Классы: основные понятия

Основные элементы класса: поля, методы, конструкторы, декструкторы, свойства.

Понятие класса

- *Класс является* модулем, архитектурной единицей построения программной системы.
- *Класс* является типом данных, определяемым пользователем. Он должен представлять собой одну логическую сущность, например, являться моделью реального объекта или процесса. *Элементами* класса являются *данные* и *функции*, предназначенные для их обработки.
- Все классы .NET имеют общего предка — класс object, и организованы в единую иерархическую структуру.
- Внутри нее классы логически сгруппированы в пространства имен, которые служат для упорядочивания имен классов и предотвращения конфликтов имен: в разных пространствах имена могут совпадать. Пространства имен могут быть вложенными.
- Любая программа использует пространство имен System.

Описание класса

[атрибуты] [модификаторы] class имя_класса [: предки] тело_класса

- Имя класса задается по общим правилам.
- Тело класса — список описаний его элементов, заключенный в фигурные скобки.
- Атрибуты задают дополнительную информацию о классе.
- Модификаторы определяют свойства класса, а также доступность класса для других элементов программы.

Спецификаторы класса

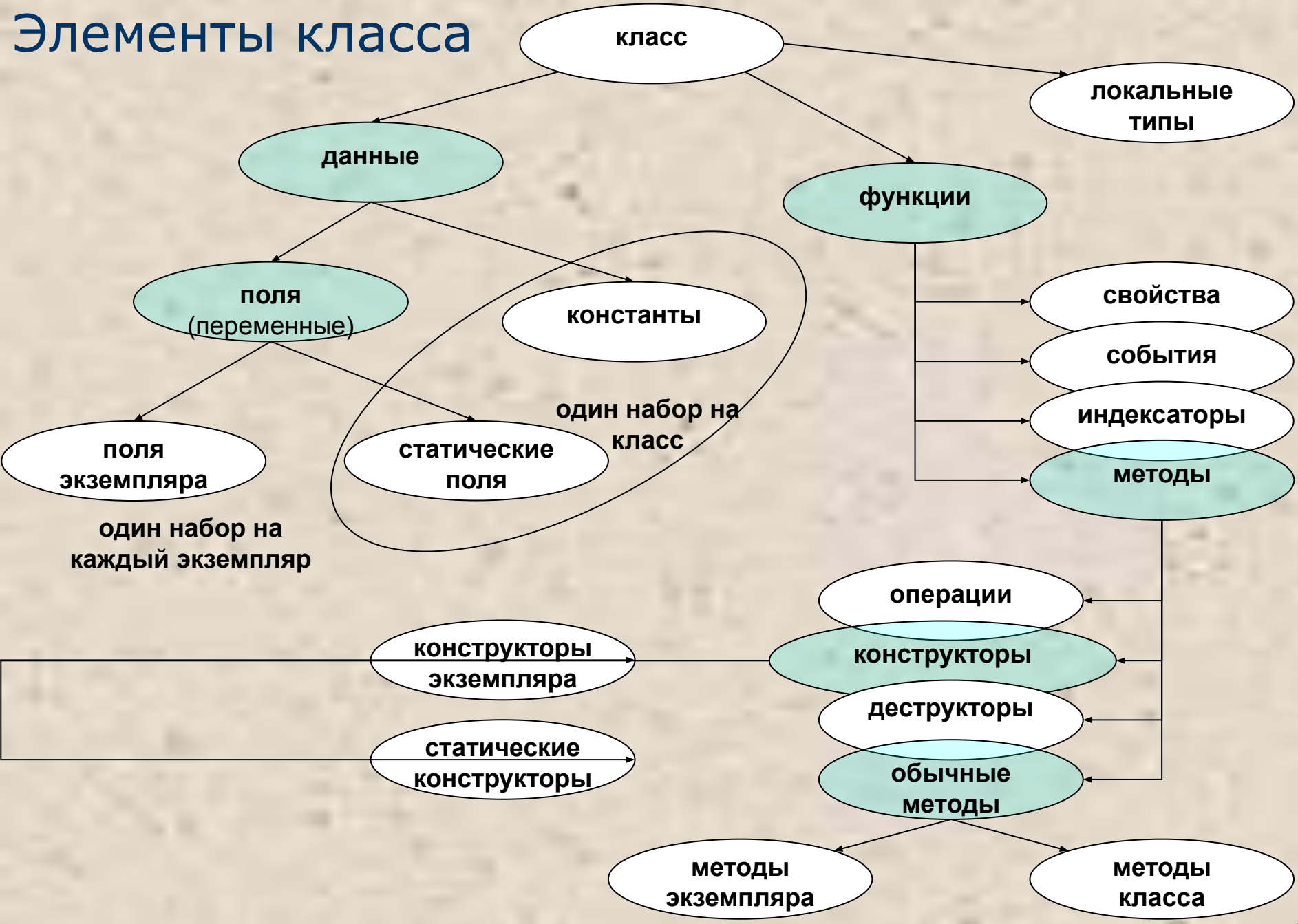
Спецификатор	Описание
new	Используется для вложенных классов. Задает новое описание класса взамен унаследованного от предка. Применяется в иерархиях объектов
public	Доступ не ограничен
protected	Используется для вложенных классов. Доступ только из элементов данного и производных классов
internal	Доступ только из данной программы (сборки)
Protected internal	Доступ только из данного и производных классов или из данной программы (сборки)
private	Используется для вложенных классов. Доступ только из элементов класса, внутри которого описан данный класс
abstract	Абстрактный класс. Применяется в иерархиях объектов.
sealed	Бесплодный класс. Применяется в иерархиях объектов.
static	Статический класс. Введен в версию языка 2.0.

