ИНТЕРФЕЙСЫ

Основные понятия

В ООП иногда требуется определить, что класс должен делать, а не как он будет это делать. Такой подход может быть реализован с помощью абстрактного класса, при этом в абстрактном классе часть методов может быть реализована, а часть нет. Кроме этого в C# предусмотрена возможность полностью отделить структуру класса от его реализации. Это делается с помощью интерфейсов.

Интерфейс – это «крайний» случай абстрактного класса, в котором не предусмотрена ни одна реализация члена класса. Таким образом, интерфейс описывает функциональность классов, но не определяет способа ее реализации. Каждый класс, реализуя интерфейс, может задавать его элементы по-своему. Так достигается полиморфизм – объекты разных классов по-разному реагируют на вызовы одноименного метода.

Синтаксис интерфейса:

```
[атрибуты] [спецификаторы] interface имя_интерфейса : [предки]
{
   //объявление функциональных членов интерфейса без реализации
   …
}
```

Для интерфейса могут использоваться спецификаторы new, public, internal и private. Спецификатор new применяется для вложенных интерфейсов и имеет такой же смысл, как и соответствующий спецификатор метода класса. По умолчанию интерфейс доступен только из сборки, в которой он описан, т.е., по умолчанию используется спецификатор internal.

Все функциональные члены интерфейса по определению являются открытыми (public) и абстрактными (abstract), поэтому при описании метода указывается только тип возвращаемого им значения и сигнатура.

В качестве функциональных членов в интерфейсе можно объявлять сигнатуры методов, свойств, индексаторов и событий. Интерфейсы не могут содержать члены данных, конструкторы, деструкторы и операторные методы (методы, переопределяющие операции). Ни один член интерфейса не может быть объявлен статическим.

Напомним, что класс может наследовать один базовый класс и реализовывать несколько интерфейсов. Класс, реализующий интерфейс, должен делать это в полном объеме, т.е., не может остаться ни одного не реализованного метода. Т.к. функциональные члены, объявленные внутри интерфейса, являются открытыми, то их реализация также должна быть открытой. Кроме того, сигнатура функционального члена в реализации должна в точности совпадать с сигнатурой, заданной в определении интерфейса.

В качестве примера рассмотрим интерфейс IPoint и его реализацию для классов PointPlane и PointSpace.

Описание интерфейса:

```
using System;
namespace MyProgram
{
   public interface IPoint
   {
      void Show();     //объявление метода
```

```
        double Distance();    //объявление метода
        //объявление свойства, доступного только для чтения
        string Name {get;}
        //объявление индексатора, доступного для чтения-записи
        int this [int i]{get;set;}
    }
}
```

Класс PointPlane реализует интерфейс IPoint. Обратите внимание на то, что при определении наследуемых членов спецификатор override не используется.

```
using System;
namespace MyProgram
{
   public class PointPlane: IPoint
   {
      protected int x;
      protected int y;
      readonly string name = "point";
      public PointPlane(int x, int y)
      {
         this.x = x;
         this.y = y;
      }
      public string Name
      {
         get
         {
            return name;
         }
      }
      public void Show()
      {
         Console.WriteLine("({0}, {1})", x, y);
      }
      public double Distance()
      {
         return Math.Sqrt(x*x+y*y);
      }
      public int this [int i]
      {
         get
         {
            if (i == 1)
            {
               return x;
            }
            else
            {
               if (i == 2)
               {
```

```
                    return y;
                }
                else
                {
                    Console.WriteLine("Неверно указан индекс");
                    return 0;
                }
            }
        }
        set
        {
            if (i == 1)
            {
                x = value;
            }
            else
            {
                if (i == 2)
                {
                    y = value;
                }
                else
                {
                    Console.WriteLine("Неверно указан индекс");
                }
            }
        }
    }
  }
}
```

