UNSW Business School
Information Systems and Technology Management

INFS1609/2609 Fundamentals of Business Programming

Lecture 5
Loops

yenni.tim@unsw.edu.au

INFS1609/2609 Fundamentals of Business Programming

Topics for this week:

- A loop design strategy to develop loops
- `while, do-while` and `for` loops
- Similarities and differences of different types of loops
- Nested loops
- Implement program control using `break` and `continue`

Main references
- *Textbook: Chapter 5*
- *Other online references posted on Ed*

INFS1609/2609 Fundamentals of Business Programming

Loops = repetition

The `while` loop

When to use a `while` **loop?**

- Repeat statement(s) for an **unknown** number of times until a condition becomes false

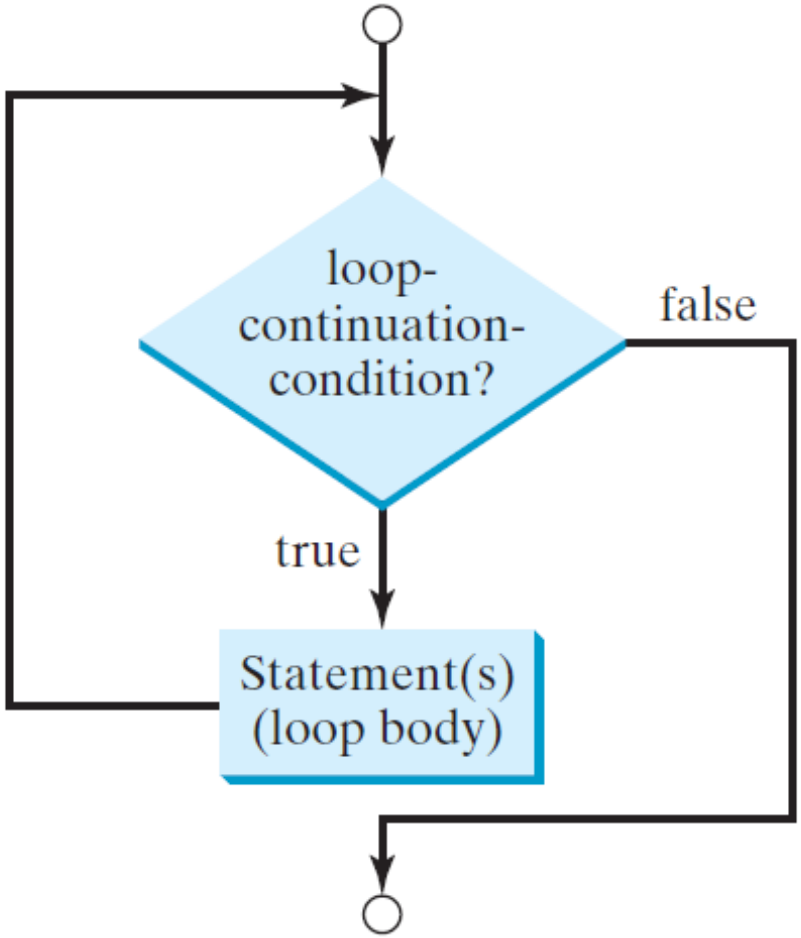
The `while` loop

Syntax:

An expression that returns a `boolean` value

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

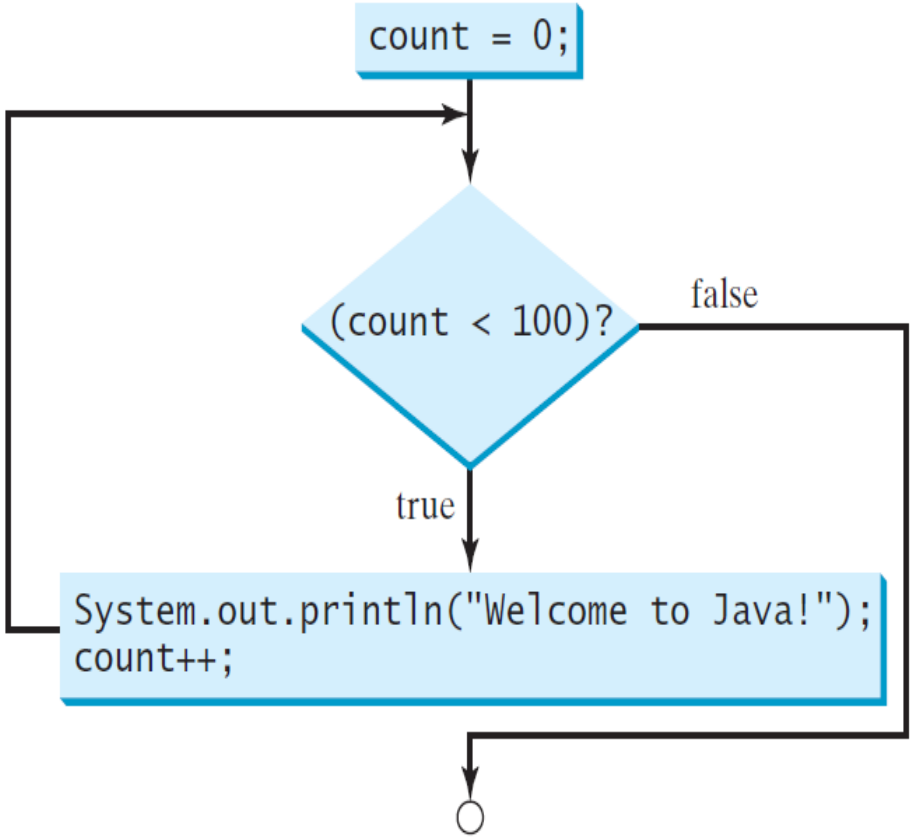
The `while` loop



Possible use cases?

