Object Oriented Programming using Java

Assignment - II

(1.) Explain about different types of control statements with an example.

In Java, there are several types of control statements that allow you to control the flow of execution in your programs. Control statements are fundamental in java programming as they allow you to make decisions, repeat tasks and handle different scenarios based on conditions. Let's discuss each type with an example :-

(1.) Conditional statement (if-else):-

Conditional statements allow you to execute certain blocks of code based on whether a condition is true or false.

Example-
```
int x = 10;
if (x>0) {
    System.out.println("X is positive");
}
else if (x<0) {
    System.out.println("X is negative");
}
else {
    System.out.println("X is zero");
}
```

In this Example.- if 'x' is greater than 0, it prints "x is positive". if 'x' is less than 0, it prints "x is negative". otherwise, it prits "x is zero".

(2.) Switch Statement:-

The switch statement allows you to select one of many code blocks to execute based on the value of an expression.

Example:-

```
int day = 2;
Switch (dy) {
        case 1:
            System.out.println("Monday");
            break;
        Case 2:
            System.out.println("Tuesday");
            break;
        Case 3:
            System.out.println("Wednesday");
            break;
        default:
            System.out.println("Invalid day");
            break;
}
```

The switch statement chaks the value of 'day' and executes the corresponding case. if no cases match, the 'default' case is executed.

(3.) Loop Statements:-

Loop statements allow you to reapeat a block of code multiple times untill a condition is met or for a specified number of times.

(1) while loop:-

Example -
```
int count = 0;
while (count < 5) {
    System.out.println("Count is: " + count);
    count++;
}
```

This 'while' loop executes the block of code inside it as long as 'count' is less than 5.

(2.) For loop:-

Example -
```
for (int i=1; i<=5; i++) {
    System.out.println("i is: " + i);
}
```

This 'For' loop iterates from 'i=1' to 'i<=5', incrementing 'i' by each time, and prints the value of 'i'.

(2.) Write about multithreading programming and explain how synchronization is achieved.

Multithreading in java allows concurrent execution of multiple threads within a single Java Program. Threads are lightweight processes within the java virtual Machine (JVM) that can execute independently and share resources such as memory. This capability is essential for applications that require multitasking, handling multiple task simultaneously, or achieving better performance by leveraging mordern multicore processors effectively.

Thread States:- Threads in java can be in different states throughout their lifecycle:

- New:- when a thread instance is created but not yet started.
- Rynnable:- when the thread is ready to run and waiting for CPU time(either running or waiting for execution).
- Blocked/Waiting:- when the thread is waiting for a monitor lock to enter a synchronized block/method or waiting indefinitely for another thread (using methods like 'wait()' or 'join()').
- Timed Waiting:- when the thread is waiting for another thread for a specified period (using methods like 'sleep()' or 'wait(timeout)').

- Terminated :- when the thread completes its execution or is terminated.

Thread Synchronization :- Concurrency in java requires careful synchronization to manage access to shared resources and avoid race conditions where multiple threads access and modify shared data simultaneously. key machanisms for synchronization include:

- Synchronized Methods :- use the 'synchronized' keyword to make methods thread-safe by allowing only one thread to execute them at a time.
- Synchronized Blocks :- use synchronized blocks to control access to critical sections of code with more flexibility than Synchronized methods.
- Volatile keyword :- use the 'volatile' keyword to ensure visibility of changes to variables across threads preventing threads from caching variables locally.

Example :-

```
Public class synchronizationExample {
    Private static int counter = 0;
    public static void main (String[] args) {
        Thread thread1 = new Thread (()-> {
            for (int i = 0; i < 100; i++) {
                incrementCounter();
            }
        });
```

```
        Thread thread2 = new Thread(() -> {
            for (int i=0; i<100; i++){
                incrementCounter();
            }
        });

        thread1.start();
        thread2.start();

        try {
            thread1.join();
            thread2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Final counter value: "+counter);
    }
    Private synchronized static void incrementCounter(){
        counter++;
    }
}
```

(3.) What is the usage and purpose of string tokenization with an example.

String tokenization refers to the process of breaking down a string into smaller components, known as tokens, based on specific delimiters (character that separate tokens). This technique is commonly used in various applications such as parsing text, reading input from files or processing data from network protocols where structured data needs to be extracted and processed.

Here are some usages and purposes of string tokenization:

Usages and Purposes of string Tokenization :-

(1.) Parsing and Extracting Data :- string tokenization is commonly used to parse structured data formats such as CSV (comma separated values), TSV (Tab separated values), and custom data formats where data elements are separated by specific delimiters. It allows breaking down a string into meaningful components or tokens based on these delimiters.

(2.) Text Processing and Analysis :- In natural language processing and text analysis, tokenization is used to split a text into words or phrases (tokens). this is crucial for tasks such as text mining, sentiment analysis, and information retrieval where understanding the structure of text is essential.

(3) Lexical Analysis in Compilers:- Tokenization plays a fundamental role in compilers and interpreters for programming languages, Here, strings of source code are tokenized into tokens representing keywords, identifiers, operators and literals.

(4) Command Line Parsing:- When developing command-line interfaces (CLI) or parsing command line arguments in programs, tokenization helps split command string into individual arguments or options.

(5) Input Validation and Sanitization:- Tokenization can be used for input ~~for~~ validation by ensuring that input conforms to expected patterns or formats, it helps in sanitizing input data and preventing Injection attacks in web applications.

Example:- Consider a simple example where we tokenize a CSV string;

```
import java.util.StringTokenizer;
Public class TokenizationExample {
    public static void main(String[] args){
        String csvData = "John,Doe,30,New York";
        StringTokenizer tokenizer = new StringTokenizer(csvData,
                                                ",");
        while (tokenizer.hasMoreTokens()){
            System.out.println(tokenizer.nextToken());
        }
    }
}
```

(4.) Write about awt controls ~~in~~ with an example.