Класс PointSpace наследует класс PointPlane и реализует интерфейс IPoint. Обратите внимание на то, что некоторые члены интерфейса IPoint в классе PointPlane были уже реализованы, но класс PointSpace их переопределяет по-своему, при этом используется спецификатор new.

```
using System;
namespace MyProgram
{
   public class PointSpace : PointPlane, IPoint
   {
      protected int z;
      public PointSpace(int x, int y, int z):base (x, y)
      {
         this.z = z;
      }
      public new void Show()
      {
         Console.WriteLine("({0}, {1}, {2})", x, y, z);
      }
      public new double Distance()
      {
```

```
      return Math.Sqrt(x * x + y * y + z * z);
   }
   public new int this [int i]
   {
      get
      {
         if (i == 1)
         {
            return x;
         }
         else
         {
            if (i == 2)
            {
               return y;
            }
            else
            {
               if (i == 3)
               {
                  return z;
               }
               else
               {
                  Console.WriteLine("Неверно указан индекс");
                  return 0;
               }
            }
         }
      }
      set
      {
         if (i == 1)
         {
            x = value;
         }
         else
         {
            if (i == 2)
            {
               y = value;
            }
            else
            {
               if (i == 3)
               {
                  z = value;
               }
               else
               {
                  Console.WriteLine("Неверно указан индекс");
               }
```

```
            }
         }
      }
   }
}
```

Теперь рассмотрим работу с данными классами. При этом учтем, что для хранения информации об объектах можно использовать массив, тип которого соответствует типу созданного интерфейса.

```
using System;
namespace MyProgram
{
   class Program
   {
      static void Main()
      {
         IPoint [] array = new IPoint[5];
         array[0] = new PointPlane(0, 3);
         array[1] = new PointPlane(4, 3);
         array[2] = new PointSpace(0, 0, 0);
         array[3] = new PointSpace(1, 1, 1);
         array[4] = new PointPlane(-1, -2);
         Console.WriteLine("ИСХОДНЫЕ ДАННЫЕ");
         foreach(IPoint item in array)
         {
            item.Show();
            Console.WriteLine("Расстояние до начала координат:
                              {0:f2}", item.Distance());
            Console.WriteLine();
         }
         array[1][1] = 100;
         array[1][2] = 100;
         array[2][1] = 1;
         array[2][2] = 2;
         array[2][3] = 3;
         Console.WriteLine("ИЗМЕНЕННЫЕ ДАННЫЕ");
         foreach(IPoint item in array)
         {
            item.Show();
            Console.WriteLine("Расстояние до начала координат:
                              {0:f2}", item.Distance());
            Console.WriteLine();
         }
      }
   }
}
```

Результат работы программы:

```
ИСХОДНЫЕ ДАННЫЕ
(0, 3)
```

```
Расстояние до начала координат: 3,00
(4, 3)
Расстояние до начала координат: 5,00
(0, 0, 0)
Расстояние до начала координат: 0,00
(1, 1, 1)
Расстояние до начала координат: 1,73
(-1, -2)
Расстояние до начала координат: 2,24
ИЗМЕНЕННЫЕ ДАННЫЕ
(0, 3)
Расстояние до начала координат: 3,00
(100, 100)
Расстояние до начала координат: 141,42
(1, 2, 3)
Расстояние до начала координат: 3,74
(1, 1, 1)
Расстояние до начала координат: 1,73
(-1, -2)
Расстояние до начала координат: 2,24
```

Задания

1. Добавьте в интерфейс IDemo свойство Y, которое позволит обращаться для чтения к значению поля y. Реализуйте работу с данным свойством в классах DemoPoint и DemoShape.
2. Добавьте свойство Z для обращения к полю z класса DemoShape. Подумайте, куда именно нужно добавить определение данного свойства и почему.

