1. **What is inheritance in java?**

**Ans:** 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. For example,

class Animal {

// methods and fields

}

// use of extends keyword

// to perform inheritance

class Dog extends Animal {

// methods and fields of Animal

// methods and fields of Dog

}

In the above example, the Dog class is created by inheriting the methods and fields from the Animal class. Here, Dog is the subclass and Animal is the superclass.

1. **What is superclass and subclass?**

**Ans:** In Java, a subclass is a class that is derived from another class (superclass). The subclass inherits all the members (fields, methods, and nested classes) from its superclass. The subclass can use all the public and protected members of the superclass. It can also have its own fields and methods in addition to the inherited ones.

Here’s an example:

class Animal {

public void eat() {

System.out.println("I can eat");

}

}

// inherit from Animal class

class Dog extends Animal {

// new method in subclass

public void display() {

System.out.println("My name is " + name);

}

}

class Main {

public static void main(String [] args) {

// create an object of the subclass

Dog labrador = new Dog ();

// access field of superclass

labrador.name = "Rohu";

labrador.display (); // call method of subclass

labrador.eat (); // call method of superclass using object of subclass

}

}

Here, Animal is the superclass and Dog is the subclass. The Dog class inherits the eat() method from the Animal class.

1. **How is inheritance implemented/achieved in Java?**

**Ans:** In Java, inheritance is implemented using the extends keyword. The extends keyword is used to create a subclass that inherits the properties of a superclass. The subclass can then add its own properties and methods in addition to the inherited ones.

Here’s an example:

class Animal {

public void eat() {

System.out.println("I can eat");

}

}

// inherit from Animal class

class Dog extends Animal {

// new method in subclass

public void display() {

System.out.println("My name is " + name);

}

}

class Main {

public static void main(String [] args) {

// create an object of the subclass

Dog labrador = new Dog ();

// access field of superclass

labrador.name = "Rohu";

labrador.display (); // call method of subclass

labrador.eat (); // call method of superclass using object of subclass

}

}

Here, Animal is the superclass and Dog is the subclass. The Dog class inherits the eat() method from the Animal class.

1. **What is polymorphism?**

**Ans:** Polymorphism is one of the four fundamental concepts of object-oriented programming (OOP). It allows objects of different classes to be treated as if they were objects of the same class. In Java, polymorphism is achieved through method overriding and method overloading.

Method overriding is when a subclass provides its own implementation of a method that is already provided by its parent class. Here’s an example:

class Animal {

public void makeSound() {

System.out.println("The animal makes a sound");

}

}

class Dog extends Animal {

public void makeSound() {

System.out.println("The dog barks");

}

}

class Cat extends Animal {

public void makeSound() {

System.out.println("The cat meows");

}

}

class Main {

public static void main(String[] args) {

Animal myAnimal = new Animal();

Animal myDog = new Dog();

Animal myCat = new Cat();

myAnimal.makeSound();

myDog.makeSound();

myCat.makeSound();

}

}

Here, Animal is the superclass and Dog and Cat are the subclasses. The Dog class overrides the makeSound() method of the Animal class to provide its own implementation.

1. **Difference between method overloading and overriding.**

**Ans:** Method overloading and method overriding are two concepts in Java that are used to achieve polymorphism.

Method overloading is when a class has two or more methods with the same name but different parameters. The methods can have different return types, but the parameter list must be different. Method overloading is used to provide different ways of calling the same method.

Here’s an example:

class Calculator {

public int add(int x, int y) {

return x + y;

}

public int add(int x, int y, int z) {

return x + y + z;

}

}

class Main {

public static void main(String[] args) {

Calculator myCalculator = new Calculator();

System.out.println(myCalculator.add(10, 20));

System.out.println(myCalculator.add(10, 20, 30));

}

}

Here, the Calculator class has two methods named add(), but they take different parameters.

Method overriding is when a subclass provides its own implementation of a method that is already provided by its parent class. The method signature (name and parameter list) must be the same in both the superclass and the subclass. Method overriding is used to provide a specific implementation of a method in a subclass.

