Design Pattern

Monday, February 10, 2025 8:45 AM

(GOF) -- Gang of Four -- categorized the design pattern into three main categories based on the three problem areas :

Creational Design Pattern : Object Creation and Initialization : Singleton , Factory , Builder

These patterns deal with object creation mechanism , trying to create objects in a manner suitable to the situation . Creational design patterns solve this problem by controlling the object creation process
(IOC)

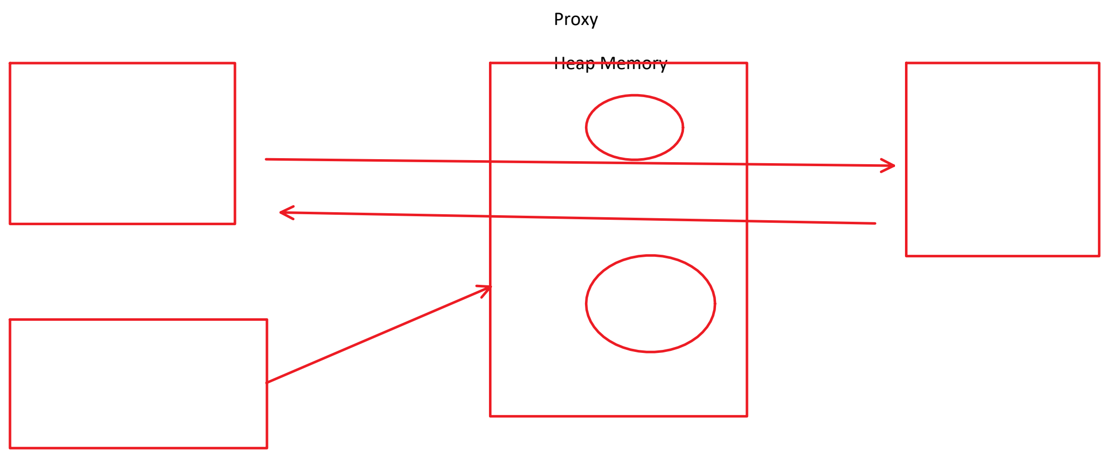
A lot of classes means we are dealing with a lot of objects . So we need to create different object(like new customer(), new Invoice() ,ne
Product()………. Creational design pattern also helps us to centralize the object , creation and initialization logic)

Structural Design Pattern : Structural changes of relation between classes and Interfaces . They help ensure that the entire structure doesn't need to change when one part of a system changes.

: **Adapter , Façade , Decorator , Composite , Proxy**

Behavioural Design Pattern : The relationship between classes and communication between Objects : These patterns are focused on communication between objects : How they interact and fulfil the requirement . They define clear patterns of communication among objects :

Chain of Responsibility design pattern, Observer design pattern, Strategy design pattern



Singleton Design Pattern :

It ensures that only one instance of the Singleton class is created throughout the application
We can do the lazy initialization - > which means it is created when it is needed for the first time , not when the application starts.
Eager initialization : The object is ready when you execute your application , no matter you needed it or not there

```
public sealed class Singleton
{
   private static int cntr = 0;

   private static Singleton Instance = null;

   public static Singleton GetInstance()
   {
      if (Instance == null)
      {
       Instance = new Singleton();
      }
      return Instance;

   }

   private Singleton() {
      cntr++;
      Console.WriteLine("Counter Value :" + cntr.ToString());
```

```
    }

    public void Display(string message) {


        Console.WriteLine(message);


            }


}



using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DesignPattern
{
    public class MainClass
    {
        public static void Main(string[] args)
        {
         //  Singleton s = new Singleton();

            Singleton user1 = Singleton.GetInstance();
            user1.Display("Data fetched by user 1");

            Singleton user2 = Singleton.GetInstance();
            user2.Display("Data fetched by user 2");

        }

    }
}
```

2. Factory Pattern -- allows to create objects without specifying their exact class , making the code more maintainable and flexible

Let's say we have a mobile phone store where customers can buy diff types of phone like android , iphone

3. Builder Pattern :

Laptop : add ons --- usp ports , hard drive , memory , battery ,, keyboad (Configuring)

Create Laptop
{

Laptop(usb , hard drive , battery , memory , keyboard)
}

Structural Design Pattern :

Adapter Pattern : converts one interface into another as per the requirement

A legacy system that outputs data in XML format but a new system expects JSON,, A legacy system refers to an old or outdated softwar
or hardware that is still in use because it is important for the organization

Phone:

ChargerAmerican -- Legacy (OLD) -- -2 pin socket
Indian Socket --- Modern (New) --- 3 pin

ChargerAdaptor --- Conversion

Decorator Pattern : decorating the base

Adds the behaviour to an object dynamically without modifying its structure

Use Case : When you need flexibility in extending the functionalities

Behavioural Design Pattern

Observer Pattern

Notifies multiple object when the state of one object changes
Use Case is -- News subscription

Where we can save all the multiple objects -- Collections