Q. 1 Explain major features of Object Oriented programming.

Answer:

FEATURES OF OOP:

1. Object
2. Class
3. Data Hiding and Encapsulation
4. Dynamic Binding
5. Message Passing
6. Inheritance
7. Polymorphism

OBJECT: Object is a collection of number of entities. Objects take up space in the memory. Objects are instances of classes. When a program is executed, the objects interact by sending messages to one another. Each object contain data and code to manipulate the data. Objects can interact without having know details of each others data or code.

CLASS: Class is a collection of objects of similar type. Objects are variables of the type class. Once a class has been defined, we can create any number of objects belonging to that class.

Eg: grapes bananas and orange are the member of class fruit.

Example:

Fruit orange;

In the above statement object mango is created which belong to the class fruit.

NOTE: Classes are user define data types.

DATA ABSTRACTION AND ENCAPSULATION:

Combining data and functions into a single unit called class and the process is known as Encapsulation. Data encapsulation is important feature of a class. Class contains both data and functions. Data is not accessible from the outside world and only those function which are present in the class can access the data. The insulation of the data from direct access by the program is called data hiding or information hiding. Hiding the complexity of program is called Abstraction and only essential features are represented. In short we can say that internal working is hidden.

DYNAMIC BINDING: Refers to linking of function call with function definition is called binding and when it is take place at run time called dynamic binding.

MESSAGE PASSING: The process by which one object can interact with other object is called message passing.

INHERITANCE: it is the process by which object of one class aquire the properties or features of objects of another class. The concept of inheritance provide the idea of reusability means we can add additional features to an existing class without Modifying it. This is possible by
driving a new class from the existing one. The new class will have the combined features of both the classes.

Example: Robine is a part of the class flying bird which is again a part of the class bird.

POLYMORPHISM: A greek term means ability to take more than one form. An operation may exhibit different behaviours in different instances. The behaviour depends upon the types of data used in the operation.

Example:
- Operator Overloading
- Function Overloading

Q.2 What is the significance of data types in a programming language? Discuss various data types in java.

Answer: In simple language, if variables are container then data type is the type of container. Type of container actually tells that what kind of stuff can be put in it. For example you don't want to put cookies in a bottle similarly you don't want to store an integer value in a variable of data type String.

There are eight primitive data types supported by Java. Primitive data types are predefined by the language and named by a keyword. Let us now look into detail about the eight primitive data types.

byte:
- Byte data type is an 8-bit signed two's complement integer.
- Minimum value is -128 (-2^7)
- Maximum value is 127 (inclusive) (2^7 -1)
- Default value is 0
- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an int.
- Example: byte a = 100 , byte b = -50

short:
- Short data type is a 16-bit signed two's complement integer.
- Minimum value is -32,768 (-2^15)
- Maximum value is 32,767 (inclusive) (2^15 -1)
- Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an int.
- Default value is 0.
- Example: short s = 10000, short r = -20000

int:
- Int data type is a 32-bit signed two's complement integer.
- Minimum value is -2,147,483,648.(-2^31)
- Maximum value is 2,147,483,647(inclusive).(2^31 -1)
- Int is generally used as the default data type for integral values unless there is a concern about memory.
- The default value is 0.
- Example: int a = 100000, int b = -200000
long:
- Long data type is a 64-bit signed two's complement integer.
- Minimum value is -9,223,372,036,854,775,808. (-2^63)
- Maximum value is 9,223,372,036,854,775,807 (inclusive). (2^63 -1)
- This type is used when a wider range than int is needed.
- Default value is 0L.
- Example: long a = 100000L, int b = -200000L

float:
- Float data type is a single-precision 32-bit IEEE 754 floating point.
- Float is mainly used to save memory in large arrays of floating point numbers.
- Default value is 0.0f.
- Float data type is never used for precise values such as currency.
- Example: float f1 = 234.5f

double:
- double data type is a double-precision 64-bit IEEE 754 floating point.
- This data type is generally used as the default data type for decimal values, generally the default choice.
- Double data type should never be used for precise values such as currency.
- Default value is 0.0d.
- Example: double d1 = 123.4

boolean:
- boolean data type represents one bit of information.
- There are only two possible values: true and false.
- This data type is used for simple flags that track true/false conditions.
- Default value is false.
- Example: boolean one = true

char:
- char data type is a single 16-bit Unicode character.
- Minimum value is \u0000 (or 0).
- Maximum value is \uffff (or 65,535 inclusive).
- Char data type is used to store any character.
- Example: char letterA = 'A'

