

Walter Savitch
Frank M. Carrano

Flow of Control: Loops

Chapter 4

Java Loop Statements: Outline

- The **while** statement
- The **do-while** statement
- The **for** Statement

Java Loop Statements

- A portion of a program that repeats a statement or a group of statements is called a *loop*.
- The statement or group of statements to be repeated is called the *body* of the loop.
- A loop could be used to compute grades for each student in a class.
- There must be a means of exiting the loop.

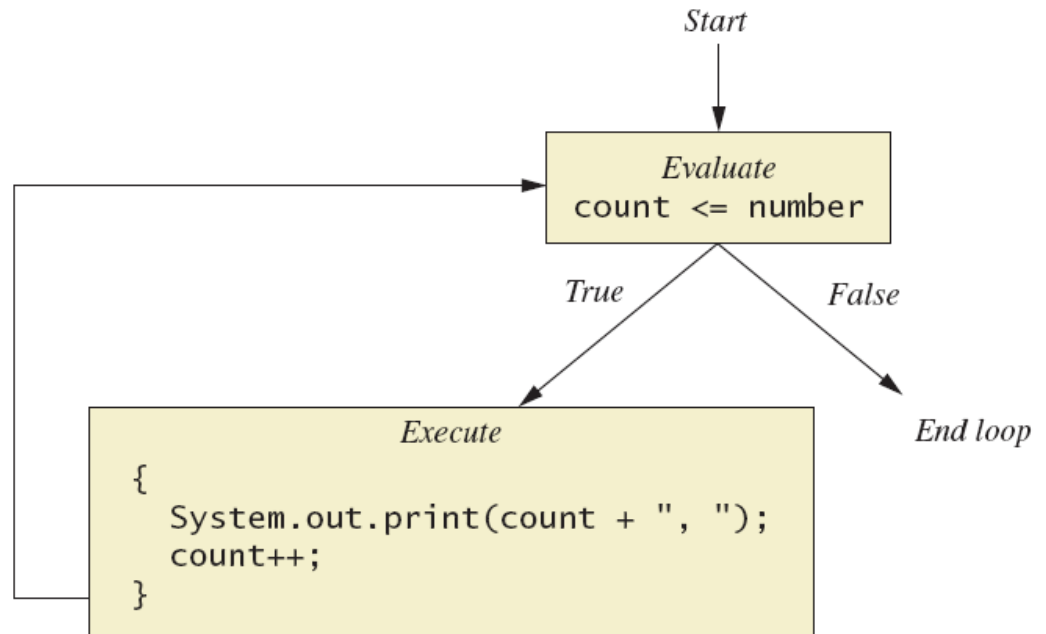
The **while** Statement

- Also called a **while** loop
- A **while** statement repeats while a controlling boolean expression remains true
- The loop body typically contains an action that ultimately causes the controlling boolean expression to become false.

The `while` Statement

- Download, compile, and run `WhileDemo`

```
while (count <= number)
{
    System.out.print(count + ", ");
    count++;
}
```



The *while* Statement

- Syntax

```
while (Boolean_Expression)  
{  
    First_Statement  
    Second_Statement  
    ...  
}
```

The **do-while** Statement

- Also called a **do-while** loop
- Similar to a **while** statement, except that the loop body is executed at least once

- Syntax

do

{

Body_Statement(s)

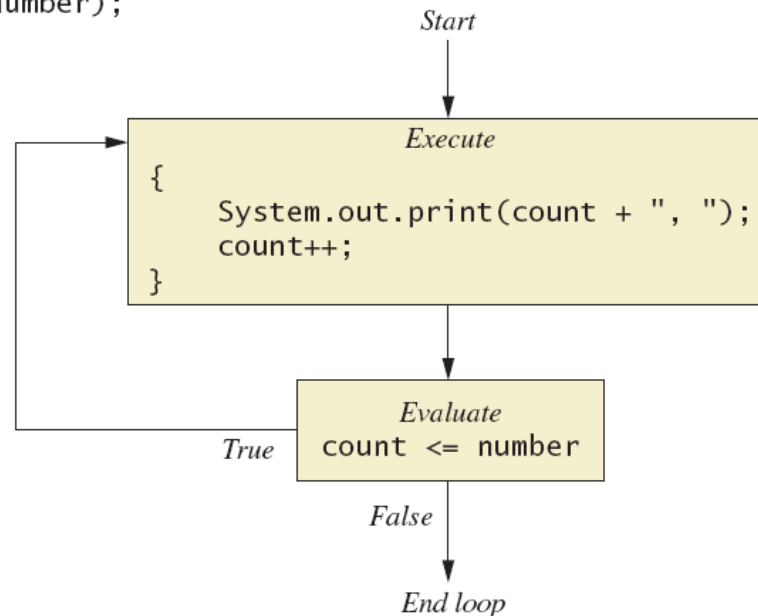
} while (Boolean_Expression);

- Don't forget the semicolon!

