Appendix A

The contents of this appendix refer to the script made during the process of course materials development. The material might contain grammar mistakes or uncompleted sections due to the fact that it is a draft.

Objects and Classes

In order to create a program there are 2 ways:

1. Using one class
2. Using multiple classes that are linked to each other

Think of this as having a school
A school teach multiple subjects, not just one!

What is the best option?

1. Put all the people in a single classroom - huge one
2. Split the people in smaller groups then set a timetable and put them in more classrooms which would increase the engagement and interactiveness between teacher and students

Which one is more efficient?What are the pros/cons?
If we put all the students in one room then if there are questions it will take long to answer all of them and maybe the others would want to go home not stay another hour just for Q/A
If we split them then it is tricky to do the timetables for each group - in the idea that unlimited amount of students can attend at any time of the year - no starting/ending date

After you learn the basics of Java programming you should know the components that create a program. Let's learn how can we use them to create one!

First, what is a class?
In our school example, let's say that each classroom has a label on the door with the subject taught in that classroom, so students would move around after each hour and search for their next class/subject in their timetable(ex. class Mathematics).
In order to identify the classes and make use of them in programming, we need to label/name them.

What elements are in a class?
A class can contain different type of elements, from variables to methods which have more categories

How do we define a class in programming?
We can simply just write class and then give it a name.
The file in which we write the code has to have the same name as the class defined in it.

A quick example:

```
-----------------------------------------------------------------
Mathematics.java
-----------------------------------------------------------------
public class Mathematics {
        …...
}
-----------------------------------------------------------------
```

First, what is in a class?
Local variables - as you might already know the local variables are the ones inside a method, a block or in a constructor(we will talk about this in a minute)
For example:

```
-----------------------------------------------------------------
```

```
Mathematics.java
----------------------------------------------------------------
public class Mathematics {
        public static void main(String args[]) {
                int myVariable = 8;
            ……
      }
}
----------------------------------------------------------------
```

'myVariable' is a classic local variable example, it can have any type of data type from byte, short, int to char.

Instance variables

These types of variables are the opposite to the local variables since they are declared outside any method, block or constructor.

```
----------------------------------------------------------------
Mathematics.java
----------------------------------------------------------------
public class Mathematics {
        int seatsAvailable;
        int booksAvailable;
        String teacherName;
 }
----------------------------------------------------------------
```

In the mathematics classroom we have to know how many seats are there, books so that students will have one on each table at least, the teacher's name, etc.

What do we do with these instance variables?

We use them in methods called **instance methods**. These methods make use of the variables in order to return a certain result based on each method.

Mathematics.java

```
public class Mathematics {
        int seatsAvailable;
        int booksAvailable;
        String teacherName;

        public void getTeacherName()
        {
                System.out.println("The mathematics teacher is called" +
teacherName);
        }
 }
```

In this instance method we want to retrieve the mathematics teacher's name. What can be the problem? We have the variable declared, the method as well, but how do we set a name to that teacher?

Constructor Method

Before we declare the instance methods and after the instance variables declaration we have to create a constructor method. This is going to be the control center which will assign each variable with a value.

 Mathematics.java

```
public class Mathematics {
        int seatsAvailable;
        String teacherName;

        public Mathematics(int seats, String name) {
                seatsAvailable = seats;
                teacherName = name;
        }

        public void getSeatsAvailable()
        {
                System.out.println("The number of seats available in the
        Mathematics classroom is " + seatsAvailable);
        }

        public void getTeacherName()
        {
                System.out.println("The mathematics teacher is called" +
        teacherName);
        }

 }
```

Basically the constructor method will have 3 parameters, each of them being assigned to the instance variables declared above the constructor method. After this assign action then the variables will be able to be used in the instance methods…. But how?

Java Objects

So far the concept of class is good for organising a program so when a problem appears the programmer would search for the class related to that problem instead of going through 100.000+ lines of code, but what is a class in essence?

Let's stick to the school example. In the school there are tons of objects, from pencils to tables to students and everything that is existing in it(students count as objects).
Each object has properties and behaviour. For example, the students, each one of them can be seen as an object that has different properties/looks to behaviour.

A class is basically a template that would be used to create objects.
For the mathematics classroom we have so far the number of seats and the teacher's name which represent some of the **classroom properties**.

The mathematics classroom is basically an object in the school. Here we can see a hierarchy of classes in the school case scenario:

```
                              class School

class Mathematics      class Biology              class Spanish ….

            Pens, seats, tables, students, books, etc.
```

In programming terms, a School class is the root of every other class(room) simply because each classroom from Mathematics to Biology, Spanish etc are included into it. All these classrooms have seats available, books available teachers and their name, pens, so common properties. Each of these classes is an object in the class School.

