Object Oriented Programming using Java

Assignment - I

Q1. Write about benefits of OPPs?

object-oriented programming or OOPs refer to Languages that use objects in programming, they use objects as a primary source to implement what is to happen in the code. objects are seen by the ~~viver~~ viewer or user, performing tasks assigned by you. object-oriented programming aims to implement real world entities like inheritance, hiding, polymorphism etc. in programming.

Benefits of OOPs in java:-

(1.) Modularity:- OOPs organizes code into Smaller parts (objects), making it easier to understand and maintain.

(2.) Reusability:- you can reuse code through ~~interfa~~ inheritance (one class can inherit traits from another) and Composition (objects can be made of other objects).

(3.) Flexibility:- OOP allows for polymorphism, meaning you can use objects interchangeably, which makes your code more adaptable to changing requirements.

(4.) Easier Debugging:- Problems are easiar to isolate with in objects, and changes are less likely to impect other part of the Code.

(5.) Improved Organization:- OOPs organizes your Code into Logical, understanding able units (objects), making it easier to manage and update.

(6.) Better Problem Solving :- OOPs models real-world scenarios effectively, helping you solve problems more intuitively.

(7.) Scalability :- OOPs is suitable for both small and large Projects, allowing you to break down complex tasks into manageable pieces.

(8.) Security :- OOP's encapsulation protects data from being accessed or modified unintentionally.

(9.) Code Reusability :- Through inheritance and composition, you can reuse existing classes without modifying them, saving time and effort in coding.

(10) Abstraction :- OOPs allows you to focus on the essential features of an object while hiding unnecessary details. This simplifies complex system. and improves efficiency.

(11) Extensibility :- OOPs allows you to extend existing Software with ~~real~~ new features without changing existing code, through mechanisms such as subclassing and interfaces.

(2.) List out and Explain about charader and byte stream classes?

In java, stream are a fundamental concept for handling input and output (I/o) operations. streams can be broadly categoized into two types.

(1.) character stream

(2.) Byte stream.

Each type serves different purposes based on whether they deal with raw byte (byte stream) or character data (character streams).

(1.) Character Streams:- character sterams are designed to address character based records, which includes taxtual records inclusive of letters, digits, symboles, and other characters. These streams are represented by way of training that quit with the phrase "Reader" or "Writer" of their names, inclusive of File Reader, BufferReader, File Writer, and Bufferwriter.

List of Character Streams:-

(1.) Reader and Writer:-

(i) Reader — Abstract class for reading characters.

(ii) Writer — Abstract class for writing characters.

(2.) File Reader and File Writer:-

(i) FileReader — Reads character from a file.

(ii) FileWriter — Writes character to a file.

(3.) InputStreamReader and OutputStream~~P~~Writers:-

(i) InputStreamReader — Reads bytes and decode them into characters.

(ii) OutputStreamWriter — Encodes character into bytes for writing.

(4.) BufferedReader and BufferedWriter:-

(i) BufferedReader — Adds buffering to a 'Reader'

(ii) BufferedWriter — Adds buffering to a 'writer'.

(2) Byte Streams :- Byte stream are deal with raw binary data, which includes all kind of data, including characters, Pictures, audio and video. These streams are represented through classes that cease with the word 'InputStream' or 'OutputStream' of their names, along with FileInputStream, BufferOutputStream, FileOutputStream, BufferInputstream.

Byte Stream List :-

(1.) Inputstream and output stream:-

(i) InputStream - Abstract class for reading bytes.

(ii) OutputStream — Abstract class for writing bytes.

(2.) FileInputStream and FileoutputStream:-

(i) FileInputStream — Reads bytes from a file.

(ii) File output Stream — Writes bytes ~~from~~ a file.

(3.) ByteArrayInputStream and ByteArrayOutputStream:-

(i) ByteArrayInputStream — Reads bytes from an array.

(ii) ByteArrayOutputStream — Writes bytes to a byte array.

(4.) BufferInputstream and Buffered output stream:-

(i) BufferedInput Stream — Adds buffering to an input stream.

(ii) BufferedOutputStream — Adds buffering to an output stream.

(3) Describe collection classes along with a suitable example?

Collections class - in java is one of the utility classes in java Collections Framework. The java.util package contains the Collections class in java. Java Collections class is used with the static methods that operate on the collections or return the collection. All the methods of this class throw the NullPointerExpception if the Collection or object passed to the methods is null.

Collection Class declaration:-

public class Collections extends Object
- object is the parent class of all the classes.

Java Collection Class:- Collection Framework contains both both classes and interfaces. Although both seem the same but there are certain different between Collection classes and the collection framework. There are some classes in java as mentioned below:

(1.) ArrayList - Array List is a class implemented using a list interface, in that Provides the functionality of a dynamic array where the size of the array is not fixed.

