NCP2103: Object-Oriented Programming
(Java Programming)

Looping

Module 6

Errol John M. Antonio
Assistant Professor
Computer Engineering Department
University of the East – Manila Campus

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
  - Structure that allows repeated execution of a block of statements
- Loop body
  - Block of statements
  - Executed repeatedly
- Iteration
  - One execution of any loop

Learning About the Loop Structure (cont'd.)

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

Learning About the Loop Structure (cont'd.)

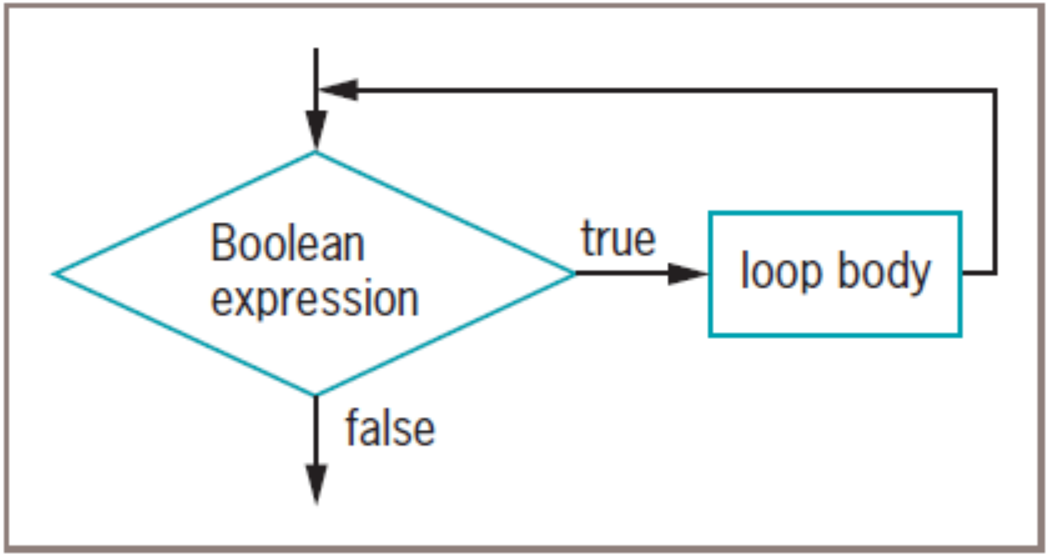


Figure 6-1 Flowchart of a loop structure

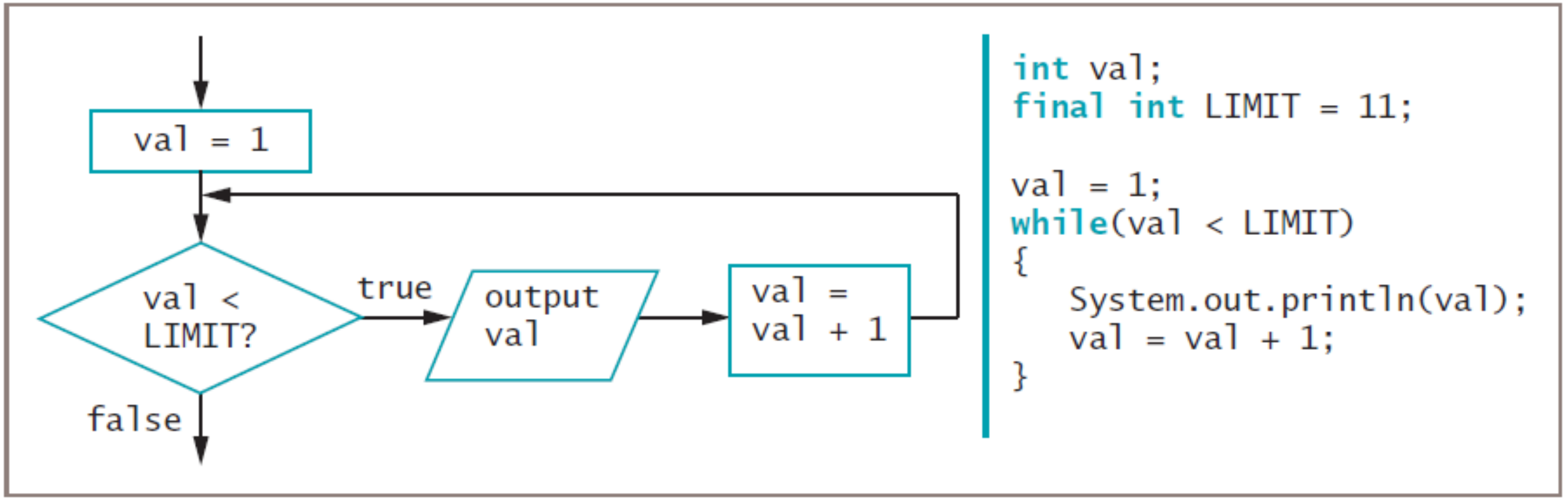
Creating `while` Loops

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

Writing a Definite `while` Loop

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

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



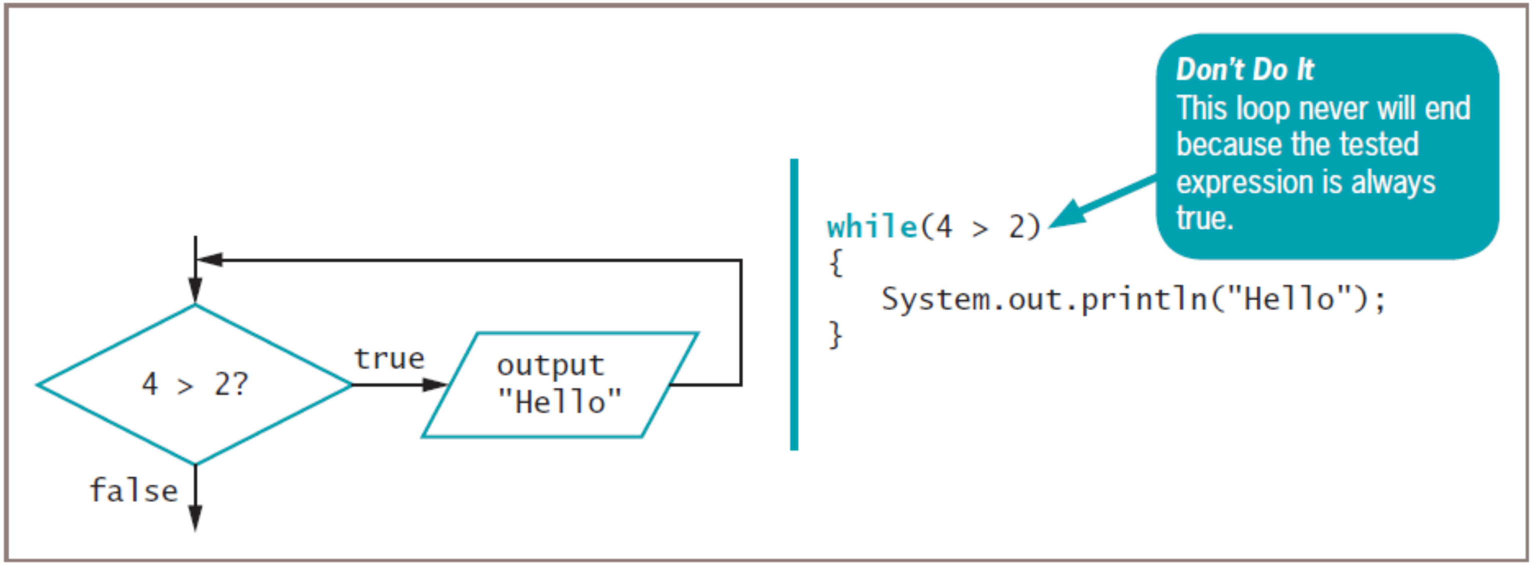
```
int val;
final int LIMIT = 11;

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

Figure 6-2 A `while` loop that displays the integers 1 through 10

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

- Write a definite loop (cont'd.)
  - Body of loop
    - Must include statement that alters loop control variable
- Infinite loop
  - Loop that never ends
  - Can result from mistake in `while` loop
  - Do not write intentionally

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



