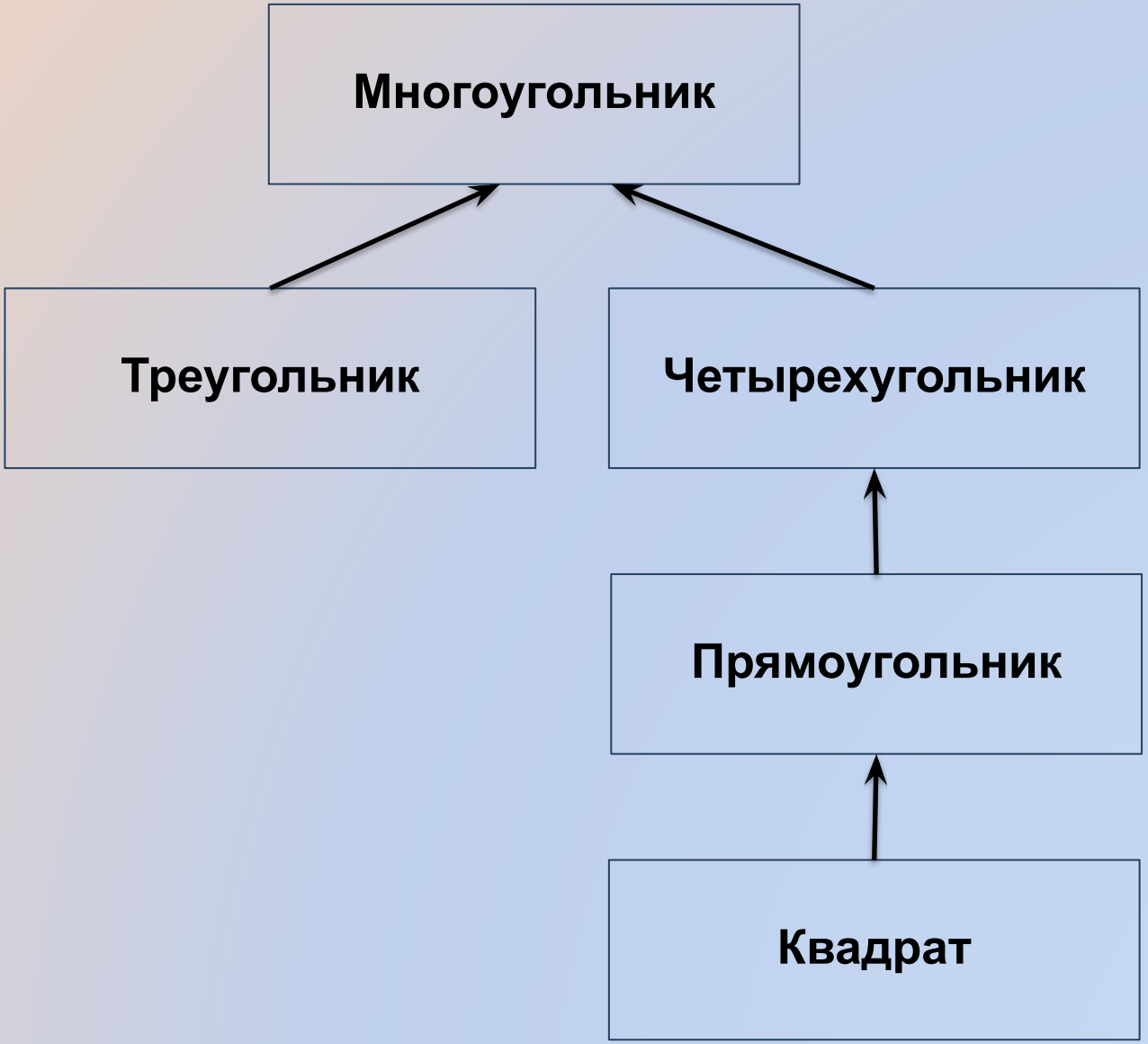
Лекция 8. Наследование: базовые понятия и примеры

Половикова О.Н.

