



Computer Programming 2

Module 3: Control Flow Statements

Name (LN, FN, MN): _____ Program/Yr/Block: _____

I. Introduction

The statements inside your source files are generally executed from top to bottom, in the order that they appear. Control flow statements, however, break up the flow of execution by employing decision making, looping, and branching, enabling your program to conditionally execute particular blocks of code. This module will cover the decision-making statements (if-then, if-then-else, switch), the looping statements (for, while, do-while), and the branching statements (break, continue, return) supported by the Java programming language.

Please feel free to use other references and resources which you think will greatly help you in your programming journey. Enjoy coding!

II. Learning Objectives

After completing this module, you should be able to design and write programs that use:

1. Decision statements or constructs.
2. Looping/Repetition/Iteration statements or constructs.
3. Branching statements.

III. Topics and Key Concepts

In the programs that we have examined to this point, each of the statements is executed once, in the order given. Most programs are more complicated because the sequence of statements and the number of times each is executed can vary. We use the term control flow to refer to statement sequencing in a program.

A. Program Control Flow

In computer programming, control flow or flow of control is the order function calls, instructions, and statements are executed or evaluated when a program is running. Many programming languages have what are called control flow statements, which determine what section of code is run in a program at any time.

By default, statements are executed in sequence (one line after another) -- like following a recipe.

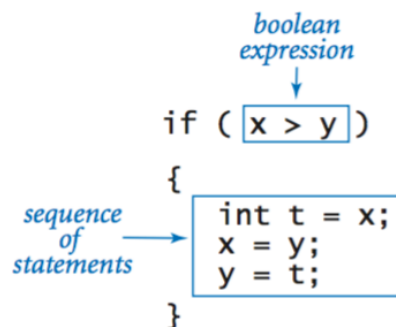
There are 3 types of control flow statements supported by the Java programming language.



1. Decision-making statements : if-then, if-then-else, switch
2. Looping statements : for, while, do-while
3. Branching statements : break, continue, return

B. The if-then statement

The if-then statement is the most basic of all the control flow statements. It tells your program to execute a certain section of code only if a particular test evaluates to true.



The code fragment as shown uses an if statement to put the smaller of two int values in x and the larger of the two values in y, by exchanging the values in the two variables if necessary.

Another example, the Bicycle class could allow the brakes to decrease the bicycle's speed only if the bicycle is already in motion. One possible implementation of the applyBrakes method could be as follows:

```
void applyBrakes() {
    // the "if" clause: bicycle must be moving
    if (isMoving){
        // the "then" clause: decrease current speed
        currentSpeed--;
    }
}
```

If this test evaluates to false (meaning that the bicycle is not in motion), control jumps to the end of the if-then statement.

In addition, the opening and closing braces are optional, provided that the "then" clause contains only one statement:

```
void applyBrakes() {
    // same as above, but without braces
    if (isMoving)
        currentSpeed--;
}
```



Deciding when to omit the braces is a matter of personal taste. Omitting them can make the code more brittle. If a second statement is later added to the "then" clause, a common mistake would be forgetting to add the newly required braces. The compiler cannot catch this sort of error; you'll just get the wrong results.

C. The if-then-else Statement

The if-then-else statement provides a secondary path of execution when an "if" clause evaluates to false. You could use an if-then-else statement in the applyBrakes method to take some action if the brakes are applied when the bicycle is not in motion. In this case, the action is to simply print an error message stating that the bicycle has already stopped.

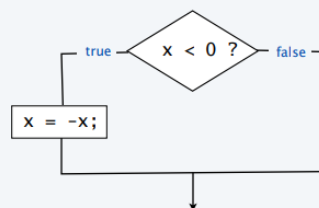
```
void applyBrakes() {  
    if (isMoving) {  
        currentSpeed--;  
    } else {  
        System.err.println("The bicycle has already stopped!");  
    }  
}
```

The if statement

Execute certain statements depending on the values of certain variables.

