Design Patterns:

*Characteristics of Bad Design:*

1. Any Change done affect other parts of the System (Rigidity)
2. When we make a Change , unexpected part of the System breaks(Fragility)
3. Not able to reuse a Component, Since it has tight coupling with the application (Immobility)
4. Easy to do the wrong thing but hard to do the right thing resulting in tendencies of the developers to avoid 'clean fix'(Viscosity).

*Characteristics of Good Design:*

1. Robust
2. Extendable
3. Maintainable
4. Reusable

*S.O.L.I.D Principles (OO Design):*

S – Single Responsibility Principles:

1. “There should not be more that one reason for a class get Change”
2. Every class must have single responsibility and should be entirely encapsulated by the class.

O – Open Close Principles:

1. “Software entities (Class, module, function) must be Open for extension but Close for Modification”
2. Any new functionality must be added as new Class or new Method without changing the existing code.(Bug Fix is a different Case)

L – Liskov’s Substitution Principles:

1. “In Class hierarchies, subclass should be complete substitutable for their base Class.”
2. Must make sure the derived class are extending the base classes without changing their behaviors.
3. Design by Contract(DbC)

I-Interface Segregation principle:

1. “Client should not be forced to depend upon the interfaces which they do not use.”
2. Many Client Specific Interface are better than one general purpose interface.
3. Aim is to split the interfaces, which are very large into small and SPECIFIC, so the client known methods only which there are interested.

D- Dependency Inversion Principle:

1. “Depend on abstraction, not on concretions”.
2. High level module should not depend on the low level module. High level module must not use the directly use the low-level module.
3. High level must use the low level indirectly thorough the interface of the low level (Run time polymorphism)

What is Design Patterns?

1. General repeatable solution for a Commonly occurring problem in software Design
2. Its ‘n a finished design that can be transformed directly into code.

***Class Pattern*s**: All the patterns that works on class will be classified as Class Patterns

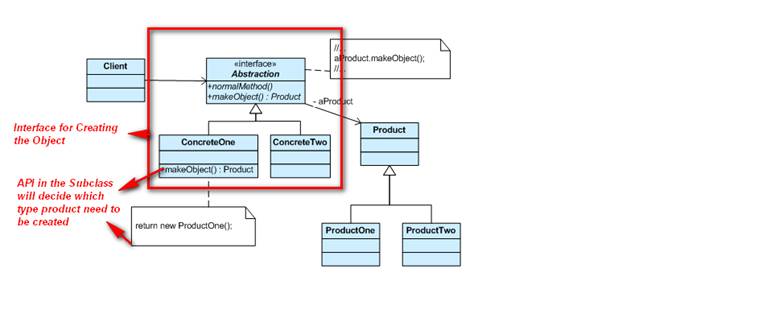
***Object Patterns***: All the patterns that works on Object will be classified as Object Patterns

***Creation Design Pattern:***

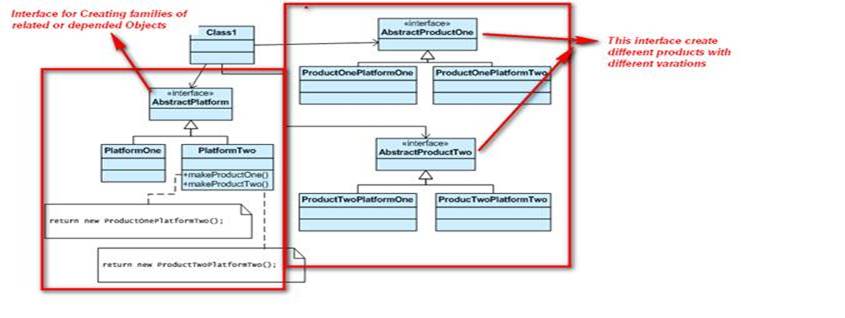
   Class – Creation Pattern – Patterns use the inheritance effectively in the Class instantiation process

   Object – Creation patterns – Delegation effectively for the Object creation.

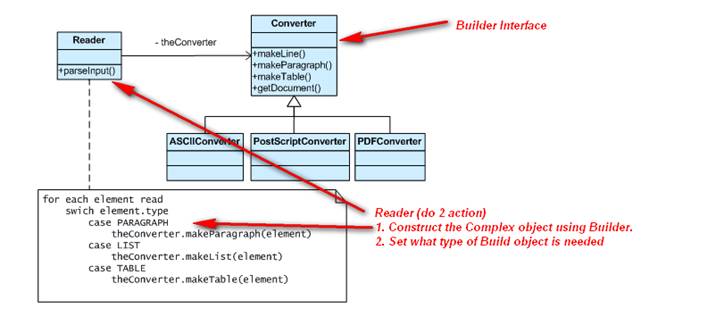
1. Singleton Design Pattern: (Object Patterns)
2. “Only one instance of the class is created and provide a global point to access it”
3. If the Single tone object depend on the other singleton object the Order of Creation of the singleton is important
4. When a Singleton class is inherited, make sure the subclass constructor is private.
5. Lazy loading – Object are created, when its request for first time.
6. Factor Method : (Class Patterns)
7. “Define a Interface for creating object, but let the subclass decide which class to instantiate”



1. Abstract Factory Design Patterns (Class Patterns)
2. “Provide an interface for creating families of related or dependent objects without specifying their concrete classes.”

