MEGA ASSIGNMENT

Faheem Akhtar Rajput (Java OOP)

JULY 8, 2022

MUHAMMAD TAYYAB BHUTTO (023-21-0286)
Sukkur IBA

Mega Assignment

1. **A java program that displays the roots of quadratic equation $ax^2+bx+c=0$ and Calculating the discriminate D to describe the nature of roots. The program should prompt user for input during run time.**

DESCRIPTION:

The Quadratic Formula**.** The quadratic equation

$$ax^2 + bx + c = 0$$

Have the solutions

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

```
import java.lang.Math;
import java.util.Scanner;
public class QuadraticEquation{
    static void quadraticEquation(double a, double b, double c){
        double firstRoot = 0, secondRoot = 0;
        double determinent = (b*b)-(4*a*c);
        double squareRoot = Math.sqrt(determinent);
        if (determinent > 0){
            firstRoot = (-b + squareRoot) / (2 * a);
            secondRoot = (-b + squareRoot) / (2 * a);
            System.out.println("First Root is " + firstRoot + " and " +
" Second Root is " + secondRoot);
        }
        else if (determinent == 0){
             System.out.println("Root is " + (-b + squareRoot) / (2 *
a));
        }

    }

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter value of a: ");
        int a = input.nextInt();
        System.out.print("Enter value of b: ");
        int b = input.nextInt();
        System.out.print("Enter value of c: ");
        int c = input.nextInt();
```

```
        quadraticEquation(a, b, c);
    }
}
```

```
Enter value of a: 15
Enter value of b: 68
Enter value of c: 3
First Root is -0.044555558333472335 and  Second Root is -0.044555558333472335
```

2. **A java program to display the Fibonacci sequence. The program should prompt user for input during run time.**

DESCRIPTION:

In mathematics, the Fibonacci numbers or Fibonacci series or Fibonacci sequence are the

numbers in the following integer

$$0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,\ 89,\ 144,\ \ldots$$

By definition, the first two numbers in the Fibonacci sequence are 0 and 1, and each subsequent number is the sum of the previous two.

In mathematical terms, the sequence F_n of Fibonacci numbers is defined by the recurrence

$$F_n = F_{n-1} + F_{n-2},$$

With see values

$$F_0 = 0,\ F_1 = 1.$$

Here is a simplest Java Program to generate Fibonacci

```
import java.util.Scanner;

public class FibonacciSeries {
    static int Fibonacci(int numbers){
        if (numbers == 0){
            return numbers;
        }
        if (numbers == 1 || numbers == 2){
            return 1;
        }
        return (Fibonacci(numbers-1) + Fibonacci(numbers-2));
    }
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
```

```
        System.out.println("Enter Number. ");
        int number = input.nextInt();
        for (int i = 0; i < number; i++){
            System.out.print(Fibonacci(i) + " ");
        }
    }
}
```

```
Enter Number. 10
0 1 1 2 3 5 8 13 21 34
```

3. Write a java program to give example for command line arguments

DESCRIPTION:

The java command-line argument is an argument i.e. passed at the time of running the java program. The arguments passed from the console can be received in the java program and it can be used as an input. Command line arguments represent the values passed to main() method. Here, args[] is one dimensional array of String type. So it can store a group of strings, passed to main() from outside by the user i.e, at the time of running the program. The prototype of main( ) when it supports command line arguments is as follows:

public static void main(String[] args)

```
import java.util.Scanner;

public class CommadineArgument {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.println("Command Line argument is "+args[0]);
    }
}
```

```
PS F:\docs\Java\Mega Assignment> javac .\CommadineArgument.java
PS F:\docs\Java\Mega Assignment> java CommadineArgument Muhammad Tayyab Bhutto
Command Line argument is Muhammad
```

4. Write a java program to sort given list of numbers ,

DESCRIPTION:

The algorithm works by comparing each item in the list with the item next to it, and swapping them if required. In other words, the largest element has bubbled to the top of the array. The algorithm repeats this process until it makes a pass all the way through the list without swapping any items

```java
import java.util.Scanner;
public class BubbleSorting {
    static void Sort(int arr[]){
        int temp;
        for (int i = 0; i < arr.length; i++) {
            for (int j = i+1; j < arr.length; j++) {
                if (arr[i] > arr[j]){
                    temp = arr[i];
                    arr[i] = arr[j];
                    arr[j] = temp;
                }
            }
        }
        System.out.print("Sorted Numbers: ");
        for (int i : arr) {
            System.out.print(i + " ");
        }
    }
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int arr[];
        System.out.print("How many numbers you have? ");
        int len = input.nextInt();
        arr = new int[len];
        System.out.print("Enter Numbers: ");
        for (int i = 0; i < len; i++){
            arr[i] = input.nextInt();
        }
        Sort(arr);
    }
}
```

```
How many numbers you have? 5
Enter Numbers: 1 5 3 4 2
Sorted Numbers: 1 2 3 4 5
```

5. Write a java program to search for an element in a given list of elements (Linear Search)

DESCRIPTION:

Linear search is one of the basic search techniques that we've now. Although this is not a very good search technique, one should understand this concept. Let's consider our aim to search for a key element in an array of elements. We loop through all the array elements and

check for existence of the key element. Since we go element by element, this search is called as linear search or sequential search. Search element is called as key element.

```java
import java.util.Scanner;
public class LinearSearch {
    static int Search(int number){
        int arr[] = {1,2,3,45,5,4,6,7,8,9,10};
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == number){
                return i;
            }
        }
        return -1;
    }
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the numner you want to search. ");
        int searchingNumber = input.nextInt();
        if (Search(searchingNumber) == -1){
            System.out.println("Not found.");
        }
        else{
                    System.out.println("Element  found  at  index  "  +
Search(searchingNumber));
        }
    }
}
```

```
Enter the numner you want to search. 5
Element found at index 4
```

6. Write a java program to search for an element in a given list of elements (Binary Search)

DESCRIPTION:

Generally, to find a value in unsorted array, we should look through elements of an array one by one, until searched value is found. In case of searched value is absent from array, we go through all elements. In average, complexity of such an algorithm is proportional to the length of the array. Situation changes significantly, when array is sorted. If we know it, random access capability can be utilized very efficiently to find searched value quick. Cost of searching algorithm reduces to binary logarithm of the array length. For reference, $\log_2(1\ 000\ 000) \approx 20$. It means, that **in worst case**, algorithm makes 20 steps to find a value in sorted array of a million elements or to say, that it doesn't present it the array.

Note: Elements should be in sorted order in the given array

```java
import java.util.Scanner;

public class BinarySearch {
    static int Search(int[] arr, int element, int beg, int end) {
        int mid = (beg + end) / 2;
        if (beg <= end) {
            if (element == arr[mid]) {
                return mid;
            } else if (element < arr[mid]) {
                return Search(arr, element, beg, mid - 1);
            } else {
                return Search(arr, element, mid + 1, end);
            }
        } else {
            return - 1;
        }
    }

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int[] arr = {1,2,3,4,5,6,7,8,9,10};
        System.out.print("Enter element. ");
        int element = input.nextInt();
        int index = Search(arr, element, 0, arr.length-1);
        if (index == -1){
            System.out.println("element not found.");
        } else {
            System.out.println("element found on index " + index);
        }
    }
}
```

```
Enter element. 10
element found on index 9
```

7. Write a java program for the following

Example for call by value Example for call by reference.

DESCRIPTION:

Java supports call by value by passing a value and returning the value. Call by reference is way of passing object into setter method and returning object as type. Steps to call are:

1. Create a object which contains setters and getters.
2. Use the above object into implementation class and pass Object as a parameter into method and also pass any primitive value into method.
3. Return the value and Object as return types.

```java
public class CallByValueAndReference {
    private String name;
    private int id;
    private CallByValueAndReference obj;
    public void setName(String name){
        this.name = name;
    }
    public void setId(int id){
        this.id = id;
    }
    public String getName(){
        return name;
    }
    public int getId(){
        return id;
    }
    public void setValue(CallByValueAndReference obj){
        obj.name = "Muhammad Tayyab Bhutto";
        obj.id = 286;
        this.obj = obj;
    }
    public CallByValueAndReference getValue(){
        return obj;
    }

    public static void main(String[] args) {
        CallByValueAndReference obj = new CallByValueAndReference();
        obj.setName("Muhammad Muzammil Bhutto");
        obj.setId(786);
           System.out.println("Call  By  Value.  "  +  obj.getName()  +"   "+
obj.getId());
        obj.setValue(obj);
        System.out.println("Call By Reference. " + obj.name + "  "+obj.id);
    }
}

```

```
Call By Value. Muhammad Muzammil Bhutto  786
Call By Reference. Muhammad Tayyab Bhutto  286
```

44. Write a java program to demonstrate static variables, methods, and blocks.

DESCRIPTION:

Static Keyword in Java is used to access the data without creating Object. Also static creates the single memory allocation for all Objects. Static method can accept only static variables. Static block has height priority than static methods (even there is main) as well as instance methods.

```java
// Why there is a need of static: it will execute first in program and
will destroy when program will end.
public class StaticBlockVarMeth {
    static int count = 0;
    static int Calculate(int val){
        return val * count;
    }
    public static void main(String[] args) {
        count = 10;
        System.out.println("Calling Calculate Method in Main Method. "
+ Calculate(10));
         System.out.println("Value of Static variable in Main after
updating. " + count);
    }
    static {
        System.out.println("Hey this static Block And Calling Calculate
Method in static block. " + Calculate(9));
    }
}
```

```
Hey this static Block And Calling Calculate Method in static block. 0
Calling Calculate Method in Main Method. 100
Value of Static variable in Main after updating. 10
```

45. Write a java program to give the example for ‘super’ keyword

DESCRIPTION:

Reusability is very important feature in Inheritance, where accessing base class properties and methods is needed. Super keyword is such object which do the job, by handling the super class properties, methods and constructors. Invoking super keyword can be done in following way:

Variable➋super.variable_name;

Methods➋super.method_name();

Constructor➋super(parameter_list);

```java
// Three usages of super Keyword
// Variable having same name in super class and in subclass
```

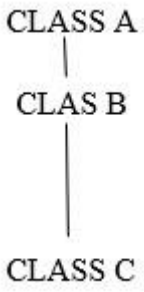
```java
// Method having same name in super class and in subclass
// Calling Constructor
class SuperClass {
    String name;
    SuperClass(String name){
        this.name = name;
    }
    void Display(){
        System.out.println("Name " + name);
    }
}
class SubClass extends SuperClass{
    String name;
    SubClass(String name){
        super(name);    // Super With Constructor
        name = super.name;  // Super With Variable Name
    }
    void Display(){
        super.Display();
    }
}
public class SuperKeyword {

    public static void main(String[] args) {
       SuperClass superClass = new SuperClass("Muhammad Tayyab Bhutto");
        System.out.println("before using super keyword.");
        superClass.Display();
        SubClass subClass = new SubClass("Muhammad Muzammil Bhutto");
        System.out.println("after using super keyword.");
        subClass.Display();
    }
}
```

```
before using super keyword.
Name Muhammad Tayyab Bhutto
after using super keyword.
Name Muhammad Muzammil Bhutto
```

10. Write a java program for multi-level inheritance?

DESCRIPTION:

Inheritance is one way of implementing super-sub class relationship, for data usability and readability. Implementing Multi-level inheritance is better one for Objects whose data will be accessed by many objects there by increasing readability.




```
class Animal {
    private String name;
    public void setName(String name){
        this.name = name;
    }
    public String getName(){
        return name;
    }
}
class Dog extends Animal {
    private String name;
    public void setName(String name){
        super.setName(name);
        this.name = super.getName();
    }
    public String getName(){
        return name;
    }
}
class Cat extends Animal {
    private String name;
    public void setName(String name){
        super.setName(name);
        this.name = super.getName();
    }
    public String getName(){
        return name;
    }
}
public class MultiLevelInheritence {
    public static void main(String[] args) {
```

```
        Animal animal = new Animal();
        Dog dog = new Dog();
        Cat cat = new Cat();
        animal.setName("Animals");
        dog.setName("Sheru");
        cat.setName("Billy");
        System.out.println(animal.getName());
        System.out.println(dog.getName());
        System.out.println(cat.getName());

    }
}
```

```
Animals
Sheru
Billy
```

11. Write a java program to differentiate method overloading and overriding?

DESCRIPTION:

Polymorphism is foremost concern in java to implement methods behavior in class and in one or more classes. Having one or forms for the desire of data sharing. Overloading is the one, whose methods have same name but different type signatures. Type promotion is supported in Overloading.

Overriding on the other hand, methods with same name and parameters in more than one class which are of 'IS-A' relationship.

```
// Overloading and Overriding and two different kinds
// Overloading: when two or more methods having same name and different
parameters is know as overloading it is compile time pollymorphism
// Overriding: when two or more methods having same name and same
parameters is know as overriding it is run time pollymorphism
class Application {
    private String appName;
    private String userName;
    private String password;

    // setData is an overloading method
    public void setData(String appName){
        this.appName = appName;
    }
    public void setData(String userName, String password){
        this.userName = userName;
```

```
        this.password = password;
    }
    public String getAppName(){
        return appName;
    }
    public String getUsername(){
        return userName;
    }
    public String getPassword(){
        return password;
    }


}
class Profile extends Application{
    private String userName;
    // Overriding
    public void setData(String userName){
        this.userName = userName;
    }
    public String getUsername(){
        return userName;
    }
}
public class OvelodingAndOverloading {
    public static void main(String[] args) {
        Application application = new Application();
        Profile profile = new Profile();
        application.setData("My App");
        application.setData("Muhammad Tayyab Bhutto", "asd123");
        profile.setData("muhammad-tayyab-bhutto");
        System.out.println("App Name: " + application.getAppName() + "
\nUser Name: " + application.getUsername());
        System.out.println("Profile id: " + profile.getUsername());
    }
}
```

```
App Name: My App
User Name: Muhammad Tayyab Bhutto
Profile id: muhammad-tayyab-bhutto
```

12. Java program to differentiate method overloading and constructor overloading.

DESCRIPTION:

This program describes the difference between method overloading and constructor loading. Method overloading is a feature that allows a class to have two or more methods having

same name, if their argument lists are different. Constructor overloading that allows a class to have more than one constructor having different argument lists. When overload method is called, java looks for a match between the arguments used to call the method and the method parameters. Finally it matches and displays the output.

```
class Application {
    private String appName;
    private String userName;
    private String password;
    // Constructor overloading
    Application(){}
    Application(String appName){
        this.appName = appName;
    }
    Application(String userName, String password){
        this.userName = userName;
        this.password = password;
    }
    // setData is an overloading method
    public void setData(String appName){
        this.appName = appName;
    }
    public void setData(String userName, String password){
        this.userName = userName;
        this.password = password;
    }
    public String getAppName(){
        return appName;
    }
    public String getUsername(){
        return userName;
    }
    public String getPassword(){
        return password;
    }


}
public class MethodAndConstructorOverloading {
    public static void main(String[] args) {
        Application application = new Application();
         System.out.println("========================================
==================================");
        System.out.println("Constructor Overloading");
        Application application1 = new Application("CodeSmashers");
```

```
          Application application2 = new Application("Muhammad Tayyab
Bhutto" + "qwe123");
        System.out.println("App Name: " + application1.getAppName() + "
\nUser Name: " + application2.getUsername());
         System.out.println("=======================================
==================================");
        System.out.println("Method Overloading");
        application.setData("My App");
        application.setData("Muhammad Tayyab Bhutto", "asd123");

        System.out.println("App Name: " + application.getAppName() + "
\nUser Name: " + application.getUsername());
    }
}
```

```
========================================================================
Constructor Overloading
App Name: CodeSmashers
User Name: null
========================================================================
Method Overloading
App Name: My App
User Name: Muhammad Tayyab Bhutto
PS F:\docs\Java\Mega Assignment>
```

13. Write a JAVA program for example of try and catch block. In this check whether the given array size is negative or not. Also demonstrate the use of finally block.

DESCRIPTION:

Program explains exceptions using try and catch block. In the try block array is created. Check the array size is negative or not. If the array size is positive then it will be display array size otherwise it throws an error like java.lang.NegativeArraySizeException.

```
import java.util.Scanner;

public class TryCatch {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int number;
        int[] arr;
        try{
            System.out.print("Enter Size Of Array: ");
            number = input.nextInt();
            arr = new int[number];

        } catch (NegativeArraySizeException e){
            System.out.println(e);
```

```
            System.out.print("Enter Again Size Of Array: ");
            number = input.nextInt();
            arr = new int[number];
        }
        finally{
            System.out.println("finally executed.");
        }
    }
}
```

```
Enter Size Of Array: -1
java.lang.NegativeArraySizeException: -1
Enter Again Size Of Array: 5
finally executed.
PS F:\docs\Java\Mega Assignment>
```

14. Write a JAVA program for creation of user defined exception.

DESCRIPTION: If you are creating your own Exception that is known as custom exception or user-defined exception Java custom exceptions are used to customize the exception according to user need.

By the help of custom exception, you can have your own exception and message.

```
import java.util.Scanner;
class EmployeeIdException extends Exception{
    public EmployeeIdException(String exception){
        super(exception);
    }
}
class Employee{
    private int id;
    public void setId(int id) throws EmployeeIdException{
        if (id <= 0 || id > 9999){
            throw new EmployeeIdException("Invalid Employee ID.");
        }
        else {
            this.id = id;
        }
    }
    public int getId(){
        return id;
    }
}
public class UserDefinedExceptions{
```

```
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        Employee employee = new Employee();
        int id;
        try {
            System.out.print("Enter ID: ");
            id = input.nextInt();
            employee.setId(id);

        } catch (EmployeeIdException e) {
            System.out.println(e.getMessage());
            try{
                System.out.print("Please Enter ID Again: ");
                id = input.nextInt();
                employee.setId(id);
            }
            catch(EmployeeIdException ex){
                System.out.println(ex.getMessage());
                  System.out.println("Thanks your Account is temporary
Blocked contact us for more.");
            }
        }
    }
}
```

```
Invalid Employee ID.
Please Enter ID Again: -4564
Invalid Employee ID.
Thanks your Account is temporary Blocked contact us for more.
PS F:\docs\Java\Mega Assignment>
```

15. Write a java program to illustrate multiple inheritance using interfaces?

DESCRIPTION: Interfaces are completely abstract classes in Java that provide you with a uniform way to properly delineate the structure or inner workings of your program from its publicly available interface, with the consequence being a greater amount of flexibility and reusable code as well as more control over how you create and interact with other classes. More precisely, they are a special construct in Java with the additional characteristic that allow you to perform a kind of multiple inheritance i.e. classes that can be upcast to more than one class; a technique from which you are severely restricted from undertaking (and with good reason) when working exclusively with classes. To create an interface in Java, you simply use the special interface keyword.

