Software Design & Analysis

Engr. Abdul-Rahman Mahmood

DPM, MCP, QMR(ISO9001:2000)

armahmood786@yahoo.com
alphapeeler.sf.net/pubkeys/pkey.htm
pk.linkedin.com/in/armahmood
www.twitter.com/alphapeeler
www.facebook.com/alphapeeler
abdulmahmood-sss
alphasecure
armahmood786@hotmail.com
http://alphapeeler.sf.net/me

alphasecure@gmail.com
http://alphapeeler.sourceforge.net
http://alphapeeler.tumblr.com
armahmood786@jabber.org
alphapeeler@aim.com
mahmood_cubix
48660186
alphapeeler@icloud.com
http://alphapeeler.sf.net/acms/

An Introduction to Design Patterns

What is Design Pattern

- Design pattern is a general **reusable** solution to a commonly occurring problem in software design.
- A design pattern is not a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

Why Design Patterns

- •To **design a new** software system quickly and efficiently.
- •To **understand a existing** software system.

Introduction

- Promote reuse.
- Use the experiences of software developers.
- A shared library/lingo used by developers.
- “Design patterns help a designer get a design right faster”.

Introduction

- **Based on** the principles of object-oriented programming: abstraction, inheritance, polymorphism and association.
- Are **solutions to recurring problems** to software design.
- Are **independent of the application domain**.
- Example – Variability of interfaces – the modeller view controller (MVC) pattern.

The Downside

- Although design patterns are useful in promoting flexibility, this maybe at the expense of a more **complicated design**.
- There does not exist
  - A **standardization** for indexing patterns
  - **General practices/processes** for using design patterns during the design process have not as yet been established.

Object-Oriented Principles

- Involves identifying:
  - **Classes** and **objects**
  - What to **encapsulate**
  - **Association** hierarchies
  - **Inheritance** hierarchies
  - **Interface** hierarchies
- Object-oriented designs are evaluated in terms of how **reusable**, **extensible** and **maintainable** they are.

Types of Pattern – Catalog 1

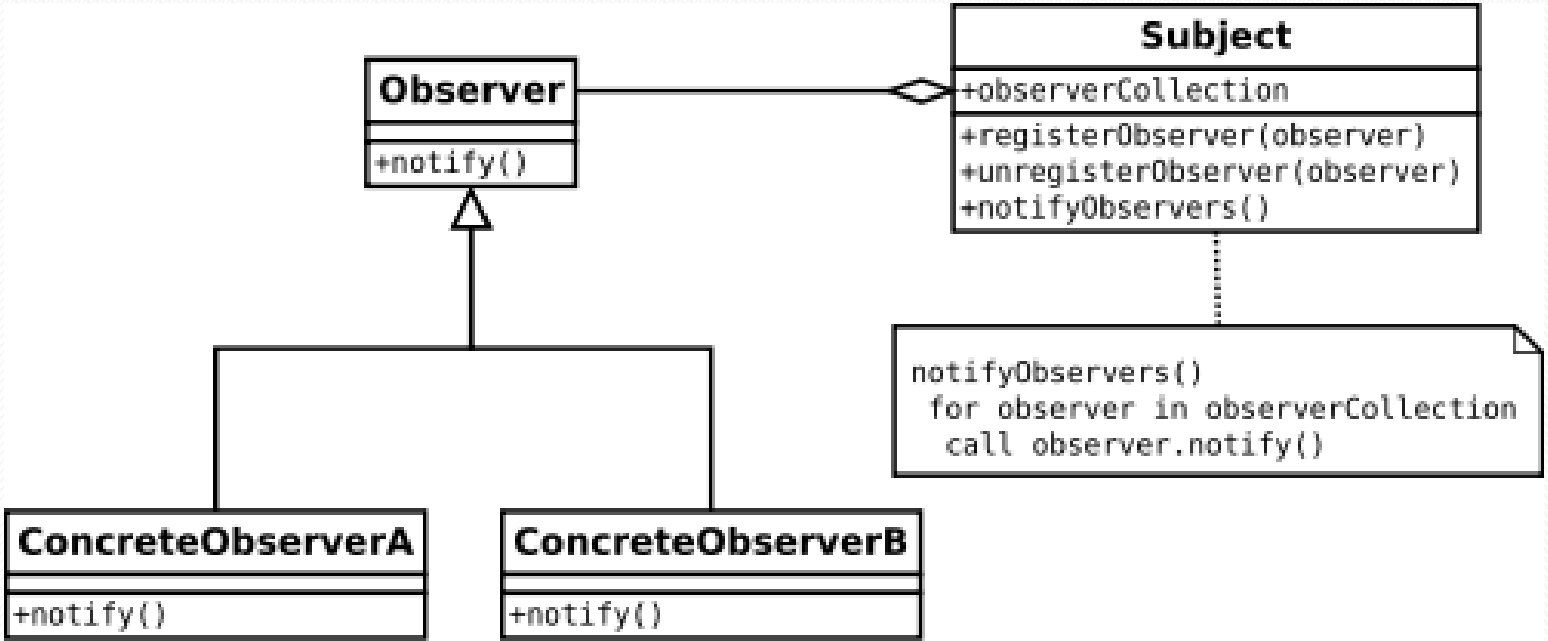
- Creational patterns
  - Focus on Object creation.
  - Focus on the best way to create instances of objects to promote flexibility, e.g. factory pattern.
- Structural patterns
  - Focus on Relationship between entities.
  - Focus on the composition of classes and objects into larger structures, e.g. the adapter pattern.
- Behavioural patterns
  - Focus on Communication between objects
  - Focus on the interaction between classes or objects, e.g. the observer pattern.

