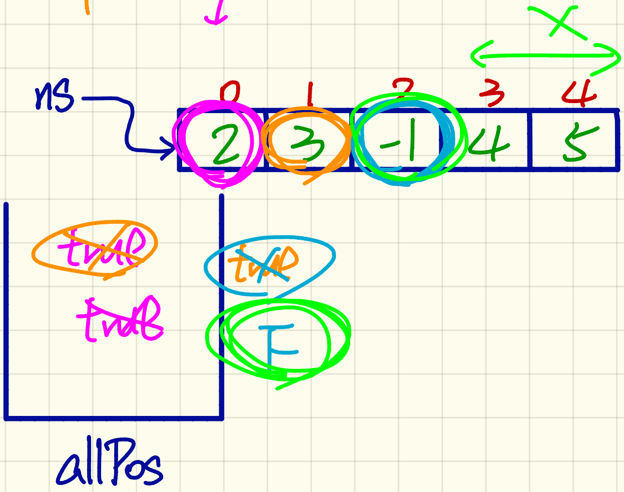
println(allPos) $\boxed{F}$

$\underline{allPos} = \underbrace{ns[0] > 0}_{T}$

$allPos = \underbrace{ns[1] > 0}_{T}$

$\underset{F}{allPos} = \underbrace{allPos[2]}_{-1} > 0$

i	i < ns.length && allPos	ns[i] > 0
0	0<5 && true T	2 > 0 T
1	1<5 && true T T	3 > 0 T
2	2<5 && true T	-1 > 0 F
3	3<5 && F T F	



isSorted

ns[0] <= ns[1]

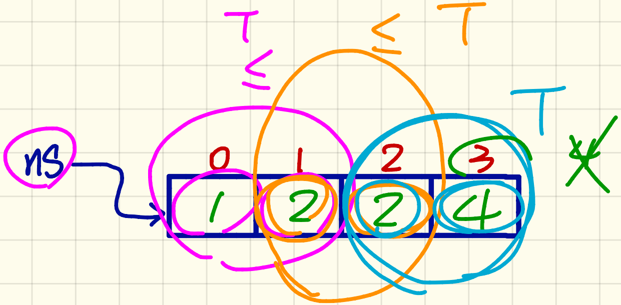
&& ns[1] <= ns[2]

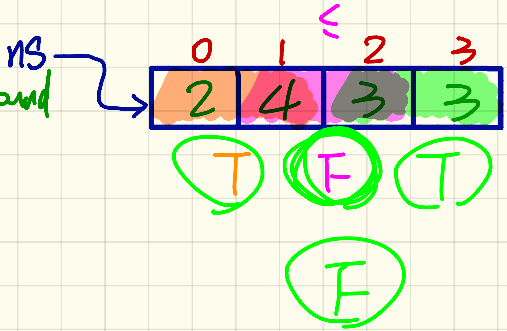
&& ns[2] <= ns[3]

&& ns[3] <= ~~ns[4]~~

ArrayIndexOutOfBound

all elements smaller than or equal to their immediately right neighbours

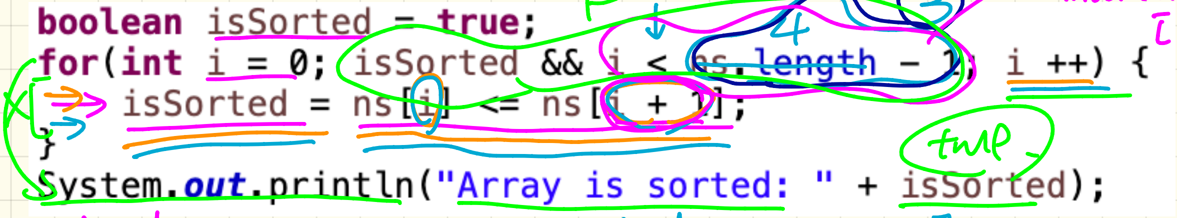




Computational Problem : Is the array sorted?

Correct: $i < ns.length - 1$

incorrect: $i < ns.length$ X



```
boolean isSorted = true;
for(int i = 0; isSorted && i < ns.length - 1; i ++) {
  isSorted = ns[i] <= ns[i + 1];
}
System.out.println("Array is sorted: " + isSorted);
```

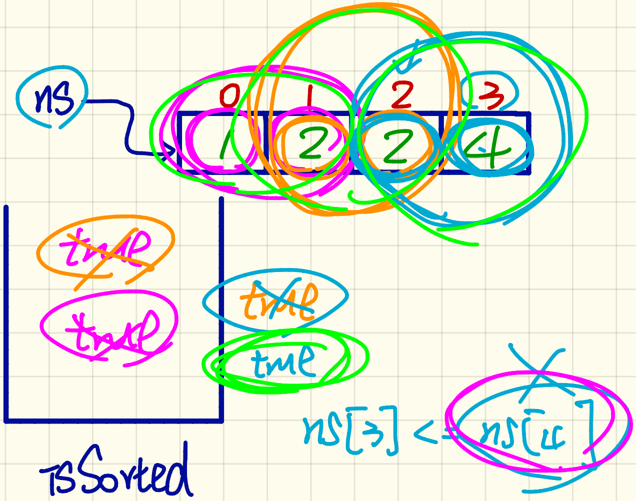
isSorted = ns[0] <= ns[1]

isSorted = ns[1] <= ns[2]

isSorted = ns[2] <= ns[3]

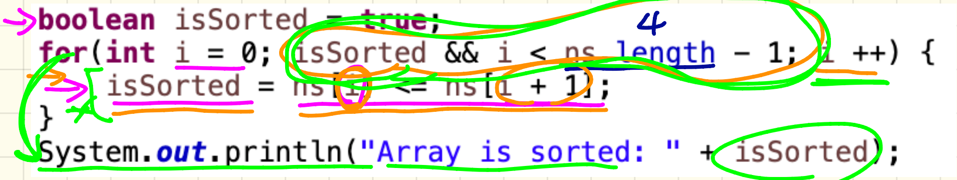
Test Case 1

i	isSorted && i<3	a[i] <= a[i+1]
0	true && 0<3 T T	ns[0] <= ns[1] T
1	true && 1<3 T T	ns[1] <= ns[2] T
2	true && 2<3 T. T	ns[2] <= ns[3] T
3	true && 3<3 F	F



~~ns[3] <= ns[4]~~

Computational Problem: Is the array sorted?



```
boolean isSorted = true;
for(int i = 0; isSorted && i < ns.length - 1; i ++) {
  isSorted = ns[i] <= ns[i + 1];
}
System.out.println("Array is sorted: " + isSorted);
```

4

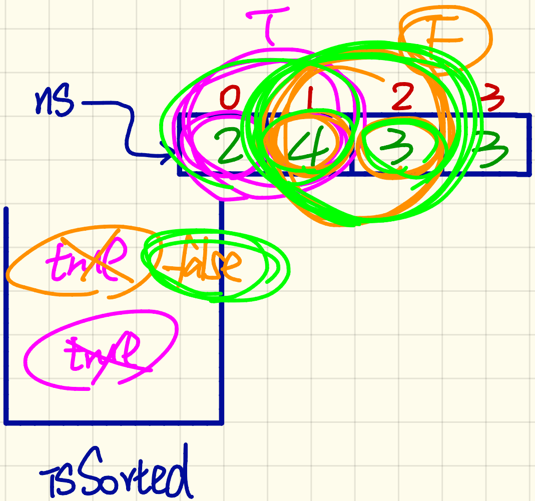
false

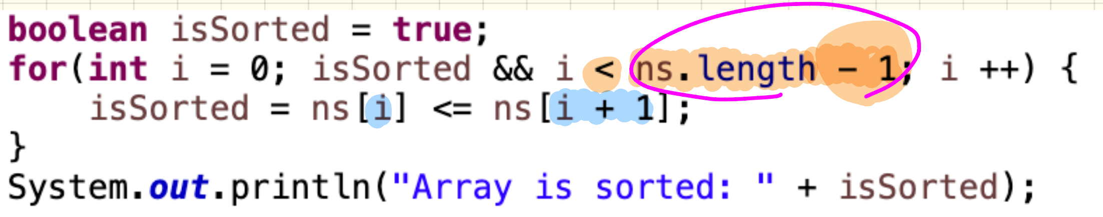
isSorted = ns[0] <= ns[1]

isSorted = ns[1] <= ns[2]

Test Case 2

i	isSorted && i<3	a[i] <= a[i+1]
0	true && 0<3 T T	ns[0] <= ns[1] T
1	true && 1<3 T T	ns[1] <= ns[2] 4 <= 3 F
2	false && 2<3 T F	

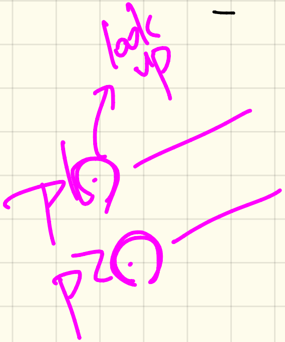




```java
boolean isSorted = true;
for(int i = 0; isSorted && i < ns.length - 1; i ++) {
    isSorted = ns[i] <= ns[i + 1];
}
System.out.println("Array is sorted: " + isSorted);
```

Object-Oriented Programming (OOP)

- Templates (compile-time Java classes)
  - ~ attributes (characteristics)
  - ~ methods
    - • constructor (create new instances)
    - • mutator (modify attribute values)
    - • accessor (query)
- Instances / Entities (runtime objects)
  - ~ calling constructor to create objects
  - ~ use of "dot notation" to
    - • get attribute values
    - • call accessor or mutator



Model: From Entities to Classes

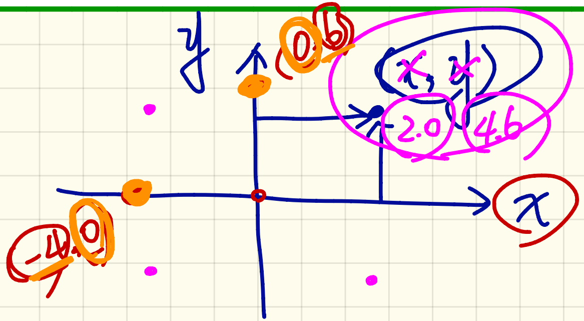
Identify Critical Nouns & Verbs

Nouns → classes, attributes

Verbs → methods

Example

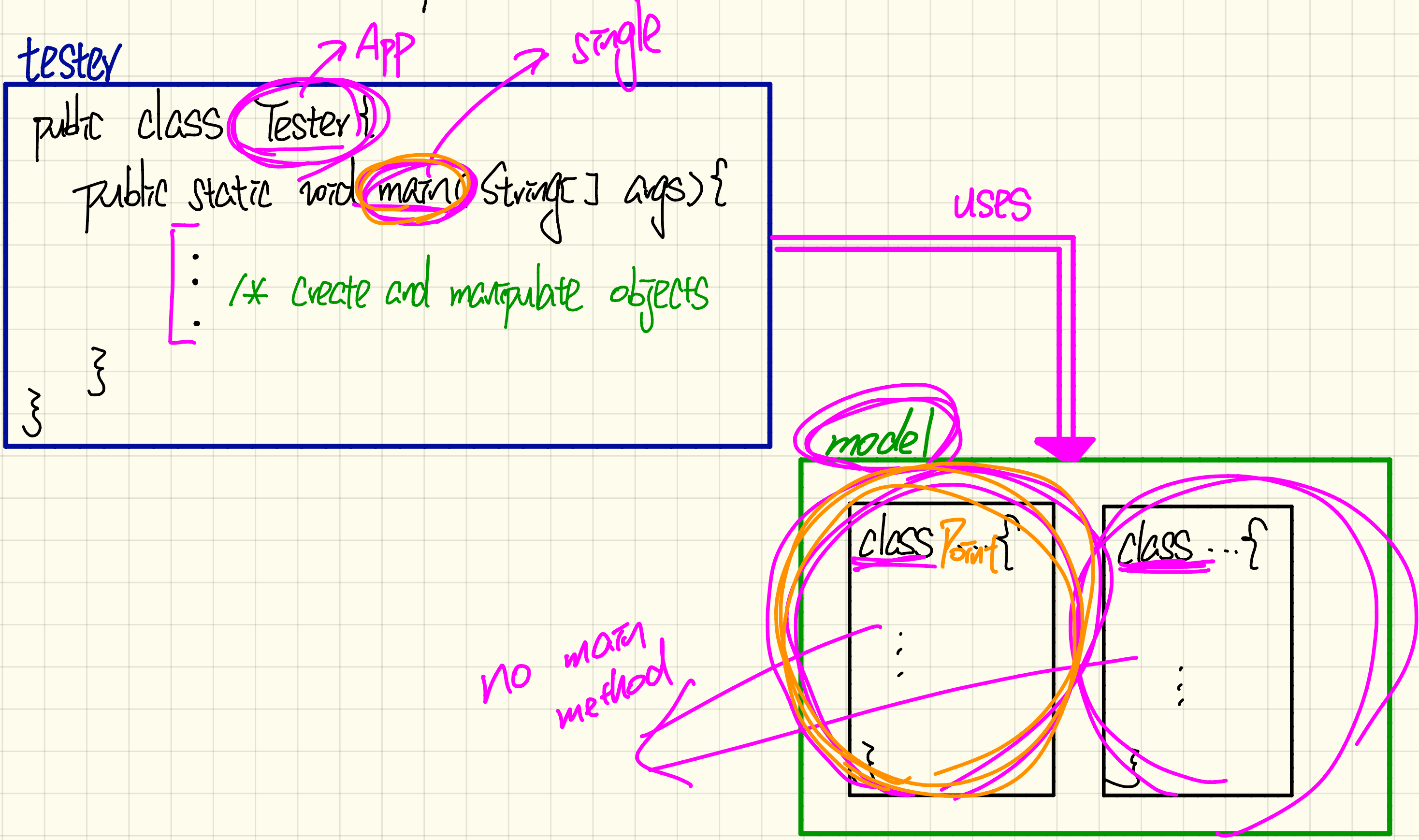
Points on a two-dimensional plane are identified by their signed distances from the X- and Y-axises. A point may move arbitrarily towards any direction on the plane. Given two points, we are often interested in knowing the distance between them.