So in the end we have a 'class School' with different topic objects that has to be created in there. How?

First of all let's take a look back at the example with the complete Mathematics class. In order to get the result for the teacher name we need to supply the instance variable with a name.

```
-------------------------------------------------------------------
School.java
-------------------------------------------------------------------
public class School {
        public static void main(String args[]) {
        Mathematics mathClassroom = new Mathematics(32, “John Bury”);
        }
```

Mathematics.java

```
public class Mathematics {
	int seatsAvailable;
	String teacherName;

	public Mathematics(int seats, String name) {
		seatsAvailable = seats;
		teacherName = name;
	}
}
```

What we can see in the example is a school class with its main method and an object called mathClassroom. An object as you might guess is created by typing the class name followed by what name you want that object to have and then the ‘new’ keyword followed by the class name. After the class name we have some parameters passed in the brackets. Are these the suppliers for the variables?

Think of the syntax above as some kind of method call. In the left side we have the class name followed by the object name that we choose. In the right hand side we have the keyword ‘new’ that creates that object and then the constructor method is provided with arguments. The arguments must be provided in the order of the parameters declared in the constructor method.

```
public Mathematics(int seats, String name)
Mathematics(32, “John Bury”)
```

So in the end we create an object called mathClassroom in the class School that has 32 Seats available and a teacher called John Bury. How can we display the results of the instance methods in the class Mathematics?

Remember! In the example above we have JUST created an object, nothing else.

```
-----------------------------------------------------------------
School.java
-----------------------------------------------------------------
public class School {
        public static void main(String args[]) {
        Mathematics mathClassroom = new Mathematics(32, “John Bury”);

        mathClassroom.getSeatsAvailable();
        mathClassroom.getTeacherName();
        }
}
-----------------------------------------------------------------
```

```
-----------------------------------------------------------------
OUTPUT
-----------------------------------------------------------------
The number of seats available in the Mathematics classroom is 32
The mathematics teacher is called John Bury
```

In the above example we can see that in order to access/call the instance methods from a class, first we need to reference using a ‘.’. We take the object name then reference it using the dot and then type the instance method’s name followed by the parenthesis that can/not have arguments passed based on what type of method is it - accessor or mutator.

Now that we know how to create/populate a class, create an object and use it we will talk about mutator and accessor methods.

What is an accessor method?

In the example above all the instance methods were accessor. Those types of methods are used to retrieve/return an information. We were able to successfully retrieve the number of seats available in the classroom and the teacher's name.

The accessor methods are distinguished by the 'get' keyword used in front of the name given to the method.

```
public void getTeacherName()
public void getSeatsAvailable()
```

What is a mutator method?

A mutator method is used to set a value to a private variable field. The mutator method is having a naming convention scheme that make use of the keyword 'set' that is the prefix of the method name.

Let's complete the code example:

Mathematics.java

```
public class Mathematics {
        private int seatsAvailable;

        public Mathematics(int seats) {
                seatsAvailable = seats;
        }

        public void setSeatsAvailable(int seatsNumber) {
                seatsAvailable = seatsNumber;
        }

        public void getSeatsAvailable()
        {
                System.out.println(“The number of seats available in the
Mathematics classroom is ” + seatsAvailable);
        }
}
------------------------------------------------------------------
```
Here we create the mutator method that will update the private variable ‘seatsAvailable’

```
------------------------------------------------------------------
School.java
------------------------------------------------------------------
public class School {
        public static void main(String args[]) {
                Mathematics mathClassroom = new Mathematics(32);

                mathClassroom.setSeatsAvailable(45);
                mathClassroom.getSeatsAvailable();
        }
}
------------------------------------------------------------------
OUTPUT
------------------------------------------------------------------
The number of seats available in the Mathematics classroom is 45
```

We created the object mathClassroom with 32 initial seats available and then we ‘overwrite’ the value by using the ‘setSeatsAvailable’ mutator method into 45 available seats.

Multiple use of an outside class

In all the examples above we have learned how a class is created, its elements and how to define an object, just one! In order to understand the power of the classes and the objects then we will create more objects in a class by making use of the diagram used above.

