# **Factory Design Pattern**

* Creation design pattern
* Define an Interface and let subclasses decide which class to instantiate

**Advantage:-**

* Loose Coupling: - application will work with any class that implements Interface

**Usage:-**

* When client needs to specify class name to create objects

E.g.:

interface IGet

{

string ConC(string s1, string s2);

}

class clsFirst : IGet

{

public string ConC(string s1, string s2)

{

string Final = "From First: " + s1+" and " + s2;

return Final;

}

}

class clsFactory

{

static public IGet CreateandReturnObj(int cChoice)

{

IGet ObjSelector = null;

switch (cChoice)

{

case 1:

ObjSelector = new clsFirst();

break;

case 2:

ObjSelector = new clsSecond();

break;

default:

ObjSelector = new clsFirst();

break;

}

return ObjSelector;

}

}

class clsSecond : IGet

{

public string ConC(string s1, string s2)

{

string Final = "From Second: " + s1 + " and " + s2;

return Final;

}

}

private void cmbSelect\_SelectedIndexChanged(object sender, EventArgs e)

{

IGet ObjIntrface = null;

ObjIntrface = clsFactory.CreateandReturnObj(cmbSelect.SelectedIndex + 1);

string res = ObjIntrface.ConC("First", "Second");

lblResult.Text = res;

}

# **Adapter Pattern**

* **Structural design pattern**
* It acts as wrapper between two objects i.e. converts the interface of one class into another interface as per requirement
* Also known as **Wrapper**

**Advantage:-**

* Allows interaction of **incompatible** objects
* **Reusability** of existing functionality

**Usage:-**

* Using modern classes in Legacy code

e.g:-

public interface ITarget

{

string GetRequest();

}

//incomptaible interface with current code

public class Adaptee

{

public string GetSpecificRequest()

{

return "Specific request.";

}

}

//Adapter makes Adaptee's interface compatible with target's interface

class Adapter : ITarget

{

private readonly Adaptee \_adaptee;

public Adapter(Adaptee adaptee)

{

this.\_adaptee = adaptee;

}

public string GetRequest()

{

return $"This is '{this.\_adaptee.GetSpecificRequest()}'";

}

}

public static void Main(string[] args)

{

Adaptee adaptee = new Adaptee();

ITarget target = new Adapter(adaptee);

Console.WriteLine("Adaptee interface is incompatible with the client.");

Console.WriteLine("But with adapter client can call it's method.");

Console.WriteLine(target.GetRequest());

}

# **Observer Pattern**

* **Behavioral pattern**
* A **subscription** mechanism to notify dependent objects whenever state of one object changes
* One-to-many dependency between objects
* Also known as **publisher-subscribe**

**Advantages:-**

* Allows coupling between objects and observers
* Support for broadcast type communication

**Usage:-**

* When change in 1 object needs to be reflected in other object without tight coupling

**Subject and observers:-**

* Subject is an object which takes the responsibility to notify the observers of any changes e.g. database changes, property change etc.
* Observer is an object listening to subject changes

e.g :-

//subject interface

internal interface ISubject

{

void Register(IObserver observer);

void UnRegister(IObserver observer);

void Notify();

}

//observer interface

internal interface IObserver

{

void Update(double price);

}

//concrete subject

internal class ServiceSubject : ISubject

{

private double \_price;

private List<IObserver> listObservers = new List<IObserver>();

public double price

{

get { return \_price; }

set

{

if (\_price != value)

{

\_price = value;

Notify();

}

}

}

public void Notify()

{

listObservers.ForEach(x => x.Update(price));

}

public void Register(IObserver observer)

{

listObservers.Add(observer);

}

public void UnRegister(IObserver observer)

{

listObservers.Remove(observer);

}

}

//concrete observer

internal class ConcreteObserver : IObserver

{

private string \_name;

public ConcreteObserver(string name)

{

this.\_name = name;

}

public void Update(double price)

{

Console.WriteLine($"Notified: updated price of stock {\_name} is {price}");

Console.ReadLine();

}

}

//Main class

public static void Main(string[] args)

{

ServiceSubject serviceSubject = new ServiceSubject();

// Add stocks

serviceSubject.Register(new ConcreteObserver("IBM"));

serviceSubject.Register(new ConcreteObserver("Google"));

serviceSubject.Register(new ConcreteObserver("Reliance"));

//Stock prices update

serviceSubject.price = 100;

serviceSubject.price = 200;

}