Lecture 4
Loops

Department of Electrical and Computer Engineering,

North South University

Motivations

Suppose that you need to print a string (e.g., "Welcome to Java!") a hundred times. It would be tedious to have to write the following statement a hundred times:

```
System.out.println("Welcome to Java!");
```

So, how do you solve this problem?

Opening Problem

Problem:

100 times

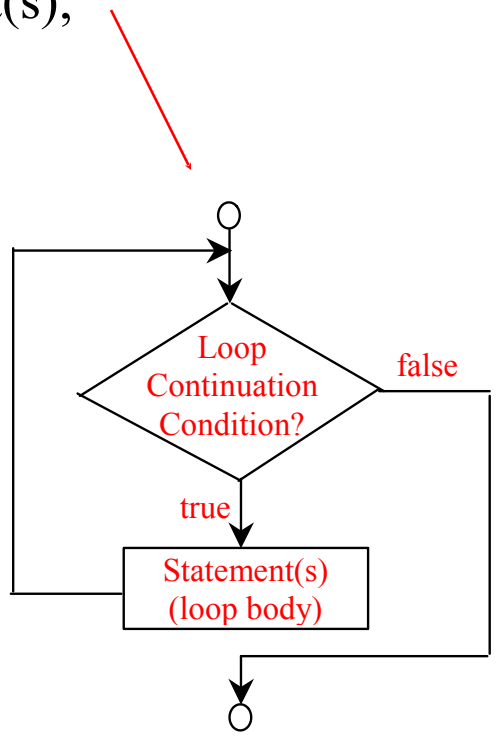
```
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");

...

...

...
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
```

Introducing while Loops

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

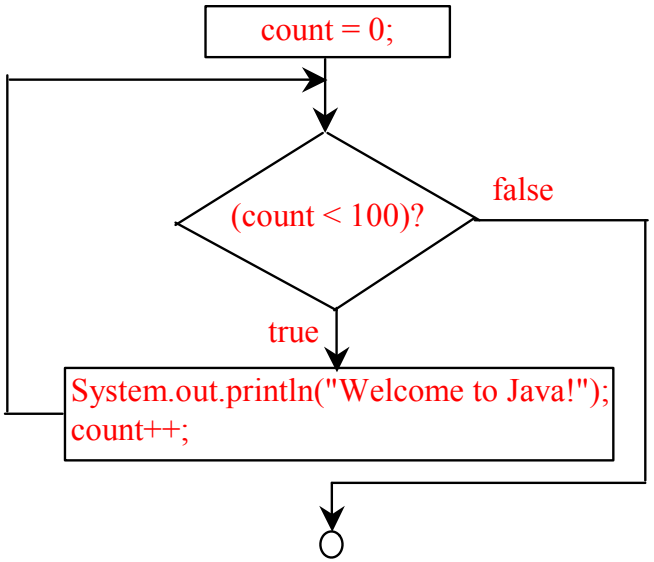
`while` Loop Flow Chart

```
while (loop-continuation-condition) {
  // loop-body;
  Statement(s);
}
```



(A)

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



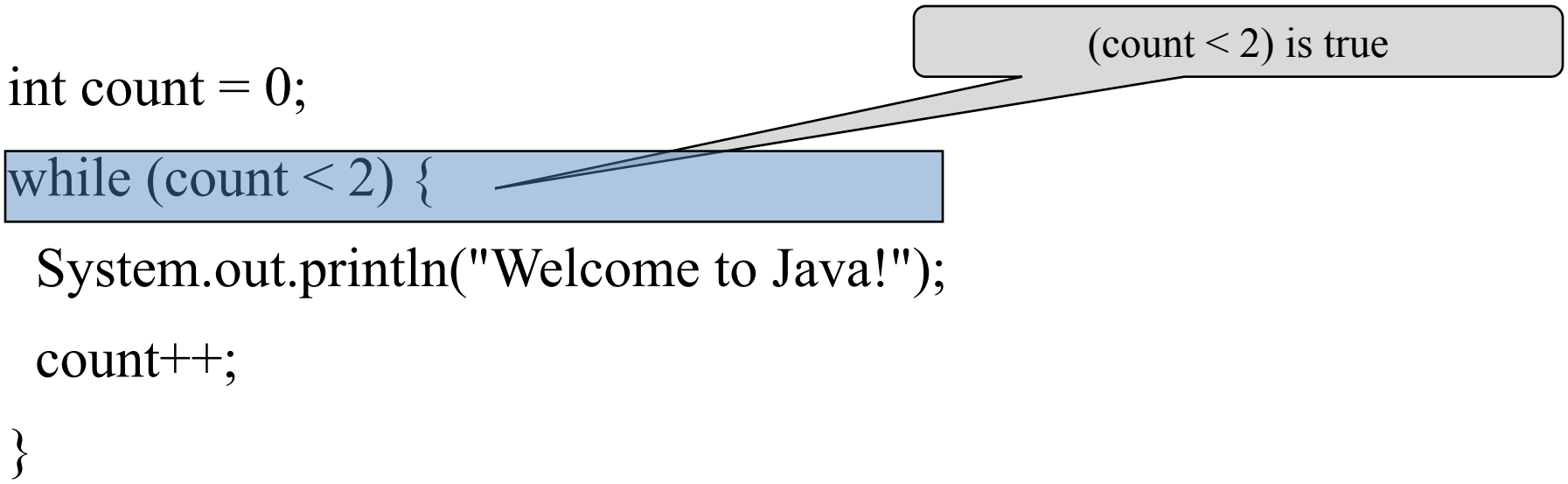
(B)

