K. K. Wagh Institute of Engineering Education and Research, Nashik

(An Autonomous Institute from A. Y. 2022-23)

End-Sem Examination-Winter 2025
Model Answer Key

Exam Seat No.										

Academic Year: 2025-2026	**Semester: I**
Name of Programme: MCA	**Pattern: 2022**
Name of Course: Object Oriented Programming	**Course Code: MCA222001**
Max. Marks: 60	**Duration: 2:30Hr.**

Instructions: Candidates should read carefully the instructions printed on the Question Paper and on the cover page of the Answer Book, which is provided for their use.

1. This question paper contains 02 page(s).
2. Answer to each new question is to be started on a new page.
3. Assume suitable data wherever required, but justify it.
4. Draw the neat labeled diagrams, wherever necessary.
5. The last column indicates the Course Outcome of the Question/sub-question

<table>
<tr><th>Q. No.</th><th>Details</th><th>Max. Marks</th><th>CO No.</th><th>Bt level</th></tr>
<tr><td>Q.1</td><td>Explain the concept of Object-Oriented Programming and its advantages over procedural programming. (6 Marks)
Answer:
Object-Oriented Programming (OOP) is a programming paradigm centered around the concept of objects, which are instances of classes. These objects encapsulate data (attributes) and behavior (methods) that operate on the data. OOP promotes modularity, reusability, and abstraction.
Key Principles of OOP:
Encapsulation: Bundling data and methods within a class to protect internal state.

Inheritance: Creating new classes from existing ones to promote code reuse.

Polymorphism: Allowing objects to be treated as instances of their parent class, enabling flexibility.

Abstraction: Hiding complex implementation details and exposing only essential features.
Advantages of OOP over Procedural Programming:
<table><tr><th>Feature</th><th>Object-Oriented Programming (OOP)</th><th>Procedural Programming</th></tr><tr><td>Modularity</td><td>Code is organized into classes and objects</td><td>Code is organized into functions and procedures</td></tr></table></td><td>[6]</td><td>CO1</td><td>L2</td></tr>
</table>

Reusability	Inheritance and polymorphism promote reuse	Limited reuse; functions must be rewritten
Maintainability	Easier to update and manage due to encapsulation	Changes may affect multiple parts of the code
Scalability	Better suited for large, complex applications	Becomes harder to manage as size increases
Security	Encapsulation protects data from unintended access	Data is exposed and less protected
Real-world modeling	Objects represent real-world entities naturally	Less intuitive for modeling real-world systems

OOP provides a structured and scalable approach to software development, making it ideal for modern applications. Its emphasis on abstraction, encapsulation, and reuse makes it superior to procedural programming in terms of maintainability, flexibility, and design clarity.

Q.2 Differentiate between function overloading and function overriding with examples (6 Marks) — [6] — CO3 — L3

Answer:

Feature	Function Overloading	Function Overriding
Location	Same class	Base and derived classes
Polymorphism Type	Compile-time	Runtime
Method Signature	Must differ in parameters	Must be same as base class method
Return Type	Can differ	Must be same or covariant
Keyword Used	None required	virtual in base, override in derived

Q.3 a) Write a Java program to define a class Student with data members and methods to display student details. (8 Marks) — [16] — CO5 — L3

Answer:

```
// Define the Student class
class Student {
    // Data members (attributes)
    int rollNumber;
```

```
    String name;
    String course;
    // Constructor to initialize student details
    Student(int rollNumber, String name, String course) {
        this.rollNumber = rollNumber;
        this.name = name;
        this.course = course;
    }
    // Method to display student details
    void displayDetails() {
        System.out.println("Student Details:");
        System.out.println("Roll Number: " + rollNumber);
        System.out.println("Name       : " + name);
        System.out.println("Course     : " + course);
    }
}
// Main class to test Student
public class Main {
    public static void main(String[] args) {
        // Create a Student object
        Student s1 = new Student(101, "Archana Patil", "MCA");
        // Display student details
        s1.displayDetails();
    }
}
```

Output:

```
Student Details:
Roll Number: 101
Name       : Roshan Patil
Course     : MCA
```

b) Define an interface Printable and implement it in a class Document with a method to print content. (8 Marks)

Answer:

Definition

	An **interface** in Java is a reference type that contains only abstract methods and constants. It is used to achieve **abstraction** and **multiple inheritance**. Interfaces define a contract that implementing classes must fulfill.
	