Опеределение

- Это принцип **создание класса на базе уже существующего**, при этом у нас есть **возможность пользоваться функционалом (свойствами и методами) базового**. Классы созданные таким образом называются производными или дочерними, а на базе которого создаются — родителем или базовым.
- Этот механизм в объектно ориентированном программировании имеет большое значение: так как в несколько раз экономит время на создание проекта, а также не нагружает его повторяющимся кодом.

Схема


Многоугольник
Треугольник
Четырехугольник
Прямоугольник
Квадрат

Способы передачи функциональности

Агрегирование

Многоугольник

- Отрезок [ ] a;
- Вершина[ ] b;

Отрезок

Вершина

Полями или свойствами класса являются объекты других классов (ссылки на объекты)

Наследование

Синтаксис объявления наследования

class имя_производного_класса : имя_базового_класса {
// тело класса
}

```
class Shape {
        public double Width;
        public double Height;
        public void Show() {
        Console.WriteLine("Ширина и высота равны " + Width + " и " + Height);
        }
 }
```

```
// Класс Triangle, производный от класса Shape
class Triangle : Shape {
public string Style; // тип треугольника
public double Area() =>return Width * Height / 2;  // Возвратить площадь треугольника
// Показать тип треугольника.
public void ShowStyle() =>Console.WriteLine("Треугольник " + Style);
 }
```

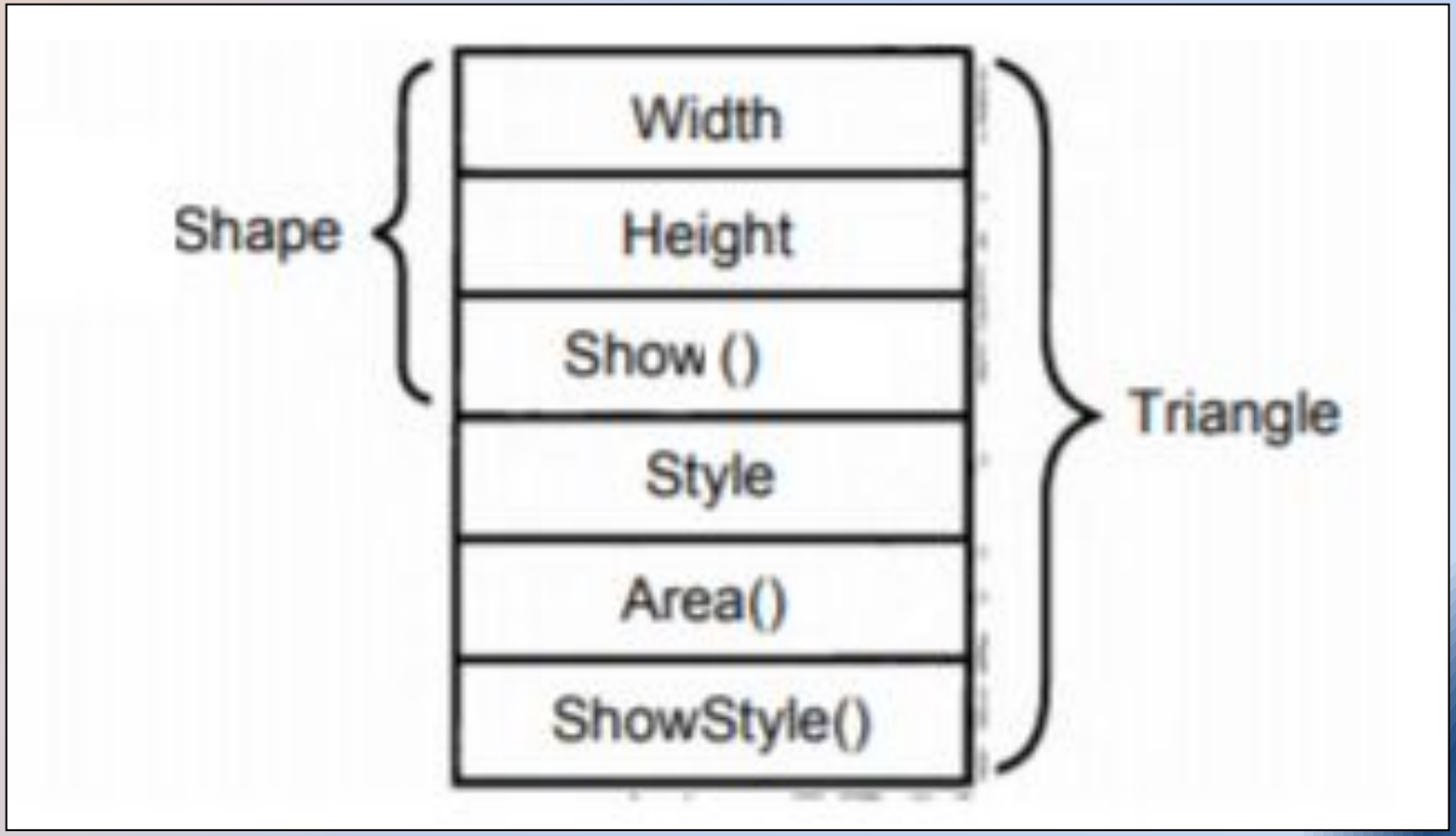
Создание объекта класса *наследника*

```
static void Main() {
//создание объекта с инициализатором
Triangle A = new Triangle(){ Width = 4.0, Height = 4.0, Style = "равнобедренный“};
Triangle  B = new Triangle(){Width = 8.0, Height = 12.0, Style ="прямоугольный"};

A.ShowStyle(); // свой метод
Show(); //унаследованный метод
Console.WriteLine("Площадь равна " + A.Area());// свой метод
```

