The
Gentlemen Team
(Dipak Shah, Prakash Bista & Suresh Rawat)
(COEM Student's)

Advanced Programming with Java Old Questions Solution 2021 to 2017

2021 'Fall'

1. What do you mean by architectural-neutral? What are wrapper classes? Explain. 7

Answer:

Architectural-neutral is a term that describes a program or a language that can run on different types of hardware or software platforms without any changes. For example, Java is an architectural-neutral language because it compiles into bytecode, which can be executed by a Java Virtual Machine (JVM) on any platform that supports JVM. This means that a Java program can run on Windows, Linux, Mac, or any other operating system as long as they have JVM installed. This makes Java portable and adaptable to various environments.

Java Wrapper Class:

Java uses primitive types such as int or double, to hold the basic data types supported by the language. Primitive types, rather than objects, are used for these quantities for the sake of performance. Using objects for these values would add an unacceptable overhead to even the simplest of calculations. Thus, the primitive types are not part of the object hierarchy, and they do not inherit Object.

Despite the performance benefit offered by the primitive types, there are times when you will need an object representation. For example, you can't pass a primitive type by reference to a method. Also, many of the standard data structures implemented by Java operate on objects, which means that you can't use these data structures to store primitive types. To handle these (and other) situations, Java provides type wrappers, which are classes that encapsulate a primitive type within an object.

Wrapper class provides a mechanism to convert primitive data types into wrapper class objects.

Need of Java Wrapper Class

- They convert the primitive data types into objects.
- Objects are needed to modify the arguments in a method.
- The classes in java.util package only works with objects.
- Data structures in the collection framework only store objects.
- Objects helps in synchronization in multithreading.

b) What is Singleton class? What is the difference between abstract class and interface? 8

Answer:

A singleton class in Java is a special class that allows only one instance (or object) of itself to be created. Imagine it like a unique key that opens a special door. No matter how many times you try to create a new object from a singleton class, you'll always get the same instance that was created initially.

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.
9)**Example:**	**Example:**

public abstract class Shape{
public abstract void draw();
}
	public interface Drawable{
	void draw();
	}

2. **a) Define Inheritance and Composition. With the help of a suitable program illustrate the difference between them.** **7**

Answer:

Inheritance is an important part of OOPs (Object Oriented programming system). In Java, it is a mechanism in which one object acquires all the properties and behaviors of a parent object.

It shows the IS-A relationship, which is also called as parent-child relationship. As an instance, the car is a vehicle, the cat is an animal, and many more.

In Java, with the help of inheritance, we can create new classes that are built upon existing classes. When we inherit from an existing class, the methods and fields of the parent class can be reused. We can also add new methods and fields to our current class.

Inheritance is used for method overriding in order to achieve the runtime polymorphism. It is also used for code reusability. Code reusability is a way that facilitates us to reuse the fields and methods of the existing class when we create a new class. We can use the same fields and methods already defined in the previous class.

The extends keyword represents the creation of a new class that is derived from an existing class. The word "extends" means to increase the functionality.

In Java, there are three types of inheritance on the basis of class: single inheritance, multilevel inheritance, and hierarchical inheritance. Multiple inheritance and hybrid inheritance can be supported by using interface in Java. It is not supported through class in Java.

Example of inheritance:

```
class School{
    String name = "John";
    }
    class Student extends School{
     int rollno = 25;
     public static void main(String args[]){
    Student s1 = new Student();
```

```
        System.out.println();
        System.out.println("Student's Name: " + s1.name);
        System.out.println("Student's Roll.no.: " + s1.rollno);
    }
    }
```

Composition

Inheritance and Composition both are design techniques. The Composition is a way to design or implement the "has-a" relationship whereas, the Inheritance implements the "is-a" relationship. The "has-a" relationship is used to ensure the code reusability in our program. In Composition, we use an instance variable that refers to another object.

Although both inheritance and composition provide the code reusability, the difference between both terms is that in composition, we do not extend the class. The composition relationship of two objects is possible when one object contains another object, and that object is fully dependent on it. The composition represents the part of relationship. For instance, A house ***has a*** living room (living room is a part of house), A person ***has a*** heart (heart is a part of the human body), and many more.

Composition has various benefits like it allows code reusability, it is helpful in achieving the multiple inheritance, it provides better test-ability of a class. It also allows us to easily replace the composed class implementation with a better and improved version. Using composition concept, we can dynamically change the behavior of our program by changing the member objects at run time.

Example:

```
class Car
    {
        String part;
          public void getData(String name)
      {
        part = name;
        System.out.println("Car has a "+part);
    }
     }
```

```
class Part
{
        public String getPartName()
 {
          return "Engine";
}
}
        public class Composition {
          public static void main(String[] args) {
        Car c1 = new Car();
           Part p = new Part();
          c1.getData(p.getPartName());
   }
 }
```

b) What is method overloading? Can you override a private or static method in Java? Explain with an example. **8**

Answer:

Overloading: Overloading is also a feature of OOP languages like Java that is related to compile-time (or static) polymorphism. This feature allows different methods to have the same name, but different signatures, especially the number of input parameters and type of input parameters.

In Java, you cannot override a private or static method in the traditional sense:

Private methods: These methods are not visible to subclasses and are thus not subject to overriding. If a subclass defines a method with the same signature as a private method in the superclass, it's considered a new method, not an override.

Static methods: While you can declare a static method with the same signature in a subclass, it doesn't constitute overriding; it's called method hiding. The static method in the subclass hides the static method in the superclass, and which method is called depends on the reference type at compile time, not the actual object type at runtime. This is because static methods are resolved at compile time based on the reference type, not the object type.

Example:

```
class Superclass {
   private void privateMethod() {
```

```
        System.out.println("Private method in superclass");
    }

    static void staticMethod() {
        System.out.println("Static method in superclass");
    }
}

class Subclass extends Superclass {
    // Attempt to override private method - Not possible
    /* @Override
    private void privateMethod() {
        System.out.println("Private method in subclass");
    } */

    // Method hiding for static method
    static void staticMethod() {
        System.out.println("Static method in subclass");
    }
}

public class Main {
    public static void main(String[] args) {
        Superclass superObj = new Superclass();
        Subclass subObj = new Subclass();

        superObj.staticMethod(); // Output: Static method in superclass
        subObj.staticMethod();   // Output: Static method in subclass (method hiding)

        // Attempt to call private method from subclass - Not possible
        // subObj.privateMethod(); // Error: privateMethod() has private access in
Superclass
    }
}
```

3. a) Explain about File Reader and Buffered Writer class. How do you create your own exception class? Explain with suitable programming example. 8

Answer:

```
// Define your custom exception class by extending Exception or one of its subclasses
class CustomException extends Exception {
  // Constructor with a message
  public CustomException(String message) {
    super(message);
  }
}

public class Example {
  // A method that throws your custom exception
  public static void checkValue(int value) throws CustomException {
    if (value < 0) {
      // If the value is negative, throw your custom exception
      throw new CustomException("Negative value not allowed");
    }
    // If the value is non-negative, continue with the operation
    System.out.println("Value is valid: " + value);
  }

  public static void main(String[] args) {
    try {
      // Call the method that may throw your custom exception
      checkValue(-5);
    } catch (CustomException e) {
      // Catch your custom exception and handle it
      System.out.println("Custom exception caught: " + e.getMessage());
    }
  }
}
```

CustomException Class: We define a custom exception class named CustomException, which extends the Exception class. This class can have constructors to initialize the exception message and can provide additional functionality if needed.

checkValue Method: This method checks if the provided value is negative. If it's negative, it throws a new instance of CustomException with a specific error message. Otherwise, it proceeds with the operation.

Main Method: In the main method, we call checkValue with a negative value (-5). Since this value violates our condition, it throws a CustomException. We catch this exception using a try-catch block and handle it appropriately.

b) Create an applet that plays a sample audio file. Also create a GUI that allows users to play, stop or replay the audio. Add a functionality to automatically stop the audio if the applet is inactive. **7**

Answer:

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

public class AudioPlayerApplet extends Applet implements ActionListener,
Runnable {

   AudioClip audioClip;
   Button playButton, stopButton, replayButton;
   Thread audioThread;

   public void init() {
      playButton = new Button("Play");
      stopButton = new Button("Stop");
      replayButton = new Button("Replay");

      playButton.addActionListener(this);
      stopButton.addActionListener(this);
      replayButton.addActionListener(this);

      add(playButton);
      add(stopButton);
      add(replayButton);

      try {
         URL url = new URL(getCodeBase(), "sample_audio.wav");
         audioClip = Applet.newAudioClip(url);
      } catch (MalformedURLException e) {
         e.printStackTrace();
      }
   }
```

```
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == playButton) {
            if (audioThread == null) {
                audioThread = new Thread(this);
                audioThread.start();
            }
        } else if (e.getSource() == stopButton) {
            if (audioClip != null) {
                audioClip.stop();
                audioThread = null;
            }
        } else if (e.getSource() == replayButton) {
            if (audioThread != null) {
                audioClip.stop();
                audioThread = null;
            }
            audioThread = new Thread(this);
            audioThread.start();
        }
    }

    public void run() {
        audioClip.play();
    }

    public void stop() {
        if (audioThread != null) {
            audioClip.stop();
            audioThread = null;
        }
    }
}
```

4. a) What do you mean by delegation event model? Explain various event listener interfaces with an example. **7**

Answer:

The Delegation Event Model is an event handling mechanism used in Java to manage events and event listeners. In this model, components delegate the

responsibility of handling events to separate listener objects, allowing for a clean separation of concerns and more flexible event handling.

In the Delegation Event Model, there are typically three main components:

Event Source: This is the object that generates the event. It could be a graphical component like a button or a menu item.

Event Object: This object represents the event itself. It contains information about the event, such as its type and any associated data.

Event Listener: This is an object that is responsible for handling the event. It listens for events from one or more event sources and reacts accordingly.

Java provides several event listener interfaces for different types of events. Some of the commonly used event listener interfaces include:

ActionListener: This interface is used for handling action events, typically generated by components like buttons and menu items when they are clicked or activated.

MouseListener: This interface is used for handling mouse events such as mouse clicks, mouse enters, and mouse exits.

KeyListener: This interface is used for handling keyboard events such as key presses and key releases.

ComponentListener: This interface is used for handling component events such as component resizing, component moving, and component visibility changes.

FocusListener: This interface is used for handling focus events such as when a component gains or loses focus.

Here's an example that demonstrates the usage of the ActionListener interface:

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

public class ButtonExample {
   private Frame frame;
   private Button button;

   public ButtonExample() {
      frame = new Frame("Button Example");
      button = new Button("Click Me");

      // Add an ActionListener to the button
      button.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            // This method is called when the button is clicked
            System.out.println("Button clicked");
         }
      });
```

```
        // Add the button to the frame
        frame.add(button);

        // Set frame properties
        frame.setSize(300, 200);
        frame.setLayout(new FlowLayout());
        frame.setVisible(true);
    }

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

b) What advantage do java's layout managers provide over traditional windowing systems? How do you create menu in Swing? Explain with an Example. **8**

Answer:

Java's layout managers provide several advantages over traditional windowing systems:

Platform Independence: Layout managers ensure that the graphical user interface (GUI) components are arranged properly and consistently across different platforms and screen resolutions. This ensures that the GUI looks and behaves consistently regardless of the underlying operating system.

Dynamic Layouts: Layout managers allow for dynamic resizing and repositioning of GUI components. This ensures that the GUI adapts to changes in window size or screen resolution, providing a responsive user experience.

Ease of Development: Layout managers simplify the process of designing complex GUIs by automating the arrangement of components. Developers can focus on designing the functionality of the application without worrying about pixel-perfect positioning of components.

Localization Support: Layout managers support localization by allowing GUI components to adjust their size and position based on the text content. This ensures that the GUI remains readable and visually appealing when translated into different languages.

To create menus in Swing, you can use the JMenuBar, JMenu, and JMenuItem classes. Here's an example demonstrating how to create a simple menu in Swing:

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

public class MenuExample {
   public MenuExample() {
      // Create a new JFrame
      JFrame frame = new JFrame("Menu Example");

      // Create a new JMenuBar
      JMenuBar menuBar = new JMenuBar();

      // Create a new JMenu
      JMenu fileMenu = new JMenu("File");

      // Create JMenuItem objects for the File menu
      JMenuItem newItem = new JMenuItem("New");
      JMenuItem openItem = new JMenuItem("Open");
      JMenuItem saveItem = new JMenuItem("Save");
      JMenuItem exitItem = new JMenuItem("Exit");

      // Add ActionListeners to handle menu item actions
      newItem.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            // Add your action for the "New" menu item here
            JOptionPane.showMessageDialog(null, "New file created");
         }
      });

      openItem.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            // Add your action for the "Open" menu item here
            JOptionPane.showMessageDialog(null, "Open file dialog");
         }
      });

      saveItem.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            // Add your action for the "Save" menu item here
```

```
            JOptionPane.showMessageDialog(null, "Save file dialog");
        }
    });

    exitItem.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            // Add your action for the "Exit" menu item here
            int option = JOptionPane.showConfirmDialog(null, "Are you sure you want to
exit?");
            if (option == JOptionPane.YES_OPTION) {
                System.exit(0);
            }
        }
    });

    // Add menu items to the File menu
    fileMenu.add(newItem);
    fileMenu.add(openItem);
    fileMenu.add(saveItem);
    fileMenu.addSeparator(); // Add a separator line between menu items
    fileMenu.add(exitItem);

    // Add the File menu to the menu bar
    menuBar.add(fileMenu);

    // Set the menu bar for the frame
    frame.setJMenuBar(menuBar);

    // Set frame properties
    frame.setSize(300, 200);
    frame.setLayout(null); // Use null layout for demonstration purposes
    frame.setVisible(true);
  }

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

5. a) As per a poll conducted in a class of 48 students, 25 students are supportive of virtual classroom while 15 are against it and remaining 8 students chose to stay neutral. Create a pie chart to illustrate this poll result. **8**

Answer: **Out of Syllabus**

b)Write a TCP socket program to accept a line of string input from the client and send it to server. The server checks if. the string is Palindrome or not and send result back to client. Client machine displays the string received from server. **7**

Answer:

Server:

```
import java.io.*;
import java.net.*;

public class Server {
   public static void main(String[] args) {
      try {
         ServerSocket serverSocket = new ServerSocket(12345);
         System.out.println("Server is running and waiting for connections...");

         while (true) {
            Socket socket = serverSocket.accept();
            System.out.println("Client connected: " + socket);

            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

            String input = in.readLine();
            String result = isPalindrome(input) ? "Palindrome" : "Not Palindrome";
            out.println(result);

            socket.close();
         }
      } catch (IOException e) {
         e.printStackTrace();
      }
   }

   private static boolean isPalindrome(String str) {
```

```
        int left = 0;
        int right = str.length() - 1;

        while (left < right) {
            if (str.charAt(left++) != str.charAt(right--)) {
                return false;
            }
        }

        return true;
    }
}
```

Client:

```
import java.io.*;
import java.net.*;

public class Client {
    public static void main(String[] args) {
        try {
            Socket socket = new Socket("localhost", 12345);
            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

            String input = "level"; // Change this to any string you want to check
            out.println(input);

            String result = in.readLine();
            System.out.println("Result from server: " + result);

            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

6. a) What are the differences between JDBC and ODBC? What are the **7**

steps to connect to the database in java?

