Практическая работа №19

Тема: Методы оптимизации и рефакторинга

Цель работы: изучение приемов рефакторинга программного кода в Visual Studio

Задачи:

– изучение и выполнение приемов рефакторинга программного кода;
– применение методов рефакторинга C# в Visual Studio.

Материально-техническое обеспечение:

Место проведения: Компьютерный класс.

Время на выполнение работы: 4 часа.

Оборудование: ПК

Средства обучения: операционная система, текстовый процессор MS Word, программные средства определенного вида

Исходные данные:

1. Конспект занятия.
2. Задание для практической работы.

Перечень справочной литературы:

1) Программирование на языке высокого уровня. Программирование на языке C++: учеб. пособие / Т. И. Немцова, С. Ю. Голова, А. И. Терентьев ; под ред. Л. Г. Гагариной. – М. : ИД «ФОРУМ» : ИНФРА-М, 2018. – 512 с. – (Среднее профессиональное образование).

Краткие теоретические сведения:

Рефакторинг кода

При создании приложений многие разработчики сначала заботятся об их функциональности, а когда она обеспечена, переделывают приложения таким образом, чтобы они были более управляемыми и удобочитаемыми. Это называется ***рефакторингом (refactoring)***. Под **рефакторингом** понимается *процесс переделки кода для повышения удобочитаемости и производительности приложений, а также обеспечения безопасности типов и приведения кода к такому виду, в котором он лучше соответствует рекомендуемым приемам объектно-ориентированного программирования*. В Visual Studio можно достаточно хорошо автоматизировать этот процесс.

За счет использования меню ***Правка - Выполнить рефакторинг,*** которое становится доступным при открытом файле кода, а также соответствующих клавиатурных комбинаций быстрого вызова, смарт-тегов (smart tags) и/или вызывающих контекстные меню щелчков, можно существенно видоизменять код с минимальным объемом усилий. В следующей таблице перечислены некоторые наиболее распространенные приемы рефакторинга, которые распознаются в Visual Studio:

Прием рефакторинга	Описание
Extract Method (Извлечение метода)	Позволяет определять новый метод на основе выбираемых операторов программного кода
Encapsulate Field (Инкапсуляция поля)	Позволяет превращать общедоступное поле в приватное, инкапсулированное в форму свойство C#
Extract Interface (Извлечение интерфейса)	Позволяет определять новый тип интерфейса на основе набора существующих членов типа
Reorder Parameters (Переупорядочивание параметров)	Позволяет изменять порядок следования аргументов в члене
Remove Parameters (Удаление параметров)	Позволяет удалять определенный аргумент из текущего списка параметров
Rename (Переименование)	Позволяет переименовывать используемый в коде метод, поле, локальную переменную и т.д. по всему проекту

Извлечение метода рефакторинга (C#)

Извлечение метода — это операция рефакторинга, которая обеспечивает простой способ создания нового метода из фрагмента кода в существующем члене.

Используя **метод извлечения** , вы можете создать новый метод, извлекая фрагмент кода из блока кода существующего члена. Новый извлеченный метод содержит выбранный код, а выбранный код в существующем члене заменяется вызовом нового метода. Превращение фрагмента кода в отдельный метод позволяет быстро и точно реорганизовать код для лучшего повторного использования и удобочитаемости.

Метод извлечения имеет следующие преимущества:

- Поощряет лучшие практики кодирования, делая упор на дискретные, многократно используемые методы.
- Поощряет самодокументирующийся код благодаря хорошей организации.

Когда используются описательные имена, высокоуровневые методы могут больше напоминать серию комментариев.

- Поощряет создание более мелких методов для упрощения переопределения.
- Уменьшает дублирование кода.

Рефакторинг переименования (C#)

Переименование — это функция рефакторинга в интегрированной среде разработки (IDE) Visual Studio, которая обеспечивает простой способ переименования идентификаторов символов кода, таких как поля, локальные переменные, методы, пространства имен, свойства и типы. **Rename** можно использовать для изменения имен в комментариях и в строках, а также для изменения объявлений и вызовов идентификатора.

Примечание

При использовании системы управления версиями для Visual Studio получите последнюю версию исходных кодов, прежде чем пытаться выполнить рефакторинг с переименованием.

