[ГЛАВА1.МЕТРИКИОБЪЕКТНО-ОРИЕНТИРОВАННЫХ](#_bookmark1)

[ПРОГРАММНЫХСИСТЕМ 7](#_bookmark1)

* 1. [ВычислениеметриксредствамиVisualStudio 7](#_bookmark2)
  2. [МетрикиЧидамбераиКемерера 10](#_bookmark3)

[Вопросыкглаве 14](#_bookmark4)

[ГЛАВА2.РЕФАКТОРИНГПРОГРАММНОГО](#_bookmark5)

[ОБЕСПЕЧЕНИЯ 15](#_bookmark5)

[Вопросыкглаве 28](#_bookmark6)

[ГЛАВА3.ПАТТЕРНЫПРОЕКТИРОВАНИЯ](#_bookmark7)

[ПРОГРАММНЫХСИСТЕМ 29](#_bookmark7)

* 1. [Классификацияпаттернов 29](#_bookmark8)
  2. [Каталогпаттерновпроектирования 30](#_bookmark9)
  3. [Какиспользоватьпаттерны 32](#_bookmark10)
  4. [Примериспользованияпаттернов 33](#_bookmark11)

[Вопросыкглаве 37](#_bookmark12)

[ГЛАВА4.МОДУЛЬНОЕТЕСТИРОВАНИЕ](#_bookmark13)

[ПРОГРАММНОГООБЕСПЕЧЕНИЯ 38](#_bookmark13)

* 1. [Омодульномтестировании 38](#_bookmark14)
  2. [СозданиемодульноготестасредствамиVisualStudio 40](#_bookmark15)
  3. [ПространствоименMicrosoft.VisualStudio.TestTools.](#_bookmark16)

[UnitTesting 48](#_bookmark16)

[Вопросы кглаве 53](#_bookmark17)

# ГЛАВА 1. МЕТРИКИ ОБЪЕКТНО-ОРИЕНТИРОВАННЫХПРОГРАММНЫХСИСТЕМ

Метрика программного обеспечения (англ.*Software metric*) – этонекая мера определенного свойства программного обеспечения илиже его спецификаций. Метрики кода дают количественные оценкиразличныххарактеристиккода,атакжепозволяютопределитьсложностькодаиегоизолированныеобласти,которыемогутпривестикпроблемамприсопровожденииприложения.

Объектно-ориентированныеметрикивводятсясцелью:

* улучшитьпониманиекачествапродукта;
* оценитьэффективностьпроцессаконструирования;
* улучшитькачествоработынаэтапепроектирования.

# ВычислениеметриксредствамиVisualStudio

*Microsoftпредоставляет*встроенноев*VisualStudioсредство,*котороепозволяетоценитькодвашегопроекта (этафункциональностьдоступнав*VisualStudio2008TeamSystemиVisual Studio*2010, начиная с*Premium*версии). Получить оценку кодаможно,нажавправойкнопкойнапроектеивыбравпункт

«Рассчитатьметрикикода»(рисунок 1.1).

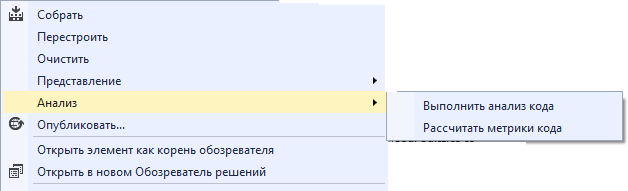


Рисунок 1.1.Окно «Рассчитатьметрикикода»

Результатысодержат5метриквашегокодарисунок1.2.

Рассмотрим,какрассчитываютсяэтиметрики:

Сложностьорганизациициклов (CyclomaticComplexity)‒метрика показывает структурную сложность кода, т. е. количестворазличных ветвей в коде. Чем больше этот показатель, тем большетестовдолжно бытьнаписано дляполногопокрытиякода.

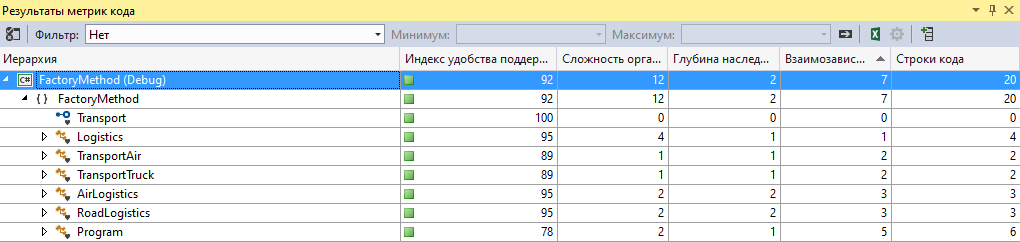


Рисунок1.2.Окно «Результатыметриккода»

Глубинанаследования (DepthofInheritance)‒метрикапоказываетдлякаждогокласса,какойонпосчетувцепочкенаследования. Например, есть 3 класса A, B, C. Класс B унаследованот А, а класс С унаследован от В, то значение этой метрики дляклассовA,BиC будет равно соответственно1, 2и3.

Взаимозависимостьклассов (ClassCoupling)–показываетстепеньзависимостиклассовдругсдругом.Хорошийдизайнпрограммногообеспеченияпредполагаетнебольшоеколичествосвязанных классов. Чем их больше, тем сложнее в дальнейшем ещераз использовать этот класс, а также поддерживать, т. к. существуеточеньмногозависимостей.

Например,нарисунке1.3класс«*Account*» связансдвумядругими классами «*Address*» и «*Order*», тогда как класс «*Сountry*» независитотдругихклассов.

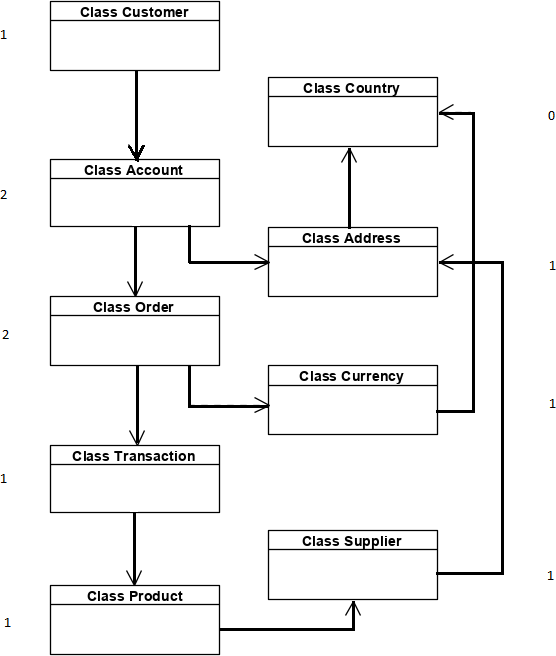


Рисунок1.3.Взаимозависимостьклассов

Количество строк кода (Lines of Code)– показывает количествостроккода.Этотпоказательпоказываетнеточноеколичествострокввашемфайле,т.к.врасчетнеберутсяпустыестрочки,комментарии, строчки со скобками, объявление типов и пространствимен. Большое количество строк в методе/классе может показыватьна ошибки в проектировании и на то, что этот код можно разделитьнанесколькочастей.

Индексудобстваподдержки(MaintainabilityIndex)–комплексный показатель качества кода. Этот показатель разработанспециалистами из Carnegie Mellon Software Engineering Institute. Вформулурасчетаэтойметрикивходятследующие метрики:

* вычислительная сложность(Halstead Volume) – чем большеоператоров,тембольшезначениеэтойметрики;
* количествостроккода;
* сложностьорганизациициклов.

Этаметрикаможетприниматьзначенияот0до100ипоказывает относительную сложность поддержки кода. Чем большезначениеэтойметрики,темлегчеподдерживатькод.Низкоезначениеметрикиуказываетнасложныйкод,которыйсложноподдерживать.Столбец«Индексудобстваподдержки»такжесодержитзначок,которыйдаетбыстроеуказаниенаобщуюремонтопригодностьисложностьконкретногоэлемента.

Втаблице1.1показандиапазон,прикоторомотображаетсязначок:

Таблица1.1

Степеньремонтопригодностикода

|  |  |  |
| --- | --- | --- |
| Значок | Уровень | Спектр |
| зеленый | Высокаяремонтопригодность | От20до100включительно |
| желтый | Умереннаяремонтопригодность | От10до 19включительно |
| красный | Низкаяремонтопригодность | От0до9включительно |

Эти значки позволяют увидеть проблемные места, на которыхнужносфокусироватьсяилиисправлятьошибки.

# МетрикиЧидамбераиКемерера

В 1994 году Чидамбер и Кемерер предложили шесть проектныхметрик,ориентированныхнаклассы.Нижеприведеныописаниярасчетапоказателейметрикикритерииоценки,введенныедлякаждойметрики[2].

Взвешенныеметодынакласс(WeightedMethodsperClass)–подсчитываетсяколичествометодоввклассе,атакжесложностькаждогометода(например,цикломатическаясложность),затем

суммируетсяпоказательсложностивсехметодоввклассе.Численныйпоказательсложностидолженбытьразумнонизким.Можноиспользоватьупрощенныйспособрасчета,вэтомслучаеподсчитывается толькоколичество методов в классе, чем меньшеметодовсодержиткласс,темвышеего оценкакачества.

Высотадереванаследования(DepthofInheritanceTree)–значением метрики для класса будет длина максимального пути отданного класса до корневого класса в иерархии классов. С однойстороны,большоезначениеэтойметрикиуказываетнавысокуюсложность проекта (в связи с использованием большого количестваклассов и методов), с другой стороны, подразумевает, что многиеметоды могут использоваться многократно, что является хорошимпоказателем.

