**Mediator Pattern**

**Ciao, appassionati di programmazione! Bentornati a .Net Core Central. Nel post del blog di oggi, ci addentreremo nell'intrigante regno del design pattern Mediator. Questo design pattern comportamentale funge da potente strumento per gestire le interazioni tra un set di oggetti, promuovendo un accoppiamento debole e una variazione indipendente delle loro interazioni.**

**Understanding the Mediator Pattern**

**L'essenza del pattern Mediator risiede nell'avere un oggetto che incapsula la comunicazione tra un gruppo di oggetti. In questo modo, elimina i riferimenti diretti tra questi oggetti, favorendo un accoppiamento debole. Similmente ad altri pattern comportamentali, l'obiettivo primario qui è stabilire flessibilità nelle interazioni tra più oggetti.**

**Real-world Example: Call Center-based Cab Booking System**

**Per illustrare il modello Mediator, prendiamo in considerazione uno scenario reale: un sistema di prenotazione taxi basato su call center. Prima delle app di ride-sharing come Uber, prenotavamo i taxi tramite call center. Ogni taxi si registrava presso il call center e quando un passeggero chiamava per prenotare un taxi, il call center gestiva la comunicazione tra il passeggero e i taxi disponibili.**

**Implementation of Mediator Pattern in .NET Core**

**Per la nostra dimostrazione, creeremo tre classi: CallCenter, Cab e Passenger. La classe CallCenter è responsabile della registrazione dei taxi e della gestione delle prenotazioni dei taxi. I taxi e i passeggeri, a loro volta, interagiscono con il call center anziché comunicare direttamente tra loro.**

**Cab Class**

**La classe Cab rappresenta un taxi, con proprietà quali nome, posizione e disponibilità. I ​​taxi si registrano presso il call center e possono essere assegnati ai passeggeri entro un certo raggio.**

**namespace MediatorPattern.Demo;**

**internal interface ICab**

**{**

**string Name { get; }**

**int CurrentLocation { get; }**

**bool IsFree { get; }**

**void Assign(string name, string address);**

**}**

**internal class Cab : ICab**

**{**

**private readonly string \_name;**

**private readonly int \_location;**

**private readonly bool \_free;**

**public Cab(string name, int location, bool free)**

**{**

**\_name = name;**

**\_location = location;**

**\_free = free;**

**}**

**public string Name => \_name;**

**public int CurrentLocation => \_location;**

**public bool IsFree => \_free;**

**public void Assign(string name, string address) =>**

**Console.WriteLine($"Cab {Name}, assigned to passenger: {name}, {address}");**

**}**

**CallCenter Class**

**La classe CallCenter gestisce un dizionario di taxi registrati e gestisce il processo di prenotazione. Controlla i taxi disponibili entro un certo raggio dalla posizione del passeggero e assegna il taxi di conseguenza.**

**namespace MediatorPattern.Demo;**

**internal interface ICabCallCenter**

**{**

**void Register(ICab cab);**

**void BookCab(IPassenger passenger);**

**}**

**internal class CabCallCenter : ICabCallCenter**

**{**

**private readonly Dictionary<string, ICab> cabs**

**= new Dictionary<string, ICab>();**

**public void BookCab(IPassenger passenger)**

**{**

**foreach (var cab in cabs.Values.Where(c => c.IsFree))**

**{**

**if(IsWithin5MileRadius(cab.CurrentLocation, passenger.Location))**

**{**

**cab.Assign(passenger.Name, passenger.Address);**

**passenger.Acknowledge(cab.Name);**

**}**

**}**

**}**

**public void Register(ICab cab)**

**{**

**if (!cabs.ContainsValue(cab)) cabs.Add(cab.Name, cab);**

**}**

**private bool IsWithin5MileRadius(int cabLocation, int passengerLocation)**

**=> Math.Abs(cabLocation - passengerLocation) < 5;**

**}**

**Passenger Class**

**La classe Passenger rappresenta un passeggero, che contiene informazioni come nome, indirizzo e posizione. I passeggeri interagiscono con il call center per prenotare i taxi.**

**namespace MediatorPattern.Demo;**

**internal interface IPassenger**

**{**

**string Name { get; }**

**string Address { get; }**

**int Location { get; }**

**void Acknowledge(string name);**

**}**

**internal class Passenger : IPassenger**

**{**

**private string \_name;**

**private string \_address;**

**private int \_location;**

**public Passenger(string name, string address, int location)**

**{**

**\_name = name;**

**\_address = address;**

**\_location = location;**

**}**

**public string Name => \_name;**

**public string Address => \_address;**

**public int Location => \_location;**

**public void Acknowledge(string name) =>**

**Console.WriteLine($"Passenger {Name}, Cab: {name}");**

**}**

**Putting it Into Action**

**Istanziamo il CallCenter e creiamo istanze di Cab e Passenger. I taxi si registrano presso il call center e i passeggeri chiamano il centro per prenotare i taxi. Il call center gestisce quindi la comunicazione, assegnando i taxi disponibili ai passeggeri entro il raggio specificato.**

**var callCenter = new CabCallCenter();**

**var passenger1 = new Passenger("Passender1", "123 Street", 10);**

**var passenger2 = new Passenger("Passender2", "456 Street", 25);**

**var cab1 = new Cab("Cab1", 11, true);**

**var cab2 = new Cab("Cab2", 22, true);**

**callCenter.Register(cab1);**

**callCenter.Register(cab2);**

**callCenter.BookCab(passenger1);**

**callCenter.BookCab(passenger2);**

**Console.ReadLine();**