OOP Design Patterns

Introduction to common OOP Patterns

Outline

- What are Design Patterns?
- Creational Design Patterns
  - ➢ Object Creation
- Behavioral Design Patterns
  - ➢ Relationship between Entities
- Structural Design Patterns
  - ➢ Communication between Objects

Motivation

- Can you imagine how many lines of code is Facebook?
- Most software systems contain certain common aspects that are frequently reinvented for each system.
- Solutions to these common problems may vary in quality from system to system.
- Design patterns seeks to communicate these classic solutions in an easy to understand manner.

What are Design Patterns?

“In software engineering, a design pattern is a general reusable solution to a commonly occurring problem within a given context in software design. A design pattern is not a finished design that can be transformed directly into source or machine code. It is a description or template for how to solve a problem that can be used in many different situations.” --Wikipedia

DRY

Do Not Repeat Yourself !!

Most of Design Patterns try to follow the DRY principle, to avoid repeating code as much as possible. Repeating similar code increases redundancy and coding time!

Creational Design Patterns

- Singleton Design Pattern
- Factory Design Pattern

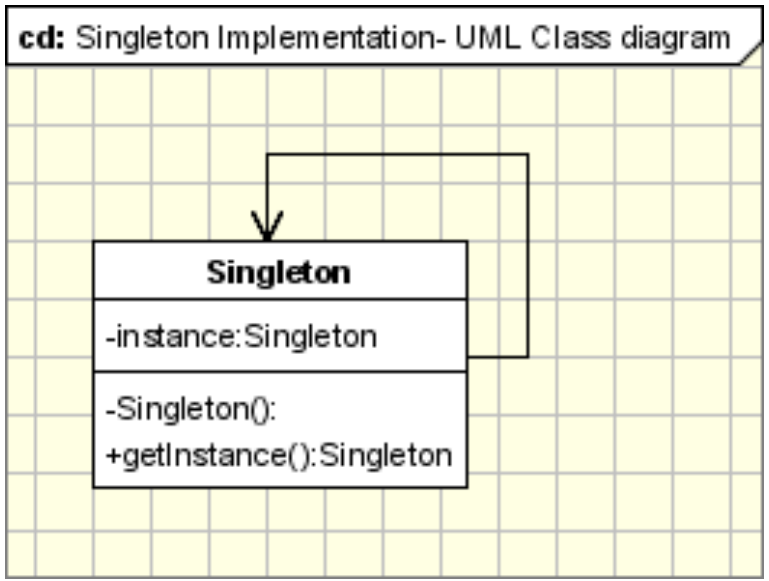
Singleton Design Pattern

- Singleton Pattern ensures that the class has at most one instance and that other classes have global access to it.

- Logging is one example that utilizes the Singleton Pattern. Only one instance of the Logger class is needed in a project and you need a global access to it everywhere in your project.

