GAM250: Advanced Games Programming

2: Design Patterns

Learning outcomes

- **Describe** the concept of Design Patterns
- **Understand** some of the classic ’Gang of Four’ Design Patterns
- **Implement** some of the most commmon design patterns

OO Design Basics

Learning Outcomes

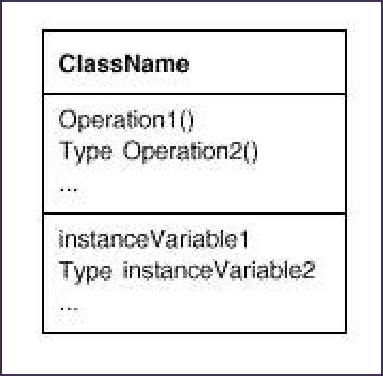
In this section you will learn how to...

- **Illustrate** the role of UML in communicating software design
- **Explain** basic OO design principles, including abstraction and polymorphism
- **Explain** the role of design patterns in object-orientated software design
- **Identify** the key components of a pattern

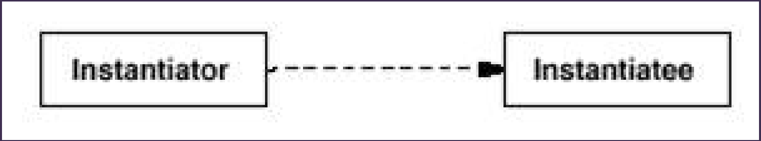
Object Modelling Techniques

- Used to describe patterns in the GO4 book
- Uses UML to graphical represent different OO relationships:
  - **class diagrams**: show the static relationship between classes
  - **object diagrams**: show the state of a program as a series of related objects
  - **interaction diagrams**: illustrate execution of the program as an interaction among related objects

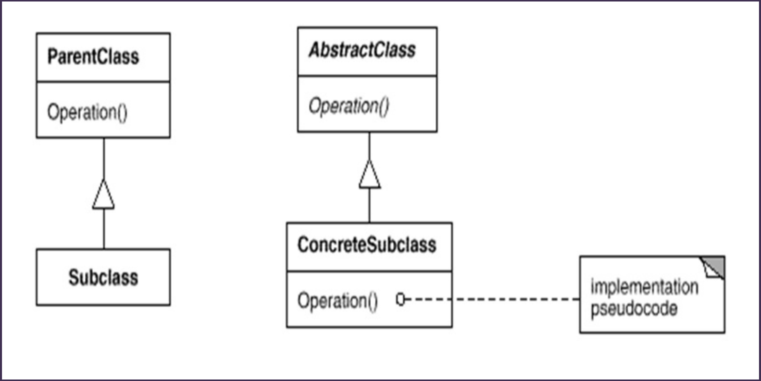
Classes


ClassName
Operation1()
Type Operation2()
...
instanceVariable1
Type instanceVariable2
...