Количестводетей(NumberOfChildren)–значениемэтойметрикибудет количествонепосредственныхклассов-наследниковрассматриваемогокласса.Сувеличениемколичестванаследниковвозрастаетмногократностьиспользованияметодовисвойствклассов,чтоявляетсяхорошейхарактеристикой,ноприэтомвозникаетрискнеправильногоиспользованиянекоторыхклассов-наследников.Поэтомузначениеданнойметрикидолжнобытьразумнонизким.

Сцепление между классами объектов (Coupling Between ObjectClasses) – рассчитывается количество классов, с которыми соединенрассматриваемый класс. Соединение образуется в результате вызоварассматриваемымклассомметодаилисвойствадругогокласса.Низкая величина сцепления для класса более предпочтительна, таккакбольшоеколичествосцепленийвлияетначувствительностьвсегопроектаклюбымизменениям:изменениеводноммодулеможетвестикнеобходимостиизмененийвомногихдругихмодулях,

все необходимые изменения трудно отследить. Это влечет за собойувеличениеколичестваошибок притестировании.

Откликдлякласса(ResponseForaClass)–прирасчетепоказателядляклассаиспользуетсяколичествособственныхметодов класса, а также количество методов, которые могут бытьвызваныизрассматриваемогокласса.Большоезначениеэтойметрики свидетельствует о высокой сложности класса, что негативносказываетсянапроцессетестированияиотладки.

Недостаток связности в методах (Lack of Cohesion in Methods) –значение метрики показывает, насколько в классе количество парметодов,несвязанныхпосвойствамкласса,преобладаетнадколичеством пар методов, связанных по свойствам класса. Высокаясвязность класса по методам является хорошей характеристикой, таккак это содействует инкапсуляции. Следовательно, высокое значениеэтойметрики,напротив,говорито низкойсвязности.

Пример расчета метрик Чидамбера и Кемерера для диаграммыUML,изображеннойнарисунке1.4,представленвтаблице1.2.

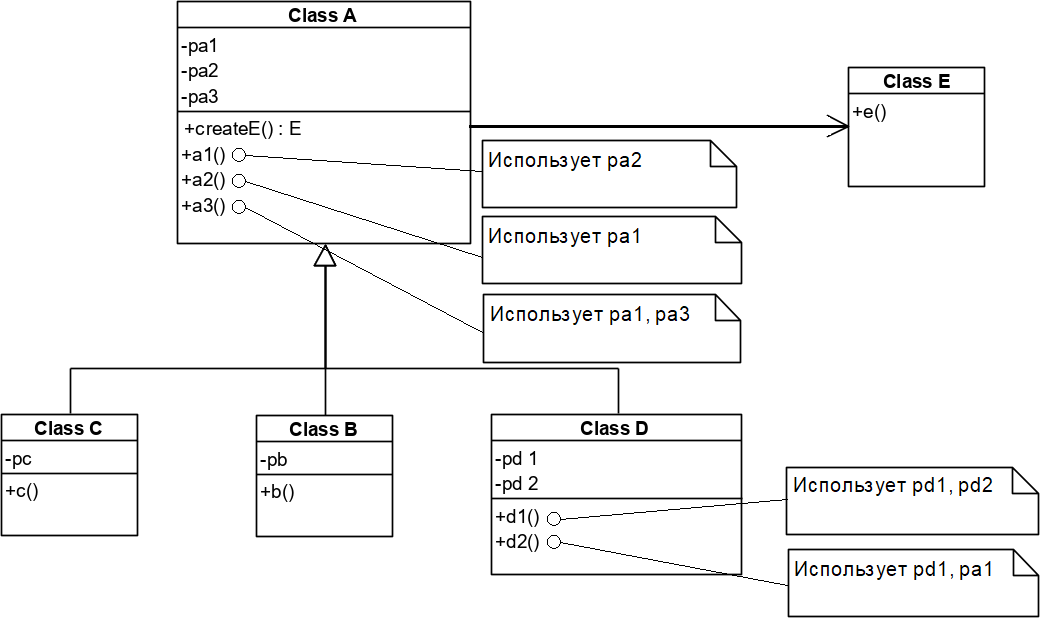


Рисунок1.4.ДиаграммаUMLдлярасчетаметрикЧидамбера иКемерера

Таблица1.2

Примеррасчетаметрик

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Имякласса | Взвешенныеметоды накласс | Высотадереванаследо-  вания | Количестводетей | Сцеплениемеждуклассами  объектов | Откликдлякласса | Недостатоксвязностивметодах |
| Class А | 4 | 0 | 3 | 1 | 5 | 4 |
| Class В | 1 | 1 | 0 | 0 | 1 | 0 |
| Class С | 1 | 1 | 0 | 0 | 1 | 0 |
| Class D | 2 | 1 | 0 | 0 | 2 | 0 |
| Class E | 1 | 0 | 0 | 0 | 1 | 0 |

Прокомментируем результаты расчета. Класс Class А имеет триметода(a1(),а2(),аЗ()),трехдетей(ClassВ,ClassС,ClassD)иявляется корневым классом. Поэтому метрики «взвешенные методынакласс»,«количестводетей»и«высотадереванаследования»имеют,соответственно,значения3,3и0.Метрика«сцеплениемежду классами объектов» для класса Class А равна 1, так как ониспользуетодинметодиздругогокласса(методе()изклассаClassE, он вызывается из метода сreateE()). Метрика «отклик для класса»для класса Class А равна 5, так как в ответ на прибытие в этот класссообщенийвозможновыполнениепятиметодов(четыреобъявленыв этом классе, а пятый метод е() вызывается из сreateE()). Рассчитаемметрику «недостаток связности в методах», поскольку в классе Ачетыреметода,возможнышестьпар:a1()&a2(),a1()&a3(),a1()&сreateE(),a2()&a3(), a2()&сreateE(), a3&сreateE().Методы a2()иa3()связаныпопеременнойpa1(),остальныенеимеютобщихпеременных.Такимобразом,количествонесвязанныхпарравнопяти, а связанных один.Следовательно, «недостаток связности вметодах»равен четырем.

# Вопросыкглаве:

* + 1. Охарактеризуйтекласспрограмм,длякоторыхпредложенныеметрикиявляютсянаиболееэффективными.
    2. Приведитенедостаткипредложенныхметрик.
    3. Какие свойства объектно-ориентированных программ неучитываютметрикиЧидамбераиКемерера?
    4. Что означают высокие значения метрики «недостатоксвязностивметодах»?

# ГЛАВА 2. РЕФАКТОРИНГ ПРОГРАММНОГООБЕСПЕЧЕНИЯ

Рефакторинг ([англ.](https://ru.wikipedia.org/wiki/Английский_язык)*refactoring*),или перепроектированиекода,переработкакода,равносильноепреобразованиеалгоритмов ‒процессизменениявнутреннейструктуры[программы](https://ru.wikipedia.org/wiki/Компьютерная_программа),незатрагивающий ее внешнего поведения и имеющий целью облегчитьпониманиеее работы.

В основе рефакторинга лежитпоследовательностьнебольшихэквивалентных (то есть сохраняющих поведение) преобразований.Посколькукаждоепреобразованиемаленькое,разработчикулегчепроследитьзаегоправильностью,ивтожевремявсяпоследовательностьможетпривестиксущественнойперестройкепрограммыиулучшению еесогласованности ичеткости.

Чтобы избегать разрушения программного обеспечения (ПО) иподдерживатьцелостностьархитектуры,необходимочастопроводить рефакторинг,чтобыбытьуверенным,чтокодостаетсячистымилаконичнымисминимальнымизависимостямимеждумодулями.Качествокодаиметрикипроектированияпредлагаютобъективныерекомендацииприопределенииучастковкодавприложении,которыеявляютсяпервоочередными кандидатамидля рефакторинга. Метрики покрытия позволяют выполнять работыпорефакторингу.

Проблемывкоде,требующиерефакторинга:

* [дублированиекода](https://ru.wikipedia.org/wiki/Дублирование_кода);
* длинныйметод;
* большойкласс;
* длинныйсписокпараметров;
* «жадные»функции‒этометод,которыйчрезмернообращаетсякданнымдругого объекта;
* избыточныевременныепеременные;
* классыданных;
* не сгруппированные данные.Методырефакторинга:
* Инкапсуляцияполя(EncapsulateField);
* Выделениекласса(ExtractClass);
* Выделениеинтерфейса(ExtractInterface);
* Выделениелокальнойпеременной(ExtractLocalVariable);
* Выделениеметода(ExtractMethod);
* Генерализациятипа(GeneralizeType);
* Встраивание(Inline);
* Введениефабрики(IntroduceFactory);

-Введениепараметра(IntroduceParameter);

* Подъемполя/метода(PullUp);
* Спускполя/метода(PushDown);
* Замена условного оператора полиморфизмом (Replacing aconditionaloperatorwith apolymorphism).

В системе*Visual Studio*существуют механизмы для проведениярефакторинга. Существует много способов вызвать инструменты длярефакторинга,включаяконтекстноеменю,котороеоткрываетсяпослещелчкаправойкнопкоймыши,интеллектуальныедескрипторыименюRefactor.

Нарисунке2.1показаноконтекстноеменюRefactor,предназначенноедляразработчиков,работающихнаязыкепрограммирования C#. Полный список действий по рефакторингу,доступных для разработчиков C# в системе*Visual Studio*, включаетоперации*«Переименовать»*,«*Извлечьметод»*,«*Инкапсулироватьполе»*,«*Извлечьинтерфейс»*,«*Удалитьпараметры»и*

«*Упорядочитьпараметры»*.Можнотакжеиспользоватьоперации

«*GenerateMethodStub»и*«*OrganizeUsings»*,которыеснекоторойнатяжкойтакжеможно отнести крефакторингу.

