Reg.No | | | | | | | |

SNS COLLEGE OF TECHNOLOGY

A

(Autonomous)

MCA- Internal Assessment –I (April 2024)

Academic Year 2023-2024(Even) / Second Semester

23CAT606 – Java Programming - Answer key

Time: $1^{1/2}$ Hours **Maximum Marks: 50**

Answer All Questions

PART - A (5 x 2 = 10 Marks)

		CO	BL
1	**What is meant by package.** Package is a way of organizing classes and interfaces into namespaces, similar to folders in a file system. Packages help in organizing and managing Java code by grouping related classes and interfaces together.	CO1	Und
2	**Does java support multiple inheritance?** No, Java does not support multiple inheritance of classes. Interfaces in Java provide a way to achieve the benefits of multiple inheritance for specifying contracts without the complications of multiple inheritance of implementations.	CO1	Ana
3	**Show what is the purpose of the finally clause of a try-catch-finally statement?** The finally clause in a try-catch-finally statement in Java is used to define a block of code that will be executed whether an exception is thrown or not.	CO1	Ana
4	**Assess what happens when we call repaint() method.** In Java, the repaint() method is used to request that a component or a portion of a component be redrawn.	CO2	Eva
5	**What is the lifecycle of an Applet?** Init(), Start(), Run(), Stop(), Destroy()	CO2	Und

PART - B (2 x 13 = 26, 1x 14=14Marks)

6 (a) **Narrate any one control statement and looping and write a program to perform Sum of all values in an array and find the greatest number in an array.** CO1 App

If Statement Syntax:

The "if" statement is used for conditional execution. It evaluates a boolean expression and executes a block of code if the expression evaluates to true. Optionally, an "else" block can be added to execute a different block of code if the expression evaluates to false.

```
if (booleanExpression) {
   // Code block to execute if booleanExpression is true
} else {
   // Code block to execute if booleanExpression is false
}
```

For Looping Syntax:

The "for" loop in Java is used to iterate over a range of values or elements. It consists of an initialization statement, a boolean expression (loop condition), an iteration statement, and a code block.

```
for (initialization; booleanExpression; iteration) {
   // Code block to execute repeatedly
}
```

For the control statement, we'll use the "if" statement, and for the looping construct, we'll use the "for" loop.

```
public class ArrayOperations {
   public static void main(String[] args) {
      // Define an array of integers
      int[] numbers = { 10, 7, 15, 23, 8, 17, 12 };

      // Calculate the sum of all values in the array
      int sum = 0;
      for (int num : numbers) {
         sum += num;
      }
```

```
        // Print the sum
        System.out.println("Sum of all values in the array: " + sum);

        // Find the greatest number in the array
        int max = numbers[0]; // Assume the first element is the greatest initially
        for (int i = 1; i < numbers.length; i++) {
            if (numbers[i] > max) {
                max = numbers[i];
            }
        }

        // Print the greatest number
        System.out.println("Greatest number in the array: " + max);
    }}
```

(Or)

(b) **Develop a simple java program to illustrate the concept of packages and its types in detail.** CO1 App

Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces.

```
// import the Vector class from util package.
import java.util.vector;

// import all the classes from util package
import java.util.*;
```

Package:

1. Built in: java.lang, java.io, java.util, java.applet, java.awt, java.net
2. User defined Package:

Packages that are defined by the user. First we create a directory **myPackage** (name should be same as the name of the package). Then create the **MyClass** inside the directory with the first statement being the **package names.**

```
package myPackage;

public class MyClass
{
    public void getNames(String s)
    {
        System.out.println(s);
    }
}
import myPackage.MyClass;

public class PrintName
{
    public static void main(String args[])
    {
        // Initializing the String variable
        // with a value
        String name = "GeeksforGeeks";

        // Creating an instance of class MyClass in
        // the package.
        MyClass obj = new MyClass();

        obj.getNames(name);
    }
}
```

7 (a) **Evaluate a try block that is likely to generate three types of exception and then incorporate necessary catch blocks and handle them appropriately.** CO2 Eva

```
public class ExceptionHandlingExample {
    public static void main(String[] args) {
        try {
            // Code that may throw different types of exceptions
            int[] numbers = {1, 2, 3};
            int index = 4; // Index out of bounds
```

```
            // Division by zero
            int result = numbers[1] / (index - 3);

            // Null pointer exception
            String str = null;
            int length = str.length();

            // ArrayIndexOutOfBoundsException
            int value = numbers[4];
        } catch (ArithmeticException e) {
            // Handle division by zero
            System.out.println("ArithmeticException caught: " + e.getMessage());
        } catch (NullPointerException e) {
            // Handle null pointer exception
            System.out.println("NullPointerException caught: " + e.getMessage());
        } catch (ArrayIndexOutOfBoundsException e) {
            // Handle array index out of bounds exception
            System.out.println("ArrayIndexOutOfBoundsException caught: " + e.getMessage());
        } catch (Exception e) {
            // Catch-all block for other types of exceptions
            System.out.println("Exception caught: " + e.getMessage());
        }
    }
}
```