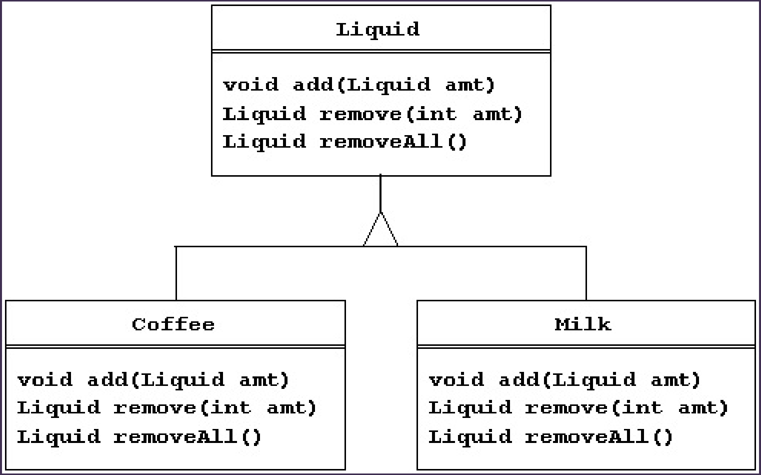
Object Instantiation



Subclassing and Abstract Classes

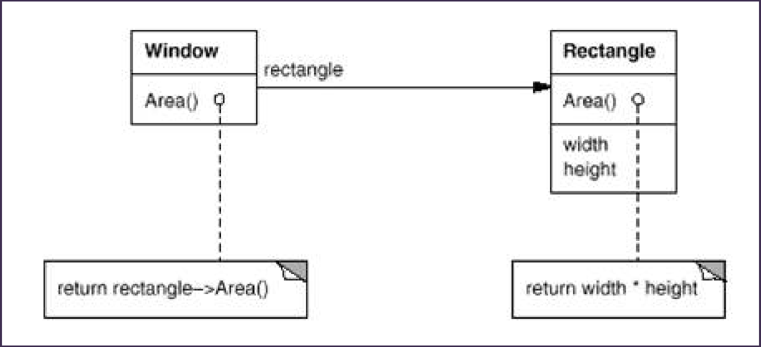


Abstraction and Polymorphism

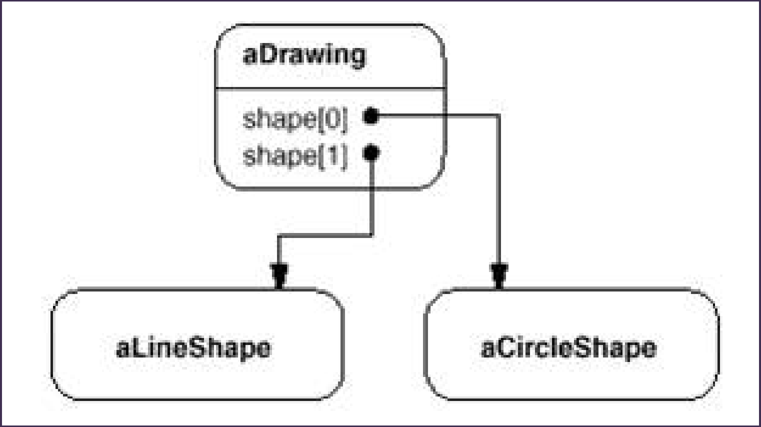


FALMOUTH UNIVERSITY

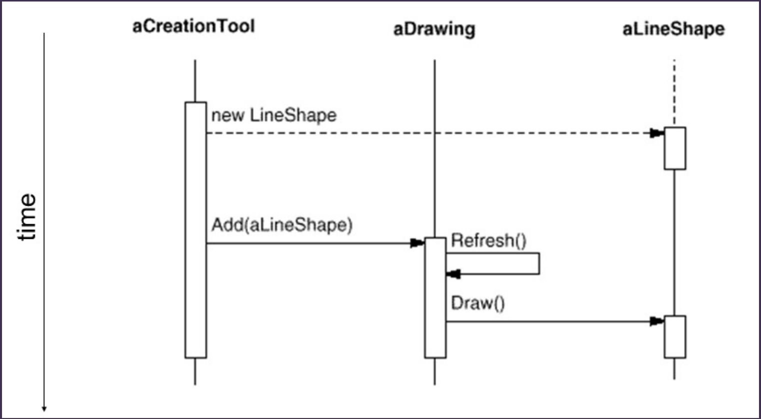
Pseudo-code and Containing



Object Diagrams



Interaction Diagrams



Role of Design Patterns

- OO design is more than just drawing diagrams, it is craftsmanship
- Good drafters are good designers
- OO design skill comes with deliberate practice and project experience
- A powerful form of abstraction and resuse is *design* abstraction and re-use

Role of Design Patterns

Object orientated systems tend to exhibit recurring structures that promote:

- Abstraction
- Flexibility
- Modularity
- Elegance

Role of Design Patterns

- Therein lies valuable design knowledge.
- The challenge, of course, is to...
  - capture
  - communicate
  - and apply
- ...this knowledge.

Role of Design Patterns

A design pattern...

