CS-2011 Introduction to Programming with Java

California State University, Los Angeles
Computer Science Department

Loops II

The `for` Loop

The `for` Loop

- **`for`** loops are the third kind of loop structure that Java provides, and have a very concise syntax for writing them
- Syntax:

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

- ***initial-action***: is often used to initialize a control variable (i.e. a counter variable)
- ***action-after-each-iteration***: usually increments or decrements the control variable
- ***loop-continuation-condition***: tests whether the control variable has reached a termination value and can be any valid Boolean expression.
- **NOTE:** these three parts of the loop must be separated by **semicolons**

The `while` Loop

- Syntax:

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

- a **`while`** loop executes statements repeatedly while the condition is true.
- **loop body:** the part of the loop that contains the statements to be repeated. the statements can be anything we have seen so far (including if/else statements and other loops)
- **iteration** (or **repetition**) **of the loop:** a one-time execution of the loop body
  - 1 iteration = 1 execution of the loop body
- **loop-continuation-condition:**
  - is a Boolean expression that controls the execution of the body
  - is evaluated each time the loop repeats to determine if the loop body should be executed
  - if it evaluates to true the loop body is executed, false the entire loop terminates.

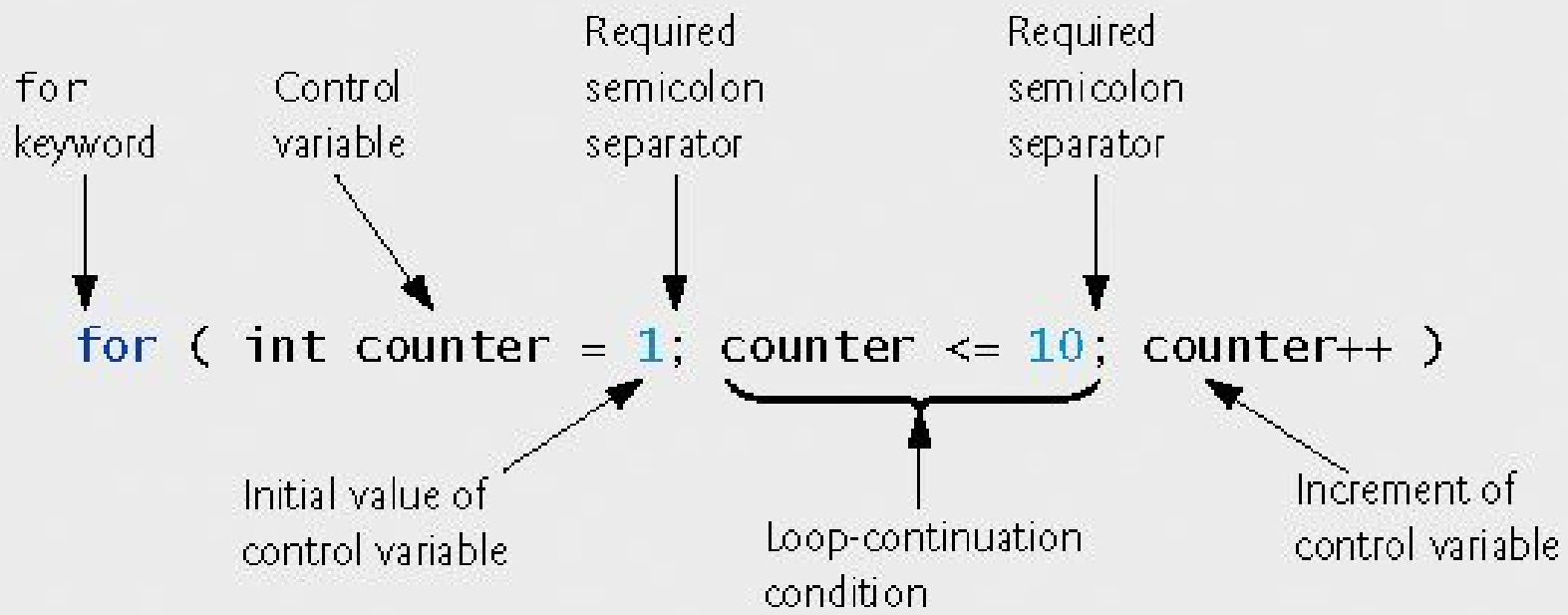
Example

- The following for loop prints "Welcome to Java!" 100 times

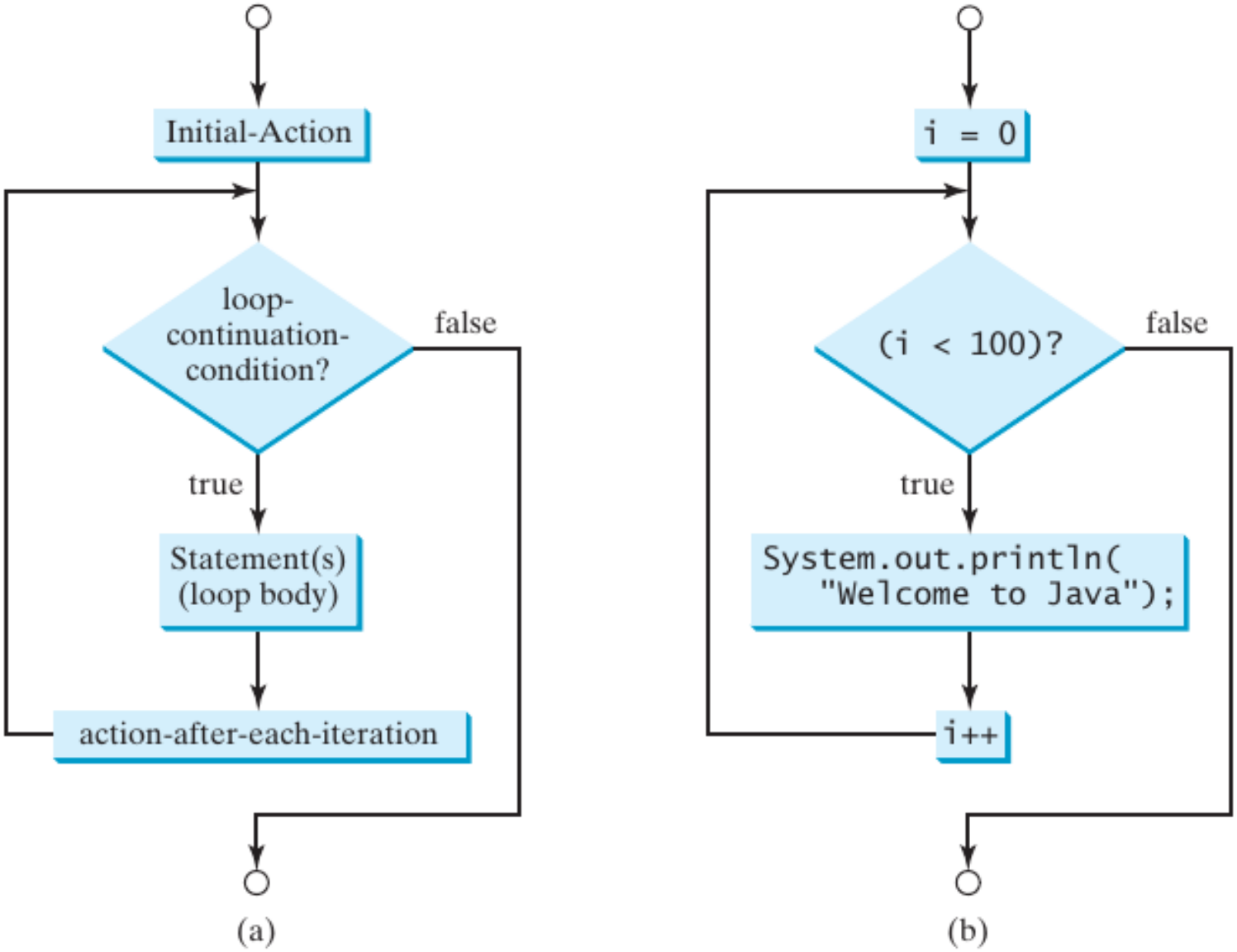
```
for (int i = 0 ; i < 100 ; i++) {
  System.out.println("Welcome to Java!");
}
```

- the **initial-action**, `int i = 0`, initializes an integer variable called `i` with a value of `0`
- the **loop-continuation-condition**, `i < 100`, is a Boolean expression. It is evaluated right after initializing, and at the beginning of each loop iteration. If the condition is `true`, the loop body is executed, if it is `false` the loop terminates.
- the action-after-each-iteration, `i++`, adjusts the control variable. Eventually this control variable should change its value so that it makes the loop-continuation-condition `false`, otherwise the loop is infinite.

