Design Pattern

Milad Abolhassani

tuxgeek.ir
milad@tuxgeek.ir

Slides:
http://slideshare.com/miladas
http://miladas.github.io/slides

Attention


EVERY TIME YOU WRITE CRAPPY CODE
GOD KILLS A KITTEN
memegenerator.net

OOP

- Class
- Property
- Method
- Inheritance
- Polymorphism
- Interface
- Abstract class
- Encapsulation
- Public/protected/private
- Final

Where did all of these come from?

SOLID

SOLID

SOLID

- Single Responsibility
- Open/Closed
- Liskov substitution
- Interface Segregation
- Dependency Injection

Single Responsibility

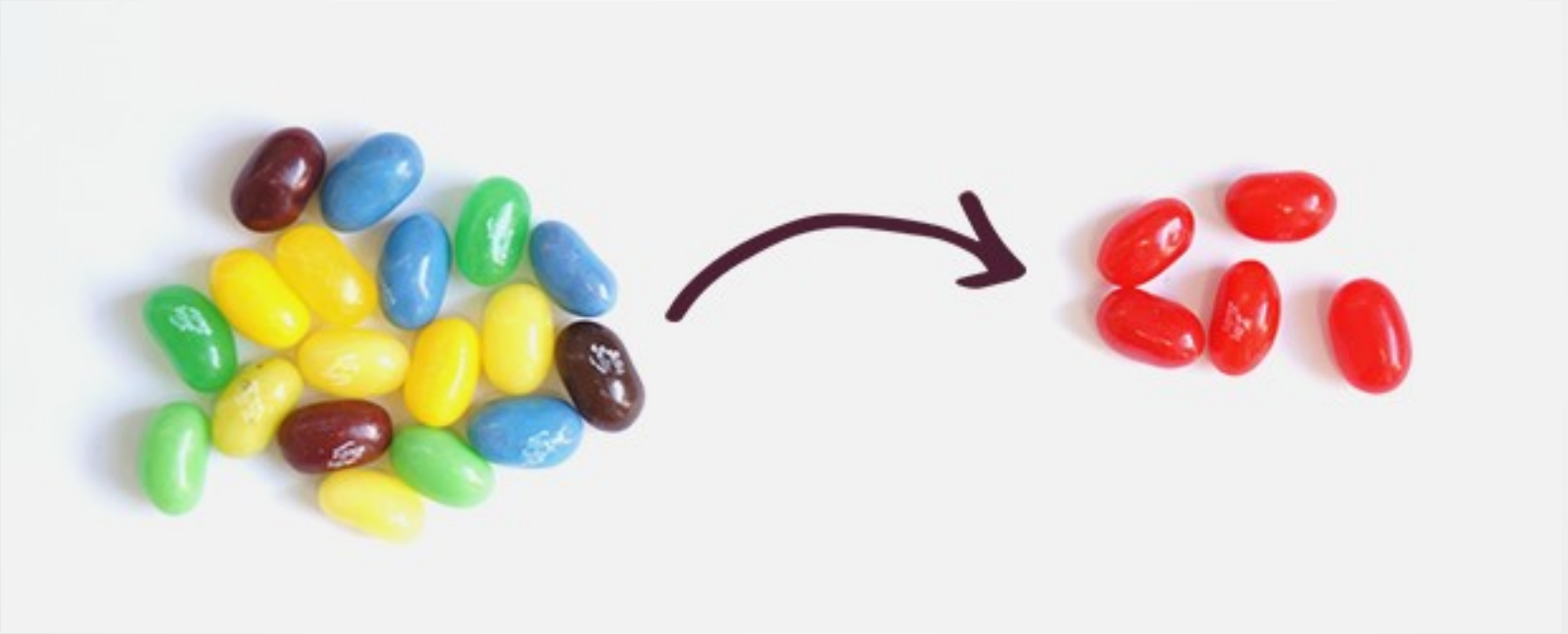
Open/Closed

- Meyer's open/closed
  - Once class completed, never should modified
- Polymorphic open/closed principle
  - Interface (public methods)
  - The interface is open for extension but close for modification