- Abstracts a recurring design structure
- Comprises class and/or object
  - dependencies
  - structures
  - interactions
  - conventions
- names and specifies the design structure explicitly
- and thereby distils design experience

Components of a Design Pattern

A design pattern is comprised of:

- A name
- Common aliases — *also known as...*
- Real-world examples
- Contexts
- Common problems solved
- Solution
- Structure
- Diagrams
- Consequences

Components of a Design Pattern

- Design patterns are often tacit knowledge made explicit.
- You will develop tacit knowledge of patterns through regular design practice.
- You are expected to engage in constant research and reflection when designing software to learn all of these different patterns.
- They will help you communicate and design in the future.
- Additional research will be required as the number of patterns greatly exceeds those that can be covered in workshops.

FALMOUTH UNIVERSITY

Design Patterns

Learning Outcomes

In this section you will learn how to...

- **Distinguish** between creational, structural, and behavoral design patterns
- **Compare and contrast** different design patterns
- **Suggest** the most appropriate design pattern for a given context

Types of Design Pattern

Design patterns come in three main flavours:

- **creational**: concerned with the process of creating and managing the creation of objects.
- **structural**: dealing with the composition of objects.
- **behavioural**: characterizing the different means by which objects can interact with others.

Types of Design Pattern

- **Creational**
- Singleton
- Typesafe Enum
- Factory
- Prototype
- Builder

- **Structural**
- Adapter
- Bridge
- Proxy
- Facade
- Decorator

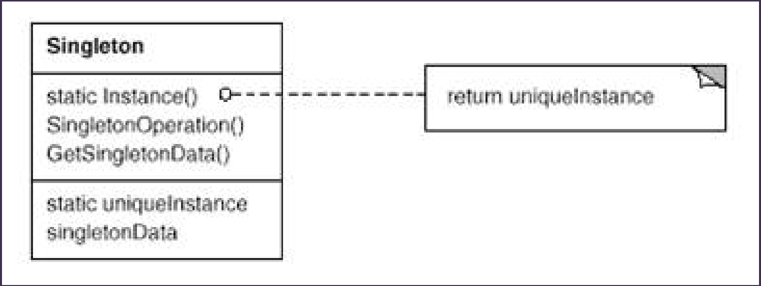
- **Behavioural**
- Template
- State
- Observer
- Visitor
- Strategy

Design Patterns

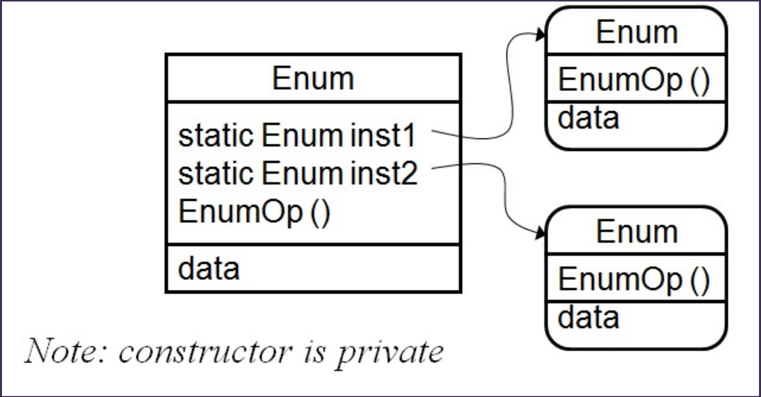
We will now briefly examine these patterns. Throughout this section...

- **Please** make notes on Slack
- **Link** to on-line resources
- **Ask** questions
- **Think** about how the patterns may apply to your own projects
- **Conduct** further research