```
while(4 > 2)
{
   System.out.println("Hello");
}
```

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

Figure 6-3 A loop that displays “Hello” infinitely

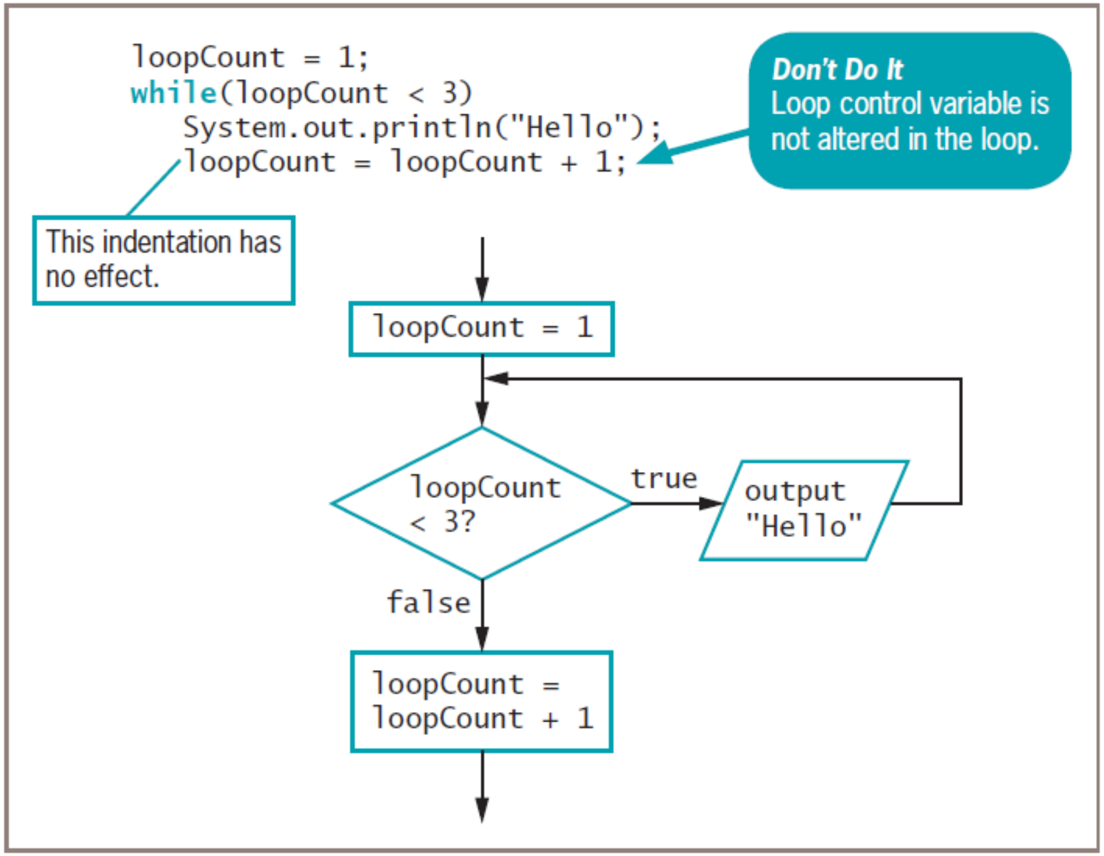
Writing a Definite `while` Loop (cont'd.)

- Suspect infinite loop
  - Same output displayed repeatedly
  - Screen remains idle for extended period of time
- Exit from infinite loop
  - Press and hold Ctrl
    - Press C or Break

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

- Prevent `while` loop from executing infinitely
  - Named loop control variable initialized to starting value
  - Loop control variable tested in `while` statement
  - If test expression `true`
    - Body of `while` statement takes action
      - Alters value of loop control variable
    - Test of `while` statement must eventually evaluate to `false`

Writing a Definite while Loop (cont'd.)



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

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

This indentation has no effect.

Figure 6-5 A while loop that displays "Hello" infinitely because loopCount is not altered in the loop body

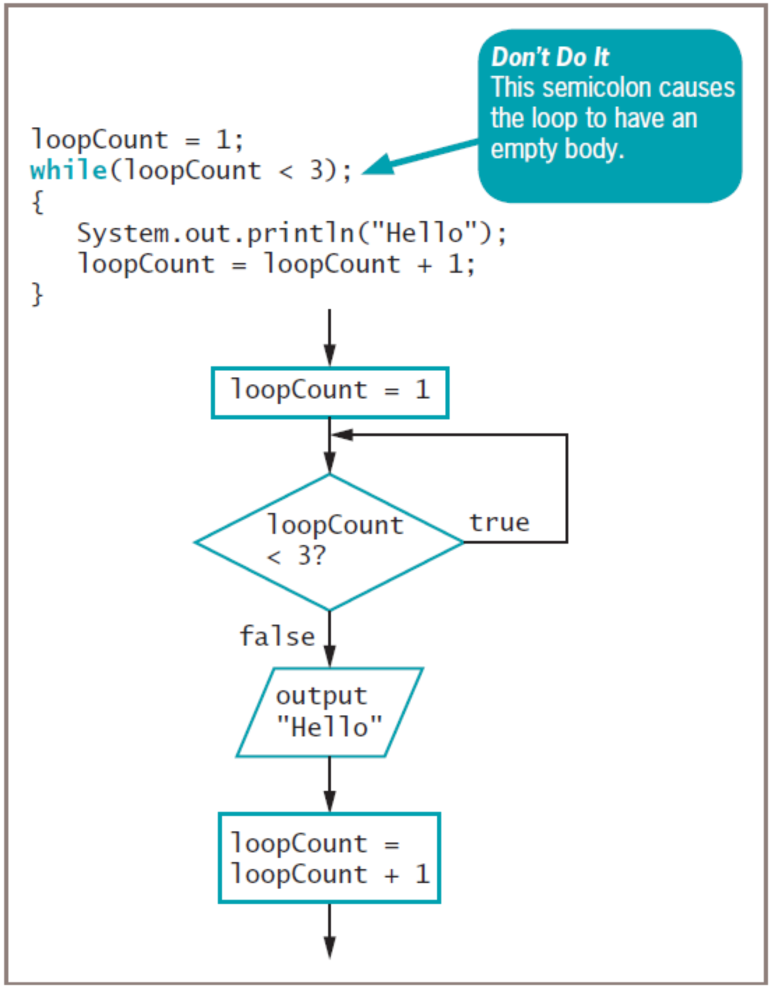
Writing a Definite `while` Loop (cont'd.)

- Loop control variable
  - Variable altered and stored with new value

