Software Engineering Lab (CS29006)

Assignment 2: Java Programming

Time: 2:00 pm - 4:55 pm **Date : 30/01/2019**

Instructions for submission

- Give meaningful comments to explain the functionality of each class and function used in your program.
- Make a zip file with and give the name of the zip file as **A2_ < YourRollNo >**.
- The zip file should contain codes for the practice problems and the assignments.
- Submit the zip file to the Moodle system.
- Submit your solution latest by **4:55 pm** on **30/01/2019**.

---

1 Problems for Practice

1.1 Working with Stack

Type the following code and fix any error that might be present in the code.

```
import java.util.*;

public class StackDemo{
  public static void main(String[] args) {
    Stack stack=new Stack();
    stack.push(new Integer(10));
    stack.push("a");
    System.out.println("The contents of Stack is" + stack);
    System.out.println("The size of an Stack is" + stack.size());
    System.out.println("The number poped out is" + stack.pop());
    System.out.println("The number poped out is " + stack.pop());
    //System.out.println("The number poped out is" + stack.pop());
    System.out.println("The contents of stack is" + stack);
    System.out.println("The size of an stack is" + stack.size());
  }
}
```

1.2 Access Modifiers

Type the following code. Some statements in this class may cause compilation error. Find them and comment them out.

```
class BaseClass {
  public int x = 10;
  private int y = 10;
  protected int z = 10;
  int a = 10; //Implicit Default Access Modifier

  public int getX() {
    return x;
  }

  public void setX(int x) {
    this.x = x;
  }

  private int getY() {
    return y;
  }

  private void setY(int y) {
    this.y = y;
  }

  protected int getZ() {
    return z;
  }

  protected void setZ(int z) {
    this.z = z;
  }

  int getA() {
    return a;
  }

  void setA(int a) {
    this.a = a;
  }
}
```

```
public class SubclassInSamePackage extends BaseClass {
  public static void main(String args[]) {
    BaseClass rr = new BaseClass();
    rr.z = 0;
    SubclassInSamePackage subClassObj = new SubclassInSamePackage();
    //Access Modifiers - Public
    System.out.println("Value of x is : " + subClassObj.x);
    subClassObj.setX(20);
    System.out.println("Value of x is : " + subClassObj.x);
       System.out.println("Value of y is : "+subClassObj.y);
    subClassObj.setY(20);
    System.out.println("Value of y is : "+subClassObj.y);

    //Access Modifiers - Protected
    System.out.println("Value of z is : " + subClassObj.z);
    subClassObj.setZ(30);
    System.out.println("Value of z is : " + subClassObj.z);
    //Access Modifiers - Default
    System.out.println("Value of x is : " + subClassObj.a);
    subClassObj.setA(20);
    System.out.println("Value of x is : " + subClassObj.a);
  }
}
```

1.3 Interfaces and Inheritance

Type the following code. Some statements from the class *B1* has been removed. Complete them so that the resulting code compiles.

```
interface I1 {
  void methodI1(); //public static by default
}

interface I2 extends I1 {
  void methodI2(); //public static by default
}



class A1 {
  public String methodA1() {
     String strA1 = "I am in methodC1 of class A1";
     return strA1;
  }

  public String toString() {
     return "toString() method of class A1";
  }
}

class B1 extends A1 implements I2 {

     System.out.println("I am in methodI1 of class B1");
  }


     System.out.println("I am in methodI2 of class B1");
  }
```

1.4 Exception Handling

Type the following code and check the run-time error.

```
1 public class ExceptionHandlingExample {
2     public static void main(String[] args) {
3         int divisor = 0;
4         int dividend = 11;
5
6         System.out.println("The result is: " + dividend / divisor);
7     }
8 }
```

Type the following code and check the output. Change the divisor to 1 and check the output.

```
 1 public class ExceptionHandlingExample {
 2     public static void main(String[] args) {
 3         int divisor = 0;
 4         int dividend = 11;
 5
 6         try {
 7             int result = dividend / divisor;
 8             System.out.println("The result is: " + result);
 9         } catch (ArithmeticException ae) {
10             System.out.println("Division by zero!");
11         } catch (Exception e) {
12             System.out.println("An exception occurred!");
13         } finally {
14             System.out.println("We're done!");
15         }
16     }
17 }
```

Revise the code for division and check the output.

```java
public class ExceptionHandlingExample {
    public static void main(String[] args) {
        int divisor = 0;
        int dividend = 11;

        try {
            // The block of code that can throw exception(s) comes here
            System.out.println("The result is: " + dividend / divisor);
        } catch (ArithmeticException ae) {
            // What to do if an exception arises?
            System.out.println("Division by zero.");
        } finally {
            // Time to leave the try-catch
            System.out.println("We're done!");
        }
    }
}
```

1.5 String Handling

Type the following code and check the output.

