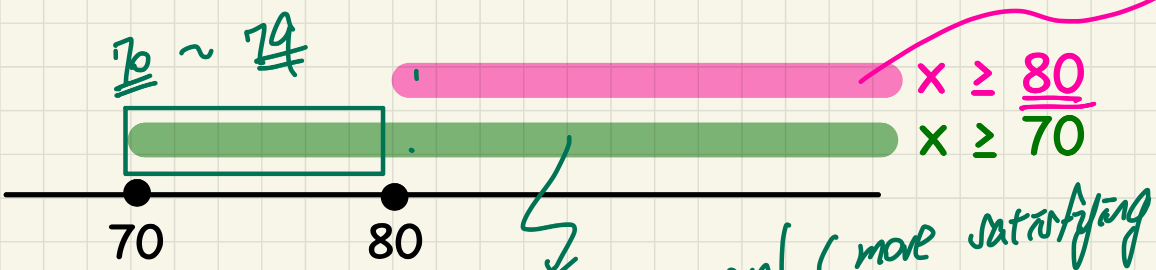
Lecture 2

Part I

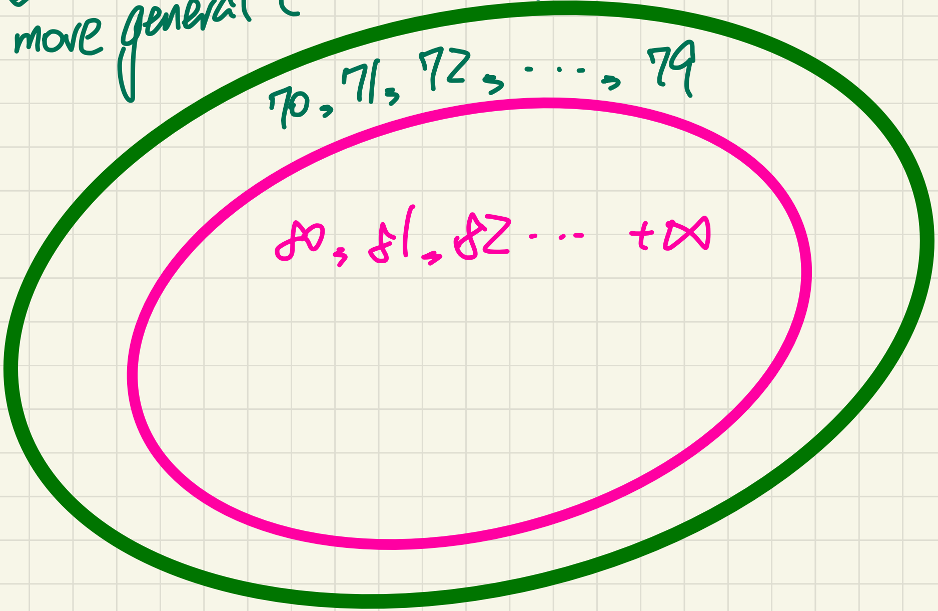
Selections -
Single If-Stmts
Conditions: General vs Specific

Overlapping Conditions: General vs. Specific



more specific (fewer satisfying values)

more general (more satisfying values)

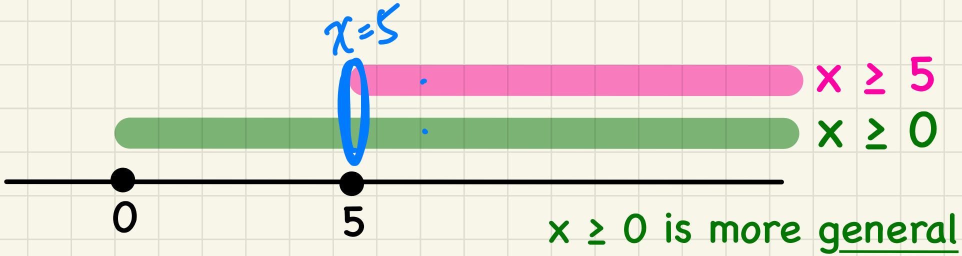


$x \geq 70$ is more general

$x \geq 80$ is more specific

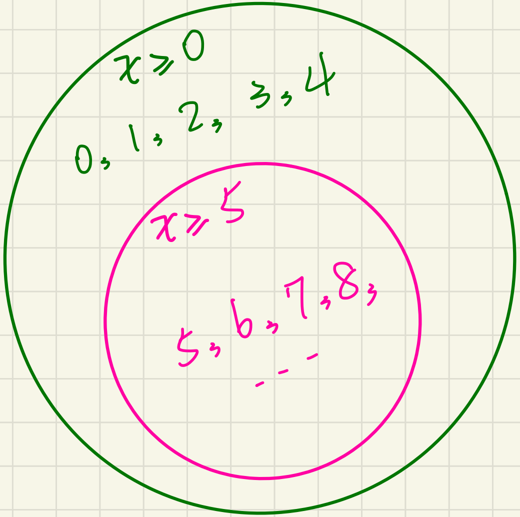
Boolean condition → set of satisfying values

Overlapping Conditions in a Single If-Statement



x ≥ 0 is more general

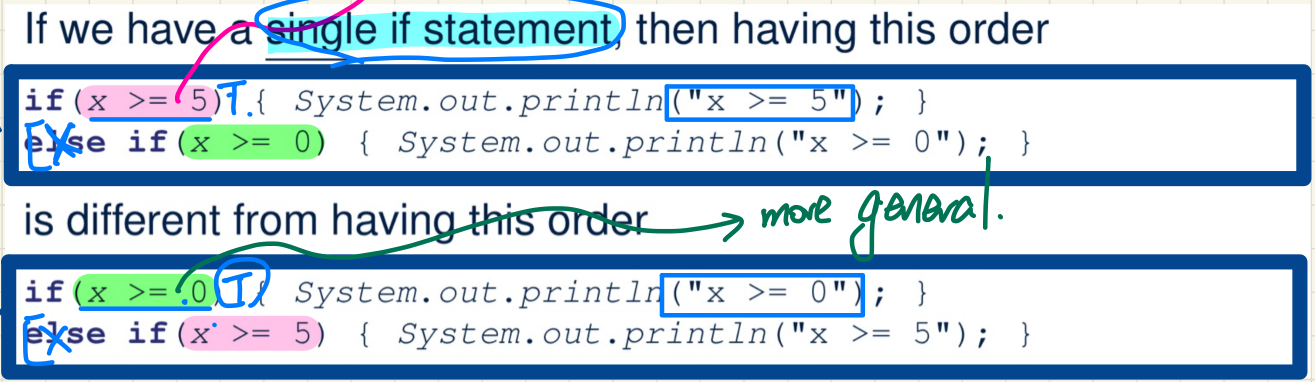
x ≥ 5 is more specific



Test Inputs:

x = 5

more specific



If we have a single if statement then having this order

v1

```
if (x >= 5) { System.out.println("x >= 5"); }
else if (x >= 0) { System.out.println("x >= 0"); }
```

T

Ex

$x >= 5$

is different from having this order

more general.

