ИЕРАРХИЯ КЛАССОВ

Управлять большим количеством разрозненных классов довольно сложно. С этой проблемой можно справиться путем упорядочивания и ранжирования классов, то есть объединяя общие для нескольких классов свойства в одном классе и используя его в качестве базового. Эту возможность предоставляет механизм *наследования.*

Наследование применяется для следующих взаимосвязанных целей:

1) исключения из программы повторяющихся фрагментов кода;
2) упрощения модификации программы;
3) упрощения создания новых программ на основе существующих.

Кроме этого, наследование является единственной возможностью использовать объекты, исходный код которых недоступен, но в которые требуется внести изменения.

Кроме механизма наследования в данном разделе мы рассмотрим такие важные понятия ООП как полиморфизм и инкапсуляцию, которые также принимают участие в формировании иерархии классов.

Наследование

Вспомним синтаксис класса:

```
[атрибуты] [спецификаторы] class имя_класса [: предок]
{
   тело_класса
}
```

При описании класса имя его предка записывается в заголовке класса после двоеточия. Класс, который наследуется, называется базовым. Класс, который наследует, называется производным. Производный класс наследует все переменные, методы, свойства, операторы и индексаторы, определенные в базовом классе, кроме того, в производный класс могут быть добавлены уникальные элементы, или переопределены существующие. Если имя предка явным образом не указано, то предком считается базовый класс всех типов данных в языке C#, т.е., тип object.

Рассмотрим наследование классов на примере геометрических фигур. В качестве базового класса создадим класс PointPlane (точка на плоскости), а в качестве производного класса от PointPlane – класс PointSpace (точка в пространстве):

```
using System;
namespace MyProgram
{
   public class PointPlane //Базовый класс -  точка на плоскости
   {
      public int x;
      public int y;
      public void Show()
      {
         Console.WriteLine("({0}, {1})", x, y);
      }
   }
}
using System;
namespace MyProgram
```

```
{
   //Производный класс - точка в пространстве
   public class PointSpace: PointPlane
   {
      public int z;
      public void Show()
      {
         Console.WriteLine("({0}, {1}, {2})", x, y, z);
      }
   }
}
```

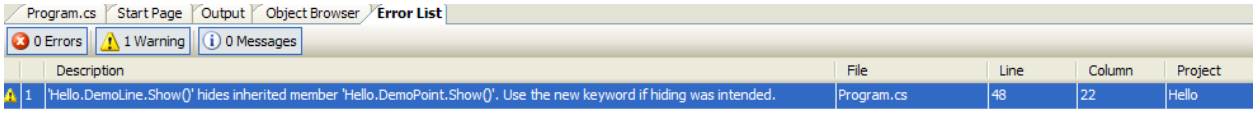
Рассмотрим на примере использование созданных классов:

```
using System;
namespace MyProgram
{
   class Program
   {
      static void Main()
      {
         PointPlane pointP = new PointPlane();
         pointP.x = 10;
         pointP.y = 10;
         pointP.Show();
         PointSpace points = new PointSpace();
         pointS.x = 1;
         pointS.y = 2;
         pointS.z = 3;
         pointS.Show();
      }
   }
}
```

Результат работы программы:

```
(10, 10)
(1, 2, 3)
```

Экземпляр класса PointSpace с одинаковой легкостью использует как собственные поля, так и унаследованные от класса PointPlane. Но в двух классах определен метод Show, поэтому компилятором будет сгенерировано предупреждение:



Чтобы избежать подобного предупреждения, необходимо перед одноименным членом производного класса (в данном случае перед методом Show в классе PointSpace) поставить спецификатор new. Данный спецификатор скрывает одноименный член базового класса от производного, и позволяет полностью его переопределить.

Использование защищенного доступа

В нашем примере поля x и y базового класса были открыты для доступа (public). Если убрать public, то поля автоматически станут закрытыми для доступа (private), в том числе, и для

доступа из производного класса. Решить проблему доступа к закрытым полям базового класса из производного можно двумя способами: используя свойства класса, или спецификатор protected. При объявлении какого-то члена класса с помощью спецификатора protected, он становится закрытым для всех классов, кроме производных.

Наследование конструкторов

В иерархии классов как базовые, так и производные классы могут иметь собственные конструкторы. При этом конструктор базового класса инициализирует часть объекта, соответствующую базовому классу, а конструктор производного класса – часть объекта, соответствующую производному классу. Так как базовый класс не имеет доступа к элементам производного класса, а производный класс – к закрытым полям базового, то их конструкторы должны быть раздельными.

