United International University (UIU)

Dept. of Computer Science & Engineering (CSE)

Final Exam, Trimester: Spring 2025
Course Code: CSE 1115, Course Title: Object Oriented Programming
Total Marks: 40, Duration: 2 Hours

Any examinee found adopting unfair means will be expelled from the trimester / program as per UIU disciplinary rules.
Answer all Five Questions.

QUESTION 1

a) Analyze the java code given below and write the output of the following code for each of the test cases given in the "Sample Input" table. [4 MARKS] [CO1]

```
class TryCatchExample
{
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      String[] arr = {"10","0",null};

      System.out.print("Enter index (0-2): ");
      int index = sc.nextInt();

      try {
         int value = Integer.parseInt(arr[index]);
         try {
            int result = 100 / value;
            System.out.println("Result = " + result);
         }
       catch (ArithmeticException e) {
            System.out.println("Inner catch: ArithmeticException");
         }
        finally {
            System.out.println("Inner finally block executed.");
         }
      }
     catch (ArithmeticException e) {
         System.out.println("Outer catch: ArithmeticException");
      }
      catch (NumberFormatException e) {
         System.out.println("Outer catch: NumberFormatException");
      }
      finally {
         System.out.println("Outer finally block executed.");
      }
   }
}
```

Sample Input

Enter index (0-2): 0

Output: ??

Enter index (0-2): 1

Output: ??

Enter index (0-2): 2

Output: ??

b) Suppose you are designing a banking application in Java. You need to write a program that simulates a withdrawal from a bank account. The account balance is initialized to 5000. The user will enter the amount to withdraw. If the amount entered is greater than the current balance, a user-defined exception named **InsufficientFundsException** should be thrown. The exception should display a meaningful message like: "**Withdrawal amount exceeds current balance**." The program must also include a finally block that always prints "**Transaction processing completed**."

You're given a skeleton code for the above scenario. You don't need to rewrite the portion given in the question. **Only write the codes necessary to complete the tasks.** [3 + 3 MARKS] [CO1]

```
// task1: Write the InsufficientFundsException class

public class Bank {
  private int balance;

  // Constructor
  public Bank(int initialBalance) {
     this.balance = initialBalance;
  }

  // task2: Write the withdraw method

  // Main method to test the Bank class
  public static void main(String[] args) {
     Bank myAccount = new Bank(5000);
     try {
           myAccount.withdraw(6000);
        } catch (InsufficientFundsException e) {
           System.out.println("Exception: " + e.getMessage());
        } finally {
           System.out.println("Transaction processing completed.");
        }
  }
```

QUESTION 2 [10 MARKS] [CO2]

Write a Java program that reads a file named "student.txt" with multiple lines where each line contains a student's ID (string), Name (string), mark1(integer) & mark2(integer) separated by a single space. Your task is to split this into two separate files:

- "info.txt" – containing Id & Name.
- "mark.txt" – containing Name & summation of both marks.

Both the input and output files should be located in the "src" directory. Check the following example for clarification:

student.txt	info.txt	mark.txt
701 Rabbi 15 12	701 Rabbi	Rabbi 27
702 Sohel 19 11	702 Sohel	Sohel 30
703 Roni 5 10	703 Roni	Roni 15

QUESTION 3 [5 MARKS] [CO1]

Consider the following code:

```
interface Computable {
  int compute(int a, int b);
}
abstract class Processor {
  protected String id;
  public Processor(String id) {
     this.id = id;
  }
  public abstract void process();

  public void printId() {
     System.out.println("Processor ID: " + id);
  }
}
```

Write an interface named "**AdvancedComputable**" which inherits "Computable" and a self-method named "*max",* return type: int. Now, write a class named "**Adder**" that inherits from "AdvancedComputable" and "Processor". Implement the necessary methods so that the class "**Adder"** contains the functionalities that:

i. find the sum of two integers (**a** and **b**).
ii. return the maximum of two integers (**a** and **b**).
iii. print the message "**Processing Addition**" when processing.

QUESTION 4 [5 MARKS] [CO2]

Consider the following code. Complete the tasks given in the table without adding any new instance variables in the "Product" class.

```
public class Product {
  String name;
  int amount;
  double unit_price;

  public Product(String name, int amount,
double unit_price) {
     this.name = name;
     this.amount = amount;
     this.unit_price = unit_price;
  }
}
class ProductList{
  public static void main(String[] args) {
     //write codes here
  }
}
```

Tasks:

i. Declare an arraylist of "Product"
ii. Insert the following products into the arraylist:
("Mango", 5, 20)
("Apple", 4, 24)
("Litchi", 20, 3)
iii. Insert a new product ("Banana", 10, 12) at index 1.
iv. Update the unit_price of "Apple" to 30.
v. Sort the arraylist based on the total price (*total price = amount * unit_price*) of the products in descending order.

QUESTION 5 [10 MARKS] [CO2]

Suppose that you are required to create a simple **Rock-Paper-Scissors** game using **Java Swing**. The game will be played by two players using buttons, and the result will be displayed based on randomly selected choices. The GUI contains three buttons labeled **Player 1 Play**, **Player 2 Play**, and **Check Winner**, along with labels to display each player's move and the final result. When **Player 1 Play** button or **Player 2 Play** button is clicked, it should randomly assign one of the three values (**Rock**, **Paper**, or **Scissors**) to the corresponding player label. When **Check Winner** button is clicked, it should compare both player choices and display either **Player 1 Wins!, Player 2 Wins!**, or **Draw** in the result label.

Write only the event-handling code for the three buttons. You may assume that the GUI components (JButtons, JLabels, etc.) have been already created.

Consider the following code:

```
public class RockPaperScissorsGUI {

// To generate a random choice, use the following function:
  private static String getRandomChoice() {
      String[] options = {"Rock", "Paper", "Scissors"};
      return options[(int)(Math.random() * 3)];
  }

public static void main(String[] args) {
      JFrame frame = new JFrame("Rock Paper Scissors");

      JLabel label1 = new JLabel("Player 1");
      JLabel label2 = new JLabel("Player 2");
      JLabel resultLabel = new JLabel("Result will be shown here");

      JButton leftButton = new JButton("Player 1 Play");
      JButton rightButton = new JButton("Player 2 Play");
      JButton checkButton = new JButton("Check Winner");

   // Write only the event-handling code

}
```

Before Click

After Click