	Step 1: Define the Interface Printable
	java
	interface Printable {
	void print(); // Abstract method to be implemented
	}
	The Printable interface declares a method print() which must be implemented by any class that uses this interface.
	Step 2: Implement the Interface in Class Document
	java
	class Document implements Printable {
	String content;
	
	// Constructor to initialize content
	Document(String content) {
	this.content = content;
	}
	
	// Implementing the print method
	public void print() {
	System.out.println("Document Content: " + content);
	}
	}
	The Document class implements Printable and provides a concrete definition for the print() method.
	Step 3: Demonstrate Usage
	java
	public class Main {
	public static void main(String[] args) {
	Document doc = new Document("Welcome to MCA Department!");
	doc.print(); // Output: Document Content: Welcome to MCA Department!
	}
	}
	
	Key Concepts Demonstrated
	Interface declaration and implementation
	Abstraction and polymorphism
	Method overriding via interface
	Object creation and method invocation			

	c) Define a constructor in Java. Illustrate the different types of constructors used to initialize object data, and demonstrate the concept of constructor with suitable examples. (8 Marks) **Answer:** Definition of Constructor A constructor in Java is a special method used to initialize objects. It has the same name as the class, does not have a return type, and is automatically invoked when an object is created.		CO5	L3

Types of Constructors in Java

Java supports three main types of constructors:

1. Default Constructor

A default constructor is provided by Java when no constructor is defined by the programmer.

It initializes object data members with default values.

Example:

```
class Demo1 {
   int x;
   Demo1() {   // Default constructor
      x = 10;
   }
   void show() {
      System.out.println("Value of x: " + x);
   }
   public static void main(String args[]) {
      Demo1 d = new Demo1();
      d.show();
   }
}
```

2. Parameterized Constructor

A constructor that accepts parameters and allows the user to initialize objects with custom values.

Example:

```
class Demo2 {
   int a, b;
   Demo2(int x, int y) {   // Parameterized constructor
      a = x;
      b = y;
   }
   void display() {
      System.out.println("a = " + a + ", b = " + b);
   }
   public static void main(String args[]) {
```

```
        Demo2 d = new Demo2(5, 15);
        d.display();
    }
}
```

3. Copy Constructor

Java does not have an inbuilt copy constructor,

but programmers can create one to copy data from one object to another.

Example:

```
class Demo3 {
    int num;
    Demo3(int n) {      // Parameterized constructor
        num = n;
    }
    Demo3(Demo3 d) {    // Copy constructor
        num = d.num;
    }
    void show() {
        System.out.println("Number: " + num);
    }
    public static void main(String args[]) {
        Demo3 d1 = new Demo3(50);
        Demo3 d2 = new Demo3(d1); // Copying object
        d1.show();
        d2.show();
    }
}
```

OR

d) Create an inner class within a Java class and access its members from the outer class.(8 Marks)

Answer:

Definition

An inner class in Java is a class defined inside another class.

The outer class can create an object of the inner class and access its members.

