Oops
Assignment

Q1. What is Inheritance in Java?

Ans. Java, Inheritance is an important pillar of OOP(Object-Oriented Programming). It is the mechanism in Java by which one class is allowed to inherit the features(fields and methods) of another class.

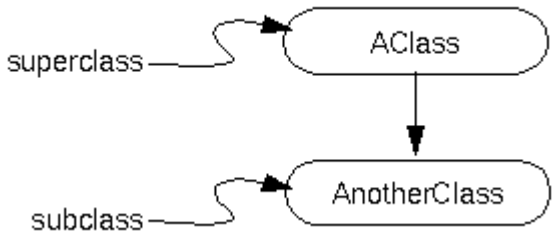
In Java, Inheritance means creating new classes based on existing ones. A class that inherits from another class can reuse the methods and fields of that class. In addition, you can add new fields and methods to your current class as well.

Q2. What is superclass and subclass in Java?

Ans. In Java, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:

- subclass (child) - the class that inherits from another class
- superclass (parent) - the class being inherited from

To inherit from a class, use the `extends` keyword.



Q3. How is inheritance implemented /achieved in Java?

Ans. The extends keyword is used for inheritance in Java. Using the extends keyword indicates you are derived from an existing class. In other words, “extends” refers to increased functionality.

Syntax :

```
class DerivedClass extends BaseClass
{
  //methods and fields
}
```

Q4. What is polymorphism?

Ans. 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.

Q5. Differentiate between method overloading and overriding?

Ans.

Method Overloading	Method Overriding
Method overloading is a compile-time polymorphism.	Method overriding is a run-time polymorphism.
Method overloading helps to increase the readability of the program.	Method overriding is used to grant the specific implementation of the method which is already provided by its parent class or superclass.
It occurs within the class.	It is performed in two classes with inheritance relationships.
Method overloading may or may not require inheritance.	Method overriding always needs inheritance.

In method overloading, methods must have the same name and different signatures.	In method overriding, methods must have the same name and same signature.
In method overloading, the return type can or can not be the same, but we just have to change the parameter.	In method overriding, the return type must be the same or co-variant.
Static binding is being used for overloaded methods.	Dynamic binding is being used for overriding methods.
Private and final methods can be overloaded.	Private and final methods can't be overridden.
The argument list should be different while doing method overloading.	The argument list should be the same in method overriding.

Q6. What is abstraction explained with an example?

Ans. In Java, abstraction is achieved by interfaces and abstract classes. We can achieve 100% abstraction using interfaces.

Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details. The properties and behaviours of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

1. An abstract class is a class that is declared with an abstract keyword.
2. An abstract method is a method that is declared without implementation.
3. An abstract class may or may not have all abstract methods. Some of them can be concrete methods
4. A method-defined abstract must always be redefined in the subclass, thus making overriding compulsory or making the subclass itself abstract.
5. Any class that contains one or more abstract methods must also be declared with an abstract keyword.
6. There can be no object of an abstract class. That is, an abstract class can not be directly instantiated with the *new operator*.
7. An abstract class can have parameterized constructors and the default constructor is always present in an abstract class.

Example-:

```
abstract class Animal {
    public abstract void animalSound();
    public void sleep() {
    System.out.println("Zzz");
```

```
  }
}

class Pig extends Animal {
  public void animalSound() {
        System.out.println("The pig says: wee wee");
  }
}

class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig();
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Q7. What is the difference between an abstract method and final method in java?Explain with an Example?

Ans.

S.No.	ABSTRACT METHOD	FINAL METHOD
1.	Uses the “abstract”	Uses the “final” keyword.

	keyword.	
2.	This helps to achieve abstraction.	This helps to restrict other classes from accessing its properties and methods.
3.	For later use, all the abstract methods should be overridden	Overriding concept does not arise as final class cannot be inherited
4.	A few methods can be implemented and a few cannot	All methods should have implementation
5.	Cannot create immutable objects (infact, no objects can be created)	Immutable objects can be created (eg. String class)

6.	Abstract class methods functionality can be altered in subclass	Final class methods should be used as it is by other classes
7.	Can be inherited	Cannot be inherited
8.	Cannot be instantiated	Can be instantiated
9.	Abstract class may have final methods.	Final class does not have abstract methods or final methods.
10.	Abstract class helps in to achieve Abstraction.	Final class can help to restrict the other classes from accessing the properties and methods.

Example - : final method -