'x' x-axis

'y' y-axis

Test Driven Development (TDD)



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

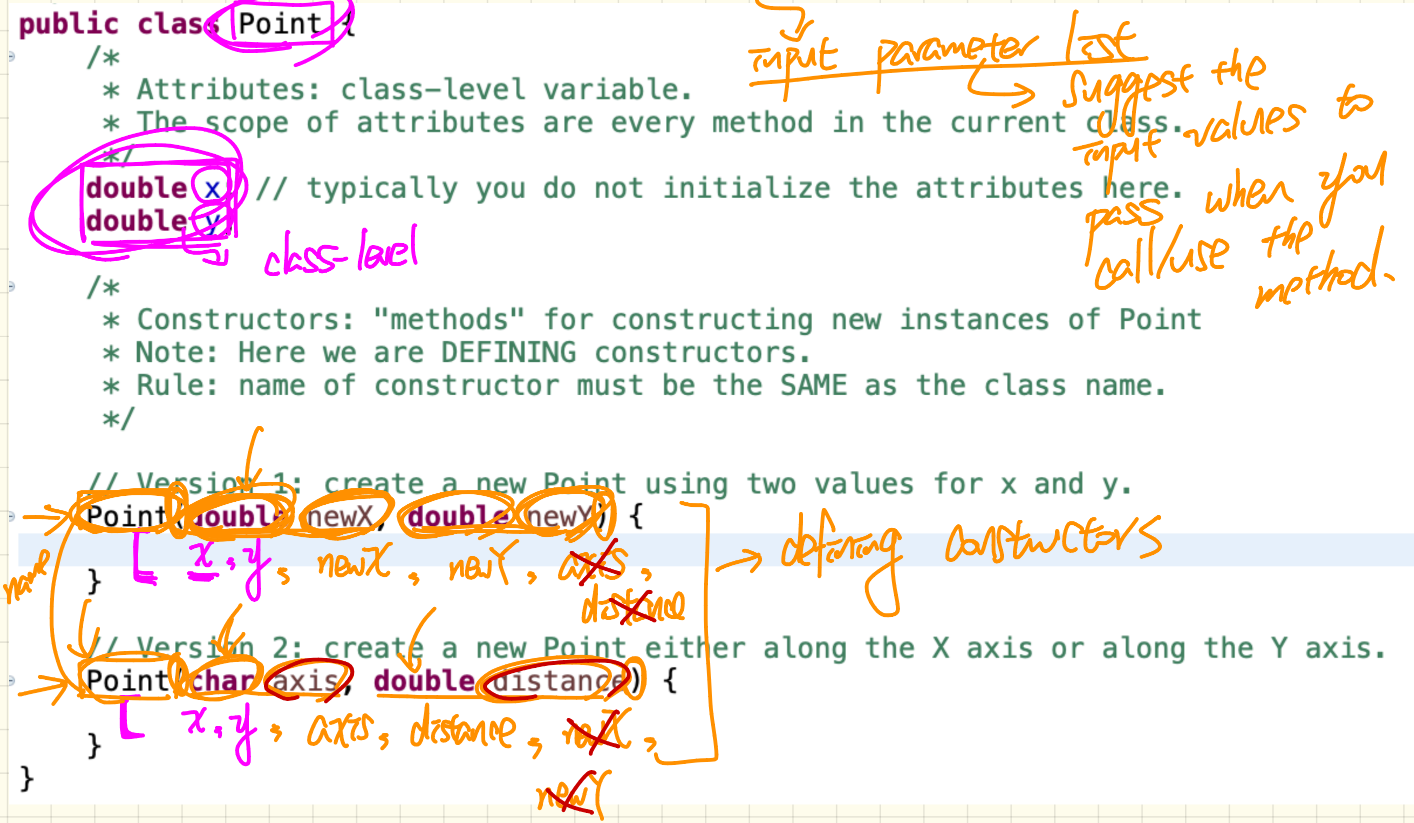
```
class Point {
    :
}
```

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

Point( )

(... , ... , ... ,)

input parameter list → suggest the input values to pass when you call/use the method.



```
public class Point {
    /*
     * Attributes: class-level variable.
     * The scope of attributes are every method in the current class.
     */
    double x; // typically you do not initialize the attributes here.
    double y;

    /*
     * Constructors: "methods" for constructing new instances of Point
     * Note: Here we are DEFINING constructors.
     * Rule: name of constructor must be the SAME as the class name.
     */

    // Version 1: create a new Point using two values for x and y.
    Point(double newX, double newY) {

    }

    // Version 2: create a new Point either along the X axis or along the Y axis.
    Point(char axis, double distance) {

    }
}
```

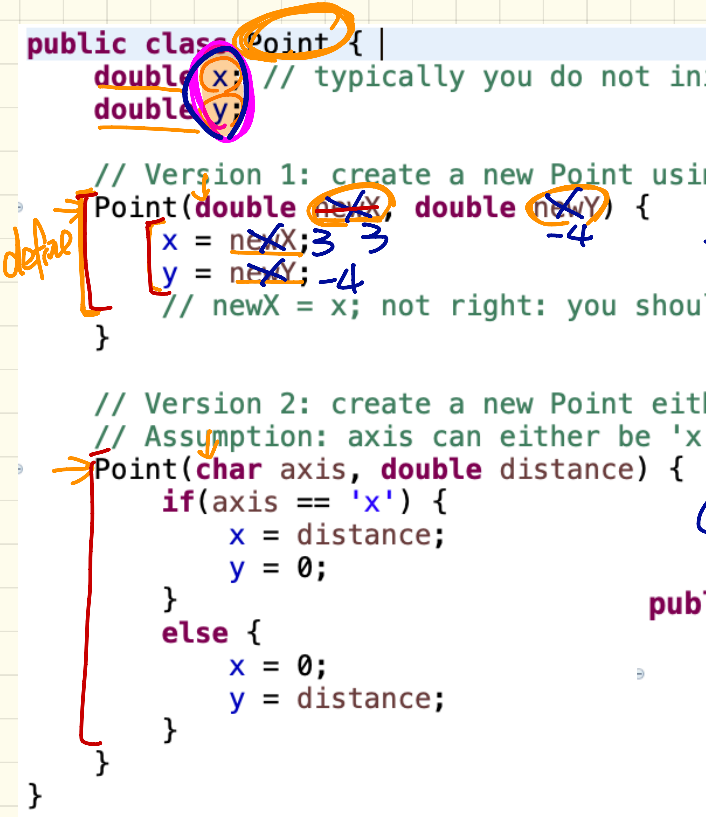
class-level

name

x, y, newX, newY, ~~axis~~, ~~distance~~

x, y, axis, distance, ~~newX~~, ~~newY~~

→ defining constructors



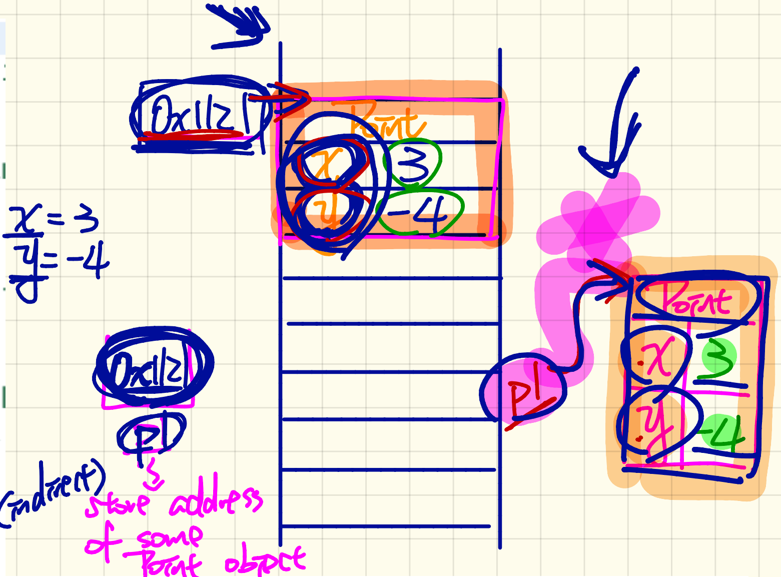
```
public class Point {
    double x; // typically you do not in
    double y;

    // Version 1: create a new Point usi
    Point(double newX, double newY) {
        x = newX;
        y = newY;
        // newX = x; not right: you shou
    }

    // Version 2: create a new Point eit
    // Assumption: axis can either be 'x
    Point(char axis, double distance) {
        if(axis == 'x') {
            x = distance;
            y = 0;
        }
        else {
            x = 0;
            y = distance;
        }
    }
}
```

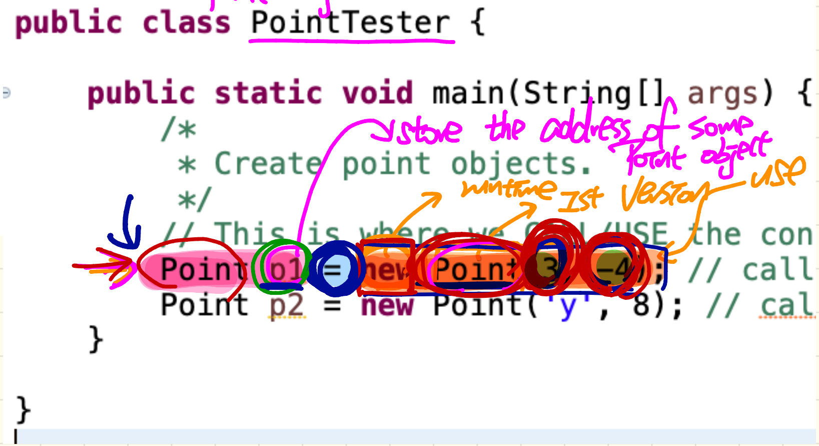
define

3 3

-4

-4





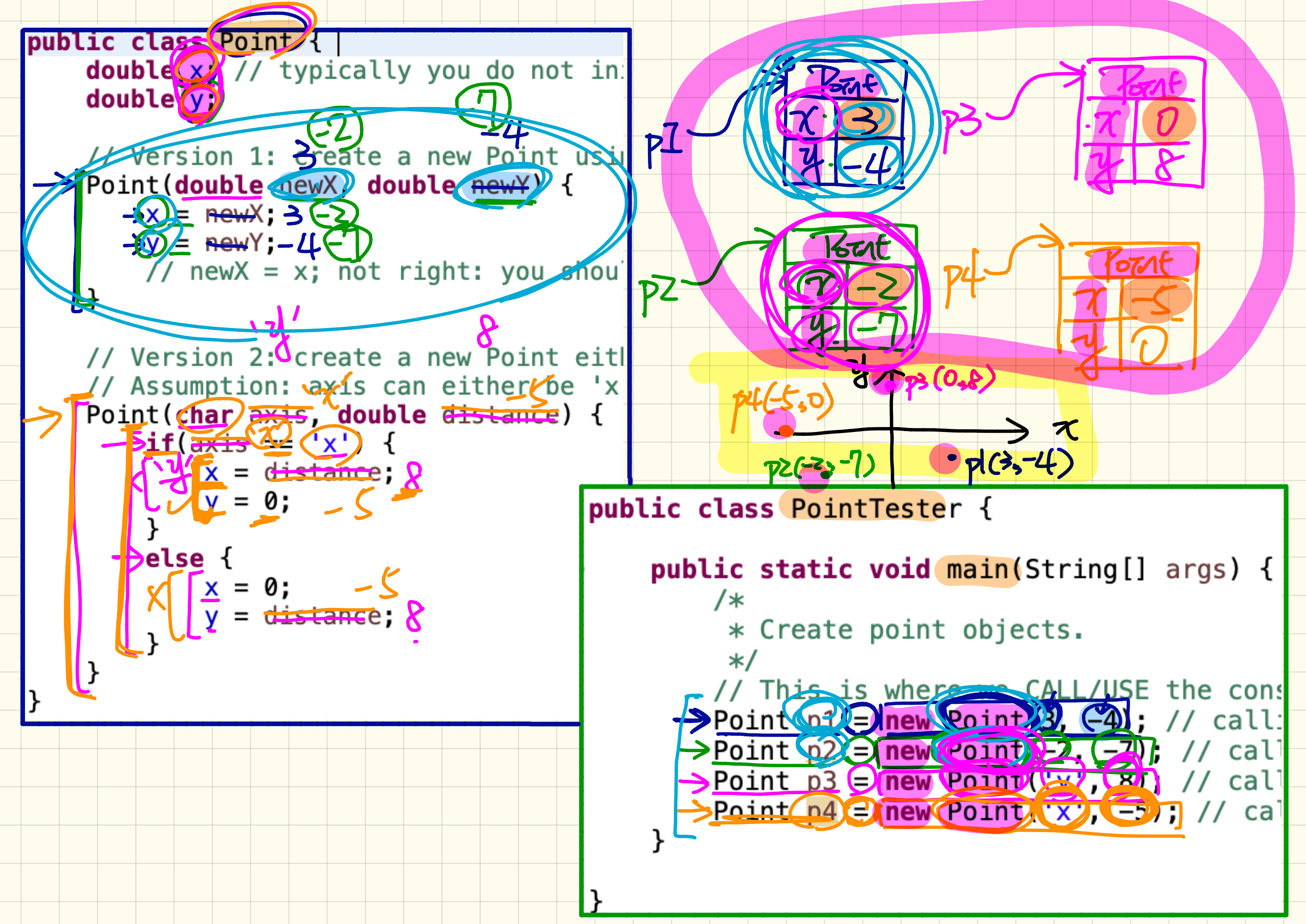
```
public class PointTester {

    public static void main(String[] args) {
        /*
         * Create point objects.
         */
        // This is where we CALL/USE the con
        Point p1 = new Point(3, -4); // call
        Point p2 = new Point('y', 8); // cal
    }

}
```

store the address of some Point object

runtime

1st version use

Method: a reusable block of code.

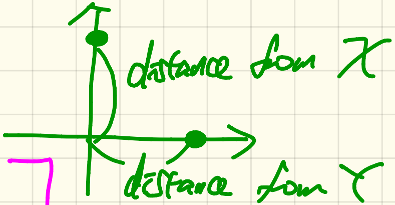


```
public class Point {
    double x; // typically you do not in
    double y;

    // Version 1: create a new Point usi
    Point(double newX, double newY) {
        x = newX;
        y = newY;
        // newX = x; not right: you shou
    }

    // Version 2: create a new Point eit
    // Assumption: axis can either be 'x
    Point(char axis, double distance) {
        if(axis == 'x') {
            x = distance;
            y = 0;
        }
        else {
            x = 0;
            y = distance;
        }
    }
}
```

```
public class PointTester {

    public static void main(String[] args) {
        /*
         * Create point objects.
         */
        // This is where we CALL/USE the cons
        Point p1 = new Point(3, -4); // call
        Point p2 = new Point(-2, -7); // call
        Point p3 = new Point('y', 8); // call
        Point p4 = new Point('x', -5); // ca
    }

}
```

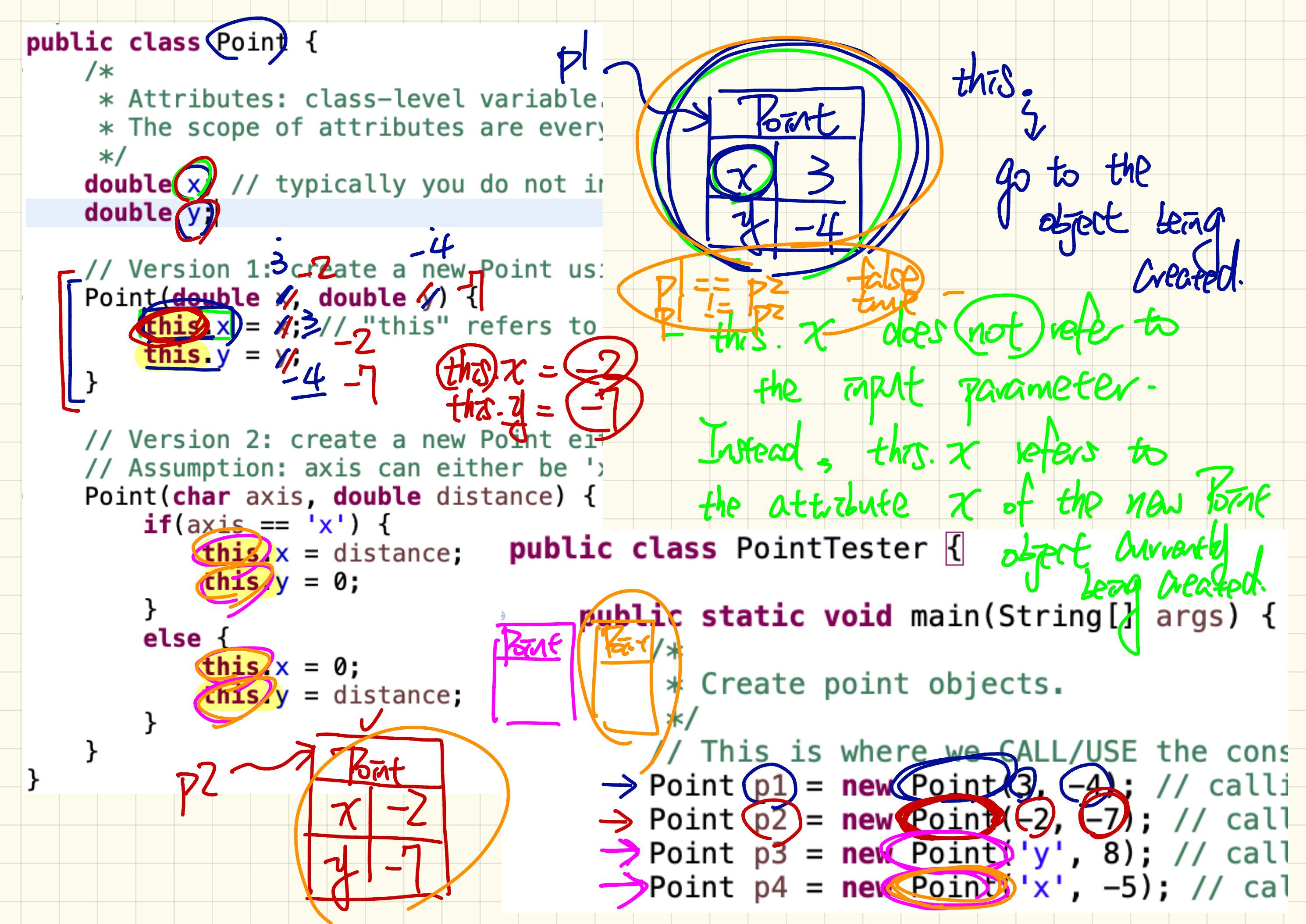
Versions of Constructors

```
Point(double newX, double newY)
→ Point(double, double)
→ Point(char axis, double distance)
→ Point(char, double)
```



```
→ Point (double distanceFromXAxis){
  } - Point(double)
→ Point (double distanceFromYAxis){
  } - Point(double)
```

```
new Point(3.4);
```



```
public class Point {
    /*
     * Attributes: class-level variable
     * The scope of attributes are every
     */
    double x; // typically you do not i
    double y;

    // Version 1: create a new Point us
    Point(double x, double y) {
        this.x = x; // "this" refers to
        this.y = y;
    }

    // Version 2: create a new Point ei
    // Assumption: axis can either be 'x
    Point(char axis, double distance) {
        if(axis == 'x') {
            this.x = distance;
            this.y = 0;
        }
        else {
            this.x = 0;
            this.y = distance;
        }
    }
}
```

```
public class PointTester {

    public static void main(String[] args) {
        /*
         * Create point objects.
         */
        // This is where we CALL/USE the cons
        Point p1 = new Point(3, -4); // calli
        Point p2 = new Point(-2, -7); // call
        Point p3 = new Point('y', 8); // call
        Point p4 = new Point('x', -5); // cal
```