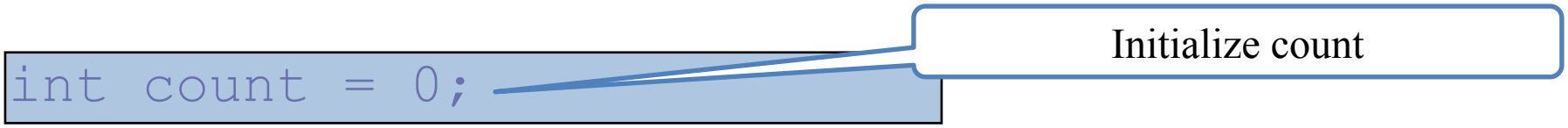
Prompt for user input
until input = 0

The `while` loop – example



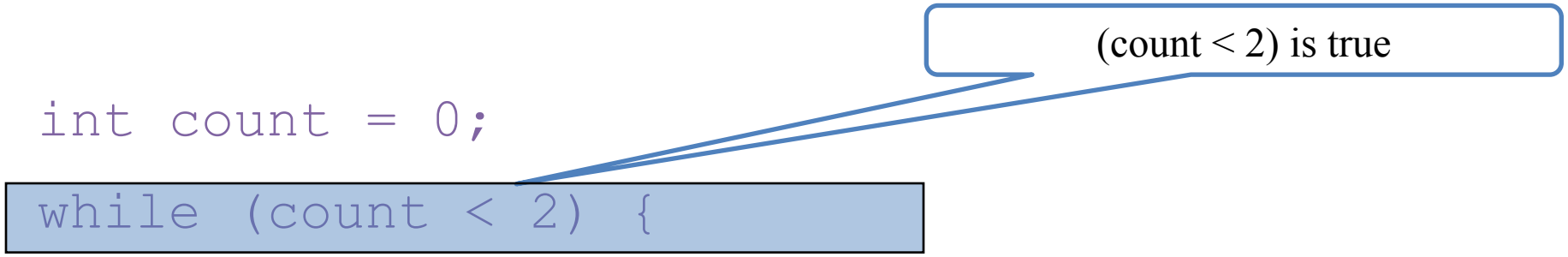

```
int count = 0;

while (count < 10) {

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

   count++;

}
```

Tracing a `while` loop

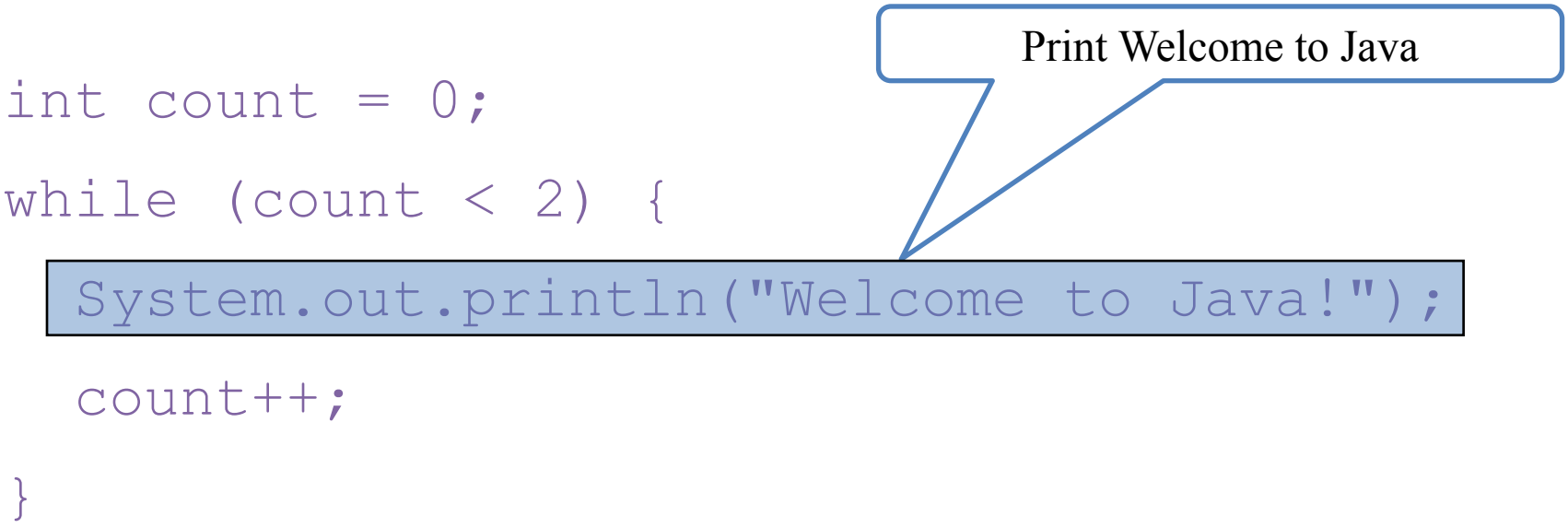



```
int count = 0;
while (count < 2) {
  System.out.println("Welcome to Java!");
  count++;
}
```