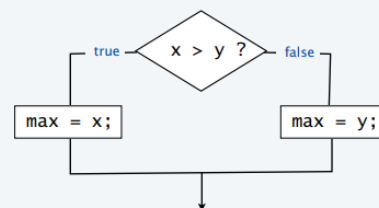
- Evaluate a boolean expression.
- If true, execute a statement.
- The **else** option: If false, execute a different statement.

Example: if (x < 0) x = -x;



Replaces x with the absolute value of x

Example: if (x > y) max = x;
else max = y;



Computes the maximum of x and y

The following program, IfElseDemo, assigns a grade based on the value of a test score: an A for a score of 90% or above, a B for a score of 80% or above, and so on.



```
class IfElseDemo {  
    public static void main(String[] args) {  
  
        int testscore = 76;  
        char grade;  
  
        if (testscore >= 90) {  
            grade = 'A';  
        } else if (testscore >= 80) {  
            grade = 'B';  
        } else if (testscore >= 70) {  
            grade = 'C';  
        } else if (testscore >= 60) {  
            grade = 'D';  
        } else {  
            grade = 'F';  
        }  
        System.out.println("Grade = " + grade);  
    }  
}
```

The output from the program is:

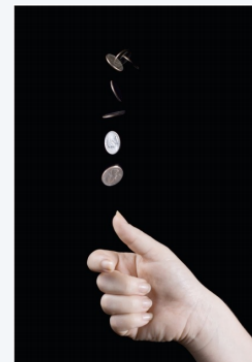
Grade = C

You may have noticed that the value of testscore can satisfy more than one expression in the compound statement: $76 \geq 70$ and $76 \geq 60$. However, once a condition is satisfied, the appropriate statements are executed (grade = 'C';) and the remaining conditions are not evaluated.

Example of if statement use: simulate a coin flip

```
public class Flip  
{  
    public static void main(String[] args)  
    {  
        if (Math.random() < 0.5)  
            System.out.println("Heads");  
        else  
            System.out.println("Tails");  
    }  
}
```

```
% java Flip  
Heads  
  
% java Flip  
Heads  
  
% java Flip  
Tails  
  
% java Flip  
Heads
```





D. The switch Statement

Unlike if-then and if-then-else statements, the switch statement can have a number of possible execution paths. A switch works with the byte, short, char, and int primitive data types. It also works with enumerated types the String class, and a few special classes that wrap certain primitive types: Character, Byte, Short, and Integer.

The following code example, SwitchDemo, declares an int named month whose value represents a month. The code displays the name of the month, based on the value of month, using the switch statement.

```
public class SwitchDemo {  
    public static void main(String[] args) {  
  
        int month = 8;  
        String monthString;  
        switch (month) {  
            case 1: monthString = "January";  
                    break;  
            case 2: monthString = "February";  
                    break;  
            case 3: monthString = "March";  
                    break;  
            case 4: monthString = "April";  
                    break;  
            case 5: monthString = "May";  
                    break;  
            case 6: monthString = "June";  
                    break;  
            case 7: monthString = "July";  
                    break;  
            case 8: monthString = "August";  
                    break;  
            case 9: monthString = "September";  
                    break;  
            case 10: monthString = "October";  
                    break;  
            case 11: monthString = "November";  
                    break;  
            case 12: monthString = "December";  
                    break;  
            default: monthString = "Invalid month";  
                    break;  
        }  
        System.out.println(monthString);  
    }  
}
```



In this case, August is printed to standard output.

The body of a switch statement is known as a switch block. A statement in the switch block can be labeled with one or more case or default labels. The switch statement evaluates its expression, then executes all statements that follow the matching case label.

You could also display the name of the month with if-then-else statements:

```
int month = 8;  
if (month == 1) {  
    System.out.println("January");  
} else if (month == 2) {  
    System.out.println("February");  
}  
... // and so on
```

Deciding whether to use if-then-else statements or a switch statement is based on readability and the expression that the statement is testing. An if-then-else statement can test expressions based on ranges of values or conditions, whereas a switch statement tests expressions based only on a single integer, enumerated value, or String object.

Another point of interest is the break statement. Each break statement terminates the enclosing switch statement. Control flow continues with the first statement following the switch block. The break statements are necessary because without them, statements in switch blocks fall through: All statements after the matching case label are executed in sequence, regardless of the expression of subsequent case labels, until a break statement is encountered.

