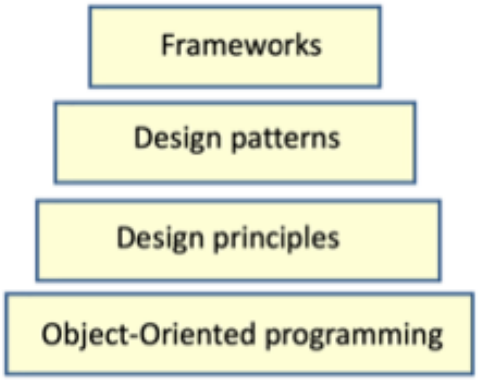
Categories of patterns

Creational	Structural	Behavioral
▪ Factory method ▪ Abstract factory ▪ Builder ▪ Singleton	▪ Composite ▪ Decorator ▪ Adapter ▪ Façade ▪ Proxy	▪ Command ▪ Iterator ▪ Mediator ▪ Chain of responsibility ▪ Observer ▪ State ▪ Strategy ▪ Template method

Design principles

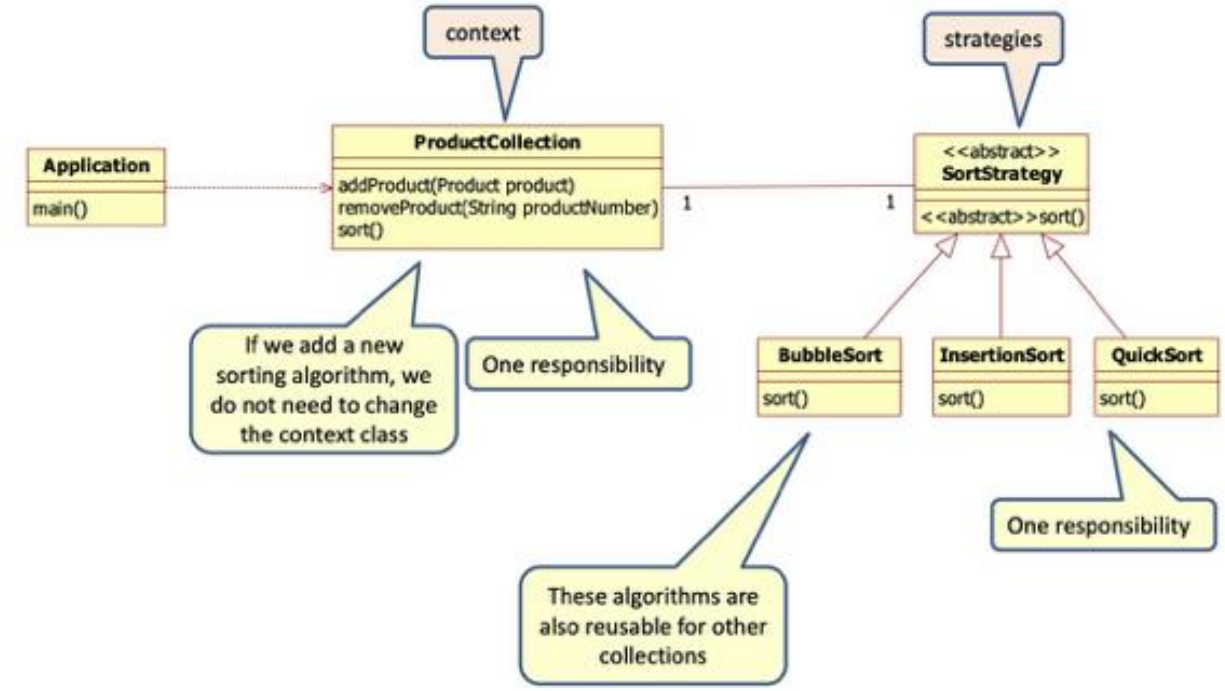
- Keep it simple
- Keep it flexible
- Loose coupling
- Separation of concern
- Information hiding
- Principle of modularity
- DRY: Don't repeat yourself
- Encapsulate what varies
- Solid
  - **S**ingle Responsibility Principle (SRP)
  - **O**pen-Closed Principle (OCP)
  - **L**iskov Substitution Principle (LSP)
  - **I**nterface Segregation Principle (ISP)
  - **D**ependency Inversion Principle (DIP)

