УДК 004.43
ББК 32.973.26-018.1
В19

Васильев, Алексей Николаевич.

В19 Программирование на C# для начинающих. Особенности языка / Алексей Васильев. — Москва : Эксмо, 2019. — 528 с. — (Российский компьютерный бестселлер).

ISBN 978-5-04-092520-9

Вторая книга известного российского автора самоучителей по программированию, посвященная особенностям языка C# и его практическому применению. Из этой книги вы узнаете, какие основные структурные единицы языка существуют, научитесь писать программы, используя все основные методы и интерфейсы, и овладеете одним из самых востребованных и популярных языков семейства C.

УДК 004.43
ББК 32.973.26-018.1

ISBN 978-5-04-092520-9

ОГЛАВЛЕНИЕ

Введение
РАСШИРЕННЫЕ ВОЗМОЖНОСТИ C#

Я мечтал об этом всю свою сознательную жизнь.

из к/ф «Ирония судьбы или с легким паром»

Вниманию читателя предлагается вторая часть книги о языке программирования C#. В первой части книги рассмотрены базовые синтаксические конструкции языка, основные типы данных, управляющие инструкции (условный оператор, оператор выбора, операторы цикла) и массивы. Также в ней описывались способы создания классов и объектов, методы их использования. Еще в первой части состоялось знакомство с индексаторами и свойствами, операторными методами. Наследование, перегрузка и переопределение методов также относятся к первой части книги. Предполагается, что читатель знаком (хотя бы на начальном уровне) с этими темами. В противном случае, перед изучением материала из второй части книги, следует освежить в памяти материал из первой части (или обратиться к другому аналогичному учебному пособию).

О чем пойдет речь

Я бы на вашем месте за докторскую диссертацию немедленно сел.

из к/ф «Иван Васильевич меняет профессию»

Далее мы рассмотрим различные темы, связанные с наиболее актуальными и перспективными механизмами языка C#:

- Сначала мы познакомимся с абстрактными классами. Узнаем, в чем особенность абстрактных классов, зачем они нужны и как используются.
- Мы рассмотрим способы создания и использования интерфейсов.

- Важные механизмы связаны с использованием делегатов. Мы узнаем, как объявляются делегаты, как на их основе создаются экземпляры и какую роль при этом играют ссылки на методы. Также состоится знакомство с анонимными методами и лямбда-выражениями.
- Кроме полей, методов, свойств и индексаторов, в классах могут быть такие члены, как события. Зачем они нужны и как используются, описывается в этой книге.
- Мы познакомимся с перечислениями — специальными типами, возможный диапазон значений которых определяется набором констант.
- Кроме классов, в языке C# широко используются структуры. Работе со структурами в книге уделяется отдельное внимание.
- Мощный механизм, связанный с выполнением операций с памятью, базируется на использовании указателей. Мы рассмотрим и эту тему.
- Отдельная глава посвящена перехвату и обработке исключений (ошибок, которые возникают в процессе выполнения программы).
- Язык C# имеет средства поддержки многопоточного программирования, позволяющие одновременно выполняться нескольким частям программы. Способы создания многопоточных приложений описываются в этой книге.
- Обобщенные типы — элегантный механизм, позволяющий создавать красивый и эффективный программный код. Данной теме в книге уделено достаточно внимания.
- Одна из глав книги посвящена вопросам создания приложений с графическим интерфейсом.
- Также у нас состоится краткое знакомство с коллекциями, а еще мы научимся выполнять различные операции с файлами.

Будет и кое-что еще.

Необходимые навыки и ресурсы

> Наконец-то все закончится. И я смогу спокойно поиграть в шахматы. И вообще, пора на пенсию.
>
> *из к/ф «Гостья из будущего»*

Если читатель предварительно ознакомился с содержанием первой части этой книги, то полученных знаний и навыков будет вполне достаточно для того, чтобы разобраться с представленным далее материалом. Вообще же для успешной и эффективной работы со второй частью книги нужно иметь представление о структуре программы в языке C#, базовых типах данных, операторах и управляющих инструкциях. Понадобятся знания в плане создания и использования массивов, описания классов и создания на их основе объектов. Мы будем использовать свойства и индексаторы, операторные методы, прибегать к наследованию, перегрузке и переопределению методов.

