



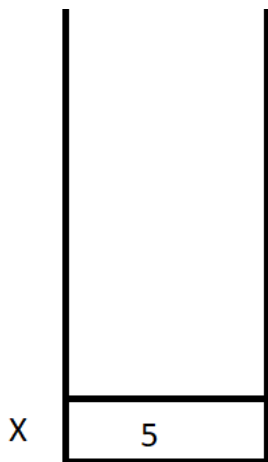
Web-разработка на C# и платформе Microsoft .NET

Ссылочные типы

Виды памяти

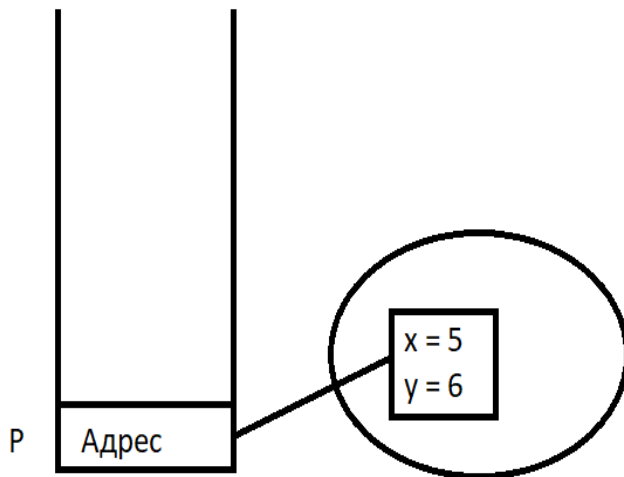
СТЕК (СТАТИЧЕСКАЯ ПАМЯТЬ)

```
int x = 5;
```



КУЧА (ДИНАМИЧЕСКАЯ ПАМЯТЬ)

```
MyPoint p = new MyPoint();  
p.x = 5; p.y = 6;
```



Виды типов объектов

СТЕК (СТАТИЧЕСКАЯ ПАМЯТЬ)

- Живут в стеке
- Гарантированно будет живым как минимум до тех пор, пока текущая «точка исполнения» находится в области видимости локальной переменной

КУЧА (ДИНАМИЧЕСКАЯ ПАМЯТЬ)

- Живут в динамической памяти
- Уничтожаются сборщиком мусора при отсутствии ссылок на них

Ссылочные типы

- Строки
`string`
- «Любой объект»
`object`
- Определяемые пользователем

```
class MyPoint
{
    public int x { get; set; }
    public int y { get; set; }
}
```

СТРУКТУРА

```
struct SPoint
{
    public int x;
    public int y;
}
```

```
SPoint p = new SPoint();
p.x = 5; p.y = 6;
SPoint p1 = p;
p.x = 10;
```

Чему равен p1.x?

КЛАСС

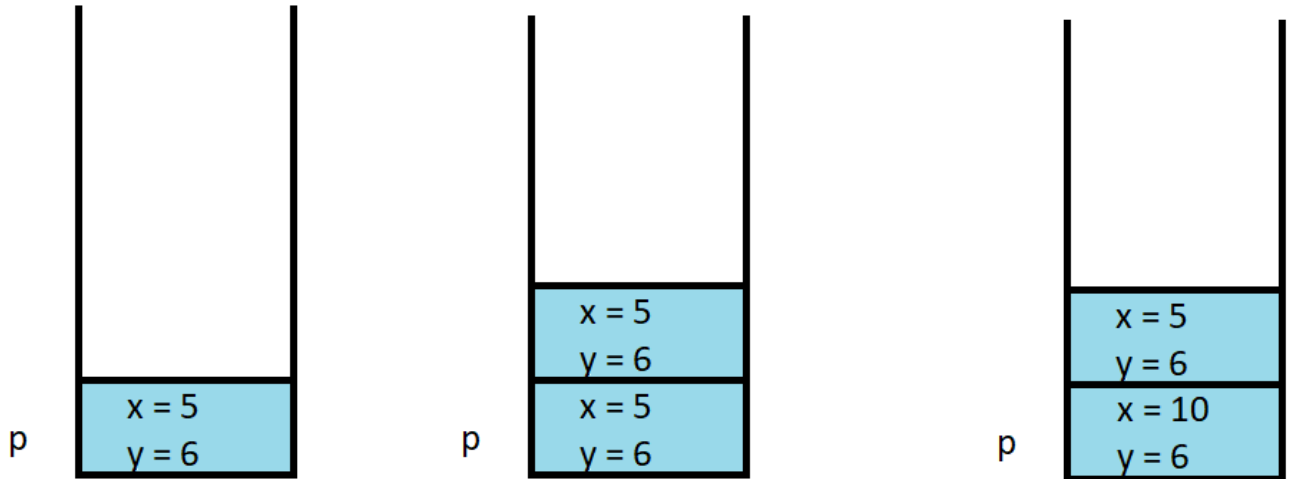
```
class CPoint
{
    public int x;
    public int y;
}
```

```
CPoint p = new CPoint();
p.x = 5; p.y = 6;
CPoint p1 = p;
p.x = 10;
```

