МИНОБРНАУКИ РОССИИ

————————————

Федеральное государственное бюджетное   
образовательное учреждение высшего образования

«Омский государственный технический университет»

**П. П. Степанов, А. А. Кабанов,   
В. А. Никонов, Т. С. Павлюченко**

**ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ   
ПРОГРАММИРОВАНИЕ**

Учебное текстовое электронное издание  
локального распространения

В трех частях

Часть 1

*Рекомендовано редакционно-издательским советом  
Омского государственного технического университета*

Омск  
Издательство ОмГТУ  
2021

——————————————————————————————————

Сведения об издании: [1](#МИНОБР), [2](#УДК) © ОмГТУ, 2021

ISBN 978-5-8149-3301-0 (ч. 1)

ISBN 978-5-8149-3300-3

УДК 004.43; 681.3.062

ББК 32.973.4

С79

Рецензенты:

*Л. В. Гайворонская*,руководитель программы ООО «Люксофт Профешнл»;

*П. А. Кульбида*,ведущий программист OmsDotNet

**Степанов, П. П.** Объектно-ориентированное программирование : учеб. пособие / П. П. Степанов, А. А. Кабанов, В. А. Никонов, Т. С. Павлюченко ; Минобрнауки России, Ом. гос. техн. ун-т. – Омск : Изд-во ОмГТУ, 2021– . – Сис­тем. требования: процессор с частотой 1,3 ГГц и выше ; 256 Мб RAM и более ; свободное место на жестком диске 300 Мб и более ; Windows XP и выше ; разрешение экрана 1024×768 и выше ; CD/DVD-ROM дисковод ; Adobe Acrobat Reader 5.0 и выше. – Загл. с титул. экрана. – ISBN 978-5-8149-3300-3.

Ч. 1. – 2021. – CD-ROM (3,61 Мб) : ил. – ISBN 978-5-8149-3301-0.

Издание состоит из трех частей. В первой части представлен теоретический материал по высокоуровневым языкам программирования, необходимый для подготовки студентов в области разработки программных систем с применением объектно-ориентированного подхода. Приведены задания для практических занятий по дисциплине «Объектно-ориентированное программирование».

Предназначено для студентов всех форм обучения по специальности 09.05.01 «Автоматизированные системы управления специального назначения» и по направлениям подготовки бакалавриата 09.03.01 «Информатика и вычислительная техника», 09.03.04 «Программная инженерия».

Редактор *К. В. Обухова*

Компьютерная верстка *Е. В. Макарениной*

*Для дизайна этикетки использованы материалы*

*из открытых интернет-источников*

–––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

Сводный темплан 2021 г. © ОмГТУ, 2021

Подписано к использованию 15.09.21.

Объем 3,61 Мб.

ВВЕДЕНИЕ

Значимость такой дисциплины, как «Объектно-ориентированное программирование», в современном мире сложно переоценить. Всё большее   
и большее число звеньев производственных, экономических, финансовых или коммуникационных цепочек, создаваемых человечеством, содержат   
в своём составе, а то и целиком состоят из средств вычислительной техники. И каждый вычислительный узел требует своего набора программного обеспечения.

При этом существуют различные подходы к решению задачи программирования, интерпретируемой как создание программы – набора инструкций вычислительному устройству, предназначенному для обработки информации.

Объектно-ориентированная парадигма – это определённый класс таких подходов, представляющий собой набор шаблонов и методик создания программного обеспечения. Центральным элементом данной парадигмы является понятие *объекта* как нечто, обладающего:

* набором *параметров или свойств*, однозначно характеризующим его среди прочих объектов;
* набором *методов*, которые позволяют ему осуществлять некоторые действия как с самим собой, так и с другими объектами, по сути описывая поведение.

Следующее понятие объектно-ориентированного программирования (ООП) – это *класс*, более абстрактное относительно *объекта* понятие, описывающее множество *объектов*, обладающих одинаковыми *свойст­вами* (но не обязательно одинаковыми значениями этих свойств) и *методами*. Иначе говоря, *класс* описывает структуру *свойств* и *методов* объектов, которые являются его экземплярами.

Реализация понятий объекта и класса в конкретном языке программирования идёт с соблюдением следующих принципов:

* *Абстракция* – заключается в выделении наиболее значимых в контексте решаемой задачи свойств и особенностей поведения объектов.
* *Инкапсуляция* – обеспечивает сокрытие конкретных деталей реализации свойств и методов объектов.
* *Наследование* – избавляет от необходимости дублирования повторяющихся свойств и методов в родственных классах.
* *Полиморфизм* – позволяет модифицировать наследуемые свойства и методы классов-потомков.

Таким образом, задача программиста, создающего программу в объектно-ориентированной парадигме, заключается в создании *объектно-ориентированной модели* предметной области, представив её в виде совокупности взаимодействующих объектов, относящихся к определённым классам.

Объекты как конкретные экземпляры классов фактически возникают лишь в момент выполнения программы. Структурной основой для них оказывается совокупность описаний классов, составляющих основное содержание программы. По сути классы оказываются кирпичиками, обеспечивающими выстраивание ООП-модели, а их свойства и методы обеспечивают выстраивание связей между ними.

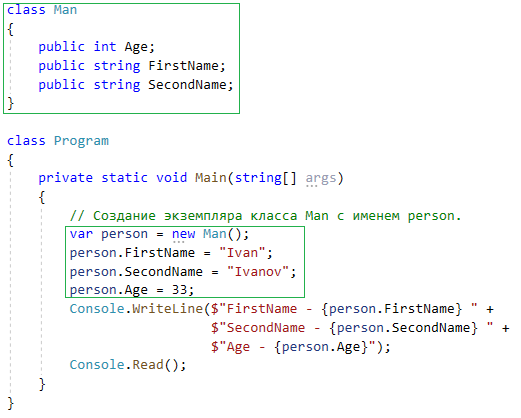
Основное содержание пособия представляет собой описание различных аспектов создания классов на языке C# (версия 7.0). В приложении приведены исходные коды на C# для обучения объектно-ориен­ти­ро­ванному программированию.

Авторы выражают благодарность Каракозову Дмитрию, Янковой Еле­не, Надточию Павлу и Огорокову Сергею за замечания и дополнения, сделанные в процессе работы над пособием.

1. КЛАССЫ, СВОЙСТВА, ПЕРЕЧИСЛЕНИЯ,   
ПЕРЕГРУЗКА ОПЕРАТОРОВ,   
МЕТОДЫ БАЗОВОГО КЛАССА OBJECT

1.1. Классы

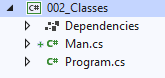
Классы – это основная единица в объектно-ориентированном программировании [1]. Основная задача класса – это моделирование вещей из реального мира. На рис. 1.1 приведен пример создания класса Man (человек), содержащего три публичных переменных (целочисленное открытое поле Age (возраст), строковое открытое поле FirstName (Имя), строковое открытое поле SecondName (Фамилия)). Для того чтобы создать экземпляр класса в языке C# служит ключевое слово **new** (рис. 1.1).

****

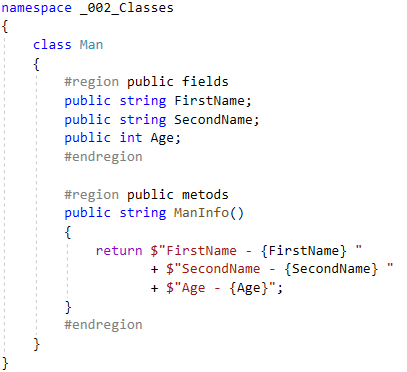
*Рис. 1.1.* Пример класса Man

1.2. Пространства имен

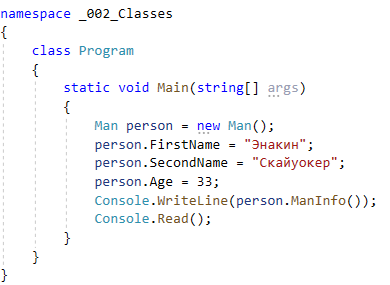
Для того чтобы классы из разных файлов были доступны друг другу, они должны находиться в одном пространстве имен (рис. 1.2). На рис. 1.3 представлен класс Man в пространстве имен \_002\_Classes, на рис. 1.4 – пример использования класса Man.



*Рис. 1.2.* Пример структуры проекта



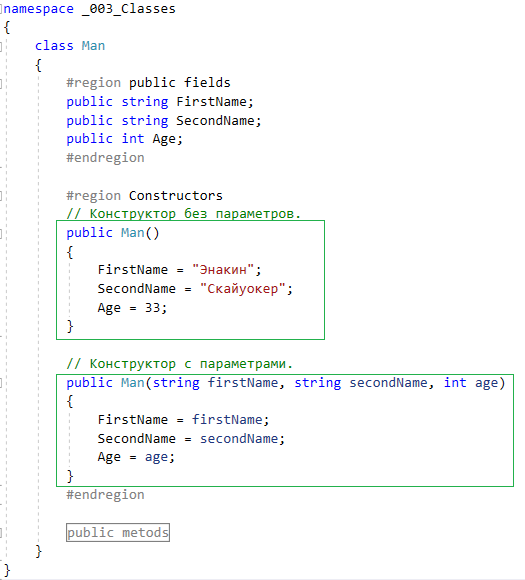
*Рис. 1.3.* Пример класса Man   
из пространства имен \_002\_Classes



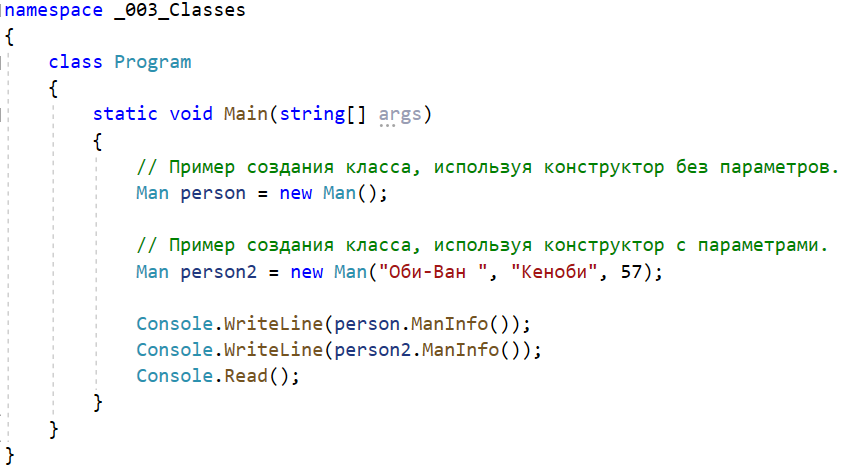
*Рис. 1.4.* Пример использования класса Man   
в одном пространстве имен \_002\_Classes

1.3. Конструкторы классов

Каждый раз, когда создается экземпляр класса, вызывается его конструктор. Класс содержит конструктор по умолчанию, который устанавливает всем полям класса значения по умолчанию. Язык C# позволяет создавать свою реализацию конструкторов. Можно реализовать как конструктор без параметров, так и с параметрами. Конструкторов с параметрами может быть сколько угодно, отличия должны быть в сигнатуре методов по числу и типу принимаемых параметров. В языке C# метод-кон­структор должен иметь то же имя, что и класс, и не иметь тип возвращаемого значения. На рис. 1.5 приведен пример класса, содержащего два конструктора: не принимающего никаких параметров и принимающего три параметра (два строковых значения и одно целочисленное). На рис. 1.6 приведен пример создания экземпляров классов с помощью этих конструкторов.



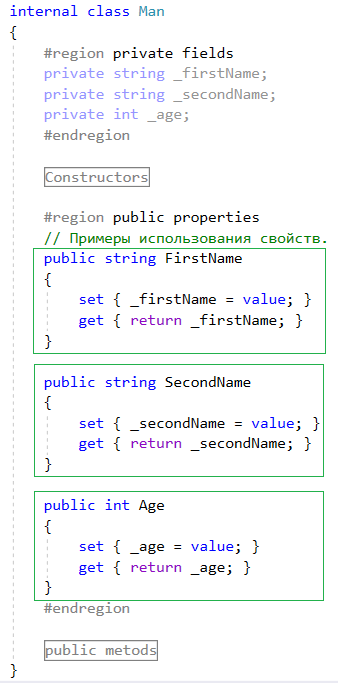
*Рис. 1.5.* Пример класса, содержащего конструктор по умолчанию   
и конструктор с параметрами



*Рис. 1.6.* Пример создания экземпляров классов с использованием   
конструктора по умолчанию и конструктора с параметрами

1.4. Свойства

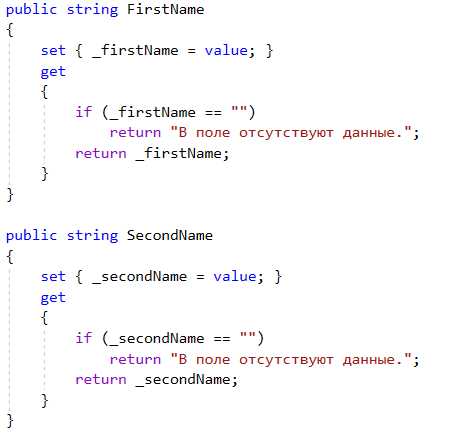
Свойства – это особый стереотип языка C#, позволяющий работать   
с методами так, словно это поля класса. На самом деле свойства – это лишь «синтаксический сахар» и при компиляции кода, написанного на C#, в IL-код свойства заменяются методами (одним или двумя в зависимости от того, указали ли мутатор (**set**) и аксессор (**get**)). Свойства могут использоваться для инкапсуляции доступа к переменным класса. На рис. 1.7 приведен пример класса, содержащего три приватных поля (недоступных из других классов) и три открытых свойства, инкапсулирующих доступ к этим полям.



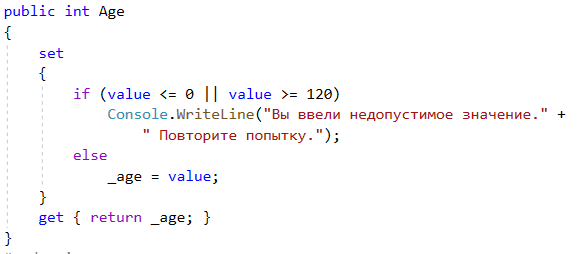
*Рис. 1.7.* Пример класса, содержащего свойства

Чтобы свойство было доступно для чтения, в его теле нужно реализовать конструкцию get, которая возвращает какое-либо значение при обращении к этому свойству.

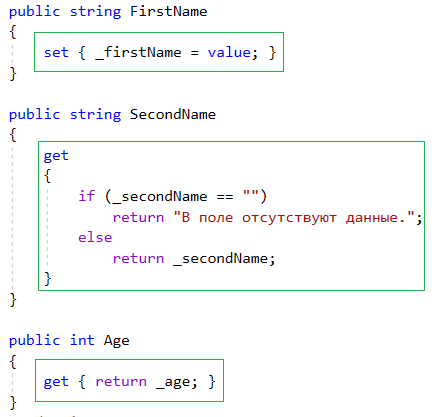
Чтобы свойство было доступно для записи, в его теле нужно реализовать конструкцию set, которая вызывается, если свойству пытаются присвоить значение. Для работы со значением, которое присвоено свойству, используется ключевое слово value. В теле конструкций **get**, **set** можно вставлять различные валидации. На рис. 1.8 приведен пример валидации на пустые строки, на рис. 1.9 – пример валидации на допустимый диапазон значений. Свойства можно сделать доступными только для чтения или записи. Для этого в теле свойства нужно реализовать только один мутатор (**set**) или аксессор (**get**) (рис. 1.10).



*Рис. 1.8.* Пример свойств, содержащих валидацию   
на пустые строки

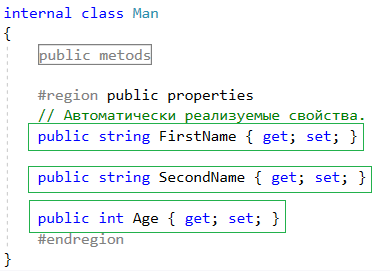


*Рис. 1.9.* Пример свойства, содержащего проверку на диапазон значений



*Рис. 1.10.* Пример свойств, доступных только для чтения (Age)   
или записи (SecondName)

Язык C# позволяет использовать автосвойства (рис 1.11), для этого   
в теле свойства достаточно указать пустые мутатор (**set**) и аксессор (**get**). Внутри при компиляции в IL-код будет создана приватная переменная для значения этого свойства.

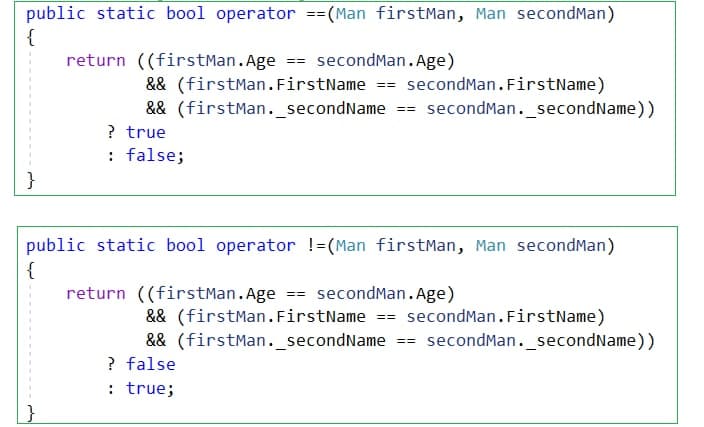


*Рис. 1.11.* Пример автосвойств

## 

1.5. Перегрузка операторов

Язык C# позволяет выполнить перегрузку операторов для пользовательских типов данных, т. е. добавить реализацию оператора, где один или оба операнда являются данными пользовательского типа [3].



*Рис. 1.12.* Пример перегрузки операторов «==» и «!=»,   
где оба операнда являются экземплярами класса Man

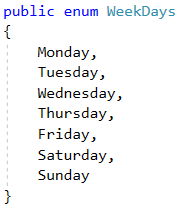
На рис. 1.12 приведен пример перегрузки операторов равенства и неравенства для класса Man.

1.6. Перечисления (Enums)

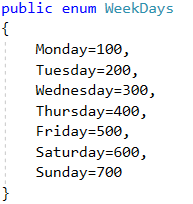
Перечисления – это значимый тип (значения хранятся в стеке потока, а не на управляемой куче), содержащий в себе наборы констант и связанных с ними значений [4]. На рис. 1.13 представлен пример перечисления. По умолчанию первое значение перечисления имеет значение 0 (Monday), а каждое последующее больше на единицу приводящего (Tuesday =1, Wednesday =2 и т. д.). Перечисления позволяют задавать произвольные значения константам (рис. 1.14). При работе с перечислением можно явно указать, каким типом являются константы. Для это используется «:», которое также используется для наследования в классах. На рис. 1.15 и 1.16 приведены примеры перечислений с явным указанием типа.

Использование перечислений позволяет решить две проблемы:

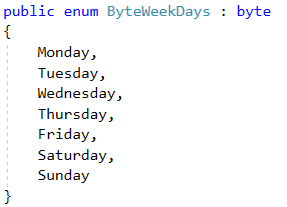
1. придать осмысленность магическим числам;
2. избежать ситуации, когда пользователь ввел значение вне доступного диапазона.



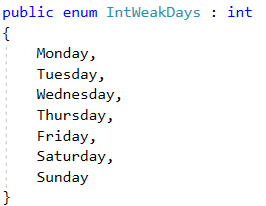
*Рис. 1.13.* Пример перечисления   
со значениями по умолчанию



*Рис. 1.14.* Пример перечисления со значениями,   
заданными пользователем



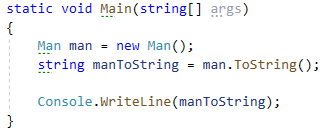
*Рис. 1.15.* Пример перечисления с типом byte



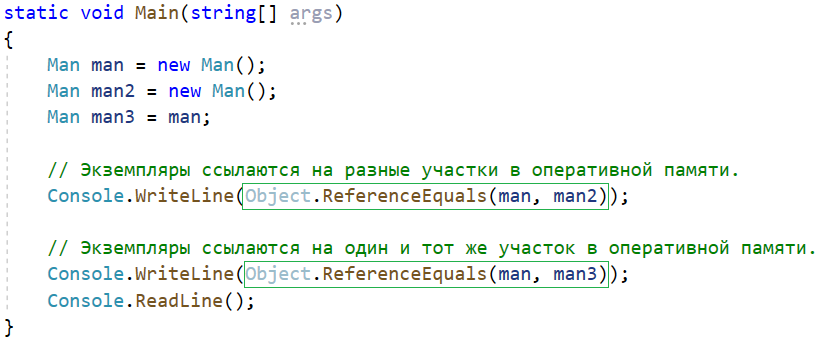
*Рис. 1.16.* Пример перечисления с типом int

1.7. Методы класса object

Object – это базовый класс, от которого наследуются все остальные классы. Поэтому методы, которые доступны в типе object, доступны в любых классах. Виртуальный метод (более подробно о виртуальных методах рассмотрено в следующей главе) **ToString()** преобразует тип в строку,   
по умолчанию возвращает полный тип в строковом представлении [5]   
(рис. 1.17). Открытый статический метод **ReferenceEquals()** служит для того, чтобы проверить, что два экземпляра класса ссылаются на один и тот же объект в памяти [6] (рис. 1.18).

