Builder Pattern in Java - Documentation

1. Overview

The Builder Pattern is a creational design pattern used to construct complex objects with many optional and required fields. It helps to create objects step-by-step and provides better readability, maintainability, and flexibility.

2. Structure

Target Class:

```java
public class User {
private final String firstName;
private final String lastName;
private final int age;
private final String phone;
private final String address;

private User(UserBuilder builder) {
this.firstName = builder.firstName;
this.lastName = builder.lastName;
this.age = builder.age;
this.phone = builder.phone;
this.address = builder.address;
}
```

```
public static class UserBuilder {
private final String firstName;
private final String lastName;
private int age;
private String phone;
private String address;

public UserBuilder(String firstName, String lastName) {
this.firstName = firstName;
this.lastName = lastName;
}

public UserBuilder age(int age) {
this.age = age;
return this;
}

public UserBuilder phone(String phone) {
this.phone = phone;
return this;
}

public UserBuilder address(String address) {
this.address = address;
return this;
}
```

```
public User build() {
return new User(this);
}
}

@Override
public String toString() {
return "User: " + this.firstName + " " + this.lastName +
", Age: " + this.age +
", Phone: " + this.phone +
", Address: " + this.address;
}
}
```

Usage Example:

```java
public class BuilderDemo {
public static void main(String[] args) {
User user = new User.UserBuilder("John", "Doe")
.age(30)
.phone("1234567890")
.address("New York")
.build();

System.out.println(user);
```

```
}

}
```

3. Boundaries and Best Practices

- Immutability: Keep the target class immutable.
- Validation: Perform field validation inside the build() method.
- Fluent Interface: Ensure each builder method returns this.
- Avoid Overuse: Use when object construction is complex.

4. Common Interview Questions (with Answers)

Q1: What is the Builder Pattern and why is it used?

A1: It's used to construct complex objects step-by-step. It helps in avoiding large constructors with many parameters and improves code readability.

Q2: Difference between Builder and Factory Patterns?

A2: Factory Pattern creates objects without specifying the exact class. Builder Pattern is used to build a complex object step-by-step.

Q3: Advantages of the Builder Pattern?

A3: Immutability, better readability, easier to maintain, no telescoping constructors, and object creation with optional parameters.

Q4: Why not use telescoping constructors?

A4: They become hard to read, maintain, and error-prone when many parameters are involved.

Q5: Can the builder class be non-static?

A5: Technically yes, but it defeats the purpose as it would require an instance of the enclosing class.

Q6: Is Builder Pattern only for immutability?

A6: No, it's mainly for object construction. Immutability is a commonly associated benefit.

Q7: Builder vs Prototype Pattern?

A7: Builder builds a fresh object step-by-step. Prototype clones an existing object.

5. Summary

The Builder Pattern is a powerful solution for managing complex object creation. It makes the code more readable and maintainable by separating the construction logic from the object representation.