OOP practical part

Must have patterns:

Behavioral:

- Chain of Responsibility
- Iterator
- Observer
- Strategy
- Visitor

Creational:

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

Structural:

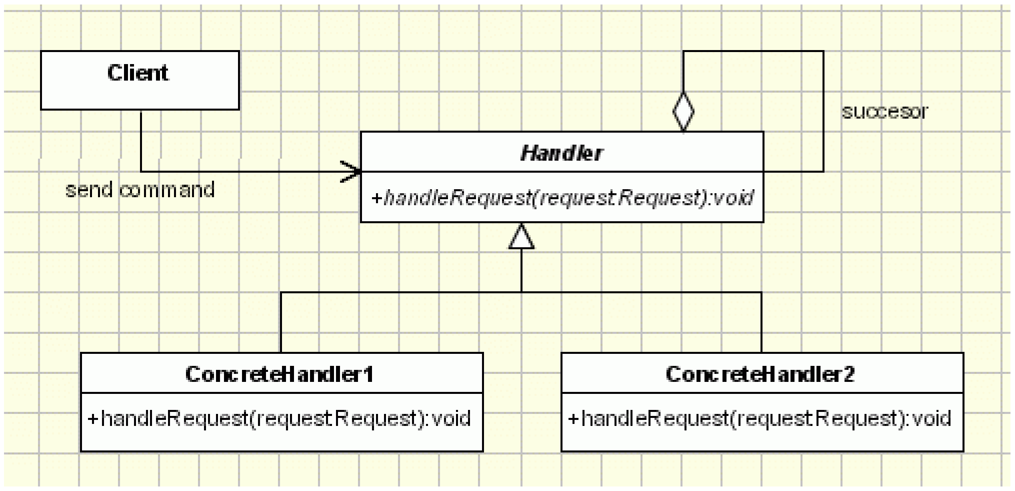
- Adapter
- Composite
- Decorator
- Facade
- Proxy

(The upcoming diagrams are just default illustrations, probably there will be different diagrams)

Behavioral patterns:

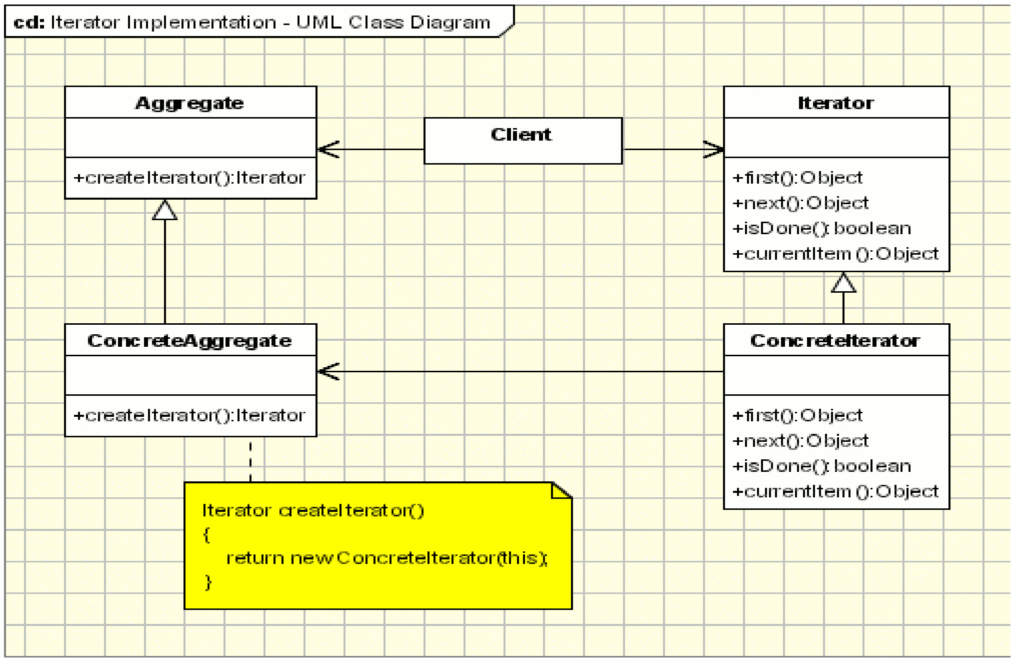
Chain of Responsibility

https://refactoring.guru/design-patterns/chain-of-responsibility


Client
send command
Handler
+handleRequest(request:Request):void
succesor
ConcreteHandler1
+handleRequest(request:Request):void
ConcreteHandler2
+handleRequest(request:Request):void