The **do-while** Statement

- Figure 4.3 The action of the **do-while** loop in **DoWhileDemo**

```
do
{
    System.out.print(count + ", ");
    count++;
} while (count <= number);
```



The **do-while** Statement

- First, the loop body is executed.
- Then the boolean expression is checked.
 - As long as it is true, the loop is executed again.
 - If it is false, the loop is exited.
- Equivalent **while** statement

```
Statement(s)_S1  
while (Boolean_Condition) {  
    Statement(s)_S1  
}
```

Using **while** to Read Input

- Download **whileWithScanner.java** from the Examples link on the course webpage
- Compile and run a few times until you are comfortable with what the program does and how it does it.

Exercise

- Modify **whileWithScanner.java** so that it reads one line of doubles, prints the intermediate results, and finally prints the sum for the entire line.
 - Read a whole line from keyboard into a String
 - Create a Scanner object on that String
 - Proceed as before, but with the new Scanner

Infinite Loops

- A loop which repeats without ever ending is called an *infinite loop*.
- If the controlling boolean expression never becomes false, a **while** loop or a **do-while** loop will repeat without ending.
- Make sure a loop contains a statement which causes the boolean expression to become false eventually.

Nested Loops

- The body of a loop can contain any kind of statements, including another loop.

Exercise

- Modify `WhileWithScanner.java`
 - Add an outer loop that reads an entire line of input
 - The inner loop calculates the sum for the current line of input

The **for** Statement

- A **for** statement executes the body of a loop a fixed number of times.
- Example

```
for (count = 1; count < 3; count++)  
{  
    System.out.println(count);  
}
```

The **for** Statement

- Syntax

*for (Initialization; Condition; Update)
Body_Statement*

- **Body_Statement** can be either a simple statement or a compound statement in **{ }**.