this.x = -2
this.y = -7

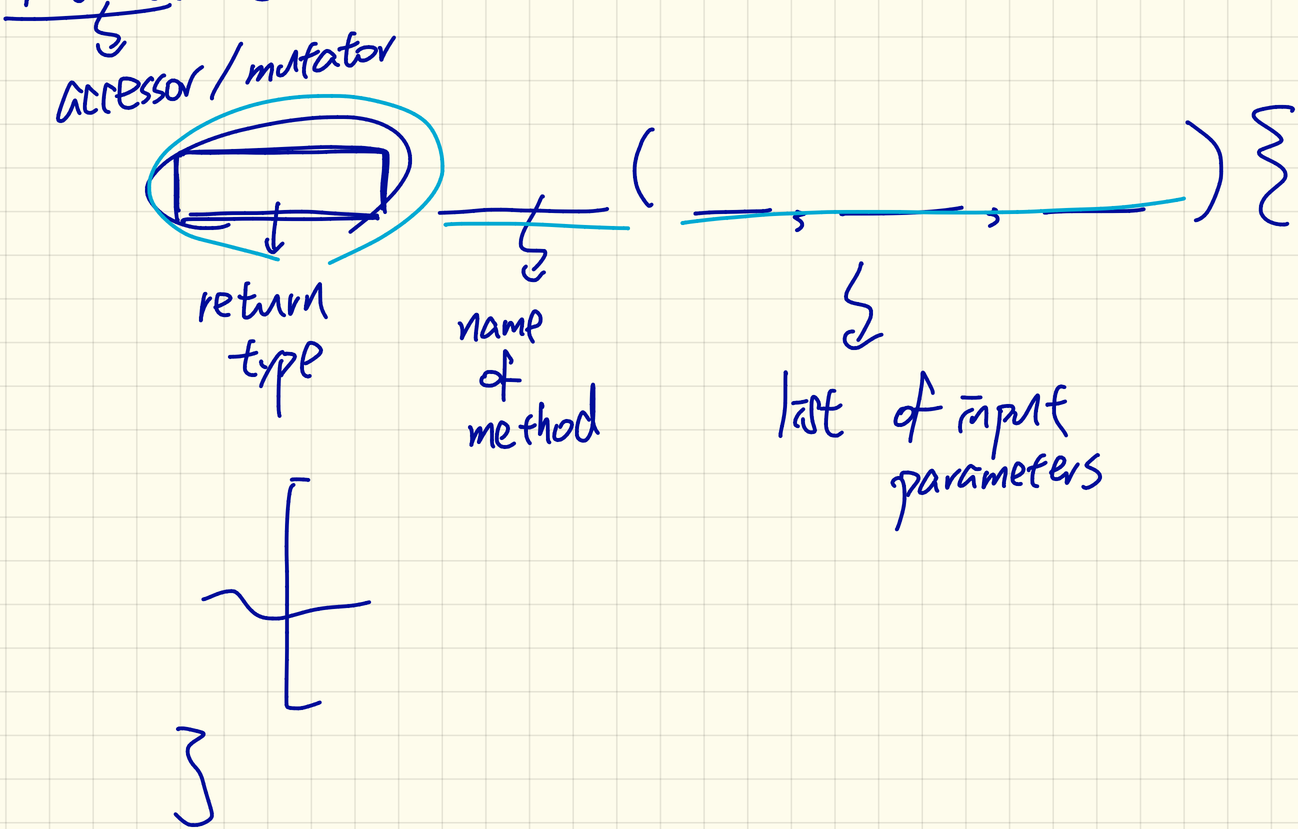
this ↓ go to the object being created.

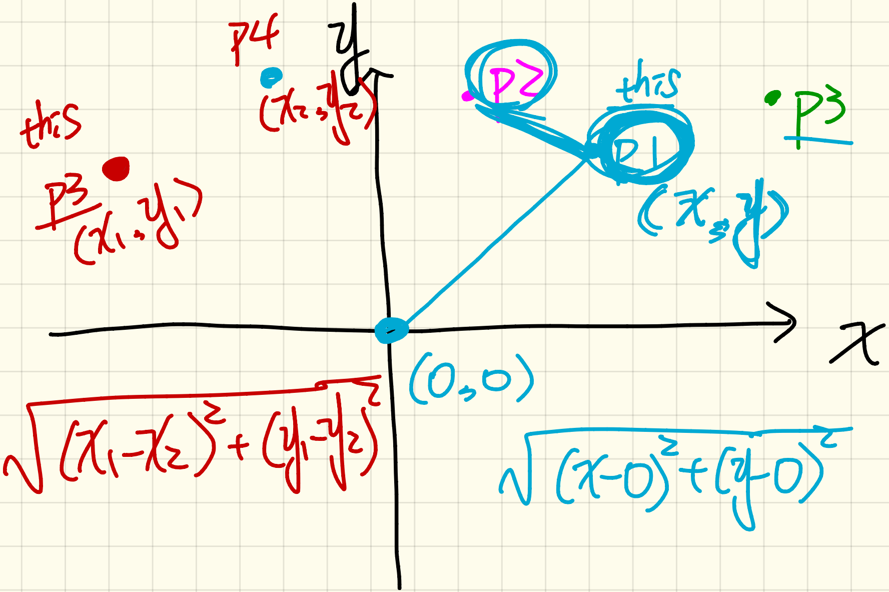
p1 == p2 false

p1 != p2 true

this.x does not refer to the input parameter. Instead, this.x refers to the attribute x of the new Point object currently being created.

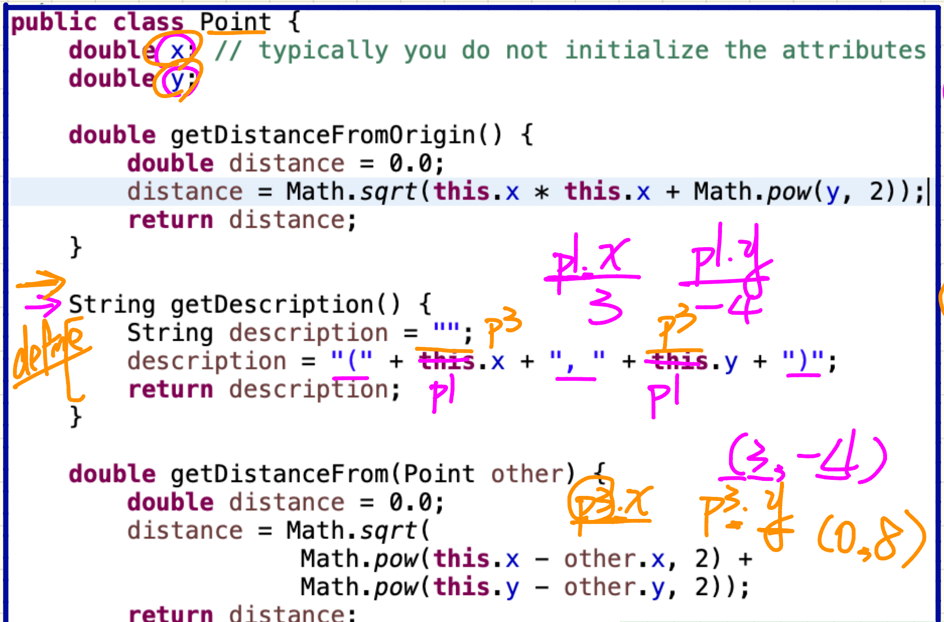
method (other than constructor)





$$\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$$

$$\sqrt{(x-0)^2+(y-0)^2}$$



```
public class Point {
    double x; // typically you do not initialize the attributes
    double y;

    double getDistanceFromOrigin() {
        double distance = 0.0;
        distance = Math.sqrt(this.x * this.x + Math.pow(y, 2));
        return distance;
    }

    String getDescription() {
        String description = "";
        description = "(" + this.x + ", " + this.y + ")";
        return description;
    }

    double getDistanceFrom(Point other) {
        double distance = 0.0;
        distance = Math.sqrt(
                        Math.pow(this.x - other.x, 2) +
                        Math.pow(this.y - other.y, 2));
        return distance;
    }
}
```

define

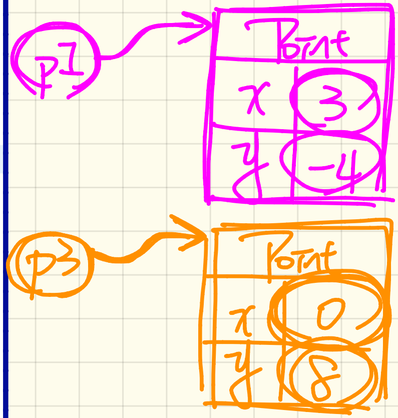
p3 p1 p3 p1

p1.x 3 p1.y -4

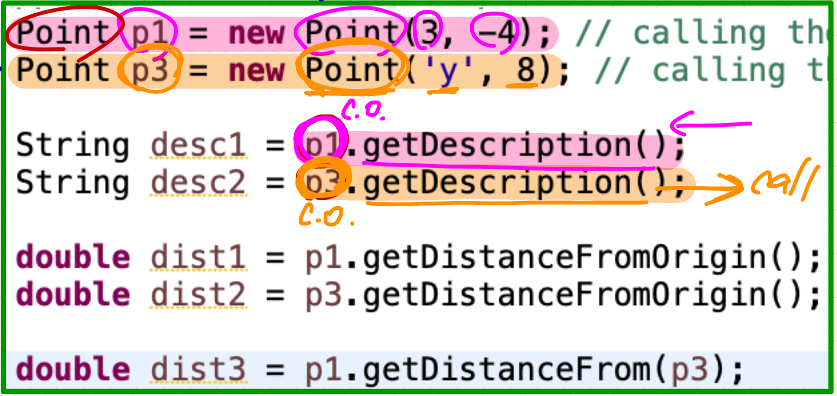
(3, -4)

p3.x p3.y 0 8

(0, 8)



PointTester.main



```
Point p1 = new Point(3, -4); // calling the
Point p3 = new Point('y', 8); // calling t

String desc1 = p1.getDescription();
String desc2 = p3.getDescription();

double dist1 = p1.getDistanceFromOrigin();
double dist2 = p3.getDistanceFromOrigin();

double dist3 = p1.getDistanceFrom(p3);
```

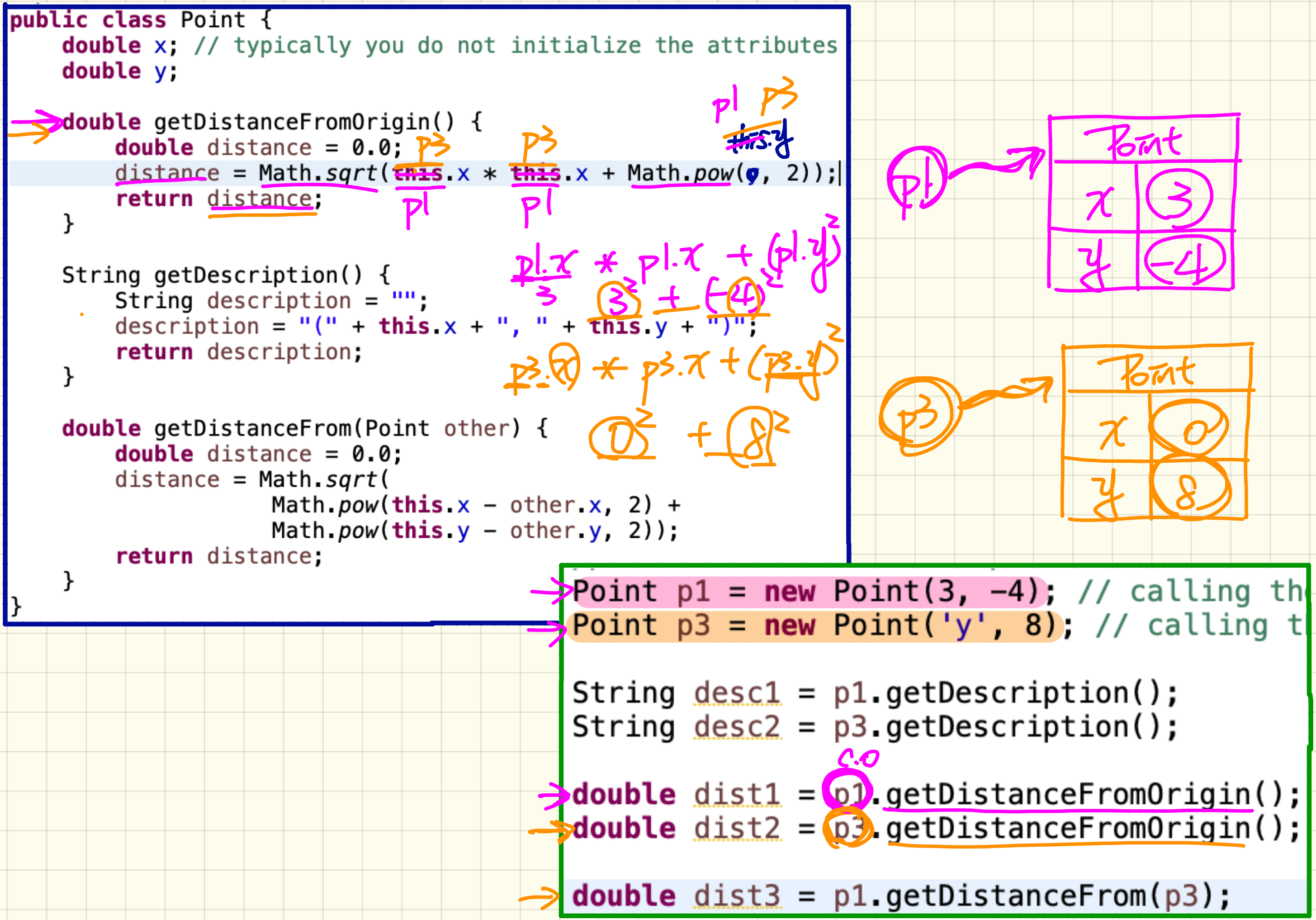
c.o. c.o. → call

p1.getDescription() method call

context object

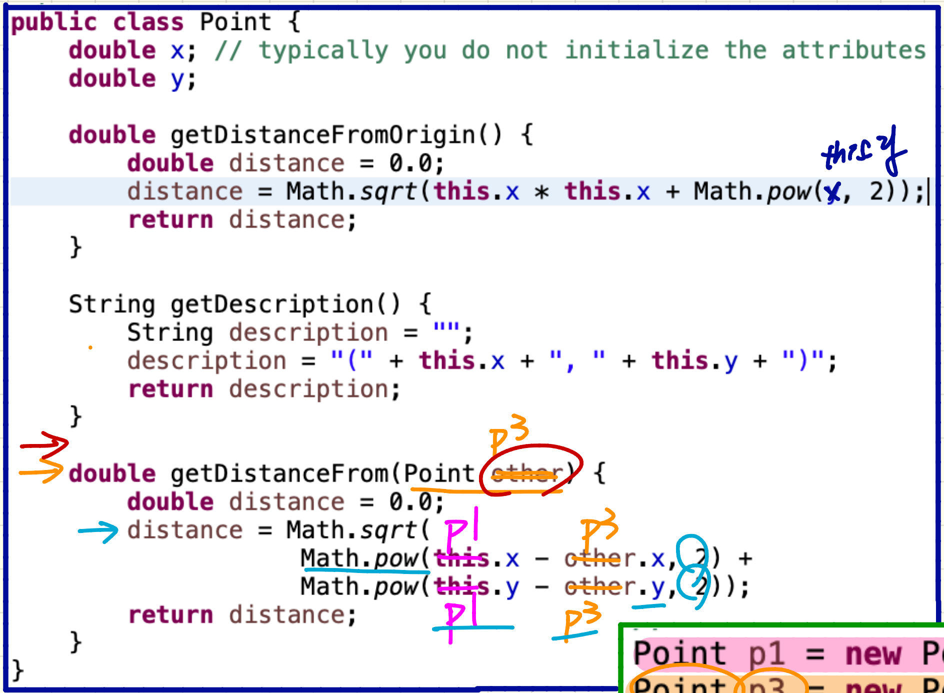
method being executed

locate the object whose addr. is stored in p1.

