جامعة بيرزيت

BIRZEIT UNIVERSITY

Computer Science Department

Laboratory Workbook

Comp231

Advanced Programming

Prepared by:

Approved by:

Computer Science Department

2017/2018

Introduction

The aim of this lab manual is to help students of the “advanced programming” course to understand and apply a variety of object oriented programming and design concepts. Every lab session is provided with lab objectives, a brief discussion about the experiment’s topics or concepts that strength the student understanding to the lab material; a Java language syntax for the commands or statements that will be used; and some exercises that allow the student to completely understand the topic. The exercises in this manual are carefully prepared, studied and revised by several members of the academic staff in the computer science department at Birzeit University.

The lab manual starts explaining Object Oriented Programming using Java language from scratch. It allows the students to learn about the structure of the Java programs and provides students with sufficient knowledge to understand object-oriented programming concepts using Java language. Further, the manual covers all object-oriented programming concepts such as the use of objects and classes, inheritance, polymorphism, abstract classes and interfaces. In addition, students will learn how to make a graphical user interface programs and event driven programming. Finally, this manual gives students a complete understanding to the above topics in which they will be able to proceed with the advanced courses during their study.

Table of Contents

Lab 1: Program Structure in Java

Objectives

- To understand the layout and the structure of a simple Java program.
- To be able to create, compile and run a simple Java program using command line (cmd).
- To apply **Scanner** and **Loops**

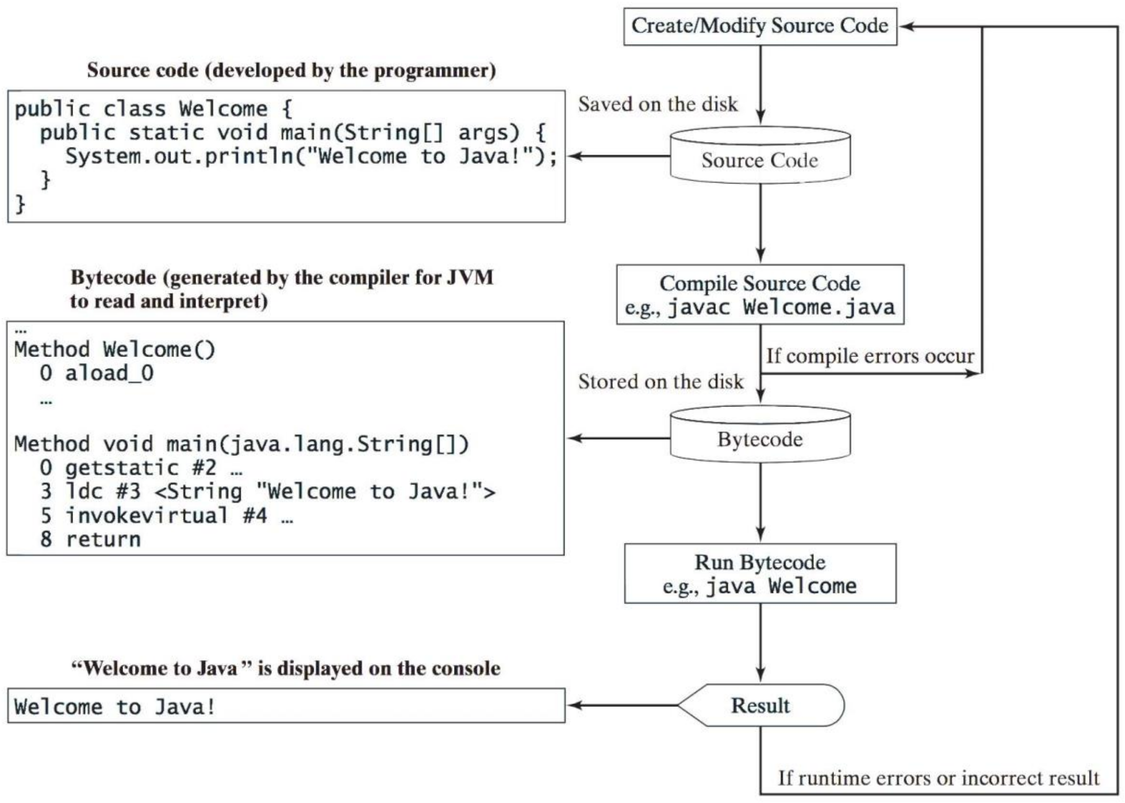
Simple Java Program

This program will allow you to understand the layout to Java Programs:

```
// Text printing program - First lab
public class Welcome {
      // main method begins execution of Java application
      public static void main(String[] args) {
            for (int i = 0; i < 3; i++)
                  System.out.println("Welcome to comp231");
      } // end method
} // end class Welcome
```

Compiling, and Executing a Java Program

Sun releases each version of J2SE with a **Java Development Toolkit** (JDK).


Create/Modify Source Code
Source code (developed by the programmer)
Saved on the disk
Source Code
Bytecode (generated by the compiler for JVM to read and interpret)
Compile Source Code e.g., javac Welcome.java
If compile errors occur
Stored on the disk
Bytecode
Run Bytecode e.g., java Welcome
"Welcome to Java" is displayed on the console
Result
If runtime errors or incorrect result

```
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

```
…
Method Welcome()
  0 aload_0
  …

Method void main(java.lang.String[])
  0 getstatic #2 …
  3 ldc #3 <String "Welcome to Java!">
  5 invokevirtual #4 …
  8 return
```

```
Welcome to Java!
```

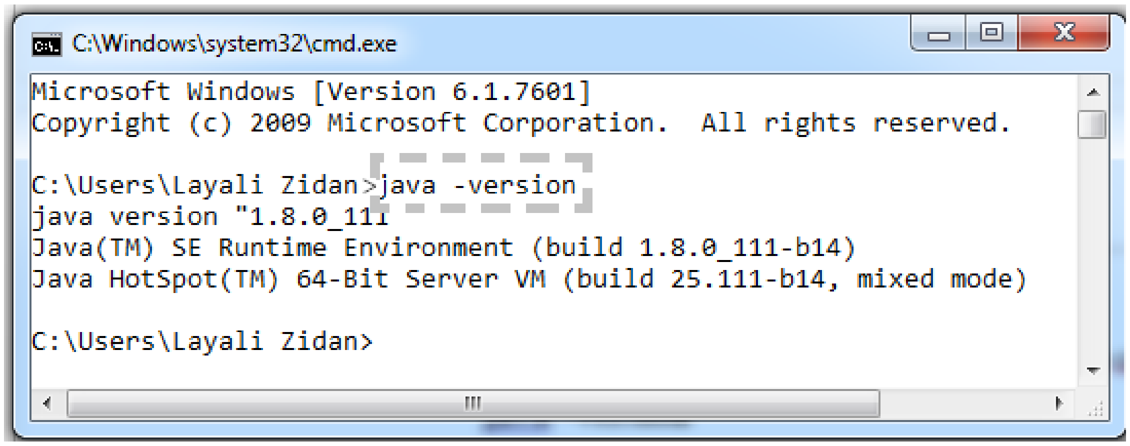
Figure1.1 (process diagram)

To check that you have installed the JDK, run command line interface (cmd). For example, in windows 10 you can use Cortana to search for and lunch the command prompt, Inside Cortana's search field, enter command or cmd. Then click or tab on he command prompt result.

Then type the following command:

java –version

This command will show you the version of the JDK you have installed (Figure1.2),:.



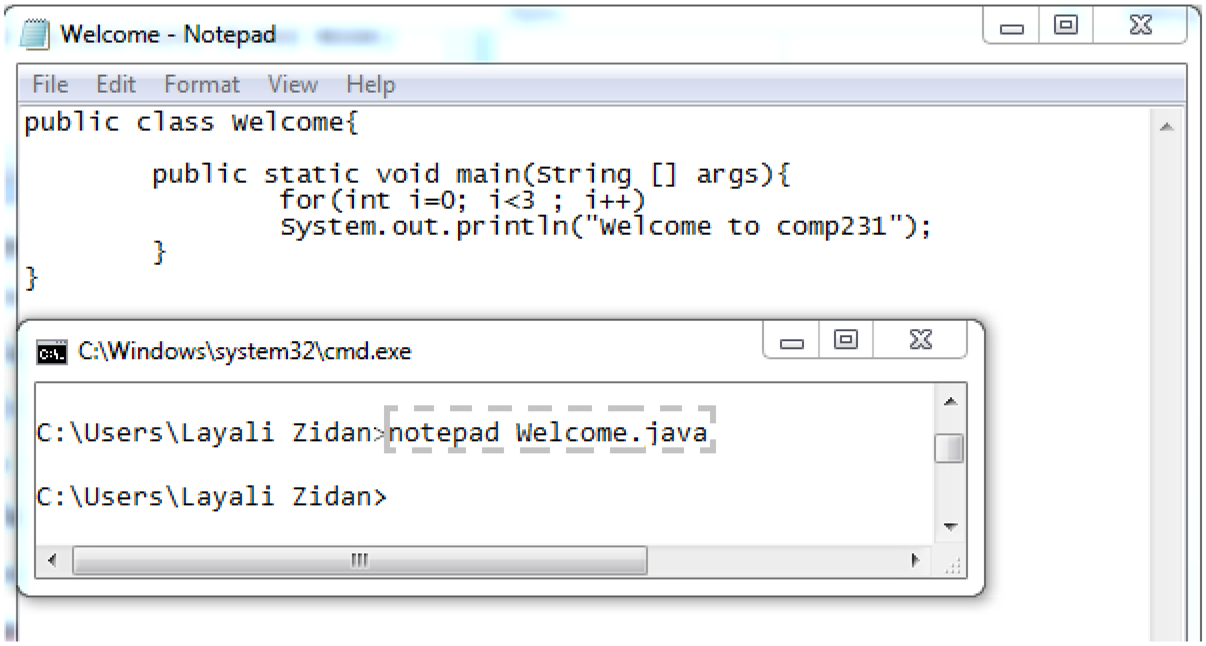
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Layali Zidan>java -version
java version "1.8.0_111
Java(TM) SE Runtime Environment (build 1.8.0_111-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.111-b14, mixed mode)

C:\Users\Layali Zidan>
```

