

# JAVA<sup>™</sup> PROGRAMMING



## Chapter 6: Looping

### Objectives

- Learn about the loop structure
- Create `while` loops
- Use shortcut arithmetic operators
- Create `for` loops
- Create `do...while` loops
- Nest loops
- Improve loop performance



## Learning About the Loop Structure

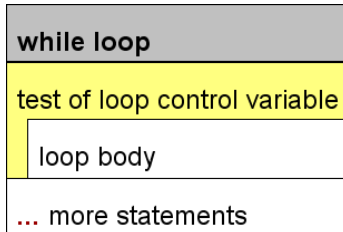
- **Loop**
  - A structure that allows repeated execution of a block of statements
- **Loop body**
  - A block of statements
  - Executed repeatedly
- **Iteration**
  - One execution of any loop



## Learning About the Loop Structure (cont'd.)

- Three types of loops
  - `while`
    - The loop-controlling Boolean expression is the first statement
  - `for`
    - A concise format in which to execute loops
  - `do...while`
    - The loop-controlling Boolean expression is the last statement

## Learning About the Loop Structure (cont'd.)



A while loop structure (is general to most looping structures)

## Creating while Loops

- **while loop**
  - Executes a body of statements continually
    - As long as the Boolean expression that controls entry into the loop continues to be `true`
  - Consists of:
    - The keyword `while`
    - Followed by a Boolean expression within parentheses
    - Followed by the body of the loop; can be a single statement or a block of statements surrounded by curly braces

## Writing a Definite `while` Loop

- **Definite loop**
  - Performs a task a predetermined number of times
  - Also called a counted loop
- Write a definite loop
  - Initialize the loop control variable
    - The variable whose value determines whether loop execution continues
  - While the loop control variable does not pass a limiting value, the program continues to execute the body of the `while` loop

## Writing a Definite `while` Loop (cont'd.)

while loop
LIMIT = 11
lcv = 1
lcv < LIMIT
output lcv
lcv = lcv + 1
... more statements

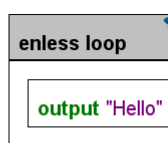
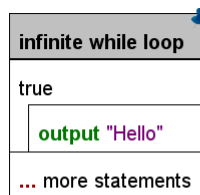
```
int lcv;  
final int LIMIT = 11;  
  
lcv = 1;  
while(lcv < LIMIT) {  
    System.out.println(lcv);  
    lcv = lcv + 1;  
}
```

A `while` loop that displays the integers 1 through 10

## Writing a Definite `while` Loop (cont'd.)

- Write a definite loop (cont'd.)
  - The body of the loop must include a statement that alters the loop control variable
- **Infinite loop**
  - A loop that never ends
  - Can result from a mistake in the `while` loop
  - Do not write intentionally

## Writing a Definite `while` Loop (cont'd.)



*Don't Do It*  
This loop will never end because the tested expression is always true

```
while(true) {  
    System.out.println("Hello");  
}
```

A `while` loop that displays "Hello" infinitely

## Writing a Definite `while` Loop (cont'd.)

- Suspect an infinite loop when:
  - The same output is displayed repeatedly
  - The screen remains idle for an extended period of time
- To exit an infinite loop, press and hold Ctrl, then press C or Break

## Pitfall: Failing to Alter the Loop Control Variable Within the Loop Body

- Prevent the `while` loop from executing infinitely
  - The named loop control variable is initialized to a starting value
  - The loop control variable is tested in the `while` statement
  - If the test expression is `true`, the body of the `while` statement takes action
    - Alters the value of the loop control variable
  - The test of the `while` statement must eventually evaluate to `false`

## Pitfall: Failing to Alter the Loop Control Variable Within the Loop Body (cont'd.)

This indentation has no effect

```
loopCount = 1;
while(loopCount < 3)
    System.out.println("Hello");
    loopCount = loopCount + 1;
```

### infinite while loop (poor coding practice)

loopCount = 1

loopCount < 3

output "Hello"

loopCount = loopCount + 1

... more statements

**Don't Do It**  
Loop control variable is not altered in the loop

A while loop that displays "Hello" infinitely because loopCount is not altered in the loop body

## Pitfall: Unintentionally Creating a Loop with an Empty Body

- **Loop control variable**
  - A variable that is altered and stored with a new value

```
loopCount = loopCount + 1
```