The for Loop


for keyword
Control variable
Required semicolon separator
Required semicolon separator
for (int counter = 1; counter <= 10; counter++)
Initial value of control variable
Loop-continuation condition
Increment of control variable

The for Loop



A **for** loop performs an initial action once, then repeatedly executes the statements in the loop body, and performs an action after an iteration when the **loop-continuation-condition** evaluates to **true**.

Tracing a `for` Loop

declare and
initialize `i` to `0`

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

Tracing a for Loop

(i < 2) is true
(0 < 2) is true

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

Tracing a `for` Loop

print
"Welcome to Java"

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

Tracing a `for` Loop

increase **`i`** by **`1`**
`i` is now **`1`**

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

Tracing a for Loop

(i < 2) is true
(1 < 2) is true

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

Tracing a `for` Loop

print
"Welcome to Java"

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

Tracing a `for` Loop

increase **`i`** by **`1`**
`i` is now **`2`**

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

Tracing a `for` Loop

`(i < 2)` is `false`
`(2 < 2)` is `false`

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

`for` Loop Tips

- The control variable:
  - can be declared inside the loop control structure, or before the loop.
  - most commonly declared inside the loop control structure
  - if the variable is declared this way it cannot be referenced outside of the loop.
- the initial-action can be a list of zero or more comma-separated variable declaration statements or assignment expressions:

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

for Loop Tips

- 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.
- The statement in (a) is also the same as in (b).
- Nevertheless, it is better to use the equivalent loop in (c) to avoid confusion:

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

(a)

Equivalent

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

(b)

Equivalent

This is better

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

(c)

Loops and Strings

Common Loop Errors

Common Loop Errors – Incorrect Semicolon

- Putting semicolons at the end of the **`for`** or **`while`** loop headers.
- This makes the body of the **`for`** or **`while`** loop an empty body.
- This is a **logic error**, hard to detect, and might cause an infinite loop in some instances.

```
for (int i = 0; i < 2; i++);
  //WRONG! Empty loop body!
{
  System.out.println("Hello");
}
```

```
int i = 0;
while (i < 10);
    //WRONG Empty loop body!
{
    System.out.println("Hello");
    i=i+1;
}
```

Common Loop Errors – Infinite Loops

- ***infinite loop:*** a loop which repeats without ever ending
- Avoid INFINITE loops!!
  - Don't forget to change the counter variable correctly.
  - Don't forget to encode the loop continuation condition correctly.
- Notice what's wrong with the following?

```
for (int i = 0; i < 2; i--) {
  System.out.println("Hello");
}
```

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

Common Loop Errors – Incorrect # of Iterations

- Make sure the number of repetitions is correct!
- Each of these loops should iterate 10 times.
- How many times does each loop actually iterate in the following examples?

```
for (int i = 1; i < 10; i++) {
  System.out.println("Hello");
}
```

```
int i = 0;
while (i <= 10) {
    System.out.println("Hello");
    i = i + 1;
}
```

Common Loop Errors – Commas in `for` Loops

- Don't replace the semicolons in a for loop with commas.
- This is a **syntax error**.

```
for (int i = 0, i < 10, i++) { //Wrong!
  System.out.println("Hello");
}
```

Common Loop Errors – Control Variable Scope

- When a **`for`** loop's control variable is declared in the initialization section of the **`for`** loop's header, using the control variable after the **`for`** loop's body is a compilation error.
- Don't forget variable scope rules!

```
for (int i = 0; i < 10; i++) {
  System.out.println("Hello");
}
i = i + 5; //Wrong!
```

Loop Exercises

- Do the following two loops result in the same value of sum?

```
for (int i = 0; i < 10; ++i) {
    sum += i;
}
```

```
for (int i = 0; i < 10; i++) {
    sum += i;
}
```

How many time does each loop iterate? What values of i are printed in each example?

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

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

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

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

Hand Trace Exercise

- Hand trace the following code using 2 3 4 5 2 for the input:

```
Scanner input = new Scanner(System.in);


int number, sum = 0, count;


for (count = 0 ; count < 5 ; count++) {
   number = input.nextInt();
   sum += number;
}


System.out.println("sum is " + sum);
System.out.println("count is " + count);
```

Which Loop to Use?