Answer:

ODBC	JDBC
1. ODBC Stands for Open Database Connectivity.	1. JDBC Stands for Java database connectivity.
2. Introduced by Microsoft in 1992.	2. Introduced by SUN Micro Systems in 1997.
3. We can use ODBC for any language like C, C++, Java etc.	3. We can use JDBC only for Java languages.
4. We can choose ODBC only Windows platform.	4. We can use JDBC on any platform.
5. Mostly ODBC Driver is developed in native languages like C, and C++.	5. JDBC Stands for Java database connectivity.
6. For Java applications it is not recommended to use ODBC because performance will be down due to internal conversion and applications will become platform-dependent.	6. For Java applications it is highly recommended to use JDBC because there are no performance & platform dependent problems.
7. ODBC is procedural.	7. JDBC is object-oriented.

Load the JDBC Driver: Load the appropriate JDBC driver for your database. This is typically done using the Class.forName() method with the fully qualified name of the JDBC driver class.

Establish a Connection: Use the DriverManager.getConnection() method to establish a connection to the database. Pass the connection URL, username, and password as parameters to this method.

Create a Statement: Create a Statement or PreparedStatement object to execute SQL queries against the database. This is typically done using the *Connection.createStatement()* or *Connection.prepareStatement()* method.

Execute SQL Queries: Use the executeQuery() method of the Statement or PreparedStatement object to execute SQL SELECT queries. Use executeUpdate() for SQL DML (Data Manipulation Language) queries like INSERT, UPDATE, DELETE.
Process the Results: If executing a SELECT query, iterate through the ResultSet returned by the executeQuery() method to retrieve and process the results.
Close Resources: Close the ResultSet, Statement, and Connection objects to release database and JDBC resources. Use the close() method of each object to do this.
Handle Exceptions: Use try-catch blocks to handle any exceptions that may occur during database connection and query execution. This ensures graceful error handling and prevents resource leaks.

b) A database "testdb" contains a table "Student" with fields id, name and address. Write a program that asks user to enter the information and save them in the database. The program should prompt the user to press y/n to either continue or exit the program after every successful entry. **8**

Answer:

```
import java.sql.*;
import java.util.Scanner;

public class reinsert {

   static Scanner sc = new Scanner(System.in);
  public static void main(String[] args) throws ClassNotFoundException,
SQLException {
        Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
     reinsert obj =new reinsert();
      Connection cn = obj.getConnection();
      obj.add(cn);

}
  public Connection getConnection () throws SQLException {
      String url =
"jdbc:sqlserver://localhost\\sqlexpress:1433;databaseName=testdb;encrypt=true;t
rustServerCertificate=true";
      String username = "sa";
      String password = "";
      Connection cn = DriverManager.getConnection(url, username, password);
      return cn;
```

```
    }
    public void add (Connection cn) throws SQLException{
        System.out.println("Enter student name");
        String name = sc.next();
        System.out.println("Enter student address");
        String address = sc.next();
        String sql = "insert into Student(name,address) values(?,?)";
        try (PreparedStatement ps = cn.prepareStatement(sql)) {
            ps.setString(1, name);
            ps.setString(2, address);
            int result = ps.executeUpdate();
            if (result == 1) {
                System.out.println("Student added successfully");
                System.out.println("Do you want to enter another Student detail (y/n)?");
                String choice = sc.next();
                if(!choice.equalsIgnoreCase("y")){
                    ps.close();
                    cn.close();
                    sc.close();
                }else{
                    add(cn);
                }
            } else {
                System.out.println("Student not added");
            }
        }
            cn.close();
            sc.close();
    }
}
```

7. **Write short notes on: (Any two)**
    a. **Abstract class**

    The abstract keyword is a non-access modifier, used for classes and methods:

    Abstract class: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

Abstract method: can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from)

```
abstract class Animal {
  public abstract void animalSound();
  public void sleep() {
    System.out.println("Zzz");
  }
}
```

✗ `Animal myObj = new Animal(); // will generate an error`

```
// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}
```

b. An Applet Skeleton

An applet skeleton is a basic template for creating applets in Java. An applet is a small Java program that runs in a web browser or an applet viewer. An applet skeleton typically consists of the following methods:

- *init():* This method is called once when the applet is loaded. It is used to initialize variables and resources.
- *start():* This method is called after init() and whenever the applet is restarted. It is used to start or resume the applet's execution.
- *stop():* This method is called when the applet is stopped or moved out of view. It is used to pause or suspend the applet's execution.
- *destroy():* This method is called when the applet is terminated or removed from the browser. It is used to release any resources or perform any cleanup activities.
- *paint(Graphics g):* This method is called when the applet's output needs to be displayed or refreshed. It is used to draw graphics or text on the applet's window.

c. Inner Class

An inner class is a class that is declared inside another class or interface. It has some advantages, such as:

- It can access the private members of the outer class or interface.
- It can group logically related classes together, making the code more readable and maintainable.
- It can provide different implementations of an interface or an abstract class.

There are four types of inner classes in Java:

- Nested inner class: It is a non-static class that is declared inside another class. It can access any member of the outer class, but it cannot have any static members itself. It can be instantiated only through an object of the outer class.
- Method local inner class: It is a class that is declared inside a method of the outer class. It can access the final local variables of the method, but it cannot be accessed from outside the method. It can be instantiated only within the method where it is defined.
- Static nested class: It is a static class that is declared inside another class or interface. It can access only the static members of the outer class or interface, but it can have its own static and non-static members. It can be instantiated without an object of the outer class or interface.
- Anonymous inner class: It is a class that has no name and is declared and instantiated at the same time. It is usually used to provide a one-time implementation of an interface or an abstract class. It can access the final local variables of the enclosing scope, but it cannot have any constructors or static members.

2020 'Fall'

1 a) Explain the Intermediate Working principal of java programming language that makes it runnable on any platform. **8**

Answer:

The intermediate working principle of Java that enables it to run on any platform is based on the concept of the Java Virtual Machine (JVM) and bytecode compilation.

Compilation to Bytecode: When you compile a Java source code file (with a .java extension), the Java compiler (javac) translates the source code into an

intermediate representation called bytecode. Bytecode is a platform-independent binary format that is not specific to any particular hardware or operating system.
Platform-Independent Bytecode: Bytecode is designed to be executed by the Java Virtual Machine (JVM), which is a software-based runtime environment. The bytecode instructions are designed to be architecture-neutral and can be interpreted or compiled into native machine code by the JVM at runtime.
Java Virtual Machine (JVM): The JVM is a virtual machine that provides an abstract execution environment for Java bytecode. It acts as an interpreter or just-in-time (JIT) compiler, executing bytecode instructions on the host system. The JVM is responsible for loading and executing Java classes, managing memory, handling exceptions, and providing access to system resources.
Write Once, Run Anywhere (WORA): Because Java programs are compiled to bytecode and executed by the JVM, they can run on any device or platform that has a compatible JVM implementation. This principle is known as "Write Once, Run Anywhere" (WORA), which means that you can write Java code once and run it on any platform that supports Java.
JVM Implementations: JVM implementations are available for various operating systems and hardware platforms, including Windows, macOS, Linux, and others. Each JVM implementation is responsible for translating bytecode into native machine code suitable for the underlying platform.
Standardization of Java Specifications: The Java language and platform are standardized by the Java Community Process (JCP), ensuring compatibility and consistency across different implementations and versions of Java. This standardization helps maintain the platform independence and portability of Java programs.

b) Why multiple inheritance is not supported in Java. Provide a simple code example to support yours answer.

Multiple inheritance is not supported in Java primarily to avoid the "diamond problem," where conflicts arise when a class inherits from two or more classes that have a common ancestor. Java opts for single inheritance with interfaces to provide a solution for achieving multiple inheritance of types.
Here's a simple explanation with a code example:

```
class A {
   public void methodA() {
      System.out.println("Method A");
   }
}

class B extends A {
```

```
    public void methodB() {
        System.out.println("Method B");
    }
}

class C extends A {
    public void methodC() {
        System.out.println("Method C");
    }
}

class D extends B, C { // This is not allowed in Java
    // Class D attempts to inherit from both B and C
}
```

2a) Why Reflection is regarded as a powerful aspect in java? WAP to demonstrate your answer. **8**

b) Define exception and explain why it is necessary to handle exceptions? WAP to write bytes of data to a file called "exams.txt" and reading it until end of file is obtained, using proper exception handling. **7**

Answer:

An exception in Java is an event that disrupts the normal flow of a program's execution. It occurs when an unexpected condition or error arises during the runtime of a program. Exceptions can be caused by various factors such as invalid input, network failures, file not found, arithmetic errors, and more.

It is necessary to handle exceptions in Java for several reasons:

- **Error Detection:** Exceptions help detect and identify errors and abnormal conditions that may occur during program execution. They provide information about the cause of the error, which helps in diagnosing and fixing the problem.
- **Error Reporting:** Handling exceptions allows the program to gracefully handle errors and report them to the user or log them for further analysis. This improves the usability and reliability of the software by providing meaningful error messages to the user.

- P**rogram Robustness:** Exception handling improves the robustness and resilience of the program by preventing it from crashing or terminating abruptly when errors occur. It allows the program to recover from errors and continue executing or gracefully terminate without causing damage.
- **Resource Management:** Exceptions are commonly used to handle resources such as files, network connections, and database connections. Proper exception handling ensures that resources are released and cleaned up properly, even in the event of an error or exception.
- **Control Flow:** Exception handling allows the program to control the flow of execution in response to different error conditions. It provides mechanisms for catching and handling exceptions, retrying operations, and taking appropriate actions based on the type of exception.

Example:

```
import java.io.*;
public class FileReadWriteExample {
   public static void main(String[] args) {
      // Write bytes of data to a file
      try (FileOutputStream fos = new FileOutputStream("exams.txt")) {
         String data = "Java Exception Handling Example";
         fos.write(data.getBytes());
         System.out.println("Data successfully written to file.");
      } catch (IOException e) {
         System.err.println("Error writing data to file: " + e.getMessage());
      }

      // Read bytes of data from the file
      try (FileInputStream fis = new FileInputStream("exams.txt")) {
         int byteRead;
         while ((byteRead = fis.read()) != -1) {
            System.out.print((char) byteRead);
         }
         System.out.println("\nEnd of file reached.");
      } catch (IOException e) {
         System.err.println("Error reading data from file: " + e.getMessage());
      }
   }
}
```

3a) Define Applet and explain its lifecycle. Create a simple Applet and demonstrate how we can run a Java Applet. **8**

Answer: Refer to 2019 Spring ‘3a’

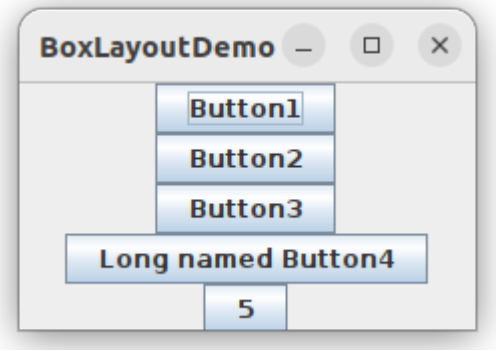
b) Create a Frame with two Text Fields, one of which should show whether the mouse pointer is inside or outside the Frame and the other text field should display the X and Y coordinates of the pointer when the use moves the mouse pointer inside the Frame area. **7**

Answer:

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

public class MousePositionFrame extends JFrame {
   private JTextField mouseStatusField;
   private JTextField coordinatesField;

   public MousePositionFrame() {
      super("Mouse Position Tracker");

      mouseStatusField = new JTextField(20);
      mouseStatusField.setEditable(false);

      coordinatesField = new JTextField(20);
      coordinatesField.setEditable(false);

      JPanel panel = new JPanel(new GridLayout(2, 1));
      panel.add(mouseStatusField);
      panel.add(coordinatesField);

      add(panel);

      addMouseListener(new MouseAdapter() {
         @Override
         public void mouseEntered(MouseEvent e) {
            mouseStatusField.setText("Mouse inside the frame");
         }

         @Override
         public void mouseExited(MouseEvent e) {
```

```
            mouseStatusField.setText("Mouse outside the frame");
        }
    });

    addMouseMotionListener(new MouseAdapter() {
        @Override
        public void mouseMoved(MouseEvent e) {
            int x = e.getX();
            int y = e.getY();
            coordinatesField.setText("X: " + x + ", Y: " + y);
        }
    });

    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(300, 200);
    setLocationRelativeTo(null);
    setVisible(true);
  }

  public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> new MousePositionFrame());
  }
}
```

4a) State the different types of layout manager in swing. WAP to generate the 7 following output in java using Box Layout.



FlowLayout

FlowLayout arranges components in a left-to-right, top-to-bottom flow. Components are placed next to each other until the container is full, and then a new row is started.

It's straightforward and suitable for small forms or dialogs.
JPanel panel = new JPanel(new FlowLayout());

BorderLayout

BorderLayout divides the container into five regions: North, South, East, West, and Center.
Components are placed in these regions, and the Center region takes up any remaining space.
It's useful for creating simple, main-content layouts.
JPanel panel = new JPanel(new BorderLayout());

GridLayout

GridLayout organizes components in a grid, with a specified number of rows and columns.
All components are given equal space in the grid.
It's useful for creating forms with a consistent grid structure.
JPanel panel = new JPanel(new GridLayout(rows, columns));

BoxLayout

BoxLayout arranges components in a single line, either horizontally or vertically.
It's useful for creating rows or columns of components with consistent alignment.
JPanel panel = new JPanel();
panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));

GridBagLayout

GridBagLayout is a flexible and powerful layout manager that allows precise control over the
placement and sizing of components.
It uses constraints to define how components should be arranged.
It's suitable for complex and customized layouts.
JPanel panel = new JPanel(new GridBagLayout());
Example:

```
import javax.swing.*;
import java.awt.*;
public class Sample {
 public static void main(String[] args){
    JFrame frame = new JFrame("BoxLayoutDemo");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setBounds(0,0,400, 170);
        Container container= frame.getContentPane();
    BoxLayout boxlayout=new BoxLayout(container,BoxLayout.Y_AXIS);
    container.setLayout(boxlayout);
    JButton button1 = new JButton("Button1");
```

