Loops and Conditions

if, if-else, if-else if

If like in all other languages `if` meant for checking conditions.

```
public class IfConditionExample {
    public static void main(String s[]) {
        int i = 10;
        int j = 5;
        if(i>j) {
            System.out.println("i is bigger than j");
        }
    }
}
```

If - else

The block of code following else is executed when the condition checked by `if` is false.

```
public class IfConditionExample {
    public static void main(String s[]) {
        int i = 10;
        int j = 5;
        if(i>j) {
            System.out.println("i is bigger than j");
        } else {
            System.out.println("i is smaller than j");
        }
    }
}
```

Activity: Ask the user for a number input and accept the number input and see if the number is greater than 10

Let's use the If-else if - We use this when we have more than two conditions to check. Before we see this example, let's look at another class we will be using in the example to understand if-else if. LocalDateTime is a class which we can use to get the date/time. This class is a part of the time package and is not available by default. We have to import it.

```
import java.time.LocalDateTime;

//This class prints the date when you run it
public class PrintDateAndTime {
    public static void main(String s[]) {
        LocalDateTime localDateTime = LocalDateTime.now();
        System.out.println(localDateTime.getDayOfMonth() +
                " "+ localDateTime.getMonth();
```

```
                    + localDateTime.getMonth()+
                " "+ localDateTime.getYear());
    }
}
```

Let's use the LocalDateTime to print an appropriate statement for each day of the week.

```
import java.time.LocalDateTime;
public class IfElseIfConditionExample {
    public static void main(String[] s) {
        LocalDateTime localDateTime = LocalDateTime.now();
        String dayOfWeek = localDateTime.getDayOfWeek().toString();
        if(dayOfWeek.equalsIgnoreCase("Monday")) {
            System.out.println("It is the first day of the week!");
        } else if (dayOfWeek.equalsIgnoreCase("Tuesday")) {
            System.out.println("It is the second day of the week!");
        } else if (dayOfWeek.equalsIgnoreCase("Wednesday")) {
            System.out.println("Mid-week already!");
        } else if (dayOfWeek.equalsIgnoreCase("Thursday")) {
            System.out.println("One more day before you say TGIF");
        } else if (dayOfWeek.equalsIgnoreCase("Friday")) {
            System.out.println("Hurray! The last day of the work week");
        } else if (dayOfWeek.equalsIgnoreCase("Saturday")) {
            System.out.println("Weekend! Blissful!");
        } else if (dayOfWeek.equalsIgnoreCase("Sunday")) {
            System.out.println("Make hay while the Sun(day) shines!");
        }

    }
}
```

Activity Use combination of `if` and `logical operators` to print whether the current day is a weekday or weekend.

Switch ....case

Another elegant way to handle if...else if... else in Java is to use switch case. switch-case statements can be used for any kind of objects for which the equality can be checked. We will use switch-case in the above example instead of if-else if.

```
package com.myob.examples;

import java.time.LocalDateTime;

public class SwitchCaseExample {
    public static void main(String[] s) {
        LocalDateTime localDateTime = LocalDateTime.now();
        String dayOfWeek = localDateTime.getDayOfWeek().toString();
        System.out.println("It is a "+dayOfWeek);
        switch(dayOfWeek) {
            case "MONDAY" :     System.out.println("It is the first day of the week!");
```

```
                                break;
            case "TUESDAY" :    System.out.println("It is the second day of the week!");
                                break;
            case "WEDNESDAY" :  System.out.println("Mid-week already!");
                                break;
            case "THURSDAY" :   System.out.println("One more day before you say TGIF");
                                break;
            case "FRIDAY" :     System.out.println("Hurray! The last day of the work week");
                                break;
            case "SATURDAY" :   System.out.println("Weekend! Blissful!");
                                break;
            case "SUNDAY" :     System.out.println("Make hay while the Sun(day) shines!");
        }
    }
}
```

Activity - Why is the `break ` imperative ? What happens if you remove the `break`?

Formatted output

We used `+` to concatenate strings in the previous examples that we worked out. Java offers a method called `format` in String class.

```
public class StringFormatOutput {
    public static void main(String s[]) {
        String piStr = "Pi";
        double pi = 3.142;
        // %.2f prints only 2 digits after decimal
        System.out.println(String.format("The value of %s is %.2f",piStr,pi));
    }
}
```

Loops in Java:

There are three kinds of loops in Java.