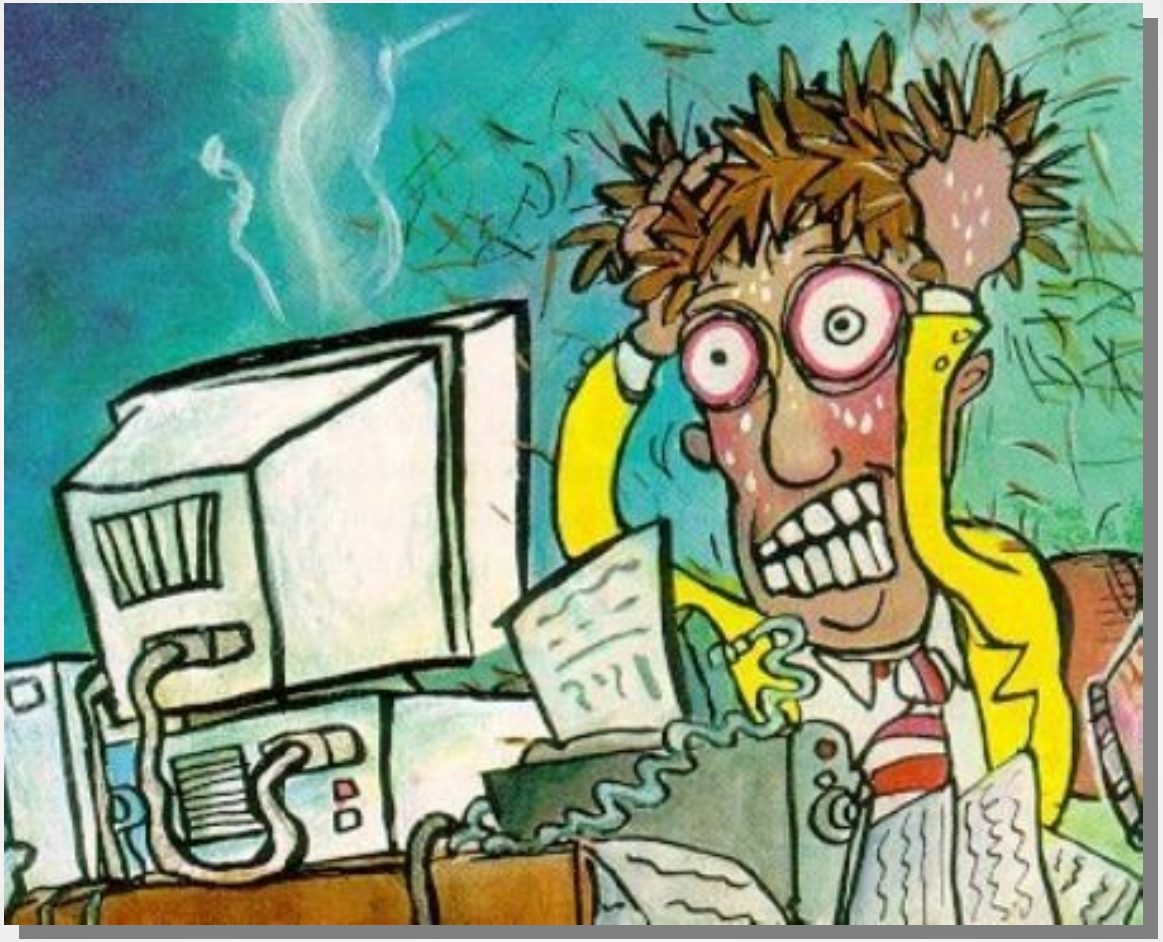
Liskov substitution

- Program to an interface rather than of it’s implementation
- Design by contract
- Do not extend the interface

Interface Segregation



Dependency injection



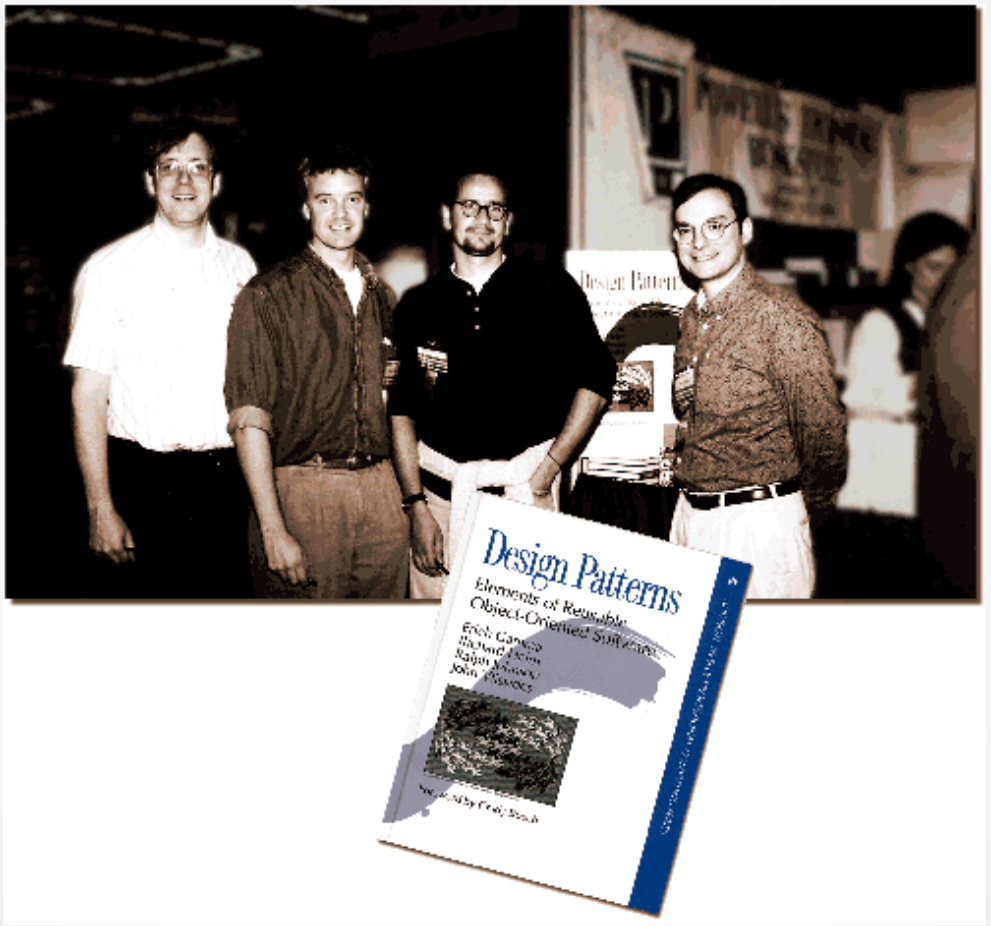
What is design pattern?