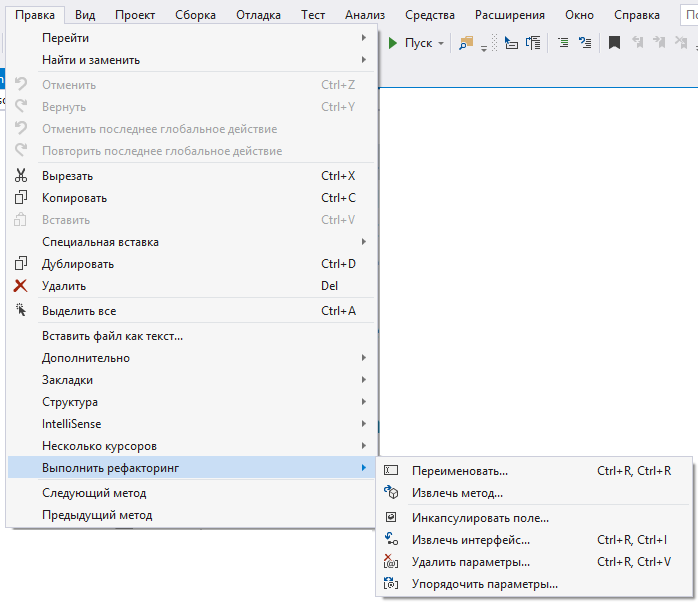


Рисунок 2.1.Контекстноеменю «Выполнитьрефакториг»

РассмотримпримерыиспользованиявстроеннойподдержкирефакторингавязыкеC#.

Чтобыначатьрефакторингдлинногометода,лучшевсегоразбить его на несколько небольших частей. Для этого необходимовыделить область кода, которую необходимо удалить из исходногометода,ивыбратьоперацию«*Извлечьметод»*.ВязыкеC#этоприведет к появлению приглашения ввести имя нового метода, какпоказанонарисунке2.2.Есливудаляемомблокекодаестьпеременные,которыеиспользовалисьранее,ониавтоматическистановятсяпеременнымивсигнатуреновогометода.Послеподтверждения имени нового метода его код вставляется сразу послеисходногометода.Удаленныйблокметодазаменяетсявызовомновогометода.

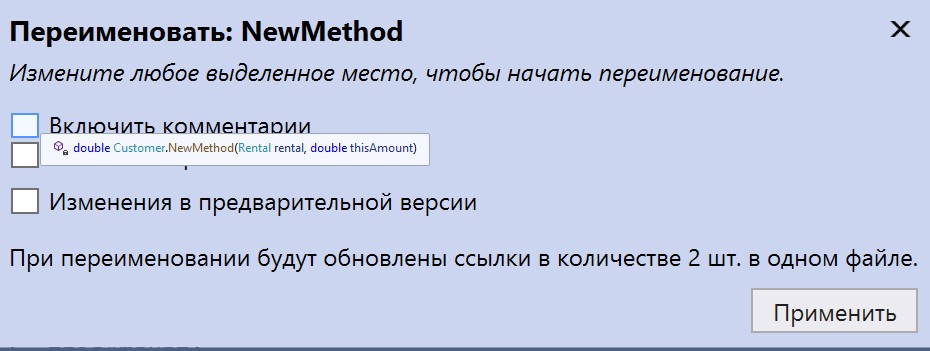


Рисунок2.2.Окно«Извлечьметод»

Вследующемпримере,длятогочтобывынестиусловноелогическое выражение в отдельный метод, необходимо выделить кодивыбрать операцию«*Извлечь метод»*.

publicstringStatement()

{

doubletotalAmount =0;

intfrequentRenterPoints=0;

stringresult=string.Format("Учетарендыдля{0}:",name);

foreach(Rentalrentalinrentals)

{

doublethisAmount =0;

// Выделите текст с этого местаswitch(rental.Movie.PriceCode)

{

caseMovie.Regular:

thisAmount+=2;

if(rental.DaysRented>2)

thisAmount += (rental.DaysRented - 2) \* 1.5;break;

caseMovie.New\_Release:thisAmount+=rental.DaysRented\*3;

break;

caseMovie.Childrens:if(rental.DaysRented> 3)

thisAmount += (rental.DaysRented - 3) \* 1.5;break;

}

//досюда,ивызовите команду«Извлечь метод»

/\* ... Продолжение длинного метода ... \*/frequentRenterPoints++;

if((rental.Movie.PriceCode == Movie.New\_Release) && (rental.DaysRented >1))

frequentRenterPoints++;

result +="\t"+ rental.Movie.Title +"\t"+ thisAmount;totalAmount+=thisAmount;

}

result+="Суммасоставляет"+totalAmount +"\n";

result +="Вы заработали"+ frequentRenterPoints +"очков за активность";returnresult;

}

publicstringStatement()

{

doubletotalAmount =0;

intfrequentRenterPoints=0;

stringresult=string.Format("Учетарендыдля{0}:",name);

foreach(Rentalrentalinrentals)

{

метода

doublethisAmount=0;

//Выделите текстсэтого места

thisAmount=NewMethod(rental,thisAmount);//вызовизвлеченного

//досюда,ивызовитекоманду«Извлечьметод»

/\* ... Продолжение длинного метода ... \*/frequentRenterPoints++;

if((rental.Movie.PriceCode == Movie.New\_Release) &&(rental.DaysRented > 1))

frequentRenterPoints++;

result +="\t"+ rental.Movie.Title +"\t"+ thisAmount;totalAmount +=thisAmount;

}

result+="Суммасоставляет"+ totalAmount+"\n";

result +="Вы заработали"+ frequentRenterPoints +"очков за активность";returnresult;

}

Врезультатеполучимновыйметод:

privatestaticdoubleNewMethod(Rentalrental,doublethisAmount)

{

switch(rental.Movie.PriceCode)

{

caseMovie.Regular:thisAmount+=2;

if(rental.DaysRented>2)

thisAmount += (rental.DaysRented - 2) \* 1.5;break;

caseMovie.New\_Release:

thisAmount += rental.DaysRented \* 3;break;

caseMovie.Childrens:

if(rental.DaysRented>3)

thisAmount += (rental.DaysRented - 3) \* 1.5;break;

}

returnthisAmount;

}

Прирефакторингечастотребуетсяинкапсулироватьвсуществующийкласснекоесвойство.Дляэтогоиспользуетсяоперация«*Инкапсулироватьполе»*.Длявыполнения

этого действия необходимо выбрать переменную, которую требуетсяинкапсулировать,ивыбратьсоответствующуюоперациювконтекстномменю,указавсвойствоивыбравместо,вкоторомследуетискатьссылку нанее.

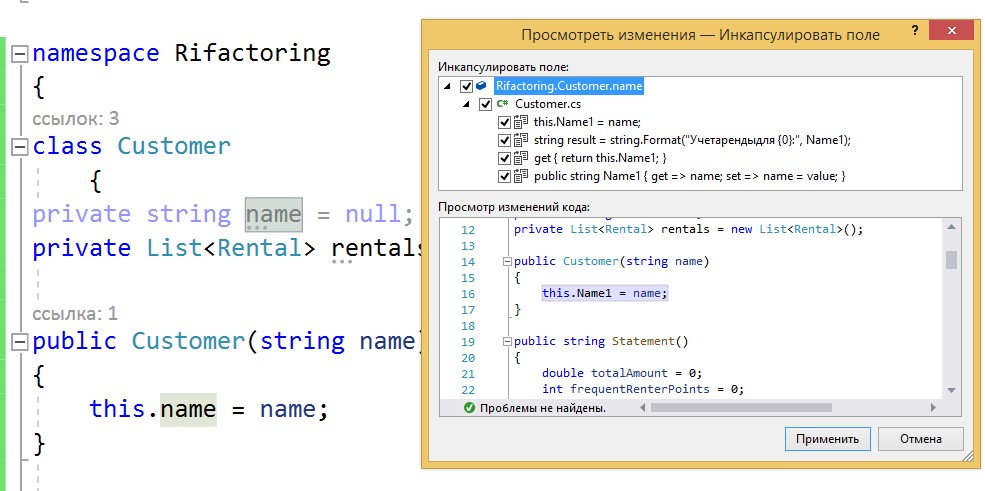


Рисунок2.3.Окно«Инкапсулироватьполе»

На следующем шаге после указания нового свойства следуетопределить, какие ссылки на переменную класса следует заменитьссылкойнановоесвойство.Откроетсяокнопредварительногопросмотрапозавершениипоискассылок,изображенноенарисунке

* 1. В верхней части окна размещено дерево, в котором необходимоуказать поля и методы, имеющие ссылки на данную переменную.Флажок,расположенныйпослекаждойстроки,указывает,следуетливыполнятьзамену.Выбравстрокувверхнейчастиокна,необходимоустановитьнанеефокусвнижнейпанели.Инкапсуляциясчитаетсязавершенной,еслиобеспеченакорректностькаждойссылки.Переменнаяклассаустанавливаетсязакрытойисоответствующиессылкиобновляются.

Длявыделенияосновныхметодоввинтерфейсиспользуетсяоперациярефакторинга«*Извлечьинтерфейс».Данная*операцияпозволяетизвлечьинтерфейс,используялюбоеколичествометодовв классе. Прежде для этого приходилось копировать весь метод вновыйфайлиудалятьегосодержимое,оставляялишьзаглушкуинтерфейса.Привыполненииэтойоперацииоткрываетсядиалоговое окно, показанное на рисунке 2.4, в котором предлагаетсявыбратьметод,которыйнеобходимовключитьвинтерфейс.Выбранныеметодыдобавляютсявновыйинтерфейс,которыйсоздаетсявновомфайле.Класс,изкоторогобылизвлеченинтерфейс,наследуетсяотэтогоинтерфейса.

