USN										

Internal Assessment Test I– NOV-2024---Scheme

Sub:	OBJECT ORIENTED PROGRAMMING WITH JAVA					Sub Code:	BCS306A	Branch:	ISE
Date:	08-11-24	Duration:	90 min	Max Marks:	50	Sem/Sec:	III A,B,C		OBE

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1a	**Explain the operations of the following operators with example for each: i) >>> ii) && iii) >>**	[5]	CO1	L1

i) >>> (Unsigned Right Shift Operator)

The >>> operator shifts the bits of a number to the right, filling the leftmost bits with **0**, regardless of the sign of the number. This operator is mainly used in **Java** and some other programming languages, but not in Python.

- **Operation**: Unsigned right shift.
- **Example in Java**:

```
public class Main {
    public static void main(String[] args) {
        int num = -8; // Binary: 11111111 11111111 11111111 11111000
        int result = num >>> 2; // Shift 2 bits right: 00111111 11111111 11111111 11111110
        System.out.println(result); // Output: 1073741822
    }
}
```

Explanation:

- -8 in binary (32-bit): 11111111 11111111 11111111 11111000
- After shifting right by 2 bits: 00111111 11111111 11111111 11111110
- The result is **1073741822** (unsigned value).

ii) && (Logical AND Operator)

The && operator is a **logical AND** operator used in programming languages like Java, C, and JavaScript. It evaluates two boolean expressions and returns true only if both expressions are true. This operator is **short-circuit**, meaning if the first condition is false, the second condition is not evaluated.

- **Operation**: Logical AND.
- **Example in Java**:

```
public class Main {
    public static void main(String[] args) {
        int a = 5, b = 10;
        if (a > 2 && b < 20) {
            System.out.println("Both conditions are true");
```

	} } } **Output**: Both conditions are true **Explanation**: • a > 2 evaluates to true. • b < 20 evaluates to true. • Both conditions are true, so the if block executes. **iii) >> (Signed Right Shift Operator)** The >> operator shifts the bits of a number to the right, **preserving the sign**. It fills the leftmost bits with 0 if the number is positive or 1 if the number is negative. This operator is used in many programming languages like Java and Python. • **Operation**: Signed right shift. • **Example in Java**: public class Main { public static void main(String[] args) { int num = -8; // Binary: 11111111 11111111 11111111 11111000 int result = num >> 2; // Shift 2 bits right: 11111111 11111111 11111111 11111110 System.out.println(result); // Output: -2 } } **Output**: -2 **Explanation**: • -8 in binary (32-bit): 11111111 11111111 11111111 11111000 • After shifting right by 2 bits: 11111111 11111111 11111111 11111110 (sign bit remains 1). • The result is **-2**.			
1b	What is type casting? Illustrate with an example, what is meant by automatic Type Casting? Type casting is the process of converting a variable from one data type to another. It can either be explicit (manual) or implicit (automatic). Explicit Type Casting: The programmer explicitly specifies the conversion. Implicit Type Casting (Automatic Typecasting): The programming language automatically converts the type when it is safe to do so, such as converting smaller data types to larger data types (e.g., int to float). Explicit Type Casting The programmer explicitly specifies the conversion using syntax like (targetType). Example in Java:	[5]	CO1	L1