Which Loop to Use?

- **`while`** and **`for`** loops are **pretest loops**
  - they check the continuation condition **before** executing the loop body.
- **`do-while`** loops are **post-test loops**
  - they check the continuation condition **after** executing the loop body.
- **`while`**, **`do-while`**, and **`for`**, are equivalent i.e. you can generally write a loop in any of these three forms.
  - in most cases some loops are better than others
  - all depends on what you are trying to do

Which Loop to Use?

- 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)

Which Loop to Use?

- If you know how many times the loop will iterate:
  - use a **`for`** loop.
- If you don't know how many times the loop will iterate:
  - If its possible to iterate 0 times, use a **`while`** loop
  - If it will always iterate at least once, use a **`do-while`** loop
- Generally, a **`while`** loop is a safe choice.

Designing Loops

Designing Loops: Initializing Statements

- Some variables should have a value before the loop begins i.e.
  - Summing a list of numbers (sum should initialize to 0)
  - Product a list of numbers (product should initialize to 1)
  - Generating a String (string should be initialized to "" (empty string))
- Other variables get values only while the loop is iterating
  - counter control variables
- Identify which variables should be initialized before the loop, and which can be initialized inside the loop.

Designing a Loop: The Loop Body

- Write out the actions the code should accomplish

- Look for a repeated pattern:
  - the repeated pattern will be the body of the loop
  - the pattern does not have to start with the first action
    - i.e. using a **`while`** loop and getting user input, the first action (first user input request) can be outside the loop.
  - some actions might have to be done after the pattern stops repeating
    - i.e. printing out the final sum

Designing a Loop: Ending the Loop

- If the number of iterations is known before the loop starts, the loop is a ***count-controlled loop***
  - use a **`for`** loop.

- Asking the user before each iteration if it is time to end the loop is an ***ask-before-iterating technique***.
  - appropriate for a small number of iterations
  - use a **`while`** or **`do-while`** loop

- For large input lists, a ***sentinel value*** can be used to end the list and the loop
  - sentinel should be different from all other possible inputs
  - i.e. a negative number after a long list of exam scores

Loop Design: Summary

- Questions you should be able to answer
  - What initialization is necessary for the loop's test expression?
  - What initialization is necessary for the loop's processing?
  - What causes the loop to terminate?
  - What actions should the loop perform?
  - What actions are necessary to prepare for the next iteration of the loop?
  - What conditions are true and what conditions are false when the loop is terminated?

Nested Loops

Nested Loops

- 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
- Example 1: print out the following pattern (3 lines of 8 * each)

```
********

********

********
```

- you are only allowed to use the following two output statements:
  - `System.out.print( "*" );`
  - `System.out.println();`

Nested Loops

```
for (int i = 0; i < 3; i++){
    for (int j = 0; j < 8; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```

Exercises

- Show the output of the following code segments:

```
1.
for (int i = 1; i < 5; i++){
  int j = 0;
  while (j < i){
    System.out.print(j + " ");
    j++;
  }
  System.out.println();
}
```

```
2.
int i= 0;
while (i < 5){
  for (int j = i; j > 1; j--){
    System.out.print(j + " ");
  }
  System.out.println("****");
  i++;
}
```

The `break` and `continue` Statements

The `break` and `continue` Statements

- Two keywords, **`break`** and **`continue`**, can by used in loop statements to provide additional control

- **`break`**: immediately ends the innermost loop that contains it.
  - break breaks out of a loop entirely

- **`continue`**: ends only the current iteration
  - Program control goes to the end of the loop body
  - Continue breaks out of an iteration

- Usually, the two keywords are used with an if statement to determine when they should be triggered.

- Code Examples:

  `TestBreak.java`

  `TestContinue.java`

break and continue Exercises

- Show the output of the two code segments:

```
for (int i = 1; i < 4; i++){
    for (int j = 1; j < 4; j++){
        if ((i + j) == 2)
            break;
        System.out.println((i+j));
    }
}
```

```
for (int i = 1; i < 4; i++){
    for (int j = 1; j < 4; j++){
        if ((i + j) == 2 )
            continue;
        System.out.println((i+j));
    }
}
```