Introduction to Java

baldoino@ic.ufal.br

First Program

Input/Output and Operations

Printing a Line of Text

A compilation error occurs if a public class's filename is not exactly same name as the class

```
1  // Fig. 2.1: Welcome1.java
2  // Text-printing program.
3
4  public class Welcome1
5  {
6     // main method begins execution of Java application
7     public static void main(String[] args)
8     {
9        System.out.println("Welcome to Java Programming!");
10    } // end method main
11 } // end class Welcome1
```

Use *blank lines* and *spaces* to enhance program readability

Input/Output and Operations

Printing Text

```
1  // Fig. 2.3: Welcome2.java
2  // Printing a line of text with multiple statements.
3
4  public class Welcome2
5  {
6     // main method begins execution of Java application
7     public static void main(String[] args)
8     {
9        System.out.print("Welcome to ");
10       System.out.println("Java Programming!");
11    } // end method main
12 } // end class Welcome2
```

```
System.out.println("Welcome\nto\nJava\nProgramming!");
```

```java
// Fig. 2.7: Addition.java
// Addition program that inputs two numbers then displays their sum.
import java.util.Scanner; // program uses class Scanner

public class Addition
{
   // main method begins execution of Java application
   public static void main(String[] args)
   {
      // create a Scanner to obtain input from the command window
      Scanner input = new Scanner(System.in);

      int number1; // first number to add
      int number2; // second number to add
      int sum; // sum of number1 and number2

      System.out.print("Enter first integer: "); // prompt
      number1 = input.nextInt(); // read first number from user

      System.out.print("Enter second integer: "); // prompt
      number2 = input.nextInt(); // read second number from user

      sum = number1 + number2; // add numbers, then store total in sum

      System.out.printf("Sum is %d%n", sum); // display sum
   } // end method main
} // end class Addition
```

import Declarations

Java operation	Operator	Algebraic expression	Java expression
Addition	+	$f + 7$	`f + 7`
Subtraction	–	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Remainder	%	$r \bmod s$	`r % s`

Operator(s)	Operation(s)	Order of evaluation (precedence)
* / %	Multiplication Division Remainder	Evaluated first. If there are several operators of this type, they're evaluated from *left to right*.
+ -	Addition Subtraction	Evaluated next. If there are several operators of this type, they're evaluated from *left to right*.
=	Assignment	Evaluated last.

Algebraic operator	Java equality or relational operator	Sample Java condition	Meaning of Java condition
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y

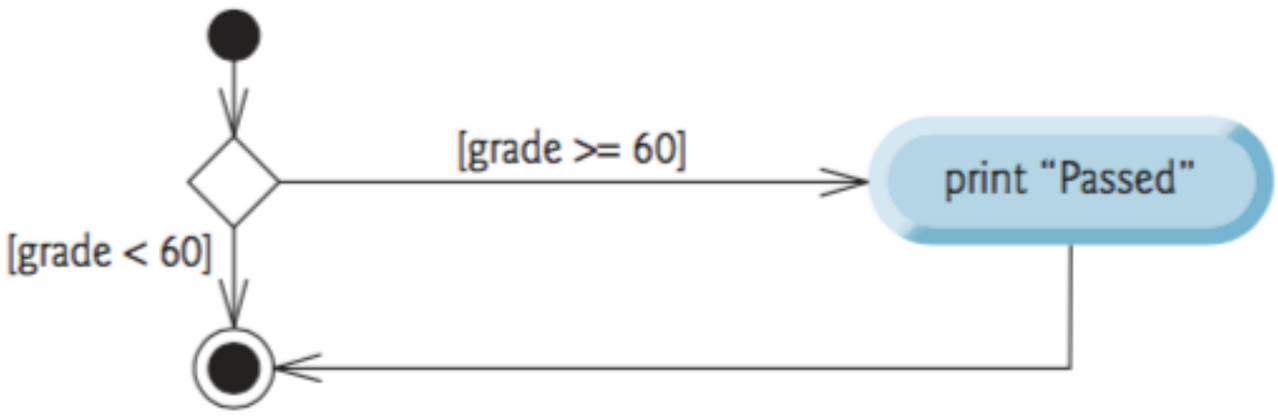
Algebraic operator	Java equality or relational operator	Sample Java condition	Meaning of Java condition
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

```java
import java.util.Scanner; // program uses class Scanner

public class Comparison
{
   // main method begins execution of Java application
   public static void main(String[] args)
   {
      // create Scanner to obtain input from command line
      Scanner input = new Scanner(System.in);

      int number1; // first number to compare
      int number2; // second number to compare

      System.out.print("Enter first integer: "); // prompt
      number1 = input.nextInt(); // read first number from user

      System.out.print("Enter second integer: "); // prompt
      number2 = input.nextInt(); // read second number from user

      if (number1 == number2)
         System.out.printf("%d == %d%n", number1, number2);

      if (number1 != number2)
         System.out.printf("%d != %d%n", number1, number2);

      if (number1 < number2)
         System.out.printf("%d < %d%n", number1, number2);

      if (number1 > number2)
         System.out.printf("%d > %d%n", number1, number2);

      if (number1 <= number2)
         System.out.printf("%d <= %d%n", number1, number2);

      if (number1 >= number2)
         System.out.printf("%d >= %d%n", number1, number2);
   } // end method main
} // end class Comparison
```

