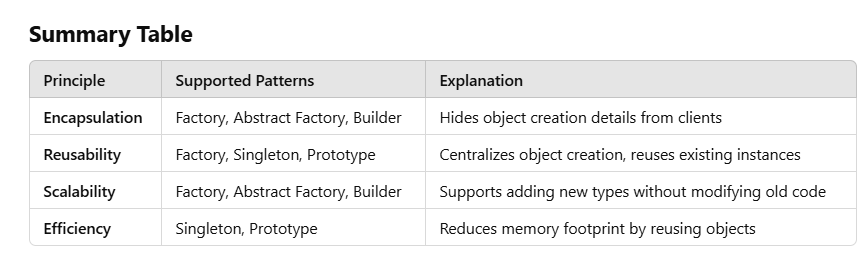
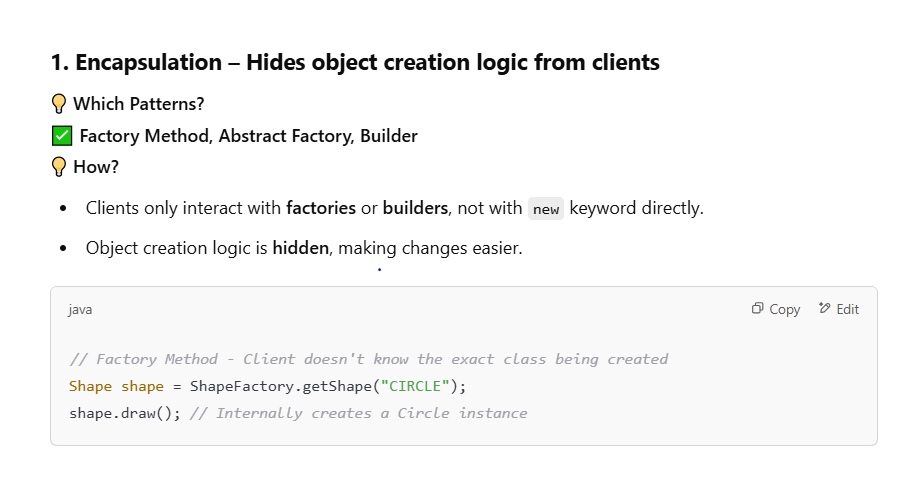
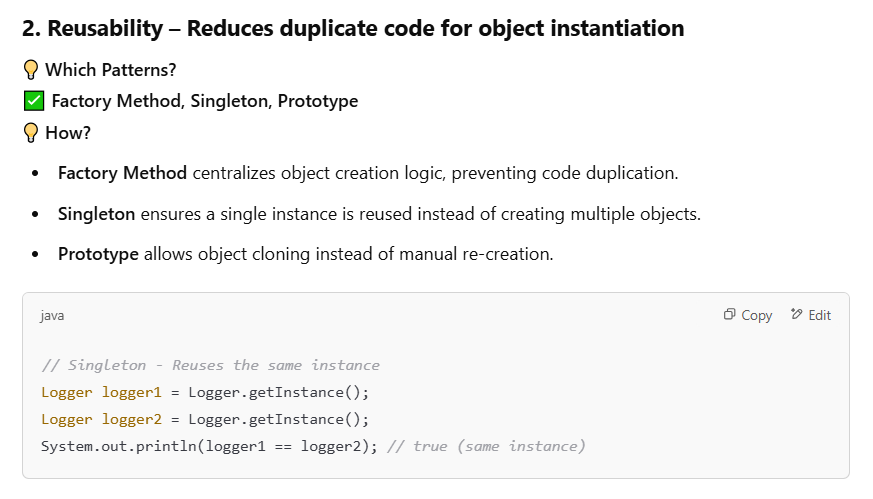
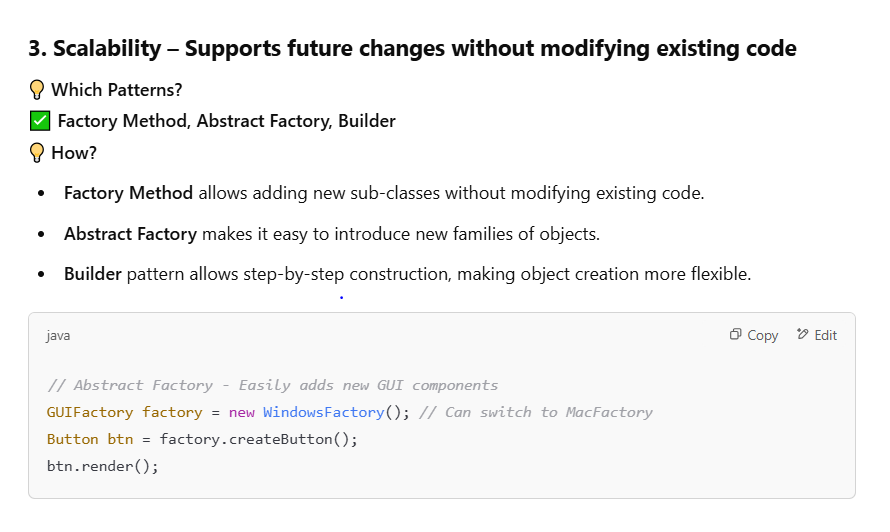
### ****Creational Design Pattern in Java****