Рефакторинг переименования доступен в следующих функциях Visual Studio:

Характерная черта	**Поведение при рефакторинге в IDE**
Редактор кода	В редакторе кода рефакторинг переименования доступен при наведении курсора на определенные типы символов кода. Когда курсор находится в этом положении, вы можете вызвать команду «**Переименовать**», набрав сочетание клавиш (CTRL + R, CTRL + R) или выбрав команду «**Переименовать**» из смарт-тега, контекстного меню или меню **рефакторинга** .
Представление класса	Когда вы выбираете идентификатор в представлении классов, рефакторинг переименования доступен из контекстного меню и меню **рефакторинга** .
Браузер объектов	Когда вы выбираете идентификатор в обозревателе объектов, рефакторинг переименования доступен только в меню « **Рефакторинг» .**
Сетка свойств конструктора Windows Forms	В **сетке свойств** конструктора Windows Forms изменение имени элемента управления инициирует операцию переименования для этого элемента управления. Диалоговое окно « **Переименовать**» не появится.
Обозреватель решений	В **обозревателе решений** в контекстном меню доступна команда «**Переименовать».** Если выбранный исходный файл содержит класс, имя класса которого совпадает с именем файла, вы можете использовать эту команду для одновременного переименования исходного файла и выполнения рефакторинга переименования. Например, если вы создаете приложение для Windows по умолчанию, а затем переименовываете Form1.cs в TestForm.cs, имя исходного файла Form1.cs изменится на TestForm.cs, а класс Form1 и все ссылки на этот класс будут переименованы в ТестФорма. **Примечание.** Команда « **Отменить** » (CTRL+Z) только отменяет рефакторинг переименования в коде и не возвращает исходное имя файла. Если выбранный исходный файл не содержит класса, имя которого совпадает с именем файла, команда « **Переименовать** » в **обозревателе решений** только переименует исходный файл и не выполнит рефакторинг переименования.

Переименовать операции

Когда вы выполняете **Rename**, механизм рефакторинга выполняет операцию переименования, характерную для каждого символа кода, как описано в следующей таблице.

Код Символ	Переименовать операцию
Поле	Изменяет объявление и использование поля на новое имя.
Локальная переменная	Изменяет объявление и использование переменной на новое имя.
Метод	Изменяет имя метода и все ссылки на этот метод на новое имя. **Примечание.** Когда вы переименовываете метод расширения, операция переименования распространяется на все экземпляры метода, находящиеся в области действия, независимо от того, используется ли метод расширения как статический метод или как метод экземпляра.
Пространство имен	Изменяет имя пространства имен на новое имя в объявлении, во всех usingинструкциях и полные имена. **Примечание.** При переименовании пространства имен Visual Studio также обновляет свойство «**Пространство имен по умолчанию**» на странице «**Приложение » конструктора проектов** . Это свойство нельзя сбросить, выбрав **Undo** в меню **Edit.** Чтобы сбросить значение свойства **Пространство имен по умолчанию**, необходимо изменить это свойство в **конструкторе проектов**.
Имущество	Изменяет объявление и использование свойства на новое имя.
Тип	Изменяет все объявления и все использования типа на новое имя, включая конструкторы и деструкторы. Для частичных типов операция переименования распространяется на все части.

Инкапсуляция рефакторинга полей (C#)

Операция рефакторинга **Encapsulate Field** позволяет быстро создать свойство из существующего поля, а затем легко обновить код, добавив ссылки на новое свойство.

Когда поле является общедоступным, другие объекты имеют прямой доступ к этому полю и могут изменять его, не обнаруживая этого объекта, которому принадлежит это поле. Используя свойства для инкапсуляции этого поля, вы можете запретить прямой доступ к полям.

Чтобы создать новое свойство, операция **Encapsulate Field** изменяет модификатор доступа для поля, которое вы хотите инкапсулировать, на **private**, а затем создает методы доступа **get** и **set** для этого поля. В некоторых случаях **get** генерируется только метод доступа, например, когда поле объявлено доступным только для чтения.

Механизм рефакторинга обновляет ваш код ссылками на новое свойство в областях, указанных в разделе «**Обновить ссылки**» диалогового окна «**Инкапсулировать поле».**

Извлечение рефакторинга интерфейса (C#)

Извлечение интерфейса — это операция рефакторинга, которая обеспечивает простой способ создания нового интерфейса с элементами, происходящими из существующего класса, структуры или интерфейса.

Когда несколько клиентов используют одно и то же подмножество членов из класса, структуры или интерфейса или, когда несколько классов, структур или интерфейсов имеют общее подмножество членов, может быть полезно реализовать подмножество членов в интерфейсе.

Извлечь интерфейс создает интерфейс в новом файле и устанавливает курсор в начало нового файла. Вы можете указать, какие члены следует извлечь в новый интерфейс, имя нового интерфейса и имя сгенерированного файла с помощью диалогового окна **Извлечь интерфейс .**

Рефакторинг удаления параметров (C#)

Remove Parameters— это операция рефакторинга, предоставляющая простой способ удаления параметров из методов, индексаторов или делегатов. Remove Parameters изменяет объявление; в любом месте, где вызывается член, параметр удаляется, чтобы отразить новое объявление.

Вы выполняете операцию удаления параметров, сначала устанавливая курсор на метод, индексатор или делегат. Пока курсор находится в нужном положении, чтобы вызвать Parameters операцию удаления, щелкните меню **Правка – Выполнить рефакторинг**, нажмите сочетание клавиш или выберите команду из контекстного меню.

Примечание

Вы не можете удалить первый параметр в методе расширения.