    - The equal sign assigns a value to the variable on the left
  - The variable should be altered within the body of the loop
- **Empty body**
  - A body with no statements
  - Caused by misplaced semicolons

## Altering a Definite Loop's Control Variable

- **Incrementing** the variable
  - Alter the value of the loop control variable by adding 1
- **Decrementing** the variable
  - Subtract 1 from the loop control variable
- **Clearest and best method**
  - Start the loop control variable at 0 or 1
  - Increment by 1 each time through the loop
  - Stop when the loop control variable reaches the limit

## Altering a Definite Loop's Control Variable (cont'd.)

```
loopCount = 3;  
while(loopCount > 1)  
{  
    System.out.println("Hello");  
    loopCount = loopCount - 1;  
}
```

**Figure 6-7** A while loop that displays "Hello" twice, decrementing the loopCount variable in the loop body



## Writing an Indefinite `while` Loop

- Indefinite loop
  - Altered by user input
    - Controlled by the user
    - Executed any number of times
- Validating data
  - Ensure a value falls within a specified range
  - Use indefinite loops to validate input data
  - If a user enters incorrect data, the loop repeats

## Writing an Indefinite `while` Loop (cont'd.)

```
import java.util.Scanner;
public class BankBalance
{
    public static void main(String[] args){
        double balance;
        int response;
        int year = 1;
        final double INT_RATE = 0.03;
        Scanner keyboard = new Scanner(System.in);
        System.out.print("Enter initial bank balance > ");
        balance = keyboard.nextDouble();
        System.out.println("Do you want to see next year's balance?");
        System.out.print("Enter 1 for yes");
        System.out.print(" or any other number for no >> ");
        response = keyboard.nextInt();
        while(response == 1){
            balance = balance + balance * INT_RATE;
            System.out.println("After year " + year + " at " + INT_RATE +
                " interest rate, balance is $" + balance);
            year = year + 1;
            System.out.println("\nDo you want to see the balance " +
                "at the end of another year?");
            System.out.print("Enter 1 for yes");
            System.out.print(" or any other number for no >> ");
            response = keyboard.nextInt();
        }
    }
}
```

Figure 6-8 The BankBalance application



## Validating Data

- Ensuring data falls within a specific range
- **Priming read**
  - Input retrieved before the loop is entered
  - Within a loop, the last statement retrieves the next input value and checks the value before the next entrance of the loop

## Validating Data (cont'd.)

```
import java.util.Scanner;
public class EnterSmallValue{
    public static void main(String[] args){
        int userEntry;
        final int LIMIT = 3;
        Scanner input = new Scanner(System.in);
        System.out.print("Please enter an integer no higher than " +
            LIMIT + " > ");
        userEntry = input.nextInt();
        while(userEntry > LIMIT){
            System.out.println("The number you entered was too high");
            System.out.print("Please enter an integer no higher than " +
                LIMIT + " > ");
            userEntry = input.nextInt();
        }
        System.out.println("You correctly entered " + userEntry);
    }
}
```

Figure 6-8 The EnterSmallValue application

## Using Shortcut Arithmetic Operators

- **Accumulating**
  - Repeatedly increasing a value by some amount
- Java provides shortcuts for incrementing and accumulating
  - += add and assign operator**
  - = subtract and assign operator**
  - \*= multiply and assign operator**
  - /= divide and assign operator**
  - %= remainder and assign operator**

## Using Shortcut Arithmetic Operators (cont'd.)

- **Prefix increment operator and postfix increment operator**
  - `++someValue, someValue++`
  - Use only with variables
  - Unary operators
    - Use with one value
  - Increase a variable's value by 1
    - No difference between operators (unless other operations are in the same expression)

## Using Shortcut Arithmetic Operators (cont'd.)

```
int value;  
  
value = 24;  
++value; // Result: value is 25  
          // (incremented and then used)  
  
value = 24;  
value++; // Result: value is 25  
          // (value is used then incremented)  
  
value = 24;  
value = value + 1; // Result: value is 25  
  
value = 24;  
value += 1; // Result: value is 25
```