```
    container.add(button1);
    JButton button2 = new JButton("Button2");
    container.add(button2);
    JButton button3 = new JButton("Button3");
    container.add(button3);
    JButton button4 = new JButton("Long named Button4");
    container.add(button4);
    // For Adding space between two button
   //  container.add(Box.createRigidArea(new Dimension(0,20)));
    JButton button5 = new JButton("Button5");
    container.add(button5);
    //Adding Button Alignment
    button1.setAlignmentX(JComponent.CENTER_ALIGNMENT);
    button2.setAlignmentX(JComponent.CENTER_ALIGNMENT);
    button3.setAlignmentX(JComponent.CENTER_ALIGNMENT);
    button4.setAlignmentX(JComponent.CENTER_ALIGNMENT);
    button5.setAlignmentX(JComponent.CENTER_ALIGNMENT);

    frame.setVisible(true);
  }

}
```

b) As per a recent survey, among all the Javascript frameworks 40% prefers React while Angular comes second with 30% developers preferring it and 20% of developers prefer Vue while remaining 10% use other frameworks. Create a pie to show case the survey information. **8**

Answer: **Out of Syllabus**

5 a) What are the different socket types in java? Draw a diagram with appropriate methods to describe client server interaction in java.

Answer: Do your Self

b) Explain the use of URL and URL connection class with a suitable example. **7**

Answer:

The URL (Uniform Resource Locator) class in Java represents a Uniform Resource Identifier (URI) and provides methods for accessing the various components of a URL, such as the protocol, host, port, path, query, and fragment. It allows you to

construct URL objects from string representations and retrieve information about the resource the URL points to.

The URLConnection class in Java represents a communication link between the application and a URL. It provides methods for establishing a connection to a URL, retrieving data from the URL, setting request headers, and handling various aspects of the communication process.

Example:

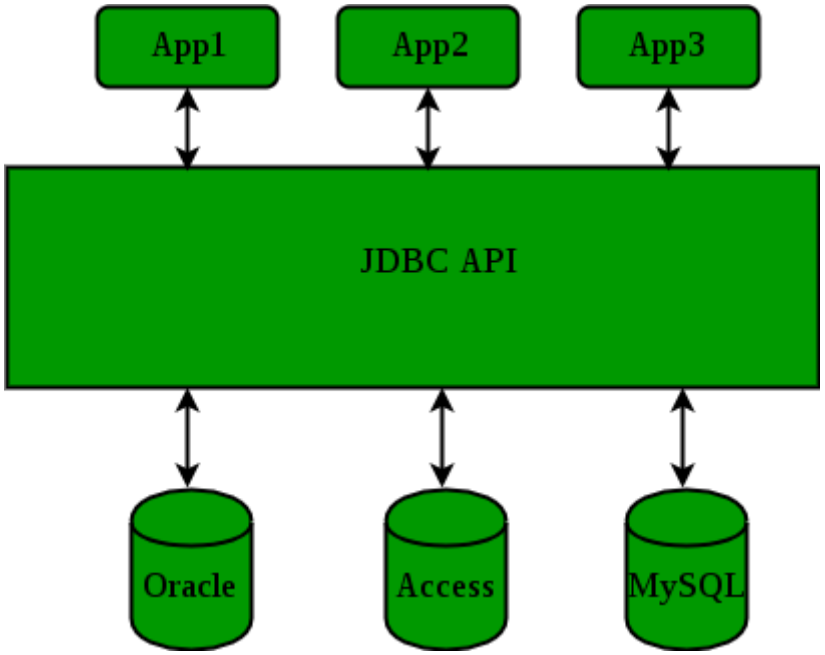
```
import java.io.*;
import java.net.*;

public class URLExample {
   public static void main(String[] args) {
      try {
         // Create a URL object representing the web page to be accessed
         URL url = new URL("https://example.com");

         // Open a connection to the URL
         URLConnection connection = url.openConnection();

         // Get an input stream for reading data from the URL
         InputStream inputStream = connection.getInputStream();
         BufferedReader reader = new BufferedReader(new
InputStreamReader(inputStream));

         // Read and print each line of the web page content
         String line;
         while ((line = reader.readLine()) != null) {
            System.out.println(line);
         }

         // Close the input stream
         reader.close();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

6a) What is JDBC? Draw JDBC architecture and explain JDBC drivers with their **7**

advantages and disadvantages.

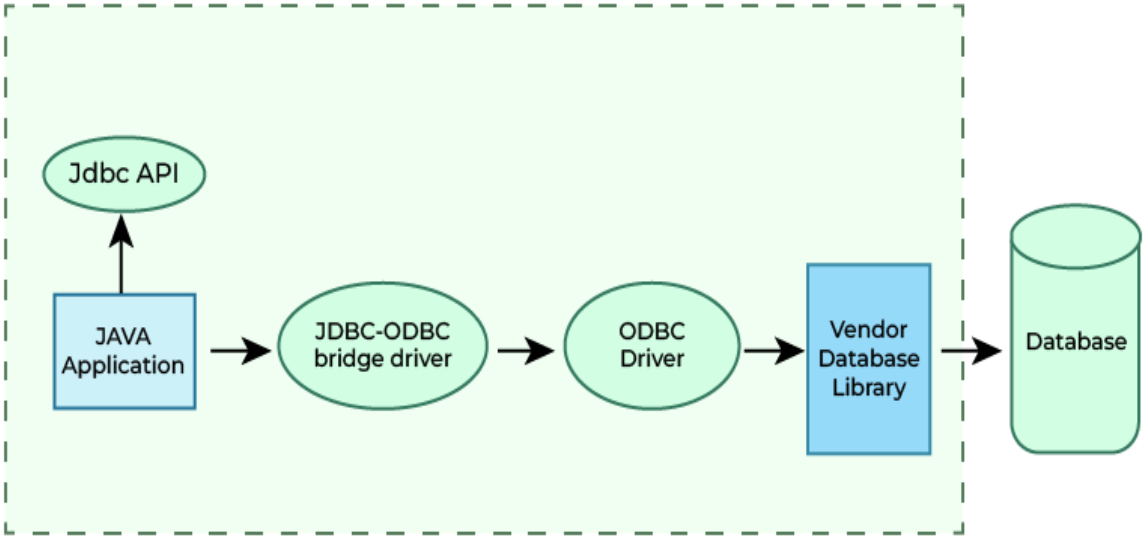
Java Database Connectivity (JDBC) is an application programming interface (API) for the programming language Java, which defines how a client may access any kind of tabular data, especially a relational database. JDBC Drivers uses JDBC APIs which was developed by Sun Microsystem, but now this is a part of Oracle. There are 4 types of JDBC drivers. It is part of the Java Standard Edition platform, from Oracle Corporation. It acts as a middle-layer interface between Java applications and databases.

The JDBC classes are contained in the Java Package java.sql and javax.sql.



1. JDBC-ODBC bridge driver – Type 1 driver

Type-1 driver or JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. Type-1 driver is also called Universal driver because it can be used to connect to any of the databases.



Advantages

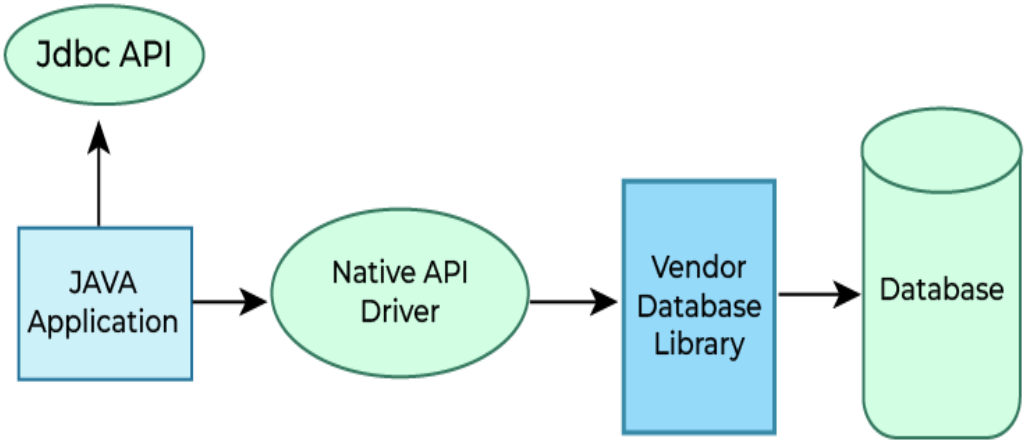
- This driver software is built-in with JDK so no need to install separately.
- It is a database independent driver.

Disadvantages

- As a common driver is used in order to interact with different databases, the data transferred through this driver is not so secured.
- The ODBC bridge driver is needed to be installed in individual client machines.
- Type-1 driver isn't written in java, that's why it isn't a portable driver.

2. Native-API driver – Type 2 driver (Partially Java driver)

The Native API driver uses the client -side libraries of the database. This driver converts JDBC method calls into native calls of the database API. In order to interact with different database, this driver needs their local API, that's why data transfer is much more secure as compared to type-1 driver. This driver is not fully written in Java that is why it is also called Partially Java driver.



Advantage

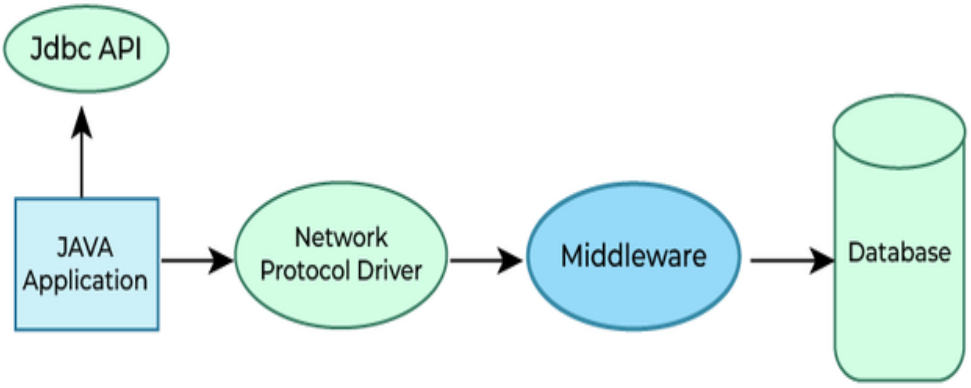
- Native-API driver gives better performance than JDBC-ODBC bridge driver.

Disadvantages

- Driver needs to be installed separately in individual client machines
- The Vendor client library needs to be installed on client machine.
- Type-2 driver isn't written in java, that's why it isn't a portable driver
- It is a database dependent driver.

3. Network Protocol driver – Type 3 driver (fully Java driver)

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. Here all the database connectivity drivers are present in a single server, hence no need of individual client-side installation.



Advantages

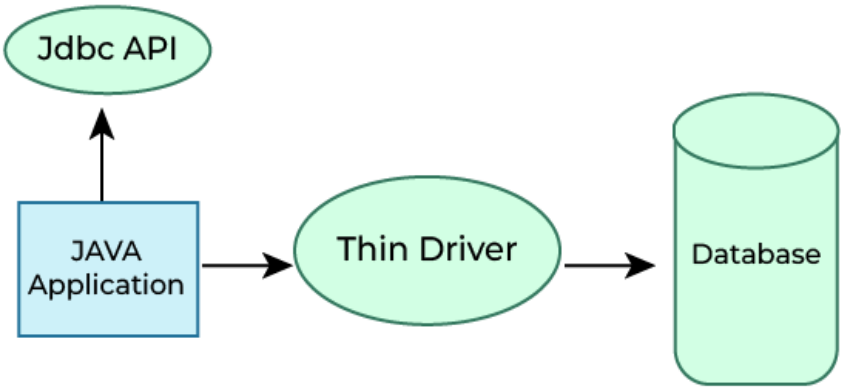
- Type-3 drivers are fully written in Java, hence they are portable drivers.
- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.
- Switch facility to switch over from one database to another database.

Disadvantages

- Network support is required on client machine.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

4. Thin driver – Type 4 driver (fully Java driver)

Type-4 driver is also called native protocol driver. This driver interact directly with database. It does not require any native database library, that is why it is also known as Thin Driver.



Advantages

- Does not require any native library and Middleware server, so no client-side or server-side installation.
- It is fully written in Java language, hence they are portable drivers.

Disadvantage:

- If the database varies, then the driver will carry because it is database dependent.

b) WAP to insert data of any student with schemas (name. age. Semetar. University. Status.) By default. Sqlconnector is to be used (if comfortable with any database connector. free to describe and use accordingly) **8**

Answer:

```
import java.sql.*;
import java.util.Scanner;

public class reinsert {

    static Scanner sc = new Scanner(System.in);
   public static void main(String[] args) throws ClassNotFoundException,
SQLException {
          Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
       reinsert obj =new reinsert();
        Connection cn = obj.getConnection();
        obj.add(cn);
```

```
}
  public Connection getConnection () throws SQLException {
      String url =
"jdbc:sqlserver://localhost\\sqlexpress:1433;databaseName=Java;encrypt=true;tru
stServerCertificate=true";
      String username = "sa";
      String password = "S2cond@2nd";
      Connection cn = DriverManager.getConnection(url, username, password);
      return cn;
  }
  public void add (Connection cn) throws SQLException{
      System.out.println("Enter student name");
      String name = sc.next();
      System.out.println("Enter student age");
      String age = sc.next();
System.out.println("Enter student semester");
      String semester = sc.next();
System.out.println("Enter student Universityname");
      String university = sc.next();
System.out.println("Enter student Status");
      String status = sc.next();

      String sql = "insert into Student(name,age,semester,University,Status)
values(?,?,?,?,?)";
      try (PreparedStatement ps = cn.prepareStatement(sql)) {
         ps.setString(1, name);
         ps.setString(2, age);
         ps.setString(3, semester);
         ps.setString(4,university);
         ps.setString(5, status);

         int result = ps.executeUpdate();
         if (result == 1) {
            System.out.println("Student added successfully");
                  } else {
            System.out.println("Student not added");
         }
      }
        ps.close();
         cn.close();
```

```
        sc.close();
    }
}
```

7 Write short notes on: (Any two)

a. Wrapper Class.

A wrapper class is a class that wraps or contains a primitive data type, such as int, char, or boolean, and provides methods to manipulate or access the value. Wrapper classes are useful for converting primitive types to objects, which can be used in collections, multithreading, serialization, etc. For example, the Integer class is a wrapper class for the int type, and it has methods like intValue(), toString(), and compareTo().

Some examples of wrapper classes and their corresponding primitive types are:

Primitive Type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

b. Life cycle of an applet.

An applet is a Java program that can be embedded into a web page and runs inside the web browser. It works on the client-side and generates dynamic content. An applet goes through four stages of its life cycle: initialization, start, stop, and destroy. These stages correspond to the applet methods init(), start(), stop(), and destroy() respectively. These methods are invoked by the browser or the applet container to manage the applet's execution. There is also a paint() method that belongs to the awt.Component class and is used to draw graphics on the applet.

The applet life cycle can be summarized as follows:

Initialization: The init() method is the first method to run that initializes the applet. It is invoked only once when the applet is loaded into the browser. It is used to set up the applet's user interface, variables, and resources.

Start: The start() method contains the actual code of the applet and starts the applet. It is invoked immediately after the init() method and every time the applet is refreshed or resumed. It is used to perform any actions that the applet needs to do when it becomes active, such as starting threads, playing sounds, or updating data.

Stop: The stop() method stops the execution of the applet. It is invoked whenever the applet is paused or hidden, such as when the browser is minimized or switched to another tab. It is used to perform any actions that the applet needs to do when it becomes inactive, such as stopping threads, pausing sounds, or saving data.

Destroy: The destroy() method destroys the applet after its work is done. It is invoked when the applet is removed from the browser or the browser is closed. It is used to perform any actions that the applet needs to do before it is terminated, such as releasing resources, closing connections, or cleaning up memory.

Paint: The paint() method is used to draw graphics on the applet. It is invoked after the start() method and whenever the applet needs to be redrawn, such as when the browser or the applet window is resized. It is used to display the applet's content, such as shapes, images, or text.

c. TCP and UDP socket.