В предыдущем примере классы создавались за счет автоматического вызова средствами C# конструктора по умолчанию. Добавим конструктор только в производный класс PointSpace.

```
using System;
namespace MyProgram
{
   public class PointPlane //Базовый класс
   {
         //поля доступны только из производных классов
         protected int x;
         protected int y;
         public void Show()
         {
            Console.WriteLine("({0}, {1})", x, y);
         }
   }
}

using System;
namespace MyProgram
{
   public class PointSpace:PointPlane
   {
      protected int z;//поле доступно только из производных классов
      //конструктор производного класса
      public PointSpace(int x, int y, int z)
      {
         this.x = x;
         this.y = y;
         this.z = z;
      }
      public new void Show()
      {
         Console.WriteLine("({0}, {1}, {2})", x, y, z);
      }
   }
}
```

В данном случае конструктор определяется только в производном классе, поэтому часть объекта, соответствующая базовому классу, создается автоматически с помощью

конструктора по-умолчанию, а часть объекта, соответствующая производному классу, создается собственным конструктором.

Обратите внимание на то, что для производного класса можно использовать параметр base, который действует подобно параметру this, за исключением того, что base всегда ссылается на базовый класс. В данном контексте параметр base позволяет получить доступ к члену базового класса, который скрыт за членом производного класса. Формат использования параметра:

```
base.член_класса
```

В качестве *член_класса* можно указывать либо метод, либо поле экземпляра.

Рассмотрим использование модифицированных классов на следующем примере:

```
PointPlane pointP = new PointPlane();
pointP.Show();
PointSpace pointS = new PointSpace(1, 2, 3);
pointS.Show();
```

Результат работы фрагмента программы:

```
(0, 0)
(1, 2, 3)
```

Задания

1. *Объясните, почему для объекта pointP были выведены координаты (0, 0).*
2. *Объясните, допустима ли команда pointP.x=1, или PointS.x=1 и почему?*
3. *Добавьте в классы PointPlane и PointSpace функциональные члены, позволяющие осуществлять доступ к закрытым полям.*

Теперь добавим конструктор в базовый класс.

```
using System;
namespace MyProgram
{
   public class PointPlane
   {
      protected int x;
      protected int y;
      public PointPlane(int x, int y) //конструктор базового класса
      {
         this.x = x;
         this.y = y;
      }
      public void Show()
      {
         Console.WriteLine("({0}, {1})", x, y);
      }
   }
}
```

Если мы попытаемся скомпилировать эту программу, то получим сообщение об ошибке. Дело в том, что в .NET объявление в классе конструктора с параметрами автоматически приводит к удалению конструктора по-умолчанию, если он не был явно задан разработчиком.

При этом конструктор класса-потомка автоматически пытается обратиться к конструктору по-умолчанию, которого теперь больше нет.

Для того, чтобы решить эту проблему, необходимо явно указать в конструкторе класса-потомка, какой конструктор класса-родителя необходимо вызвать. Это делается с помощью служебного слова base следующим образом:

```
public PointSpace(int x, int y, int z)
:base (x, y) //конструктор производного класса
{
   this.z = z;
}
```

В данном случае команда base(x, y), стоящая перед телом конструктора производного класса, вызывает конструктор базового класса, передавая ему в качестве параметров список переменных.

Рассмотрим использование модифицированных классов на примере:

```
PointPlane pointP = new PointPlane(10, 10);
pointP.Show();
PointSpace pointS = new PointSpace(1, 2, 3);
pointS.Show();
```

Результат работы фрагмента программы:

```
(10, 10)
(1, 2, 3)
```

Задание

Объясните, допустима ли следующая команда:

```
PointPlane pointP = new PointPlane();
```

и почему?

В общем случае с помощью параметра base можно вызвать конструктор любой формы, определенный в базовом классе. Реально же выполнится тот конструктор, список формальных параметров которого будет соответствовать списку аргументов, переданных base.

Рассмотрим модификацию классов PointPlane и PointSpace, содержащих по несколько конструкторов:

```
using System;
namespace MyProgram
{
   public class PointPlane
   {
      protected int x;
      protected int y;
      public PointPlane()
      {
      }
      public PointPlane(int a)
      {
         x = a;
```

```
            y = a;
         }
         public PointPlane(int x, int y)
         {
            this.x = x;
            this.y = y;
         }
         public void Show()
         {
            Console.WriteLine("({0}, {1})", x, y);
         }
      }
   }

   using System;
   namespace MyProgram
   {
      public class PointSpace:PointPlane
      {
         protected int z;
         public PointSpace()
         {
         }
         public PointSpace(int a)
         :base (a)
         {
            z = a;
         }
         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);
         }
      }
   }
```

