public class Person {

    private final String firstName;

    private final String lastName;

    private final int age;

    private final String address;

    private Person(Builder builder) {

        this.firstName = builder.firstName;

        this.lastName = builder.lastName;

        this.age = builder.age;

        this.address = builder.address;

    }

    public static class Builder {

        private String firstName;

        private String lastName;

        private int age;

        private String address;

        public Builder firstName(String firstName) {

            this.firstName = firstName;

            return this;

        }

        public Builder lastName(String lastName) {

            this.lastName = lastName;

            return this;

        }

        public Builder age(int age) {

            this.age = age;

            return this;

        }

        public Builder address(String address) {

            this.address = address;

            return this;

        }

        public Person build() {

            return new Person(this);

        }

    }

    @Override

    public String toString() {

        return "Person{" +

                "firstName='" + firstName + '\'' +

                ", lastName='" + lastName + '\'' +

                ", age=" + age +

                ", address='" + address + '\'' +

                '}';

    }

}

**Use Case**

Imagine you need to create a Person object with various attributes. Using the Builder Pattern, you can construct the object step-by-step, making the code more readable and maintainable.

public class Main {

    public static void main(String[] args) {

        Person person = new Person.Builder()

            .firstName("John")

            .lastName("Doe")

            .age(30)

            .address("123 Main St")

            .build();

        System.out.println(person);

    }

}

While the Builder Pattern offers many advantages, such as improved readability and flexibility, it also has some drawbacks:

1. **Increased Complexity**: The Builder Pattern introduces additional classes (e.g., the builder class) and methods, which can make the codebase more complex
2. **Overhead**: For simple objects or situations where a straightforward constructor would suffice, using the Builder Pattern can be overkill and add unnecessary overhead
3. **Maintenance**: Maintaining the builder class alongside the main class can be cumbersome, especially if the class structure changes frequently
4. **Learning Curve**: For developers unfamiliar with the pattern, there can be a learning curve to understand and implement it correctly
5. **Verbose Code**: The pattern can lead to more verbose code, as it requires defining multiple methods for setting each attribute
6. **Potential Misuse**: If not used appropriately, the Builder Pattern can clutter the code and make it harder to read and maintain

Despite these drawbacks, the Builder Pattern is very useful for constructing complex objects and is widely used in scenarios where immutability and readability are important.