Примеры объявления простейших классов

```
class Demo {}          // пустой класс

public class Rational {тело_класса}
```

Элементы класса



Данные: поля и константы

- Данные, содержащиеся в классе, могут быть переменными или константами.
- Переменные, описанные в классе, называются полями класса.
- При описании полей можно указывать атрибуты и спецификаторы, задающие различные характеристики элементов:

[атрибуты] [модификаторы] [const] тип имя

[= начальное_значение]

- Все поля сначала автоматически инициализируются нулем соответствующего типа (например, полям типа int присваивается 0, а ссылкам на объекты — значение null). После этого полю присваивается значение, заданное при его явной инициализации.

Модификаторы полей и констант класса

Спецификатор	Описание
new	Новое описание поля, скрывающее унаследованный элемент класса
public	Доступ к элементу не ограничен
protected	Доступ только из данного и производных классов
internal	Доступ только из данной сборки
protected internal	Доступ только из данного и производных классов и из данной сборки
private	Доступ только из данного класса
static	Одно поле для всех экземпляров класса
readonly	Поле доступно только для чтения
volatile	Поле может изменяться другим процессом или системой

Методы

- Метод — функциональный элемент класса, реализующий вычисления или другие действия. Методы определяют поведение класса и составляют его **интерфейс**.
- Метод — законченный фрагмент кода, к которому можно обратиться по имени. Он описывается один раз, а вызываться может столько раз, сколько необходимо.
- Один и тот же метод может обрабатывать различные данные, переданные ему в качестве аргументов.

```
double a = 0.1;
double b = Math.Sin(a);
Console.WriteLine(a);
```

Синтаксис метода

[атрибуты] [спецификаторы] тип имя_метода (**[параметры]**) тело_метода

- Спецификаторы: new, **public**, protected, internal, protected internal, private, static, virtual, sealed, override, abstract, extern.
- Метод класса имеет непосредственный доступ к его полям.
- Пример:

```
class Demo {
    double y;                          // закрытое поле класса

    public void Sety( double z ) {     // открытый метод класса
        y = z;
    }
}
… Demo x = new Demo();          // где-то в методе другого класса
  x.Sety(3.12); …               // вызов метода
```

Пример

```
class Demo    {
        public int a = 1;
        public const double c = 1.66;
        static string s = "Demo";
        double y;
        public double Gety() { return y;  }            // метод получения y
        public void Sety( double y_ ){  y = y_; }      // метод установки y
        public static string Gets() { return s; }      // метод получения s
}
class Class1 {
       static void Main()
        {  Demo x = new Demo();
           x.Sety(0.12);                               // вызов метода установки y
           Console.WriteLine(x.Gety());                // вызов метода получения y
           Console.WriteLine(Demo.Gets());             // вызов метода получения s
//          Console.WriteLine(Gets());                 // вариант вызова из того же
}}                                                     // класса
```