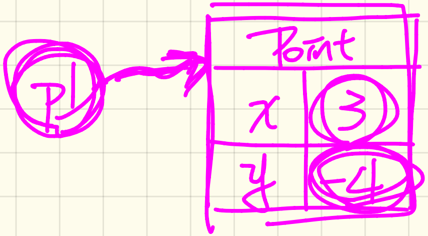


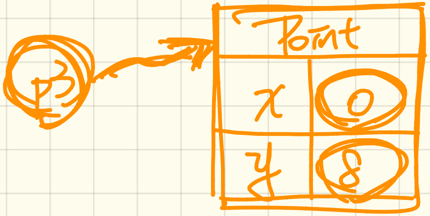
```java
public class Point {
    double x; // typically you do not initialize the attributes
    double y;

    double getDistanceFromOrigin() {
        double distance = 0.0;
        distance = Math.sqrt(this.x * this.x + Math.pow( , 2));
        return distance;
    }

    String getDescription() {
        String description = "";
        description = "(" + this.x + ", " + this.y + ")";
        return description;
    }

    double getDistanceFrom(Point other) {
        double distance = 0.0;
        distance = Math.sqrt(
                        Math.pow(this.x - other.x, 2) +
                        Math.pow(this.y - other.y, 2));
        return distance;
    }
}
```

```java
Point p1 = new Point(3, -4); // calling th
Point p3 = new Point('y', 8); // calling t

String desc1 = p1.getDescription();
String desc2 = p3.getDescription();

double dist1 = p1.getDistanceFromOrigin();
double dist2 = p3.getDistanceFromOrigin();

double dist3 = p1.getDistanceFrom(p3);
```



```java
public class Point {
    double x; // typically you do not initialize the attributes
    double y;

    double getDistanceFromOrigin() {
        double distance = 0.0;
        distance = Math.sqrt(this.x * this.x + Math.pow(x, 2));
        return distance;
    }

    String getDescription() {
        String description = "";
        description = "(" + this.x + ", " + this.y + ")";
        return description;
    }

    double getDistanceFrom(Point other) {
        double distance = 0.0;
        distance = Math.sqrt(
                    Math.pow(this.x - other.x, 2) +
                    Math.pow(this.y - other.y, 2));
        return distance;
    }
}
```



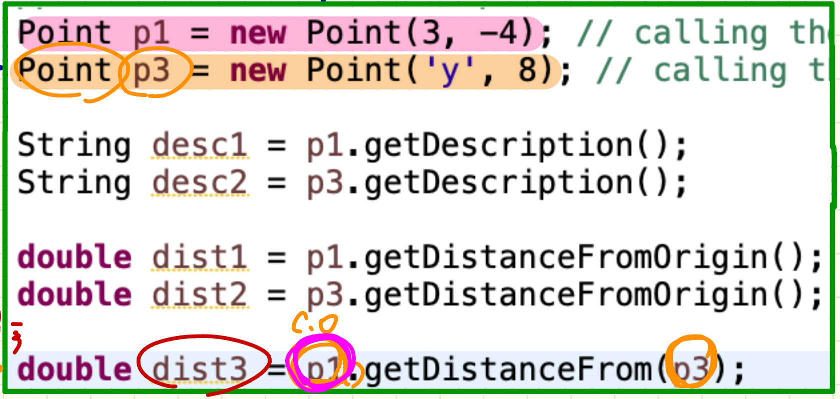


distance = $\sqrt{(\overset{3}{p1.x} - \overset{0}{p3.x})^2 + (\overset{-4}{p1.y} - \overset{8}{p3.y})^2}$

$= \sqrt{(0-3)^2 + (8-(-4))^2}$

double dist4 = p3.getDistanceFrom(p1);

c.o.



```java
Point p1 = new Point(3, -4); // calling th
Point p3 = new Point('y', 8); // calling t

String desc1 = p1.getDescription();
String desc2 = p3.getDescription();

double dist1 = p1.getDistanceFromOrigin();
double dist2 = p3.getDistanceFromOrigin();

double dist3 = p1.getDistanceFrom(p3);
```

Point

Attributes

x
y

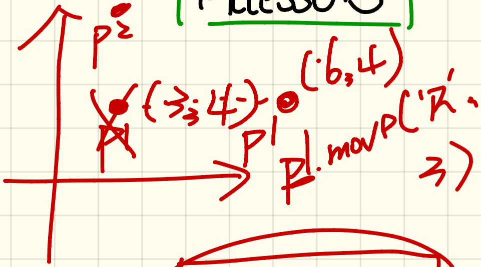
character what an instance is like

Constructors

Point(double, double)
Point(char, double)

constructing new instances

Accessors



p1. p3.

String getDescription()
double getDistanceFromOrigin()
double getDistanceFrom(Point other)

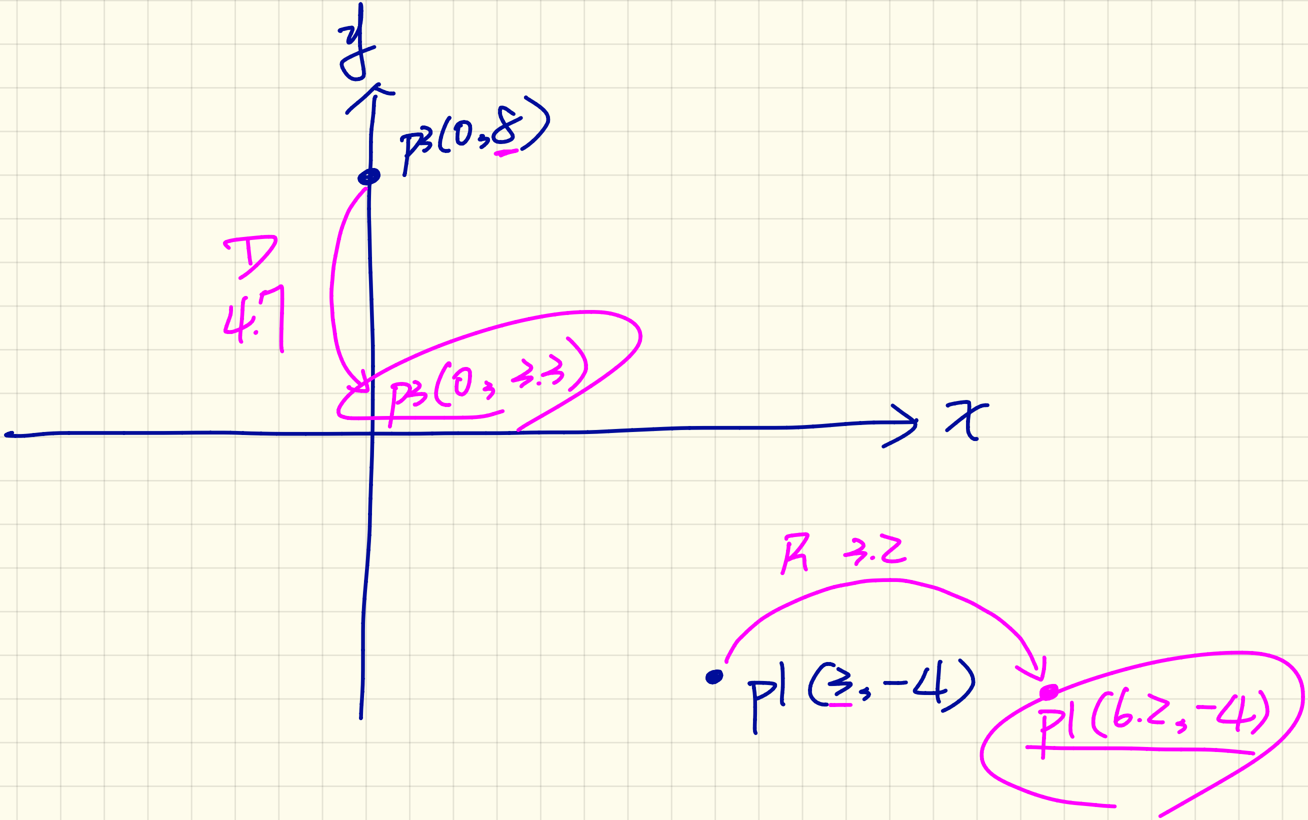
inquiring about the context object

Mutators

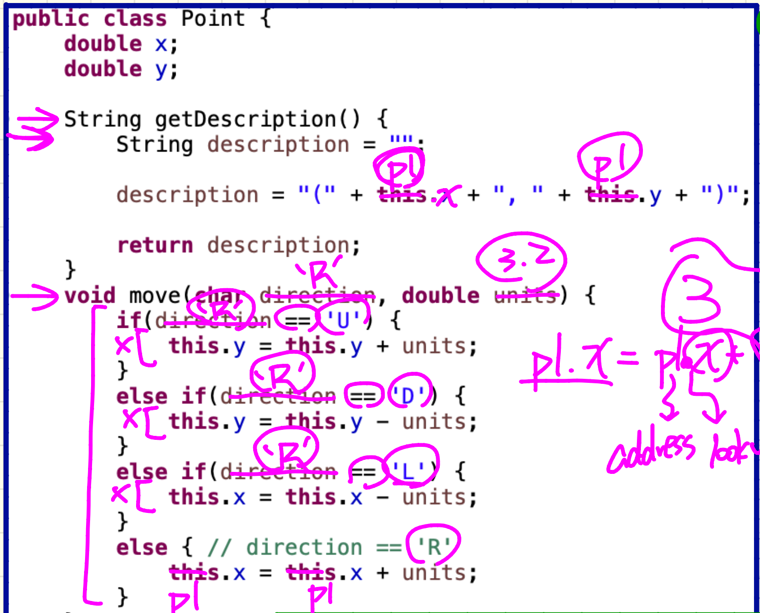
void move(char, double)

return nothing ← void

mutating the attribute values of the context object


y
x
P3(0,8)
D
4.7
P3(0,3.3)
R 3.2
P1(3,-4)
P1(6.2,-4)

Con. obj . method

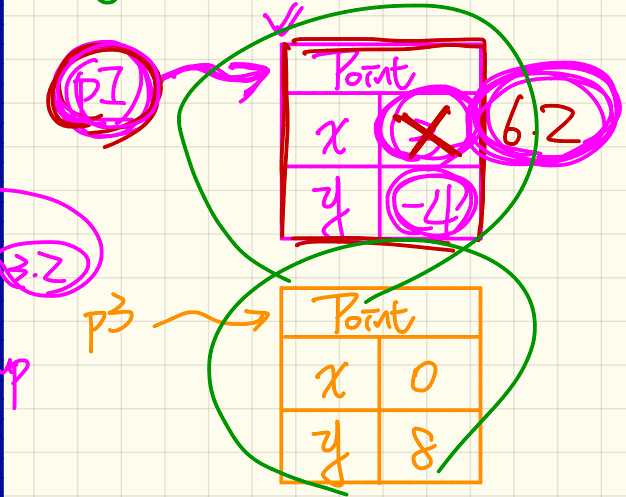


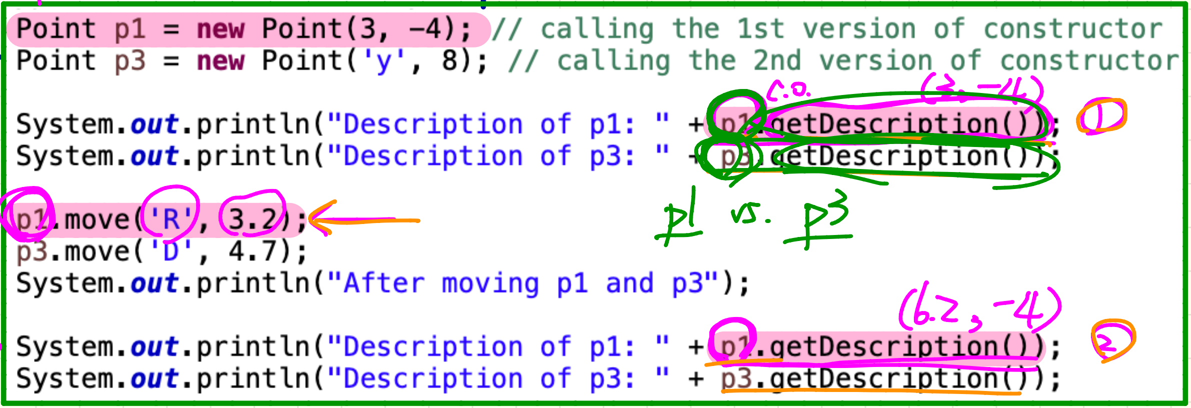
```
public class Point {
    double x;
    double y;

    String getDescription() {
        String description = "";

        description = "(" + this.x + ", " + this.y + ")";

        return description;
    }
    void move(char direction, double units) {
        if(direction == 'U') {
            this.y = this.y + units;
        }
        else if(direction == 'D') {
            this.y = this.y - units;
        }
        else if(direction == 'L') {
            this.x = this.x - units;
        }
        else { // direction == 'R'
            this.x = this.x + units;
        }
    }
}
```

p1.x = p1.x + 3.2

3

address lookup



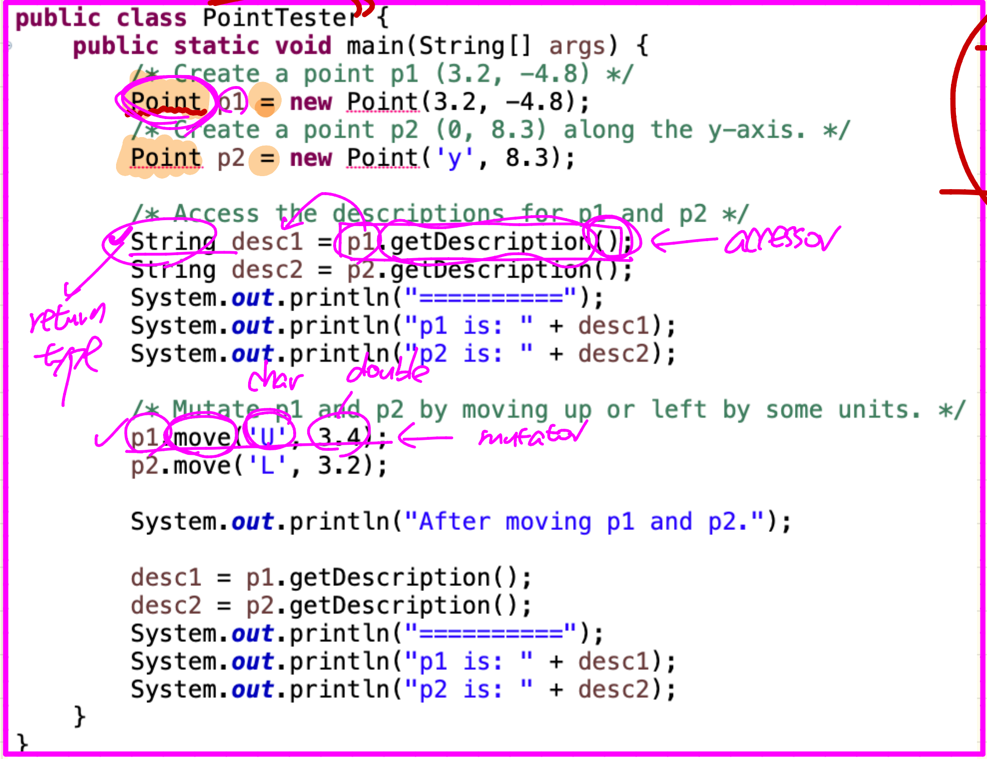


```
Point p1 = new Point(3, -4); // calling the 1st version of constructor
Point p3 = new Point('y', 8); // calling the 2nd version of constructor

System.out.println("Description of p1: " + p1.getDescription());
System.out.println("Description of p3: " + p3.getDescription());

p1.move('R', 3.2);
p3.move('D', 4.7);
System.out.println("After moving p1 and p3");

System.out.println("Description of p1: " + p1.getDescription());
System.out.println("Description of p3: " + p3.getDescription());
```

Given: Problem Description

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

Your Task:
Create missing classes



```
public class PointTester {
    public static void main(String[] args) {
        /* Create a point p1 (3.2, -4.8) */
        Point p1 = new Point(3.2, -4.8);
        /* Create a point p2 (0, 8.3) along the y-axis. */
        Point p2 = new Point('y', 8.3);

        /* Access the descriptions for p1 and p2 */
        String desc1 = p1.getDescription();
        String desc2 = p2.getDescription();
        System.out.println("==========");
        System.out.println("p1 is: " + desc1);
        System.out.println("p2 is: " + desc2);

        /* Mutate p1 and p2 by moving up or left by some units. */
        p1.move('U', 3.4);
        p2.move('L', 3.2);

        System.out.println("After moving p1 and p2.");

        desc1 = p1.getDescription();
        desc2 = p2.getDescription();
        System.out.println("==========");
        System.out.println("p1 is: " + desc1);
        System.out.println("p2 is: " + desc2);
    }
}
```

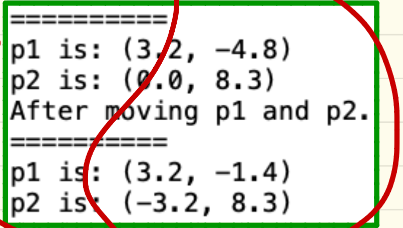
accessor

return type

char

double

mutator



```
==========
p1 is: (3.2, -4.8)
p2 is: (0.0, 8.3)
After moving p1 and p2.
==========
p1 is: (3.2, -1.4)
p2 is: (-3.2, 8.3)
```

Given: Expected Output

classes you create must not contain any System.out.println.