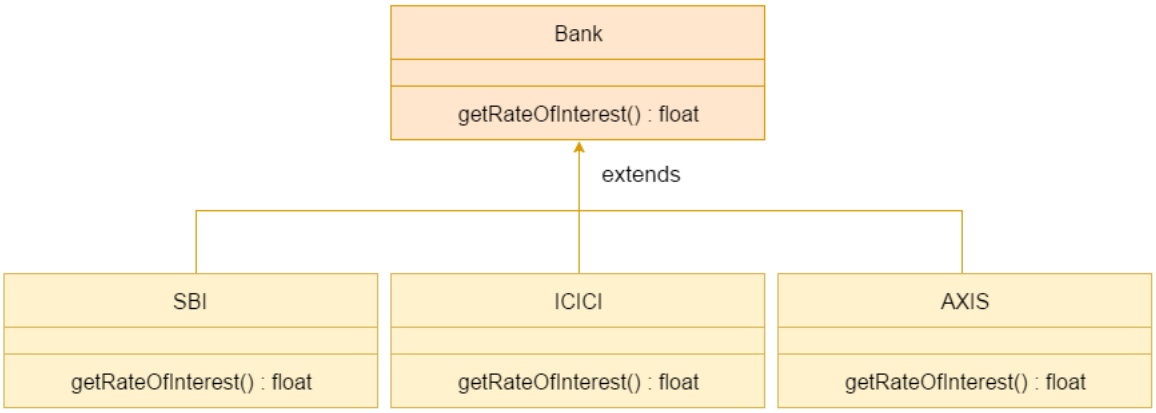
```
public class ComputeInitials {
public static void main(String[] args) {
String myName = "Fred F. Flintstone";
String myInitials = new String();
int length = myName.length();

for (int i = 0; i < length; i++) {
if (Character.isUpperCase(myName.charAt(i))) {
myInitials += myName.charAt(i);
}
}
System.out.println("My initials are: " + myInitials);
}
}
```

---

2 Assignments

1. Consider a scenario where Bank is a class that provides functionality to get the rate of interest. However, the rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7%, and 9% rate of interest.



Write a Java program that calculates the rate of interest for each bank. Use java method overriding.

2. Write a Java program to check whether a given array size is negative or not. Create a *try-catch* block. Inside the *try* block — read the size of the array, declare the array, store the array elements, and print the array elements. Inside the *catch* block, handle *negativeArraySizeException* and print an appropriate error message.

3. In an online banking system, if a customer wants to debit some amount of money, the system initially checks whether his/her current balance is below a threshold or not. If the customer's current balance is above the threshold then the system debits and shows the current balance to the customer. If the customer's current balance is below the threshold then the system penalizes him/her and shows the penalty amount. Write a Java program to implement the above concept. Throw an exception whenever the current balance is below the threshold while debiting some money.

4. Write a Java program to create a String object. Using constructor, initialize this object with your name. Print the length of your name using appropriate String method. Find whether character "a" is in your name or not; if yes find the number of occurrences of "a" in your name. Print the positions of occurrences of "a" . Repeat same for a different String object.

5. The Java library provides the **ArrayList** class, which can be considered as a dynamic array that occupies as much space as the number of elements in it. Based on the following sample code,

   - Create an **ArrayList** to store **Integer** types.
   - Using a for loop, store the squares of the first 10 natural numbers in the list so created. Use the **add()** method of **ArrayList** for this purpose.
   - Once again, use a for loop to print all the elements.
   - Print the size of the list (you should not count the items individually).
   - Remove all the elements from the list using a single method call.
   - Print whether or not the list is empty using only a single method call (no comparison allowed).

```
1 import java.util.ArrayList;
2
3 public class ArrayListDemo {
4     public static void main(String[] args) {
5         // A list to store items of type String
6         ArrayList<String> alist = new ArrayList<>();
7     }
8 }
```

   Hint: Find the relevant methods from the API documentation of ArrayList [1].

   Note: In Java, <> is called the diamond operator, and is used to represent generics. For example, an **ArrayList** is a generic type, that can store objects of any type — String, Double, Integer, Planet, Star, ArrayList, and so on. In contrast to type casting, generics are *type safe*.

[1] https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html