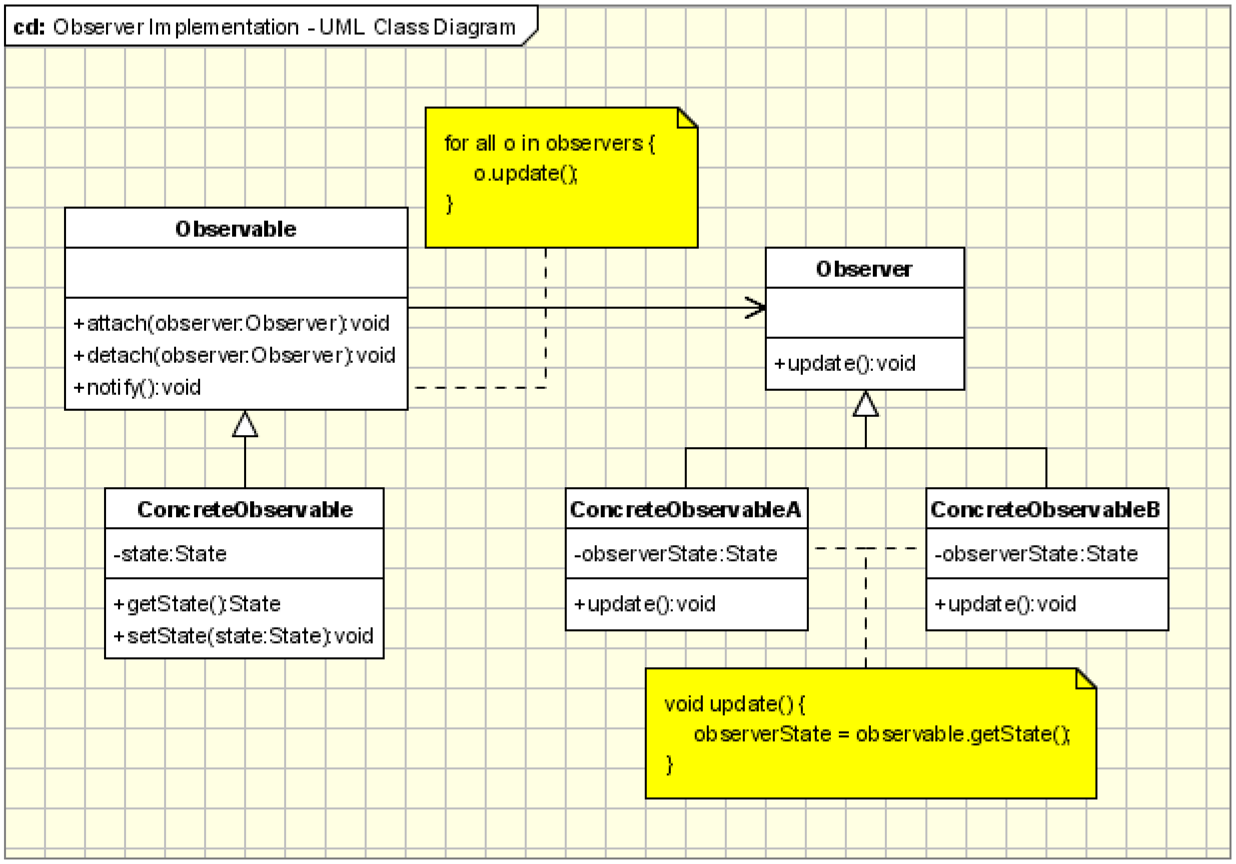
Iterator

https://refactoring.guru/design-patterns/iterator



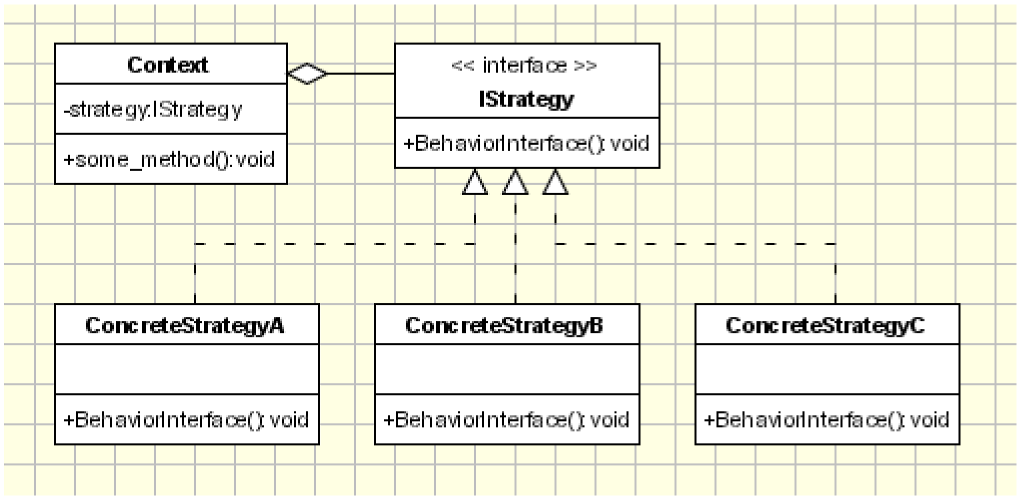
Observer

https://refactoring.guru/design-patterns/observer



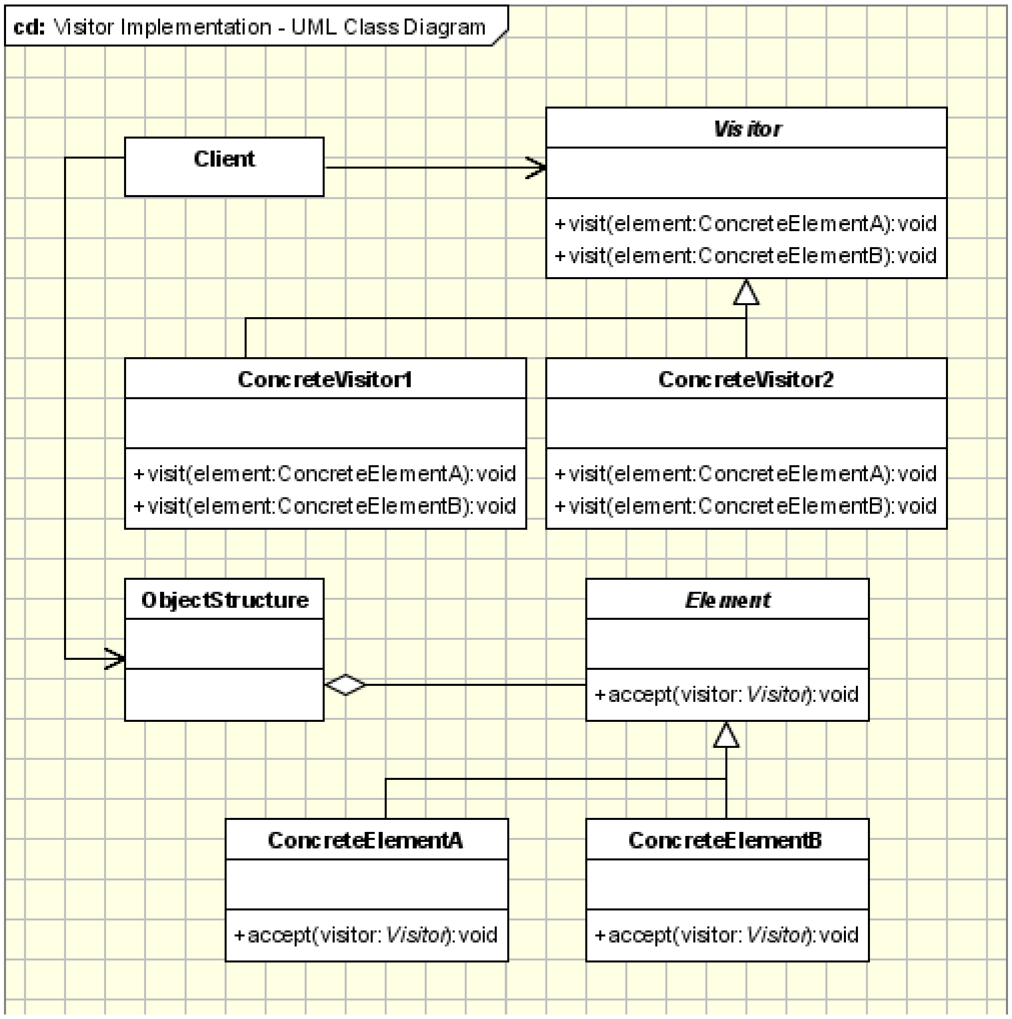
Strategy

https://refactoring.guru/design-patterns/strategy