```
loopCount = loopCount + 1
```

    - Equal sign assigns value to variable on left
  - Variable should be altered within body of loop
- Empty body
  - Body with no statements
  - Caused by misplaced semicolons

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



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

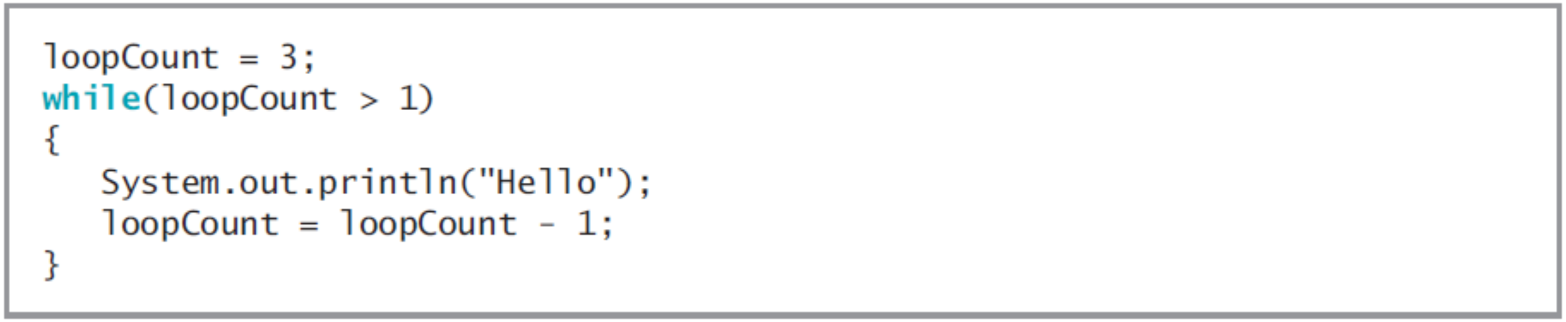
Don't Do It
This semicolon causes the loop to have an empty body.

Figure 6-6 A while loop that loops infinitely with no output because the loop body is empty

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

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

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



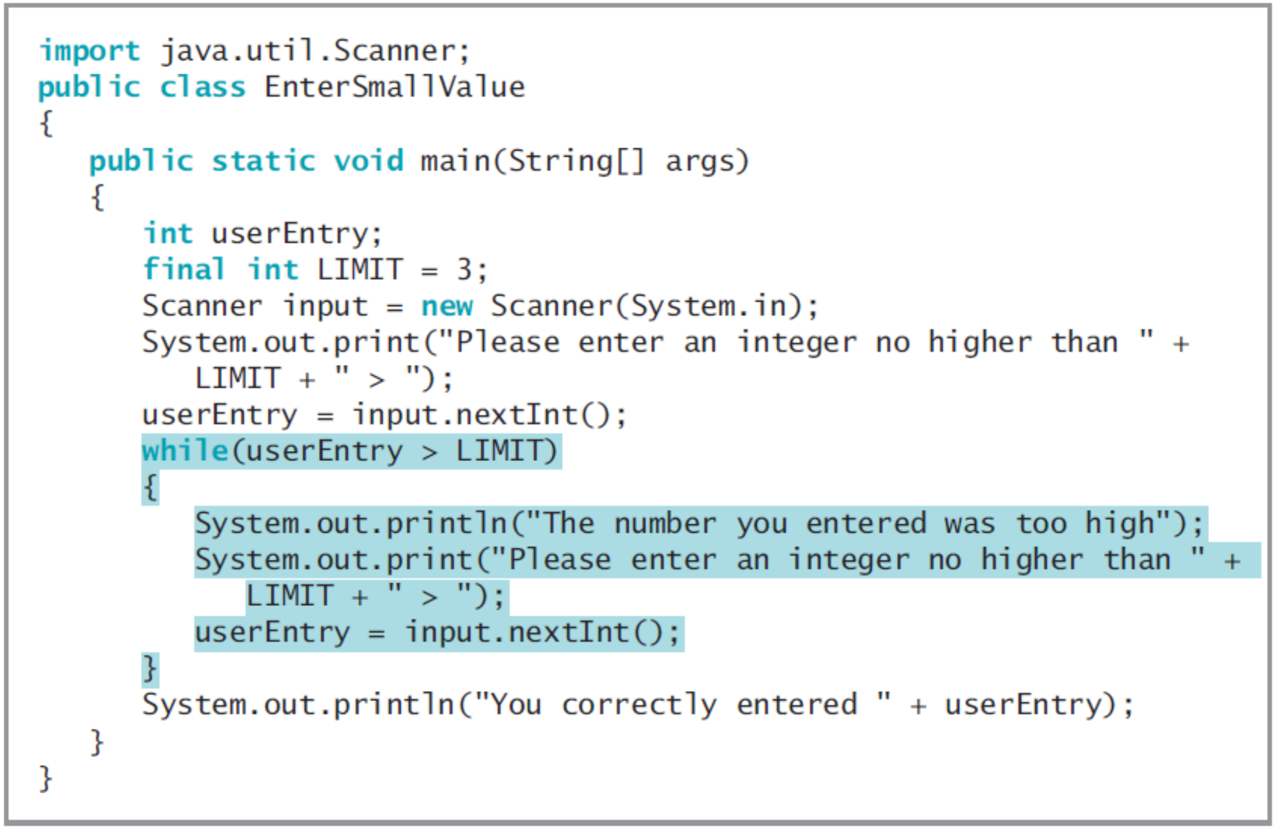
```
loopCount = 3;
while(loopCount > 1)
{
   System.out.println("Hello");
   loopCount = loopCount - 1;
}
```

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 user
    - Executed any number of times
- Validating data