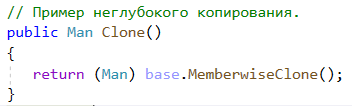


*Рис. 1.17.* Пример использования метода ToString

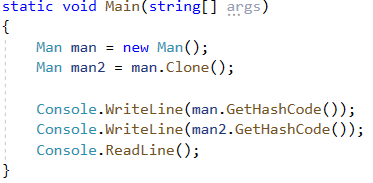


*Рис. 1.18.* Пример использования метода ReferenceEquals

Метод **MemberwiseClone()** [7] служит для неглубокого копирования объекта в памяти. При использовании метода **MemberwiseClone()** копируется дамп участка оперативной памяти (рис. 1.19). На практике не рекомендуется копировать экземпляр методом **MemberwiseClone()**, рекомендуется   
у класса реализовывать интерфейс **IClonable**. Виртуальный метод **GetHashCode()** возвращает хеш-код объекта, который уникален для каждого экземпляра класса. Пример использования метода **GetHashCode()** представлен на рис. 1.20.

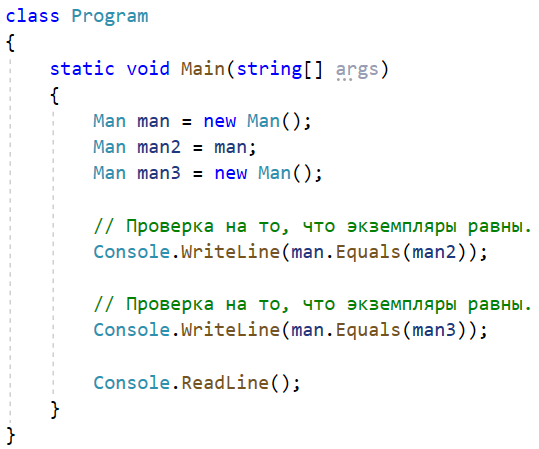


*Рис. 1.19.* Пример использования   
метода MemberwiseClone

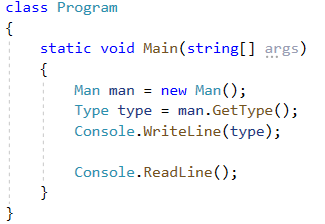


*Рис. 1.20.* Пример использования   
метода GetHashCode

Виртуальный метод **Equals()** (рис. 1.21) в отличие от метода **ReferenceEquals()** может быть перегружен у класса потомка и служит для проверки того, что классы совпадают. Считается хорошей практикой вызывать внутри метода **Equals()** метод **GetHashCode()**. Следовательно, при переопределении одного из этих методов должен быть переопределен и второй (подробно о переопределении методов рассмотрено в следующей главе). Метод **GetType()** возвращает экземпляр типа Type, содержащий информацию о типе объекта (рис. 1.22).



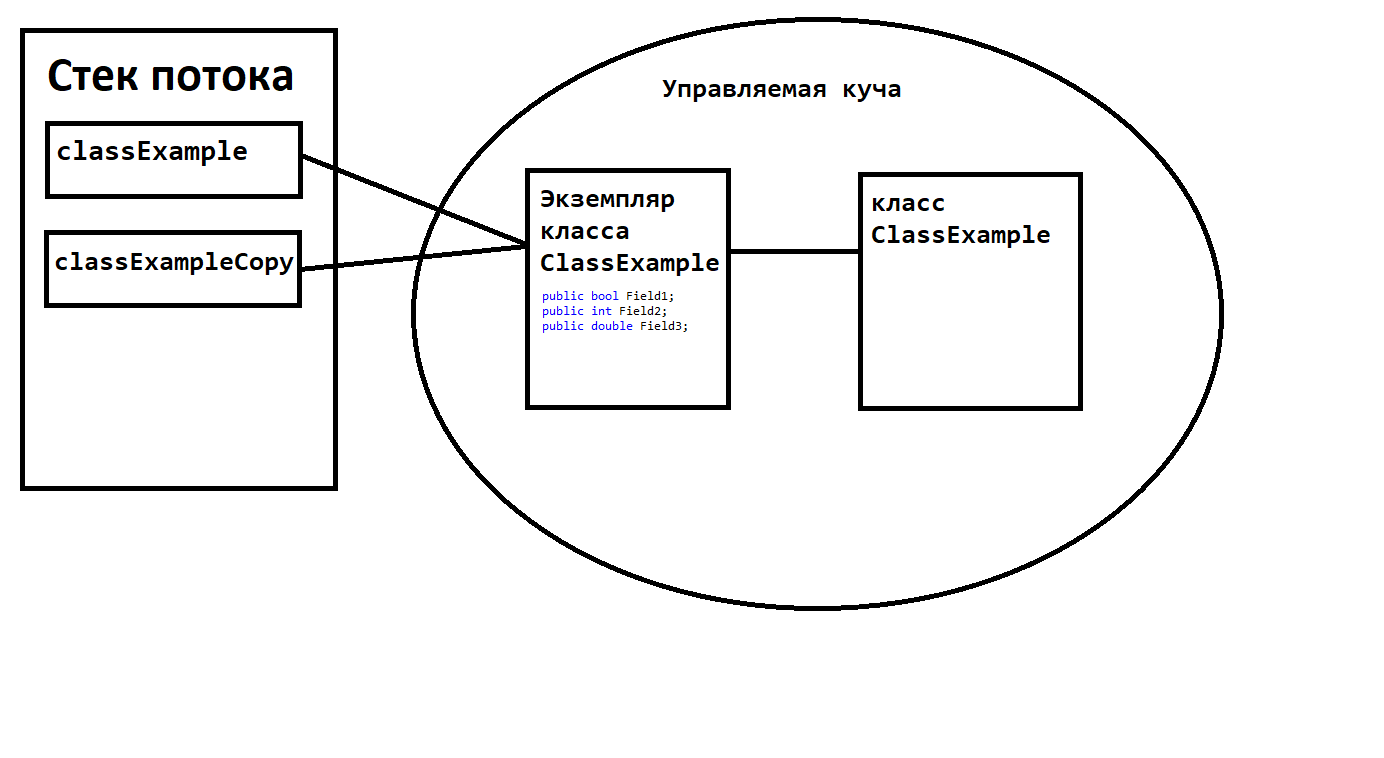
*Рис. 1.21.* Пример использования   
метода Equals()



*Рис. 1.22.* Пример использования   
метода GetType

1.8. Тип данных struct

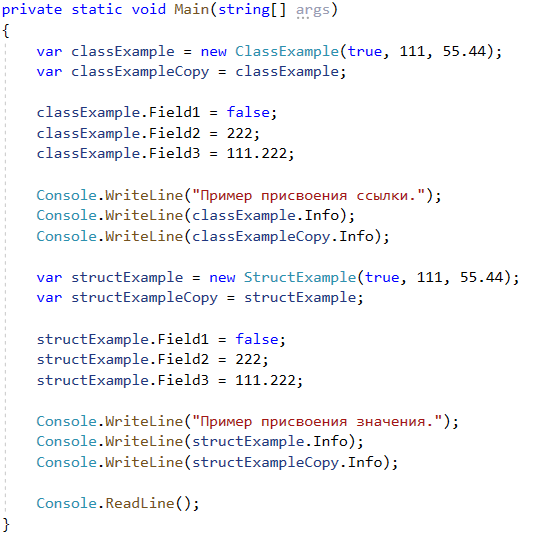
Структуры, как и классы, могут содержать состояние и поведение объектов. В отличие от классов все структуры неявно унаследованы от типа **ValueType**,который переопределяет механизм хранения в оперативной памяти. Классы хранят в стеке потока указатель на область памяти   
в управляемой куче, где хранится объект (рис. 1.23). В отличие от классов структуры хранят в стеке потока сами значения (рис. 1.24). При использовании оператора«=»у экземпляра класса происходит копирование ссылки на управляемую кучу (рис. 1.25, 1.26). При этом оба объекта ссылаются на одну и ту же область на управляемой куче, и, следовательно, изменение одного экземпляра класса меняет состояние всех экземпляров классов, ссылающихся на одну и ту же область на управляемой куче. При использовании оператора «=»уструктуры создается новая копия объекта, и изменение состояния одного из экземпляров меняет состояние только текущего экземпляра структуры (рис. 1.25, 1.26). По умолчанию при передаче экземпляра класса в качестве параметра метода происходит передача ссылки и при изменении переменной внутри метода изменится я и переменная извне (рис. 1.27, 1.28). По умолчанию при передаче экземпляра структуры в качестве параметра метода происходит копирование экземпляра. При изменении внутри метода, меняется только локальная переменная метода.



*Рис. 1.23.* Пример хранения экземпляров класса

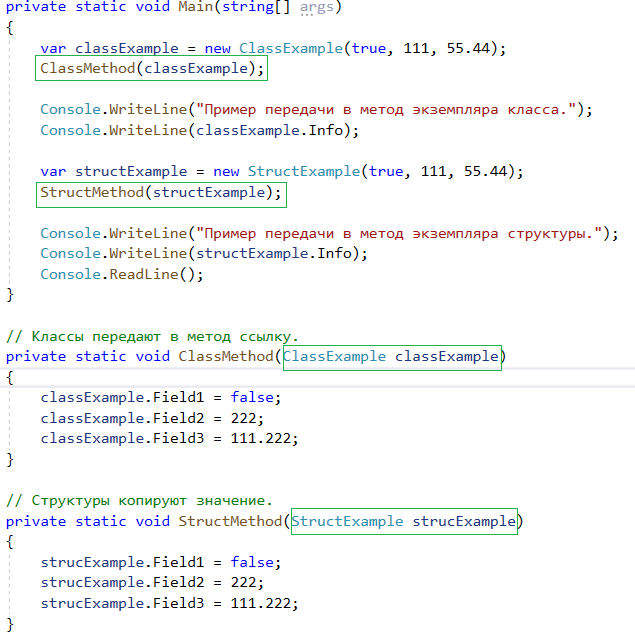


*Рис. 1.24.* Пример хранения экземпляров структуры



*Рис. 1.25.* Пример использования оператора «=» для классов и структур

|  |  |
| --- | --- |
| *Рис. 1.26.* Результат работы  программы |  |



*Рис. 1.27.* Пример передачи экземпляров классов и структур   
в качестве параметра метода

|  |  |
| --- | --- |
| *Рис. 1.28.* Результат работы  программы |  |

## 

Для расширения и углубления знаний по данной теме рекомендуется изучить информационные источники [1–7, 11–14, 20].

ПРАКТИЧЕСКОЕ ЗАДАНИЕ 1

*Задача 1*

Используя Visual Studio, необходимо создать проект по шаблону Con­sole Application.

Требуется создать класс с именем Animal:

1. в теле класса требуется создать закрытые поля:

* Имя;
* Возраст;
* Количество конечностей;
* Вид животного (травоядное, хищник и т. д.) (Enum);
* Среда обитания (Enum);
* Континент, на котором обычно обитает (Enum).

2) реализовать базовый конструктор и конструктор с параметрами;

3) перегрузить операторы «==», «!=»;

4) реализовать методы GetSound(), CanFly(), CanSwim(), ExistsTail() (Наличие хвоста);

5) реализовать по одному свойству, доступному только для записи,   
чтения;

6) создать несколько экземпляров класса Animal.

*Задача 2*

Реализовать класс Figure:

1) класс содержит закрытое поле \_square;

2) класс имеет несколько конструкторов с параметрами, согласно которым вычисляется площадь фигур;

3) класс имеет открытое, доступное только для чтения свойство Square, возвращающее значение поля \_square.

Контрольные вопросы

1. Как создать класс в языке C#?

2. Что такое конструктор по умолчанию и конструктор с параметрами?

3. Как вызвать конструктор по умолчанию и конструктор с пара­метрами?

4. Что такое свойство?

5. Для чего нужны операторы set и get?

6. Что такое автосвойcтво?

7. Что такое перечисление?

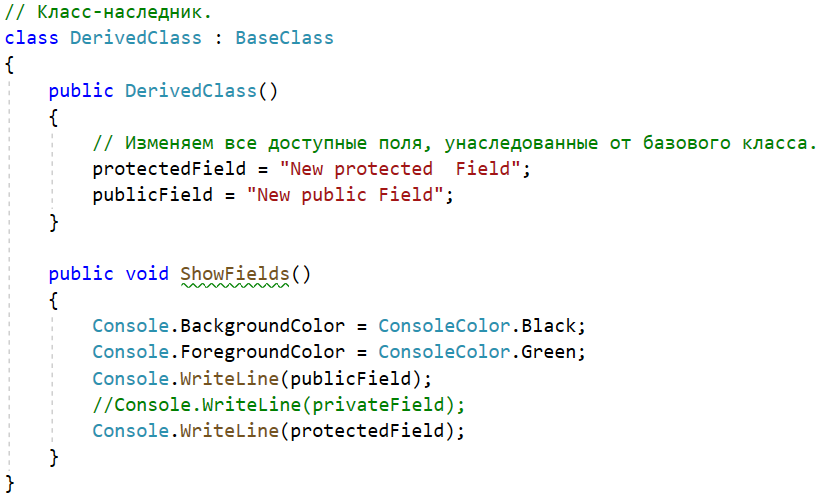
8. Для чего нужны перечисления?

9. От каких классов может «наследоваться» перечисление?

10. Назовите методы базового класса Object.

2. НАСЛЕДОВАНИЕ, МОДИФИКАТОРЫ ДОСТУПА

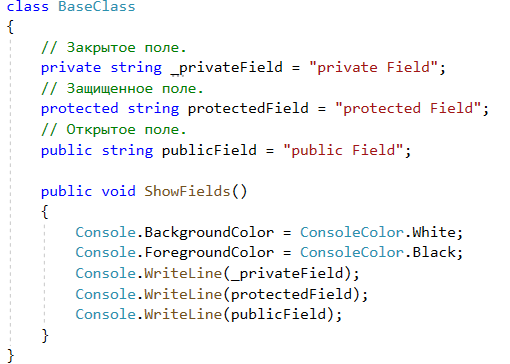
Наследование – одна из базовых парадигм объектно-ориентирован­но­го программирования. Наследование служит для того, чтобы создавать новые классы на основе уже существующих, копируя их члены и методы. В языке C# оператором наследования является «:» (рис. 2.1).

**

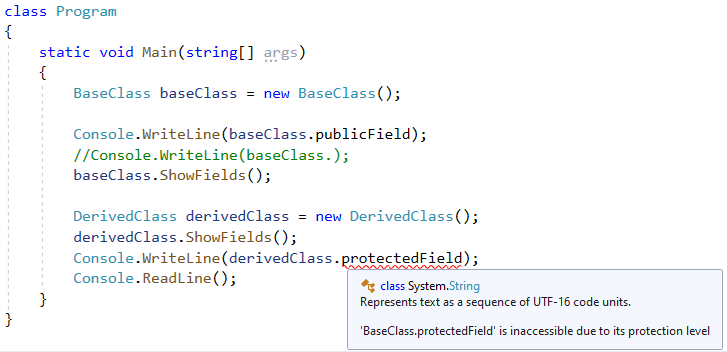
*Рис. 2.1.* Пример наследования

2.1. Модификаторы доступа private, public, protected

По умолчанию в языке C# все поля и методы класса имеют модификатор **private**. Поля и методы, имеющие этот модификатор, могут быть вызваны только в текущем классе и не наследуются. Поля и методы класса, по­меченные модификатором доступа **public**, доступны из любых классов   
и наследуются в дочерних классах. Поля и методы класса, помеченные модификатором **protected**, наследуются в дочерних классах и доступны в текущем классе и классах-наследниках. На рис. 2.1 представлен базовый класс, содержащий три поля, помеченные разными модификаторами доступа. На рис. 2.2 представлен класс-наследник BaseClass. Класс-наследник DevivedClass унаследовал от базового класса две переменные, помеченные модификаторами доступа **public** и **protected**. Поле базового класса, помеченное модификатором доступа **private**, унаследовано не было. Однако хоть поле базового класса, помеченное модификатором доступа **protected**, и было унаследовано, оно доступно только в классе-наследнике (рис. 2.3).



*Рис. 2.2.* Пример базового класса

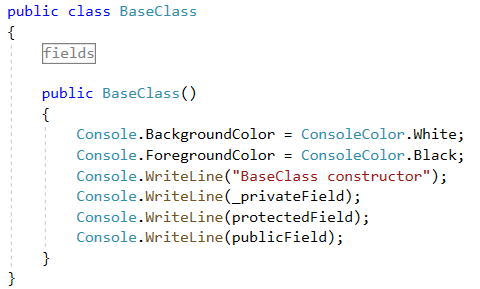


*Рис. 2.3.* Попытка обращения к полю или методу класса, помеченных   
модификатором доступа **protected** извне текущего и дочернего классов

2.2. Вызов конструкторов при наследовании

При создании класса вызываются конструкторы всех базовых классов по цепочке, начиная от самого базового класса в языке C# – **Object**.   
На рис. 2.4 приведен пример конструктора базового класса, который устанавливает цвет фона белым и цвет текста черным, выводит строку, что это конструктор базового класса, и значения полей на экран. На рис. 2.5 представлен пример класса-наследника, который изменяет значения полей, унаследованных из базового класса, меняет цвет фона на черный, а цвет текста на зеленый и выводит строку, что это конструктор класса-потомка и значения полей. На рис. 2.6 представлен пример создания экземпляра дочернего класса.

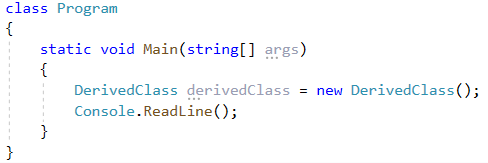
На рис. 2.7 представлен результат работы программы, на котором видно, что сначала вызывается конструктор базового класса, затем дочернего. По умолчанию вызывается конструктор базового класса без параметров, однако если родительский класс содержит несколько конструкторов (рис. 2.8), то при вызове конструктора дочернего класса можно явно указать, какой конструктор базового класса вызвать с помощью оператора «:» и ключевого слова **base** (рис. 2.9). На рис. 2.9 показан пример вызова конструктора базового класса, придающего два строковых параметра.