- In software engineering, a design pattern is a general reusable solution to a commonly occurring problem.

History

- Patterns originated as an architectural concept by Christopher Alexander (1977/79).
- Design patterns gained popularity in computer science after the book Design Patterns: Elements of Reusable Object-Oriented Software was published in 1994.
  - Gang of Four
  - Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John

Gang of four


Design Patterns
Elements of Reusable
Object-Oriented Software

Why design pattern?

- Clarity
- Correctness
- Same vocabularies
  - Avoid miss communications
- Reuse
- Save time, trial and error
- Can speed up the development process
  - by providing tested, proven development paradigms
- We are not reusing code, we are reusing experience

Category

- Creational design patterns
- Structural design patterns
- Behavioral design patterns

Creational design patterns

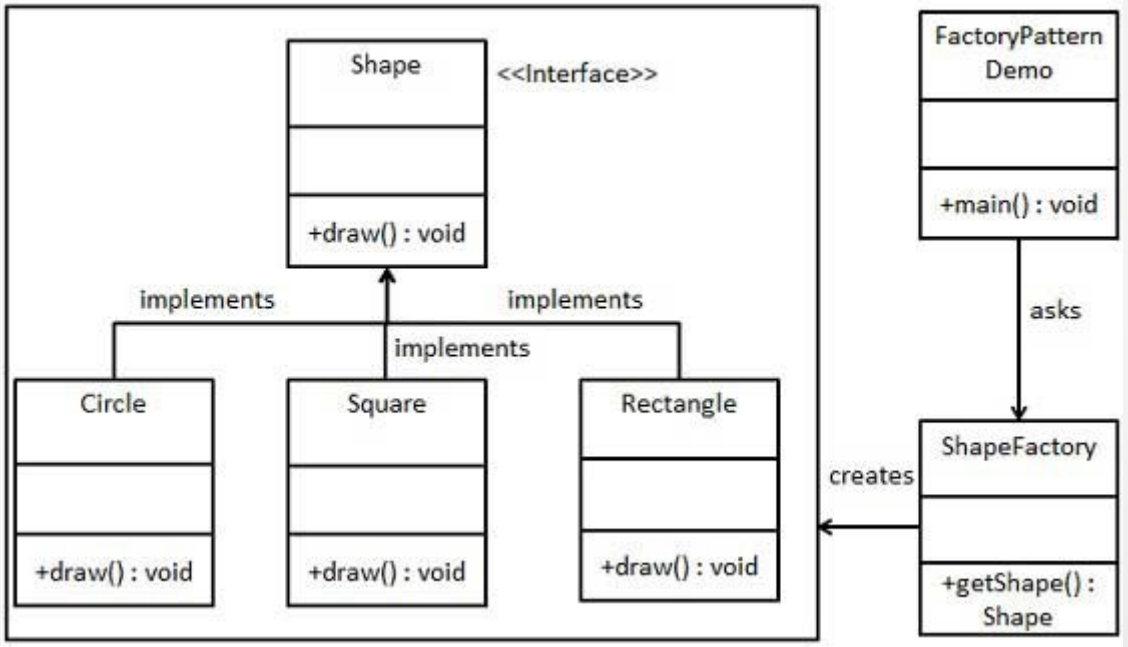
- These design patterns provide a way to create objects while hiding the creation logic.