Types of Pattern – Catalog 2

- Architectural patterns
  - Focus on the form of the **overall system**.
- Design patterns
  - Focus on the form of the **subsystems** making up the overall system and essentially **provides schemes** for refining them.

Observer Design Pattern

- •Observer Design Pattern is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods.
- •Type : Behavioral pattern.

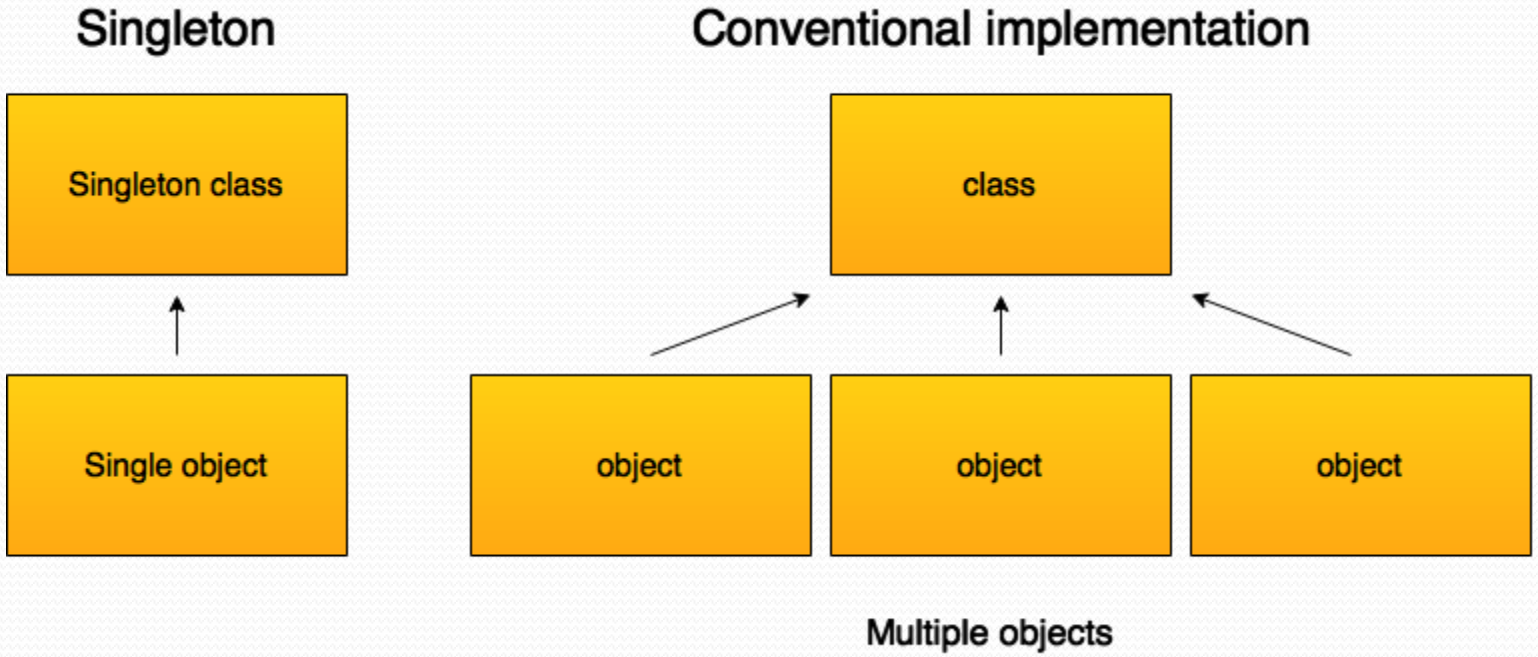


Factory Design Pattern

- Define an interface for creating an object, but let the subclasses decide which class to instantiate. The Factory method lets a class defer instantiation to subclasses.
- Type : Creational pattern.