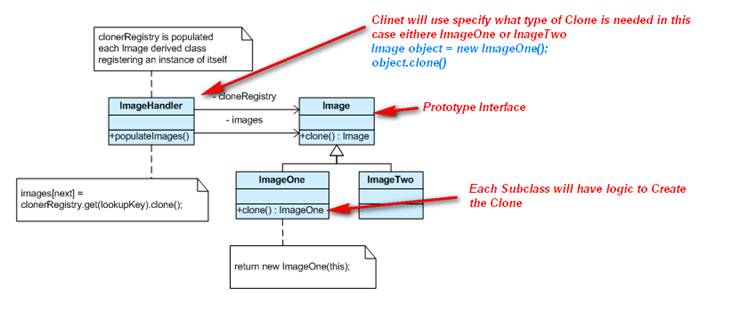


1. Builder Design Pattern:
2. “Separation Construction of Complex object from its representation- allow client object to construct a complex object by specifying only its type and content being shielded from the details related to the object’s representation. This way the construction process can be used to create different representations. “



1. Prototype Design Pattern

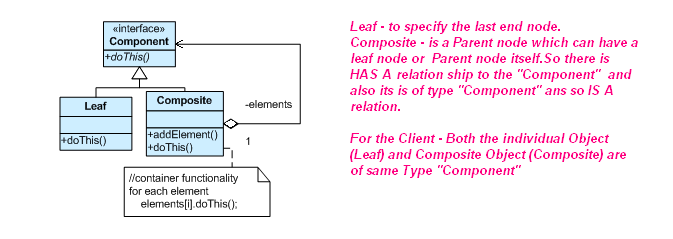
“Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.”



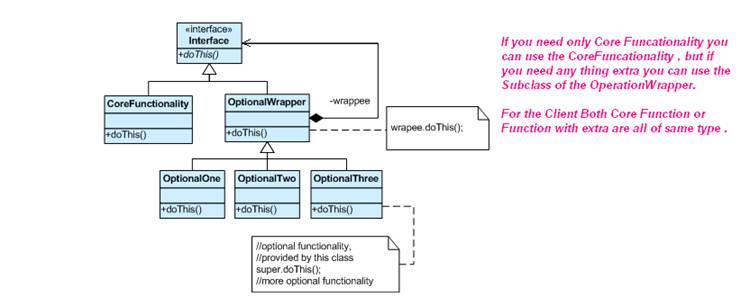
***Structural Patterns: -*** Defines the Structure of the objects and Classes that works to together and define how the relations can be defined.

1. Composite Design Patterns:

“Compose Object into tree Structure to represent the whole-part hierarchies. For the Client the individual object and Composited Object will be treated as Same”

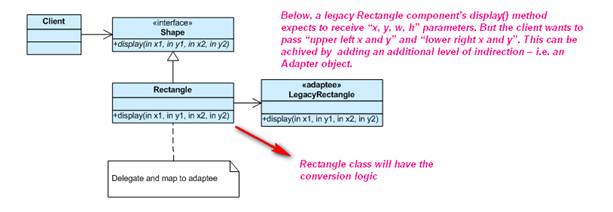


1. Decorate Pattern:
2. “Extending the Functionality   dynamically at Run time.”



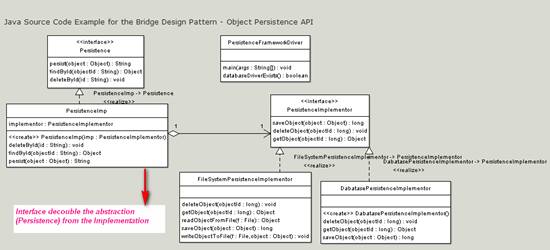
C. Adapter Patter :

      1. “Convert the interface of the class into another Class the client Expected”



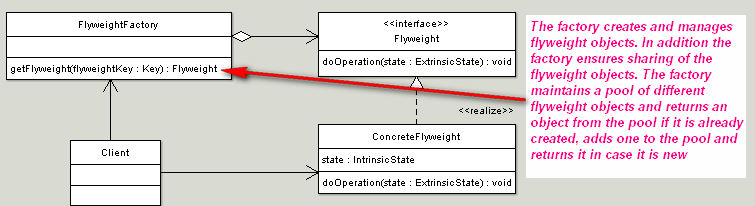
D. Bridge Pattern:

    1. “Decouple an abstraction from its implementation so that the two can vary independently”



E. Fly Weight Pattern:

1. “Sharing large number of fine- grained object efficiently”

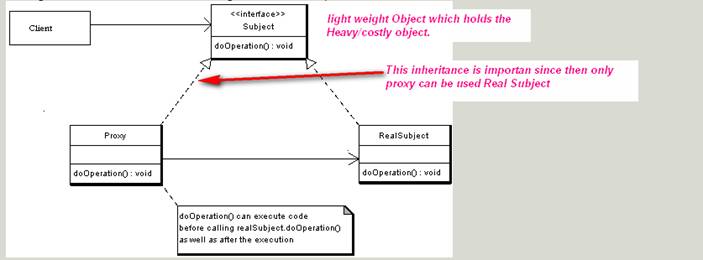


F. Proxy Pattern

 1. “ Provide a Surrogate or placeholder for another object to control access to it”

   2. If only few methods of the heavy /Costly Object is needed instead of initializing the heavy object, we can use the light object exposing the same interface as the heavy object.