*Рис. 2.4.* Пример конструктора базового класса



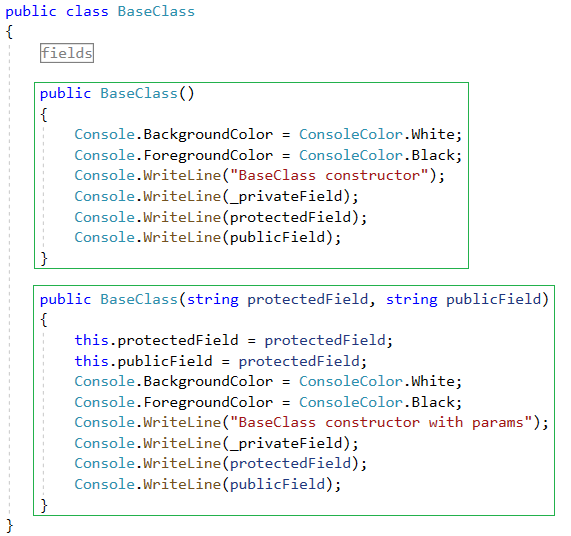
*Рис. 2.5.* Пример конструктора дочернего класса



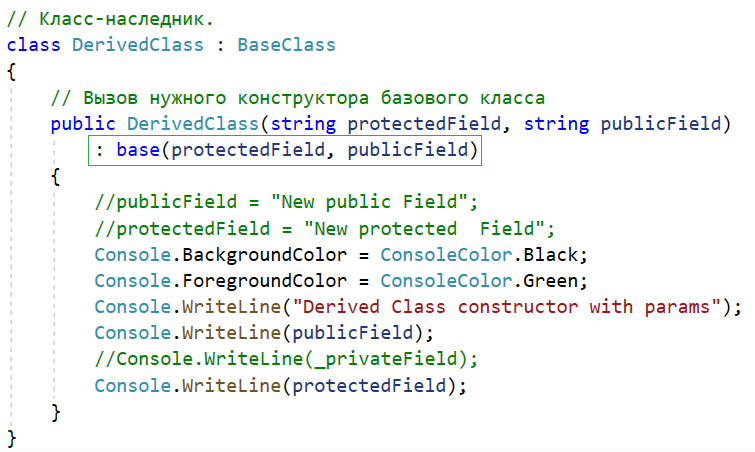
*Рис. 2.6.* Пример создания экземпляра дочернего класса



*Рис. 2.7.* Результат работы программы



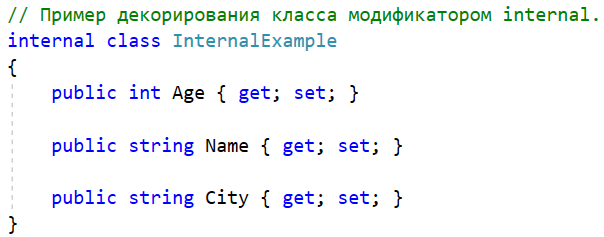
*Рис. 2.8.* Пример базового класса с двумя конструкторами



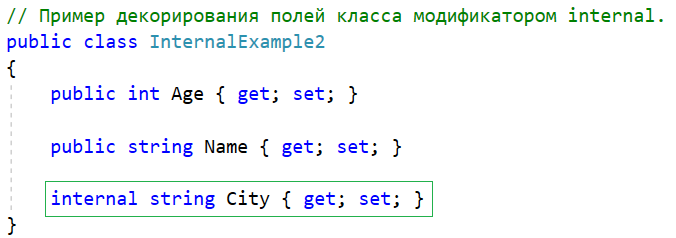
*Рис. 2.9.* Пример вызова конструктора базового класса,   
принимающего два строковых параметра

2.3. Модификатор доступа internal

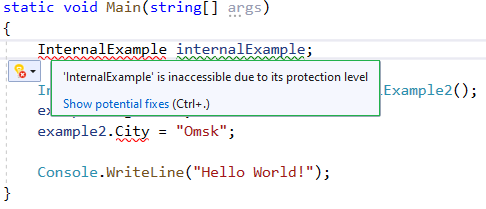
Модификатор доступа **internal** применим как к классам (рис. 2.10), так и к полям и методам класса (рис. 2.11). Если класс декорирован модификатором доступа **internal**, то он доступен для работы только в рамках текущей сборки (рис. 2.12). Если поля и методы класса декорированы модификатором доступа internal, то они доступны только в рамках текущей сборки, при этом сам класс и поля, а также методы, не помеченные модификатором доступа internal, доступны и в других сборках (рис. 2.13).



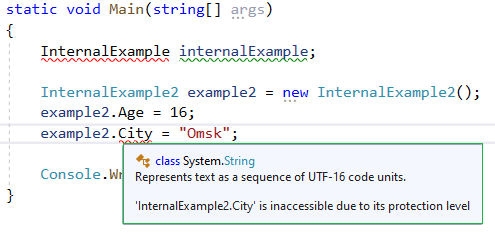
*Рис. 2.10.* Пример декорирования класса   
модификатором internal



*Рис. 2.11.* Пример декорирования полей класса   
модификатором internal



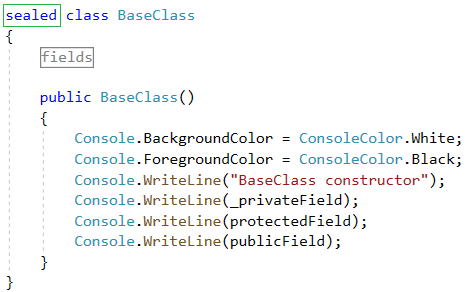
*Рис. 2.12.* Пример попытки использования класса,   
помеченного модификатором доступа internal в другой сборке



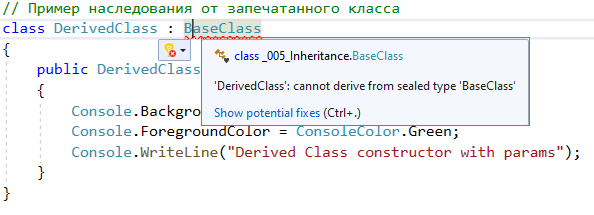
*Рис. 2.13.* Пример попытки использования метода класса,   
помеченного модификатором доступа internal в другой сборке

2.4. Модификатор доступа sealed

Модификатор доступа **sealed** применим к классам и служит для того, чтобы запретить наследование от класса, помеченного этим модификатором. Такие классы называют запечатанными. На рис. 2.14 приведен пример базового класса, помеченного модификатором доступа sealed, и у этого класса не может быть наследников. Попытка наследоваться от класса, помеченного модификатором доступа **sealed**, приведет к ошибке компиляции (рис. 2.15). Также модификатор доступа **sealed** может использоваться для запрета перегрузки виртуальных свойств и методов в дочерних классах.



*Рис. 2.14.* Пример класса, декорированного   
модификатором доступа sealed



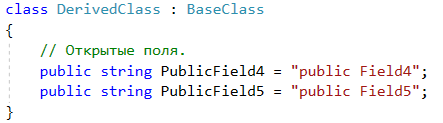
*Рис. 2.15.* Пример попытки наследоваться от класса,   
помеченного модификатором доступа sealed

2.5. Up Cast и Down Cast

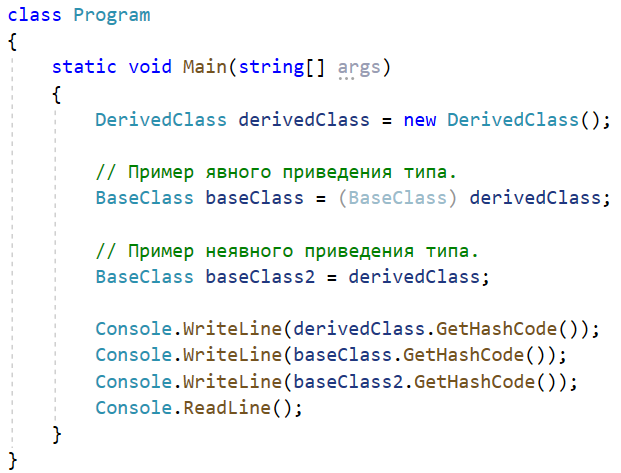
Up Cast – это приведение класса потомка к типу базового класса. При этом поля и методы, которые были объявлены в классе-потомке, становятся недоступными. На рис. 2.16 представлен базовый класс, содержащий три открытых переменных. На рис. 2.17 представлен класс-наследник, содержащий два открытых поля. На рис. 2.18 показаны примеры как явного, так и неявного преобразования типа к базовому типу (**Up Cast**).



*Рис. 2.16.* Пример базового класса

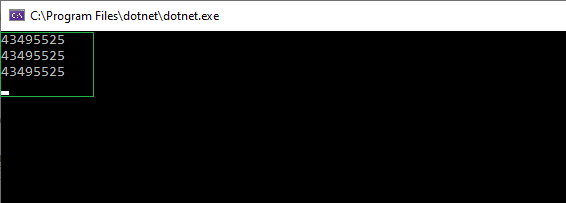


*Рис. 2.17.* Пример класса-наследника

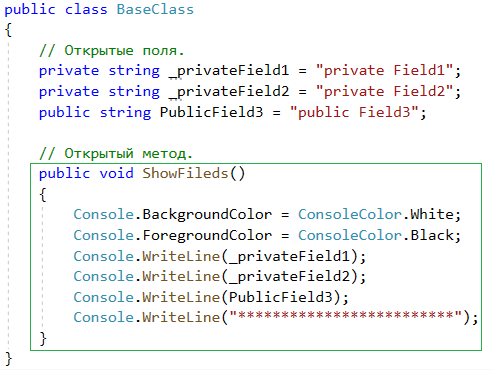


*Рис. 2.18.* Пример явного и неявного приведения типа

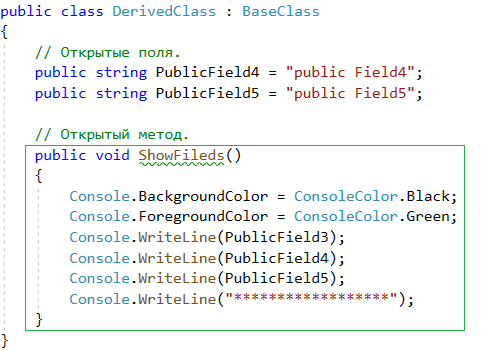
После приведения класса к базовому типу поля publicField4 и publicField5 ста­новятся недоступными. На самом деле эти поля никуда   
не пропадают, просто CLR скрывает их от пользователя. Более того, все три переменные, приведенные на рис. 2.18, ссылаются на одну и ту же область в оперативной памяти. Убедиться в этом можно, вызвав метод **GetHashCode()** на все переменные. При этом они будут равны (рис. 2.19). Если в классе-наследнике и классе-потомке есть методы с одинаковыми именами и одинаковой сигнатурой, то после приведения класса-наследника к базовому классу вызовется именно реализация метода базового класса – такая техника называется замещением методов. На рис. 2.20 представлен базовый класс, имеющий реализацию метода ShowFileds(), на рис. 2.21 – класс-наслед­ник, имеющий свою реализация метода ShowFileds().



*Рис. 2.19.* Пример использования метода GetHashCode

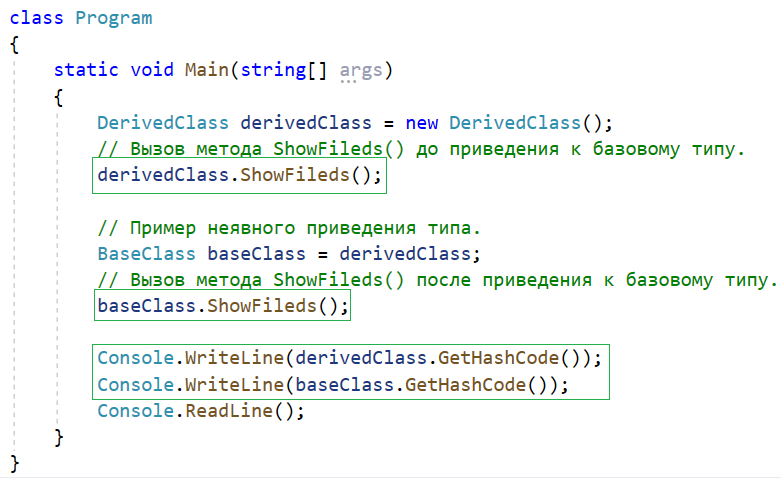


*Рис. 2.20.* Пример базового класса,   
содержащего реализацию метода ShowFileds()



*Рис. 2.21.* Пример класса-потомка,   
содержащего свою реализацию метода ShowFileds()

На рис. 2.22 приведен пример вызова ShowFileds() из класса-наслед­ника и вызова метода ShowFileds() после приведения к базовому классу.

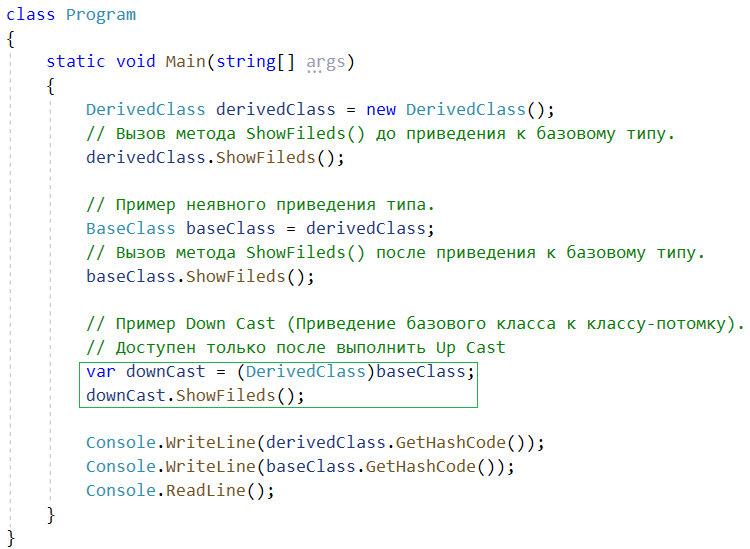


*Рис. 2.22.* Пример вызова метода ShowFileds()   
до приведения к базовому типу и после приведения к базовому типу

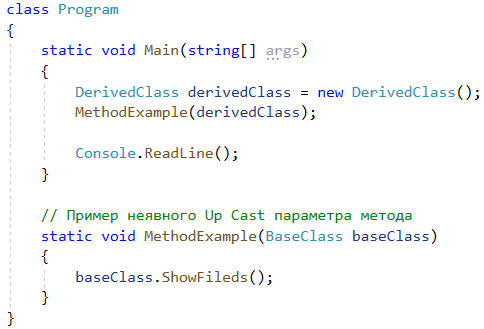
Результат работы программы представлен на рис. 2.23. **Down Cast** – это приведение базового класса к классу-потомку. Возможно только тогда, когда был произведён **Up Cast**, так как в противном случае в области памяти физически не будет полей и методов класса-наследника (рис. 2.24). На рис. 2.25 показан еще один пример неявного приведения к базовому классу.



*Рис. 2.23.* Результат работы программы



*Рис. 2.24.* Пример Down Cast

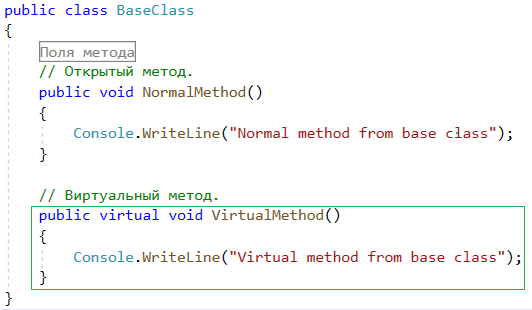


*Рис. 2.25.* Пример неявного приведения типов

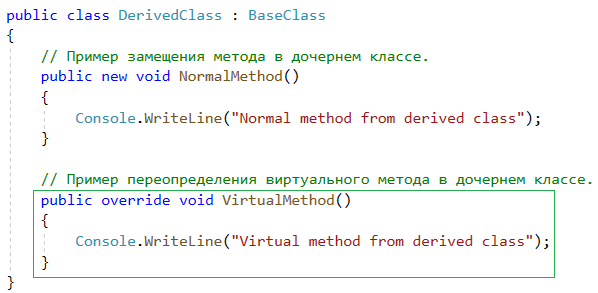
2.6. Виртуальные методы (virtual, override)

Язык C# позволяет объявить метод, свойство или индексатор (подробнее об индексаторах рассмотрено в главе 4) виртуальными с помощью ключевого свойства **virtual** (рис. 2.26). Позже в классе-потомке виртуальное свойство может быть переопределено с помощью ключевого слова **override** (рис. 2.27), такая техника носит название «переопределение виртуального метода». Отличие замещения от переопределения заключается в том, что если метод замещен в классе-потомке, то после приведения его к базовому типу вызовется реализация метода из базового типа. При переопределении виртуального метода всегда вызовется реализация переопределенного метода даже при проведении к базовому классу.

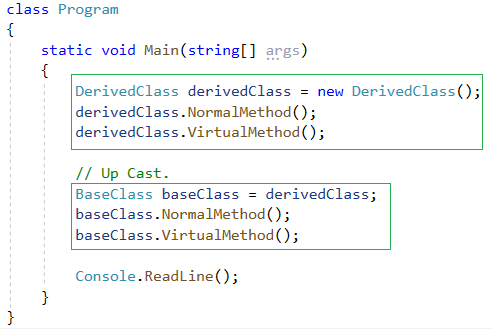
На рис. 2.28 и 2.29 приведен пример вызова обычного и виртуального методов. Как можно увидеть на рис. 2.29, даже после приведения к базовому классу вызывается реализация переопределённого метода. На рис. 2.30 представлен пример использования виртуальных свойств и их пе­регрузка (рис. 2.31). Используя ключевое слово **sealed**, можно запретить в дальнейшем переопределение метода (рис. 2.32).



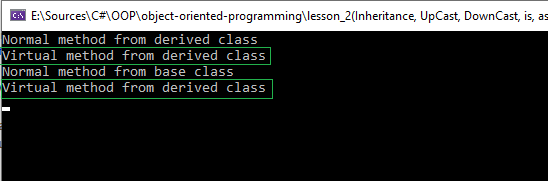
*Рис. 2.26.* Пример виртуального метода



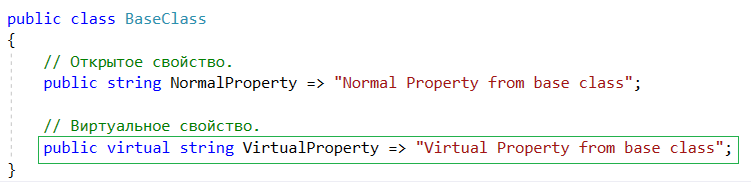
*Рис. 2.27.* Пример переопределения виртуального метода



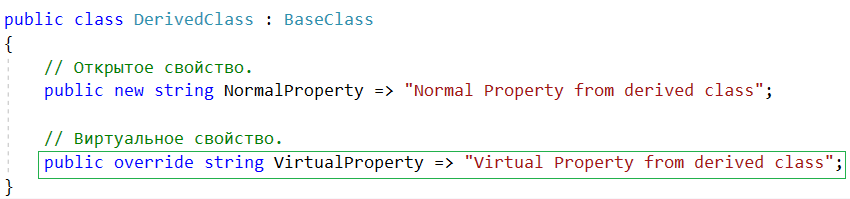
*Рис. 2.28.* Пример вызова виртуального и обычного методов   
до и после приведения к базовому типу



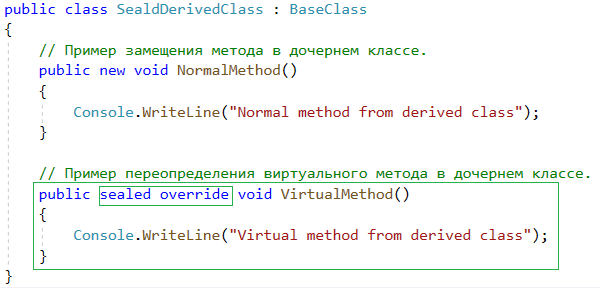
*Рис. 2.29.* Результат работы программы



*Рис. 2.30.* Пример виртуального свойства



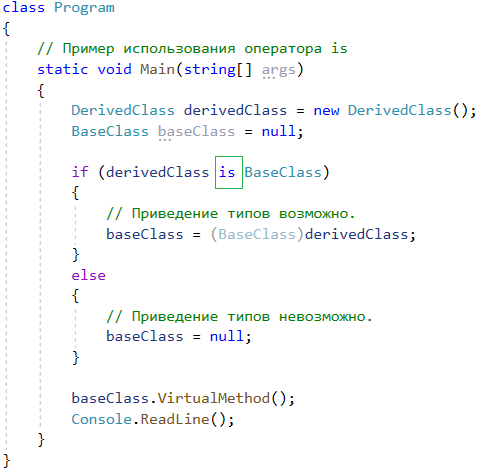
*Рис. 2.31.* Пример перегрузки   
виртуального свойства



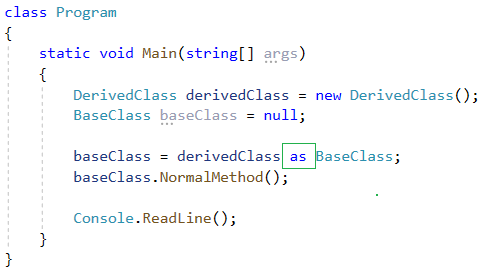
*Рис. 2.32.* Пример запрета   
дальнейшего переопределения метода

2.7. Операторы is и as

Оператор **is** служит для проверки того, можно ли привести один тип   
к другому. В случае, если приведение типов возможно, оператор **is** возвращает **true**, в противном случае **false**. На рис. 2.33 приведен пример использования оператора **is**. Оператор **as** выполняет преобразование заданного экземпляра к заданному типу, если такое преобразование возможно, в противном случае возвращает **null**. На рис. 2.34 приведен пример использования оператора **as**.