Explanation:

1. Inside the try block, we have code that may throw three types of exceptions:
    - `ArithmeticException`: Occurs when attempting to divide by zero.
    - `NullPointerException`: Occurs when trying to invoke a method or access a property of a null object.
    - `ArrayIndexOutOfBoundsException`: Occurs when trying to access an element of an array with an invalid index.
2. We have separate catch blocks for each type of exception:
    - `catch (ArithmeticException e)`: Catches and handles `ArithmeticException`.
    - `catch (NullPointerException e)`: Catches and handles `NullPointerException`.
    - `catch (ArrayIndexOutOfBoundsException e)`: Catches and handles `ArrayIndexOutOfBoundsException`.
3. We also have a catch block with `Exception` as its parameter to catch any other types of exceptions that were not caught by the specific catch blocks above.

(Or)

(b) **Can you write a program to draw 2D geometrical shapes and Fill those shapes with solid colors.** CO2 Eva

Java program using the Java 2D API to draw 2D geometrical shapes (a rectangle, an ellipse, and a polygon) and fill those shapes with solid colors.

```
import javax.swing.*;
import java.awt.*;

public class DrawShapes extends JPanel {

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;

        // Set rendering hints for better quality
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
```

```
        // Set colors
        g2d.setColor(Color.RED);
        // Draw and fill a rectangle
        g2d.fillRect(50, 50, 100, 80);
        // Set colors
        g2d.setColor(Color.BLUE);
        // Draw and fill an ellipse
        g2d.fillOval(200, 50, 100, 80);
        // Set colors
        g2d.setColor(Color.GREEN);
        // Define the points of a polygon
        int[] xPoints = {350, 400, 450, 400};
        int[] yPoints = {50, 150, 150, 50};
        int nPoints = 4;
        // Draw and fill a polygon
        g2d.fillPolygon(xPoints, yPoints, nPoints);
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("Draw Shapes");
            frame.setSize(500, 200);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.add(new DrawShapes());
            frame.setVisible(true);
        });
    }
}
```

8 (a) **How would you find a given two one dimensional arrays A and B which are sorted in ascending order. Write a Java program to merge them into a single sorted array, see that is contains every item from array A and B, in ascending order.** CO1 App

To merge two sorted arrays A and B into a single sorted array containing all elements from both arrays, you can use a merging algorithm similar to the merge step of merge sort.

```
public class MergeSortedArrays {
    public static int[] mergeArrays(int[] A, int[] B) {
        int[] mergedArray = new int[A.length + B.length];
        int i = 0, j = 0, k = 0;

        // Merge elements from arrays A and B into mergedArray
        while (i < A.length && j < B.length) {
            if (A[i] <= B[j]) {
                mergedArray[k++] = A[i++];
            } else {
                mergedArray[k++] = B[j++];
            }
        }

        // Copy remaining elements from array A, if any
        while (i < A.length) {
            mergedArray[k++] = A[i++];
        }

        // Copy remaining elements from array B, if any
        while (j < B.length) {
            mergedArray[k++] = B[j++];
        }

        return mergedArray;
    }

    public static void main(String[] args) {
        int[] A = {1, 3, 5, 7, 9};
        int[] B = {2, 4, 6, 8, 10};

        int[] mergedArray = mergeArrays(A, B);
```

```
        // Print the merged array
        System.out.println("Merged Array:");
        for (int num : mergedArray) {
            System.out.print(num + " ");
        }
    }
}
```

Explanation:

1. The mergeArrays method takes two sorted arrays A and B as input and returns a single sorted array containing all elements from both arrays.
2. Inside the mergeArrays method:
    - We create a new array mergedArray to store the merged result. Its length is the sum of the lengths of arrays A and B.
    - We use three pointers i, j, and k to track the positions in arrays A, B, and mergedArray, respectively.
    - We compare elements from arrays A and B one by one, adding the smaller element to mergedArray and advancing the corresponding pointer.
    - After merging, if any elements are left in either array A or B, we copy them to mergedArray.
3. In the main method, we create two sorted arrays A and B, call the mergeArrays method to merge them, and print the merged array.