Using a break is recommended so that modifying the code is easier and less error prone. The default section handles all values that are not explicitly handled by one of the case sections.

Ensure that the expression in any switch statement is not null to prevent a NullPointerException from being thrown.

E. The while and do-while Statements

Many computations are inherently repetitive. The while loop enables us to execute a group of statements many times. This enables us to express lengthy computations without writing lots of code.

The while statement continually executes a block of statements while a particular condition is true. Its syntax can be expressed as:

initialization is a separate statement
`int power = 1;`
loop-continuation condition
`while (power <= n/2)`
braces are optional when body is a single statement
`{`
`power = 2*power;`
`}`
body

The code fragment shown computes the largest power of 2 that is less than or equal to a given positive integer n.

The while statement continually executes a block of statements while a particular condition is true.

The while loop

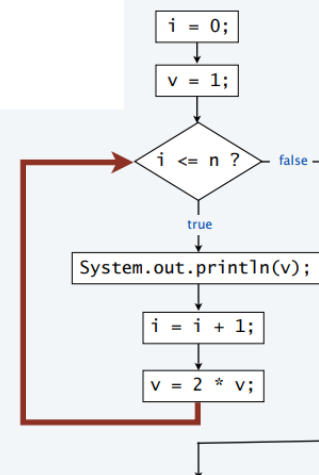
Execute certain statements repeatedly until certain conditions are met.

- Evaluate a boolean expression.
- If true, execute a sequence of statements.
- Repeat.

Example:

```
int i = 0;
int v = 1;
while (i <= n)
{
    System.out.println(v);
    i = i + 1;
    v = 2 * v;
}
```

Prints the powers of two from 2^0 to 2^n .
[stay tuned for a trace]



The while statement evaluates expression, which must return a boolean value. If the expression evaluates to true, the while statement executes the statement(s) in the while block. The while statement continues testing the expression and executing its block until the expression evaluates to false.

Using the while statement to print the values from 1 through 10 can be accomplished as in the following WhileDemo program:



```
class WhileDemo {  
    public static void main(String[] args){  
        int count = 1;  
        while (count < 11) {  
            System.out.println("Count is: " + count);  
            count++;  
        }  
    }  
}
```

You can implement an infinite loop using the while statement as follows:

```
while (true){  
    // your code goes here  
}
```

Example of while loop use: print powers of two

A [trace](#) is a table of variable values after each statement.

```
public class PowersOfTwo  
{  
    public static void main(String[] args)  
    {  
        int n = Integer.parseInt(args[0]);  
        int i = 0;  
        int v = 1;  
        while (i <= n)  
        {  
            System.out.println(v);  
            i = i + 1;  
            v = 2 * v;  
        }  
    }  
}
```

Prints the powers of two from 2^0 to 2^n .

i	v	i <= n
0	1	true
1	2	true
2	4	true
3	8	true
4	16	true
5	32	true
6	64	true
7	128	false

4096	256	8	32
64	512	128	4
32	16	64	2
4	2	8	4

```
% java PowersOfTwo 6  
1  
2  
4  
8  
16  
32  
64
```

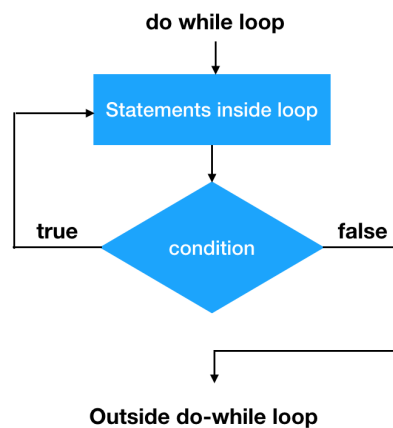
The Java programming language also provides a do-while statement, which can be expressed as follows:

```
do {  
    statement(s)  
} while (expression);
```

The difference between do-while and while is that do-while evaluates its expression at the bottom of the loop instead of the top. Therefore, the statements within the do block are always executed at least once, as shown in the following DoWhileDemo program:



```
class DoWhileDemo {  
    public static void main(String[] args){  
        int count = 1;  
        do {  
            System.out.println("Count is: " + count);  
            count++;  
        } while (count < 11);  
    }  
}
```