Figure1.2 (java version)

To run a java program using cmd we use the following steps:

1- Open *Notepad* and name it as ClassName.java using the following command, then write your source code inside it (figure1.3) :

Notepad className.java



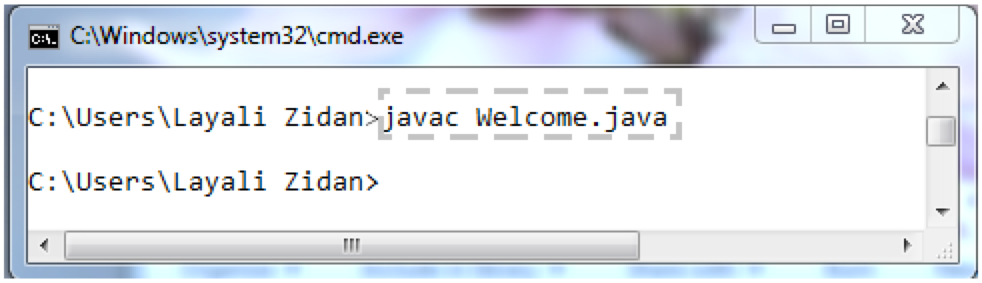
```
public class Welcome{

        public static void main(String [] args){
                for(int i=0; i<3 ; i++)
                System.out.println("Welcome to comp231");
        }
}
```

```
C:\Users\Layali Zidan>notepad Welcome.java

C:\Users\Layali Zidan>
```

Figure1.3 (open notepad file)

2- After completing your source code, open cmd (run → cmd) then you have to compile it using the following command (figure 1.4):

3- **javac ClassName.java**

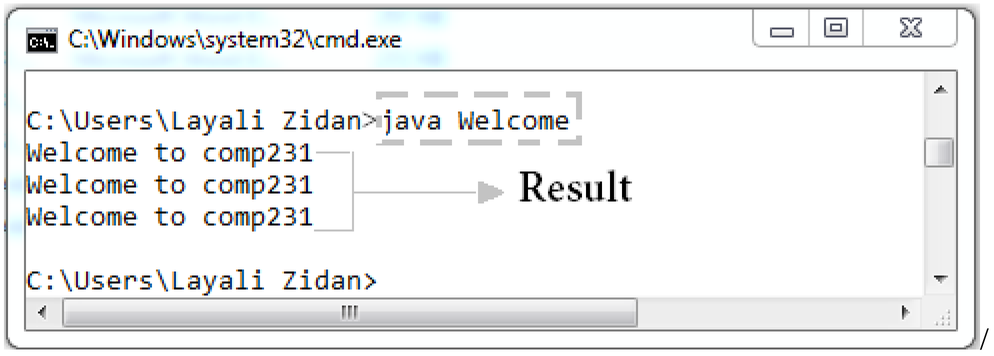


```
C:\Users\Layali Zidan>javac Welcome.java

C:\Users\Layali Zidan>
```

Figure1.4 (compiling source code)

4- If no compilation errors appear, this means that you can run your program using the following command (figure 1.5):

java ClassName



```
C:\Users\Layali Zidan>java Welcome
Welcome to comp231
Welcome to comp231
Welcome to comp231

C:\Users\Layali Zidan>
```

Figure1.5 (Running byte code)

<u>Read from consol:</u>

Console input is not directly supported in Java, but you can use the **Scanner** class to create an object to read input from **System.in**, as follows:

```
Scanner input = new Scanner(System.in);
```

The syntax **new Scanner (System.in)** creates an object of the **Scanner** type. The syntax **Scanner input** declares that **input** is a variable whose type is **Scanner**. The whole line

Scanner input = new Scanner (System.in)

creates a **Scanner** object and assigns its reference to the variable **input**.

An object has methods that can be invoked as functions in c programming language. When someone invokes an object's method, he/she is asking the object to perform a certain task. For example, you can invoke the **nextDouble()** method to read a **double** value as follows:

```
double radius = input.nextDouble();
```

`Scanner methods that the Scanner object can use:`

boolean hasNext()	This method returns true if this scanner has another token in its input.
String next()	This method finds and returns the next complete token from this scanner.
double nextDouble()	This method scans the next token of input as double.
int nextInt()	This method scans the next token of input as an int.
String nextLine()	This methods advances this scanner past the current line and returns the input that was skipped.

<u>Exercises</u>

1. Write a java application using **notepad** that prints your name, id, course name (Which is "advanced programming comp231") and your instructor's name with the section number next to it. Then from console use JDK to compile and run the program.

2. Use notepad to write a java application that asks the user to enter 10 numbers and calculates their average then displays the result. From console use JDK to compile and run the program.

Lab 2: Structured Programming - Revision

Objectives

- To introduce Eclipse debugger - an important tool for finding and fixing Run-time errors.
- To be able to write simple Java application using output and input streams.
- To become familiar with primitive data types.
- To become familiar with the conditional statements in Java (i.e. **if, if else, switch**).
- To become familiar with the control structures in Java (i.e. **while, for, do while**).
- To create methods, invoke methods and pass arguments to a method.
- To determine the scope of variables.

Conditional Statements in Java

1. **Single Condition**

```
if (expression ) {
   statement(s);
}
```

Multiple conditions:

```
if (expression ) {
    statement(s);
}
else {
    statement(s);
}
```

2. Multiple Selections

```
switch (expression) {
 case constant-expression:
      Statement(s);
      break;
 default:
      Statement(s);
      break;
}
```

Control Structures (Loops)

1. ***while*** **loop:**

```
while (expression) {
      Statement(s);
}
```

2. ***do-while*** **loop:**

```
do {
     Statement(s);
} while (expression);
```

3. ***for*** loop :

```
for (initial-action; expression ;action-after-each-iteration) {
     Statement(s);
}
```

Eclipse IDE

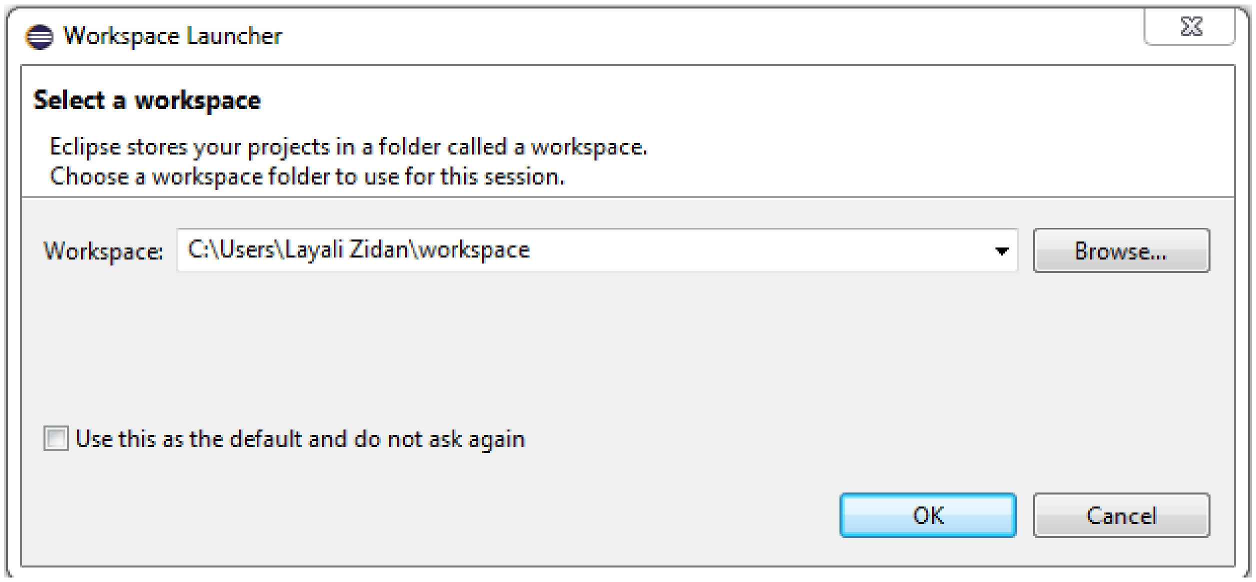
This lab will introduce you to Eclipse, a full-featured and very versatile Integrated Development Environment(IDE). During the assignments and labs in this course you will be using Eclipse extensively to develop Java programs.

You can download it from the following link:

https://eclipse.org/mars/

Running the Program

1- To open Eclipse, double click its application icon, then chose the workspace (folder) as in figure2.1. All your work will be saved in this folder, so make sure that you chose it and remember where you have saved it (the path).



Workspace Launcher

Select a workspace

Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Users\Layali Zidan\workspace

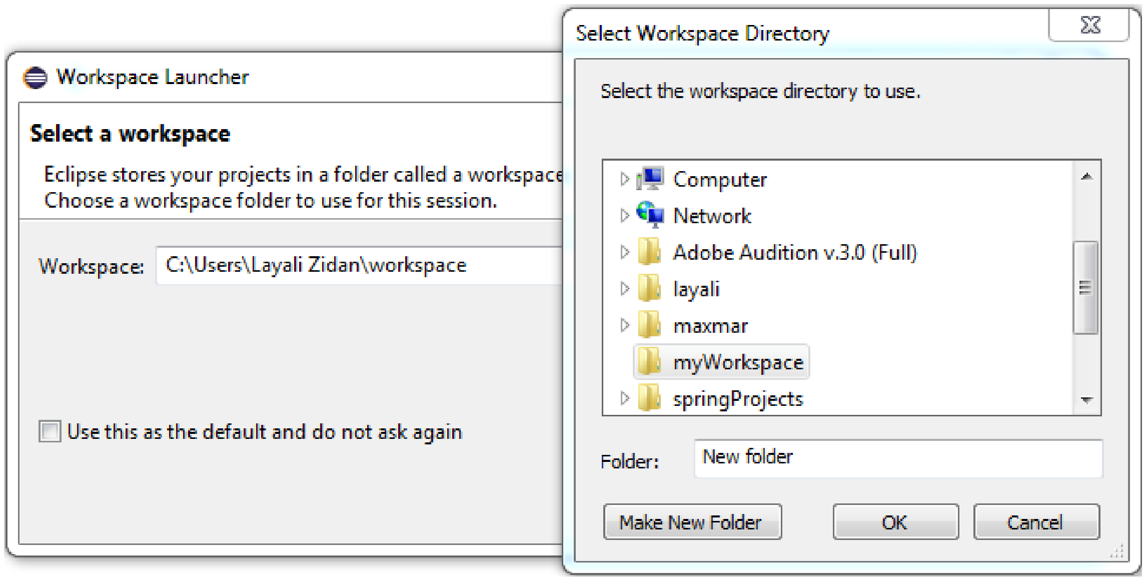
Browse...