v2

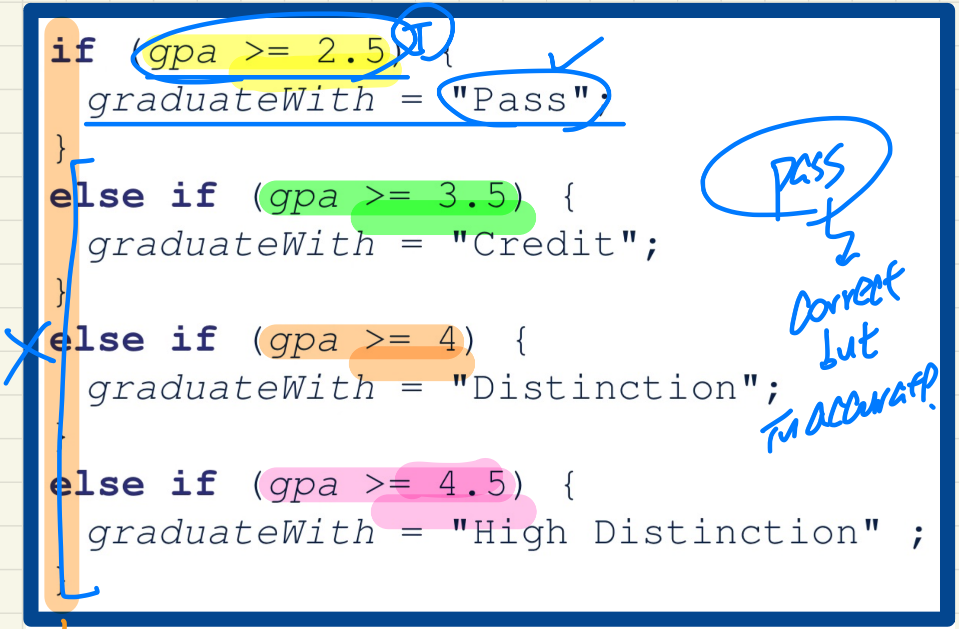
```
if (x >= 0) { System.out.println("x >= 0"); }
else if (x >= 5) { System.out.println("x >= 5"); }
```

T

Ex

$x >= 0$

Single If-Stmt with General to Specific Branching Conditions



```
if (gpa >= 2.5) {
  graduateWith = "Pass";
}
else if (gpa >= 3.5) {
  graduateWith = "Credit";
}
else if (gpa >= 4) {
  graduateWith = "Distinction";
}
else if (gpa >= 4.5) {
  graduateWith = "High Distinction" ;
}
```

T

Pass

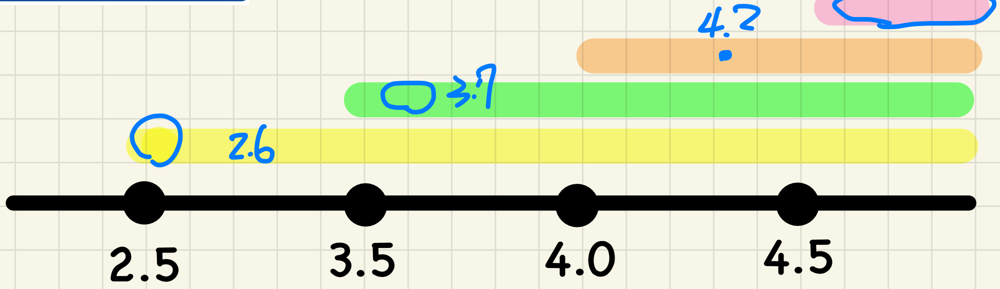
Correct but Inaccurate

single if-stmt.

branching conditions sorted from most general to most specific

Test Inputs:

gpa = 4.8



Lecture 2

Part J

Selections -
Short-Circuit Effect of && and ||

F means no need to evaluate b2.

b1 && b2

T.

T T

F F

↳ as long as one operand is false, result is F

T means no need to evaluate b2.

b1 || b2

F.

T ∴ T.

F ∴ F

↳ as long as one operand is true, result is T.

Short-Circuit Evaluation: &&

Q.* y/x > 2 && x != 0

10/0 crash!

Left Operand op1	Right Operand op2	op1 && op2
true	*true*	*true*
true	*false*	*false*
false	*true*	*false*
false	*false*	*false*

SCE would not help.

Test Inputs:

x = 0, y = 10

x = 5, y = 10

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

= 0 5

10 10

(guarding construct)

protects the division y/x

y/x > 2

≤ 2

F

✓

0 != 0 && 10/0 > 2

F

unnecessary to evaluate.

5 != 0 && 10/5 > 2

T

F

Q.* y/x>2 || x==0

Short-Circuit Evaluation: ||

Exercise Justify this version using || is equivalent to the previous version using &&.

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

Test Inputs:

x = 0 y = 10

x = 5 y = 10

```
System.out.println("Enter x:");
int x = input.nextInt();                0
System.out.println("Enter y:");
int y = input.nextInt();                10
if(x == 0 || y / x > 2)* {
  if(x == 0) {
    System.out.println("Error: Division by Zero");
  }
  else {
    System.out.println("y / x is greater than 2");
  }
}
else { /* !(x == 0 || y / x > 2) == (x != 0 && y / x <= 2) */
  System.out.println("y / x is not greater than 2");
}
```

guarding constraint

y/x > 2

y/x ≤ 2

0 == 0 || 10/0 > 2

T

not necessary to evaluate

F.

5 == 0 || 10/5 > 2

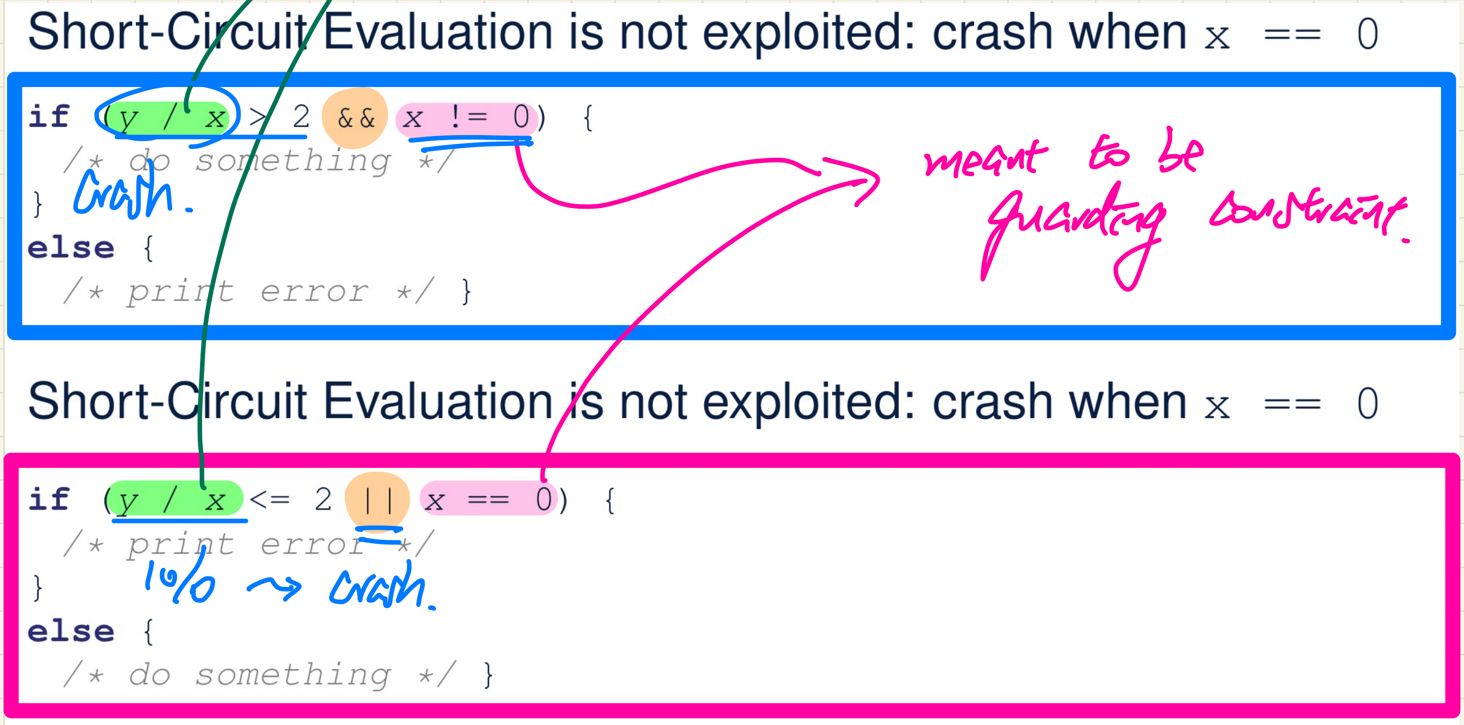
F F

Short-Circuit Evaluation: Common Errors

Test Inputs:

x = 0 y = 10

division to protect/guard.