Рефакторинг параметров изменения порядка (C#)

Reorder Parameters— это операция рефакторинга Visual C#, которая предоставляет простой способ изменить порядок параметров для методов, индексаторов и делегатов. Reorder Parameters изменяет объявление, и во всех местах, где вызывается член, параметры перестраиваются, чтобы отразить новый порядок.
Чтобы выполнить Reorder Parameters операцию, поместите курсор на метод, индексатор или делегат или рядом с ним. Когда курсор находится в нужном положении, вызовите Reorder Parameters операцию, нажав сочетание клавиш или щелкнув команду из контекстного меню.
Примечание
Вы не можете изменить порядок первого параметра в методе расширения.

Ход работы:

Требования к содержанию отчета:

- Номер и название практической работы.
- Цель работы.
- По каждому заданию (задаче/примеру) экранные формы (при наличии) и листинг исходного программного кода и кода с изменением в процессе рефакторинга, показывающие порядок выполнения практической работы, и результаты, полученные в ходе её выполнения.
- Ответы на контрольные вопросы в тетради.

Порядок выполнения работы:

Все проекты практической работы и файл отчета под именем **Пр19_Фамилия.docx** размещать в своей сетевой в новой папке **Пр19_Фамилия.**

Внимание!!! В файл отчета помещать скрин окон, содержащий полный программный код с используемыми директивами и пространством имен.

В начале каждого файла проекта установить комментарии: пр.р.№_____ (указать номер), свою Фамилию. Формулировку задания

Задание 1. Использование метода извлечения

1. Создайте консольное приложение с именем **pr19_1_Фамилия**, а затем замените Program приведенным ниже примером кода.

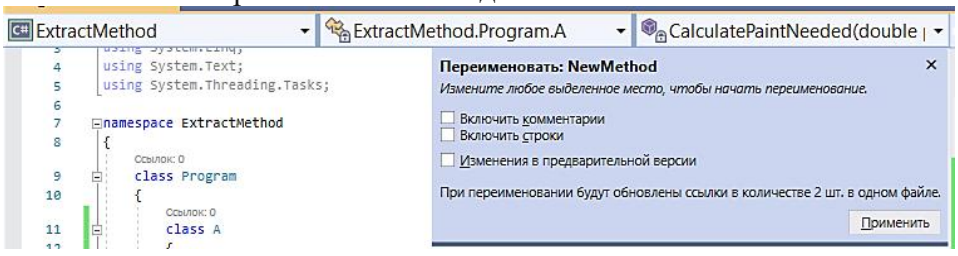
```
class Program
{
    Ссылок: 0
    class A
    {
        const double PI = 3.141592;

        Ссылок: 0
        double CalculatePaintNeeded(double paintPerUnit, double radius)
        {
            //Выберите любое из следующих действий:
            // 1. Вся следующая строка кода.
            // 2. Правая часть следующей строки кода.
            // 3. Просто «PI *» правой части следующей строки
            //кода (для просмотра запроса на расширение выбора).
            // 4. Весь код в теле метода.
            //... Затем вызовите метод извлечения.

            double area = PI * radius * radius;

            return area / paintPerUnit;
        }
    }
}
```

2. Отобразите в отчете исходный код и затем программный код с изменениями.
3. Выберите фрагмент кода, который вы хотите извлечь:

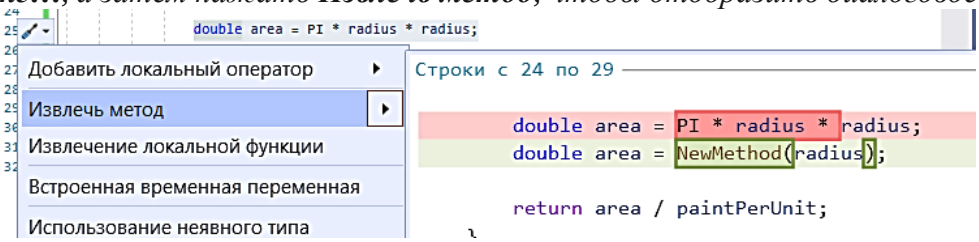
double area = PI * radius * radius;

4. В меню **Правка** выберите **Выполнить рефакторинг…** и далее **Извлечь метод**…

Появится диалоговое окно «Переименовать метод»**.**



*Вы также можете щелкнуть правой кнопкой мыши выбранный код, указать **Быстрые действия и рефакторинг…**, а затем нажать **Извлечь метод**, чтобы отобразить диалоговое окно.*



*Кроме того, вы также можете нажать сочетание клавиш (английская раскладка) **CTRL+R** и затем сразу **CTRL+M***

5. Укажите имя нового метода, например, **CircleArea**.

Предварительный просмотр новой сигнатуры метода отображается в разделе **Предварительный просмотр сигнатуры метода**.

6. Нажмите **ОК** .
7. Отобразите в отчете изменения, полученные в программном коде.