☐ Use this as the default and do not ask again

OK Cancel

Figure2.1 (workspace)

If you want to change it, click brows and select new place for it as in figure 2.2. The new place that we have chosen is myWorkspace, then click ok.


Workspace Launcher

Select a workspace

Eclipse stores your projects in a folder called a workspace
Choose a workspace folder to use for this session.

Workspace: C:\Users\Layali Zidan\workspace

☐ Use this as the default and do not ask again

Select Workspace Directory

Select the workspace directory to use.

Folder: New folder

Figure2.2 (changing workspace)

2- Now you can create your first project by going to File $\longrightarrow$ New $\longrightarrow$ Java project. Then name your project as HelloWorld and click finish.

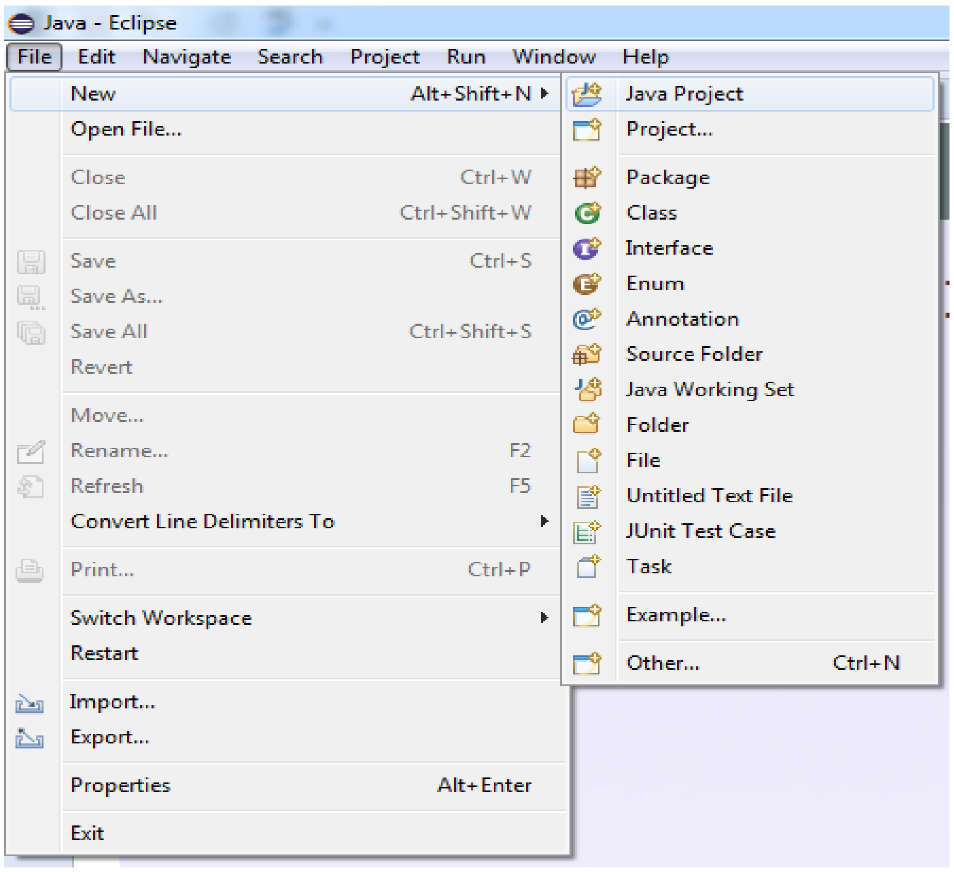
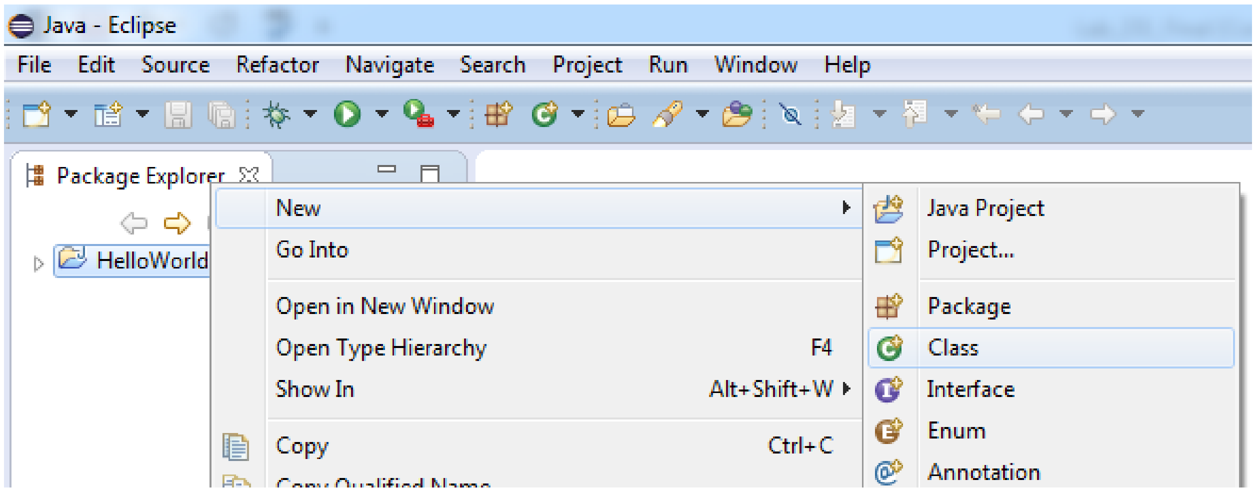


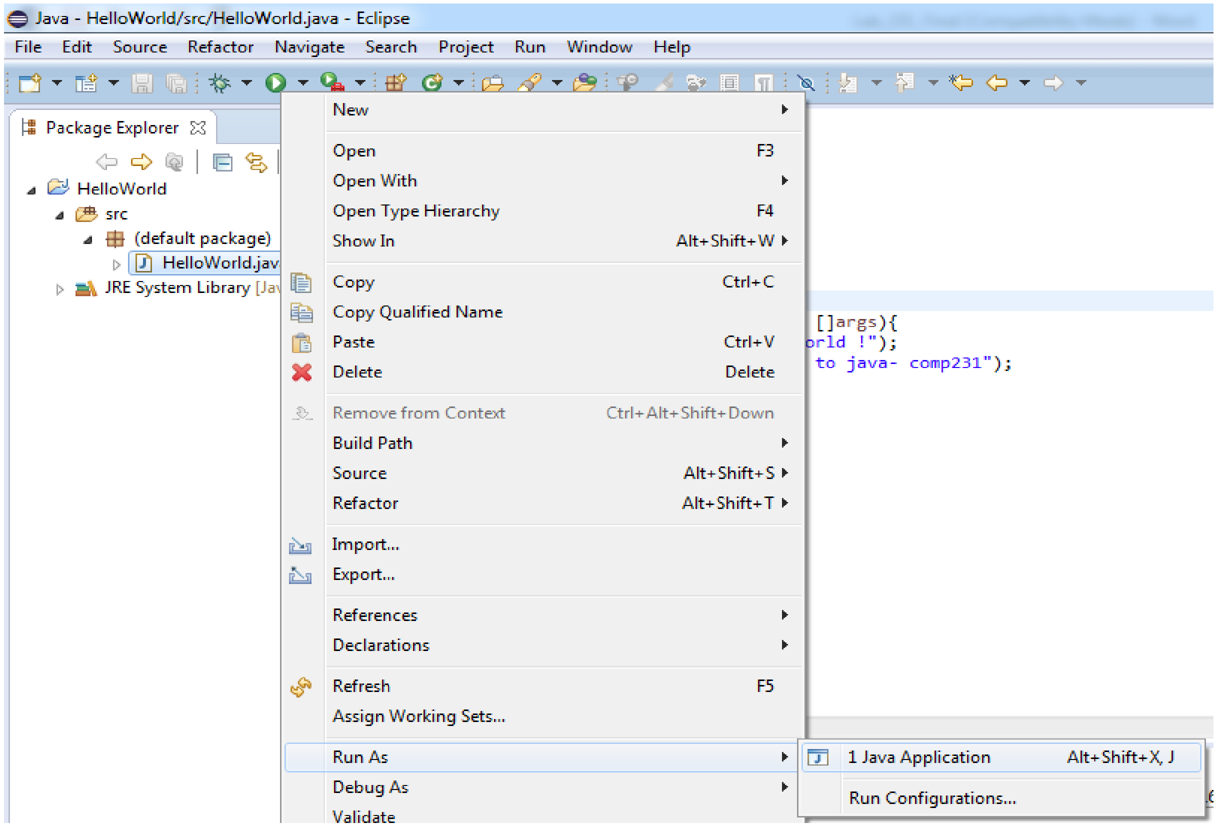
Figure2.3 (new java project)

3- To create a new class, right click on your project name (HelloWorld), then New ⟶Class



4- After creating your class, you are ready to write your code inside it. Later on, to run your program, right-click on your class that contains the main method

Choose **run** then choose **java application** as in figure 2.4



```
[]args){
orld !");
to java- comp231");
```

Figure2.4(run java application)

5- The results of your program will appear in the console.

Methods:

Syntax

[*modifiers*] data-type ***method-name*** ([*parameter-declaration*]) **{…}**

Calling a Method

When creating a method, you give a definition of what the method is supposed to do. To use a method, you have to call or invoke it.