Q.3 Strings are immutable. Explain the statement.
Answer: String is immutable means that you cannot change the object itself, but you can change the reference to the object. When you called a = "ty", you are actually changing the reference of a to a new object created by the String literal "ty". Changing an object means to use its methods to change one of its fields (or the fields are public and not final, so that they can be updated from outside without accessing them via methods), for example:
Foo x = new Foo("the field");
x.setField("a new field");
System.out.println(x.getField()); // prints "a new field"
While in an immutable class (declared as final, to prevent modification via inheritance)(its methods cannot modify its fields, and also the fields are always private and recommended
to be final), for example String, you cannot change the current String but you can return a new String, i.e:
String s = "some text";
s.substring(0,4);
System.out.println(s); // still printing "some text"
String a = s.substring(0,4);
System.out.println(a); // prints "some"

**Q.4 What are collections in Java? When they are used?**

**Answer:** A *collection* — sometimes called a container — is simply an object that groups multiple elements into a single unit. Collections are used to store, retrieve, manipulate, and communicate aggregate data. Typically, they represent data items that form a natural group, such as a poker hand (a collection of cards), a mail folder (a collection of letters), or a telephone directory (a mapping of names to phone numbers). If you have used the Java programming language — or just about any other programming language — you are already familiar with collections.

The Java Collections Framework provides the following benefits:

- Reduces programming effort:
- Increases program speed and quality:
- Allows interoperability among unrelated APIs:
- Reduces effort to learn and to use new APIs:
- Reduces effort to design new APIs:
- Fosters software reuse:

**Part-2 Attempt any three.**

**Q.1 What is UML? Mention the different kinds of modelling diagrams used.**

**Answer:** UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. UML was created by Object Management Group and UML 1.0 specification draft was proposed to the OMG in January 1997. Differe

**Q.2 What are exceptions? When they are used? Explain with a code snippet?**

**Answer:** An *exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions. When an error occurs within a method, the method creates an object and hands it off to the runtime system. The object, called an *exception object*, contains information about the error, including its type and the state of the program when the error occurred. Creating an exception object and handing it to the runtime system is called *throwing an exception*. 
After a method throws an exception, the runtime system attempts to find something to handle it. The set of possible "somethings" to handle the exception is the ordered list of methods that had been called to get to the method where the error occurred.

```java
private List<Integer> list;
private static final int SIZE = 10;

public void writeList() {
    PrintWriter out = null;
    try {
        System.out.println("Entered try statement");
        out = new PrintWriter(new FileWriter("OutFile.txt"));
        for (int i = 0; i < SIZE; i++) {
            out.println("Value at: " + i + " = " + list.get(i));
        }
    }
    catch and finally blocks . . .
}
```

**Q.3 What is an interface in Java? Discuss with an example.**

**Answer:** An interface is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

An interface is not a class. Writing an interface is similar to writing a class, but they are two different concepts. A class describes the attributes and behaviors of an object. An interface contains behaviors that a class implements.

Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

**Example:**

```java
// File name : NameOfInterface.java
import java.lang.*;
//Any number of import statements

public interface NameOfInterface {
    //Any number of final, static fields
    //Any number of abstract method declarations
}
```

**Q.4 Explain operator overloading with suitable examples.**

**Answer:** C++ allows you to specify more than one definition for a function name or an operator in the same scope, which is called function overloading and operator overloading respectively.

An overloaded declaration is a declaration that had been declared with the same name as a previously declared declaration in the same scope, except that both declarations have different arguments and obviously different definition (implementation).
When you call an overloaded function or operator, the compiler determines the most appropriate definition to use by comparing the argument types you used to call the function or operator with the parameter types specified in the definitions. The process of selecting the most appropriate overloaded function or operator is called overload resolution.

Example:-
#include <iostream>
using namespace std;

class printData {
    public:
    void print(int i) {
        cout << "Printing int: " << i << endl;
    }

    void print(double f) {
        cout << "Printing float: " << f << endl;
    }

    void print(char* c) {
        cout << "Printing character: " << c << endl;
    }
};

int main(void) {
    printData pd;

    // Call print to print integer
    pd.print(5);
    // Call print to print float
    pd.print(500.263);
    // Call print to print character
    pd.print("Hello C++");

    return 0;
}

Part-3 Attempt any three.