*Рис. 2.33.* Пример использования оператора is



*Рис. 2.34.* Пример использования оператора as

2.8. Композиция и агрегация

Наследование – это не единственная связь между объектами в объектно-ориентированном программирование. Связь, когда один класс содержит в себе ссылки на экземпляры других классов и память, под эти объекты выделяется в конструкторе класса, следовательно и удаление   
в памяти основного объекта приведет к удалению всех составных объектов. На рис. 2.35 приведен пример **композиции**. **Агрегация** отличается от **композиции** тем, что жизненный цикл объектов, из которых состоит класс, напрямую не зависит от него (память выделяется вне класса, содержащего в себе экземпляры других классов). На рис. 2.36 приведен пример агрегации, где память под поля не выделяется в конструкторе, как в прошлом примере, и в конструктор передаются переменные, под которые уже выделена память. Так, при удалении объекта Computer его поля \_hdd, \_monitor, \_ram останутся нетронутыми, если на них есть ссылки   
у других переменных.

|  |  |
| --- | --- |
| *Рис. 2.35.* Пример  использования композиции | X:\5. РАБОТЫ 2021\С редактированием\ЭЛЕКТРОННЫЕ\Степанов, Кабанов, Никонов, Павлюченко\Оригинал\Правки скринов\2.35.png |

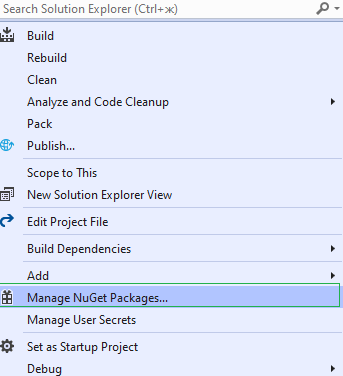
|  |  |
| --- | --- |
| *Рис. 2.36.* Пример  использования агрегации |  |

Для расширения и углубления знаний по данной теме рекомендуется изучить информационные источники [6–10, 20, 21].

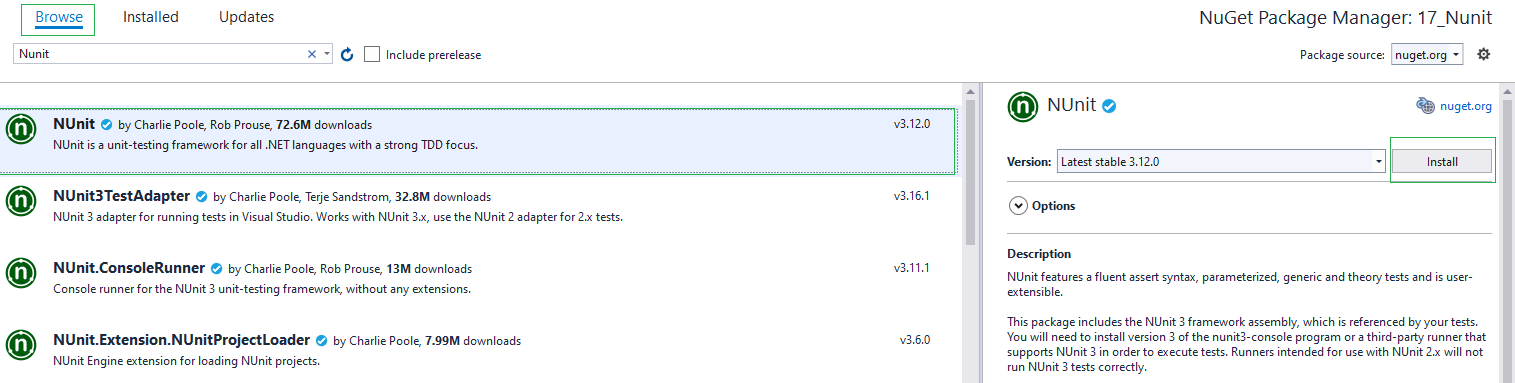
ПРАКТИЧЕСКОЕ ЗАДАНИЕ 2

## *Задача 1.* Unit-тестирование с использованием Nunit.

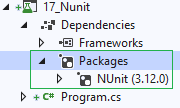
Для того чтобы работать с фреймворком **Nunit**,сперва нужно установить Nuget-пакет, содержащий его. Это можно сделать, нажав правой кнопкой мыши по проекту в Visual Studio, после чего необходимо нажать на пункт **Manage NuGet Packages** (рис. 2.37). Далее необходимо перейти на вкладку Browse (рис. 2.38), в окне поиска ввести Nunit (рис. 2.38), выбрать последнею версию и нажать кнопку Install (рис. 2.38). После успешной установки в проекте появится ссылка на NuGet-пакет (рис. 2.39). После установки **Nunit** необходимо в классы, которые будут использоваться как unit-тесты, подключить пространство имен NUnit.Framework (рис. 2.40). Далее можно приступить к написанию тестов. Для того чтобы пометить класс как класс, содержащий unit-тесты, его необходимо декорировать атрибутом **[TestFixture]** (рис. 2.40).



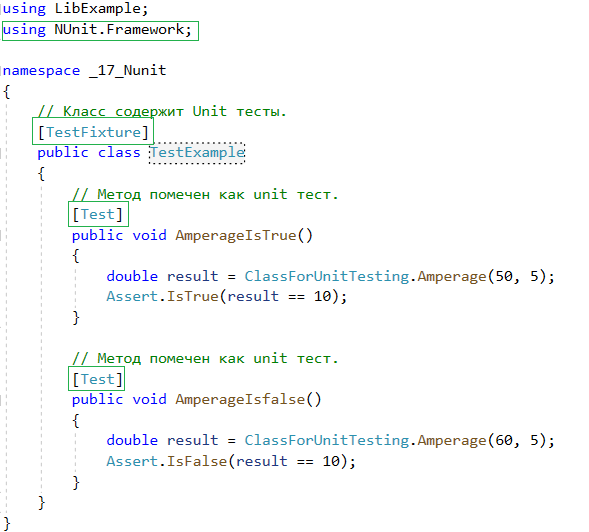
*Рис. 2.37.* Пример открытия пункта   
NuGet Package Manager



*Рис. 2.38.* Пример установки NuGet-пакета

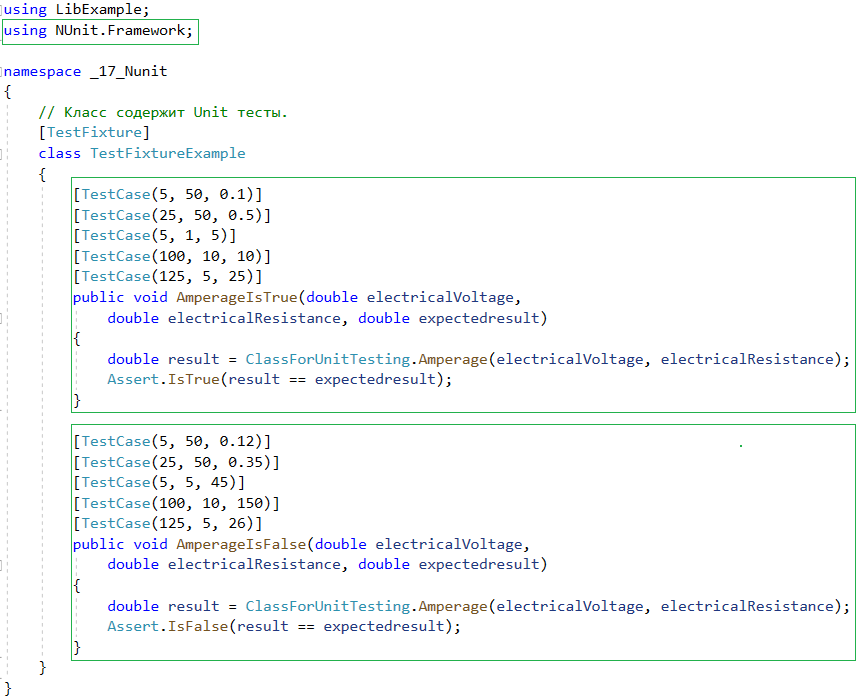


*Рис. 2.39.* Пример успешной установки   
NuGet-пакета



*Рис. 2.40.* Пример написания unit-тестов

А для того, чтобы указать, что метод содержит в себе unit-тест, его необходимо декорировать атрибутом **[Test]** (рис. 2.41). Использование атрибута **[Test]** может оказаться неудобным в том случае, если мы хотим проверить один о тот же функционал. В тестах отличаются только входные данные. Тогда всякий раз, когда нужно будет проверить разные входные данные, необходимо будет писать новый тест. Чтобы этого избежать, существует атрибут **[TestCase]**, который вызывает метод им декорированный с параметрами, указанными в атрибуте.



*Рис. 2.41.* Пример использования атрибута [TestCase]

*Задача 2*

Реализовать программу «Банкомат».

Программа позволяет:

– внести деньги на счет;

– вывести деньги со счета.

Дополнительно реализовать:

– несколько типов счетов (текущий, дебетовый, кредитный), сделать их запечатанными;

– перевод между счетами;

– правила связи между счетами;

– запрет на снятие с любого счета более чем 30 000 за сеанс;

– запрет на работу с дебетовым счетом при наличии кредитного счета с балансом более минус 20 000;

– автоматическое поступление +2000 на дебетовый счет при пополнении текущего счета более чем на 1 000 000 за операцию;

– статическое поле «Баланс», общее для всех счетов;

– написать 10 unit-тестов, проверяющих корректность работы программы.

*Задача 3*

Используя Visual Studio, создать проект по шаблону Console Application.

Требуется:

– создать класс, представляющий учебный класс ClassRoom;

– создать класс Pupil (ученик). В теле класса создать методы: Study(); Read(); Write(); Relax();

– создать три производных класса на основе базового класса Pupil: ExcelentPupil; GoodPupil; BadPupil;

– преобразовать перечисленные три производных класса запечатанными и переопределить каждый из методов в зависимости от успеваемости ученика.

Конструктор класса ClassRoom принимает аргументы типа Pupil, класс должен состоять из четырех учеников.

Необходимо предусмотреть возможность того, что пользователь может передать два или три аргумента.

* Реализовать статическое поле «Количество учеников» в классе ClassRoom.
* К классу Pupil реализовать виртуальное свойство, доступное только для чтения GetCurrentGrade, позволяющее возвращать текущую оценку ученика (случайное число от 2 до 5).
* В классах-наследниках перегрузить свойство GetCurrentGrade так, чтобы вероятность получения хорошей оценки у способных учеников была выше.
* В классе ClassRoom реализовать свойство GetRoundGrade, вычисляющее средний балл по группе.
* Написать 10 unit-тестов, проверяющих корректность работы программы.

*Задача 4*

Используя Visual Studio, создать проект по шаблону Console Application.

Требуется:

* создать класс DocumentWorker;
* в теле класса создать три метода: OpenDocument(), EditDocument(), SaveDocument(). В тело каждого из методов добавить вывод на экран соответствующих строк: "Документ открыт", "Редактирование документа доступно в версии Про", "Сохранение документа доступно в версии Про";
* создать производный класс ProDocumentWorker;
* переопределить соответствующие методы, при переопределении методов вывести следующие строки: "Документ отредактирован", "Документ сохранен в старом формате, сохранение в остальных форматах доступно в версии Эксперт";
* создать производный класс ExpertDocumentWorker от базового класса ProDocumentWorker. Переопределить соответствующий метод. При вызове данного метода необходимо вывести на экран "Документ сохранен в новом формате";
* в теле метода Main() необходимо реализовать возможность приема от пользователя номера ключа доступа pro и exp. Если пользователь   
  не вводит ключ, он может пользоваться только бесплатной версией (создается экземпляр базового класса). Если пользователь ввел номера ключа доступа pro и exp, то должен создаться экземпляр соответствующей версии класса, приведенный к базовому, – DocumentWorker;
* написать 10 unit-тестов, проверяющих корректность работы программы.

Контрольные вопросы

1. Опишите, что такое наследование, приведите примеры.

2. Расскажите про модификатор доступа private.

3. Расскажите про модификатор доступа public.

4. Расскажите про модификатор доступа protected.

5. Расскажите про модификатор доступа internal.

6. Расскажите про модификатор доступа sealed.

7. Что такое Up Cast?

8. Что такое Down Cast?

9. Что такое виртуальный метод? Что такое виртуальное свойство?

10. Чем отличается замещение от переопределения?

11. Расскажите про оператор is.

12. Расскажите про оператор as.

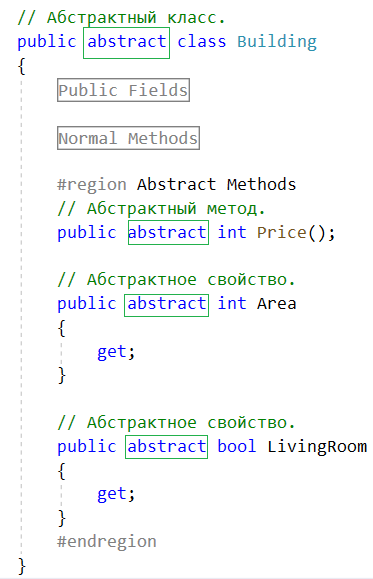
13. Расскажите про композицию и агрегацию.

14. Расскажите, что такое атрибуты NUnit.

3. АБСТРАКТНЫЕ КЛАССЫ, ИНТЕРФЕЙСЫ

3.1. Абстрактные классы

Если классы объявлены как абстрактные, то их экземпляры нельзя создавать напрямую. Абстрактные классы используются для создания абстракций, от которых можно только наследоваться. Ключевое слово **abstract** (рис. 3.1) служит для указания, что класс является абстрактным. Абстрактные классы могут содержать абстрактные методы (методы, свойства, итераторы). Абстрактные методы и свойства необходимо тоже декорировать атрибутом **abstract** (рис. 3.1). На самом деле абстрактные методы и свойства являются виртуальными, но не содержат реализацию и обязаны быть переопределенными в классе-наследнике с помощью ключевого слова **override** (рис. 3.2).

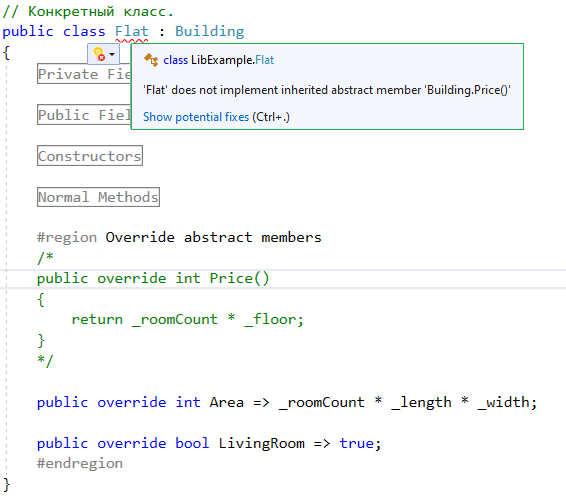


*Рис. 3.1.* Пример абстрактного класса,   
содержащего абстрактный метод и абстрактные свойства

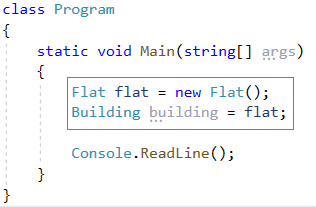


*Рис. 3.2.* Пример класса, унаследованного от абстрактного

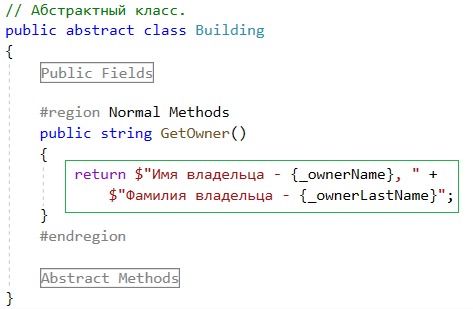
Класс-наследник, унаследованный от абстрактного класса, обязан содержать переопределение всех абстрактных методов и свойств, в противном случае будет ошибка компиляции (рис. 3.3). Создать экземпляр абстрактного класса нельзя, но можно привести класс потомок (рис. 3.4). Абстрактный класс не может быть декорирован модификаторами **static** или **sealed**. Абстрактные классы могут содержать не абстрактные методы и свойства. На рис. 3.5 приведен абстрактный класс, содержащий обычный метод. Не абстрактные методы и свойства доступны в классе-на­следнике (рис. 3.6). Абстрактные классы могут содержать конструкторы. Абстрактный класс может быть унаследован от обычного класса или абстрактного класса. И если класс, унаследованный от абстрактного класса, содержит переопределение абстрактных методов, свойств, перечислений, то его потомки не обязаны содержать переопределения методов абстрактного класса.



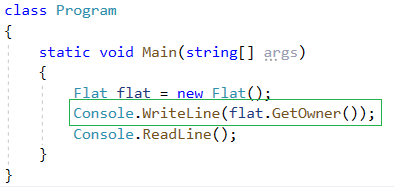
*Рис. 3.3.* Пример класса, унаследованного от абстрактного,   
не содержащего явную реализацию абстрактных методов и свойств



*Рис. 3.4.* Пример приведения потомка   
к абстрактному классу



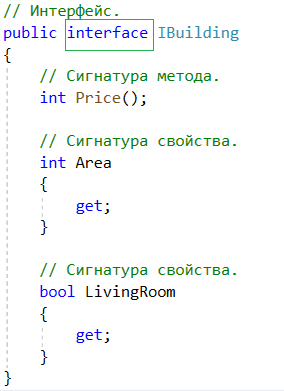
*Рис. 3.5.* Пример абстрактного класса,   
содержащего не абстрактный метод



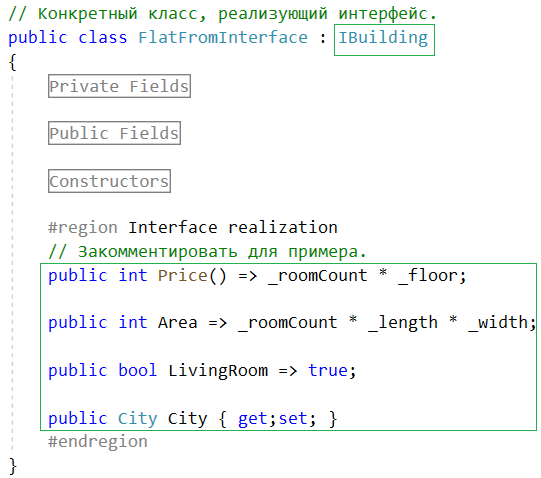
*Рис. 3.6.* Пример использования не абстрактного метода   
в классе-потомке

3.2. Интерфейсы

Интерфейс – это своего рода «контракт», который класс обязан соблюдать. Интерфейсы содержат в себе сигнатуры методов, свойств без модификаторов доступа. На рис. 3.7 приведен пример интерфейса, содержащего в себе сигнатуру метода и двух свойств. Чтобы класс реализовывал интерфейс, его нужно указать в операторе как «:» (рис. 3.8).

