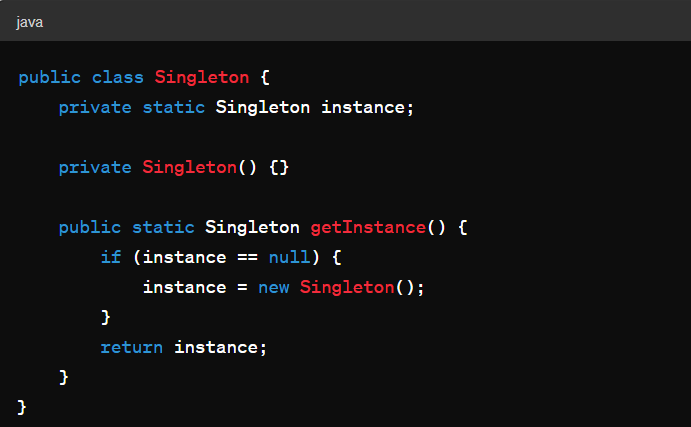
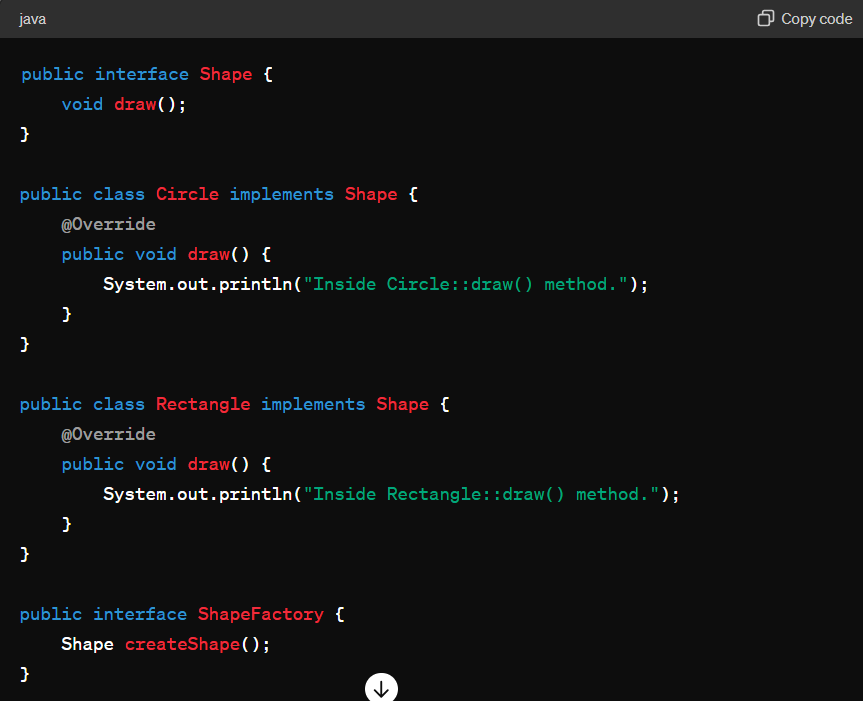
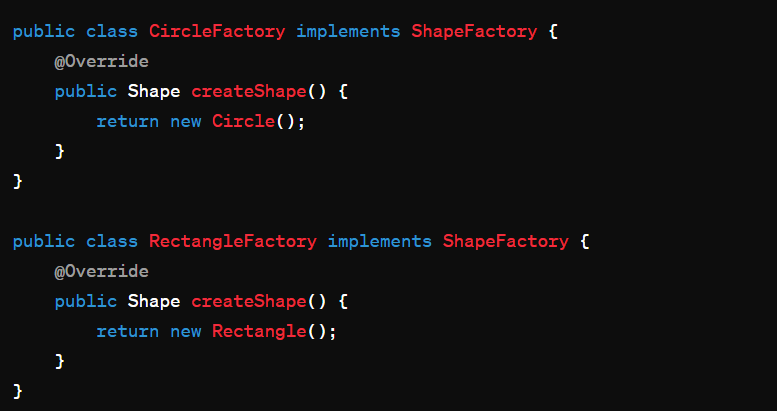
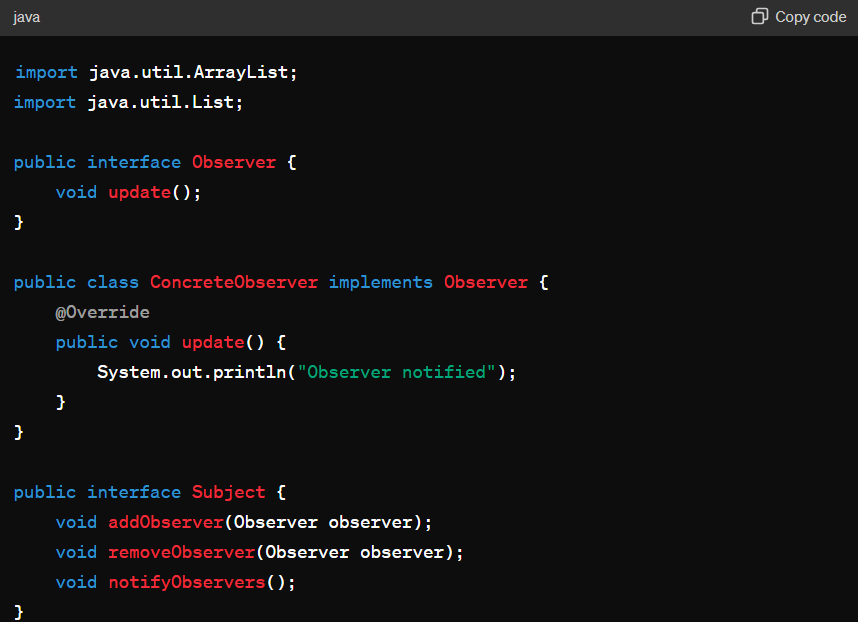
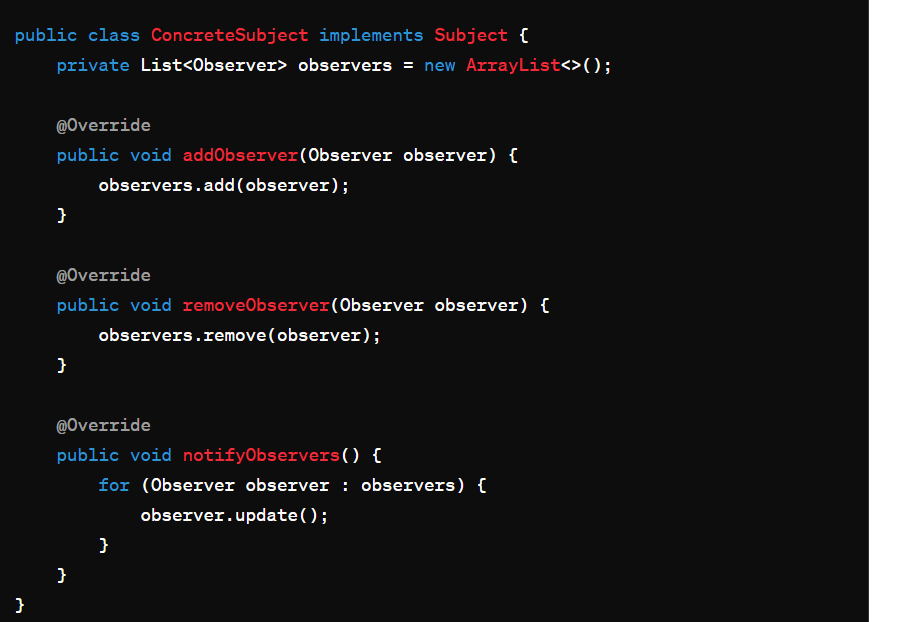
Design patterns are general reusable solutions to common problems encountered during software development. They provide a template or guideline for solving recurring design issues. Here are some of the most commonly used types of design patterns:

1. **Creational Patterns**:
   * **Singleton Pattern**: Ensures that a class has only one instance and provides a global point of access to that instance.
   * 
   * **Factory Method Pattern**: Defines an interface for creating objects, but allows subclasses to alter the type of objects that will be created.
   * 
   * 
   * **Abstract Factory Pattern**: Provides an interface for creating families of related or dependent objects without specifying their concrete classes.
   * **Builder Pattern**: Separates the construction of a complex object from its representation, allowing the same construction process to create different representations.
   * **Prototype Pattern**: Creates new objects by copying an existing object, known as a prototype, during runtime.
2. **Structural Patterns**:
   * **Adapter Pattern**: Allows objects with incompatible interfaces to work together by providing a wrapper that translates one interface into another.
   * **Decorator Pattern**: Attaches additional responsibilities to an object dynamically, providing a flexible alternative to subclassing for extending functionality.
   * **Facade Pattern**: Provides a simplified interface to a complex system, hiding its complexities from clients.
   * **Proxy Pattern**: Provides a surrogate or placeholder for another object to control access to it.
3. **Behavioral Patterns**:
   * **Observer Pattern**: Defines a one-to-many dependency between objects, so that when one object changes state, all its dependents are notified and updated automatically.
   * 
   * 
   * **Strategy Pattern**: Defines a family of algorithms, encapsulates each one, and makes them interchangeable. It lets the algorithm vary independently from clients that use it.
   * **Command Pattern**: Encapsulates a request as an object, thereby allowing for parameterization of clients with queues, requests, and operations.
   * **Iterator Pattern**: Provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
   * **Chain of Responsibility Pattern**: Allows an object to send a command without knowing which object will handle it, and it allows multiple objects to handle the command without explicit knowledge of each other.
4. **Concurrency Patterns**:
   * **Immutable Object Pattern**: Ensures that an object's state cannot be modified after it is created, which simplifies concurrent programming.
   * **Producer-Consumer Pattern**: Provides a solution for communicating between two threads: one that produces data and another that consumes it.
   * **Reader-Writer Lock Pattern**: Allows multiple readers or a single writer to access a resource concurrently, but not both simultaneously.
   * **Thread Pool Pattern**: Manages and reuses a pool of threads to execute tasks asynchronously, improving performance in multi-threaded applications.

These are just a few examples of design patterns. Each type of pattern addresses different aspects of software design and development, providing solutions to common challenges encountered in various scenarios.

**4. Template Method Pattern**

The RestTemplate and JdbcTemplate classes in Spring are examples of the template method pattern, providing a way to execute boilerplate code and allowing customization through callbacks.