Singleton Design Pattern

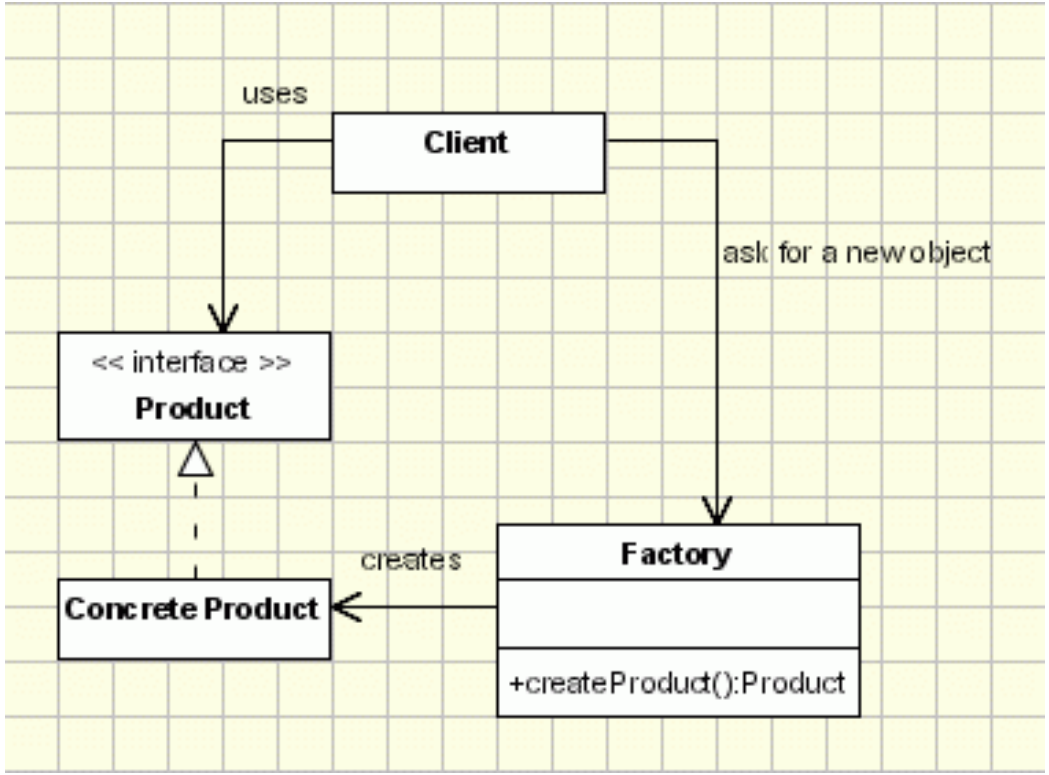
```
public class Singleton{
        private static Singleton instance;
        private Singleton(){
                ...
        }
        public static Singleton getInstance(){
                if (instance == null)
                        instance = new Singleton();
                return instance;
        }
        ...
        public void doSomething(){
                ...
        }
}
```



Factory Design Pattern

- Factory Pattern is useful when a class constructor needs a lot of processing or logic before passing it it’s arguments.
- Factory Pattern creates objects without exposing the instantiation logic to the client.
- Refers to the newly created object through a common interface.