	public class Main { public static void main(String[] args) { double num = 9.7; int intNum = (int) num; // Explicitly cast double to int System.out.println(intNum); // Output: 9 } } Explanation: double num = 9.7: The variable num is of type double. (int) num: Converts double to int, truncating the fractional part. Automatic Typecasting (Implicit Casting) In automatic typecasting, the conversion is done by the compiler or interpreter without explicit instruction. This happens when the conversion does not result in data loss, such as widening conversions (e.g., int to double). Example in Java: public class Main { public static void main(String[] args) { int num = 10; double doubleNum = num; // Automatic typecasting from int to double System.out.println(doubleNum); // Output: 10.0 } } Explanation: int num = 10: The variable num is of type int. double doubleNum = num: Automatically converts int to double without data loss.			
2a	**Discuss any three OOP Principles.** Explanation on Encapsulation,Abstraction,Polymorphism	[5]	CO1	L1
2b	**Write a java program to add two matrices of the same size by scanning the order of N and elements of an array through the console.** import java.util.Scanner; public class MatrixAddition { public static void main(String[] args) { Scanner scanner = new Scanner(System.in); // Input the order of the matrix (N x N) System.out.print("Enter the number of rows (N): "); int rows = scanner.nextInt();	[5]	CO1	L1

```
System.out.print("Enter the number of columns (N): ");
int columns = scanner.nextInt();

// Declare matrices
int[][] matrix1 = new int[rows][columns];
int[][] matrix2 = new int[rows][columns];
int[][] result = new int[rows][columns];

// Input elements for the first matrix
System.out.println("Enter elements of the first matrix:");
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < columns; j++) {
        System.out.print("Enter element at position [" + i + "][" + j + "]: ");
        matrix1[i][j] = scanner.nextInt();
    }
}

// Input elements for the second matrix
System.out.println("Enter elements of the second matrix:");
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < columns; j++) {
        System.out.print("Enter element at position [" + i + "][" + j + "]: ");
        matrix2[i][j] = scanner.nextInt();
    }
}

// Add the two matrices
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < columns; j++) {
        result[i][j] = matrix1[i][j] + matrix2[i][j];
    }
}

// Display the result matrix
System.out.println("The resultant matrix after addition is:");
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < columns; j++) {
```

```
                System.out.print(result[i][j] + " ");
            }
            System.out.println();
        }
        scanner.close();
    }
}
```

3a	**Create a program demonstrating the use of this, super, final, and static in Java. Explain the role of each keyword in the code and discuss scenarios where each keyword would be essential.**	[5]	CO2	L3

```
class Parent {
    // Static variable
    static int staticVar = 100;

    // Final variable
    final String finalMessage = "This is a final variable";

    // Method in the parent class
    void display() {
        System.out.println("This is the parent class display method.");
    }
}

class Child extends Parent {
    String name;

    // Constructor using 'this'
    Child(String name) {
        this.name = name; // 'this' refers to the current instance
    }

    // Overriding method using 'super'
    @Override
    void display() {
        super.display(); // 'super' calls the parent class method
        System.out.println("This is the child class display method.");
    }
```

	`` void showDetails() { System.out.println("Child name: " + this.name); // 'this' to refer to instance variable System.out.println("Accessing final variable: " + finalMessage); System.out.println("Accessing static variable: " + staticVar); } } public class KeywordsDemo { // Static method static void staticMethod() { System.out.println("This is a static method."); } public static void main(String[] args) { // Create an instance of the child class Child child = new Child("John"); // Demonstrate 'this' and 'super' child.display(); child.showDetails(); // Demonstrate static method KeywordsDemo.staticMethod(); } } ```			
3b	**Develop a java program to find the factorial of a given number using recursion.**	[5]	CO2	L3
4a	**Explain in brief about types of variables supported by Java and its scope with suitable Java Program.** Explaining in Brief with example about 1. Local Variable, 2. Instance Variable and 3. Static Variable	[5]	CO2	L1
4b	**Explain in brief different Access Controls supported by Java.** Explaining in Brief about Default,Public,Protected and Private Access Specifier	[5]	CO2	L1
5a	**Explain any three types of inheritance with suitable programs.** Explaining 1) Single Inheritance 2) Multilevel Inheritance 3) Hierarchical Inheritance	[5]	CO3	L1

