**Fall**

16

**nCino**

A Summary of Design Patterns

Amanda Fouts

A short summary of Design Patterns

08

**Fall**

# Decorators

An easy way to remember the decorator pattern is that it provides decoration to functions/etc. already in place. For example, adding headlights to a bike would be decorating the bike. After implementing a bike interface, one might decorate it through a headlight function. Another good example may be pizza toppings and how they are calculated.

The decorator pattern is also known as the Wrapper pattern. It allows behavior to be added to an object, either statically or dynamically, without affecting the behavior of other objects from the same class. It is used to adhere to the single responsibility pattern.

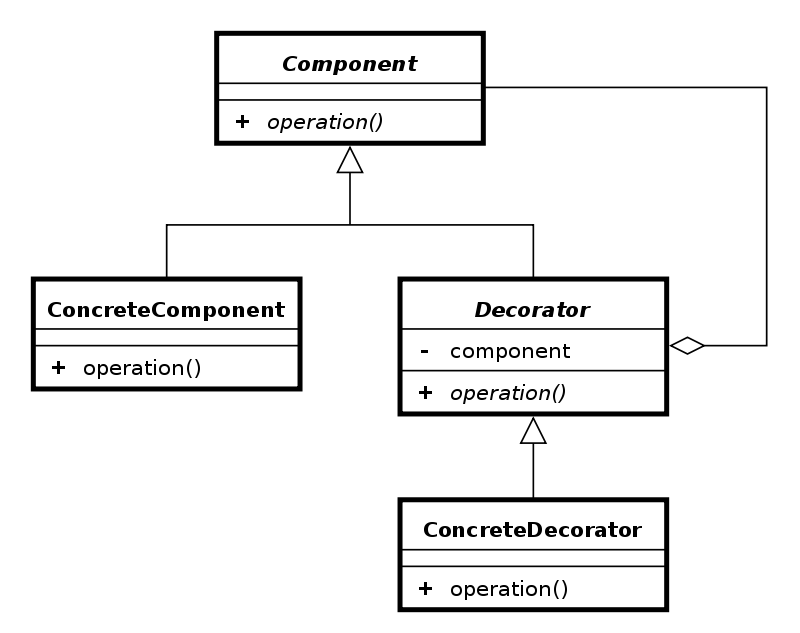


Figure - Decorator UML

# Bridge Pattern

The bridge pattern is used to decouple abstraction from implementation in order for the two be able to vary independently. Bridge pattern uses encapsulation, aggregation, and inheritance to separate responsibilities in to two or more classes. It is useful when the class (abstraction) and what it does (implementation) vary often. Think of two layers of abstraction.

# 

Figure - Bridge Pattern UML

# Builder Pattern

The builder pattern is an object creation design pattern that is used to enable polymorphism. If your constructors look messy or you are using too many constructors, you might want to look to the builder pattern because it is commonly used as a solution to the telescoping constructor anti-pattern. A builder is used in place of numerous constructors.

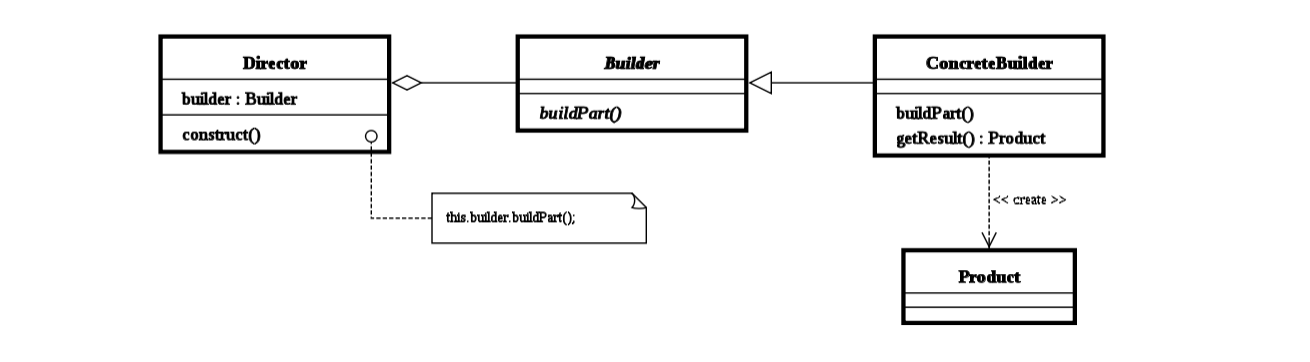


Figure - Builder Pattern UML

# Chain of Responsibility Pattern

Chain of responsibility might look like this when implemented:

Request.isIncluded(var).isIncluded(var)….

Each processing object has logic within it that defines the types of command object that it can handle. The rest are passed to the next processing object in the chain. A mechanism is also in place for adding new processing objects at the end of this chain. It is related to the chaining pattern.

# Façade Pattern

A façade provides a simplified interface to lots of code. A façade might make a software library easier to use, test, read, etc. Façade patterns reduce dependencies since most code uses the façade. Another use might be to wrap a lot of poorly designed code within the better code of the Façade. The Façade pattern is often used with a system that is difficult to understand or complex.

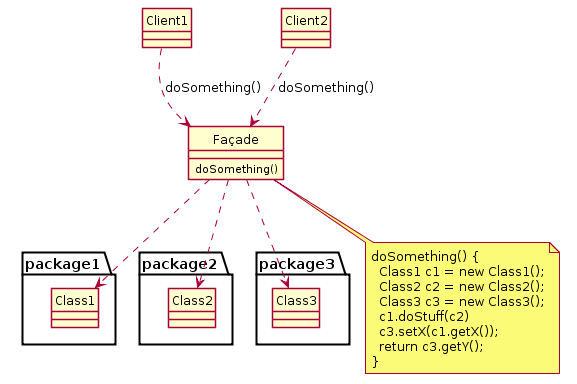


Figure - Facade Pattern UML

# Singleton

The singleton pattern restricts the instantiation of a class to one object. Facades are most of the time singletons because only one façade object is required. Singletons should be used when one object is necessary to coordinate actions across the system. Some consider the singleton to be an anti-pattern.

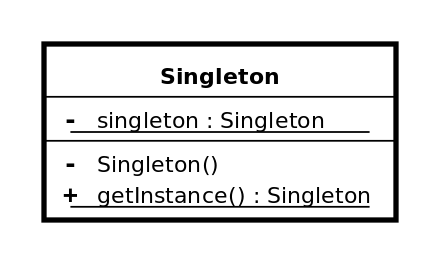


Figure - Singleton UML

# References

1. Wikipedia