Note: Interface consists only abstract methods and static final members.

```
package Animal;

public abstract class Animal {
    protected int legs;
    protected Animal(int legs){
        this.legs = legs;
    }
    public void walk(){
        System.out.println("Animal walks.");
    }
    public abstract void eat();
}
```

```
package Animal;

public class Spider extends Animal{
    protected Spider(){
        super(8);
    }

    @Override
    public void eat() {
        System.out.println("Spider eats insects");
    }
}
```

```
package Animal;

public interface Pet {
    public abstract String getName();
    public abstract void setName(String name);
    public abstract void play();
}
```

```
package Animal;

public class Cat extends Animal implements Pet{
    String name;
    Cat(String name){
        super(4);
        this.name = name;
    }
    Cat(){
        this("Cat");
    }
    @Override
    public void eat() {
        System.out.println("Cat eats mice, fish, hens");
    }
    @Override
    public void setName(String name){
        this.name = name;
    }
    @Override
    public String getName(){
        return name;
```

```
    }
    @Override
    public void play(){
        System.out.println("Cat is playing");
    }

}
```

```
package Animal;

public class Fish extends Animal implements Pet{
    String name;
    protected Fish(){
        super(0);
    }
    @Override
    public void walk(){
        System.out.println("Fish can't walk and don't have legs.");
    }
    @Override
    public void eat(){
        System.out.println("Fish eat feed.");
    }

    @Override
    public void play() {
        System.out.println("Fish is playing in water.");
    }

    @Override
    public void setName(String name) {
        this.name = name;
    }

    @Override
    public String getName() {
        return name;
    }

}
```

```
package Animal;

public class Main {
    public static void main(String[] args) {
        Fish d = new Fish();
        Cat c = new Cat("Fluffy");
        Animal a = new Fish();
        Animal e = new Spider();
        Pet p = new Cat();
        d.eat();
        d.walk();
        c.setName("Cat");
        System.out.println(c.getName());
        c.play();
        c.eat();
        a.walk();
        a.eat();
        e.eat();
```

```
        e.walk();
        p.setName("Fish");
        System.out.println(p.getName());
        p.play();
    }
}
```

```
Fish eat feed.
Fish can't walk and don't have legs.
Cat
Cat is playing
Cat eats mice, fish, hens
Fish can't walk and don't have legs.
Fish eat feed.
Spider eats insects
Animal walks.
Fish
Cat is playing

Process finished with exit code 0
```

16. Write a java program to create a package called employee and implement this package out of the package?

DESCRIPTION: Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations and annotations easier, etc.

Some of the existing packages in Java are −

- **java.lang** − bundles the fundamental classes
- **java.io** − classes for input , output functions are bundled in this package

```
package Employee;
public class Employee {
    private String firstName;
    private String lastName;
    private String nationalIdCardNumber;
    public Employee() {
    }
    public Employee(String firstName, String lastName, String
nationalIdCardNumber) {
        this.firstName = firstName;
```

```
        this.lastName                   =                   lastName;
        this.nationalIdCardNumber       =       nationalIdCardNumber;
    }
    void              setFirstName(String               firstName){
        this.firstName                  =                  firstName;
    }
    String                                              getFirstName(){
        return                                              firstName;
    }
    void               setLastName(String               lastName){
        this.lastName                   =                  lastName;
    }
    String                                             getLastName(){
        return                                              lastName;
    }
    void    setNationalIdCardNumber(String    nationalIdCardNumber){
        this.nationalIdCardNumber       =       nationalIdCardNumber;
    }
    String                                  getNationalIdCardNumber(){
        return                                  nationalIdCardNumber;
    }
    public              String              toString()              {
        return   firstName+   "   "   +   lastName   +   "   CNIC   "   +
nationalIdCardNumber;
    }
    double                                              earnings(){
        return                                                 0.0;
    }
}

//Main
```

```
import Employee.*;
public class Main {
    public static void main(String[] args) {
        Employee employee = new Employee("Muhammad" , "Tayyab Bhutto", "023-21-
0286");
        System.out.println(employee.toString());
    }
}
```

```
Muhammad Tayyab Bhutto CNIC 023-21-0286

Process finished with exit code 0
```

17. Write a JAVA program to create a dialog box and menu.

DESCRIPTION:

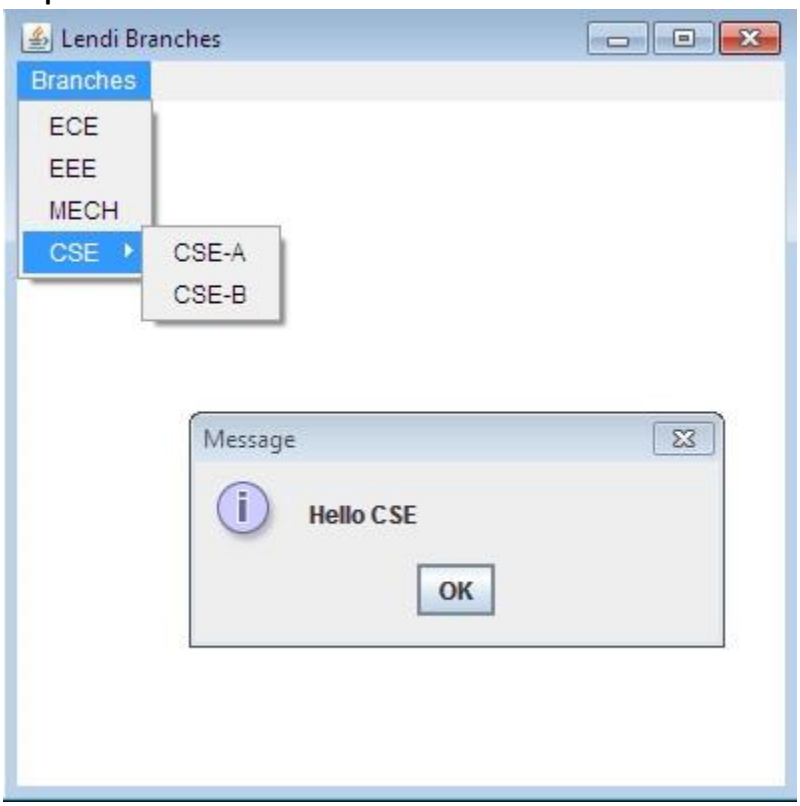
Dialog control represents a top-level window with a title and a border used to take some form of input from the user. This class inherits methods from the following classes:

- java.awt.Window

- java.awt.Component
- java.lang.Object

package: **java.awt.Dialog**

output

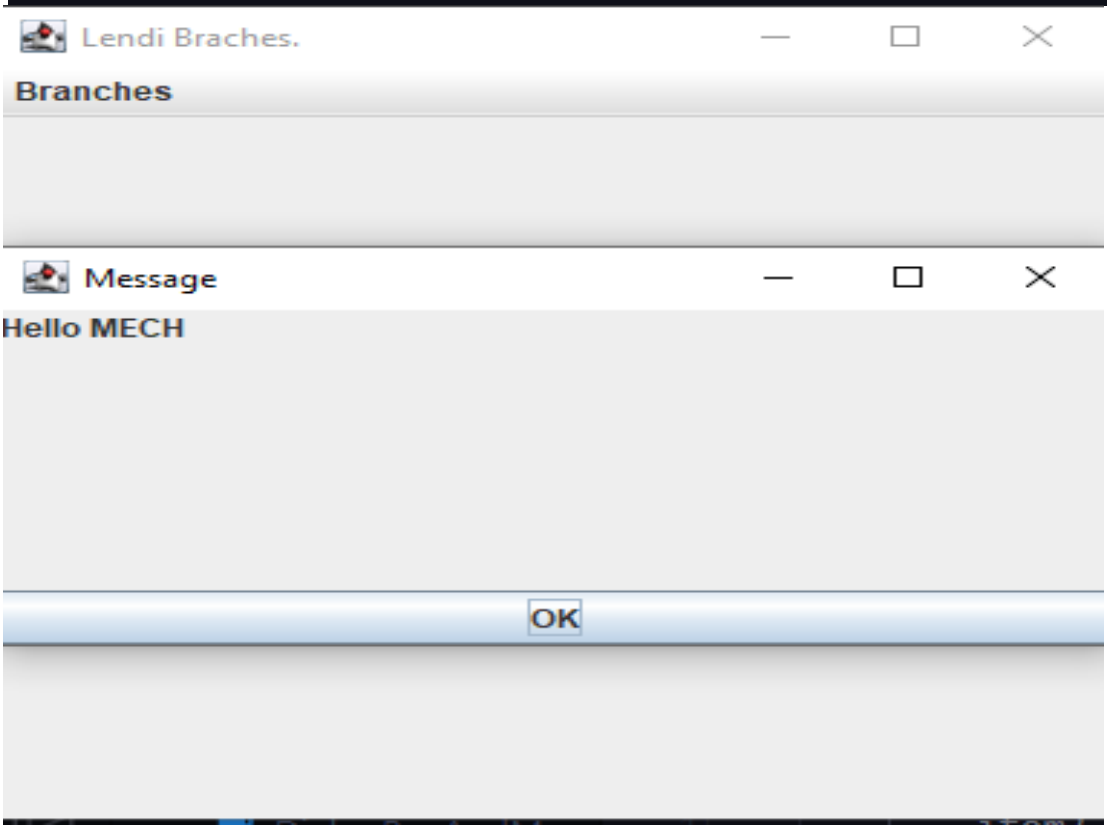



```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
public class DialogBoxAndMenu extends JFrame {
    JFrame jFrame;
    JMenu menu, submenu;
    JMenuItem item1, item2, item3, item4, item5;
    public DialogBoxAndMenu(){
        jFrame = new JFrame("Lendi Braches.");
        JMenuBar jmenubar = new JMenuBar();
        menu = new JMenu("Branches");
        submenu = new JMenu("CSE");
        item1 = new JMenuItem("ECE");
        item2 = new JMenuItem("EEE");
        item3 = new JMenuItem("MECH");
        item4 = new JMenuItem("CSE-A");
```

```
        item5 = new JMenuItem("CSE-B");
        menu.add(item1);
        menu.add(item2);
        menu.add(item3);
        submenu.add(item4);
        submenu.add(item5);
        menu.add(submenu);
        jmenubar.add(menu);


        item1.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                JFrame jFrame1 = new JFrame("Message");
                JLabel label = new JLabel();
                JButton ece = new JButton("OK");
                ece.setSize(50,50);
                label.setText("Hello ECE");
                jFrame1.add(label, BorderLayout.NORTH);
                jFrame1.add(ece, BorderLayout.SOUTH);
                jFrame1.setSize(400,200);
                jFrame1.setDefaultCloseOperation(EXIT_ON_CLOSE);
                jFrame1.setVisible(true);

            }
        });

        item2.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                JFrame jFrame1 = new JFrame("Message");
                JLabel label = new JLabel();
                JButton ece = new JButton("OK");
                ece.setSize(50,50);
                label.setText("Hello EEE");
                jFrame1.add(label, BorderLayout.NORTH);
                jFrame1.add(ece, BorderLayout.SOUTH);
                jFrame1.setSize(400,200);
                jFrame1.setDefaultCloseOperation(EXIT_ON_CLOSE);
                jFrame1.setVisible(true);

            }
        });

        item3.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
```

```
            JFrame jFrame1 = new JFrame("Message");
            JLabel label = new JLabel();
            JButton ece = new JButton("OK");
            ece.setSize(50,50);
            label.setText("Hello MECH");
            jFrame1.add(label, BorderLayout.NORTH);
            jFrame1.add(ece, BorderLayout.SOUTH);
            jFrame1.setSize(400,200);
            jFrame1.setDefaultCloseOperation(EXIT_ON_CLOSE);
            jFrame1.setVisible(true);

        }
    });

    item4.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            JFrame jFrame1 = new JFrame("Message");
            JLabel label = new JLabel();
            JButton ece = new JButton("OK");
            ece.setSize(50,50);
            label.setText("Hello CSE-A");
            jFrame1.add(label, BorderLayout.NORTH);
            jFrame1.add(ece, BorderLayout.SOUTH);
            jFrame1.setSize(400,200);
            jFrame1.setDefaultCloseOperation(EXIT_ON_CLOSE);
            jFrame1.setVisible(true);

        }
    });

    item5.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            JFrame jFrame1 = new JFrame("Message");
            JLabel label = new JLabel();
            JButton ece = new JButton("OK");
            ece.setSize(50,50);
            label.setText("Hello CSE-B");
            jFrame1.add(label, BorderLayout.NORTH);
            jFrame1.add(ece, BorderLayout.SOUTH);
            jFrame1.setSize(400,200);
            jFrame1.setDefaultCloseOperation(EXIT_ON_CLOSE);
            jFrame1.setVisible(true);

        }
    });
```

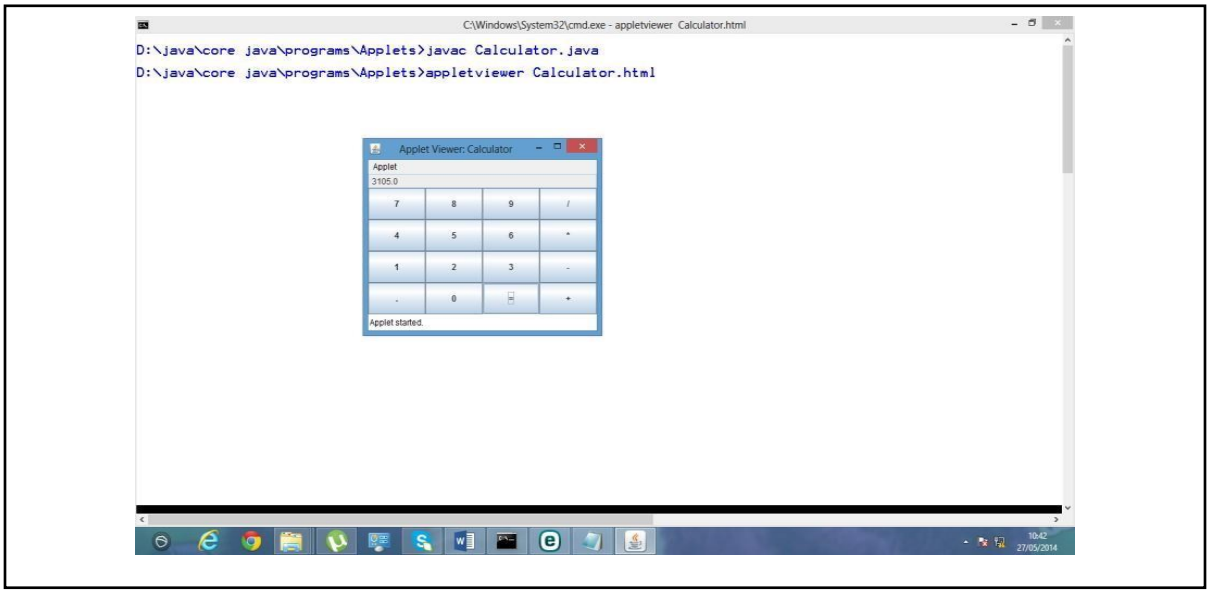
```
        jFrame.setJMenuBar(jmenubar);
        jFrame.setSize(400,400);
        jFrame.setLayout(null);
        jFrame.setDefaultCloseOperation(EXIT_ON_CLOSE);
        jFrame.setVisible(true);

    }
    public static void main(String[] args) {
        DialogBoxAndMenu dialogBoxAndMenu = new DialogBoxAndMenu();
    }
}
```



18. Write a java program to create a simple calculator.

DESCRIPTION:

This program is used to create a simple calculator having basic arithmetic operations like addition, subtraction, multiplication, division and modulo division. This calculator is having buttons and these buttons are created using Button() method. Also this having text field to display result.




```java
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.util.*;
import java.util.List;

class solveExpression {
    private String exp;
    protected int returnType = 0; /* 0, 1, 2 */
    protected String ans[] = {"Syntax Error !", "Math Error !", ""};

    private StringTokenizer element;
    private List<String> infix, prefix;

    protected solveExpression(String exp) {
        this.exp = exp;
        if (!error()) {
            infixToPrefix();
            calculatePrefix();
        }
    }

    protected String Answer() {
        return ans[returnType];
    }

    private boolean error() {
```

```
        element = new StringTokenizer(exp, "+-x/()*", true);
        infix = new ArrayList<String>();
        while (element.hasMoreElements())
            infix.add(new String(element.nextElement().toString()));

        for (int i = 0; i < infix.size(); i++) {
            String s = infix.get(i);
            if (s.equals("+") || s.equals("-") || s.equals("(")) {
                if (i == infix.size()-1) return true;
                String t = infix.get(i+1);
                  if (t.equals("x") || t.equals("/") || t.equals(")"))
return true;
                if (s.equals("+") || s.equals("-")) {
                    if (t.equals("+")) {
                        infix.remove(i+1);
                        i--;
                    }
                    else if (t.equals("-")) {
                        infix.set(i, s.equals("+") ? "-" : "+");
                        infix.remove(i+1);
                        i--;
                    }
                    else {
                        if (i == 0) {infix.add(i--, "0"); continue;}
                        String g = infix.get(i-1);
                        if (t.equals("(")) {
                            if (g.equals("x") || g.equals("/")) {
                                 infix.set(i, s.equals("+") ? "1" : "-
1");
                                infix.add(i+1, g);
                                i--;
                            }
                            else if (g.equals("(")) {
                                if (s.equals("+")) infix.remove(i--);
                                else infix.add(i--, "0");
                            }
                            continue;
                        }
                            if (g.equals("(") || g.equals("x") ||
g.equals("/")) {
                            if (s.equals("+")) infix.remove(i--);
                            else {
                                infix.remove(i);
                               StringBuilder num = new StringBuilder("-
");
                                num.append(infix.get(i));
                                infix.set(i, new String(num));
```

```
                                i--;
                            }
                        }
                    }
                }
            }
            else if (s.equals("x") || s.equals("/")) {
                if (i == 0 || i == infix.size()-1) return true;
                String t = infix.get(i+1);
                  if (t.equals("x") || t.equals("/") || t.equals(")"))
return true;
            }
            else if (s.equals(")")) {
                if (i == 0) return true;
                if (i == infix.size()-1) continue;
                String t = infix.get(i+1);
                 if (t.equals("+") || t.equals("-") || t.equals("x") ||
t.equals("/") || t.equals(")")) continue;
                if (t.equals("(")) {
                    infix.add(i+1, "x"); continue;
                }
                return true;
            }
            else {
                try {
                    double t = (double) Double.parseDouble(s);
                }
                catch (NumberFormatException e) {
                    return true;
                }
                if (i == infix.size()-1) continue;
                String t = infix.get(i+1);
                if (t.equals("(")) infix.add(i+1, "x");
            }
        }
        return checkOpenClose();
    }

    private boolean checkOpenClose() {
        int cnt = 0;
        for (String s : infix) {
            if (s.equals("(")) cnt++;
            else if (s.equals(")")) {
                if (cnt == 0) return true;
                cnt--;
            }
        }
```

```
        for (; cnt > 0; cnt--) infix.add(")");
        return cnt == 0 ? false : true;
    }

    private int rank(String s) {
        if (s.equals("x") || s.equals("/")) return 2;
        if (s.equals("+") || s.equals("-")) return 1;
        return 0;
    }

    private void infixToPrefix() {
        Collections.reverse(infix);
        Deque<String> st = new LinkedList<String>();
        prefix = new ArrayList<String>();

        for (String t : infix) {
              if (t.equals("+") || t.equals("-") || t.equals("x") ||
t.equals("/")) {
                while (!st.isEmpty() && rank(st.peek()) > rank(t)) {
                    prefix.add(st.pop());
                }
                st.push(t);
            }
            else if (t.equals(")")) {
                st.push(t);
            }
            else if (t.equals("(")) {
                while (!st.isEmpty() && !st.peek().equals(")")) {
                    prefix.add(st.pop());
                }
                st.pop();
            }
            else {
                prefix.add(t);
            }
        }
        while (!st.isEmpty()) prefix.add(st.pop());
        Collections.reverse(prefix);
    }

    private void calculatePrefix() {
        Collections.reverse(prefix);
        Deque<Double> st = new LinkedList<Double>();
        for (String t : prefix) {
              if (t.equals("+") || t.equals("-") || t.equals("x") ||
t.equals("/")) {
                double a = st.pop(), b = st.pop();
```

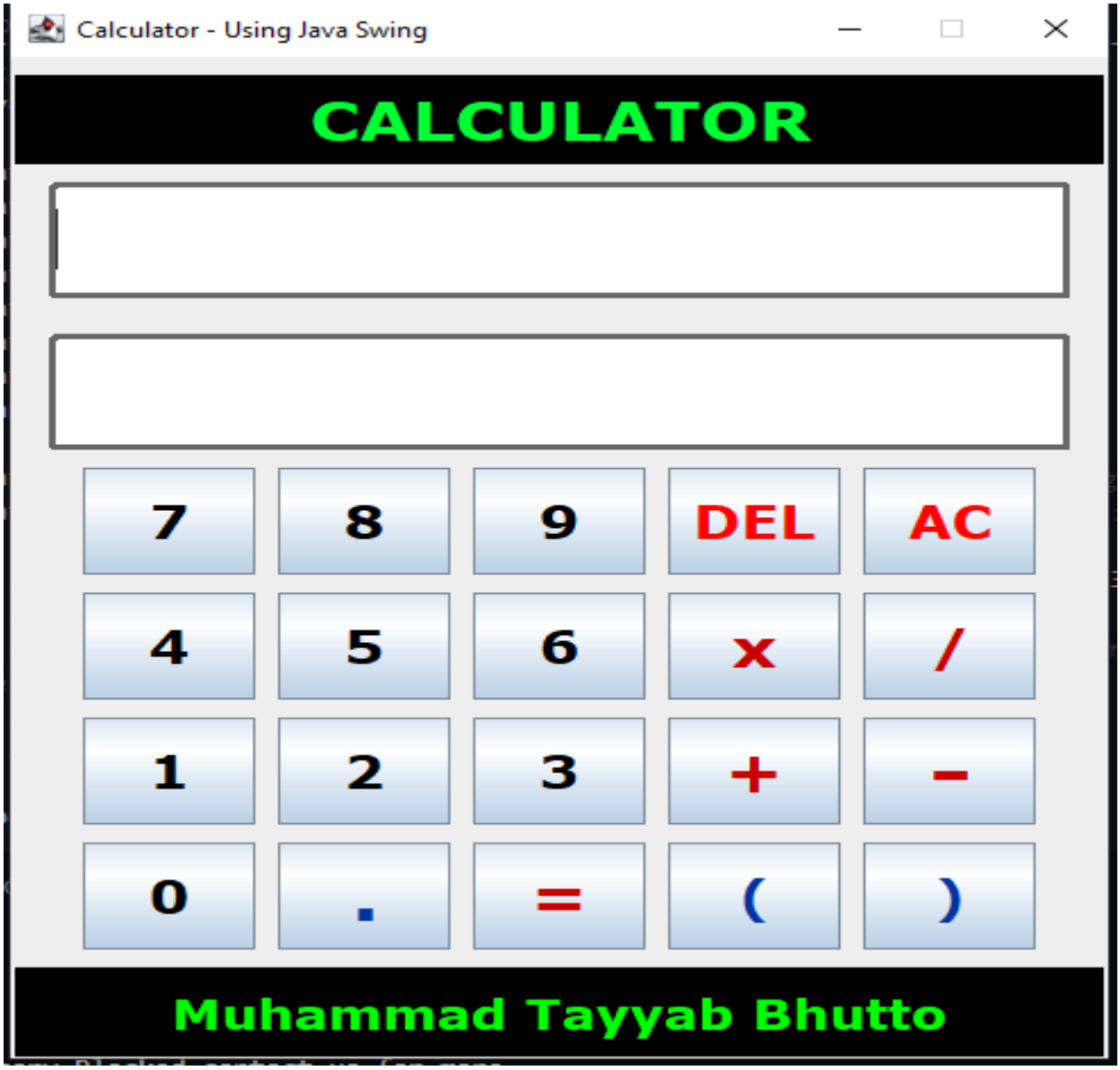
```
                if (t.equals("+")) a += b;
                else if (t.equals("-")) a -= b;
                else if (t.equals("x")) a *= b;

                else {
                    if (Math.abs(b) <= Double.MIN_NORMAL) {
                        returnType = 1;
                        return ;
                    }
                    a /= b;
                }
                st.push(a);
            }
            else {
                st.push(Double.parseDouble(t));
            }
        }
        returnType = 2;
        ans[2] = st.pop().toString();
    }
}
public class SimpleCalculator extends JFrame {

    public SimpleCalculator() {
        this.initComponent();
        this.pack();
        this.setLocationRelativeTo(null);
        this.setVisible(true);
    }

    private void initComponent() {
        labelCalculator = new JLabel();
        ScreenUserInterface = new JPanel();
        screenInput = new JTextField();
        screenOutput = new JTextField();
        ButtonUserInterface = new JPanel();
        copyright = new JLabel();

         setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_C
LOSE);
        setAutoRequestFocus(false);
        setBackground(new java.awt.Color(255, 255, 255));
        setPreferredSize(new java.awt.Dimension(500, 600));
        setResizable(false);
                                          getContentPane().setLayout(new
java.awt.FlowLayout(java.awt.FlowLayout.CENTER, 0, 10));
        setTitle("Calculator - Using Java Swing ");
```

```
        setFocusTraversalKeysEnabled(false);

        labelCalculator.setBackground(new Color(0, 0, 0));
        labelCalculator.setFont(new Font("Verdana", 1, 30));
        labelCalculator.setForeground(new Color(0, 255, 51));
        labelCalculator.setHorizontalAlignment(SwingConstants.CENTER);
        labelCalculator.setText("CALCULATOR");
        labelCalculator.setOpaque(true);
        labelCalculator.setPreferredSize(new Dimension(480, 50));
        getContentPane().add(labelCalculator);

       ScreenUserInterface.setPreferredSize(new java.awt.Dimension(450,
150));
        ScreenUserInterface.setLayout(new java.awt.GridLayout(2, 0, 0,
20));

        screenInput.setFont(new java.awt.Font("Courier New", Font.BOLD,
30));
         screenInput.setHorizontalAlignment(javax.swing.JTextField.LEFT
);
           screenInput.setBorder(new  javax.swing.border.LineBorder(new
java.awt.Color(102, 102, 102), 3, true));
        screenInput.setSelectedTextColor(Color.WHITE);
        screenInput.setSelectionColor(Color.DARK_GRAY);

        addScreenInputEvent();

        screenOutput.setEditable(false);
       screenOutput.setFont(new java.awt.Font("Courier New", Font.BOLD,
30));
        screenOutput.setForeground(new java.awt.Color(0, 51, 204));
        screenOutput.setOpaque(true);
        screenOutput.setBackground(Color.WHITE);
         screenOutput.setHorizontalAlignment(javax.swing.JTextField.RIG
HT);
          screenOutput.setBorder(new  javax.swing.border.LineBorder(new
java.awt.Color(102, 102, 102), 3, true));
        screenOutput.setSelectedTextColor(Color.WHITE);
        screenOutput.setSelectionColor(Color.DARK_GRAY);

        ScreenUserInterface.add(screenInput);
        ScreenUserInterface.add(screenOutput);
        getContentPane().add(ScreenUserInterface);

       ButtonUserInterface.setPreferredSize(new java.awt.Dimension(420,
270));
```

```
        ButtonUserInterface.setLayout(new java.awt.GridLayout(0, 5, 10,
10));

        for (int i = 0; i < stringButton.length; i++) {
            b[i] = new JButton(stringButton[i]);
            b[i].setFont(new java.awt.Font("Verdana", 1, 25));
            b[i].setMargin(new java.awt.Insets(0, 0, 0, 0));
            b[i].setForeground(Color.black);
        }

        b[10].setForeground(new Color(0,50,162));
        b[11].setForeground(new Color(0,50,162));
        b[12].setForeground(new Color(0,50,162));
        b[19].setForeground(new Color(200, 0, 0));
         for(int i=13;i<=16;i++) b[i].setForeground(new Color(200, 0,
0));
        b[17].setForeground(Color.red);
        b[18].setForeground(Color.red);
        b[10].setFont(new java.awt.Font("Verdana", 1, 35));
        b[13].setFont(new java.awt.Font("Verdana", 1, 30));
        b[14].setFont(new java.awt.Font("Verdana", 1, 40));
        b[15].setFont(new java.awt.Font("Verdana", 1, 30));
        b[19].setFont(new java.awt.Font("Verdana", 1, 30));

        addButtonEvent();

              for (int i = 0; i < stringButton.length; i++)
ButtonUserInterface.add(b[orderButtonDisplay[i]]);
        getContentPane().add(ButtonUserInterface);

        copyright.setBackground(new Color(0, 0, 0));
        copyright.setFont(new Font("Verdana", 1, 23));
        copyright.setForeground(new Color(0, 255, 0));
        copyright.setHorizontalAlignment(SwingConstants.CENTER);
        copyright.setText("Muhammad Tayyab Bhutto");
        copyright.setOpaque(true);
        copyright.setPreferredSize(new Dimension(480, 50));
        getContentPane().add(copyright);
    }

    private void updateScreenInput(String add) {
        StringBuilder cur = new StringBuilder(screenInput.getText());
        int pos = screenInput.getCaretPosition();
        cur.insert(pos, add);
        screenInput.setText(new String(cur));
        screenInput.setCaretPosition(pos+add.length());
    }
```

```
    private void addScreenInputEvent() {
        screenInput.addKeyListener(new KeyAdapter() {
            @Override
            public void keyTyped(KeyEvent e) {
                e.consume();
                char c = e.getKeyChar();
                String sc = Character.toString(c);
                for (int i = 0; i < 17; i++) {
                    if (sc.equals(stringButton[i])) {
                        updateScreenInput(sc); break;
                    }
                }
                if (c == '*') updateScreenInput("x");
                if (c == '=' || c == KeyEvent.VK_ENTER) {
                    String exp = screenInput.getText();
                                                screenOutput.setText(new
solveExpression(exp).Answer());
                }
            }
        });
    }

    private void addButtonEvent() {
        for (int i = 0; i <= 16; i++) {
            String g = stringButton[i];
            b[i].addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    updateScreenInput(g);
                }
            });
        }

        b[17].addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                screenInput.setText("");
                screenOutput.setText("");
            }
        });

        b[18].addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String s = screenInput.getText();
                if (s.length() == 0) return ;
```

```
                int pos = screenInput.getCaretPosition();
                if (pos == 0) return ;
                String t = s.substring(0, pos-1);
                          screenInput.setText(t.concat(s.substring(pos,
s.length())));
                screenInput.setCaretPosition(pos-1);
            }
        });

        b[19].addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String exp = screenInput.getText();
               screenOutput.setText(new solveExpression(exp).Answer());
            }
        });
    }

    /* Variables declaration*/
    private javax.swing.JPanel ButtonUserInterface;
    private javax.swing.JPanel ScreenUserInterface;
    private javax.swing.JLabel labelCalculator;
    private javax.swing.JLabel copyright;
    private javax.swing.JTextField screenInput;
    private javax.swing.JTextField screenOutput;
                    private        String        stringButton[]        =
{"0","1","2","3","4","5","6","7","8","9",
                                                        ".","(",")","+","-
","x","/","AC","DEL","="};
    private JButton b[] = new JButton[stringButton.length];
    private int orderButtonDisplay[] = {
        7,  8,  9,  18, 17,
        4,  5,  6,  15, 16,
        1,  2,  3,  13, 14,
        0,  10, 19, 11, 12,
    };

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



19. Basic tasks for classes

Download the recommended textbook named “Java: How to Program by Deital and Deital”. Go to page 441 and write the codes for the following exercises

8.3 8.8 8.9 8.15

Static
8.16 8.17

