

НГТУ
НЭТИ

Факультет прикладной
математики
и информатики

М. Г. ЗАЙЦЕВ

УНИВЕРСАЛЬНЫЕ КОЛЛЕКЦИИ ЯЗЫКА C#

НОВОСИБИРСК
2022

Министерство науки и высшего образования Российской Федерации
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

М. Г. ЗАЙЦЕВ

УНИВЕРСАЛЬНЫЕ КОЛЛЕКЦИИ ЯЗЫКА C#

Утверждено
Редакционно-издательским советом университета
в качестве учебного пособия

НОВОСИБИРСК
2022

УДК 004.432(075.8)
З-177

Рецензенты:
канд. техн. наук, доц. *В. Г. Кобылянский*,
ст. преп. кафедры ТС и ВС СибГУТИ *Л. Ф. Лебеденко*

Зайцев М. Г.
З-177 Универсальные коллекции языка C# : учебное пособие / М. Г. Зайцев. – Новосибирск : Изд-во НГТУ, 2022. – 112 с.

ISBN 978-5-7782-4628-7

Учебное пособие предназначено для студентов факультета ПМИ, обучающихся по направлению: 02.03.03 – Математическое обеспечение и администрирование информационных систем по дисциплине «Технологии разработки программного обеспечения». Пособие охватывает часть материала, изучаемого по дисциплине (ТРПО). Излагаются вопросы, касающиеся универсальных коллекций языка C#, а также приведены указания к выполнению расчетно-графического задания и варианты его выполнения. Примеры упражнений выполнены в среде Visual Studio на языке C#.

Работа подготовлена кафедрой теоретической
и прикладной информатики

Зайцев Михаил Георгиевич

УНИВЕРСАЛЬНЫЕ КОЛЛЕКЦИИ ЯЗЫКА C#

Учебное пособие

Редактор *И. Л. Кескевич*
Выпускающий редактор *И. П. Бровановa*
Корректор *И.Е. Семенова*
Дизайн обложки *А. В. Ладыжская*
Компьютерная верстка *Н. В. Гаврилова*

Налоговая льгота – Общероссийский классификатор продукции
Издание соответствует коду 95 3000 ОК 005-93 (ОКП)

Подписано в печать 16.03.2022. Формат 60 × 84 1/16. Бумага офсетная
Тираж 50 экз. Уч.-изд. л. 6,51. Печ. л. 7,0. Изд. № 244/21. Заказ № 115. Цена договорная

Отпечатано в типографии
Новосибирского государственного технического университета
630073, г. Новосибирск, пр. К. Маркса, 20

УДК 004.432(075.8)

ISBN 978-5-7782-4628-7

© Зайцев М. Г., 2022
© Новосибирский государственный
технический университет, 2022

КОЛЛЕКЦИИ C#

Коллекция представляет собой способ организации и обработки совокупности связанных объектов. В C# имеется немало интерфейсов и классов, в которых определяются и реализуются различные типы коллекций. Коллекции упрощают решение многих задач программирования благодаря тому, что предлагают готовые решения для создания целого ряда типичных, но порой трудоемких для разработки структур данных. Например, в C# встроены коллекции, предназначенные для поддержки динамических массивов, связных списков, стеков, очередей и хеш-таблиц. Коллекции являются современным технологическим средством, заслуживающим пристального внимания всех, кто программирует на C#.

Первоначально существовали только классы необобщенных коллекций, но с внедрением обобщений в версии C# 2.0 среда .NET Framework была дополнена многими новыми обобщенными классами и интерфейсами. Вместе с библиотекой распараллеливания задач (TPL) в версии 4.0 среды .NET Framework появился ряд новых классов коллекций, предназначенных для применения в тех случаях, когда доступ к коллекции осуществляется из нескольких потоков. Прикладной интерфейс Collections API составляет значительную часть среды .NET Framework.

Во многих приложениях требуется создавать группы связанных объектов и управлять ими. Существует два способа группировки объектов: создать массив объектов и создать коллекцию.

Массивы удобнее всего использовать для создания фиксированного числа строго типизированных объектов и работы с ними.

Коллекции предоставляют более гибкий способ работы с группами объектов. В отличие от массивов коллекция, с которой вы работаете, может при необходимости динамически расти или уменьшаться.

Некоторые коллекции допускают назначение ключа любому объекту, который добавляется в коллекцию, чтобы в дальнейшем можно было быстро извлечь связанный с ключом объект из коллекции.

Коллекция является классом, поэтому необходимо объявить экземпляр класса перед добавлением в коллекцию элементов.

Если коллекция содержит элементы только одного типа данных, можно использовать один из классов в пространстве имен System. Collections.Generic. Универсальная (обобщенная) коллекция обеспечивает строгую типизацию, так что в нее нельзя добавить другие типы данных. Тип данных элементов задается при описании коллекции. При извлечении элемента из универсальной коллекции не нужно определять или преобразовывать его тип данных. Именно универсальные, или обобщенные, коллеции мы рассмотрим далее в этом пособии.