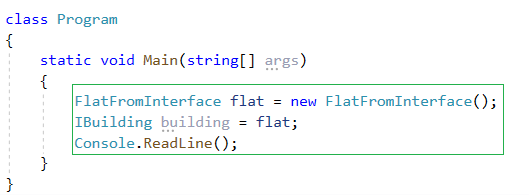


*Рис. 3.7.* Пример интерфейса

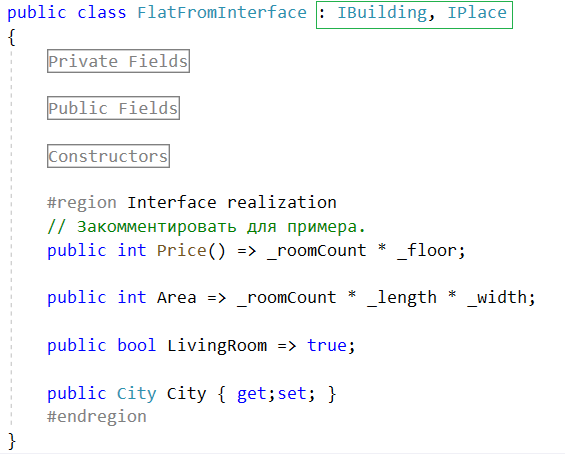


*Рис. 3.8.* Пример класса, реализующего интерфейс

У класса может быть один предок, но класс может реализовывать множество интерфейсов. Так же как и в случае с абстрактными классами, экземпляры интерфейсов нельзя создавать напрямую, но можно выполнить Up Cast класса, реализующего интерес к этому интерфейсу (рис. 3.9). На рис. 3.10 приведен пример класса, реализующего два интерфейса (класс может реализовывать произвольное количество интерфейсов).

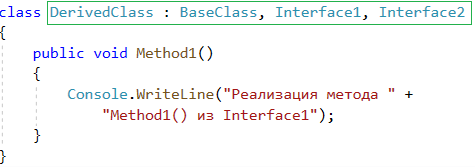


*Рис. 3.9.* Пример приведения класса к интерфейсу



*Рис. 3.10.* Пример класса, реализующего два интерфейса

На рис. 3.11 приведен пример класса, который унаследован от класса предка и реализует два интерфейса. Если базовый класс содержит методы или свойства, совпадающие с методами и свойствами интерфейса, то их можно не реализовывать в классе-потомке.



*Рис. 3.11.* Пример класса, имеющего одного предка   
и реализующего два интерфейса

Для расширения и углубления знаний по данной теме рекомендуется изучить информационные источники [4–7, 13–17, 21].

ПРАКТИЧЕСКОЕ ЗАДАНИЕ 3

*Задача 1*

Создать программу, вычисляющую общую выручку за рейс (выручка складывается из суммы оплаты проезда пассажирами), используя предметную область «Пассажирский транспорт». Обязательно наличие нескольких видов транспорта, например автобус, такси, электричка с разными характеристиками (для каких-то видов транспорта выручка вычисляется с учётом километража (такси), для каких-то – с учётом количества человек, льготных билетов и т. д.).

Написать 10 unit-тестов, проверяющих корректность работы программы.

*Задача 2*

Создать программу, позволяющую определять возможный для каждой категории клиентов размер кредита и процентную ставку, используя прикладную область «Выдача кредитов банком». Есть несколько категорий клиентов: сами сотрудники банков как привилегированная категория; обычные граждане; граждане с плохой кредитной историей; льготники (например, пенсионеры). Алгоритм расчёта размера кредита и процентной ставки отличается для всех категорий (сам алгоритм расчёта не столь важен, важно умение применить полиморфизм).

Написать 10 unit-тестов, проверяющих корректность работы программы.

Контрольные вопросы

1. Что такое абстрактный класс?

2. Что такое абстрактный метод?

3. Может ли абстрактный класс содержать не абстрактные члены?

4. Может ли абстрактный класс содержать конструкторы?

5. Может ли абстрактный класс быть декорированным атрибутом static?

6. Может ли абстрактный класс быть декорированным атрибутом sealed?

7. Опишите, что такое интерфейс.

8. Может ли интерфейс содержать не абстрактные члены?

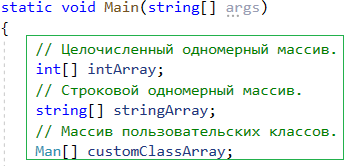
9. Могут ли методы или свойства в интерфейсе быть декорированы модификаторами доступа.

10. Сколько интерфейсов может реализовывать класс?

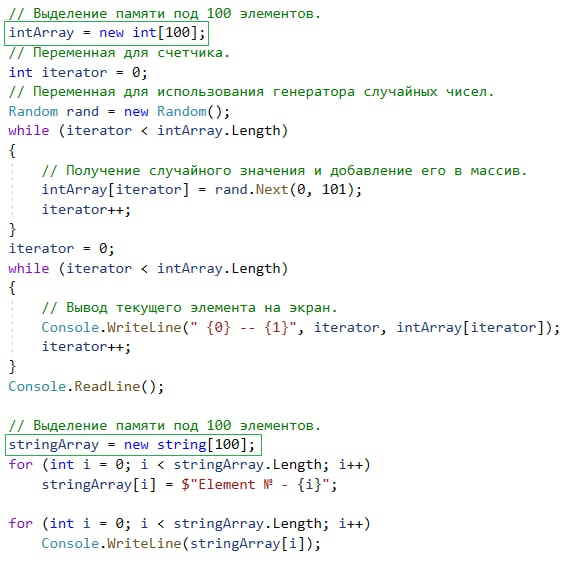
4. МАССИВЫ, ИНДЕКСАТОРЫ, КОЛЛЕКЦИИ,   
ПАРАМЕТРИЗИРОВАННЫЕ КОЛЛЕКЦИИ

4.1. Одномерные массивы

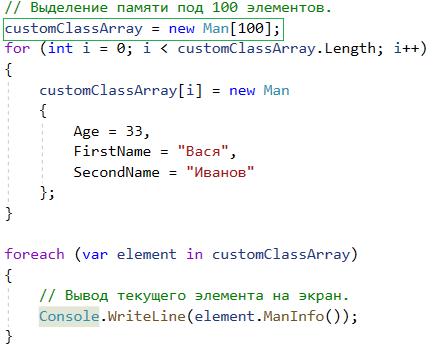
Для того чтобы объявить массив в языке C#, используется оператор «**[]**». На рис. 4.1 показаны примеры объявления массив целочисленных значений, строк и пользовательских классов. После того как массив объявлен, необходимо указать, для скольких элементов необходимо выделить память. На рис. 4.2 представлен пример выделения памяти под 100 элементов для целочисленных и строковых значений, заполнения массива значениями и вывода массивов на экран. На рис. 4.3 приведен пример выделения памяти для массива, содержащего 100 экземпляров типа Man, заполнения массива значениями и вывода его не экран. С помощью оператора «**{}**» массив можно заполнить значениями сразу после выделения памяти. На рис. 4.4 приведен пример заполнения массивов значениями после выделения памяти. Для заполнения массивов значениями можно использовать методы. На рис. 4.5 приведен пример статического метода, который генерирует экземпляр класса Man, заполняет его случайными значениями из строковых массивов, содержащих имена и фамилии, и устанавливает возраст случайным значением от 1 до 100. На рис. 4.6 показа пример использования метода **GetMan()** внутри оператора «**{}**». Также на рис. 4.6 показано, что при использовании оператора «**{}**» можно явно не указывать, под сколько элементов массива нужно выделять память, можно даже не указывать тип массива, а просто указывать **new []**.



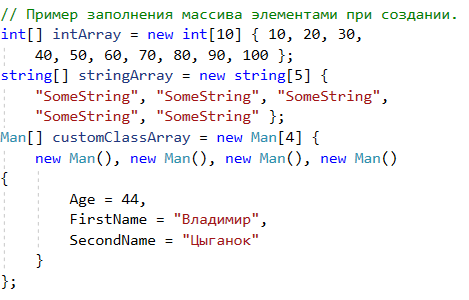
*Рис. 4.1.* Примеры объявления различный массивов



*Рис. 4.2.* Пример выделения памяти и заполнения строкового   
и целочисленного массивов



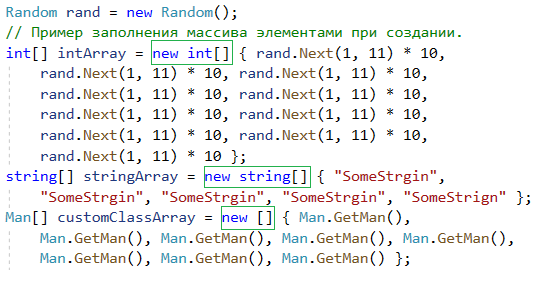
*Рис. 4.3.* Пример выделения памяти и заполнения массива,   
содержащего экземпляры пользовательских классов



*Рис. 4.4.* Пример заполнения массива элементами при его создании



*Рис. 4.5.* Пример метода, возвращающего экземпляр класса

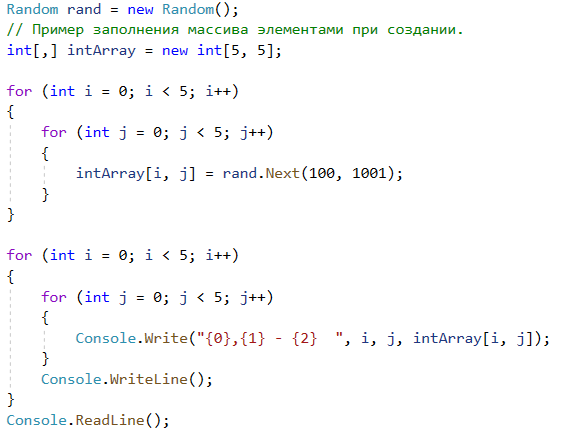


*Рис. 4.6.* Пример использования методов в операторе {}

## 

4.2. Многомерные массивы

Язык C# позволяет создавать многомерные массивы: **[,]** – двумерный,   
**[, ,]** – трехмерный и т. д. На рис. 4.7 приведен пример объявления, выделения памяти, заполнения случайными значениями целочисленного массива.

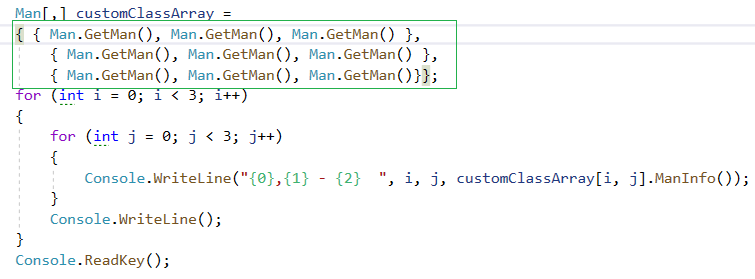
****

*Рис. 4.7.* Пример целочисленного двумерного массива

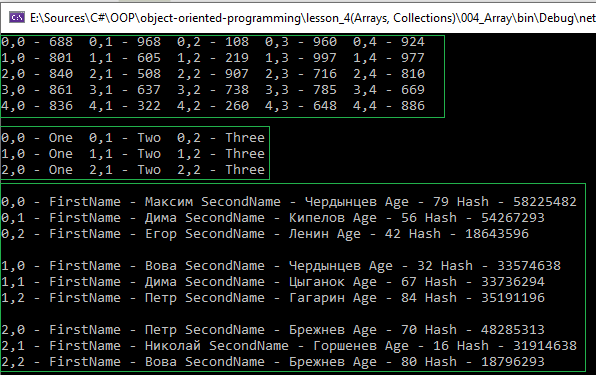
На рис. 4.8 приведен пример строкового двумерного массива, а также пример использования оператора **{}** для двумерного массива. Так же, как   
и с одномерным массивом, при использовании оператора **{}** можно явно не указывать размерность массива.На рис. 4.9 приведен пример дву­мерного массива, содержащего экземпляры пользовательского класса.   
На рис. 4.9 показано, что, как и для одномерных массивов, можно использовать методы для инициализации массивов. На рис. 4.9 также показано, что при использовании оператора **{}** можно вообще не указывать **new []**. На рис. 4.10 показан результат выполнения программы.

****

*Рис. 4.8.* Пример строкового двумерного массива

****

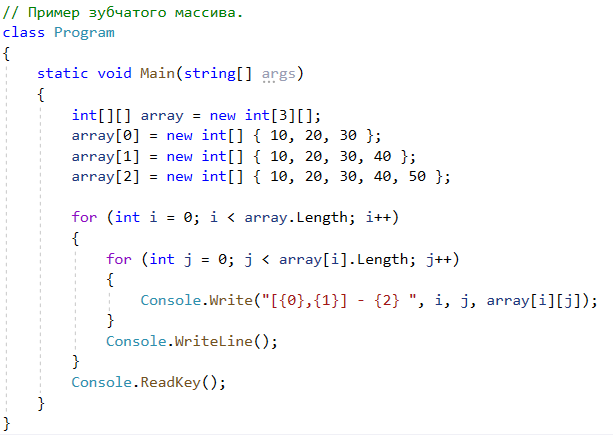
*Рис. 4.9.* Пример двумерного массива,   
состоящего из экземпляров пользовательского типа



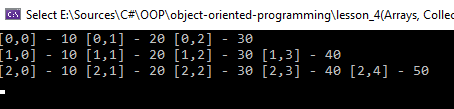
*Рис. 4.10.* Пример работы программы

4.3. Зубчатые массивы

Язык C# позволяет создавать зубчатые массивы произвольной размерности. На рис. 4.11 приведен пример создания целочисленного зубчатого массива. В зубчатом массиве каждая строка массива может иметь различную длину. На рис. 4.12 приведен пример работы программы.

****

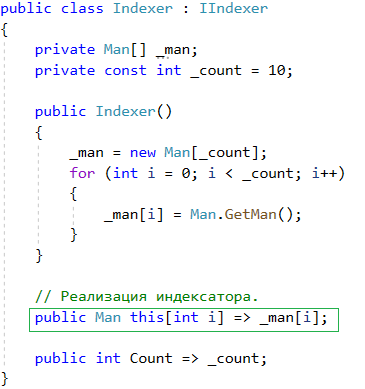
*Рис. 4.11.* Пример зубчатого массива



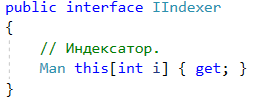
*Рис. 4.12.* Пример работы программы

4.4. Индексаторы

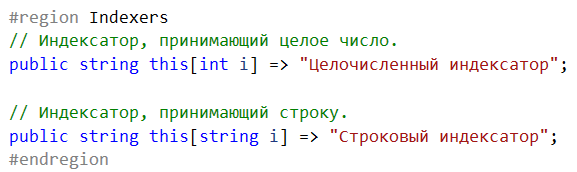
Индексаторы позволяют обращаться к классу как к массиву [12].   
На рис. 4.13 приведен пример индексатора. Индексаторы похожи на свойства, только для описания индексатора используется ключевое слово **this [тип индекса]**.Интерфейсы могут содержать сигнатуры индексаторов (рис. 4.14) [12]. В классе может быть сколь угодно много индексаторов, различающихся по сигнатуре (рис. 4.15).



*Рис. 4.13.* Пример индексатора



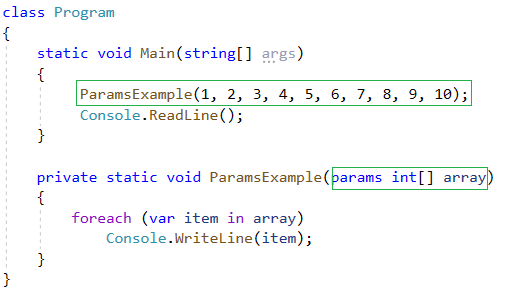
*Рис. 4.14.* Пример интерфейса,   
содержащего индексатор



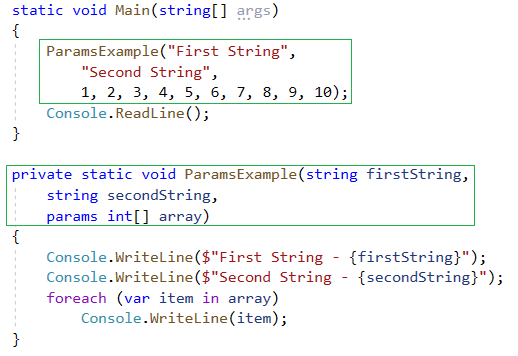
*Рис. 4.15.* Пример класса, содержащего несколько индексаторов

4.5. Использование ключевого слова params

Используя ключевое слово **params**, можно передавать в метод неограниченное число параметров. На рис. 4.16 приведён пример использования ключевого слова **params**;переданные параметры в функцию преобразуются в массив. Если метод принимает несколько параметров, то атрибутом **params** может быть декорирован только последний параметр (рис. 4.17).



*Рис. 4.16.* Пример использования ключевого слова params



*Рис. 4.17.* Пример использования ключевого слова params в методе,   
принимающем несколько параметров

## 

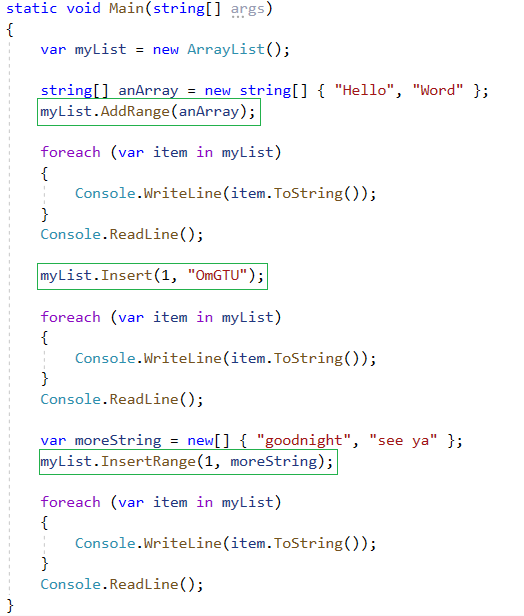
4.6. Коллекции

4.6.1. ArrayList

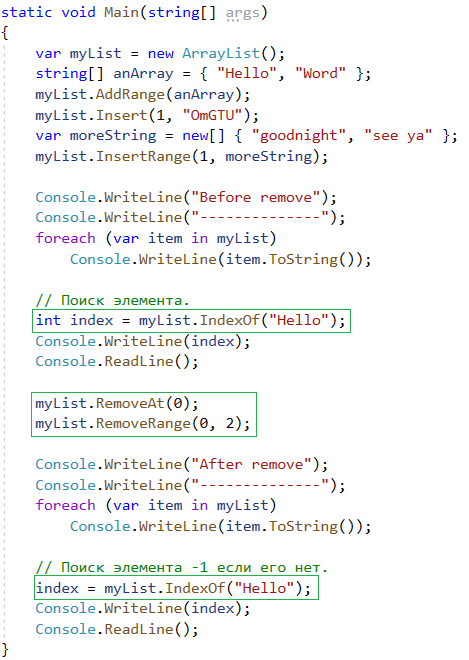
**ArrayList** – это динамическая коллекцияиз пространства имен System.Collections, которая содержит объекты типа **object**,который в языке **C#** является родителем для все типов, и, следовательно, экземпляр любого типа может быть помещен в коллекцию. На рис. 4.18 приведен пример того, что при использовании метода **Add()** вколлекцию **ArrayList** может быть помещен экземпляр любого класса.На рис. 4.19 представлен пример добавления сразу нескольких значений в коллекцию при использовании метода **AddRange()**.Также на рис. 4.19 приведен пример использования методов **Insert()** и **InsertRange()**,добавляющих одно и или несколько значений по указанному индексу коллекции. Для поиска первого элемента в массиве или проверке того, что коллекция **ArrayList** не содержит искомый элемент, используется метод **IndexOf()**, который возвращает или первый индекс найденного элемента в массиве, или –1, если элемент не найден в массиве (рис. 4.20).



*Рис. 4.18.* Пример использования коллекции ArrayList

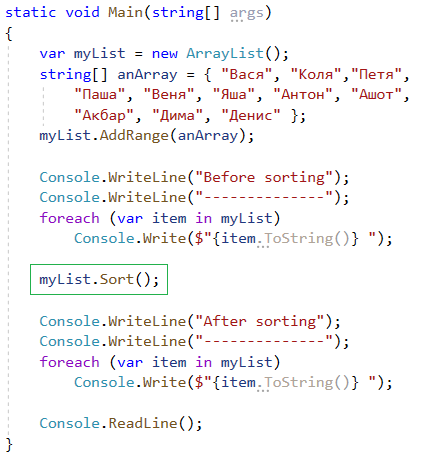


*Рис. 4.19.* Пример использования коллекции ArrayList

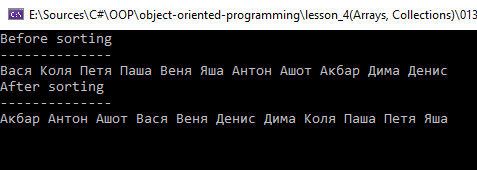


*Рис. 4.20.* Пример удаления значений из коллекций

Метод **RemoveAt()** удаляет элемент из массива по указанному индексу (рис. 4.20). Метод удаления **RemoveRange()** принимает два параметра: первый параметр – индекс элемента коллекции, с которого нужно начинать удаление, второй – сколько элементов необходимо удалить из коллекции (рис. 4.20). Коллекция **ArrayList** содержит метод **Sort()**, который сортирует коллекцию от меньшего значения до большего (рис. 4.21).   
На рис. 4.22 приведен пример сортировки.