```
abstract class Employee {
    private String firstName;
    private String lastName;
    private String socialSecurityNumber;

    // three-argument constructor
    public Employee(String first, String last, String ssn) {
        firstName = first;
        lastName = last;
        socialSecurityNumber = ssn;
    } // end three-argument Employee constructor
      // set first name

    public void setFirstName(String first) {
        firstName = first; // should validate
    } // end method setFirstName
      // return first name

    public String getFirstName() {
        return firstName;
    } // end method getFirstName
      // set last name

    public void setLastName(String last) {
        lastName = last; // should validate
    } // end method setLastName
      // return last name

    public String getLastName() {
        return lastName;
    } // end method getLastName
      // set social security number

    public void setSocialSecurityNumber(String ssn) {
        socialSecurityNumber = ssn; // should validate
    } // end method setSocialSecurityNumber
      // return social security number

    public String getSocialSecurityNumber() {
        return socialSecurityNumber;
    } // end method getSocialSecurityNumber
      // return String representation of Employee object

    @Override
```

```
    public String toString() {
        return String.format("%s %s\nsocial security number: %s",
getFirstName(), getLastName(), getSocialSecurityNumber());
    } // end method toString
      // abstract method overridden by concrete subclasses

    public abstract double earnings(); // no implementation here
} // end abstract class Employee

class SalariedEmployee extends Employee {
    private double weeklySalary;
    double baseSalary;
    // four-argument constructor
    public SalariedEmployee(String first, String last, String ssn, double
salary) {
        super(first, last, ssn); // pass to Employee constructor
        setWeeklySalary(salary); // validate and store salary
    } // end four-argument SalariedEmployee constructor

    // set salary
    public void setWeeklySalary(double salary) {
        if (salary >= 0.0) {
            baseSalary = salary;
        } else {
            throw new IllegalArgumentException("Weekly salary must be");
        }
    }

    // calculate earnings; override abstract method earnings in Employee
    // return salary
    public double getWeeklySalary() {
        return weeklySalary;
    } // end method getWeeklySalary

    @Override
    public double earnings() {
        return getWeeklySalary();
    } // end method earnings
      // return String representation of SalariedEmployee object

    @Override
    public String toString() {
        return String.format("salaried employee: %s\n%s: $%,.2f",
                super.toString(), "weekly salary", getWeeklySalary());
    } // end method toString
} // end class SalariedEmployee
```

```java
class HourlyEmployee extends Employee {
    private double wage; // wage per hour
    private double hours; // hours worked for week
    // five-argument constructor

    public HourlyEmployee(String first, String last, String ssn,
            double hourlyWage, double hoursWorked) {
        super(first, last, ssn);
        setWage(hourlyWage); // validate hourly wage
        setHours(hoursWorked); // validate hours worked
    } // end five-argument HourlyEmployee constructor
      // set wage

    public void setWage(double hourlyWage) {
        if (hourlyWage >= 0.0) {
            wage = hourlyWage;
        } else {
            throw new IllegalArgumentException("Hourly wage must be >= 0.0");
        }
    } // end method setWage
      // return wage

    public double getWage() {
        return wage;
    } // end method getWage
      // set hours worked

    public void setHours(double hoursWorked) {
        if ((hoursWorked >= 0.0) && (hoursWorked <= 168.0)) {
            hours = hoursWorked;
        } else {
            throw new IllegalArgumentException("Hours worked must be >= 0.0 and
<= 168.0");
        }
    } // end method setHours
      // return hours worked

    public double getHours() {
        return hours;
    } // end method getHours
      // calculate earnings; override abstract method earnings in Employee