Параметры методов

- Параметры определяют множество значений аргументов, которые можно передавать в метод.
- Список аргументов при вызове как бы накладывается на список параметров, поэтому они должны попарно соответствовать друг другу.
- Для каждого параметра должны задаваться его тип, имя и, возможно, вид параметра.
- Имя метода вкупе с количеством, типами и спецификаторами его параметров представляет собой сигнатуру метода — то, по чему один метод отличают от других.
- В классе не должно быть методов с одинаковыми сигнатурами.
- Метод, описанный со спецификатором static, должен обращаться только к статическим полям класса.
- Статический метод вызывается через имя класса, а обычный — через имя экземпляра.

Вызов метода

1. Вычисляются выражения, стоящие на месте аргументов.
2. Выделяется память под параметры метода.
3. Каждому из параметров сопоставляется соответствующий аргумент. При этом проверяется соответствие типов аргументов и параметров и при необходимости выполняется их преобразование. При несоответствии типов выдается диагностическое сообщение.
4. Выполняется тело метода.
5. Если метод возвращает значение, оно передается в точку вызова; если метод имеет тип void, управление передается на оператор, следующий после вызова.

Описание объекта: SomeObj obj = new SomeObj();

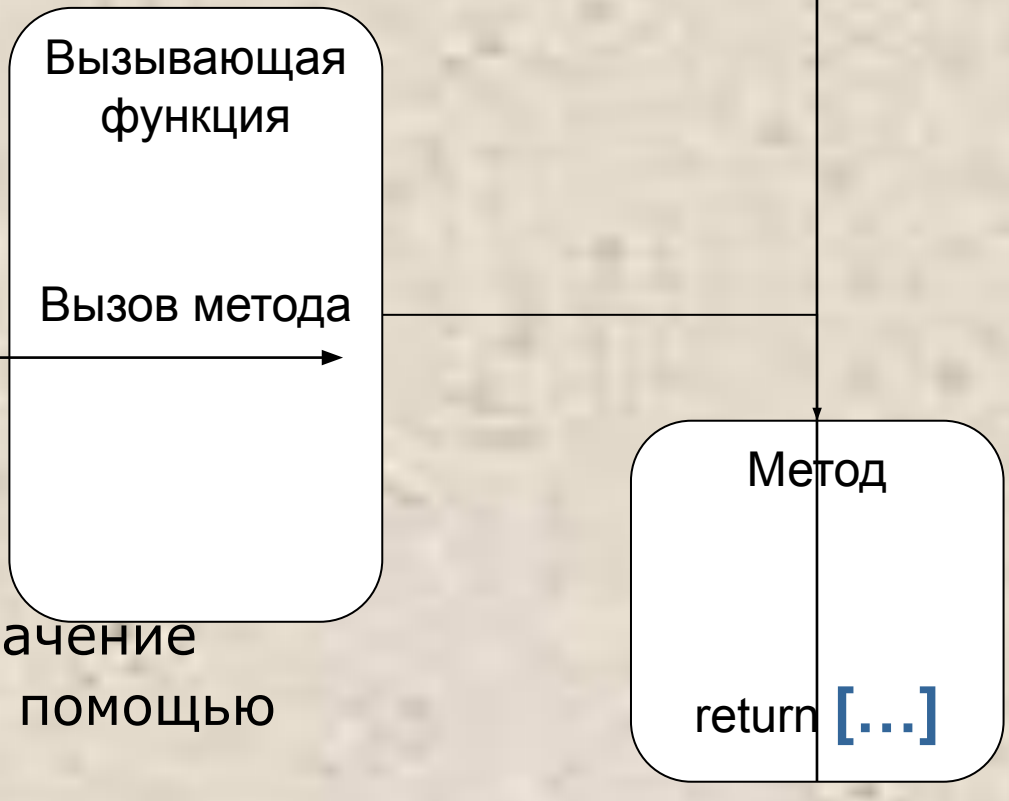
Описание аргументов: int b; double a, c;