Visitor

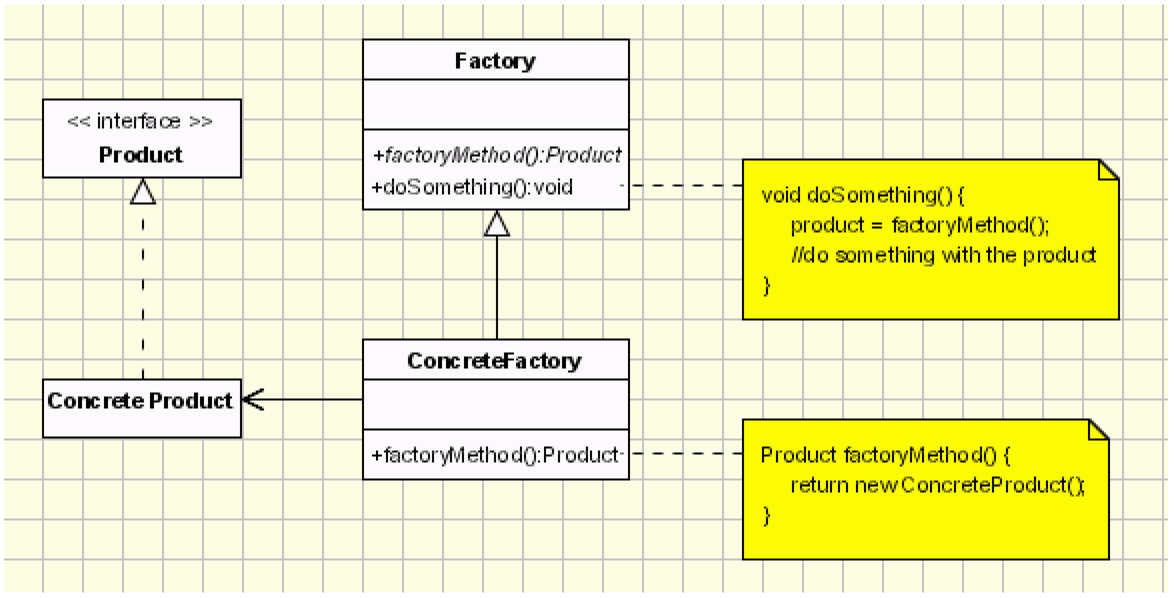
https://refactoring.guru/design-patterns/visitor


cd: Visitor Implementation - UML Class Diagram
Client
Visitor
+visit(element:ConcreteElementA):void
+visit(element:ConcreteElementB):void
ConcreteVisitor1
+visit(element:ConcreteElementA):void
+visit(element:ConcreteElementB):void
ConcreteVisitor2
+visit(element:ConcreteElementA):void
+visit(element:ConcreteElementB):void
ObjectStructure
Element
+accept(visitor: Visitor):void
ConcreteElementA
+accept(visitor: Visitor):void
ConcreteElementB
+accept(visitor: Visitor):void

Creational patterns:

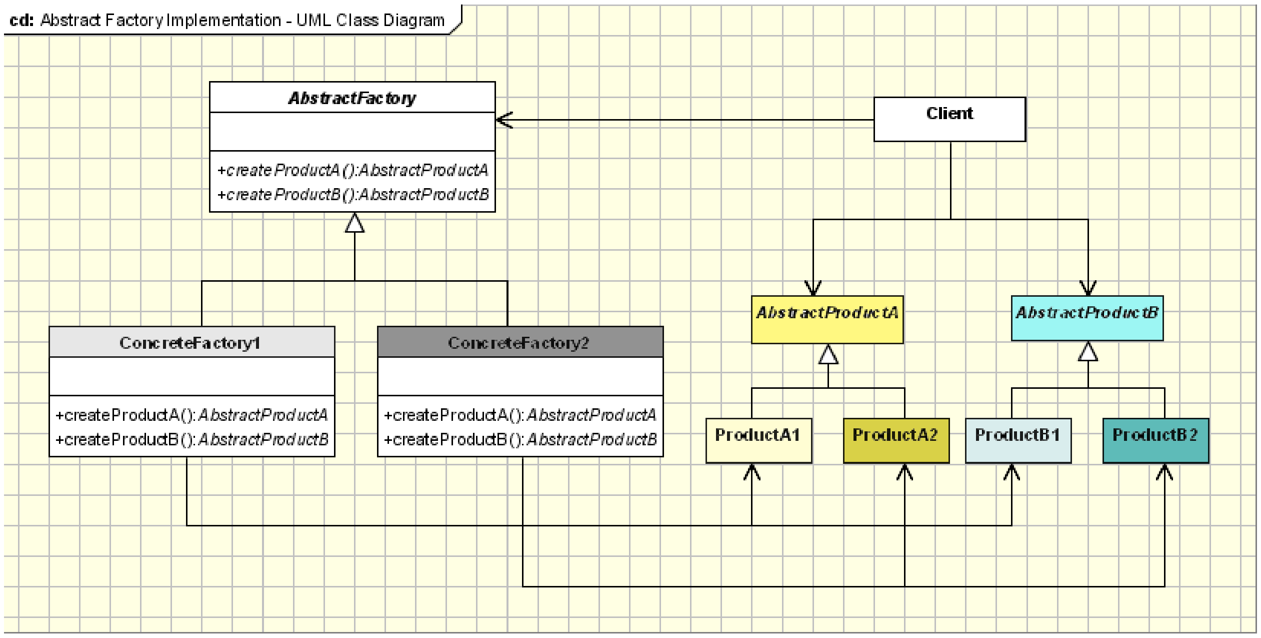
Factory

https://refactoring.guru/design-patterns/factory-method



Abstract Factory

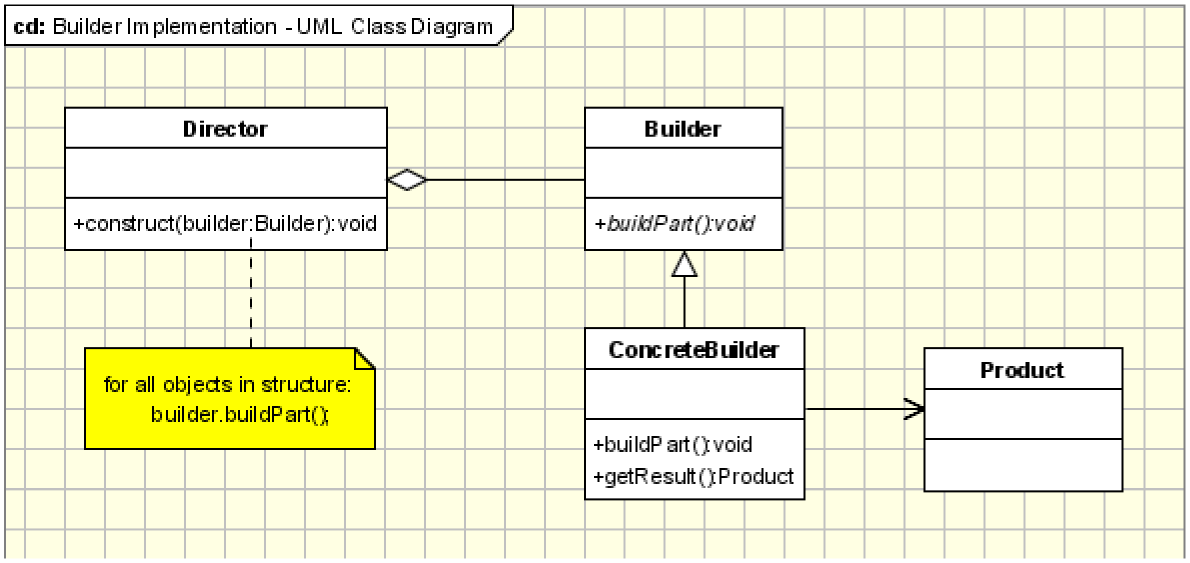
https://refactoring.guru/design-patterns/abstract-factory



cd: Abstract Factory Implementation - UML Class Diagram
AbstractFactory
+createProductA():AbstractProductA
+createProductB():AbstractProductB
Client
ConcreteFactory1
+createProductA():AbstractProductA
+createProductB():AbstractProductB
ConcreteFactory2
+createProductA():AbstractProductA
+createProductB():AbstractProductB
AbstractProductA
AbstractProductB
ProductA1
ProductA2
ProductB1
ProductB2