Предполагается, что читатель знаком со способами создания приложений на языке C# и обладает необходимыми навыками для работы со средой разработки. В качестве последней предлагается использовать приложение Microsoft Visual Studio (или некоммерческую версию Microsoft Visual Studio Express). Примеры из книги тестировались именно в этой среде разработки. Минимальная, но вполне достаточная для успешной работы информация о среде разработки есть в первой части книги. В этой, второй части в некоторых случаях (когда в этом есть необходимость) также даются пояснения по поводу особенностей работы со средой разработки.

Обратная связь с автором

Автор книги — *Васильев Алексей Николаевич*, доктор физико-математических наук, профессор кафедры теоретической физики физического факультета Киевского национального университета имени Тараса Шевченко. Информацию об этой и других книгах автора можно найти на сайте `www.vasilev.kiev.ua`. Вопросы, замечания и предложения можно отправлять автору по адресу электронной почты `alex@vasilev.kiev.ua`.

Глава 1
АБСТРАКТНЫЕ КЛАССЫ И ИНТЕРФЕЙСЫ

Мерзавец, а, мерзавец, ты, значит, здесь вместо работы латынь изучаешь?

из к/ф «Формула любви»

В этой главе мы в некотором смысле продолжим тему наследования, рассмотренную в первой части книги. Но сейчас речь пойдет скорее о «сопутствующих» технологиях. Нам предстоит познакомиться с абстрактными классами и интерфейсами. Мы узнаем:

- что такое абстрактный класс и зачем он нужен;
- что такое интерфейс, как он описывается и как используется;
- в чем особенность интерфейсных переменных;
- как реализуется расширение интерфейсов;
- что такое явная реализация интерфейса и в чем ее специфика.

Также мы рассмотрим дополнительные аспекты, имеющие отношение к использованию абстрактных классов и интерфейсов.

Знакомство с абстрактными классами

— Астронавты! Которая тут цаппа?
— Там... ржавая гайка, родной.

из к/ф «Кин-дза-дза»

Есть такое понятие, как *абстрактный метод*. Абстрактным называется метод, который в классе только объявлен, но не описан. Под объявлением метода подразумевается ситуация, когда в теле класса указан тип результата метода, его название и приведен список аргументов (после

закрывающей круглой скобки ставится точка с запятой), а тела метода нет. То есть блока из фигурных скобок с командами, выполняемыми при вызове метода, нет вообще. Но чтобы метод был абстрактным, недостаточно описать его без тела с командами. В описании метода необходимо явно указать, что метод абстрактный. Для этого используется ключевое слово `abstract`. Шаблон описания абстрактного метода представлен ниже (жирным шрифтом выделены ключевые элементы шаблона):

```
доступ abstract тип имя(аргументы);
```

Сначала указывается спецификатор уровня доступа, затем ключевое слово `abstract`, после него — идентификатор типа результат метода, название и список аргументов. При этом аргументы описываются, как и в обычном (не абстрактном) методе, с указанием типа аргумента и его формального названия. В конце ставится точка с запятой.

Если в классе есть хотя бы один абстрактный метод, то такой класс также считается *абстрактным*. Абстрактный класс, как и абстрактный метод, описывается с ключевым словом `abstract`.

Поскольку у абстрактного метода нет тела и не известно, какие команды должны выполняться при вызове метода, то такой метод вызвать нельзя. Поэтому, как вы могли догадаться, на основе абстрактного класса нельзя создать объект. В этом случае все логично: если бы такой объект можно было создать, то у него был бы метод, который нельзя вызвать. А это, как минимум, странно и непоследовательно. Но тогда возникает вопрос: а зачем вообще нужны абстрактные классы? Ответ состоит в том, что абстрактный класс может (и должен) использоваться как базовый при наследовании. Это его главное и наиболее важное назначение — быть базовым классом. Общая схема такова: создается абстрактный класс, а затем путем наследования на основе абстрактного класса создаются обычные (не абстрактные) производные классы. При этом в производных классах абстрактные методы переопределяются: описывается полная версия метода, и, как при переопределении обычных методов, в описании указывается ключевое слово `override`. В производном классе должны быть переопределены (описаны) все абстрактные методы из абстрактного базового класса.