Operators				Associativity	Type
*	/	%		left to right	multiplicative
+	-			left to right	additive
<	<=	>	>=	left to right	relational
==	!=			left to right	equality
=				right to left	assignment

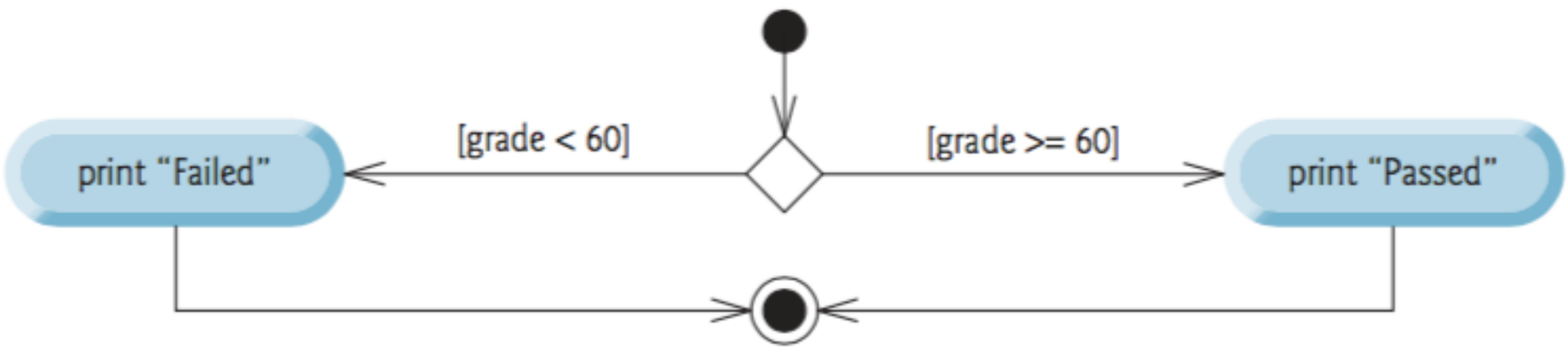
If student's grade is greater than or equal to 60
*   Print "Passed"*

```
if (studentGrade >= 60)
   System.out.println("Passed");
```



```
If student's grade is greater than or equal to 60
      Print "Passed"
Else
      Print "Failed"
```

```
if (grade >= 60)
   System.out.println("Passed");
else
   System.out.println("Failed");
```



```
If student's grade is greater than or equal to 90
      Print "A"
else
      If student's grade is greater than or equal to 80
            Print "B"
      else
            If student's grade is greater than or equal to 70
                  Print "C"
            else
                  If student's grade is greater than or equal to 60
                        Print "D"
                  else
                        Print "F"
```

```
if (studentGrade >= 90)
   System.out.println("A");
else if (studentGrade >= 80)
   System.out.println("B");
else if (studentGrade >= 70)
   System.out.println("C");
else if (studentGrade >= 60)
   System.out.println("D");
else
   System.out.println("F");
```

```
if (x > 5)
   if (y > 5)
      System.out.println("x and y are > 5");
else
   System.out.println("x is <= 5");
```

```
if (x > 5)
{
   if (y > 5)
      System.out.println("x and y are > 5");
}
else
   System.out.println("x is <= 5");
```

```
if (x > 5)
   if (y > 5)
      System.out.println("x and y are > 5");
   else
      System.out.println("x is <= 5");
```

```
System.out.println(studentGrade >= 60 ? "Passed" : "Failed");
```

```
 3  public class Student
 4  {
 5     private String name;
 6     private double average;
 7
 8     // constructor initializes instance variables
 9     public Student(String name, double average)
10     {
11        this.name = name;
12
13        // validate that average is > 0.0 and <= 100.0; otherwise,
14        // keep instance variable average's default value (0.0)
15        if (average > 0.0)
16           if (average <= 100.0)
17              this.average = average; // assign to instance variable
18     }
19
20     // sets the Student's name
21     public void setName(String name)
22     {
23        this.name = name;
24     }
25
26     // retrieves the Student's name
27     public String getName()
28     {
29        return name;
30     }
31
32     // sets the Student's average
33     public void setAverage(double studentAverage)
34     {
35        // validate that average is > 0.0 and <= 100.0; otherwise,
36        // keep instance variable average's current value
37        if (average > 0.0)
38           if (average <= 100.0)
39              this.average = average; // assign to instance variable
40     }
```