(Or)

(b) **Classify the components used in AWT. Design the Student registration form using AWT components and containers.** CO2 App

In AWT (Abstract Window Toolkit), components are GUI elements like buttons, labels, text fields, checkboxes, radio buttons, etc., while containers are components that can contain other components. AWT provides several components and containers for building GUI applications. Here's a classification of some commonly used components and containers in AWT:

Components:

1. **Button**: A clickable button that triggers an action when clicked.
2. **Label**: A non-interactive display area for text or images.
3. **TextField**: A single-line text input field for user input.
4. **TextArea**: A multi-line text input area for user input.
5. **Checkbox**: A checkbox for selecting one or more options.
6. **RadioButton**: A radio button for selecting one option from multiple options.
7. **List**: A list box for displaying a list of items, allowing single or multiple selection.
8. **Choice**: A drop-down list of selectable items.
9. **Scrollbar**: A scrollbar for scrolling content.
10. **Canvas**: A blank area for drawing graphics or custom components.

Containers:

1. **Frame**: A top-level window with a title bar and border.
2. **Panel**: A generic container for grouping and organizing components.
3. **Window**: An abstract base class for top-level windows.
4. **Dialog**: A top-level window with a title bar that can be modal or modeless.
5. **Applet**: A container for running Java applets within a web browser.

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

public class StudentRegistrationForm extends Frame {
```

```
    private Label nameLabel, emailLabel, genderLabel;
    private TextField nameField, emailField;
    private Choice genderChoice;
    private Button submitButton;

    public StudentRegistrationForm() {
        setTitle("Student Registration Form");
        setSize(400, 200);
        setLayout(new GridLayout(4, 2));

        nameLabel = new Label("Name:");
        add(nameLabel);

        nameField = new TextField();
        add(nameField);

        emailLabel = new Label("Email:");
        add(emailLabel);

        emailField = new TextField();
        add(emailField);

        genderLabel = new Label("Gender:");
        add(genderLabel);

        genderChoice = new Choice();
        genderChoice.add("Male");
        genderChoice.add("Female");
        genderChoice.add("Other");
        add(genderChoice);

        submitButton = new Button("Submit");
        submitButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String name = nameField.getText();
                String email = emailField.getText();
                String gender = genderChoice.getSelectedItem();

                System.out.println("Name: " + name);
                System.out.println("Email: " + email);
                System.out.println("Gender: " + gender);
            }
        });
        add(submitButton);

        setVisible(true);
    }

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

Reg.No | | | | | | | |

SNS COLLEGE OF TECHNOLOGY

(Autonomous)

B

MCA- Internal Assessment –I (April 2024)

Academic Year 2023-2024(Even) / Second Semester

23CAT606 – Java Programming – Answer Key

Time: 1 1/2 Hours **Maximum Marks: 50**

Answer All Questions

PART - A (5 x 2 = 10 Marks)

		CO	BL
1	**Java is platform Independent. JVM is platform dependent. Justify** Java's platform independence stems from its design principle of "write once, run anywhere" (WORA). This means that Java programs written on one platform (such as Windows) can be executed on any other platform (such as Linux or macOS) without modification, as long as a Java Virtual Machine (JVM) is available for that platform. JVM itself is platform-dependent, with separate implementations tailored to different operating systems and hardware architectures.	CO1	Ana
2	**Differentiate class and interface.** • **Class**: A class in Java is a blueprint or template for creating objects. It defines the properties (fields) and behaviors (methods) that objects of that class will have. • **Interface**: An interface in Java is a reference type that defines a set of abstract methods that a class implementing the interface must provide. It specifies a contract for what a class can do but does not provide the implementation of those methods.	CO1	Ana
3	**Assess why java generates unreachable code error during multiple catch statements?** In Java, the "unreachable code" error occurs when the compiler detects that a certain block of code cannot be executed under any circumstances. This typically happens when there's a logical flow issue in the code, such as unreachable statements after a `return`, `throw`, `break`, or `continue` statement.	CO1	Eva
4	**State the order of execution of applet methods.** Init(), Start(), Run(), Stop() and Destroy	CO2	Rem
5	Which containers use a border Layout as their default layout? The containers that use BorderLayout as their default layout manager include: 1. **Frame**: A top-level window with a title bar and border. 2. **Window**: An abstract base class for top-level windows. 3. **Dialog**: A top-level window with a title bar that can be modal or modeless. 4. **Applet**: A container for running Java applets within a web browser. `frame.add(new Button("North"), BorderLayout.NORTH);`	CO2	Und