First example will use 2 classes, the Mathematics Class and the School Class. The School class will make use of the Mathematics class more than just once.

```
--------------------------------------------------------------------------------
Mathematics.java
--------------------------------------------------------------------------------
public class Mathematics {
        // Part 1
        private int seatNumber;
        private String personName;
        private int penNumber;
        private boolean lateFlag;

        // Part 2
        public Mathematics(String name, int seat, int pen, boolean late) {
                personName = name;
                seatNumber = seat;
                penNumber = pen;
                lateFlag = late;
        }

        // Part 3
        public void studentInfo()
        {
                System.out.println(“This is ” + personName);
                System.out.println(“His/Her seat number is: ” + seatNumber);
                System.out.println(“He/She got: ” + penNumber + “ pens”);
```

```
		System.out.println(“It is ” + lateFlag + “ that he/she is late”);
	}
}
```

In our example there are 3 parts:
The first part defines 4 instance variables that will retain the student information.
In the second part we have a constructor method that will be the middle between the School class and the Mathematics class varible which will assign whatever value you set in the School class to each instance variable in the Mathematics Class.
The third part has a method of displaying the information about each student. Once a student is created in the School class we can call this method that will give us some details about his/her mathematics class.

School.java

```
public class School {
	public static void main(String args[]) {
		// Create them
		Mathematics person1= new Mathematics(“John”, 33, 2, true);
		Mathematics person2= new Mathematics(“Mark”, 24, 1, false);
		Mathematics person3= new Mathematics(“Felix”, 12, 5, true);
		Mathematics person4= new Mathematics(“Jess”, 2, 3, false);

		// Introduce them
		person1.studentInfo();
		person2.studentInfo();
		person3.studentInfo();
		person4.studentInfo();
	}
}
```

Above we have the School class in which we have created 4 persons which are students in the school and each of them belong in the mathematics classroom. Each of them has a name, a seat number, a number of pens and some are late some are not.

If we run this School class the output will be:

```
This is John
His/Her seat number is: 33
He/She got: 2 pens
It is true that he/she is late
This is Mark
His/Her seat number is: 24
He/She got: 1 pens
It is false that he/she is late
This is Felix
His/Her seat number is: 12
He/She got: 5 pens
It is true that he/she is late
This is Jess
His/Her seat number is: 2
He/She got: 3 pens
It is false that he/she is late
```

Basically we make use of the mathematics class to create 4 people with different specs each. They are all taking the mathematics course and they are all created using a constructor method that receives each person's specs and assign them to the instance variables.

Remember!

The constructor is able to do the same thing as the mutator method but the reason we use the method is because it is easy to change the value afterwards. If we use a constructor to assign values then we want these values to be changed then we can make use of mutator methods to change the values.

Also pay attention at the fact that the ‘studentInfo’ method is just an accessor method since it just takes the values passed in the School Class and retrieve them in sentences in order to have a meaning.

Modifier Types
After finishing with the basics of classes and objects we can move on with understanding certain keywords that appear in the text which were not explained before.

There are 2 types of modifiers in Java: Access and Non Access Modifiers.

Access Modifiers
Public
A class, method, variable, constructor, etc. that is declared as being public will be able to be accessed from any other class.

For example, the main method has to be public all the time since it is the most important component of a program that has to be accessed by the other classes interfaces, etc.

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

Important!
No keyword in front of the component refers to the fact that they are by default declared as being public.

Private

When we put the ‘private’ keyword in front of a method, variable etc. then that method, variable or component can only be accessed within the class it is defined.

Important!
Interfaces and classes cannot be private.
The only way we can access components outside the class they are defined is by using the accessor and mutator methods which are declared as being public.

Example from above:

```
public class Mathematics {
	private int seatsAvailable;

	public Mathematics(int seats) {
		seatsAvailable = seats;
	}

	public void setSeatsAvailable(int seatsNumber) {
		seatsAvailable = seatsNumber;
	}

	public void getSeatsAvailable()
	{
		System.out.println(“The number of seats available in the
Mathematics classroom is ” + seatsAvailable);
	}
}
```

In the code above we can see that the instance variable is declared as being private. The getters and setters are declared public so the outside class can get access to the variable by using the getters and setters.

Practice: Make a complex student system for different classrooms

Inner classes

After learning about how to create a class, make use of it and create different objects we will look at how to have multiple classes inside one class and how to make instances of them.

We will use the School example above.

In the example above we were having a class for each classroom and all the classes(Mathematics, Biology, etc.) were linked in the School class which was the main class that has them all.
Now we will use the School Class as an outer-class and all the other classes that were linked to it will be inner classes. Check the example below:

```
public class School{
        public class Mathematics{
                public void getName(){
                System.out.println(“This is the Mathematics class”);
        }
        }
        public class Biology{
                public void getName(){

                System.out.println(“This is the Biology class”);
        }
        }
        public class Spanish{
                public void getName(){
                System.out.println(“This is the Spanish class”);
        }
        }
        public class Physics{
                public void getName(){
                System.out.println("This is the Physics class");
        }
        }
}
```

Here we can see that there is the School Class that has all the other Classes inside it, each having a print statement that specifies the class name.