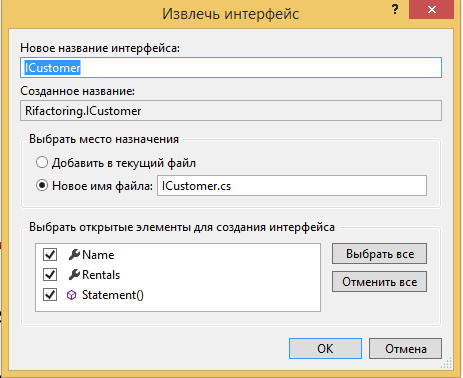


Рисунок2.4.Окно«Извлечьинтерфейс»

В следующем примере продемонстрировано извлечение вотдельныйинтерфейспервого метода:

classClass1

{

public voidShouldBeInInterface(floatd)

{/\*... \*/}

publicvoidNormalMethod\_1(inta,intb)

{/\*... \*/}

publicvoidNormalMethod\_2(intc)

{/\*... \*/}

}

Вновомфайлеформируетсяновыйинтерфейс:

interfaceIClass1

{

voidShouldBeInInterface(floatd);

}

Аисходныйфайлобновляетсяследующимобразом:

classClass1:IClass1

{

publicvoidShouldBeInInterface(floatd)

{/\*... \*/}

publicvoidNormalMethod\_1(inta,intb)

{/\*... \*/}

publicvoidNormalMethod\_2(intc)

{/\*... \*/}

}

Для переупорядочивания параметров используетсяоперация

«*ReorderParametersdialog»,которая*позволяетпереставлятьпараметрывспискевсоответствиистребуемымпорядком.Этаоперацияможетпомочьповыситьчитабельностькода,аиногдаявляетсянеобходимойприреализацииинтерфейсов.

При выполнении этой операции открывается диалоговое окно,показанноенарисунке.2.5.

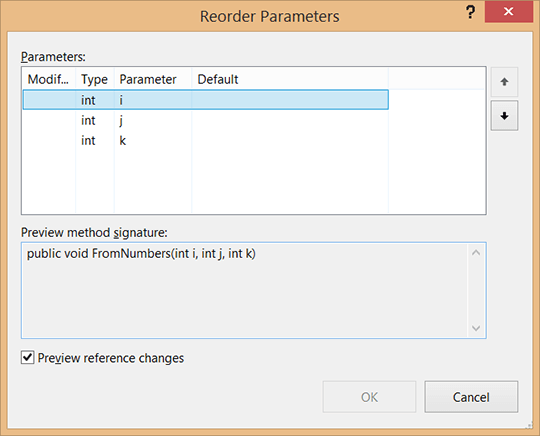


Рисунок2.5.Окно «ReorderParametersdialog»

Поумолчаниюпараметрывкаждомвызоведанногометодапереставляютсяавтоматическивсоответствиисновымпорядком.

Удаление параметра из метода с помощью операции «*Удалитьпараметры»значительно*сокращаетобъемпоискаошибоккомпиляции, которые могут при этом возникнуть. Кроме того, этаоперация очень полезна при многочисленных перегрузках метода,прикоторыхудалениепараметранеможетпорождатьошибкикомпиляции. В этом случае ошибка во время выполнения программыможет возникнуть только по семантической, а не по синтаксическойпричине.

Нарисунке2.6показанодиалоговоеокно«*Удалитьпараметры»*,котороеиспользуетсядляудаленияпараметровизсписка параметров. Если параметр был удален случайно, его легковосстановить. Как указывает предупреждение, размещенное в этомдиалоговомокне,удалениепараметровчастоприводиткнеожиданнымфункциональнымошибкам,поэтомуважноконтролироватьвнесенныеизменения.Длятогочтобыоценить

внесенные изменения, можно снова использовать окнопредварительногопросмотра.

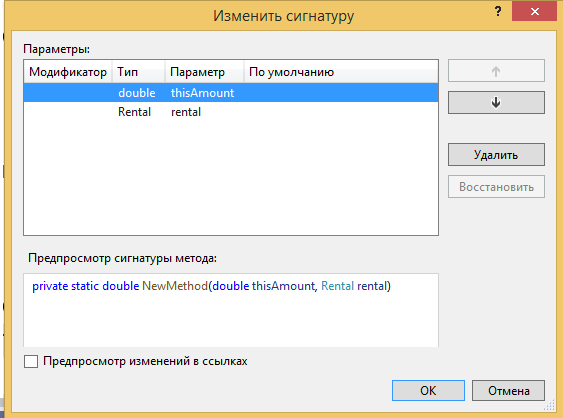


Рисунок2.6. Окно«Удалитьпараметры»

ОперацияRenameобеспечиваетпереименованиепеременных,методовиклассов.Диалоговоеокно«Rename»дляязыкаC#приведенонарисунке2.7.

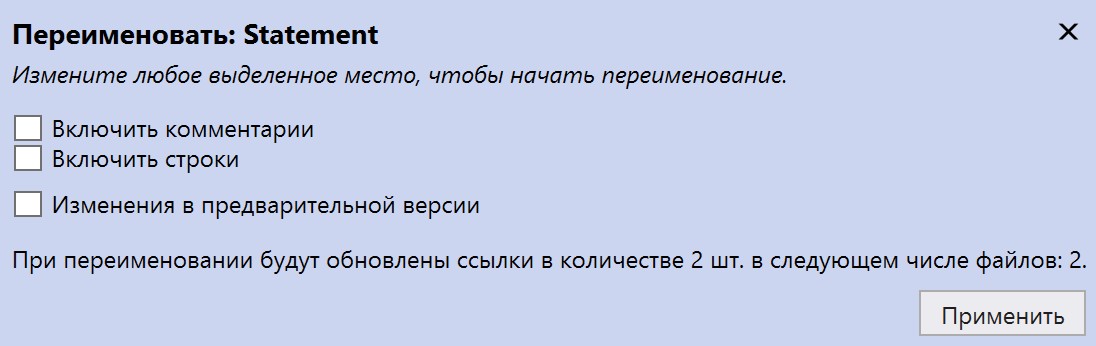


Рисунок2.7.Окно «Rename»

Операция«*GenerateMethodStub»используется,*когдануженвызовметода,которыйещененаписан.Следующийпример

иллюстрируетновыйметод,которыйдолженбытьсгенерированпозже.

publicvoidMethodA()

{

stringInputA;doubleInputB;

intOutputC=NewMethodIJustThoughtOf(InputA,InputB);

}

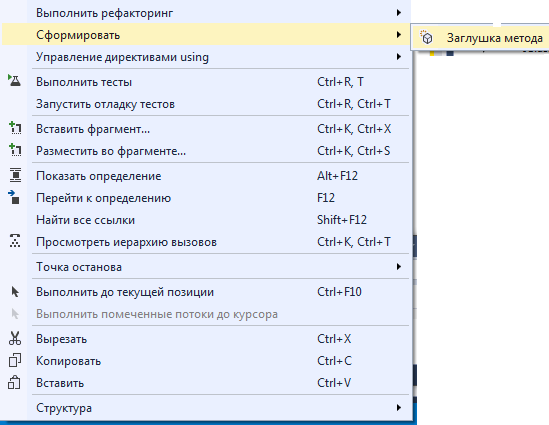


Рисунок2.8.Генерациязаглушкиметода

Этоткодпорождаетошибкуприсборке,посколькувызываемыйметодещенеопределен.Используя операциюрефакторинга«*GenerateMethodStub»*(доступнуюспомощьюинтеллектуального указателя в самом коде), можно сгенерироватьзаглушку метода рисунок 2.8. Как видно из следующего примера,заглушка метода содержит входные параметры и тип возвращаемогозначения.

publicvoidMethodA()

{

stringInputA;doubleInputB;

intOutputC=NewMethodIJustThoughtOf(InputA,InputB);

}

privateintNewMethodIJustThoughtOf(stringInputA,doubleInputB)

{

thrownewNotImplementedException();

}

Очень полезно поддерживать упорядоченный список директивusingвкаждомфайле(наязыкеC#)иссылатьсятольконатепространства имен, которые действительнонеобходимы в данномфайле. Операция «*Organize Usings»*, доступная в контекстном менюрисунок2.9,котороеоткрываетсяпослещелчкапользователемправой кнопкой мыши в окне редактора кода, поможет в обоих этихслучаях.

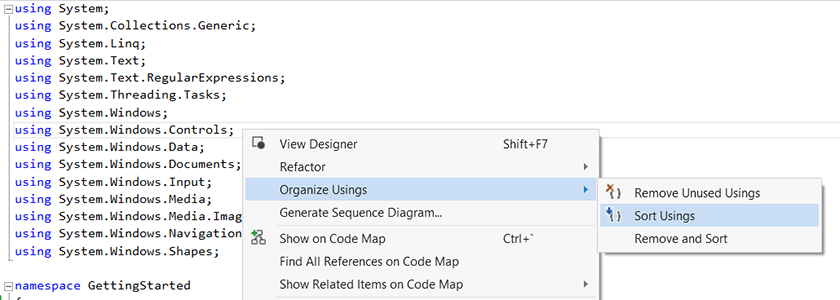


Рисунок2.9.Операция«OrganizeUsings»

Послерефакторингаможетвыясниться,чтовначалефайласодержитсямножестводирективusing,которыебольшенеиспользуются.Чтобыопределить,какиеизэтихдирективиспользуются,акакиенет,дляэтогодостаточнощелкнутьправой

кнопкой мыши в окне редактора кода, выбрать команду «*OrganizeUsings»и*далее«*RemoveUnusedUsings»*.Неиспользуемыедирективыusing,ихальтернативныеименаивнешниеальтернативныеименасборокизисходногофайлабудутудалены.

Оченьполезноупорядочиватьдирективыusingвалфавитномпорядке, чтобы легко управлять пространствами имен, на которыеониссылаются.Длятогочтобынеделатьэтовручную,нужнощелкнутьправойкнопкоймышивокнередакторакода,выбратькоманду «*Organize Usings»и*далее«*Sort Usings»*.

Если для пространства имен задано альтернативное имя, то онопереноситсявконецсписка,аесливпроектеиспользуютсяальтернативные имена внешних сборок (с помощью ключевого словаextern вязыке C#), то ониперемещаютсявначалосписка.