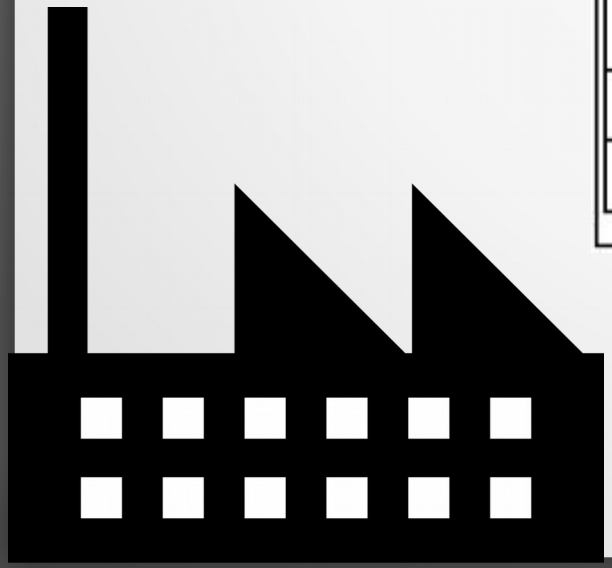
Creational design patterns

- Factory method
- Singleton
- Prototype
- Builder

Factory Method

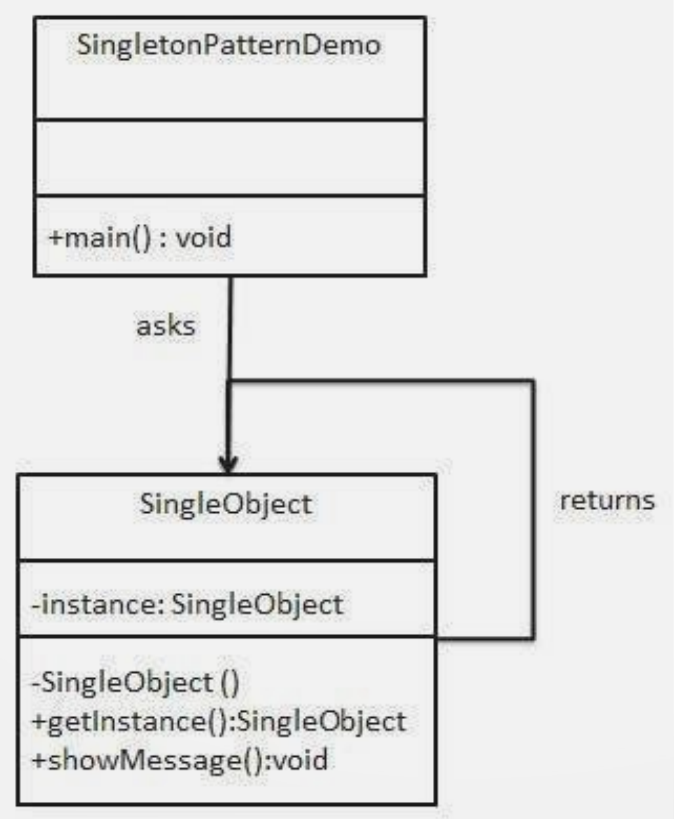
- Factory Method lets a class defer instantiation to subclasses.





Singleton

- This pattern involves a single class which is responsible to create an object while making sure that only single object gets created.

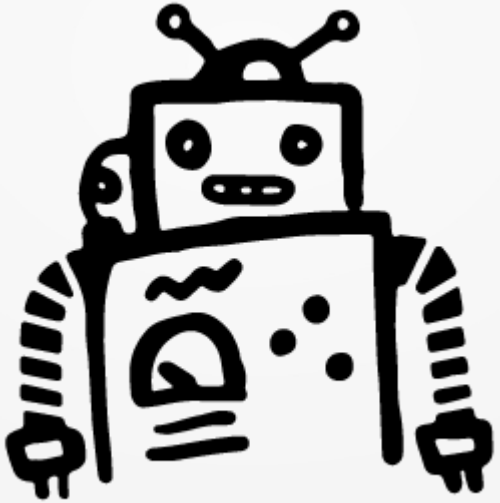


Prototype

- Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

Builder

- Separate the construction of a complex object from its representation so that the same construction process can create different representations.



Programmer

Structural design patterns

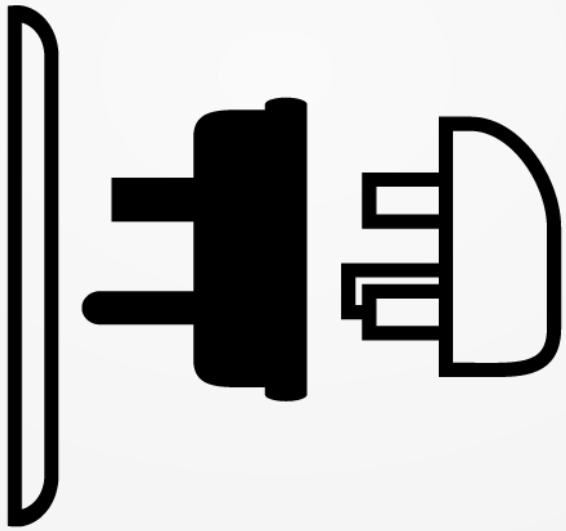
- These design patterns are all about Class and Object composition. Concept of inheritance is used to compose interfaces and define ways to compose objects to obtain new functionalities.

