



Final Exam. : : Trimester: Spring - 2017

Course Code: CSI 211 , Course Title: Object-Oriented Programming

Total Marks: 40

Duration: 2:00 Hours

There are 7 questions. Answer any 5 questions. Figures in the right-hand margin indicate full marks.

1. Find the output of the programs below.

[4+4]

a)	<pre>public class TestThread { public static void main (String [] args) { Thread t = new MyThread() { public void run() { System.out.print(" are you running?"); } }; t.start(); } } class MyThread extends Thread { MyThread() { System.out.print("MyThread"); } public void run() { System.out.print(" running"); } public void run(String s) { System.out.print(s + " is running again"); } }</pre>
b)	<pre>import java.util.InputMismatchException; public class TestException { public static void main(String[] args) { int num = 4; try{ try{ throwException(num); }catch(InputMismatchException e){ System.out.println(e.getMessage()); throwException(++num); }finally{ System.out.println("I am out of here."); } }catch(Exception e){ System.out.println(e.getMessage()); } } public static void throwException(int num){ if(num%2==0) throw new InputMismatchException("Can't be an even number."); else if(num%5==0) throw new NumberFormatException("Can't be a multiple of 5."); else System.out.println("Input: "+5); } }</pre>

2. Suppose, you are given an ArrayList named **listNum** of 15 Integer numbers and listNum is already initialized with 15 random values: 37, 46, 8, 55, 39, 76, 96, 15, 77, 57, 47, 18, 86, 11, 71. Write a multi-threaded program where each thread will remove 5 items (one at a time) from **listNum**. Note that **listNum** is commonly accessed by all the threads. From **main**, create and start **three** threads. Use appropriate precaution so that **listNum** is updated properly, or in other words no two threads update **listNum** at the same moment. So, after running three threads, the ArrayList (**listNum**) should be empty. [8]
3. You are helping a programming contest organizer to select the teams for the contest. Each team can have minimum of 3 members and maximum 5 members and each team member must complete 100 credit hours to participate. A partial program has already been written as shown below. You are asked to rewrite the **checkTeamCount(int)** and **checkCredit(int)** method as instructed below. Also you need to implement the 2 custom exceptions mention in those 2 methods. **Implement the InvalidTeamMemberException, InvalidMemberCountException** in such a way so that you can specify/pass the credit, minimum and maximum member count into the exception class and set the exception message using those parameters. For **InvalidTeamMemberException** the exception message should be set to "Must complete [minCredit] credits to participate in the contest" and for **InvalidMemberCountException** the exception message should say "We need [min] to [max] team members to participate in the contest." [8]

```
import java.util.Scanner;
public class ProgrammingContest {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Please enter team member count");
        int teamCount = scan.nextInt();
        try{
            checkTeamCount(teamCount);
        }catch(InvalidMemberCountException e){
        }
        int creditComp=0;


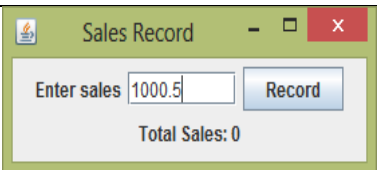
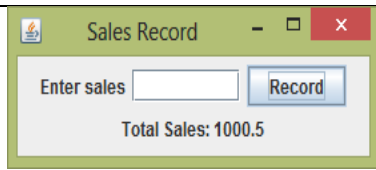

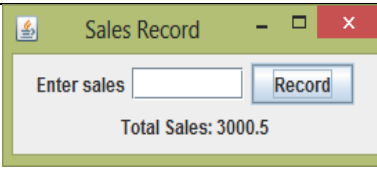
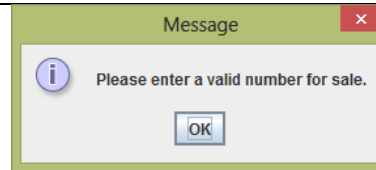
        for(int i=0; i<teamCount; i++){
            System.out.println("Please enter the credit completed by member "+ (i+1));
            creditComp = scan.nextInt();
            try {
                checkCredit(creditComp);
            } catch (InvalidTeamMemberException e) {
            }
        }
    }
    public static void checkTeamCount(int count) {
        //throw InvalidMemberCountException when count is less than 3 or greater than 5.
        //pass 3 for min & 5 for max count while creating InvalidMemberCountException
    }

    public static void checkCredit(int credit) {
        // throw InvalidTeamMemberException if credit is less than 100.
        // and pass 100 as minCredit while creating InvalidTeamMemberException object
    }
}
```