Figure 6-13 Four ways to add 1 to a value

## Using Shortcut Arithmetic Operators (cont'd.)

- **Prefix increment operator and postfix increment operator (cont'd.)**
  - **Prefix ++**
    - The result is calculated and stored
    - Then the variable is used
  - **Postfix ++**
    - The variable is used
    - Then the result is calculated and stored
- **Prefix and postfix decrement operators**
  - someValue
  - someValue--
  - Similar logic to increment operators

## Using Shortcut Arithmetic Operators (cont'd.)

```
public class PrefixPostfixDemo{
    public static void main(String[] args){
        int myNumber, answer;
        myNumber = 17;
        System.out.println("Before incrementing, myNumber is " +
            myNumber);
        answer = ++myNumber;
        System.out.print("After prefix increment, myNumber is " +
            myNumber);
        System.out.println(" and answer is " + answer);
        myNumber = 17;
        System.out.println("Before incrementing, myNumber is " +
            myNumber);
        answer = myNumber++;
        System.out.print("After postfix increment, myNumber is " +
            myNumber);
        System.out.println(" and answer is " + answer);
    }
}
```

```
Before incrementing, myNumber is 17
After prefix increment, myNumber is 18 and answer is 18
Before incrementing, myNumber is 17
After postfix increment, myNumber is 18 and answer is 17
```

Figure 6-14 The PrefixPostfixDemo application

## Creating a for Loop

- **for loop**
  - Used when a definite number of loop iterations is required
  - One convenient statement indicates:
    - The starting value for the loop control variable
    - The test condition that controls loop entry
    - The expression that alters the loop control variable

## Creating a `for` Loop (cont'd.)

```
int val;
for(val = 1; val < 11; val++){
    System.out.println(val);
}

val = 1;
while(val < 11){
    System.out.println(val);
    val++;
}
```

Figure 6-18 A `for` loop and a `while` loop that displays the integers 1 through 10

## Creating a `for` Loop (cont'd.)

- Other uses for the three sections of a `for` loop
  - Initialization of more than one variable
    - Place commas between separate statements
  - Performance of more than one test using AND or OR operators
  - Decrementing or performance of some other task
  - Altering more than one value
- You can leave one or more portions of a `for` loop empty
  - Two semicolons are still required as placeholders

## Creating a `for` Loop (cont'd.)

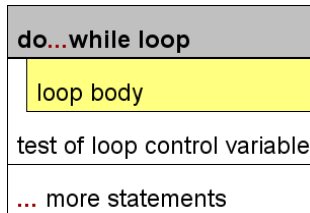
- Use the same loop control variable in all three parts of a `for` statement
- To pause a program:
  - Use the `for` loop that contains no body (**do-nothing** loop)  

```
for(x = 0; x < 100000; x++);
```
  - Or use the built-in `sleep()` method

## Learning How and When to Use a `do...while` Loop

- **`do...while` loop**
  - A **posttest loop**
  - Checks the value of the loop control variable
    - At the bottom of the loop
    - After one repetition has occurred
  - Performs a task at least one time
  - You are never required to use this type of loop
  - Use curly braces to block the statement
    - Even with a single statement

## Learning How and When to Use a do...while Loop (cont'd.)



General structure of a do...while loop

## Learning How and When to Use a do...while Loop (cont'd.)

```
import java.util.Scanner;
public class BankBalance2{
    public static void main(String[] args){
        double balance;
        int response;
        int year = 1;
        final double INT_RATE = 0.03;
        Scanner keyboard = new Scanner(System.in);
        System.out.print("Enter initial bank balance >> ");
        balance = keyboard.nextDouble();
        do{
            balance = balance + balance * INT_RATE;
            System.out.println("After year " + year + " at " + INT_RATE +
                " interest rate, balance is $" + balance);
            year = year + 1;
            System.out.println("\nDo you want to see the balance " +
                "at the end of another year?");
            System.out.print("Enter 1 for yes");
            System.out.print(" or any other number for no >> ");
            response = keyboard.nextInt();
        } while(response == 1);
    }
}
```

