IT2030 – OBJECT ORIENTED PROGRAMMING – MOCK FINAL EXAM

Question 1 – Threads (This question is NOT related to the students who follows subject code as IE2021)

a) In this question you will create two Thread classes and implement
   a. Implement a Thread class called PrintThread that prints numbers 1 to 100. The PrintThread class should extend the Thread class in your implementation.
   b. Implement a Thread class called SLIITThread that prints the name "SLIIT" a 100 times.
   c. Implement a class called ThreadMain class and in the main() function create one thread each for each of the classes and run the two threads.

   Save the program as Ques1a.java

b) In this question you will display the status of running threads.
   a. Implement a Thread class called NumbersThread that print numbers 1 to 100. You should also print the name of the current Thread being printed.
   b. Implement a class called ThreadBase and in the main() function create three threads of the NumbersThread class.
   c. Name the three Threads as "Red", "Blue" and "Green"
   d. After starting the three threads, the main() function should print the word "SLLIT" 100 times.
   e. At the end of the main program Print the state and if the first thread is alive using the methods getState() and isAlive()

   Save the program as Ques1b.java

c) In this question you will do a Calculation and get multiple threads to perform the calculation.
   a. Implement a class called Calculation. In this class have a method called **sum**(int start, int end) that calculates the sum of the numbers between start and end. It should calculate this value and store it in a property called total.
   b. Implement a **getTotal**() method in the **Calculation** class to return total.
   c. Write a Thread class that stores a Calculation object, start and end values as properties and get them from the constructor. The run method should call the **sum** method of the **Calculation** class.
   d. Create a class called **ParallelTest** and in the main function perform the following.
      i. Create a **Calculation** type object
      ii. Create two threads that are used to calculate sum of the numbers 1 to 100000. Divide the work equally among the threads.
      iii. Finally print the result obtained.

Save the program as Ques1c.java

Question 2 – Exception Handling, Collections, String, Generics *(Note this question only focuses on Exception Handling)*

Save your program as Ques03.java

1) Consider the following **BankDemo** Application to perform deposit and withdraw amount from the customer account. To perform these operations, you should create an **Account** class and validate the withdrawal amount lest make the account **overdue.** You should create custom exception class **"InsufficientBalanceException".**

The sample **BankDemo** Application main program is given below with sample output. Your implementation should satisfy the same.

```
public class BankDemo {

    public static void main(String[] args) {

        Account account = new Account(123);
        System.out.println("Depositing Rs.10,000");
        account.deposit(10000.00);

        try {
            System.out.println("\nWithdrawing Rs.6,000/=");
            account.withdraw(6000.00);

            System.out.println("\nWithdrawing Rs.8,000/=");
            account.withdraw(8000.00);

        } catch (InsufficientBalanceException e) {
            System.out.println("Sorry, your account remains only Rs." + e.getAmount());
            e.printStackTrace();
        }
    }
}
```

When you withdraw more than the existing account throw **InsufficientBalanceException.** When you run the program out put should be as follows.

```
Console  Problems  Javadoc  Declaration  Serve
<terminated> BankDemo [Java Application] C:\Program Files\Java
Depositing Rs.10,000

Withdrawing Rs.6,000/=

Withdrawing Rs.8,000/=
Sorry, your account remains only Rs.4000.0
InsufficientBalanceException
        at Account.withdraw(Account.java:18)
        at BankDemo.main(BankDemo.java:15)
```

