COMP2511

23T1 Week 9

WEDNESDAY 1PM - 4PM (W13B)

FRIDAY 11AM - 2PM (F11A)

Slides by Alvin Cherk (z5311001)

This week

- Template method pattern
- Decorator pattern

There are assignment-ii interviews week 10. You have to do it otherwise you just get 0.

No blogs, no marks. (Including access, layout, MRs).

If you are unsure if I can access your blog, feel free to email me a link to check if I can view it.

Week 10: Kahoot & Revision. Email me if you want to cover something specific.

MyExperience

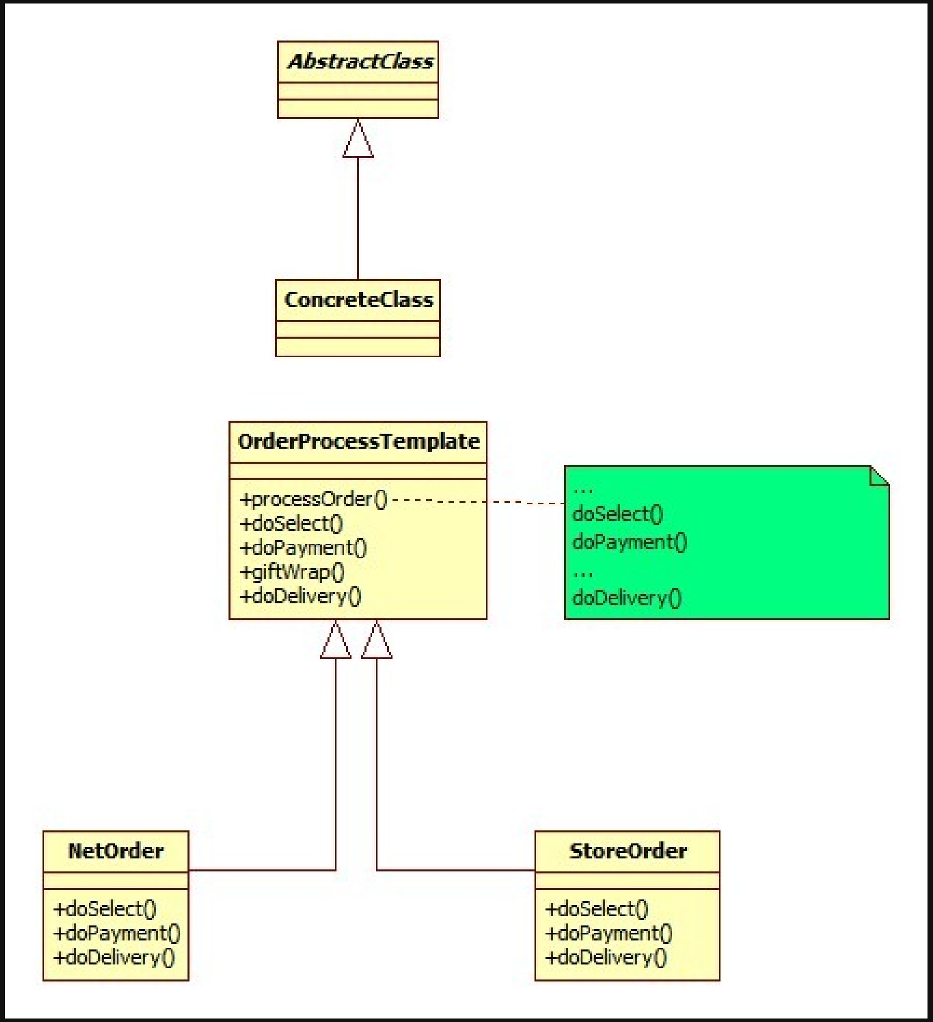
Template Pattern

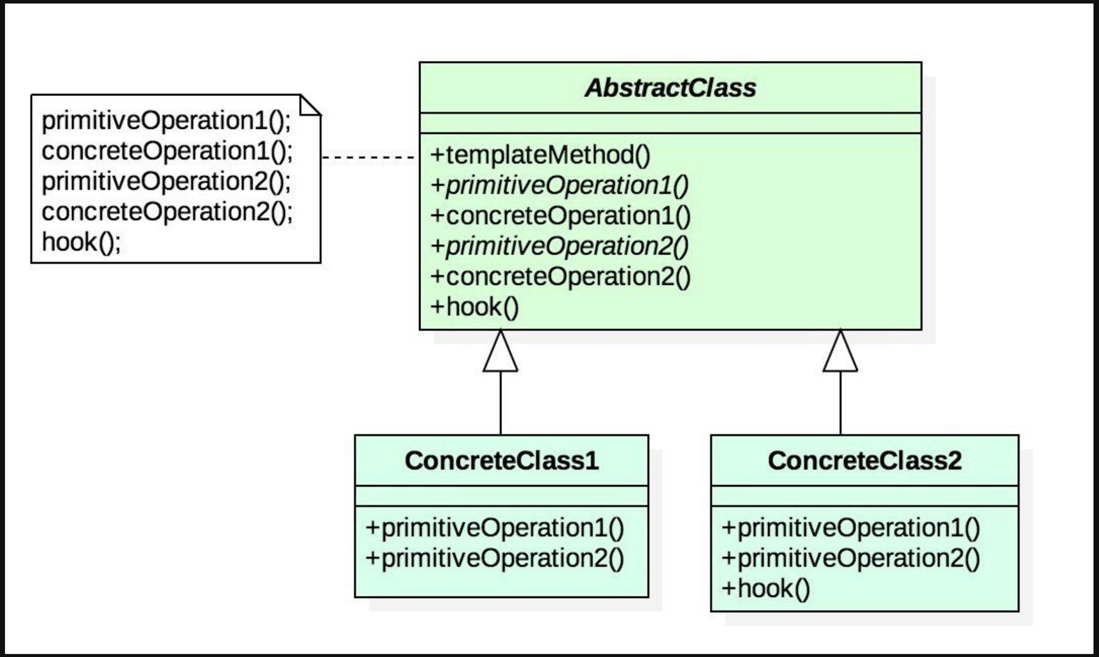
Template Pattern

What kind of design pattern is it?