PART - B (2 x 13 = 26, 1x14=14 Marks)

6	(a) Classify conditional and looping statements in java. 1. **Conditional Statements**: o **if**: Executes a block of code if a specified condition is true. o **if-else**: Executes one block of code if a specified condition is true and another block if the condition is false.	CO1	Ana

- **if-else if-else**: Executes one of several blocks of code, depending on the evaluation of multiple conditions.

```
if (x > 0) {
            System.out.println("x is positive");
        } else if (x == 0) {
            System.out.println("x is zero");
        } else {
            System.out.println("x is negative");
        }
char grade = 'B';
        switch (grade) {
            case 'A':
                System.out.println("Excellent");
                break;
            case 'B':
                System.out.println("Good");
                break;
            case 'C':
                System.out.println("Fair");
                break;
            default:
                System.out.println("Need Improvement");
        }
for (int i = 1; i <= 5; i++) {
            System.out.println("Iteration " + i);
        }

        int j = 1;
        while (j <= 5) {
            System.out.println("While Loop Iteration " + j);
            j++;
        }

        int k = 1;
        do {
            System.out.println("Do-While Loop Iteration " + k);
            k++;
        } while (k <= 5);
```

switch: Evaluates the value of an expression and executes a block of code based on the matched case value.

2. **Looping Statements**:
   - **for**: Executes a block of code a specified number of times. It consists of an initialization statement, a boolean expression (loop condition), an iteration statement, and a code block.
   - **while**: Executes a block of code as long as a specified condition is true. It evaluates the condition before each iteration.
   - **do-while**: Similar to the while loop, but it executes the block of code at least once before evaluating the loop condition.
   - **for-each (enhanced for loop)**: Iterates over elements of an array or a collection sequentially without using an explicit loop counter. It simplifies iterating over arrays and collections.

(Or)

(b) Inference the use of Interface in java program with an example. CO1 Ana

Interfaces in Java provide a way to define a contract for classes. They specify a set of methods that a class implementing the interface must implement. This allows for achieving abstraction and polymorphism in Java programs.

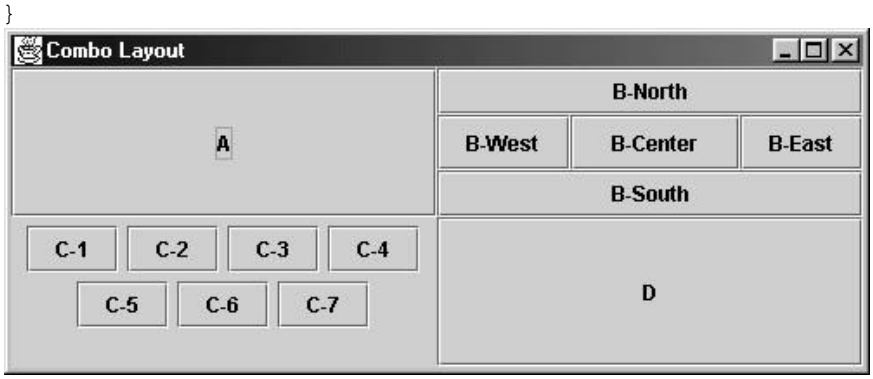
```
public class Rectangle implements Shape {
    private double length;
    private double width;

    public Rectangle(double length, double width) {
```

```
        this.length = length;
        this.width = width;
    }

    @Override
    public double calculateArea() {
        return length * width;
    }

    @Override
    public double calculatePerimeter() {
        return 2 * (length + width);
    }
}
// Circle.java
public class Circle implements Shape {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    @Override
    public double calculateArea() {
        return Math.PI * radius * radius;
    }

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

7 (a) Write a java program to stimulate the layouts in AWT CO2 App

```
import java.awt.*;

public class AWTLayoutExample {
    public AWTLayoutExample() {
        Frame frame = new Frame("AWT Layout Example");

        // FlowLayout
        Panel flowLayoutPanel = new Panel(new FlowLayout());
        flowLayoutPanel.add(new Button("Button 1"));
        flowLayoutPanel.add(new Button("Button 2"));
        flowLayoutPanel.add(new Button("Button 3"));
        frame.add(flowLayoutPanel, BorderLayout.NORTH);

        // BorderLayout
        Panel borderLayoutPanel = new Panel(new BorderLayout());
        borderLayoutPanel.add(new Button("North"),
BorderLayout.NORTH);
        borderLayoutPanel.add(new Button("South"),
BorderLayout.SOUTH);
        borderLayoutPanel.add(new Button("East"),
BorderLayout.EAST);
        borderLayoutPanel.add(new Button("West"),
BorderLayout.WEST);
        borderLayoutPanel.add(new Button("Center"),
BorderLayout.CENTER);
        frame.add(borderLayoutPanel, BorderLayout.CENTER);