Инициируем вызов различных конструкторов, и посмотрим, что из этого получится:

```
PointPlane pointP1 = new PointPlane();
pointP1.Show();
PointPlane pointP2 = new PointPlane(1);
pointP2.Show();
PointPlane pointP3 = new PointPlane(2, 3);
pointP3.Show();
PointSpace pointS1 = new PointSpace();
pointS1.Show();
PointSpace pointS2 = new PointSpace(4);
```

```
pointS2.Show();
PointSpace pointS3 = new PointSpace(5, 6, 7);
pointS3.Show();
```

Результат работы фрагмента программы:

```
(0, 0)
(1, 1)
(2, 3)
(0, 0, 0)
(4, 4, 4)
(5, 6, 7)
```

Задание

Объясните, как при вызове конструктора производного класса инициируется вызов конструктора базового класса.

Класс object

Все типы языка C#, включая размерные типы, унаследованы от класса object. Следовательно, ссылку типа object можно использовать в качестве ссылки на любой другой тип, в том числе, размерный.

Если ссылка типа object указывает на значение нессылочного типа, то происходит приведение к ссылочному типу (boxing). В результате этого процесса значение нессылочного типа сохраняется в динамической памяти, подобно любому экземпляру класса. Другими словами, "необъектное" значение помещается в объектную оболочку и размещается в динамической памяти. Такой "необъектный" объект можно затем использовать подобно любому другому объекту. Приведение к объектному типу происходит автоматически.

Восстановление значения из "объектного образа" называется unboxing. Это действие выполняется с помощью операции приведения типа, т.е., приведения ссылки на объект класса object к значению желаемого типа.

Рассмотрим простой пример, который иллюстрирует приведение значения к объектному типу и его восстановление.

```
int x = 1;
object ob = x;    //boxing
int y = (int) ob;  //unboxing
Console.WriteLine("y={0}", y);
```

Результат работы фрагмента программы:

```
y=1
```

Так как C# является языком со строгой типизацией, в нем требуется строгое соблюдение совместимости типов (с учетом стандартных преобразований). Поэтому ссылка одного типа обычно не может ссылаться на объект другого ссылочного типа, за одним небольшим исключением – ссылочная переменная базового класса может ссылаться на объект любого производного класса.

В нашем случае допустимой будет следующая команда:

```
PointPlane pointS = new PointSpace(1,2,3);
```

Более того, т.к. тип object является базовым для всех типов данных, становится допустимой следующая последовательность команд:

```
object ob1 = new PointPlane(4, 5);
object ob2 = new PointSpace(6, 7, 8);
```

Ошибка возникнет при попытке обратиться к методу Show. Например, команда

```
pointS.Show();
```

вместо ожидаемого (1, 2, 3) выведет нам (1, 2). А команда:

```
ob1.Show();
```

вообще не сможет выполниться, т.к. для класса object метод Show не определен.

Давайте разберемся, почему вместо (1, 2, 3) мы получили (1, 2). Как уже упоминалось ранее, все методы класса находятся в памяти в единственном экземпляре и используются всеми объектами одного класса совместно. Это решение вполне логично, так как если у нас есть 1000 объектов типа PointPlane, то 1000 раз дублировать код метода Show нет никакого смысла.

В результате, в случае с присваиванием ссылке на класс PointPlane объекта PointSpace возникает конфликт между двумя таблицами методов – базового класса и класса-потомка.

Так как потомков у базового класса может быть много (в том числе потомков третьего и более высоких уровней) и поиск по всем возможным таблицам методов является достаточно длительным процессом, компилятор считает, что, если не сказано обратное, таблица, в которой производится поиск метода, определяется типом ссылки на объект. Для того чтобы явно указать на необходимость поиска нужного метода по типу объекта, а не ссылки, используются спецификаторы virtual и abstract, а обозначенные ими методы принято называть виртуальными и абстрактными соответственно.

Виртуальные методы

Виртуальный метод – это метод, который объявлен в базовом классе с использованием ключевого слова virtual, а затем переопределен в производном классе с помощью ключевого слова override. При этом, если реализовано наследование, в том числе и многоуровневое, то каждый производный класс может иметь свою собственную версию виртуального метода.

Этот факт особенно полезен в случае, когда доступ к объекту производного класса осуществляется через ссылочную переменную базового класса. В этой ситуации CLR сама выбирает, какую версию виртуального метода нужно вызвать. Этот выбор производится по типу объекта, на который ссылается данная ссылка.