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

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

crash.

meant to be guarding constraint.

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

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

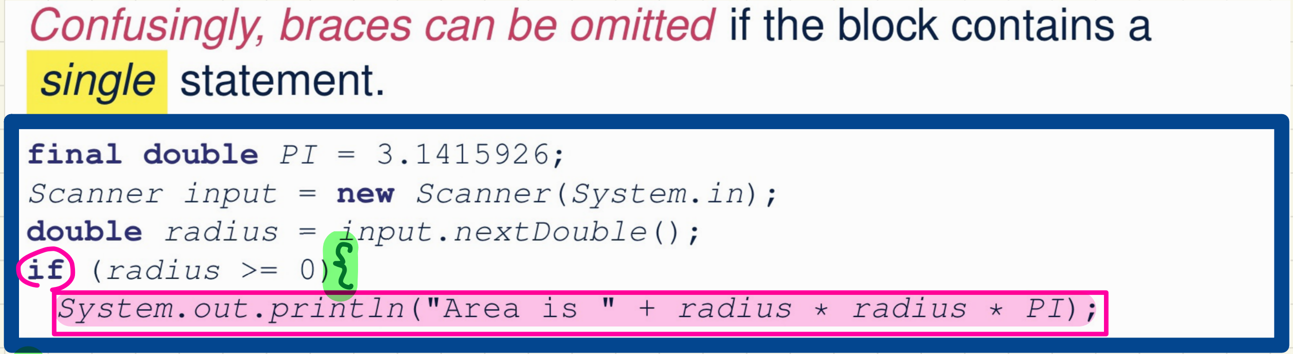
10/0 → crash.

Lecture 2

Part K

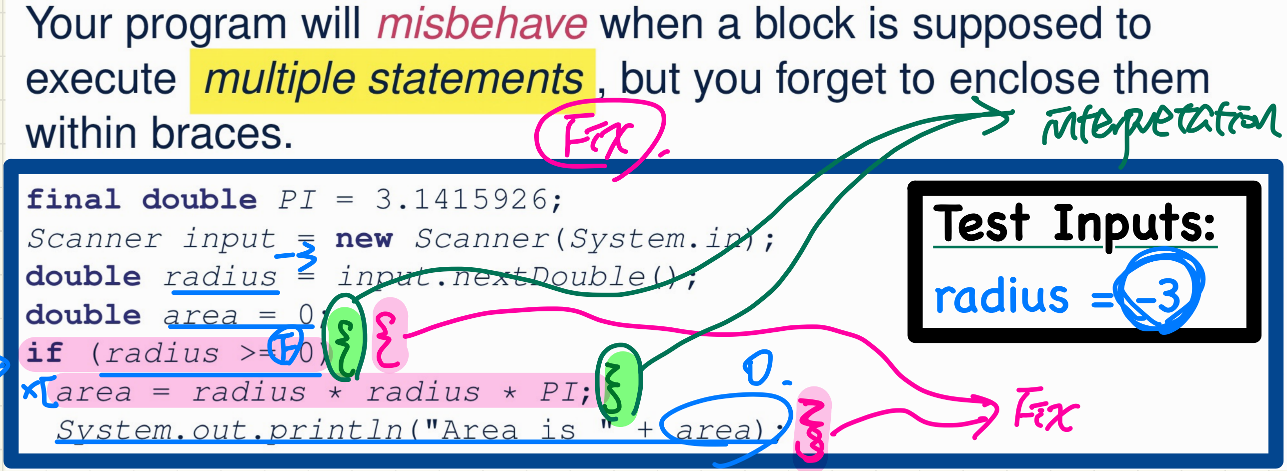
Selections -
More Common Errors and Pitfalls

Common Errors: Missing Braces



Confusingly, braces can be omitted if the block contains a *single* statement.

```
final double PI = 3.1415926;
Scanner input = new Scanner(System.in);
double radius = input.nextDouble();
if (radius >= 0){
  System.out.println("Area is " + radius * radius * PI);
}
```



Your program will *misbehave* when a block is supposed to execute *multiple statements*, but you forget to enclose them within braces.

Fix.

```
final double PI = 3.1415926;
Scanner input = new Scanner(System.in);
double radius = input.nextDouble();
double area = 0;
if (radius >= 0){
  area = radius * radius * PI;}
  System.out.println("Area is " + area);
```

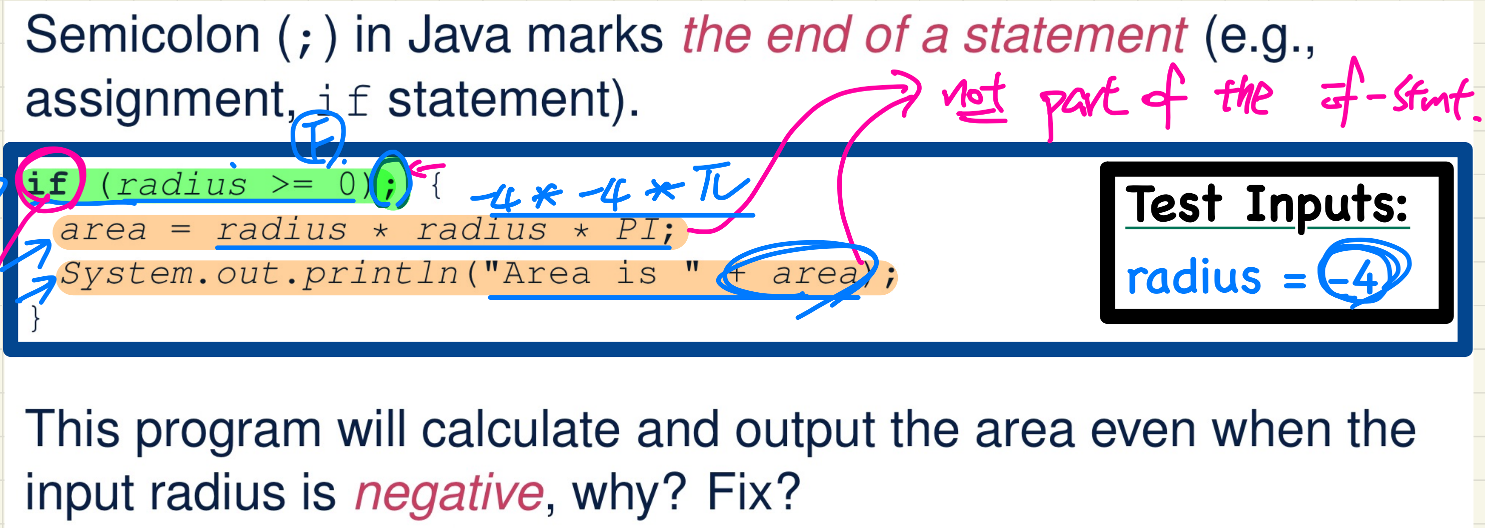
Test Inputs:

radius = -3

→ interpretation by Java compiler

if { } were missing.

Common Errors: Misplaced Semicolon



Semicolon (;) in Java marks *the end of a statement* (e.g., assignment, if statement).

F

```
if (radius >= 0); {
  area = radius * radius * PI;
  System.out.println("Area is " + area);
}
```

-4 * -4 * π

not part of the if-stmt.

