# Mediator – Посредник

**Посредник** (на [английски](http://bg.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8_%D0%B5%D0%B7%D0%B8%D0%BA): *Mediator*) е поведенчески [шаблон за дизайн](http://bg.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D0%B8_%D0%B7%D0%B0_%D0%B4%D0%B8%D0%B7%D0%B0%D0%B9%D0%BD), който се използва в [обектно-ориентираното](http://bg.wikipedia.org/wiki/%D0%9E%D0%B1%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%B0%D0%BD%D0%BE_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B8%D1%80%D0%B0%D0%BD%D0%B5) [програмиране](http://bg.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B8%D1%80%D0%B0%D0%BD%D0%B5).

Дефинира обект, който капсулира връзките между списък от обекти.

Медиаторът поддържа шаблона Разхлабени връзки( loose coupling) като предпазва директно обвързване между обектите, давайки възможност това да се случва на по-високо ниво.

|  |
| --- |
| // Mediator pattern -- Structural example |
| using System;    namespace DoFactory.GangOfFour.Mediator.Structural  {    /// <summary>    /// MainApp startup class for Structural    /// Mediator Design Pattern.    /// </summary>    class MainApp    {      /// <summary>      /// Entry point into console application.      /// </summary>      static void Main()      {        ConcreteMediator m = new ConcreteMediator();          ConcreteColleague1 c1 = new ConcreteColleague1(m);        ConcreteColleague2 c2 = new ConcreteColleague2(m);          m.Colleague1 = c1;        m.Colleague2 = c2;          c1.Send("How are you?");        c2.Send("Fine, thanks");          // Wait for user        Console.ReadKey();      }    }      /// <summary>    /// The 'Mediator' abstract class    /// </summary>    abstract class Mediator    {      public abstract void Send(string message,        Colleague colleague);    }      /// <summary>    /// The 'ConcreteMediator' class    /// </summary>    class ConcreteMediator : Mediator    {      private ConcreteColleague1 \_colleague1;      private ConcreteColleague2 \_colleague2;        public ConcreteColleague1 Colleague1      {        set { \_colleague1 = value; }      }        public ConcreteColleague2 Colleague2      {        set { \_colleague2 = value; }      }        public override void Send(string message,        Colleague colleague)      {        if (colleague == \_colleague1)        {          \_colleague2.Notify(message);        }        else        {          \_colleague1.Notify(message);        }      }    }      /// <summary>    /// The 'Colleague' abstract class    /// </summary>    abstract class Colleague    {      protected Mediator mediator;        // Constructor      public Colleague(Mediator mediator)      {        this.mediator = mediator;      }    }      /// <summary>    /// A 'ConcreteColleague' class    /// </summary>    class ConcreteColleague1 : Colleague    {      // Constructor      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);      }    }      /// <summary>    /// A 'ConcreteColleague' class    /// </summary>    class ConcreteColleague2 : Colleague    {      // Constructor      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);      }    }  } |

