Chapter 3: Selection and Comments

Learning Outcomes:

- ✓ Identify the importance of selection and where/how to use it
- ✓ Implement conditional statements in a program
- ✓ Identify the importance of using comments

Prerequisite Knowledge:

- ✓ Complete Chapter 2
- ✓ Be familiar with the Boolean operators used in Java (`> < <= >= != ==`)
- ✓ Be familiar with flow-chart diagrams and how they are used to demonstrate program logic

Keywords: Selection IF ELSE IF ELSE SWITCH CASE Batch

3.1 The Theory: Selection

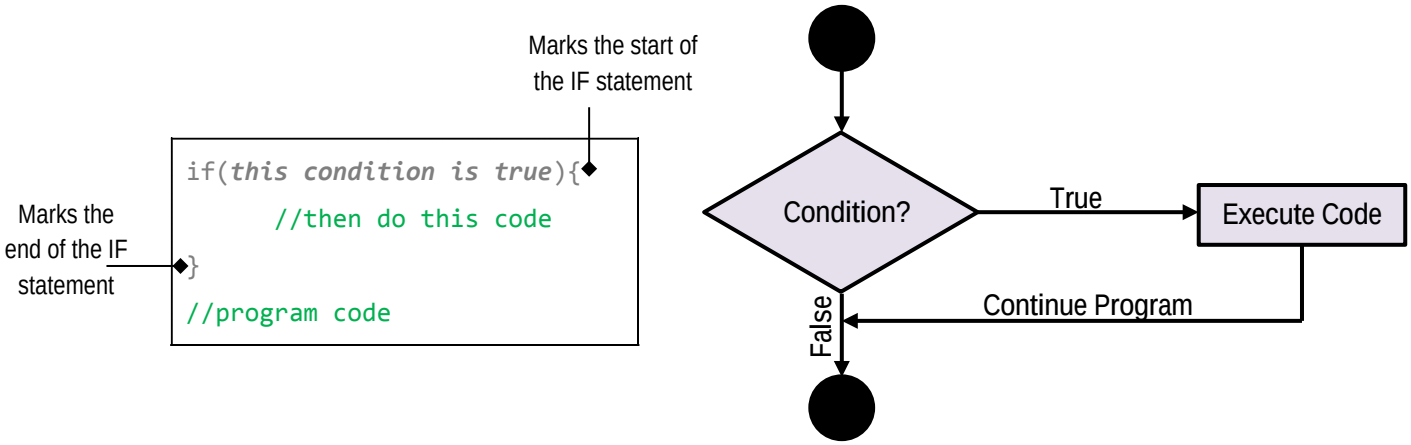
All of the program examples written in the previous chapters were linear and provided no choices. Most modern programs today (unless they are specifically written batch programs) require choices to make them interactive for a user; for example, an iPhone would not be much use to the user if they could not 'choose' to call somebody or to open a specific application. This 'choice' is made available through selection. There are different selection techniques available in Java; one of the most common is an 'IF statement'.

Programming Tips!

In Java, the single equals sign is used to assign a variable a value, i.e. `z=5`. However, two equals signs are used when making comparisons, i.e. `if(x==6)`, to check if 'x' and '6' are equal in value.

IF Statements

IF statements are fundamental in all programming languages; they allow selection in a program. In other words, they allow alternative paths in a program to be followed or avoided. The principles of an IF statement are the same in all languages; only the syntax differs. IF statements use Boolean operators to determine whether a condition is true or false. Common operators include `<` (less than), `<=` (less than or equal to), `==` (equal to), `!=` (not equal to), `>` (greater than) and `>=` (greater than or equal to). The logic of an IF statement is demonstrated in the diagram below:



Java syntax stipulates that the condition is placed between the brackets, following the [`if`] keyword; the condition uses Boolean operators to determine whether the condition is true. If the condition is true, then the code between both curly brackets [`{ }`] is executed. If the condition is false, then the code inside of the IF statement is skipped and the program continues the execution of the rest of the program code.

3.2 Practical: Using Selection

My First IF Program

The example below demonstrates a simple IF statement that compares the value of `x` to determine whether a condition is true or false. Notice that if the condition is true (`x` has a value of less than 10) then both statements are printed to the command-line. If the condition is false, only the end statement is printed. Here's a conundrum: 'if `x` has the value 10, what will the output be?'