Inner classes help in encapsulation and logically grouping classes.

```
class Outer {
```

	private String outerMsg = "Message from Outer Class";
	
	Code:
	// Inner class
	class Inner {
	String innerMsg = "Message from Inner Class";
	
	void showInner() {
	System.out.println(innerMsg);
	}
	}
	
	// Method in outer class accessing inner class members
	void display() {
	Inner in = new Inner(); // create inner class object
	in.showInner(); // call inner class method
	System.out.println("Accessing inner variable: " + in.innerMsg);
	}
	}
	
	public class TestInner {
	public static void main(String[] args) {
	Outer outer = new Outer();
	outer.display();
	}
	}
	Output:
	Message from Inner Class
	Accessing inner variable: Message from Inner Class			

Q.4	a) Write a Java program using inheritance where a class Employee is extended by a class Manager. (8 Marks)
	Answer:
	// Superclass
	
	class Employee {
	
	String name;
	
	int empId;
	
	// Constructor
	
	Employee(String name, int empId) {
	
	this.name = name;
	
	this.empId = empId;
	
	}
	
	// Method to display employee details
	
	void display() {
	
	System.out.println("Employee Name: " + name);
	
	System.out.println("Employee ID: " + empId);
		[16]	CO2	L2

```
    }
}
// Subclass extending Employee
class Manager extends Employee {
    String department;
    // Constructor
    Manager(String name, int empId, String department) {
        super(name, empId); // call superclass constructor
        this.department = department;
    }
    // Method to display manager details
    void showManagerDetails() {
        display(); // call method from Employee
        System.out.println("Department: " + department);
    }
}
// Main class
public class InheritanceDemo {
    public static void main(String[] args) {
        // Create Manager object
        Manager m = new Manager("Rahul", 101, "IT");
        m.showManagerDetails();
    }
}
```

d)Create an inner class within a Java class and access its members from the outer class.(8 Marks)

Here's a complete answer for d) Create an inner class within a Java class and access its members from the outer class (8 Marks):

Explanation

In Java, an inner class is a class defined inside another class.

The outer class can create an object of the inner class and access its members.

Inner classes are useful for logically grouping classes that are only used in one place, increasing encapsulation.

Example Code

```
// Outer class
class Outer {
   private String outerMsg = "Hello from Outer Class!";

   // Inner class
   class Inner {
      String innerMsg = "Hello from Inner Class!";
          void displayInner() {
         System.out.println(innerMsg);
      }
   }
   // Method in outer class accessing inner class members
   void accessInner() {
      // Create object of Inner class
      Inner in = new Inner();
      in.displayInner(); // Access inner class method
      System.out.println("Accessing inner variable: " + in.innerMsg);
   }
}
// Main class
public class InnerClassDemo {
   public static void main(String[] args) {
      Outer outer = new Outer();
      outer.accessInner(); // Outer class accessing inner class members
   }
}
```

Output

```
Hello from Inner Class!
```

Accessing inner variable: Hello from Inner Class!

OR

b) Implement a multithreaded Java application using Runnable interface to simulate two threads printing numbers. (8 Marks)

Answer:

In Java, multithreading allows concurrent execution of two or more parts of a program.

The Runnable interface is commonly used to define tasks for threads.

Each thread runs independently and can perform tasks simultaneously.

Code:

```
// Class implementing Runnable interface
class NumberPrinter implements Runnable {
   private String threadName;

   NumberPrinter(String name) {
      this.threadName = name;
   }

   // run() method defines the thread task
   public void run() {
      for (int i = 1; i <= 5; i++) {
         System.out.println(threadName + " prints: " + i);
         try {
            Thread.sleep(500); // pause for clarity
         } catch (InterruptedException e) {
            System.out.println(threadName + " interrupted.");
         }
      }
   }
}
```

```
// Main class
public class MultiThreadDemo {
    public static void main(String[] args) {
        // Create two Runnable objects
        NumberPrinter np1 = new NumberPrinter("Thread-1");
        NumberPrinter np2 = new NumberPrinter("Thread-2");

        // Create Thread objects
        Thread t1 = new Thread(np1);
        Thread t2 = new Thread(np2);

        // Start threads
        t1.start();
        t2.start();
    }
}
```

Output:

Thread-1 prints: 1
Thread-2 prints: 1
Thread-1 prints: 2
Thread-2 prints: 2
Thread-1 prints: 3
Thread-2 prints: 3
Thread-1 prints: 4
Thread-2 prints: 4
Thread-1 prints: 5
Thread-2 prints: 5

c) Illustrate the synchronization method in Java using a shared resource accessed by multiple threads. (8 Marks)

Answer:

Synchronization in Java ensures that only one thread can access a shared resource at a time.