F. The for Statement

The for statement provides a compact way to iterate over a range of values. Programmers often refer to it as the "for loop" because of the way in which it repeatedly loops until a particular condition is satisfied. The general form of the for statement can be expressed as follows:

```
for (initialization; termination;  
    increment) {  
    statement(s)  
}
```

When using this version of the for statement, keep in mind that:

- The initialization expression initializes the loop; it's executed once, as the loop begins.
- When the termination expression evaluates to false, the loop terminates.
- The increment expression is invoked after each iteration through the loop; it is perfectly acceptable for this expression to increment or decrement a value.

The following program, ForDemo, uses the general form of the for statement to print the numbers 1 through 10 to standard output:



```
class ForDemo {  
    public static void main(String[] args){  
        for(int i=1; i<11; i++){  
            System.out.println("Count is: " + i);  
        }  
    }  
}
```

Notice how the code declares a variable within the initialization expression. The scope of this variable extends from its declaration to the end of the block governed by the for statement, so it can be used in the termination and increment expressions as well. If the variable that controls a for statement is not needed outside of the loop, it's best to declare the variable in the initialization expression. The names i, j, and k are often used to control for loops; declaring them within the initialization expression limits their life span and reduces errors.

```
int power = 1;  
for (int i = 0; i <= n; i++)  
{  
    System.out.println(i + " " + power);  
    power = 2*power;  
}
```

The for notation

The three expressions of the for loop are optional; an infinite loop can be created as follows:

```
// infinite loop  
for ( ; ; ) {  
  
    // your code goes here  
}
```

The for statement also has another form designed for iteration through Collections and arrays. This form is sometimes referred to as the enhanced for statement, and can be used to make your loops more compact and easier to read. To demonstrate, consider the following array, which holds the numbers 1 through 10:

```
int[] numbers = {1,2,3,4,5,6,7,8,9,10};
```



The following program, EnhancedForDemo, uses the enhanced for to loop through the array:

```
class EnhancedForDemo {  
    public static void main(String[] args){  
        int[] numbers =  
            {1,2,3,4,5,6,7,8,9,10};  
        for (int item : numbers) {  
            System.out.println("Count is: " + item);  
        }  
    }  
}
```

In this example, the variable `item` holds the current value from the numbers array.

It is recommended to use this form of the for statement instead of the general form whenever possible.

The for loop

An alternative repetition structure. ← Why? Can provide code that is more compact and understandable.

- Evaluate an *initialization statement*.
- Evaluate a *boolean expression*.
- If true, execute a *sequence of statements*, then execute an *increment statement*.
- Repeat.

Example:

```
int v = 1;  
for (int i = 0; i <= n; i++)  
{  
    System.out.println( i + " " + v );  
    v = 2*v;  
}
```

Prints the powers of two from 2^0 to 2^n

initialization statement

boolean expression

increment statement

Every for loop has an equivalent while loop:

```
int v = 1;  
int i = 0;  
while (i <= n; )  
{  
    System.out.println( i + " " + v );  
    v = 2*v;  
    i++;  
}
```

G. Branching Statements: the break statement

In some situations, we want to immediate exit a loop without letting it run to completion. Java provides the `break` statement for this purpose.

The `break` statement has two forms: labeled and unlabeled. You saw the unlabeled form in the previous discussion of the `switch` statement. You can also use an unlabeled `break` to terminate a `for`, `while`, or `do-while` loop.

Note that the `break` statement does not apply to `if` or `if-else` statements.



```
class BreakDemo {
    public static void main(String[] args) {

        int[] arrayOfInts =
            { 32, 87, 3, 589,
              12, 1076, 2000,
              8, 622, 127 };
        int searchfor = 12;

        int i;
        boolean foundIt = false;

        for (i = 0; i < arrayOfInts.length; i++) {
            if (arrayOfInts[i] == searchfor) {
                foundIt = true;
                break;
            }
        }

        if (foundIt) {
            System.out.println("Found " + searchfor + " at index " + i);
        } else {
            System.out.println(searchfor + " not in the array");
        }
    }
}
```

The program shown searches for the number 12 in an array. The break statement, shown in boldface, terminates the for loop when that value is found. Control flow then transfers to the statement after the for loop. This program's output is:

Found 12 at index 4

In this module, we will only cover unlabeled break and continue statements.

