



Walter Savitch

# 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.
- Example: 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

- Syntax

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

# The **while** Statement

- Download, compile, and run **WhileDemo.java**

Enter a number:

2

1, 2,

Buckle my shoe.

Sample  
screen  
output

Enter a number:

3

1, 2, 3,

Buckle my shoe.

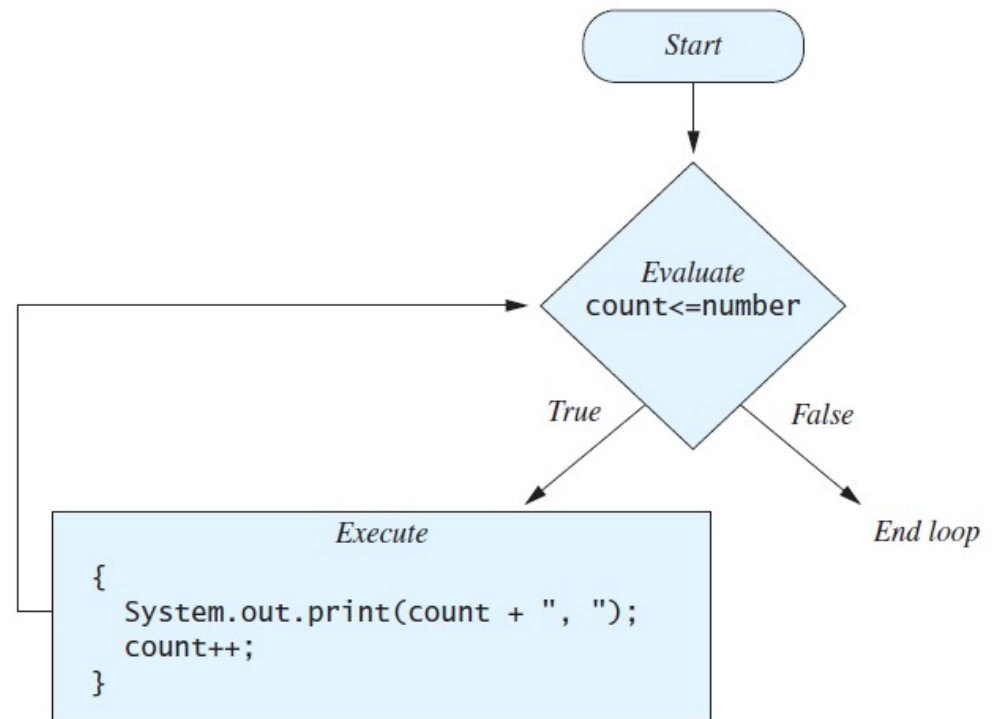
Enter a number:

0

Buckle my shoe.

The loop body is  
iterated zero times.

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



# 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
```

```
} while (Boolean_Expression);
```

- Don't forget the semicolon!

# The **do-while** Statement

- View **DoWhileDemo.java**

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

Enter a number:

2

1, 2,

Buckle my shoe.

Enter a number:

3

1, 2, 3,

Buckle my shoe.

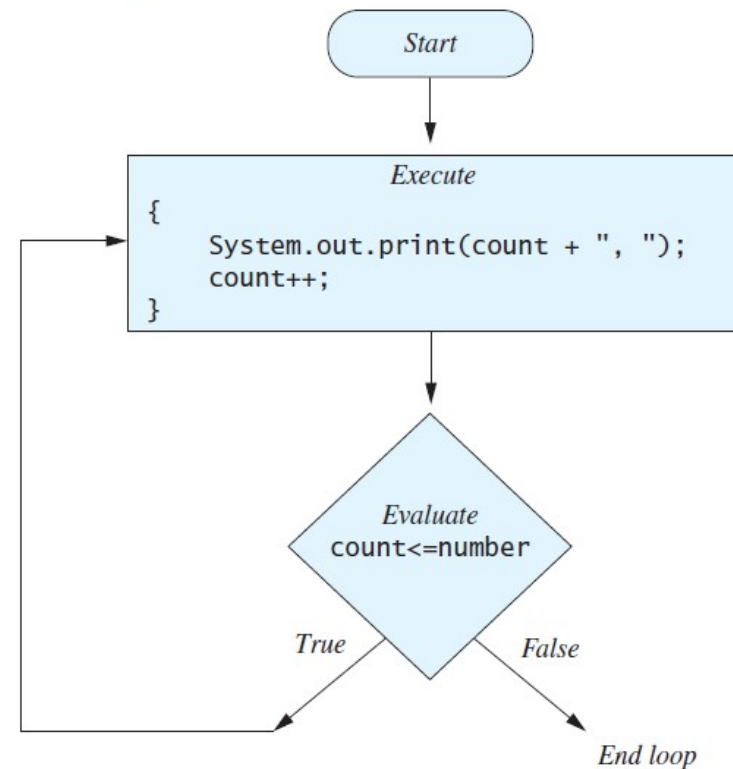
Sample  
screen  
output

Enter a number:

0

1, ←  
Buckle my shoe.

The loop body always  
executes at least once.





# 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.
- View **ExamAverager.java**

