1. Сформулируйте назначение и цель применения паттерна

Паттерн Посредник (Mediator) представляет такой шаблон проектирования, который обеспечивает взаимодействие множества объектов без необходимости ссылаться друг на друга. Тем самым достигается слабосвязанность взаимодействующих объектов.

1. Постройте UML-диаграмму паттерна и дайте пояснения основным элементам



Конкретный посредник помогает общаться конкретным коллегам. Например, менеджер проекта помогает общаться программисту и тестеру. Тогда, когда готова новая версия программы или получены новые результаты тестов, менеджер сообщает результаты одному из коллег.

1. Укажите области применения
2. Когда имеется множество взаимосвязанных объектов, связи между которыми сложны и запутаны.
3. Когда необходимо повторно использовать объект, однако повторное использование затруднено в силу сильных связей с другими объектами.
4. Особенности паттерна
5. Устраняется сильная связанность между объектами Colleague
6. Упрощается взаимодействие между объектами: вместо связей по типу "все-ко-всем" применяется связь "один-ко-всем"
7. Взаимодействие между объектами абстрагируется и выносится в отдельный интерфейс
8. Централизуется управления отношениями между объектами
9. Приведите примеры реализации

class Program

{

static void Main(string[] args)

{

ManagerMediator mediator = new ManagerMediator();

Colleague customer = new CustomerColleague(mediator);

Colleague programmer = new ProgrammerColleague(mediator);

Colleague tester = new TesterColleague(mediator);

mediator.Customer = customer;

mediator.Programmer = programmer;

mediator.Tester = tester;

customer.Send("Есть заказ, надо сделать программу");

programmer.Send("Программа готова, надо протестировать");

tester.Send("Программа протестирована и готова к продаже");

Console.Read();

}

}

abstract class Mediator

{

public abstract void Send(string msg, Colleague colleague);

}

abstract class Colleague

{

protected Mediator mediator;

public Colleague(Mediator mediator)

{

this.mediator = mediator;

}

public virtual void Send(string message)

{

mediator.Send(message, this);

}

public abstract void Notify(string message);

}

// класс заказчика

class CustomerColleague : Colleague

{

public CustomerColleague(Mediator mediator)

: base(mediator)

{ }

public override void Notify(string message)

{

Console.WriteLine("Сообщение заказчику: " + message);

}

}

// класс программиста

class ProgrammerColleague : Colleague

{

public ProgrammerColleague(Mediator mediator)

: base(mediator)

{ }

public override void Notify(string message)

{

Console.WriteLine("Сообщение программисту: " + message);

}

}

// класс тестера

class TesterColleague : Colleague

{

public TesterColleague(Mediator mediator)

: base(mediator)

{ }

public override void Notify(string message)

{

Console.WriteLine("Сообщение тестеру: " + message);

}

}

class ManagerMediator : Mediator

{

public Colleague Customer { get; set; }

public Colleague Programmer { get; set; }

public Colleague Tester { get; set; }

public override void Send(string msg, Colleague colleague)

{

// если отправитель - заказчик, значит есть новый заказ

// отправляем сообщение программисту - выполнить заказ

if (Customer == colleague)

Programmer.Notify(msg);

// если отправитель - программист, то можно приступать к тестированию

// отправляем сообщение тестеру

else if (Programmer == colleague)

Tester.Notify(msg);

// если отправитель - тест, значит продукт готов

// отправляем сообщение заказчику

else if (Tester == colleague)

Customer.Notify(msg);

}

}