Модифицируем методы Show в классах PointPlane и PointSpace с учетом сказанного.

```
public virtual void Show() //в классе PointPlane
{
   Console.WriteLine("({0}, {1})", x, y);
}
public override void Show()//в классе PointSpace
{
   Console.WriteLine("({0}, {1}, {2})", x, y, z);
}
```

Теперь рассмотрим следующий фрагмент программы:

```
PointPlane []array = new PointPlane [6];
array[0] = new PointPlane();
array[1] = new PointPlane(1);
```

```
array[2] = new PointPlane(2, 3);
array[3] = new PointSpace();
array[4] = new PointSpace(4);
array[5] = new PointSpace(5, 6, 7);
foreach (PointPlane item in array)
{
   item.Show();
}
```

Результат работы фрагмента программы:

```
(0, 0)
(1, 1)
(2, 3)
(0, 0, 0)
(4, 4, 4)
(5, 6, 7)
```

Таким образом, благодаря полиморфизму, через ссылочную переменную базового класса можно обращаться к объектам разного типа, а также с помощью одного и того же имени выполнять различные действия.

Задание

Добавьте в базовый класс PointPlane виртуальный метод Distance, позволяющий вычислить расстояние от начала координат до заданной точки. Переопределите его в производном классе PointSpace с учетом того, что точка задана в пространстве. Продемонстрируйте работу данного метода.

Абстрактные методы и классы

Иногда полезно создать базовый класс, определяющий только своего рода "пустой бланк", который унаследуют все производные классы, причем каждый из них заполнит этот "бланк" собственной информацией. Такой класс определяет структуру методов, которые производные классы должны реализовать, но сам класс при этом не обеспечивает реализации этих методов. Подобная ситуация может возникнуть тогда, когда базовый класс попросту не в состоянии реализовать метод. В данной ситуации разрабатываются *абстрактные методы*, или целые *абстрактные классы*.

Абстрактный метод описывается с помощью спецификатора abstract. Он не имеет тела и, следовательно, не реализуется базовым классом, а производные классы должны его обязательно переопределить. Абстрактный метод автоматически является виртуальным и использовать спецификатор virtual не нужно. Более того, если вы попытаетесь использовать оба спецификатора (abstract и virtual) одновременно, то компилятор выдаст сообщение об ошибке.

Задание

Подумайте, можно ли спецификатор abstract сочетать со спецификатором static. И почему?

Если класс содержит хотя бы один абстрактный метод, его также нужно объявить как абстрактный, используя спецификатор abstract перед class. Поскольку абстрактный класс полностью не реализован, то невозможно создать экземпляр класса с помощью операции

new. Например, если класс Demo определен как абстрактный, то попытка создать экземпляр класса Demo повлечет ошибку:

```
Demo a = new Demo();
```

Однако можно создать массив ссылок, тип которых соответствует абстрактному классу:

```
Demo [] Ob = new Demo[5];
```

Если производный класс наследует абстрактный, то он должен полностью переопределить все абстрактные методы базового класса, или также быть объявлен как абстрактный. Таким образом, спецификатор abstract наследуется до тех пор, пока в производном классе не будут реализованы все абстрактные методы.

Рассмотрим пример использования абстрактных методов и классов.

```
using System;
namespace MyProgram
{
   abstract public class Point //абстрактный класс
   {
      abstract public void Show();
      abstract public double Distance();
   }
}
```

Класс Point содержит объявление двух абстрактных методов, которые при наследовании необходимо будет реализовать. В общем случае абстрактный класс может содержать не только абстрактные методы. Используя абстрактный класс Point, модифицируем классы PointPlane и PointSpace.

```
using System;
namespace MyProgram
{
   public class PointPlane: Point
   {
      protected int x;
      protected int y;
      public PointPlane()
      {
      }
      public PointPlane(int a)
      {
         x = a;
         y = a;
      }
      public PointPlane(int x, int y)
      {
         this.x = x;
         this.y = y;
      }
      //переопределяем абстрактный метод
      public override void Show()
      {
```

```
            Console.WriteLine("({0}, {1})", x, y);
         }
         //переопределяем абстрактный метод
         public override double Distance()
         {
            return Math.Sqrt(x*x+y*y);
         }
      }
   }

   using System;
   namespace MyProgram
   {
      public class PointSpace:PointPlane
      {
         protected int z;
         public PointSpace()
         {
         }
         public PointSpace(int a):base (a)
         {
            z = a;
         }
         public PointSpace(int x, int y, int z) :base (x, y)
         {
            this.z = z;
         }
         //переопределяем абстрактный метод
         public override void Show()
         {
            Console.WriteLine("({0}, {1}, {2})", x, y, z);
         }
         //переопределяем абстрактный метод
         public override double Distance()
         {
            return Math.Sqrt(x*x + y*y + z*z);
         }
      }
   }
```