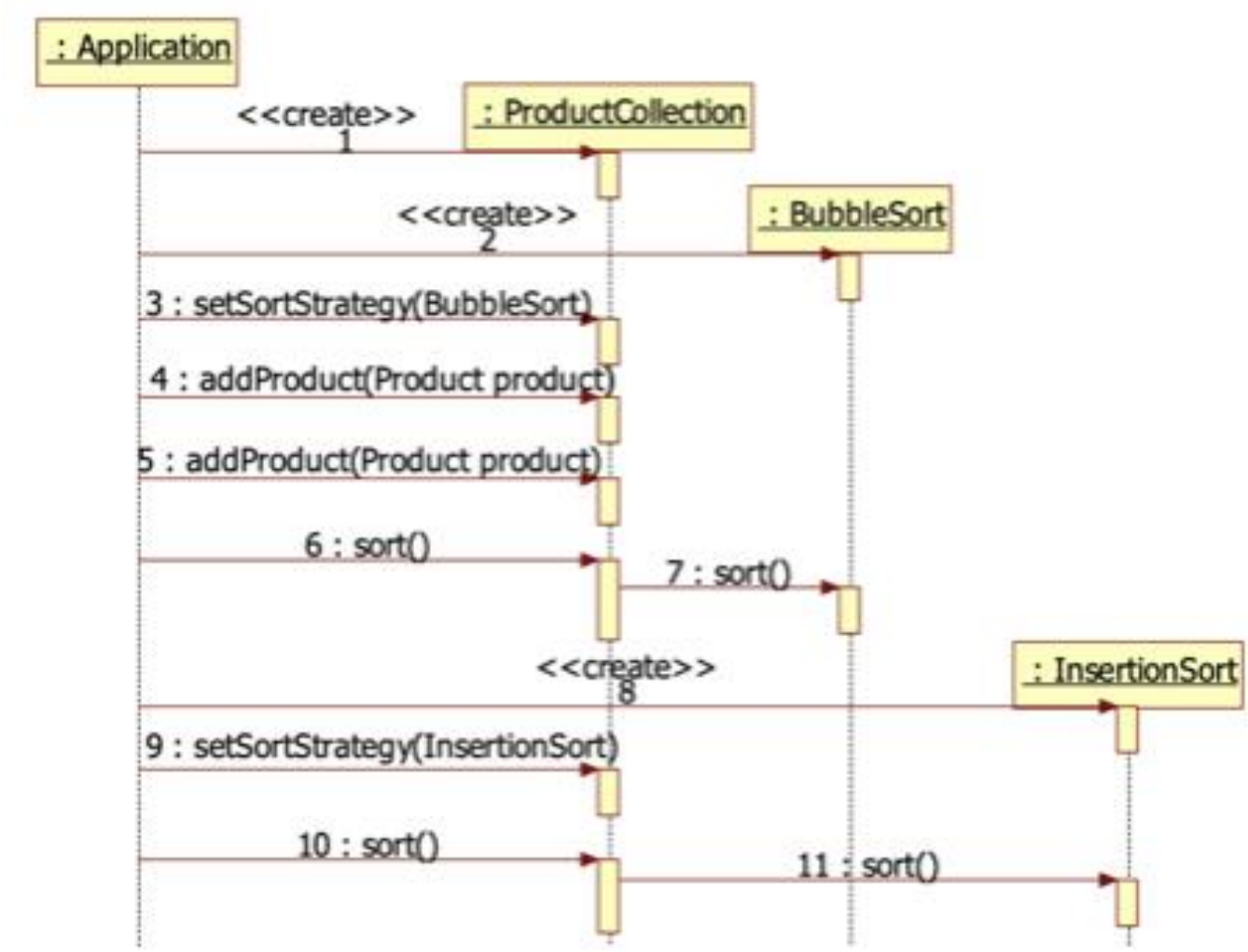


Strategy Pattern

The strategy pattern extracts algorithms (strategies) from a certain class (context class) and makes a different class for every single algorithm. This gives the following advantages

- We can easily add new algorithms without changing the context class
- The strategies are better reusable