Test Inputs:

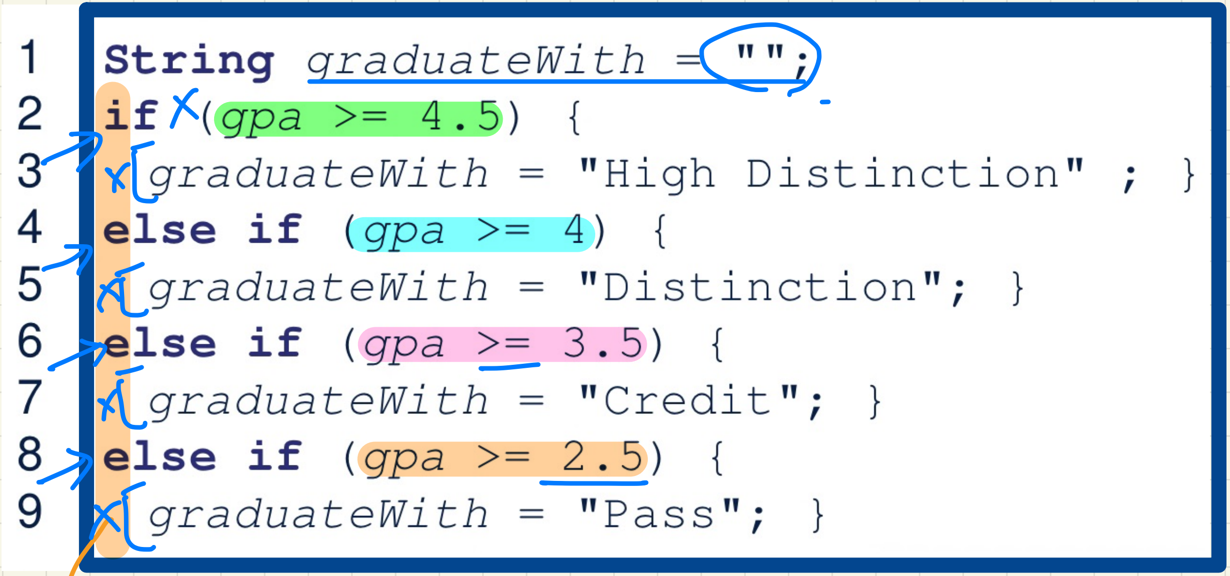
radius = -4

This program will calculate and output the area even when the input radius is *negative*, why? Fix?

≡.

```
if (radius >= 0) {
    // do nothing.
}
```

Common Errors: Variable Not Properly Re-Assigned

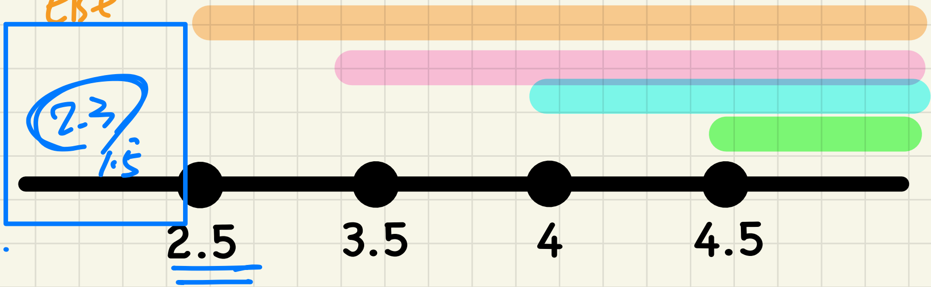


```
String graduateWith = "";
if (gpa >= 4.5) {
  graduateWith = "High Distinction" ; }
else if (gpa >= 4) {
  graduateWith = "Distinction"; }
else if (gpa >= 3.5) {
  graduateWith = "Credit"; }
else if (gpa >= 2.5) {
  graduateWith = "Pass"; }
```

Test Inputs:

gpa = 1.5

single if-stmt without an "else"



Common Errors: Ambiguous "else"

"dangling" else.

v1

```
if (x >= 0)
   if (x > 100) {
      System.out.println("x is larger than 100");
   }
else {
  System.out.println("x is negative");
}
```

T F

Test Inputs:

x = 20

v2

```
if (x >= 0)
   if (x > 100) {
      System.out.println("x is larger than 100");
   }
else {
  System.out.println("x is negative");
}
```

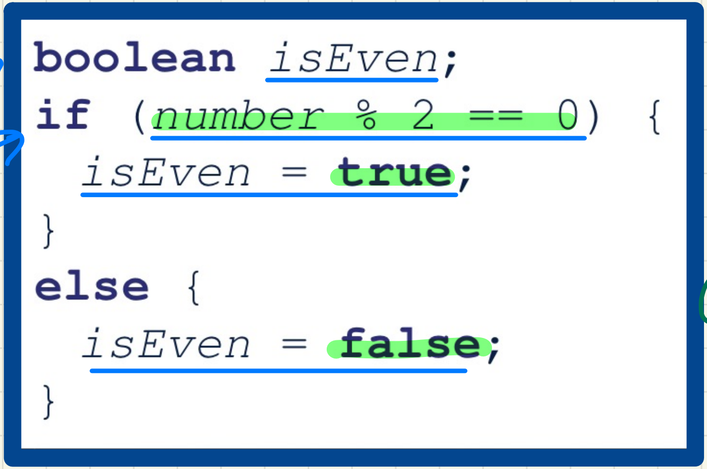
T F

Test Inputs:

x = 20

x is negative.

Common Pitfall: Simplifiable Boolean Expressions



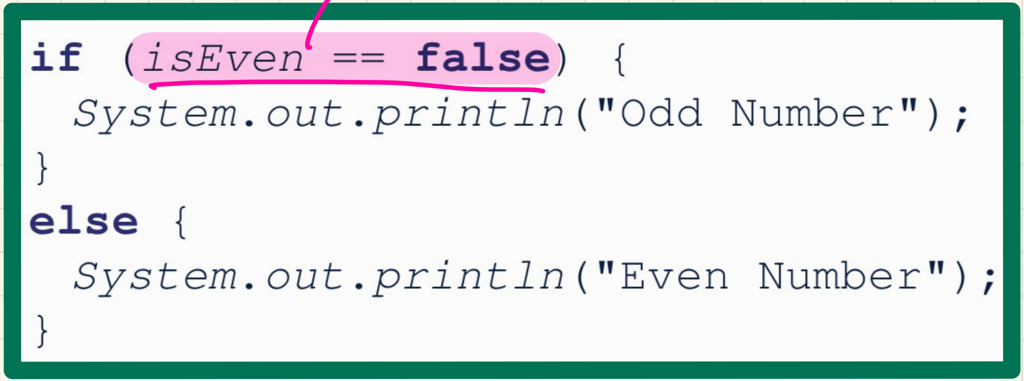
```
boolean isEven;
if (number % 2 == 0) {
  isEven = true;
}
else {
  isEven = false;
}
```

boolean isEven = number % 2 == 0;

isEven	isE == False	!isEven
T	T==F F	F
F	F==F T	T

F

! isEven



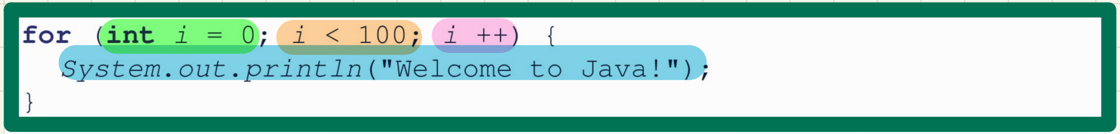
```
if (isEven == false) {
  System.out.println("Odd Number");
}
else {
  System.out.println("Even Number");
}
```