Примечания

*При использовании команды «**Извлечь метод**» новый метод вставляется после исходного элемента в том же классе.*

Частичные типы

*Если класс является частичным типом, **метод извлечения** создает новый метод сразу после исходного члена. **Метод извлечения** определяет сигнатуру нового метода, создавая статический метод, когда код в новом методе не ссылается на данные экземпляра.*

Общие параметры типа

Когда вы извлекаете метод с неограниченным параметром универсального типа, сгенерированный код не добавит refмодификатор к этому параметру, пока ему не будет присвоено значение. Если извлеченный метод будет поддерживать ссылочные типы в качестве аргумента универсального типа, вам следует вручную добавить refмодификатор к параметру в сигнатуре метода.

Задание 1.2. Использование метода извлечения

1. Создайте консольное приложение с именем **pr19_1.2_Фамилия**, а затем замените Program приведенным ниже примером кода.

```
Ссылок: 0
class Program
{
    Ссылок: 0
    static void Main(string[] args)
    {
        // Настраиваем консольный интерфейс (CUI)
        Console.Title = "Мое приложение";
        Console.ForegroundColor = ConsoleColor.Yellow;
        Console.BackgroundColor = ConsoleColor.Blue;
        Console.WriteLine("Привет, это мой проект!");
        Console.BackgroundColor = ConsoleColor.Black;
        Console.ReadLine();

    }
}
```

В таком, как он есть виде, в этом коде нет ничего неправильного, но давайте представим, что возникло желание сделать так, чтобы данное приветственное сообщение отображалось в различных местах по всей программе. В идеале вместо того, чтобы заново вводить ту же самую отвечающую за настройку консольного интерфейса логику, было бы неплохо иметь вспомогательную функцию, которую можно было бы вызывать для решения этой задачи. С учетом этого, попробуйте применить к существующему коду прием *рефакторинга Extract Method (Извлечение метода)*.

2. Для этого выделите в окне редактора все содержащиеся внутри Main() операторы, кроме последнего вызова *Console.ReadLine()*.
3. Выберите в меню пункт **Правка – Выполнить рефакторинг… - Извлечь метод**.
4. Назначьте новому методу имя **MyConfigCUI()**

5. Отобразите в отчете изменения, полученные в программном коде.

После этого метод Main() станет вызывать новый только что сгенерированный метод MyConfigCUI(), внутри которого будет содержаться выделенный ранее код:

Нетрудно заметить, подобные мастера позволяют довольно легко производить рефакторинг кода не только на одной странице, но и во всем приложении.

Задание 1.3. Анонимные методы

Если вы попытаетесь извлечь часть анонимного метода, включающую ссылку на локальную переменную, объявленную или указанную за пределами анонимного метода, Visual Studio предупредит вас о возможных семантических изменениях.

Когда анонимный метод использует значение локальной переменной, значение получается в момент выполнения анонимного метода. Когда анонимный метод извлекается в другой метод, значение локальной переменной получается в момент вызова извлеченного метода.

Следующий пример иллюстрирует это семантическое изменение. Если этот код будет выполнен, то в консоль будет выведено **11 .**

1) Создайте консольное приложение **pr19_1.3_Фамилия.** Напишите следующий программный код. Проверьте результат, отобразите в отчете исходный код и результат

```
class Program
{
    delegate void D();
    D d;
    Ссылок: 0
    static void Main(string[] args)
    {
        Program p = new Program();
        int i = 10;
        /*начать извлечение*/
        p.d = delegate { Console.WriteLine(i++); };
        /*закончить извлечение*/
        i++;
        p.d();
    }
}
```

2) Примените метод извлечения для извлечения области кода, помеченной комментариями к коду, в отдельный метод, а затем выполняете рефакторинговый код.

Отобразите в отчете изменения, полученные в программном коде и результат его выполнения. Что изменилось? Объясните результаты работы программных кодов

Задание 2. Рефакторинг переименования

Чтобы переименовать идентификатор

1. Создайте консольное приложение с именем **pr19_2_Фамилия**, а затем замените Program приведенным ниже примером кода.

```
9   class Program
10  {
        Ссылок: 2
11      class ProtoClassA
12      {
13          // Вызвать метод 'MethodB'.
            ссылка: 1
14          public void MethodB(int i, bool b) { }
15      }
16
        Ссылок: 0
17      class ProtoClassC
18      {
            Ссылок: 0
19          void D()
20          {
21              ProtoClassA MyClassA = new ProtoClassA();
22
23              // Вызвать метод 'MethodB'.
24              MyClassA.MethodB(0, false);
25          }
26      }
27  }
```

2. Отобразите в отчете исходный код и затем программный код с изменениями.

3. Поместите курсор на **MethodB**, либо в объявлении метода, либо в вызове метода.
4. В меню **Правка – Выполнить рефакторинг…** выберите **Переименовать…**. Появится диалоговое окно «**Переименовать».**
5. Введите новое имя **MethodC**.
6. В диалоговом окне «**Переименовать»** установите флажок «**Включить комментарии**».
7. Нажмите **Применить** .
8. Отобразите в отчете программный код с изменениями.