*Рис. 4.21.* Пример использования метода Sort



*Рис. 4.22.* Результат работы программы

4.6.2. Queue (Очередь)

**Queue** – коллекция, работающая по принципу «первый вошел – первый вышел». На рис. 4.23 приведен пример работы с коллекцией типа **Queue**,метод **Enqueue()** добавляет элемент в очередь. Метод **Peek()**(рис. 4.23)возвращает элемент, который указан вершиной коллекции.   
Метод **Dequeue()** (рис. 4.23) возвращает элемент, который являетсявершиной коллекции и удаляет его из коллекции. Свойство **Count** возвращает число элементов в коллекции (рис. 4.23).

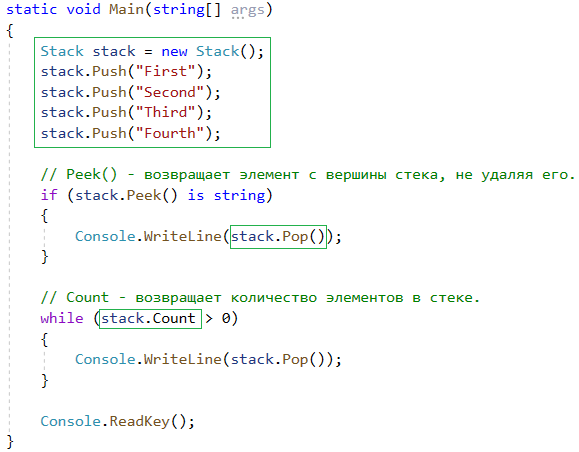


*Рис. 4.23.* Пример работы с коллекцией Queue

## 

4.6.3. Stack (Стек)

**Stack** – коллекция, работающая по принципу «первый зашел – по­с­ледний вышел». На рис. 4.24 приведен пример работы с коллекцией типа **Stack**,метод **Push()** добавляет элемент в очередь. Метод **Peek()** (рис. 4.24)возвращает элемент, который указан вершиной коллекции. Метод **Pop()** (рис. 4.24) возвращает элемент, который являетсявершиной коллекции   
и удаляет его из коллекции. Свойство **Count** возвращает число элементов в коллекции (рис. 4.24).



*Рис. 4.24.* Пример работы с коллекцией Stack

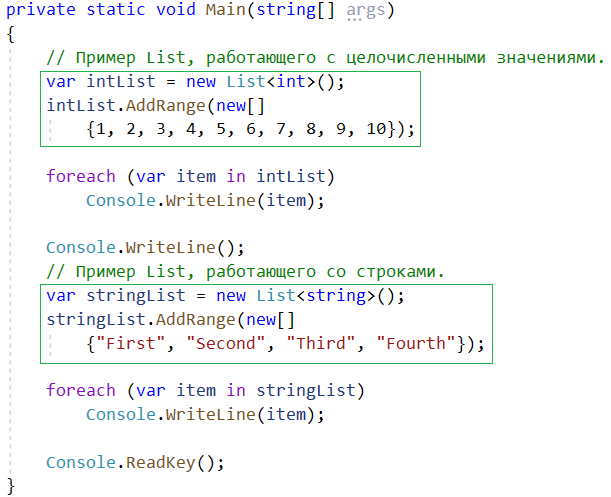
4.7. Параметризованные коллекции

4.7.1. Generic List

Все коллекции, рассмотренные ранее, содержали в себе объекты типа **object**.Класс **object** является базовым типом данных для всех типов   
в языке C#, и экземпляр любого другого типа данных можно привести   
к типу **object**. Поэтому в рассмотренных ранее примерах коллекции могли содержать экзе­м­пляры любых классов. Однако такой подход содержит   
в себе две проблемы:

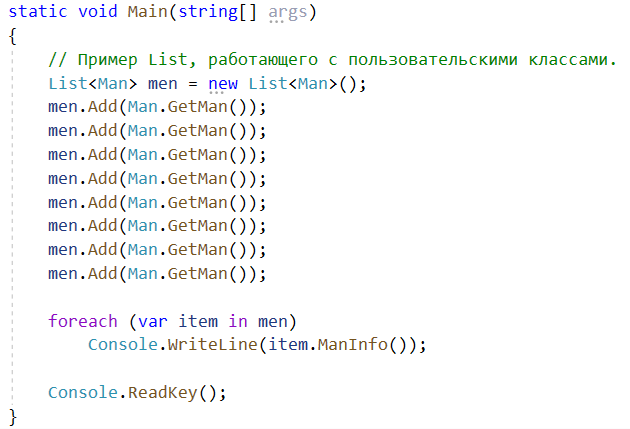
1. нужно точно знать, к какому типу привести значение, доставаемое из коллекции, в противном случае возникнет исключение;
2. проблема **Boxing** (Упаковка) и **Unboxing** (Распаковка), связанная   
   с тем, что Object – ссылочный тип данных, означающий что в стеке потока хранится только адрес на ссылку в управляемой куче, где содержится значение. Проблема возникает, когда в коллекцию помещается значимый тип данных (структура и перечисление), при помещении копируется значение из стека потока в управляемую кучу, и наоборот: при извлечении значение каждый раз копируется из управляемой кучи в стек потока.

Чтобы избежать этих проблем, Microsoft добавили новое пространство имен **System.Collections.Generic**, содержащее шаблонные коллекции. Каждая такая коллекция может работать только с определенным типом данных, и, если пользователь попытается добавить в нее значение другого типа, произойдет ошибка на этапе компиляции. На рис. 4.25 приведен пример работы с коллекцией типа **List**.На рис. 4.25 созданы два экземпляра коллекции **List**:один для работы с целочисленными значениями, другой для работы со строками.



*Рис. 4.25.* Пример работы с коллекцией List

На рис. 4.26 приведен пример коллекции **List**, cодержащей в себе экземпляры пользовательского класса Man. Методы для добавления, уда­ления и поиска элементов идентичны коллекции **ArrayList**,рассмотренной ранее.



*Рис. 4.26.* Пример работы с коллекцией List,   
содержащей экземпляры класса Man

4.7.2. Generic Dictionary

Еще одна коллекция – это **Dictionary**,которая содержит в себе пары «ключ-значение». На рис. 4.27 представлен пример коллекции **Dictionary**,в котором в качестве ключа используется целочисленное значение, а в качестве значения используется строковое значение. На рис. 4.27 представлены примеры получения элементов и ключей по индексу и в цикле.   
На рис. 4.28 представлен пример коллекции **Dictionary**, в которой в качестве ключа используется целочисленные значения, а в качестве значения – пользовательский класс Man.

Для расширения и углубления знаний по данной теме рекомендуется изучить информационные источники [7–21].

|  |  |
| --- | --- |
|  | *Рис. 4.27.* Пример работы  с коллекцией Dictionary |
|  | *Рис. 4.28.* Пример работы  с коллекцией Dictionary,  содержащей экземпляры  класса Man |

ПРАКТИЧЕСКОЕ ЗАДАНИЕ 4

*Задача 1*

1. Создать массив, содержащий Фамилии всех одногруппников.
2. Создать массив, содержащий Имена всех одногруппников.
3. Объединить массивы в один массив так, чтобы по нечетным индексам стояли фамилии, по четным индексам – имена.

*Задача 2*

Создать массив, содержащий Фамилии всех одногруппников.

Отсортировать его пузырьком:

## <https://en.wikipedia.org/wiki/Bubble_sort>.

*Задача 3*

Смоделировать механизм очереди на отчисление в институте, используя класс Queue. Нужно написать класс Студент, содержащий статический метод GenerateStudent, который создает рандомного студента. Для этого   
в классе необходимо создать массив имен, фамилий и отчеств (можно использовать данные одногруппников), из которого выбирается одно случайное значение по следующим предметам:

* Программирование;
* Философия;
* Сети;
* Методы оптимизации.

Реализовать интерфейс, содержащий сигнатуры методов:

* string GetStudentInfo();
* bool GetDecision().

Продумать логику, когда, в зависимости от каких оценок и по каким предметам студента нужно отчислять.

Написать пять unit-тестов, проверяющих метод GetDecision().

*Задача 4*

Создать англо-русский словарь (new Dictionary< string, string>();),   
в котором в качестве ключа выступают английские слова, а в качестве значения – русские.

*Задача 5*

Переписать третью задачу:

* Заменить Queue на массив;
* Написать класс-обертку, реализовать в нем индексатор.

*Задача 6*

1. Создать массив, содержащий Фамилии всех одногруппников по четным индексам и имена по нечетным.
2. Разбить массив на два массива так, чтобы в один массив попали фамилии одногруппников, в другой – имена.

*Задача 7*

Создать массив, содержащий Имена всех одногруппников.

Отсортировать его сортировкой вставками:

<https://en.wikipedia.org/wiki/Insertion_sort>

*Задача 8*

Смоделировать механизм мыслей в голове, используя класс Stack.

Нужно реализовать класс Think, содержащий статический метод Gene­rate Think, который создает случайную мысль в голове человека. Для этого необходимо реализовать Enum TypeThink, в котором содержатся типы мыслей: Мысли об учебе; Мысли о Еде; Мысли о компьютерных играх.

Для каждого значения перечисления создать строковой массив, содержащий контурные мысли.

Например, для значения Мысли об Еде:

* Не пойти ли мне поесть;
* Хочу есть;
* Хочу в KFC;
* Опять потолстел.

Реализовать интерфейс, содержащий сигнатуры методов:

1) string GetThinkInfo();

2) bool GetDecision().

Продумать логику: когда, в зависимости от каких значений типа мыслей считается, что мысль хорошая или плохая.

Написать пять unit-тестов, проверяющих метод GetDecision().

*Задача 9*

Создать русско-английский словарь (new Dictionary< string, string>();), в котором в качестве ключа выступают русские слова, а в качестве значения – английские.

*Задача 10*

Переписать третье задание:

* Заменить Stack. на массив;
* Написать класс-обертку, реализовать в нем индексатор.

Контрольные вопросы

1. Что такое массив?

2. Что такое одномерный массив?

3. Что такое многомерный массив?

4. Что такое индексатор?

5. Сколько индексаторов может быть у класса?

6. Приведите пример использования ключевого слова params в языке C#.

7. Расскажите про коллекцию ArrayList.

8. Расскажите про коллекцию Queue.

9. Расскажите про коллекцию Stack.

10 Недостатки коллекции, содержащей элементы типа object.

11. Расскажите про коллекцию List.

12. Расскажите про коллекцию Dictionary.

5. РАБОТА С ФАЙЛОВОЙ СИСТЕМОЙ

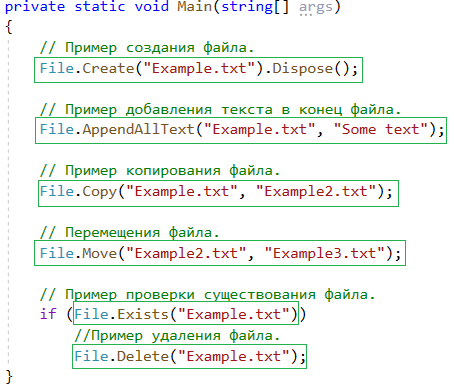
5.1. Получение информации о логических дисках   
компьютера

Для работы с файловой системой компьютера необходимо включить пространство имен **System.IO**. В неместь класс **DriveInfo**, который содержит методы для получения информации о логических дисках на компьютере, а также статический метод **GetDrives()**,возвращающий массив типа **DriveInfo**, где каждый элемент содержит информацию об одном логическом диске. Экземпляр класса **DriveInfo** содержит свойства, дающие информацию о диске. Например, **Name** – метка диска, **DriveFormat** – формат жесткого диска (FAT, NTFS), **TotalSize** – размер диска, **TotalFreeSpace** – объём незанятого места. Также экземпляр класса **DriveInfo** содержит свойство **RootDirectory**, которое возвращает корневую директорию диска, из которой можно получить список дочерних файлов и директорий (рис. 5.1).

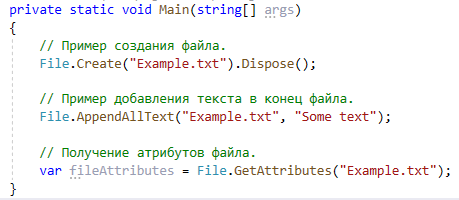
|  |  |
| --- | --- |
| X:\5. РАБОТЫ 2021\С редактированием\ЭЛЕКТРОННЫЕ\Степанов, Кабанов, Никонов, Павлюченко\Оригинал\Правки скринов\5.1.png | *Рис. 5.1.* Пример  получения информации  о логических дисках  компьютера |

5.2. File

Статический класс **File** содержит в себе статические методы для работы с файлами. Статический метод **Create()** создает файл и возвращает поток, связанный с этим файлом. Для того чтобы закрыть поток, можно воспользоваться методом **Dispose()** (рис. 5.2). Статический метод **AppendAllText()** добавляет текст в конец файла.Статический метод **Copy()** копирует файл по указанному пути. Статический метод **Move()** перемещает файл по указанному пути.



*Рис. 5.2.* Пример работы со статическим классом File



*Рис. 5.3.* Пример работы со статическим классом File

## Статический метод **Exists()** проверяет существование файла, возвращает **true**, если файл существует. Статический метод **Delete()** удаляет указанный файл (рис. 5.2). Статический класс File содержит статический метод **GetAttributes()**, возвращающий список атрибута файла (рис. 5.3).

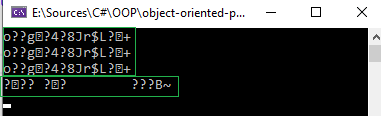
5.3. Получение MD5-хеша файла

Для того чтобы однозначно идентифицировать файл, можно использовать его **MD5-хеш**, который не меняется при переименовании файла,   
а изменяется только при изменении самого файла. На рис. 5.4 приведен пример создания и копирования файла с разными именами.



*Рис. 5.4.* Пример вычисления MD5-хеша

Так как содержание файлов одинаковое, хеш-значения этих файлов будут совпадать. На рис. 5.5 представлен результат работы программы. Первые три значения – это значения **MD5**-файлов, имеющих одинаковое содержимое.

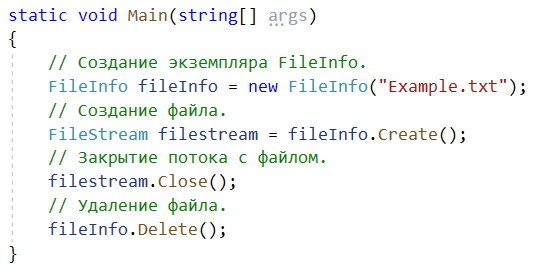


*Рис. 5.5.* Пример работы программы

## 

5.4. FileInfo

Класс **FileInfo**, как и класс **File**, служит для работы с файлами. Отличие в том, что File – это статический класс, для работы **FileInfo** необходимо создавать экземпляры классов. На рис. 5.6 приведен пример создания экземпляра класса, связанного с файлом Example.txt, пример создания потока, связанного с файлом с помощью метода **Create()**, закрытия потока   
с помощью метода **Dispose()**,а такжепример удаления файла с помощью оператора **Delete()**.



*Рис. 5.6.* Пример работы с классом FileInfo

Класс **FileInfo** содержит в себенесколько сигнатур метода Open. Одна из сигнатур метода Open принимает в себе перечисление типа **FileMode**,которое содержит следующие значения (рис. 5.7):

* **Append** – открывает файл и устанавливает курсор в конец файла.
* **Create** – создает новый файл; если он уже существует, удаляется   
  и создается новый.
* **CreateNew** – создает новый файл; если он уже существует, возбуждается исключение.
* **Open** – указывает на то, что файл должен быть открыт; если файла не существует, возбуждается исключение.
* **OpenOrCreate** – файл должен быть открыт; если его не существует, должен быть создан новый файл.

Еще один вариант метода **Open()** принимает в себя два параметра: перечисление **FileMode** и перечисление **FileAccess**. Перечисление **FileAccess** указывает, с какими правами открывается поток, связанный с файлом. Перечисление **FileAccess** содержит следующие значения:

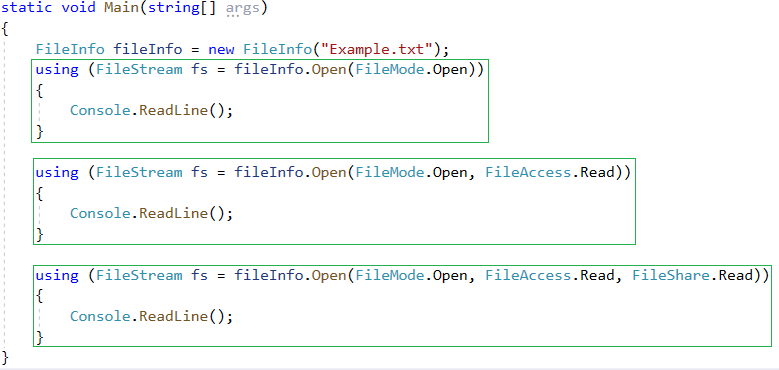
* **Read** – открытие файла на чтение.
* **ReadWrite** – открытие файла на чтение и запись.
* **Write** – открытие файла на запись.

Еще один вариант метода **Open()** принимает в себя три параметра: перечисление **FileMode**, перечисление **FileAccess** и перечисление **FileShare**. Перечисление **FileShare** содержит следующие значения:

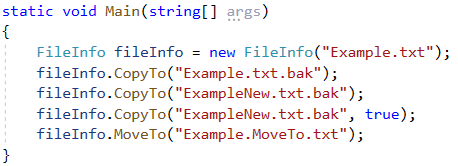
* **Delete** – разрывает удаление файлов из других потоков или приложений.
* **None** – запрещает общий доступ к файлу.
* **Read** – разрешает открытие файла для чтения из другого потока или приложения.
* **ReadWrite** – разрешает открытие файла на чтение и запись из другого потока или приложения.
* **Write** – разрешает открытие файла на запись из другого потока или приложения.

На рис. 5.8 показаны примеры использования методов CopyTo()   
и MoveTo(). Метод CopyTo() имеет две сигнатуры:

* принимает один параметр – путь, куда нужно скопировать.
* принимает два параметра – путь, куда нужно скопировать, и булевой параметр, указывающий, что нужно делать, если файл уже существует (true – перезапись файла).



*Рис. 5.7.* Пример различных методов Open класса FileInfo



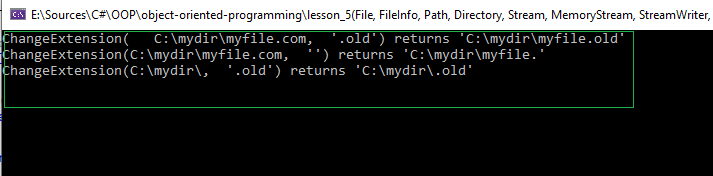
*Рис. 5.8.* Пример копирования и перемещения файлов   
с помощью класса FileInfo

5.5. Path

Класс Path содержит в себе множество методов для работы и путей   
к файлам расширения и т. д. На рис. 5.9 показан пример использования метода **ChangeExtension()**, который заменяет расширение файла во входной строке на указанное. Данный метод только возвратит строку и не проводит никаких манипуляций с файловой системой. На рис. 5.10 показаны примеры метода **ChangeExtension()** для разных строк. На рис. 5.11 показаны примеры использования метода **GetFileName()**, который из строки возвращает только имя файла с расширением. Метод **GetFileNameWithoutExtension()** возвращает из переданной в него строки имя файла без расширения (рис. 5.11). Метод **GetExtension()** возвращает из переданной строки расширение файла (рис. 5.11). Метод **GetDirectoryName()** из переданной строки возвращает имя директории, в которой расположен файл (рис. 5.11). Метод **GetPathRoot()** возвращает имя корневого каталога, в котором расположен файл (рис. 5.12). На рис. 5.12 приведен результат работы программы.   
На рис. 5.13 представлен пример использования метода **GetFullPath()**,который возвращает полный путь до файла.