        // GridLayout
        Panel gridLayoutPanel = new Panel(new GridLayout(2, 3));
        gridLayoutPanel.add(new Button("Button 1"));
        gridLayoutPanel.add(new Button("Button 2"));
        gridLayoutPanel.add(new Button("Button 3"));
        gridLayoutPanel.add(new Button("Button 4"));
        gridLayoutPanel.add(new Button("Button 5"));
```

```
        gridLayoutPanel.add(new Button("Button 6"));
        frame.add(gridLayoutPanel, BorderLayout.SOUTH);

        // CardLayout
        Panel cardLayoutPanel = new Panel(new CardLayout());
        cardLayoutPanel.add(new Button("Card 1"), "card1");
        cardLayoutPanel.add(new Button("Card 2"), "card2");
        cardLayoutPanel.add(new Button("Card 3"), "card3");
        frame.add(cardLayoutPanel, BorderLayout.WEST);

        frame.setSize(400, 300);
        frame.setVisible(true);
    }

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



(Or)

(b) Develop a java program to design the APPLET window and explain the life cycle. CO2 App

```
import java.applet.*;
import java.awt.*;

public class MyApplet extends Applet {

    // Initialization method
    public void init() {
        System.out.println("Applet initialized");
    }

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

    // Paint method
    public void paint(Graphics g) {
        System.out.println("Applet painted");
        g.drawString("Hello, World!", 20, 20);
    }

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

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

8 (a) Create a simple java program to illustrate the concept of packages and its types in detail. CO1 App

Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces.

```
// import the Vector class from util package.
import java.util.vector;

// import all the classes from util package
import java.util.*;
```

Package:

1. Built in: java.lang, java.io, java.util, java.applet, java.awt, java.net
2. User defined Package:

Packages that are defined by the user. First we create a directory **myPackage** (name should be same as the name of the package). Then create the **MyClass** inside the directory with the first statement being the **package names.**

```
package myPackage;

public class MyClass
{
   public void getNames(String s)
   {
      System.out.println(s);
   }
}
import myPackage.MyClass;

public class PrintName
{
  public static void main(String args[])
  {
    // Initializing the String variable
    // with a value
    String name = "GeeksforGeeks";

    // Creating an instance of class MyClass in
    // the package.
    MyClass obj = new MyClass();

    obj.getNames(name);
  }
}
```

(Or)

(b) Build a program to create a frame with the following menus, such that the corresponding geometric object is created when a menu is clicked. a. Circle. b. Rectangle. c. Line. d. Diagonal for the rectangle. CO2 App

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

public class GeometricObjectCreator extends JFrame implements
ActionListener {
    public GeometricObjectCreator() {
        setTitle("Geometric Object Creator");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Create menu bar
        JMenuBar menuBar = new JMenuBar();

        // Create menus
        JMenu shapeMenu = new JMenu("Shapes");

        // Create menu items
        JMenuItem circleItem = new JMenuItem("Circle");
        circleItem.addActionListener(this);
        shapeMenu.add(circleItem);

        JMenuItem rectangleItem = new JMenuItem("Rectangle");
        rectangleItem.addActionListener(this);
        shapeMenu.add(rectangleItem);
```

```
        JMenuItem lineItem = new JMenuItem("Line");
        lineItem.addActionListener(this);
        shapeMenu.add(lineItem);

        JMenuItem diagonalItem = new JMenuItem("Diagonal for
Rectangle");
        diagonalItem.addActionListener(this);
        shapeMenu.add(diagonalItem);

        // Add menus to menu bar
        menuBar.add(shapeMenu);

        // Set menu bar
        setJMenuBar(menuBar);
    }

    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();

        switch (command) {
            case "Circle":
                JOptionPane.showMessageDialog(this, "Circle
created");
                // Code to create circle object
                break;
            case "Rectangle":
                JOptionPane.showMessageDialog(this, "Rectangle
created");
                // Code to create rectangle object
                break;
            case "Line":
                JOptionPane.showMessageDialog(this, "Line
created");
                // Code to create line object
                break;
            case "Diagonal for Rectangle":
                JOptionPane.showMessageDialog(this, "Diagonal for
Rectangle created");
                // Code to create diagonal for rectangle object
                break;
            default:
                break;
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            GeometricObjectCreator frame = new
GeometricObjectCreator();
            frame.setVisible(true);
        });
    }
}
```