Given: Tester (not compiling)

Given:

- Problem description
- Tester (not compiling) → compile
- Expected output

never change the given Tester

Tasks:

- Add in missing classes
- Add in missing methods → Constructor, accessor, mutator
- Add attributes and implement methods property

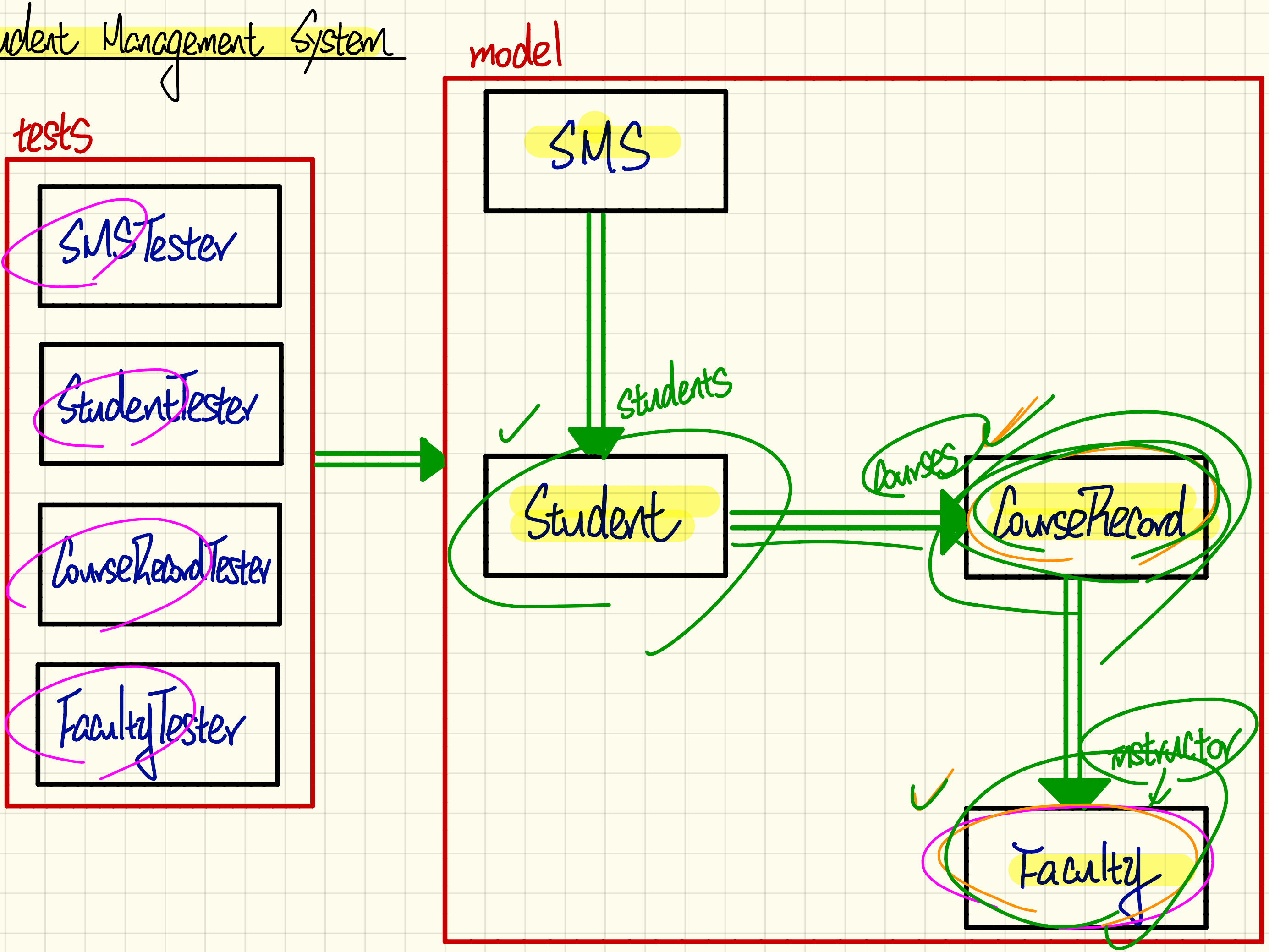
Object-Oriented Programming (OOP)

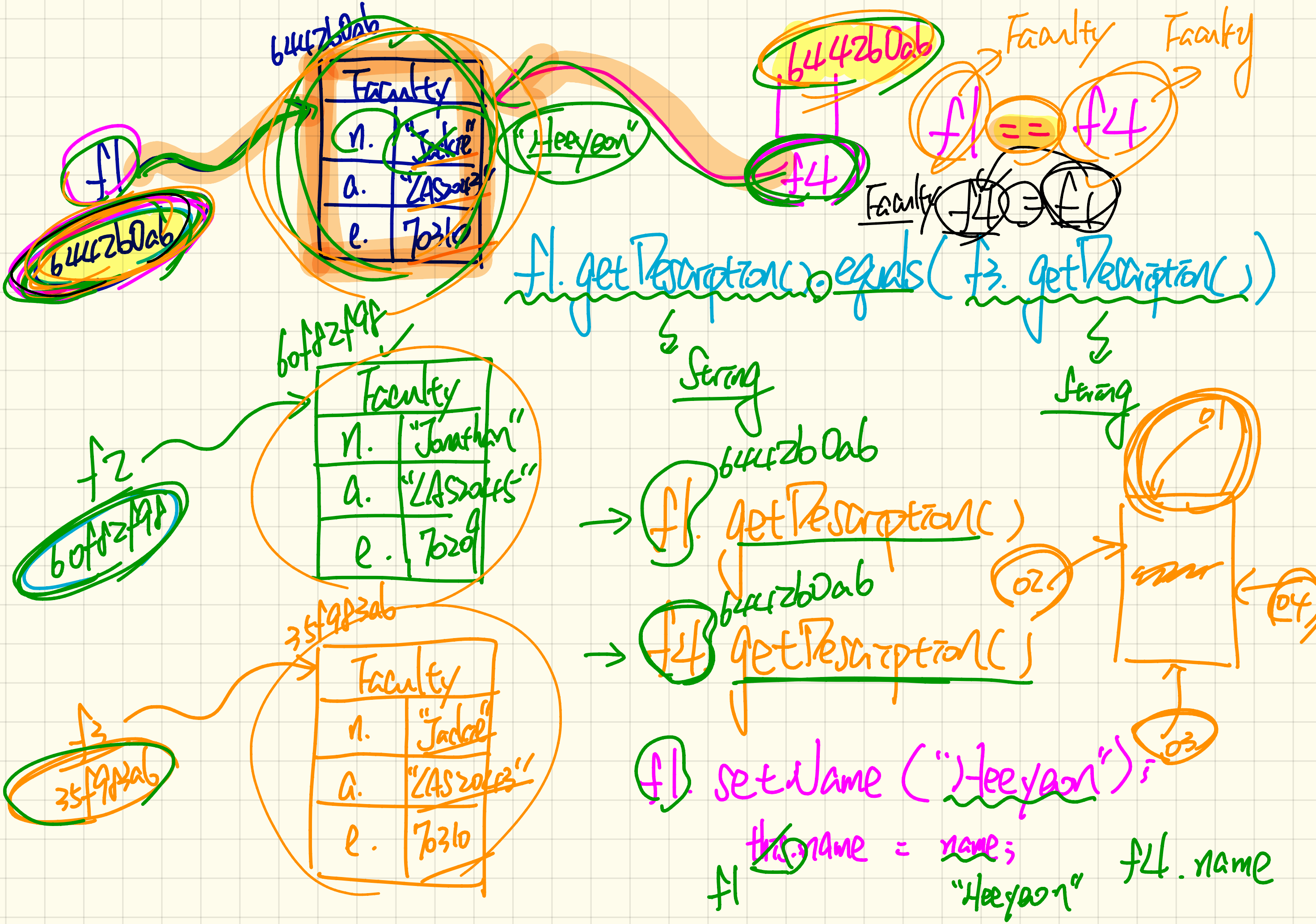
- Templates (Compile-time Java classes)
    - attributes
    - methods
        - Constructor
        - mutator
        - accessor

this

- Instances / Entities (runtime objects)
    - calling constructor to create objects
    - use of "dot notation" to
        - get attribute values
        - call accessor or mutator

Student Management System

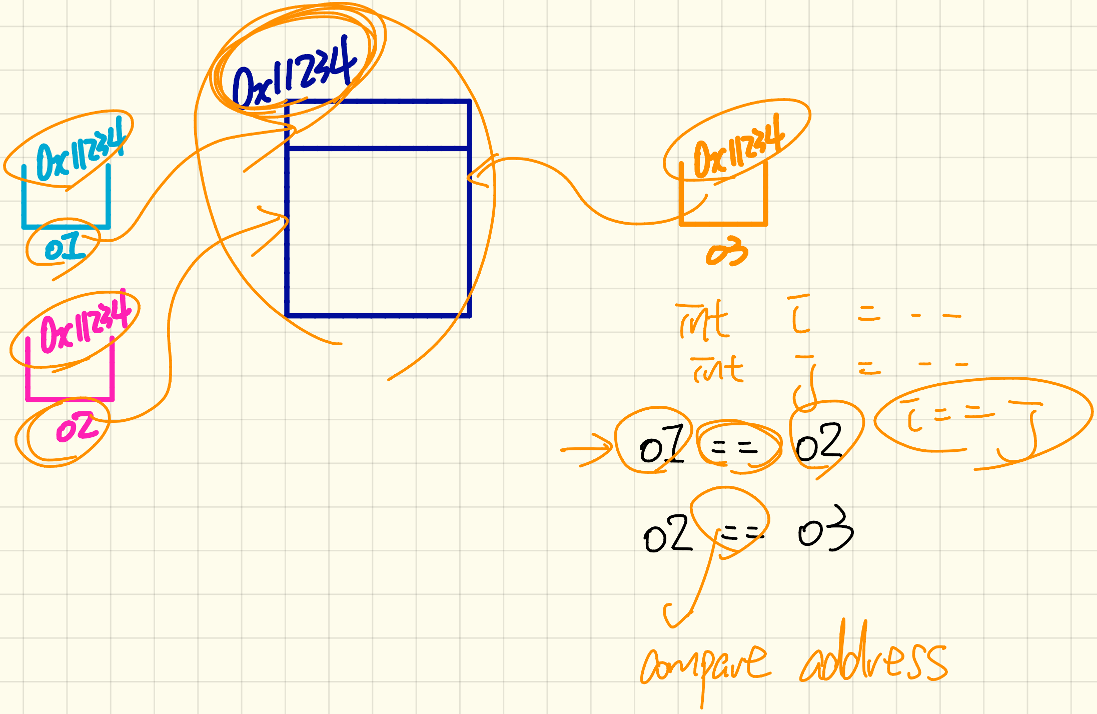




```
f1.getDescription().equals(f3.getDescription())
f1.setName("Heeyeon");
~~this~~.name = name;
```

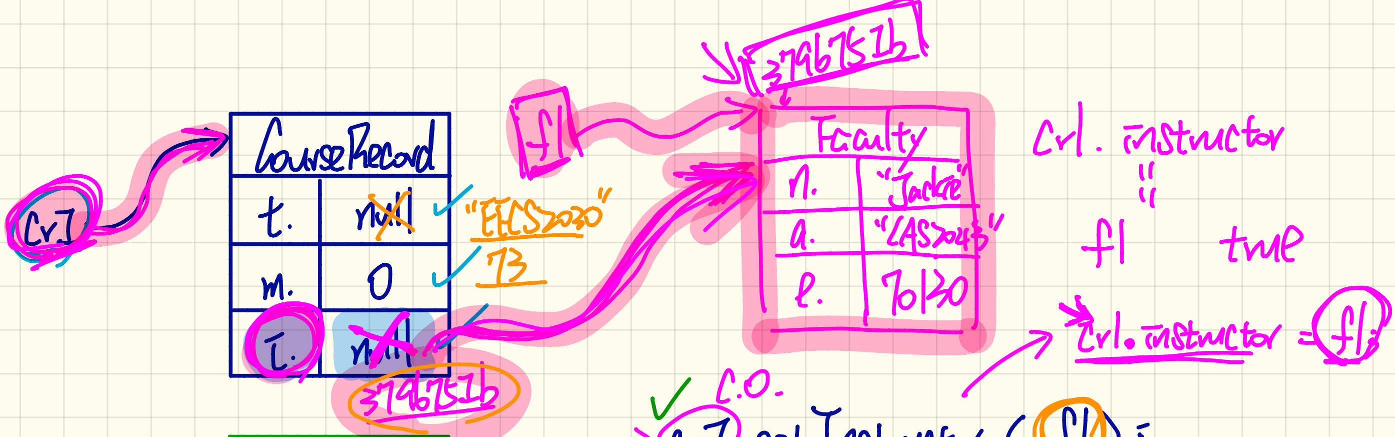
Aliasing

Multiple variables store copies of the same address.

