Verisoft

SDET Course

Design Patterns - Chain of Responsibility

3 Types of Design Patterns

- Creational
  - Singleton
  - Builder
  - Prototype
  - Factory Method
  - Abstract Factory

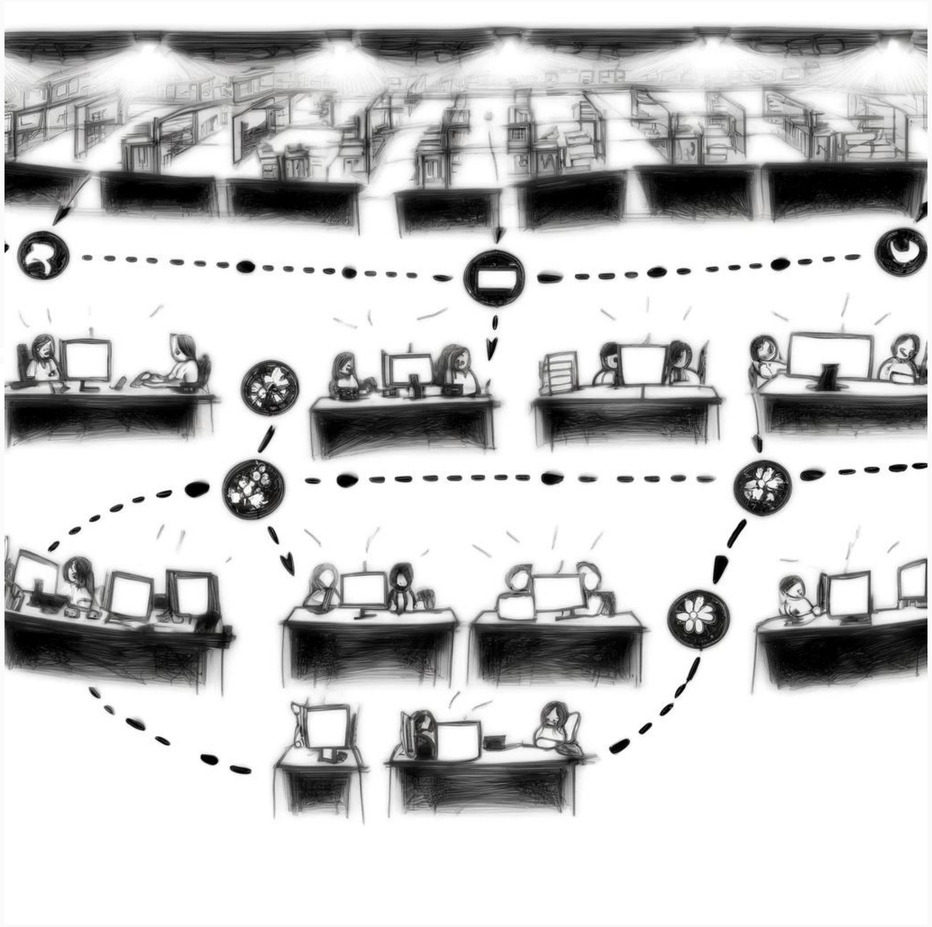
- Structural
  - Adapter
  - Composite
  - Proxy
  - Flyweight
  - Bridge
  - Facade
  - Decorator

- Behavioral
  - Strategy
  - Observer
  - Command
  - Memento
  - State
  - Template Method
  - Mediator
  - **Chain of Responsibility**
  - Interpreter
  - Visitor
  - Iterator

Agenda

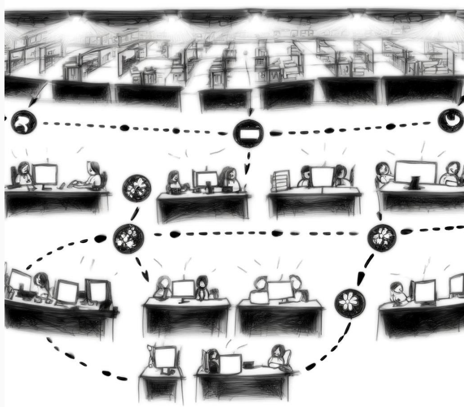
- Description
- Diagram
- Code sample (Java)
- Use cases