TCP socket

A TCP socket is a connection-oriented socket that establishes a reliable and ordered data transfer between two devices. A TCP socket uses a three-way handshake to create a connection, and then sends and receives data in segments that are acknowledged by the receiver. A TCP socket also

implements flow control and congestion control mechanisms to ensure the optimal rate and quality of the data transmission. A TCP socket is suitable for applications that require high reliability and consistency, such as web browsing, file transfer, or email.

UDP socket

A UDP socket is a connectionless socket that sends and receives data in datagrams that are independent and unordered. A UDP socket does not establish a connection or guarantee the delivery, order, or integrity of the data. A UDP socket also does not implement any flow control or congestion control mechanisms, so it can send and receive data at any rate and quality. A UDP socket is suitable for applications that require low latency and high speed, such as video streaming, voice over IP, or online gaming.

2019 'Fall'

1. **a) What are the most powerful features of java? Describe how it is a platform neutral language by a diagram.** **7**

Java is a popular and versatile programming language that has many powerful features. Some of the most important features of Java are:

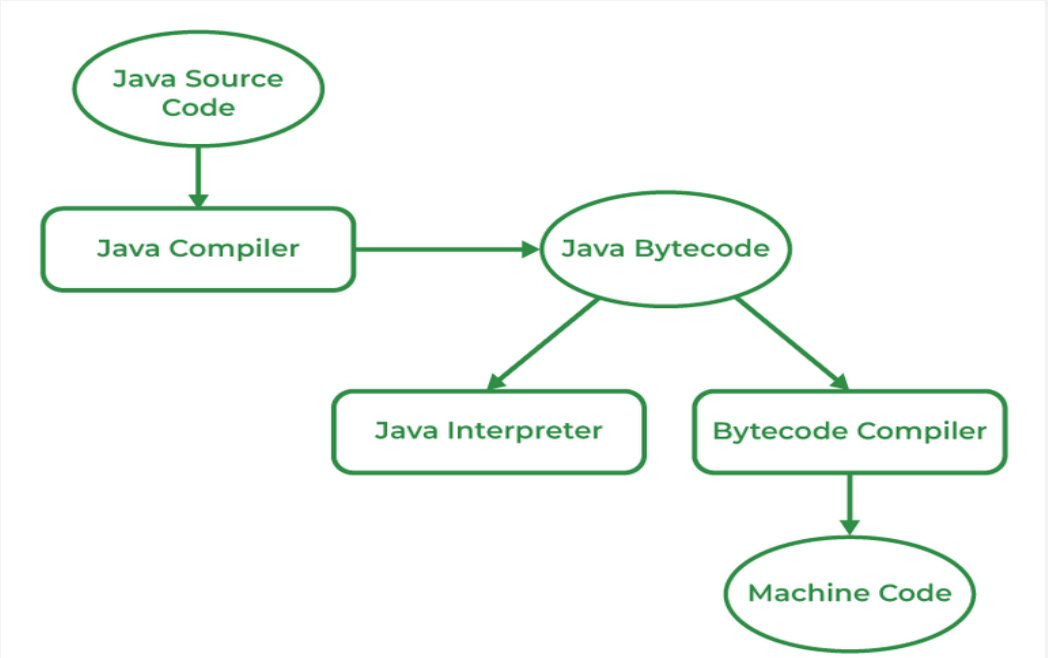
- Object-oriented: Java is based on the concept of objects, which are entities that have attributes and behaviors. Objects can be organized into classes, which define the common properties and methods of a group of objects. Java supports the principles of object-oriented programming, such as inheritance, polymorphism, abstraction, and encapsulation.
- Simple: Java is designed to be easy to learn and use, with a clear and concise syntax. Java eliminates some of the complex and rarely-used features of other languages, such as pointers, multiple inheritance, and operator overloading. Java also provides automatic memory management and garbage collection, which free the programmer from manual memory allocation and deallocation.
- Robust: Java is a reliable and error-free language, with built-in mechanisms to handle exceptions and errors. Java performs strict compile-time and run-time checks to ensure the validity and consistency of the code. Java also supports multithreading, which allows concurrent execution of multiple tasks within a single program.
- Secure: Java is a secure language, with features that prevent unauthorized access and manipulation of data and resources. Java runs the code inside a virtual machine, which isolates it from the underlying operating system and hardware. Java also does not allow direct memory access or pointer arithmetic, which can

cause security breaches. Java provides cryptographic and authentication tools to ensure the integrity and confidentiality of the data.

- Interpreted and compiled: Java is both an interpreted and a compiled language, which means that it combines the advantages of both approaches. Java code is first compiled into bytecode, which is a platform-independent intermediate representation of the code. The bytecode is then executed by a Java virtual machine, which interprets it and translates it into machine instructions for the specific platform. This allows Java code to run on any platform that has a Java virtual machine installed, without requiring recompilation.

Java is a platform-neutral language, which means that it can run on any platform that supports a Java virtual machine, regardless of the hardware and operating system. This is possible because of the following steps:

- The Java compiler converts the source code into bytecode, which is a standard and uniform format that is independent of any specific platform.
- The Java virtual machine interprets the bytecode and converts it into machine instructions that are specific to the platform on which it is running. The Java virtual machine acts as an abstraction layer between the bytecode and the platform, hiding the differences and complexities of the underlying hardware and operating system.
- The Java application programming interface (API) provides a set of libraries and classes that are available to the Java code, regardless of the platform. The Java API defines the functionality and behavior of the Java code, such as input/output, networking, graphics, database, etc. The Java API is implemented by the Java virtual machine, which ensures that the Java code can access the same features and services on any platform**.**



b) Explain multiple inheritance. How is it possible to achieve multiple inheritance in java? **8**

Answer:

Multiple inheritance is a feature of some programming languages that allows a class to inherit properties and behavior from more than one parent class. In other words, a subclass can have multiple superclasses. This can lead to complexities such as the diamond problem, where ambiguity arises when a subclass inherits from two superclasses that have a common ancestor.

Java does not support multiple inheritance of classes, meaning a class can only extend one superclass. This decision was made to avoid the complexities and ambiguities associated with multiple inheritance, particularly the diamond problem. However, Java does support multiple inheritance of interfaces.

In Java, a class can implement multiple interfaces, which allows it to inherit abstract methods and constants from multiple sources. This is achieved using the implements keyword followed by a comma-separated list of interface names.

Here's an example:

```
interface Interface1 {
   void method1();
}

interface Interface2 {
   void method2();
}
```

```
class MyClass implements Interface1, Interface2 {
    public void method1() {
        System.out.println("Method 1 implementation");
    }

    public void method2() {
        System.out.println("Method 2 implementation");
    }
}
```

2. a) What is interface? WAP to show interface implementation in java. 8

Answer:

An interface in Java is a reference type similar to a class that can contain only abstract methods, default methods, static methods, and constant (final) variables. It defines a contract for classes to implement, specifying a set of methods that the implementing class must provide. Interfaces allow for multiple inheritance in Java, as a class can implement multiple interfaces.

```
// Define an interface
interface Animal {
    void sound();
    void eat();
}

// Implement the interface in a class
class Dog implements Animal {
    public void sound() {
        System.out.println("Dog barks");
    }

    public void eat() {
        System.out.println("Dog eats bones");
    }
}

// Implement another interface in a class
interface Bird {
    void chirp();
}
```

```
class Parrot implements Bird {
    public void chirp() {
        System.out.println("Parrot chirps");
    }
}

public class InterfaceExample {
    public static void main(String[] args) {
        // Create objects of classes implementing the interface
        Dog dog = new Dog();
        Parrot parrot = new Parrot();

        // Call methods defined in the interface
        dog.sound();
        dog.eat();

        parrot.chirp();
    }
}
```

b) Create a program to write "Test on File Handling" inside a file "abc.txt". **7**

3. a) What is an Applet? Explain life cycle of applet. **8**

Solved in above set

b) What are the differences between GridLayout and GridBagLayout? **7**
Explain FlowLayout.

Answer:

Feature	GridLayout	GridBagLayout
Layout Behavior	Divides the container into a grid of equally sized cells, with components placed in each cell.	Allows for more flexible layout by specifying constraints for each component individually.
Component Sizing	All cells in the grid have the same size.	Components can have different sizes and span multiple cells.
Alignment	Components are aligned along the grid's edges.	Supports component alignment and resizing within cells.

Ease of Use	Simple to use, especially for layouts with uniform grid-based structure.	More complex and flexible, suitable for intricate layouts.
Flexibility	Limited flexibility in component placement and sizing.	Highly flexible, allowing precise control over component positioning and sizing.
Constraint Support	Does not support specifying constraints for individual components.	Requires specifying constraints for each component using GridBagConstraints.

FlowLayout

FlowLayout arranges components in a left-to-right, top-to-bottom flow.
Components are placed next to each other until the container is full, and then a new row is started.
It's straightforward and suitable for small forms or dialogs.
JPanel panel = new JPanel(new FlowLayout());

4. a) Create a swing GUI that contains a button, and two text fields. When the button is clicked the value of first text field should be checked and display "odd number" or "even number" in the second text field. 8

Answer:

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

public class OddEvenGUI extends JFrame {
    private JTextField textField1;
    private JTextField textField2;
    private JButton checkButton;

    public OddEvenGUI() {
        super("Odd/Even Checker");

        textField1 = new JTextField(10);
        textField2 = new JTextField(10);
        textField2.setEditable(false);
        checkButton = new JButton("Check");

        // Add ActionListener to the button
        checkButton.addActionListener(new ActionListener() {
```

```
        @Override
        public void actionPerformed(ActionEvent e) {
            // Get the text from the first text field
            String inputText = textField1.getText();
            try {
                // Parse the text as an integer
                int number = Integer.parseInt(inputText);
                // Check if the number is odd or even
                if (number % 2 == 0) {
                    textField2.setText("Even number");
                } else {
                    textField2.setText("Odd number");
                }
            } catch (NumberFormatException ex) {
                // If the input is not a valid integer
                textField2.setText("Invalid input");
            }
        }
    });

    // Create a panel and add components
    JPanel panel = new JPanel();
    panel.add(new JLabel("Enter a number: "));
    panel.add(textField1);
    panel.add(checkButton);
    panel.add(new JLabel("Result: "));
    panel.add(textField2);

    // Set layout and add panel to the frame
    setLayout(new FlowLayout());
    add(panel);

    // Set frame properties
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(300, 150);
    setLocationRelativeTo(null); // Center the frame
    setVisible(true);
}

public static void main(String[] args) {
```

```
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new OddEvenGUI();
            }
        });
    }
}
```

b) Create a graphics application to display "Pokhara University" in blue color with font name: Times New Roman, type:Bold and size:20. Also write HTML to embed the applet. **7**

Answer: Out of Syllabus

5. **a) Draw a diagram illustrating how client server interaction occurs in UNIX based system, with detailed methods in each step.** **7**

Answer: Out of Syllabus

b) Create a TCP client/server program in which client sends an integer to the server and the server responds to client by sending square of the number sent by client. **8**

Answer:

Server:

```
import java.io.*;
import java.net.*;

public class Server {
    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(12345);
            System.out.println("Server is running and waiting for connections...");

            while (true) {
                Socket socket = serverSocket.accept();
                System.out.println("Client connected: " + socket);

                BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
                PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

                int num = Integer.parseInt(in.readLine());
```

```
            int square = num * num;
            out.println(square);

            socket.close();
          }
        } catch (IOException e) {
          e.printStackTrace();
        }
    }
}
```

Client:

```
import java.io.*;
import java.net.*;

public class Client {
    public static void main(String[] args) {
        try {
            Socket socket = new Socket("localhost", 12345);
            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

            int num = 5; // Change this to the number you want to send to the server
            out.println(num);

            int square = Integer.parseInt(in.readLine());
            System.out.println("Square received from server: " + square);

            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

6. **a) Briefly describe the JDBC-ODBC types of bridge and driver in java.** **7**

   Answer:

The JDBC-ODBC Bridge is a mechanism provided by Java to enable Java applications to interact with databases via ODBC (Open Database Connectivity) drivers. It serves as a bridge between JDBC (Java Database Connectivity) API and ODBC API, allowing Java applications to access databases using ODBC drivers. Let's delve into the types of bridge and driver involved in the JDBC-ODBC Bridge in detail:

Bridge Types:

- **Type 1 Bridge (JDBC-ODBC Bridge):** The Type 1 JDBC-ODBC Bridge is the original bridge implementation provided by Java. It translates JDBC method calls into corresponding ODBC function calls. This bridge comes bundled with the Java Development Kit (JDK) and is available for use without any additional installation or configuration. It relies on the presence of an installed and properly configured ODBC driver on the system to establish a connection to the target database. While it provides a way to access databases via ODBC drivers, it has several limitations and is not recommended for production use due to performance overhead and security concerns.

Driver Types:

- **JDBC Driver:** The JDBC driver is a software component that allows Java applications to connect to databases using the JDBC API. There are four main types of JDBC drivers, categorized based on their architecture and implementation:
    - **Type 1 Driver (JDBC-ODBC Bridge):** As mentioned earlier, the Type 1 JDBC-ODBC Bridge is included in the JDK and acts as a bridge between JDBC and ODBC APIs. It translates JDBC calls into ODBC calls, enabling Java applications to access databases via ODBC drivers.
    - **Type 2 Driver (Native-API Driver):** The Type 2 JDBC driver uses a native library specific to the database vendor's platform. It communicates directly with the database using a vendor-specific API, bypassing the need for ODBC. While it offers better performance compared to the JDBC-ODBC Bridge, it is platform-dependent and may require additional installation steps.
    - **Type 3 Driver (Network Protocol Driver):** The Type 3 JDBC driver uses a middleware server to connect to the database. It translates JDBC calls into a vendor-independent protocol, which is then converted to the database-specific protocol by the middleware server. This driver provides platform independence and better performance than the Type 1 driver but may introduce additional latency due to network communication.

- **Type 4 Driver (Thin Driver):** The Type 4 JDBC driver communicates directly with the database server using a vendor-specific protocol. It does not require any additional middleware or native libraries, making it highly portable and easy to deploy. This driver offers the best performance and is the recommended choice for most applications.

b) Write a program to display only those records whose salary is more than 5000 from a table that contains id, name, post and salary of some employees. **8**

```
import java.util.Scanner;
import java.sql.*;
public class crudOp {
    static Scanner sc = new Scanner(System.in);
  public static void main(String[] args) throws ClassNotFoundException,
SQLException {
        Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
      crudOp obj = new crudOp();
      Connection cn = obj.getConnection();
      int choice;
      obj.showMenu();

  }

  public Connection getConnection() throws SQLException {
      String url =
"jdbc:sqlserver://localhost\\sqlexpress:1433;databaseName=Java;encrypt=true;tru
stServerCertificate=true";
      String username = "sa";
      String password = "";
      Connection cn = DriverManager.getConnection(url, username, password);
      return cn;

  }


  public void showAllStudents(Connection cn) throws SQLException {
      String sql = "select * from employee where salary >5000";
      Statement st = cn.createStatement();
      ResultSet rs = st.executeQuery(sql);
```

```
        while (rs.next()) {
            System.out.println("ID: " + rs.getInt("id") +  "      Name: " +
rs.getString("name") + "       Post: "
                    + rs.getString("post")+ "      Salary: "
                    + rs.getString("salary"));
            System.out.println("*********************************");
        }
    }
}
```

7. Write short notes on: (Any two)

a. Reflection API

The Reflection API is a feature of Java that allows you to inspect and manipulate the properties and behavior of classes, methods, fields, and constructors at runtime. With the Reflection API, you can access information about the structure and functionality of any Java object, even if you don't know its name or type at compile time. You can also create new instances, invoke methods, get or set field values, and modify the access level of members using the Reflection API.