Стандартные интерфейсы .Net. Интерфейс IComparable

В библиотеке классов .Net определено множество стандартных интерфейсов, задающих желаемую функциональность объектов. Например, интерфейс IComparable задает метод сравнения объектов по принципу больше и меньше, что позволяет переопределить соответствующие операции в рамках класса, реализующего данный интерфейс. Реализация интерфейсов IEnumerable и IEnumerator дает возможность просматривать содержимое объекта с помощью оператора foreach.

Можно создавать собственные классы, реализующие стандартные интерфейсы, что позволит использовать объекты этих классов стандартными способами. Более подробно мы рассмотрим стандартный интерфейс IComparable.

Интерфейс IComparable определен в пространстве имен System и содержит единственный метод CompareTo, возвращающий результат сравнения двух объектов: текущего и переданного ему в качестве параметра:

```
interface IComparable
{
   int CompareTo(object obj);
}
```

Реализация данного метода должна возвращать:

1) 0, если текущий объект и параметр равны;
2) отрицательное число, если текущий объект меньше параметра;
3) положительное число, если текущий объект больше параметра.

Использование стандартного интерфейса IComparable рассмотрим на примере модификации класса PointPlane из предыдущего раздела.

```
public class PointPlane: IPoint, IComparable
{
    …
   //добавили в класс реализацию метода CompareTo
   public int CompareTo (object obj)
   {
      IPoint b = (IPoint) obj; //преобразуем параметр к типу IPoint
      //определяем критерии сравнения текущего объекта с параметром
      //в зависимости от удаленности точки от начала координат
      if (this.Distance() == b.Distance())
      {
         return 0;
      }
      else
      {
         if (this.Distance() > b.Distance())
         {
            return 1;
         }
         else
         {
            return -1;
         }
      }
   }
}
```

Обратите внимание на то, что мы изменили только класс PointPlane, класс PointSpace мы не меняли. А теперь рассмотрим следующий фрагмент программы:

```
IPoint[] array = new IPoint[5];
array[0] = new PointPlane(0, 3);
array[1] = new PointPlane(4, 3);
array[2] = new PointSpace(0, 0, 0);
array[3] = new PointSpace(1, 1, 1);
array[4] = new PointPlane(-1, -2);
Array.Sort(array); //вызываем стандартную сортировку массива
foreach(IPoint item in array)
{
   item.Show();
   Console.WriteLine("Расстояние до начала координат: {0:f2}",
                     item.Distance());
   Console.WriteLine();
}
```

Результат работы фрагмента программы

```
(0, 0, 0)
Расстояние до начала координат: 0,00
(1, 1, 1)
Расстояние до начала координат: 1,73
```

```
(-1, -2)
Расстояние до начала координат: 2,24
(0, 3)
Расстояние до начала координат: 3,00
(4, 3)
Расстояние до начала координат: 5,00
```

Обратите внимание на то, что во время реализации метода CompareTo в качестве параметра передавалась ссылка на объект типа object. Напомним, что класс object является корневым классом для всех остальных в C#. Поэтому, он может ссылаться на объект любого типа. Но для того, чтобы потом получить доступ к членам объекта произвольного класса, нужно выполнить приведение типов.

Таким образом, переопределив метод CompareTo, нам удалось отсортировать массив, используя стандартный метод сортировки, определенный в классе Array.

Задания

1. Объясните, почему реализуя метод CompareTo мы привели объектную ссылку к типу IPoint, а не к типу PointPlane.
2. Объясните, почему в классе PointSpace мы не переопределяли метод CompareTo.
3. Измените реализацию метода CompareTo так, чтобы метод Sort сортировал массив точек по убыванию расстояния между точкой и началом координат.

Используя собственную реализацию метода CompareTo можно перегрузить операции отношения. Напомним, что операции отношения должны перегружаться парами: < и >, <= и >=, == и !=.