Теперь рассмотрим следующий фрагмент программы:

```
Point []array = new Point[6];
array[0] = new PointPlane();
array[1] = new PointPlane(1);
array[2] = new PointPlane(2, 3);
array[3] = new PointSpace();
array[4] = new PointSpace(4);
array[5] = new PointSpace(5, 6, 7);
foreach (Point item in array)
{
```

```
        item.Show();
        Console.WriteLine("Расстояние до начала координат: {0:f2}",
                          item.Distance());
        Console.WriteLine();
    }
```

Результат работы фрагмента программы:

```
(0, 0)
Расстояние до начала координат: 0.00
(1, 1)
Расстояние до начала координат: 1.41
(2, 3)
Расстояние до начала координат: 3.61
(0, 0, 0)
Расстояние до начала координат: 0.00
(4, 4, 4)
Расстояние до начала координат: 6.93
(5, 6, 7)
Расстояние до начала координат: 10.49
```

Задание

Добавьте в класс Point абстрактный метод, позволяющий определить, является ли объект-точка началом координат. Реализуйте данный метод в классах PointPlane и PointSpace и продемонстрируйте его работу.

Запрет наследования

В C# при описании класса может использоваться спецификатор sealed, который запрещает производить наследование от данного класса. Например:

```
sealed class Demo {   … }
class newDemo: Demo {   …  }  // ошибка
```

Задание

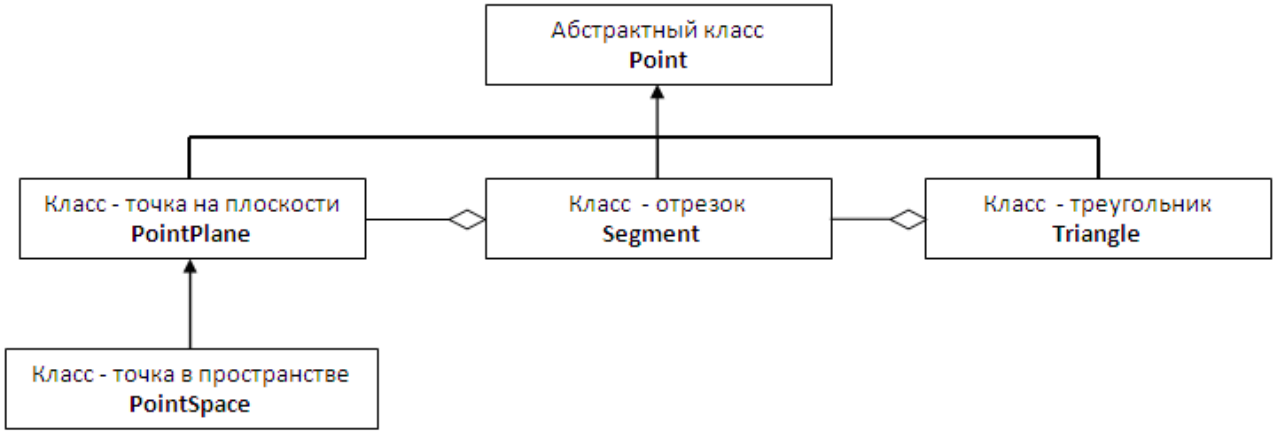
Подумайте:

1. *Для чего может создаваться класс, от которого нельзя наследовать?*
2. *Можно ли использовать сочетание спецификаторов sealed и abstract при описании класса, и почему?*

Многоуровневая иерархия

До сих пор мы рассматривали простой тип иерархии классов, который состоит из одного базового и одного производного класса. В общем случае можно построить иерархию классов, состоящую из любого количества уровней наследования.

Рассмотрим структуру следующей иерархии классов:



Стрелки с окончанием в виде черного треугольника показывают направление наследования, а стрелки с белым ромбом на конце – агрегацию, т.е., говорят о том, что поля класса Segment будут являться ссылками на объекты класса PointPlane, а поля класса Triangle, в свою очередь, будут являться ссылками на объекты класса Segment.

Рассмотрим реализацию каждого класса.