Activity 3.1

Experiment with the value of x in the program below and check the results. Try the following numbers: 9, 10, 11 and 12.

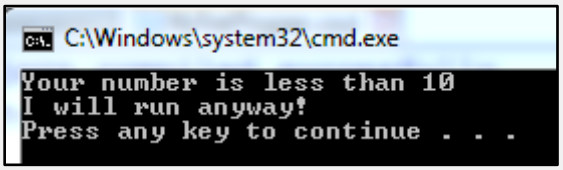
Try the code

```
public class MyFirstIf{

	public static void main(String[] args){
		int x = 5;
		if(x < 10){
		    System.out.println("Your number is less than 10");
		}
		System.out.println("I will run anyway!");
	}
}
```

Step by step

1. Type out the above code
2. Save the program as MyFirstIf.java
3. Compile the program (use the javac compiler)
4. Run the MyFirstIf.class file
5. Check the results of the program



More Conditions – AND & OR

A second (and more) condition(s) can be added using the keyword [`&&`], which means 'AND', between two condition, i.e. [`if(y > 5 && y < 10)`]. The previous condition now states that `y` has to be more than 5 and less than 10. When using the [`&&`] keyword, both conditions must be true before the code contained in the IF statement will execute. Alternatively, the keyword [`||`], which is short for 'OR', can be used to execute code if any of the two (or more) conditions are found to be true. The Java syntax for using multiple conditions is shown below:

If z is equal to 5 or y is equal to 10 then...

```
if(z == 5 || y == 10) {
      //then do this code
}

//program code
```

Activity 3.2

Write a program that implements an IF statement to test variable `z`, to determine if its value is greater than or equal to 10 and less than or equal to 20. Change the value of `z` several times to test the statement.

Activity 3.3

The Driving Licence Checker – write a program that uses a single IF statement to check the status of somebody's driving licence. The program should use a variable called `points`. If the `points` variable has a value of 12 or more, it should inform the user that they are disqualified. Change the value of the variable to ensure the program functions correctly.

