Explain polymorphism in JAVA

Polymorphism is a fundamental concept in Object-Oriented Programming (OOP) that allows objects to be treated as instances of their parent class, enabling a single interface to represent different underlying forms (data types). In Java, polymorphism comes in two main types: compile-time (also known as static) and runtime (also known as dynamic).

1. Compile-Time Polymorphism (Static Polymorphism)

Compile-time polymorphism is achieved through method overloading and operator overloading (though Java does not support operator overloading). It allows multiple methods in the same class to have the same name but different parameters (different type or number of parameters).

Example of Method Overloading:

```
class MathOperations {
   // Overloaded methods
   int add(int a, int b) {
      return a + b;
   }
```

```
    double add(double a, double b) {
        return a + b;
    }

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

public class Main {
    public static void main(String[] args) {
        MathOperations math = new MathOperations();
        System.out.println(math.add(5, 10));          // Calls add(int, int)
        System.out.println(math.add(5.5, 10.5));      // Calls add(double,
double)
        System.out.println(math.add(5, 10, 15));      // Calls add(int, int,
int)
    }
}
```

In this example, the add method is overloaded with different parameter types and counts.

2. Runtime Polymorphism (Dynamic Polymorphism)

Runtime polymorphism is achieved through method overriding. This occurs when a subclass provides a specific implementation of a method that is already defined in its superclass. The method to be executed is determined at runtime based on the object being referenced.

Example of Method Overriding:

```
class Animal {
   void sound() {
      System.out.println("Animal makes a sound");
   }
}

class Dog extends Animal {
   @Override
   void sound() {
      System.out.println("Dog barks");
   }
}

class Cat extends Animal {
   @Override
   void sound() {
        System.out.println("Cat meows");
    }
}


public class Main {
    public static void main(String[] args) {
        Animal myAnimal;


        myAnimal = new Dog();
        myAnimal.sound();  // Output: Dog barks


        myAnimal = new Cat();
        myAnimal.sound();  // Output: Cat meows
    }
}
```

In this example, the sound method is overridden in the Dog and Cat classes. The method that gets executed is determined at runtime based on the actual object type (Dog or Cat), even though the reference type is Animal.

Benefits of Polymorphism

Code Reusability: You can write methods that work on the parent class type but can handle all derived class types.

Flexibility and Maintainability: Polymorphism makes it easier to add new classes and methods without altering existing code, promoting the Open/Closed Principle.

Dynamic Method Resolution: Allows for more dynamic and flexible code, as the method that gets called is determined at runtime.