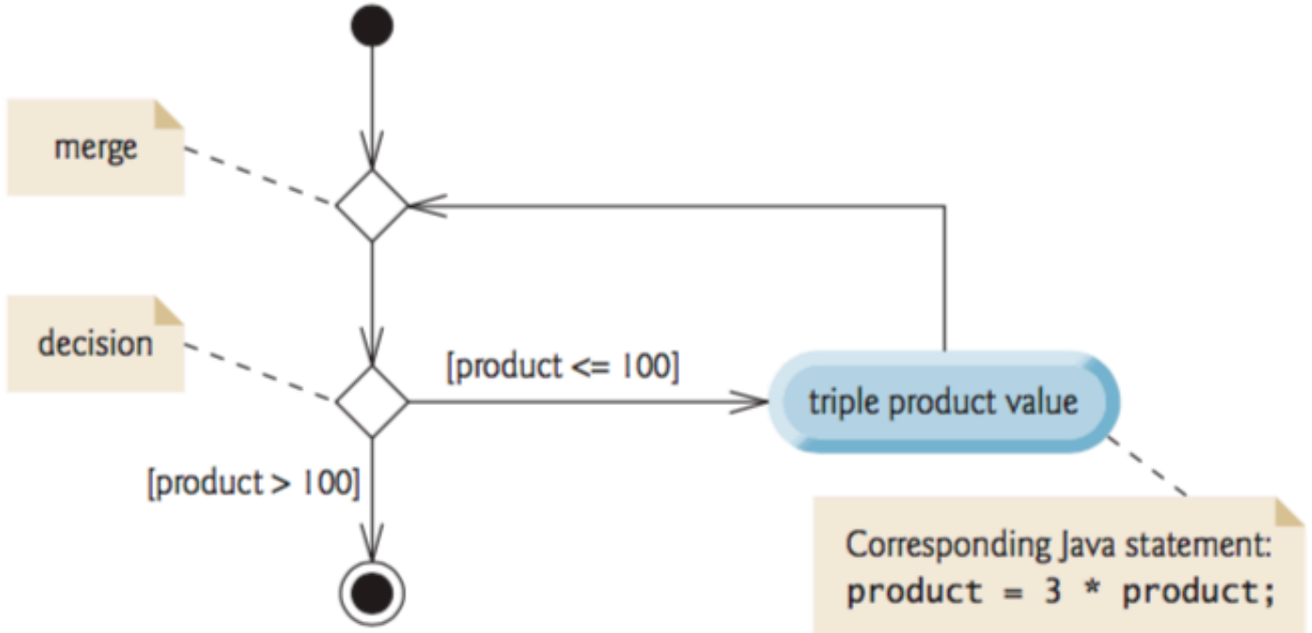
```
41
42      // retrieves the Student's average
43      public double getAverage()
44      {
45          return average;
46      }
47
48      // determines and returns the Student's letter grade
49      public String getLetterGrade()
50      {
51          String letterGrade = ""; // initialized to empty String
52
```

```
53              if (average >= 90.0)
54                 letterGrade = "A";
55              else if (average >= 80.0)
56                 letterGrade = "B";
57              else if (average >= 70.0)
58                 letterGrade = "C";
59              else if (average >= 60.0)
60                 letterGrade = "D";
61              else
62                 letterGrade = "F";
63
64              return letterGrade;
65           }
66     } // end class Student
```

```java
// Fig. 4.5: StudentTest.java
// Create and test Student objects.
public class StudentTest
{
   public static void main(String[] args)
   {
      Student account1 = new Student("Jane Green", 93.5);
      Student account2 = new Student("John Blue", 72.75);

      System.out.printf("%s's letter grade is: %s%n",
         account1.getName(), account1.getLetterGrade());
      System.out.printf("%s's letter grade is: %s%n",
         account2.getName(), account2.getLetterGrade());
   }
} // end class StudentTest
```

While there are more items on my shopping list
    Purchase next item and cross it off my list

```
while (product <= 100)
   product = 3 * product;
```

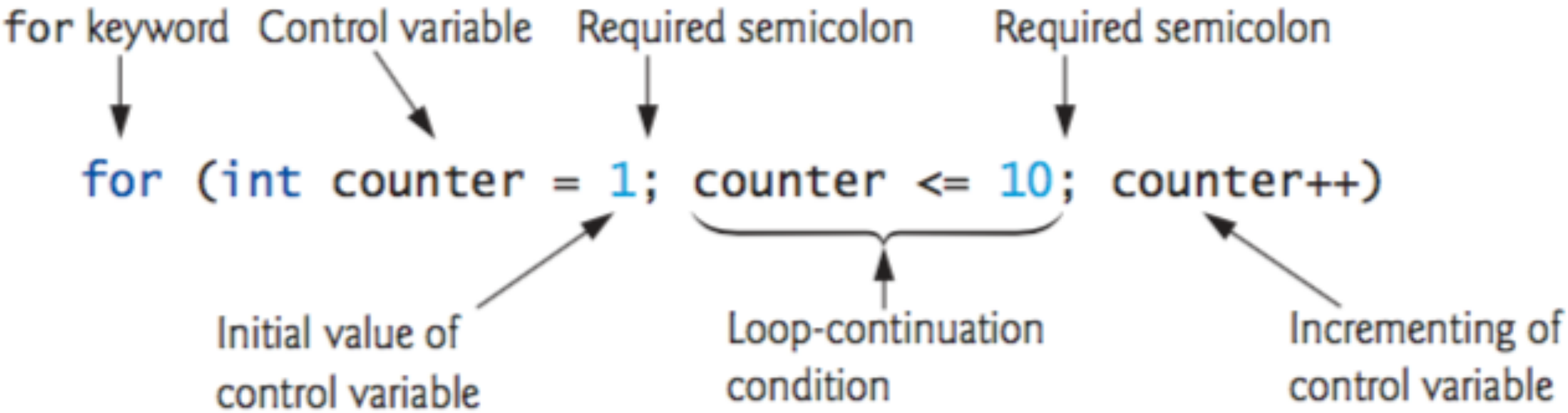


Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`c += 7`	`c = c + 7`	10 to c
`-=`	`d -= 4`	`d = d - 4`	1 to d
`*=`	`e *= 5`	`e = e * 5`	20 to e
`/=`	`f /= 3`	`f = f / 3`	2 to f
`%=`	`g %= 9`	`g = g % 9`	3 to g