Lecture 3

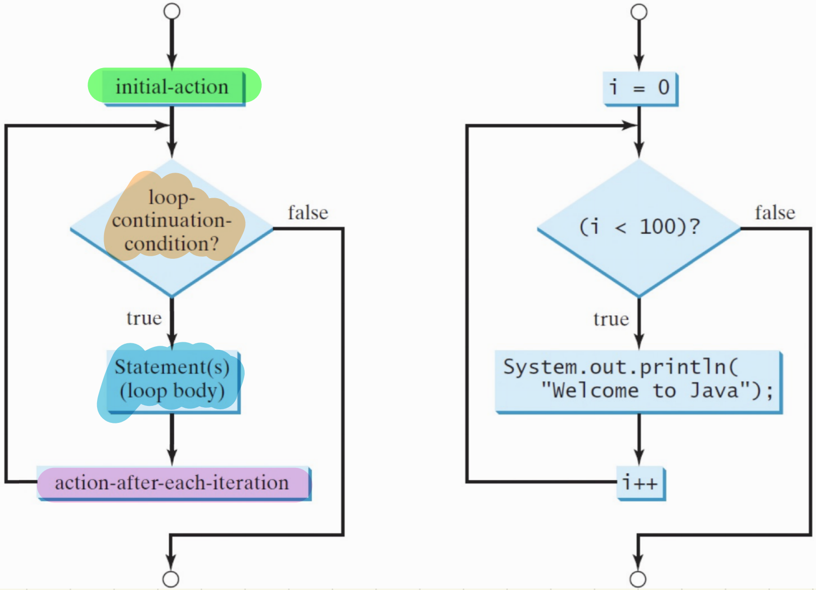
Part A

Loops -
for-Loop vs. while-Loop
Syntax and Semantics

for-Loop: Syntax and Semantics



```
for (int i = 0; i < 100; i ++) {
  System.out.println("Welcome to Java!");
}
```


initial-action
loop-continuation-condition?
false
true
Statement(s) (loop body)
action-after-each-iteration
i = 0
(i < 100)?
false
true
System.out.println("Welcome to Java");
i++

Q. How many times is the stsy condition (i < 100) checked?

Q. How many times is the loop body (println) executed?

[1, 3) ~> 1, 2   [1, 3]   1, 2, 3

for-Loop: Tracing

i < 100

(99-0)+1 ~> 109.

0 : 99

T   100   F

```
for (int i = 0; i < 100; i ++) {
  System.out.println("Welcome to Java!");
}
```

inclusive [n, m] size? m-n+1.

lower   upper

[23, 24] ~> 24-23+1 ~> 2.

i	i < 100	Enter/Stay Loop?	Iteration	Actions
0	0 < 100	*True*	1	print, i ++
1	1 < 100	*True*	2	print, i ++
2	2 < 100	*True*	3	print, i ++
. . .				
99	99 < 100	*True*	100	print, i ++
100	100 < 100	*False*	–	–

100 ~> iterations

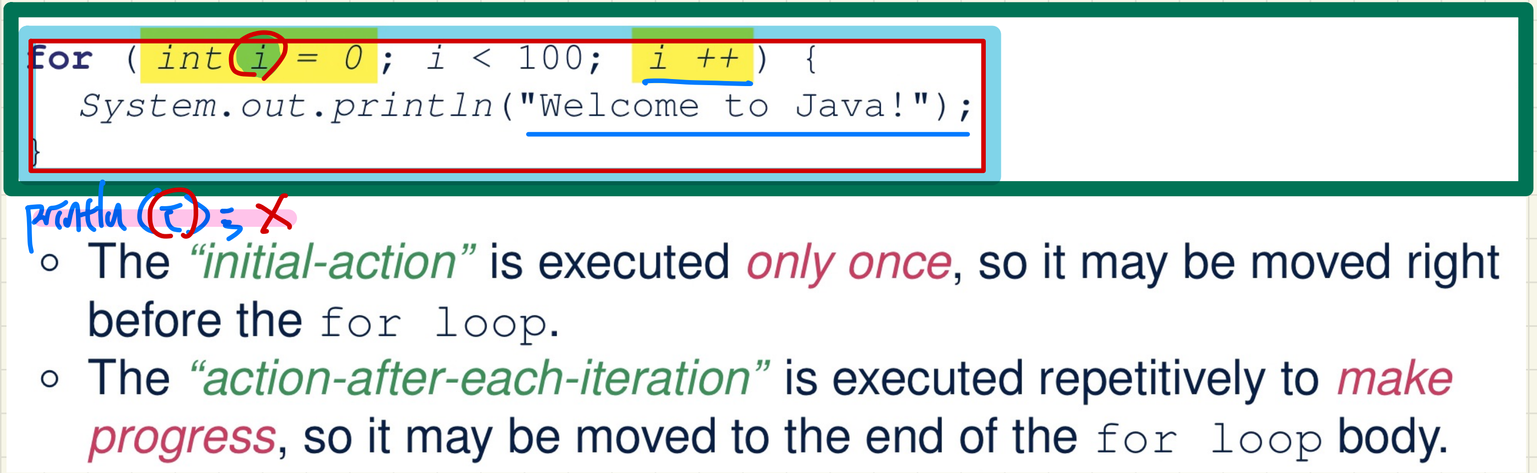
~> 1

~> no infinite loop.

Q. How many times is the stsy condition (i < 100) checked? 101

Q. How many times is the loop body (println) executed? 100

for-Loop: Alternative Syntax



```
for ( int i = 0; i < 100; i ++) {
   System.out.println("Welcome to Java!");
}
```

println(i); ✗

- The *"initial-action"* is executed *only once*, so it may be moved right before the `for loop`.
- The *"action-after-each-iteration"* is executed repetitively to *make progress*, so it may be moved to the end of the `for loop` body.

So the above for-loop may be re-written as:

```
int i=0;
for( ; i<100; ){
     println(..);
     i++;
} println(i); ✓
```

for-Loop: Exercises (1)

v1

```java
for (int count = 0; count < 100; count ++) {
  System.out.println("Welcome to Java!");
}
```

× 100

v2

```java
for (int count = 1; count < 201; count += 2) {
  System.out.println("Welcome to Java!");
}
```

↳ × 100

Q. Are the outputs same or different?

$Count = 2i - 1$

[0, 99]

100

count	count < 100	Iteration
0	T	1
1	T	2
⋮		
99	T	100
100	F	

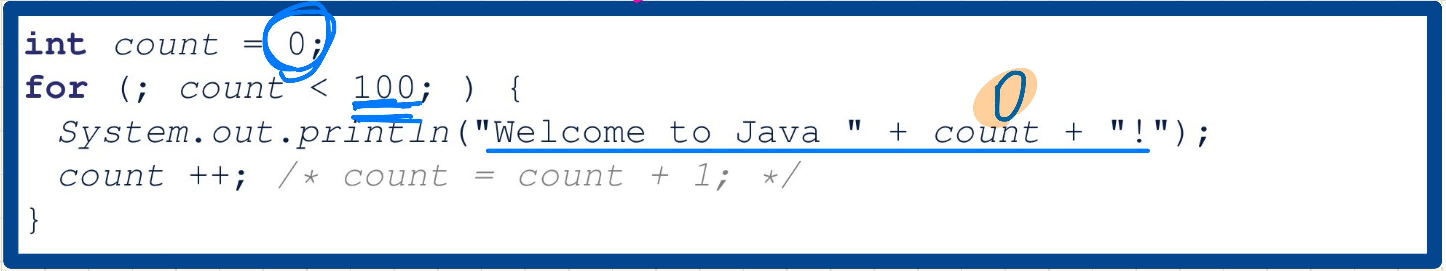
count	count < 201	Iteration (i)
1	T	1
3	T	2
5	T	3
7	T	4
⋮		
→ 199	T	100
201	F	

$199 = 2i - 1$

$i = 100$

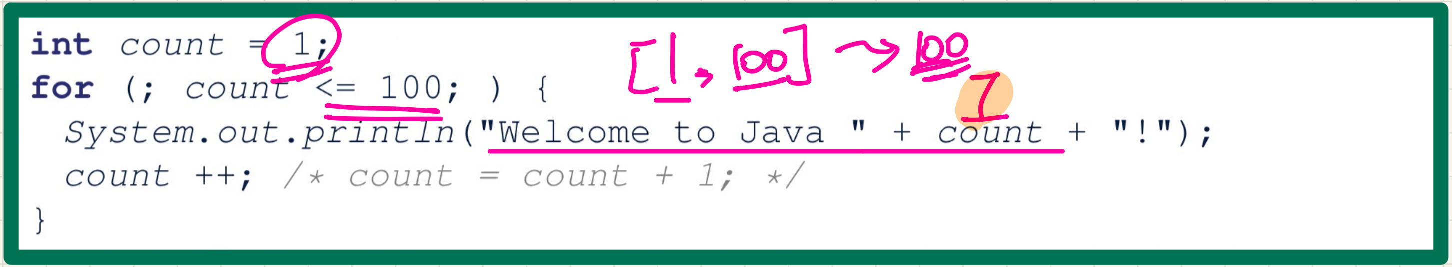
for-Loop: Exercises (2)