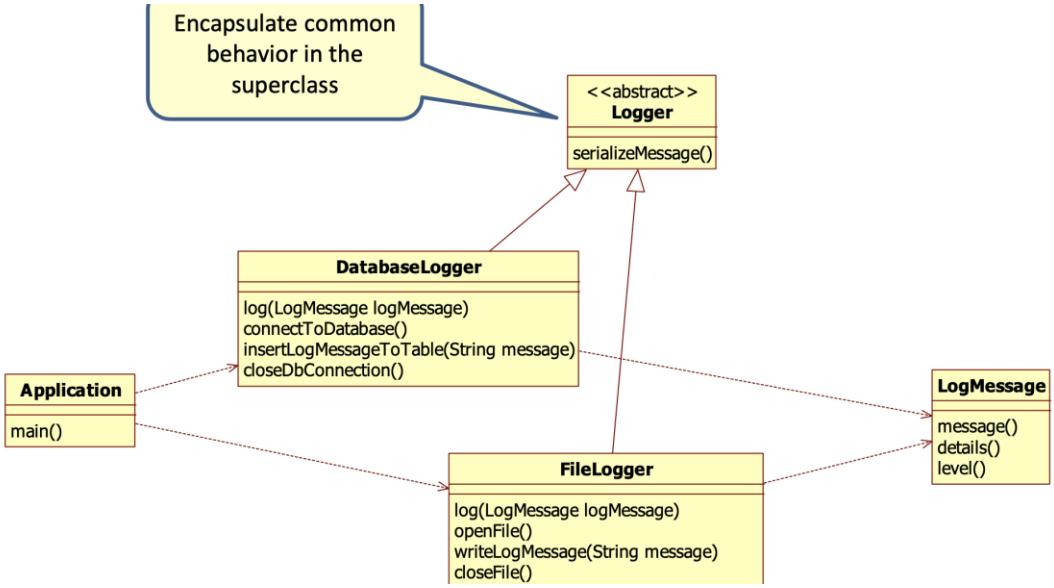


Advantages of the Strategy Pattern

- It's easy to switch between different algorithms (strategies) in runtime because you're using polymorphism in the interfaces.
- Clean code because you avoid conditional-infested code (not complex).
- More clean code because you separate the concerns into classes (a class to each strategy).

TEMPLATE PATTERN

The template method pattern defines the skeleton of an algorithm in the superclass but lets subclasses override specific steps of the algorithm without changing its structure