Tracing a `while` loop

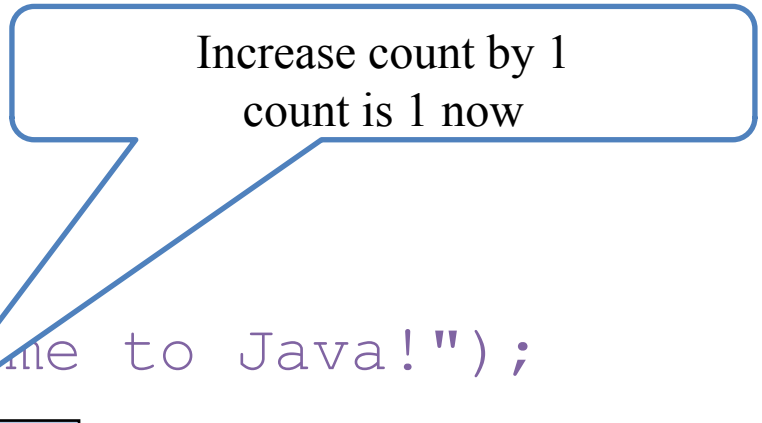


Tracing a `while` loop

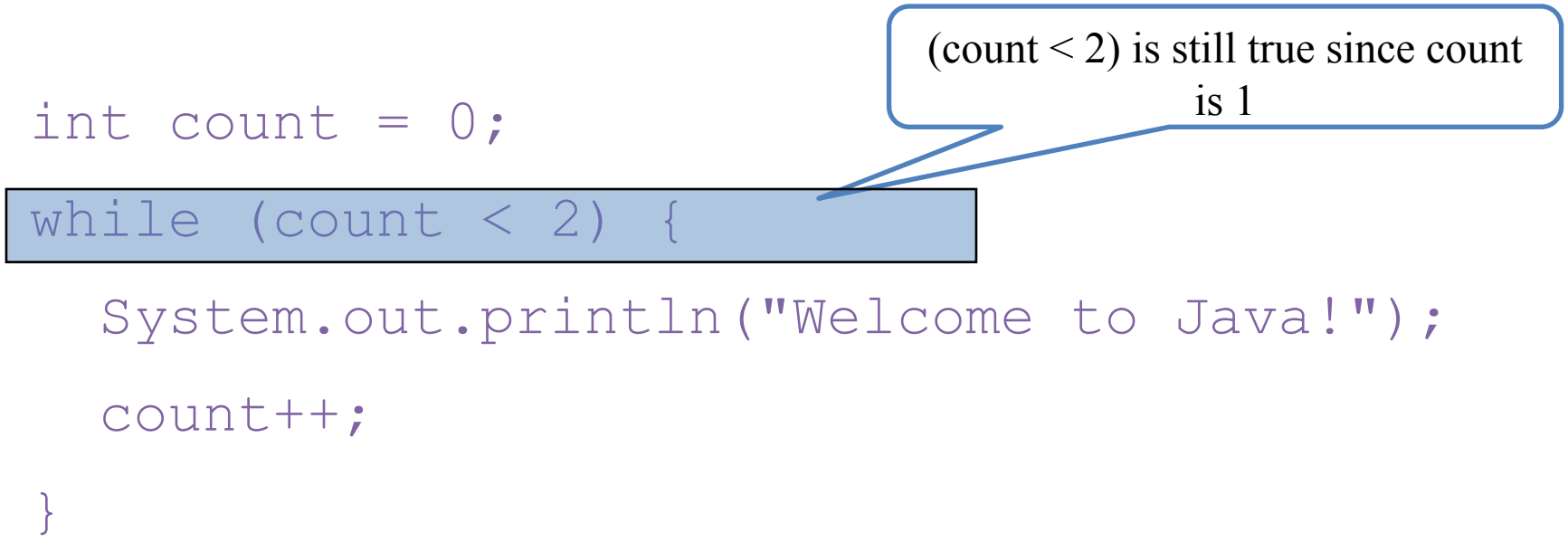


Tracing a `while` loop

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

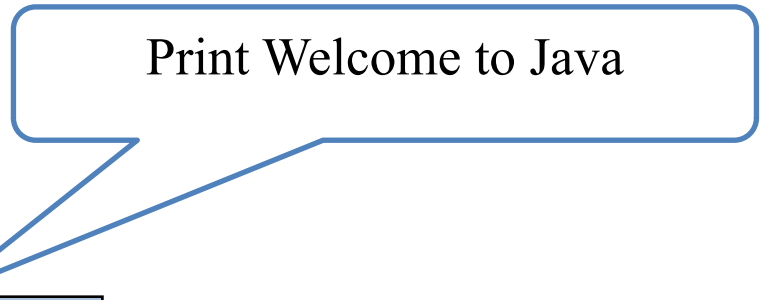


Tracing a `while` loop



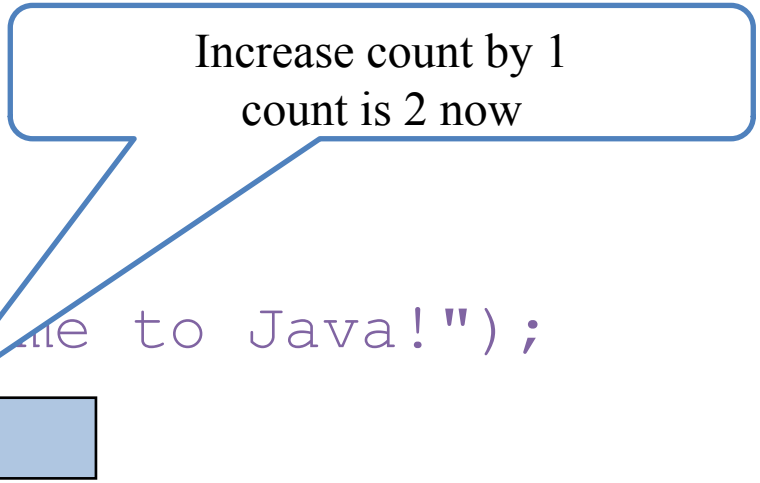
Tracing a `while` loop

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



Tracing a `while` loop

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



Tracing a `while` loop

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

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

Tracing a `while` loop

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

The loop exits. Execute the next statement after the loop.