[0, 99] → 100



```
int count = 0;
for (; count < 100; ) {
  System.out.println("Welcome to Java " + count + "!");
  count ++; /* count = count + 1; */
}
```

0



```
int count = 1;
for (; count <= 100; ) {
  System.out.println("Welcome to Java " + count + "!");
  count ++; /* count = count + 1; */
}
```

[1, 100] → 100

1

Q. Are the outputs same or different?

for-Loop: Exercises (3)

Compare the behaviour of the following three programs:

```
for (int i = 1; i <= 5 ; i ++) {
  System.out.print(i); }
```

Output: 12345

```
int i = 1;
for ( ; i <= 5 ; ) {
  System.out.print(i);
  i ++; }
```

Output: 12345

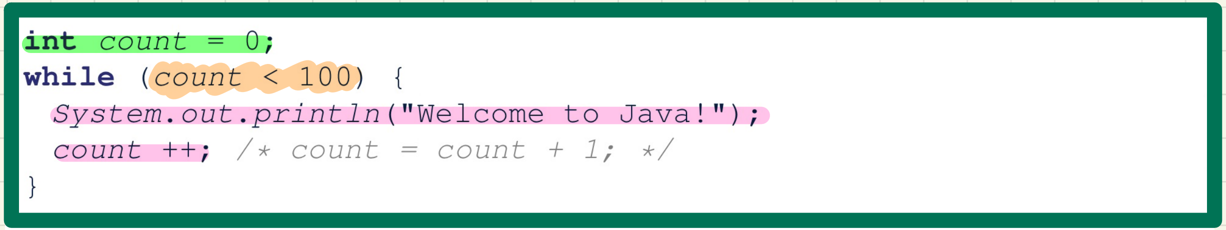
2 3 4 5 6

```
int i = 1;
for ( ; i <= 5 ; ) {
  i ++;
  System.out.print(i); }
```

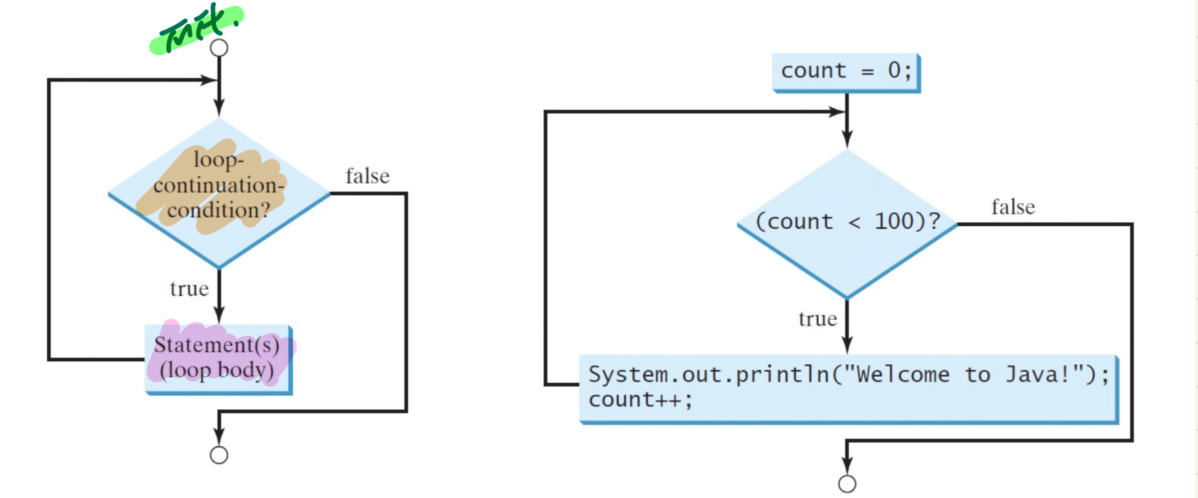
i	i<=5	i+	i++
1	T	1	2
2	T	2	3
3	T	3	4
4	T	4	5
5	T	5	6
	F		

Output: 23456

while-Loop: Syntax and Semantics



```
int count = 0;
while (count < 100) {
  System.out.println("Welcome to Java!");
  count ++; /* count = count + 1; */
}
```



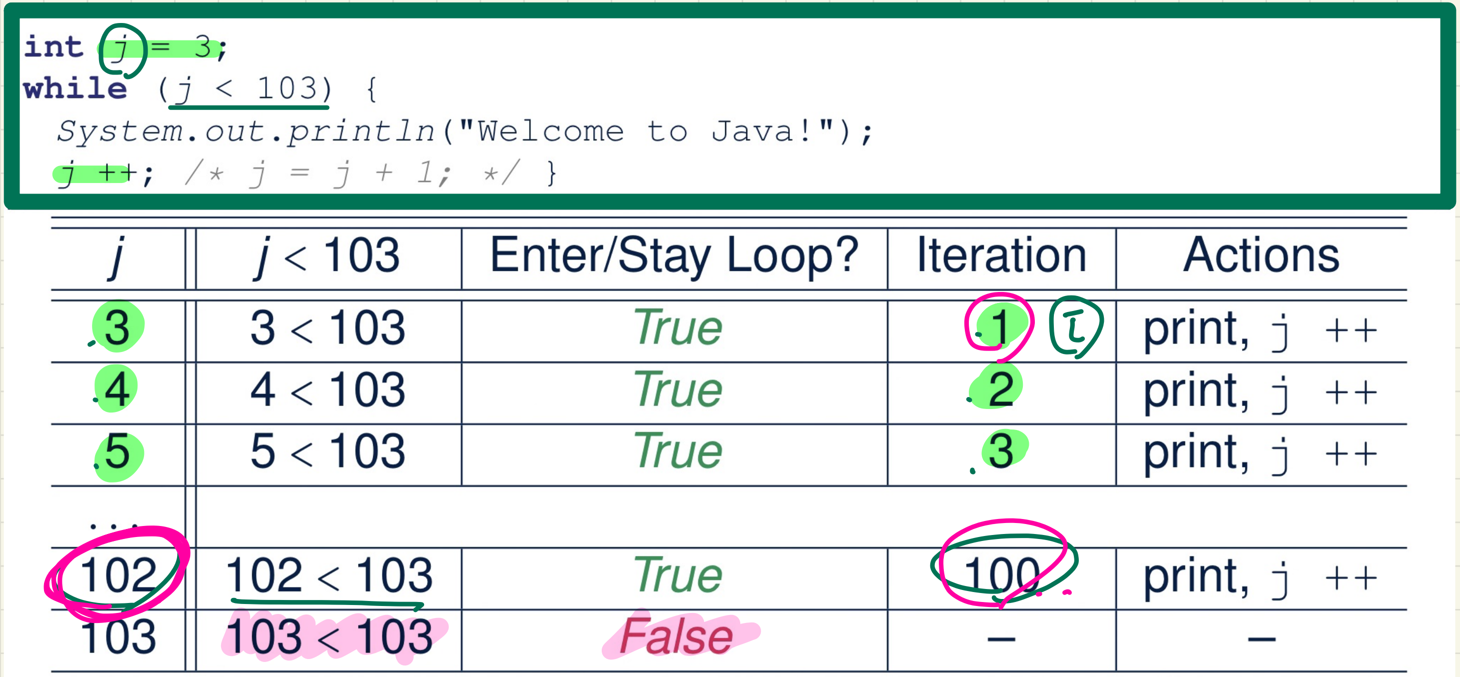
Q. How many times is the stsy condition (i < 100) checked?

