Flow of control – Part 1

All Java statements that we have seen so far are executed from the top to the bottom of the program in the order that they appear. Flow of control allows a break up of the flow of execution by using selection statement, repetition statement and branching so that a program can be conditionally executed a particular block of code.

Selection Statement → Allows a program to perform decision-making

1. **Single Selection Statement**
   As in most programming languages, Java provides flow of control with a single selection statement known as "**if statement**". This kind of statement chooses an action based on condition provided. Literally, an "if statement" says that "if a condition provided is true, then executes this particular action." The example below shows a Java's "if statement" that evaluates an integer acquired from a user whether that particular integer is a positive number or not. A condition to be evaluated as either true or false can be constructed as a "**Boolean expression**" which two operands are compared.

```
1  import java.util.Scanner;
2  public class Example {
3    public static void main(String [] args) {
4      Scanner input = new Scanner(System.in);
5      System.out.print("Enter an integer:");
6      int number = input.nextInt();
7      if(number > 0) {
8        System.out.printf("%d is positive.", number);
9      }
10   }
11 }
```

   On line 7, the code shows the syntax of "if statement" that it begins with a Java's keyword "if" followed by a condition known as "Boolean expression". When the Boolean expression is used with an "if statement", it must be enclosed in parentheses. "if(number > 0)" is actually translated as "if number is greater than zero". An "if statement" must contain its action enclosed in **curly brackets as shown on line 7 and 9.** The action inside the brackets will only be executed when the Boolean expression of the "if statement" is evaluated as "true".

For example:

- When the program is run and an integer "1" is entered, the output is displayed because 1 is greater than 0, so (number > 0) is true.

```
> run Example
Enter an integer: 1

1 is positive.
```

- When the program is run and an integer "-1" is entered, there is no output being displayed because -1 is not greater than 0, so (number > 0) is false.

```
> run Example
Enter an integer: -1
```

- When the program is run and an integer "0" is entered, there is no output being displayed as well because 0 is not greater than 0, so (number > 0) is false.

```
> run Example
Enter an integer: 0
```

2. **Double Selection Statement**

To handle the other case in this program where the number is not positive, Java provides another type of selection statement called **"if-else statement"**. The example below shows a Java's "if –else statement" that evaluates an integer acquired from a user whether that particular integer is a positive number or not. The "if-else statement" chooses between two alternation actions based on the value of a Boolean expression.

```
1  import java.util.Scanner;
2  public class Example {
3    public static void main(String [] args) {
4      Scanner input = new Scanner(System.in);
5      System.out.print("Enter an integer:");
6      int number = input.nextInt();
7      if(number > 0) {
8        System.out.printf("%d is positive.", number);
9      }
10     else {
11       System.out.printf("%d is not positive.", number);
12     }
13   }
14 }
```

On line 10, the code shows the syntax of an "else statement" that it begins with a Java's keyword "else". The "else" statement does not need a Boolean expression because it handles any case that is evaluated as "false" from the "if statement" above it. Similarly, the "else statement" must contain its action enclosed in **curly brackets as shown on line 10 and 12**.

The program now can handle both cases, positive and non-positive numbers. If "number > 0" is evaluated as false, the "else" statement will handle the situation.

For example:

- When the program is run and an integer "1" is entered, the output is displayed because 1 is greater than 0, so (number > 0) is true.

```
> run Example
Enter an integer: 1

1 is positive.
```

- When the program is run and an integer "-1" is entered, there is now an output being displayed as well. Once the Boolean expression of "if statement", (number > 0), is evaluated as false, the "else" statement then takes the action.

```
> run Example
Enter an integer: -1

-1 is not positive.
```

- When the program is run and an integer "0" is entered, there is now an output being displayed as well. Once the Boolean expression of "if statement", (number > 0), is evaluated as false, the "else" statement then takes the action.

```
> run Example
Enter an integer: 0

0 is not positive.
```