Trace while Loop

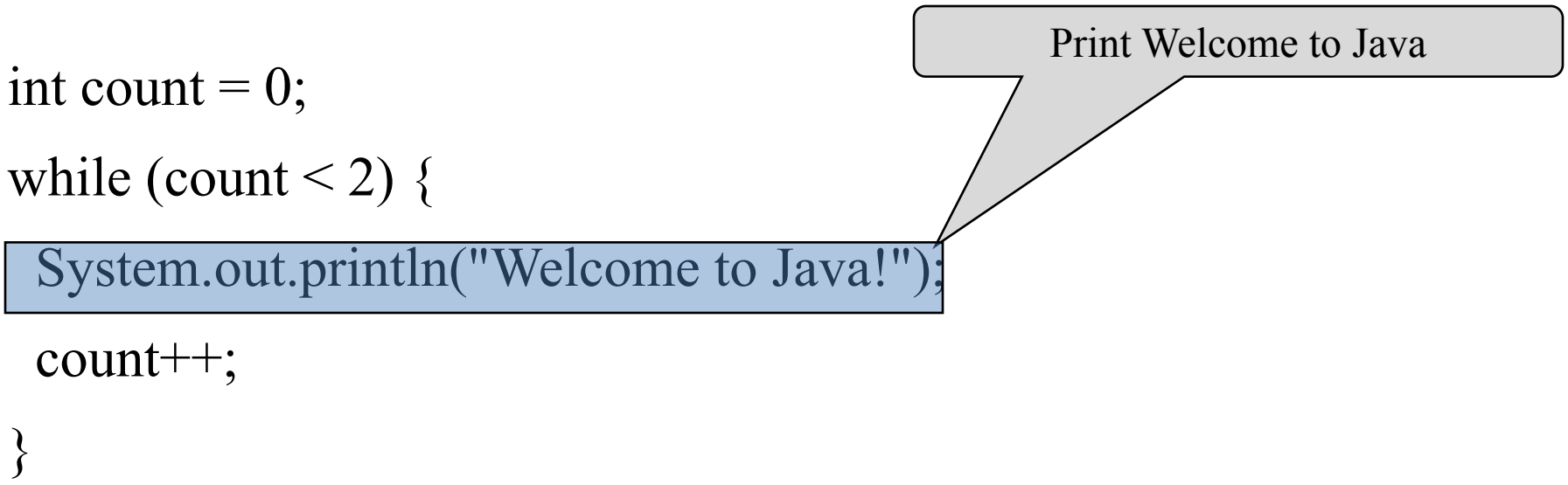
```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Initialize count

Trace while Loop, cont.

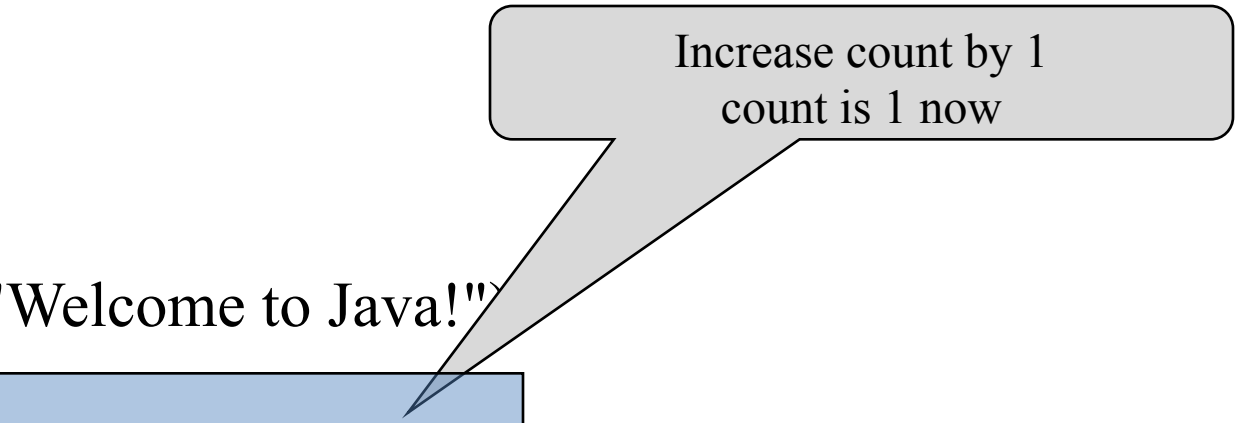


Trace while Loop, cont.



Trace while Loop, cont.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

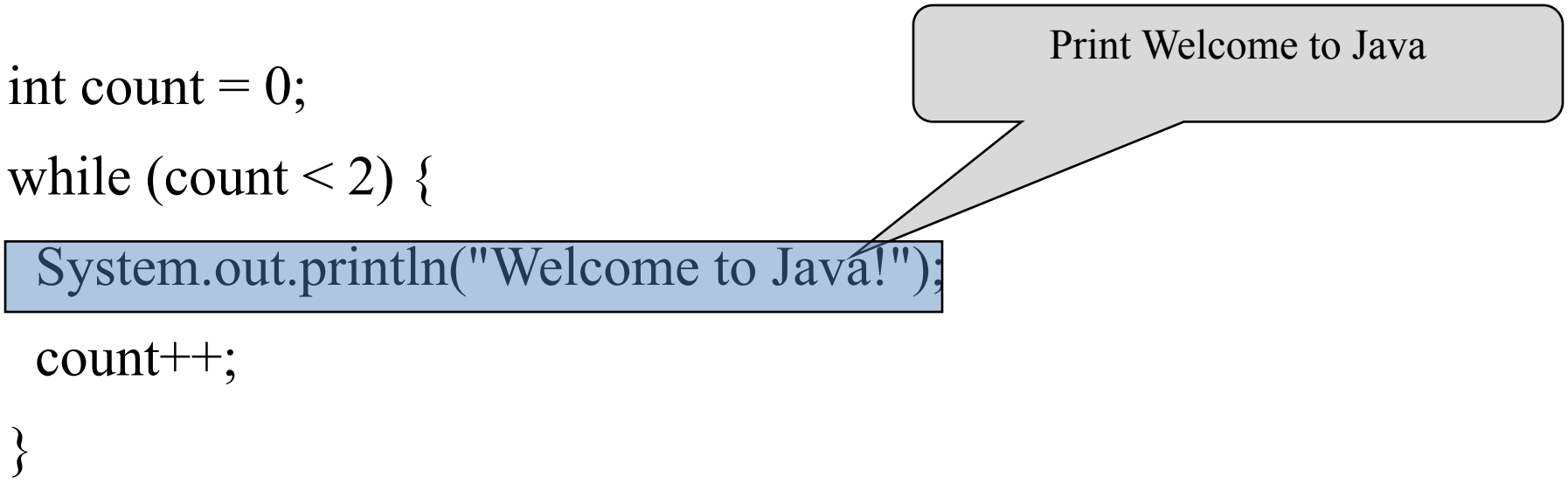


Trace while Loop, cont.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

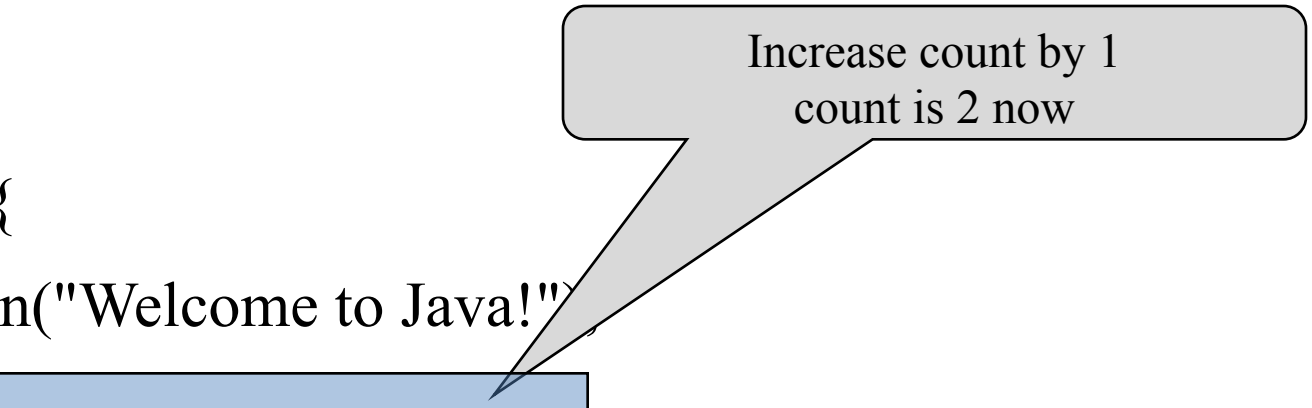
(count < 2) is still true since count is 1

Trace while Loop, cont.