There are two ways to call a method; the choice is based on whether the method returns a value or not:

1. If the method returns a value, a call to the method is usually treated as a value. For example:

   data-type variable-name = **method-name**(passed-arguments);

   Calls **method-name** (passed-arguments) and assigns the result of the method to the variable variable-name.

   Another example of a call that is treated as a return-value is:

   ```
   System.out.println(method-name(passed-arguments));
   ```

   Which prints the return value of the method call **method-name**(passed-arguments)

2. If the method returns void it should be called using a statement. For example, the method **println** returns void. The following call is a statement:

   ```
   System.out.println("Welcome to Java!");
   ```

Exercises

1. The body mass index (BMI) is a ratio of person's weight and height. The index can be used to determine if a weight is unhealthy for certain height. Here is the formula:

$$BMI = \frac{weight(pounds) \times 703}{height^2(inches)}$$

Write a java program that reads values for weight in **kilograms** and height in **centimetres** and prints out the BMI.

Hence, the above formula uses **pounds** and **inches**, so you have to include the following two methods to do the needed conversion.

- Converts from Kilograms to Pounds:

$$pounds = kilogram \times 2.2$$

```
public static double kilogramToPound(double kilogram)
```

- Converts from Centimetres to Inches:

$$inches = centimeter \times 0.39$$

```
public static double centimetresToInches(double centimetres)
```

Your program should also print on screen an interpretation of the BMI. Use the following scale.

BMI	Interpretation
15 <= BMI < 19	Underweight
19 <=BMI < 25	Normal
25<= BMI <30	Overweight

2. Write a java application that will determine the salary for each of several employees. The company pays "straight-time" for the first 40 hours, if the employee worked more than 40 hours, we consider the extra hours as overtime and the employee gets 50% extra for each overtime hour. You are given a list of employees of the company, each employee has a number of worked hours and hourly rate.

Your application should input this information (worked hours and hourly rate) for each employee then it should determine and display the employee's salary.

The program will stop if we enter -1.

Output may appear as follow:

```
Enter hours worked (-1 to end): 39
Enter hourly rate of the worker ($00.00):10.00
Salary is $390.0

Enter hours worked (-1 to end): 41
Enter hourly rate of the worker ($00.00):10.00
Salary is $415.0

Enter hours worked (-1 to end): -1
```

3. A prime number is a number that has no positive divisors other than 1 and itself. Your program should stop if we entered a number less than 2. Write a java program that finds the prime numbers between 2 and 500 using the following method signature, your program will stop if the user enter a number less than 2

```
public static boolean isPrime( int number)
```

Now improve your program to allow the user to enter a number then check if it is prime or not.

Lab 3: Methods

Objectives

- To practice more about methods (create, invoke and pass arguments).
- To determine the scope of variables.

Exercises

1. The exponential ***e^x*** can be calculated using the following formula:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!}$$

The factorial of non-negative integer n is written n! And is defined as follows:

$$n! = n \cdot (n-1).(n-2).\cdots.1$$

$$n! = 1 \ (if \ \ n = 0)$$

Write a java program that contains two methods ***factorial*** and ***exp according to the formulas that we mentioned above.*** The main task for this program is to ask the user to enter a number and calculate its exponential then display the result.

- Note that **exp** method will call **factorial** method.

2. An integer number is said to be *perfect number* if the sum of its factors including 1 (but not the number itself) is equal to the number itself.
For example 6 is perfect because 6 = 1 + 2 + 3.

Write a method ***isPerfectNumber*** using the following signature that determines if parameter number is perfect number. Use this method in a main method that determine and prints all the perfect numbers between 1 and 1000.

```
public boolean isPerfectNumber( int number)
```

3. Write a java method that accepts a binary number and converts it to decimal then display the result. For Example:
$(110)_2 = (6)_{10}$
$(2^2 *1)+ (2^1 *1) + (2^0*0) = 6$
Additional task: write a method that accepts a decimal and converts it to binary.

Lab 4: Arrays and Object Use

Objectives

- To introduce the concept of arrays in Java.
- To understand how to declare an array, initialize it and refer to individual elements inside the array.
- To understand how to pass an array of elements to a method.
- To understand the concept of array of ***Objects***.

Syntax

To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference.

dataType[ ] arrayName = new dataType[sizeOfAnArray];

dataType arrayName[ ] = new dataType[sizeOfAnArray];

Passing an array to a method

Arrays are objects therefore they are passed to the method **by reference**.
To pass an array to a method as a parameter you have to write the following:

modifiers data-type ***method-name*** (dataType [ ]arrayName) **{…}**

Exercises

1. A building with ***n*** apartments is to be represented in an array.
   - Write an application that reads the number of apartments and use it as the size of the array. The apartment number will serve as the index of the array.

     The values in the array represent the number of people who live in the apartment.

     For example: a building which has 4 apartments can be represented as follows :

     Building:

2	**3**
4	**2**
6	**1**
2	**0**

Now, create a building (array on integers) as above then print the following information:

1- Number of people in the building.

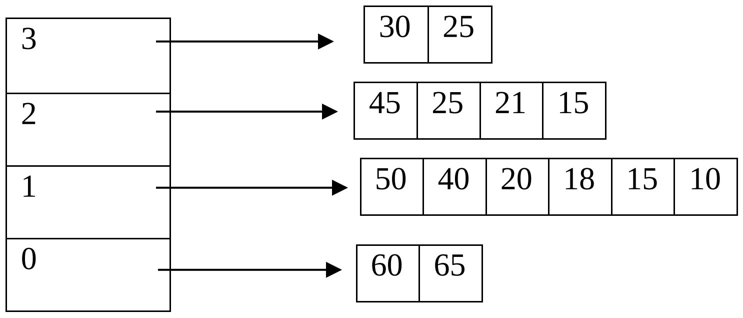
2- The average number of people per apartment

3- The number of apartments with above-average occupancy, and the number with below-average occupancy.

- Modify the above application to read the number of people in the apartment and their ages. Still, the apartment number serves as an index into an array of apartments. But, the values of the apartment represent the ages of people who live in the apartment instead of their number. Add a new method ***averageAges()*** to find the average age of the building residents.

  Hint: use 2 dimensional arrays as in the following figure and build it at run time.

  Building:



2. Write a java program that has a method called linearSearch as in the following signature which accepts an array of integers and an integer number to search for it inside the array. If the number is found in the array, it will return its index, else if the number is not found it will return -1.

```
public static int linearSearch( int [ ] myArray , int number)
```

3. Write an application that creates a one dimensional array of 5 Students. Then write a test program to print the student details in the list.

Tips for solution:

Use the following ***Student*** class to solve the question.

```java
 public class Student {
      private int studentId;
      private String studentName;

      public Student() {
      }

      public Student(int stId, String stName) {
            studentId = stId;
            studentName = stName;
      }

      public void setStudentId(int number) {
            studentId = number;
      }

      public int getStudentId() {
            return studentId;
      }

      public void setStudentName(String name) {
            studentName = name;
      }

      public String getStudentName() {
            return studentName;
      }
}
```

To create an array of 4 Students do the following:

```java
      Student []array = new Student[4];
```

To create a new student, do the following:

```java
      array[index] = new Student(120, "Ali Jumah");
```

Lab 5: Object-Oriented Programming

Objectives

- To understand objects and classes, and the differences between them.
- To use UML (Unified Modeling Language) graphical notations to describe classes and objects.
- To understand and apply the role of constructors.

Unified Modeling Language (UML)

UML is a standardized general-purpose modeling language in the field of software engineering. UML includes a set of graphical notation techniques to create abstract models of specific systems, referred to as UML model.

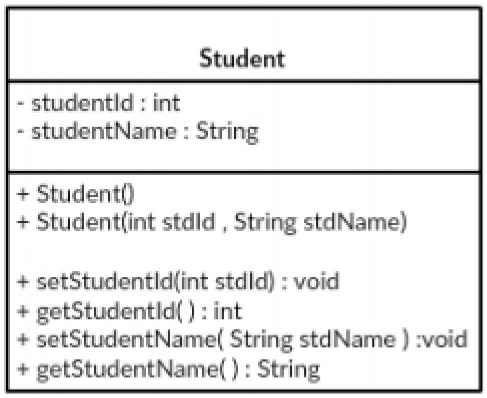
Classes always start with upper case letter while **objects** start with lower case letters. **Data members** and **member methods** start with lower case letters.

Objects are **instances** from classes.

An object has attributes (properties) and behavior.

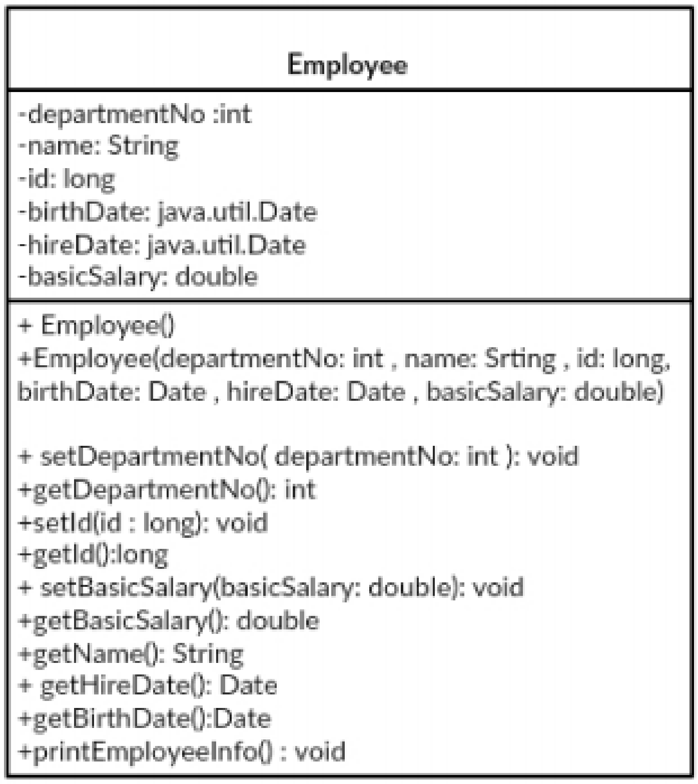
- The **attributes** (states) of an object is represented by data fields (also known as properties).
- The **behavior** of an object is defined by a set of methods.