Figure 6-23 A do...while loop for the BankBalance2 application

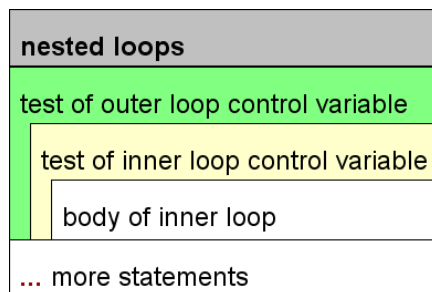


## Learning About Nested Loops

- **Inner loop and outer loop**
  - An inner loop must be entirely contained in an outer loop
  - Loops can never overlap
- To print three mailing labels for each of 20 customers:

```
for(customer = 1; customer <= 20; customer++)  
    for(color = 1; color <= 3; color++)  
        outputLabel ();
```

## Learning About Nested Loops (cont'd.)



Nested loops



## Improving Loop Performance

- Make sure a loop does not include unnecessary operations or statements
- Consider the order of evaluation for short-circuit operators
- Make comparisons to zero (0)
- Employ loop fusion to combine loops



## Avoiding Unnecessary Operations

- Do not use unnecessary operations or statements:
  - Within a loop's tested expression
  - Within the loop body

- **Avoid:**

```
while (x < a + b)
// loop body
```

- **Instead use:**

```
int sum = a + b;
while(x < sum)
// loop body
```

## Considering the Order of Evaluation of Short-Circuit Operators

- Short-circuit evaluation
  - Each part of an AND or an OR expression is evaluated only as much as necessary to determine the value of the expression
- Important to consider the number of evaluations that take place
  - When a loop might execute many times

## Comparing to Zero

- Making a comparison to zero (0) is faster than making a comparison to any other value
- To improve loop performance, compare the loop control variable to zero (0)
- Do-nothing loop
  - Performs no actions other than looping

## Comparing to Zero (cont'd.)

```
import java.time.*;

public class CompareLoopTimes {
    public static void main(String[] args) {
        int startTime, endTime;
        final int REPEAT = 100_000;
        final int FACTOR = 1_000_000;
        LocalDateTime now;
        now = LocalDateTime.now();
        startTime = now.getNano();
        for(int x = 0; x <= REPEAT; x++)
            for(int y = 0; y <= REPEAT; y++);
        now = LocalDateTime.now();
        endTime = now.getNano();
        System.out.println("Time for loops starting from 0: " +
            (endTime - startTime) / FACTOR + " milliseconds");
        now = LocalDateTime.now();
        startTime = now.getNano();
        for(int x = REPEAT; x >= 0; x--)
            for(int y = REPEAT; y >= 0; y--);
        now = LocalDateTime.now();
        endTime = now.getNano();
        System.out.println("Time for loops ending with 0: " +
            (endTime - startTime) / FACTOR + " milliseconds");
    }
}
```

```
----jGRASP wedge2: ubuntu: Command Shell [ 511x300px 100% ]
Time for loops starting from 0: 47 milliseconds
Time for loops ending with 0: 8 milliseconds
```

Figure 6-29 The CompareLoopTimes application

## Employing Loop Fusion

- **Loop fusion**
  - A technique of combining two loops into one
  - Will not work in every situation



## Don't Do It

- Don't insert a semicolon at the end of a `while` clause
- Don't forget to block multiple statements that should execute in a loop
- Don't make the mistake of checking for invalid data using a decision instead of a loop
- Don't ignore subtleties in the boundaries used to stop loop performance
- Don't repeat steps within a loop that could just as well be placed outside the loop

Java Programming, Eighth Edition

46



## Summary

- The loop structure allows repeated execution of a block of statements
  - Infinite loop
  - Definite loop
  - Nest loop
- You must change the loop control variable within the looping structure
- Use the `while` loop to execute statements while some condition is `true`

Java Programming, Eighth Edition

47



## Summary (cont'd.)

- Execute the `while` loop
  - Initialize the loop control variable, test in the `while` statement, and alter the loop control variable
- Prefix `++` and postfix `++`
  - Increase a variable's value by 1
  - The variable is used
    - The result is calculated and stored
- Unary operators
  - Use with one value



## Summary (cont'd.)

- Binary operators
  - Operate on two values
- Shortcut operators `+=`, `-=`, `*=`, and `/=`
  - Perform operations and assign the result in one step
- `for` loop
  - Initializes, tests, and increments in one statement
- `do...while` loop
  - Tests a Boolean expression after one repetition
- Improve loop performance
  - Do not include unnecessary operations or statements