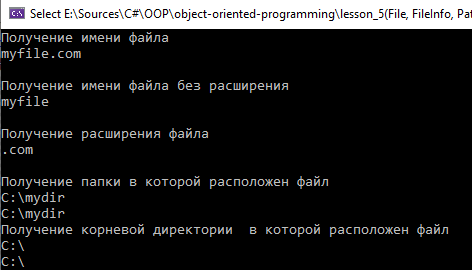
*Рис. 5.9.* Пример использования метода ChangeExtension класса Path



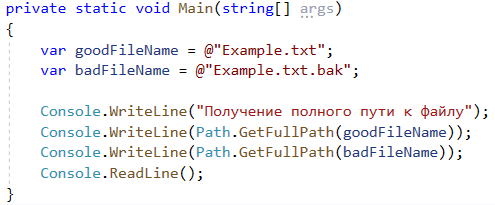
*Рис. 5.10.* Пример работы программы



*Рис. 5.11.* Пример работы с методами класса Path

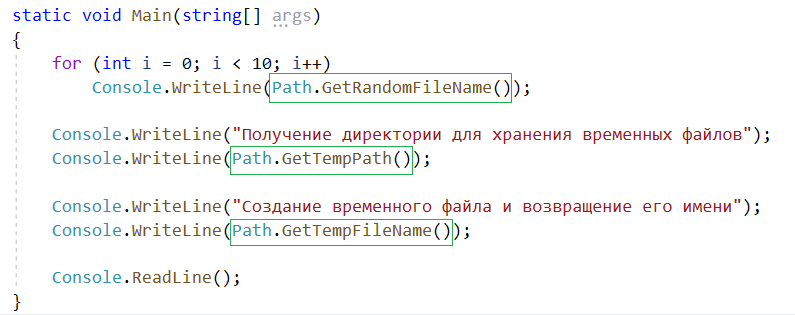


*Рис. 5.12.* Пример работы программы

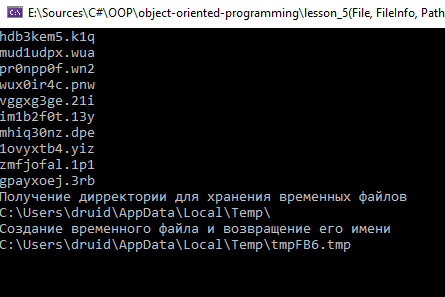


*Рис. 5.13.* Пример получения полного пути к файлу

На рис. 5.14 приведен пример работы метода **GetRandomFileName()**,который возвращает строку, содержащую случайное имя файла. Также на   
рис. 5.14 приведен пример метода **GetTempPath()**, которыйвозвращает путь до папки, содержащей временные файлы, и пример метода **GetTemp FileName()**, который создает файл в директории для временных файлов   
и возвращает его имя. Пример работы программы представлен на рис. 5.15.



*Рис. 5.14.* Пример работы с методами класса Path

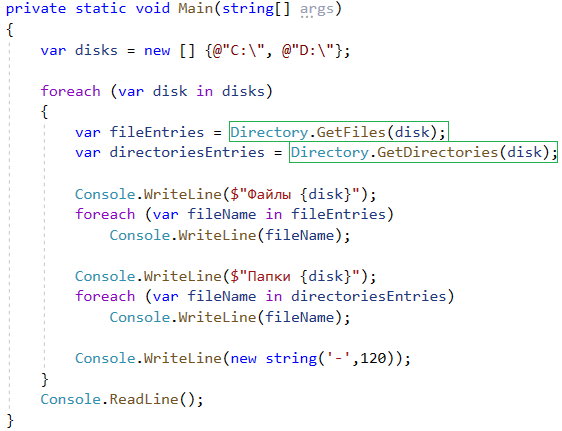


*Рис. 5.15.* Пример работы программы

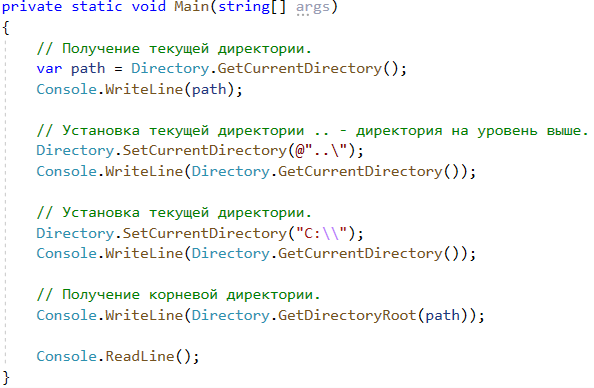
## 

5.6. Directory

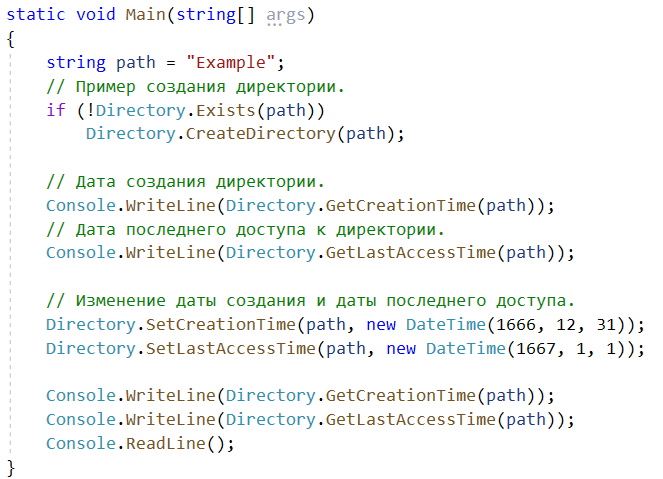
Статический класс **Directory** содержит в себе методы по работе с директориями. На рис. 5.16 показан пример использования метода **GetFiles()**, который возвращает строковой массив, состоящий из имен файлов в директории. Также на рис. 5.16 приведен пример использования метода **GetDirectories()**,который возвращает список директорий   
из директории. На рис. 5.17 представлен пример использования ме­тода **GetCurrentDirectory()**, который возвращает текущую директорию  (по умолчанию та директория, где находится исполняемый файл).   
На рис. 5.17 показан пример того, как можно изменить текущую директорию, используя метод **SetCurrentDirectory()**,также показан метод **GetDirectoryRoot()**, который из переданной в него строки возвращает конечную директорию. На рис. 5.18 приведен пример использования оператора **Exists()**,проверяющего, существует ли указанная директория. Также приведен пример использования метода **CreateDirectory()**, создающего указанную директорию. Метод **GetCreationTime()** возвращает дату создания директории (рис. 5.18), а метод **GetLastAccessTime()** возвращает дату последнего доступа к директории (рис. 5.18). Методы **SetCreationTime()**   
и **SetLastAccessTime()** служат для изменения дата создания и последнего доступа директории. На рис. 5.19 приведен пример перемещения директории, содержащей файлы, с помощью метода **Move()**.

****

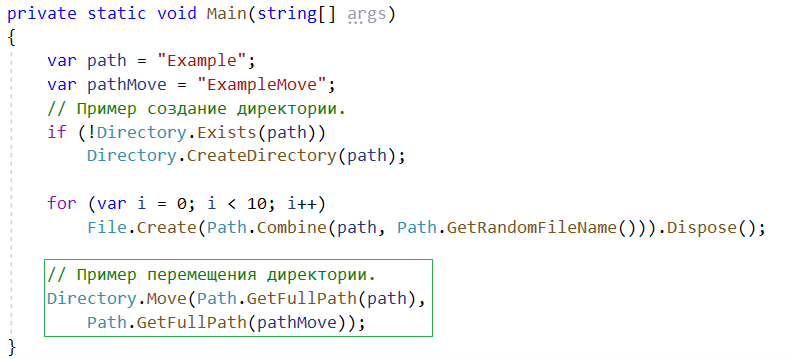
*Рис. 5.16.* Пример получения файлов и директорий



*Рис. 5.17.* Пример чтения и установки   
значения текущей директории



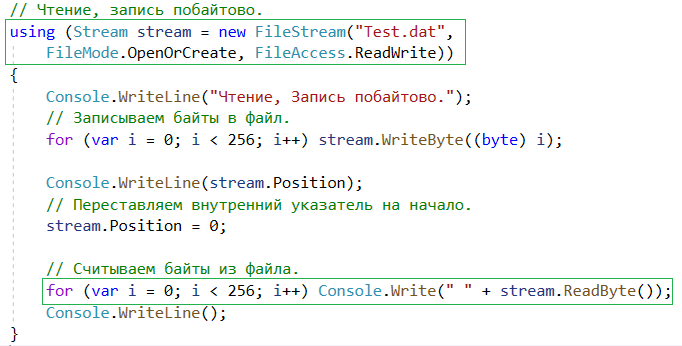
*Рис. 5.18.* Пример чтения и установки   
даты создания и последнего доступа



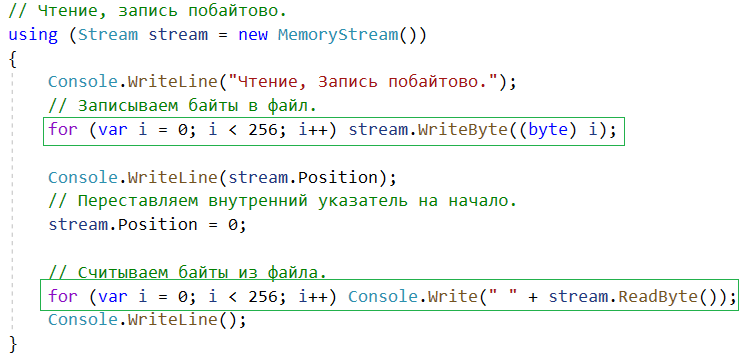
*Рис. 5.19.* Пример перемещения директории

5.7. Stream

Класс **Stream** является базовым классом для всех потоков, например для **FileStream** (рис. 5.20) или **MemoryStream** (рис. 5.21). Класс **Stream** работает с байтами, а не с символами. На рис. 5.20 и 5.21 приведены примеры метода **WriteByte()**, который записывает один байт в поток и то, что с ним связано, а также пример установки в начало потока и пример использования **ReadByte()**, который считывает один байт из потока.

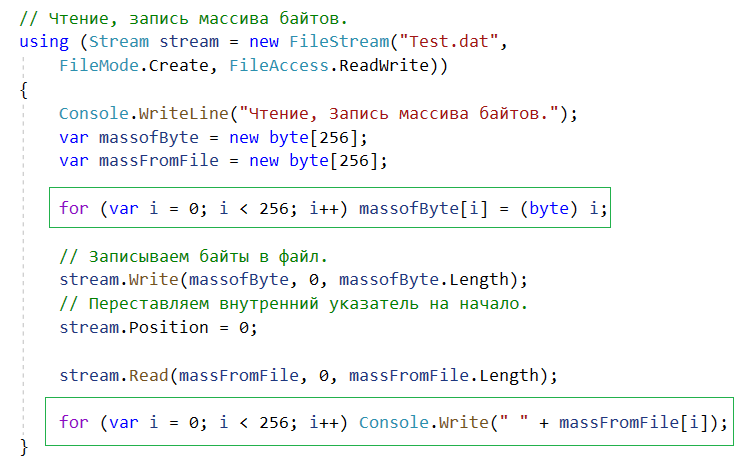


*Рис. 5.20.* Пример приведения класса FileStream к классу Stream,   
побайтовое чтение и запись

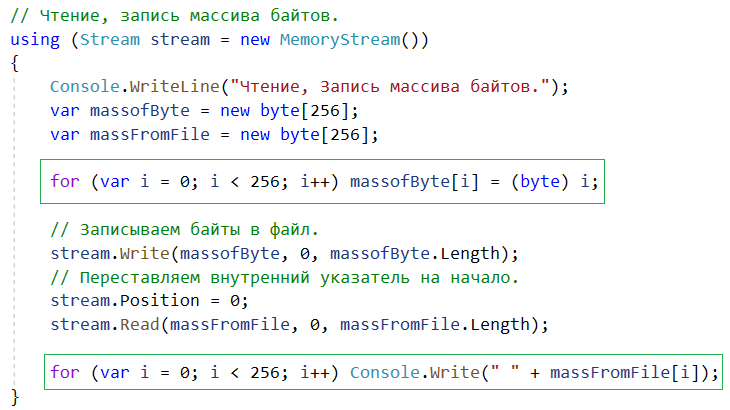


*Рис. 5.21.* Пример приведения класса MemoryStream к классу Stream,   
побайтовое чтение и запись

Класс **Stream** содержит в себе методы **Write()** и **Read()** (рис. 5.22–5.23) для чтения и записи массива байтов.

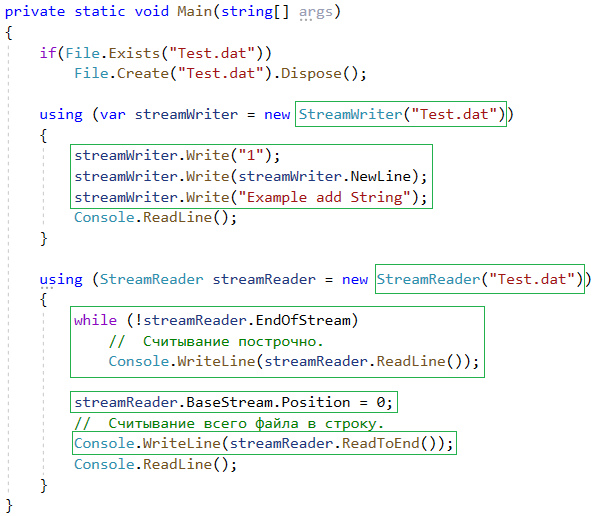


*Рис. 5.22.* Пример приведения класса MemoryStream к классу Stream,   
чтение и запись массива байтов



*Рис. 5.23.* Пример приведения класса MemoryStream к классу Stream,   
чтение и запись массива байтов

5.8. StreamWriter

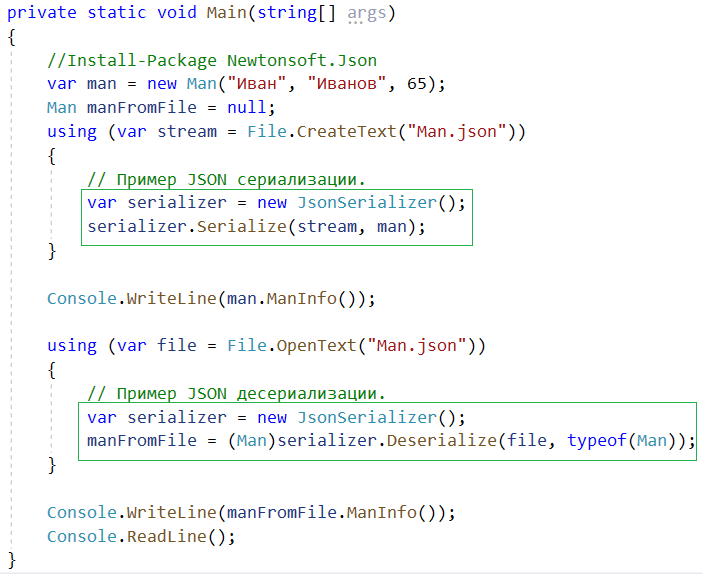


*Рис. 5.24.* Пример использования классов   
StreamWriter и StreamReader

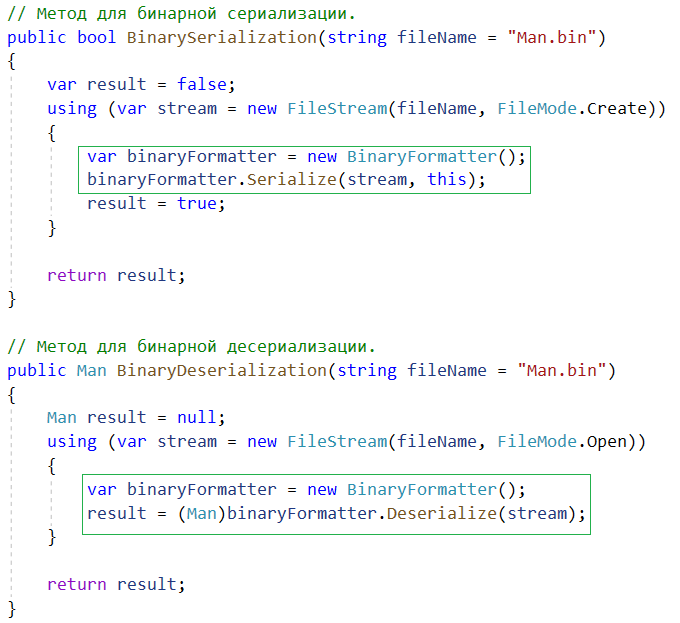
**StreamWriter** –класс, работающий не с байтами, а со строками, причем только на запись (класс **Stream** умеет работать только с байтами).Конструктор **StreamWriter** принимает либо строку, либо экземпляр класса **Stream**.На рис. 5.24 приведен пример работы с **StreamWriter**.Класс **StreamReader** может считывать символы и строки из потока (рис. 5.24).

5.9. Бинарная сериализация (BinaryFormatter)

Сериализация – это процесс сохранения состояния объекта [13]. На рис. 5.25 представлен пример сериализации и десериализации объекта типа Man. Для того чтобы класс поддерживал сериализацию, его необходимо декорировать атрибутом **[Serializable]** (рис. 5.26). На рис. 5.26 показан пример использования сериализации и десериализации в методе с помощью ключевого слова **this**.



*Рис. 5.25.* Пример двоичной сериализации

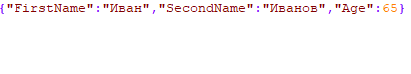


*Рис. 5.26.* Пример двоичной сериализации   
с помощью ключевого слова this

## 

5.10. Сериализация JSON

Для работы JSON-сериализации необходимо установить NuGet-пакет **Newtonsoft.Json**. Отличие JSON-сериализации от бинарной в том, что экземпляр класса сериализуется в формате JSON [14], который является хорошо читаемым и понимаемым для человека (рис. 5.27). На рис. 5.28 приведен пример JSON-сериализации.



*Рис. 5.27.* Результат JSON-сериализации



*Рис. 5.28.* Пример JSON-сериализации

Для расширения и углубления знаний по данной теме рекомендуется изучить информационные источники [1, 15–21].

ПРАКТИЧЕСКОЕ ЗАДАНИЕ 5

**Вариант 1. Класс File.**

*Задача 1*

* создать Текстовый документ {НазваниеГруппы}.txt;
* записать в файл имена и фамилии всех одногруппников;
* сделать резервную копию файла;
* удалить файл.

*Задача 2*

* обойти все диски;
* обойти все папки на дисках;
* сохранить полные пути к документам в List<srting> docs;
* сохранить коллекцию docs в файл.

*Задача 3*

* обойти все диски;
* обойти все папки на дисках;
* скопировать exe-файл во все директории, к которым есть доступ;
* написать программу, которая:
* обходит все диски;
* обходит все папки;
* проверяет, есть ли в заданных папках файлы с расширением .exe. Если MD5-сумма файла совпадает с искомой, программа удаляет файл.

**Вариант 2. Класс FileInfo.**

*Задача 1*

* создать текстовый документ {НазваниеГруппы}.txt;
* записать в файл имена и фамилии всех одногруппников;
* сделать резервную копию файла;
* удалить файл.

*Задача 2*

* обойти все диски;
* обойти все папки на дисках;
* сохранить полные пути к картинкам в List<srting> pictures;
* сохранить коллекцию pictures в файл.

*Задача 3*

Написать программу для резервного копирования папки.

* Программа проверяет, если копирование в указанную папку не осуществлялось, то в папке для копирования создается подпапка с версией 1 и в нее копируется содержимое.
* При повторном запуске программы, если исходная папка изменилась (нужно взять последнюю версию для копирования и проверить количество файлов, наименование, MD5-суммы файлов), то в папке для копирования создается подпапка с новой версией и копируется содержимое.
* Если в папке для копирования нет изменений, ничего не происходит.

Контрольные вопросы

1. Как получить список логических дисков в языке C#?

2. Расскажите о классе File.

3. Расскажите о классе FileInfo.

4. В чем отличие классов File и FileInfo?

5. Для чего служит класс Path?

6. Расскажите о классе Directory.