3.3 The Theory: Extended Selection

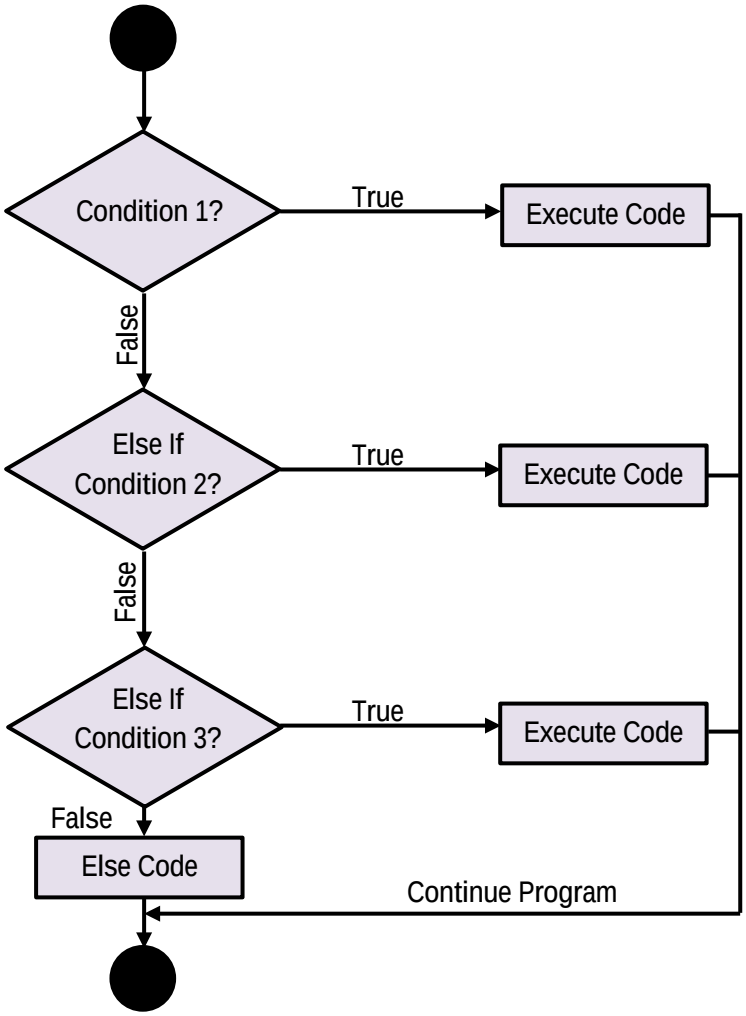
Extending IF, with ELSE and ELSE IF

Sometimes programmers find that a single conditional statement is just not enough. Consider the driving licence program from earlier; what if the programmer wanted to display a message for 3 points and 6 points too? The programmer cannot use separate IF statements because if a user had more than 12 points, then all three statements would execute as all conditions would return true; see below:

```
int points = 13;
if(points >= 12){
     System.out.println("Disqualified!");
}
if(points >= 6){
     System.out.println("Warning!");
}
if(points >= 3){
     System.out.println("Drive Safer!");
}
```

```
C:\Windows\system32\cmd.exe
Disqualified!
Warning!
Drive Safer!
Press any key to continue . . .
```

However, the [`else if`] keyword can be used to extend an IF statement by adding additional conditions to check whether the previous condition returns false. This method is much more efficient than writing many individual IF statements. The [`else`] keyword is also useful and can be used to catch any other possibilities if all conditions return false. The logic of an extended IF statement is shown to the right:




```
if(this condition is true){
      //then do this code
} else if(this condition is true){
      //then do this code
} else if(this condition is true){
      //then do this code
} else if(this condition is true){
      //then do this code
} else {
       //if all else fails...
       //then do this code
}
//program code
```


3.4 Practical: Using Extended Selection

My Extended IF Program

The example below demonstrates an extended IF statement that compares the value of `x` against multiple conditions; if a condition returns true, then its code will be executed and the flow of the IF statement will end. If all conditions in the statement return false however, then the `else` code will be executed.

Activity 3.4

Experiment with the value of `x` (in the program below) and check the results. Try the following numbers: 9, 10, 45, 50, 55, 70, 100 and 110.

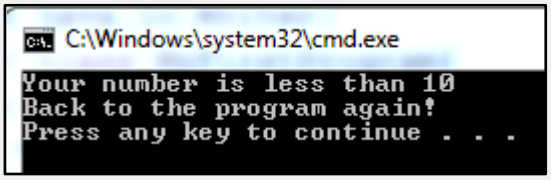
Try the code

```
public class MyExtendedIf{

	public static void main(String[] args){
		int x = 5;
		if(x < 10){
		    System.out.println("Your number is less than 10");
		}else if(x < 50){
		    System.out.println("Your number is less than 50");
		}else if(x < 100){
		    System.out.println("Your number is less than 100");
		}else{
		    System.out.println("Your number is 100 or more!");
		}
		System.out.println("Back to the program again!");
	}
}
```

Step by step

1. Type out the above code
2. Save the program as MyExtendedIf.java
3. Compile the program (use the javac compiler)
4. Run the MyExtendedIf.class file
5. Check the results of the program



The order of an IF statement is important and it will impact on a written program. Consider this example:

```
if(x > 5) {//say more than 5}
else if(x > 10) {//say more than 10}
else if(x > 15) {//say more than 15}
```

If `x` has the value 16, which condition will execute? The answer is '*say more than 5*'. Unfortunately, many beginners expect the answer to be '*say more than 15'*. They fall for the trap of thinking that a program will choose the most logical condition out of all of the possible conditions and execute its contained code. Remember, programs always work in sequence; each condition is checked in order, starting with the first condition. If that condition is true, none of the other conditions will execute. The following code would be much more efficient.

```
if(x > 15) {//say more than 15}
else if(x > 10) {//say more than 10}
else if(x > 5) {//say more than 5}
```

Activity 3.5

Password Application – write a program that has a variable called `myPassword`. If the `myPassword` variable is set to 'letmein' then the application should say 'Access Granted', else it should return 'Access Denied'. To compare two strings, for now, use the double equals operator (`==`). Change the value of the `myPassword` variable to ensure that the program works as expected.