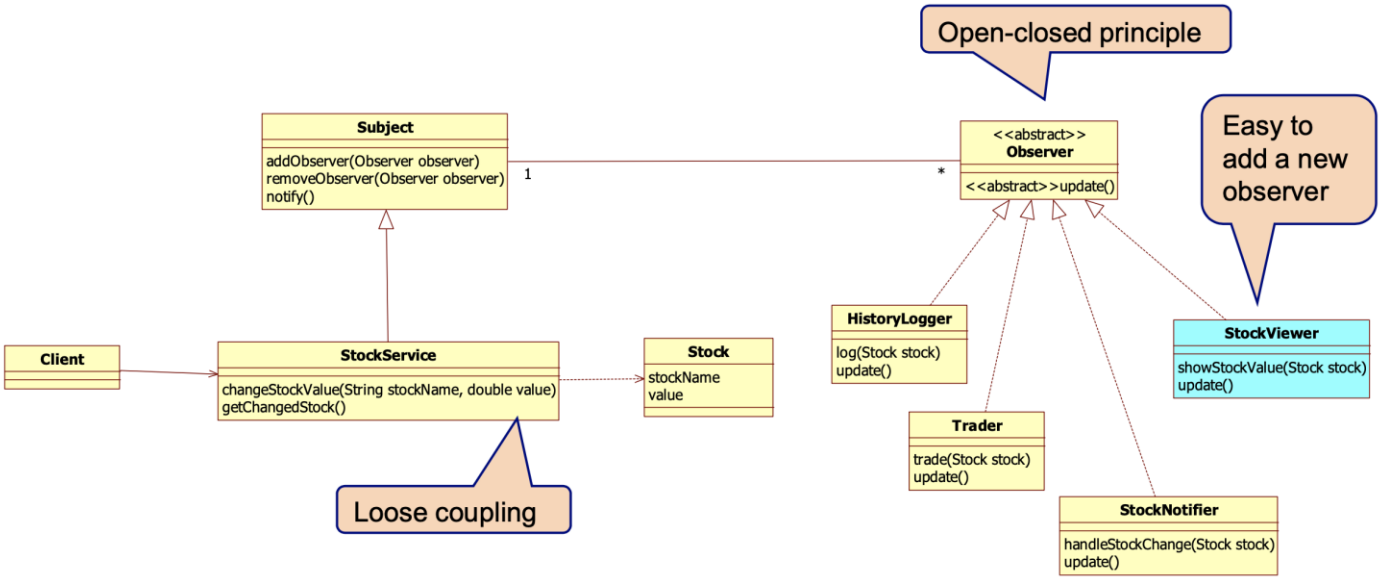


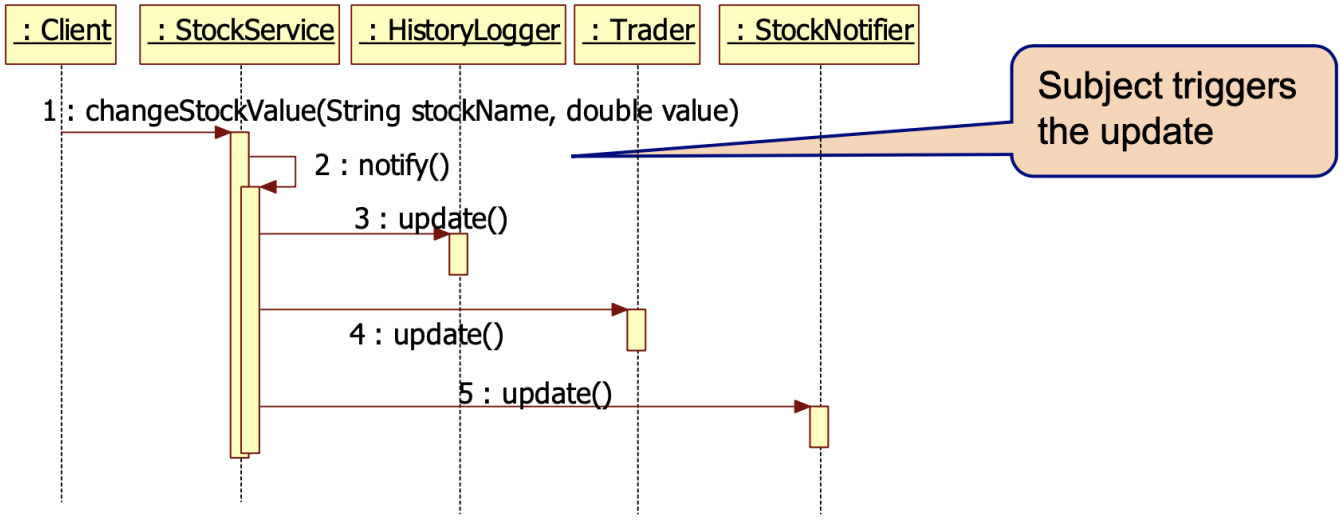
The Template Method pattern provides you with the following advantages:

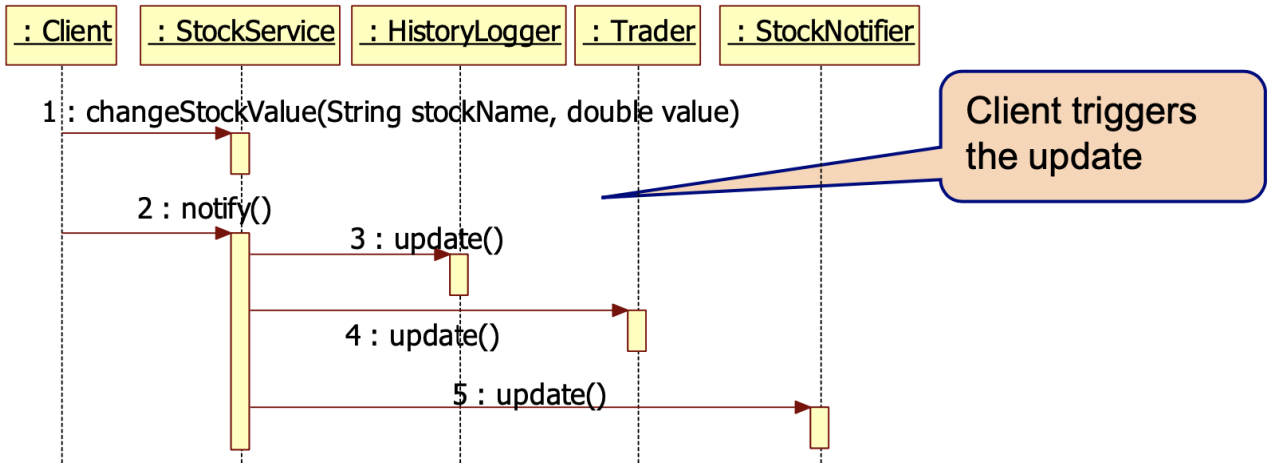
- As we saw earlier in the chapter, there is no code duplication.
- Code reuse happens with the Template Method pattern as it uses inheritance and not composition. ...
- Flexibility lets subclasses decide how to implement steps in an algorithm.

OBSERVER PATTERN

The Observer design pattern lets several observer objects be notified when a subject is changed in some way.