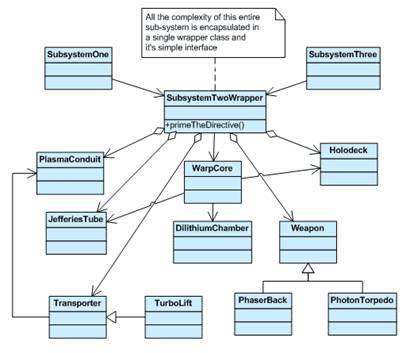
That light Object is called Proxy Object, which will have a place hold to the Heavy Object.



G. Façade Pattern:

   “Providing a unified interface to the set of interface in the subsystem”.

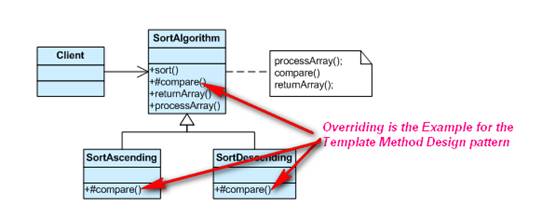
   “Providing high level interface make the subclass easy to use”.



***Behavioral Patterns***-Identify the Common Communication between the Objects and realize these patterns

1. Template Method Design Pattern:

“Subclass redefines certain steps of algorithm without changing the algorithm”



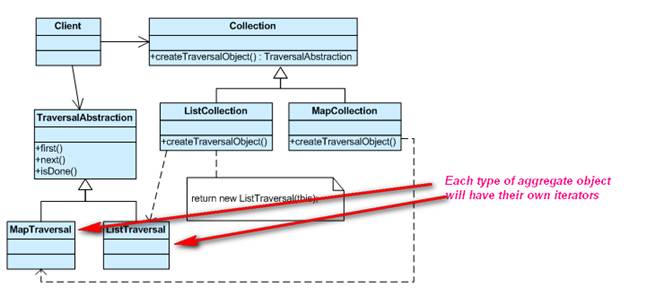
1. Chain of Responsibility Design Pattern:

“Remove the Coupling between the Sender of the request and the receiver, by giving more than one object a chance to handle the Request. Object are chained to do that ”.

Filters are the example of the this Design patterns.

1. Iterator Design Pattern:

“Provide a way to access element of the aggregate object without exposing its underlying representation”

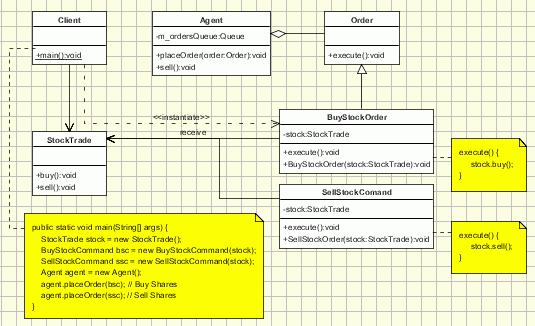


1. Observer Design Pattern:

“Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.”

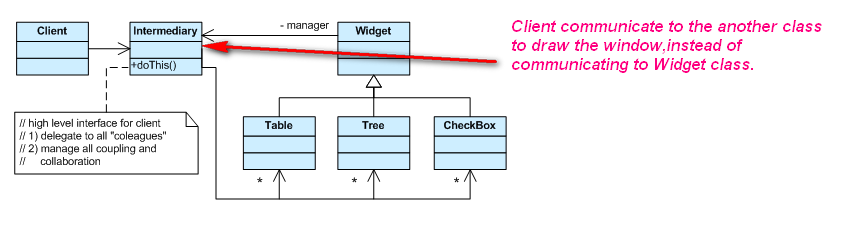
1. Command Design Patterns:

“Command design pattern encapsulates commands (method calls) in objects allowing us to issue requests without knowing the requested operation or the requesting object.”



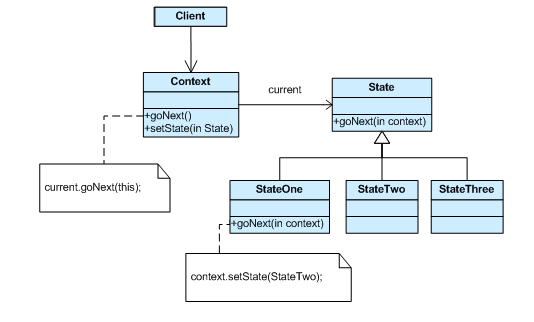
1. Mediator Design Pattern:

“Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.”



1. State Design Pattern:

“Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.”



1. Strategy Design pattern:

* “Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from the clients that use it.