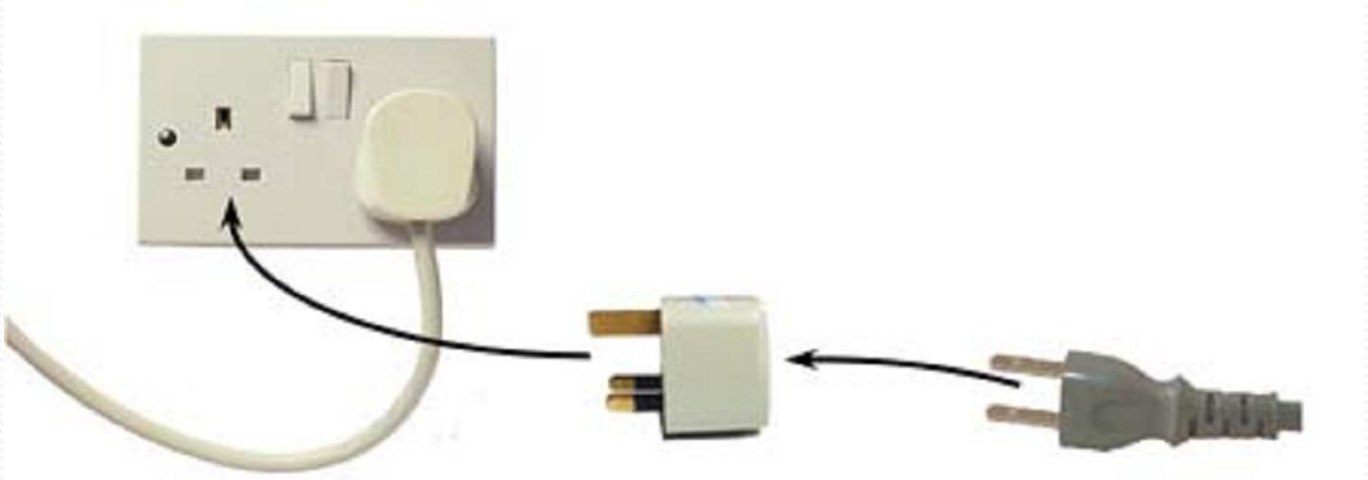
Singleton Design Pattern

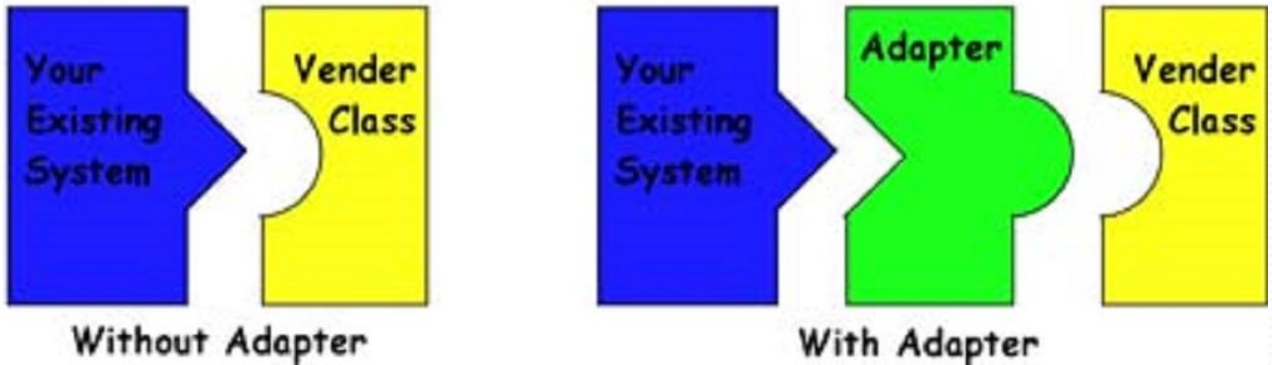
- Ensure a class has only one instance, and provide a global point of access to it.
- Encapsulated “just-in-time initialization” or “initialization on first use”.
