LAB ASSIGNMENT I

PART A

Instructions

- Part A of this assignment must be done with ***gedit*** text editor and use the terminal to type in the commands for compiling and executing the java program.
- Part B of this assignment must be done with ***eclipse IDE***, go through the ***eclipse IDE*** *handout* for getting yourself familiarized with the IDE.

Getting started with Java

Let's start by compiling and running the short sample program shown here. As you will see, this involves a little more work than you might imagine.

```
/*
      This is a simple Java program.
      Call this file "Example.java".
*/
class Example
{
      // Your program begins with a call to main().
      public static void main(String [] args)
      {
            System.out.println("This is a simple Java program.");
      }
}
```

Entering the Program

- For most computer languages, the name of the file that holds the source code to a program is immaterial. However, this is not the case with Java.
- The first thing that you must learn about Java is that the name you give to a source file is very important. For this example, the name of the source file should be **Example.java**.
- In Java, a source file is officially called a *compilation unit*. It is a text file written with the ***gedit*** text editor and that contains (among other things) one or more class definitions. (For now, we will be using source files that contain only one class.)
- The Java compiler requires that a source file use the **.java** filename extension.
- As you can see by looking at the program, the name of the class defined by the program is also **Example**. This is not a coincidence. In Java, all code must reside inside a class. By convention, the name of the main class should match the name of the file that holds the program.
- You should also make sure that the capitalization of the filename matches the class name. The reason for this is that Java is case-sensitive.

Compiling the Program

- To **compile** the file, open your terminal and type

  javac Example.java
- The ***javac*** compiler creates a file called **Example.class** that contains the bytecode version of the program.
- The **Java bytecode** is the intermediate representation of your program that contains instructions the Java Virtual Machine will execute.
- Thus, the output of ***javac*** is not code that can be directly executed.
- To actually run the program, you must use the Java application launcher called ***java***.
- To do so, pass the class name **Example** as a command-line argument, as shown here

  java Example

Some Facts

- When Java source code is compiled, each individual class is put into its own output file named after the class and using the **.class** extension.
- This is why it is a good idea to give your Java source files the same name as the class they contain—the name of the source file will match the name of the **.class** file.
- When you execute **java** as just shown, you are actually specifying the name of the class that you want to execute. It will automatically search for a file by that name that has the **.class** extension. If it finds the file, it will execute the code contained in the specified class.

PART B

Object Oriented Programming Basics

Class and Object are two main building blocks of OO that encapsulate attributes (fields) and operations (methods) on them. A class can be likened to a data type (although it is more than that) while the object can be likened to variable of that type. We will explain with a simple example.

Step 1: Defining a class

Let's define a class for representing fractions which has 2 attributes: numerator and denominator.

```
 --------------Fraction.java --------------

public class Fraction {
  private int numerator;
  private int denominator;
}
-----------------------------------------
```

- Now object of Fraction class can be created using the following syntax.

```
Fraction f = new Fraction();  // meaning f is an instance of Fraction
```

- The keyword '`public`' implies Fraction class can be accessed from outside.
- The keyword '`private`' for attributes imply they can't be accessed from 'outside'.

  I.e. one cannot access them by *f.numerator=2* or *f.denominator=3*. Try doing this.

- '`public`', '`protected`' and '`private`' are referred to as **access specifiers**. They are used to safeguard the attributes from improper direct access.
- How to assign values for numerator and denominator?
  - By defining methods that assign them

Step 2: Defining setters - Assigning values to attributes

Let's define two methods to set the values of the attributes (setters)

```
--------------Fraction.java --------------

public class Fraction {
  private int numerator;
  private int denominator;

  public void setNumerator(int n) {
    this.numerator = n;
  }

  public void setDenominator(int d) {
    this.denominator = d;
  }
}
-----------------------------------------
```

- Now one can set the values for the attributes by using these setter methods.

```
Fraction f = new Fraction();
      f.setNumerator(2);
      f.setDenominator(3);
```

- Alternately one may have a single setter method that takes two integer arguments and assign them to the attributes.
- There are no hard and fast rules to number of setter methods. It depends on the requirement.
- The next question is how to access (or read) the values of these attributes.

Step 3: Defining getters - Reading values of attributes

- Getters are methods defined to return the values of the attributes. We define two methods
  `getNumerator()` and `getDenominator()` towards this end.

```
--------------Fraction.java --------------

public class Fraction {
  private int numerator;
  private int denominator;

  public void setNumerator(int n) {
    this.numerator = n;
  }

  public void setDenominator(int d) {
    this.denominator = d;
  }

  public int getNumerator() {
    return numerator;
  }

  public int getDenominator() {
    return denominator;
  }
}
-----------------------------------------
```

- Now one can read the values and print them as follows.

```
Fraction f = new Fraction();
f.setNumerator(2);
f.setDenominator(3);
System.out.println("The fraction is " + f.getNumerator() + "/" +
f.getDenominator());
```

- Typically we define a separate "driver" class to instantiate Fraction, set, read and print them. This is to separate the definition of a class from its usage.

- Driver class code is given below.

```
--------------Driver.java--------------

public class Driver {
  public static void main(String[] args) {
    Fraction f = new Fraction();
    f.setNumerator(2);
    f.setDenominator(3);
    System.out.println("The fraction is " + f.getNumerator() + "/" +
f.getDenominator());
  }
}
```

Compiling and execution

- Refer the handout to execute in ***Eclipse IDE***
- Only the class that contains `main()` can be executed. In this case "`Driver`".
- You may also implement a separate method `print()` in Fraction to print the attribute values.

```
public void print() {
        System.out.println("The fraction is " + this.numerator + "/" +
  this.denominator);
}
```

- Hence, in the Driver class you may call this method to print the fraction.

```
Fraction f = new Fraction();
   f.setNumerator(2);
   f.setDenominator(3);
   f.print();
```