Example 1

Addition Quiz: Using a while loop, prompts the user to enter an answer for a question on addition of two single digits. Let the user enter a new answer until it is correct

https://liveexample.pearsoncmg.com/html/RepeatAdditionQuiz.html

Extra

The `random` method

- Generates a random double value greater than or equal to `0.0` and less than `1.0` (`0 <= Math.random() < 1.0`)

```
a + Math.random() * b
```
→ Returns a random number between a and a + b, excluding a + b.

Example

```
(int)(Math.random() * 10)
```
→ Returns a random integer between 0 and 9.

```
50 + (int)(Math.random() * 50)
```
→ Returns a random integer between 50 and 99.

Example 2

Guessing numbers: Write a program that randomly generates an integer between 0 and 100, inclusive. The program prompts the user to enter a number continuously until the number matches the randomly generated number. For each user input, the program tells the user whether the input is too low or too high, so the user can choose the next input intelligently

Let's start with this and convert it to a program using loops!

https://liveexample.pearsoncmg.com/html/GuessNumberOneTime.html

Ending a loop

- At times when the number of times a loop is executed is not predetermined, you use an input value to signify the end of the loop
- Such a value is known as *sentinel value*

Example: Write a program that reads and calculates the sum of an unspecified number of integers. The input 0 signifies the end of the input

https://liveexample.pearsoncmg.com/html/SentinelValue.html

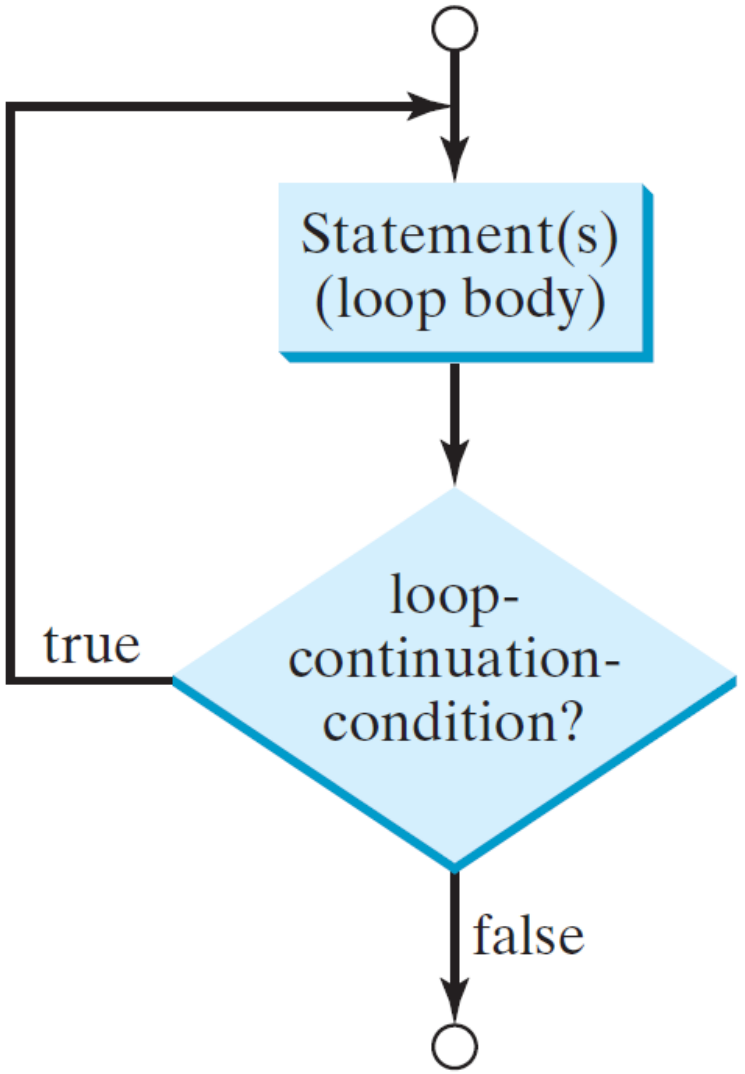
A `do-while` loop

Syntax:

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

A `do-while` loop

replace a while loop if the loop body has to be executed before testing the continuation condition

