**DoFactory Chain of Responsibility Design Pattern**

**Notes: -**

**1-Chain of responsibility: Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.**

**Apply Chain of responsibility with using register and execute all with optimize way**

**Example: -**

**using System.Collections.Generic;**

**using System.Collections.ObjectModel;**

**namespace DoFactoryChainOfResp.Handlers{**

**public class Handler{**

**protected int request;**

**private ICollection<Handler> handlers;**

**public Handler(){handlers = new Collection<Handler>();}**

**public void Register(Handler handlerType){handlers.Add(handlerType);}**

**public void ExecuteAll(){**

**foreach (var handler in handlers){handler.HandleRequest(handler.request);}}**

**public virtual void HandleRequest(int request) { }}}**

**//Sub class Handler**

**using System;**

**namespace DoFactoryChainOfResp.Handlers{**

**class ConcreteHandler1 : Handler{**

**public ConcreteHandler1(int request){base.request = request;}**

**public override void HandleRequest(int request){**

**if (request >= 0 && request < 10){**

**Console.WriteLine("{0} handled request {1}",**

**this.GetType().Name, request);}}}}**

**using System;**

**namespace DoFactoryChainOfResp.Handlers{**

**class ConcreteHandler2 : Handler{**

**public ConcreteHandler2(int request){base.request = request;}**

**public override void HandleRequest(int request){**

**if (request >= 10 && request < 20){**

**Console.WriteLine("{0} handled request {1}",**

**this.GetType().Name, request);}}}}**

**using System;**

**namespace DoFactoryChainOfResp.Handlers{**

**class ConcreteHandler3 : Handler{**

**public ConcreteHandler3(int request){base.request = request;}**

**public override void HandleRequest(int request){**

**if (request >= 20 && request < 30){**

**Console.WriteLine("{0} handled request {1}",**

**this.GetType().Name, request);}}}}**

**using DoFactoryChainOfResp.Handlers;**

**using System;**

**namespace DoFactoryChainOfResp{**

**//chain of responsibility: Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request.**

**//Chain the receiving objects and pass the request along the chain until an object handles it.**

**//this way is preferred because you combine the register and calling them on the same line as below**

**class Program{**

**static void Main(string[] args){**

**// Setup Chain of Responsibility**

**var handler = new Handler();**

**handler.Register(new ConcreteHandler1(1));**

**handler.Register(new ConcreteHandler2(11));**

**handler.Register(new ConcreteHandler3(21));**

**handler.ExecuteAll();**

**Console.ReadKey();}}}**

**Example: -**

**using DoFactory2ChainOfResp.Models;**

**//we make field that refer to the same class that all sub classes inherit from it and using it to call abstract method as below**

**namespace DoFactory2ChainOfResp.Handlers{**

**abstract class Approver{**

**//we make instance of the successor and using as parameter to assign to the Approver class and //using it to call the ProcessRequest method**

**protected Approver successor;**

**public void SetSuccessor(Approver successor){this.successor = successor;}**

**public abstract void ProcessRequest(Purchase purchase);}}**

**using DoFactory2ChainOfResp.Models;**

**using System;**

**namespace DoFactory2ChainOfResp.Handlers{**

**class Director : Approver{**

**//on each child class it using the successor which is the Approver instance**

**public override void ProcessRequest(Purchase purchase){**

**if (purchase.Amount < 10000.0){**

**Console.WriteLine("{0} approved request# {1}",**

**this.GetType().Name, purchase.Number);}**

**else if (successor != null){**

**successor.ProcessRequest(purchase);}}}}**

**using DoFactory2ChainOfResp.Models;**

**using System;**

**namespace DoFactory2ChainOfResp.Handlers{**

**class President : Approver{**

**//on each child class it using the successor which is the Approver instance**

**public override void ProcessRequest(Purchase purchase){**

**if (purchase.Amount < 100000.0){**

**Console.WriteLine("{0} approved request# {1}",**

**this.GetType().Name, purchase.Number);}**

**else{**

**Console.WriteLine("Request# {0} requires an executive meeting!",**

**purchase.Number);}}}}**

**using DoFactory2ChainOfResp.Models;**

**using System;**

**namespace DoFactory2ChainOfResp.Handlers{**

**//on each child class it using the successor which is the Approver instance**

**class VicePresident : Approver{**

**public override void ProcessRequest(Purchase purchase){**

**if (purchase.Amount < 25000.0){**

**Console.WriteLine("{0} approved request# {1}",**

**this.GetType().Name, purchase.Number);}**

**else if (successor != null){successor.ProcessRequest(purchase);}}}}**

**namespace DoFactory2ChainOfResp.Models{**

**public class Purchase{**

**private int \_number;**

**private double \_amount;**

**private string \_purpose;**

**// Constructor**

**public Purchase(int number, double amount, string purpose){**

**this.\_number = number;**

**this.\_amount = amount;**

**this.\_purpose = purpose;}**

**// Gets or sets purchase number**

**public int Number{get { return \_number; }set { \_number = value; }}**

**// Gets or sets purchase amount**

**public double Amount{get { return \_amount; }set { \_amount = value; }}**

**// Gets or sets purchase purpose**

**public string Purpose{get { return \_purpose; }set { \_purpose = value; }}}}**

**using DoFactory2ChainOfResp.Handlers;**

**using DoFactory2ChainOfResp.Models;**

**using System;**

**namespace DoFactory2ChainOfResp{**

**//this way is not prefered becouse you are not register these sub classes and call them all on the same line**

**class Program{**

**static void Main(string[] args){**

**// Setup Chain of Responsibility**

**Approver larry = new Director();**

**Approver sam = new VicePresident();**

**Approver tammy = new President();**

**//we assign the instance of the child into the partent**

**larry.SetSuccessor(sam);**

**sam.SetSuccessor(tammy);**

**// Generate and process purchase requests which make process to the same instance multiple stages**

**Purchase p = new Purchase(2034, 350.00, "Assets");**

**larry.ProcessRequest(p);**

**p = new Purchase(2035, 32590.10, "Project X");**

**larry.ProcessRequest(p);**

**p = new Purchase(2036, 122100.00, "Project Y");**

**larry.ProcessRequest(p);**

**// Wait for user**

**Console.ReadKey();}}}**