  - Ensures value falls within specified range
  - Use indefinite loops to validate input data
  - If user enters incorrect data
    - Loop repeats

Writing an Indefinite `while` Loop (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 6-10 The EnterSmallValue application

Using Shortcut Arithmetic Operators

- Accumulating
  - Repeatedly increasing value by some amount
- Java provides shortcuts for incrementing and accumulating:

+=	add and assign
−=	subtract and assign
*=	multiply and assign
/=	divide and assign
%=	remainder and assign

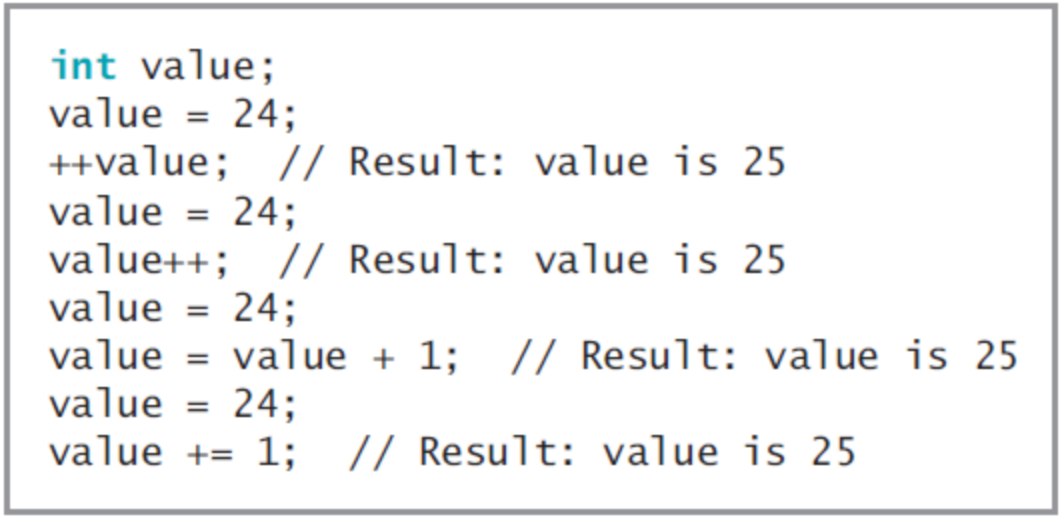
Using Shortcut Arithmetic Operators (cont'd.)

- Prefix and postfix increment operators

  `++someValue, someValue++`

  - Use only with variables
  - Unary operators
    - Use with one value
  - Increase variable's value by 1
    - No difference between operators (unless other operations in same expression)

Using Shortcut Arithmetic Operators (cont'd.)