Now let's create a new class called 'MyClassroom' that will create instances of what classes a student is attending.

```
public class MyClassroom{
        public static void main(String args[]){
                School school = new School();

                School.Spanish spanish = school.new Spanish();
                spanish.getName();
}
}
```

Here we make an instance of the outer class which is School Class and then we make an instance of the inner class which is Spanish Class. The Spanish Class has a getter which prints the class name. We use the instance to call the getter afterwards.

OUTPUT

```
This is the Spanish class
```

Method Local Inner Class

Beside having all the classes in one class we can also have classes inside methods. For the example above we grouped all the course classes into one class called School.

In school we can have a method for example called totalCourses that will retrieve the number of classes that are in the School. For our example above there are 4 classes. Let’s see:

```
public class School {
        public void totalCourses() {
                public class Print{
                        public void printTotalNumber() {
                                System.out.println(“4 Classes”);
                        }
                }
        }

        public class Mathematics{
                …..
        }

        public class Biology{
                …..
        }
        …...
        Print printer = new Printer();
        printer.printTotalNumber();
}
```

As you see here we have the outer class ‘School’, then inside is a method called ‘totalCourses’. In the ‘totalCourses’ method there is a class created called ‘Print’ with a method called printTotalNumber that prints the number of classes inside the School Class.

In order to make it more simple to execute the totalCourses method, we make an instance of the Print class before closing the outer class bracket

Now let’s take a look at the main method inside our MyClassroom class:

```
public class MyClassroom{
        public static void main(String args[]){
                School school = new School();
                school.totalCourses();
        }
}
```

In the main method we only make an instance of the outer class and then we call the method and the final **output** will be:

```
4 Classes
```

Static Nested Class

The difference with the static nested class is the fact that you do not need to make an instance of the outer class. Let’s take the examples from the inner class beginning.

```
public class MyClassroom{
        public static void main(String args[]){
                School school = new School();

                School.Spanish spanish = school.new Spanish();
                spanish.getName();
}
}
```

Here is the example when we made an instance of the outer class 'School'. If we adapt it for the static nested class then we get:

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

        School.Spanish classroom = new School.Spanish();
        classroom.getName();
        }
}
```

Inheritance

Inheritance is the process in which a class acquires the properties(methods, etc.) of another class.

The class which inherits the properties is called **Subclass** and the class that has the properties inherited is called **Superclass**.

Let's say we have two classrooms: Mathematics and Biology. We want to have the same seats in the Biology classroom as there are in the Mathematics class so we need to inherit the properties of the seats(Brand, color, size, etc.). How do we do that?

```
public class Mathematics{}
public class Biology{}
```

extends Keyword

In order to inherit the properties of the Mathematics class we use the following syntax:

public class Mathematics{}
public class Biology **extends** Mathematics{}

PRACTICE

You are going to have a quizz to test your understanding so far! Good Luck!

1. Select the right definition of a class called “Human”:
    a. public class main Humans
    b. class main Human
    c. public class Humans
    d. public static Human
    e. public class Human

2. What is a class for object-creating?
    a. A template for creating another class
    b. A class is a template for creating classrooms for the school
    c. A class is an object used in school
    d. A template for creating different objects which defines its properties and behaviors
    e. A blueprint for creating Mathematics class

3. Complete the following constructor method:

```
-----------------------------------------------------------------
Phone.java
------------------------------------------------------------------
public class Phone {
        int megapixels;
        String phoneName;

        public Phone(int mPixels, String pName) {
                ……?……...
        }
}
```
----

a. Megapixels = mPixels;
   phoneName = pName;

b. mPixels = megapixels;
   pName = mPixels;

c. pName = mPixels;
   Megapixels = pName;

d. megapixels = mPixels;
   phoneName = pName;

e. megapixels = pName;
   PhoneName = pName;

4. Create an object called tea that will output:

```
        It takes 3 minutes to make a tea
        I like the green tea.
```

Tea.java

```
public class TeaBags {
        int minutesToMake;
        String teaType;

        public TeaBags(int min, String type) {
                minutesToMake = min;
                teaType = type;
        }

        public void info() {
          System.out.println(“It takes “ + minutesToMake + “ minutes to make
a tea”);
          System.out.println(“I like the “ + teaType + “ tea.”);
         }
}
```

----

a.

b. mPixels = megapixels;
   pName = mPixels;

c. pName = mPixels;
   Megapixels = pName;

d. megapixels = mPixels;
   phoneName = pName;

e. megapixels = pName;
   PhoneName = pName;