Класс Point:

```
using System;
namespace MyProgram
{
   abstract public class Point
   {
      abstract public string Name();   //возвращает имя объекта
      abstract public void Show();     //выводит объект на экран
      abstract public double Distance();//рассчитывает длину
      //устанавливает поля объекта
      abstract public void Set(params int[] ar);
   }
}
```

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 ()
      {
         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 void Set(params int [] ar)
        {
           if (ar.Length == 2)
           {
              this.x = ar[0];
              this.y = ar[1];
           }
           else
           {
              Console.WriteLine("Неверное количество аргументов");
           }
        }
        public int X
        {
           get
           {
              return x;
           }
           set
           {
              x = value;
           }
        }
        public int Y
        {
           get
           {
              return y;
           }
           set
           {
              y = value;
           }
        }
     }
  }
```

Класс PoinSpace:

```
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 override void Show()
   {
      Console.WriteLine("({0}, {1}, {2})", x, y, z);
   }
   public override double Distance()
   {
      return Math.Sqrt(x * x + y * y + z * z);
   }
   public override void Set(params int [] ar)
   {
      if (ar.Length == 3)
      {
         this.x = ar[0];
         this.y = ar[1];
         this.z = ar[2];
      }
      else
      {
         Console.WriteLine("Неверное количество аргументов");
      }
   }
   public int Z
   {
      get
      {
         return z;
      }
      set
      {
         z = value;
      }
   }
}
```

Класс Segment:

```
using System;
namespace MyProgram
{
   class Segment : Point
   {
      internal PointPlane begin;
      internal PointPlane end;
```

```
readonly string name = "segment";
public Segment(int x1, int y1, int x2, int y2)
{
   begin = new PointPlane(x1,y1);
   end = new PointPlane(x2, y2);
}
public override string Name ()
{
   return name;
}
public override void Show ()
{
   Console.Write("Начало отрезка");
   begin.Show();
   Console.Write("Конец отрезка");
   end.Show();
}
public override double Distance ()
{
   return Math.Sqrt(Math.Pow(begin.X - end.X,2) +
                    Math.Pow(begin.Y - end.Y, 2));
}
public override void Set(params int [] ar)
{
   if (ar.Length == 4)
   {
      begin.Set(ar[0], ar[1]);
      end.Set(ar[2], ar[3]);
   }
   else
   {
      Console.WriteLine("Неверное количество аргументов");
   }
}
public PointPlane Begin
{
   get
   {
      return begin;
   }
   set
   {
      begin = value;
   }
}
public PointPlane End
{
   get
   {
      return end;
```

```
            }
            set
            {
               end = value;
            }
         }
      }
   }
```

Класс Triangle:

```
using System;
namespace MyProgram
{
   public class Triangle:Point
   {
      internal Segment sideA;
      internal Segment sideB;
      internal Segment sideC;
      readonly string name = "triangle";
      public Triangle(int x1, int y1, int x2, int y2,
                      int x3, int y3)
      {
         sideA = new Segment(x1, y1, x2, y2);
         sideB = new Segment(x2, y2, x3, y3);
         sideC = new Segment(x3, y3, x1, y1);
      }
      public override string Name()
      {
         return name;
      }
      public override void Show()
      {
         Console.Write("Первая вершина");
         sideA.Begin.Show();
         Console.Write("Вторая вершина");
         sideB.Begin.Show();
         Console.Write("Третья вершина");
         sideC.Begin.Show();
      }
      public override double Distance()
      {
         return sideA.Distance()+sideB.Distance()+sideC.Distance();
      }
      public override void Set(params int[] ar)
      {
         if (ar.Length == 6)
         {
            sideA.Set(ar[0], ar[1], ar[2], ar[3]);
            sideB.Set(ar[2], ar[3], ar[4], ar[5]);
            sideC.Set(ar[4], ar[5], ar[0], ar[1]);
         }
```

```
            else
            {
               Console.WriteLine("Неверное количество аргументов");
            }
         }
      }
   }
```

Используя разработанные классы, решим задачу. Пусть в текстовом файле хранится информация: в первой строке – количество объектов (N); в последующих строках – данные об N объектах, в качестве которых могут быть точка на плоскости (задается двумя координатами x, y), точка в пространстве (задается тремя координатами x, y, z), отрезок (задается координатами начала и конца отрезка x1, y1 и x2, y2 соответственно) и треугольник на плоскости (задается координатами трех вершин: x1, y1, x2, y2, x3 и y3 соответственно). Например:

text.txt	Соответствие
8	N
0 0	данные о точке на плоскости
1 2 3	данные о точке в пространстве
4 5 6 7	данные об отрезке на плоскости
0 4 3 0 -3 0	данные о треугольнике
8 9	данные о точке на плоскости
0 1 2 3	данные об отрезке на плоскости
4 5 6	данные о точке на плоскости
0 0 3 4 0 4	данные о треугольнике

Необходимо считать данные об объектах из файла и вывести полную информацию о них на экран.

