1. **What is Inheritance in Java?**

**Ans.** Inheritance is a fundamental concept in object-oriented programming (OOP) that allows a class to inherit properties and behaviors from another class.

1. **What is super-class and subclass?**

**Ans.** The superclass is the existing class from which properties and behaviors are inherited, while the subclass is the new class that inherits those properties and behaviors.

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

**Ans.** Inheritance is achieved by using the **extends** keyword to establish a relationship between a superclass and a subclass.

1. **What is polymorphism?**

**Ans.** If one thing exist in more than one form then it is called polymorphism.

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

**Ans.** Method overloading refers to defining multiple methods in the same class with the same name but different parameters.

Method overriding refers to redefining a method in a subclass that is already defined in its superclass, with the same name, return type, and parameters.

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

**Ans.** Abstraction fundamental involves simplifying complex systems or ideas by focusing on essential characteristics while hiding unnecessary details.

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

**Ans.**

1. Abstract Method: An abstract method is a method that is declared in an abstract class or an interface but does not have an implementation. It serves as a placeholder for the method that must be implemented in any concrete subclass or class implementing the interface. Abstract methods are meant to be overridden by the subclasses, providing their own implementation.

Example:

abstract class Animal {

abstract void makeSound(); // Abstract method declaration

void sleep() {

System.out.println("Zzzz");

}

}

class Cat extends Animal {

void makeSound() { // Implementing the abstract method

System.out.println("Meow");

}

}

public class Main {

public static void main(String[] args) {

Cat cat = new Cat();

cat.makeSound(); // Output: Meow

cat.sleep(); // Output: Zzzz

}

}

1. Final Method: A final method is a method that cannot be overridden or modified by any subclass. When a method is declared as final, its implementation in the superclass is considered the final and definitive version. This prevents any subclass from changing the behaviour of the method.

Example:

class Parent {

final void display() { // Final method declaration

System.out.println("Parent's display method");

}

}

class Child extends Parent {

// Trying to override the final method, which is not allowed

void display() {

System.out.println("Child's display method");

}

}

public class Main {

public static void main(String[] args) {

Child child = new Child();

child.display(); // Output: Parent's display method

}

}

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

**Ans.** A final class is a class that cannot be subclassed or extended by any other class. When a class is declared as final, it means that it cannot be used as a superclass for inheritance.

In a final class, all its methods are automatically considered final as well.

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

**Ans.**

|  |  |  |
| --- | --- | --- |
|  | **Abstraction** | **Encapsulation** |
| **Definition** | Representing essential features and behaviour while hiding unnecessary details. | Bundling data and methods together, protecting the internal state of an object. |
| **Focus** | What the object does and what properties it has. | How the object's data and behaviour are organized and accessed. |
| **Purpose** | Simplify complexity, provide a high-level view, and define contracts. | Ensure data integrity, control access, and hide implementation details. |
| **Achieved through** | Abstract classes and interfaces. | Access modifiers (public, private, protected) and class boundaries. |
| **Key concept** | Defining the interface and behaviour of an object. | Data hiding and controlling access to the object's internal state. |
| **Relationship to other classes** | Defines relationships between classes, often using inheritance or interfaces. | Protects the internal state from direct access by other classes. |
| **Visibility** | Exposes necessary information and functionality to the outside world. | Limits access to internal members and provides controlled access through methods. |

1. **Differentiate between Runtime and compile time polymorphism explain with an example.**

**Ans.**

1. Compile-Time Polymorphism (Static Polymorphism): Compile-time polymorphism is determined during the compilation phase of a program. It involves method overloading, where multiple methods with the same name but different parameters or types are defined within a class. The appropriate method to invoke is resolved by the compiler based on the method signature at compile-time.

Example:

class Calculator {

int add(int num1, int num2) {

return num1 + num2;

}

double add(double num1, double num2) {

return num1 + num2;

}

}

public class Main {

public static void main(String[] args) {

Calculator calculator = new Calculator();

int sum1 = calculator.add(2, 3); // Invokes the int add method

double sum2=calculator.add(2.5, 3.5);// Invokes the double add method

System.out.println("Sum 1: " + sum1); // Output: Sum 1: 5

System.out.println("Sum 2: " + sum2); // Output: Sum 2: 6.0

}

}

1. Runtime Polymorphism (Dynamic Polymorphism): Runtime polymorphism occurs during the execution of a program. It involves method overriding, where a subclass provides its own implementation for a method defined in its superclass. The method to be executed is determined dynamically at runtime based on the actual object type.

Example:

class Animal {

void makeSound() {

System.out.println("Animal makes a sound");

}

}

class Cat extends Animal {

void makeSound() {

System.out.println("Cat says meow");

}

}

public class Main {

public static void main(String[] args) {

Animal animal = new Animal();

Animal cat = new Cat();

animal.makeSound(); // Output: Animal makes a sound

cat.makeSound(); // Output: Cat says meow

}

}