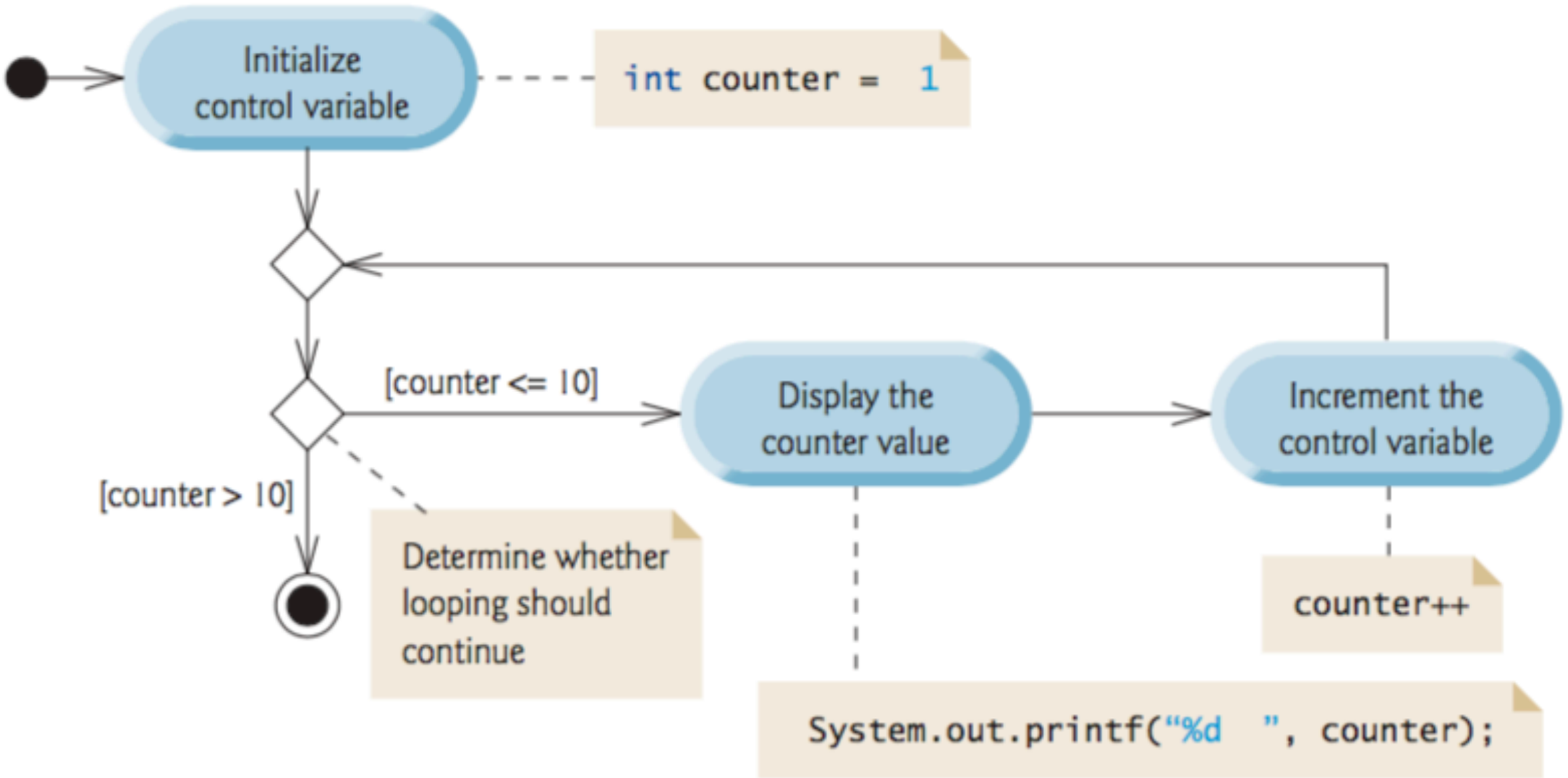
Operator	Operator name	Sample expression	Explanation
++	prefix increment	++a	Increment a by 1, then use the new value of a in the expression in which a resides.
++	postfix increment	a++	Use the current value of a in the expression in which a resides, then increment a by 1.
--	prefix decrement	--b	Decrement b by 1, then use the new value of b in the expression in which b resides.
--	postfix decrement	b--	Use the current value of b in the expression in which b resides, then decrement b by 1.

```java
// Fig. 4.15: Increment.java
// Prefix increment and postfix increment operators.

public class Increment
{
    public static void main(String[] args)
    {
        // demonstrate postfix increment operator
        int c = 5;
        System.out.printf("c before postincrement: %d%n", c); // prints 5
        System.out.printf("    postincrementing c: %d%n", c++); // prints 5
        System.out.printf(" c after postincrement: %d%n", c); // prints 6

        System.out.println(); // skip a line

        // demonstrate prefix increment operator
        c = 5;
        System.out.printf(" c before preincrement: %d%n", c); // prints 5
        System.out.printf("     preincrementing c: %d%n", ++c); // prints 6
        System.out.printf("  c after preincrement: %d%n", c); // prints 6
    }
} // end class Increment
```