Structural design patterns

- Adapter
- Decorator
- Facade
- Flyweight
- Proxy

Adapter

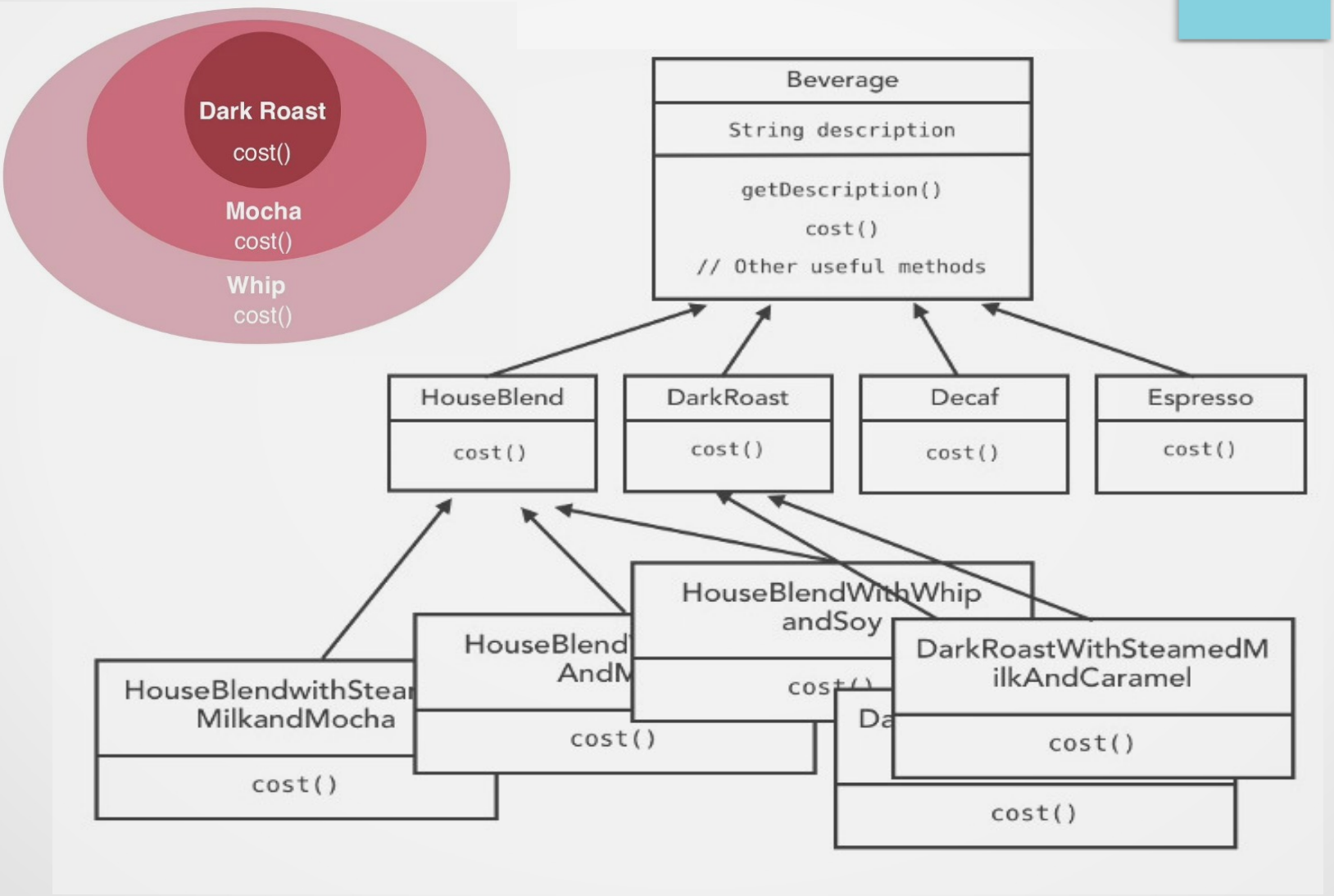
- Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.



Decorator

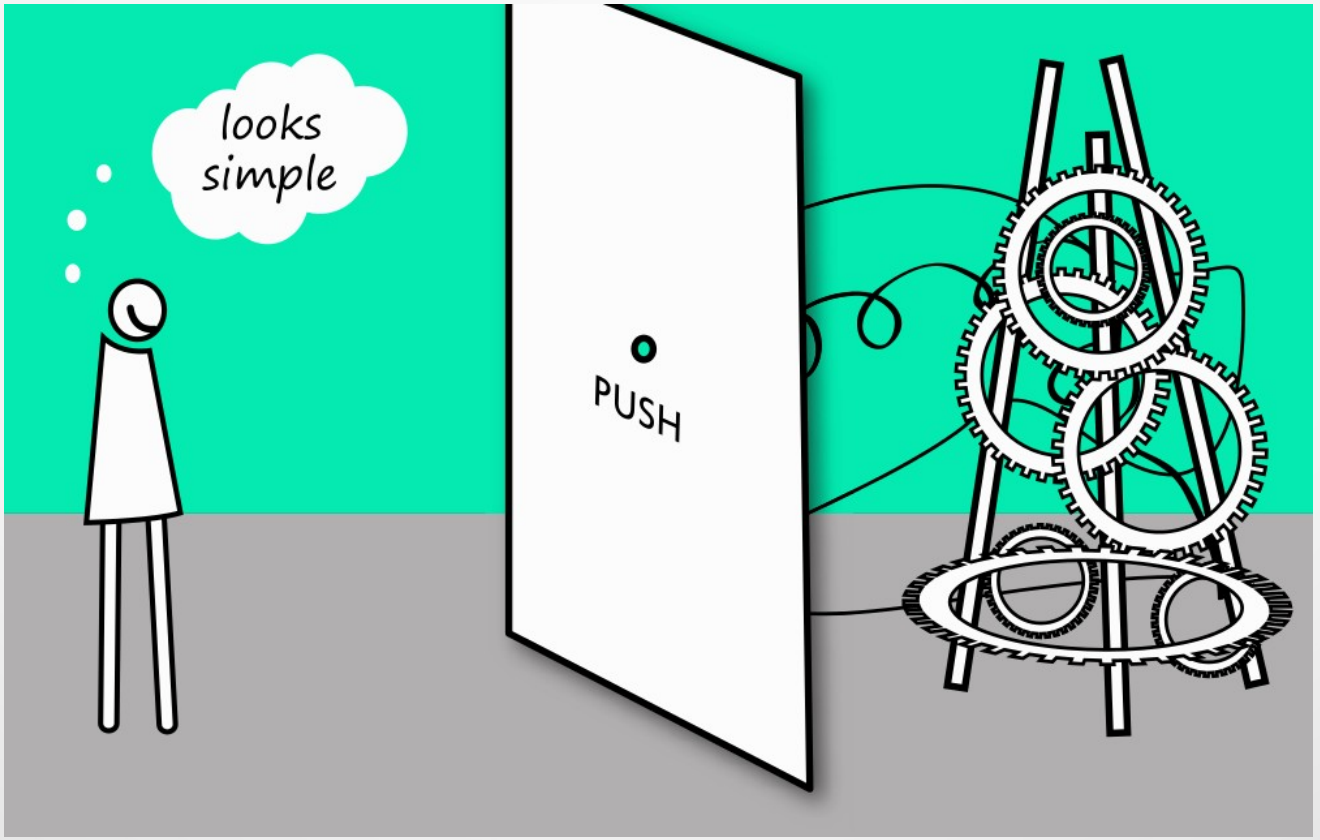
- Attach additional responsibilities to an object dynamically.