	with programming Example			
5b	**Explain different types of Polymorphism with suitable programming examples.** **Polymorphism** in Java refers to the ability of a single action to behave differently depending on the object or context. It allows objects to take many forms and is broadly classified into two types: --- **1. Compile-Time Polymorphism (Method Overloading)** ● **Definition**: Achieved by defining multiple methods with the same name but different parameter lists (type, number, or both) within the same class. ● **Binding**: The method to call is determined at compile time. ● **Usage**: Increases code readability and usability. **Example**: class Calculator { // Method with one parameter int add(int a) { return a + a; } // Method with two parameters int add(int a, int b) { return a + b; } // Method with three parameters int add(int a, int b, int c) { return a + b + c; } } public class CompileTimePolymorphism { public static void main(String[] args) { Calculator calc = new Calculator(); System.out.println("Addition of one number: " + calc.add(5)); // Calls the first method System.out.println("Addition of two numbers: " + calc.add(5, 10)); // Calls the second method System.out.println("Addition of three numbers: " + calc.add(5, 10, 15)); // Calls the third method }	[5]	CO3	L1

```
}
```

Output:

Addition of one number: 10

Addition of two numbers: 15

Addition of three numbers: 30

---

2. Runtime Polymorphism (Method Overriding)

- **Definition**: Achieved by defining a method in a child class with the same name, return type, and parameters as in its parent class.
- **Binding**: The method to call is determined at runtime based on the object.
- **Usage**: Implements dynamic method dispatch and supports runtime decisions.

Example:

```
class Animal {
    void sound() {
        System.out.println("Animals make sounds.");
    }
}

class Dog extends Animal {
    @Override
    void sound() {
        System.out.println("Dog barks.");
    }
}

class Cat extends Animal {
    @Override
    void sound() {
        System.out.println("Cat meows.");
    }
}

public class RuntimePolymorphism {
    public static void main(String[] args) {
        Animal animal;

        // Runtime polymorphism
        animal = new Dog();
```

	animal.sound(); // Calls the Dog's overridden method animal = new Cat(); animal.sound(); // Calls the Cat's overridden method } } **Output**: Dog barks. Cat meows			
6	**Develop a JAVA program to create an abstract class Shape with abstract methods calculateArea() and calculatePerimeter(). Create subclasses Circle and Triangle that extend the Shape class and implement the respective methods to calculate the area and perimeter of each shape.** // Abstract class Shape abstract class Shape { // Abstract methods to calculate area and perimeter public abstract double calculateArea(); public abstract double calculatePerimeter(); } // Subclass Circle that extends Shape class Circle extends Shape { private double radius; // Constructor to initialize the radius public Circle(double radius) { this.radius = radius; } // Implementing the calculateArea method @Override public double calculateArea() { return Math.PI * radius * radius; }	[10]	CO3	L3

```
    // Implementing the calculatePerimeter method
    @Override
    public double calculatePerimeter() {
        return 2 * Math.PI * radius;
    }
}

// Subclass Triangle that extends Shape
class Triangle extends Shape {
    private double side1, side2, side3;

    // Constructor to initialize the sides of the triangle
    public Triangle(double side1, double side2, double side3) {
        this.side1 = side1;
        this.side2 = side2;
        this.side3 = side3;
    }

    // Implementing the calculateArea method using Heron's formula
    @Override
    public double calculateArea() {
        double semiPerimeter = (side1 + side2 + side3) / 2;
        return Math.sqrt(semiPerimeter * (semiPerimeter - side1) *
(semiPerimeter - side2) * (semiPerimeter - side3));
    }

    // Implementing the calculatePerimeter method
    @Override
    public double calculatePerimeter() {
        return side1 + side2 + side3;
    }
}

// Main class to test the Shape, Circle, and Triangle classes
public class Main {
    public static void main(String[] args) {
        // Create a Circle object with radius 5
```

```
        Shape circle = new Circle(5);
        System.out.println("Circle Area: " + circle.calculateArea());
        System.out.println("Circle Perimeter: " + circle.calculatePerimeter());

        // Create a Triangle object with sides 3, 4, and 5
        Shape triangle = new Triangle(3, 4, 5);
        System.out.println("Triangle Area: " + triangle.calculateArea());
        System.out.println("Triangle Perimeter: " +
triangle.calculatePerimeter());
    }
}
```

Faculty Signature CCI Signature HOD Signature