Now if `x` has the value 16, the first condition will be true and thus execute the appropriate code. However, if `x` has a lesser value, the first condition will return false and the following conditions will be checked instead.

3.5 Practical: Using SWITCH CASE

My SWITCH CASE Program

'SWITCH CASE' is another selection technique available in Java and sometimes a possible alternative to an IF statement. The principles of SWITCH CASE, however, are similar to that of an IF statement; both techniques can sometimes be used interchangeably. SWITCH CASE is particularly useful when testing the value of a variable. The logic of a SWITCH CASE statement is shown in the code below:

```
String myVariable = "hello Kelly";
switch (myVariable){
      case "hello Joe":
            //then do this code
      break;
      case "hello Kelly":
            //then do this code
      break;
      default:
            //then do this code
      break;
}
//program code
```

SWITCH CASE tests the content of a variable against several cases and executes any code accordingly. SWITCH CASE also executes in order, checking the 'cases' in sequence until a condition is found to be true. Notice that the variable to be tested is placed between the brackets following the keyword [`switch`]. The value to be matched is then placed after the keyword [`case`] and the value is proceeded by a colon [`:`]. If the variable being tested is integer, then no quotes are required after the [`case`] keyword. The code to be executed goes on the following line(s) after the [`case`] keyword. The [`break;`] keyword is placed after the code to be executed for each case; this is to 'break' the flow (exit) of the SWITCH CASE statement and return to the program code. Finally, the keyword [`default:`] can be used to execute code if all 'cases' return false; this is similar to the [`else`] of an IF statement.

Activity 3.6

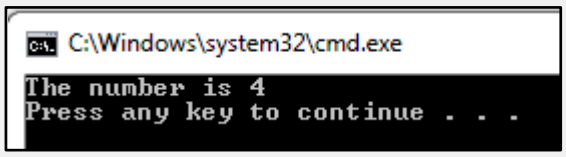
Experiment with the value of `x` (in the program below) and check the results. Try the following numbers; 1, 2, 3, 4 and 5.

Try the code

```
public class MySwitchCase{
      public static void main(String[] args){
            int x = 4;
            switch (x){
                  case 2:
                        System.out.println("The number is 2");
                  break;
                  case 4:
                        System.out.println("The number is 4");
                  break;
                  default:
                        System.out.println("Number unknown");
                  break;
            }
      }
}
```

Step by step

1. Type out the above code
2. Save the program as MySwitchCase.java
3. Compile the program (use the javac compiler)
4. Run the MySwitchCase.class file
5. Check the results of the program



3.6 Practical: Using Comments

Applying Comments to Code

Throughout this chapter comments have been added to some of the examples, i.e. [`//then do this code`]. In Java, single-line comments are added using two forward slashes [`//`]. Any text or code that follows the forward slashes will be ignored by the compiler. Comments can be added pretty much anywhere in programming code and are not executed. Comments act as visual aids for a programmer and can be used to:

- Specify the author and version of software
- 'Comment out' unwanted or old code
- Test code (by commenting out segments of code)
- Record changes made to software
- Leave notes for other programmers to follow

Programming Tips!

In the previous activity, the double equals sign (==) was used to compare two strings. However, this is not efficient and can cause problems in a program. Instead the [`equals()`] method, from the `String` class, should be used. An example of this is shown below:

```
String x = "match me";
if(x.equals("match me"){

System.out.println("matched")
;
}
```

The example below shows a program that has been annotated with comments. Notice that comments have been used to explain specific parts of the code as well as general software details:

```
// Version: #1.01
// Last Modified: 23/02/2014
// Modified By: Joseph Hopwood
// Author: James Olivando

public class MySwitchCase{ // this is a class

      public static void main(String[] args){ // this is a method
            int x = 4; // this is an integer variable
            switch (x){ // start of switch case statement
                  case 2:
                        System.out.println("The number is 2");
                        break;
                  case 4:
                        System.out.println("The number is 4");
                        break;
                  default:
                        System.out.println("Number unknown");
            } // end of switch case statement
      } // end of method
} // end of class
```

Activity 3.7

Open up an existing program and add comments of your own explaining how the program works.