Trace while Loop, cont.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```



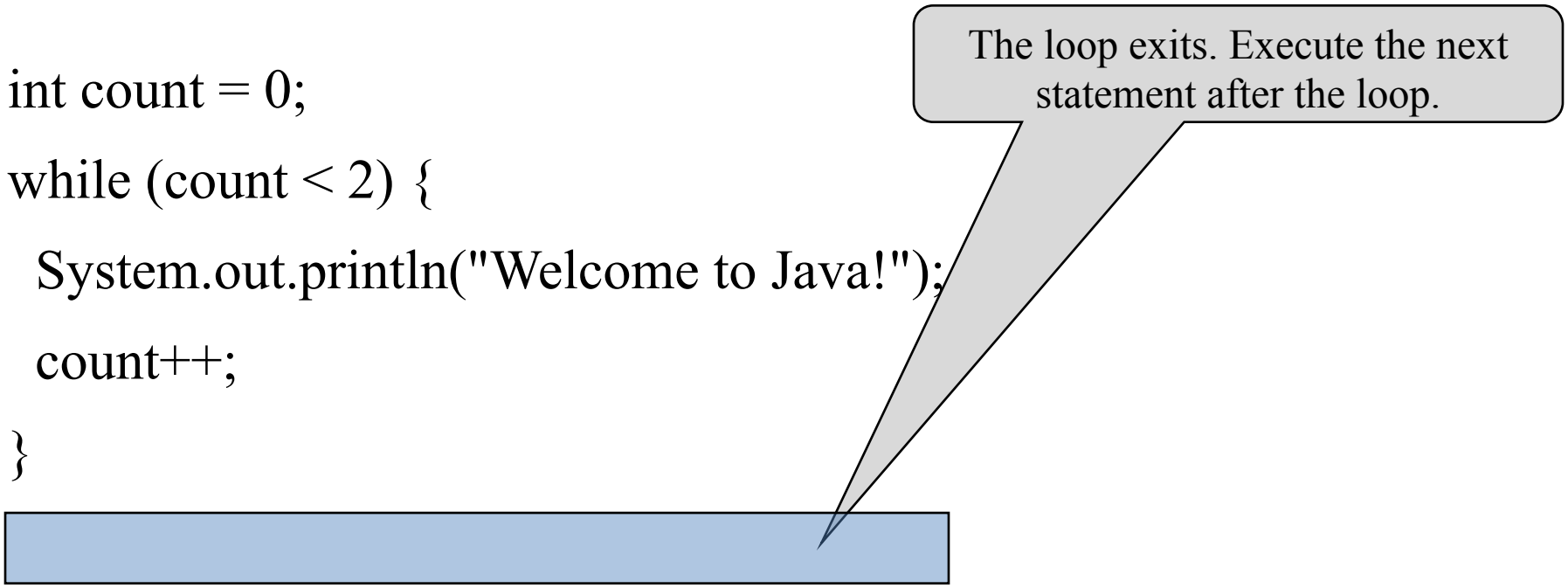
Trace while Loop, cont.

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

(count < 2) is false since count is 2 now

Trace while Loop

```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```



Ending a Loop with a Sentinel Value

- Often the number of times a loop is executed is not predetermined. You may use an input value to signify the end of the loop. Such a value is known as a *sentinel value*.