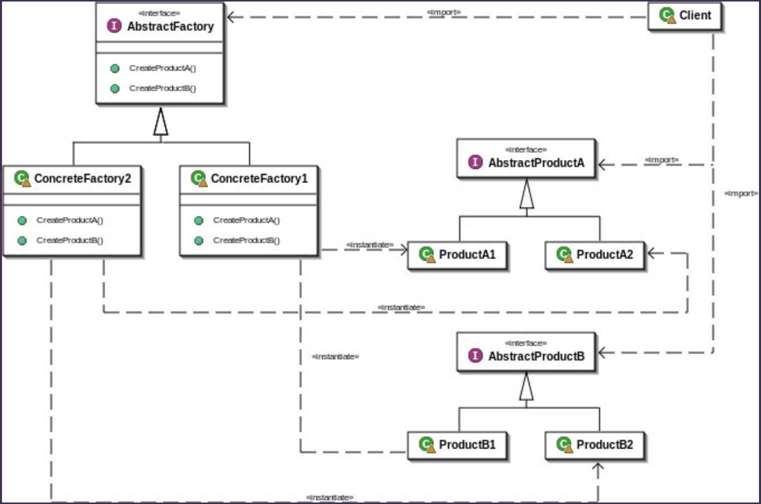
Singleton


Singleton
static Instance()
SingletonOperation()
GetSingletonData()
static uniqueInstance
singletonData
return uniqueInstance

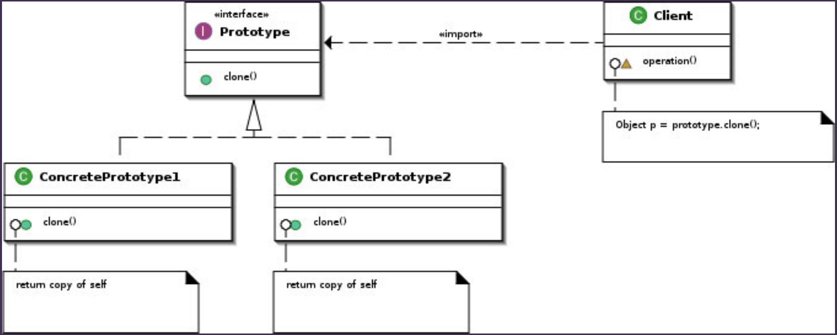
Typesafe Enum



Abstract Factory

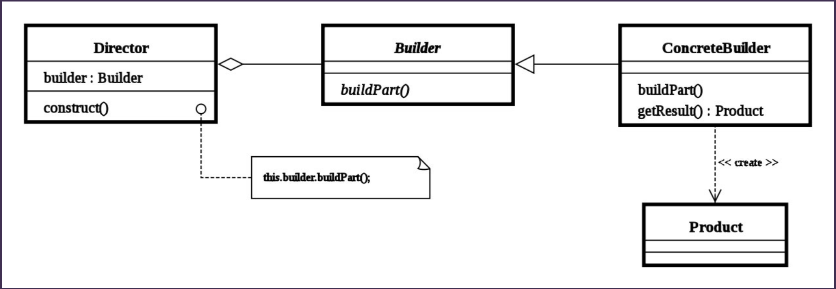


Prototype

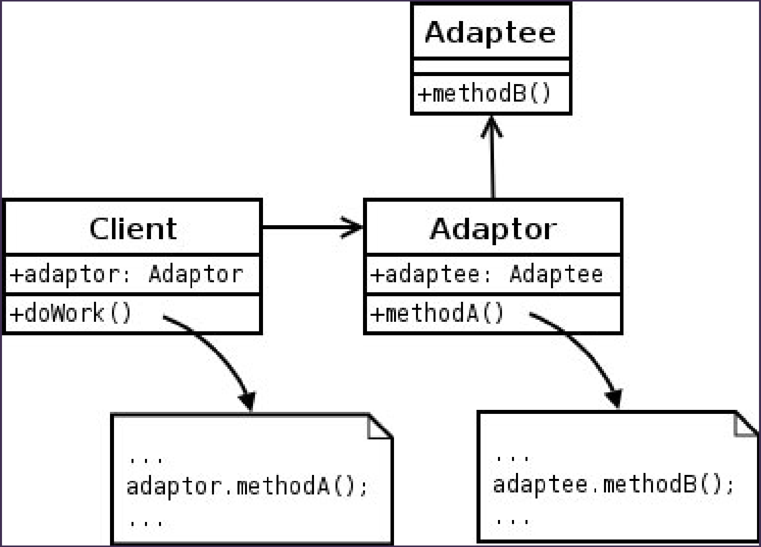


«interface»
Prototype
clone()
«import»
Client
operation()
Object p = prototype.clone();
ConcretePrototype1
clone()
return copy of self
ConcretePrototype2
clone()
return copy of self