Factory Design Pattern



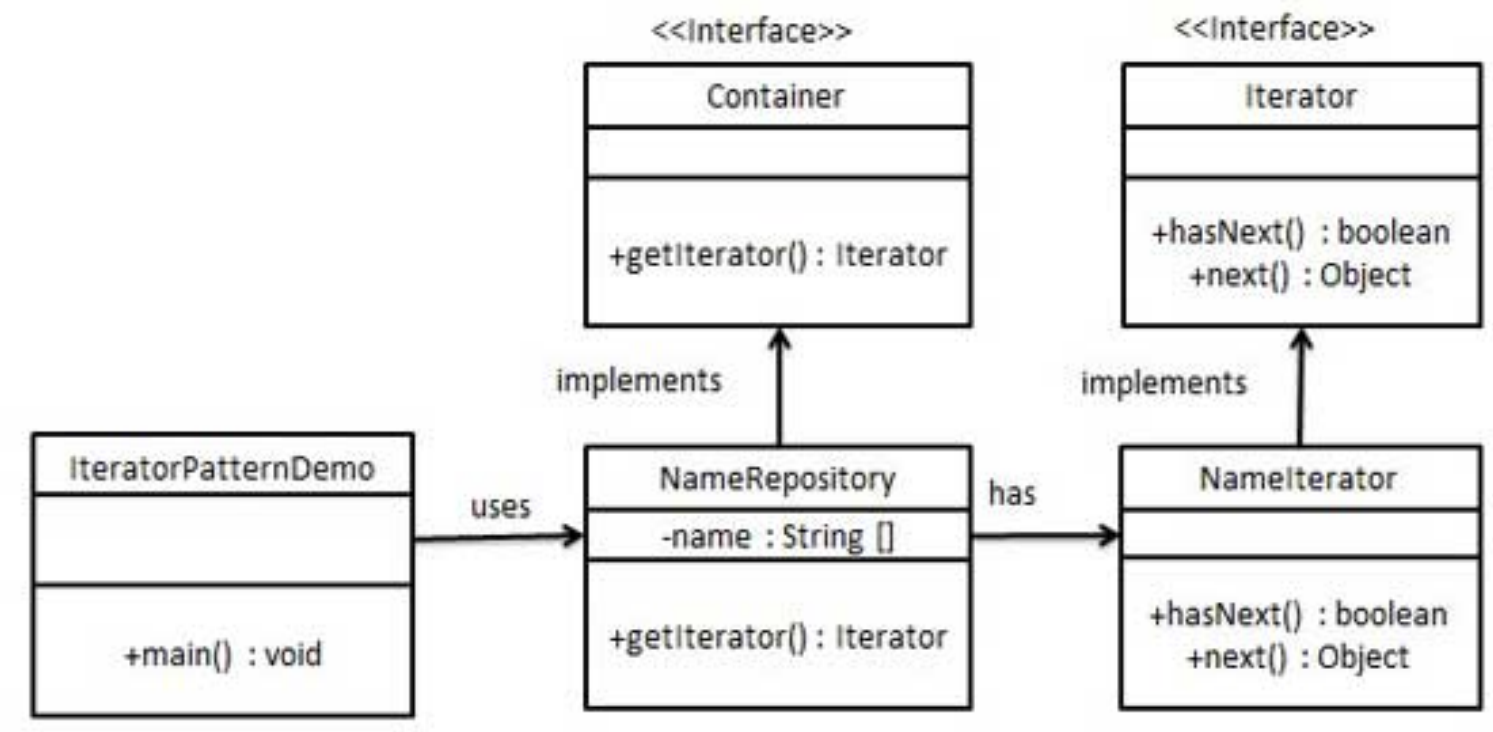
Behavioral Design Patterns

- Iterator Design Pattern
- Template Design Pattern

Iterator Design Pattern

- A collection is just a grouping of some objects, a data structure. It can be a list, an array, a tree and the examples can continue.
- A collection should provide a way to access its elements without exposing its internal structure. It have a mechanism to traverse in the same way a list or an array. It doesn't matter how they are internally represented.
- The Iterator pattern takes the responsibility of accessing and passing through the objects of the collection and put it in the iterator object. The iterator object will maintain the state of the iteration, keeping track of the current item and having a way of identifying what elements are next to be iterated.