Behavioural

- Template method pattern defines a **skeleton** (structure) of a behaviour.
- The template method calls primitive operations, that could be implemented by subclasses OR has default implementations in abstract super class.
- Subclasses can redefine only certain parts of the behaviour without changing the other parts of the structure.





- Primitive operations: Operations that have default implementations, or must be implemented by subclass
- Final operations: Concrete operations that cannot be overriden
- Hook operations: Concrete operations that do nothing by default and can be redefined by subclass if necessary. This gives the subclass the ability to "hook into" the algorithm at various points

Template vs Strategy

- Template method works at the class level, so its **static**
- Strategy works on the object level, letting you switch behaviours at run-time
- Template method is based on inheritance: Alter parts of the algorithm by extending those parts in subclasses
- Strategy is based on composition: You alter parts of the object's behaviour by supplying it with a different strategy
- Strategy can change their behaviour after creation (supply with new behaviour), templates cannot change behaviour after construction

Decorator Pattern

Decorator Pattern

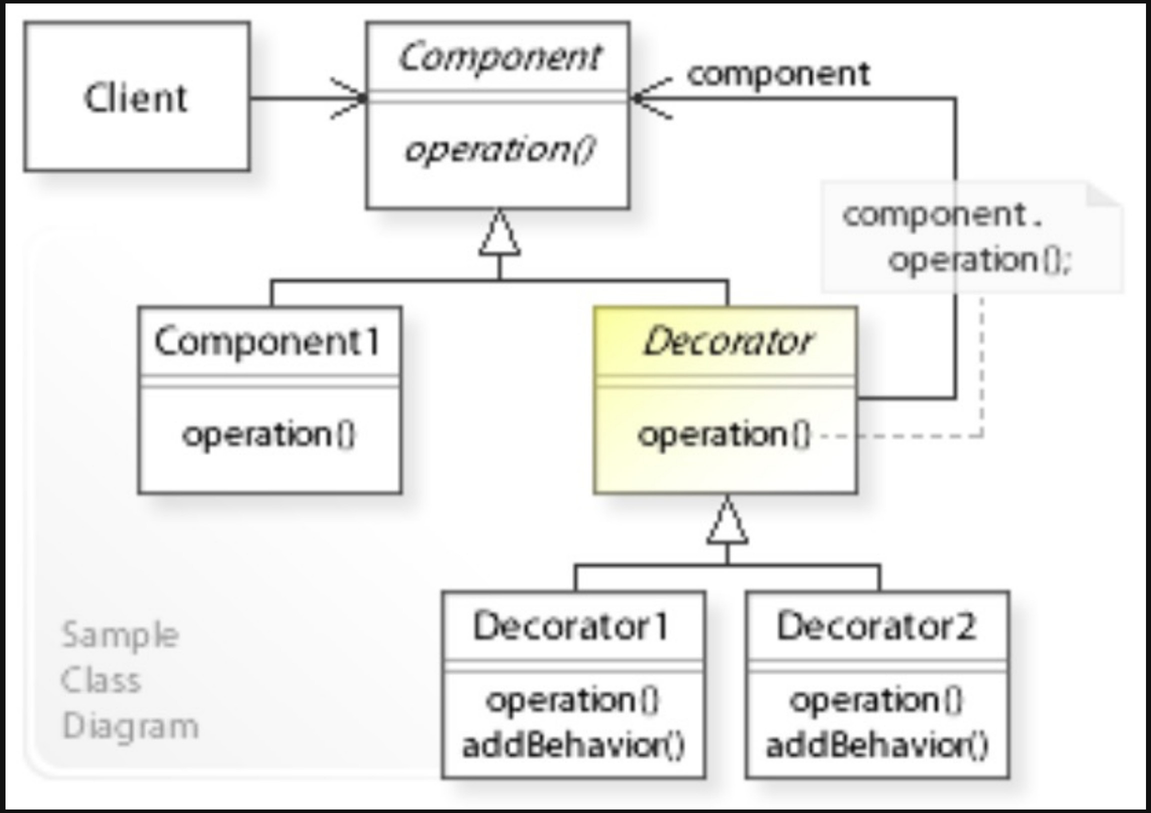
What kind of design pattern is it?

Structural

- Adds functionality to a class at run-time. Used when subclassing would result in an exponential rise in new classes
- Attaches additional responsibilities to an object dynamically
- Avoids implementing all possible functionality in one complex class
- Prefers composition over inheritance

Adding behaviour to an object, without opening the object up (i.e., rewriting its contents) and changing it.

Decorator Pattern



- Client: refers to component interface
- Component: defines a common interface for Component1 and Decorator objects
- Component1: Defines an object that gets decorated
- Decorator: maintains a reference to a Component object, and forwards requests to this component object (component.operation())
- Decorator1, Decorator2, ...: implement additional functionality (addBehaviour() to be performed before and/or after forwarding a request)

Decorator Pattern

```java
public interface Component {
  void doOperationA();
  void doOperationB();
}

public class ConcreteComponent implements Component {
  @Override
  void doOperationA();

  @Override
  void doOperationB();
}

public abstract class Decorator implements Component {
  private Component cc;
}

public class ConcreteDecoratorX extends Decorator {}
```

Code Demo

Template

Link

Attendance

Feedback

https://forms.gle/R4sMTTQzPC4vqXSN8