3. **Multiple Selection Statement**

   To handle multiple cases in this program where the number is positive, negative and zero, Java provides another type of selection statement called **"if-else if statement"**. The example below shows a Java's "if–else if statement" that evaluates an integer acquired from a user that this particular integer is a positive number, negative number or zero. The "if-else if statement" chooses between multiple alternation actions based on the value of a Boolean expression.

```
1  import java.util.Scanner;
2  public class Example {
3    public static void main(String [] args) {
4      Scanner input = new Scanner(System.in);
5      System.out.print("Enter an integer:");
6      int number = input.nextInt();
7      if(number > 0) {
8        System.out.printf("%d is positive.", number);
9      }
10     else if(number < 0) {
11       System.out.printf("%d is negative.", number);
12     }
13     else if(number == 0) {
14       System.out.printf("%d is zero.", number);
15     }
16   }
17 }
```

On line 10, the code shows the syntax of an "else if statement" that begins with a Java's keyword "else if" followed by a Boolean expression. Similarly, the "else if statement" must contain its action enclosed in **curly brackets as shown on line 10 and 12**. The Boolean expression, (number < 0), of "else if statement" on line 10 will be evaluated when the Boolean expression, (number > 0), of "if statement" on line 7 is false otherwise it will be ignored. If both Boolean expressions, (number > 0) and (number < 0), of "if statement" and "else if statement" on line 7 and 10 respectively are both false, the Boolean expression, (number == 0), of "else if statement" on line 13 will be evaluated.

Even though, this code of multiple selection statements works perfectly, code optimization can be employed here to improve the program's performance. In this example, there are only three possibilities that a number can only be positive number, negative number or zero. To optimize this code, if the two Boolean expressions on line 7 and 10 are false, it is then not necessary to evaluate the last Boolean expression, (number == 0), because if a number is neither positive nor negative, it must then be a zero. In this particular case, instead of using if, else if and else if, we can use if, else if and else to ignore the evaluation on the last Boolean expression and improve performance. Here is the optimized version.

```
1  import java.util.Scanner;
2  public class Example {
3    public static void main(String [] args) {
4      Scanner input = new Scanner(System.in);
5      System.out.print("Enter an integer:");
6      int number = input.nextInt();
7      if(number > 0) {
8        System.out.printf("%d is positive.", number);
9      }
10     else if(number < 0) {
11       System.out.printf("%d is negative.", number);
12     }
13     else {
14       System.out.printf("%d is zero.", number);
15     }
16   }
17 }
```

On line 13, instead of using else if(number == 0), it now uses "else statement" to avoid using the last Boolean expression to be evaluated. With this code, the "else statement" will take action when both Boolean expressions on line 7 and line 10 are false without having to evaluate the integer.

Comparison Operators

When a Boolean expression is constructed, a comparison operator must be used to determine if one operand is greater than, less than, equal to, or not equal to another operand. The result of the comparison of the two operands is either true or false. Here are the symbols used in Java to perform comparison.

- > greater than
- < less than
- >= greater than or equal to
- <= less than or equal to
- == equal to
- != not equal to

Logical Operators

Java also provides logical operators for defining multiple Boolean expressions in the selection statements. These logical operators are related to Geometry's Truth Tables. In this section, logical AND and OR operator will be stated.

Geometry's Truth Tables for AND operator

C1	C2	C1 AND C2
T	T	T
T	F	F
F	T	F
F	F	F

Geometry's Truth Tables for OR operator

C1	C2	C1 OR C2
T	T	T
T	F	T
F	T	T
F	F	F

Symbols used in Java for AND and OR operator

&&	Logical AND	Short-Circuiting
||	Logical OR	Short-Circuiting
&	Logical AND	Not Short-Circuiting
|	Logical OR	Not Short-Circuiting

The advantage of the short-circuiting ones is that the number of evaluation on the Boolean expression could be less. C

Considering the following Boolean expressions:
Assume that x = 1 and y = 2;
if(x == 0 & y == 2)
With this expression, x == 0 is false and y == 2 is true which results as false. Logically, with the AND operator, when there is a false statement, the result is always false. Since the first expression, x == 0, is false, the evaluation should have stopped there and do not need to evaluate the second expression, y == 2, because it doesn't matter what result the second expression produces, the result will be false.

On the other hand, when using **a short-circuiting one as shown below**, the second expression, y == 2, will never need to be evaluated.
if(x == 0 && y == 2)