Класс Triangle это особый тип класса Shape. В класс Triangle входят все члены класса Shape (не все доступны, но входят все), к которым, добавляются методы Area() и ShowStyle(). Добавляется ещё поле, отвечающее за тип треугольника (Style), метод Area(): вычисляет площадь.

Что есть в производном классе?

В класс Triangle входят все члены его базового класса Shape: поля Width и Height доступны для метода Area(). Объекты A и B в методе Main() могут обращаться непосредственно к переменным Width и Height, словно они являются членами класса Triangle.

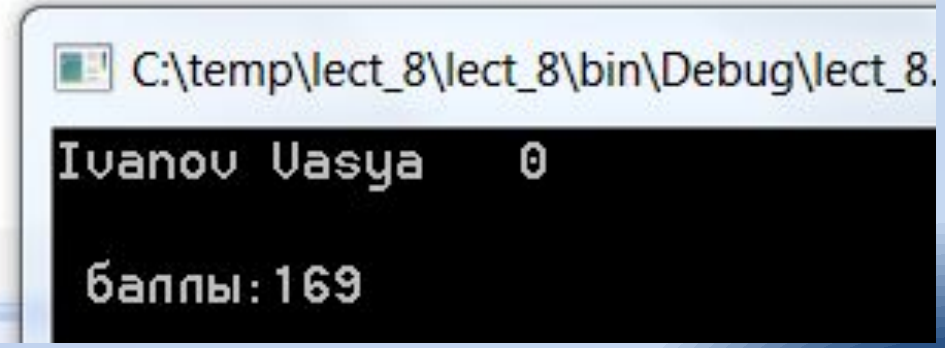


Пример наследования

```
class Person{
    private string name;
    protected long weight;
    public string Name{
        get { return name; }
        set { name = value; }
    }
    public void Print(){
        Console.WriteLine(Name+ "    "+weight.ToString());
    }
}
    class Student : Person{
        int[] balls = null;
        public int SumBalls{
            get { int sum = 0;
                if (balls != null)
                    for (int i= 0; i< balls.Length; i++)
                        sum += balls[i];
                return sum;
                }
        }
        public void Set(params int[] a){

            if (a != null) {
                balls = new int[a.Length];
                a.CopyTo(balls, 0);
            }
        }
```