1. for loop
2. while loop
3. do...while loop

For loop

There are two kinds of for loop. First one is where we mention the initial value, condition and the new value of the loop variable.

```
public class ForLoopExample {
    public static void main(String s[]) {
        int j = 10;
        //start with i=1. Continue till i<=12. Increment i by 1 after each iteration.
        for(int i=1; i<=12; i++) {
            System.out.println(String.format("%d X %d = %d",j,i,j*i));
        }
```

```
    }
}
```

Nested loop

Nested loop is having one loop inside another. We will print the times table for all the numbers from 1 to 5.

```
public class ForLoopExample {
    public static void main(String s[]) {
        for(int j=1; j<=5 ; j++) {
            System.out.println(String.format("\n\n%d Table\n", j));
            for (int i = 1; i <= 12; i++) {
                System.out.println(String.format("%d X %d = %d", j, i, j * i));
            }
        }
    }
}
```

Labelling the loop

Labelling the loop is used when you want to break or continue a specific loop when you have nested loops. Take the case when we want to print entire times tables (till times 12) for all numbers from 1 to 20 except 5, 10 and 15. For these 3 numbers we will only print till times 5.

```
public class ForNestedLoopExample {
    public static void main(String s[]) {
        outerLoop: for(int j=1; j<=20 ; j++) {
            innerLoop: for (int i = 1; i <= 12; i++) {
                if((j == 5 || j== 10 || j == 15)  && i>5) {
                    break innerLoop;
                }
                System.out.println(String.format("%d X %d = %d", j, i, j * i));
            }
        }
    }
}
```

Breaking from the inner loop is same as continuing the outerloop. Replace the code above with this.

```
public class ForNestedLoopExample {
    public static void main(String s[]) {
        outerLoop: for(int j=1; j<=20 ; j++) {
            innerLoop: for (int i = 1; i <= 12; i++) {
                if((j == 5 || j== 10 || j == 15)  && i>5) {
                    continue outerLoop;
                }
                System.out.println(String.format("%d X %d = %d", j, i, j * i));
            }
        }
```

```
        }
    }
}
```

💡 **Tip** : When there is no condition specified in the for loop, it gets into an infinite loop.

Activity Write code to print Arithmetic progression as long the user wants to. Take the initial value and incremental values from the user. Arithmetic progression is where the initial values is incremented by the incremental value.

while Loop

`while` loop is also a conditional loop, but there is no inital value or incremental value. Let's print 'Hello World' after 3 seconds.

```
import java.time.LocalDateTime;

public class WhileLoop {
    public static void main(String s[]) {
        LocalDateTime localDateTime = LocalDateTime.now();
        System.out.println("Start time : "+localDateTime);
        LocalDateTime untilTime = localDateTime.plusSeconds(3);
        System.out.println("End time : "+untilTime);
        while(LocalDateTime.now().compareTo(untilTime) <= 0) {
            //Just keep looping through for 3 seconds doing nothing
        }
        System.out.println("Hello World!!");
    }
}
```

do...while

The difference between `while` and `do while` is that `while` gets into the loop only when the condition is true. But in the case of `do while` the block of code which is meant to loop is executed atleast once before the condition is checked. The code below is going to get user input to create a customer object. It gets the data as long as the user wants to add.

```
import java.util.Scanner;

public class DoWhile {
    public static void main(String s[]) {
        Scanner scanner = new Scanner(System.in);
        String reply = "n";
        do {
            System.out.println("Enter your first name");
            String firstName = scanner.nextLine();
            System.out.println("Enter your last name");
            String secondName = scanner.nextLine();
            System.out.println("Enter your email");
            String email = scanner.nextLine();
            System.out.println("Enter your account number");
```

```
            System.out.println("Enter your account number");
            String accountNumber = scanner.nextLine();
            System.out.println("Enter your account type");
            String typeOfAccount = scanner.nextLine();
            System.out.println("Enter your address");
            String address = scanner.nextLine();

            Customer c1 = new Customer();
            c1.setFirstName(firstName);
            c1.setLastName(secondName);
            c1.setEmail(email);
            c1.setAccountNumber(accountNumber);
            c1.setTypeOfAccount(typeOfAccount);
            c1.setAddressForCommunication(address);

            System.out.println(c1);
            System.out.println("Do you want to enter another custiomer details? ");

            reply = scanner.nextLine();
        } while (reply.equalsIgnoreCase("Y"));
        scanner.close();
    }
}
```