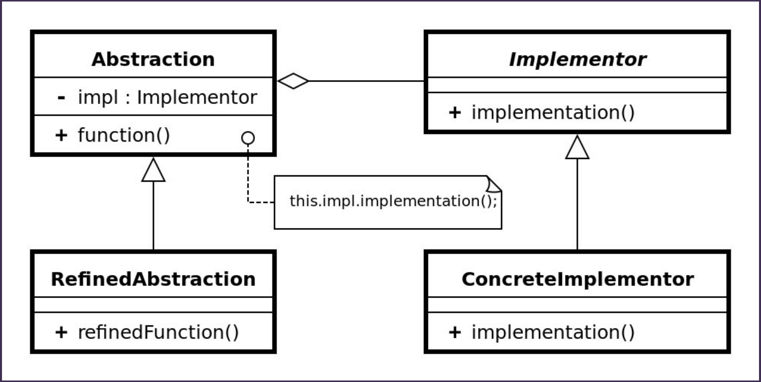
Builder


Director
builder : Builder
construct()
Builder
buildPart()
ConcreteBuilder
buildPart()
getResult() : Product
this.builder.buildPart();
<< create >>
Product

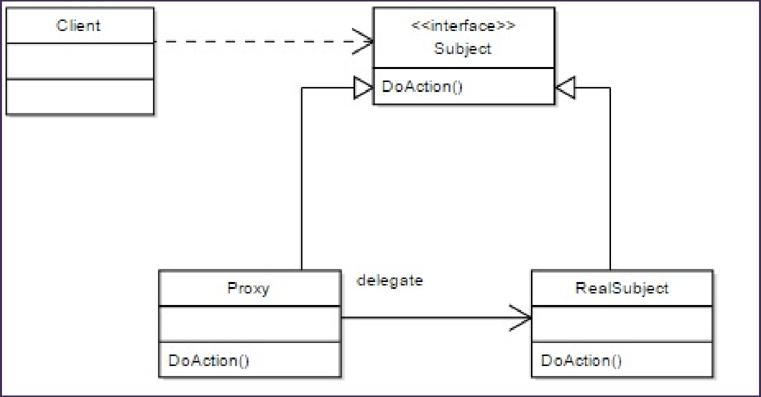
Adapter


Adaptee
+methodB()
Client
+adaptor: Adaptor
+doWork()
Adaptor
+adaptee: Adaptee
+methodA()
...
adaptor.methodA();
...
...
adaptee.methodB();
...

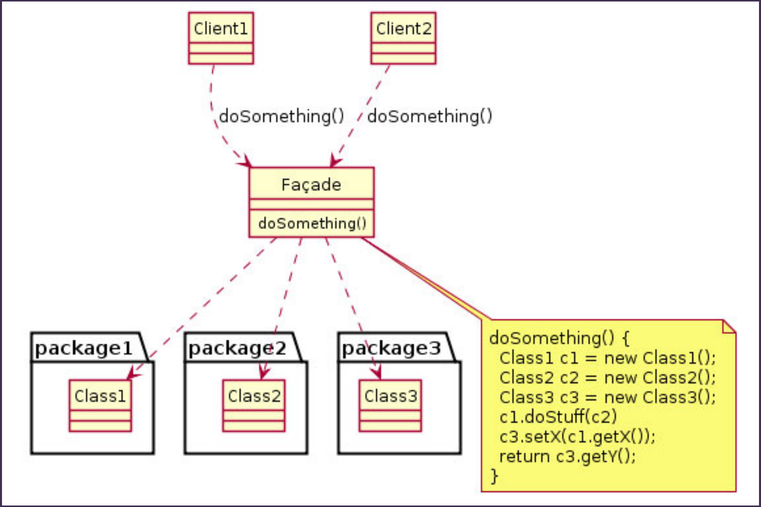
Bridge


Abstraction
- impl : Implementor
+ function()
Implementor
+ implementation()
this.impl.implementation();
RefinedAbstraction
+ refinedFunction()
ConcreteImplementor
+ implementation()