Пример наследования

```
class Program
{
    public static void Main(string[] args)
    {
        Student A = new Student();
        A.Name = "Ivanov Vasya";
        A.Print();
        A.Set(12, 23, 56, 78);
        Console.Write("\n баллы:"+A.SumBalls.ToString());
        Console.ReadKey(true);
    }
}
```

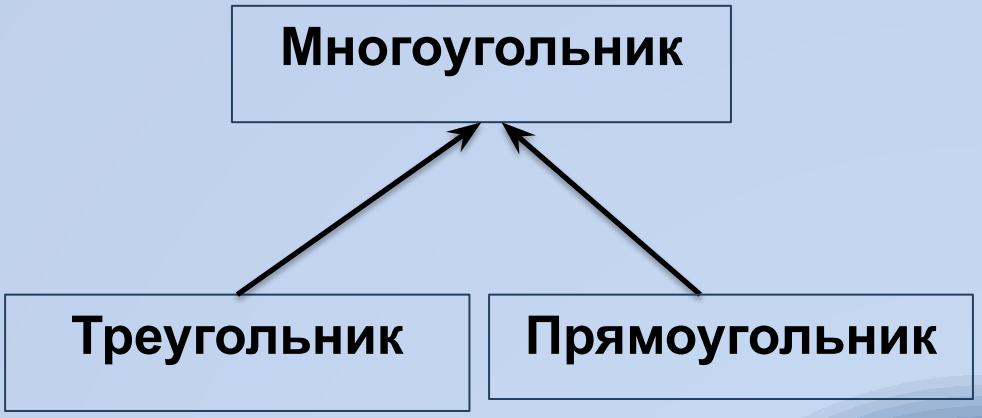


Несколько наследников

Преимущество наследования: как только будет создан базовый класс, в котором определены общие для множества объектов свойства, поля, методы, этот класс можно использовать для создания любого числа более конкретных производных классов (наследников).

В каждом производном классе может быть также выстроена своя собственная классификация (иерархия наследования).

```
class Rectangle : Shape {
// Возвратить логическое значение true, если
// прямоугольник является квадратом
public bool IsSquare() {
if(Width == Height) return true;
return false;
}
// Возвратить площадь прямоугольника
 public double Area() =>return Width * Height;
```



Закрытые члены классы

Закрытый член класса остается закрытым в своем классе. Он не доступен из кода за пределами своего класса, включая и производные классы (наследники).

Такое ограничение (на первый взгляд) является трудно преодолимым, поскольку оно не даёт во многих случаях возможности пользоваться унаследаванными закрытыми членами.

Для преодоления данного ограничения в C# предусмотрены разные способы. **Один из них состоит в использовании защищенных (protected) членов класса**, **а второй — в применении открытых свойств для доступа к закрытым данным**.

Закрытые члены классы

Родительский класс (базовый)

```
public enum places: int{Барнаул = 1, Новоалтайск, Заринск, посёлок, другое = 0};
//public enum places1: char{Барнаул = 'Б', Новоалтайск='Н', Заринск = 'З', другое = '-'};
class Person{
    private string name;
    protected double weight = 0.0;// защищённый
    protected places address = places.другое;// защищённый
    public string Name{ // св-во к закрытому полю
        get { return name; }
        set { name = value; }
    }
    public void Print(){
        Console.WriteLine("--------------print--------------------");
        Console.WriteLine(Name+ "    "+weight.ToString());
        Console.WriteLine("Address: {0}:({1})", address, (int)address);
    }
    public void SetAddress(places a){
        address = a;
    }
}
```