The Observer pattern provides you with the following advantages:

- It supports the principle of loose coupling between objects that interact with each other
- It allows sending data to other objects effectively without any change in the `Subject` or `Observer` classes
- `Observers` can be added/removed at any point in time

The following are the disadvantages of the Observer pattern:

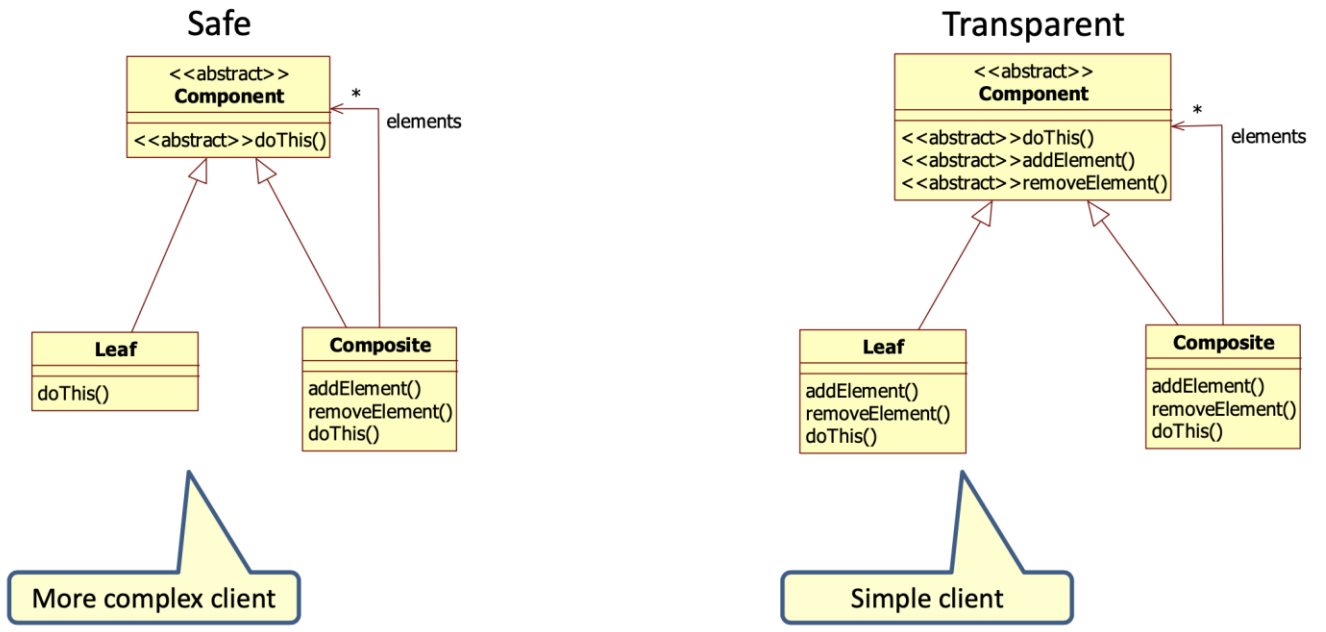
- The Observer interface has to be implemented by `ConcreteObserver`, which involves inheritance. There is no option for composition, as the Observer interface can be instantiated.
- If not correctly implemented, the `Observer` can add complexity and lead to inadvertent performance issues.
- In software application, notifications can, at times, ...

Composite pattern

The intent of this pattern is to compose objects into tree structures to represent part-whole hierarchies.

Composite lets clients treat individual objects and compositions of objects uniformly.