```
Want to average another exam?  
Enter yes or no.  
yes  
  
Enter all the scores to be averaged.  
Enter a negative number after  
you have entered all the scores.  
90  
70  
80  
-1  
The average is 80.0  
Want to average another exam?  
Enter yes or no.  
no
```

Sample  
screen  
output

# 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*



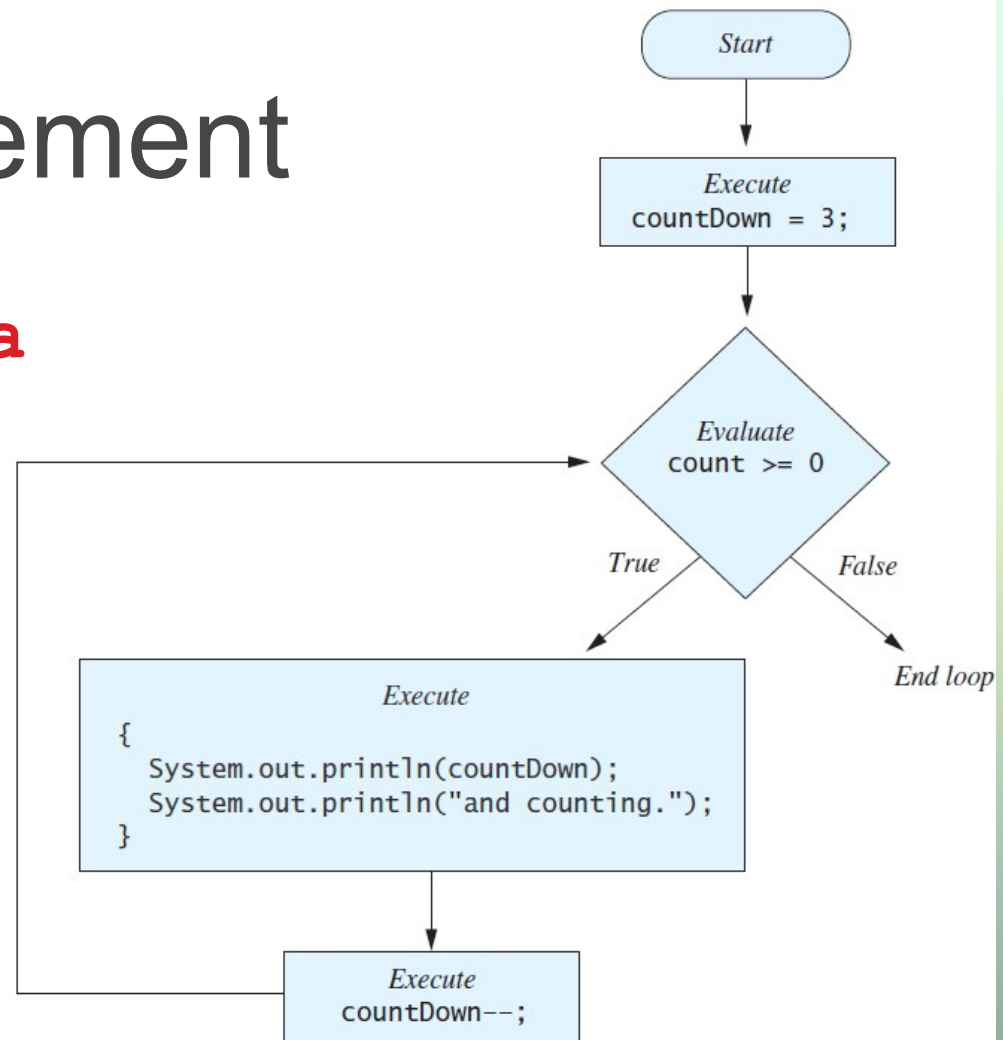
# The **for** Statement

- View **ForDemo.java**

```
3  
and counting.  
2  
and counting.  
1  
and counting.  
0  
and counting.  
Blast off!
```

Sample  
screen  
output

```
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 * n;
}
```

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

# for 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

# for Exercise

```
public class ReverseWords {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        System.out.print("Enter text to be reversed: ");
        String line = keyboard.nextLine();
        for (int i = line.length()-1; i >= 0; i--) {
            System.out.print(line.charAt(i));
        }
        System.out.println("");
    }
}
```

# The **for-each** Statement

- Possible to step through values of an enumeration type
- Example

```
enum Suit {CLUBS, DIAMONDS, HEARTS, SPADES}  
for (Suit nextSuit : Suit.values())  
{  
    System.out.print(nextSuit + " ");  
}  
System.out.println();
```

# 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.

# Controlling Number of Loop Iterations

- For large input lists, a *sentinel value* can be used to signal the end of the list.
  - The sentinel value must be different from all the other possible inputs.
  - A negative number following a long list of nonnegative exam scores could be suitable.

90

0

10

-1

# Controlling Number of Loop Iterations

- Example: reading a list of scores followed by a sentinel value

```
int next = keyboard.nextInt();  
while (next >= 0)  
{  
    Process_The_Score  
    next = keyboard.nextInt();  
}
```



# Controlling Number of Loop Iterations

- Using a boolean variable to end the loop
- View **BooleanDemo.java**

```
Enter nonnegative numbers.  
Place a negative number at the end  
to serve as an end marker.
```

```
1 2 3 -1
```

```
The sum of the numbers is 6
```

Sample  
screen  
output

# for Exercise

- Modify **ReverseWords.java** so that it reads one line of text, prints the reverse order of that line, and then reads the next line etc.
- The program will read and reverse new lines until the user enters only a period (sentinel value) on one line.

# for Exercise

```
public class ReverseWords {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        String line = "";
        boolean inputContinues = true;

        while (inputContinues) {
            System.out.print("Enter text to be reversed "
                + "(period ends program): ");
            line = keyboard.nextLine();
            if (!line.equals(".")) {
                for (int i = line.length()-1; i >= 0; i--) {
                    System.out.print(line.charAt(i));
                }
                System.out.println("");
            } else {
                inputContinues = false;
            }
        }
    }
}
```


# 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
  - Introduce 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.
  - For example, you can't get out of debt when the monthly penalty exceeds the monthly payment.