Задание 3. Инкапсуляция рефакторинга полей (C#)
Чтобы создать свойство из поля

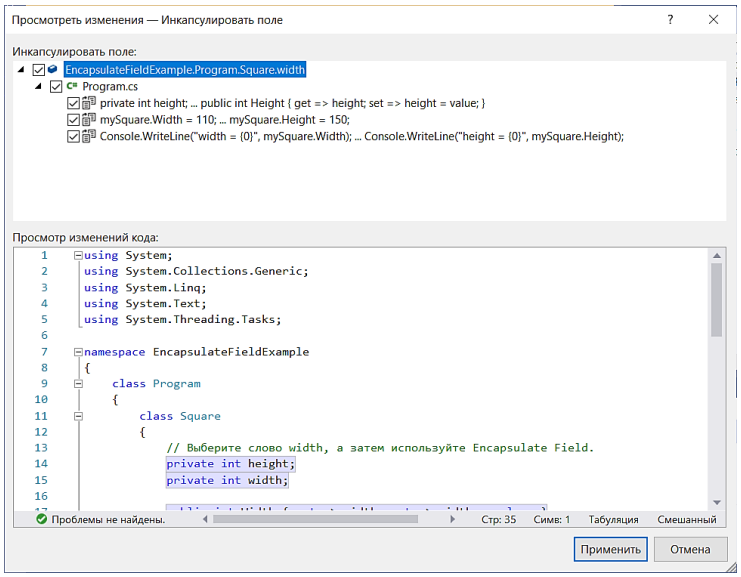
1. Создайте консольное приложение с именем **pr19_3_Фамилия**, а затем замените Program приведенным ниже примером кода.

```
class Program
{
    Ссылок: 2
    class Square
    {
        // Выберите слово width, а затем используйте Encapsulate Field.
        public int width, height;
    }
    Ссылок: 0
    class MainClass
    {
        Ссылок: 0
        public static void Main()
        {
            Square mySquare = new Square();
            mySquare.width = 110;
            mySquare.height = 150;
            // Выходные значения ширины и высоты.
            Console.WriteLine("width = {0}", mySquare.width);
            Console.WriteLine("height = {0}", mySquare.height);
        }
    }

}
```

2. Отобразите в отчете исходный код и затем программный код с изменениями.
3. В редакторе кода поместите курсор в объявление на имя поля, которое вы хотите инкапсулировать. В приведенном ниже примере поместите курсор на слово **width**:

public int width, height;

4. В меню **Правка – Выполнить рефакторинг…** выберите **Инкапсулировать поле…** Появится диалоговое окно « **Инкапсулировать поле ».**



Вы также можете нажать сочетание клавиш CTRL+R, CTRL+E, чтобы отобразить диалоговое окно ***Encapsulate Field .***

5. Укажите настройки. Нажмите кнопку **Применить**.
6. Отобразите в отчете программный код с изменениями.

Примечания

Операция ***Encapsulate Field*** *возможна только в том случае, если курсор находится на той же строке, что и объявление поля.*

Для объявлений, которые объявляют несколько полей, ***Encapsulate Field*** *использует запятую в качестве границы между полями и инициирует рефакторинг поля, ближайшего к курсору и на той же строке, что и курсор. Вы также можете указать, какое поле вы хотите инкапсулировать, выбрав имя этого поля в объявлении.*

Код, сгенерированный этой операцией рефакторинга, моделируется функцией инкапсуляции фрагментов кода поля. Фрагменты кода можно изменить.

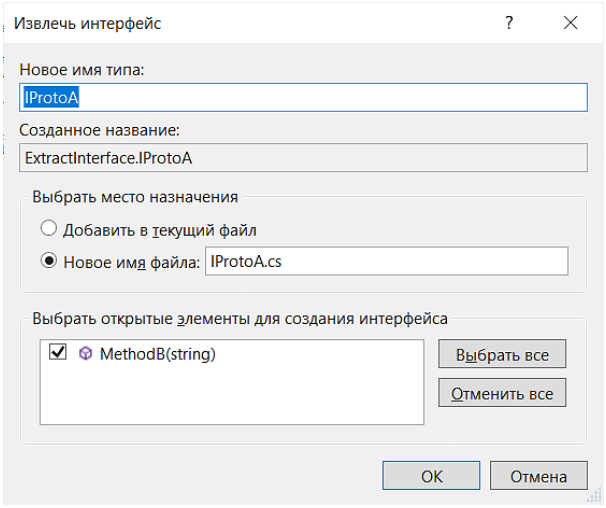
Задание 4. Извлечение рефакторинга интерфейса (C#)
Использование интерфейса извлечения

1. Создайте консольное приложение с именем **pr19_4_Фамилия**, а затем замените Program следующим кодом.

```
class Program
{
    //Вызвать интерфейс извлечения на ProtoA.
    //Примечание: извлеченный интерфейс будет создан в новом файле.
    Ссылок: 0
    class ProtoA
    {
        Ссылок: 0
        public void MethodB(string s) { }
    }
}
```

2. Отобразите в отчете исходный код и затем программный код с изменениями.
3. Поместив курсор в **MethodB**, нажмите **Извлечь интерфейс…** в меню **Правка – Выполнить рефакторинг…**

Появится диалоговое окно **Извлечь интерфейс .**



*Вы также можете нажать сочетание клавиш CTRL+R, CTRL +I, чтобы отобразить диалоговое окно «**Извлечение интерфейса».***

4. Щелкните **Выбрать все**.
5. Нажмите **ОК** .
6. Отобразите в отчете программный код с изменениями и структуру решения. Что изменилось? Отобразите все изменения в файле отчета.