Here’s an example:

class Animal {

public void makeSound() {

System.out.println("The animal makes a sound");

}

}

class Dog extends Animal {

public void makeSound() {

System.out.println("The dog barks");

}

}

class Cat extends Animal {

public void makeSound() {

System.out.println("The cat meows");

}

}

class Main {

public static void main(String[] args) {

Animal myAnimal = new Animal();

Animal myDog = new Dog();

Animal myCat = new Cat();

myAnimal.makeSound();

myDog.makeSound();

myCat.makeSound();

}

}

Here, Animal is the superclass and Dog and Cat are the subclasses. The Dog class overrides the makeSound() method of the Animal class to provide its own implementation.

1. **What is an abstraction explained with an example?**

**Ans:** Abstraction is one of the four fundamental concepts of object-oriented programming (OOP). It is the process of hiding certain details and showing only essential information to the user. In Java, abstraction can be achieved with either abstract classes or interfaces.

An abstract class is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class). It can have both abstract and regular methods. An abstract method is a method that does not have a body and can only be used in an abstract class. The body is provided by the subclass (inherited from).

Here’s an 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();

}

}

Here, Animal is an abstract class that has an abstract method named animalSound(). The Pig class extends the Animal class and provides its own implementation of the animalSound() method.

1. **What is the difference between an abstract method and final method in java? Explain with an example.**

**Ans:** An abstract method is a method that is declared but does not have an implementation. It must be overridden by the subclass that extends the abstract class. An abstract method is declared using the abstract keyword.

Here’s an example:

abstract class Animal {

public abstract void animalSound();

}

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();

}

}

Here, Animal is an abstract class that has an abstract method named animalSound(). The Pig class extends the Animal class and provides its own implementation of the animalSound() method.

A final method is a method that cannot be overridden by any subclass. It is declared using the final keyword.

Here’s an example:

class Animal {

public final void animalSound() {

System.out.println("The animal makes a sound");

}

}

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();

}

}

Here, Animal is a class that has a final method named animalSound(). The Pig class extends the Animal class but cannot override the animalSound() method because it is declared as final.

1. **What is the final class in java?**

**Ans:** A final class in Java is a class that cannot be extended by any subclass. It is declared using the final keyword.

Here’s an example:

final class Animal {

// class members

}

class Dog extends Animal {

// compile-time error

}

Here, Animal is a final class that cannot be extended by any subclass. The Dog class extends the Animal class but cannot do so because it is declared as final.

1. **Differentiate between abstraction and encapsulation?**

**Ans:** Abstraction and encapsulation are two fundamental concepts of object-oriented programming (OOP).

Abstraction is the process of hiding certain details and showing only essential information to the user. It can be achieved with either abstract classes or interfaces.

Encapsulation is the process of hiding the internal details of an object from the outside world. It is achieved by declaring the variables of a class as private and providing public getter and setter methods to access and modify the values of those variables.

Here’s an example:

class Person {

private String name;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

class Main {

public static void main(String[] args) {

Person person = new Person();

person.setName("John");

System.out.println(person.getName());

}

}

Here, Person is a class that has a private variable named name. The getName() and setName() methods are public methods that can be used to access and modify the value of the name variable.

1. **Difference between Runtime and Complile time polymorphism explain with an example.**

**Ans:** Polymorphism is the ability of an object to take on many forms. In Java, polymorphism can be achieved through method overloading and method overriding.

Method overloading is an example of compile-time polymorphism. It is when a class has two or more methods with the same name but different parameters.

Method overriding is an example of runtime polymorphism. It is when a subclass provides its own implementation of a method that is already provided by its parent class. The method signature (name and parameter list) must be the same in both the superclass and the subclass.

Here’s an example:

class Animal {

public void makeSound() {

System.out.println("The animal makes a sound");

}

}

class Dog extends Animal {

public void makeSound() {

System.out.println("The dog barks");

}

}

class Main {

public static void main(String[] args) {

Animal myAnimal = new Animal();

Animal myDog = new Dog();

myAnimal.makeSound();

myDog.makeSound();

}

}

Here, Animal is the superclass and Dog is the subclass. The Dog class overrides the makeSound() method of the Animal class to provide its own implementation.