Закрытые члены классы

Дочерний класс (производный)

```
class Student : Person{
    int[] balls = null;

    public int SumBalls{// сво-во для закрытого поля
        get {
            int sum = 0;
            if (balls != null)
                for (int i= 0; i< balls.Length; i++)
                    sum += balls[i];
            return sum;
            }
    }
    public void Set(params int[] a){

        foreach (var x in Enum.GetValues(typeof(places)) ){ //перебор значений
            Console.Write("\n" + x + " 1/0:");
            if (Console.ReadLine() == "1"){
                address = (places)x; //изменяем защищённое поле
                break;
            }
        }
        if (a != null) {
            balls = new int[a.Length];
            a.CopyTo(balls, 0);
        }
    }}
```

Закрытые члены классы

Создание объекта дочернего класса (доступность членов класса)

```
class Program
{
    public static void Main(string[] args)
    {
        Student A = new Student();
        A.Name = "Ivanov Vasya";
        A.SetAddress(places.Барнаул);
        A.Print();
        A.Set(12, 23, 56, 78);

        A.Print();
        Console.Write("\n баллы:"+A.SumBalls.ToString());
        Console.ReadKey(true);
    }
}
```

```
--------------print---------
Ivanov Vasya   0
Address: Барнаул:(1)

другое 1/0:0

Барнаул 1/0:0

Новоалтайск 1/0:1
--------------print---------
Ivanov Vasya   0
Address: Новоалтайск:(2)

 баллы:169
```

Закрытые члены классы

Дочерний класс (производный)

```
class Student : Person{
    int[] balls = null;

    public int SumBalls...
    public void Set(params int[] a){

        foreach (var x in Enum.GetValues(typeof(places)) ){ //перебор значений
            Console.Write("\n" + x + " 1/0:");
            if (Console.ReadLine() == "1"){
                //address = (places)x; //изменяем защищённое поле
                //изменяем защищённое поле, используя метод базового класса
                SetAddress((places)x);
                break;
            }
        }
        if (a != null) {
            balls = new int[a.Length];
            a.CopyTo(balls, 0);
        }
    }}
```

Для изменения значения защищённого поля можно использовать также метод базового класса.

Выводы:

1. При наследовании производный класс получает поля, сво-ва, методы базового класса, а также может иметь свои (отличные от базового) поля, сво-ва методы.
2. Открытые и защищенные члены базового класса доступны без ограничений в производном классе, работа с ними осуществляется так же как со своими членами.
3. Закрытые члены базового класса наследуются производным классов, но доступ к ним возможен либо через открытые (защищённые) свойства, либо открытыми (защищёнными) методами.
4. Доступ к членам производного класса вне объявление класса (внутри функции Main класса Programm) выполняется средствами оператора “.” (точка) к имени объекта: ***A.Print()***. Вне класса видны только откры́ые его члены (методы, св-ва, поля).

Конструкторы и наследование

Конструкторы базового класса не наследуются в производный!
Нужно понимать смысл данного утверждения.

1. Для создания объекта производного класса нужны свои конструкторы (один или несколько, либо конструктор по умолчанию без параметров).
2. Так как объект производного класса является также объектом и базового класса, то для его полноценного построения нужно задействовать и конструктор базового класса. Конструктор производного класса отвечает за инициализацию своих полей, конструктор базового класса также обеспечивает инициалтзацию своих полей (унаследованных).
3. Конструктор класса (если он производный) должен вызывать конструктор базового класса.

Конструкторы и наследование

```
class Car{
    public Car() : this(“Good Car”){..}
    public Car(string name){..}
…
}
class Van: Car{
    public Van(string name, string fullname): base(name){…}
…
}
…..
Van Obj = new Van(“…”, ”…”);
```

Вызов конструктора базового класса осуществляется через **ключевое слово *base***