Descri
on

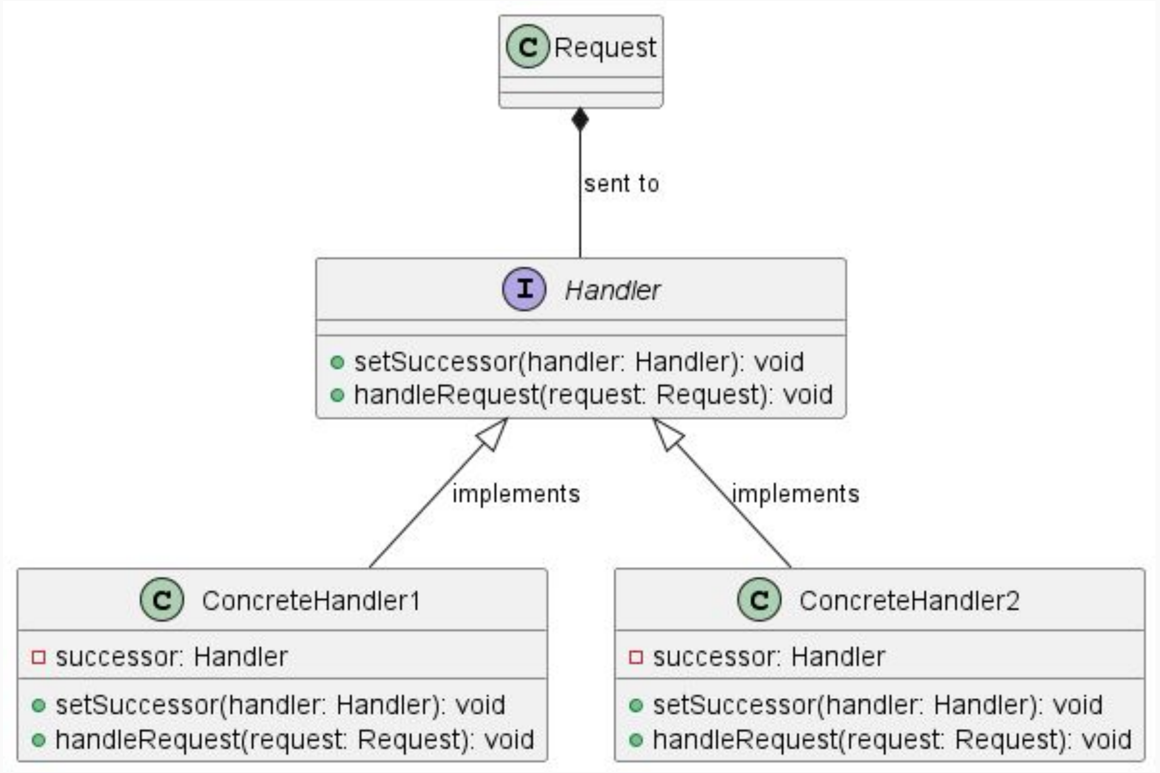


Description

The Chain of Responsibility design pattern is a behavioral design pattern used to pass the request along a chain of handlers. Upon receiving a request, each handler decides either to process the request or to pass it to the next handler in the chain. This pattern allows an object to send a command without knowing which object will handle the request. It effectively decouples the sender and the receiver by giving multiple objects a chance to handle the request. The chain can be dynamically composed at runtime, and the handlers in the chain do not need to know about the overall structure of the chain. This pattern is particularly useful when multiple objects can handle a request, but the specific handler isn't determined a priori, thus promoting loose coupling and flexibility in the distribution of responsibilities.

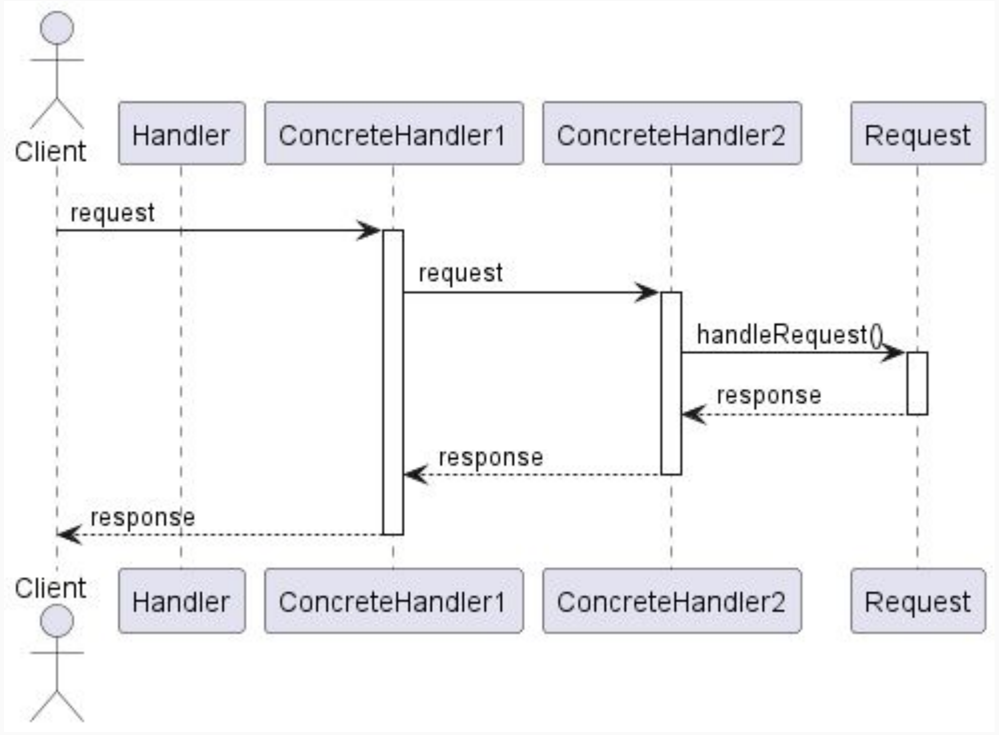


Class Diagram



C Request
sent to
I Handler
● setSuccessor(handler: Handler): void
● handleRequest(request: Request): void
implements
implements
C ConcreteHandler1
□ successor: Handler
● setSuccessor(handler: Handler): void
● handleRequest(request: Request): void
C ConcreteHandler2
□ successor: Handler
● setSuccessor(handler: Handler): void
● handleRequest(request: Request): void

Sequence Diagram



Code Sample

Use cases

- General
  - Logging Frameworks
  - Approval Processes

- In Test Automation
  - Test Result Processing
  - Automated Test Fixtures (Junit 5 extensions)
  - Middleware for API Testing

Happy Coding