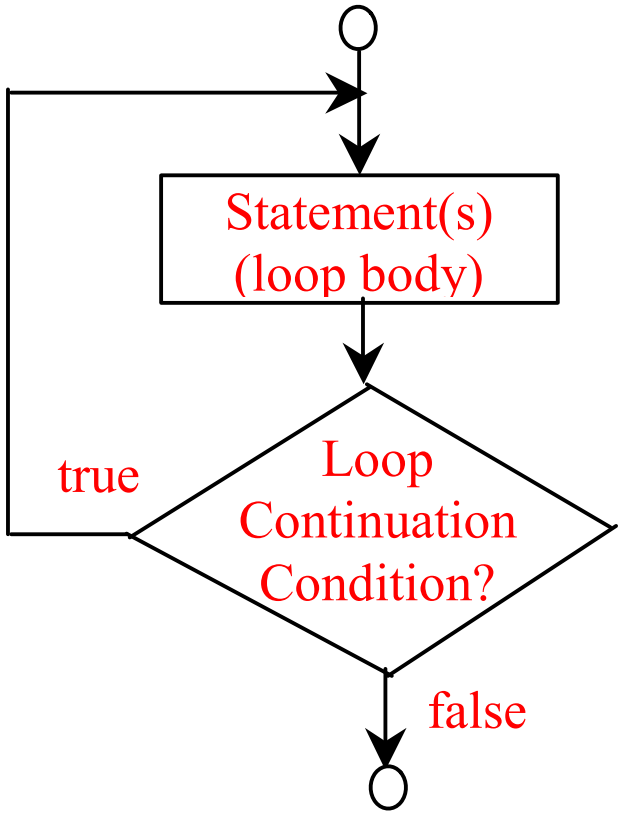
Caution

- Don't use floating-point values for equality checking in a loop control. Since floating-point values are approximations for some values, using them could result in imprecise counter values and inaccurate results. Consider the following code for computing 1 + 0.9 + 0.8 + ... + 0.1:

```
double item = 1; double sum = 0;
while (item != 0) { // No guarantee item will be 0
  sum += item;
  item -= 0.1;
}
System.out.println(sum);
```

Variable item starts with 1 and is reduced by 0.1 every time the loop body is executed. The loop should terminate when item becomes 0. However, there is no guarantee that item will be exactly 0, because the floating-point arithmetic is approximated. This loop seems OK on the surface, but it is actually an infinite loop.

do-while Loop

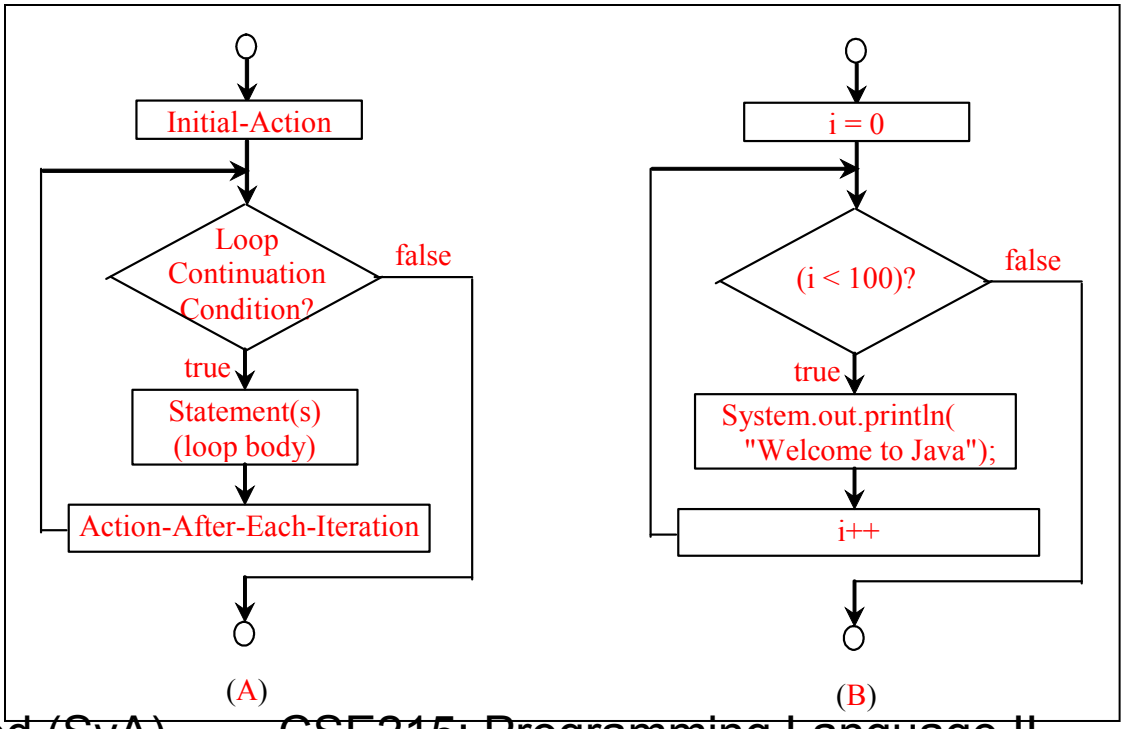



```
do {
   // Loop body;
   Statement(s);
} while (loop-continuation-condition);
```