This prevents data inconsistency when multiple threads operate on the same object.

Achieved using the synchronized keyword.

CO2 | L2

Code

```
// Shared resource class
class Counter {
   private int count = 0;

   // synchronized method
   public synchronized void increment() {
      count++;
      System.out.println("Count: " + count);
   }
}

// Thread class
class MyThread extends Thread {
   Counter c;

   MyThread(Counter c) {
      this.c = c;
   }

   public void run() {
      for (int i = 0; i < 5; i++) {
         c.increment();
      }
   }
}

// Main class
public class SyncDemo {
   public static void main(String[] args) {
      Counter counter = new Counter();

      // Two threads sharing same resource
      MyThread t1 = new MyThread(counter);
      MyThread t2 = new MyThread(counter);

      t1.start();
      t2.start();
   }
}
```

Output:
Count: 1
Count: 2
Count: 3
...
Count: 10

OR

d)Create a user-defined exception InvalidAgeException and use it in a program to validate age input.
(8 Marks)

Answer:

Java allows creation of user-defined exceptions by extending the Exception class.

Here, InvalidAgeException is thrown if age is less than 18.
Code:

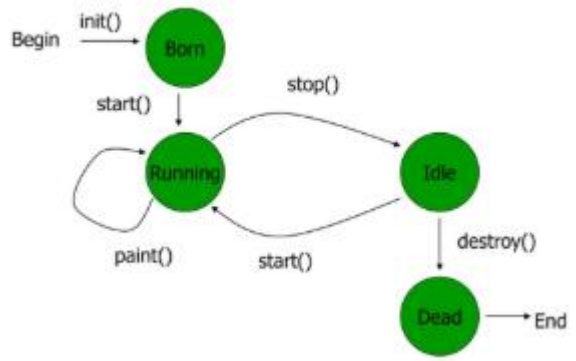
```
// User-defined exception
class InvalidAgeException extends Exception {
    InvalidAgeException(String msg) {
        super(msg);
    }
}

// Main class
public class AgeValidation {
    // Method to validate age
    static void validate(int age) throws InvalidAgeException {
        if (age < 18) {
            throw new InvalidAgeException("Age must be 18 or above!");
        } else {
            System.out.println("Valid age: " + age);
        }
    }

    public static void main(String[] args) {
        try {
            validate(15); // invalid case
        } catch (InvalidAgeException e) {
            System.out.println("Exception caught: " + e.getMessage());
        }

        try {
            validate(22); // valid case
        } catch (InvalidAgeException e) {
            System.out.println("Exception caught: " + e.getMessage());
        }
    }
}
```

Output:
Exception caught: Age must be 18 or above!
Valid age: 22

Q.5 a) Illustrate applet life cycle in detail. (8 Marks) [16] CO4 L3

Answer:

An **applet** is a Java program that runs inside a web browser or applet viewer. The **life cycle of an applet** is controlled by the browser and consists of specific methods:



Applet Life Cycle Methods
init()

Called once when the applet is first loaded.

Used for initialization (e.g., setting up variables, UI components).

start()

Called after init() and every time the applet becomes active.

Used to start animations, threads, or tasks.

paint(Graphics g)

Called whenever the applet needs to redraw itself.

Used for displaying output on the screen.

stop()

Called when the user navigates away from the page.

Used to suspend threads or pause activities.

destroy()

Called when the applet is terminated.

Used to release resources.
Code:

```
import java.applet.Applet;
import java.awt.Graphics;

public class AppletLifeCycle extends Applet {
    public void init() {
        System.out.println("Applet initialized");
    }

    public void start() {
        System.out.println("Applet started");
    }

    public void paint(Graphics g) {
        g.drawString("Applet Life Cycle Demo", 20, 20);
    }

    public void stop() {
        System.out.println("Applet stopped");
    }

    public void destroy() {
        System.out.println("Applet destroyed");
    }
}
```

Output:

Applet initialized
Applet started
Applet stopped
Applet destroyed

OR

b) Create a GUI application using AWT that includes a label, text field, and button to display user input. (8 Marks)

Answer:

AWT (Abstract Window Toolkit) is used to create GUI applications in Java.