Q.1 Explain abstract method and abstract class. Write a program for employee class where salary as an abstract method with full implementation.
Answer:- An abstract class is a class that is declared abstract—it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed.
An abstract method is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:
abstract void moveTo(double deltaX, double deltaY);
If a class includes abstract methods, then the class itself must be declared abstract, as in:
public abstract class GraphicObject {
    // declare fields
    // declare nonabstract methods
    abstract void draw();
}
When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, then the subclass must also be declared abstract.

Q.2 What is an applet? Explain lifecycle of an applet with method declaration. Write a java program which shows a simple Applet.
Answer:- An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.
There are some important differences between an applet and a standalone Java application, including the following:
- An applet is a Java class that extends the java.applet.Applet class.
- A main() method is not invoked on an applet, and an applet class will not define main().
- Applets are designed to be embedded within an HTML page.
- When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.
- A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
- The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
- Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.
- Other classes that the applet needs can be downloaded in a single Java Archive (JAR) file.

Life Cycle of an Applet:
Four methods in the Applet class give you the framework on which you build any serious applet:
- 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.
- start: This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.
stop: This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.

destroy: This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.

paint: Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

A "Hello, World" Applet:
The following is a simple applet named HelloWorldApplet.java:
import java.applet.*;
import java.awt.*;

class HelloWorldApplet extends Applet {
    public void paint(Graphics g) {
        g.drawString("Hello World", 25, 50);
    }
}

Q.3 Explain multithreading in Java. What is the effect in program when we use multithreading in a program. Write a java program for z=sin(x) +cos(x) using multithreading.

Answer:- Java is a *multithreaded programming language* which means we can develop multithreaded program using Java. A multithreaded program contains two or more parts that can run concurrently and each part can handle different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

By definition multitasking is when multiple processes share common processing resources such as a CPU. Multithreading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application.

Multithreading enables you to write in a way where multiple activities can proceed concurrently in the same program.

```java
import java.lang.Math;
class MathSin extends Thread {
p    public double deg;
p    public double res;
p    public MathSin(int degree) {
p        deg = degree;
    }
```
public void run() {
    System.out.println("Executing sin of "+deg);
    double Deg2Rad = Math.toRadians(deg);
    res = Math.sin(Deg2Rad);
    System.out.println("Exit from MathSin. Res = "+res);
}
}
class MathCos extends Thread {
    public double deg;
    public double res;
    public MathCos(int degree) {
        deg = degree;
    }
    public void run() {
        System.out.println("Executing cos of "+deg);
        double Deg2Rad = Math.toRadians(deg);
        res = Math.cos(Deg2Rad);
        System.out.println("Exit from MathCos. Res = "+res);
    }
}

class MathThreads {
    public static void main(String args[]) {
        MathSin st = new MathSin(45);
        MathCos ct = new MathCos(60);
        st.start();
        ct.start();
        try {// wait for completion of all thread and then sum
            st.join();
            ct.join();//wait for completion of MathCos object
            double z = st.res + ct.res;
            System.out.println("Sum of sin, cos = "+z);
        } catch(InterruptedException IntExp) {
        }
    }
}

Q.4 Design a class using Java to create a singly linked list. Then also write methods for
adding a node to the linked list in the beginning and also search a given node.

public class Node<T>{
    Node<T> prev;
    Node<T> next;
    T element;
    public Node() {
    }
public class SinglyLinkedList<T> {
    private Node<T> head;
    private Node<T> tail;
    private Node<T> temp;
    private int size;
    public SinglyLinkedList() {
        head = null;
        tail = null;
        size = 0;
    }
    public void addLast(T elem) {
        if(head == null) {
            head = new Node<T>();
            head.element = elem;
            tail = head;
            size++;
        } else {
            temp = new Node<T>();
            temp.element = elem;
            tail.next = temp;
            temp.prev = tail;
            tail = temp;
            size++;
        }
    }
    void print(){
        temp = head;
        while(temp != null){
            System.out.println(temp.element);
            temp = temp.next;
        }
    }
    public static void main(String[] args) {
        SinglyLinkedList<Integer> list = new SinglyLinkedList<>();
        list.addLast(1);
        list.addLast(2);
        list.addLast(3);
        list.addLast(4);
        list.addLast(5);
        list.addLast(6);
        list.print();
    }
}