The following UML represents Student class that we have seen in our previous lab:


Student
- studentId : int
- studentName : String
+ Student()
+ Student(int stdId , String stdName)
+ setStudentId(int stdId) : void
+ getStudentId( ) : int
+ setStudentName(String stdName) :void
+ getStudentName( ) : String

Exercises

1. Create an ***Employee*** class according to the following UML.



```
Employee
-departmentNo :int
-name: String
-id: long
-birthDate: java.util.Date
-hireDate: java.util.Date
-basicSalary: double

+ Employee()
+Employee(departmentNo: int , name: Srting , id: long,
birthDate: Date , hireDate: Date , basicSalary: double)

+ setDepartmentNo( departmentNo: int ): void
+getDepartmentNo(): int
+setId(id : long): void
+getId():long
+ setBasicSalary(basicSalary: double): void
+getBasicSalary(): double
+getName(): String
+ getHireDate(): Date
+getBirthDate():Date
+printEmployeeInfo() : void
```

Write a driver class that creates an array of 4 Employees then passes the array to a method named largestSalary which will return the employee **object** with the largest salary.

```
public static Employee largestSalary ( Employee [ ] employees)
```

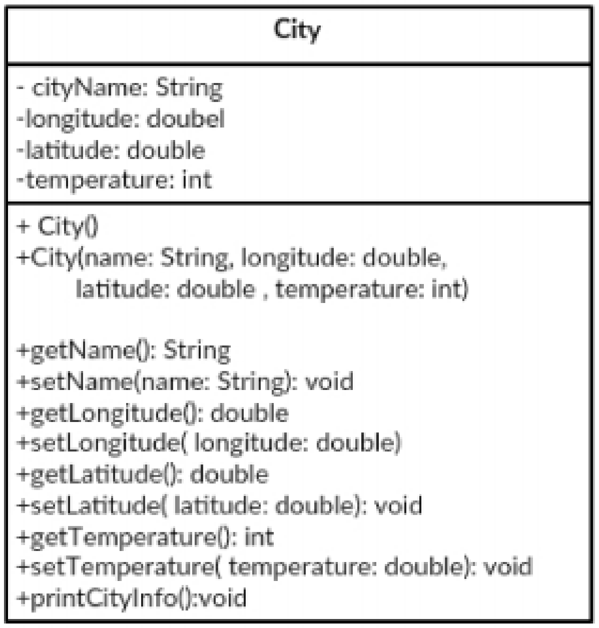
2. Develop **MyArray** class. The internal representation of this class is an array of integers. The following methods should be included in the class to deal with the array:

    MyArray(int []) // Constructor
    min( ): int // returns the minimum element in the array
    max( ): int // returns the maximum element in the array
    average( ):double // returns the average of the elements
    printArray( ):void // prints the elements of the array
    getSize( ):int //returns the length of the array

    Average (***mean)*** is $\bar{x} = \frac{x_{1+}\ x_{12} + \ldots + x_N}{N} = \frac{1}{N}\sum_{i=1}^{N} x_i$

    Develop a driver class to create objects (array) from the created class and test their behaviors.

3. Create **City** class using the following UML:



```
City
- cityName: String
-longitude: doubel
-latitude: double
-temperature: int
+ City()
+City(name: String, longitude: double,
      latitude: double , temperature: int)

+getName(): String
+setName(name: String): void
+getLongitude(): double
+setLongitude( longitude: double)
+getLatitude(): double
+setLatitude( latitude: double): void
+getTemperature(): int
+setTemperature( temperature: double): void
+printCityInfo():void
```

Create a class City according to the above UML class diagram, then create an array of cities which contains 4 city objects. Then pass the array to a method named **belowAverage** that accepts an array of City objects and an integer which represents certain temperature. This method shall print all information about all cities that have a temperature below the given temperature.

```
public static void belowAverage( City [ ] cities , int avgTemp)
```

Lab 6: Strings

Objectives

1. To be able to create and manipulate non-modifiable (immutable) string objects of class ***String***.
2. To be able to create and manipulate modifiable string objects of class ***StringBuffer/ StringBuilder*** class.

Syntax

Strings are Objects in Java and Strings can be declared and created using one of the following:

String stringName = new String("String Litral");

String stringName = new String(char [] arrayOfCharacters);

String stringName = "String Litral";

String class methods:

The following are methods to deal with Strings:

Method	Description
length()	Returns the number of characters in this string.
charAt()	Returns the character at the specific index from this string.
toUpperCase()	Returns a new string with all letters in upper case.
toLowerCase()	Returns a new string with all letters in lowercase.
trim()	Returns a new string with whitespaces characters trimmed on both sides .

Method	Description
equals(s1)	Returns true if this string equal to string s1.
equalsIgnoreCase(s1)	Returns true if this string is equal to string s1; it is case insensitive.
compareTo(s1)	Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than, equal or less than s1.
compareToIgnoreCase(s1)	same as compareTo except that the comparison is case insensitive.
startsWith(prefix)	Returns true if this string starts with specified prefix.
endsWith(suffix)	Returns true if this string ends with specified suffix.
contains(s1)	Returns true if s1 is substring in this string.
index(ch)	Returns the index of the first occurrence of ch in the string. Returns -1 if not matched.
indexOf(ch, fromIndex)	Returns the index of the first occurrence of ch after from Index in the string, Returns -1 if not matched.
indexOf(s)	Returns the index of the first occurrence of string s in this string. Returns -1 if not matched.
indexOf(s, fromIndex)	Returns the index of the first occurrence of string s in this string after from Index. Returns -1 if not matched.
lastIndexOf(ch)	Returns the index of the last occurrence of ch in this string. Returns -1 if not matched.
lastIndexOf(s, fromIndex)	Returns the index of the last occurrence of ch in before fromIndex in this string. Returns -1 if not matched.
lastIndexOf(s)	Returns the index of the last occurrence of string s in this string. Returns -1 if not matched.
lastIndexOf(s, fromIndex)	Returns the index of the last occurrence of string s in this string before fromIndex. Returns -1 if not matched.

<u>Exercises</u>

1. You will create a class named **<u>MyString</u>** that will perform different processing on Strings that are sent to it. All of the methods you create will be static, and the class should work in a similar manner to the Math class. Only the four methods listed below should be public.

1) Create a method ***reverseString*** that receives a String and returns a String that is the exact reversal of the characters in the first String.

2) Create a method ***isPalindrome*** that receives a String and returns a boolean value of true if the String is a Palindrome and false if it is not. A word is a palindrome if it reads the same forwards and backwards. For example, the word level is a palindrome.
The idea of a palindrome can be extended to phrases or sentences if we ignore details like punctuation, so you need to create a method that cleans the sentence from punctuation and spaces, then convert the sentence to lower or upper case after that we can use the method isPalindrome with the new sentence. Here are two familiar examples:
Madam, I'm Adam → *madamimadam*
A man, a plan, a canal: Panama → *amanaplanacanalpanama*

3) Create a method ***ShortHnaded*** that receives a String and returns the String converted into shorthand. The simplified shorthand form of a string is defined as follows:

   1. Replace these four words: "and" with "&", "to" with "2", "you" with "U", and "for" with "4".
   2. Remove all vowels ('a', 'e', 'i', 'o', 'u', whether lowercase or uppercase), do not remove 'u' and 'I' if they did not occurred in a word.

4) Also add the following methods:

1- numberOfSentences: a method that accepts a string and return number of sentences on it. A sentence can be determined if it ends with one of those punctuation (. , ? !).

2- numberOfWords: a methods that accepts a string and return number of words, a word consists of more than 3 characters.

2. Design a java program that we can use to encrypt a sentence by using the following methods respectively:

a. Convert al letters to capital.
b. reverseString: reverse the sentence
c. toNumbers: a method that converts the following letters to numbers:
   "O" to zero (0), "S" to $, "L" to 1
d. beginAndEnd: that add ** at the beginning and at the end of the sentence.

Finally, you should display the encrypted sentence.

Hint: use stringBuilder.

Lab 7: Inheritance and Polymorphism

Objectives

- To understand the concepts of inheritance and polymorphism.
- To be able to develop a subclass from a superclass through inheritance.
- To understand the concepts of dynamic binding, and generic programming.
- To be able to restrict access to data and methods using the *protected* visibility modifier.

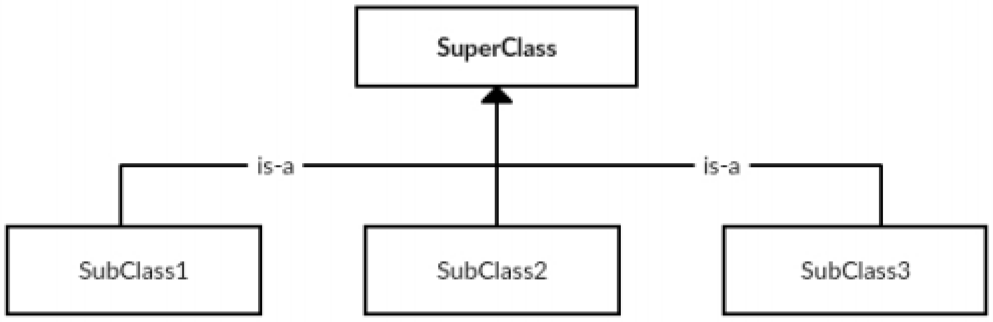
Definition

Inheritance: allows you to derive new classes from existing classes. Inheritance represents the is-a relationship.

Polymorphism: is to behave differently with different subclasses.

As objects do not exist by themselves but are instances of a ***class***, a class can inherit the features of another class and add its own modifications. (This could mean restrictions or additions to its functionality). Inheritance aids in the reuse of code.

Classes can have 'Children' that is, one class can be created out of another class. The original or parent class is known as the ***SuperClass*** (or base class). The child class is known as the ***SubClass*** (or derived class).



A **SubClass** inherits all the *attributes* and *behaviors* of the **SuperClass**, and may have additional attributes and behaviors.

