**<http://blackwasp.co.uk/GofPatterns.aspx>**

**Why we create a software?**

To solve business problems

**Business problems consist of what?**

Consist of real life entities/objects

**How will one/Developer create a software**

By writing “code” using some “programming language”

The developer can’t write a code without knowing the relationship between the real life “entities/objects”

The developer has to use an approach which is oriented towards real life entities

The language which the developer is using to create a software should allow the developer to “map” real life entities into the software

Such languages are known as “object oriented languages”

While developing softwares we may face some recurring software design problems like :

1. How do I create object of a class and populate it with some data?
2. How do I iterate through a collection of objects?

Design pattern is a tried and tested solution to a commonly occurring/recurrent software problem.

**Why design patterns:**

* Time proven solution to recurrent software problems.
* No need to reinvent the wheel.

**Gang of Four:**

* Erich Gamma
* Richard Helm
* Ralph Johnson
* John Vlissides

These four peoples are known as “Gang of Four” they written one book named:

“Design Patterns : Reusable Elements of Object Oriented Softrware”

In the above book they defined 23 design patterns which are known as GoF patterns.

They divided these patterns into 3 main categories like :

1. **Creational Patterns**
2. **Structural Patterns**
3. **Behavioral Patterns**

**Creational Patterns**:-

Deal with object creation.

**Objective** – To Isolate/Separate the logic of object creation from the client code.

We have following patterns under this:

* Singleton
* Factory method
* Abstract factory
* Prototype
* Builder

**Singleton:**

**Objective** –

* Throughout the lifetime of the app, there must be only one instance of some desired class.
* Resourse Sharing
* One and only one instance
* Logic for singleton is within the object itself
* Private parameter less constructor
* Sealed class
* Lazy loaded and thread safe

In .NET 4.0 microsoft introduced class Lazy<T>

* It allows lazy initialization
* It is thread safe

**Factory Method:**

Int x = Convert.ToInt32(“20”)

Double d = Convert.ToDouble(“20.566”)

* Create a method (Factory method) for object creation
* Invoke the factory method

**Abstract Factory:**

* It is simply an extension of factory
* It’s actually a factory of factories
* Encapsulate a set of factories having a common requirement without exposing the concrete classes

**Prototype:**

* Create objects by cloning existing ones
* Cloning logic is inside the class whose object needs to be cloned

Eg – order in Zomato – repeat order

**Structural patterns**:

* Focus is on relationship between classes
* How objects are composed to form complex structures

We have following patterns under this:

* Adopter
* Composite
* Decorator
* Proxy
* flyweight
* façade

**Adapter:**

* deal with incompatible interfaces between client and service provider

**Decorator:**

* To add additional behavior to an object at runtime without modifying the code

**Façade :**

* Provides a high level easy to use interface to the client by shielding the internal interfaces of the sub systems.
* Provides an abstracted view of the subsystems by hiding the complexities

**Behavioural Patterns**

* How objects communicate with each other
* How objects interact with each other in a loosely coupled manner

We have following patterns under this:

* Chain of responsibilities
* Interpreter
* Iterator
* Mediator
* Memento
* Observer
* State
* Strategy
* Template method
* Visitor

**Iterator:**

* Allows your code to access individual elements of an aggregate object sequentially
* Abstracts/Hides the internals of the aggregate object from the client

Eg – foreach loop

**Observer :**

* Define one to many dependencies between objects such that when the state of an object changes all dependant objects are notified
* Called as “pub/sub” pattern

**Memento :**

* Token Pattern
* Allows you to save the state of an object external to the object and restore the state as and when needed.
* A way to create a snapshot of an objects state and restore back to a specific snapshot as and when required
* Used in apps which need to implement some sort of undo/redo functionality