AWT (Abstract Window Toolkit) controls in java are a set of graphical user Interface (GUI) components provided by the "java.awt" package. These controls form the foundation for building GUI applications in Java, allowing developers to create windows, dialogs, buttons, text fields, checkboxes, radio buttons, lists, menus, and more. AWT controls are lightweight and platform-independent, making them suitable for basic GUI development in java.

Common AWT Controls :-

(1.) Frame :- A 'Frame' is a top-level window with a title and border. it is created using the 'Frame' class and serves as the main container for other AWT components.

(2.) Panel :- A 'Panel' is a container that can hold other AWT components. it is used to organize and group related GUI components.

(3.) Label :- A 'Lebel' is used to display text or an image on the GUI. it is created using the 'Label' class and typically information to the user.

(4.) Button :- A Button repereserts a push-button that triggers an action when clicked. It is created using the 'Button' class and can be used to perform task or submit forms.

(5.) TextField:- A TextField allows users to input a single line of text. It is created using the 'TextField' class and can be used to capture user input.

(6.) Checkbox:- A checkbox allows user to select one or more options from a list. it is created using the "Checkbox" class and is typically used for boolean options.

(7.) Choice:- A choice is a drop-down menu that allows users to select one option from a list of predefined choices. it is created using the 'choice' class and is useful for selecting among several options.

(8.) List:- A List presents a list of items that users can select from. it is created using the 'List' class and supports single or multiple selections.

Example:- Creating a simple GUI with AWT Controls:

```
import java.awt.*;
import java.awt.event.*;
public class AWTDemo extends Frame implements ActionListener{
    Private Label label;
    private TextField textField;
    private Button button;
    private Checkbox checkbox;

    public AWTDemo(){
        SetLayout(new FlowLayout());
        label = new Label("Enter your name:");
        add(label);
```

```
        textFeild = new TextField(20);
        add(textFeild);

        button = new Button("Submit");
        add(button);
        button.addActionListener(this);
        checkbox = new Checkbox("I agree to the terms and
                                  Conditions");
        add(checkbox);
        SetTitle("AWT Demo");
        SetSize(300,150);
        SetVisible(true);
    }
    Public void actionPerformed(AdionEvent e){
        if(e.getsource() == button){
            string name = textField.getText();
            boolean agreed = checkbox.getState();
            if(agreed){
                system.out.println("Hello, "+name+"! you
                    agreed to the terms.");
            }
            else{
                system.out.println("Hello, "+name+"!
                    please agree to the terms.");
            }
        }
    }
    public static void main(String[] args){
        new AWTDemo();
    }
}
```

(5.) Write about java network programming with an Example.

Java network programming allows developers to create applications that communicate over the network, using sokets. Sokets enables bidirectional communication between client and server applications, facilitating data exchange over TCP or UDP protocols.

Java Network Programming Concepts:

(1.) Socket Programing:-

(i) Socket – A socket is an endpoint for communication between two machines.

(ii) ServerSocket – waits for incoming client requests.

(iii) Client Socket – Initiates communication with the Server.

Java Provides classes like 'Socket' and 'serverSocket' in the 'java.net' package for Socket Programming.

(2.) TCP vs UDP:-

(i) TCP – (Transmission Control Protocol):- Provides reliable, ordered, and error-checked delivery of data between applications. Used for applications where accurate transmission is important (e.g., file transfer, email).

(ii) UDP – (User Datagram Protocol):- Connectionless Protocol that Sends data packets (datagrams) without checking whether they arrive or not. Used for applications that can tolerate some data loss (eg., Streaming media, online gaming).

(3.) Networking APIs in Java:-

(i) Socket API - ('Java.net.Socket', 'Java.net.ServerSocket'):- used for creating client-server applications over TCP/IP networks.

(ii) URL and URLConnection:- used for accessing resources via URLs.

(iii) DatagramPacket and DatagramSocket:- used for communi-cation via UDP.

(iv) InetAddress:- Represents an Ip Address.

(v) URLConnection:- Abstract class for representing a communication link between the application and URL resource.

Example:- Example of Java Network Programming:-

Server (EchoServer.java):-

```
import java.io.*;
import java.net.*;
public class EchoServer{
    public static void main(String[] args){
        try{
            ServerSocket serversocket = new ServerSocket(8888);
            System.out.println("Server Started. Waiting for a client...");
            Socket clientSocket = serversocket.accept();
            System.out.println("Client connected: "+clientSocket);
            BufferReader in = new BufferReader(new InputStreamReader
            (clientSocket.getInputStream()));
            PrintWriter out = new PrintWriter(clientSocket.getoutput
            Stream(), true);
            String message;
```

```
        while((message = in.readLine()) != null){
            System.out.println("Received from client: "+message);
            out.println("Server echoed: "+message);
        }
        in.close();
        out.close();
        clientSocket.close();
        serverSocket.close();
    }catch(IOException e){
        e.printStackTrace();
    }
}
}
```

Client (EchoClient.java):-

```
import java.io.*;
import java.net.*;
Public class Echoclient{
    public static void main(String[]args){
        try{
            Socket socket = new Socket("localhost",8888);
            BufferedReader in = new BufferedReader(new InputStream
            Reader(socket.getInputStream()));
            PrintWriter out = PrintWriter(socket.getoutputStream(),true);

            out.println("Hello from client!");
            string response = in.readLine();
            System.out.println("Server response: "+response);

            out.println("How are you?");
            response = in.readLine();
            System.out.println("Server response: "+response);

            in.close();
            out.close();
            Socket.close();
        }catch(IOException e){
            e.printStackTrace();
        }
    }
}
```