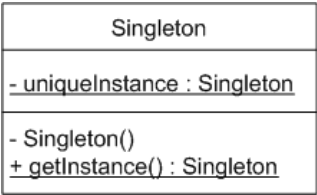
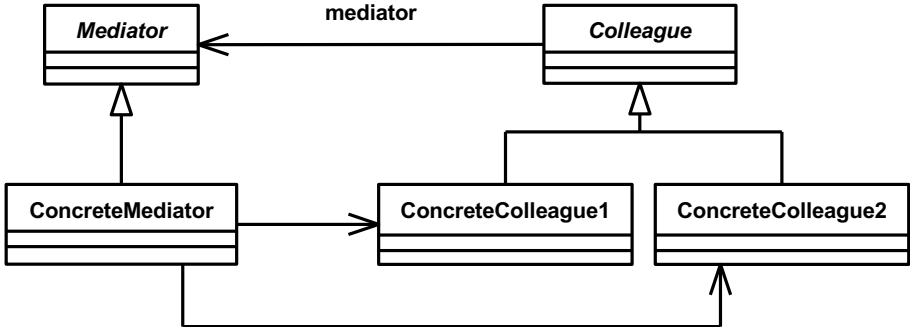
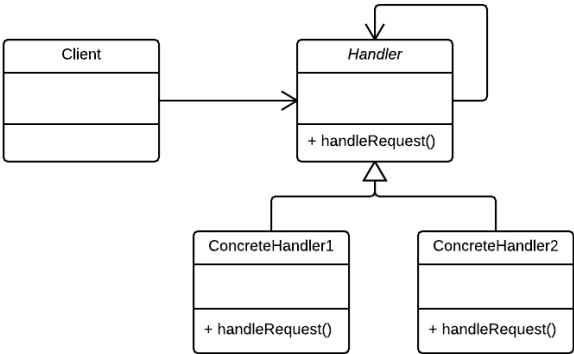


Annexe Patrons - final

Nom du patron	Structure générique
Stratégie (Strategy)	<pre> classDiagram class Context { ContextInterface() } class Strategy { AlgorithmInterface() } class ConcreteStrategyA { AlgorithmInterface() } class ConcreteStrategyB { AlgorithmInterface() } class ConcreteStrategyC { AlgorithmInterface() } Context "1" *-- "1" Strategy : strategy Strategy < -- ConcreteStrategyA Strategy < -- ConcreteStrategyB Strategy < -- ConcreteStrategyC </pre>
Observateur (Observer)	<pre> classDiagram class Subject { + attach(o: Observer) + detach(o: Observer) + notifyObservers() } class ConcreteSubject { - subjectState + getState() + setState() } class Observer { << interface >> + Update() } class ConcreteObserver { - observerState + Update() } Subject "1" *-- "*" Observer : observers Subject < -- ConcreteSubject Observer < .. ConcreteObserver ConcreteSubject ..> ConcreteObserver : subject Note for Subject "for all o in observers { o.update(); }" Note for ConcreteObserver "observerState = subject.getState()" </pre>
Itérateur (Iterator)	<pre> classDiagram class Aggregate { CreateIterator() } class ConcreteAggregate { CreateIterator() } class Iterator { First() Next() IsDone() CurrentItem() } class ConcreteIterator { } class Client { } Aggregate < -- ConcreteAggregate Iterator < .. ConcreteIterator ConcreteAggregate ..> ConcreteIterator : return new ConcreteIterator(this) Client --> Aggregate Client --> Iterator </pre>
Décorateur (Decorator)	<pre> classDiagram class Component { operation() } class ConcreteComponent { operation() } class Decorator { operation() } class ConcreteDecoratorA { addedState operation() } class ConcreteDecoratorB { operation() addedBehavior() } Component < -- ConcreteComponent Component < -- Decorator Decorator < -- ConcreteDecoratorA Decorator < -- ConcreteDecoratorB Decorator "1" *-- "1" Component : component Decorator ..> Component : component -> operation(); ConcreteDecoratorA ..> Decorator : Decorator -> operation(); addedBehavior(); ConcreteDecoratorB ..> Decorator : Decorator -> operation(); addedBehavior(); </pre>

Nom du patron	Structure générique
Methode Template (Template Method)	<pre> classDiagram class AbstractClass { TemplateMethod() PrimitiveOperation1() PrimitiveOperation2() } class ConcreteClass { PrimitiveOperation1() PrimitiveOperation2() } AbstractClass < -- ConcreteClass </pre>
Composite	<pre> classDiagram class Client class Component { operation() } class Leaf { operation() } class Composite { operation() add(Component) remove(Component) getChild(int): Component } Client --> Component Component < -- Leaf Component < -- Composite Composite o--> Component : children Composite --> Component : forall component in children component.operation(); </pre>
Commande (Command)	<pre> classDiagram class Client class Command { <<interface>> execute() } class ConcreteCommand { state execute() } Client ..> Command Command < .. ConcreteCommand </pre>
Méthode Fabrique (Factory)	<pre> classDiagram class Creator { <<interface>> factoryMethod() } class Product { <<interface>> } class ConcreteCreator { } class ConcreteProduct { } Creator < .. ConcreteCreator Product < .. ConcreteProduct ConcreteCreator ..> ConcreteProduct </pre>
Memento (Snapshot)	<pre> classDiagram class Originator { SetMemento(Memento m) CreateMemento() state } class Memento { GetState() SetState() state } class Caretaker { } Originator ..> Memento Caretaker o--> Memento : memento </pre>

Nom du patron	Structure générique
Singleton	
Adaptateur (Adapter)	<pre>graph TD Client --> Target Adapter -- > Target Adapter o-- Adaptee Adapter -.-> Note[Calls adapteeMethod()]</pre>
Médiateur (Mediator)	
Chaîne de responsabilité (Chain of Responsibility)	

Nom du patron	Structure générique
Façade	
Visiteur (Visitor)	
Proxy	
Prototype	