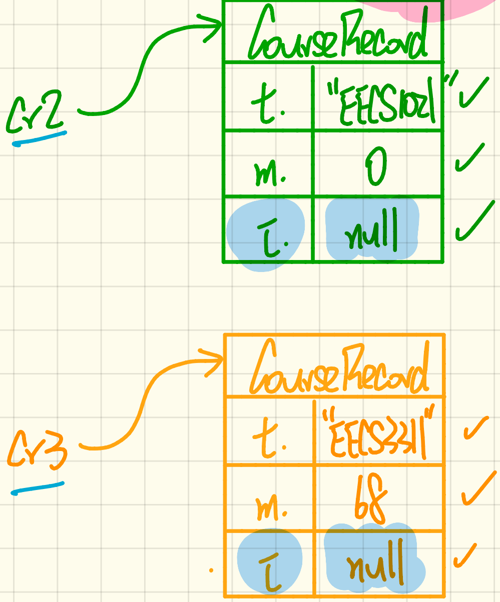


```
int i = ---
int j = ---
o1 == o2     i == j
o2 == o3
```

compare address



→ cr1.setInstructor(f1);



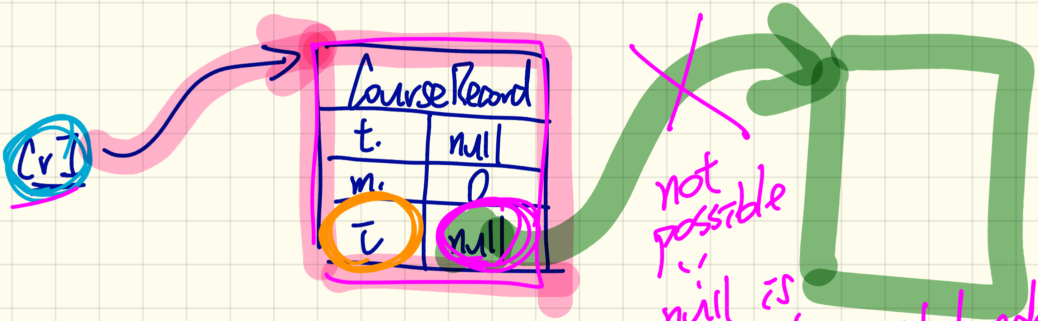
class CourseRecord {

Faculty instructor;

→ void setInstructor(Faculty ~~instructor~~ f1) {

~~this~~ cr1 .instructor = ~~instructor~~ f1;

}



not possible
∵
null is
not a valid address

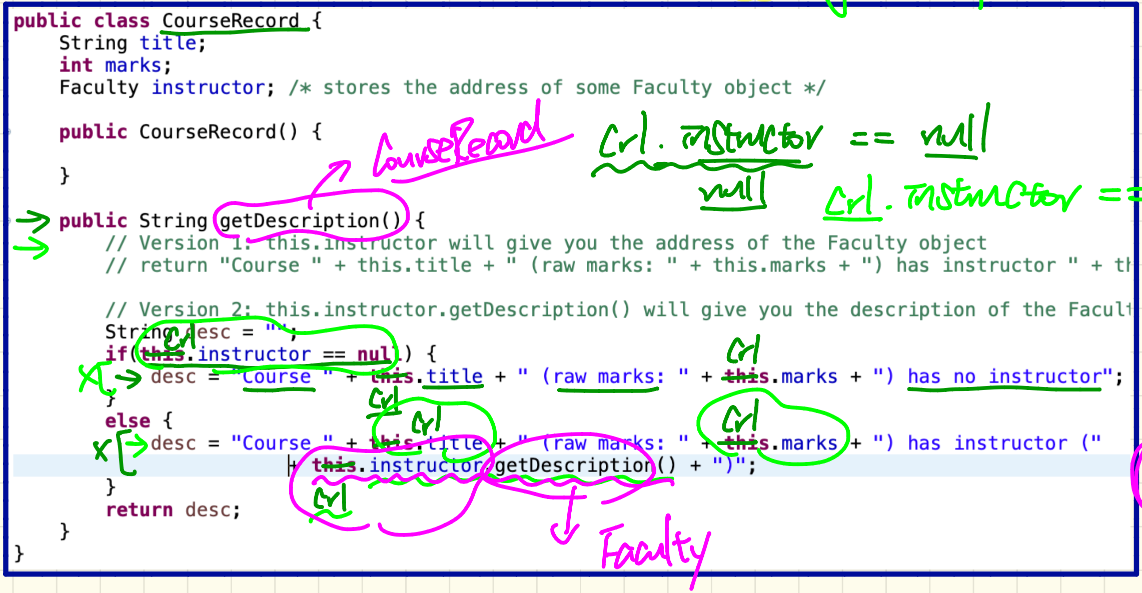
crl.instructor.getDescription()

crl.getDescription()

C.O

class CourseRecord{
    Faculty instructor;
→ String getDescription(){
        return -- ~~this~~.instructor.getDescription()
                  crl
    }
}

cr1 . instructor . getDescription ( )



```
public class CourseRecord {
    String title;
    int marks;
    Faculty instructor; /* stores the address of some Faculty object */

    public CourseRecord() {

    }

    public String getDescription() {
        // Version 1: this.instructor will give you the address of the Faculty object
        // return "Course " + this.title + " (raw marks: " + this.marks + ") has instructor " + th
        
        // Version 2: this.instructor.getDescription() will give you the description of the Facult
        String desc = "";
        if(this.instructor == null) {
            desc = "Course " + this.title + " (raw marks: " + this.marks + ") has no instructor";
        }
        else {
            desc = "Course " + this.title + " (raw marks: " + this.marks + ") has instructor ("
                    + this.instructor.getDescription() + ")";
        }
        return desc;
    }
}
```

CourseRecord

cr1 . instructor == null

null

cr1 . instructor == null

F

cr1

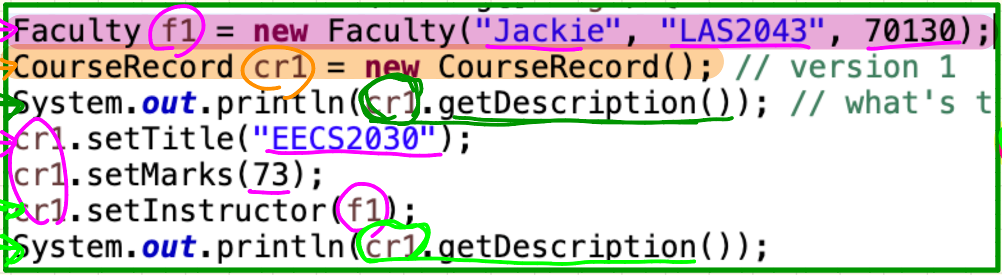
cr1

cr1

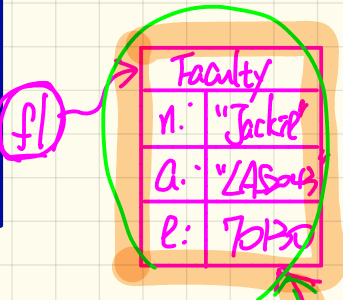
cr1

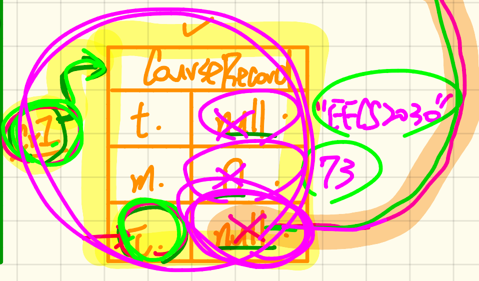
cr1

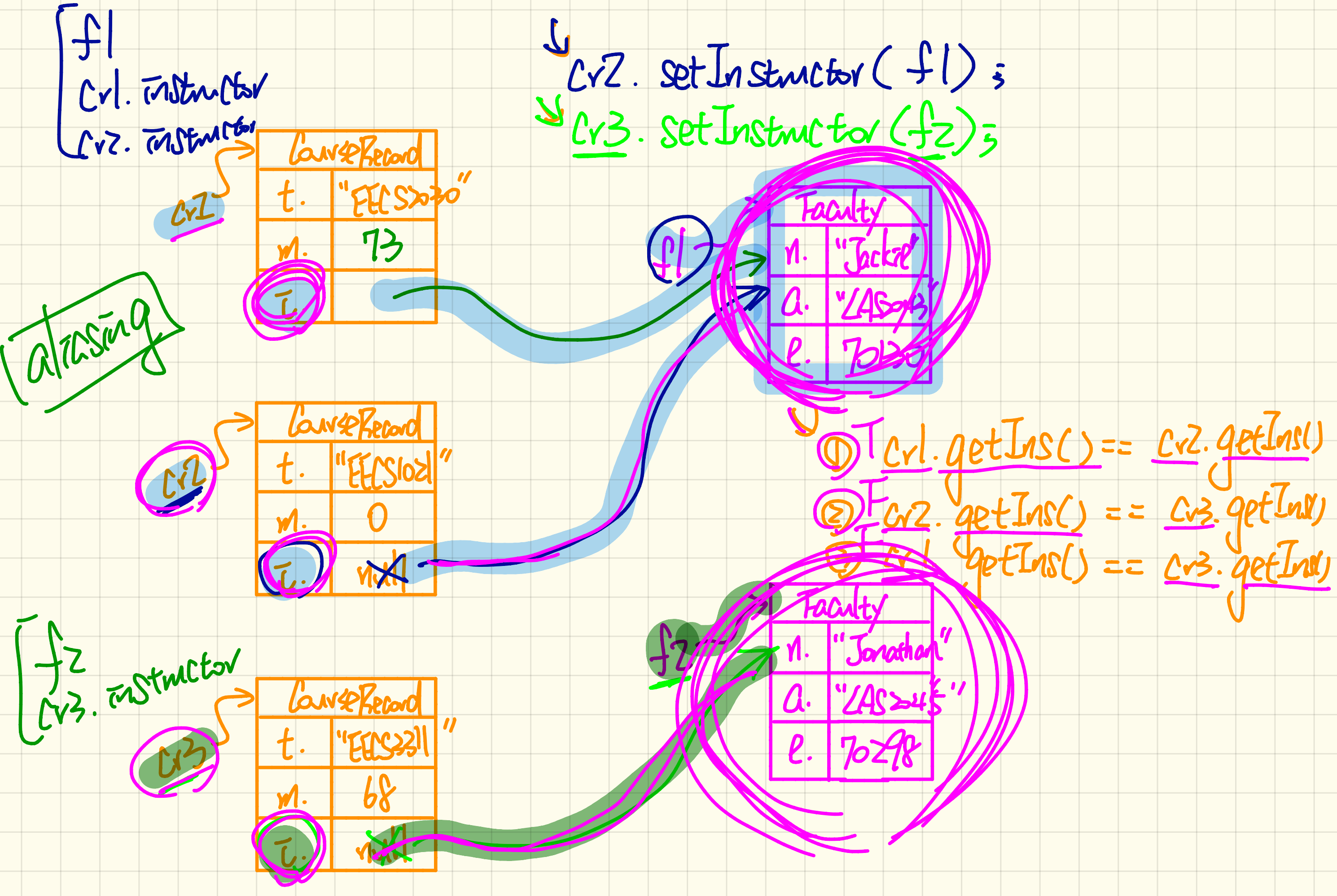
Faculty



```
Faculty f1 = new Faculty("Jackie", "LAS2043", 70130);
CourseRecord cr1 = new CourseRecord(); // version 1
System.out.println(cr1.getDescription()); // what's t
cr1.setTitle("EECS2030");
cr1.setMarks(73);
cr1.setInstructor(f1);
System.out.println(cr1.getDescription());
```







```
cr2.setInstructor(f1);
cr3.setInstructor(f2);
```

f1
cr1.instructor
cr2.instructor

cr1

CourseRecord	
t.	"EECS2030"
m.	73
i.	

aliasing

cr2

CourseRecord	
t.	"EECS1022"
m.	0
i.	~~null~~

f2
cr3.instructor

cr3

CourseRecord	
t.	"EECS3311"
m.	68
i.	~~null~~

f1

Faculty	
n.	"Jackie"
a.	"LAS2043"
e.	70130

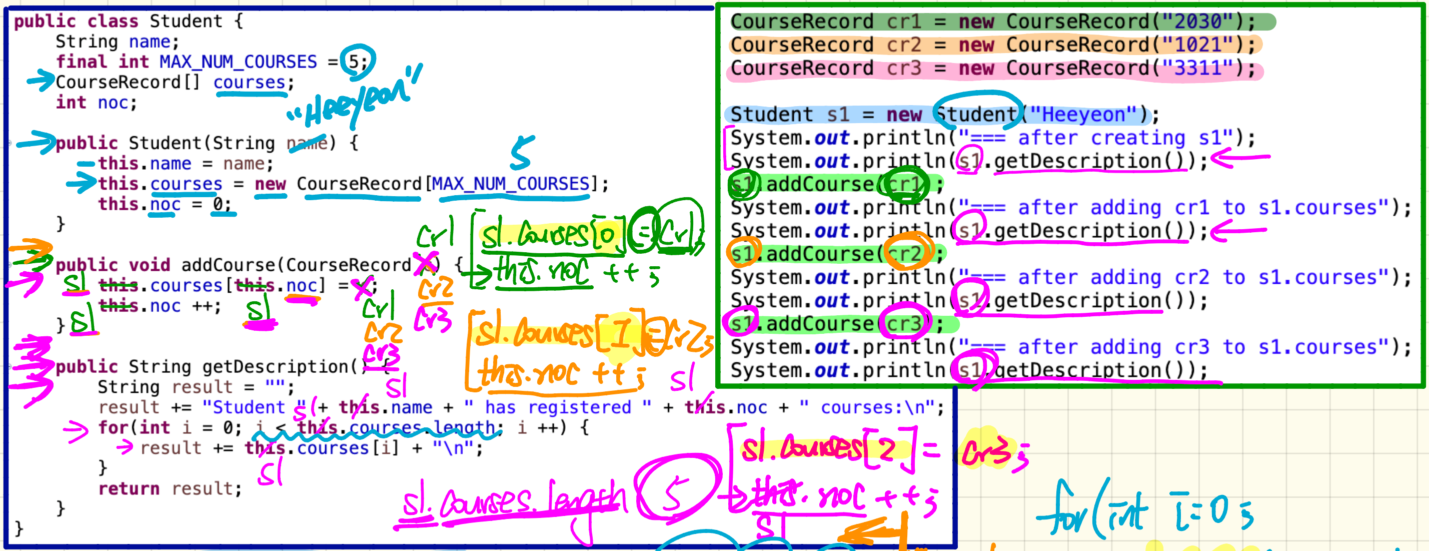
f2

Faculty	
n.	"Jonathan"
a.	"LAS2045"
e.	70298

① T cr1.getIns() == cr2.getIns()

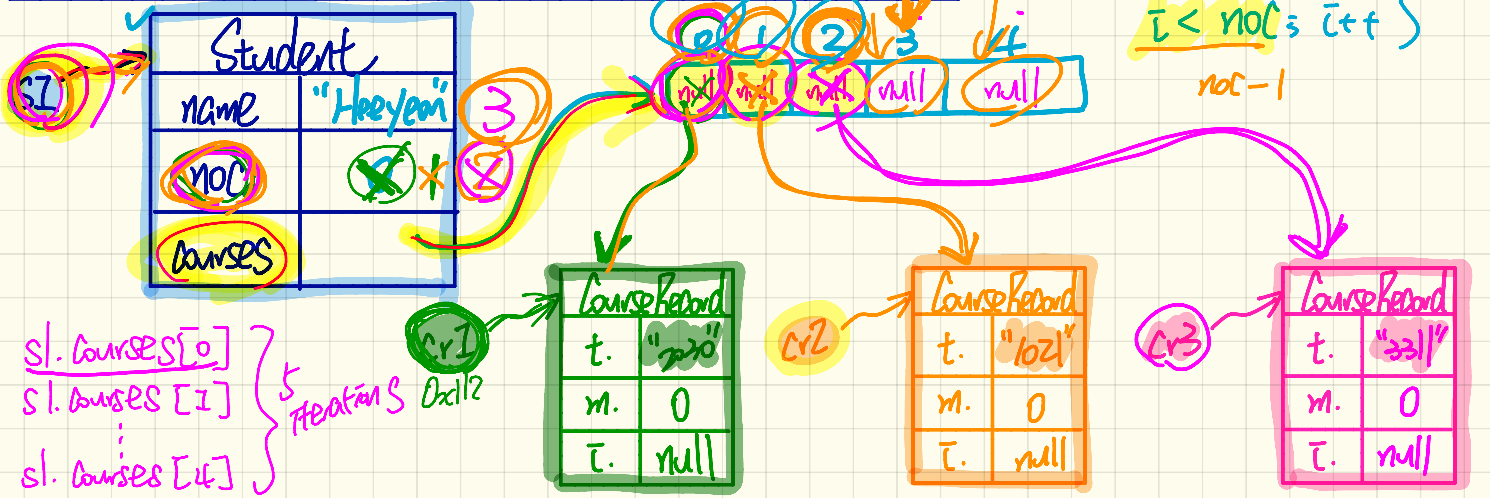
② F cr2.getIns() == cr3.getIns()

