Assignment-17

Java Oops

1. **What is inheritance in Java ?**
   - ➔ **Inheritance is a process where a child class acquires all the properties and behaviours of the parent class .**

2. **What is Super-Class and Sub-Class ?**
   - → **A class from where a subclass inherits features is called superclass.**
   - → **It is also called base class or parent class.**

   - ➔ **A class that inherits all the members (fields, method, and nested classes) from another class is called a subclass.**
   - ➔ **It is also called a derived class, child class, or extended class.**

3. **How is Inheritance implemented/achieved in Java?**

   → **Inheritance can be implemented or achieved by using two keywords: extends: extends is a keyword that is used for developing the inheritance between two classes and two interfaces. implements: implements keyword is used for developing the inheritance between a class and interface.**

4. **What is polymorphism?**
   - ➔ **Polymorphism in OOP is the ability of an entity to take several forms. In other words, it refers to the ability of an object (or a reference to an object) to take different forms of objects. It allows a common data-gathering message to be sent to each class. Polymorphism encourages what is called 'extendibility' which means an object or a class can have its uses extended**

5. **Differentiate between method overloading and method overriding ?**

- **Method Overloading → Method overloading involves defining multiple methods in a class with the same name but with different parameters (either different types or different numbers of parameters).**
- **Overloaded methods must have the same name but may have different return types.**
- **Overloaded methods are resolved at compile time based on the number and types of arguments passed during method invocation.**

- **Method Overriding → Method overriding occurs when a subclass provides a specific implementation of a method that is already defined in its superclass.**

- **The method signature (name, parameters, and return type) of the overriding method in the subclass must match exactly with the method in the superclass.**
- **Method overriding is used to achieve runtime polymorphism, where the actual method invoked is determined by the type of the object at runtime.**
- **Method overriding is applicable only to inherited methods, i.e., the method must be defined in both the superclass and subclass.**
- **The @Override annotation (in Java) is used to indicate that a method is intended to override a method in the superclass, helping to catch errors if the method signature does not match exactly.**

Aspect	**Method Overloading**	**Method Overriding**
Definition	Multiple methods with the same name	Subclass provides specific implementation of a method
Inheritance	Not related to inheritance	Related to inheritance, occurs between superclass-subclass
Signature	Different method signatures	Method signatures must match exactly
Resolving	Compile-time resolution	Runtime resolution
Return type	May or may not have the same return type	Must have the same return type
Annotations	No specific annotation needed	**@Override** annotation can be used (in Java)
Usage	Providing multiple behaviors for a method	Changing or extending behavior of inherited method

6. What is an abstraction explained with an Example?

➔ **Abstraction is nothing but the quality of dealing with ideas rather than events. It basically deals with hiding the internal details and showing the essential things to the user.**

➔ **Abstract class Sports { // abstract class sports Abstract void jump(); // abstract method }**

7. What is the difference between an abstract method and final method in Java ? Explain with an example .

→ The abstract method is incomplete while the final method is regarded as complete. The only way to use an abstract method is by overriding it, but you cannot override a final method in Java.

8. What is the final class in Java?

➔ A class declared with the final keyword is known as the final class. A final class can't be inherited by subclasses. By using the final class, we can restrict the inheritance of the class. We can create a class as a final class only if it is complete in nature, which means it must not be an abstract class. In java, all the wrapper classes are final classes like String, Integer, etc. If we try to inherit a final class, then the compiler throws an error at compilation time. We can't create a class as immutable without the final class.

```
          final class ParentClass

{    void

 showData()     {

     System.out.println("This is a method of final Parent class");

   }

} //It will throw compilation error class ChildClass extends ParentClass

{     void showData()      {

    System.out.println("This is a method of Child class");

      }

} class MainClass {

 public static void main(String arg[])      {

   ParentClass obj = new ChildClass();       obj.showData();

   }

}
```

9. Differentiate between abstraction and encapsulation.

- Abstraction→ Definition: Abstraction is the process of simplifying complex systems by representing only the essential features and hiding unnecessary details.
- Focus: Abstraction focuses on what an object does rather than how it does it.
- Example: For example, consider a car. From an abstract perspective, we might think of a car as an object that can be driven, refueled, and parked. We don't need to know the intricate details of how the engine works or how the transmission system operates to understand these high-level functionalities.

- Encapsulation→ Definition: Encapsulation is the bundling of data (attributes) and methods (behaviors) that operate on the data into a single unit, called a class.
- Focus: Encapsulation focuses on hiding the internal state of an object and exposing only the necessary operations to interact with that state.

9. Difference between Runtime and compile time polymorphism explain with an example.

Runtime Polymorphism→
Definition: Compile-time polymorphism, also known as static polymorphism, refers to the polymorphic behavior that is resolved during compile time.

- **Mechanism: In compile-time polymorphism, the method or function overloading is used to achieve polymorphic behavior. The decision about which method to call is made by the compiler based on the number and types of arguments provided at the call site.**

Compile time Polymorphism→

- **Definition: Runtime polymorphism, also known as dynamic polymorphism, refers to the polymorphic behavior that is resolved during runtime.**
- **Mechanism: In runtime polymorphism, method overriding is used to achieve polymorphic behavior. The decision about which method to call is made by the JVM (Java Virtual Machine) based on the actual type of the object at runtime.**
- **Example:**