Конструкторы и наследование

Родительский класс (базовый)

```
class Person{
    private string name;
    protected double weight = 0.0;// защищённый
    protected places address = places.другое;// защищённый
    public string Name{ // св-во к закрытому полю
        get { return name; }
        set { name = value; }
    }
    public void Print(){
        Console.WriteLine("--------------print--------------------");
        Console.WriteLine(Name+ "    "+weight.ToString());
        Console.WriteLine("Address: {0}:({1})", address, (int)address);
    }
    public void SetAddress(places a){
        address = a;
    }
    public Person(string name, double weight){
        Name = name;
        this.weight = weight;
    }
    public Person(){
        Name = "Petrov";
        this.weight = 12.5;
    }
}
```

Конструкторы и наследование

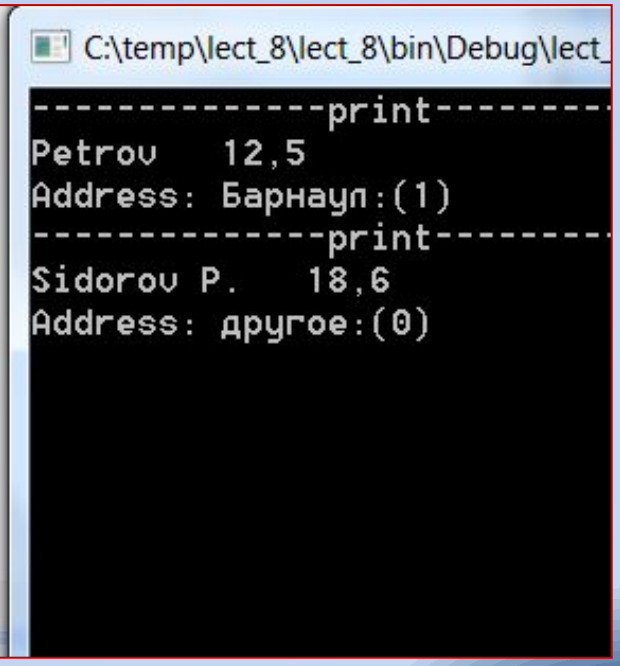
Дочерний класс (производный)

```
class Student : Person{
    int[] balls = null;

    public Student(string name, double weight): base(name, weight){
        balls = new int[]{0,0,0};
    }
    public Student(): base(){

    }

    public int SumBalls...
    public void Set(params int[] a)...}
```

Конструкторы и наследование

Создание объекта дочернего класса

```
class Program
{
    public static void Main(string[] args)
    {
        Student A = new Student();//конструктор без параметров
        A.SetAddress(places.Барнаул);
        //A.Set(12, 23, 56, 78);
        A.Print();
        Student B = new Student("Sidorov P.", 18.6);//другой конструктор
        B.Print();

        Console.ReadKey(true);
    }
}
```