В следующем примере для класса PointPlane перегрузим операции == и != таким образом, чтобы при сравнении двух объектов возвращалось значение true, если точки находятся на равном удалении от начала координат, и false в противном случае. Для этого в класс PointPlane добавим следующие методы:

```
public static bool operator == (PointPlane a, PointPlane b)
{
   return (a.CompareTo(b) == 0);
}
public static bool operator != (PointPlane a, PointPlane b)
{
   return (a.CompareTo(b) != 0);
}
```

Рассмотрим следующий фрагмент программы:

```
PointPlane a = new PointPlane(0, 3);
PointPlane b = new PointPlane(3, 0);
PointPlane c = new PointPlane(-1, -2);
if (a == b)
{
   Console.WriteLine("точки a и b равноудалены от начала
                      координат");
}
```

```
else
{
   Console.WriteLine("точки a и b находятся на разном расстоянии от
                      начала координат");
}
if (a==c)
{
   Console.WriteLine("точки a и c равноудалены от начала
                      координат");
}
else
{
   Console.WriteLine("точки a и c находятся на разном расстоянии от
                      начала координат");
}
```

Результат работы фрагмента программы:

```
точки a и b равноудалены от начала координат
точки a и c находятся на разном расстоянии от начала координат
```

Однако если мы попытаемся выполнить следующий фрагмент программы:

```
IPoint a = new PointPlane(0, 3);
IPoint b = new PointPlane(3, 0, 0);
if (a == b)
{
   Console.WriteLine("точки a и b равноудалены от начала
                      координат");
}
else
{
   Console.WriteLine("точки a и b находятся на разном расстоянии от
                      начала координат");
}
```

То получим следующий *результат*:

```
точки a и b находятся на разном расстоянии от начала координат
```

Давайте разберемся, что произошло. Дело в том, что операторы проверки на равенстово были перегружены в классе PointPlane. Интерфейс IPoint про них ничего не знает и использует стандартные операторы, которые проверяют равенствно ссылок, а не объектов. Например, если заменить код создания объектов на такой:

```
PointPlane a = new PointPlane(0, 3);
PointPlane b = new PointSpace(3, 0, 0);
```

то сравнение пройдет правильно, так как ссылки на PointPlane уже знают о правилах перегрузки операторов.

Решить эту проблему нам поможет создание следующей иерархии классов:

Рассмотрим каждый компонент этой иерархии.

Абстрактный класс Point:

```
using System;
namespace MyProgram
{
   abstract public class Point:IComparable
   {
      abstract public string Name {get;}
      abstract public void Show();
      abstract public double Distance();
      abstract public int this [int i]{get; set;}
      public int CompareTo (object obj)
      {
         Point b = (Point) obj;
         if (this.Distance() == b.Distance())
         {
            return 0;
         }
         else
         {
            if (this.Distance() > b.Distance())
            {
               return 1;
            }
            else
            {
               return -1;
            }
         }
      }
      public static bool operator == (Point a, Point b)
      {
         return (a.CompareTo(b) == 0);
      }
```

```
        public static bool operator != (Point a, Point b)
        {
           return (a.CompareTo(b) != 0);
        }
     }
  }
```

Класс PointPlane:

```
  using System;
  namespace MyProgram
  {
     public class PointPlane : Point
     {
        protected int x;
        protected int y;
        readonly string name = "point";
        public PointPlane(int x, int y)
        {
           this.x = x;
           this.y = y;
        }
        public override string Name
        {
           get
           {
              return name;
           }
        }
        public override void Show()
        {
           Console.WriteLine("({0}, {1})", x, y);
        }
        public override double Distance()
        {
           return Math.Sqrt(x * x + y * y);
        }
        public override int this [int i]
        {
           get
           {
              if (i == 1)
              {
                 return x;
              }
              else
              {
                 if (i == 2)
                 {
                    return y;
                 }
                 else
```

```
                {
                   Console.WriteLine("Неверно указан индекс");
                   return 0;
                }
             }
          }
          set
          {
             if (i == 1)
             {
                x = value;
             }
             else
             {
                if (i == 2)
                {
                   y = value;
                }
                else
                {
                   Console.WriteLine("Неверно указан индекс");
                }
             }
          }
       }
    }
}
```