Iterator Design Pattern

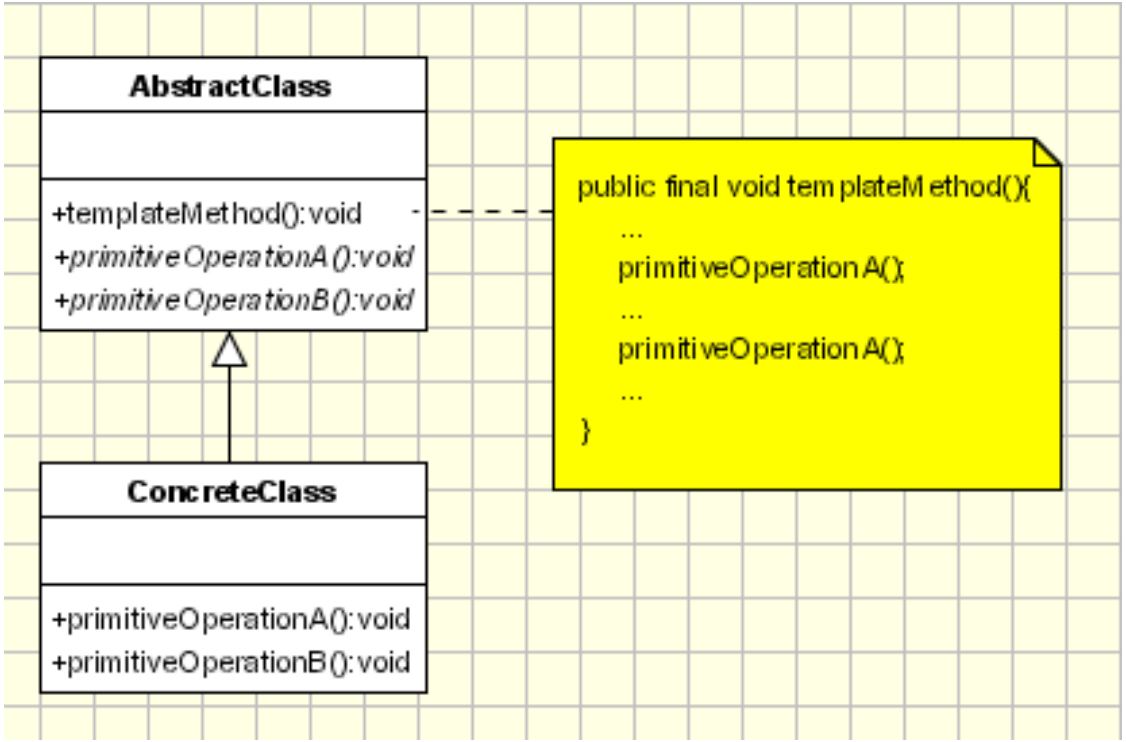


Template Design Pattern

A template method defines an algorithm in a base class using abstract operations that subclasses override to provide concrete behavior without changing the algorithm itself.

```
public abstract class CaffeineBeverage {

    void final prepareRecipe() {
        boilWater();

        brew();

        pourInCup();

        addCondiments();
    }

    abstract void brew();

    abstract void addCondiments();

    void boilWater(){
        // implementation
    }

    void pourInCup(){
        // implementation
    }
}
```

Template Design Pattern



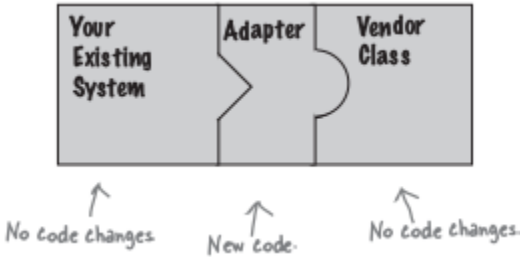
Structural Design Patterns

- Adapter Design Pattern
- Decorator Design Pattern

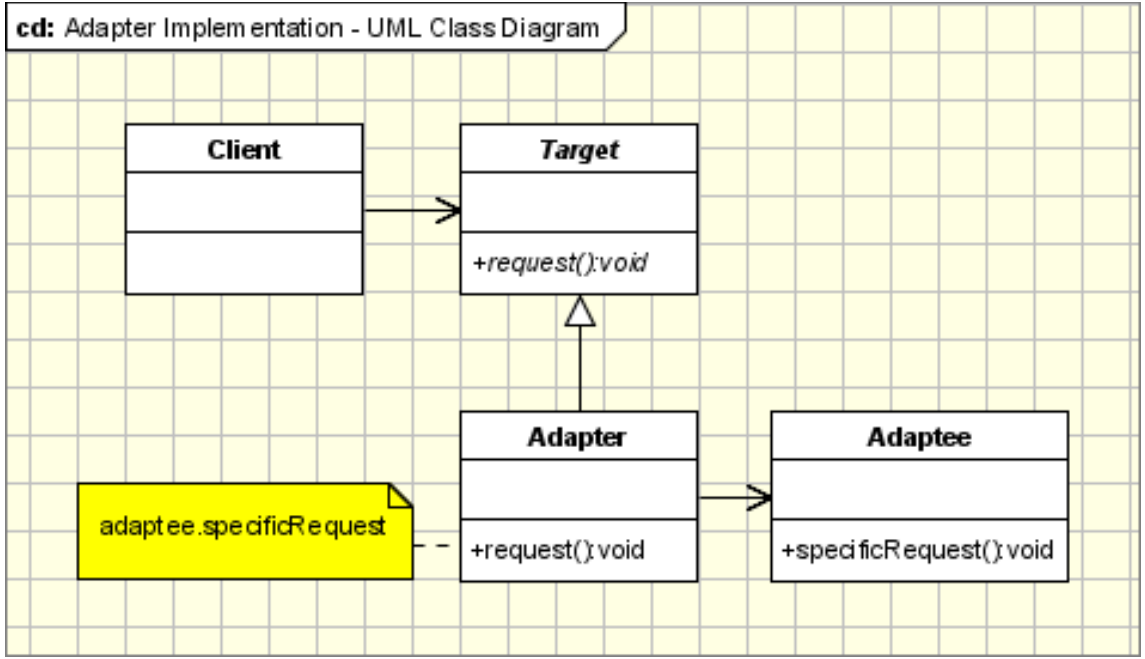
Adapter Design Pattern

The adapter pattern is adapting between classes and objects. Like any adapter in the real world it is used to be an interface, a bridge between two objects.