4. UIU CSE admin is storing the section wise student count for every course in a text file named “Sections.txt”. Each line stores record of one Course in the format [courseID-secACount-secBCount-secCCount...]. Write a program that will read the file one line at a time, calculate the **total student** of that course and then **write** that info in a separate file named “Output.txt” in the format [courseID-totalStudents]. See below for the **sample input and output file**. [8]

Sections.txt	Output.txt
CSI121-30-35-29	CSI121-94
CSI122-20-24-25-20	CSI122-89
CSI211-34-26-30	CSI211-90
CSI212-20-20-20-25	CSI212-85

5. An electronics shop needs a simple GUI to record their sales information, where they will enter their sales amount each time they sell and it will give them the total sales. So create a simple GUI as shown below; initially the total sales will be 0. When user enters the sale in the **textfield** and hit the “Record” button, it will add that sale amount to the total sales and display in the label below it, it will also clear the textfield. If user enters anything other than number, a pop-up message should display saying “Please enter a valid number for sale” as shown in the picture below. [8]

1.Initial State 	2. User entered Sales amount 	3. User clicked “Record” 
4. User entered another Sales amount 	5. User clicked “Record” 	6. User entered invalid number e.g. “a” in Sales, a popup message display 

6. Find out if the following JAVA programs have any error. **List the errors** if any. Fix the code and **rewrite** after the errors list. You cannot **delete any line** of code. However, you are **allowed to edit or add** any code as per requirement. [4+4]

a)	<pre> public class TestThread1 implements Runnable{ String name; public TestThread1(String name) { this.name = name; } public void run(int n){ System.out.printf("Running:%s %d times.\n", name, n); } public static void main(String[] args) { TestThread1 t1 = new TestThread1("First Thread"); t1.start(); t1.join(); } </pre>
----	--

	<pre> } b) import java.io.IOException; public class TestExceptionError { public static void main(String[] args) { try { whoIs("IOException"); } finally{ System.out.println("Always executes."); } System.out.println("Which exception should be handled?"); catch (Exception e) { e.printStackTrace(); } catch (IOException e) { e.printStackTrace(); } } public static void whoIs(String n) throws IOException{ System.out.println("Who is "+n+"?"); } } </pre>
--	---

7. a) Create a generic **static** method named **sort(...)** that will take **2 parameters**; first one is an ArrayList of Generic type and 2nd one is a boolean. The method will **sort** the ArrayList in **ascending** order if the boolean parameter is true, descending otherwise. [5]

- b) Modify the following **BankAccount** class in such a way so that following code generates the expected output as shown below i.e. the **ArrayList** should be sorted in ascending order of balance. [3]

Code	Expected Output
<pre> public class TestGeneric { public static void main(String[] args) { ArrayList<BankAccount> accounts=new ArrayList<BankAccount>(); accounts.add(new BankAccount("Rasha", "011123", 12000)); accounts.add(new BankAccount("Keya", "011124", 10500)); accounts.add(new BankAccount("Asad", "011125", 100000)); Collections.sort(accounts); System.out.println("Accounts sorted in Ascending order:"); for(BankAccount b: accounts) System.out.println(b); } } class BankAccount { String name, id; double balance; public BankAccount(String name, String id, double balance){ this.name = name; this.id = id; this.balance = balance; } } </pre>	<p>Accounts sorted in Ascending order: Keya-011124-10500.0 Rasha-011123-12000.0 Asad-011125-100000.0</p>