7. Расскажите о типе потока Stream.

8. Расскажите о потоках StreamWrite и StreamRead.

9. Расскажите про бинарную сериализацию.

10. Расскажите про JSON-сериализацию.

ЗАКЛЮЧЕНИЕ

В пособии на примере конструкций языка C# были рассмотрены основополагающие принципы объектно-ориентированной парадигмы программирования:

* классы и объекты (глава 1);
* свойства классов (глава 1);
* абстракция как в форме перегрузки операторов (глава 1), так   
  и в виде абстрактных классов и интерфейсов (глава 3);
* наследование (глава 2);
* инкапсуляция в форме различных модификаторов доступа свойств и методов (глава 2).

Также рассмотрены некоторые примеры реализаций объектно-ориен­ти­рованных моделей на основе библиотек фреймворка .Net:

* класс Object как наиболее общая модель произвольного объекта (глава 1);
* различные подходы к созданию массивов, коллекций (глава 4);
* классы библиотеки System.IO, предназначенные для работы с файловой системой (глава 5).

Для углублённого изучения дисциплины можно обратить внимание на книгу Джеффри Рихтера «CLR via C#. Программирование на платформе Microsoft.NET Framework 4.5 на языке C#», где подробно описывается внутреннее устройство и функционирование общеязыковой исполняющей среды (CLR) Microsoft .NET Framework версии 4.5.

Во второй части пособия будут раскрыты следующие вопросы: обработка исключений; использование конструкций LINQ; делегаты; события; рефлексия; многопоточность; асинхронность.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Рихтер, Джеффри. CLR via C#. Программирование на платформе Microsoft.NET Framework 4.5 на языке C# / Джеффри Рихтер. – Москва : Питер, 2019. – 896 c. – ISBN: 978-5-4461-1102-2.
2. Троелсен, Эндрю. Язык программирования C#7 и платформы .NET   
   и .NET Core / Эндрю Троелсен, Филипп Джепикс ; пер. с англ. Ю. Н. Артеменко. – 8-е изд. – Москва : Диалектика ; Санкт-Петербург : Диалектика, 2018. – 1328 с. – ISBN 978-5-6040723-1-8.
3. Албахари, Джозеф. C# 7.0. Справочник : полное описание языка : пер. с англ. / Джозеф Албахари и Бен Албахари. – 7-е изд. – Москва : Диалектика ; Санкт-Петербург : Альфа-книга, 2018. – 1023 с. – ISBN 978-5-6040043-7-1.
4. Албахари, Джозеф. C# 7.0. Карманный справочник : скорая помощь для программистов на C# 7.0 / Джозеф Албахари, Бен Албахари ; пер. с англ. и ред. Ю. Н. Артеменко. – Москва : Диалектика, 2017. – 224 с. – ISBN 978-5-9909446-1-9.
5. Ошероув, Рой. Искусство автономного тестирования с примерами на C# / Р. Ошероув ; пер. с англ. А. А. Слинкин. – Москва : ДМК Пресс,   
   2014. – 360 с. – ISBN 978-5-94074-945-5.
6. A tour of the C# language. – URL: <https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/> (date accessed: 21.06.2021).
7. Classes (C# Programming Guide). – URL: [https://docs.microsoft. com/ru-ru/dotnet/csharp/tour-of-csharp/classes-and-objects](https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/classes-and-objects) (date accessed: 21.06.2021).
8. Properties (C# Programming Guide). – URL: [https://docs.microsoft. com/ru-ru/dotnet/csharp/programming-guide/classes-and-structs/properties](https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/classes-and-structs/properties) (date accessed: 21.06.2021).
9. C# operators and expressions (C# reference). – URL: [https:// docs.microsoft. com/ru-ru/dotnet/csharp/language-reference/operators/operator-verloading](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/operator-verloading) (date accessed: 21.06.2021).
10. Enumeration types (C# reference). – URL: [https://docs.microsoft.com/ ru-ru/dotnet/csharp/language-reference/builtin-types/enum](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/builtin-types/enum) (date accessed: 21.06.2021).
11. Object. ToString Method. – URL: <https://docs.microsoft.com/ru-ru/dotnet/api/system.object.tostring?view=netcore-3.1#System_Object_ToString> (date accessed: 21.06.2021).
12. Object. Reference Equals (Object, Object) Method. – URL: [https://docs.microsoft.com/ru-ru/dotnet/api/system.object.referenceequals?view =netcore-3.1#System\_Object\_ReferenceEquals\_System\_Object\_System\_   
    Object\_](https://docs.microsoft.com/ru-ru/dotnet/api/system.object.referenceequals?view=netcore-3.1#System_Object_ReferenceEquals_System_Object_System_Object_) (date accessed: 21.06.2021).
13. Object. MemberwiseClone Method. – URL: [https://docs.microsoft. com/ru-ru/dotnet/api/system.object.memberwiseclone?view=netcore-3.1#Syst em\_Object\_MemberwiseClone](https://docs.microsoft.com/ru-ru/dotnet/api/system.object.memberwiseclone?view=netcore-3.1#System_Object_MemberwiseClone) (date accessed: 21.06.2021).
14. Object. GetHashCode Method. – URL: [https://docs.microsoft.com/ru-ru/dotnet/api/system.object.gethashcode?view=netcore-3.1#System\_Object\_ GetHashCode](https://docs.microsoft.com/ru-ru/dotnet/api/system.object.gethashcode?view=netcore-3.1#System_Object_GetHashCode) (date accessed: 21.06.2021).
15. Indexers (C# Programming Guide). – URL: [https://docs.   
    microsoft.com/ru-ru/dotnet/csharp/programming-guide/indexers/](https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/indexers/) (date acces­sed: 21.06.2021).
16. Binary serialization. – URL: [https://docs.microsoft.com/ru-ru/dotnet/   
    standard/serialization/binary-serialization](https://docs.microsoft.com/ru-ru/dotnet/%20standard/serialization/binary-serialization) (date accessed: 21.06.2021).
17. JSON serialization and deserialization (marshalling and unmarshalling) in .NET – overview. – URL: [https://docs.microsoft.com/ru-ru/dotnet/standard/ serialization/system-text-json-overview](https://docs.microsoft.com/ru-ru/dotnet/standard/%20serialization/system-text-json-overview) (date accessed: 21.06.2021).
18. Видео курс C# базовый (ООП). – URL: [https://itvdn.com/ru/video/ csharp-essential](https://itvdn.com/ru/video/csharp-essential) (date accessed: 21.06.2021).
19. C# для профессионалов (ООП). – URL: [https://itvdn.com/ru/video/ csharp-for-professional-renewed](https://itvdn.com/ru/video/csharp-for-professional-renewed) (date accessed: 21.06.2021).
20. Свидетельство о регистрации программы для ЭВМ № 2020663836 Российская Федерация. Комплект программ на основе методологии объектно-ориентированного программирования : № 2020663836 ; заявл. 03.10.2020 ; опубл. 03.10.2020 / П. П. Степанов, Г. В. Никонова, А. А. Кабанов, Т. С. Павлюченко ; заявитель и правообладатель ОмГТУ. – 1 с.
21. Свидетельство о регистрации программы для ЭВМ № 2020663837 Российская Федерация. Программы обработки данных на основе структурного программирования : № 2020663837 ; заявл. 03.10.2020 ; опубл. 03.10.2020 / П. П. Степанов, Г. В. Никонова, А. А. Кабанов, Т. С. Павлюченко ; заявитель и правообладатель ОмГТУ. – 1 с.

ПРИЛОЖЕНИЕ

ЛИСТИНГИ ПРОГРАММ

namespace LibExample

{

public class ClassExample

{

public bool Field1;

public int Field2;

public double Field3;

public string Info => $"{Field1} {Field2} {Field3}";

public ClassExample(bool field1, int field2, double field3)

{

Field1 = field1;

Field2 = field2;

Field3 = field3;

}

}

}

namespace LibExample

{

public class Man

{

#region private fields

private string \_firstName;

private string \_secondName;

private int \_age;

#endregion

#region Constructors

// Конструктор по умолчанию.

public Man()

{

this.\_firstName = "Валерий";

this.\_secondName = "Кипелов ";

this.\_age = 0;

}

// Конструктор с параметрами.

public Man(string firstName, string secondName, int age)

{

this.\_firstName = firstName;

this.\_secondName = secondName;

this.\_age = age;

}

#endregion

#region public properties

// Примеры использования Свойств.

public string FirstName

{

get

{

if (\_firstName == "")

return "В поле отсутствуют данные.";

else

return \_firstName;

}

set { \_firstName = value; }

}

public string SecondName

{

get

{

if (\_secondName == "")

return "В поле отсутствуют данные.";

else

return \_secondName;

}

set { \_secondName = value; }

}

public int Age

{

get

{

return \_age;

}

}

#endregion

#region public metods

public string ManInfo()

{

return $"FirstName - {this.\_firstName} SecondName - {this.SecondName} Age - {this.Age}";

}

// Пример неглубокого копирования.

public Man MemberwiseClone()

{

return (Man) base.MemberwiseClone();

}

#endregion

#region operators

public static bool operator ==(Man firstMan, Man secondMan)

{

if (firstMan == null || secondMan == null)

return false;

return ((firstMan.Age == secondMan.Age)

&& (firstMan.FirstName == secondMan.FirstName)

&& (firstMan.\_secondName == secondMan.\_secondName))

? true

: false;

}

public static bool operator !=(Man firstMan, Man secondMan)

{

if (firstMan == null || secondMan == null)

return true;

return ((firstMan.Age == secondMan.Age)

&& (firstMan.FirstName == secondMan.FirstName)

&& (firstMan.\_secondName == secondMan.\_secondName))

? false

: true;

}

#endregion

}

}

namespace LibExample

{

public struct StructExample

{

public bool Field1;

public int Field2; // = 112233;

public double Field3;

public string Info => $"{Field1} {Field2} {Field3}";

public StructExample(bool field1, int field2, double field3)

{

Field1 = field1;

Field2 = field2;

Field3 = field3;

}

}

}

using System;

namespace \_001\_Classes

{

class Man

{

public int Age;

public string FirstName;

public string SecondName;

}

class Program

{

private static void Main(string[] args)

{

// Создание экземпляра класса Man с именем person.

var person = new Man();

person.FirstName = "Ivan";

person.SecondName = "Ivanov";

person.Age = 33;

Console.WriteLine("FirstName - {0} " +

"SecondName - {1} " +

"Age - {2}",

person.FirstName, person.SecondName,

person.Age);

Console.Read();

}

}

}

namespace \_002\_Classes

{

class Man

{

#region public fields

public string FirstName;

public string SecondName;

public int Age;

#endregion

#region public metods

public string ManInfo()

{

return string.Format("FirstName - {0} " +

"SecondName - {1} " +

"Age - {2}",

FirstName, SecondName, Age);

}

#endregion

}

}

using System;

namespace \_002\_Classes

{

class Program

{

static void Main(string[] args)

{

Man person = new Man();

person.FirstName = "Энакин";

person.SecondName = "Скайуокер";

person.Age = 33;

Console.WriteLine(person.ManInfo());

Console.Read();

}

}

}

namespace \_003\_Classes

{

class Man

{

#region public fields

public string FirstName;

public string SecondName;

public int Age;

#endregion

#region Constructors

// Конструктор по умолчанию.

public Man()

{

FirstName = "Энакин";

SecondName = "Скайуокер";

Age = 33;

}

// Конструктор с параметрами.

public Man(string firstName, string secondName, int age)

{

FirstName = firstName;

SecondName = secondName;

Age = age;

}

#endregion

#region public metods

public string ManInfo()

{

return $"FirstName - {FirstName} SecondName - {SecondName} Age - {Age}";

}

// Значение переменной по умолчанию.

public string ManInfoWithPlanet(string planet = "Татуин")

{

return $"FirstName - {FirstName} SecondName - {SecondName} Age - {Age} Planet - {planet}";

}

#endregion

}

}

using System;

namespace \_003\_Classes

{

class Program

{

static void Main(string[] args)

{

// Пример создания класса с использованием конструктора   
по умолчанию.

Man person = new Man();

// Пример создания класса с использованием конструктора   
с параметрами.

Man person2 = new Man("Оби-Ван", "Кеноби", 57);

Console.WriteLine(person.ManInfo());

Console.WriteLine(person2.ManInfo());

Console.Read();

}

}

}

namespace \_004\_Properties

{

internal class Man

{

#region public metods

public string ManInfo()

{

return $"FirstName - {FirstName} SecondName - {SecondName} Age - {Age}";

}

#endregion

#region Constructors

// Конструктор по умолчанию.

public Man()

{

FirstName = "Энакин";

SecondName = "Скайуокер";

Age = 33;

}

// Конструктор с параметрами.

public Man(string firstName, string secondName, int age)

{

FirstName = firstName;

SecondName = secondName;

Age = age;

}

#endregion

#region public properties

// Примеры использования cвойств.

public string FirstName { set; get; }

public string SecondName { set; get; }

public int Age { set; get; }

#endregion

}

}

using System;

namespace \_004\_Properties

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Hello World!");

}

}

}

using System;

namespace \_005\_Properties

{

internal class Man

{

#region public metods

public string ManInfo()

{

return $"FirstName - {FirstName} SecondName - {SecondName} Age - {Age}";

}

#endregion

#region private fields

private string \_firstName;

private string \_secondName;

private int \_age;

#endregion

#region Constructors

// Конструктор по умолчанию.

public Man()

{

\_firstName = "";

\_secondName = "";

\_age = 0;

}

// Конструктор с параметрами.

public Man(string firstName, string secondName, int age)

{

\_firstName = firstName;

\_secondName = secondName;

\_age = age;

}

#endregion

#region public properties

// Примеры использования свойств.

public string FirstName

{

set => \_firstName = value;

get

{

if (\_firstName == "")

return "В поле отсутствуют данные.";

return \_firstName;

}

}

public string SecondName

{

set => \_secondName = value;

get

{

if (\_secondName == "")

return "В поле отсутствуют данные.";

return \_secondName;

}

}

public int Age

{

set

{

if (value <= 0 || value >= 120)

Console.WriteLine("Вы ввели недопустимое значение. Повторите попытку.");

else

\_age = value;

}

get => \_age;

}

#endregion

}

}

using System;

namespace \_005\_Properties

{

class Program

{

static void Main(string[] args)

{

var person = new Man();

Console.WriteLine($"{person.FirstName}, {person.SecondName}, {person.Age}");

person.FirstName = "Ivan";

Console.WriteLine($"{person.FirstName}, {person.SecondName}, {person.Age}");

Console.ReadKey();

}

}

}

namespace \_006\_Properties

{

class Man

{

#region private fields

private string \_firstName;

private string \_secondName;

private int \_age;

#endregion

#region Constructors

// Конструктор по умолчанию.

public Man()

{

this.\_firstName = "Валерий";

this.\_secondName = "Кипелов ";

this.\_age = 0;

}

// Конструктор с параметрами.

public Man(string firstName, string secondName, int age)

{

this.\_firstName = firstName;

this.\_secondName = secondName;

this.\_age = age;

}

#endregion

#region public properties

// Примеры использования cвойств.

public string FirstName

{

set { \_firstName = value; }

}

public string SecondName

{

get

{

if (\_secondName == "")

return "В поле отсутствуют данные.";

else

return \_secondName;

}

}

public int Age

{

get

{

return \_age;

}

}

#endregion

#region public metods

public string ManInfo()

{

return $"FirstName - {this.\_firstName} SecondName - {this.SecondName} Age - {this.Age}";

}

#endregion

}

}

using System;

namespace \_006\_Properties

{

class Program

{

static void Main(string[] args)

{

Man person = new Man();

//Console.WriteLine(person.FirstName);

// person.FirstName = "Энакин";

//Console.WriteLine(person.SecondName);

person.SecondName = "Скайуокер";

Console.ReadKey(); ;

}

}

}

namespace \_008\_Operators

{

class Man

{

#region private fields

private string \_firstName;

private string \_secondName;

private int \_age;

#endregion

#region Constructors

// Конструктор по умолчанию.

public Man()

{

this.\_firstName = "Валерий";

this.\_secondName = "Кипелов ";

this.\_age = 0;

}

// Конструктор с параметрами.

public Man(string firstName, string secondName, int age)

{

this.\_firstName = firstName;

this.\_secondName = secondName;

this.\_age = age;

}

#endregion

#region public properties

// Примеры использования свойств.

public string FirstName

{

get

{

if (\_firstName == "")

return "В поле отсутствуют данные.";

else

return \_firstName;

}

set { \_firstName = value; }

}

public string SecondName

{

get

{

if (\_secondName == "")

return "В поле отсутствуют данные.";

else

return \_secondName;

}

set { \_secondName = value; }

}

public int Age

{

get

{

return \_age;

}

}

#endregion

#region public metods

public string ManInfo()

{

return $"FirstName - {this.\_firstName} SecondName - {this.SecondName} Age - {this.Age}";

}

#endregion

public static bool operator ==(Man firstMan, Man secondMan)

{

if (firstMan == null || secondMan == null)

return false;

return ((firstMan.Age == secondMan.Age)

&& (firstMan.FirstName == secondMan.FirstName)

&& (firstMan.\_secondName == secondMan.\_secondName))

? true

: false;

}

public static bool operator !=(Man firstMan, Man secondMan)

{

if (firstMan == null || secondMan == null)

return true;

return ((firstMan.Age == secondMan.Age)

&& (firstMan.FirstName == secondMan.FirstName)

&& (firstMan.\_secondName == secondMan.\_secondName))

? false

: true;

}

}

}

using System;

using System.IO;

namespace \_018\_StreamWrite

{

internal class Program

{

private static void Main(string[] args)

{

if(File.Exists("Test.dat"))

File.Create("Test.dat").Dispose();

using (var streamWriter = new StreamWriter("Test.dat"))

{

streamWriter.Write("1");

streamWriter.Write(streamWriter.NewLine);

streamWriter.Write("Example add String");

Console.ReadLine();

}

using (StreamReader streamReader = new StreamReader("Test.dat"))

{

while (!streamReader.EndOfStream)

// Считывание построчно.

Console.WriteLine(streamReader.ReadLine());

streamReader.BaseStream.Position = 0;

// Считывание всего файла в строку.

Console.WriteLine(streamReader.ReadToEnd());

Console.ReadLine();

}

}

}

}

using System;

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

using LibExample;

namespace \_019\_Serialize

{

class Program

{

static void Main(string[] args)

{

Man man = new Man("Иван", "Иванов", 65);

Man manFromFile = null;

using (FileStream stream = new FileStream("Man.bin", FileMode.Create))

{

// Пример двоичной сериализации.

BinaryFormatter binaryFormatter = new BinaryFormatter();

binaryFormatter.Serialize(stream, man);

}

Console.WriteLine(man.ManInfo());

using (FileStream stream = new FileStream("Man.bin", FileMode.Open))

{

// Пример двоичной десериализации.

BinaryFormatter binaryFormatter = new BinaryFormatter();

manFromFile = (Man)binaryFormatter.Deserialize(stream);

}

Console.WriteLine(manFromFile.ManInfo());

Console.ReadLine();

}

}

}

using System;

using LibExample;

namespace \_020\_Serialize

{

class Program

{

static void Main(string[] args)

{

Man man = new Man("Иван", "Иванов", 65);

man.BinarySerialization();

Man manFromFile = Man.StaticBinaryDeserialization();

Console.WriteLine(man.ManInfo());

Console.WriteLine(manFromFile.ManInfo());

Console.ReadLine();

}

}

}

using System;

using System.IO;

using LibExample;

using Newtonsoft.Json;

namespace \_021\_Serialize

{

internal class Program

{

private static void Main(string[] args)

{

//Install-Package Newtonsoft.Json -Version 12.0.2

var man = new Man("Иван", "Иванов", 65);

Man manFromFile = null;

using (var stream = File.CreateText("Man.json"))

{

// Пример JSON-сериализации.

var serializer = new JsonSerializer();

serializer.Serialize(stream, man);

}

Console.WriteLine(man.ManInfo());

using (var file = File.OpenText("Man.json"))

{

// Пример JSON-десериализации.

var serializer = new JsonSerializer();

manFromFile = (Man) serializer.Deserialize(file, typeof(Man));

}

Console.WriteLine(manFromFile.ManInfo());

Console.ReadLine();

}

}

}