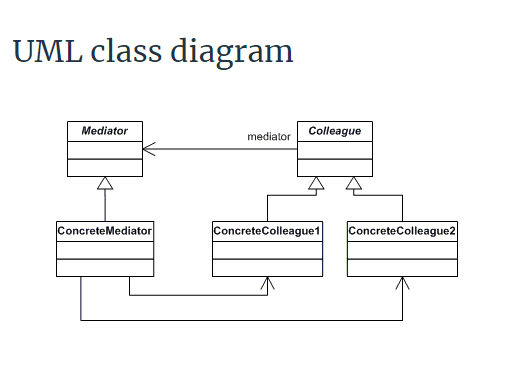
**Section17 Mediator Design Pattern**

**Notes: -**

**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.**



**(So when we want to call ConcreteShape1 that it indirectly call Mediator function because we reference the same instance to each other as above)**

Concrete Shape 1

Mediator

Concrete Shape 2

**(so the main of using Mediator design pattern when you want to provide loosely coupled on the Concrete Shape and Mediator and make Concrete Shape call the mediator and do some operation and then call the Concrete Shape)**

**Example: -**

**//1-In the below example we see that the abstract class and his children contains the Mediator**

**//to assign the Mediator instance inside the child of Colleague**

**//2-and on the ConcreteMediator it contains the whole child classes as below**

**namespace DoFactoryMediatorPro.Mediator{**

**abstract class Colleague{**

**protected Mediator mediator;**

**public Colleague(Mediator mediator){**

**this.mediator = mediator;}}}**

**using System;**

**namespace DoFactoryMediatorPro.Mediator{**

**class ConcreteColleague1 : Colleague{**

**public ConcreteColleague1(Mediator mediator): base(mediator){}**

**public void Send(string message){mediator.Send(message, this);}**

**public void Notify(string message){Console.WriteLine("Colleague1 gets message: "+ message);}}}**

**using System;**

**namespace DoFactoryMediatorPro.Mediator{**

**class ConcreteColleague2 : Colleague{**

**public ConcreteColleague2(Mediator mediator): base(mediator){}**

**public void Send(string message){mediator.Send(message, this);}**

**public void Notify(string message){**

**Console.WriteLine("Colleague2 gets message: "+ message);}}}}**

**Example: -**

**using DoFactoryMediator2Pro.Mediator;**

**using System;**

**namespace DoFactoryMediator2Pro.Command{**

**//on this abstract class we have instnace of the Chartoom Mediator Concrete which used to call it //through the child class**

**class Participant{**

**private Chatroom \_chatroom;**

**private string \_name;**

**public Participant(string name){this.\_name = name;}**

**public string Name{get { return \_name; }}**

**public Chatroom Chatroom{set { \_chatroom = value; }get { return \_chatroom; }}**

**// Sends message to given participant**

**public void Send(string to, string message){\_chatroom.Send(\_name, to, message);}**

**// Receives message from given participant**

**public virtual void Receive(string from, string message){**

**Console.WriteLine("{0} to {1}: '{2}'",from, Name, message);}}}**

**using System;**

**namespace DoFactoryMediator2Pro.Command{**

**class NonBeatle : Participant{**

**public NonBeatle(string name): base(name){}**

**public override void Receive(string from, string message){**

**Console.Write("To a non-Beatle: ");**

**base.Receive(from, message);}}}**

**using System;**

**namespace DoFactoryMediator2Pro.Command{**

**class Beatle : Participant{**

**public Beatle(string name): base(name){}**

**public override void Receive(string from, string message){**

**Console.Write("To a Beatle: ");**

**base.Receive(from, message);}}}**

**using DoFactoryMediator2Pro.Command;**

**namespace DoFactoryMediator2Pro.Mediator{**

**abstract class AbstractChatroom{**

**public abstract void Register(Participant participant);**

**public abstract void Send(**

**string from, string to, string message);}}**

**using DoFactoryMediator2Pro.Command;**

**using System.Collections.Generic;**

**namespace DoFactoryMediator2Pro.Mediator{**

**//on the Concrete Mediator we have two methods Register which register the Participant inside the Dictionary<string,Participatn> internally**

**//and the another is send which used to send to target**

**class Chatroom : AbstractChatroom{**

**private Dictionary<string, Participant> \_participants =**

**new Dictionary<string, Participant>();**

**public override void Register(Participant participant){**

**//it will assign the new Participant on the Mediator**

**if (!\_participants.ContainsValue(participant)){\_participants[participant.Name] = participant;}**

**//it will assign the Chatromm with the instance itself**

**participant.Chatroom = this;}**

**public override void Send(string from, string to, string message){**

**Participant participant = \_participants[to];**

**if (participant != null){participant.Receive(from, message);}}}}**