```
1. class Bike{
2.   final void run(){System.out.println("running");}
```

```
3. }
4.
5. class Honda extends Bike{
6.   void run(){System.out.println("running safely with 100kmph");}
7.
8.   public static void main(String args[]){
9.   Honda honda= new Honda();
10.  honda.run();
11.  }
12.}
```

abstract method -:

```
abstract class A {
        abstract void m1();
 void m2()
    {
        System.out.println("This is "
                           + "a concrete method.");
    }
}

class B extends A {
       void m1()
    {
        System.out.println("B's "
                           + "implementation of m1.");
    }
}
public class AbstractDemo {
   public static void main(String args[])
    {
        B b = new B();
```

```
        b.m1();
        b.m2();
    }
}
```

Q8. What is the final class in Java?

Ans. The final method in Java is used as a non-access modifier applicable only to a variable, a method, or a class. It is used to restrict a user in Java.

Final classes: When a class is declared as final, it cannot be extended by a subclass. This is useful for classes that are intended to be used as is and should not be modified or extended.

Example

```
1. final class Bike{}
2. class Honda1 extends Bike{
3.   void run(){System.out.println("running safely with 100kmph");
4. }
5.   public static void main(String args[]){
6.   Honda1 honda= new Honda1();
7.   honda.run();
8.   }
9. }
```

Q9. Differentiate between abstraction and encapsulation.

Ans.

Abstraction	Encapsulation

Abstraction is process of hiding the implementation details and showing only the functionality to the users.	Encapsulation is a process of binding data and methods together in a single unit, providing controlled access to data.
Main feature: reduce complexity, promote maintainability, and also provide clear separation between the interface and its concrete implementation.	Main feature: data hiding, providing access control and modularity.
In abstraction, problems are solved at the design or interface level.	While in encapsulation, problems are solved at the implementation level.
We can implement abstraction using abstract class and interfaces.	Whereas encapsulation can be implemented using by access modifier i.e. private, protected and public and nested classes.

In abstraction, implementation complexities are hidden using abstract classes and interfaces.	While encapsulation uses private access modifier to hide the data and use getter and setter to provide controlled access to data.
The objects that help to perform abstraction are encapsulated.	Whereas the objects that result in encapsulation need not be abstracted.
Abstraction provides access to specific part of data.	Encapsulation hides data, preventing the users from directly accessing it, (providing controlled access) which is also known as data hiding.
Abstraction focuses on “what” the object does .	Encapsulation focuses on “How” the object does it.
Example: CAR – the driver of the car only needs to know how to drive it. Not how its engine and the gear box and other internal components work.	Example: A bank can be thought of as a fully encapsulated class that provides access to the customers through various methods (getters and setters). Rest of the data

	inside bank is hidden to protect from outside world.

Q10. Difference between Runtime and compile time polymorphism explained with an example.

Ans.

Compile Time Polymorphism	Run time Polymorphism
In Compile time Polymorphism, the call is resolved by the compiler.	In Run time Polymorphism, the call is not resolved by the compiler.
It is also known as Static binding, Early binding and overloading as well.	It is also known as Dynamic binding, Late binding and overriding as well.
It is achieved by method overloading	It is achieved by virtual functions and pointers.
It provides fast execution because the method that needs	It provides slow execution as compare to early binding because

to be executed is known early at the compile time.	the method that needs to be executed is known at the runtime.
Compile time polymorphism is less flexible as all things execute at compile time.	Run time polymorphism is more flexible as all things execute at run time.
Inheritance is not involved.	Inheritance is involved.

Example-: runtime-:

```
class Cde {

        public void method()
    {
         System.out.println("Method 1");
    }
}

public class GFG extends Test {

         public void method()
    {
         System.out.println("Method 2");
    }

    public static void main(String args[])
    {
         Test test = new GFG();

         test.method();
    }
}
```

Compile time-:

```java
public class Abc {


        public static int add(int a, int b)
    {
         return a + b;
    }


    public static double add(
         double a, double b)
    {
         return a + b;
    }
    public static void main(String args[])
    {
              System.out.println(add(2, 3));
             System.out.println(add(2.0, 3.0));
    }
}
```