Safe or transparent

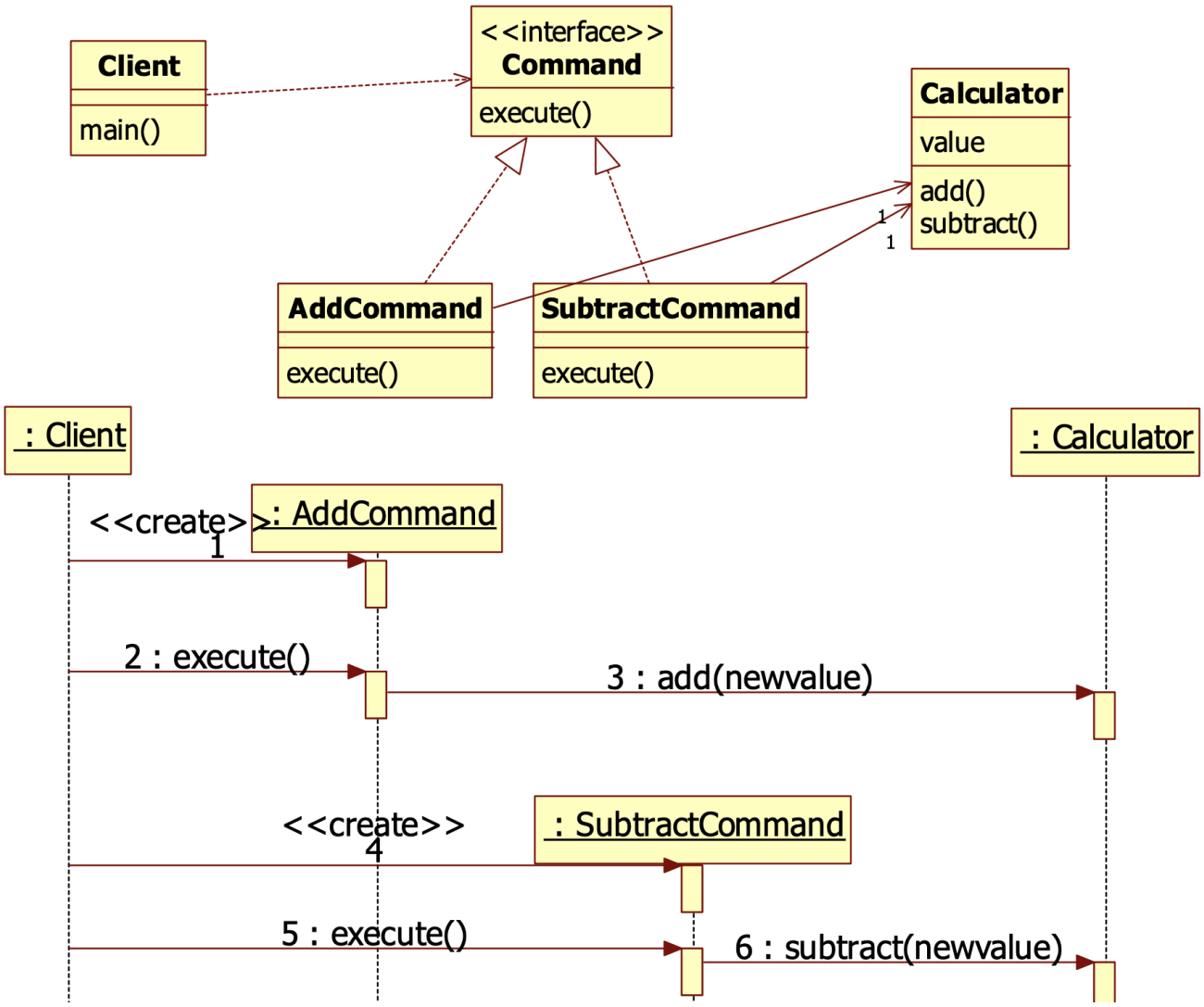


Advantage of Composite Design Pattern

- It defines class hierarchies that contain primitive and complex objects.
- It makes easier to you to add new kinds of components.
- It provides flexibility of structure with manageable class or interface.

Command pattern

- Encapsulate a request into a single object
- Advantages:
  - Command objects can be logged
  - Command objects can be used for undo/redo functionality
  - Command objects can be replayed