Примечания

Эта функция доступна только в том случае, если курсор находится в классе, структуре или интерфейсе, содержащем члены, которые вы хотите извлечь. Когда курсор окажется в этом положении, вызовите операцию рефакторинга Extract Interface.

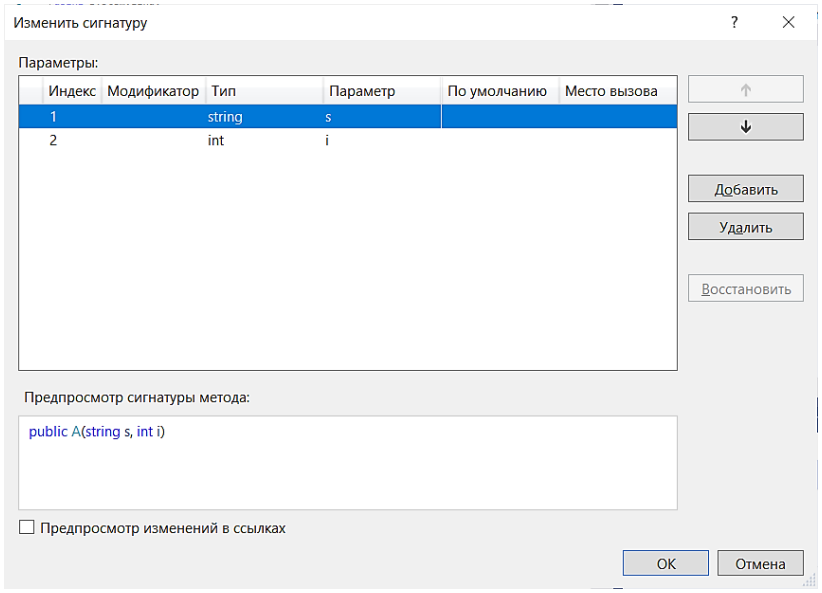
Когда вы вызываете интерфейс извлечения для класса или структуры, список баз и интерфейсов изменяется, чтобы включить новое имя интерфейса. Когда вы вызываете извлечение интерфейса для интерфейса, список баз и интерфейсов не изменяется.

Задание 5. Рефакторинг удаления параметров (C#)
Удаление параметров

1. Создайте консольное приложение с именем **pr19_5_Фамилия**, а затем замените Program следующим кодом.

```
9   class Program
10  {
        Ссылок: 3
11      class A
12      {
13          // Вызвать для 'A'.
            ссылка: 1
14          public A(string s, int i) { }
15      }
16
        Ссылок: 0
17      class B
18      {
            Ссылок: 0
19          void C()
20          {
21              // Вызвать для 'A'.
22              A a = new A("a", 2);
23          }
24      }
25  }
```

2. Отобразите в отчете исходный код.
3. Поместите курсор на метод **A** либо в объявлении метода, либо в вызове метода.
4. В меню **Правка - Выполнить рефакторинг…** выберите **Удалить параметры**, чтобы отобразить диалоговое окно «**Удалить параметры…**».



*Вы также можете нажать сочетание клавиш CTRL+R, CTRL+V, чтобы отобразить диалоговое окно «**Удалить параметры ».***

5. Используя поле **Параметр**, поместите курсор на **int i**, а затем нажмите **Удалить**. Отобразите новое окно параметров в отчете.
6. Нажмите **ОК** .
7. Отобразите в отчете программный код с изменениями.

Примечания

Вы можете удалить параметры из объявления метода или вызова метода. Поместите курсор в объявление метода или имя делегата и вызовите функцию удаления параметров.

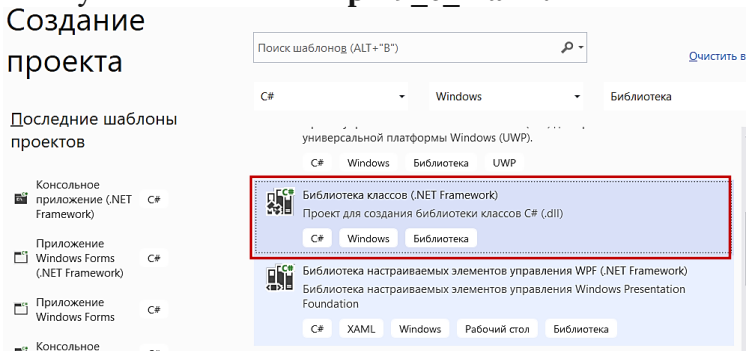
Внимание!