Для того чтобы одновременно упорядочить директивы using иудалитьнеиспользуемые,достаточнощелкнутьправойкнопкоймыши в окне редактора кода, выбрать команду «*Organize Usings»*идалее«*Remove» и«Sort»*.

# Вопросыкглаве:

* + 1. Перечислитеметодырефакторинга.
    2. Зачемнужнопроводитьрефакторинг?
    3. Когдаследуетпроводитьрефакторинг?
    4. Почемурефакторингприноситположительныерезультаты?
    5. Какрефакторингсвязанспроектированием?

# ГЛАВА 3.ПАТТЕРНЫ ПРОЕКТИРОВАНИЯПРОГРАММНЫХСИСТЕМ

Шаблонпроектирования, или паттерн (англ.*designpattern*),вразработкепрограммногообеспечения ‒этоповторноиспользующаяся архитектурная конструкция, представляющая собойрешениепроблемы проектирования врамкахнекоторогочастовозникающегоконтекста.

# Классификацияпаттернов

Паттерныпроектированияпрограммныхсистемделятсянаследующие категории:

* + - Архитектурныепаттерны;
    - Паттерныпроектирования;
    - Идиомы.

Архитектурные паттерны, являются наиболеевысокоуровневымипаттернамииописываютструктурнуюсхемупрограммнойсистемывцелом.Вданнойсхемеуказываютсяотдельныефункциональныесоставляющиесистемы,называемыеподсистемами,атакжевзаимоотношениямеждуними.Примеромархитектурногопаттерна являетсяхорошоизвестнаяпрограммнаяпарадигма«модель-представление-контроллер»(model-view-controller - MVC).

Подсистемы могут также состоять из архитектурных единицуровнемниже. Паттерныпроектирования описываютсхемыдетализации программных подсистем и отношений между ними, приэтом они не влияют на структуру программной системы в целом исохраняютнезависимостьотреализацииязыкапрограммирования.

Идиомы, являясь низкоуровневыми паттернами, имеют дело свопросами реализации какой-либо проблемы с учетом особенностейданногоязыкапрограммирования.Приэтомчастоодниитеже

идиомы для разных языков программирования выглядят по-разномуилинеимеютсмысла вовсе.

Также существуют и другие виды паттернов, не относящиеся кпроектированиювообще,например,паттерныанализа,тестирования,документированияидр.

В данном учебном пособии мы остановимся на рассмотрениитолько паттерновпроектирования.

# Каталогпаттерновпроектирования

Внастоящеевремянаиболеепопулярнымипаттернамиявляютсяпаттерныпроектирования.Однойизраспространенныхклассификаций таких паттернов является классификация по степенидетализациии уровнюабстракциирассматриваемыхсистем.

Паттернпредставляетопределенныйспособпостроенияпрограммногокодадлярешениячастовстречающихсяпроблемпроектирования.Вданномслучаепредполагается,чтоестьнекоторыйнаборобщихформализованныхпроблем,которыедовольночастовстречаются,ипаттерныпредоставляютрядпринциповдлярешенияэтих проблем.

Применениепаттерновсталосчитатьсяхорошейпрактикойпрограммирования.Принаписаниипрограмммыможемформализовать проблему в виде классов, объектов и связей междуними.Иприменитьодинизсуществующихпаттерновдляеерешения. В итоге нам не надо ничего придумывать. У нас уже естьготовый шаблон, и нам только надо его применить в конкретнойпрограмме.

Посвоемудействиюпаттерныможнообъединитьврядгрупп.

Восновеклассификацииосновныхпаттерновположенацельилизадачи, которыеопределенныйпаттернвыполняет.

**Порождающиепаттерны‒**этопаттерны,которыеабстрагируютпроцессинстанцированияили,инымисловами,процесспорожденияклассовиобъектов.

Ктакимшаблонамотносятся:

* Абстрактнаяфабрика(AbstractFactory);
* Строитель(Builder);
* Фабричныйметод(FactoryMethod);
* Прототип(Prototype);
* Одиночка(Singleton).

**Структурныепаттерны**‒рассматривают,какклассыиобъектыобразуютболеекрупныеструктуры-болеесложныепохарактеруклассыиобъекты.

Ктакимшаблонамотносятся:

* Адаптер(Adapter);
* Мост(Bridge);
* Компоновщик(Composite);
* Декоратор(Decorator);
* Фасад(Facade);
* Приспособленец(Flyweight);
* Заместитель(Proxy).

**Поведенческие паттерны‒** определяют алгоритмы ивзаимодействиемеждуклассамииобъектами,тоестьихповедение.

Ктакимшаблонамотносятся:

* Цепочкаобязанностей(Chainofresponsibility);
* Команда(Command);
* Интерпретатор(Interpreter);
* Итератор(Iterator);
* Посредник(Mediator);
* Хранитель(Memento);
* Наблюдатель(Observer);
* Состояние(State);
* Стратегия(Strategy);
* Шаблонныйметод(Templatemethod);
* Посетитель(Visitor).

Существуют и другие классификации паттернов в зависимостиоттого,относитсяпаттернкклассамилиобъектам.Паттерныклассовописываютотношениямеждуклассамипосредствомнаследования.Паттерныобъектов описываютотношениямеждуобъектами.

Различных шаблонов проектирования гораздо больше. Одни изнихтольконачинаютприменяться,другиеявляютсяпопулярнымина текущий момент, а некоторые уже менее распространены, чемраньше.

# Какиспользоватьпаттерны

Преждевсего,прирешениикакой-нибудьпроблемынадовыделитьвсеиспользуемыесущностиисвязимеждунимииабстрагировать их от конкретной ситуации. Затем надо посмотреть,вписывается ли абстрактная форма решения задачи в определенныйпаттерн.Например,сутьрешаемойзадачиможетсостоятьвсозданииновыхобъектов.Вэтомслучае,возможно,стоитпосмотретьнапорождающиепаттерны.Причемлучшенесразувзять какой-то определенный паттерн ‒ первый, который показалсянужным,апосмотретьнанесколькородственныхпаттерновизоднойгруппы,которыерешаютоднуитужезадачу.

При этом важно понимать смысл и назначение паттерна, явнопредставлятьегоабстрактнуюорганизациюиеговозможныеконкретныереализации.Одинпаттернможетиметьразличныереализации,ичемчащевыбудетесталкиватьсясэтимиреализациями,тем лучшевыбудетепониматьсмыслпаттерна.

# Примериспользованияпаттернов

Мы рассмотрим принципы использования патеров на примерепаттерна«Фабричныйметод»(FactoryMethod)применительнокязыкуC#.Болееподробноореализациииприменениидругихпаттерновможнопрочитатьвкниге«Приемыобъектно-ориентированногопроектирования.Паттерныпроектирования»авторов[Ральф Джонсон](https://www.litres.ru/r-dzhonson/),[Джон Влиссидес](https://www.litres.ru/dzhon-vlissides/),[Ричард Хелм](https://www.litres.ru/richard-helm/),[Эрих Гамма](https://www.litres.ru/erih-gamma/)[3].

**Фабричныйметод‒**этопаттерн,которыйопределяетинтерфейсдлясозданияобъектовнекоторогокласса,нонепосредственное решение отом, объект какогоклассасоздаватьпроисходит в подклассах. То есть паттерн предполагает, что базовыйклассделегирует созданиеобъектовклассам-наследникам.

Когданадоприменятьпаттерн:

* Когда заранее неизвестно, объекты, каких типов необходимосоздавать.
* Когдасистемадолжнабытьнезависимойотпроцессасоздания новых объектов и расширяемой: в нее можно легко вводитьновыеклассы,объектыкоторыхсистемадолжнасоздавать.
* Когда создание новых объектов необходимо делегировать избазовогоклассаклассамнаследникам.

Создадимпрограммууправлениягрузовымиперевозками.Сначалавырассчитываетеперевозитьтоварытольконаавтомобилях.Поэтомувесьвашкодработает с объектамикласса «*TransportTruck»*.Вкакой-томоментпоявляетсянеобходимость добавить в программу поддержку Авиа логистики.Большаячастьсуществующегокодапривязанакклассам «*TransportTruck»*, и для того чтобы добавить в программуклассы«*TransportAir»*, понадобится перелопатитьвсю программу.Более того, если вы потом решите добавить в программу еще одинвидтранспорта,товсюэтуработупридетсяповторить.Паттерн

«Фабричныйметод»предлагаетсоздаватьобъектыненапрямую,используя оператор*new*, а через вызовособого*фабричного*метода. Этодаетвозможностьпереопределитьфабричныйметодвподклассе,чтобыизменитьтипсоздаваемогопродукта. Чтобы эта система работала, все возвращаемые объектыдолжныиметьобщийинтерфейс.Подклассысмогутпроизводитьобъектыразличныхклассов,следующиходномуитомужеинтерфейсу.

Например,классы «*TransportTruck»и «TransportAir»реализуют интерфейсTransport*с методом*Deliver()*. Каждый из этихклассов реализует метод по-своему: грузовики везут грузы по земле,асамолетыповоздуху.Фабричныйметодвклассе«*RoadLogistics»вернет* объект-грузовик, акласс «*AirLogistics»*‒ объект-судно. Для клиента фабричного методанет разницы между этими объектами, так как он будет трактовать ихкак некий абстрактный «*Transport»*. Для него будетважно, чтобыобъектимелметод доставить,акакконкретноонработает ‒неважно[4].