```
int value;
value = 24;
++value;  // Result: value is 25
value = 24;
value++;  // Result: value is 25
value = 24;
value = value + 1;  // Result: value is 25
value = 24;
value += 1;  // Result: value is 25
```

Figure 6-12 Four ways to add 1 to a value

Using Shortcut Arithmetic Operators (cont'd.)

- Prefix and postfix increment operators (cont'd.)
  - Prefix ++
    - Result calculated and stored
    - Then variable used
  - Postfix ++
    - Variable used
    - Then result calculated and stored
- Prefix and postfix decrement operators

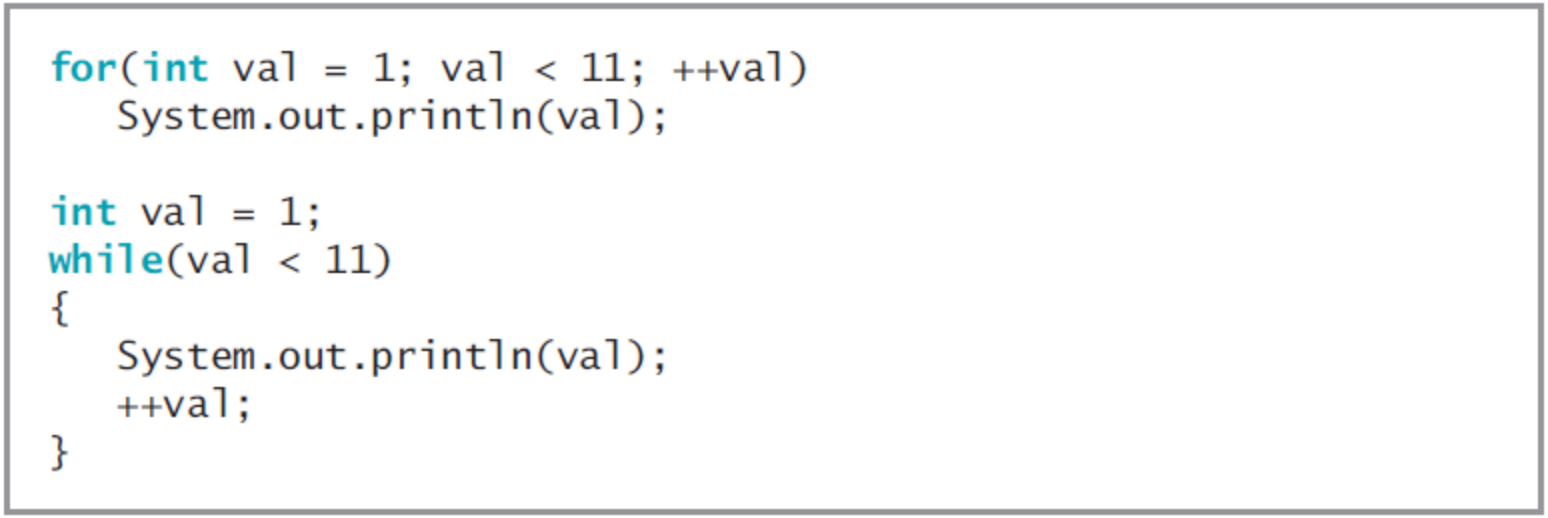
```
--someValue
someValue--
```

  - Similar logic to increment operators

Creating a `for` Loop

- `for` Loop
  - Used when definite number of loop iterations is required
  - One convenient statement
    - Assign starting value for loop control variable
    - Test condition that controls loop entry
    - Alter loop control variable

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



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

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

Figure 6-15 A `for` loop and a `while` loop that display the integers 1 through 10

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

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

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

- Use same loop control variable in all three parts of `for` statement
- To pause program
  - Use `for` loop that contains no body