Operators						Associativity	Type
++	--					right to left	unary postfix
++	--	+	-	(*type*)		right to left	unary prefix
*	/	%				left to right	multiplicative
+	-					left to right	additive
<	<=	>	>=			left to right	relational
==	!=					left to right	equality
?:						right to left	conditional
=	+=	-=	*=	/=	%=	right to left	assignment


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

```
1  // Fig. 5.2: ForCounter.java
2  // Counter-controlled repetition with the for repetition statement.
3
4  public class ForCounter
5  {
6     public static void main(String[] args)
7     {
8        // for statement header includes initialization,
9        // loop-continuation condition and increment
10       for (int counter = 1; counter <= 10; counter++)
11          System.out.printf("%d  ", counter);
12
13       System.out.println();
14    }
15 } // end class ForCounter
```


Initialize
control variable
int counter = 1
[counter <= 10]
[counter > 10]
Display the
counter value
Increment the
control variable
counter++
Determine whether
looping should
continue
System.out.printf("%d ", counter);

a) Vary the control variable from 1 to 100 in increments of 1.

```
for (int i = 1; i <= 100; i++)
```

b) Vary the control variable from 100 to 1 in *decrements* of 1.

```
for (int i = 100; i >= 1; i--)
```

c) Vary the control variable from 7 to 77 in increments of 7.

```
for (int i = 7; i <= 77; i += 7)
```

d) Vary the control variable from 20 to 2 in *decrements* of 2.

```
for (int i = 20; i >= 2; i -= 2)
```

e) Vary the control variable over the values 2, 5, 8, 11, 14, 17, 20.

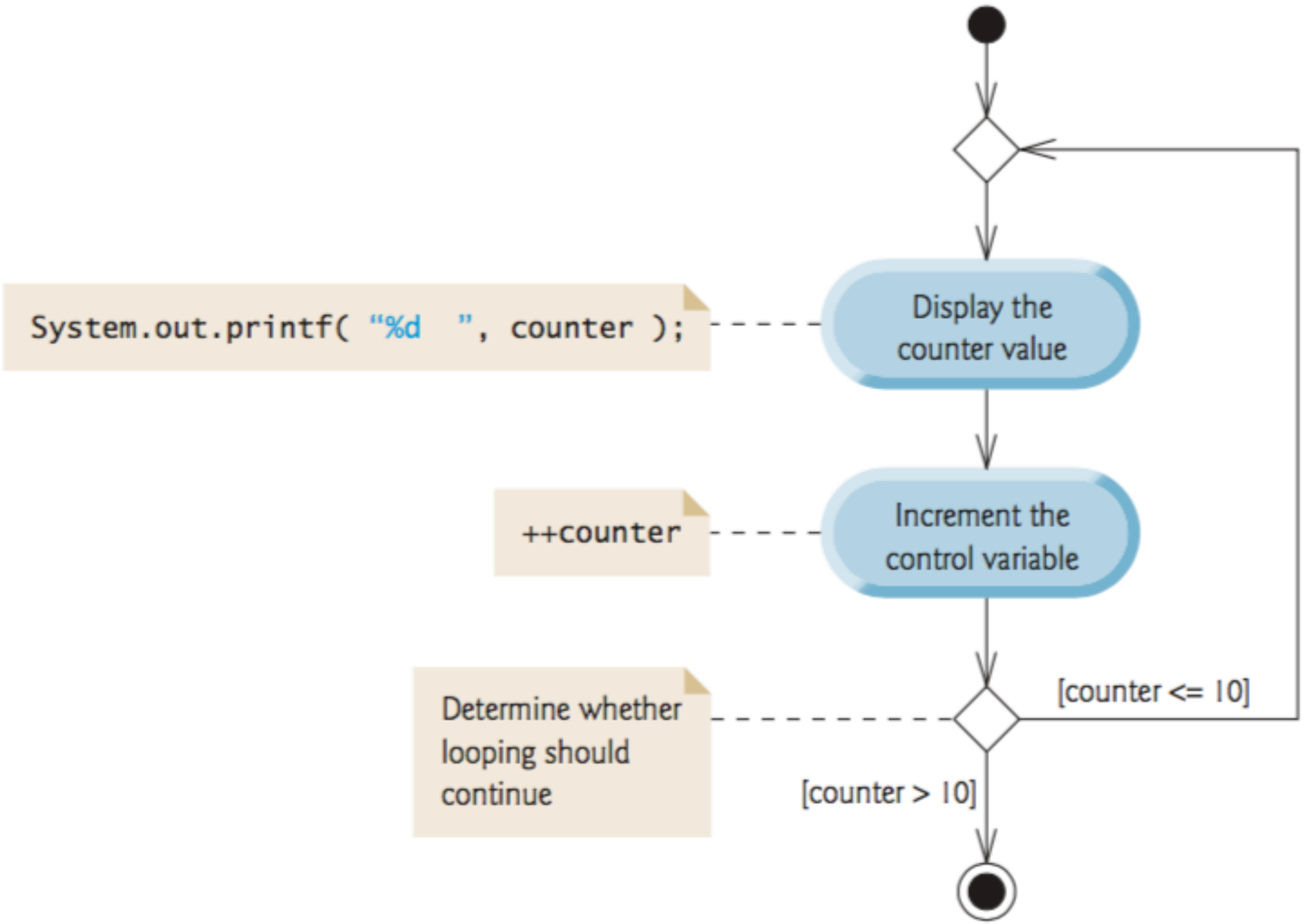
```
for (int i = 2; i <= 20; i += 3)
```