Syntex -

ArrayList<_type_> var_name = new ArrayList<_type_>();

(2.) Vector :- Vector is a part of the collection class that implements a dynamic array that can grow or shrink its size as required.

Syntex –

Public class Vector<E> extends AbstractList<E> implements List<E>, RandomAccess, Cloneable, Serializable.

(3.) Stack :- Stack is a part of java Collection class that models and implements a stack data structure. It is based on the basic principle of last-in-first-out (LIFO).

Syntex –

public class Stack<E> extends Vector<E>

(4.) Linked List :- LinkedList class is implementation of the LinkedList data structure. It can store the elements that are not stored in contiguous locations and every element is a seprate object with a different data and different address part.

Syntex –

LinkedList name = new LinkedList();

(5) HashSet :- HashSet is implimenented using the Hashtable data structure. It offers constant time performance for the performing operations like add, remove, contains, and size.

Syntex –

Public class Hashset<E> extends AbstractSet<E> implements Set<E>, Cloneable, Serializable.

(6.) HashMap:- HashMap class is similar to HashTable but the data unsynchronized. it stores the data in(Key, Value) pairs, and you can access them by an index of another type.

Syntex -

Public class HashMap <K,V> extends AbstractMap<K,V> implements Map<K,V>, cloneable, serializable.

Example of HashMap:-

```
import java.util.HashMap

public class HashMapExample {
    public static void main(string[] args) {
        HashMap <Integer, string> studentMap = new HashMap<>();
        student.put(1, "Alice");
        student.put (2, "Bob");
        Student.put (3, "charlie");

        system.out.println("Student with ID 2: "+studentMap.get(2));

        system.out.println(" All Student : ");
        for (integer id: studentMap.keyset()) {
            string name = studentMap.get(id);
            system.out.println("ID: "+id+", Name: "+name);
        }
    }
}
```

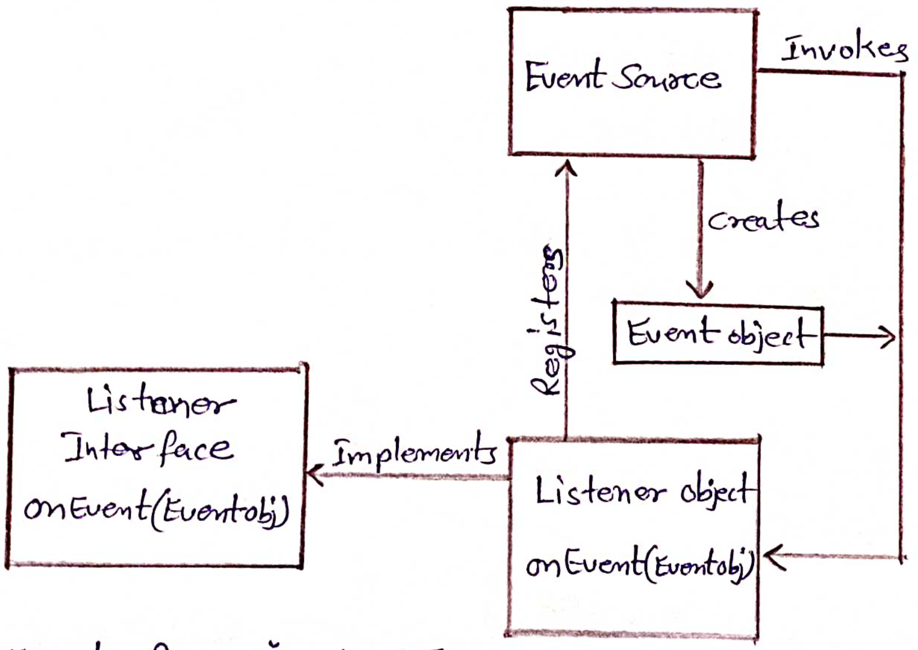
other Collection classes - (7) LinkedHashset (8) TreeSet

(9) PriorityQueue (10) ArrayDeque (11) EnumMap

(12) AbstractMap (13) TreeMap

(1.) List out event listener interfaces and write about event delegation model

Event Delegation Model :- The Event Delegation model in Java is a java is a design pattern used to handle events generated by user ~~inter~~ interactions with graphical user interface (GUI) components. it promote efficient event handling by delegating the ~~res~~ responsibility. of event management and processing to specialized event listeners rather than the components themselves. Here's how the Event Delegation Model works —



Event Processing in Java

In Java, event Listener iterfaces are used to handle events generated by user interactions or system operations. These interfaces follow the observer design pattern, where

Listeners (observers) register themselves to be notified of events and provide specific methods to handle those events. Here are some commonly used event listener interfaces in java:

(1.) Action Listener:- used for Handling action events, such as button click or menu selections.

method - "void actionPerformed(ActionEvent e)" invoked when an action occurs.