```
for(x = 0; x < 100000; ++x);
```

  - Or built-in `sleep()` method

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

- `do...while` loop
  - Posttest loop
  - Checks value of loop control variable
    - At bottom of loop
    - After one repetition has occurred
  - Performs task at least one time
  - Never required to use this type of loop
  - Use curly brackets to block statement
    - Even with single statement

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

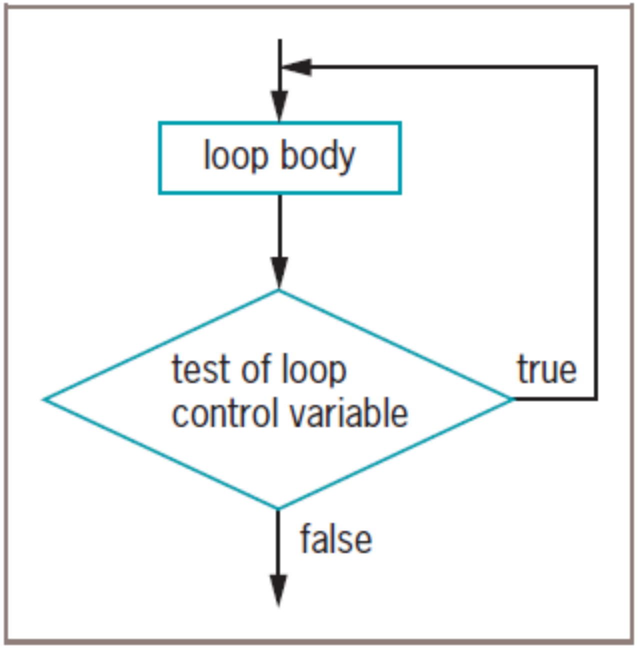
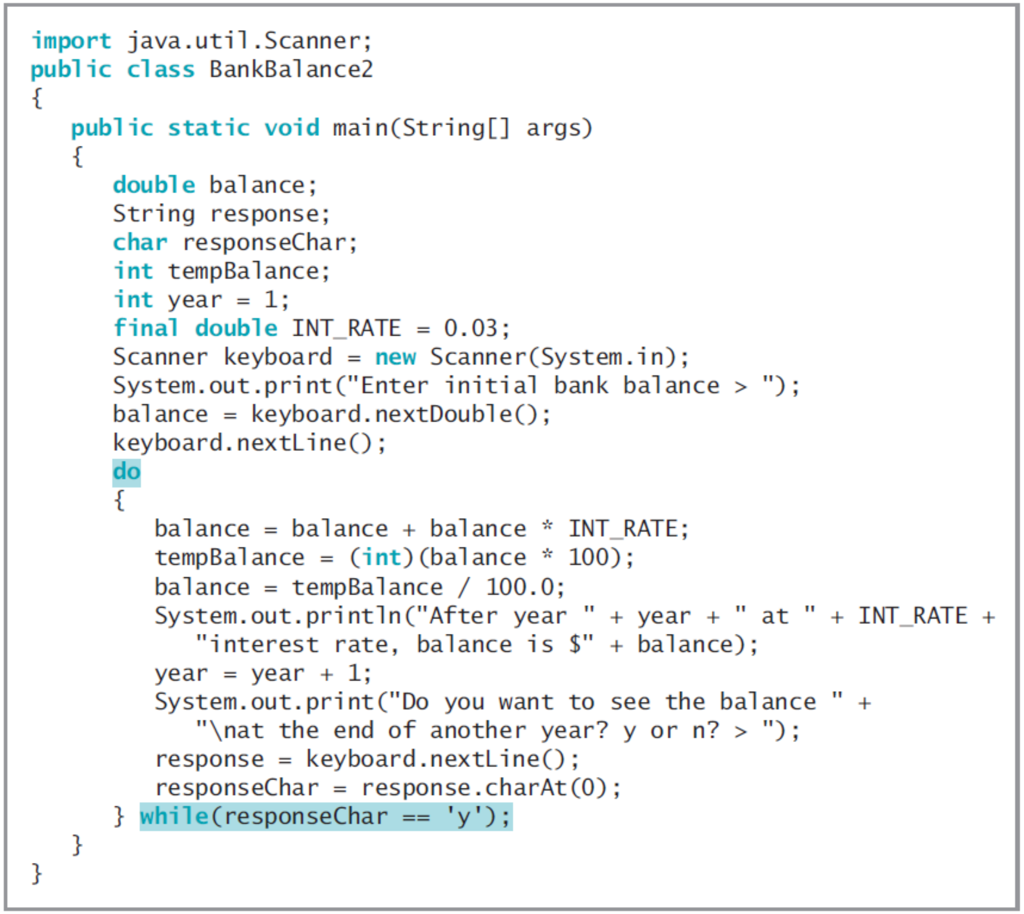


Figure 6-16 General structure of a do...while loop

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;
      String response;
      char responseChar;
      int tempBalance;
      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();
      keyboard.nextLine();
      do
      {
         balance = balance + balance * INT_RATE;
         tempBalance = (int)(balance * 100);
         balance = tempBalance / 100.0;
         System.out.println("After year " + year + " at " + INT_RATE +
            "interest rate, balance is $" + balance);
         year = year + 1;
         System.out.print("Do you want to see the balance " +
            "\nat the end of another year? y or n? > ");
         response = keyboard.nextLine();
         responseChar = response.charAt(0);
      } while(responseChar == 'y');
   }
}
```