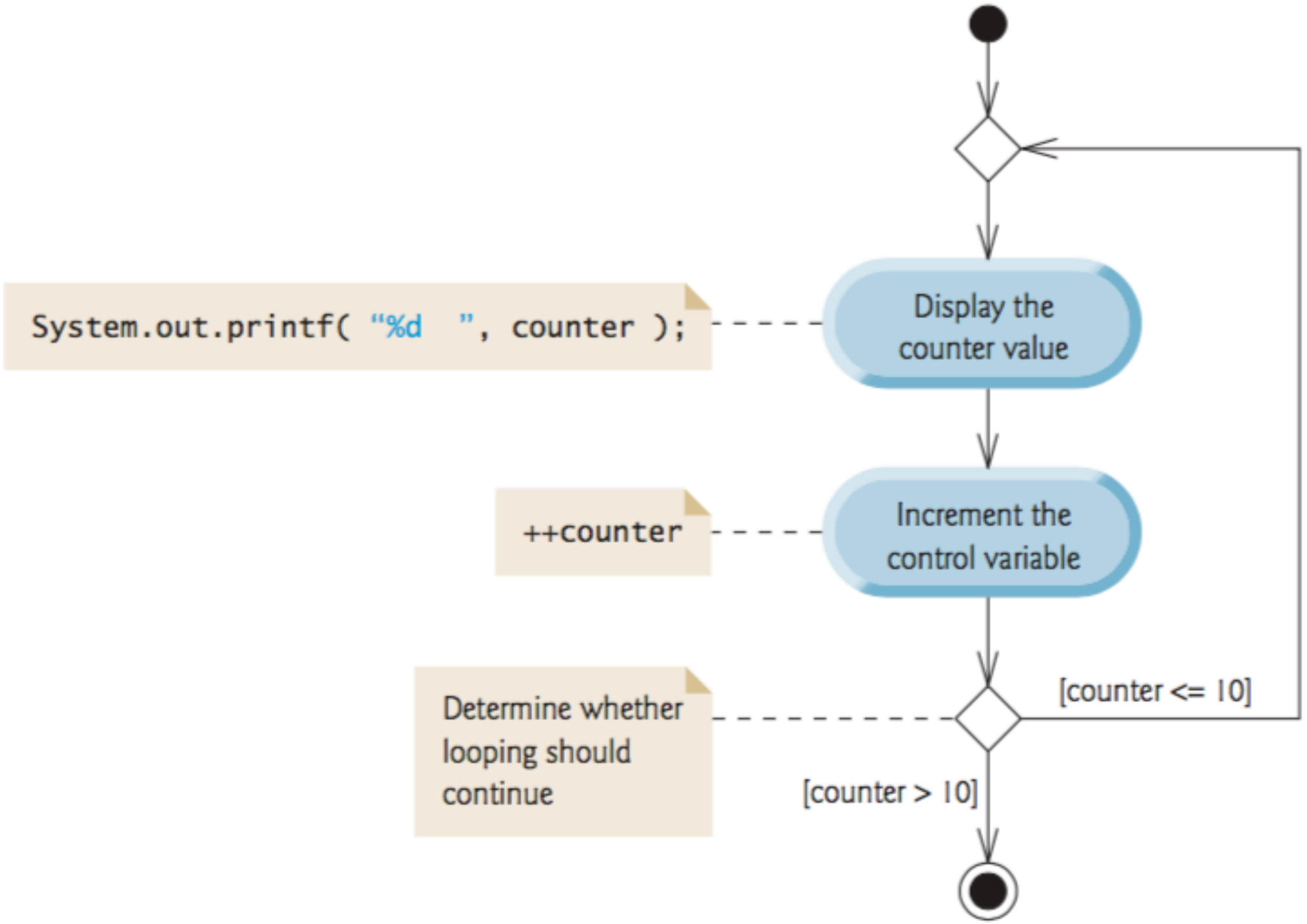
f) Vary the control variable over the values 99, 88, 77, 66, 55, 44, 33, 22, 11, 0.

```
for (int i = 99; i >= 0; i -= 11)
```

```
do
{
    statement
} while (condition);
```

```
1   // Fig. 5.7: DoWhileTest.java
2   // do...while repetition statement.
3
4   public class DoWhileTest
5   {
6      public static void main(String[] args)
7      {
8         int counter = 1;
9
10        do
11        {
12           System.out.printf("%d  ", counter);
13           ++counter;
14        } while (counter <= 10); // end do...while
15
16        System.out.println();
17     }
18  } // end class DoWhileTest
```


System.out.printf("%d ", counter);
Display the counter value
++counter
Increment the control variable
Determine whether looping should continue
[counter <= 10]
[counter > 10]