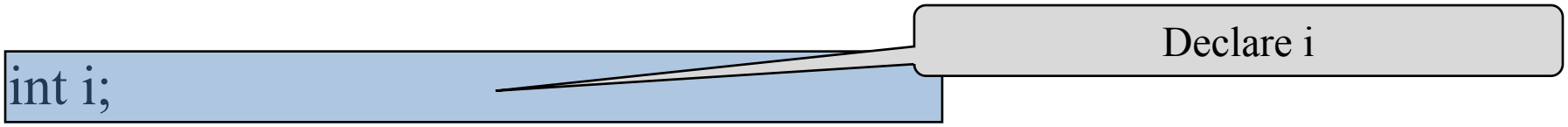
`for` Loops

```
for (initial-action; loop-
    continuation-condition;
    action-after-each-iteration) {
  // loop body;
  Statement(s);
}
```

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

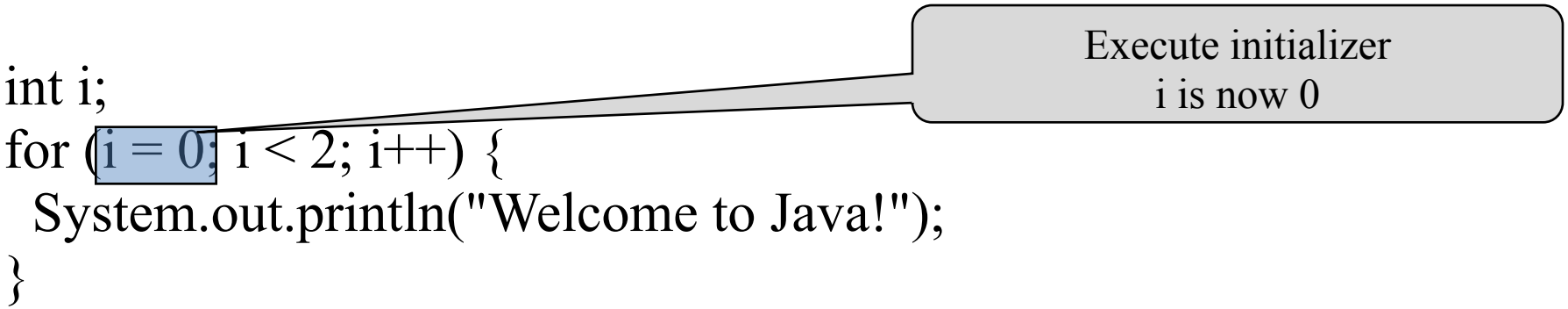


Trace for Loop

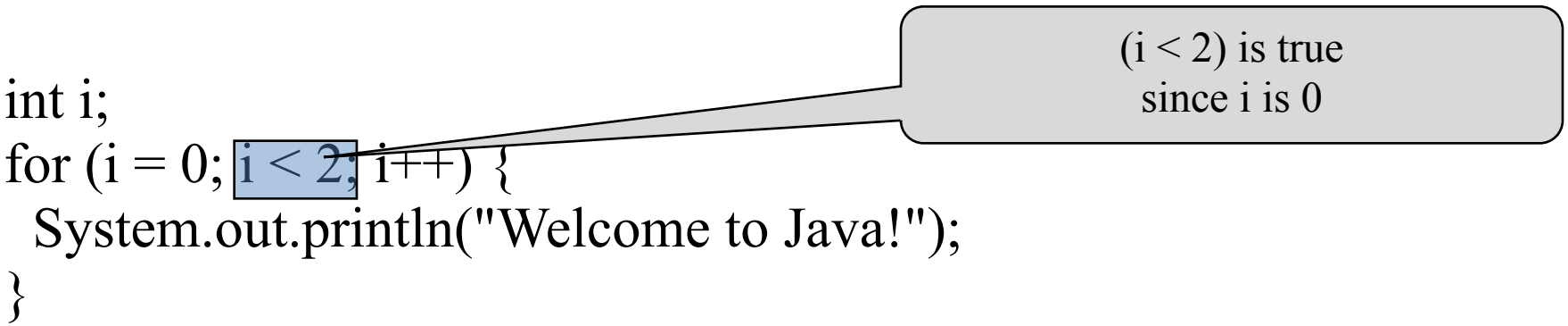



```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