- Type : Creational pattern.



Adaptor Design pattern

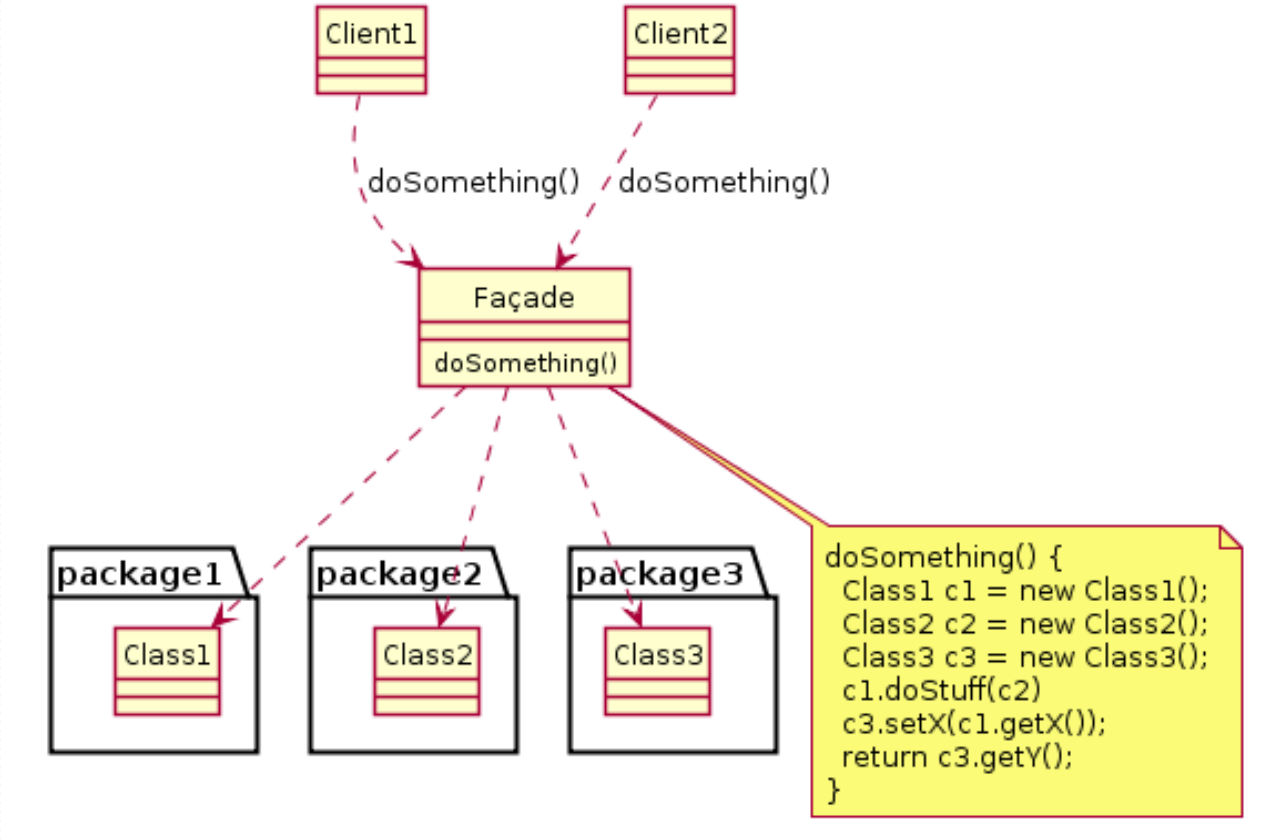
- The adapter pattern (often referred to as the wrapper pattern or simply a wrapper) is a design pattern that ***translates***one interface for a class into a compatible interface.