System.out.printf("%d ", counter);
Display the counter value
++counter
Increment the control variable
Determine whether looping should continue
[counter <= 10]
[counter > 10]

```
public class AutoPolicy
{
   private int accountNumber; // policy account number
   private String makeAndModel; // car that the policy applies to
   private String state; // two-letter state abbreviation

   // constructor
   public AutoPolicy(int accountNumber, String makeAndModel, String state)
   {
      this.accountNumber = accountNumber;
      this.makeAndModel = makeAndModel;
      this.state = state;
   }

   // sets the accountNumber
   public void setAccountNumber(int accountNumber)
   {
      this.accountNumber = accountNumber;
   }

   // returns the accountNumber
   public int getAccountNumber()
   {
      return accountNumber;
   }

   // sets the makeAndModel
   public void setMakeAndModel(String makeAndModel)
   {
      this.makeAndModel = makeAndModel;
   }

   // returns the makeAndModel
   public String getMakeAndModel()
   {
      return makeAndModel;
   }

   // sets the state
   public void setState(String state)
   {
      this.state = state;
   }

   // returns the state
   public String getState()
   {
      return state;
   }

   // predicate method returns whether the state has no
   public boolean isNoFaultState()
   {
      boolean noFaultState;

      // determine whether state has no-fault auto insu
      switch (getState()) // get AutoPolicy object's st
      {
         case "MA": case "NJ": case "NY": case "PA":
            noFaultState = true;
            break;
         default:
            noFaultState = false;
            break;
      }

      return noFaultState;
   }
} // end class AutoPolicy
```

```java
// Fig. 5.12: AutoPolicyTest.java
// Demonstrating Strings in switch.
public class AutoPolicyTest
{
   public static void main(String[] args)
   {
      // create two AutoPolicy objects
      AutoPolicy policy1 =
         new AutoPolicy(11111111, "Toyota Camry", "NJ");
      AutoPolicy policy2 =
         new AutoPolicy(22222222, "Ford Fusion", "ME");

      // display whether each policy is in a no-fault state
      policyInNoFaultState(policy1);
      policyInNoFaultState(policy2);
   }

   // method that displays whether an AutoPolicy
   // is in a state with no-fault auto insurance
   public static void policyInNoFaultState(AutoPolicy policy)
   {
      System.out.println("The auto policy:");
      System.out.printf(
         "Account #: %d; Car: %s; State %s %s a no-fault state%n%n",
         policy.getAccountNumber(), policy.getMakeAndModel(),
         policy.getState(),
         (policy.isNoFaultState() ? "is": "is not"));
   }
} // end class AutoPolicyTest
```

```java
// Fig. 5.13: BreakTest.java
// break statement exiting a for statement.
public class BreakTest
{
   public static void main(String[] args)
   {
      int count; // control variable also used after loop terminates

      for (count = 1; count <= 10; count++) // loop 10 times
      {
         if (count == 5)
            break; // terminates loop if count is 5

         System.out.printf("%d ", count);
      }

      System.out.printf("%nBroke out of loop at count = %d%n", count);
   }
} // end class BreakTest
```

```java
// Fig. 5.14: ContinueTest.java
// continue statement terminating an iteration of a for statement.
public class ContinueTest
{
   public static void main(String[] args)
   {
      for (int count = 1; count <= 10; count++) // loop 10 times
      {
         if (count == 5)
            continue; // skip remaining code in loop body if count is 5

         System.out.printf("%d ", count);
      }

      System.out.printf("%nUsed continue to skip printing 5%n");
   }
} // end class ContinueTest
```

ClassName.*methodName*(*arguments*)

```
Math.sqrt(900.0)
```

Method	Description	Example

`abs(`*x*`)`	absolute value of *x*	`abs(23.7)` is `23.7`
		`abs(0.0)` is `0.0`
		`abs(-23.7)` is `23.7`

`ceil(`*x*`)`	rounds *x* to the smallest integer not less than *x*	`ceil(9.2)` is `10.0` `ceil(-9.8)` is `-9.0`
`cos(`*x*`)`	trigonometric cosine of *x* (*x* in radians)	`cos(0.0)` is `1.0`
`exp(`*x*`)`	exponential method e^x	`exp(1.0)` is `2.71828` `exp(2.0)` is `7.38906`
`floor(`*x*`)`	rounds *x* to the largest integer not greater than *x*	`floor(9.2)` is `9.0` `floor(-9.8)` is `-10.0`
`log(`*x*`)`	natural logarithm of *x* (base *e*)	`log(Math.E)` is `1.0` `log(Math.E * Math.E)` is `2.0`
`max(`*x*`, `*y*`)`	larger value of *x* and *y*	`max(2.3, 12.7)` is `12.7` `max(-2.3, -12.7)` is `-2.3`
`min(`*x*`, `*y*`)`	smaller value of *x* and *y*	`min(2.3, 12.7)` is `2.3` `min(-2.3, -12.7)` is `-12.7`
`pow(`*x*`, `*y*`)`	*x* raised to the power *y* (i.e., x^y)	`pow(2.0, 7.0)` is `128.0` `pow(9.0, 0.5)` is `3.0`
`sin(`*x*`)`	trigonometric sine of *x* (*x* in radians)	`sin(0.0)` is `0.0`
`sqrt(`*x*`)`	square root of *x*	`sqrt(900.0)` is `30.0`
`tan(`*x*`)`	trigonometric tangent of *x* (*x* in radians)	`tan(0.0)` is `0.0`