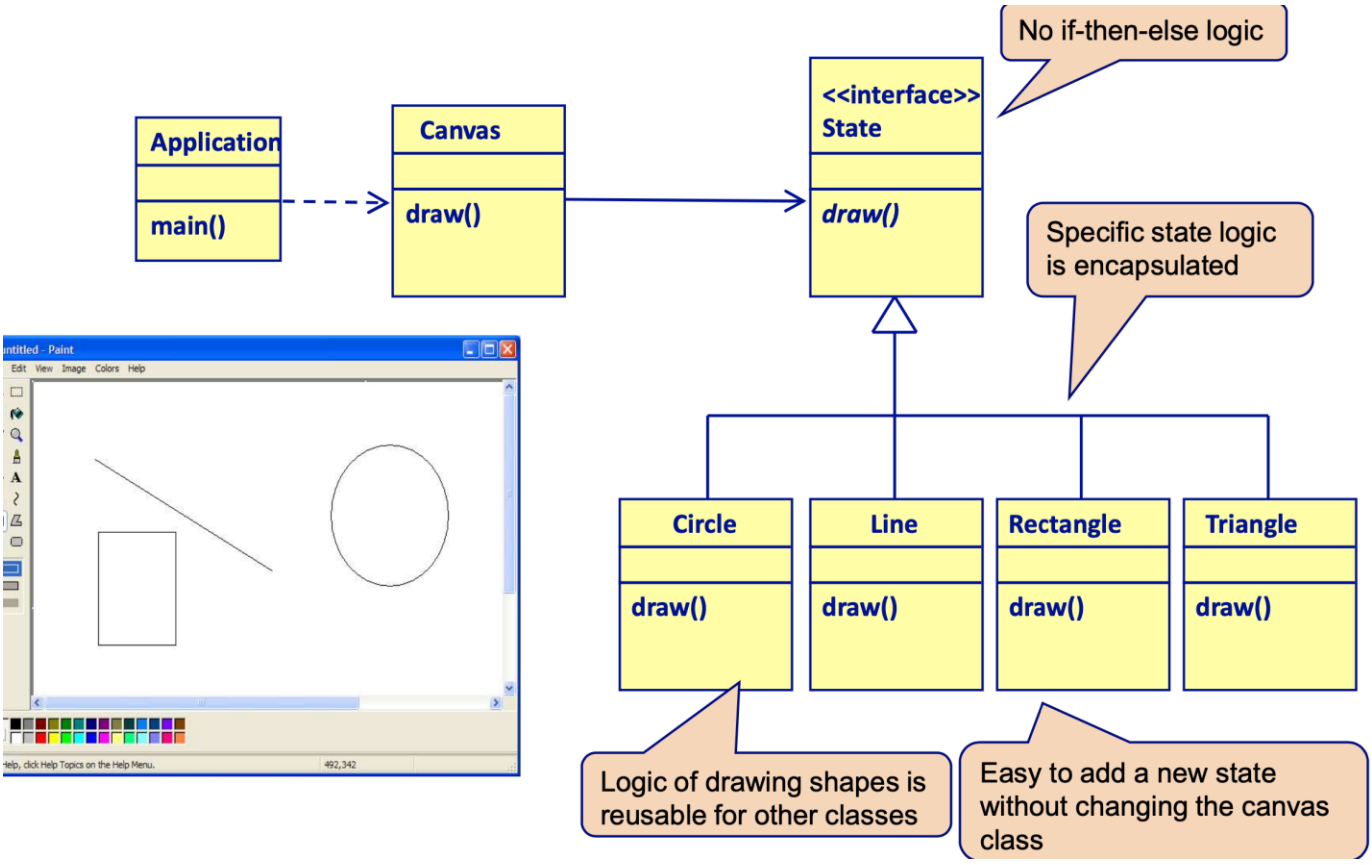


What problem does it solve?

- *Whenever you need to know the actions taken by a user, you can use the command pattern.*
- *One important application of this pattern is undo/redo functionality.*
- *Providing support for macros (recording and playback of macros).*

State pattern

The state pattern is a design pattern that allows an object to completely change its behavior depending upon its current internal state.



State or strategy?

Strategy

- The context can have different algorithms
- Strategies do not know each other
  - The context has one strategy

State

- The context can have different states
- States know the local state transitions
  - The context has currently one state, but that state will change over time.

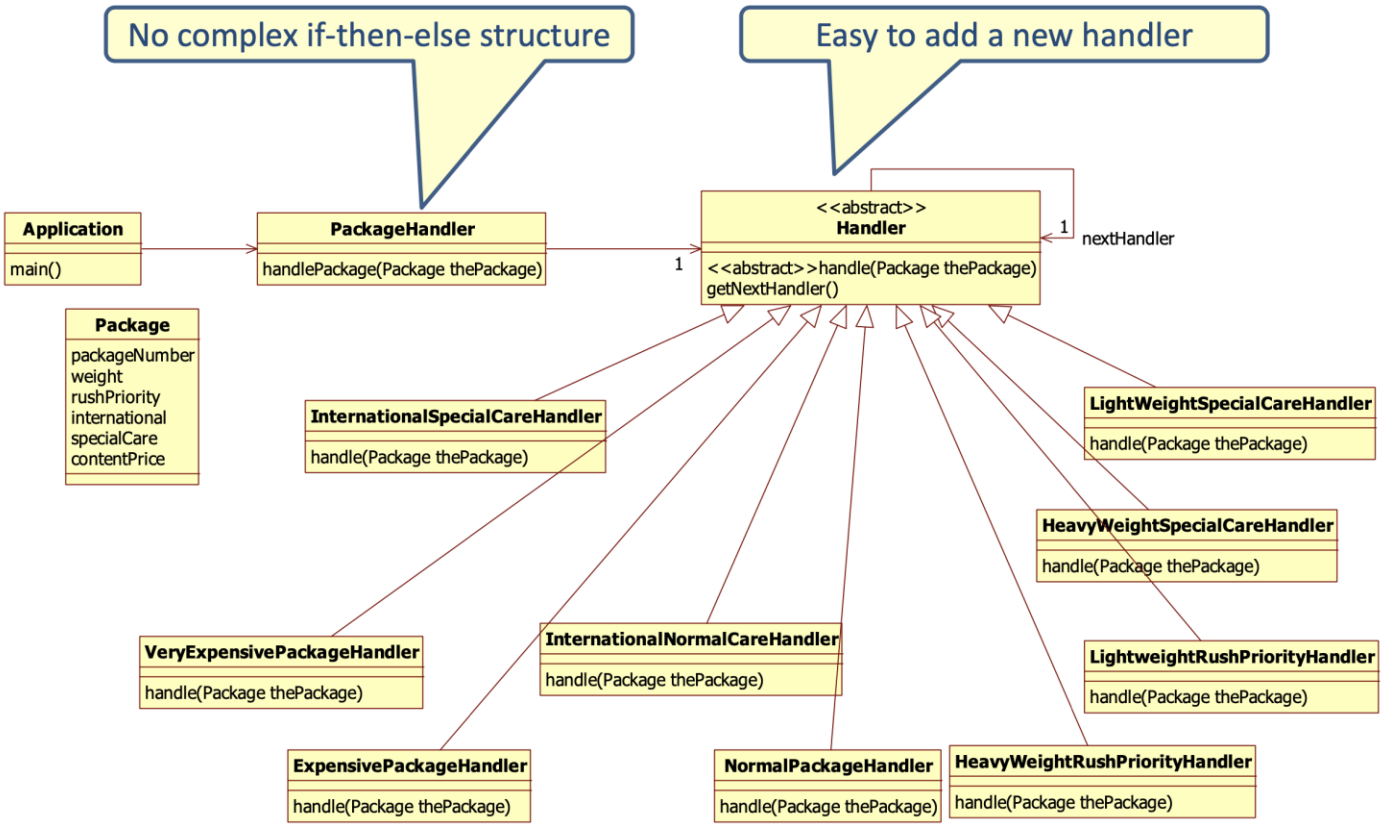
Advantages. The State pattern **minimizes conditional complexity**, eliminating the need for if and switch statements in objects that have different behavior requirements unique to different state transitions.