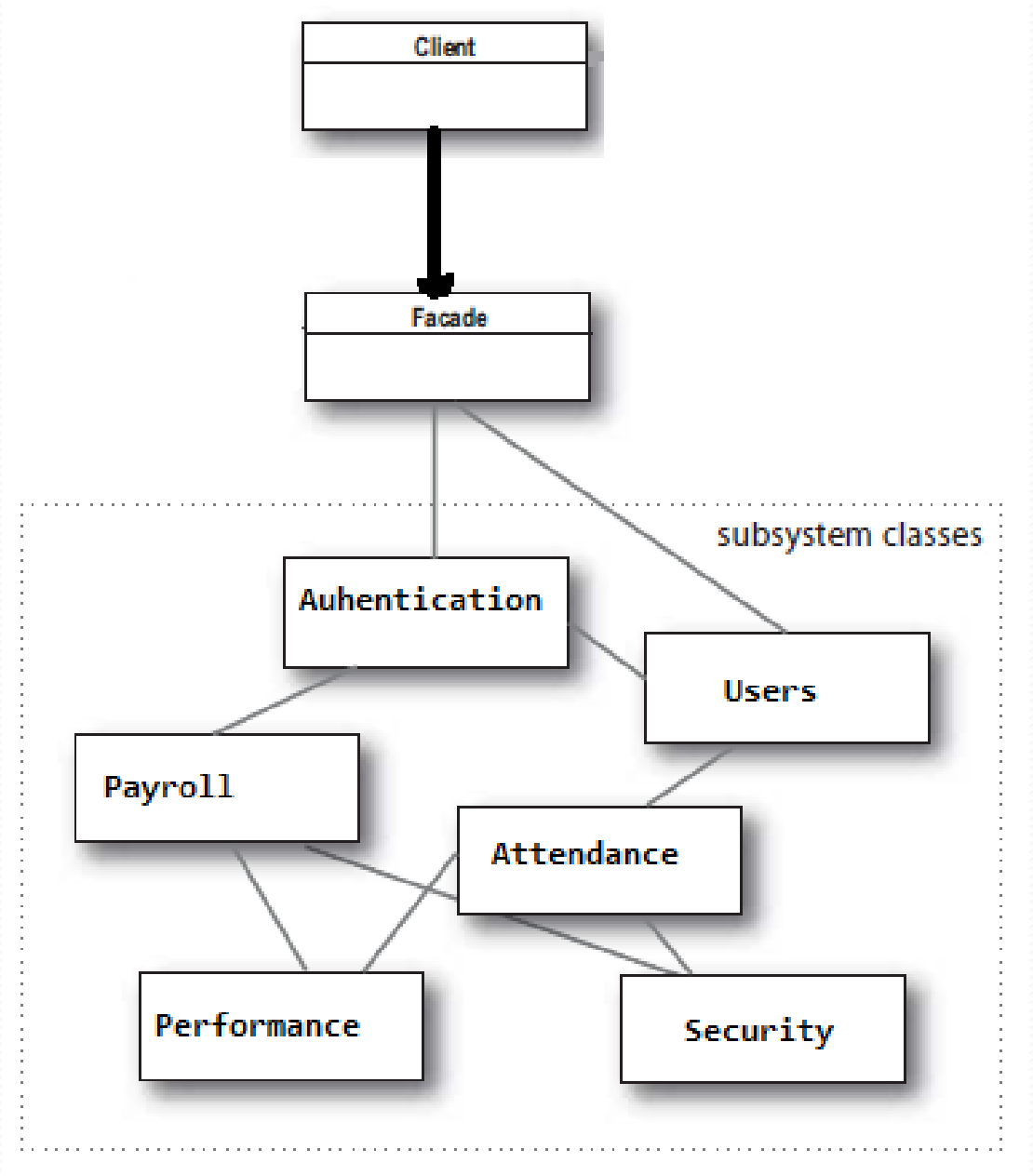


Façade pattern

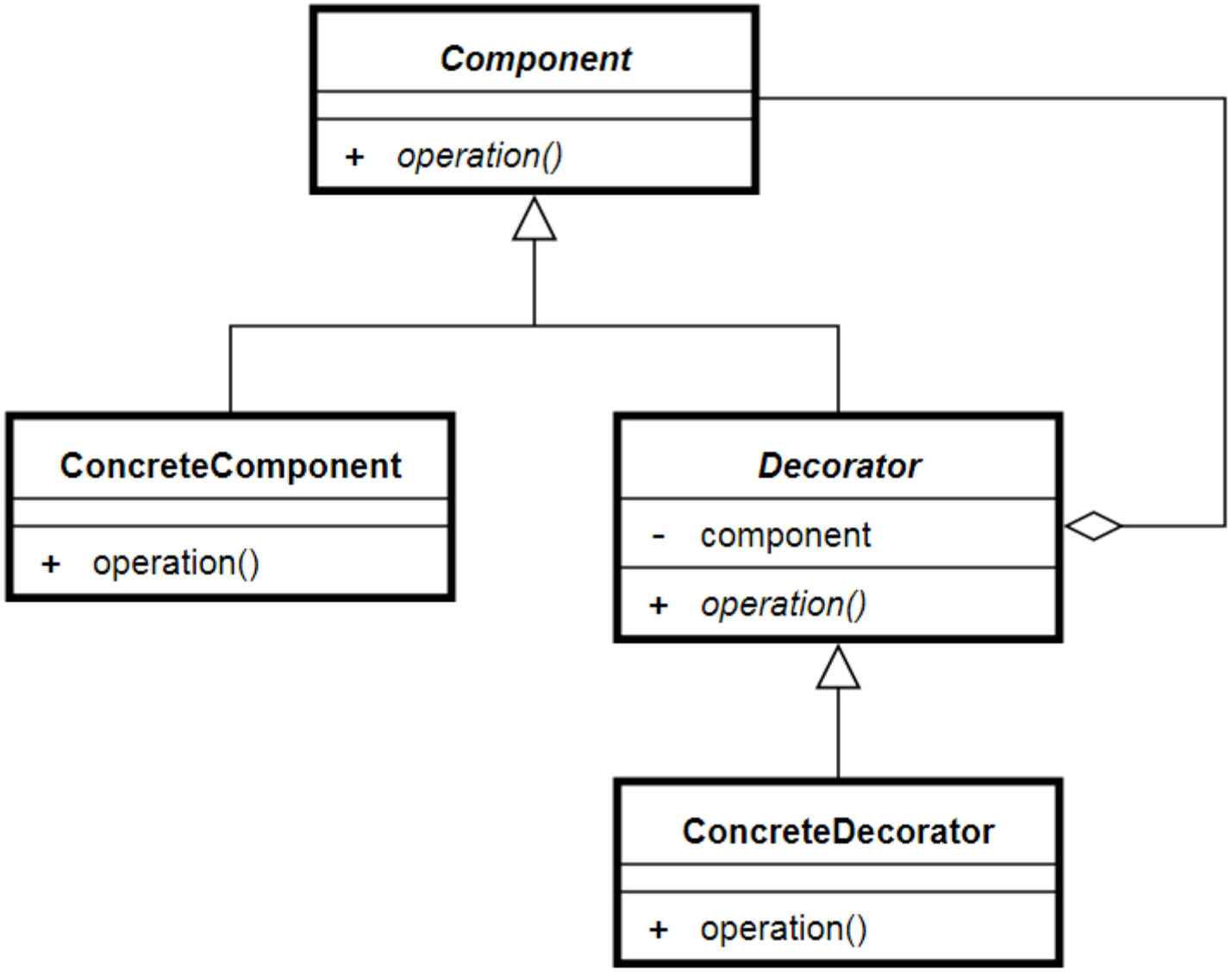
- A facade is an object that provides a **simplified interface** to a larger body of code, such as a class library.
- Type: Structural Design Pattern.