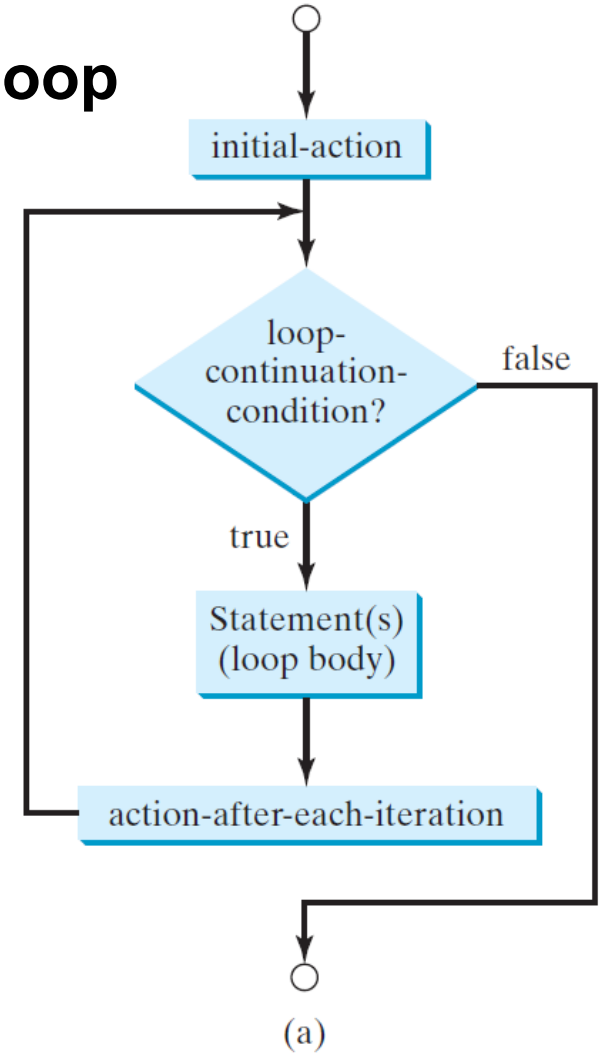


The `for` loop

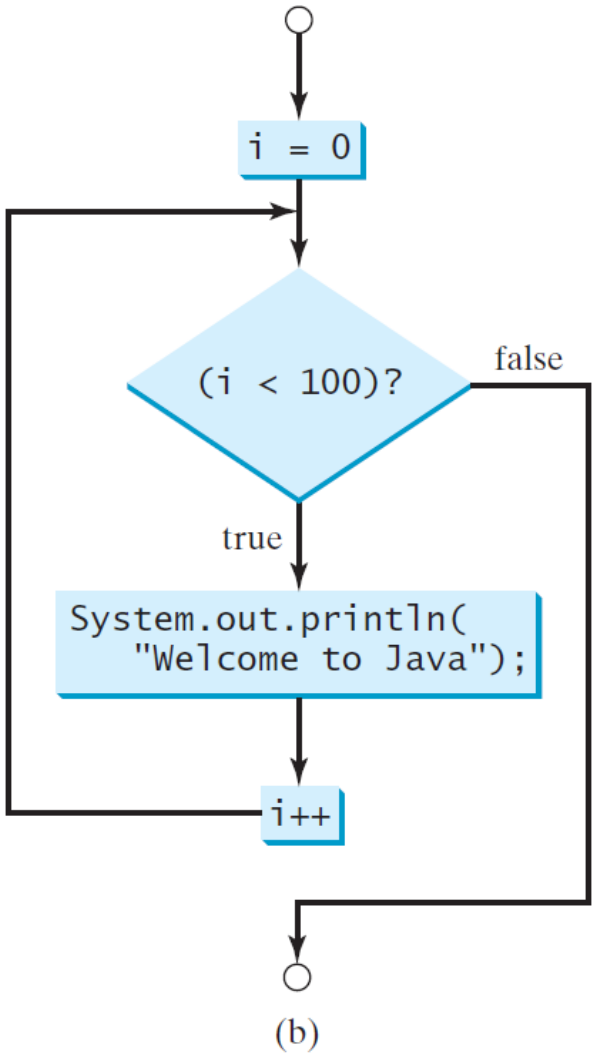
```
for (initial-action; loop-continuation-
   condition; action-after-each-iteration) {
    // loop body;
    Statement(s);
}
```

- Initial-action is executed **only once**
- Then the **loop continuation condition** is evaluated
- If the condition is evaluated to **true**:
  - Codes inside the loop body is executed
- Then the **action-after-each-iteration** is executed
- The process goes on until the continuation-condition is evaluated to **false**

The `for` loop

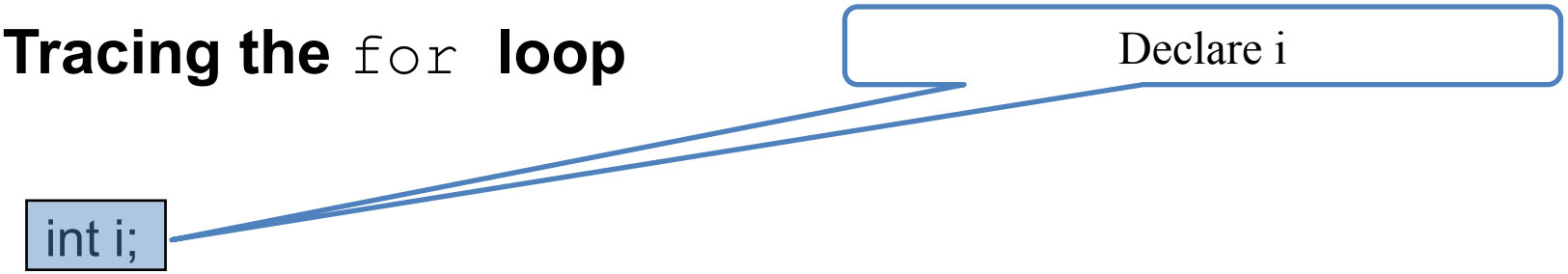


(a)



(b)