Type	Valid promotions
`double`	None
`float`	`double`
`long`	`float` or `double`
`int`	`long`, `float` or `double`
`char`	`int`, `long`, `float` or `double`
`short`	`int`, `long`, `float` or `double` (but not `char`)
`byte`	`short`, `int`, `long`, `float` or `double` (but not `char`)
`boolean`	None (`boolean` values are not considered to be numbers in Java)

```
4  public class Scope
5  {
6     // field that is accessible to all methods of this class
7     private static int x = 1;
8
9     // method main creates and initializes local variable x
10    // and calls methods useLocalVariable and useField
11    public static void main(String[] args)
12    {
13       int x = 5; // method's local variable x shadows field x
14
15       System.out.printf("local x in main is %d%n", x);
16
17       useLocalVariable(); // useLocalVariable has local x
18       useField(); // useField uses class Scope's field x
19       useLocalVariable(); // useLocalVariable reinitializes local x
20       useField(); // class Scope's field x retains its value
21
22       System.out.printf("%nlocal x in main is %d%n", x);
23    }
24
25    // create and initialize local variable x during each call
26    public static void useLocalVariable()
27    {
28       int x = 25; // initialized each time useLocalVariable is called
29
30       System.out.printf(
31          "%nlocal x on entering method useLocalVariable is %d%n", x);
32       ++x; // modifies this method's local variable x
33       System.out.printf(
34          "local x before exiting method useLocalVariable is %d%n", x);
35    }
36
37    // modify class Scope's field x during each call
38    public static void useField()
39    {
40       System.out.printf(
41          "%nfield x on entering method useField is %d%n", x);
42       x *= 10; // modifies class Scope's field x
43       System.out.printf(
44          "field x before exiting method useField is %d%n", x);
45    }
46 } // end class Scope
```

```java
// Fig. 6.10: MethodOverload.java
// Overloaded method declarations.

public class MethodOverload
{
   // test overloaded square methods
   public static void main(String[] args)
   {
      System.out.printf("Square of integer 7 is %d%n", square(7));
      System.out.printf("Square of double 7.5 is %f%n", square(7.5));
   }

   // square method with int argument
   public static int square(int intValue)
   {
      System.out.printf("%nCalled square with int argument: %d%n",
         intValue);
      return intValue * intValue;
   }

   // square method with double argument
   public static double square(double doubleValue)
   {
      System.out.printf("%nCalled square with double argument: %f%n",
         doubleValue);
      return doubleValue * doubleValue;
   }
} // end class MethodOverload
```

Arrays and ArrayLists

```java
// Fig. 7.2: InitArray.java
// Initializing the elements of an array to default values of zero.

public class InitArray
{
   public static void main(String[] args)
   {
      // declare variable array and initialize it with an array object
      int[] array = new int[10]; // create the array object

      System.out.printf("%s%8s%n", "Index", "Value"); // column headings
```

```java
// Fig. 7.3: InitArray.java
// Initializing the elements of an array with an array initializer.

public class InitArray
{
   public static void main(String[] args)
   {
      // initializer list specifies the initial value for each element
      int[] array = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };

      System.out.printf("%s%8s%n", "Index", "Value"); // column headings

      // output each array element's value
      for (int counter = 0; counter < array.length; counter++)
         System.out.printf("%5d%8d%n", counter, array[counter]);

} // end class InitArray
```

```java
// Fig. 7.12: EnhancedForTest.java
// Using the enhanced for statement to total integers in an array.

public class EnhancedForTest
{
   public static void main(String[] args)
   {
      int[] array = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
      int total = 0;

      // add each element's value to total
      for (int number : array)
         total += number;

      System.out.printf("Total of array elements: %d%n", total);
   }
} // end class EnhancedForTest
```

```
3
4  public class InitArray
5  {
6     // create and output two-dimensional arrays
7     public static void main(String[] args)
8     {
9        int[][] array1 = {{1, 2, 3}, {4, 5, 6}};
10       int[][] array2 = {{1, 2}, {3}, {4, 5, 6}};
11
12       System.out.println("Values in array1 by row are");
13       outputArray(array1); // displays array1 by row
14
15       System.out.printf("%nValues in array2 by row are%n");
16       outputArray(array2); // displays array2 by row
17    }
18
19    // output rows and columns of a two-dimensional array
20    public static void outputArray(int[][] array)
21    {
22       // loop through array's rows
23       for (int row = 0; row < array.length; row++)
24       {
25          // loop through columns of current row
26          for (int column = 0; column < array[row].length; column++)
27             System.out.printf("%d  ", array[row][column]);
28
29          System.out.println();
30       }
31    }
32 } // end class InitArray
```

```
4  public class VarargsTest
5  {
6     // calculate average
7     public static double average(double... numbers)
8     {
9        double total = 0.0;
10
11       // calculate total using the enhanced for statement
12       for (double d : numbers)
13          total += d;
14
15       return total / numbers.length;
16    }
17
18    public static void main(String[] args)
19    {
20       double d1 = 10.0;
21       double d2 = 20.0;
22       double d3 = 30.0;
23       double d4 = 40.0;
24
25       System.out.printf("d1 = %.1f%nd2 = %.1f%nd3 = %.1f%nd4 = %.1f%n%n",
26          d1, d2, d3, d4);
27
28       System.out.printf("Average of d1 and d2 is %.1f%n",
29          average(d1, d2)  );
30       System.out.printf("Average of d1, d2 and d3 is %.1f%n",
31          average(d1, d2, d3)  );
32       System.out.printf("Average of d1, d2, d3 and d4 is %.1f%n",
33          average(d1, d2, d3, d4)  );
34    }
35 } // end class VarargsTest
```