# 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 ~~viwer~~ 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 easier to isolate with in objects, and changes are less likely to impact 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 character and byte stream classes?

In java, stream are a fundamental concept for handling input and output (I/o) operations. streams can be broadly categorized 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 stereams 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 FileReader, 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.) FileReader and FileWriter:-
   (i) FileReader — Reads character from a file.
   (ii) FileWriter — Writes character to a file.

(3.) InputStreamReader and OutputStreamWriter:-

(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 OutputStream — Writes bytes to 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) Buffered OutputStream — 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);
        }
    }
}
```

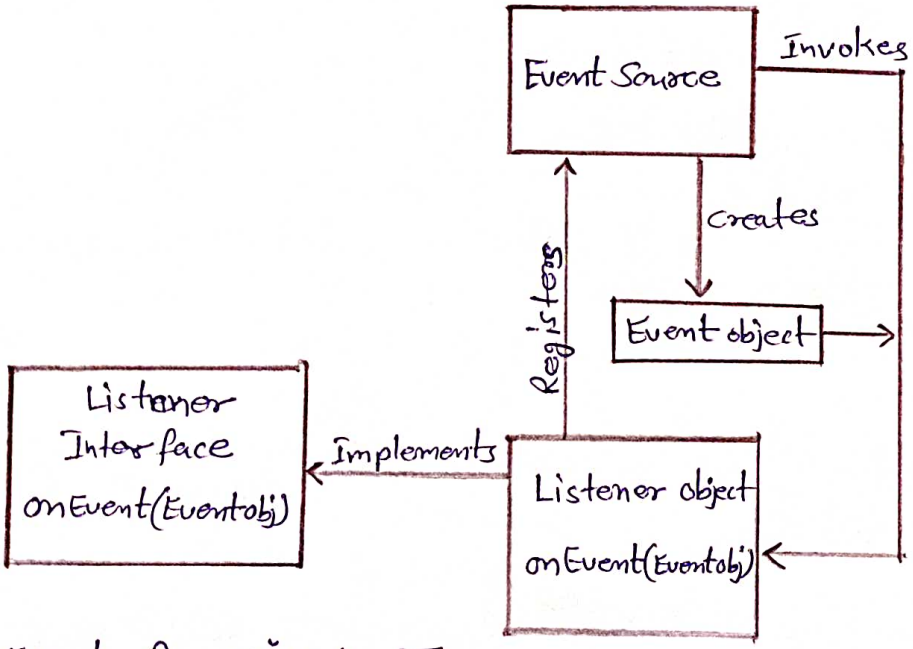
Other Collection classes — ⑦ LinkedHashSet ⑧ TreeSet

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

(12) AbstractMap (13) TreeMap

(4.) 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 interactions with graphical user interface (GUI) components. it promote efficient event handling by delegating the 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 interfaces 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.) ActionListener:- 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(KeyEvent e)
(iii) void keyTyped(KeyEvent e)

(5.) FocusListener:- 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 explanation along with an example for both reading and writing files:

Reading from a file in Java:-

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

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

(3.) Closing the file:- close the file to release system resources.

Example:- 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()' reds 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 'FileOutputStream' 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 (BufferedWriter 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());
        }
    }
}
```

Explanation:- '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.