Figure 6-17 A do…while loop for the BankBalance2 application

Learning About Nested Loops

- Inner and outer loops
  - Inner loop must be entirely contained in 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.)

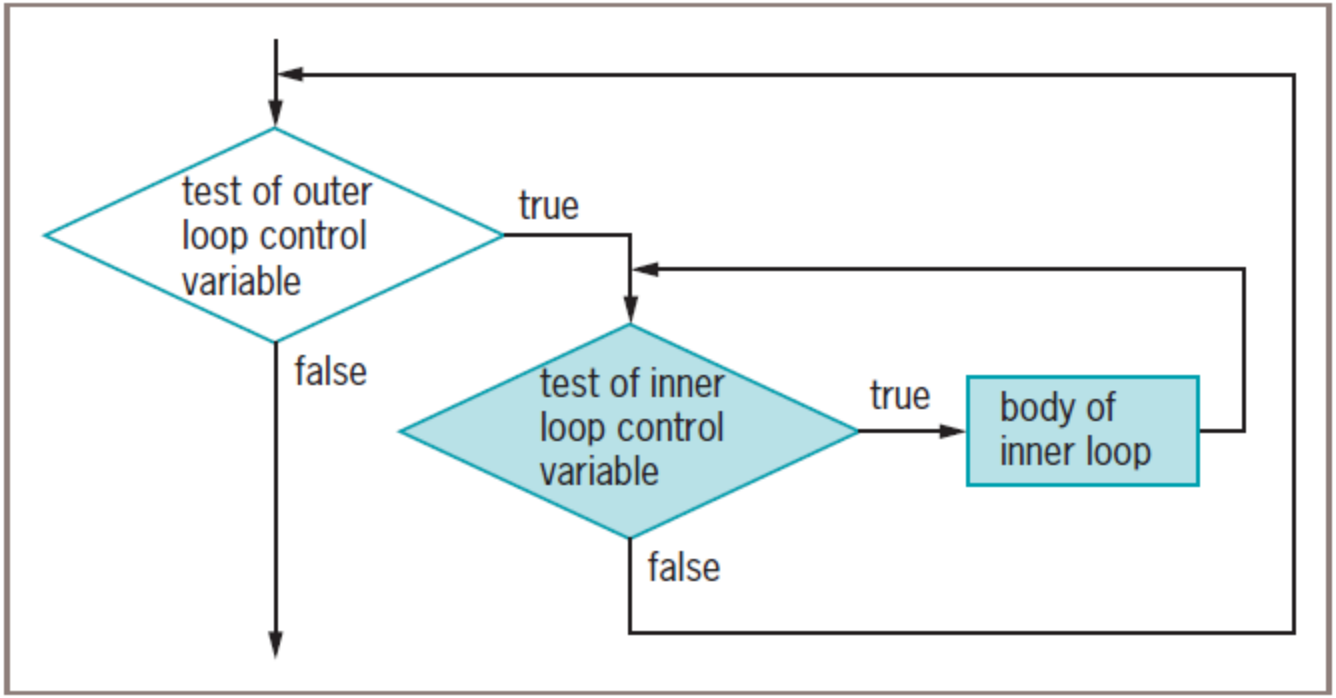

test of outer loop control variable
true
false
test of inner loop control variable
true
body of inner loop
false

Figure 6-18 Nested loops

Improving Loop Performance

- Make sure loop does not include unnecessary operations or statements
- Consider order of evaluation for short-circuit operators
- Make comparisons to 0
- Employ loop fusion