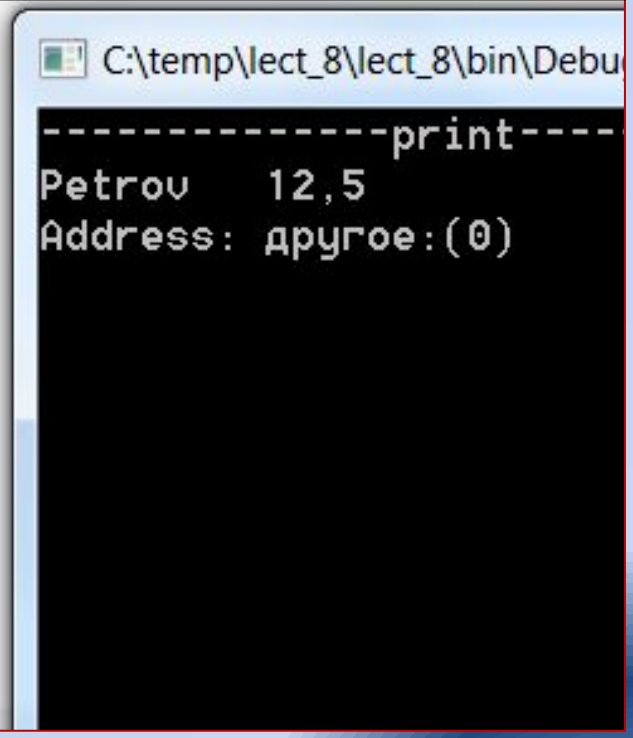


Конструкторы и наследование

Конструктор производного класса явно не вызывает конструктор базового класса

```
class Student : Person{
    int[] balls = null;

    public Student(string name, double weight): base(name, we
        balls = new int[]{0,0,0};
    }
    public Student(){
    }

    public int SumBalls...
    public void Set(params int[] a)...}
class Program
{
    public static void Main(string[] args)
    {
        Student A = new Student();//конструктор без параметров
        //A.SetAddress(places.Барнаул);
        //A.Set(12, 23, 56, 78);
        A.Print();
```

```
class Person{
    private string name;
    protected double weight = 0.0;// защищённый
    protected places address = places.другое;//
    public string Name...
    public void Print()...
    public void SetAddress(places a){
        address = a;
    }
    public Person(string name, double weight)...
    public Person()...
}
```



Базовый конструктор будет вызван неявно. Будет вызван конструктор базового класса по умолчанию (без параметров).

Конструкторы и наследование

Конструктор производного класса явно не вызывает конструктор базового

```
8   class Person{
9       private string name;
10      protected double weight = 0.0;// защищённый
11      protected places address = places.другое;// защищённый
12      public string Name...
16      public void Print()...
21      public void SetAddress(places a){
22          address = a;
23      }
24      public Person(string name, double weight)...
28      /* ... */
32  }
33      class Student : Person{
34          int[] balls = null;
35
36          public Student(string name, double weight): base(name, weight){
37              balls = new int[]{0,0,0};
38          }
39          public Student(){
40          }
41
```

Errors

1 Errors | 0 Warnings | 0 Messages

!	Line	Description	File
⊗	39	lect_8.Person не содержит конструктор, который принимает 0 аргументов (CS172...	Program.cs

Базовый конструктор будет вызван неявно. <u>Если конструктора по умолчанию нет в базовом классе, будет ошибка.</u>

Модификаторы доступа

Модификатор	К чему относится	Описание
public	К любым типам или членам	Элемент видим любому другому коду
protected	К любым членам типа и любым вложенным типам	Элемент видим только любому производному типу
internal	К любым типам или членам	Элемент видим только в пределах включающей его сборки
private	К любым членам типа и любым вложенным типам	Элемент видим только в пределах типа, которому он принадлежит
protected internal	К любым членам типа и любым вложенным типам	Элемент видим только в пределах включающей его сборки, а также в любом коде внутри производного типа

Объявление класса могут быть внутренними (internal) или открытыми (public) в зависимости от того, нужно обеспечить видимость за пределами текущего файла или нет.

Модификаторы доступа

Объявлять класс модификаторами **protected**, **private** или **protected internal** нельзя, поскольку эти уровни видимости не имеют смысла для обычного класса. Таким образом, эти модификаторы могут относиться только к членам (полям, свойствам).

Тем не менее, можно создавать вложенные классы (т.е. типы, содержащиеся внутри других типов) с такими модификаторами видимости, потому что в этом случае типы (подклассы) имеют статус членов.

```
public class ClassA {
    protected class ClassB {
    ….
    }
…..
}
```

Типы наследования

Существуют два отличающихся типа наследования:

- **наследование реализации и**
- **наследование интерфейса.**

Наследование **реализации означает, что класс-наследник является производным от базового класса, получая от него все члены: поля, свойства, методы**. Класс-наследник принимает реализацию каждого метода базового класса, если только в его определении не указано, что реализация метода должна быть переопределена. Такой тип наследования наиболее удобен, когда нужно добавить функциональность к существующему классу или же когда несколько связанных классов разделяют некоторый объем общей функциональности.

Наследование **интерфейса означает, что класс наследует только сигнатуры функций, но не их реализации**. Этот способ наследования наиболее удобен, когда нужно указать, что тип обеспечивает