*Удалить параметры позволяет удалить параметр, на который есть ссылка в теле члена, но не удаляет ссылки на этот параметр в теле метода. Это может привести к ошибкам сборки в вашем коде. Однако вы можете использовать диалоговое окно **Preview Changes** для просмотра кода перед выполнением операции рефакторинга.*

Задание 6. Рефакторинг параметров изменения порядка (C#)

Чтобы изменить порядок параметров

1. Создайте библиотеку классов с именем **pr19_6_Фамилия**



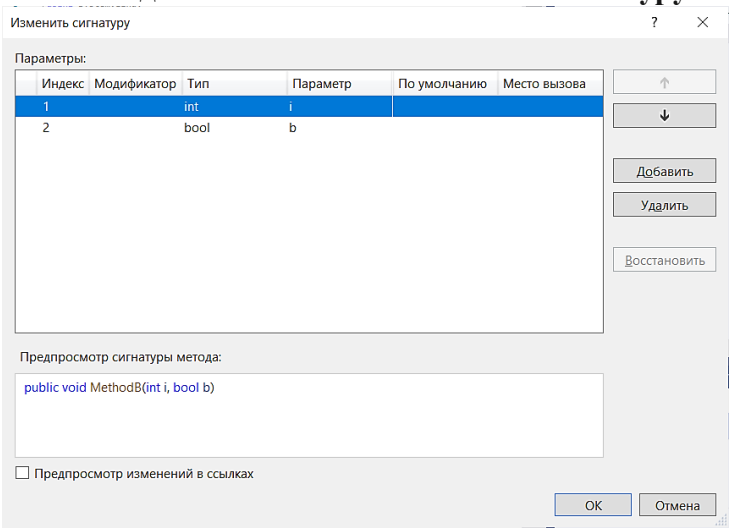
а затем замените **Class1** ее приведенным ниже примером кода.

```
class ProtoClassA
{
    // Вызвать для 'MethodB'.
    ссылка: 1
    public void MethodB(int i, bool b) { }
}

Ссылок: 0
class ProtoClassC
{
    Ссылок: 0
    void D()
    {
        ProtoClassA MyClassA = new ProtoClassA();

        // Вызвать для 'MethodB'.
        MyClassA.MethodB(0, false);
    }
}
```

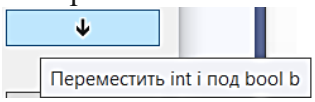
2. Отобразите в отчете исходный код.

3. Поместите курсор на **MethodB**, либо в объявлении метода, либо в вызове метода.
4. В меню **Правка – Выполнить рефакторинг…** нажмите «**Упорядочить параметры…**».

Появится диалоговое окно «**Изменить сигнатуру**».



5. В диалоговом окне выберите int i в списке « **Параметры** » и нажмите кнопку со стрелкой «Вниз».



Кроме того, вы можете перетащить **int i** после **bool b** списка параметров.

6. **нажмите « ОК »** .

MethodB в этом примере обновляются объявление метода и все сайты вызова метода.

7. Отобразите в отчете программный код с изменениями.

Примечания

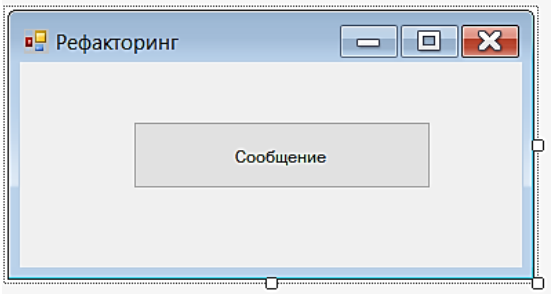
Вы можете изменить порядок параметров из объявления метода или вызова метода. Поместите курсор на или рядом с объявлением метода или делегата, но не в теле.

Задание 7. Выполните самостоятельные задания:

Для каждого самостоятельного задания в файле отчета отобразить исходный код. Описать технологию и процесс выполнения задания (со скриншот окнами выполнения). Отобразить полученные изменения программного кода и структуру каждого решения.

7.1. Необходимо вынести условное логическое выражение в отдельный метод

- Для этого создайте **приложение Windows Form** под именем **pr19_7.1_Фамилия**




```
public partial class Form1 : Form
{
    ссылка: 1
    public Form1()
    {
        InitializeComponent();
    }

    ссылка: 1
    private void button1_Click(object sender, EventArgs e)
    {

        string connectionString = Properties.Settings.Default.ConnectionString;

        if (connectionString == null)
        {
            connectionString = "Строка соединения по умолчанию";
        }

        MessageBox.Show(connectionString);
        /* ... Продолжение длинного метода ... */
    }
}
```

- Далее используя необходимый прием рефакторинга, вынесите условное логическое выражение в отдельный метод.