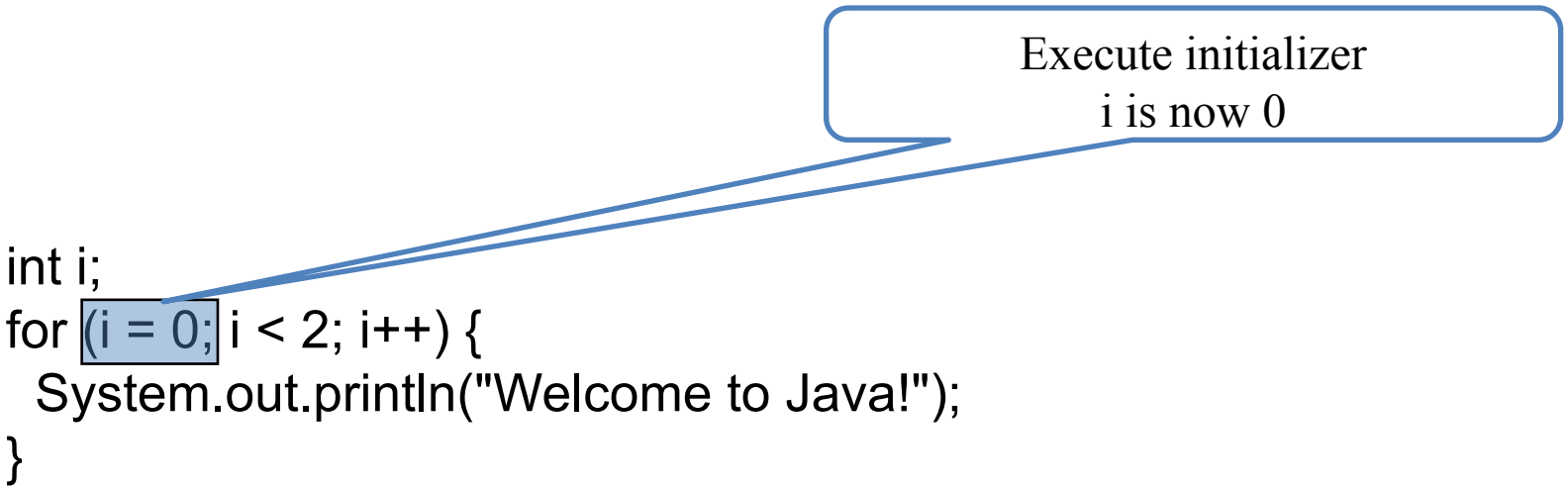
Tracing the `for` loop



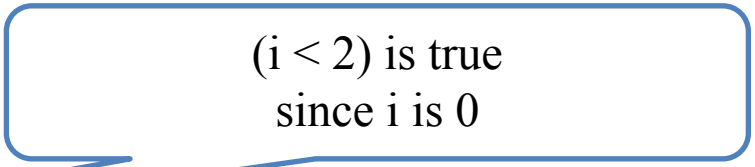
By the way, this is equivalent to:

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

Tracing the `for` loop

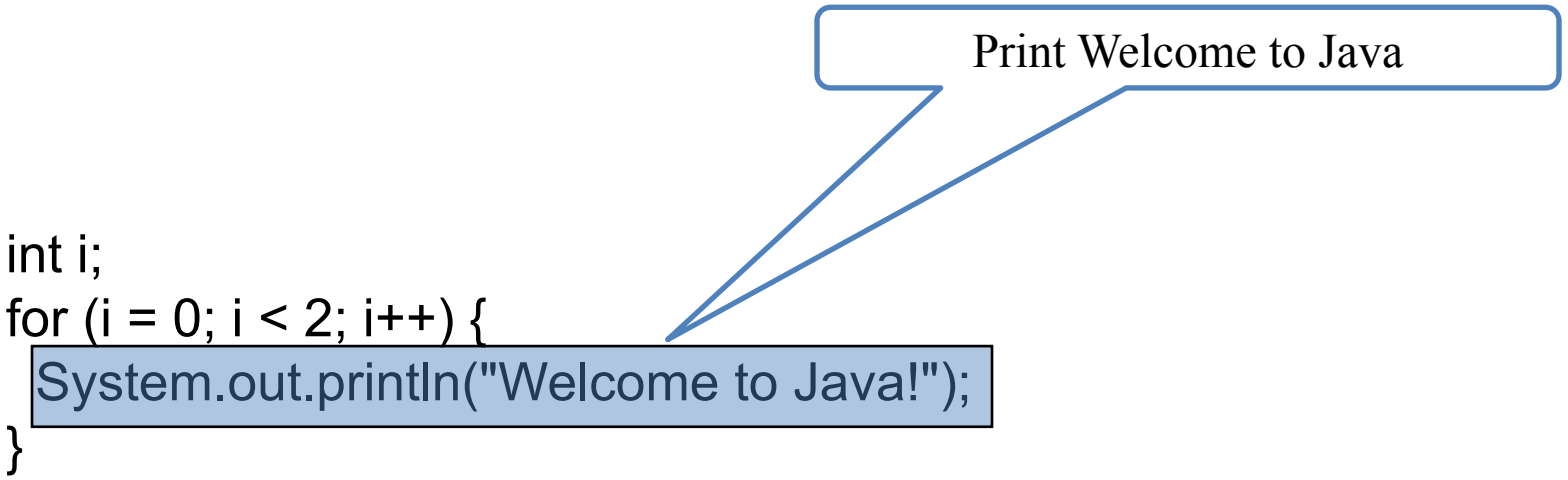


Tracing the `for` loop

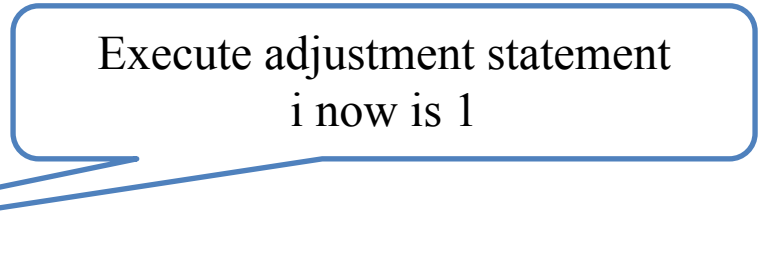



```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Tracing the `for` loop

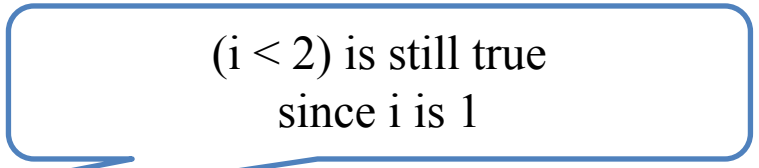


Tracing the `for` loop



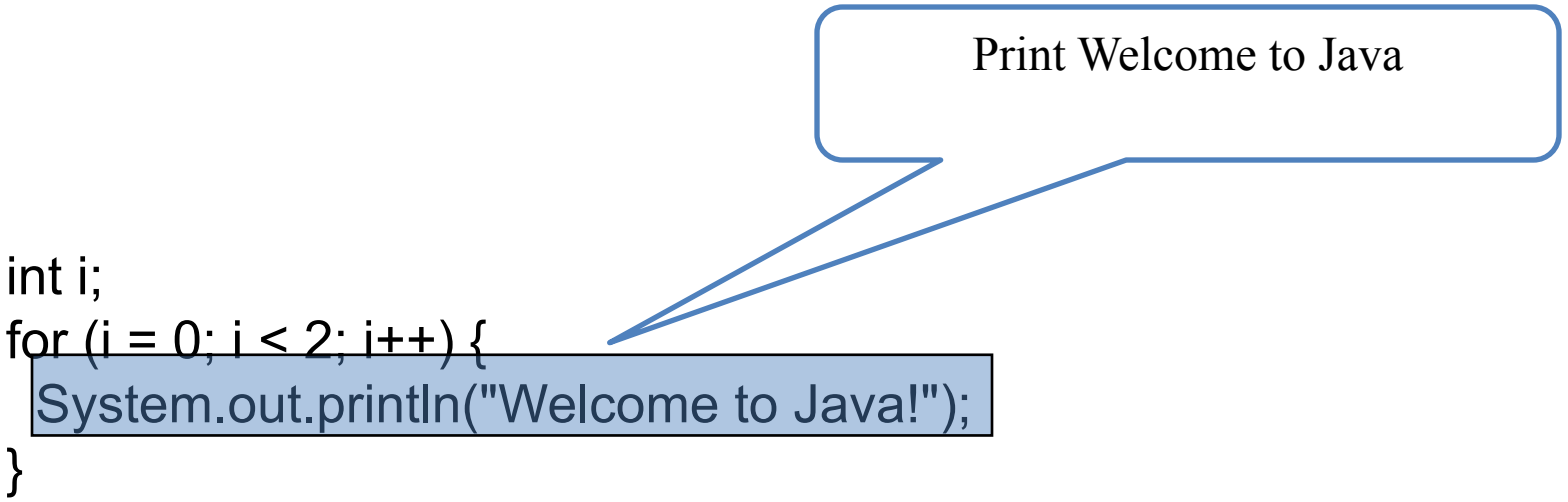

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Tracing the `for` loop

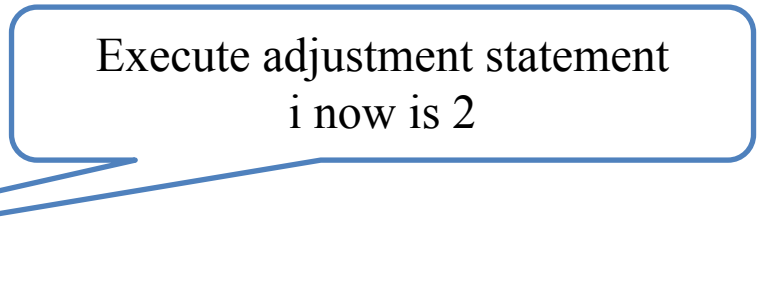



```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Tracing the `for` loop

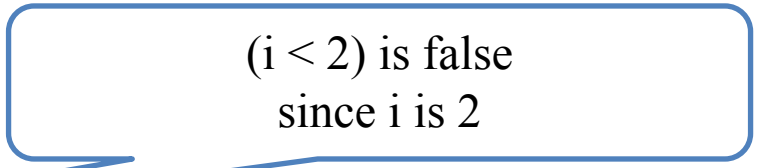


Tracing the `for` loop



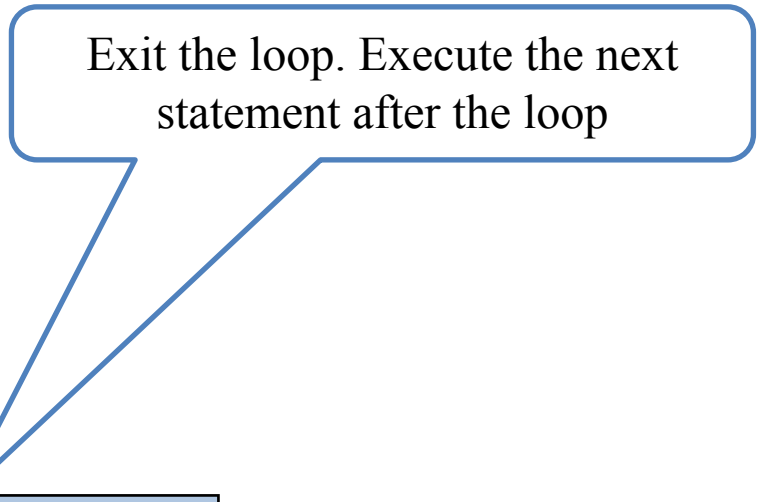

```
int i;
for (i = 0; i < 2; i++) {
  System.out.println("Welcome to Java!");
}
```