Syntax

In Java inheritance is represented using ***extends*** keyword as follows:

[*modifiers*] super-*class-name* **{ //Data members and methods …}**

[*modifiers*] sub*class-name* ***extends*** super-*class-name {…}*

Exercises

1. Design a class named **Account** that represents a bank account and contains:

- An **int** data field named **id** for the account (default 0).
- A **double** data field named **balance** for the account (default 0).
- A **Date** data field named **dateCreated** that stores the date when the account was created.
- Two constructors:
    1) **No-arg constructor** that creates a default account and initialize the **dateCreated**.
    2) **Constructor with arguments** that accepts id and balance.
- The setter and getter methods for **id and balance.**
- Getter for **dateCreated.**
- A method named **withdraw** that withdraws a specified amount from the account.
- A method named **deposit** that deposits a specified amount to the account.
- **toString** method that returns id, balance and date created.

Create two subclasses for **Checking** and **Saving** accounts from **Account** class. A **Checking** account has an overdraft limit equals to 1000, but a **savings** account cannot be overdrawn.

Step 1: Draw the UML diagram for the classes then implement the classes.

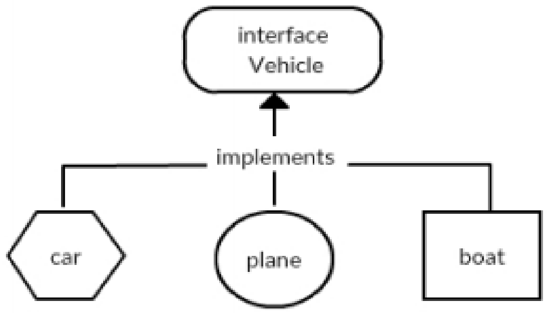
Step 2: Implement classes using eclipse.

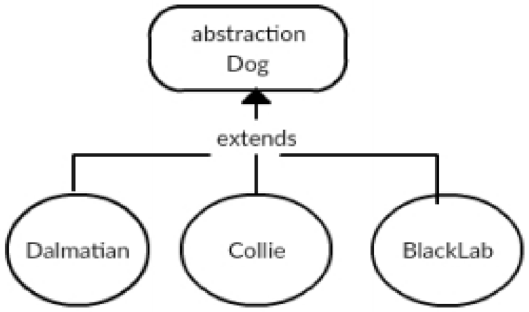
Step3: Write a test program that creates different objects of **Saving account**, and **Checking account**, and invokes their **toString()** methods and to call **withdraw** and **deposit** method.

Lab 8: Abstract Classes and Interfaces

Objectives

- To understand the concept of abstract classes.
- To be able to differentiate between abstract and concrete classes.
- To define a natural order using the Comparable interface.
- To declare custom interfaces





Theory

Abstract classes are used to declare common characteristics of subclasses. An abstract class cannot be instantiated. It can only be used as a superclass for other classes that extend the abstract class. Abstract classes are declared with the abstract keyword. Abstract classes are used to provide a template or design for concrete subclasses down the inheritance tree.

An abstract method is a method signature without implementation. Its implementation is provided by the subclasses. A class that contains abstract methods **must** be declared abstract.

An interface defines one or more constant identifiers and abstract methods. A separate class *implements* the interface class and provides the definition of the abstract methods. Interfaces are used as a design technique to help organize properties (identifiers) and behaviors (methods) the implementing classes may assume.

Differences between abstract and interface classes:

Interface	Abstract class
Interface support multiple inheritance	Abstract class does not support multiple inheritance
Interface does'n Contains Data Member	Abstract class contains Data Member
Interface does'n contains Cunstructors	Abstract class contains Cunstructors
An interface Contains only incomplete member (signature of member)	An abstract class Contains both incomplete (abstract) and complete member
An interface cannot have access modifiers by default everything is assumed as public	An abstract class can contain access modifiers for the subs, functions, properties
Member of interface can not be Static	Only Complete Member of abstract class can be Static

Syntax

Abstract Class

[*modifier*] **abstract** class Class-Name {

/** Body of the class */

}

Abstract Method

[*modifier*] **abstract** *data-type* method-name([*parameters-declarations*]) {…}

Interface

[*modifier*] **interface** InterfaceName {

/** Constant declarations */

/** Method signatures */

}

There are famous interface classes that you will use, such as **Comparable** which will help you in comparing objects by implementing the **compareTo** method and **Cloneable** which will help you cloning objects by implementing the clone method.

Exercises

1. Create the classes in the following inheritance hierarchy. An **Employee** should have a **first name**, **last name** and **ID number**. In addition, the **SalariedEmployee** should have a **weekly salary**; an **HourlyEmployee** should have a **wage** and a **number of hours worked**; a **CommissionEmployee** should have a commission **rate** and **gross sales**; and a **BaseCommissionEmployee** should have a **base salary**. Each class should have an appropriate constructor, setters and getters. Each Employee should have an **earning()** and **toString()** methods. Also make Employee comparable. The compareTo method compares between the employees based on **earnings**.

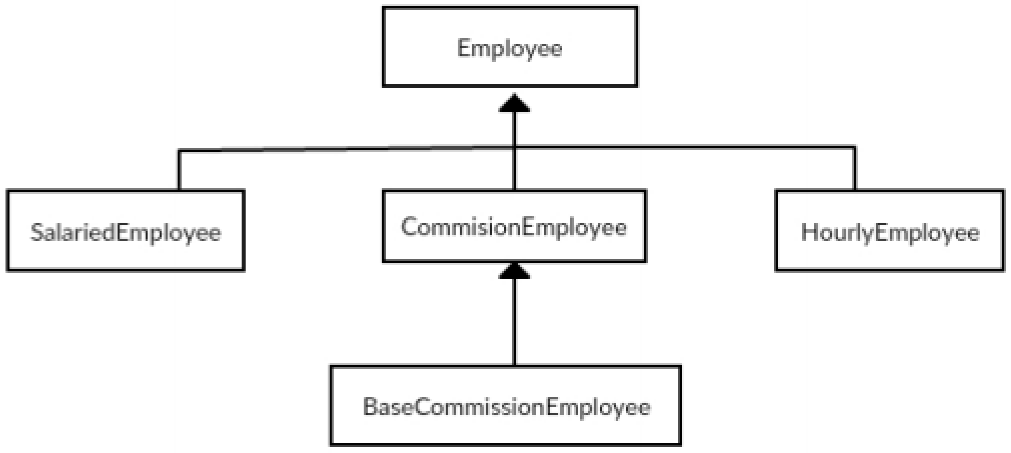
- Write a method that finds the total earnings of all the employees in an array. The method signature is:

```
public static double totalEarning(Employee[] a)
```

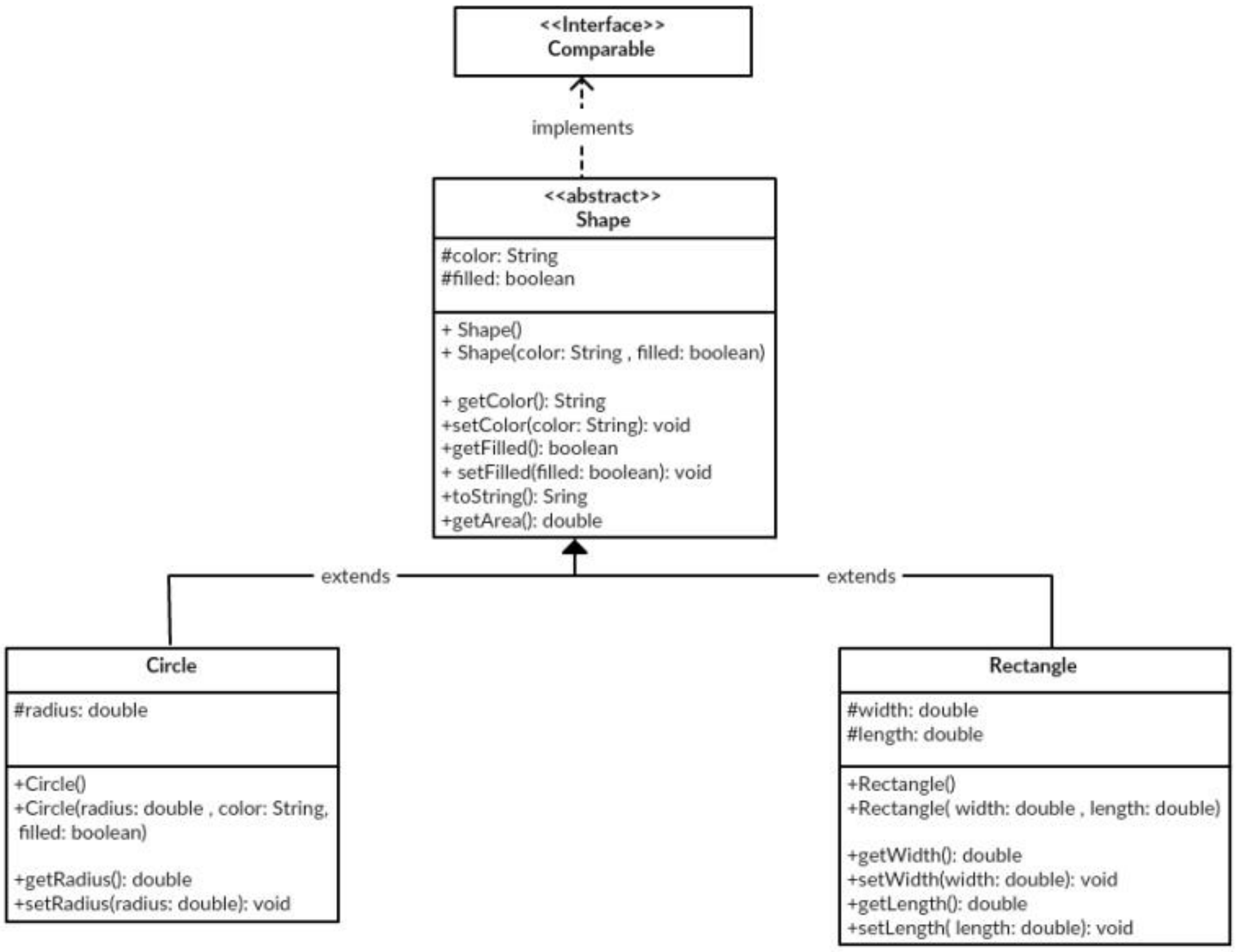
- Write another method that sorts an array of Objects using the compareTo method. The signature of this method is:

```
public static void sort(Object[] a)
```

- Write a test program that creates an array of 5 employees and computes their total earnings using the totalEarning method.



