Java Assignment:

1.What do you understand by OOP?

Ans:- oop or Object-Oriented Programming, is a programming paradigm based on the concept of "objects," which can contain data in the form of fields (often known as attributes or properties) and code in the form of procedures (often known as methods). Objects are instances of classes, which act as blueprints for creating objects.

2. What are the features of OOP?

- **Abstraction:** Abstraction in Java is the process in which we only show essential details/functionality to the user. The non-essential implementation details are not displayed to the user. In this article, we will learn about abstraction and what abstract means.
- **Encapsulation :** In Java, encapsulation is achieved by declaring the instance variables of a class as private, which means they can only be accessed within the class. To allow outside access to the instance variables, public methods called getters and setters are defined, which are used to retrieve and modify the values of the instance variables, respectively.

- **Inheritance:** Inheritance is one of the key features of OOP that allows us to create a new class from an existing class. The new class that is created is known as subclass (child or derived class) and the existing class from where the child class is derived is known as superclass (parent or base class). The extends keyword is used to perform inheritance in Java.

- **Polymorphism:** Polymorphism is considered one of the important features of Object-Oriented Programming. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word “poly” means many and “morphs” means forms, So it means many forms.

3. Write down the advantages and disadvantages of OOP.

Advantages of OOP:

Modularity: Objects can be created independently and then combined to create complex systems, promoting code reusability and simplifying maintenance.

Reusability: Through inheritance, classes can inherit attributes and methods from other classes, allowing for the reuse of code and promoting a more efficient development process.

Flexibility: Polymorphism allows objects of different classes to be treated as objects of a common superclass, enabling more flexible and adaptable code.

Easier maintenance: Encapsulation allows for the separation of internal implementation details from the external interface, making it easier to modify and maintain code.

Scalability: OOP makes it easier to scale applications by breaking them down into smaller, manageable objects.

Disadvantages of OOP:

Complexity: OOP can lead to complex designs, especially for beginners, due to the need to understand concepts such as inheritance, polymorphism, and encapsulation.

Performance Overhead: OOP can sometimes lead to slower execution compared to procedural programming, as there is overhead associated with objects and method calls.

Learning Curve: OOP requires a different way of thinking compared to procedural programming, which can make it challenging for some programmers to learn.

Overhead: OOP can lead to a larger memory footprint and slower performance compared to procedural programming, especially for small, simple programs.

4. What are the differences between OOP and structural programming?

Structural Programming	*Oop Programming*
1.It is a subset of Procedural Programming.	1.It relies on concept of objects that contain data and code.
2.Programs are divided into small programs or functions.	2.programs are divided into objects or entities.
3.It is all about facilitating creation of programs with readable code and reusable components.	3.It is all about creating objects that usually contain both functions and data.
4.Its main aim is to improve and increase quality, clarity and development time of computer program.	4.Its main aim is to improve and increase both quality and productivity of system analysis and design.
5.It simply focuses on functions and processes that usually work on data.	5.It simply focuses on representing both structure and behavior of information system into tiny or small modules that generally combines data and process both.

5. Why JAVA is not a purely object-oriented programming language?

Java is often considered a "pure" object-oriented programming language, but it is not purely object-oriented due to the following reasons:

Primitive Data Types: Java includes primitive data types (e.g., int, double, boolean) that are not objects. These types are used to represent simple values and are not part of any class hierarchy.

Static Methods: Java allows the use of static methods, which belong to the class rather than any specific object instance. Static methods can be called without creating an object of the class, which is not purely object-oriented.

Static Variables: Similar to static methods, Java allows the use of static variables, which are shared across all instances of a class and are not tied to any specific object instance.

Final Classes and Methods: Java includes final classes and methods, which cannot be subclassed or overridden, respectively. This restricts the full flexibility of object-oriented principles like inheritance and polymorphism.

6. What is the use of 'static' keyword?

The static keyword in Java is used for memory management mainly. We can apply static keyword with variables, methods, blocks and nested classes . The static keyword belongs to the class than an instance of the class. If you declare any variable as static, it is known as a static variable.

7. **What do you mean by wrapper class?**

A Wrapper class in Java is a class whose object wraps or contains primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store primitive data types. In other words, we can wrap a primitive value into a wrapper class object.

8. Explain initialization of a JAVA object.

Memory Allocation: Memory is allocated for the object on the heap. The amount of memory required is determined by the object's class definition and any additional memory needed for its instance variables.

Setting Default Values: All instance variables of the object are set to their default values. For example, numeric types are set to 0, boolean types are set to false, and reference types (objects) are set to null.

Executing Constructor: The constructor of the object's class is called to initialize the object. The constructor can initialize the object's instance variables and perform any other initialization tasks.

Initializing Instance Variables: If the constructor assigns specific values to the object's instance variables, those values are set during initialization.

Returning Reference: Finally, the new expression returns a reference to the newly created object. This reference can be stored in a variable and used to access the object's methods and fields.

9. What are the different types of inheritance?

In object-oriented programming, inheritance is a mechanism where a new class derives attributes and methods from an existing class. There are several types of inheritance:

Single Inheritance: A class is derived from only one base class.

Multiple Inheritance: A class is derived from more than one base class. This is supported in languages like C++.

Multilevel Inheritance: A class is derived from a derived class, making a chain of inheritance.

Hierarchical Inheritance: Multiple derived classes are created from a single base.

Hybrid (Virtual) Inheritance: Combination of two or more types of inheritance.

Multipath Inheritance: When a class is derived from two classes, both of which are derived from the same base class.

Cyclic Inheritance: A situation where a class is a base class of itself, directly or indirectly.

10. Why interface is required?

Abstraction: Interfaces allow you to define a contract for a set of methods without providing an implementation. This allows you to abstract the behavior of a class from its implementation, which is a fundamental principle of object-oriented programming.

Multiple Inheritance: Java classes can only inherit from one superclass, but they can implement multiple interfaces. This allows you to incorporate behaviors from multiple sources without the complexities of multiple inheritance seen in some other languages.

Polymorphism: Interfaces allow different classes to be treated as instances of the same interface type. This enables polymorphism, where you can write code that works with objects of different classes as long as they implement the same interface.

Code Reusability: Interfaces promote code reusability by allowing different classes to implement the same interface. This means that you can write methods that accept interfaces as parameters, making your code more flexible and easier to maintain.

Contractual Obligations: When a class implements an interface, it agrees to fulfill the contract defined by that interface. This ensures that classes implementing the same interface provide a consistent set of methods, which is useful for ensuring interoperability and code correctness.