Proxy

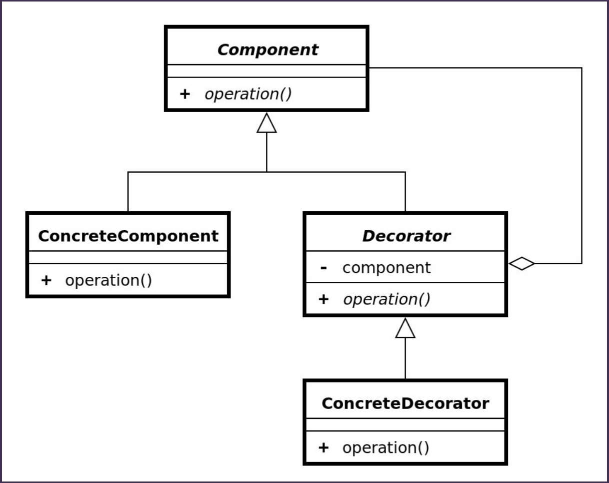

Client
<<interface>>
Subject
DoAction()
Proxy
DoAction()
delegate
RealSubject
DoAction()

Facade



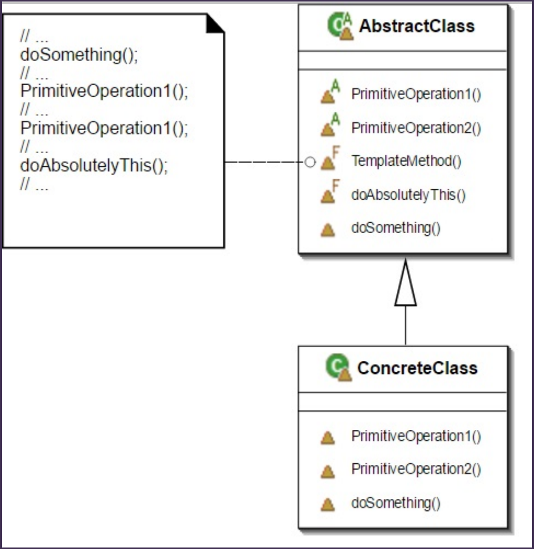
```
doSomething() {
  Class1 c1 = new Class1();
  Class2 c2 = new Class2();
  Class3 c3 = new Class3();
  c1.doStuff(c2)
  c3.setX(c1.getX());
  return c3.getY();
}
```

Decorator

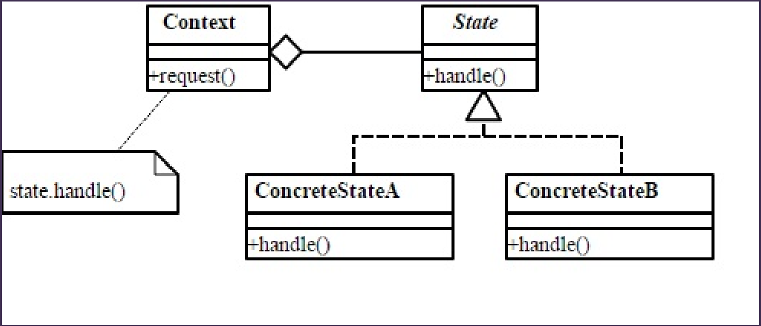


Component
+ *operation()*
ConcreteComponent
+ operation()
Decorator
- component
+ *operation()*
ConcreteDecorator
+ operation()