The Reflection API is useful for applications that need to perform dynamic operations based on the runtime characteristics of objects, such as frameworks, libraries, testing tools, or IDEs.

b. Events and event classes

Events and event classes are concepts that are related to event-driven programming, which is a paradigm where the flow of the program is determined by the occurrence of events. Events are actions or occurrences that happen in the system or the environment, such as user input, mouse movement, network activity, timer expiration, etc. Event classes are data structures that represent the information and characteristics of the events, such as the type, source, time, location, parameters, etc.

In event-driven programming, there are two main roles: event sources and event listeners. Event sources are objects that generate or trigger events, such as buttons, text fields, scroll bars, etc. Event listeners are objects that handle or respond to events, such as methods, functions, or callbacks. Event sources and event listeners communicate through a mechanism called event handling, which involves registering, dispatching, and processing events.

c. Types of JDBC drivers

JDBC drivers are software components that enable Java applications to interact with databases. There are four types of JDBC drivers:

- **Type 1: JDBC-ODBC bridge driver.** This driver uses the ODBC driver to connect to the database and converts JDBC calls into ODBC calls. This driver is discouraged because of its low performance and compatibility issues. It is also removed from Java .
- **Type 2: Native-API driver**. This driver uses the native libraries of the database and converts JDBC calls into native calls. This driver has better performance than type 1, but it requires the installation of the native libraries on the client machine. It is also not fully written in Java.
- **Type 3: Network Protocol driver**. This driver uses a middleware server that communicates with the database and converts JDBC calls into the database-specific protocol. This driver is fully written in Java and does not require any native libraries on the client machine. However, it requires the maintenance of the middleware server and may have network overhead.
- **Type 4: Thin driver**. This driver directly communicates with the database and converts JDBC calls into the database-specific protocol. This driver is also fully written in Java and does not require any native libraries or middleware servers. It has the best performance and portability among the four types.

2019 'Spring'

1. **a) List the advantages of java programming language. Why is it also known as platform independent language?** **8**

Answer:

Java programming language offers several advantages, some of which include:

Simple and Easy to Learn: Java has a syntax that is similar to C and C++, making it easy for developers who are familiar with these languages to pick up Java. Additionally, Java omits complex features such as explicit memory management and operator overloading, making it simpler to learn and use.

Object-Oriented: Java is a fully object-oriented programming language, which means it supports concepts such as encapsulation, inheritance, and polymorphism. Object-oriented programming promotes code reusability, maintainability, and modularity.

Platform Independence: Java programs can run on any device or platform with a Java Virtual Machine (JVM). This platform independence is achieved by compiling Java source code into bytecode, which can be executed by any JVM, regardless of the underlying hardware and operating system.

Robust and Secure: Java has built-in features for error handling, exception handling, and memory management, which help in creating robust and reliable applications. Additionally, Java's security features, such as bytecode verification and sandboxing, protect against security vulnerabilities and malicious code execution.

Rich Standard Library: Java provides a vast standard library (Java API) that includes classes and methods for various tasks such as file I/O, networking, database access, GUI development, and more. The rich standard library reduces development time and effort by providing pre-built components and utilities.

Multi-threading Support: Java supports multi-threading, allowing developers to create concurrent and parallel applications. Multi-threading enables efficient utilization of system resources and improves application performance by executing multiple tasks simultaneously.

Dynamic and Extensible: Java supports dynamic class loading and reflection, which allow classes to be loaded at runtime and provide access to class metadata. This dynamic nature of Java enables features such as dynamic code execution, dynamic proxy creation, and runtime type checking.

Community Support: Java has a large and active developer community, with a wealth of resources such as documentation, tutorials, forums, and libraries. The community support helps developers to learn, troubleshoot, and collaborate on Java projects effectively.

Java is known as a platform-independent language because of its ability to run on any device or platform that has a Java Virtual Machine (JVM) installed. This platform independence is achieved through the following mechanisms:

Bytecode Compilation: Java source code is compiled into platform-independent bytecode rather than native machine code. Bytecode is a low-level representation of Java code that can be executed by any JVM, regardless of the underlying hardware and operating system.

Java Virtual Machine (JVM): JVM acts as an abstract execution environment for Java bytecode. It interprets and executes bytecode instructions on the host system, providing a consistent runtime environment across different platforms.

Write Once, Run Anywhere (WORA): The platform independence of Java allows developers to write Java code once and run it on any device or platform that supports Java. This WORA principle enables code portability and reduces the need for platform-specific code modifications.

b) In what condition is the use of "this" operator mandatory? Explain with suitable program. **7**

Answer:

The this keyword in Java is used to refer to the current instance of a class. It can be used in several situations, but it is mandatory in the following scenarios:

- **To Distinguish Between Local and Instance Variables:** When a local variable has the same name as an instance variable in a class, the this keyword is used to differentiate between them. This is necessary to avoid ambiguity and ensure that the correct variable is accessed.
- **To Invoke the Current Object's Method:** When a method is called within the same class and there is no ambiguity about which method to call, the this keyword can *be used to explicitly invoke the method of the current object.*

```
public class Employee {
    private String name; // Instance variable

    public Employee(String name) {
        // Use "this" to refer to the instance variable
        this.name = name;
    }

    public void printName(String name) {
        // Use "this" to refer to the instance variable
        System.out.println("Instance variable (this.name): " + this.name);
        // Use the parameter directly (no "this" needed)
        System.out.println("Local variable (name): " + name);
    }

    public static void main(String[] args) {
        // Create an instance of Employee
        Employee emp = new Employee("John Doe");

        // Call the printName method
        emp.printName("Jane Smith");
    }
}
```

2. a) ***Why Reflection is regarded as a powerful aspect in java? WAP to demonstrate your answer.*** **8**

b) What do you mean by exception? How exception handling is different from error handing? Explain. **7**

Answer:

An exception in programming refers to an unexpected event or condition that occurs during the execution of a program, disrupting the normal flow of execution. Exceptions

can occur for various reasons, such as invalid input, resource unavailability, programming errors, or unexpected conditions at runtime.

Exception handling is the process of gracefully handling these unexpected events or conditions in a program. It involves detecting, responding to, and possibly recovering from exceptions to ensure that the program continues to execute smoothly and does not terminate abruptly. Exception handling allows developers to write robust and resilient code that can gracefully handle errors and recover from unexpected situations.

Difference between Exception Handling and Error Handling:

Nature:

- Exception handling deals with unexpected conditions that occur during the execution of a program, such as runtime errors, invalid input, or resource unavailability.
- Error handling, on the other hand, typically refers to handling fatal or unrecoverable errors that occur due to serious system-level issues, hardware failures, or catastrophic failures that prevent the program from continuing execution.

Impact:

- Exceptions are generally recoverable and do not necessarily result in program termination. By handling exceptions appropriately, the program can recover from the error and continue execution.
- Errors, on the other hand, are often unrecoverable and indicate severe issues that may require intervention at the system level. Errors usually lead to program termination or abnormal termination of the application.

Handling Mechanism:

- Exception handling is implemented using try-catch blocks, where the code that may potentially throw an exception is enclosed within a try block, and the handling code is specified in one or more catch blocks.
- Error handling may involve system-level mechanisms or specialized error-handling techniques provided by the programming language or runtime environment. Error handling may also involve logging, reporting, and alerting mechanisms to notify administrators or users about critical failures.

3. a) State the life cycle of an applet. Can it be changed into application? If yes, justify your answer. **8**

An applet is a Java program that can be embedded into a web page. It runs inside the web browser and works at client side.

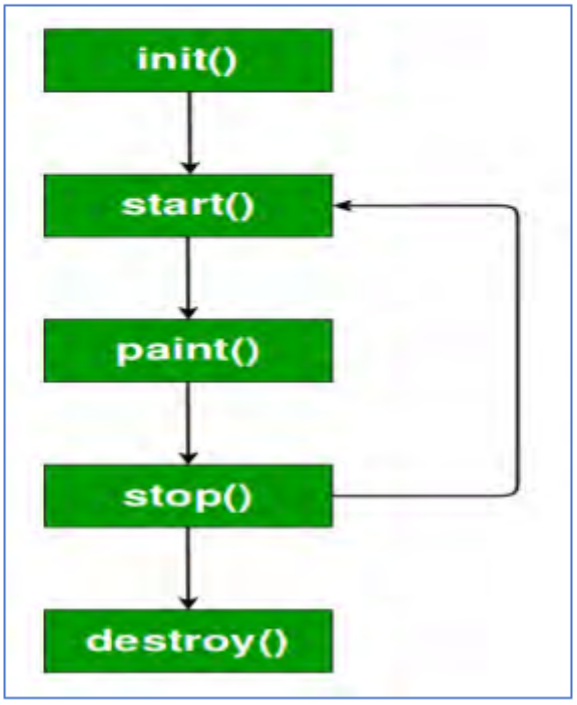
▪ An applet is embedded in an HTML page using the APPLET or OBJECT tag and hosted on a web server.

▪ Applets are used to make the website more dynamic and entertaining.

▪ To create an applet, a class must class extends java.applet.Applet class.
▪ An Applet class does not have any main () method. It is viewed using JVM. The JVM can use either a plug-in of the Web browser or a separate runtime environment to run an applet application. JVM creates an instance of the applet class and invokes init() method to initialize an Applet.

Life cycle of an Applet

- When an applet begins, the following methods are called, in this sequence:
  1. init ()
  2. start ()
  3. paint ()
- When an applet is terminated, the following sequence of method calls takes place:
  1. stop ()
  2. destroy ()



1. init(): This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed. This method is called only once during the run time of your applet.
2. start (): This method is automatically called after the browser calls the init method. It is also called to restart an applet after it has been stopped. Note that init () is called once i.e., when the first time an applet is loaded whereas start () is called each time an applet's HTML document is displayed onscreen. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.
3. paint (): Invoked immediately after the start () method, and also any time the applet needs to repaint itself in the browser. For example, the window in which the applet is running may be overwritten by another window and then uncovered. Or the applet

window may be minimized and then restored. Whatever the cause, whenever the applet must redraw its output, paint () is called. The paint () method has one parameter of type Graphics. This parameter will contain the graphics context, which describes the graphics environment in which the applet is running. This context is used whenever output to the applet is required. Its prototype is public void paint (Graphics g); where g is an object reference of class Graphic.

4. stop (): The stop () method is called when a web browser leaves the HTML document containing the applet—when it goes to another page, for example. When stop () is called, the applet is probably running. You should use stop () to suspend threads that don't need to run when the applet is not visible. You can restart them when start () is called if the user returns to the page.

5. destroy (): This method is only called when the browser shuts down normally. destroy () method is called when your applet needs to be removed completely from Memory.

Regarding whether an applet can be changed into an application, the answer is yes. An applet can be converted into a standalone application by modifying its code structure and removing its dependence on being embedded within a web page. Justifications for this include:

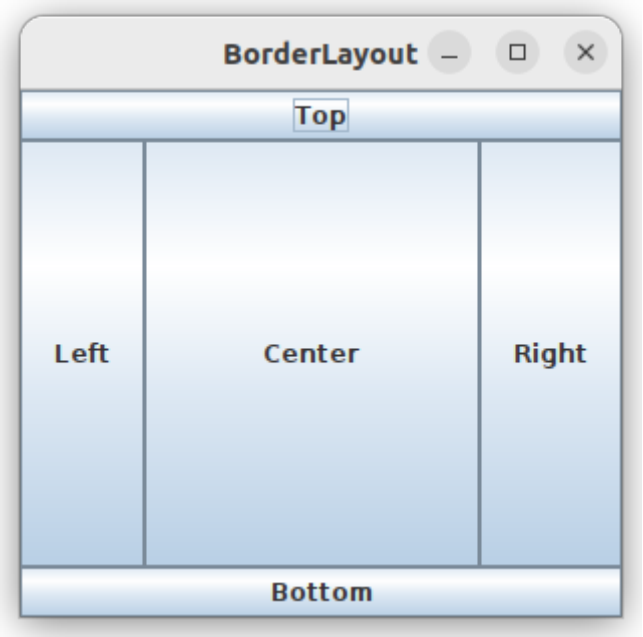
Independence from Browser: Applets run within a browser environment and are subject to browser limitations and dependencies. By converting an applet into a standalone application, you free it from these constraints, allowing it to run independently on a user's system.

Enhanced Functionality: Standalone applications have greater access to system resources and can offer more advanced functionality compared to applets. They can interact with the file system, access hardware devices, and perform other tasks that may not be possible within a browser environment.

Improved Performance: Standalone applications typically have better performance compared to applets since they do not need to contend with the overhead of running within a web browser. This can result in faster execution and better responsiveness.

Better User Experience: Standalone applications provide a more seamless user experience compared to applets, which require users to access them through a web browser. Users can launch standalone applications directly from their desktop or start menu, making them more convenient to use.

b) What is layout manager? State the different types of layout manager in swing. WAP to generate the following output in java using Boxlayout. **7**



Answer:First part in above set:

4. a) Create a swing GUI which contains a text field (to input radius of circle), a label and a button. When the button is clicked area of circle should be calculated and displayed in the label. **8**

Answer:

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

public class CircleAreaCalculator extends JFrame {
    private JTextField radiusField;
    private JLabel areaLabel;
    private JButton calculateButton;

    public CircleAreaCalculator() {
        super("Circle Area Calculator");

        radiusField = new JTextField(10);
        areaLabel = new JLabel("Area will be displayed here");
        calculateButton = new JButton("Calculate");

        calculateButton.addActionListener(new ActionListener() {
            @Override
```

```
            public void actionPerformed(ActionEvent e) {
                try {
                    // Get the radius from the text field
                    double radius = Double.parseDouble(radiusField.getText());
                    // Calculate the area of the circle
                    double area = Math.PI * radius * radius;
                    // Update the label with the calculated area
                    areaLabel.setText("Area of the circle: " + String.format("%.2f",
area));
                } catch (NumberFormatException ex) {
                    // If the input is not a valid number
                    areaLabel.setText("Invalid input");
                }
            }
        });

