OOPS ANSWERS

1(a) Explain the lifecycle of an applet. Write a simple Applet program.

Answer:

An applet is a Java program that can run in a web browser or an applet viewer. The lifecycle of an applet is managed by specific methods defined in the Applet class, which allow the applet to be started, stopped, and destroyed.

1. init(): Called when the applet is first loaded. Used to initialize variables and set up the applet environment.

2. start(): Called each time the applet becomes active, usually when the applet's page is visited or revisited.

3. stop(): Called when the applet becomes inactive, such as when navigating away from the page.

4. destroy(): Called when the applet is being removed from memory, typically when the browser is closed.

Example Applet Code:

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

public class HelloWorldApplet 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("Hello, World!", 20, 20);
    }

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

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

1(b) What is the collection framework? Draw the collection framework hierarchy.

Answer:

The Java Collection Framework is a set of classes and interfaces that provide a standard way to manage groups of objects. It is part of the java.util package and includes data structures such as lists, sets, queues, and maps.

1. Purpose: Simplifies the handling of collections of data by offering reusable interfaces and classes.

2. Hierarchy:

Collection Interface: Base interface that includes List, Set, and Queue.

List Interface: An ordered collection (e.g., ArrayList, LinkedList).

Set Interface: A collection with no duplicates (e.g., HashSet, TreeSet).

Queue Interface: A collection for holding elements prior to processing (e.g., PriorityQueue).

Map Interface: A key-value pair collection (e.g., HashMap, TreeMap).

Collection Framework Hierarchy Diagram:

Collection

├— List

│ ├— ArrayList

│ └── LinkedList

├── Set

│ ├── HashSet

│ └── TreeSet

└── Queue

└── PriorityQueue

Map

├── HashMap

└── TreeMap

2(a) What is AWT? Write AWT classes with a program.

Answer:

AWT (Abstract Window Toolkit) is a Java API used for creating graphical user interfaces (GUIs) for Java applications. It is part of the java.awt package and provides components such as windows, buttons, text fields, etc.

1. Core AWT Classes:

Frame: Represents the main window.

Button: Creates a button.

Label: Displays a text label.

TextField: Allows text input.

Panel: A container for grouping components.

Example AWT Program:

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

public class AWTExample {
   public static void main(String[] args) {
      Frame frame = new Frame("AWT Example");
      Button button = new Button("Click Me");
      TextField textField = new TextField(20);

      button.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            textField.setText("Button Clicked!");
         }
      });
```

```
        frame.add(button, BorderLayout.NORTH);
        frame.add(textField, BorderLayout.SOUTH);

        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}
```

2(b) Explain the event listener interface with a program.

Answer:

In Java AWT, an event listener is an interface that listens to events generated by GUI components like buttons or text fields. When a user interacts with a component, the corresponding listener handles the event.

1. Common Event Listener Interfaces:

ActionListener: For handling button clicks.

MouseListener: For handling mouse events.

KeyListener: For handling keyboard events.

Example Program Using ActionListener:

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

public class EventListenerExample {
   public static void main(String[] args) {
      Frame frame = new Frame("Event Listener Example");
      Button button = new Button("Press Me");

      button.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            System.out.println("Button was pressed!");
         }
      });

      frame.add(button, BorderLayout.CENTER);
      frame.setSize(200, 100);
      frame.setVisible(true);
   }
}
```

3(a) Write short notes on JButton and JList.

Answer:

1. JButton:

A Swing component that represents a button. It is part of the javax.swing package.

When clicked, a JButton triggers an action event, which can be handled by implementing an ActionListener.

Example:

```
JButton button = new JButton("Click Me");
button.addActionListener(e -> System.out.println("Button Clicked"));
```

2. JList:

A Swing component used to display a list of items from which a user can select one or more.

Supports single or multiple-selection modes.

Example:

String[] items = {"Item 1", "Item 2", "Item 3"};

JList<String> list = new JList<>(items);

3(b) What is StringTokenizer?

Answer:

The StringTokenizer class in Java is used to break a string into tokens based on a specified delimiter.

1. Purpose: Commonly used for parsing strings into individual segments.

2. Constructor Options:

StringTokenizer(String str): Uses space as the default delimiter.

StringTokenizer(String str, String delimiter): Allows specifying a custom delimiter.

3. Methods:

nextToken(): Returns the next token.

hasMoreTokens(): Checks if more tokens are available.

Example:

```
String str = "Hello,World,Java";
StringTokenizer st = new StringTokenizer(str, ",");
while (st.hasMoreTokens()) {
   System.out.println(st.nextToken());
}
// Output: Hello World Java
```

4(a) Explain about JDBC architecture.

Answer:

JDBC (Java Database Connectivity) is an API that allows Java applications to connect and interact with databases.

1. Components:

JDBC API Layer: Provides methods to perform database operations.

JDBC Driver Manager: Manages drivers and connects to the appropriate driver.

JDBC Drivers: Translate Java calls into database-specific calls (e.g., Type 1 to Type 4 drivers).

Database: Backend where data is stored.

Example:

```
Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb", "user",
"password");

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("SELECT * FROM employees");

while (rs.next()) {

   System.out.println(rs.getString("name"));

}

conn.close();
```

4(b) Explain about Daemon Threads.

Answer:

A Daemon Thread is a background thread that performs system-level tasks and does not prevent the JVM from exiting when the application finishes.

1. Purpose: Used for supporting tasks like garbage collection.

2. Lifecycle: Ends automatically when there are no active user threads.

3. Setting a Daemon Thread: Can be marked as daemon using setDaemon(true).

Example:

```
public class DaemonExample {
   public static void main(String[] args) {
      Thread daemonThread = new Thread(() -> {
         while (true) {
            System.out.println("Daemon thread running...");
         }
      });
      daemonThread.setDaemon(true);
      daemonThread.start();
      System.out.println("Main thread ending...");
   }
}
```