Tracing the `for` loop




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

Tracing the `for` loop




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

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)

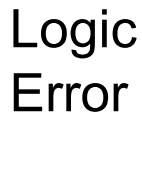
Equivalent

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

(b)

Note

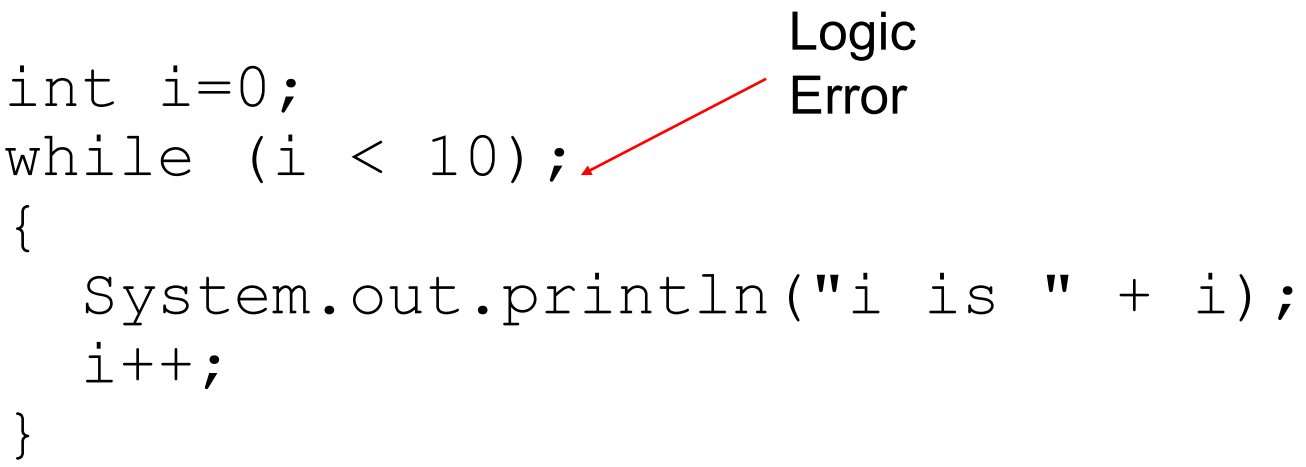
Adding a semicolon at the end of the for statement, before the loop body, is a common mistake:




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

Note

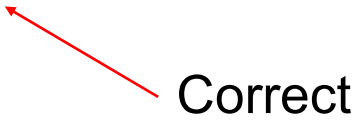
Similarly, the following loop is also wrong:



Note

However, in the case of `do-while` loop, the semicolon is **needed** to end the loop:

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



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)

Which loop to use?

- `for` loop in (a) in the following figure can generally be converted into the following while loop in (b) *except in certain special cases* (see Review Question 3.19 for one of them):

```
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)

Which loop to use?

- Use the one that is most intuitive and comfortable for you
- 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

- 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**

```
for(int i = 0; i < 3; i++){
    System.out.println("this is i: " + i);
    for(int j = 0; j < i; j++)
        System.out.println("this is j: " + j);
}
```

(printlnA)

(printlnB)

How many times the two println statements will be executed?

Nested Loops

Example: Write a program that uses nested for loops to print a multiplication table

https://liveexample.pearsoncmg.com/html/MultiplicationTable.html

Using `break` and `continue` with loops

- `break` can be used to **terminate** a `for`, `while`, or `do-while` loop

  https://liveexample.pearsoncmg.com/html/TestBreak.html

- `continue` **skips** the **current iteration** of a `for`, `while`, or `do-while` loop

  https://liveexample.pearsoncmg.com/html/TestContinue.html

Let's look at few examples:

Example 1:
Combining `for` loop, `if` statement and casting

Code is on Ed

Example 1:

Combining `for` loop, `if` statement and casting

- The “printable” range of characters from the ASCII table is 33-126
- Since the loop is counting integers, we can display the integers directly as the ASCII value (remember our casting?)
- The character ‘\t’ is used to put a tab stop between each pairs of values
- The String “: ” included in the statement is important – why?

Example 2:
Finding the Greatest Common Divisor

Write a program that prompts the user to enter two positive integers and finds their greatest common divisor

5 minutes to write down a pseudocode

Example 2:

Finding the Greatest Common Divisor

- Suppose you enter two integers 4 and 2, their greatest common divisor is 2
- Suppose you enter two integers 16 and 24, their greatest common divisor is 8
- So, how do you find the greatest common divisor? Let the two input integers be n1 and n2. You know number 1 is a common divisor, but it may not be the greatest commons divisor. So you can check whether k (for k = 2, 3, 4, and so on) is a common divisor for n1 and n2, until k is greater than n1 or n2

Example 3:

Guessing Number

Re-write the GuessingNumber program by using a break statement

https://liveexample.pearsoncmg.com/html/GuessNumberUsingBreak.html

Reflection

What have you learned today?

Thank you!