2. Implement the superclass **Shape** and its subclasses **Circle, Rectangle** and **Square** as shown in the following UML class diagram, note that the compareTo method will compare based on **area**:



Create an array list of 5 different shapes, then sort the list and print the type of each object with its area.

Lab 9: Exception-handling and text I/O

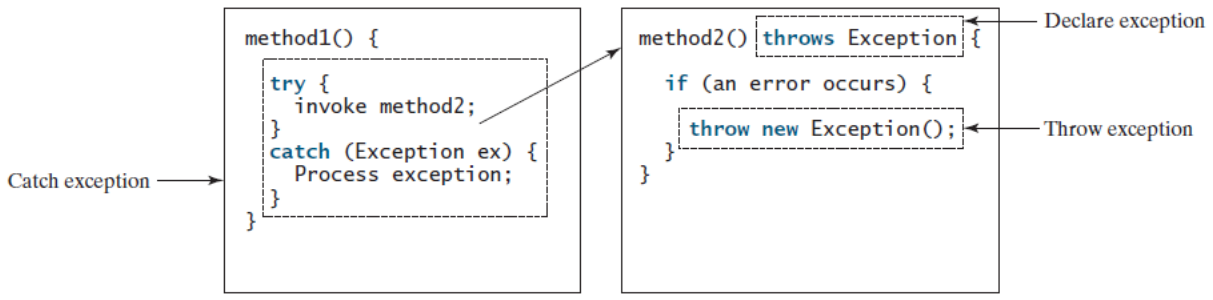
Objectives

- To understand the concept of Exception and Exception handling.
- How to throw an exception?
- How to write a try-catch block to handle exception.

Theory

A Runtime errors occur while a program is running if the JVM detects an operation that is impossible to carry out, in java, the runtime errors are thrown as exception, so the exception is an object that represents an error or condition that prevents execution from proceeding normally. If the exception is not handled, the program will terminate abnormally.

Java exception handling model is based on three operations: declaring an exception, throwing an exception, and catching an exception, as in the following figure.



Syntax

Try-catch block:

```
 try{
        statements;
}catch(Exception1 exVar1){
        Handler for exception1;
} catch(Exception2 exVar2){
        Handler for exception2;
}…
   catch(ExceptionN exVarN){
        Handler for exceptionN;
 }
```

Exercises

1) Write a **bin2Dec(String binaryString)** method that converts a binary string into a decimal number. Implement the **bin2Dec** method to throw a **NumberFormatException** if the string is not binary String.

2) Design a **Circle** class that has **radius** and **color** as attributes and allow the setRadius method to throw an **IllegalArgumentException** if the radius is negative.

3) Write a program that reads unknown number of students from file named **students.txt**, the data stored in this file as following:

```
Ahmad 90 85 78
Musa   76 80 84
Salam 85 90 99
```

Then calculate the average of each student and save the results on a file named **output.txt** as following:

```
Ahmad 84.3
Musa 80
Salam 91.3
```

Lab 10: JavaFX basics and UI controls

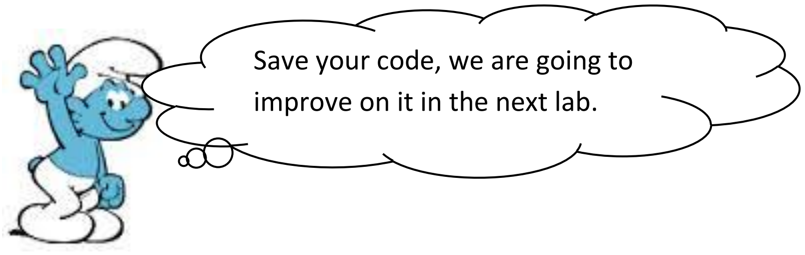
Objectives

- To write a javaFX program and understand the relationship among stages, scenes, and nodes.
- To create user interface using panes, UI controls, and shapes.
- To create a graphical user interface with various user-interface controls.
- Learn how to use several UI controls such as **Button**, **Label**, and **TextField**.

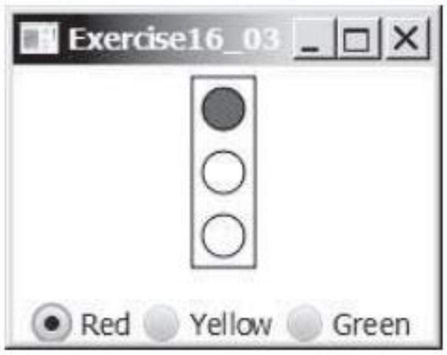
Theory

A graphical user interface (GUI) makes a system user-friendly and easy to use. Creating a GUI requires creativity and knowledge of how UI controls work. Since the UI controls in javaFX are very flexible and versatile, you can create a wide assortment of useful user interface for rich Internet applications.

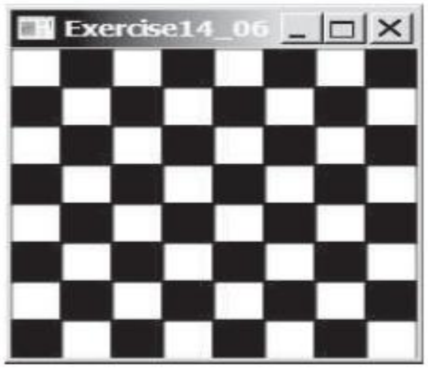
Exercises


Save your code, we are going to improve on it in the next lab.

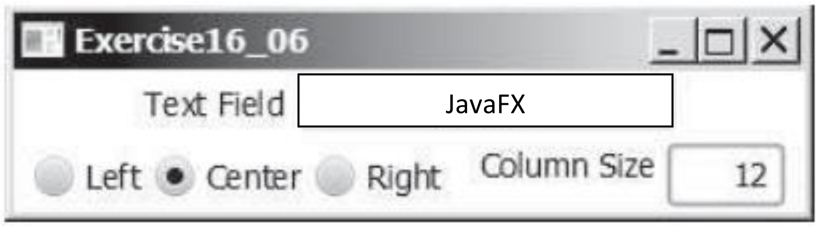
1. Write a program that display the traffic light as in the figure below:



2. Write a java program that displays a checkboard in which each white and black cell is a **Rectangle** with a fill color black or white as in the figure below:



3. Write a java program that displays the following UI:



Lab 11: Event-Driven Programming

<u>**Objectives**</u>

- To get the taste of event-driven programming.
- To describe events, even sources, and event classes.
- To define event handler classes using inner classes, and simplify event handling using lambda expression.

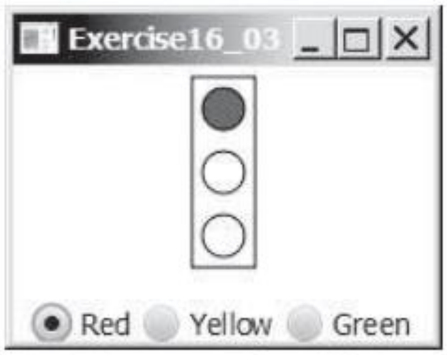
  To develop applications using **MouseEvents** and **KeyEvents**.

<u>**Theory**</u>

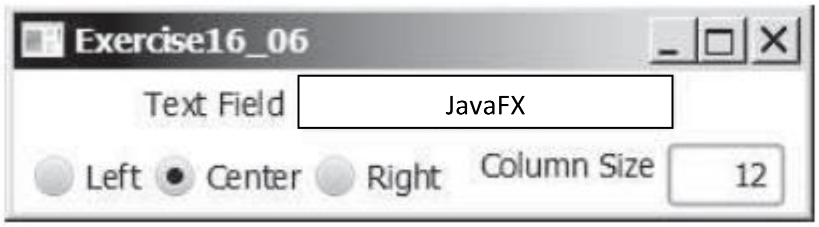
When you run a Java GUI program, the program interacts with the user, and the events drive its execution. This is called *event-driven programming.* An *event* can be defined as a signal to the program that something has happened. Events are triggered by external user actions, such as mouse movements, mouse clicks, and keystrokes. The program can choose to respond to or ignore an event.

Exercises

1. Write a program that displays the traffic light as in the figure below, the program will simulate a traffic light, the user can select one of the three lights: red, yellow, or green. When a radio button is selected, the light is turned on. Only one light can be on at a time, No light is on when the program starts.

