UNIT-IV

Structural pattern

Insoftwareengineering,structuraldesignpatternsaredesignpatternsthateasethedesignby identifying a simple way to realize relationships betweenentities.

Examples of Structural Patterns include:

- Adapter pattern: 'adapts' one interface for a class into one that a clientexpects
  - Retrofit Interface Pattern[1][2]: An adapter used as a new interface for multiple classes at the sametime.
  - Adapter pipeline: Use multiple adapters for debuggingpurposes.[3]
- Bridge pattern: decouple an abstraction from its implementation so that the two can vary independently
- Tombstone: An intermediate "lookup" object contains the real location of anobject.[4]
- Composite pattern: a tree structure of objects where every object has the sameinterface
- Facade pattern: create a simplified interface of an existing interface to ease usage for common tasks
- Flyweight pattern: a high quantity of objects share a common properties object to savespace
- Proxy pattern: a class functioning as an interface to anotherthing

Incomputerprogramming,theadapterpattern(oftenreferredtoasthewrapperpatternorsimply awrapper)isadesignpatternthattranslatesoneinterfaceforaclassintoacompatibleinterface.
An *adapter* allows classes to work together that normally could not because of incompatible interfaces, by providing its interface to clients while using the original interface. The adapter translates calls to its interface into calls to the original interface, and the amount of code necessary to do this is typically small. The adapter is also responsible for transforming data into appropriate forms. For instance, if multiple boolean values are stored as a single integer (i.e. flags) but your client requires a 'true'/'false', the adapter would be responsible for extracting the appropriate values from the integer value. Another example is transforming the format of dates (e.g. YYYYMMDD to MM/DD/YYYY or DD/MM/YYYY).

Thebridgepatternisadesignpatternusedinsoftwareengineeringwhichismeantto*"decouple an abstractionfrom its implementationso that the two canvary independently"*.[1]The *bridge* uses encapsulation, aggregation, and can use inheritanceto separate responsibilities into different classes.

When a class varies often, the features of object-oriented programmingbecome very useful

because changes to a program's codecan be made easily with minimal prior knowledge about the program. The bridge pattern is useful when both the class as well as what it does vary often. The class itself

can be thought of as the *implementation* and what the class can do as the *abstraction*. The bridge pattern can also be thought of as two layers of abstraction.

In software engineering, the composite pattern is a partitioning design pattern. The composite pattern describes that a group of objects are to be treated in the same way as a single instance of an object. The intent of a composite is to "compose" objects into tree structures to represent part-whole hierarchies. Implementing the composite pattern lets clients treat individual objects and compositions uniformly.

Thefacadepatternisasoftwaredesignpatterncommonlyusedwithobject-orientedprogramming. The name is by analogy to an architecturalfacade.

A facade is an object that provides a simplified interface to a larger body of code, such as a classlibrary. A facade can:

- make a software library easier to use, understand and test, since the facade has convenient methods for commontasks;
- make the library more readable, for the samereason;
- reduce dependencies of outside code on the inner workings of a library, since most codeuses the facade, thus allowing more flexibility in developing thesystem;
- wrap a poorly-designed collection of APIs with a single well-designed API (as per

In computer programming, flyweight is a software design pattern. A flyweight is an objectthat minimizes memoryuse by sharing as much data as possible with other similar objects; it is a way to use objects in large numbers when a simple repeated representation would use an unacceptable amount of memory. Often some parts of the object state can be shared, and it is common practice to hold them in external data structuresand pass them to the flyweight objects temporarily when they are used.

In computer programming, the proxy pattern is a software design pattern.

A proxy, in its most general form, is a class functioning as an interface to something else. The proxy could interface to anything: a network connection, a large object in memory, a file, or some other resource that is expensive or impossible to duplicate.A well-known example of the proxy pattern is a reference counting pointerobject.

STRUCTURAL PATTERN

Name	Description	In *DesignPatterns*	In *Code Complete*[17]	Other
Adapteror Wrapper orTranslator.	Convert the interface of a class into another interface clients expect. An adapter lets classes work together that could not otherwise because of incompatible interfaces. The enterprise integration pattern equivalent is the Translator.	Yes	Yes	N/A
Bridge	Decouple an abstraction from its implementation allowing the two to vary independently.	Yes	Yes	N/A
Composite	Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.	Yes	Yes	N/A
Decorator	Attach additional responsibilities to an object dynamically keeping the same interface. Decorators provide a flexible alternative to subclassing for extending functionality.	Yes	Yes	N/A
Facade	Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.	Yes	Yes	N/A
Front Controller	The pattern relates to the design of web applications. It provides a centralized entry point for handling requests.	No	Yes	N/A

Flyweight	Use sharing to support large numbers of similar	Yes	No	N/A
	objects efficiently.			
Proxy	Provide a surrogate or placeholder for another object to control access to it.	Yes	No	N/A
Module	Group several related elements, such as classes, singletons, methods, globally used, into a single conceptual entity.	No	No	N/A