(2.) ItemListener:- used for Handling item events, such as selection changes in checkboxes or list items.

method - "void itemStateChanged (ItemEvent e)" invoked when an item's state changes.

(3.) MouseListener:- used for Handling mouse events, such as click, mouseover, mousemovement, or button presses.

methods - (i) void mouseClicked(MouseEvent e)
(ii) void mousePressed (MouseEvent e)
(iii) void mouseReleased (MouseEvent e)
(iv) void mouseEntered (MouseEvent e)
(v) void mouseExited (MouseEvent e)

(4.) keyListener:- Handles keyboard events, such as key press and releases.

methods (i) void keyPressed (keyEvent e)
(ii) void keyReleased (key Event e)
(iii) void keyTyped (keyEvent e)

(5.) Focus Listener:- Handle focus events, triggered when components gain or lose focus.

methods - (i) void focusGained (FocusEvent e)
(ii) void focusLost (FocusEvent e).

(5.) Explain process of reading and writing into files with an example.

In Java, reading from and writing to files involves Several steps that ensure proper handling of file resources. Here's detailed explaination along with an example for both reading and writing files:

<u>Reading from a file in Java</u>:-

(1.) <u>Opening the file</u>:- use the 'FileInputStream' or 'BufferedReader' classes to open the file for reading.

(2.) <u>Reading Data</u>:- use methods provided by 'FileInputStream' or 'BufferedReader' classes to read data from the file.

(3.) <u>Closing the file</u>:- close the file to release system resources.

<u>Example</u>:- Read the file 'Input.txt'

```
import java.io.BufferedReader;
import java.io.FileReader;
import Java.io. IOException;

Public class ReadFileExample {

        public static void main(string[] args){
          String filePath = "Input.text;

          try(BufferedReader reader = new BufferedReader(
                              new FileReader(filePath))){
               String Line;
        while((line = reader.readLine()) != null){
            System.out.println(line);
        }
    } catch (IOException e) {
        System.err.println("Error reading from file: "+
                              e.getMessage());
    }
  }
}
```

Explanation :- 'BufferedReader' is used for efficient reading of characters, and 'FileReader' is used to read byte from the file. The 'try-with-resources' statement ensure that the 'BufferedReader' is closed automatically after use. Inside the 'try' block, 'Reader.readline()' reads lines from the file untill the end ('null' is returned when the end of the file is reached.)

Writing to a File in Java :-

(1.) Opening the file :- use 'FileOutput Stream' or 'BufferedWriter' classes to open the file for writing.

(2.) Writing Data :- use methods provided by 'FileOutputStream' or 'BufferedWriter' to write data to the file.

(3.) Closing the file :- close the file to insure all data is flushed and resources are released.

Example :- Here's an Example that writes to a file named 'output.txt':

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import Java.io.IoException;

public class WriterFileExample {
        public static void main (String[] args){
            String filePath = "output.txt";
            String data = " Hello, World\nThis is a new Line.";

            try (BufferedWritter writer = new BufferedWriter(new
                                    FileWriter(filePath))){
                writer.write(data);
                System.out.println("Data has been written to"+filePath);
            } catch(IOException e) {
                System.err.println (" Error writing to file:"+
                                        e.getMessage());
            }
        }
}
```

Explaination:- 'BufferedWriter' is used for efficient writing of characters, and 'FileWriter' is used to write bytes to the file. The 'try-with-resources' statement ensures that the 'BufferedWriter' is closed automatically after use. Inside the 'try' block, 'writer.write(data)' writes the string 'data' to the file.

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 poits "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 cheks 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 multithreeding programming and explain how synchronization is achieved.

Multithreding 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 leverraging mordern multicore Processors effectively.

<u>Thread States</u>:- Threads in java can be in different states throughout their lifecycle:

- <u>New</u>:- when a thread instance is created but not yet started.
- <u>Runnable</u>:- when the thread is ready to run and waiting for CPU time(either running or waiting for execution).
- <u>Blocked/Waiting</u>:- 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()').
- <u>Timed Waiting</u>:- 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++) {
                                increment Counter();
                        }
                });
        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 ~~as~~ 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 ~~t~~mining, sentiment analysis, and information retrieval where understanding the structure of ~~the~~ text is essential.

(3.) Lexical Analysis in Compilers:- Tokenization plays a fundamental role in compilers and iterpreters for programming languages, Here, strings of source code are tokenized into tokens repre-senting 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 ~~is~~ 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 from 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 repeseresents 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 opations.

(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(ActionEvent 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 (e.g., 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) Inet Address :- 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~~ string message;
        while((message = in.readLine()) != null){
              System.out.println("Recevied 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();
          }
      }
}
```