КРАТКИЙ ОБЗОР КОЛЛЕКЦИЙ

Главное преимущество коллекций заключается в том, что они обеспечивают стандартизацию обработки управляемых ими объектов в программе. Все коллекции разработаны на основе набора четко определенных интерфейсов. Имеется также возможность реализовать собственную коллекцию, хотя потребность в этом возникает крайне редко.

В среде .NET Framework поддерживаются пять типов коллекций: необобщенные, специальные, с поразрядной организацией, обобщенные и параллельные.

Необобщенные коллекции

Реализуют ряд основных структур данных, включая динамический массив, стек, очередь, а также словари, в которых можно хранить пары «ключ-значение». В отношении необобщенных коллекций важно иметь в виду следующее: они оперируют данными типа object. Таким образом, необобщенные коллекции могут служить для хранения данных любого типа, причем в одной коллекции допускается наличие разнотипных данных. Очевидно, что такие коллекции не типизированы, поскольку в них хранятся ссылки на данные типа object. Классы и интерфейсы необобщенных коллекций находятся в пространстве имен **System.Collections**.

Специальные коллекции

Оперируют данными конкретного типа или же делают это каким-то особым образом. Например, имеются специальные коллекции для символьных строк, а также специальные коллекции, в которых используется однонаправленный список. Специальные коллекции объявляются в пространстве имен **System.Collections.Specialized**.

Поразрядная коллекция

В прикладном интерфейсе Collections API определена одна коллекция с поразрядной организацией – это BitArray. Коллекция типа BitArray поддерживает поразрядные операции, т. е. операции над отдельными двоичными разрядами, например И, ИЛИ, исключающее ИЛИ, а следовательно, она существенно отличается своими возможностями от остальных типов коллекций. Коллекция типа BitArray объявляется в пространстве имен **System.Collections**.

Обобщенные коллекции

Обеспечивают обобщенную реализацию нескольких стандартных структур данных, включая связные списки, стеки, очереди и словари. Такие коллекции являются типизированными в силу их обобщенного характера. Это означает, что в обобщенной коллекции могут храниться только такие элементы данных, которые совместимы по типу с этой коллекцией. Благодаря этому исключается случайное несовпадение типов. Обобщенные коллекции объявляются в пространстве имен **System.Collections.Generic**.

Параллельные коллекции

Поддерживают многопоточный доступ к коллекции. Это обобщенные коллекции, определенные в пространстве имен **System.Collections.Concurrent**.

В пространстве имен System.Collections.ObjectModel находится также ряд классов, поддерживающих создание пользователями собственных обобщенных коллекций.

Основополагающим для всех коллекций является понятие *перечислителя* (*метода итератора*), который поддерживается в необобщенных интерфейсах IEnumerator и IEnumerable, а также в обобщенных интерфейсах IEnumerator<T> и IEnumerable<T>. Перечислитель обеспечивает стандартный способ поочередного доступа к элементам

коллекции. Следовательно, он перечисляет содержимое коллекции. В каждой коллекции должна быть реализована обобщенная или необобщенная форма интерфейса IEnumerable, чтобы элементы любого класса коллекции были доступны посредством методов, определенных в интерфейсе IEnumerator или IEnumerator<T>.

С перечислителем (методом итератора) непосредственно связано другое средство, называемое *итератором* (оператор foreach, см. раздел Итераторы). Это средство упрощает процесс получения последовательного доступа к элементам коллекций, поочередное обращение к которым организуется в цикле foreach.

Классы коллекций C# по своей сути подобны классам стандартной библиотеки шаблонов (Standard Template Library – STL), определенной в C++. То, что в программировании на C++ называется контейнером, в программировании на C# называется коллекцией.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое коллекция?
2. Что такое универсальная коллекция?
3. Какие типы коллекций поддерживаются в C#?
4. Что такок перечислитель и для чего он предназначен?
5. В чем преимущество коллекций перед массивами?

ИНТЕРФЕЙСЫ

Интерфейс (interface) – в переводе на русский язык означает сопряжение.

В языке C# интерфейс – это пользовательский тип данных, который определяет ряд методов (свойств, индексаторов, событий). Класс, который наследует (реализует) интерфейс, должен реализовать его методы (свойства, индексаторы, события). Интерфейсы используются для того, чтобы указать классам, что именно нужно реализовать в них.

Интерфейс определяет контракт. Любой класс (class) или структура (struct), реализующие этот контракт, должны предоставлять реализацию для членов, определенных в интерфейсе. Начиная с C# 8.0 интерфейс может определять для членов реализацию по умолчанию. Он также может определять члены как статические (static), чтобы

обеспечить единую реализацию для общих функциональных возможностей.

Использование интерфейсов эффективно в случаях, когда нужно создать альтернативу множественного наследования. Любой класс может унаследовать несколько интерфейсов. При этом все методы унаследованных интерфейсов должны быть реализованы в классе. Структура, так же как и класс, может реализовывать любое количество интерфейсов.