Класс PointSpace:

```
using System;
namespace MyProgram
{
   public class PointSpace : PointPlane
   {
      protected int z;
      public PointSpace(int x, int y, int z):base (x, y)
      {
         this.z = z;
      }
      public new void Show()
      {
         Console.WriteLine("({0}, {1}, {2})", x, y, z);
      }
      public new double Distance()
      {
         return Math.Sqrt(x * x + y * y + z * z);
      }
      public new int this [int i]
      {
         get
         {
```

```
            if (i == 1)
            {
               return x;
            }
            else
            {
               if (i == 2)
               {
                  return y;
               }
               else
               {
                  if (i == 3)
                  {
                     return z;
                  }
                  else
                  {
                     Console.WriteLine("Неверно указан индекс");
                     return 0;
                  }
               }
            }
         }
         set
         {
            if (i == 1)
            {
               x = value;
            }
            else
            {
               if (i == 2)
               {
                  y = value;
               }
               else
               {
                  if (i == 3)
                  {
                     z = value;
                  }
                  else
                  {
                     Console.WriteLine("Неверно указан индекс");
                  }
               }
            }
         }
      }
   }
}
```

Рассмотрим, как теперь будут сравниваться между собой разнотипные объекты:

```
Point a = new PointPlane(0, 3);
Point b = new PointSpace(3, 0, 0);
Point c = new PointSpace(3, 0, 1);
if (a == b)
{
   Console.WriteLine("точки a и b равноудалены от начала
                      координат");
}
else
{
   Console.WriteLine("точки a и b находятся на разном расстоянии от
                      начала координат");
}
if (a==c)
{
   Console.WriteLine("точки a и c равноудалены от начала
                      координат");
}
else
{
   Console.WriteLine("точки a и c находятся на разном расстоянии от
                      начала координат");
}
```

Результат работы фрагмента программы:

```
точки a и b равноудалены от начала координат
точки a и c находятся на разном расстоянии от начала координат
```

После рассмотрения последнего примера встает вопрос: когда следует использовать интерфейс, а когда – абстрактный класс?

Если вы полностью описываете действия класса, и не нужно уточнять, как он реализован, следует использовать интерфейс. Если же требуется включить в описание детали реализации, имеет смысл использовать абстрактный класс.

Практикум №13

Замечание

За основу берутся задания из предыдущего практикума.

Задание 1

В абстрактном классе Figure реализовать метод CompareTo так, чтобы можно было отсортировать объекты по их площадям.

Задание 2

В абстрактном классе Function реализовать метод CompareTo так, чтобы можно было отсортировать функции по коэффициенту a.

Задание 3

В абстрактном классе Издание реализовать метод CompareTo так, чтобы можно было отсортировать каталог изданий по фамилии автора.

Задание 4

В абстрактном классе Транспорт реализовать метод CompareTo так, чтобы можно было отсортировать базу данных о машинах по их грузоподъемности.

Задание 5

В абстрактном классе Персона реализовать метод CompareTo так, чтобы можно было отсортировать базу данных о персонах по дате их рождения.

Задание 6

В абстрактном классе Товар реализовать метод CompareTo так, чтобы можно было отсортировать базу данных о товарах по их цене.

Задание 7

В абстрактном классе Товар реализовать метод CompareTo так, чтобы можно было отсортировать базу данных о товарах по возрасту детей, на которых он рассчитан.

Задание 8

В абстрактном классе Телефонный_справочник реализовать метод CompareTo так, чтобы можно было отсортировать базу данных справочника по номеру телефона.

Задание 9

В абстрактном классе Клиент реализовать метод CompareTo так, чтобы можно было отсортировать базу данных о клиентах банка по дате открытия их счета.

Задание 10

В абстрактном классе Программное_обеспечение реализовать метод CompareTo так, чтобы можно было отсортировать базу данных по названию ПО.