③ F cr1.getIns() == cr3.getIns()



```
public class Student {
    String name;
    final int MAX_NUM_COURSES = 5;
    CourseRecord[] courses;
    int noc;

    public Student(String name) {
        this.name = name;
        this.courses = new CourseRecord[MAX_NUM_COURSES];
        this.noc = 0;
    }

    public void addCourse(CourseRecord c) {
        this.courses[this.noc] = c;
        this.noc ++;
    }

    public String getDescription() {
        String result = "";
        result += "Student " + this.name + " has registered " + this.noc + " courses:\n";
        for(int i = 0; i < this.courses.length; i ++) {
            result += this.courses[i] + "\n";
        }
        return result;
    }
}
```

```
CourseRecord cr1 = new CourseRecord("2030");
CourseRecord cr2 = new CourseRecord("1021");
CourseRecord cr3 = new CourseRecord("3311");

Student s1 = new Student("Heeyeon");
System.out.println("=== after creating s1");
System.out.println(s1.getDescription());
s1.addCourse(cr1);
System.out.println("=== after adding cr1 to s1.courses");
System.out.println(s1.getDescription());
s1.addCourse(cr2);
System.out.println("=== after adding cr2 to s1.courses");
System.out.println(s1.getDescription());
s1.addCourse(cr3);
System.out.println("=== after adding cr3 to s1.courses");
System.out.println(s1.getDescription());
```



```
class Student {
    CourseRecord[] courses;

    String getDescription() {

        --- . this.courses[i].getDescription();

}
```

s1.getDescription

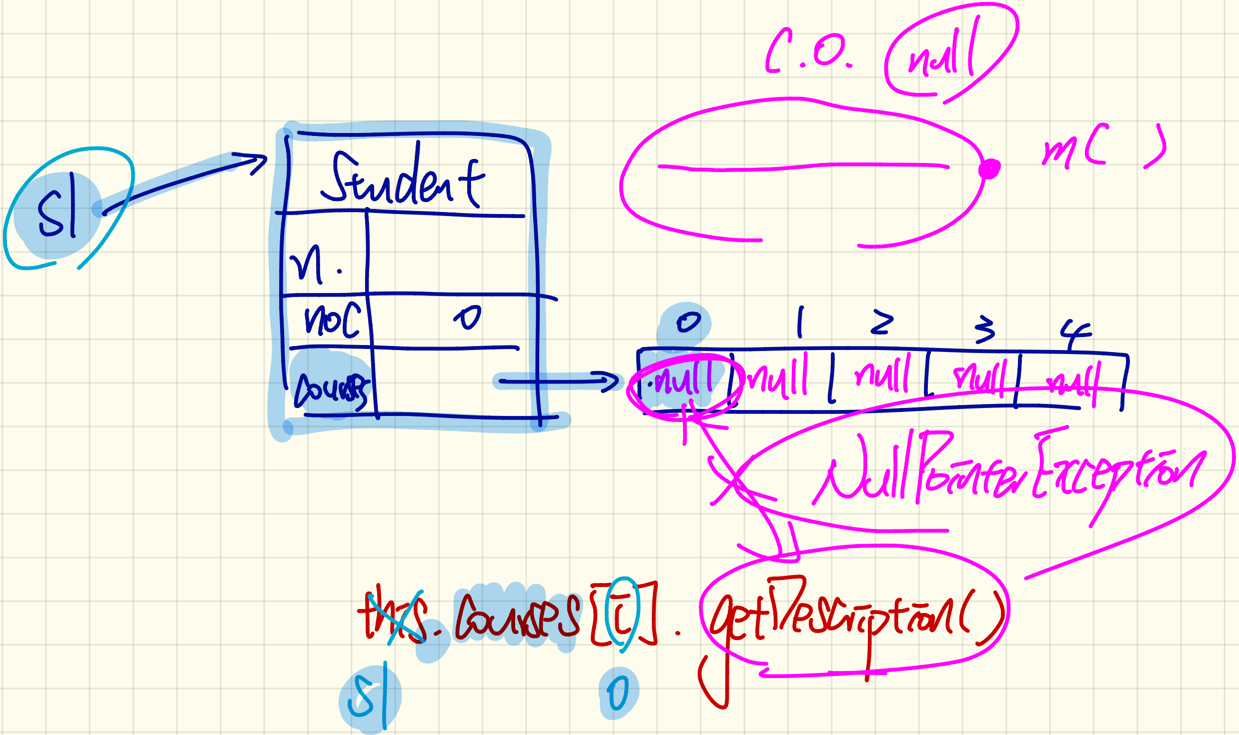
C.O. of type CourseRecord

CourseRecord[]

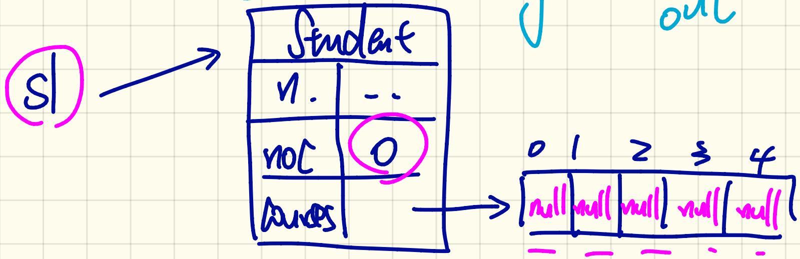
```
class CourseRecord {
    Faculty instructor;
    String getDescription() {
        --- - this.instructor.getDescription();
}
```

C.O. of type Faculty

```
class Faculty {
    n
    a
    e
    String getDescription() {
           n       a      e
}
```


s1
Student
n.
noC
0
courses
0
1
2
3
4
null
null
null
null
null
C.O.
null
m( )
NullPointerException
~~this~~.courses[i].getDescription()
s1
0

1. Empty Courses → nothing gets printed out



s1.getDescription()

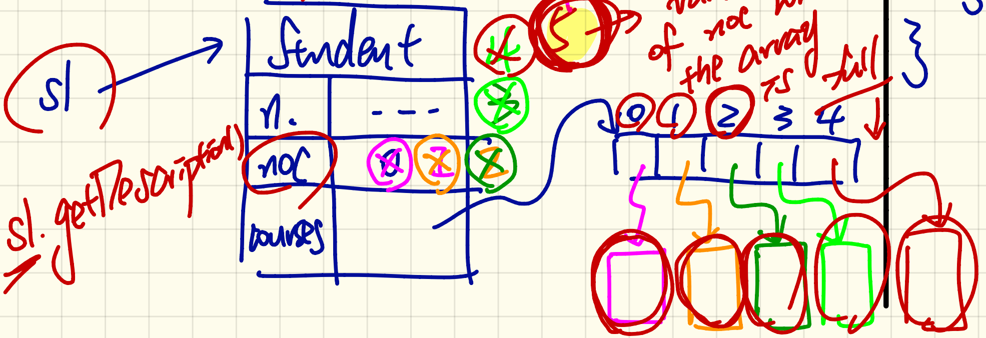
↳ no entrance to loop

↳ nothing printed.

2. Full Courses → all courses printed out

s1.courses.length

5 → value of noc when the array is full



```
class Student {

  String getDescription() {
    for(int i = 0; i < this.noc; i++) {
      this.courses[i].getDesc();
    }
  }
}
```

0 < 0 F

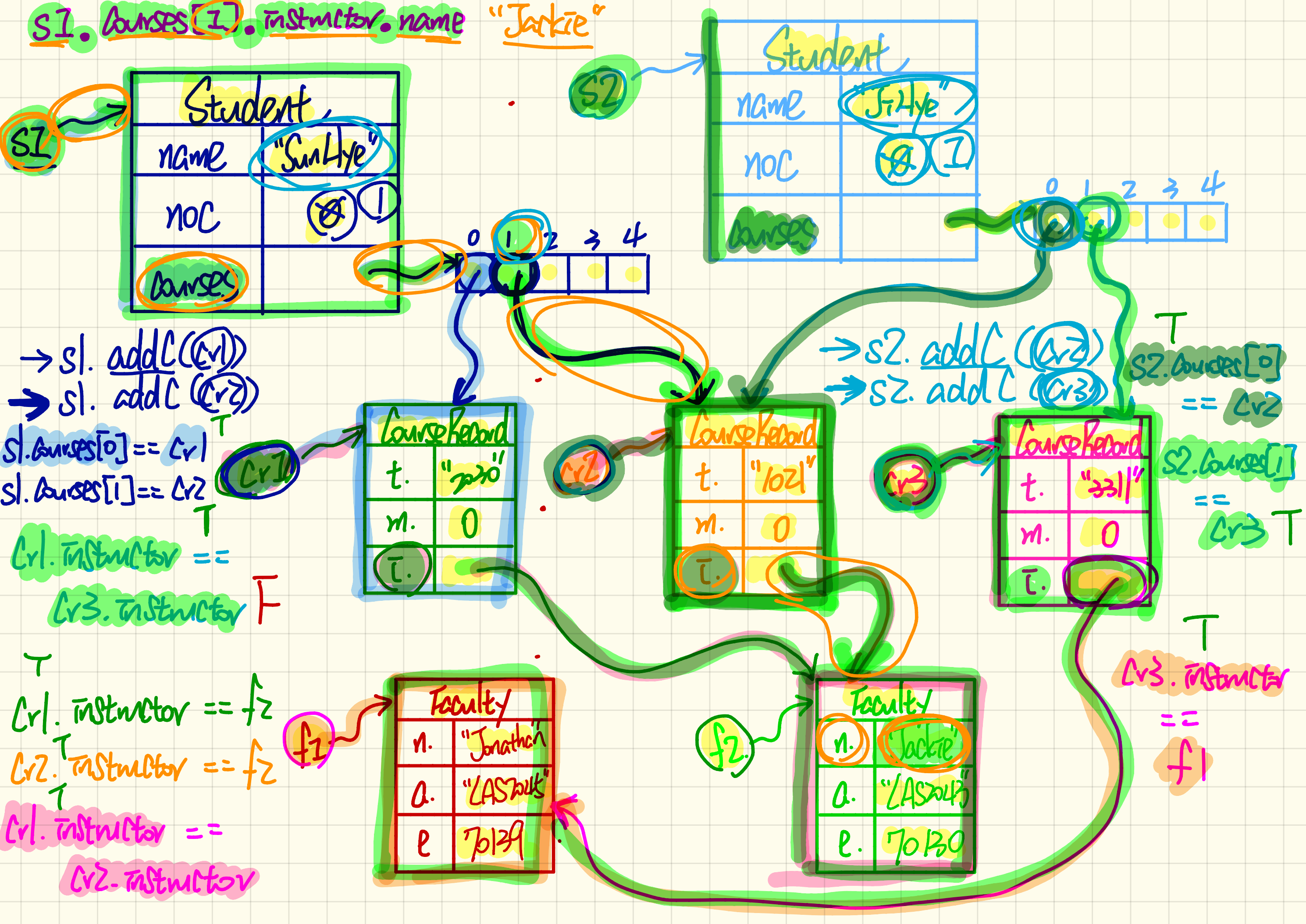
s1.noc 0

s1.noc 5

i 0 1 2 3 4 5

5 < 5 F

s1.courses[1].instructor.name "Jackie"



→ s1.addC(cr1)

→ s1.addC(cr2)

s1.courses[0] == cr1 T

s1.courses[1] == cr2

T

cr1.instructor == cr3.instructor F

T

cr1.instructor == f2

T

cr2.instructor == f2

T

cr1.instructor == cr2.instructor

→ s2.addC(cr2)

→ s2.addC(cr3)

T

s2.courses[0] == cr2

s2.courses[1] == cr3 T

T

cr3.instructor == f1

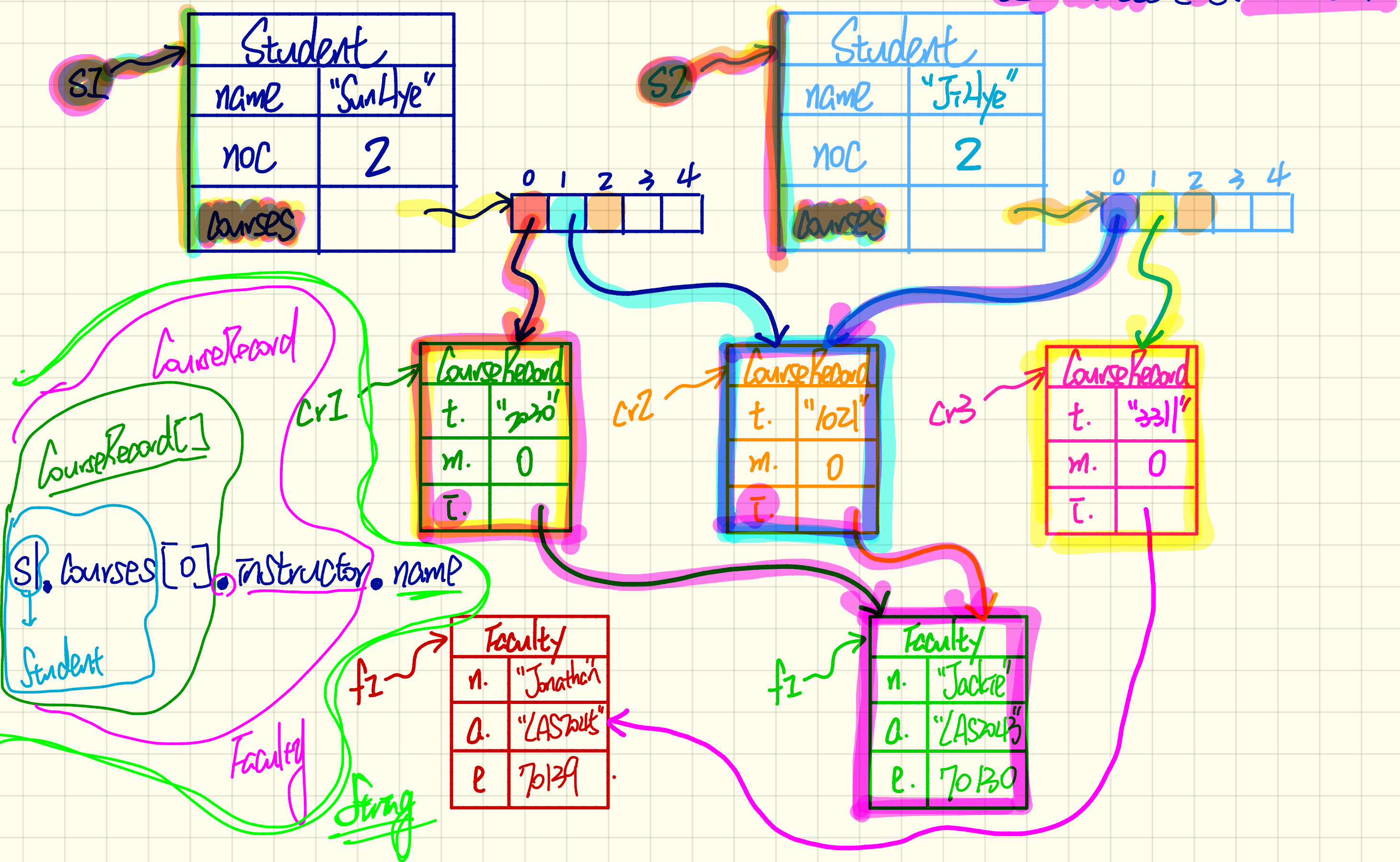
① S1. courses[0] $\overset{F}{==}$ S2. courses[1]

② S1. courses[1] $\overset{T}{==}$ S2. courses[0]

③ S1. $\overset{\text{null}}{\text{courses[2]}}$ $\overset{T}{==}$ S2. $\overset{\text{null}}{\text{courses[2]}}$