Кодпрограммыбудетвыглядетьследующимобразом:

classProgram

{

staticvoidMain(string[]args)

{

Logisticsdev =newRoadLogistics("Грузовые перевозки");Transporttransport2=dev.Create();

dev =newAirLogistics("Авиа перевозки");Transporttransport = dev.Create();Console.ReadLine();

}

}

abstractclassLogistics

{

publicstringName{get;set;}

publicLogistics(stringn)

{

Name=n;

}

//фабричный метод

abstractpublicTransportCreate();

}

classRoadLogistics:Logistics

{

publicRoadLogistics(stringn)

:base(n)

{}

public overrideTransportCreate()

{

returnnewTransportTruck();

}

}

classAirLogistics:Logistics

{

publicAirLogistics(stringn)

:base(n)

{}

public overrideTransportCreate()

{

returnnewTransportAir();

}

}

interfaceTransport

{

}

classTransportTruck:Transport

{

publicTransportTruck()

{

Console.WriteLine("Машинаназначена");

}

}

classTransportAir:Transport

{

publicTransportAir()

{

Console.WriteLine("Самолетготов");

}

}

На рисунке 3.1 представлена диаграмма UML,

иллюстрирующаяшаблонвышеприведенногопримера:

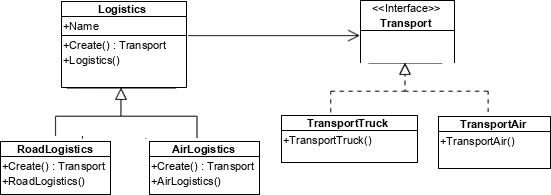


Рисунок3.1.ДиаграммаUMLреализациипаттерна«Ф[абричныйметод](https://refactoring.guru/ru/design-patterns/factory-method)»

# Вопросыкглаве:

1. Чем отличаются паттерны «Ф[абричный метод](https://refactoring.guru/ru/design-patterns/factory-method)» и

«[Абстрактнаяфабрика](https://refactoring.guru/ru/design-patterns/abstract-factory)»?

1. Какиепаттернырешаютзадачиэффективногоибезопасноговзаимодействиямеждуобъектамипрограммы?
2. Какие паттерны отвечают за построение удобных вподдержкеиерархийклассов?
3. Какиепаттерныотвечаютзаудобноеибезопасноесозданиеновыхобъектовилидажецелыхсемействобъектов?
4. Когдаимеетсмыслприменятьпаттерн«Компоновщик»?
5. Какойпаттернлучшеиспользовать,есливамнужнодобавлять обязанности объектам на лету, незаметно для кода,которыйих использует?
6. Какойпаттернлучшеиспользовать,есливыработаетесобъектом,поведениекоторогокардинальноменяетсявзависимостиотвнутреннегосостояния,причемтиповсостояниймного, иихкод часто меняется?
7. Ктоикогдапридумалпаттерны?
8. Какой паттерн проектирования гарантирует, что у класса естьтолько один экземпляр, и предоставляет к нему глобальнуюточкудоступа?
9. Так ли паттерны хороши на самом деле? Всегда ли можно ихиспользовать?
10. Можнолииспользоватьязыкпаттерноввнеразработкипрограммногообеспечения?
11. Почемуиногдапаттерныбываютвредными?
12. Какойпаттернлучшеиспользовать,есливамнужнопредставитьпростойилиурезанныйинтерфейсксложнойподсистеме?

# ГЛАВА 4.МОДУЛЬНОЕ ТЕСТИРОВАНИЕ ПРОГРАММНОГООБЕСПЕЧЕНИЯ

# 4.1.Омодульномтестировании

Модульноетестирование ([англ.](https://ru.wikipedia.org/wiki/Английский_язык)*unittesting*) – процессв[программировании](https://ru.wikipedia.org/wiki/Программирование),позволяющийпроверитьнакорректностьотдельныемодули[исходногокода](https://ru.wikipedia.org/wiki/Исходный_код)программы[5].

Тесты пишут для каждой нетривиальной функции или метода.Это позволяет быстро проверить, не привело ли изменение кода кпоявлениюошибоквужеоттестированныхместахпрограммы,атакжеоблегчаетобнаружениеиустранениетакихошибок.

Примодульномтестированииизолируютсяотдельныечастипрограммыипоказывается,чтопоотдельностиэтичастиработоспособны.

Модульноетестированиеполезноприпроведении[рефакторинг](https://ru.wikipedia.org/wiki/Рефакторинг)аигарантирует,чтомодульпо-прежнемуработаеткорректно ([регрессионное тестирование](https://ru.wikipedia.org/wiki/Регрессионное_тестирование)). Модульное тестированиеиспользуется для подхода к тестированию «снизу-вверх»: сначалатестируютсяотдельныечастипрограммы,азатемпрограммавцелом.

Модульноетестированиенеприменяют,когда:

* Решаетсякомбинаторнаязадача.Например,каждоевозможное значение булевской переменной потребует двух тестов:один ‒ на вариант TRUE, другой ‒ на вариант FALSE. В результатена каждую строку исходного кода потребуется 3-5 строк тестовогокода.
* Результатизвестен лишь приблизительно. Например,в[математическоммоделировании](https://ru.wikipedia.org/wiki/Математическая_модель)вомногихслучаяхкачествомоделированияопределяется«наглаз»,ипоследнийрезультатзаписываетсякак«опорный».Еслинайденорасхождение,новый

результатпроверяютвручнуюивыясняют,какойкачественнее:старыйилиновый.

Привыполненииюнит-тестовпроисходиттестированиекаждогоизмодулейпоотдельности.Этоозначает,чтоошибкиинтеграции,системногоуровня,функций,исполняемыхвнескольких модулях, не будут определены. Данная технология такжебесполезнадляпроведениятестовнапроизводительность.Модульноетестированиеболееэффективноприиспользованиивсочетаниисдругими методикамитестирования.

Для получения пользы от модульного тестирования требуетсястрогоследоватьтехнологиитестированиянавсемпротяжениипроцессаразработкипрограммногообеспечения(ПО).Нужнохранить не только записи обовсехпроведенныхтестах, нои обовсехизмененияхисходногокодавовсехмодулях.Сэтойцельюследуетиспользовать[системуконтроляверсийПО](https://ru.wikipedia.org/wiki/Система_управления_версиями).Еслиболеепоздняя версия ПО не проходит тест, который был успешно пройденранее,будетнесложнымсверитьвариантыисходногокодаиустранить ошибку. Также необходимо отслеживать и анализироватьнеудачныетесты.Игнорированиеэтоготребованияприведеткувеличениюнеудачных тестовых результатов.

[Экстремальноепрограммирование](https://ru.wikipedia.org/wiki/Экстремальное_программирование)предполагаеткакодинизпостулатовиспользованиеинструментовавтоматическогомодульноготестирования.Вэкстремальномпрограммированиииспользуются модульные тесты для[разработки через тестирование](https://ru.wikipedia.org/wiki/Разработка_через_тестирование).Для этого разработчик до написания кода пишет тест, отражающийтребования к модулю. Очевидно, тест до написания кода работать недолжен. Дальнейший процесс сводится к написанию кратчайшегокода, удовлетворяющего данному тесту. После разработчик пишетследующийтест, код,итакмногократно.

# 4.2СозданиемодульноготестасредствамиVisual Studio

Принаписаниимодульныхтестовследуетпридерживатьсяединого стиля написания тела теста. Отлично зарекомендовал себяподход AAA (arrange, act, assert). Напишем тест для метода, которыйпроверяет правильность вычисления суммы чисел. Для этого в*VisualStudio*создадим новый проект Visual C# и выбираем «Библиотекаклассов»(рисунок 4.1).

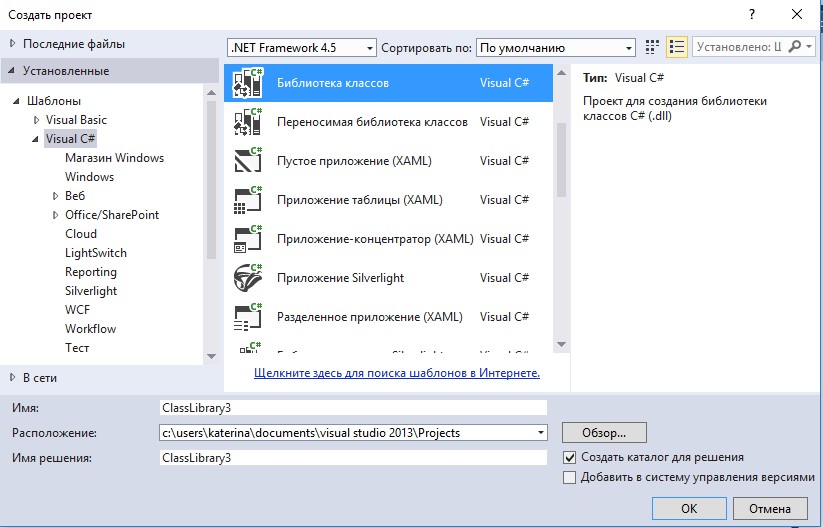


Рисунок4.1.Окно«Создатьпроект»

НазовемегоMyСalculation.«*Class1*»переименуемв «*MyCalc*».

namespaceMyCalc

{

publicclassMyCalc

{

publicintsum(intx,inty)

{

returnx+y;

}

}

}

Во времясозданиямодульноготестадлякаждогокласса,

подвергаемого тестированию, создается отдельный файл модульноготеста.Каждыйфайлмодульноготестасодержитметодтестадлякаждогометода,подвергаемоготестированию.Дляэтогонеобходимоврамкахтогожесамогорешениясоздатьещеодинпроектсоответствующеготипа.Правойкнопкойщелкнитепорешению,выберите«Добавить»изатем«Создатьпроект…»(рисунок4.2).