Decorator


Dark Roast
cost()
Mocha
cost()
Whip
cost()
Beverage
String description
getDescription()
cost()
// Other useful methods
HouseBlend
cost()
DarkRoast
cost()
Decaf
cost()
Espresso
cost()
HouseBlendWithWhip
andSoy
DarkRoastWithSteamedM
ilkAndCaramel
cost()
cost()
cost()
cost()

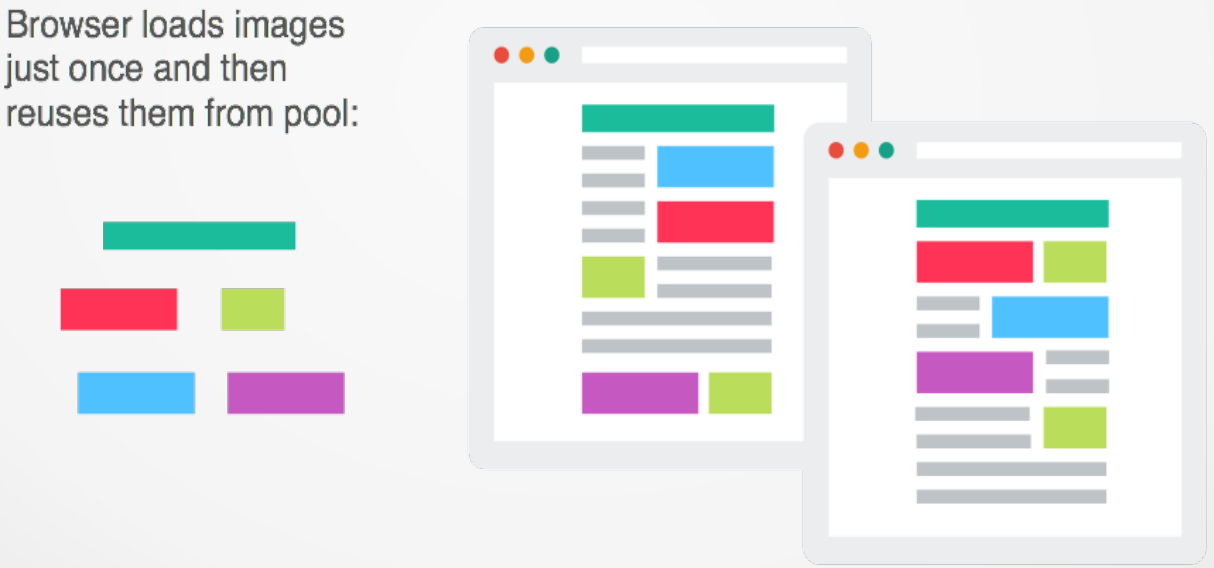
Facade

- Wrap a complicated subsystem with a simpler interface.



Flyweight

- The Flyweight uses sharing to support large numbers of objects efficiently.



Proxy

- Use an extra level of indirection to support distributed, controlled, or intelligent access.

Behavioral design patterns

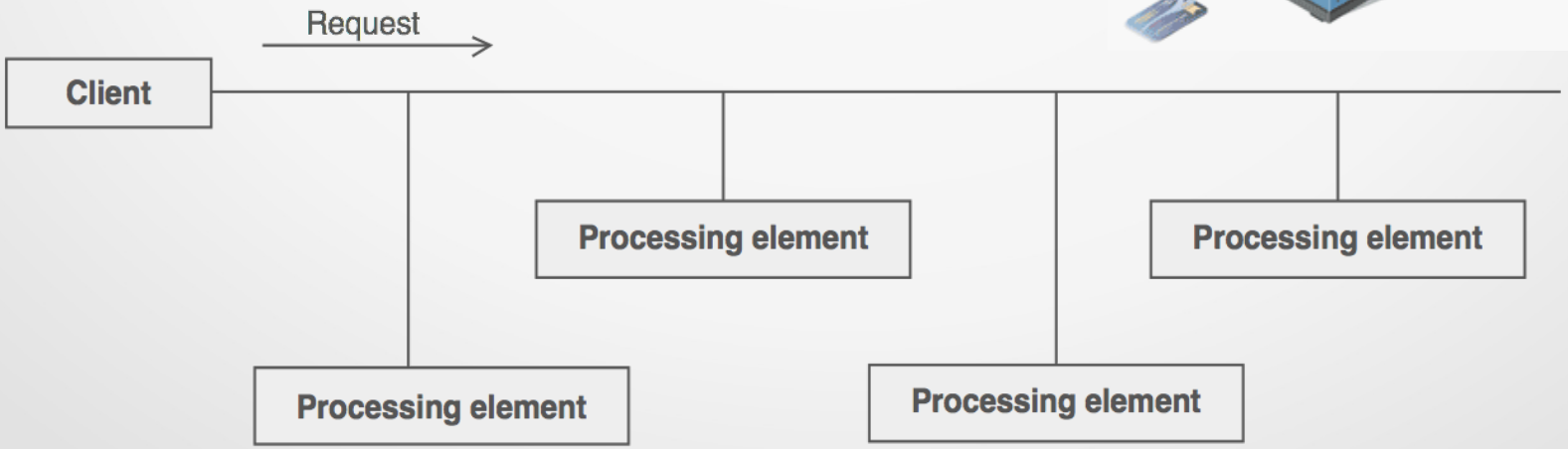
- These design patterns are all about Class's objects communication. Behavioral patterns are those patterns that are most specifically concerned with communication between objects.