```
using System;
using System.IO;

namespace MyProgram
{
   class Program
   {
      static public Point[] Input() //читаем данные из файла
      {
         using (StreamReader fileIn =
                     new StreamReader ("d:/Example/text.txt"))
         {
            int n = int.Parse( fileIn.ReadLine());
            Point[] ar = new Point[n];
            for (int i = 0; i < n; i++)
            {
               string[] text = fileIn.ReadLine().Split(' ');
               if (text.Length == 2)
               {
                  ar[i] = new PointPlane(int.Parse(text[0]),
                                         int.Parse(text[1]));
               }
               else
               {
```

```
                if (text.Length == 3)
                {
                   ar[i] = new PointSpace(int.Parse(text[0]),
                                          int.Parse(text[1]),
                                          int.Parse(text[2]));
                }
                else
                {
                   if (text.Length == 4)
                   {
                      ar[i] = new Segment(int.Parse(text[0]),
                                          int.Parse(text[1]),
                                          int.Parse(text[2]),
                                          int.Parse(text[3]));
                   }
                   else
                   {
                      ar[i] = new Triangle(int.Parse(text[0]),
                                           int.Parse(text[1]),
                                           int.Parse(text[2]),
                                           int.Parse(text[3]),
                                           int.Parse(text[4]),
                                           int.Parse(text[5]));
                   }
                }
             }
          }
          return ar;
       }
    }
    static void Print (Point []array) //выводим данные на экран
    {
       foreach (Point item in array)
       {
          item.Show();
          switch (item.Name())
          {
             case "point":
                Console.WriteLine("Расстояние до начала координат
                                   {0:f2}", item.Distance());
                break;
             case "line":
                Console.WriteLine("Длина отрезка {0:f2}",
                                   item.Distance());
                break;
             case "triangle":
                Console.WriteLine("Периметр треугольника:
                                   {0:f2}",item.Distance());
                break;
          }
          Console.WriteLine();
       }
```

```
        }
        static void Main()
        {
           Point[] array = Input();
           Print(array);
        }
     }
  }
```

Результат работы программы:

```
(0, 0)
Расстояние до начала координат: 0,00
(1, 2, 3)
Расстояние до начала координат: 3,74
Начало отрезка: (4, 5)
Конец отрезка: (6, 7)
Длина отрезка: 2,83
Первая вершина: (0, 4)
Вторая вершина: (3, 0)
Третья вершина (-3, 0)
Периметр треугольника: 16,00
(8, 9)
Расстояние до начала координат:12,04
Начало отрезка: (0, 1)
Конец отрезка: (2, 3)
Длина отрезка: 2,83
(4, 5, 6)
Расстояние до начала координат: 8,77
Первая вершина: (0, 0)
Вторая вершина: (3, 4)
Третья вершина (0, 4)
Периметр треугольника: 12,00
```


Задания

1. Измените программу так, чтобы на экран выводились данные только о точках.
2. Задокументируйте созданные классы.

Практикум №12

Замечания

1. *Полную структуру классов и их взаимосвязь продумать самостоятельно.*
2. *Для абстрактного класса определить, какие методы должны быть абстрактными, а какие обычными.*
3. *Исходные данные считывать из файла.*

Задание 1

1. Создать абстрактный класс Figure с методами вычисления площади и периметра, а также методом, выводящим информацию о фигуре на экран.
2. Создать производные классы: Rectangle (прямоугольник), Circle (круг), Triangle (треугольник) со своими методами вычисления площади и периметра.
3. Создать массив из n фигур и вывести полную информацию о фигурах на экран.

Задание 2

1. Создать абстрактный класс Function с методом вычисления значения функции y=f(x) в заданной точке.
2. Создать производные классы: Line (y=ax+b), Kub ($y=ax^2+bx+c$), Hyperbola ($y = \frac{a}{x} + b$) со своими методами вычисления значения в заданной точке.
3. Создать массив n функций и вывести полную информацию о значении данных функций в точке x.

Задание 3

1. Создать абстрактный класс Издание с методами, позволяющим вывести на экран информацию об издании, а также определить, является ли данное издание искомым.
2. Создать производные классы: Книга (название, фамилия автора, год издания, издательство), Статья (название, фамилия автора, название журнала, его номер и год издания), Электронный ресурс (название, фамилия автора, ссылка, аннотация) со своими методами вывода информации на экран.
3. Создать каталог (массив) из n изданий, вывести полную информацию из каталога, а также организовать поиск изданий по фамилии автора.

Задание 4

1. Создать абстрактный класс Транспорт с методами, позволяющими вывести на экран информацию о транспортном средстве, а также определить его грузоподъемность.
2. Создать производные классы: Легковая_машина (марка, номер, скорость, грузоподъемность), Мотоцикл (марка, номер, скорость, грузоподъемность, наличие коляски; при этом, если коляска отсутствует, то грузоподъемность равна 0), Грузовик (марка, номер, скорость, грузоподъемность, наличие прицепа; при этом, если есть прицеп, то грузоподъемность увеличивается в два раза) со своими методами вывода информации на экран, и определения грузоподъемности.
3. Создать базу (массив) из n машин, вывести полную информацию из базы на экран, а также организовать поиск машин, удовлетворяющих требованиям грузоподъемности.

Задание 5

1. Создать абстрактный класс Персона с методами, позволяющим вывести на экран информацию о персоне, а также определить ее возраст (на момент текущей даты).
2. Создать производные классы: Абитуриент (фамилия, дата рождения, факультет), Студент (фамилия, дата рождения, факультет, курс), Преподаватель (фамилия, дата рождения, факультет, должность, стаж), со своими методами вывода информации на экран, и определения возраста.
3. Создать базу (массив) из n персон, вывести полную информацию из базы на экран, а также организовать поиск персон, чей возраст попадает в заданный диапазон.

Задание 6

1. Создать абстрактный класс Товар с методами, позволяющими вывести на экран информацию о товаре, а также определить, соответствует ли он сроку годности на текущую дату.
2. Создать производные классы: Продукт (название, цена, дата производства, срок

годности), Партия (название, цена, количество, дата производства, срок годности), Комплект (название, цена, перечень продуктов) со своими методами вывода информации на экран, и определения соответствия сроку годности.

3. Создать базу (массив) из n товаров, вывести полную информацию из базы на экран, а также организовать поиск просроченного товара (на момент текущей даты).

Задание 7

1. Создать абстрактный класс Товар с методами, позволяющими вывести на экран информацию о товаре, а также определить, соответствует ли она искомому типу.
2. Создать производные классы: Игрушка (название, цена, производитель, материал, возраст, на который рассчитана), Книга (название, автор, цена, издательство, возраст, на который рассчитана), Спорт-инвентарь (название, цена, производитель, возраст, на который рассчитан) со своими методами вывода информации на экран и определения соответствия искомому типу.
3. Создать базу (массив) из n товаров, вывести полную информацию из базы на экран, а также организовать поиск товаров определенного типа.

Задание 8

1. Создать абстрактный класс Телефонный_справочник с методами, позволяющими вывести на экран информацию о записях в телефонном справочнике, а также определить соответствие записи критерию поиска.
2. Создать производные классы: Персона (фамилия, адрес, номер телефона), Организация (название, адрес, телефон, факс, контактное лицо), Друг (фамилия, адрес, номер телефона, дата рождения) со своими методами вывода информации на экран и определения соответствия искомому типу.
3. Создать базу (массив) из n записей, вывести полную информацию из базы на экран, а также организовать поиск в базе по фамилии.

Задание 9

1. Создать абстрактный класс Клиент с методами, позволяющими вывести на экран информацию о клиентах банка, а также определить соответствие клиента критерию поиска.
2. Создать производные классы: Вкладчик (фамилия, дата открытия вклада, размер вклада, процент по вкладу), Кредитор (фамилия, дата выдачи кредита, размер кредита, процент по кредиту, остаток долга), Организация (название, дата открытия счета, номер счета, сумма на счету) со своими методами вывода информации на экран и определения соответствия дате (открытия вклада, выдаче кредита, открытия счета).
3. Создать базу (массив) из n клиентов, вывести полную информацию из базы на экран, а также организовать поиск клиентов, начавших сотрудничать с банком с заданной даты.

Задание 10

1. Создать абстрактный класс Программное_обеспечение с методами, позволяющими вывести на экран информацию о программном обеспечении, а также определить соответствие возможности использования (на момент текущей даты).
2. Создать производные классы: Свободное (название, производитель), Условно-бесплатное (название, производитель, дата установки, срок бесплатного использования), Коммерческое (название, производитель, цена, дата установки, срок использования) со

своими методами вывода информации на экран и определения возможности использования на текущую дату.

3. Создать базу (массив) из n видов программного обеспечения, вывести полную информацию из базы на экран, а также организовать поиск программного обеспечения, которое допустимо использовать на текущую дату.