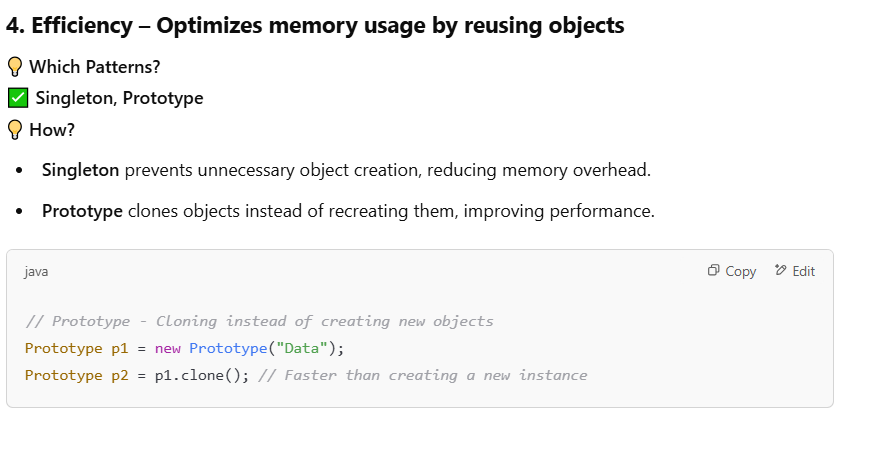
**Creational Design Patterns** focus on how objects are created in a flexible and reusable way, improving code maintainability and scalability.

****

****

****

****

****

**1:- Singleton Pattern :-**

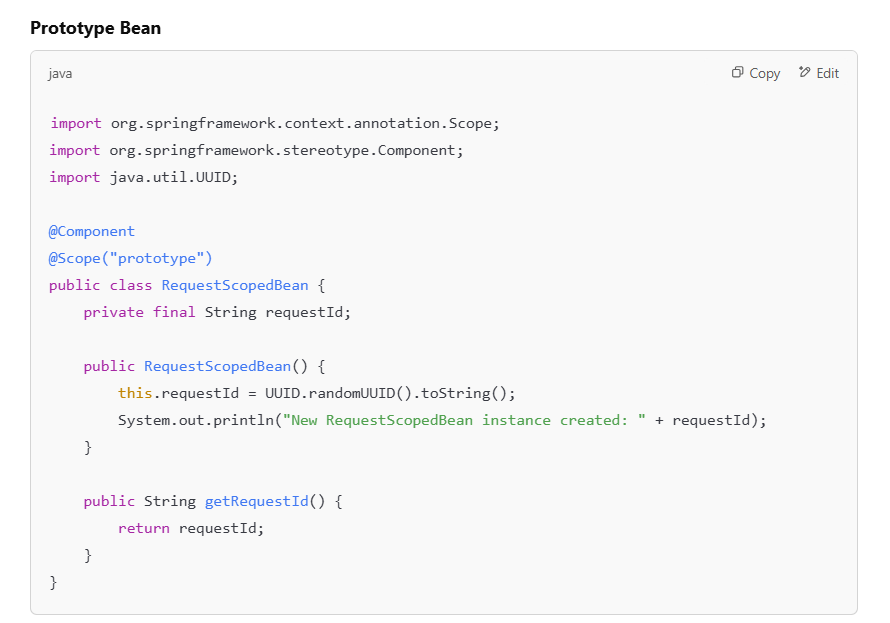
The **Singleton pattern** ensures that only one instance of a class is created and provides a global access point to that instance.