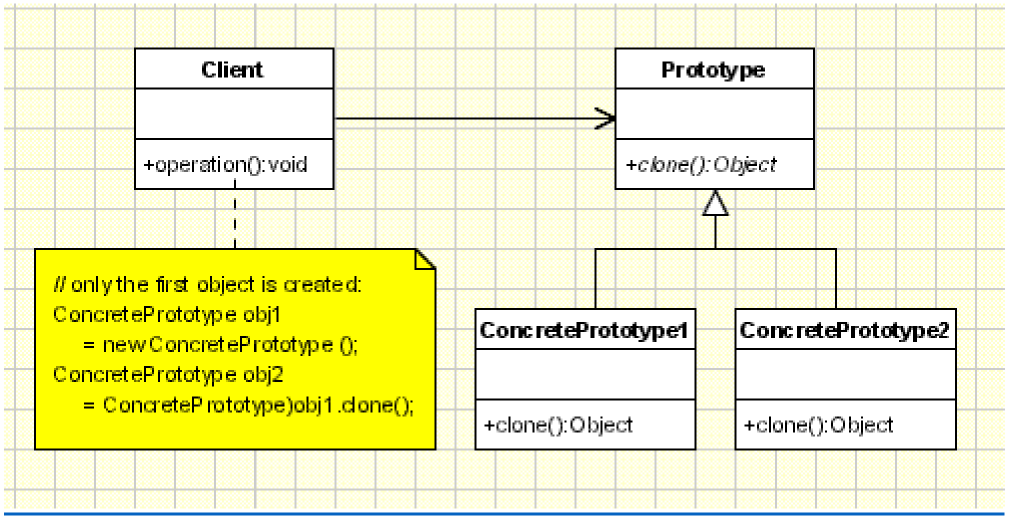
Builder

https://refactoring.guru/design-patterns/builder



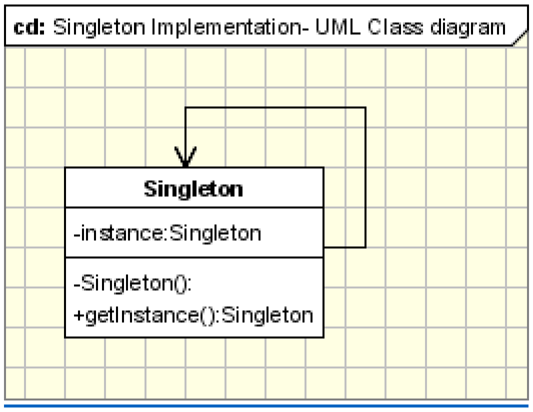
Prototype

https://refactoring.guru/design-patterns/prototype



Singleton

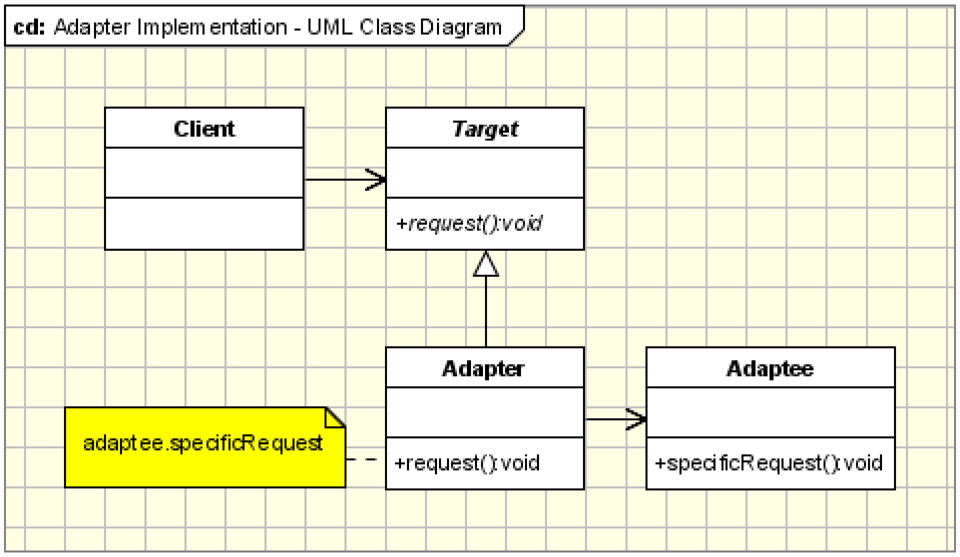
https://refactoring.guru/design-patterns/singleton



Structural patterns:

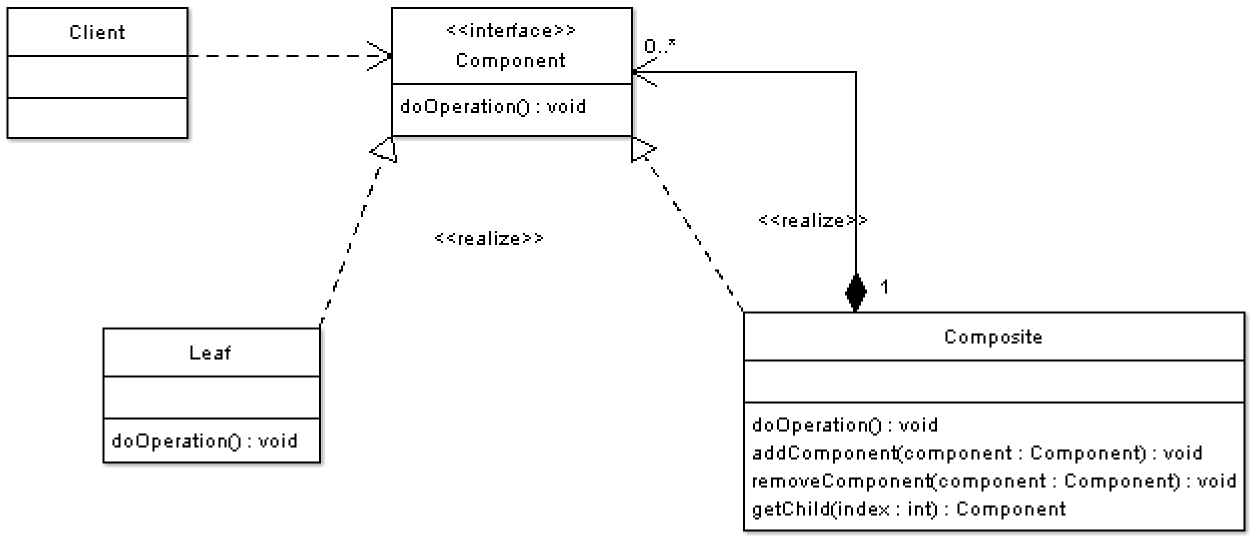
Adapter

https://refactoring.guru/design-patterns/adapter



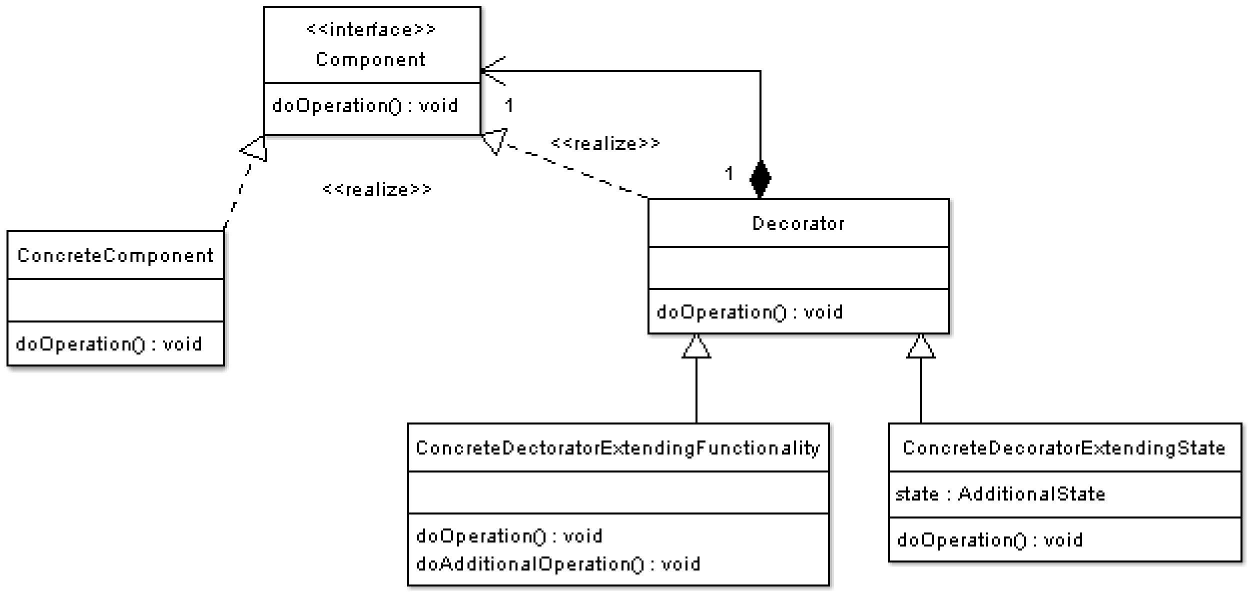
Composite

https://refactoring.guru/design-patterns/composite



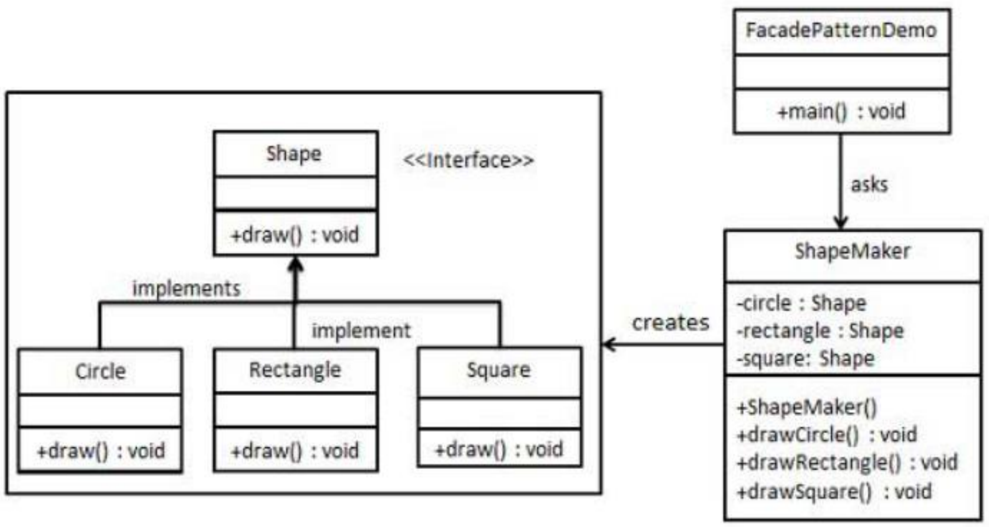
Decorator

https://refactoring.guru/design-patterns/decorator



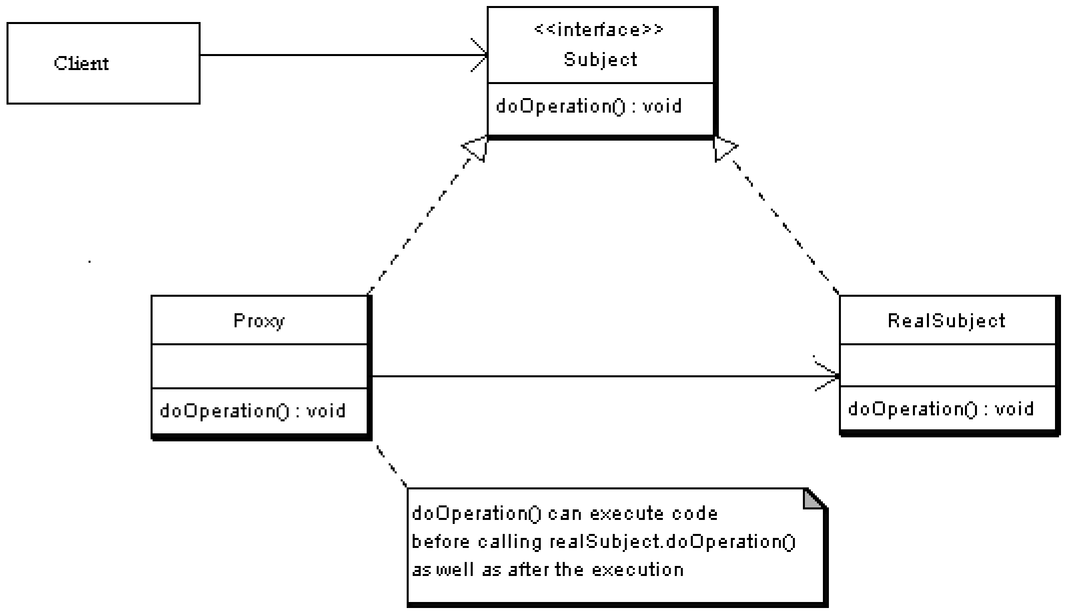
Facade

https://refactoring.guru/design-patterns/facade



Proxy

https://refactoring.guru/design-patterns/proxy


Client
<<interface>>
Subject
doOperation() : void
Proxy
doOperation() : void
RealSubject
doOperation() : void
doOperation() can execute code
before calling realSubject.doOperation()
as well as after the execution