    @Override
    public double earnings() {
        if (getHours() <= 40) // no overtime
        {
            return getWage() * getHours();
```

```
        } else {
            return 40 * getWage() + (getHours() - 40) * getWage() * 1.5;
        }
    } // end method earnings
      // return String representation of HourlyEmployee object

    @Override
    public String toString() {
        return String.format("hourly employee: %s\n%s: $%,.2f; %s: %,.2f",
                super.toString(), "hourly wage", getWage(),
                "hours worked", getHours());
    } // end method toString
}// end class HourlyEmployee

class CommissionEmployee extends Employee {
    private double grossSales; // gross weekly sales
    private double commissionRate; // commission percentage
    // five-argument constructor

    public CommissionEmployee(String first, String last, String ssn,
            double sales, double rate) {
        super(first, last, ssn);
        setGrossSales(sales);
        setCommissionRate(rate);
    } // end five-argument CommissionEmployee constructor
      // set commission rate

    public void setCommissionRate(double rate) {
        if (rate > 0.0 && rate < 1.0) {
            commissionRate = rate;
        } else {
            throw new IllegalArgumentException("Commission rate must be > 0.0
and < 1.0");
        }
    } // end method setCommissionRate
      // return commission rate

    public double getCommissionRate() {
        return commissionRate;
    } // end method getCommissionRate
      // set gross sales amount

    public void setGrossSales(double sales) {
        if (sales >= 0.0) {
            grossSales = sales;
        } else {
            throw new IllegalArgumentException("Gross sales must be >= 0.0");
```

```
        }
    } // end method setGrossSales
      // return gross sales amount

    public double getGrossSales() {
        return grossSales;
    } // end method getGrossSales
      // calculate earnings; override abstract method earnings in Employee

    @Override
    public double earnings() {
        return getCommissionRate() * getGrossSales();
    } // end method earnings
      // return String representation of CommissionEmployee object

    @Override
    public String toString() {
        return String.format("%s: %s\n%s: $%,.2f; %s: %.2f", "commission
employee", super.toString(), "gross sales",
                getGrossSales(), "commission rate", getCommissionRate());
    } // end method toString
} // end class CommissionEmployee

class BasePlusCommissionEmployee extends CommissionEmployee {
    private double baseSalary; // base salary per week
    // six-argument constructor

    public BasePlusCommissionEmployee(String first, String last,
            String ssn, double sales, double rate, double salary) {
        super(first, last, ssn, sales, rate);
        setBaseSalary(salary); // validate and store base salary
    } // end six-argument BasePlusCommissionEmployee constructor
      // set base salary

    public void setBaseSalary(double salary) {
        if (salary >= 0.0) {
            baseSalary = salary;
        } else {
            throw new IllegalArgumentException("Base salary must be >= 0.0");
        }
    } // end method setBaseSalary
      // return base salary

    public double getBaseSalary() {
        return baseSalary;
    } // end method getBaseSalary
      // calculate earnings; override method earnings in CommissionEmployee
```

```
    @Override
    public double earnings() {
        return getBaseSalary() + super.earnings();
    } // end method earnings
      // return String representation of BasePlusCommissionEmployee object

    @Override
    public String toString() {
        return String.format("%s %s; %s: $%,.2f", "base-salaried",
super.toString(), "base salary", getBaseSalary());
    } // end method toString
} // end class BasePlusCommissionEmployee

public class PayrollSystemTest {
    public static void main(String[] args) {
        // create subclass objects
        SalariedEmployee salariedEmployee = new SalariedEmployee("Muhammad
Tayyab", "Bhutto", "111-11-1111", 800.00);
        HourlyEmployee hourlyEmployee = new HourlyEmployee("Tanveer", "Ahmed",
"222-22-2222", 16.75, 40);
        CommissionEmployee commissionEmployee = new CommissionEmployee(
                "Muhammad", "Muzammil", "333-33-3333", 10000, .06);
        BasePlusCommissionEmployee basePlusCommissionEmployee = new
BasePlusCommissionEmployee(
                "Amjad", "Umar", "444-44-4444", 5000, .04, 300);
        System.out.println("Employees processed individually:\n");

        System.out.printf("%s\n%s: $%,.2f\n\n",
                salariedEmployee, "earned", salariedEmployee.earnings());
        System.out.printf("%s\n%s: $%,.2f\n\n",
                hourlyEmployee, "earned", hourlyEmployee.earnings());
        System.out.printf("%s\n%s: $%,.2f\n\n",
                commissionEmployee, "earned", commissionEmployee.earnings());
        System.out.printf("%s\n%s: $%,.2f\n\n",
                basePlusCommissionEmployee, "earned",
basePlusCommissionEmployee.earnings());

        // create four-element Employee array
        Employee[] employees = new Employee[4];
        // initialize array with Employees
        employees[0] = salariedEmployee;
        employees[1] = hourlyEmployee;
        employees[2] = commissionEmployee;
        employees[3] = basePlusCommissionEmployee;
        System.out.println("Employees processed polymorphically:\n");
        // generically process each element in array employees
```

```
        for (Employee currentEmployee : employees) {
            System.out.println(); // invokes toString
            // determine whether element is a BasePlusCommissionEmployee
            if (currentEmployee instanceof BasePlusCommissionEmployee) {
                // downcast Employee reference to
                // BasePlusCommissionEmployee reference
                BasePlusCommissionEmployee employee =
(BasePlusCommissionEmployee) currentEmployee;
                employee.setBaseSalary(1.10 * employee.getBaseSalary());
                System.out.printf("new base salary with 10%% increase is:
$%,.2f\n", employee.getBaseSalary());
            } // end if
            System.out.printf("earned $%,.2f\n\n", currentEmployee.earnings());
        } // end for
          // get type name of each object in employees array
        for (int j = 0; j < employees.length; j++) {
            System.out.printf("Employee %d is a %s\n", j,
employees[j].getClass().getName());
        }

    }
}
```

```
Employees processed individually:

salaried employee: Muhammad Tayyab Bhutto
social security number: 111-11-1111
weekly salary: $0.00
earned: $0.00

hourly employee: Tanveer Ahmed
social security number: 222-22-2222
hourly wage: $16.75; hours worked: 40.00
earned: $670.00

commission employee: Muhammad Muzammil
social security number: 333-33-3333
gross sales: $10,000.00; commission rate: 0.06
earned: $600.00

base-salaried commission employee: Amjad Umar
social security number: 444-44-4444
gross sales: $5,000.00; commission rate: 0.04; base salary: $300.00
earned: $500.00

Employees processed polymorphically:


earned $0.00


earned $670.00


earned $600.00


new base salary with 10% increase is: $330.00
earned $530.00
```

20. Inheritance exercises

1. A Bank

Look at the `Account` class `Account.java` and write a `main` method in a different class to briefly experiment with some instances of the `Account` class.

- Using the `Account` class as a base class, write two derived classes called `SavingsAccount` and `CurrentAccount`. A `SavingsAccount` object, in addition to the attributes of an `Account` object, should have an interest variable and a method which adds interest to the account. A `CurrentAccount` object, in addition to the attributes of an `Account` object, should have an overdraft limit variable. Ensure that you have overridden methods of the

`Account` class as necessary in both derived classes. ○ Now create a `Bank` class, an object of which contains an array of `Account` objects. Accounts in the array could be instances of

the `Account` class, the `SavingsAccount` class, or the `CurrentAccount` class. Create some test accounts (some of each type). ○ Write an update method in the bank class. It iterates through each account, updating it in the following ways: Savings accounts get interest added (via the method you already wrote); CurrentAccounts get a letter sent if they are in overdraft.

- The `Bank` class requires methods for opening and closing accounts, and for paying a dividend into each account.

Hints: ○ Note that the balance of an account may only be modified

through the `deposit(double)` and `withdraw(double)` methods.

- The `Account` class should not need to be modified at all.
- Be sure to test what you have done after each step.