7.2. Меню Organize Usings

Очень полезно поддерживать упорядоченный список директив using в каждом файле (на языке C#) и ссылаться только на те пространства имен, которые действительно необходимы в данном файле. После рефакторинга кода может выясниться, что в начале файла содержится множество директив using, которые больше не используются. Чтобы определить, какие из этих директив используются, а какие нет, вместо метода проб и ошибок можно выполнить операцию, предусмотренную системой Visual Studio.

Для этого достаточно щелкнуть правой кнопкой мыши в окне редактора кода и выбрать команду **Удалить и отсортировать директивы usings** (в языке C#). Неиспользуемые директивы using, их альтернативные имена и внешние альтернативные имена сборок из исходного файла будут удалены.

- Выполните удаление ненужных директив в проекте задания 7.1.

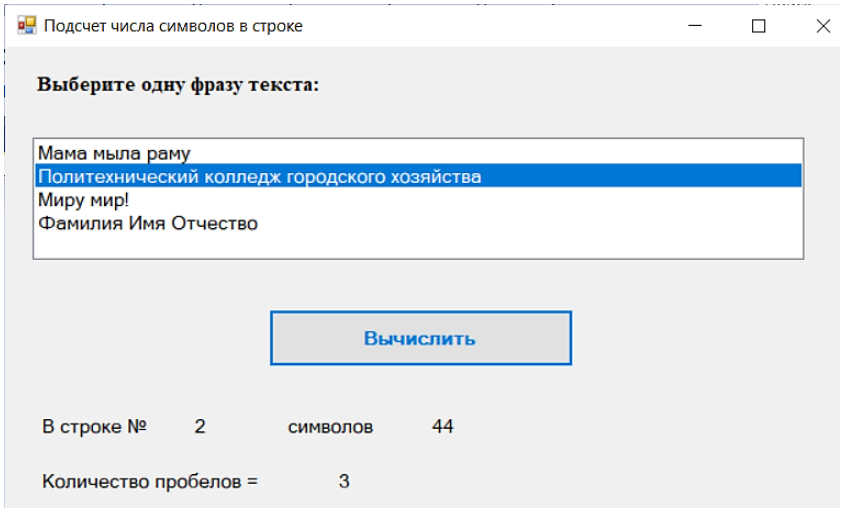
7.3. Требуется извлечь в отдельный интерфейс первый метод ShouldBeInInterface

- Создайте консольное приложение с именем **pr19_7.3_Фамилия**, а затем замените Program следующим кодом.

```
Ссылок: 0
public class ConcreteClass
{
    Ссылок: 0
    public void ShouldBeInInterface()
    { /* ... */ }

    Ссылок: 0
    public void AnotherNormalMethod(int ParameterA, int ParameterB)
    { /* ... */ }

    Ссылок: 0
    public void NormalMethod()
    { /* ... */ }
}
```

- Примените требуемый рефакторинг для первого метода.

7.4. Создайте **приложение Windows Form** под именем **pr19_7.4_Фамилия**



Фамилию Имя Отчество введите свое

Текст обработчика нажатия кнопки «**Вычислить**» приведен ниже:

```
private void button1_Click(object sender, EventArgs e)
{
    // Получаем номер выделенной строки
    int index = listBox1.SelectedIndex;
    // Считываем строку в переменную str
    string str = (string)listBox1.Items[index];
    // Узнаем количество символов в строке
    int len = str.Length;
    // Считаем, что количество пробелов равно 0
    int count = 0;
    // Устанавливаем счетчик символов в 0
    int i = 0;
    // Организуем цикл перебора всех символов в строке
    while (i < len)
    {
        // Если нашли пробел, то увеличиваем счетчик пробелов на 1
        if (str[i] == ' ')
            count++;
            i++;
    }
    label3.Text = index.ToString();
    label5.Text = len.ToString();
    label7.Text = count.ToString();
}
```

- отредактируйте программный код, чтобы нумерация строк выводилась с 1

- добавьте на форму элементы:

и программный код:

```
//определяем количество символов, заданных в поле textBox1
char input = Convert.ToChar(textBox1.Text);
int simv = 0;
for (int j = 0; j < len; j++)
    {
        if (input == str[j])
        {
            simv++;
        }
    }
label10.Text = simv.ToString();
```

- используя рефакторинг кода извлеките метод, определяющий количество пробелов в строке
- используя рефакторинг кода извлеките метод, определяющий количество заданных символов в строке
- выполните удаление ненужных директив в проекте задания 7.1.

Задание 8: Приведите примеры реализации рефакторинга с использованием следующих плагинов:
− Visual Assist X
− CodeRush.
− ReSharper
Для поиска информации используйте сеть Интернет.

Контрольные вопросы:

1. Что понимается под рефакторингом?
2. Возможности рефакторинга кода, которые распознаются **в Visual Studio** (заполните таблицу):

Прием рефакторинга	Описание	Технология выполнения