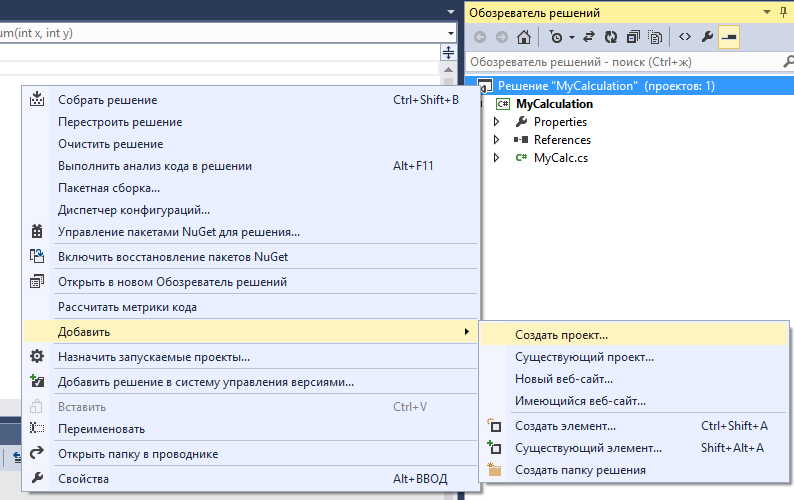


Рисунок4.2.«Добавитьпроектврешение»

В открывшемся окне в группе Visual C# щелкните «Тест», азатемвыберите«Проектмодульноготеста»рисунок4.3.Присозданиимодульныхтестовиспользуютправилаименованияпеременных,имяфайламодульноготестаполучаетсяпутем

соединения слова «Test» с именем файла, содержащего тестируемыйкод.Внашемпримереэто«MyCalcTest.cs».Введитеимяпроекта «MyCalcTests» инажмите«ОК».Такимобразомпроектбудетсоздан.

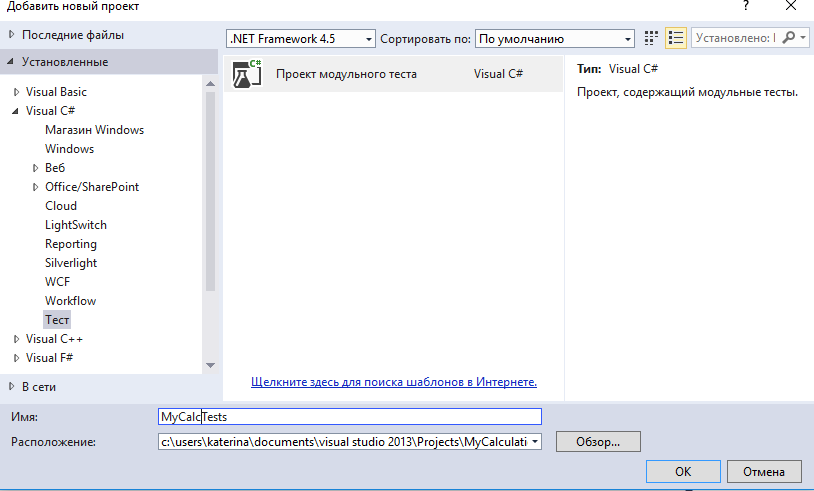


Рисунок4.3.Окно «Добавитьпроектмодульноготеста»

ПередВамипоявитсяследующийкод:

usingSystem;

usingMicrosoft.VisualStudio.TestTools.UnitTesting;

namespaceMyCalcTests

{

[TestClass]

publicclassUnitTest1

{

[TestMethod]

publicvoidTestMethod1()

{

}

}

}

Директива[TestMethod] обозначает, что далее идет метод,

содержащий модульный (unit) тест. А [TestClass] говорит о том, чтодалееидеткласс,содержащийметоды,вкоторыхприсутствуютмодульные-тесты.

Переименуемкласс«UnitTest1»в «MyCalcTests».ЗатемвReferencesпроектанеобходимодобавитьссылкунапроект,кодкоторого будем тестировать. Правой кнопкой щелкаем на Referencesи выбираем «Добавить ссылку…». В появившемся окне раскрываемгруппу «Решение», выбираем «Проекты» и ставим галочку напротивпроектаMyСalculation.

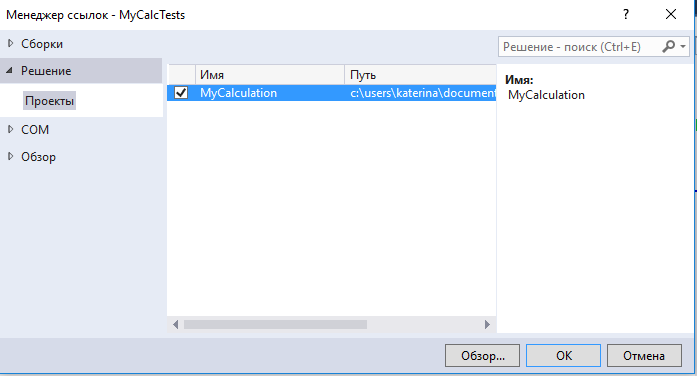


Рисунок4.4.Окно«Менеджерссылок»

Также в коде необходимо подключить с помощью директивыusingследующеепространство имен:usingMyСalculation.

Теперьнапишемтест.Проверим,правильнолипрограммавычисляетсуммучисел10и20.Ожидаемыйрезультат(правильное

решение)вданномслучаеэто число 30. Переименуемметод*TestMethod1()вsum\_10and20\_30returned()*.Новоеназваниеметода поясняет, что будет проверяться и для каких значений (10 и20)ичто ожидаетсявкачествеправильногорезультата(30).

Тестирующий метод обычно содержит три необходимыхкомпонента:

* исходныеданные:входныезначенияиожидаемыйрезультат(arrange);
* код,вычисляющийзначениеспомощьютестируемогометода

(act);

* код,сравнивающий ожидаемый результат с полученным

(assert).

Тестирующийкод будеттаким:

namespaceMyCalcTests

{

[TestClass]

publicclassMyCalcTests

{

[TestMethod]

publicvoidsum\_10and20\_30returned()

{

//arrangeintx = 10;inty=20;

intexpected=30;

//ast

MyCalcc =newMyCalc();intactual=c.sum(x,y);

//assert

Assert.AreEqual(expected,actual);

}

}

}

Длясравненияожидаемогорезультатасполученнымиспользуется метод*AreEqual()*класса «*Assert»*. Данный класс всегдаиспользуетсяпринаписаниимодульныхтестовв*Visual Studio*.

Чтобыпросмотретьвсетесты,доступныедлявыполнения,необходимо открыть окно «Обозреватель тестов». Для этого в меню*Visual Studio*щелкните на кнопку «ТЕСТ», выберите «Окна», а затемнажмитенапункт«Обозревательтестов»(рисунок 4.5).

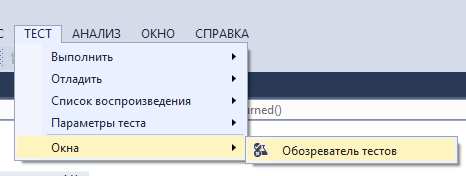


Рисунок 4.5.Добавлениеокна«Обозревательтестов»

Встудиипоявитсяследующееокно(рисунок4.6):

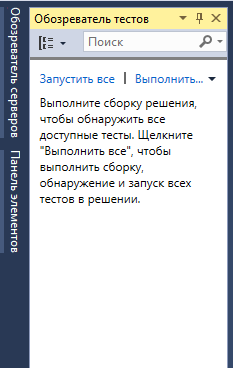


Рисунок4.6.Окно«Обозревательтестов»

В данный момент список тестов пуст, поскольку решение ещени разу не было собрано. Выполним сборку нажатием клавиш Ctrl +Shift + B. После ее завершения в «Обозревателе тестов» увидим наштест(рисунок 4.7).

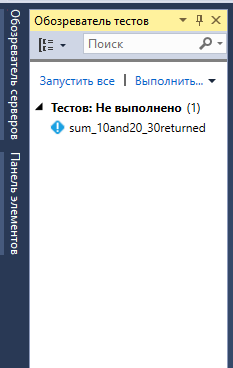


Рисунок4.7.Окно «Обозревательтестов»послезавершениясборки

Синяятабличкасвосклицательнымзнакомозначает,чтоуказанный тест никогда не выполнялся. Выполним его. Для этогонажмем правой кнопкой мыши на его имени и выберем «Выполнитьвыбранныетесты».

Зеленыйкружоксгалочкойозначает,чтомодульныйтестуспешнопройден:ожидаемыйиполученныйрезультатыравны(рисунок4.8).

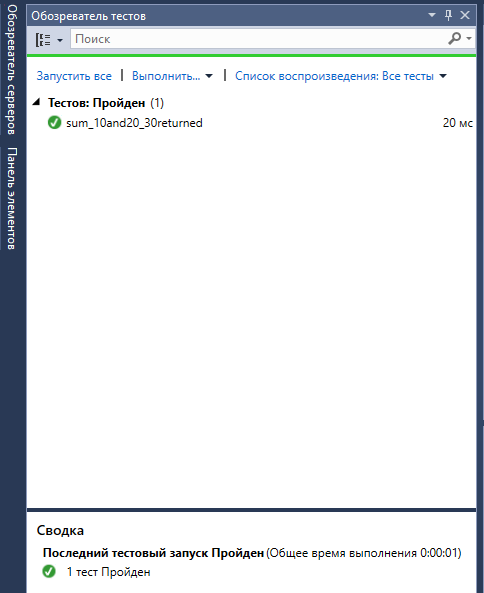


Рисунок 4.8. Окно «Обозреватель тестов» после успешного выполнениявыбранныхтестов

Изменим код метода*sum\_10and20\_30returned()*, вычисляющегосуммучисел,чтобысымитировать,чтотестнесработал,ипосмотреть,какповедетсебя*VisualStudio*.Прибавимквозвращаемому значению10. Запустимтест(рисунок4.9).

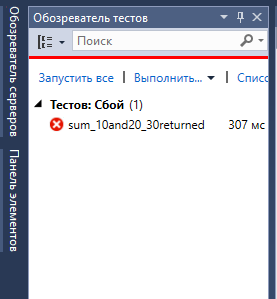


Рисунок 4.9. Окно «Обозреватель тестов»,послетогокактестнесработал

Такимобразом,мынаучилисьсоздаватьпростейшиймодульныйтест наязыке C# в*VisualStudio*.