H. Branching Statements: the continue statement

The continue statement skips the current iteration of a for, while, or do-while loop. The unlabeled form skips to the end of the innermost loop's body and evaluates the Boolean expression that controls the loop.

The following program, ContinueDemo, steps through a String, counting the occurrences of the letter "p". If the current character is not a p, the continue statement skips the rest of the loop and proceeds to the next character. If it is a "p", the program increments the letter count.



```
class ContinueDemo {
    public static void main(String[] args) {

        String searchMe = "peter piper picked a " + "peck of pickled peppers";
        int max = searchMe.length();
        int numPs = 0;

        for (int i = 0; i < max; i++) {
            // interested only in p's
            if (searchMe.charAt(i) != 'p')
                continue;

            // process p's
            numPs++;
        }
        System.out.println("Found " + numPs + " p's in the string.");
    }
}
```

Here is the output of this program:

Found 9 p's in the string.

To see this effect more clearly, try removing the continue statement and recompiling. When you run the program again, the count will be wrong, saying that it found 35 p's instead of 9.

I. **Branching Statements: the return statement**

The last of the branching statements is the return statement. The return statement exits from the current method, and control flow returns to where the method was invoked. The return statement has two forms: one that returns a value, and one that doesn't. To return a value, simply put the value (or an expression that calculates the value) after the return keyword.

```
return ++count;
```

The data type of the returned value must match the type of the method's declared return value. When a method is declared void, use the form of return that doesn't return a value.

```
return;
```

The Classes and Objects lesson will cover everything you need to know about writing methods.

J. Nesting Conditionals and Loops

Nesting conditionals and loops

Nesting

- Any "statement" within a conditional or loop may itself be a conditional or a loop statement.
- Enables complex control flows.
- Adds to challenge of debugging.



Example:

```
for (int t = 0; t < trials; t++)
{
    int cash = stake;
    while (cash > 0 && cash < goal)
        if (Math.random() < 0.5) cash++;
        else cash--;
    if (cash == goal) wins++;
}
```

if-else statement
within a while loop
within a for loop

[Stay tuned for an explanation of this code.]

Example of nesting conditionals: Tax rate calculation

Goal. Given income, calculate proper tax rate.

income	rate
0 – \$47,450	22%
\$47,450 – \$114,649	25%
\$114,650 – \$174,699	28%
\$174,700 – \$311,949	33%
\$311,950 +	35%

```
if (income < 47450) rate = 0.22;
else
{
    if (income < 114650) rate = 0.25;
    else
    {
        if (income < 174700) rate = 0.28;
        else
        {
            if (income < 311950) rate = 0.33;
            else rate = 0.35;
        }
    }
}
```

if statement
within an if statement

if statement
within an if statement
within an if statement

if statement
within an if statement
within an if statement
within an if statement

K. The Conditional Operator

The conditional operator `?:` is a ternary operator (three operands) that enables you to embed a conditional within an expression. The three operands are separated by the `?` and `:` symbols. If the first operand (a Boolean expression) is true, the result has the value of the second expression; otherwise, it has the value of the third expression.

```
int min = (x < y) ? x : y;
```



IV. Teaching and Learning Materials Resources

- PC Computer | Laptop | Android Phone
- Gordon College LAMP
- Google Meet
- Facebook Messenger

V. Learning Tasks

A. Explore (45 points)

You are tasked to create/edit, compile and run the following sample codes that were used in this module:

1. IfElseDemo.java

Your code (provide screenshot)

Sample Run: your terminal window (provide screenshot)

2. Flip.java

Your code (provide screenshot)



Sample Run: your terminal window (provide screenshot)

3. SwitchDemo.java

Your code (provide screenshot)

Sample Run: your terminal window (provide screenshot)