Behavioral design patterns

- Chain of responsibility
- Mediator
- Observer
- State
- Strategy
- Template method

Chain of responsibility

- The Chain of Responsibility pattern avoids coupling the sender of a request to the receiver by giving more than one object a chance to handle the request.



Mediator

- The Mediator defines an object that controls how a set of objects interact. Loose coupling between colleague objects is achieved by having colleagues communicate with the Mediator, rather than with each other.

Observer

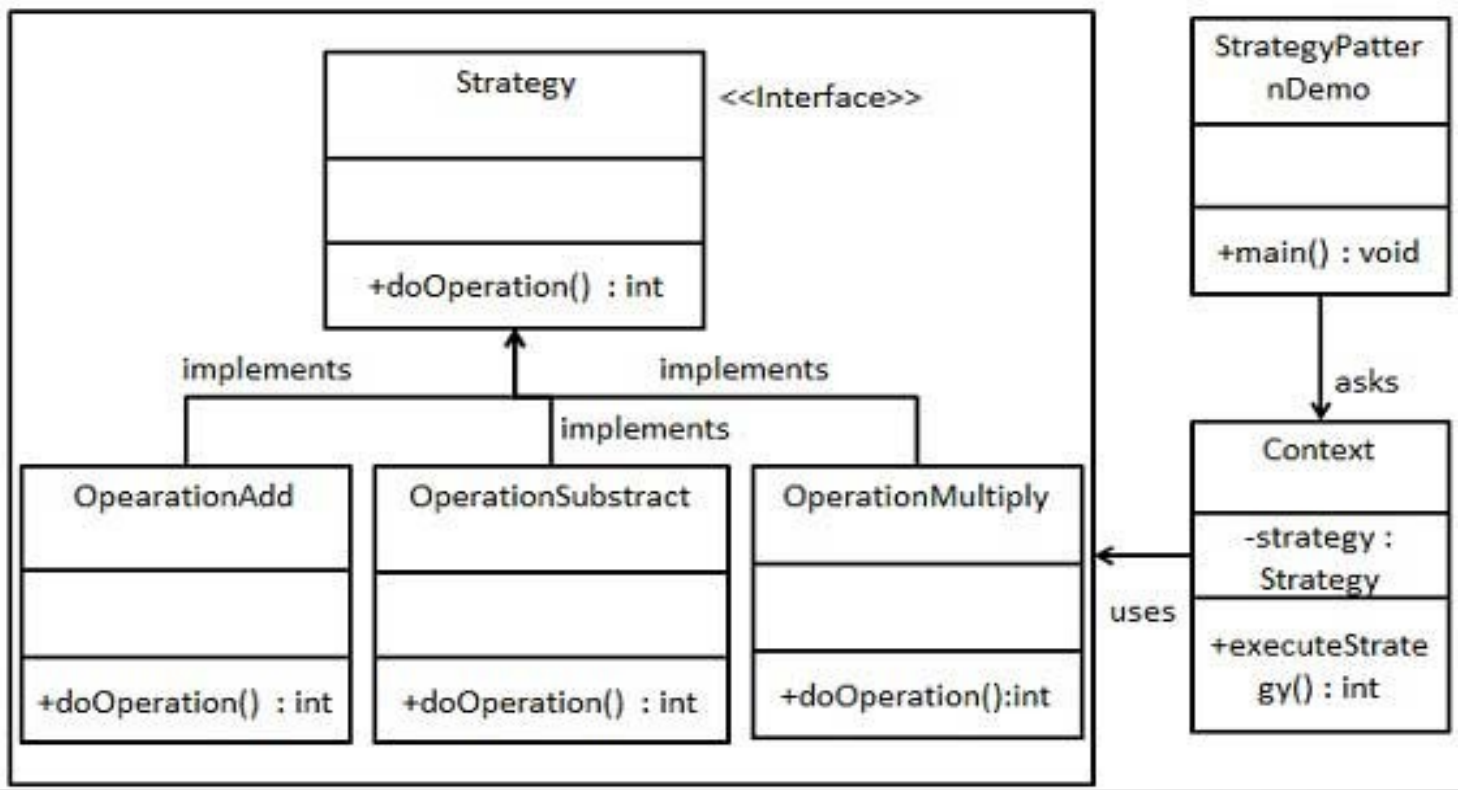
- Observer pattern is used when there is one-to-many relationship between objects such as if one object is modified, its depenedent objects are to be notified automatically.

State

- The State pattern allows an object to change its behavior when its internal state changes.