        JPanel panel = new JPanel(new FlowLayout());
        panel.add(new JLabel("Enter the radius of the circle: "));
        panel.add(radiusField);
        panel.add(calculateButton);
        panel.add(areaLabel);

        setLayout(new BorderLayout());
        add(panel, BorderLayout.CENTER);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(300, 150);
        setLocationRelativeTo(null);
        setVisible(true);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new CircleAreaCalculator();
            }
        });
    }
}
```

b) WAP to display the flag of nepal, drawing a graph along with the reasonable Coordinates. **7**

Answer: Out of Syllabus

5. a) Explain network programming. Compare TCP and UDP sockets. **7**

Answer:

Network programming is the practice of writing programs that communicate with other programs or devices over a network. This communication can occur over various types of networks, including local area networks (LANs), wide area networks (WANs), and the internet. Network programming enables applications to exchange data, share resources, and collaborate with other systems.

Common tasks in network programming include:

- Socket Programming:
- Client-Server Architecture
- Data Serialization
- Error Handling and Reliability
- Security:

	TCP	UDP
Full form	It stands for Transmission Control Protocol.	It stands for User Datagram Protocol.
Type of connection	It is a connection-oriented protocol, which means that the connection needs to be established before the data is transmitted over the network.	It is a connectionless protocol, which means that it sends the data without checking whether the system is ready to receive or not.
Reliable	TCP is a reliable protocol as it provides assurance for the delivery of data packets.	UDP is an unreliable protocol as it does not take the guarantee for the delivery of packets.
Speed	TCP is slower than UDP as it performs error checking, flow	UDP is faster than TCP as it does not guarantee the delivery of data packets.

	control, and provides assurance for the delivery of	
Header size	The size of TCP is 20 bytes.	The size of the UDP is 8 bytes.
Acknowledgment	TCP uses the three-way-handshake concept. In this concept, if the sender receives the ACK, then the sender will send the data. TCP also has the ability to resend the lost data.	UDP does not wait for any acknowledgment; it just sends the data.
Flow control mechanism	It follows the flow control mechanism in which too many packets cannot be sent to the receiver at the same time.	This protocol follows no such mechanism.
Error checking	TCP performs error checking by using a checksum. When the data is corrected, then the data is retransmitted to the receiver.	It does not perform any error checking, and also does not resend the lost data packets.
Applications	This protocol is mainly used where a secure and reliable communication process is required, like military services, web browsing, and e-mail.	This protocol is used where fast communication is required and does not care about the reliability like VoIP, game streaming, video and music streaming, etc.

b) Create a TCP client/server program in which client and server communicate by sending a message to each other. **8**

Answer:

Server:

```
import java.io.*;
import java.net.*;

public class TCPServer {
   public static void main(String[] args) {
      try {
         ServerSocket serverSocket = new ServerSocket(12345);
         System.out.println("Server is running and waiting for connections...");
```

```
        while (true) {
            Socket socket = serverSocket.accept();
            System.out.println("Client connected: " + socket);

            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

            // Read message from client
            String clientMessage = in.readLine();
            System.out.println("Client says: " + clientMessage);

            // Send response to client
            out.println("Hello from server!");

            socket.close();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
  }
}
```

Client:

```
import java.io.*;
import java.net.*;

public class TCPClient {
  public static void main(String[] args) {
    try {
        Socket socket = new Socket("localhost", 12345);
        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

        // Send message to server
        out.println("Hello from client!");

        // Receive response from server
```

```
            String serverMessage = in.readLine();
            System.out.println("Server says: " + serverMessage);

            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

6. a) Draw JDBC architecture and describe different types of JDBC Drivers. 7

Answer: In above set

b) Discuss the tags used in applet. 8

Answer:

In Java, an applet is a special type of program that runs within a web browser. To create and define the behavior of an applet, developers use various HTML tags in conjunction with Java code. Below are the commonly used HTML tags for defining and embedding Java applets within web pages:

- ➔ **<applet>:** The main tag used to define an applet within an HTML document. It has several attributes that specify the applet's properties and behavior, such as code, width, height, and parameters.
- ➔ **<param>:** Used inside the <applet> tag to specify parameters that are passed to the applet. These parameters can be accessed by the applet using the getParameter() method.
- ➔ <**object>:** An alternative tag used to embed Java applets in HTML documents. It provides more flexibility and control over applet embedding compared to the <applet> tag.
- ➔ **<embed>:** Another alternative tag used to embed applets within HTML documents, particularly in older versions of HTML. It is similar to the <object> tag but has limited support in modern web browsers.
- ➔ **<noapplet>:** A deprecated tag used to display a message or alternative content if the browser does not support Java applets.
- ➔ **<comment>:** HTML comment tags can be used to include comments within the HTML code, including comments about the applet or its behavior.

7. Write short notes on: (Any two)

a. Interface and Inner Classes

An interface is a type that defines a set of abstract methods that can be implemented by a class or another interface. An interface can also contain constants, default methods, static methods, and nested types. An interface can be used to achieve multiple inheritance, polymorphism, abstraction, and loose coupling in Java.

An inner class is a class that is defined inside another class or interface. An inner class can be either static or non-static. A static inner class is also known as a nested class, and it does not have access to the instance members of the outer class. A non-static inner class is also known as an inner class, and it has access to the instance members

b. Different event handling mechanisms

Certainly! Event handling is a crucial aspect of programming, especially in graphical user interfaces (GUIs). Let's explore the different mechanisms for handling events:

1.Foreground Events:

a.These events require user interaction to generate. Examples include:

i. Clicking on a button.
ii. Scrolling a scrollbar.
iii. Moving the cursor.

b.Foreground events are associated with components in a GUI.

c. Event classes related to foreground events:
   i. ActionEvent: Indicates that a component-defined action occurred, such as a button click or selecting an item from a menu.
   ii. ItemEvent: Indicates whether an item was selected or not.
   iii. KeyEvent: Occurs due to a sequence of keypresses on the keyboard.
   iv. MouseEvent: Includes mouse-related events like clicks and movements.
   v. TextEvent: Occurs when an object's text changes.

2.Background Events:

a.These events don't require user interaction to generate.

b.Examples:

i.Operating system failures or interrupts.

ii.Operation completion notifications.

c.Background events are typically system-generated.

3.Delegation Event Model:

a.Java follows this model for event handling.

b.It involves two main components:

i.Sources: Events are generated from sources (e.g., buttons, checkboxes, menus).

ii.Listeners: Responsible for handling events generated by sources.

c.To perform event handling:

i.Register the source with the appropriate listener.

ii.Different classes provide registration methods (e.g., addKeyListener() for KeyEvent, addActionListener() for ActionEvent).

4.Event Classes in Java:

a.Here are some commonly used event classes and their associated listener interfaces:

i.ActionEvent: Handled by ActionListener. Represents component-defined actions (e.g., button clicks).

ii.AdjustmentEvent: Emitted by an Adjustable object (e.g., Scrollbar).

iii.ComponentEvent: Indicates component movement, size changes, or visibility changes.

iv.ContainerEvent: Generated when a component is added to or removed from a container.

v.FocusEvent: Related to focus (e.g., gaining or losing focus).

vi.MouseEvent: Includes mouse clicks and movements.

vii.MouseWheelEvent: Specifies mouse wheel rotation.

d. Executing SQL queries from Java

Certainly! Executing SQL queries from Java involves several steps. Let's explore how to do this using the Java Database Connectivity (JDBC) API:

1. Establishing a Connection:
    - First, you need to establish a connection with the database you want to interact with. This connection is represented by a Connection object.
    - You'll need the appropriate JDBC driver for your database system.
    - Example code snippet to establish a connection:
    - **Java**

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DatabaseConnection {
  public static Connection getConnection() throws SQLException {
    String url = "jdbc:mysql://localhost:3306/mydb"; // Replace with your
database URL
    String username = "myuser";
    String password = "mypassword";
    return DriverManager.getConnection(url, username, password);
  }
}
```

    -
    - AI-generated code. Review and use carefully. More info on FAQ.
    -
2. Creating Statements:
    - A Statement represents an SQL statement that you want to execute.
    - There are three types of statements:
        - Statement: For simple SQL statements without parameters.
        - PreparedStatement: For precompiled SQL statements with input parameters.
        - CallableStatement: For executing stored procedures with input and output parameters.
3. Executing Queries:

- To execute a query, call an execute method from the Statement:
  - execute: Returns true if the first object returned by the query is a ResultSet (used for SELECT queries).
  - Example code snippet to execute a SELECT query:
  - Java

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class QueryExample {
    public static void main(String[] args) {
        try (Connection connection = DatabaseConnection.getConnection()) {
            String query = "SELECT COF_NAME, SUP_ID, PRICE FROM COFFEES";
            try (Statement statement = connection.createStatement()) {
                ResultSet resultSet = statement.executeQuery(query);
                while (resultSet.next()) {
                    String coffeeName = resultSet.getString("COF_NAME");
                    int supplierID = resultSet.getInt("SUP_ID");
                    float price = resultSet.getFloat("PRICE");
                    System.out.println(coffeeName + ", Supplier ID: " + supplierID + ", Price:
" + price);
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

  -
  - AI-generated code. Review and use carefully. More info on FAQ.
  -
- Adjust the query according to your database schema and requirements.

4. Processing the ResultSet:
   - The ResultSet contains the results of your query.
   - Use methods like getString, getInt, or getFloat to retrieve data from the result set.

2018 'Fall'

1. **a) How is interface different from abstract class? Explain the use of interface to achieve multiple inheritances.** **8**

Answer: Second part answer in above set

b) What are the uses of abstract keyword? Explain with a suitable program. **7**

Answer:

The main uses of the abstract keyword are:

- ➔ Creating Abstract Classes: Abstract classes serve as blueprints for concrete classes and provide a common interface for subclasses. They often contain abstract methods that define behavior without specifying the implementation details. Abstract classes can also include concrete methods and member variables.
- ➔ Defining Abstract Methods: Abstract methods are methods declared without an implementation. They are intended to be overridden by subclasses to provide specific implementations. Abstract methods are used to define behavior that must be implemented by subclasses but may vary depending on the subclass.

Example:

```
// Abstract class representing a shape
abstract class Shape {
   // Abstract method to calculate area
   public abstract double calculateArea();
}

// Concrete subclass representing a circle
class Circle extends Shape {
   private double radius;

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

   // Implementation of abstract method to calculate area
   @Override
   public double calculateArea() {
      return Math.PI * radius * radius;
   }
}
```

```
// Concrete subclass representing a rectangle
class Rectangle extends Shape {
    private double length;
    private double width;

    // Constructor
    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    // Implementation of abstract method to calculate area
    @Override
    public double calculateArea() {
        return length * width;
    }
}

public class Main {
    public static void main(String[] args) {
        // Create instances of Circle and Rectangle
        Circle circle = new Circle(5);
        Rectangle rectangle = new Rectangle(4, 6);

        // Calculate and print the area of each shape
        System.out.println("Area of Circle: " + circle.calculateArea());
        System.out.println("Area of Rectangle: " + rectangle.calculateArea());
    }
}
```

2. a) Create a class Employee with id, name, post and salary. Create a parameterized constructor to initialize the instance variables. Override the to String() method to display the employee details. **7**

b) What is exception handling? Explain. Write a program to handle Arithmetic Exception. **8**

Answer:

Exception handling is the process of managing and responding to errors or exceptional conditions that occur during the execution of a program. Exceptions are unexpected events or conditions that disrupt the normal flow of execution and may result from various factors such as invalid input, runtime errors, resource unavailability, or external factors beyond the program's control.

Example:

```
public class ExceptionHandlingExample {
    public static void main(String[] args) {
        try {
            int result = divide(10, 0); // Attempt to divide by zero
            System.out.println("Result: " + result); // This line will not be executed
        } catch (ArithmeticException e) {
            // Handle ArithmeticException
            System.out.println("Error: " + e.getMessage());
        }
    }

    // Method to perform division
    public static int divide(int dividend, int divisor) {
        // Attempt to divide by zero
        return dividend / divisor;
    }
}
```

3. a) What is a dialog box? Explain its types. Write a program to create your own dialog box. **7**

Answer:

A dialog box is a small graphical window that appears on the screen to prompt the user for input, display information, or request confirmation. Dialog boxes are commonly used in graphical user interfaces (GUIs) to interact with users and obtain their input or response. They typically contain controls such as buttons, text fields, checkboxes, and dropdown lists.

Types of Dialog Boxes:

➔ **Message Dialog Box:** This type of dialog box displays a message to the user, such as information, warning, or error message. It may contain buttons for the user to acknowledge or dismiss the message.

➔ **Input Dialog Box:** An input dialog box prompts the user to enter some input, such as text, numbers, or selections from a list. It typically contains text fields or dropdown lists for input, along with buttons for confirmation or cancellation.

➔ **Option Dialog Box:** An option dialog box presents the user with a set of options or choices to select from. It may contain checkboxes, radio buttons, or dropdown lists for selecting options, along with buttons for confirmation or cancellation.

➔ **File Dialog Box:** This type of dialog box allows the user to select files or directories from the file system. It provides a graphical interface for browsing files and directories, with options to open, save, or select files.

➔ **Custom Dialog Box:** Custom dialog boxes are user-defined or customized dialog boxes created by developers to meet specific requirements. They can contain any combination of controls and components tailored to the application's needs. Here's an example program in Java that creates a custom dialog box using Swing:

```
import javax.swing.*;

public class CustomDialogBoxExample {
    public static void main(String[] args) {
        // Create a JFrame to host the dialog box
        JFrame frame = new JFrame("Custom Dialog Box Example");
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null); // Center the frame

        // Create and display a custom dialog box
        JOptionPane.showMessageDialog(frame, "This is a custom dialog box",
                "Custom Dialog Box", JOptionPane.INFORMATION_MESSAGE);

        // Display the JFrame
        frame.setVisible(true);
    }
}
```

b) What are the different types of streams supported by java? Explain. A data file "emp.txt" contains name, address and salary of 30 employees. Write a program to display only those records who are from "Kathmandu". **8**

4. **a) Create a swing GUI that contains a two buttons (add and subtract) and three text fields. When the buttons are clicked sum or difference of values of first two text fields should be displayed in the third text field Respectively.** **8**

Answer:

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

public class CalculatorGUI extends JFrame {
   private JTextField textField1;
   private JTextField textField2;
   private JTextField resultField;
   private JButton addButton;
   private JButton subtractButton;

   public CalculatorGUI() {
      super("Calculator");

      textField1 = new JTextField(10);
      textField2 = new JTextField(10);
      resultField = new JTextField(10);
      resultField.setEditable(false); // Make result field read-only

      addButton = new JButton("Add");
      subtractButton = new JButton("Subtract");

      // Add ActionListener to the add button
      addButton.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
            try {
               // Parse text field values as integers
               int num1 = Integer.parseInt(textField1.getText());
               int num2 = Integer.parseInt(textField2.getText());
               // Calculate sum and display result
               int sum = num1 + num2;
               resultField.setText(Integer.toString(sum));
            } catch (NumberFormatException ex) {
               // If input is not valid integers
               resultField.setText("Invalid input");
            }
         }
      });
```

```
    // Add ActionListener to the subtract button
    subtractButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            try {
                // Parse text field values as integers
                int num1 = Integer.parseInt(textField1.getText());
                int num2 = Integer.parseInt(textField2.getText());
                // Calculate difference and display result
                int difference = num1 - num2;
                resultField.setText(Integer.toString(difference));
            } catch (NumberFormatException ex) {
                // If input is not valid integers
                resultField.setText("Invalid input");
            }
        }
    });

    JPanel panel = new JPanel(new FlowLayout());
    panel.add(new JLabel("Number 1: "));
    panel.add(textField1);
    panel.add(new JLabel("Number 2: "));
    panel.add(textField2);
    panel.add(addButton);
    panel.add(subtractButton);
    panel.add(new JLabel("Result: "));
    panel.add(resultField);

    setLayout(new BorderLayout());
    add(panel, BorderLayout.CENTER);

    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(300, 200);
    setLocationRelativeTo(null);
    setVisible(true);
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
```

```
            @Override
            public void run() {
               new CalculatorGUI();
            }
        });
    }
}
```

b) Explain the use of URL and URL Connection class with a suitable program. 7

Answer : Refer to answer 2020 5b

5. a) Write a program to display all records from the database table. Assume the name of database and table yourself. 7

Answer:

```
import java.sql.*;

public class ShowDatabase {
   public static void main (String[]args) throws ClassNotFoundException,
SQLException{
     Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
      String jdbcUrl =
"jdbc:sqlserver://localhost\\sqlexpress:1433;databaseName=Java;encrypt=true;tru
stServerCertificate=true";
       Connection conn = DriverManager.getConnection(jdbcUrl,"sa","S2cond2nd");
       String selectValue = "Select * from employee";
       PreparedStatement ps =conn.prepareStatement(selectValue);

           ResultSet resultSet = ps.executeQuery();
       while (resultSet.next()) {
int id = resultSet.getInt("id");
String name = resultSet.getString("name");
String post = resultSet.getString("post");
String salary  = resultSet.getString("salary");
System.out.println("ID: " + id + ", Name: " + name +", Post: " + post+", Salary: " +
salary);
   }
}
}
```

b) Differentiate between TCP and UDP. Create a TCP client application that takes input from user and sends it to the server. **8**

Answer:

Basis	Transmission Control Protocol (TCP)	User Datagram Protocol (UDP)
Type of Service	TCP is a connection-oriented protocol. Connection orientation means that the communicating devices should establish a connection before transmitting data and should close the connection after transmitting the data.	UDP is the Datagram-oriented protocol. This is because there is no overhead for opening a connection, maintaining a connection, or terminating a connection. UDP is efficient for broadcast and multicast types of network transmission.
Reliability	TCP is reliable as it guarantees the delivery of data to the destination router.	The delivery of data to the destination cannot be guaranteed in UDP.
Error checking mechanism	TCP provides extensive error-checking mechanisms. It is because it provides flow control and acknowledgment of data.	UDP has only the basic error-checking mechanism using checksums.
Acknowledgment	An acknowledgment segment is present.	No acknowledgment segment.
Sequence	Sequencing of data is a feature of Transmission Control Protocol (TCP). this means that packets arrive in order at the receiver.	There is no sequencing of data in UDP. If the order is required, it has to be managed by the application layer.
Speed	TCP is comparatively slower than UDP.	UDP is faster, simpler, and more efficient than TCP.