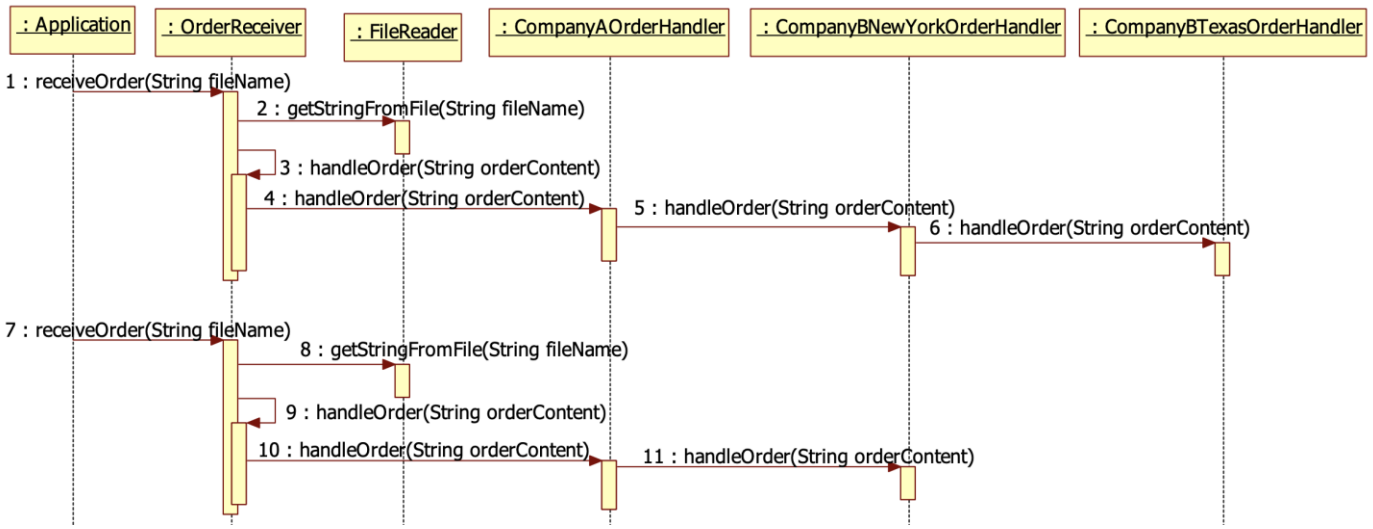
CHAIN OF RESPONSABILITY

Chain of responsibility pattern is used to achieve loose coupling in software design where a request from the client is passed to a chain of objects to process them. Later, the object in the chain will decide themselves who will be processing the request and whether the request is required to be sent to the next object in the chain or not.

Where and When Chain of Responsibility pattern is applicable :

- When you want to decouple a request's sender and receiver
- Multiple objects, determined at runtime, are candidates to handle a request
- When you don't want to specify handlers explicitly in your code
- When you want to issue a request to one of several objects without specifying the receiver explicitly.





Advantages:

1. Decouples the sender of the request and its receivers.

2. Simplifies your object as it doesn't have to know about the chain structure and keep direct references to its members.

3. Allows you to add or remove responsibilities dynamically by changing the members or order of the chain.

Disadvantage:

1. Hard to observe the run-time characteristics and debug.