Polymorphism is one of the core concepts in object-oriented programming (OOP) and plays a significant role in Java. It allows you to write code that can work with objects of different classes in a consistent way. Polymorphism is achieved through method overriding and interface implementation. There are two main types of polymorphism in Java:

1. **Compile-time Polymorphism (Static Binding)**:
   - Also known as early binding or method overloading.
   - Occurs at compile time.
   - Different methods with the same name are invoked based on the number or type of arguments.

   Example of compile-time polymorphism:

   ```java
   class Calculator {
       int add(int a, int b) {
           return a + b;
       }

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

   public class Main {
       public static void main(String[] args) {
           Calculator calculator = new Calculator();
           int result1 = calculator.add(5, 10);         // Calls the int version of add
           double result2 = calculator.add(3.5, 2.7);   // Calls the double version of add
       }
   }
   ```

2. **Run-time Polymorphism (Dynamic Binding)**:
   - Also known as late binding or method overriding.
   - Occurs at runtime.
   - The method to be invoked is determined at runtime based on the actual object's type.

   Example of run-time polymorphism:

   ```java
   class Animal {
       void makeSound() {
           System.out.println("Some sound");
```

```
    }
}

class Dog extends Animal {
    void makeSound() {
        System.out.println("Bark");
    }
}

class Cat extends Animal {
    void makeSound() {
        System.out.println("Meow");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal animal1 = new Dog();
        Animal animal2 = new Cat();

        animal1.makeSound(); // Calls Dog's makeSound() method
        animal2.makeSound(); // Calls Cat's makeSound() method
    }
}
```

In this example, the `makeSound` method is overridden in both the `Dog` and `Cat` classes. When we create objects of these classes and assign them to the `Animal` reference variables, the appropriate `makeSound` method is called based on the actual object's type at runtime.

Polymorphism enables you to write more flexible and maintainable code because you can treat objects of different classes in a uniform way, allowing for code reuse and extensibility. It's a fundamental concept in Java and OOP in general.