```
class Account {
    private int accountNumber; // account number
    private int pin; // PIN for authentication
    private double availableBalance; // funds available for withdrawal
    private double totalBalance; // funds available + pending deposits
    // Account constructor initializes attributes

    public Account(int theAccountNumber, int thePIN, double
theAvailableBalance, double theTotalBalance) {
        accountNumber = theAccountNumber;
        pin = thePIN;
        availableBalance = theAvailableBalance;
        totalBalance = theTotalBalance;
    } // end Account constructor
      // determines whether a user-specified PIN matches PIN in Account

    public boolean validatePIN(int userPIN) {
        if (userPIN == pin)
            return true;
        else
            return false;
    } // end method validatePIN
      // returns available balance

    public double getAvailableBalance() {
        return availableBalance;
    } // end getAvailableBalance
      // returns the total balance
```

```
      public void setTotalBalance(double totalBalance) {
          this.totalBalance = totalBalance;
      }
      public double getTotalBalance() {
          return totalBalance;
      } // end method getTotalBalance
        // credits an amount to the account

      public void credit(double amount) {
          totalBalance += amount; // add to total balance
      } // end method credit
        // debits an amount from the account

      public void debit(double amount) {
          availableBalance -= amount; // subtract from available balance
          totalBalance -= amount; // subtract from total balance
      } // end method debit
        // returns account number

      public int getAccountNumber() {
          return accountNumber;
      } // end method getAccountNumber
      public String toString(){
          return "Account Number: " + accountNumber + " Availble Balance: "
+ availableBalance + " Total Balance: " + totalBalance;
      }
  } // end class Account

  class SavingsAccount extends Account {
      private double interest;
      private int year;
      public SavingsAccount(int theAccountNumber, int thePIN, double
theAvailableBalance, double theTotalBalance, int year){
          super(theAccountNumber, thePIN, theAvailableBalance,
theTotalBalance);
          this.year = year;
      }
      public void getInterest(){
          double principal = 1000.0; // initial amount before interest
          double rate = 0.05; // interest rate
          // calculate amount on deposit for each of ten years
          // calculate new amount for specified year
          interest = principal * Math.pow( 1.0 + rate, year );
          // display the year and the amount
          System.out.println( "Year: "+ year+ " Interest: "+ interest);
      }
      public String toString(){
```

```
            return super.toString() + " Interest: " + interest + " Years: " +
year;
        }
    }

    class CurrentAccount extends Account {
        private double newAmount;

        public CurrentAccount(int theAccountNumber, int thePIN, double
theAvailableBalance, double theTotalBalance, double newAmount) {
            super(theAccountNumber, thePIN, theAvailableBalance,
theTotalBalance);
            this.newAmount = newAmount;
        }
        public void setNewAmount(double newAmount){
            this.newAmount = newAmount;
        }
        public void getOverDraft(){
            if (newAmount > 100000){
                System.out.println("Overdraft");
            } else {
                setTotalBalance(newAmount+getTotalBalance());
                System.out.println("Added To Your Balance. " +
getTotalBalance());
            }
        }
        public String toString(){
            return super.toString() + " Current Amount: " + newAmount;
        }
    }

    class Bank {
        private Account[] accounts; // array of Accounts

        public Bank() {
            accounts = new Account[2]; // just 2 accounts for testing
            accounts[0] = new Account(12345, 54321, 1000.0, 12000);
            accounts[1] = new Account(98765, 56789, 200.0, 200.0);
        } // end no-argument BankDatabase constructor
          // retrieve Account object containing specified account number

        private Account getAccount(int accountNumber) {
            // loop through accounts searching for matching account number
            for (Account currentAccount : accounts) {
                // return current account if match found
                if (currentAccount.getAccountNumber() == accountNumber)
                    return currentAccount;
```

```
            } // end for
            return null; // if no matching account was found, return null
        } // end method getAccount
          // determine whether user-specified account number and PIN match
          // those of an account in the database

        public boolean authenticateUser(int userAccountNumber, int userPIN) {
            // attempt to retrieve the account with the account number
            Account userAccount = getAccount(userAccountNumber);
            // if account exists, return result of Account method validatePIN
            if (userAccount != null)
                return userAccount.validatePIN(userPIN);
            else
                return false; // account number not found, so return false
        } // end method authenticateUser
          // return available balance of Account with specified account number

        public double getAvailableBalance(int userAccountNumber) {
            return getAccount(userAccountNumber).getAvailableBalance();
        } // end method getAvailableBalance
          // return total balance of Account with specified account number

        public double getTotalBalance(int userAccountNumber) {
            return getAccount(userAccountNumber).getTotalBalance();
        } // end method getTotalBalance
          // credit an amount to Account with specified account number

        public void credit(int userAccountNumber, double amount) {
            getAccount(userAccountNumber).credit(amount);
        } // end method credit

        public void debit(int userAccountNumber, double amount) {
            getAccount(userAccountNumber).debit(amount);
        } // end method debit
    } // end class BankDatabase

    public class BankMain {
        public static void main(String[] args) {
            Account accountDetails = new Account(12345,4444, 10000,50000);
            SavingsAccount savingsAccount = new SavingsAccount(12345,4444,
10000,50000,5);
            CurrentAccount currentAccount = new CurrentAccount(12345,4444,
10000,50000, 5000);
            Bank bank = new Bank();
            savingsAccount.getInterest();
            currentAccount.getOverDraft();
            System.out.println(accountDetails.toString());
```

```
            System.out.println(savingsAccount.toString());
            System.out.println(currentAccount.toString());

        }
    }
```

```
Year: 5 Interest: 1276.2815625000003
Added To Your Balance. 55000.0
Account Number: 12345 Availble Balance: 10000.0 Total Balance: 50000.0
Account Number: 12345 Availble Balance: 10000.0 Total Balance: 50000.0 Interest: 1276.2815625000003 Years: 5
Account Number: 12345 Availble Balance: 10000.0 Total Balance: 55000.0 Current Amount: 5000.0
PS F:\docs\Java\Mega Assignment>
```

2. Employees

Create a class called `Employee` whose objects are records for an employee. This class will be a derived class of the class `Person` which you will have to copy into a file of your own and compile. An employee record has an employee's name (inherited from the class `Person`), an annual salary represented as a single value of type `double`, a year the employee started work as a single value of type `int` and a national insurance number, which is a value of type `String`.

Your class should have a reasonable number of constructors and accessor methods, as well as an `equals` method. Write another class containing a `main` method to fully test your class definition.

```java
class Person {
    private String firstName;
    private String lastName;
    public Person(){}   // default constructor
    // two argument constructor
    public Person(String firstName, String lastName){
        this.firstName = firstName;
        this.lastName = lastName;
    }
    public void setFirstName(String first) {
        firstName = first; // should validate
    } // end method setFirstName
      // return first name

    public String getFirstName() {
        return firstName;
    } // end method getFirstName
```

```
      // set last name

    public void setLastName(String last) {
        lastName = last; // should validate
    } // end method setLastName
      // return last name

    public String getLastName() {
        return lastName;
    } // end method getLastName
      // set social security number

    public String toString(){
        return "First Name: " + firstName + " Last Name: " + lastName;
    }
}
class Employee extends Person {
    private double anualSalary;
    private int startedYear;
    private String insuranceNumber;
    public Employee(){} // default constructor
    // five argument constructor
    public Employee(String firstName, String lastName, double anualSalary, int
startedYear, String insuranceNumber){
        super(firstName, lastName);
        this.anualSalary = anualSalary;
        this.startedYear = startedYear;
        this.insuranceNumber = insuranceNumber;
    }
    public void setAnualSalary(double anualSalary) {
        this.anualSalary = anualSalary; // should validate
    } // end method setAnualSalary
      // return Anual Salary

    public double getAnualSalary() {
        return anualSalary;
    } // end method getAnualSalary
      // set started year

    public void setStartedYear(int startedYear) {
        this.startedYear = startedYear;// should validate
    } // end method setLastName
      // return Started Year

    public int getStartedYear() {
        return startedYear;
    } // end method getStartedYear
```

```java
    public void setInsuranceNumber(String insuranceNumber) {
        this.insuranceNumber = insuranceNumber;// should validate
    } // end method setInsuranceNumber
      // return Insurance Number

    public String getInsuranceNumber() {
        return insuranceNumber;
    } // end method getInsuranceNumber
    public boolean isEqual(Employee e1, Employee e2){
        if (e1.insuranceNumber.equals(e2.insuranceNumber)){
            return true;
        }
        return false;
    }
    public String toString(){
        return super.toString() + " Anual Salary: " + anualSalary + " Started
Year: " + startedYear + " Insurance Number: " + insuranceNumber;
    }
}
public class EmployeeMain {
    public static void main(String[] args) {
        Person person1 = new Person("Muhammad Tayyab", "Bhutto");
        Person person2 = new Person();
        Employee employee1 = new Employee("Muhammad Tayyab", "Bhutto", 5000000,
2020, "pk20-0020-2003-3892");
        Employee employee2 = new Employee();
        person2.setFirstName("Faheem");
        person2.setLastName("Akhtar");
        employee2.setFirstName("Faheem Akhtar");
        employee2.setLastName("Rajput");
        employee2.setStartedYear(2002);
        employee2.setAnualSalary(20000000);
        employee2.setInsuranceNumber("pk20-0020-2003-4455");
        System.out.println("Person 01 " + person1.toString());
        System.out.println("Person 02 " + person2.toString());
        System.out.println("Employee 01 " + employee1.toString());
        System.out.println("Employee 02 " + employee2.toString());
    }
}
```

```
Person 01 First Name: Muhammad Tayyab Last Name: Bhutto
Person 02 First Name: Faheem Last Name: Akhtar
Employee 01 First Name: Muhammad Tayyab Last Name: Bhutto Anual Salary: 5000000.0 Started Year: 2020 Insurance Number: pk20-0020-2003-3892
Employee 02 First Name: Faheem Akhtar Last Name: Rajput Anual Salary: 2.0E7 Started Year: 2002 Insurance Number: pk20-0020-2003-4455
PS F:\docs\Java\Mega Assignment>
```

BEST WISHES