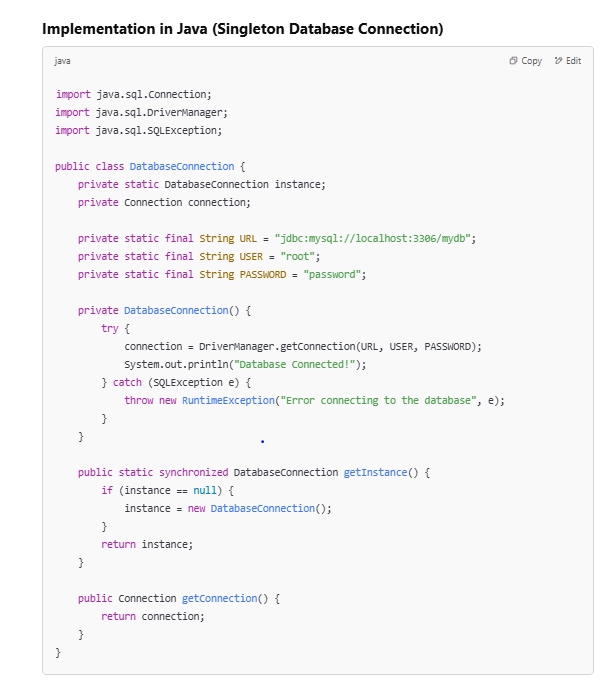
**Use Cases of Singleton in Java & Spring Boot**

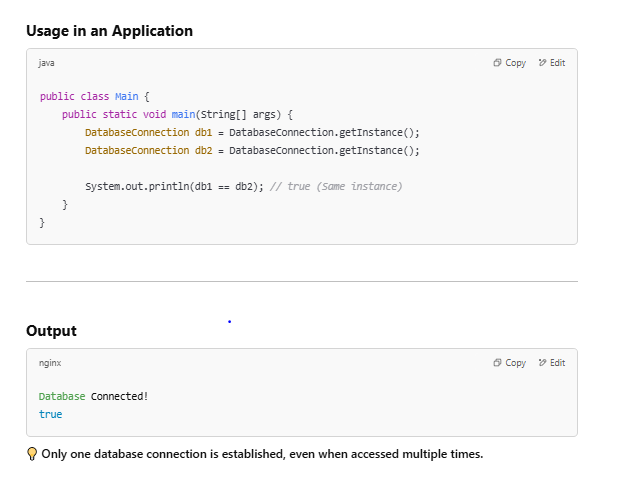
1. **Database Connection Pool** – A single instance manages multiple database connections efficiently.
2. **Caching** – A singleton class can store frequently used data to improve performance.
3. **Logging** – A single logger instance ensures consistent logging throughout the application.
4. **Configuration Management** – A single instance manages application-wide configurations.
5. **Spring Beans** – By default, Spring beans are Singleton-scoped.