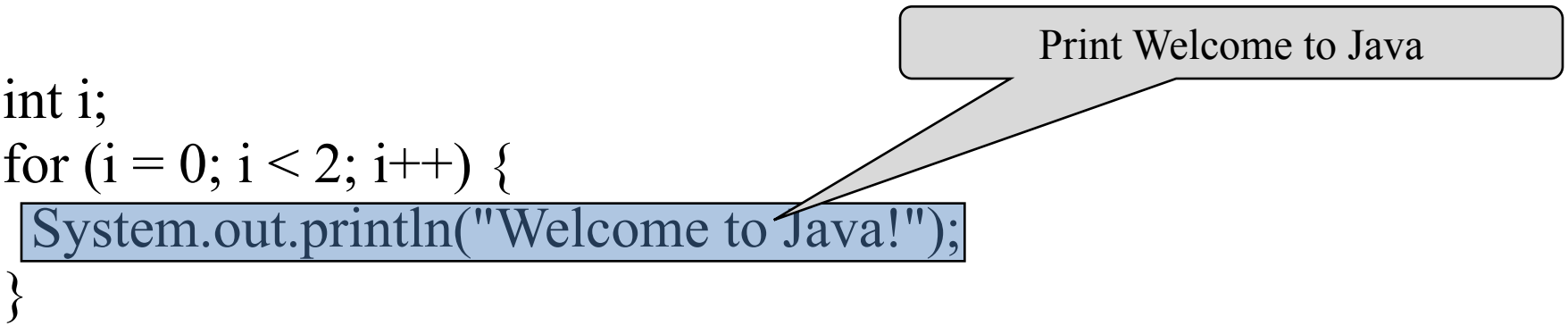
Trace for Loop, cont.



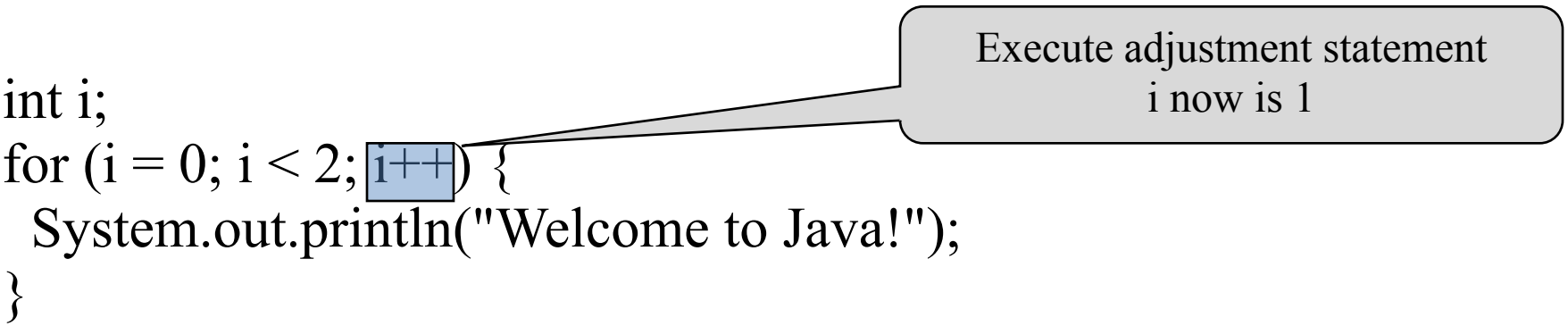
Trace for Loop, cont.



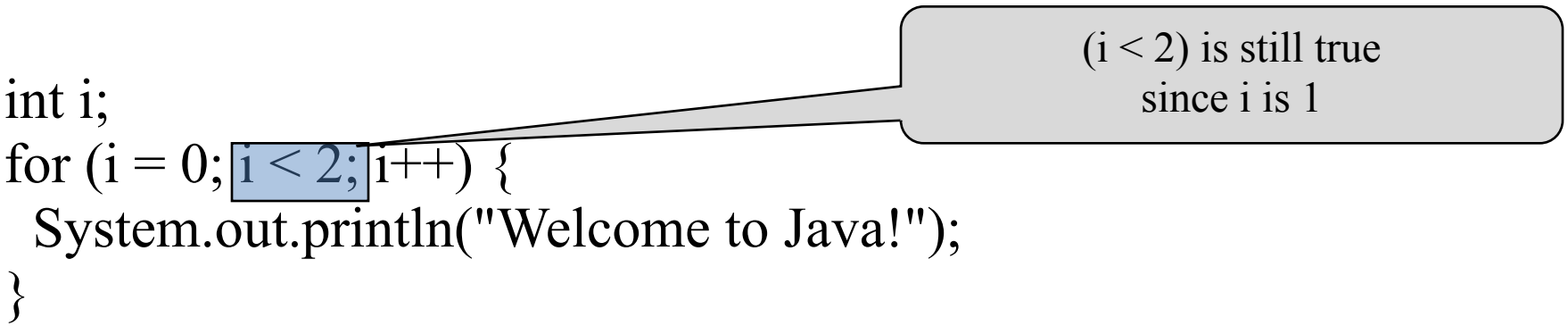
Trace for Loop, cont.



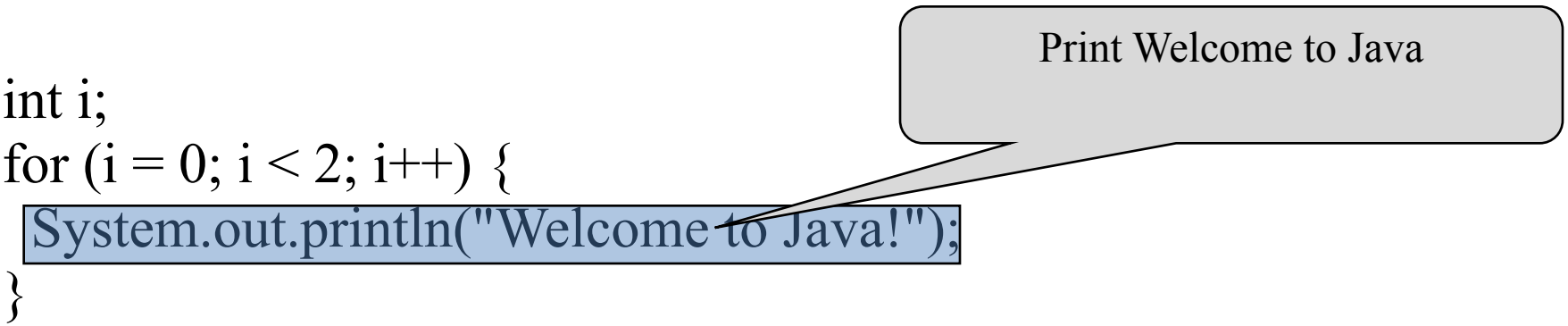
Trace for Loop, cont.



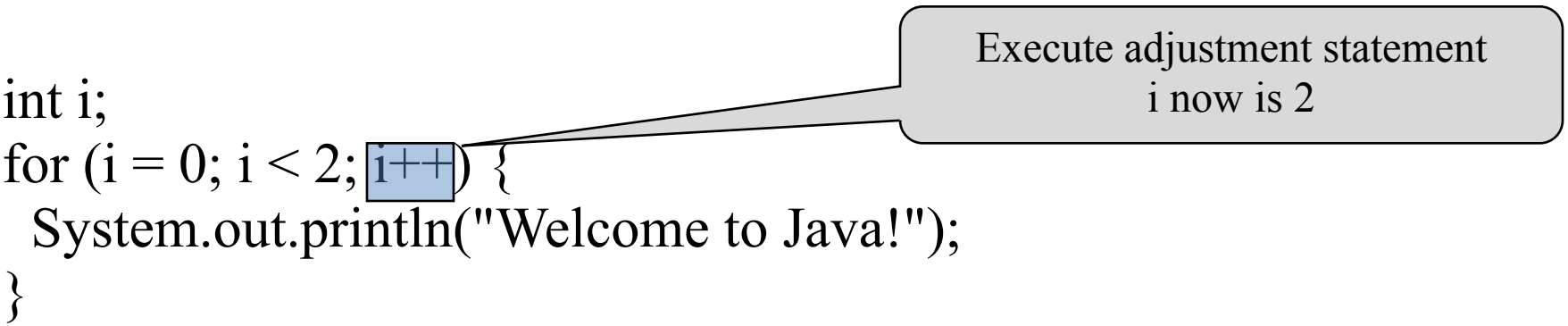
Trace for Loop, cont.



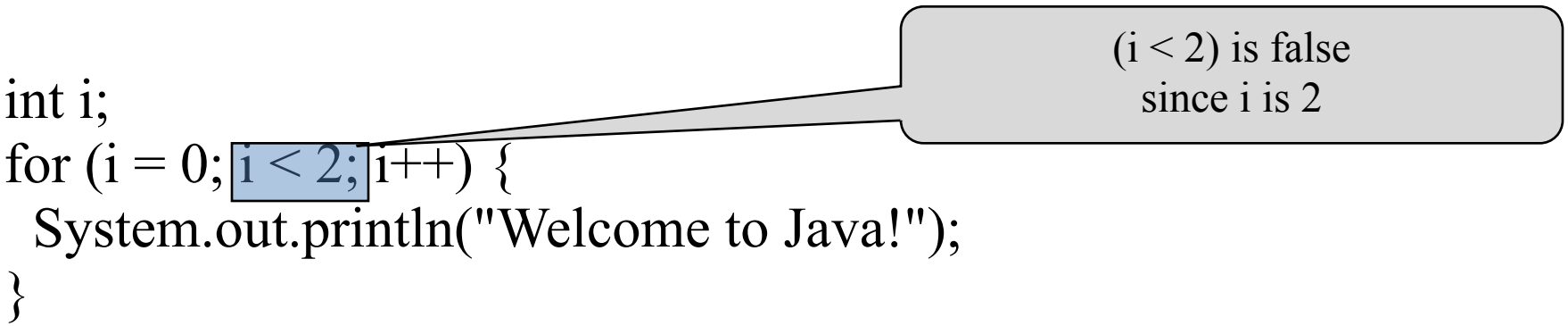
Trace for Loop, cont.



Trace for Loop, cont.

