VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of the UGC Act, 1956)

Reg. No. : 23MCA102 6

Final Assessment Test (FAT) - May 2024

Programme	M.C.A.	Semester	WINTER SEMESTER 2023 - 24
Course Title	JAVA PROGRAMMING	Course Code	PMCA502L
Faculty Name	Prof. MADHESWARI	Slot	D1+TD1
		Class Nbr	CH2023240501374
Time	3 Hours	Max. Marks	100

General Instructions:

- Write only Register Number in the Question Paper where space is provided (right-side at the top) & do not write any other details.

Answer all questions (10 X 10 Marks = 100 Marks)

01. Design a Java program to manage student records using a **'Student'** class. The **'Student'** class should include attributes like **'name'**, **'rollNumber'**, and an array **'marks'** to store marks obtained in different subjects. Implement a **constructor** in the **'Student'** class utilizing the **'this'** keyword to initialize these attributes. Additionally, include methods for displaying student details, calculating the average marks, and updating marks for a specific subject. Utilize the **'Scanner'** class for user input and ensure explicit error handling for invalid inputs, such as **non-numeric marks or marks out of range** (considering the maximum marks for each subject). Specify that the **maximum marks** for each subject are **100**, and the program should consider a **maximum of 5 subjects**. Create a **'StudentManager'** class with a **main method** to demonstrate the functionalities of the Student class. The program should enable users to create **multiple Student objects**, update their marks, and display their details. [10]

02. Design a Java program for a banking system that incorporates inheritance, method overloading, and method overriding. Implement a superclass **'Account'** with attributes like **'accountNumber'**, **'accountHolderName'**, and **'balance'**. Include methods for **'deposit'**, **'withdrawal'**, and displaying **'account information'**. In the **'withdrawal'** method of the base class **'Account'**, it takes two arguments: the **withdrawal amount (type double)** and the **withdrawal limit (type double)**. Ensure that the withdrawal limit parameter is used only for the **'SavingsAccount'** subclass. Extend the **'Account'** class to create subclasses **'SavingsAccount'** and **'CurrentAccount'**, representing different types of bank accounts. In the **'SavingsAccount'** subclass, **overload** the **'withdrawal'** method to include an additional parameter for **withdrawal limit**, set to **Rs. 50,000/-**. **Override** the **'withdrawal'** method in the **'CurrentAccount'** subclass to implement specific overdraft functionality, with an **overdraft limit** set to **Rs. 80,000/-**. [10]

Ensure input validation to handle errors:

- **For invalid withdrawals:** Ensure that the withdrawal amount is greater than zero.
- **For insufficient funds:** Check if the withdrawal amount is less than or equal to the account balance.

- **For incorrect account details**: Validate that the account number and account holder's name are not empty strings, and the balance is not negative.

03. Write a Java program that generates various patterns using loops. The program should prompt the user to select a pattern type and then print the corresponding pattern. The available pattern types are as follows: [10]

Pattern 1: A pattern with increasing number of stars on each line.

```
*
**
***
****
*****
******
```

Pattern 2: A pattern with decreasing number of stars on each line.

```
*****
****
***
**
*
```

Pattern 3: A pyramid pattern with increasing number of stars from top to bottom.

```
   *
  ***
 *****
*******
```

Pattern 4: A pyramid pattern with decreasing number of stars from top to bottom.

```
*******
 *****
  ***
   *
```

Pattern 5: A pattern with a diamond shape.

```
   *
  ***
 *****
*******
 *****
  ***
   *
```

04. Design a Java program to cover different scenarios of exception handling. Create a User Defined exception class named **'InvalidAgeException'** to handle situations where the user inputs an age less than 0 or greater than 100. Construct a class that receives user input for a number as a string and attempts to convert it to an integer, providing handling for **'NumberFormatException'** if the input string is not a valid integer. Define a class **'Person'** with attributes **'name'** and **'age'**, create an object without assigning values to its members. Handle **'NullPointerException'** when accessing the attributes of this object. Write a method to declare an array of size 5 and attempting to access the 6th element, with handling for **ArrayIndexOutOfBoundsException**. [10]

05. Create a JavaFX program for a simple **'calculator'** app. The calculator should include a **graphical user interface (GUI)** with buttons for the **numbers 0-9**, **operators (+, -, *, /)**, and **functions (clear, equal)**. It should utilize the layout **GridPane** to properly align these elements. Create code that does basic arithmetic operations **(addition, subtraction, multiplication, and division)** when the respective operator buttons are clicked. Make that the calculator displays the entered numbers as well as the operation's result. Create a code for the clear button to reset the calculator and an equal button to compute the results. [10]

06. Develop a **JavaServer Pages (JSP)** application for an **online bookstore**. Create an **index.jsp** page that greets users with the **bookstore's name** and presents a list of available books with details like **title, author,** and **price**. Include a **search form** that enables the user to search for books by title or author, with dynamic handling to display matching results. Ensure the page is well-structured, responsive, and visually appealing using **CSS** styling. Specifically, use **inline styling** to set the **background color** of the form to **light grey**. [10]

07. Explain the MVC design pattern and its importance in software development. Create a basic JavaFX application for managing a list of contacts using the Model-View-Controller (MVC) architecture. [10]

Model: Implement a Contact class with attributes such as name, email, and phone number.

View: Design a JavaFX GUI with a TableView to display the list of contacts. Include text fields for adding a new contact with fields for name, email, and phone number. Also, add buttons for adding and deleting contacts.

Controller: Write a controller class to handle user input and interact with the model. Include methods for adding and deleting contacts, as well as initializing the view and connecting it with the model.

08. You are tasked with developing a simple distributed application using **Remote Method Invocation (RMI)** in Java. The application consists of a client-server architecture where the client sends a request to the server to perform a basic mathematical operation, and the server responds with the result. Create an **interface** named **MathOperation** containing the method signatures for basic mathematical operations such as addition, subtraction, multiplication, and division. Create a **server** class that implements the **MathOperation** interface. Create a **client** class that connects to the server and invokes the remote methods to perform mathematical operations. [10]

09. Design a Java program for managing geometric shapes using interfaces and polymorphism. Define an interface named '**Shape**' with methods '**calculateArea()**' and '**calculatePerimeter()**'. Declare two classes, '**Circle**' and '**Rectangle**', each implementing the '**Shape**' interface'. The Circle class should have a **constructor** that takes the **radius** as a parameter, while the Rectangle class should take the **length** and **width**. Provide appropriate implementations for '**calculateArea()**' and '**calculatePerimeter()**' in both classes based on their respective shapes. [10]

10. Discuss the fundamentals of the Spring Framework in Java, including major features such as dependency injection (DI), inversion of control (IoC), and aspect-oriented programming. Develop a basic Spring application in Java to manage tasks. Design a Task class to represent tasks and a TaskService class to interact with tasks. Utilize Spring to configure the application context for managing beans and enable dependency injection. Provide functionality to add tasks and list all tasks. [10]