# Пространство именMicrosoft.VisualStudio.TestTools.UnitTesting

Пространство имен Microsoft.VisualStudio.TestTools.UnitTestingсодержитклассыдляработымодульныхтестов.Рассмотримэтиклассыподробнее[6].

Вклассе «*Assert*» определеннаборстатическихметодов,которыеможноиспользоватьвтестах.Класс«*Assert*»являетсяоднимизсамыхчастоприменяемых.Описаниеметодовпредставлено втаблице4.1.

Каждыйстатическийметодвклассе«*Assert*»позволяетпроверитькакой-тоаспектмодульноготеста,иеслипроверканепроходит, эти методы генерируют исключение. Чтобы модульныйтестпрошел,всеутверждениядолжнызавершитьсяуспешно.

Каждый метод в классе «*Assert*» имеет перегруженную версию,котораяпринимаетпараметр*string*.Вслучаеотрицательногорезультата утверждения эта строка помещается в элемент сообщениявнутриобъектаисключения.

Методы*AreEqual()иAreNotEqual()имеют*несколькоперегруженныхверсий,предназначенныхдлясравненияспецифическихтипов.Например,существуетверсия,котораяпозволяетсравниватьстрокибезучетарегистрасимволов.

Таблица4.1

Описаниеметодовкласса«*Assert*»

|  |  |
| --- | --- |
| Метод | Описание |
| AreEqual<T>(T,T);AreEqual<T>(T,T,string) | Утверждает,чтодваобъектатипаTимеютодноитожезначение |
| AreNotEqual<T>(T,T);  AreNotEqual<T>(T,T,string) | Утверждает,что дваобъектатипаTнеимеют одно  итожезначение |
| AreSame<T>(T,T);  AreSame<T>(T,T,string) | Утверждает,чтодвепеременныессылаютсяна  одини тот жеобъект |
| AreNotSame<T> (T, T);AreNotSame<T>(T,T,string) | Утверждает,чтодвепеременныессылаютсянаразныеобъекты |
| Fail();  Fail(string) | Отрицательныйрезультатутверждения‒никакие  условиянепроверены |
| Inconclusive();Inconclusive(string) | Показывает,чторезультатмодульноготестанеможетбыть однозначноустановлен |
| IsTrue(bool);IsTrue(bool,string) | Утверждает,чтобулевскоезначениеравноtrue‒  чащевсегоиспользуетсядляоценкивыражения,возвращающегобулевскийрезультат |
| IsFalse(bool);IsFalse(bool,string) | Утверждает,чтобулевскоезначениеравноfalse |
| IsNull(object);  IsNull(object,string) | Утверждает,чтопеременнаянеприсвоена  объектнойссылке |
| IsNotNull(object);  IsNotNull(object,string) | Утверждает,чтопеременнаяприсвоенаобъектной  ссылке |
| IsInstanceOfType(object,Type);  IsInstanceOfType(object,Type,string) | Утверждает,чтообъектотноситсякуказанному  типуилиявляетсяпроизводнымот указанноготипа |
| IsNotInstanceOfType(object,Type);  IsNotInstanceOfType(object,Type,string) | Утверждает,чтообъектнеотноситсякуказанномутипу |

Напишемтестдляметода,которыйвычисляетквадратныйкореньдлязаданногочисла.

namespaceClassAss

{

publicclassMyClass

{

publicstaticdoubleGetSgrt(doublevalue)

{

returnMath.Sqrt(value);

}

}

}

Притестированииданногометодаследуетучестьдваварианта

решений,когдапривычисленииквадратногокорнямыполучаемцелое значение ‒ метод*IsSqrtTest*(), и когда проверка значений наравенствовыполняетсясучетомпогрешности,вслучаееслирешениесодержитдробнуючасть,метод*DeltaTest()*.

namespaceClassAssTest

{

[TestClass]

publicclassMyClassTest

{

[TestMethod]publicvoidIsSqrtTest()

{

//arrange

const doubleinput = 4;constdoubleexpected=2;

//act

doubleactual=ClassAss.MyClass.GetSgrt(input);

//assert

Assert.AreEqual(expected, actual,"Sqrt of {0} should have been {1}!", input,expected);

}

[TestMethod]publicvoidDeltaTest()

{

//arrange

const doubleexpected = 3.1;constdoubledelta=0.07;

//act

doubleactual=ClassAss.MyClass.GetSgrt(10);

//assert

//проверка значений на равенство с учетом погрешностиAssert.AreEqual(expected,actual,delta,"fail mtssege!");

}

}

}

Класс«*TestContext*»используетсядляхраненияинформации,

передаваемойдлямодульныхтестов.Рассмотримработуданногоклассанапримере.

namespaceContexUnitTest

{

[TestClass]

publicclassUnitTest1

{

publicTestContextTestContext{get;set;}//свойство

[TestMethod]

publicvoidTestMethod1()

{

TestContext.WriteLine("TestContext.TestRunDirectory{0}",TestContext.TestRunDirectory);

TestContext.WriteLine("TestName{0}", TestContext.TestName);TestContext.WriteLine("CurrentTestOutcomey{0}",TestContext.CurrentTestOutcome);//результатработытеста

}

[TestCleanup]

publicvoidTestCleanup()

{

TestContext.WriteLine("TestName(Cleanup){0}",TestContext.TestName);

TestContext.WriteLine("CurrentTestOutcomey (Cleanup) {0}",TestContext.CurrentTestOutcome);

}

}

}

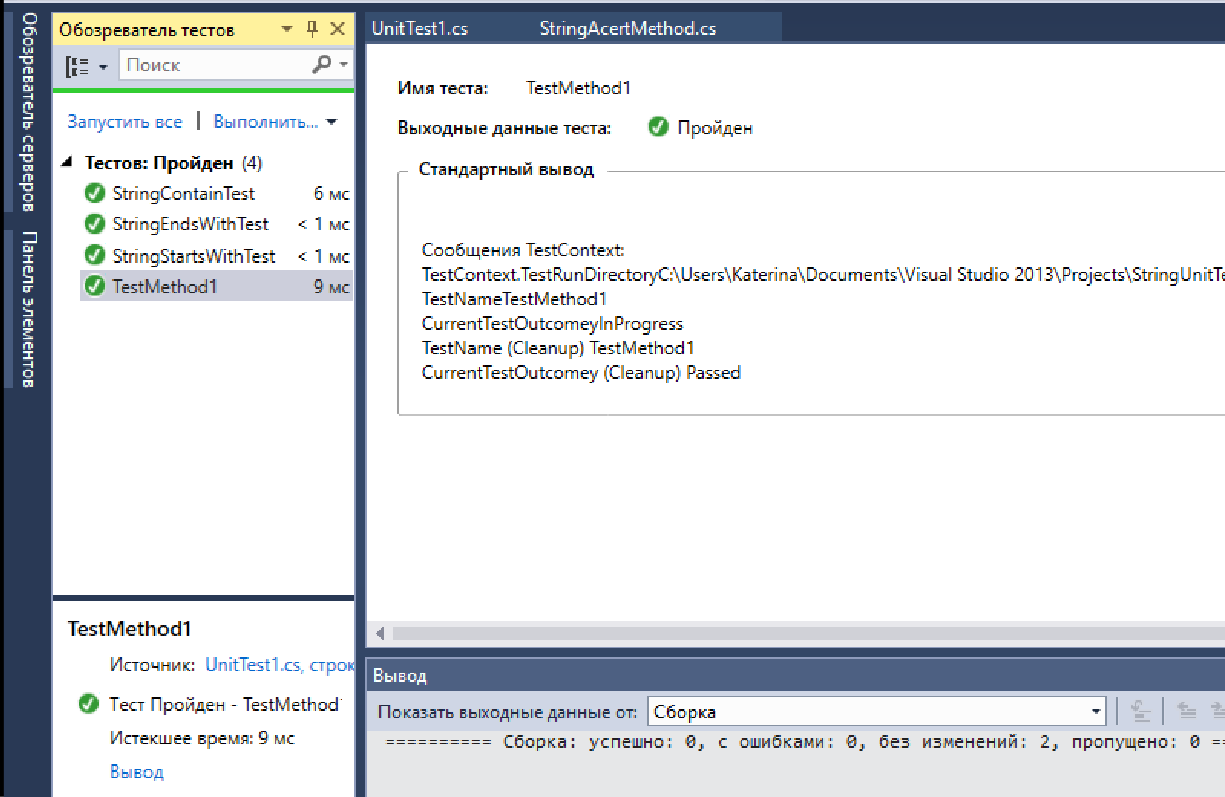
Результатработытестаможноувидетьнарисунке4.9.

Рисунок4.9.Результатиспользованиякласса«*TestContext*»

Дополнительныеатрибутытеста:

[ClassInitialize()] используетсядлявыполнениякодадовыполненияпервоготеставклассе.

[ClassCleanUp()] используется для выполнения кода послезавершениявыполнениявсехтестов вклассе.

[TestInitialize()] используетсядлявыполнениякодадовыполнениякаждоготеста.

[TestCleanUp()] используется для выполнения кода послезавершениявыполнениякаждоготеста.

# Вопросы к главе:

* + 1. ЧтотакоетестированиеПО?
    2. Чтопроисходитнаэтапеисполнениятестирования?
    3. Чтопроисходитнаэтапепланированиятестирования?
    4. Что происходит на этапе изучения и анализа предметатестирования?
    5. Какиесуществуюттипытестовпопокрытию?
    6. Какиесуществуюттестовыеактивности?
    7. Какиеэтапысоставляютпроцесстестирования?
    8. Чтовыпонимаетеподразработкойчерезтестирование?
    9. Можно ли использовать модульное тестирование дляпроведениятестовнапроизводительность?Почему?
    10. Какиекомпонентыдолженсодержатьтестирующийметод?
    11. Длячегонужен класс«*TestContext»*? Какего использоватьпритестировании?
    12. Длячегонуженкласс«*Assert»*?Какегоиспользоватьпритестировании?