В следующем примере класс ImplementationClass должен реализовать метод с именем SampleMethod, не имеющий параметров и возвращающий значение void. В C# допускается описывать ссылки на интерфейс. Если описать переменную-ссылку на интерфейс, то с ее помощью можно вызвать методы класса, который использует этот интерфейс.

```
interface ISampleInterface
{
   void SampleMethod();
}

class ImplementationClass : ISampleInterface
{
   // Явная реализация членов интерфейса.
   void ISampleInterface.SampleMethod()
   {
      // Реализация метода.
   }

   static void Main()
   {
      // Объявление экземпляра интерфейса.
      ISampleInterface obj = new ImplementationClass();

      // Вызов члена интерфейса.
      obj.SampleMethod();
   }
}
```

Интерфейс может быть членом пространства имен или класса. Объявление интерфейса может содержать объявления (сигнатуры без реализации) следующих членов: методов, свойств, индексаторов, событий.

Эти объявления членов интерфейса обычно не содержат тело. Начиная с C#8.0 член интерфейса может объявлять тело. Это называется реализацией по умолчанию. Члены с телом позволяют интерфейсу предоставлять реализацию по умолчанию для классов и структур, которые не предоставляют реализацию с переопределением. Кроме того, начиная с C#8.0 интерфейс может включать следующее:

- константы,
- операторы,
- статический конструктор,
- вложенные типы,
- статические поля, методы, свойства, индексаторы и события,
- объявления членов с использованием синтаксиса явной реализации интерфейса,
- явные модификаторы доступа (доступ по умолчанию – public).

Интерфейсы не могут содержать состояние экземпляра. Хотя статические поля теперь разрешены, поля экземпляров в интерфейсах запрещены. Автосвойства экземпляра не поддерживаются в интерфейсах, так как они неявно объявляют скрытое поле. Это правило незначительно воздействует на объявления свойств. В объявлении интерфейса следующий код не объявляет автоматически реализуемое свойство, как в классе (class) или структуре (struct). Вместо этого он объявляет свойство, которое не имеет реализации по умолчанию, но должно быть реализовано в любом типе, реализующем интерфейс.

```
public interface INamed
{
  public string Name {get; set;}
}
```

Интерфейс может наследовать от одного или нескольких базовых интерфейсов. Когда интерфейс переопределяет метод, реализованный в базовом интерфейсе, он должен использовать синтаксис явной реализации интерфейса.

Если список базовых типов содержит базовый класс и интерфейсы, базовый класс должен стоять первым в списке.

Класс, реализующий интерфейс, может явно реализовывать члены этого интерфейса. При явной реализации члена к нему можно получить доступ только через экземпляр интерфейса, но не через экземпляр класса. Кроме того, обращение к членам интерфейса по умолчанию можно осуществлять только через экземпляр интерфейса.

РЕАЛИЗАЦИЯ ИНТЕРФЕЙСА

В следующем примере показана реализация интерфейса. В этом примере интерфейс содержит объявление свойства, а класс содержит его реализацию. Любой экземпляр класса, который реализует интерфейс IPoint, имеет целочисленные свойства X и Y.

```
interface IPoint
{
  // Сигнатуры свойств:
  int X
  {
    get;
    set;
  }

  int Y
  {
    get;
    set;
  }

  double Distance
  {
     get;
  }
}

class Point : IPoint
{
  // Конструктор:
  public Point(int x, int y)
  {
    X = x;
    Y = y;
  }

  // Реализация свойств:
  public int X { get; set; }

  public int Y { get; set; }
```

```
  // Реализация свойства с помощью анонимной функции.
  public double Distance =>
    Math.Sqrt(X * X + Y * Y);

}

class Program
{
  static void PrintPoint(IPoint p)
  {
    Console.WriteLine("x={0}, y={1}", p.X, p.Y);
  }

  static void Main()
  {
    IPoint p = new Point(2, 3);
    Console.Write("My Point: ");
    PrintPoint(p);
  }
}
```
Результат: My Point: x=2, y=3

УНИВЕРСАЛЬНЫЕ ИНТЕРФЕЙСЫ

Часто требуется определить интерфейсы для универсальных классов коллекций или для универсальных классов, представляющих элементы коллекции. Для универсальных классов предпочтительнее использовать универсальные интерфейсы, такие как IComparable<T>, вместо IComparable, поскольку это позволяет избежать выполнения операция упаковки-преобразования и распаковки-преобразования для типов значений. В библиотеке классов .NET в пространстве имен System.Collections.Generic определяется несколько универсальных интерфейсов для работы с классами коллекций.

СИНТАКСИС ОБЪЯВЛЕНИЯ ОБОБЩЕННОГО ИНТЕРФЕЙСА

В языке C# допускается объявлять обобщенные интерфейсы. При объявлении обобщенного интерфейса сигнатура методов, которые объявлены в интерфейсе, может содержать ссылки на параметризованные типы. Перечень параметризованных типов (например T1,