Вызов метода: obj.P(a, b, c);

Заголовок метода P: public void P(double x, int y, double z);

Примеры методов

```
public void Sety(double z) {
    y = z;
}
public double Gety() {
   return y;
}
```



- Тип метода определяет, значение какого типа вычисляется с помощью метода
- Параметры используются для обмена информацией с методом. Параметр - локальная переменная, которая при вызове метода принимает значение соответствующего аргумента.

```
x.Sety(3.12);
double t = x.Gety();
```

Способы передачи параметров и их типы

Способы передачи параметров: по значению и по ссылке.

- *При передаче по значению* метод получает копии значений аргументов, и операторы метода работают с этими копиями.
- *При передаче по ссылке* (*по адресу*) метод получает копии адресов аргументов и осуществляет доступ к аргументам по этим адресам.

В C# четыре типа параметров:

- параметры-значения;
- параметры-ссылки (**ref**);
- выходные параметры (**out**);
- параметры-массивы (**params**).

Ключевое слово предшествует описанию типа параметра. Если оно опущено, параметр считается параметром-значением. Пример:

```
public int Calculate( int a, ref int b, out int c, params int[] d ) …
```

Пример передачи параметров

```
class Class1
   {   static int Max(int a, int b)      // выбор макс. значения
       {
          if ( a > b ) return a;
          else         return b;
       }
       static void Main()
       {
          int a = 2, b = 4;
          int x = Max( a, b );        // вызов метода Max
          Console.WriteLine( x );                 // результат: 4
          short t1 = 3, t2 = 4;
          int y = Max( t1, t2 );      // пар-ры совместимого типа
          Console.WriteLine( y );                 // результат: 4
          int z = Max( a + t1, t1 / 2 * b );   // выражения
          Console.WriteLine( z );                 // результат: 5
}}
```

Пример: параметры-значения и ссылки ref

```
using System;
namespace ConsoleApplication1
{   class Class1
    {   static void P( int a, ref int b )
        {
            a = 44; b = 33;
            Console.WriteLine( "внутри метода {0} {1}", a, b );
        }
        static void Main()
        {
            int a = 2, b = 4;
            Console.WriteLine( "до вызова     {0} {1}", a, b );
            P( a, ref b );
            Console.WriteLine( "после вызова  {0} {1}", a, b );
}}}
```

Результат работы программы:

до вызова 2 4

внутри метода 44 33

после вызова 2 33

Пример: выходные параметры out

```
using System;
namespace ConsoleApplication1
{   class Class1
    {   static void P( int x, out int y )
        {
            x = 44; y = 33;
            Console.WriteLine( "внутри метода {0} {1}", x, y );
        }
        static void Main()
        {
            int a = 2, b;         // инициализация b не требуется

            P( a, out b );
            Console.WriteLine( "после вызова  {0} {1}", a, b );
}}}
```

Результат работы программы:

внутри метода 44 33
после вызова 2 33

Правила применения параметров

1. Для параметров-значений используется передача по значению. Этот способ применяется для исходных данных метода.

- При вызове метода на месте параметра, передаваемого по значению, может находиться выражение (а также его частные случаи — переменная или константа). Должно существовать неявное преобразование типа выражения к типу параметра.

2. Параметры-ссылки и выходные параметры передаются по адресу. Этот способ применяется для передачи побочных результатов метода.

- При вызове метода на месте параметра-ссылки **ref** может находиться только имя инициализированной переменной точно того же типа. Перед именем параметра указывается ключевое слово ref.
- При вызове метода на месте выходного параметра **out** может находиться только имя переменной точно того же типа. Ее инициализация не требуется. Перед именем параметра указывается ключевое слово out.

Конструкторы

Конструктор предназначен для инициализации объекта. Он вызывается автоматически при создании объекта класса с помощью операции new. Имя конструктора совпадает с именем класса.