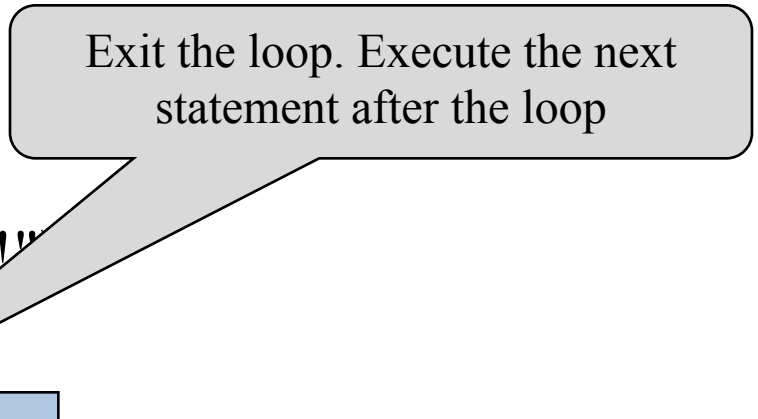


Trace for Loop, cont.



Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```



Note

The initial-action in a for loop can be a list of zero or more comma-separated expressions.

The action-after-each-iteration in a for loop can be a list of zero or more comma-separated statements.

Therefore, the following two for loops are correct. They are rarely used in practice, however.