4. WhileDemo.java

Your code (provide screenshot)



Sample Run: your terminal window (provide screenshot)

5. DoWhileDemo.java

Your code (provide screenshot)

Sample Run: your terminal window (provide screenshot)

6. ForDemo.java

Your code (provide screenshot)



Sample Run: your terminal window (provide screenshot)

7. EnhancedForDemo.java

Your code (provide screenshot)

Sample Run: your terminal window (provide screenshot)

8. BreakDemo.java

Your code (provide screenshot)



Sample Run: your terminal window (provide screenshot)

9. ContinueDemo.java

Your code (provide screenshot)

Sample Run: your terminal window (provide screenshot)

B. Explain

1. For each of the following control flow statement, draw its corresponding flowchart representation. (25 points)

1. if



2. if-else	
3. if-else-if/ switch	
4. while/for	
5. do-while	

2. If you are asked to write a program that uses loop statements, what factors will you consider in choosing a while loop over a for loop or vice versa? Explain briefly. (10 points)

--



3. What is wrong with the following code? (5 points)

```
public class PQwhile
{
    public static void main(String[] args)
    {
        int n = Integer.parseInt(args[0]);
        int i = 0;
        int v = 1;
        while (i <= n)
            System.out.println(v);
            i = i + 1;
            v = 2 * v;
    }
}
```

4. What does the following program print? (5 points)

```
public class PQfor
{
    public static void main(String[] args)
    {
        int f = 0, g = 1;
        for (int i = 0; i <= 10; i++)
        {
            System.out.println(f);
            f = f + g;
            g = f - g;
        }
    }
}
```



C. Engage (40 points)

1. Write a Hellos.java program that prints the text “Hello” the n^{th} time where n is an integer as a command-line argument. Sample run:

```
/* **** */
* Compilation: javac Hellos.java
* Execution:   java Hellos n
*
* Prints ith Hello for i = 1 to n.
*
* % java Hellos
* 1st Hello
* 2nd Hello
* 3rd Hello
* 4th Hello
* 5th Hello
* 6th Hello
* 7th Hello
* 8th Hello
* 9th Hello
* 10th Hello
* 11th Hello
* 12th Hello
* 13th Hello
* 14th Hello
* 15th Hello
* 16th Hello
* 17th Hello
* 18th Hello
* 19th Hello
* 20th Hello
* 21st Hello
* 22nd Hello
* 23rd Hello
* 24th Hello
*
* **** */
```

Your code (provide screenshot)



--

Sample Run: your terminal window (provide screenshot)

--

2. Write a program that accepts three integer arguments from the command-line and displays “equal” if all three numbers are equal, and “not equal” if they are not. Name the file as AllNumsEqual.java.

Your code (provide screenshot)

--

Sample Run: your terminal window (provide screenshot)

--








3. Write a program that generates the result of rolling a six-sided die (an integer between 1 and 6). Name the file as DieRoll.java.

Your code (provide screenshot)

Sample Run: your terminal window (provide screenshot)

4. Write a program Typhoon.java that takes the wind speed (in kilometers per hour) as an integer command-line argument and prints the category of the typhoon. Below is a table of the wind speeds according to the PAGASA.

Tropical cyclone categories		
Category	Sustained winds	
 Super Typhoon	>220 km/h >119 knots	
 Typhoon	>118–220 km/h >64–119 knots	
 Severe tropical storm	>89–117 km/h >48–63 knots	
 Tropical storm	>62–88 km/h >34–47 knots	
 Tropical depression	>62–88 km/h >34–47 knots	

l!fe THE PHILIPPINE STAR philstarlife.com @philstarlife



Your code (provide screenshot)

Sample Run: your terminal window (provide screenshot)

VI. References

- Oracle. nd. "Oracle Java Documentation". <https://docs.oracle.com/javase/tutorial/>
- Oracle Academy. 2020. "Java Programming Instructor Resources". <https://academy.oracle.com>
- Sedgewick Robert, Princeton University, Wayne, Kevin. 2017. "Introduction to Programming in Java: An Interdisciplinary Approach, 2nd Edition". <https://introcs.cs.princeton.edu/java/home/>