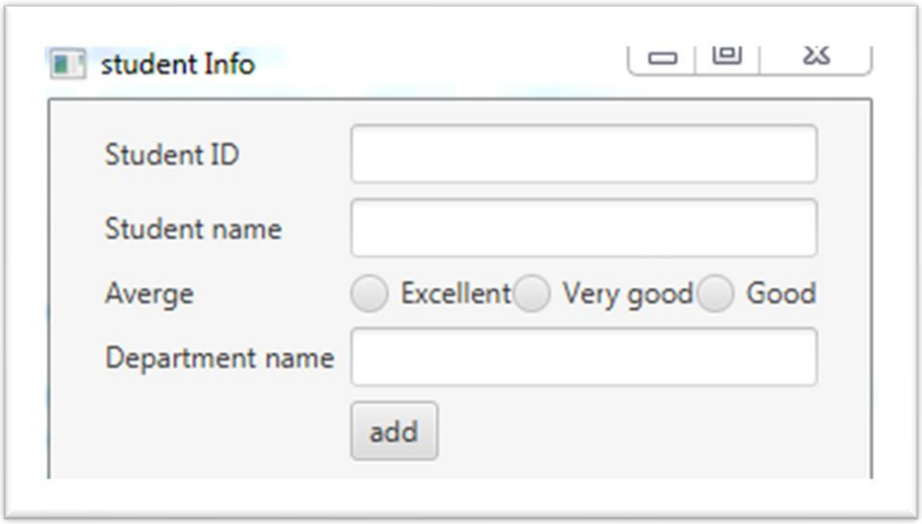


2. Write a java program that displays the following UI and sets the horizontal-alignment and column-size properties of text field dynamically:



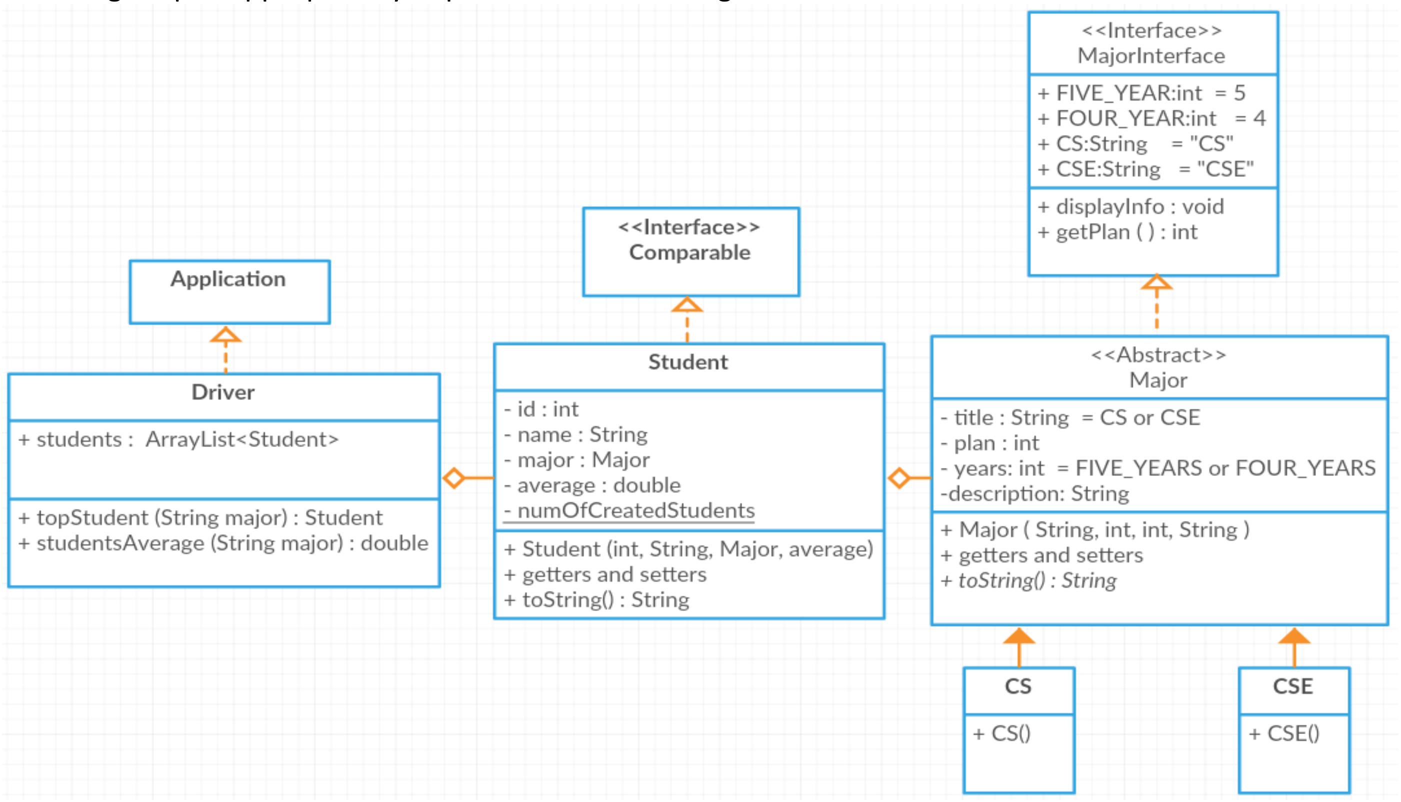
3. Write a complete Java program which will display a user interface (UI) as in the following figure using javaFX to collect student information. The collected information are:
    a. Student number
    b. Student name
    c. Average
    d. Department name

    When your program started, it should display a UI that enables the user to enter the required information and add button that will take information and saves it in a file.



Lab 12- Extra Lab: JavaFX and Event-Driven Programming

- Using Eclipse Appropriately implement the following **UML**:

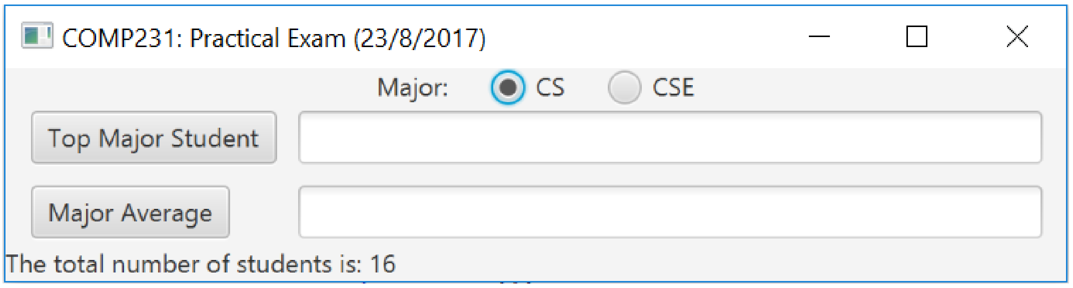


  - Add getters and setters as necessary.
  - Every time the **Student** constructor invoked, the **numOfCreatedStudents** incremented.

- Create a file “**students.txt”** to your project with the following data:
- 1120752:Majdi : CSE : 70
- 1130340:Rand : CSE : 75
- 1131913:Maen : CS : 50
- 1141339:Areen : CSE : 65
- 1141622:Deema : CSE : 80
- 1141647:Wafa : PHY : 55
- 1141945:Kasandra : PHY : 50
- 1142149:Manar : CSE : 65
- 1142529: Sami : CS : 78
- 1142753: Mohammad : CS : 88
- 1150304: Maab : MATH : 72
- 1150467: Amr : ACC : 99
- 1151504: Hadi : CSE : 82
- 1152270: Atheer : CS : 92
- 1152770: Farhat : SCI : 80
- 1153216: Isra : ENG : 76
- 1153279: Yazeed : CS : 67
- 1160063: Ayah : CS : 99
- 1160144: Feras : CSE : 97
- 1160724: Sajeda : CSE : 89
- 1161357: Bara : CS
- 1161728: Omar : CS
- 1161798: Husam : CSE : 70

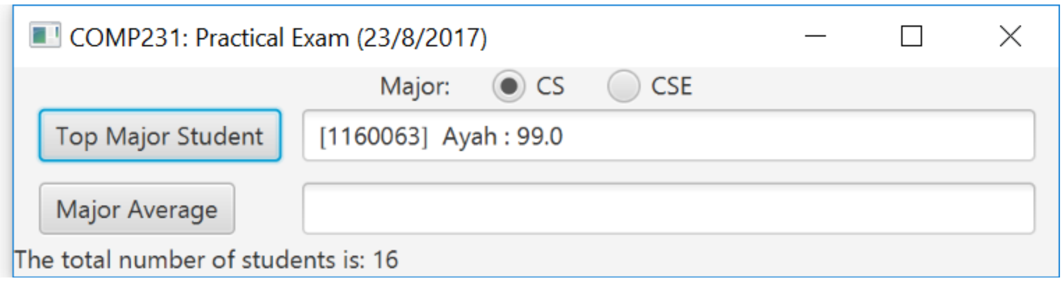
  * Each line in the file consists of: student information: ***id***, ***name***, ***major***, and average marks, separated by a (**:**).
- Create a driver class named **Driver** that has an ArrayList of **Student**s.
- The driver class does the following:
  - At start, it reads the **students.txt** file records (line by line). Converts each line to a specific student object with appropriate major and add it to the students ArrayList.
    - Notes:
      - Only consider the students with CS or CSE major.
      - Some student’s information are missing, in this case ignore this student.
  - Implement the following methods:
    - ***topStudent(major)***: returns the students that has the best average in his major.
    - ***studentsAverage(major)***: returns the average of students in the major.

- Build the following JavaFX GUI (*as close as possible*):


COMP231: Practical Exam (23/8/2017)

The total number of students is: 16

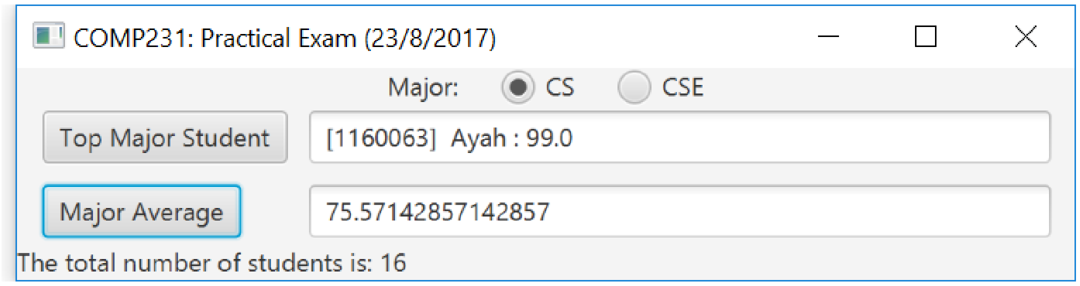
  - Clicking the "***Top Major Student***" button will execute the method ***topStudent(major)*** and pass to it the value of selected Major radio button. The output will be displayed in the TextFiled as follow:


COMP231: Practical Exam (23/8/2017)

[1160063] Ayah : 99.0

The total number of students is: 16

  - Clicking the "***Major Average***" button will execute the method ***studentsAverage(major)*** and pass to it the value of selected Major radio button. The output will be displayed in the TextFiled as follow:


COMP231: Practical Exam (23/8/2017)

[1160063] Ayah : 99.0

75.57142857142857

The total number of students is: 16