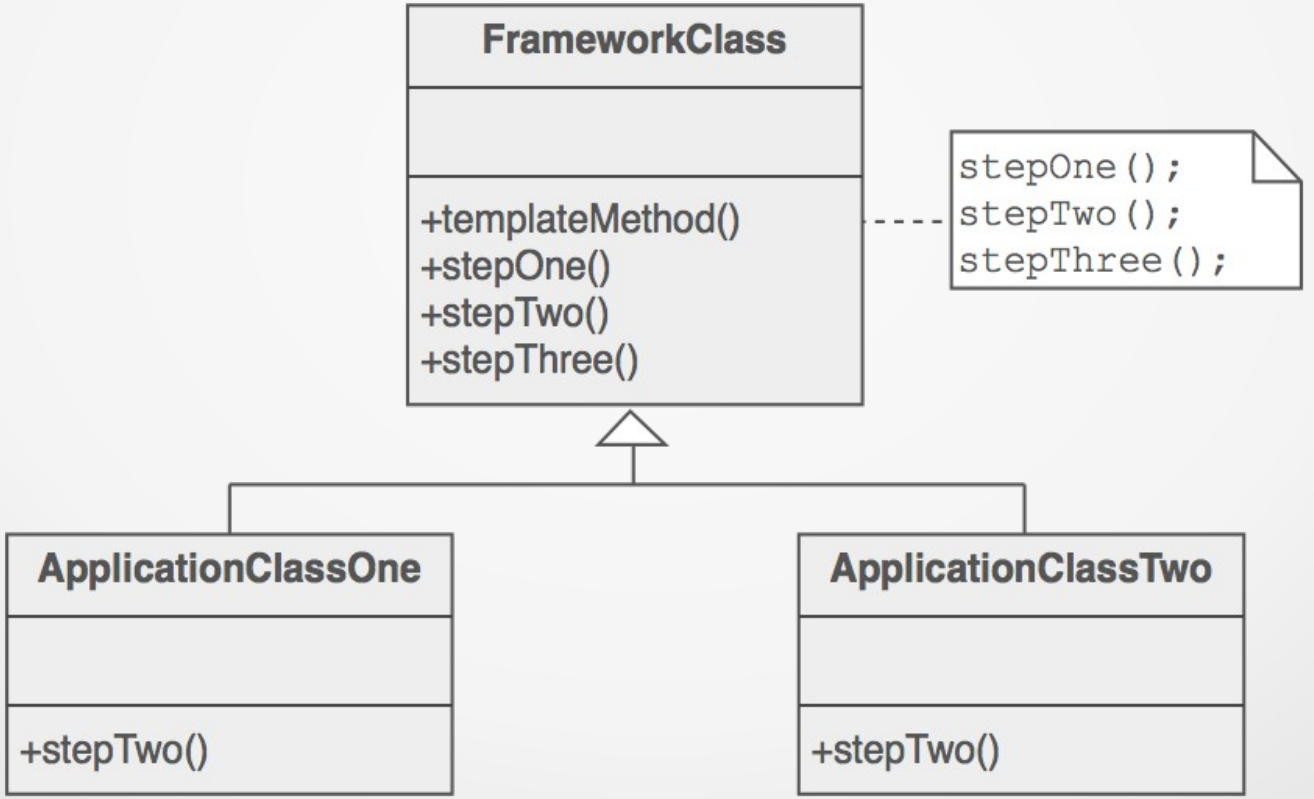
Strategy

- In Strategy pattern, a class behavior or its algorithm can be changed at run time.

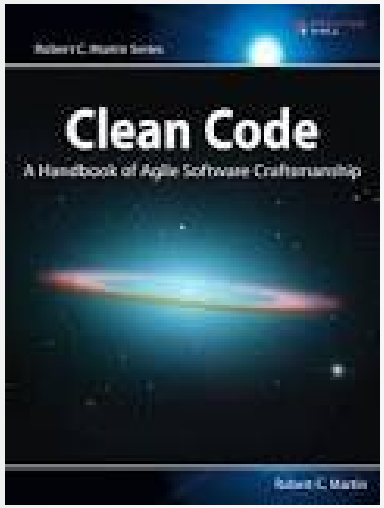


Template method

- The Template Method defines a skeleton of an algorithm in an operation, and defers some steps to subclasses.

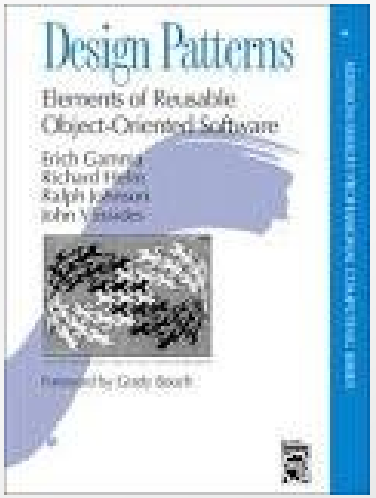


Thank you


Clean Code
A Handbook of Agile Software Craftsmanship


DESIGN PATTERNS
Explained simply


CODE
COMPLETE
2
Steve McConnell


Design Patterns
Elements of Reusable
Object-Oriented Software

References

- JavaWorld
- SourceMaking
- TutorialsPoint
- Informit
- geekswitblogs
- Wikipedia [1] [2]