Свойства конструкторов:

- Конструктор не возвращает значение, даже типа void.
- Класс может иметь несколько конструкторов с разными параметрами для разных видов инициализации.
- Если программист не указал ни одного конструктора или какие-то поля не были инициализированы, полям значимых типов присваивается нуль, полям ссылочных типов — значение null.
- Конструктор, вызываемый без параметров, называется конструктором по умолчанию.

Пример класса с конструктором

```
class Demo
    {
        public Demo( int a, double y )        // конструктор
        {
            this.a = a;
            this.y = y;
        }
        int a;
        double y;
    }

    class Class1
    {   static void Main()
        {
            Demo a = new Demo( 300, 0.002 );     // вызов конструктора
            Demo b = new Demo( 1, 5.71 );        // вызов конструктора
            …
     } }
```

Пример класса с двумя конструкторами

```
class Demo
   {
      public Demo( int a )                  // конструктор 1
      {
         this.a = a;
         this.y = 0.002;
      }
      public Demo( double y )               // конструктор 2
      {
         this.a = 1;
         this.y = y;
      }
...
   }
      ...
         Demo x = new Demo( 300 );          // вызов конструктора 1
         Demo y = new Demo( 5.71 );         // вызов конструктора 2
```

Описание объекта (экземпляра)

- Класс является обобщенным понятием, определяющим характеристики и поведение множества конкретных объектов этого класса, называемых экземплярами (объектами) класса.
- Объекты создаются явным или неявным образом (либо программистом, либо системой). Программист создает экземпляр класса с помощью операции new:

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

- Для каждого объекта при его создании в памяти выделяется отдельная область для хранения его данных.
- Кроме того, в классе могут присутствовать статические элементы, которые существуют в единственном экземпляре для всех объектов класса.
- Функциональные элементы класса всегда хранятся в единственном экземпляре.

Пример создания объектов (экземпляров)

```
class Monster { ... }
 class Class1
{
        static void Main()
        {
            Monster X = new Monster();
            X.Passport();
            Monster Vasia = new Monster( "Vasia" );
            Vasia.Passport();
            Monster Masha = new Monster( 200, 200, "Masha" );
            Console.Writeline(Masha);
        }
    }
```

Результат работы программы:
Monster Noname health = 100 ammo = 100
Monster Vasia health = 100 ammo = 100
Monster Masha health = 200 ammo = 200

Свойства

- Свойства служат для организации доступа к полям класса. Как правило, свойство определяет методы доступа к закрытому полю.
- Синтаксис свойства:

[спецификаторы] тип имя_свойства

{

[get {код_доступа}]

[set {код_доступа}]

}

При обращении к свойству автоматически вызываются указанные в нем блоки чтения (**get)** и установки (**set)**.

- Может отсутствовать либо часть get, либо set, но не обе одновременно. Если отсутствует часть set, свойство доступно только для чтения (read-only), если отсутствует get - только для записи (write-only).

Пример описания свойств

```
public class Button: Control
{  private string caption;      // поле, с которым связано свойство
   public string Caption {      // свойство
      get { return caption; }  // способ получения свойства

      set                           // способ установки свойства
         { if (caption != value) { caption = value; }
}}    ...
```

В программе свойство выглядит как поле класса:

```
Button ok = new Button();
ok.Caption = "OK";              // вызывается метод установки свойства
string s = ok.Caption;          // вызывается метод получения свойства
```

Сквозной пример класса

```
class Monster {
    public Monster()      // конструктор
    {
            this.name  = "Noname";
            this.health = 100;
            this.ammo = 100;
    }
  public Monster( string name ) : this()
    {
            this.name = name;
     }
  public Monster( int health, int ammo,
     string name )
     {
            this.name  = name;
            this.health = health;
            this.ammo = ammo;
     }
  public int GetName()          // метод
    {  return name;   }
  public int GetAmmo()        // метод
     {  return ammo;}
```