НА ЗАМЕТКУ

Абстрактный метод по определению является виртуальным. При этом ключевое слово `virtual` в объявлении абстрактного метода не используется. Также следует учитывать, что абстрактный метод не может быть статическим.

Преимущество использования абстрактного класса как базового состоит в том, что, описывая базовый класс, мы фактически создаем некий шаблон для производных классов. Все производные классы, созданные на основе одного и того же абстрактного класса, будут иметь определенный набор методов. В каждом из классов эти методы реализуются по-своему, но они есть, и мы в этом можем быть уверены.

Конечно, никто не запрещает нам описать обычный (не абстрактный) класс и затем на его основе создавать производные классы, переопределяя в них методы их базового класса. Но это не самый лучший подход, поскольку если забыть переопределить в производном классе метод (который надо переопределить), то в производном классе будет использована унаследованная версия метода из базового класса и формальной ошибки в этом не будет. А такие ситуации сложно отслеживать. Делая же методы абстрактными в базовом абстрактном классе, мы с необходимостью должны будем описать их в производном классе. Если этого не сделать, программа просто не скомпилируется.

ПОДРОБНОСТИ

Если класс описать с ключевым словом `abstract`, то он будет абстрактным, даже если в классе нет ни одного абстрактного метода. При наследовании абстрактного класса в производном классе можно переопределять не все абстрактные методы из базового класса. Но в таком случае производный класс также будет абстрактным и должен быть описан с ключевым словом `abstract`.

Также следует заметить, что хотя создать объект абстрактного класса нельзя, но можно объявить объектную переменную для абстрактного класса. Эта переменная не может ссылаться на объект абстрактного класса (поскольку такой объект создать нельзя), зато она может ссылаться на объект производного класса. А поскольку абстрактные методы по умолчанию являются виртуальными и переопределяются в производном классе, то через объектную переменную базового абстрактного класса будут вызываться именно те версии методов, что определены в производном классе. Далее перейдем к рассмотрению примеров.

ПОДРОБНОСТИ

Абстрактными могут быть также свойства и индексаторы. При описании абстрактного свойства или индексатора используется ключевое слово `abstract`. В теле свойства или метода аксессоры

не описываются. Указываются только ключевые слова `get` и `set` или только одно из них, если у свойства или индексатора только один аксессор.

В производном классе свойство или индексатор описываются с ключевым словом `override`, как при переопределении метода. При этом должны быть описаны все аксессоры, объявленные в абстрактном классе для данного свойства или индексатора.

Использование абстрактных классов

— Дядя Вова. Цаппу надо крутить, цаппу.
— На! Сам делай!
— Мне нельзя, я чатланин.

из к/ф «Кин-дза-дза»

Для начала мы рассмотрим очень простой пример, в котором описывается абстрактный класс, а затем на основе этого класса создаются производные классы. Рассмотрим программу, представленную в листинге 1.1.

Листинг 1.1. Знакомство с абстрактными классами

```
using System;
// Абстрактный класс:
abstract class Base{
   // Защищенное целочисленное поле:
   protected int num;
   // Конструктор:
   public Base(int n){
      // Вызов метода:
      set(n);
   }
   // Абстрактные методы:
   public abstract void show();
   public abstract void set(int n);
   public abstract int get();
```

```
}
// Производный класс на основе абстрактного класса:
class Alpha:Base{
   // Защищенное целочисленное поле:
   protected int val;
   // Конструктор:
   public Alpha(int n):base(n){
      // Вызов метода:
      show();
   }
   // Переопределение абстрактного метода:
   public override void show(){
      // Отображение сообщения:
      Console.WriteLine("Alpha: {0}, {1} и {2}",num,val,get());
   }
   // Переопределение абстрактного метода:
   public override void set(int n){
      // Присваивание значений полям:
      num=n;
      val=n%10;
   }
   // Переопределение абстрактного метода:
   public override int get(){
      return num/10;
   }
}
// Производный класс на основе абстрактного класса:
class Bravo:Base{
   // Защищенное целочисленное поле:
   protected int val;
   // Конструктор:
   public Bravo(int n):base(n){
```

```
      // Вызов метода:
      show();
   }
   // Переопределение абстрактного метода:
   public override void show(){
      // Отображение сообщения:
      Console.WriteLine("Bravo: {0}, {1} и {2}",num,val,get());
   }
   // Переопределение абстрактного метода:
   public override void set(int n){
      // Присваивание значений полям:
      num=n;
      val=n%100;
   }
   // Переопределение абстрактного метода:
   public override int get(){
      return num/100;
   }
}
// Класс с главным методом:
class AbstractDemo{
   // Главный метод:
   static void Main(){
      // Объектная переменная абстрактного класса:
      Base obj;
      // Создание объектов производных классов:
      Alpha A=new Alpha(123);
      Bravo B=new Bravo(321);
      // Объектной переменной базового класса присваивается
      // ссылка на объект производного класса:
      obj=A;
      Console.WriteLine("После выполнения команды obj=A");
```

```
        // Вызов методов через объектную переменную
        // базового класса:
        obj.set(456);
        obj.show();
        // Объектной переменной базового класса присваивается
        // ссылка на объект производного класса:
        obj=B;
        Console.WriteLine("После выполнения команды obj=B");
        // Вызов методов через объектную переменную
        // базового класса:
        obj.set(654);
        obj.show();
    }
}
```