- Corresponding **while** statement

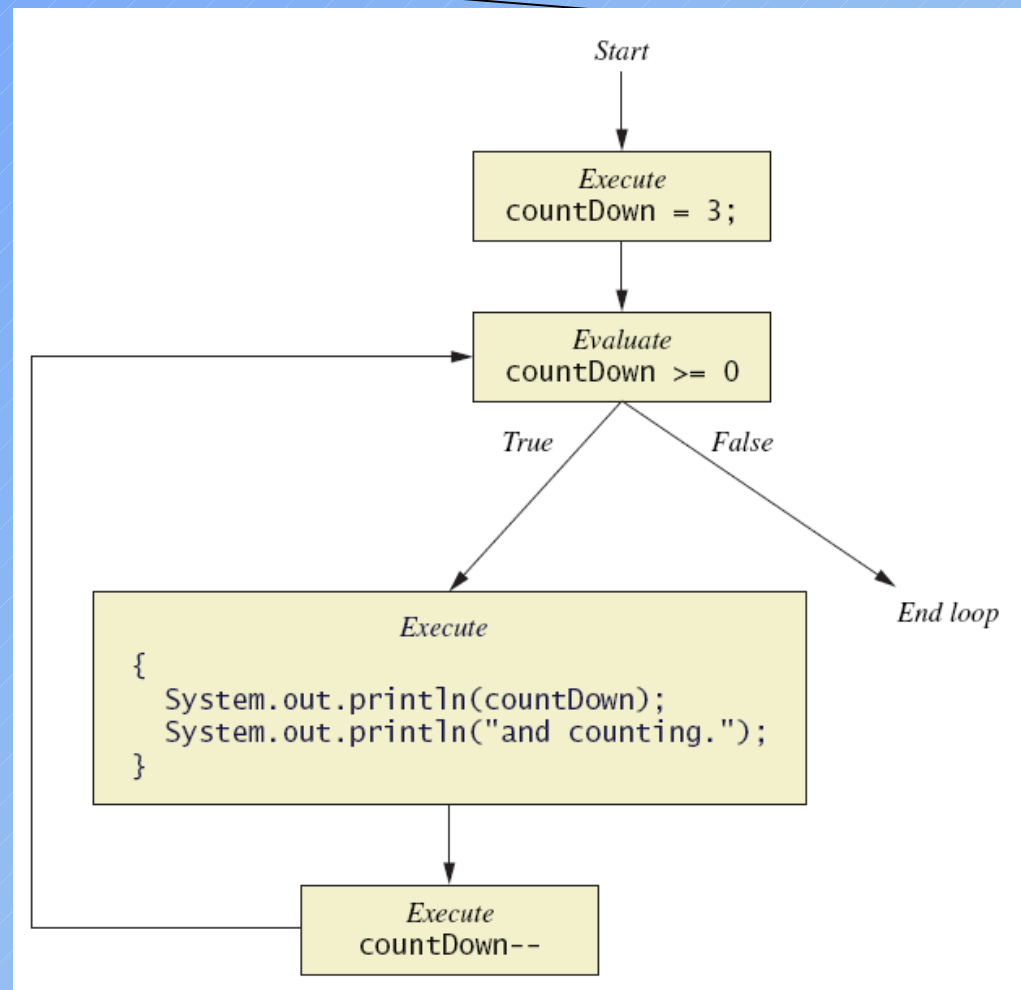
Initialization

while (Condition)

Body_Statement_Including_Update

The for Statement

- Download **ForDemo**



```
for (countDown = 3; countDown >= 0; countDown--)  
{  
    System.out.println(countDown);  
    System.out.println("and counting.");  
}
```

The **for** Statement

- Possible to declare variables within a **for** statement

```
int sum = 0;
for (int n = 1 ; n <= 10 ; n++)
{
    sum = sum + n;
}
```

- Note that variable **n** is local to the loop

Exercise

- Write a program called ReverseWords that reads lines of text from the keyboard and prints the line backwards
- Example:
 - input: what's up?
 - output: ?pu s'tahw

Controlling Number of Loop Iterations

- If the number of iterations is known before the loop starts, the loop is called a *count-controlled loop*.
 - Use a **for** loop.
- Asking the user before each iteration if it is time to end the loop is called the *ask-before-iterating technique*.
 - Appropriate for a small number of iterations
 - Use a **while** loop or a **do-while** loop.


The **break** Statement in Loops

- A **break** statement can be used to end a loop immediately.
- The **break** statement ends only the **innermost** loop or switch statement that contains the **break** statement.
- **break** statements make loops more difficult to understand.
- Use **break** statements sparingly (if ever).

The **break** Statement in Loops

- Note program fragment, ending a loop with a **break** statement, listing 4.8

```
while (itemNumber <= MAX_ITEMS)
{
    . . .
    if (itemCost <= leftToSpend)
    {
        . . .
        if (leftToSpend > 0)
            itemNumber++;
        else
        {
            System.out.println("You are out of money.");
            break;
        }
    }
    else
        . . .
}
System.out.println( . . . );
```



The **continue** Statement in Loops

- A **continue** statement
 - Ends current loop iteration
 - Begins the next one
- Text recommends avoiding use
 - Introduces unneeded complications

Tracing Variables

- *Tracing variables* means watching the variables change while the program is running.
 - Simply insert temporary output statements in your program to print of the values of variables of interest
 - Or, learn to use the debugging facility that may be provided by your system.

Tracing Variables with Drjava

- Open `WhileWithScanner.java`
- Choose **Debugger->Debug Mode**
 - A new panel is added with Stack and Thread tabs
- Right-click on the code `sum += num;` and choose **Toggle Breakpoint**
 - That line now has a red background

Tracing Variables with Drjava

- Click on the left-right arrows to the left of the Stack tab
 - A new panel is added with a **Watches** tab
- Type num into the Name field and hit return
- Type sum into the next Name field, then return
- Run the program and type some numbers when prompted to do so
- The program will stop at the breakpoint and show the values of num and sum

Tracing Variables with Drjava

- Click the **Step Over** button to move to the next statement
- Click the **Resume** button to move to the next breakpoint
- To exit the debugger and the program, click the **Reset** button located on the top panel

Loop Bugs

- Common loop bugs
 - Unintended infinite loops
 - Off-by-one errors
 - Testing equality of floating-point numbers
- Subtle infinite loops
 - The loop may terminate for some input values, but not for others.
 - Example: you can't get out of debt when the monthly penalty exceeds the monthly payment.