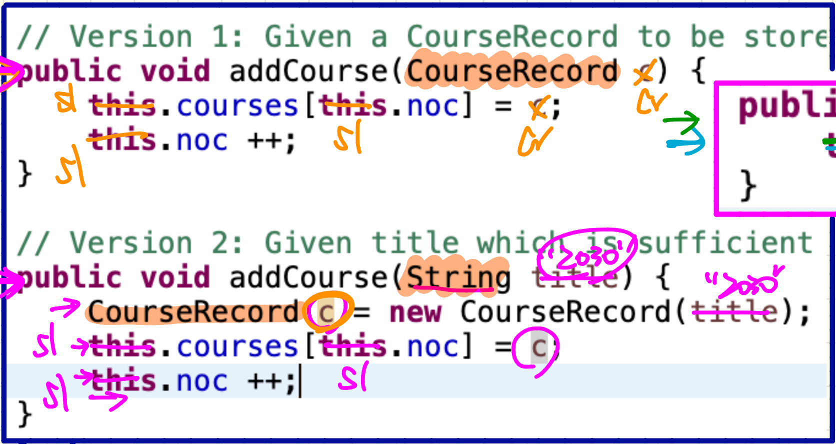
④ S1. courses[0].instructor

$\overset{T}{==}$ S2. courses[0].instructor



s1.courses[1] = 60

CourseRecord

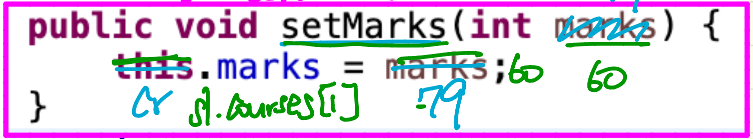


```
// Version 1: Given a CourseRecord to be store
public void addCourse(CourseRecord c) {
    this.courses[this.noc] = c;
    this.noc ++;
}

// Version 2: Given title which is sufficient
public void addCourse(String title) {
    CourseRecord c = new CourseRecord(title);
    this.courses[this.noc] = c;
    this.noc ++;
}
```

s1 cr s1 cr s1

"2030" "2030"

s1 s1 s1



```
public void setMarks(int marks) {
    this.marks = marks;
}
```

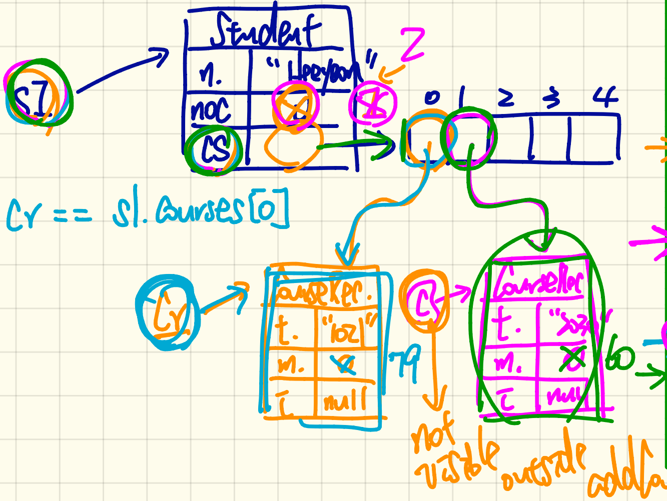
79 60 60 cr s1.courses[1] 79

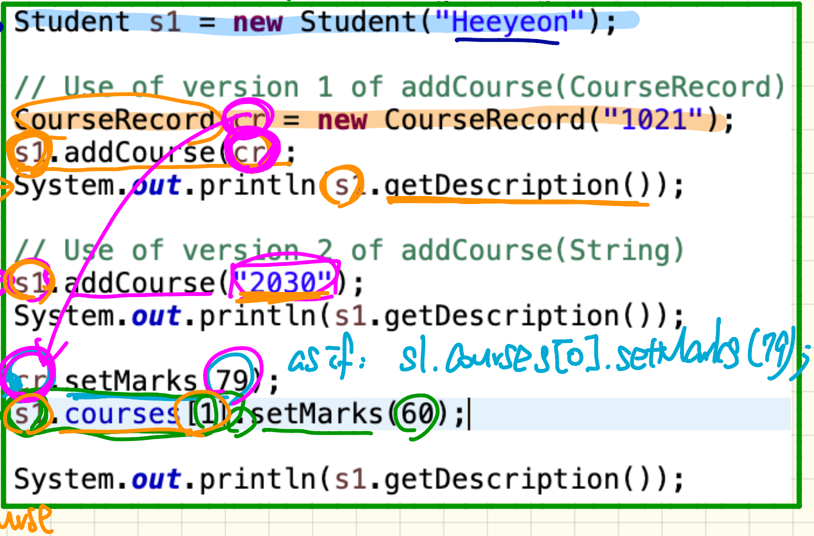
s1.courses[0] = cr;
s1.noc++;

s1.courses[1] = c;
s1.noc++;

Student.

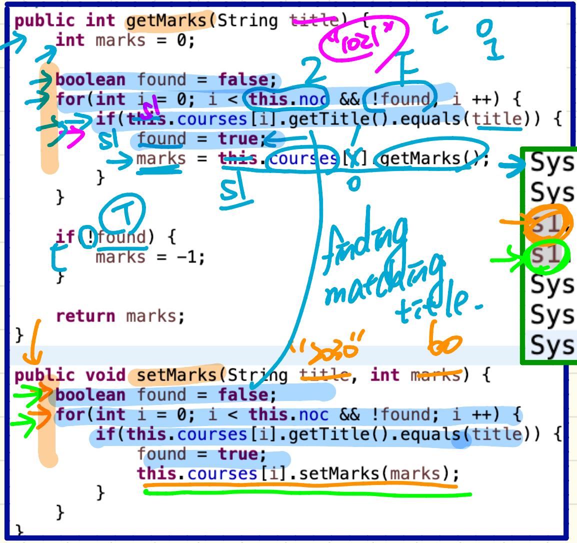
Tester





```
Student s1 = new Student("Heeyeon");

// Use of version 1 of addCourse(CourseRecord)
CourseRecord cr = new CourseRecord("1021");
s1.addCourse(cr);
System.out.println(s1.getDescription());

// Use of version 2 of addCourse(String)
s1.addCourse("2030");
System.out.println(s1.getDescription());

cr.setMarks(79);
s1.courses[1].setMarks(60);

System.out.println(s1.getDescription());
```

as if: s1.courses[0].setMarks(79);

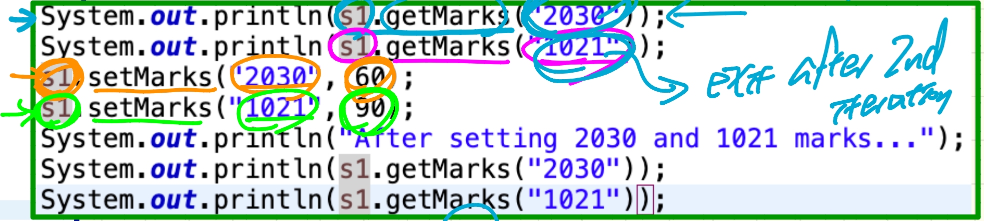


```
public int getMarks(String title) {
  int marks = 0;
  boolean found = false;
  for(int i = 0; i < this.noc && !found; i ++) {
    if(this.courses[i].getTitle().equals(title)) {
      found = true;
      marks = this.courses[i].getMarks();
    }
  }

  if(!found) {
    marks = -1;
  }

  return marks;
}

public void setMarks(String title, int marks) {
  boolean found = false;
  for(int i = 0; i < this.noc && !found; i ++) {
    if(this.courses[i].getTitle().equals(title)) {
      found = true;
      this.courses[i].setMarks(marks);
    }
  }
}
```

s1. courses[0].getTitle().equals("2030") T

"2030"

exit after 1st iteration



```
System.out.println(s1.getMarks("2030"));
System.out.println(s1.getMarks("1021"));
s1.setMarks("2030", 60);
s1.setMarks("1021", 90);
System.out.println("After setting 2030 and 1021 marks...");
System.out.println(s1.getMarks("2030"));
System.out.println(s1.getMarks("1021"));
```

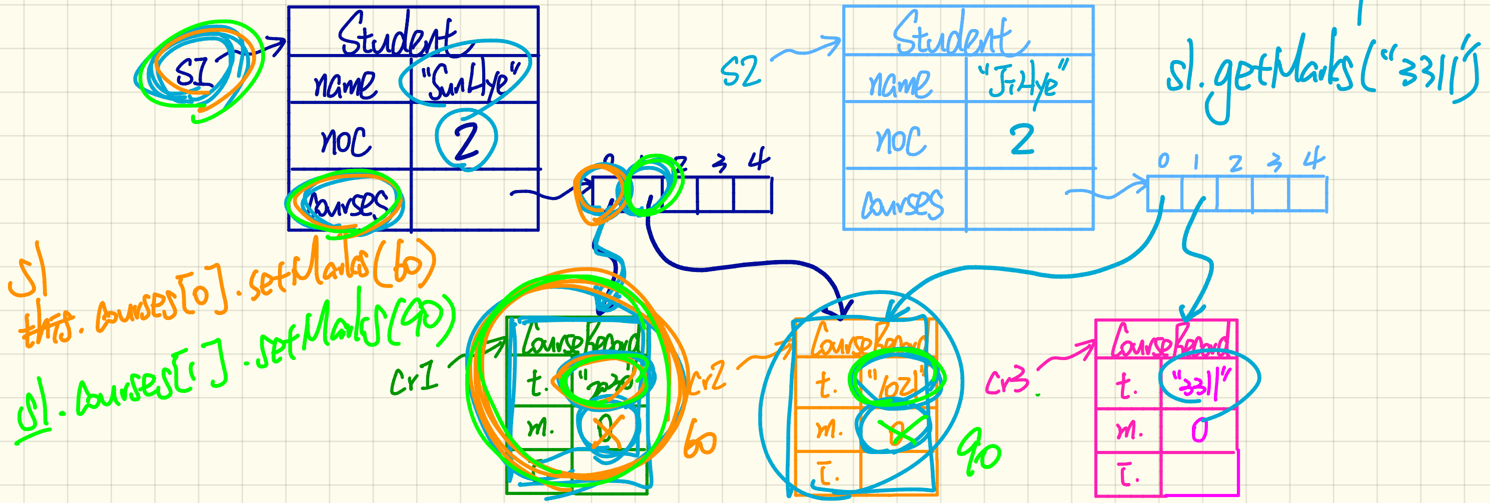
exit after 2nd iteration

s1. courses[0].getTitle().equal("1021") F

"2030" "1021"

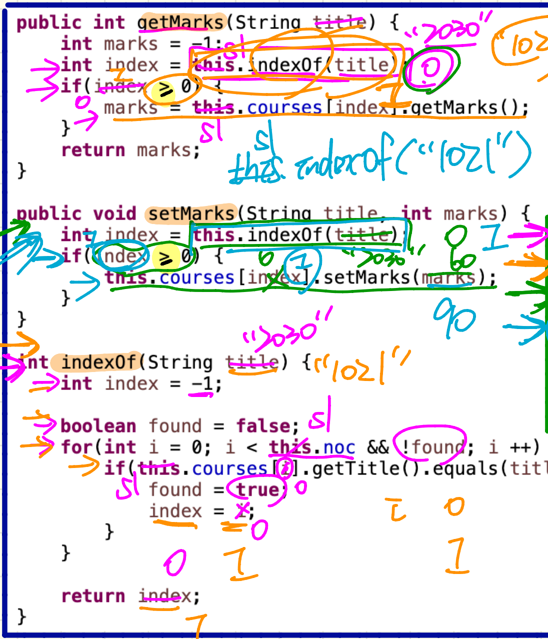
T

s1.getMarks("3311")

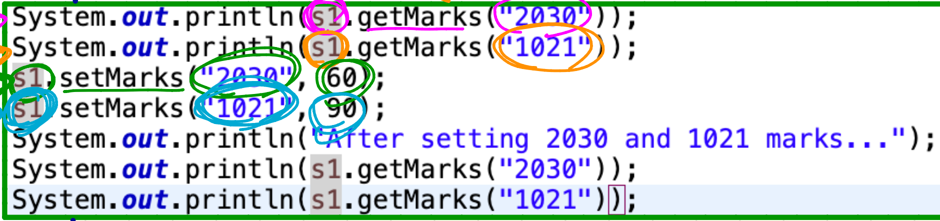


s1
this.courses[0].setMarks(60)

s1.courses[1].setMarks(90)



```
public int getMarks(String title) {
    int marks = -1;
    int index = this.indexOf(title);
    if(index >= 0) {
        marks = this.courses[index].getMarks();
    }
    return marks;
}

public void setMarks(String title, int marks) {
    int index = this.indexOf(title);
    if(index >= 0) {
        this.courses[index].setMarks(marks);
    }
}

int indexOf(String title) {
    int index = -1;

    boolean found = false;
    for(int i = 0; i < this.noc && !found; i ++) {
        if(this.courses[i].getTitle().equals(title)) {
            found = true;
            index = i;
        }
    }

    return index;
}
```



```
System.out.println(s1.getMarks("2030"));
System.out.println(s1.getMarks("1021"));
s1.setMarks("2030", 60);
s1.setMarks("1021", 90);
System.out.println("After setting 2030 and 1021 marks...");
System.out.println(s1.getMarks("2030"));
System.out.println(s1.getMarks("1021"));
```

s1.indexOf("2030")   s1.indexOf("1021")

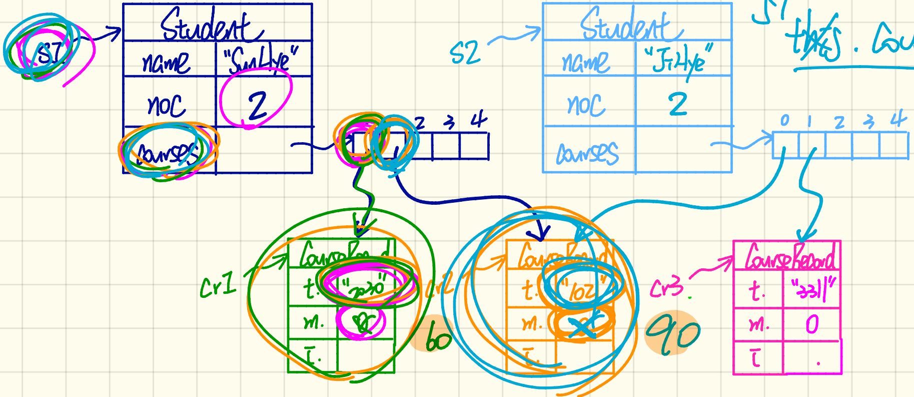
s1.courses[0].getMarks()

s1.courses[1].getMarks()

this.courses[1].getTitle().equals("1021")

s1 ~~this~~.courses[0].setMarks(60);

s1 ~~this~~.courses[1].setMarks(90)



s1. getGPA()

s1. courses[0]. getLetterGrade()

s1. courses[1]. getLG() A

s1. courses[2]. getLG() C

B

~~8~~ 0.0 ~~14~~ 21

gp

i

0

1

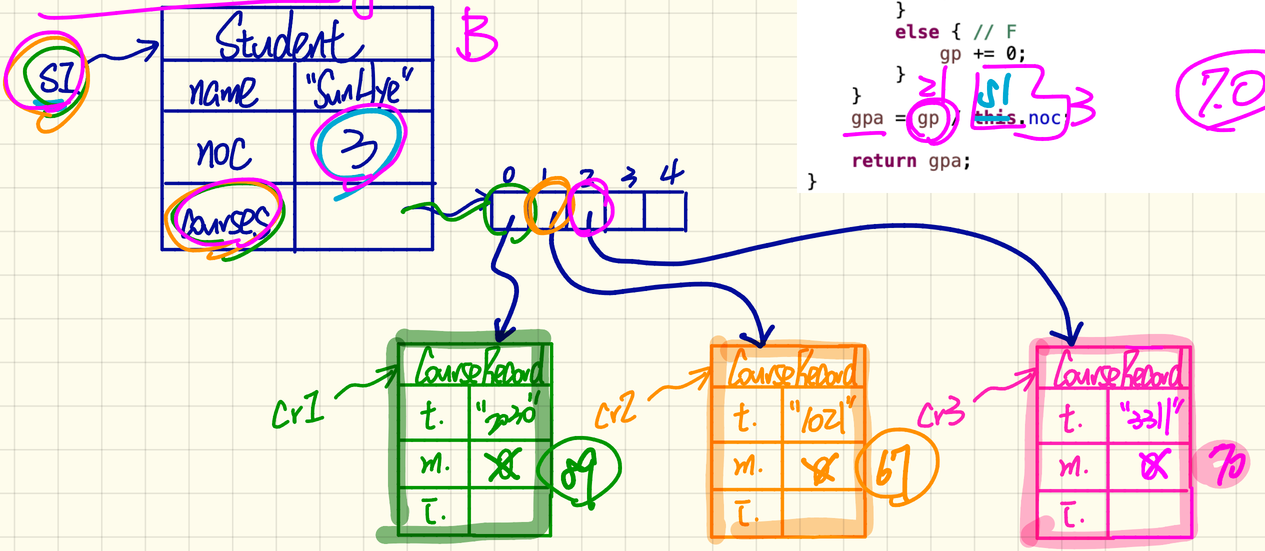
2

```
double getGPA() {
    double gpa = 0.0;
    double gp = 0.0;
    for(int i = 0; i < this.noc; i ++) {
        CourseRecord c = this.courses[i];
        String lg = c.getLetterGrade();
        String lg = this.courses[i].getLetterGrade();
        if(lg.equals("A+")) {
            gp += 9;
        }
        else if(lg.equals("A")) {
            gp += 8;
        }
        else if(lg.equals("B")) {
            gp += 7;
        }
        else if(lg.equals("C")) {
            gp += 6;
        }
        else if(lg.equals("D")) {
            gp += 5;
        }
        else { // F
            gp += 0;
        }
    }
    gpa = gp / this.noc;

    return gpa;
}
```

s1 3

21 s1 3

7.0



s1
Student
name
"SunHye"
noc
3
courses
0 1 2 3 4
cr1
CourseRecord
t.
"2030"
m.
89
i.
cr2
CourseRecord
t.
"1021"
m.
67
i.
cr3
CourseRecord
t.
"3311"
m.
70
i.