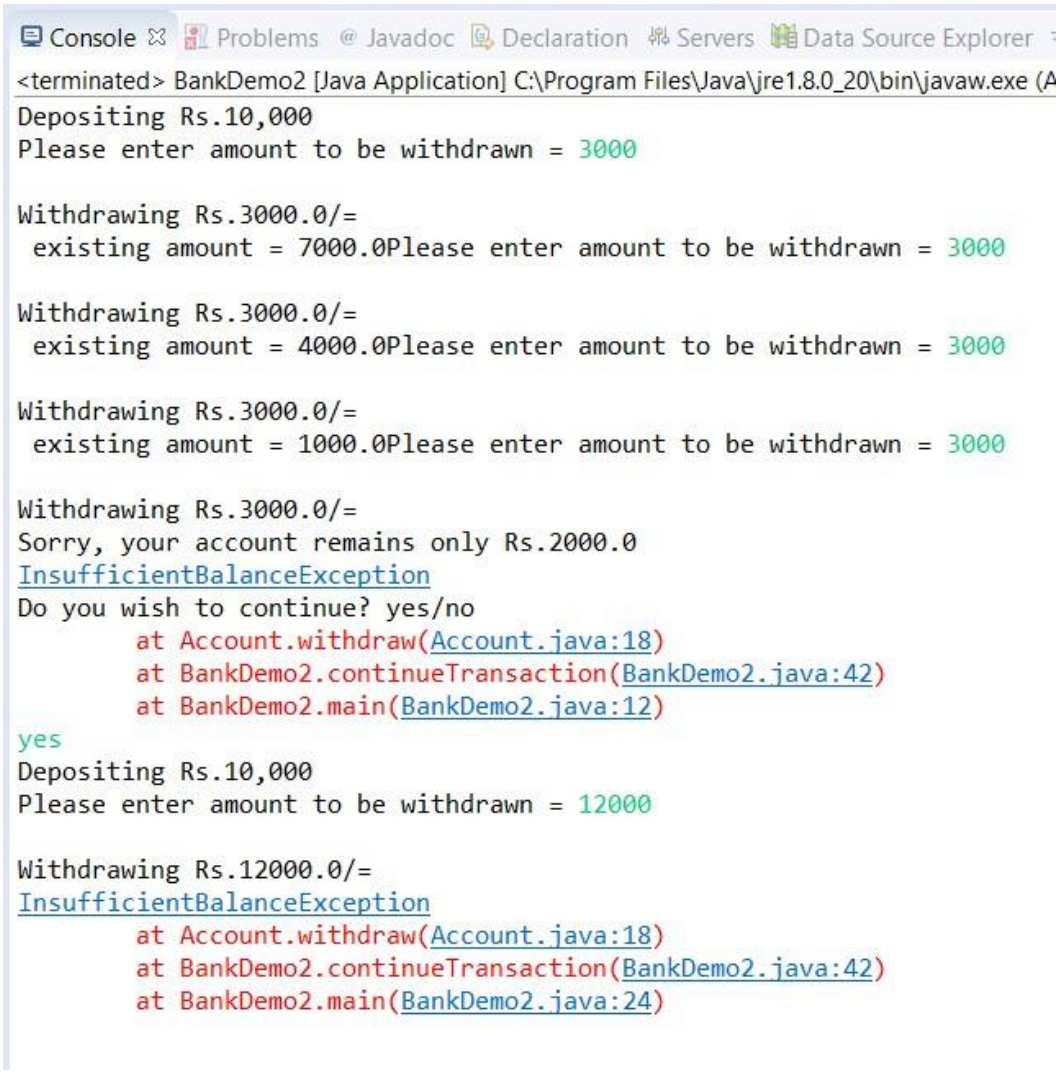
a) Create **InsufficientBalanceException** class and amount should be able to pass through the constructor of this custom exception class

b) Create **Account** class that holds **balance** and **Account No**. Implement operations to display existing balance, account number and account number can be assigned through the Constructor

c) Implement the **deposit** operation and that increases the existing balance in the account

d) Implement the withdraw operation and that reduces the balance with given value. In case if balance is not sufficient **throw InsufficientBalanceException** in the method and you should handle it in the BankDemo Application. You throw this in the withdraw operation as below
throw new InsufficientBalanceException(amount);

2) Modify the above **BankDemo** class to give the below output



(No need to consider the keyboard input validations in your implementation)

a) In the modified program user should enter the withdrawal amount as keyboard input and this activity should continue as infinite loop until user response for the question
"**Do you wish to continue ?"** If user answers as "**no**" program will terminate

b) You should extend the above exception handling with including **finally block**. In the finally block you should ask the above question "**Do you wish to continue?"**

c) If user response "**yes"** for the above question **a)** in your program should deposit the same amount for the account and continue the withdrawal process

d) Make sure you should not duplicate the logics in the program for above modification (Consider OOP concepts)

Question 3 – Object Oriented Concepts/Interfaces/Abstract Classes

Save your program as Ques04.java

1. Create an **abstract class** called **`Vehicle`**. The class should keep the following information in fields:
    - `speed`
    - `regularPrice`
    - `colour`

    a) Your class should have a constructor that initializes the three instance variables.

    b) Overload the constructor so that it accept only the `speed` and the `colour`.

    c) In addition, Car class has a method call `getRegularPrice` which returns the `regularPrice`.

2. Create a **sub class** of **`Vehicle`** class and name it as **`Truck`**. The **`Truck`** class has the following field.
    - `Weight`

    a) Your class should have a constructor that initializes all instance variables.

b) Override the `getSalePrice()` method to return the regular price based on the weight. If the weight is greater than 2000 regular price will have 10% discount. Otherwise, 20% discount.

3. Create a **sub class** of **`Vehicle`** class and name it as `Bus`. The **`Bus`** class has the following field.
    - `Year`
    - `manufacturerDiscount`

    a) Your class should have a constructor that initializes all instance variables.

    b) Override the `getSalePrice()` method to return the sales price based on the manufacturer Discount by subtracting the manufacturer discount from the sales price.

4. Create `MyOwnAutoShop` class which contains the main() method. Perform the following within the main() method.

    a) Create an instance of **`Truck`** class and initialize all the fields with appropriate values.

    b) Create an instance of the **`Bus`** class and initialize all the fields with appropriate values.

    c) Display the sale prices of all instance.