Результат выполнения программы следующий:

Результат выполнения программы (из листинга 1.1)

```
Alpha: 123, 3 и 12
Bravo: 321, 21 и 3
После выполнения команды obj=A
Alpha: 456, 6 и 45
После выполнения команды obj=B
Bravo: 654, 54 и 6
```

В программе описан абстрактный класс `Base`. Он описан с ключевым словом `abstract`. В классе объявлены три абстрактных метода (все описаны с ключевым словом `abstract`): метод `show()` без аргументов и не возвращающий результат, метод `set()` с целочисленным аргументом и не возвращающий результат, метод `get()` без аргументов и возвращающий целочисленный результат. Все эти методы должны быть описаны в производном классе, создаваемом на основе данного абстрактного класса. Но не все «содержимое» абстрактного класса является «абстрактным». В нем описано закрытое целочисленное поле `num`, а также конструктор с одним целочисленным аргументом. В теле

конструктора содержится команда `set(n)`, которой метод `set()` вызывается с аргументом `n`, переданным конструктору. Интересно здесь то, что метод `set()` абстрактный и в классе `Base` не описан, а только объявлен.

ПОДРОБНОСТИ

Хотя на основе абстрактного класса объект создать нельзя, но можно описать конструктор для абстрактного класса. Этот конструктор будет вызываться при создании объекта производного класса, поскольку в этом случае сначала вызывается конструктор базового класса. В нашем примере в теле конструктора класса `Base` вызывается абстрактный метод `set()`, в классе не описанный. Но проблемы при этом не возникает, поскольку выполняться конструктор базового класса будет при создании объекта производного класса, в котором метод `set()` должен быть описан. Проще говоря, на момент, когда метод `set()` будет вызываться, он уже будет описан.

На основе класса `Base` создается два класса: `Alpha` и `Bravo`. Эти классы очень похожи, но описываются по-разному. Начнем с общих моментов для обоих классов. И в классе `Alpha`, и в классе `Bravo` появляется дополнительное целочисленное поле `val`. В каждом из классов описан конструктор с целочисленным аргументом, который передается конструктору базового класса. В теле конструктора вызывается метод `show()`. Но описывается метод `show()` в каждом классе со своими особенностями. В классе `Alpha` при вызове метода `show()` отображаются название класса `Alpha`, значения полей `num` и `val` и результат вызова метода `get()`. Метод `show()` для класса `Bravo` описан так же, но название класса отображается другое. Метод `get()` в каждом классе также свой. В классе `Alpha` метод `get()` описан так, что результатом возвращается значение `num/10`. Это значение поля `num`, если в нем отбросить разряд единиц. В классе `Bravo` результатом метода `get()` является значение `num/100`, которое получается отбрасыванием разрядов единиц и десятков в значении поля `num`.

Метод `set()` в классе `Alpha` переопределен таким образом, что при целочисленном аргументе `n` выполняются команды `num=n` и `val=n%10`. То есть полю `num` присваивается значение аргумента, а полю `val` в качестве значения присваивается остаток от деления значения аргумента на `10` (это последняя цифра в десятичном представлении числового значения аргумента `n`).

В классе `Bravo` метод `set()` описан похожим образом, но полю `val` значение присваивается командой `val=n%100` (остаток от деления