```
for (int i = 1; i < 100; System.out.println(i++));

for (int i = 0, j = 0; (i + j < 10); i++, j++) {
  // Do something
}
```

Note

If the loop-continuation-condition in a for loop is omitted, it is implicitly true. Thus the statement given below in (a), which is an infinite loop, is correct. Nevertheless, it is better to use the equivalent loop in (b) to avoid confusion:

```
for ( ; ; ) {
  // Do something
}
```

(a)

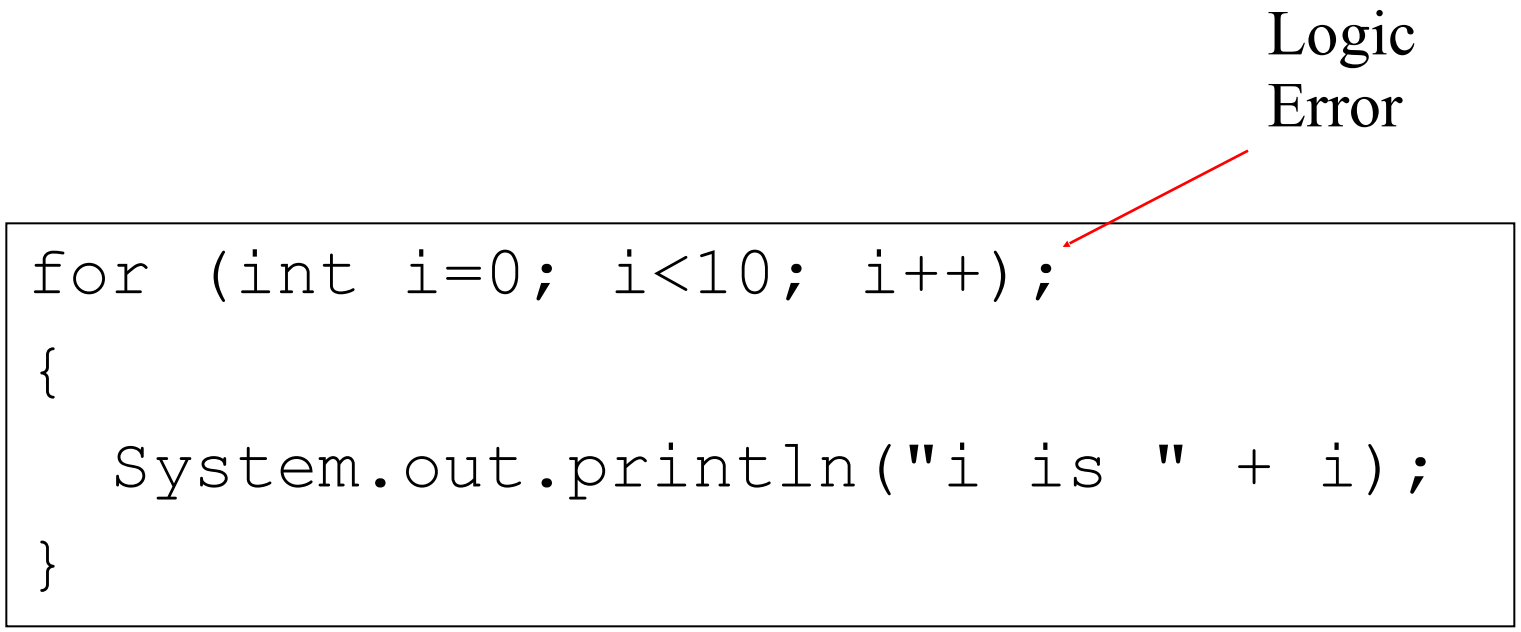
Equivalent

```
while (true) {
  // Do something
}
```

(b)

Caution

Adding a semicolon at the end of the for clause before the loop body is a common mistake, as shown below:



Caution, cont.

Similarly, the following loop is also wrong:

```
int i=0;
while (i < 10);
{
   System.out.println("i is " + i);
   i++;
}
```

Logic Error

In the case of the do loop, the following semicolon is needed to end the loop.

```
int i=0;
do {
   System.out.println("i is " + i);
   i++;
} while (i<10);
```

Correct

Which Loop to Use?

The three forms of loop statements, while, do-while, and for, are expressively equivalent; that is, you can write a loop in any of these three forms. For example, a while loop in (a) in the following figure can always be converted into the following for loop in (b):

```
while (loop-continuation-condition) {
  // Loop body
}
```

(a)

Equivalent

```
for ( ; loop-continuation-condition; )
  // Loop body
}
```

(b)

A for loop in (a) in the following figure can generally be converted into the following while loop in (b) except in certain special cases

```
for (initial-action;
     loop-continuation-condition;
     action-after-each-iteration) {
  // Loop body;
}
```

(a)

Equivalent

```
initial-action;
while (loop-continuation-condition) {
  // Loop body;
  action-after-each-iteration;
}
```

(b)

Recommendations

- Use the one that is most intuitive and comfortable for you.
- In general, a for loop may be used if the number of repetitions is known, as, for example, when you need to print a message 100 times.
- A while loop may be used if the number of repetitions is not known, as in the case of reading the numbers until the input is 0.
- A do-while loop can be used to replace a while loop if the loop body has to be executed before testing the continuation condition.

Nested Loops

- A loop can be nested inside another loop.
- Nested loops consist of an outer loop and one or more inner loops.
- Each time the outer loop is repeated, the inner loops are reentered, and started anew.

Nested Loops

The following program uses nested **for** loops to display a multiplication table.

```
14        // Display table body
15        for (int i = 1; i <= 9; i++) {
16          System.out.print(i + " | ");
17          for (int j = 1; j <= 9; j++) {
18            // Display the product and align properly
19            System.out.printf("%4d", i * j);
20          }
21          System.out.println();
22        }
```

Nested Loops

Output:

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

Keyword `break`

- Using a `break` in a loop, immediately terminate the loop.
- Listing 5.12 presents a program to demonstrate the effect of using `break` in a loop.

LISTING 5.12 TestBreak.java

```
1  public class TestBreak {
2    public static void main(String[] args) {
3      int sum = 0;
4      int number = 0;
5
6      while (number < 20) {
7        number++;
8        sum += number;
9        if (sum >= 100)
10         break;
11     }
12
13     System.out.println("The number is " + number);
14     System.out.println("The sum is " + sum);
15   }
16 }
```

Output:

```
The number is 14
The sum is 105
```

Keyword `continue`

- When it is encountered, it ends the current iteration and program control goes to the end of the loop body.
- Listing 5.13 presents a program to demonstrate the effect of using **continue** in a loop.

```
1  public class TestContinue {
2    public static void main(String[] args) {
3      int sum = 0;
4      int number = 0;
5
6      while (number < 20) {
7        number++;
8        if (number ==10 || number == 11)
9          continue;
10       sum += number;
11     }
12
13     System.out.println("The sum is " + sum);
14   }
15 }
```