Avoiding Unnecessary Operations

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

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

- Instead use

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

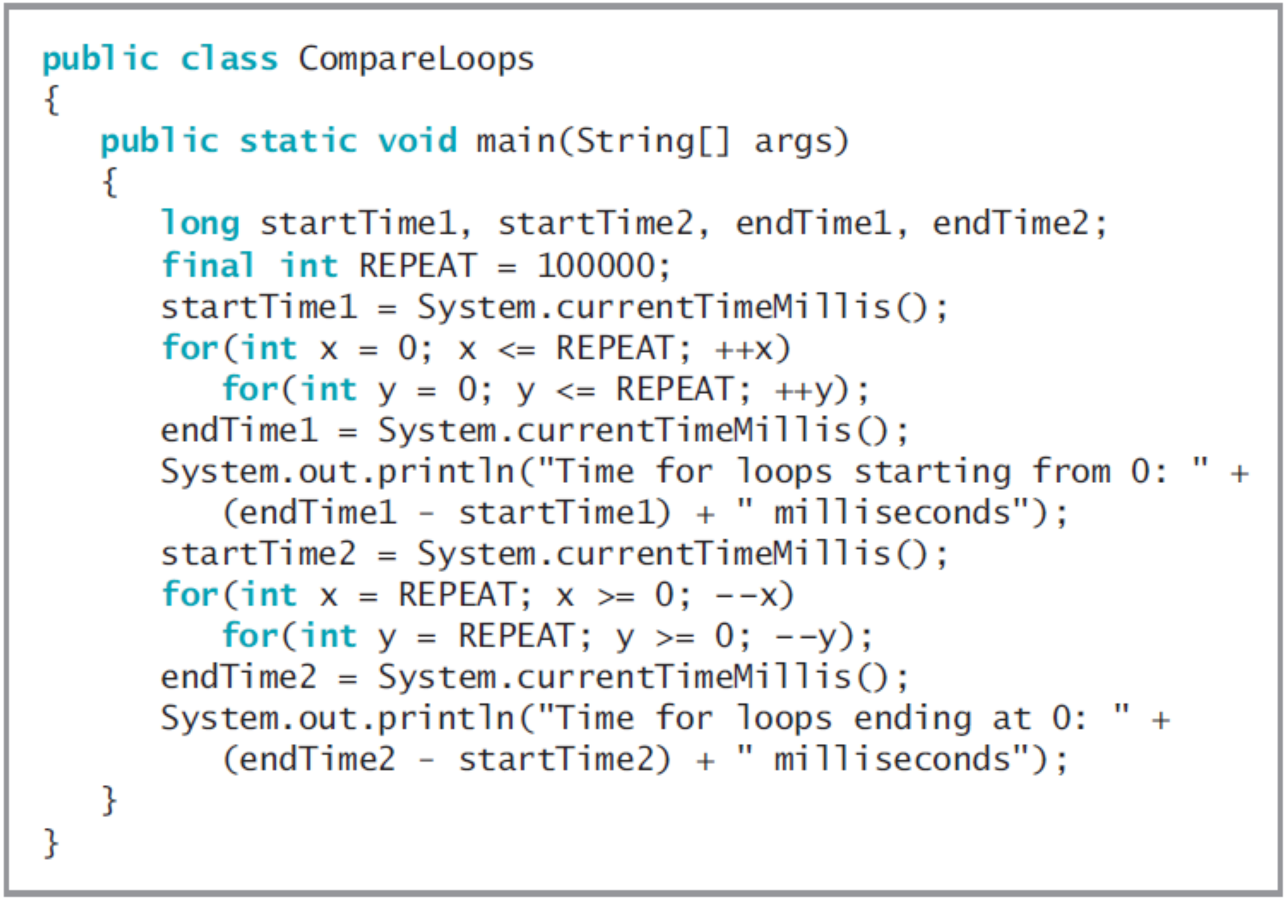
Considering the Order of Evaluation of Short-Circuit Operators

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

Comparing to Zero

- Making comparison to 0
  - Faster than making comparison to any other value
- Improve loop performance
  - Compare loop control variable to 0
- Do-nothing loop
  - Performs no actions other than looping

Comparing to Zero (cont'd.)



```
public class CompareLoops
{
   public static void main(String[] args)
   {
      long startTime1, startTime2, endTime1, endTime2;
      final int REPEAT = 100000;
      startTime1 = System.currentTimeMillis();
      for(int x = 0; x <= REPEAT; ++x)
         for(int y = 0; y <= REPEAT; ++y);
      endTime1 = System.currentTimeMillis();
      System.out.println("Time for loops starting from 0: " +
         (endTime1 - startTime1) + " milliseconds");
      startTime2 = System.currentTimeMillis();
      for(int x = REPEAT; x >= 0; --x)
         for(int y = REPEAT; y >= 0; --y);
      endTime2 = System.currentTimeMillis();
      System.out.println("Time for loops ending at 0: " +
         (endTime2 - startTime2) + " milliseconds");
   }
}
```

Figure 6-21 The CompareLoops application

Employing Loop Fusion

- Loop fusion
  - Technique of combining two loops into one
  - Will not work in every situation

You Do It

- Writing a loop to validate data entries
- Working with prefix and postfix increment operators
- Working with definite loops
- Working with nested loops

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

Summary

- Loop structure allows repeated execution of block of statements
  - Infinite loop
  - Definite loop
  - Nest loop
- Must change loop control variable within looping structure
- Use `while` loop to:
  - Execute statements while some condition is `true`

Summary (cont'd.)

- Execute `while` loop
  - Initialize loop control variable, test in `while` statement, and alter loop control variable
- Prefix ++ and postfix ++
  - Increase variable’s value by 1
  - Variable used
    - Result 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 result in one step
- `for` loop
  - Initializes, tests, and increments in one statement
- `do...while` loop
  - Tests Boolean expression after one repetition
- Improve loop performance
  - Do not include unnecessary operations or statements

End of Module.

REFERENCE:

Farrell, J. (2016). *Java Programming*. 8th Edition. Course Technology, Cengage Learning.