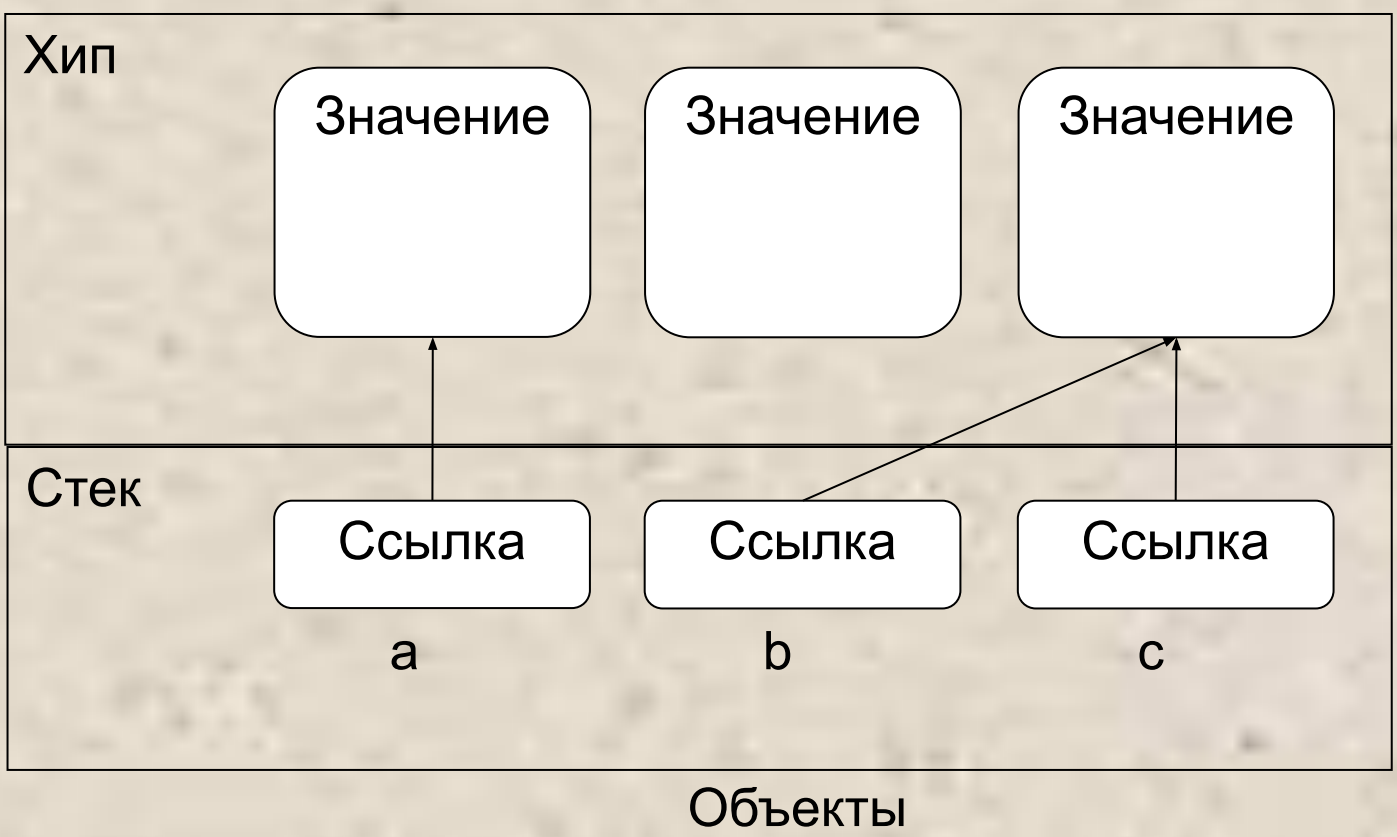
```
public int Health {                  // свойство
   get { return health;  }
   set { if (value > 0) health = value;
           else                 health = 0;
         }
 }

 public void Passport()                  // метод
     {   Console.WriteLine(
  "Monster {0} \t health = {1} \
    ammo = {2}", name, health, ammo );
      }

public override string ToString(){
     string buf = string.Format(
       "Monster {0} \t health = {1} \
     ammo = {2}", name, health, ammo);
    return buf;  }

  string name;
  int health, ammo;
}
```

Присваивание и сравнение объектов



- b = c
- Величины ссылочного типа равны, если они ссылаются на одни и те же данные (b == c, но a != b даже при равенстве их значений или если обе ссылки равны null).