### ****Example 1: Using Prototype for Generating Unique IDs****

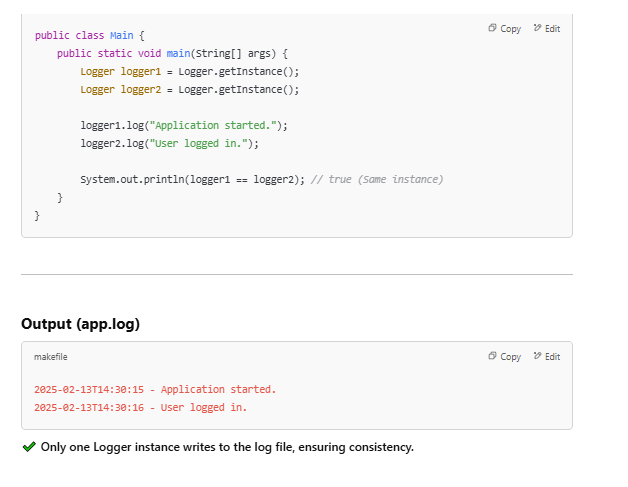
A common use case is generating **unique IDs per request**.











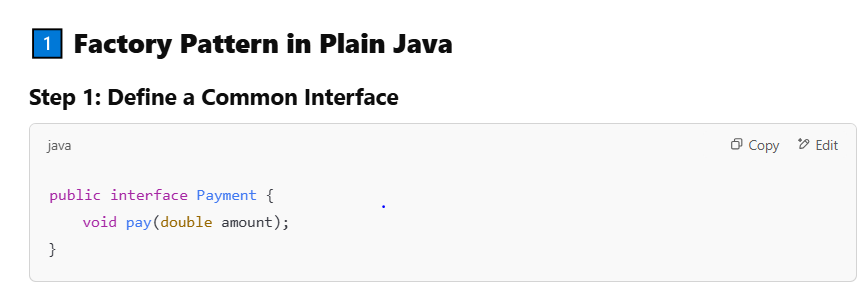
**Why is Unit Testing a Singleton in Spring Boot Difficult?**

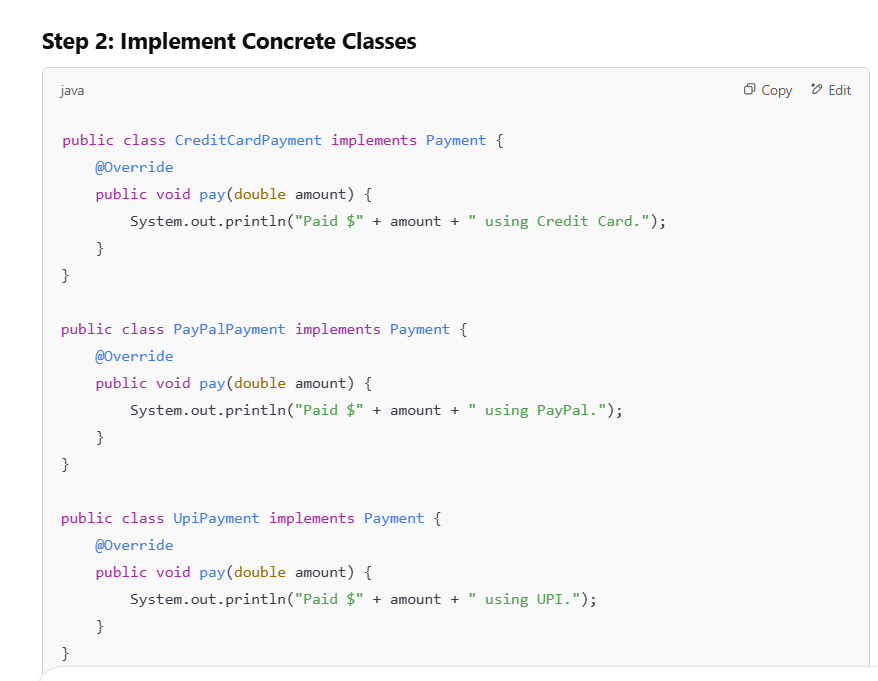
Unit testing a **Singleton** in Spring Boot can be challenging due to:

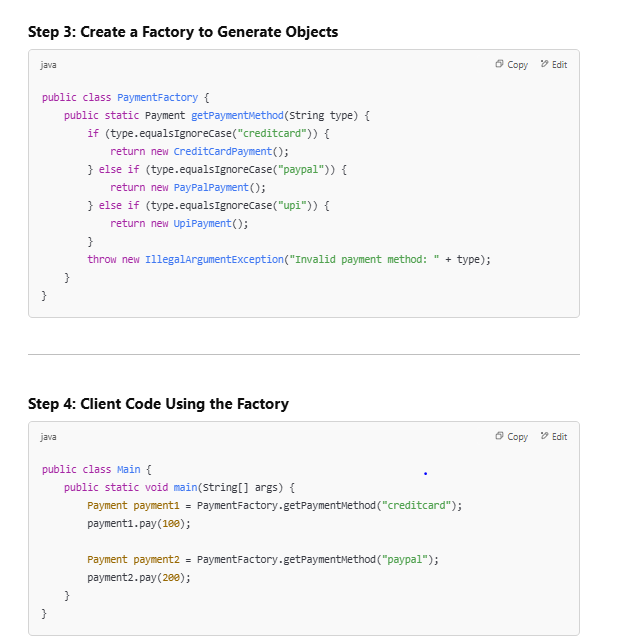
1. **State Retention** – The singleton retains state across tests, leading to unintended side effects.
2. **Dependency on Global State** – If the singleton holds shared resources (e.g., database connections, configuration), tests might affect each other.
3. **Hard to Mock** – Since only one instance exists, replacing it with a mock is difficult.
4. **Tight Coupling** – Singletons are often tightly coupled with other classes, making unit testing harder.

2:--- Factory Pattern

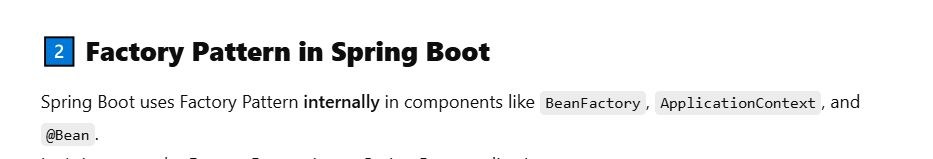








✔️ **Loose coupling** – The client (Main) does not depend on concrete implementations.  
✔️ **Easy to add new payment methods** without modifying existing code.



3:- Builder design pattern

### ****Builder Design Pattern in Java and Spring Boot****

The **Builder Design Pattern** is a **creational design pattern** used to construct a complex object step by step. It allows the construction of different types of objects using the same construction process.

### ****Use Case:****

The **Builder Pattern** is useful when:

* The object to be created is **complex** and has multiple attributes.
* We want to construct an object in **steps**, without requiring all arguments to be provided at once.
* The client doesn’t need to know the internal structure of the object being built.

#### ****Real-World Example:****

Imagine you are building a **User** profile with several attributes such as name, email, address, phone number, etc. The builder pattern helps in constructing a User object step by step.

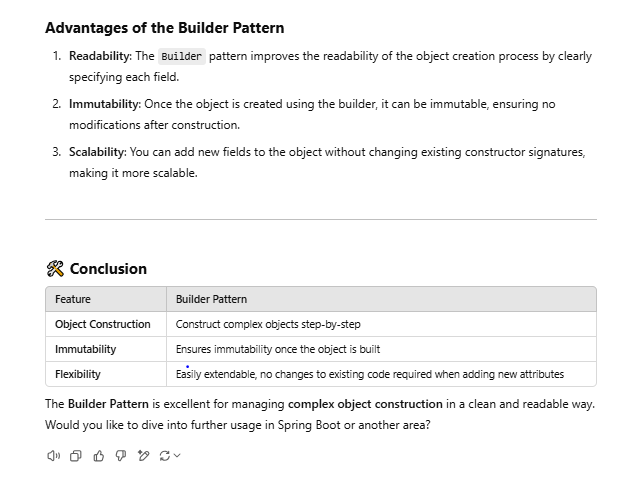
### ****Step-by-Step Example of Builder Pattern****

Let’s go through a simple example using **Java** and **Spring Boot**.









4: Prototype design pattern