Components: Label, TextField, and Button.

When the button is clicked, the text entered by the user is displayed.

Code:

```
import java.awt.*;
import java.awt.event.*;

public class AWTApp extends Frame implements ActionListener {
   Label lbl;
   TextField tf;
   Button btn;

   public AWTApp() {
      // Create components
      lbl = new Label("Enter your name:");
      tf = new TextField(20);
      btn = new Button("Display");

      // Set layout
      setLayout(new FlowLayout());

      // Add components
      add(lbl);
      add(tf);
      add(btn);

      // Add action listener
      btn.addActionListener(this);

      // Frame settings
      setSize(300, 200);
      setVisible(true);
   }

   public void actionPerformed(ActionEvent e) {
      String input = tf.getText();
      lbl.setText("Hello, " + input);
   }

   public static void main(String[] args) {
      new AWTApp();
   }
```

	} Output A window appears with: Label: Enter your name: TextField: for user input Button: Display Hello, <user input>			

	c) Design a Swing-based form using JLabel, JTextField, JButton, and JComboBox to collect user data. (8 Marks)
	Answer:
	
	Swing is a Java GUI toolkit that provides lightweight components.
	
	Common components:
	
	JLabel → displays text.
	
	JTextField → allows user input.
	
	JButton → performs actions when clicked.
	
	JComboBox → provides a drop-down list.
	
	Code:
	import javax.swing.*;
	import java.awt.*;
	import java.awt.event.*;
	
	public class SwingForm extends JFrame implements ActionListener {
	JLabel nameLabel, deptLabel, outputLabel;
	JTextField nameField;
	JComboBox<String> deptBox;
	JButton submitBtn;
	
	public SwingForm() {
	// Labels
	nameLabel = new JLabel("Enter Name:");
	deptLabel = new JLabel("Select Department:");
	outputLabel = new JLabel("");
	
	// Text field
	nameField = new JTextField(15);
	
	// Combo box
	String[] departments = {"IT", "HR", "Finance", "Marketing"};
	deptBox = new JComboBox<>(departments);
	
	// Button
	submitBtn = new JButton("Submit");
	submitBtn.addActionListener(this);
	
	// Layout
	setLayout(new FlowLayout());
			CO4	L3

```
        add(nameLabel);
        add(nameField);
        add(deptLabel);
        add(deptBox);
        add(submitBtn);
        add(outputLabel);

        // Frame settings
        setSize(300, 200);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public void actionPerformed(ActionEvent e) {
        String name = nameField.getText();
        String dept = (String) deptBox.getSelectedItem();
        outputLabel.setText("Name: " + name + ", Dept: " + dept);
    }

    public static void main(String[] args) {
        new SwingForm();
    }
}
```

Output:
A window with:

Label: Enter Name + TextField
Label: Select Department + ComboBox
Button: Submit
Ex.
Name: <user input>, Dept: <selected department>

OR

d) Compare and contrast Swing and AWT in terms of performance, look and feel, and flexibility. (8 Marks)

Answer:

Feature	AWT (Abstract Window Toolkit)	Swing (Java Foundation Classes)
Performance	Heavyweight components (depend on OS resources, slower)	Lightweight components (pure Java, faster)
Look & Feel	Native look (depends on OS)	Consistent look across platforms; supports pluggable look & feel
Flexibility	Limited components (basic UI only)	Rich set of components (tables, trees, sliders, etc.)
Portability	Less portable (OS-dependent rendering)	Highly portable (platform-independent rendering)
Event Handling	Uses old event model	Improved delegation event model
Customization	Difficult to customize	Easy to customize and extend

ALL THE BEST