Polymorphism is a fundamental concept in object-oriented programming that allows objects of different classes to be treated as objects of a common superclass during runtime. It provides a way to write flexible and reusable code by enabling a single interface to represent multiple forms (types) of objects.

There are two main types of polymorphism in Java:

Compile-time Polymorphism (Static Polymorphism):

Compile-time polymorphism is achieved through method overloading. Method overloading allows a class to have multiple methods with the same name but different parameter lists (number or type of parameters). The appropriate method to be called is determined at compile-time based on the method's signature.

Example:

java

Copy code

public class MathOperations {

public int add(int a, int b) {

return a + b;

}

public double add(double a, double b) {

return a + b;

}

}

In this example, the MathOperations class has two methods named add, one for adding integers and another for adding doubles. The appropriate method will be called based on the argument types passed during compilation.

Run-time Polymorphism (Dynamic Polymorphism):

Run-time polymorphism is achieved through method overriding and inheritance. Method overriding allows a subclass to provide a specific implementation for a method that is already defined in its superclass. When a method is called on a reference of the superclass but points to an object of the subclass, the method of the subclass is executed at runtime.

Example:

java

Copy code

class Animal {

public void makeSound() {

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

}

}

class Cat extends Animal {

@Override

public void makeSound() {

System.out.println("Meow!");

}

}

public class Main {

public static void main(String[] args) {

Animal animal = new Cat(); // Polymorphic behavior

animal.makeSound(); // Output: "Meow!"

}

}

In this example, we have a superclass Animal with a method makeSound(). The subclass Cat overrides the makeSound() method. During runtime, when makeSound() is called on a reference of type Animal but pointing to a Cat object, the overridden method in Cat is executed, demonstrating dynamic polymorphism.

Polymorphism is a powerful feature that promotes code reuse, flexibility, and extensibility in object-oriented programming. By writing code that depends on interfaces and abstractions rather than concrete implementations, you can design more maintainable and scalable applications.