Q. How many times is the loop body (println) executed?

while-Loop: Tracing

j = i+2

102 = i+2 ⇒ i=100



```
int j = 3;
while (j < 103) {
  System.out.println("Welcome to Java!");
  j ++; /* j = j + 1; */ }
```

j	$j < 103$	Enter/Stay Loop?	Iteration	Actions
3	3 < 103	*True*	1 i	print, j ++
4	4 < 103	*True*	2	print, j ++
5	5 < 103	*True*	3	print, j ++
...				
102	102 < 103	*True*	100	print, j ++
103	103 < 103	*False*	–	–

Q. How many times is the stsy condition (i < 100) checked? 101

Q. How many times is the loop body (println) executed? 100

while-Loop: Exercises (1)

```
int count = 0;
while (count < 100) {
  System.out.println("Welcome to Java!");
  count ++; /* count = count + 1; */
}
```

[0, 99] ~> 100

```
int count = 1;
while (count <= 100) {
  System.out.println("Welcome to Java!");
  count ++; /* count = count + 1; */
}
```

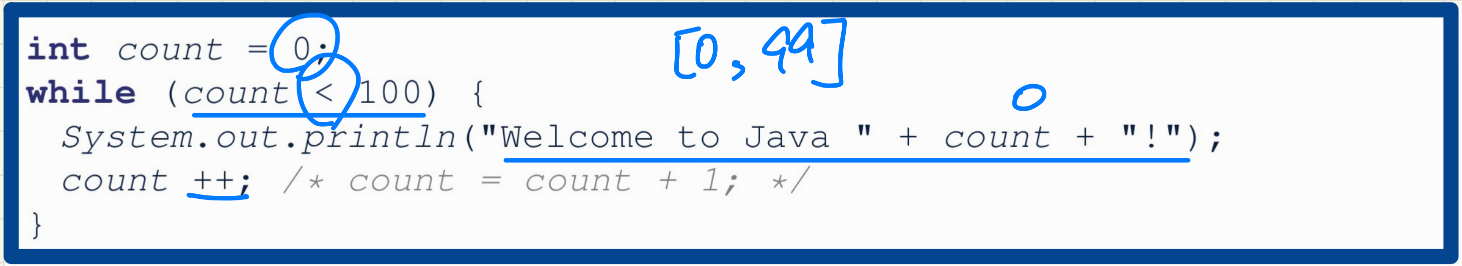
[1, 100] ~> 100

Q. Are the outputs same or different?

count	count < 100	Iteration

count	count <= 100	Iteration

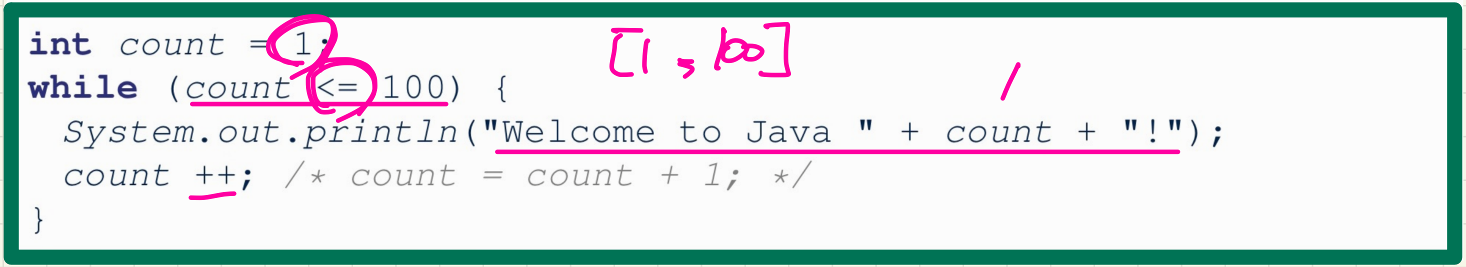
while-Loop: Exercises (2)



```
int count = 0;
while (count < 100) {
  System.out.println("Welcome to Java " + count + "!");
  count ++; /* count = count + 1; */
}
```

[0, 99]

0



```
int count = 1;
while (count <= 100) {
  System.out.println("Welcome to Java " + count + "!");
  count ++; /* count = count + 1; */
}
```

[1, 100]

1

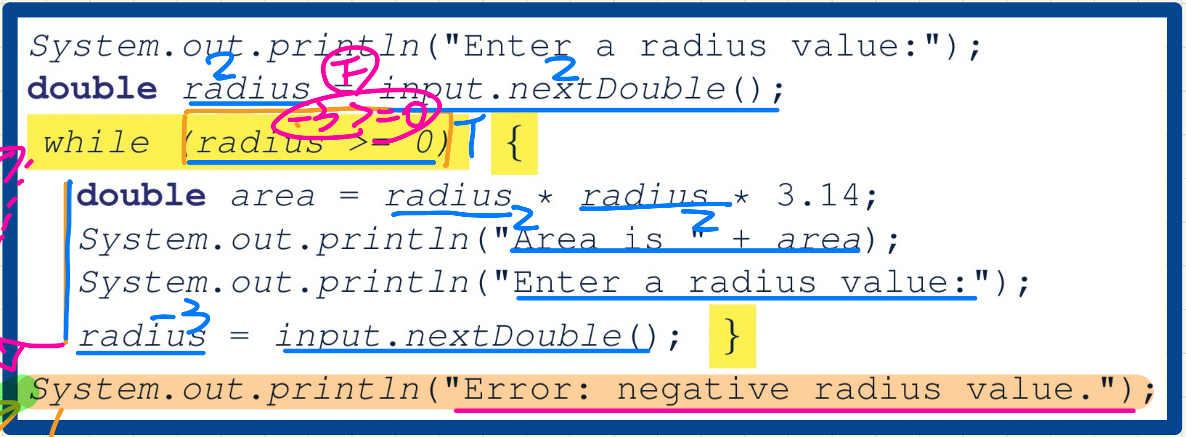
Q. Are the outputs same or different?

Lecture 3

Part B

Loops -
Compound Loops,
for-Loops vs. and while-Loops

Compound Loop: Exercises (1)



```
System.out.println("Enter a radius value:");
double radius = input.nextDouble();
while (radius >= 0) {
  double area = radius * radius * 3.14;
  System.out.println("Area is " + area);
  System.out.println("Enter a radius value:");
  radius = input.nextDouble(); }
System.out.println("Error: negative radius value.");
```

reaching this line, we already exit from loop.

↳ $!(radius >= 0)$

$\equiv$ $radius < 0$

Test Inputs:

radius = -3

Test Inputs:

radius = 2

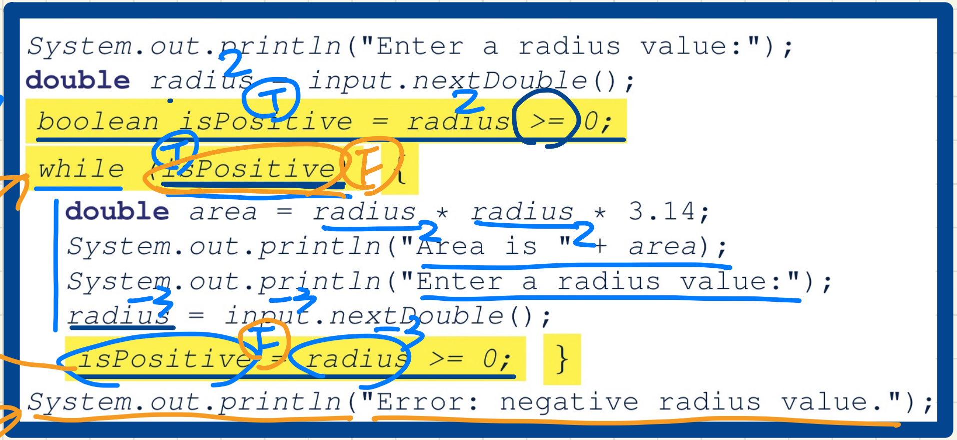
radius = -3

Test Inputs:

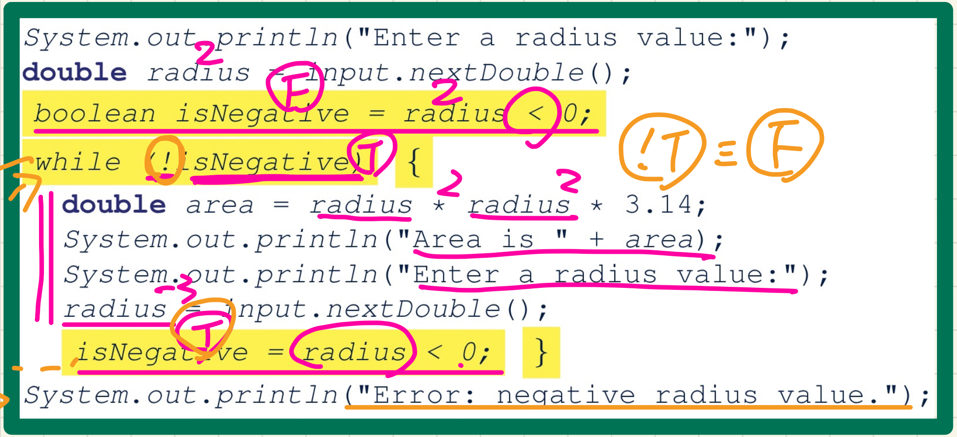
radius = 2

radius = 3

Compound Loop: Exercises (2.1)



```
System.out.println("Enter a radius value:");
double radius = input.nextDouble();
boolean isPositive = radius >= 0;
while (isPositive) {
  double area = radius * radius * 3.14;
  System.out.println("Area is " + area);
  System.out.println("Enter a radius value:");
  radius = input.nextDouble();
  isPositive = radius >= 0; }
System.out.println("Error: negative radius value.");
```



```
System.out.println("Enter a radius value:");
double radius = input.nextDouble();
boolean isNegative = radius < 0;
while (!isNegative) {
  double area = radius * radius * 3.14;
  System.out.println("Area is " + area);
  System.out.println("Enter a radius value:");
  radius = input.nextDouble();
  isNegative = radius < 0; }
System.out.println("Error: negative radius value.");
```

Test Inputs:

radius = -3

Test Inputs:

radius = 2

radius = -3

Test Inputs:

radius = 2

radius = 3

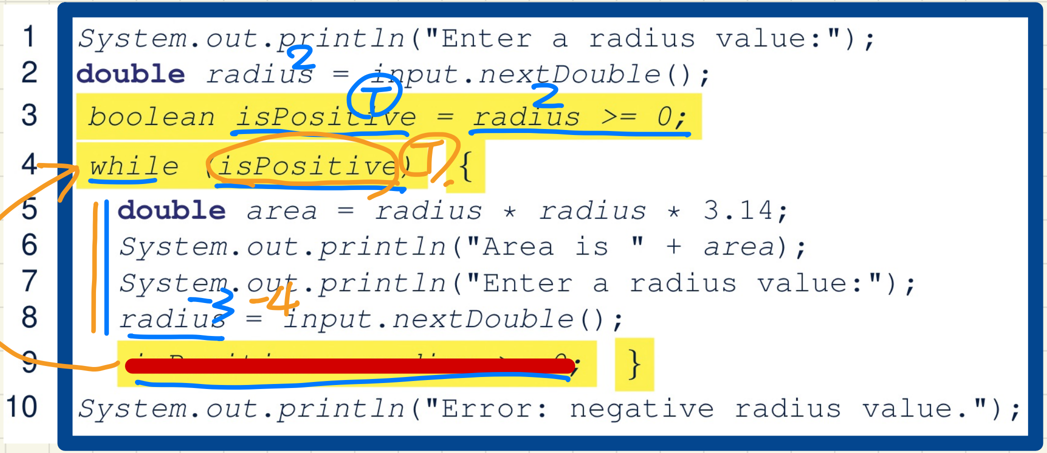
Compound Loop: Exercises (2.2)

Q. What if we delete the update at **Line 9**?

Test Inputs:

radius = 2

radius = -3



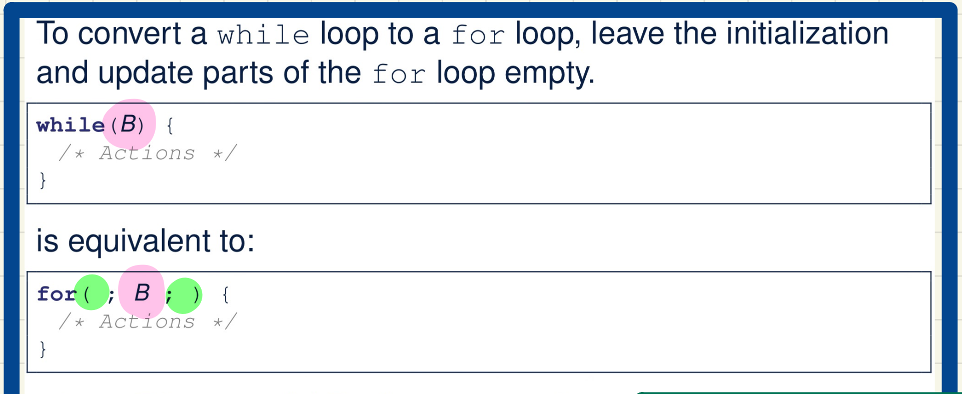
```
1  System.out.println("Enter a radius value:");
2  double radius = input.nextDouble();
3  boolean isPositive = radius >= 0;
4  while (isPositive) {
5    double area = radius * radius * 3.14;
6    System.out.println("Area is " + area);
7    System.out.println("Enter a radius value:");
8    radius = input.nextDouble();
9    ~~isPositive = radius >= 0;~~ }
10 System.out.println("Error: negative radius value.");
```

Console

?

try this on Eclipse

for-Loop vs. while-Loop



To convert a `while` loop to a `for` loop, leave the initialization and update parts of the `for` loop empty.

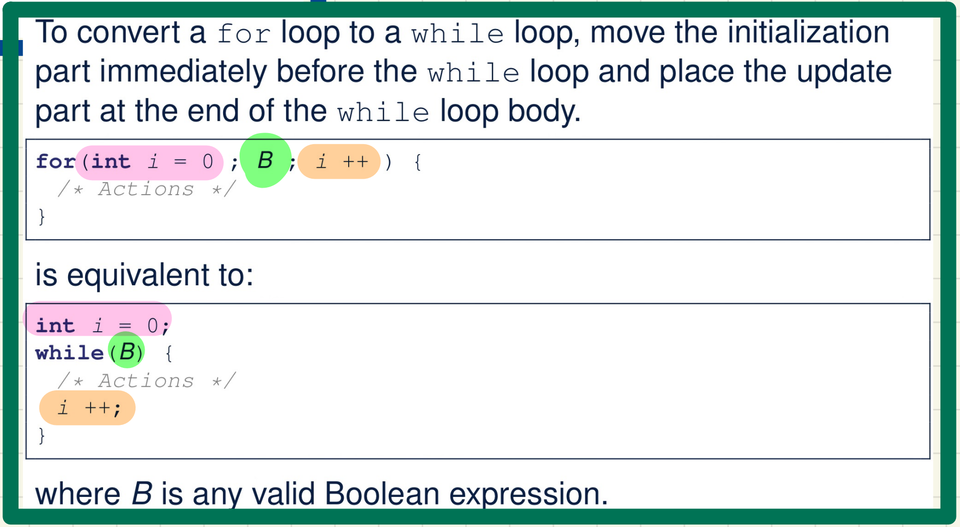
```
while(B) {
  /* Actions */
}
```

is equivalent to:

```
for( ; B ; ) {
  /* Actions */
}
```

where *B* is any valid Boolean expression.

expressive power
equivalent



To convert a `for` loop to a `while` loop, move the initialization part immediately before the `while` loop and place the update part at the end of the `while` loop body.

```
for(int i = 0 ; B ; i ++) {
  /* Actions */
}
```

is equivalent to:

```
int i = 0;
while(B) {
  /* Actions */
  i ++;
}
```

where *B* is any valid Boolean expression.