Retransmission	Retransmission of lost packets is possible in TCP, but not in UDP.	There is no retransmission of lost packets in the User Datagram Protocol (UDP).
Header Length	TCP has a (20-60) bytes variable length header.	UDP has an 8 bytes fixed-length header.
Weight	TCP is heavy-weight.	UDP is lightweight.
Handshaking Techniques	Uses handshakes such as SYN, ACK, SYN-ACK	It's a connectionless protocol i.e. No handshake
Broadcasting	TCP doesn't support Broadcasting.	UDP supports Broadcasting.
Protocols	TCP is used by HTTP, HTTPs, FTP, SMTP and Telnet.	UDP is used by DNS, DHCP, TFTP, SNMP, RIP, and VoIP.
Stream Type	The TCP connection is a byte stream.	UDP connection is a message stream.
Overhead	Low but higher than UDP.	Very low.
Applications	This protocol is primarily utilized in situations when a safe and trustworthy communication procedure is necessary, such as in email, on the web surfing, and in military services.	This protocol is used in situations where quick communication is necessary but where dependability is not a concern, such as VoIP, game streaming, video, and music streaming, etc.

```
import java.io.*;
import java.net.*;

public class TCPClient {
    public static void main(String[] args) {
        try {
```

```
        // Create a socket to connect to the server (change the IP address and
    port as needed)
        Socket socket = new Socket("localhost", 12345);

        // Create input and output streams for communication with the
    server
        BufferedReader userInput = new BufferedReader(new
    InputStreamReader(System.in));
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

        // Read user input from the console
        System.out.print("Enter message to send to server: ");
        String message = userInput.readLine();

        // Send the message to the server
        out.println(message);

        // Close the socket and streams
        out.close();
        userInput.close();
        socket.close();
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
}
```

6. **a) How is applet different from normal java program? List out the steps for converting applet into application.** **7**
   Answer:

Applets and normal Java programs (also known as applications) differ in several key aspects:

Execution Environment:

- Applets are designed to run within a web browser or applet viewer and are executed by a web browser's Java plugin or by a standalone applet viewer.
- Normal Java programs are standalone applications that run independently of a web browser or applet viewer and are executed from the command line or by double-clicking on the executable JAR file.

GUI Components:

- Applets often use AWT (Abstract Window Toolkit) or Swing components for creating user interfaces within a web browser.
- Normal Java programs can also use AWT or Swing components for creating graphical user interfaces, but they are not limited to web-based environments and can be used to create standalone desktop applications.

Security Restrictions:

- Applets are subject to security restrictions imposed by the web browser environment, such as sandboxing and access controls, to prevent potentially harmful actions.
- Normal Java programs do not have the same security restrictions and have more freedom in terms of accessing system resources and performing actions.

Deployment:

- Applets are typically deployed and accessed through a web server, and the user interacts with them within a web browser.
- Normal Java programs can be distributed as executable JAR files or standalone applications, which users can run locally on their machines without requiring a web browser.

Steps for Converting an Applet into an Application:

Remove Applet-Specific Code:

- Remove any applet-specific methods and dependencies, such as init(), start(), stop(), and destroy().
- Replace applet-specific GUI components with AWT or Swing components, if necessary.

Main Method:

- Add a main() method to the class to serve as the entry point for the application.
- Move any initialization code from the init() method to the main() method.

GUI Initialization:

- Initialize GUI components within the main() method or in other appropriate initialization methods.
- Replace any applet-specific initialization code with appropriate initialization logic for standalone applications.

Event Handling:

- Implement event handling logic using listeners or event handlers, similar to how it's done in normal Java applications.
- Handle user input and interaction using appropriate event listeners or handlers.

Compile and Run:

- Compile the modified Java class(es) using the Java compiler (javac).
- Run the compiled application using the Java Virtual Machine (java), either from the command line or by double-clicking on the executable JAR file.

b) How can you create closable frames in swing and AWT? Write a program to draw a bar chart. **8**

Answer:

Note:Remaining Part is Out of Syllabus

7. **7. Write short notes on: (Any two)**
   a. **Reflection API**
      (2017 Fall 7a same)

   b. **History of Java**

Java is a popular programming language that was developed by James Gosling at Sun Microsystems and released in 19951. The original goal of the language was to design a language for small, embedded systems in electronic appliances like set-top boxes. However, it was best suited for internet programming.

Here are some key points in the history of Java:

- The project was initiated by James Gosling, Mike Sheridan, and Patrick Naughton in June 19912.
- Initially, it was called “Greentalk” and then "Oak"2.
- In 1995, Oak was renamed as “Java” because Oak was already a trademark by Oak Technologies.
- Java was chosen as the name while James Gosling was having a cup of coffee near his office.
- Java was incorporated by Netscape and became a popular language for internet programming.
- JDK 1.0 was released on January 23, 19962.
- After the first release of Java, there have been many additional features added to the language.

c. JDBC API

The JDBC API is an application programming interface that allows Java programs to access and manipulate data in various types of databases. The JDBC API consists of two packages: java.sql and

javax.sql, which provide classes and interfaces for connecting to databases, executing SQL statements, processing query results, and managing transactions. The JDBC API also defines a standard way for Java applications to use drivers, which are libraries that mediate between the JDBC API and the specific database protocols. To use the JDBC API, you need to have a driver that supports your database and include it in your classpath.

2018 'Spring'

1. **a) What do you mean by architectural-neutral? What are wrapper classes? Explain.**
   Answer: 2021 '1a'
   b) Mention the scope of all modifiers (private, default, protected and public). Write suitable program to illustrate the concept.

In Java, access modifiers control the visibility and accessibility of classes, variables, methods, and constructors. There are four access modifiers in Java: private, default (also known as package-private), protected, and public. Here's a brief overview of each modifier's scope:

- Private: The private modifier limits the access to the member only within the same class. It cannot be accessed from outside the class, not even by subclasses.
- Default (Package-private): If no access modifier is specified (i.e., no modifier is used), it's considered to have default accessibility. The member is accessible only within the same package. It cannot be accessed from outside the package, even by subclasses in different packages.
- Protected: The protected modifier allows access within the same package, as well as by subclasses in different packages (through inheritance). It grants access to subclasses and classes in the same package.
- Public: The public modifier allows access from anywhere. The member can be accessed from any other class, regardless of the package or inheritance relationship.

Example:package com.mycompany.accessmodifiers;

// Class with different access modifiers

```
public class MyClass {
    private int privateVar = 10;
    int defaultVar = 20; // Default access modifier
    protected int protectedVar = 30;
    public int publicVar = 40;

    // Method with private access modifier
    private void privateMethod() {
        System.out.println("Private method");
    }

    // Method with default access modifier
    void defaultMethod() {
        System.out.println("Default method");
    }

    // Method with protected access modifier
    protected void protectedMethod() {
        System.out.println("Protected method");
    }

    // Method with public access modifier
    public void publicMethod() {
        System.out.println("Public method");
    }
}

// Subclass in a different package
package com.mycompany.subpackage;

import com.mycompany.accessmodifiers.MyClass;

// Subclass of MyClass
public class SubClass extends MyClass {
    public void accessVariables() {
        // Accessing variables from the superclass
        // privateVar is not accessible here
        // defaultVar is not accessible here (different package)
        System.out.println("Protected variable: " + protectedVar);
        System.out.println("Public variable: " + publicVar);
```

```
    }

    public void accessMethods() {
        // Accessing methods from the superclass
        // privateMethod is not accessible here
        // defaultMethod is not accessible here (different package)
        protectedMethod();
        publicMethod();
    }
}

// Main class to demonstrate access modifiers
package com.mycompany.main;

import com.mycompany.accessmodifiers.MyClass;
import com.mycompany.subpackage.SubClass;

public class Main {
    public static void main(String[] args) {
        MyClass myObject = new MyClass();

        // Accessing variables and methods from the same class
        // privateVar and privateMethod are not accessible here
        System.out.println("Default variable: " + myObject.defaultVar);
        System.out.println("Protected variable: " + myObject.protectedVar);
        System.out.println("Public variable: " + myObject.publicVar);
        // myObject.privateMethod(); // Error: privateMethod() has private access in MyClass
        myObject.defaultMethod();
        myObject.protectedMethod();
        myObject.publicMethod();

        // Creating an instance of the subclass to access variables and methods
        SubClass subObject = new SubClass();
        subObject.accessVariables();
        subObject.accessMethods();
    }
}
```

2. a) What is method overloading? Can you override a private or static method in Java? Explain with an example. **8**

Answer: 2021 '2b'

b) How does interface differ from abstract classes? Elaborate using code snippets to justify.

Answer: Answer in above set

3. **a) Explain about FileReader and Buffered Writer class. How do you create own exception subclasses? Explain with an example.**

   b) What are the differences between applet and normal java program?

   Create an applet with the functionalities to play, stop and repeat the audio.

   Answer in 2021 3b

4. **a) Create a swing GUI that contains a text field and a button. When the button is pressed the content in the text field should be changed into uppercase and background color of text field should be changed.**

   Answer:

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

public class UppercaseTextFieldGUI extends JFrame {
    private JTextField textField;
    private JButton changeButton;

    public UppercaseTextFieldGUI() {
        super("Uppercase Text Field GUI");

        // Create text field
        textField = new JTextField(20);

        // Create button
        changeButton = new JButton("Change to Uppercase");
        changeButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Change text to uppercase
                String text = textField.getText().toUpperCase();
                textField.setText(text);
                // Change background color
                textField.setBackground(Color.YELLOW);
            }
        });
```

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

        // Add components to the frame
        add(textField);
        add(changeButton);

        // Set window properties
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(300, 100);
        setLocationRelativeTo(null);
        setVisible(true);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new UppercaseTextFieldGUI();
            }
        });
    }
}
```

b) Demonstrate various drawing methods. How do you create, load and display image?

Answer: Out of Syllabus

5. **a) Differentiate between TCP and UDP sockets. Explain InetAddress class.**

Answer: First part refer to 2018 Fall 5b

The InetAddress class in Java is a class that represents an IP address, either an IPv4 address or an IPv6 address. It provides methods for performing various operations related to IP addresses, such as obtaining the host name corresponding to an IP address, obtaining the IP address corresponding to a host name, and checking the reachability of a given IP address or host.

```
import java.net.*;

public class InetAddressExample {
    public static void main(String[] args) {
```

```
        try {
            // Obtain InetAddress object representing the local host
            InetAddress localhost = InetAddress.getLocalHost();
            System.out.println("Local Host:");
            System.out.println("Host Name: " + localhost.getHostName());
            System.out.println("IP Address: " + localhost.getHostAddress());

            // Resolve host name to IP address
            InetAddress googleAddress = InetAddress.getByName("www.google.com");
            System.out.println("\nGoogle Address:");
            System.out.println("Host Name: " + googleAddress.getHostName());
            System.out.println("IP Address: " + googleAddress.getHostAddress());

            // Check reachability of Google
            System.out.println("\nGoogle Reachability:");
            System.out.println("Is Reachable: " + googleAddress.isReachable(5000)); //
    Timeout: 5000 ms
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

b) What are some key classes defined in java to work with datagrams? How do you get a list of IP addresses that are assigned to a network interface?

6. **a) What is the benefit of using Prepared Statement in java? What is JDBC database connection pool? How to setup in Java?**

The benefits of using a PreparedStatement in Java for database operations include:

Improved Performance: Prepared statements are precompiled SQL statements that are stored in the database server's memory. This allows the database server to optimize query execution and reuse execution plans, resulting in improved performance, especially for frequently executed queries.

Protection Against SQL Injection: Prepared statements automatically escape special characters and sanitize input data, reducing the risk of SQL injection attacks. Parameters in prepared statements are treated as placeholders and are not directly concatenated with SQL strings, making them immune to SQL injection vulnerabilities.

Parameterized Queries: Prepared statements support parameterized queries, allowing you to pass parameters dynamically to the SQL statement. This makes it

easier to create flexible and reusable queries, as you can easily substitute different parameter values without modifying the SQL statement itself.

Ease of Use: Using prepared statements simplifies the process of executing SQL queries with parameterized values. You can set parameter values using dedicated setter methods without worrying about proper escaping or formatting.

As for JDBC database connection pool, it is a pool of database connections that are created and managed by the application server or a separate connection pooling library. The purpose of connection pooling is to improve the efficiency and scalability of database access in Java applications by reusing existing database connections instead of creating new ones for each database operation.

Setting up a JDBC database connection pool in Java typically involves the following steps:

Choose a Connection Pooling Library: There are several popular connection pooling libraries available for Java, such as Apache Commons DBCP, HikariCP, and c3p0. Choose the one that best fits your requirements and integrate it into your project by adding the corresponding dependency to your build configuration (e.g., Maven or Gradle).

Configure Connection Pool Properties: Configure the connection pool properties such as maximum number of connections, minimum number of connections, timeout settings, and other performance tuning parameters. These settings can usually be specified in a configuration file or programmatically when creating the connection pool object.

Initialize the Connection Pool: Initialize the connection pool by creating an instance of the connection pool class provided by the chosen library and configuring it with the desired properties.

Acquire and Release Connections: Use the connection pool to acquire database connections when needed by requesting connections from the pool. Perform database operations using the acquired connections and release them back to the pool when they are no longer needed.

Shutdown the Connection Pool: When the application is shutting down or no longer needs the database connections, properly shut down the connection pool to release all resources and close all connections.

b) A database "testdb" contains a table "employee" with some records having id, name, post, salary. Write a program to update the salary to 50000 whose post is "Manager".

Answer:

```
import java.sql.*;

public class Update1 {
```

```
    public static void main (String[]args) throws ClassNotFoundException, SQLException{
      Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
       String jdbcUrl =
"jdbc:sqlserver://localhost\\sqlexpress:1433;databaseName=testdb;encrypt=true;trustSer
verCertificate=true";
        Connection conn = DriverManager.getConnection(jdbcUrl,"sa","S2cond2nd");
        String updateValue ="Update employee set salary= '50000' Where post='manager'";
        PreparedStatement ps =conn.prepareStatement(updateValue);
        int result=ps.executeUpdate();
        if(result>0){
           System.out.println("Update successfully");

        }
        else{
           System.out.println("Update fail");
        }
     }
}
```

7. **Write short notes on (Any Two):**
   a. **Inner Class**

   In Java, an inner class, or nested class, is a class that is declared inside another class or interface. Inner classes were introduced to logically group classes and interfaces in one place, making the code more readable and maintainable. They can access all the members of the outer class, including private data members and methods.

Example

```
class OuterClass {
  int x = 10;

  class InnerClass {
    int y = 5;
  }
}

public class Main {
  public static void main(String[] args) {
```

```
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.y + myOuter.x);
  }
}


// Outputs 15 (5 + 10)
```

b. C++ Vs Java

C++:

- C++ is platform-dependent.
- It is mainly used for system programming.
- C++ supports the goto statement.
- It supports multiple inheritance.
- C++ supports operator overloading.
- It supports pointers.
- C++ uses compiler only.
- It supports both call by value and call by reference.
- C++ supports structures and unions.
- It doesn't have built-in support for threads.
- C++ doesn't support documentation comments.
- It supports the virtual keyword.
- C++ doesn't support the >>> operator.
- It always creates a new inheritance tree.
- C++ is nearer to hardware.

Java:

- Java is platform-independent.
- It is mainly used for application programming.
- Java doesn't support the goto statement.
- It doesn't support multiple inheritance through class.
- Java doesn't support operator overloading.
- It supports pointer internally.
- Java uses both compiler and interpreter.
- It supports call by value only.
- Java doesn't support structures and unions.
- It has built-in thread support.
- Java supports documentation comment.

- It has no virtual keyword.
- Java supports unsigned right shift >>> operator.
- All classes in Java are the child of the Object class.

c. Types of JDBC drivers

Java Database Connectivity (JDBC) uses drivers to connect with the database. There are four types of JDBC drivers:

1. **JDBC-ODBC Bridge Driver (Type 1)**: This driver uses the ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls.
2. **Native-API Driver (Type 2):** Also known as a partially Java driver, this driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API.
3. **Network Protocol Driver (Type 3)**: This is a fully Java driver that uses middleware (application server) to convert JDBC calls directly or indirectly into the vendor-specific database protocol.
4. **Thin Driver (Type 4):** This is also a fully Java driver that converts JDBC calls directly into the vendor-specific database protocol.

2017 'Fall'

1. **a) Justify the statement," Java is designed for distributed application". 7**
   Explain the different types of Java variables.

Java and Distributed Applications:

Java is designed for distributed applications in several ways:

1. **Platform Independence**: Java "write once, run anywhere" principle allows developers to build applications that can run on any device, regardless of the device's architecture.
2. **Networking Support:** Java has extensive networking capabilities built into the language, making it easier to work with resources across a network.
3. **Extended Libraries**: Java libraries have features like Remote Method Invocation (RMI) to implement distributed objects. RMI allows an object to invoke methods on an object running in another JVM.
4. **Integration of New Technology**: Java provides interoperability with other software systems through bindings like COBRA.
5. **Security:** Java has a distinctive feature of moving the code between machines securely and then executing the sandbox code that allows running untrusted code in a secure way.

Types of Variables in Java:

There are three types of variables in Java:

1. **Local Variables:** These are declared within a block or method or constructor. They are created when the block is entered or the function is called and destroyed after exiting from the block or when the call returns from the function.
2. **Instance Variables**: These are declared inside the class but outside the body of the method. They are not declared as static. Their value is instance-specific and is not shared among instances.
3. **Static Variables**: These are declared as static. They cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

Each type of variable has its own lifecycle and scope, and they are used in different contexts based on the requirements of the program.

b) Explain with example: static block, static variable and static method. 8

Why main method is always static and public in Java?

Static Block, Static Variable, and Static Method in Java:

1. **Static Block**: A static block is a block of code that is run when the class is loaded into memory. It is used for static initialization of a class. The code inside the static block is executed only once: the first time the class is loaded into memory. Here an example:

```
Java
class Test {
   static int i;
   static {
      i = 10;
      System.out.println("static block called");
   }
}
```

2. **Static Variable**: A static variable is a variable that is associated with the class, not objects of the class. Static variables are initialized only once, at the start of the execution. Here an example.

```
Java
class Test {
   static int i = 10;  // static variable
}
```

3. **Static Method:** A static method is a method that belongs to the class rather than the instance of a class. The static method can access static data members and can change the value of it. Here an example:

```
Java
class Test {
   static int i = 10;  // static variable
   static void method() {  // static method
      System.out.println("Value of i: " + i);
   }
}
```

Why is the main method always static and public in Java?

The main method in Java is always static and public due to the following reasons:

1.**Static**: The main method is static so that the Java Virtual Machine (JVM) can call it without creating an instance of the class. If it were not static, the JVM would need to create an instance of the class before the main method could be called, but there is no way to call a constructor from the JVM.

2.**Public**: The main method is public because it must be accessible to the JVM, which is outside the scope of the class and package. If the main method were not public, the JVM would not be able to access it.

2. a) How can you achieve multiple inheritance in java? Explam with a suitable program. 7

Answer in above program:

b) What are the main uses of super, finalize and this keyword? Explain how a package is created and accessed while developing an application in Java. 8

Uses of super, finalize, and this keywords in Java:

1. **super**: The super keyword in Java is a reference variable that is used to refer to the parent class. It is used in three contexts:
    - To refer to the immediate parent class instance variable.
    - To invoke the immediate parent class method.
    - To invoke the immediate parent class constructor.
2. **this:** The this keyword in Java is a reference variable that is used to refer to the current class. It is used in various contexts.
    - To refer to the current class instance variable.
    - To invoke or initiate the current class constructor.
    - Can be passed as an argument in the method call.
    - Can be passed as an argument in the constructor call.
    - Can be used to return the current class instance.
3. **finalize**: The finalize() method in Java is a special method that is invoked by the garbage collector before an object is being garbage collected. This method can be used to ensure that an object terminates cleanly.

Creating and Accessing a Package in Java:

A package in Java is a mechanism to encapsulate a group of classes, sub-packages, and interfaces. Here are the steps to create and access a package:

1.Creating a Package:

a. Choose a name for the package.

b. Include the package command as the first line of code in your Java Source File.
c. The source file contains the classes, interfaces, etc., that you want to include in the package.
d. Compile to create the Java packages.

2.Accessing a Package:

a.If you want to use a package in a Java program, it is necessary to import that package at the top of the program.

b.Use the import keyword before the package name.

c.Syntax: import packageName.

3. a)Write a program to store object of a class Student into a file "student.dat" also read the objects from same file and display the state of objects on the screen. Handle possible exceptions. **7**

Java

```
import java.io.*;
class Student implements Serializable {
   String name;
   int age;

   public Student(String name, int age) {
      this.name = name;
      this.age = age;
   }

   @Override
   public String toString() {
      return "Student{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
   }
}

public class Main {
   public static void main(String[] args) {
      // Create a new student object
```

```
        Student student = new Student("John Doe", 20);

        try {
            // Write the student object to a file
            FileOutputStream fileOut = new FileOutputStream("student.dat");
            ObjectOutputStream out = new ObjectOutputStream(fileOut);
            out.writeObject(student);
            out.close();
            fileOut.close();
            System.out.println("Serialized data is saved in student.dat");
        } catch (IOException i) {
            i.printStackTrace();
        }

        student = null;

        try {
            // Read the student object from the file
            FileInputStream fileIn = new FileInputStream("student.dat");
            ObjectInputStream in = new ObjectInputStream(fileIn);
            student = (Student) in.readObject();
            in.close();
            fileIn.close();
        } catch (IOException i) {
            i.printStackTrace();
            return;
        } catch (ClassNotFoundException c) {
            System.out.println("Student class not found");
            c.printStackTrace();
            return;
        }

        // Print the state of the student object
        System.out.println("Deserialized Student...");
        System.out.println(student);
    }
}
```

b)Differentiate error with exception. Write a program to create your **8**

own exception class in java.

Error vs Exception in Java:

- **Error**: Errors are serious problems that a reasonable application should not try to catch. They are usually caused by severe issues that are outside the control of the program, such as running out of memory or a system crash. Errors are conditions that cannot be recovered from and mostly occur at runtime. Examples include OutOfMemoryError, StackOverflowError, and NoClassDefFoundError.
- **Exception**: Exceptions are conditions that a reasonable application might want to catch. They are used to handle errors that can be recovered from within the program. Exceptions occur during the execution of a program i.e., at runtime, and disrupt the normal flow of the program instructions. Examples include NullPointerException, IllegalArgumentException, and IOException.

In Java, both Errors and Exceptions are subclasses of java.lang.Throwable class.

Creating a Custom Exception Class in Java:

You can create your own exceptions in Java, known as custom exceptions or user-defined exceptions. Here an example of how to create a custom exception class:

java

```
// Creating a custom exception
class MyCustomException extends Exception {
    public MyCustomException(String message) {
        super(message);
    }
}

// Using the custom exception
public class Test {
    public static void main(String[] args) {
        try {
            throw new MyCustomException("This is a custom exception");
        } catch (MyCustomException e) {
            System.out.println(e.getMessage());
        }
    }
```

```
}
```

In this example, MyCustomException is a custom exception class that extends the Exception class. The MyCustomException class has a constructor that accepts a string parameter and passes it to the super constructor of the Exception class. This string represents the message that will be displayed when the exception is thrown.

4. **a) Explain Applet Architecture. How can you convert applet to application? Explain with suitable example.** **7**

Applet Architecture:

An applet in Java is a class that extends the java.applet.Applet class1. The Applet class provides the standard interface between the applet and the browser environment1. It has five core lifecycle methods.

1. init(): This method is used to initialize the Applet. It is invoked only once..
2. start(): This method is invoked after the init() method or when the browser is maximized. It is used to start the Applet.
3. stop(): This method is used to stop the Applet. It is invoked when the Applet is stopped or the browser is minimized.
4. destroy(): This method is used to destroy the Applet. It is invoked only once.
5. paint(Graphics g): This method is used to paint the Applet. It provides a Graphics class object that can be used for drawing.

Converting Applet to Application:

To convert an applet to an application, you need to make a few changes:

1. Replace extends Applet with extends JFrame for the class.
2. Change the init() method's name to the constructor.
3. Create a new main() method.
4. Remove the import statement for the java.applet.Applet class.
5. Add window listener's method to handle window events

In this example, the HelloWorldApplet class is converted to the HelloWorldApplication class. The paint() method is moved to a JPanel that is added to the JFrame. The main() method creates an instance of the HelloWorldApplication and makes it visible.

b) How Event is handled in Java? Write a Java program to create a swing application with 3 buttons representing your favourite colors. When a **8**

button is clicked, the background color must change to the corresponding color.

Event handling in Java is the mechanism that controls an event and decides what should happen when an event occurs. Java follows the Delegation Event Model for event handling. There are two major components in this model:

1. Event Source: Events are generated from the source. Sources can be buttons, checkboxes, list, menu-item, scrollbar, text components, windows, etc.
2. Event Listener: Listeners are used for handling the events generated from the source. Each listener represents interfaces that are responsible for handling events.

To perform event handling, we need to register the source with the listener. Different classes provide different registration methods. For example, for KeyEvent we use addKeyListener() to register.

Here's a simple Java Swing application with 3 buttons representing different colors. When a button is clicked, the background color changes to the corresponding color:

Java

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

public class ColorChanger extends JFrame {
   public ColorChanger() {
      // Create buttons
      JButton redButton = new JButton("Red");
      JButton greenButton = new JButton("Green");
      JButton blueButton = new JButton("Blue");

      // Add action listeners to buttons
      redButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            getContentPane().setBackground(Color.RED);
         }
      });

      greenButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            getContentPane().setBackground(Color.GREEN);
         }
      });
```

```
        blueButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                getContentPane().setBackground(Color.BLUE);
            }
        });

        // Add buttons to frame
        setLayout(new FlowLayout());
        add(redButton);
        add(greenButton);
        add(blueButton);

        // Set frame properties
        setSize(300, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }

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

.In this program, each button is associated with an ActionListener. When a button is clicked, the actionPerformed method is called, which changes the background color of the content pane.

5. a) Differentiate between GridLayout and GridBag Layout. Write a program to show the use of BorderLayout. **7**

First part answer in above set

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

public class BorderLayoutExample {
    public static void main(String[] args) {
        // Create a JFrame
        JFrame frame = new JFrame("BorderLayout Example");

        // Create buttons for each region of the BorderLayout
        JButton northButton = new JButton("North");
```

```
        JButton southButton = new JButton("South");
        JButton eastButton = new JButton("East");
        JButton westButton = new JButton("West");
        JButton centerButton = new JButton("Center");

        // Set layout manager to BorderLayout
        frame.setLayout(new BorderLayout());

        // Add buttons to the frame with specified regions
        frame.add(northButton, BorderLayout.NORTH);
        frame.add(southButton, BorderLayout.SOUTH);
        frame.add(eastButton, BorderLayout.EAST);
        frame.add(westButton, BorderLayout.WEST);
        frame.add(centerButton, BorderLayout.CENTER);

        // Set window properties
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }
}
```

b) What is GUI programming? Write a program to draw Nepali flag using graphics. **8**

Answer: Out of Syllabus

6. **a) What is Java URL processing? Demonstrate URLConnection class method with suitable example program.** **7**

Answer : Answer in above set

b) What are the different types of JDBC statements available? Explain with example. **8**

Answer: Answer in above set

7. **Write short notes on: (Any two).**

   a. **Socket Programming in Java**

   Socket programming in Java is used for communication between applications running on different Java Runtime Environments (JREs). It can be connection-oriented or connection-less

```
import java.io.*;
```

```
import java.net.*;

public class MyServer {

public static void main(String[] args){

try{

ServerSocket ss=new ServerSocket(6666);

Socket s=ss.accept();//establishes connection

DataInputStream dis=new DataInputStream(s.getInputStream());

String  str=(String)dis.readUTF();

System.out.println("message= "+str);

ss.close();

}catch(Exception e){System.out.println(e);}

}

}
```

b. **JDBC Driver types**
   In the above set
c. **Reflection in Java**
   2017 Fall 7a