Façade pattern



Decorator pattern



Pattern Scope

- The scope of a pattern specifies whether the pattern applies to **classes** or **objects**.
- **Class patterns** describe relationships between classes and their subclasses. These relationships are static.
- **Object patterns** describe the relationships between objects. Theses relationships can be changed at runtime.

Defining Patterns

- Are defined in terms of classes and objects and relationships between them.
- Using existing well-tested patterns saves time instead of deriving them from scratch each time.
- Patterns may consist of smaller patterns/sub-patterns.
- Class diagrams are used to express design patterns.

Core Components of a Pattern

- The problem which the pattern was used to solve and the situation giving rise to the problem. A **list of the conditions that must be met** in order to apply the pattern may also be included.
- The **core solution** to the problem in terms of a description of the design rather than implementation details.
- **Uses** of the solution

A More Detailed Definition

- Name
- Intent of the pattern
- Aliases
- The problem
- Solution
- Example/s
- Applicability
- Structure
- Participants
- Collaborations
- Implementation
- Sample code
- Known uses
- Related patterns
- Consequences

Problems Solved by Design Patterns

- Finding appropriate objects.
- Determining the granularity of objects.
- Specifying object interfaces – definition of interfaces and relationships between them.
- Specifying object implementations.
- Using reuse mechanisms – delegation and parameterised types.
- Relating runtime and compile-time structures

Designing for Change

- A design must facilitate reuse and change.
- We need to design so as to avoid redesign.
- Certain design patterns can be used to prevent particular causes of redesign.

Avoiding Redesign

- Explicitly declaring class instances instead of using an interface -abstract factory, factory method and prototype patterns.
- Dependence of specific operations, i.e. using hard-coded requests - chain of responsibility and the command patterns.
- Limit software and hardware platform dependencies - abstract factory and bridge patterns.
- Dependence on object representations and implementations - Object representations and/or implementations may need to be changed - abstract factory, bridge, memento and proxy.

Avoiding Redesign

- Algorithmic dependencies - Algorithms that are likely to change should be isolated from the definition of the objects using them -builder, iterator, strategy, template and visitor patterns.
- Tight coupling - Tight coupling does not facilitate reuse- abstract factory, bridge, chain of responsibility, command, facade, mediator and observer patterns.
- Subclassing to extend functionality - Rather than using inheritance or association it maybe more efficient to combine both by creating one subclass that is associated with existing class - bridge, chain of responsibility, composite, decorator, observer and strategy classes.
- Difficulty in altering classes - In some cases adapting a class maybe difficult, e.g. the source code may not be available adapter, creator and visitor patterns.

Applying a Pattern

- In designing a system different patterns are used to design the different aspects of the system.
- Design patterns allow parts of the system to vary independently of other parts of the system.
- Patterns are often combined. Using one pattern my introduce further patterns into the design.
- Methods for applying design patterns and deciding which one to use.

Breaking Down the Problem

- Describe the problem and its subproblems.
- Select the category of patterns that is suitable for the design task.
- Compare the problem description with each pattern in the category.
- Identify the benefits and disadvantages of using each of the patterns in the category.
- Choose the pattern that best suits the problem.

Choosing a Design Pattern

- Consider the problems solved by design problems and what solution is needed for the problem at hand.
- Consider the intent of each pattern and which is most similar to the problem at hand.
- Analyse the relationships between patterns to determine which is the correct group of patterns to use.
- Determine whether creational, structural or behavioural patterns are needed and which of the patterns in the most suitable category is/are relevant to the problem at hand.
- Look at what could cause redesign for the problem at hand and patterns that can be used to avoid this.
- Identify which aspect/s of the system need to varied independently and which patterns will cater for this.

Using a Design Pattern

- Obtain an overview of the pattern.
- Obtain an understanding of the classes and objects and relationships between them.
- Choose application-specific names for the components of the patterns.
- Define the classes.
- Choose application-specific names for the operations defined in the pattern.
- Implement the necessary operations and relationships.