- It converts the interface of a class into another interface clients expect.
- Adapter lets classes work together, that could not otherwise because of incompatible interfaces.



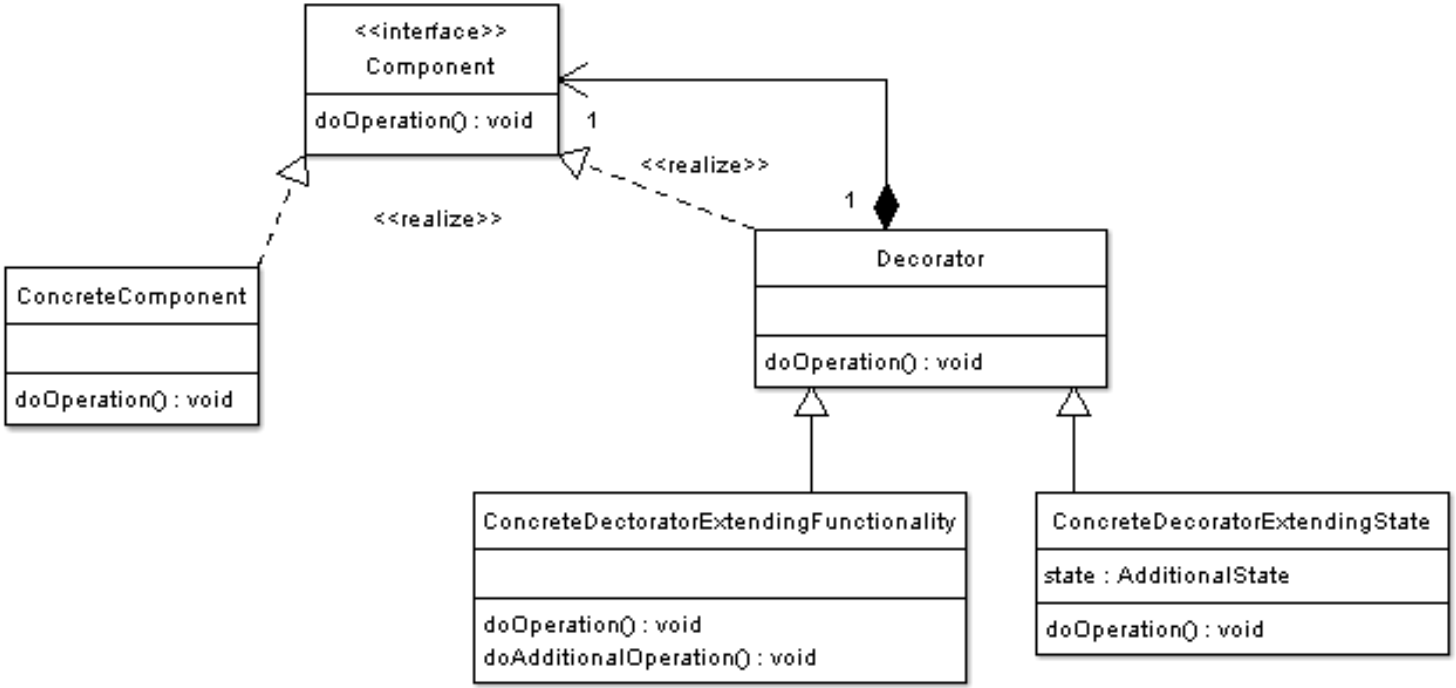
Adapter Design Pattern



Decorator Design Pattern

- Also known as Wrapper Design Pattern.
- Extending an object's functionality can be done statically (at compile time) by using inheritance however it might be necessary to extend an object's functionality dynamically (at runtime) as an object is used.
- The decorator pattern applies when there is a need to dynamically add as well as remove responsibilities to a class, and when subclassing would be impossible due to the large number of subclasses that could result or to overcome multiple inheritance.

Decorator Design Pattern



Last Advice

- There are much more design patterns and OOP analysis and design techniques for you to explore and utilize in your everyday development.
- Keep them in the back of your mind and make an effort to strive for good code, easily extensible and maintainable.

Resources

- http://www.oodesign.com/
- Head First, Object-Oriented analysis and design
  - Book by Brett McLaughlin
- Head First, Design Patterns
  - Book by Elisabeth Freeman and Eric Freeman