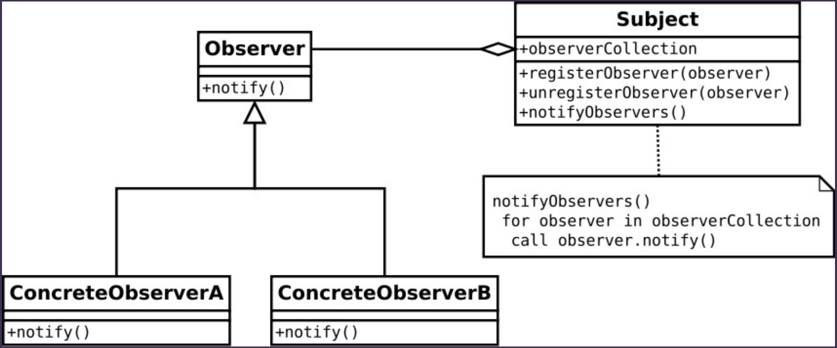
Template


```
// ...
doSomething();
// ...
PrimitiveOperation1();
// ...
PrimitiveOperation1();
// ...
doAbsolutelyThis();
// ...
```

State

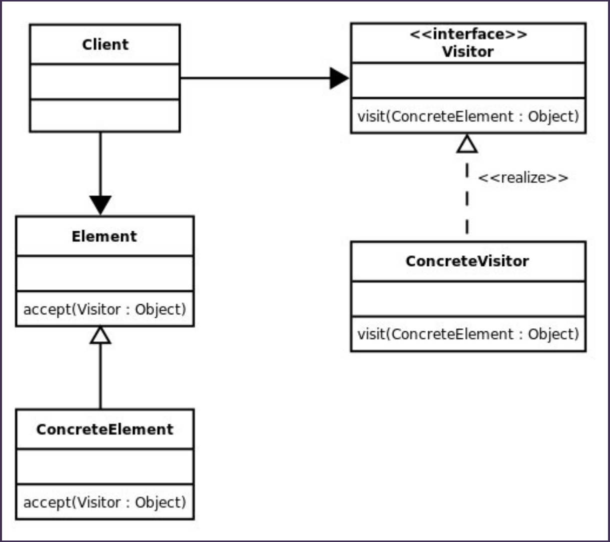

Context
+request()
State
+handle()
state.handle()
ConcreteStateA
+handle()
ConcreteStateB
+handle()

Observer



```
notifyObservers()
 for observer in observerCollection
  call observer.notify()
```

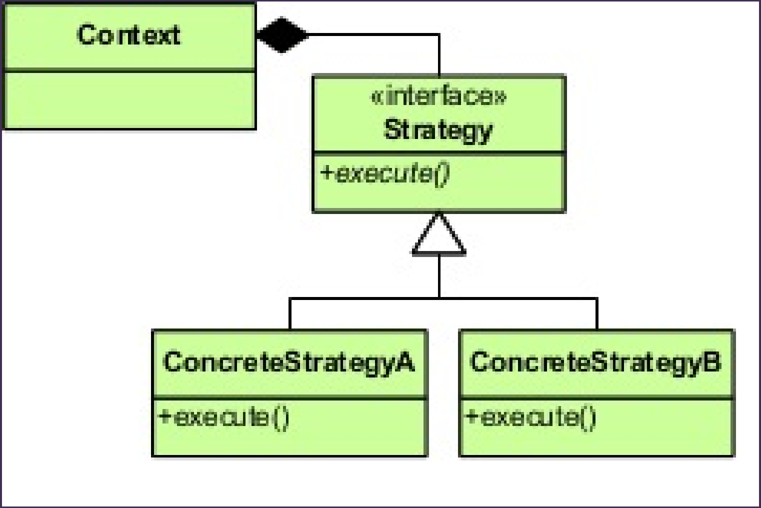
Visitor



Client
<<interface>>
Visitor
visit(ConcreteElement : Object)
<<realize>>
ConcreteVisitor
visit(ConcreteElement : Object)
Element
accept(Visitor : Object)
ConcreteElement
accept(Visitor : Object)

FALMOUTH UNIVERSITY

Strategy


Context
«interface»
Strategy
+execute()
ConcreteStrategyA
+execute()
ConcreteStrategyB
+execute()