Чему равен p1.x?

Различия между value и reference типами

СТРУКТУРА



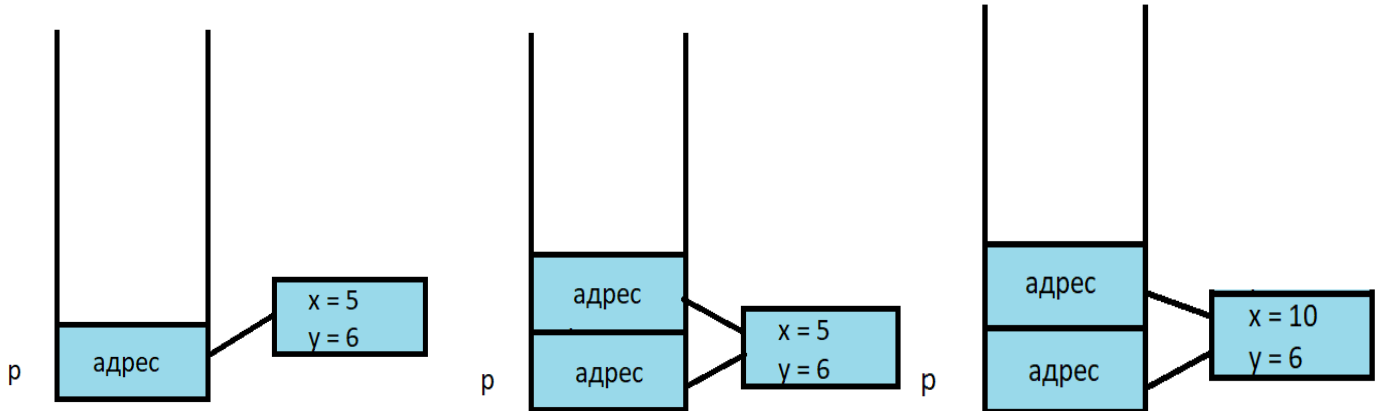
```
SPoint p = new SPoint();  
p.x = 5; p.y = 6;
```

```
SPoint p1 = p;
```

```
p.x = 10;
```

Различия между value и reference типами

КЛАСС



```
CPoint p = new CPoint();  
p.x = 5; p.y = 6;  
CPoint p1 = p;
```

```
p.x = 10;
```

Различия между value и reference типами

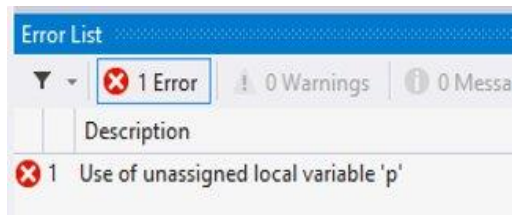
СТРУКТУРА

```
SPoint p;  
p.x = 1;  
p.y = 2;
```

```
SPoint p;  
  
void SetValue()  
{  
    p.x = 1;  
    p.y = 2;  
}
```

КЛАСС

```
CPoint p;  
p.x = 1;  
p.y = 2;
```



```
CPoint p;  
  
void SetValue()  
{  
    p.x = 1;  
    p.y = 2;  
}
```



Объявление класса

```
[атрибуты]
[спецификаторы] class имя_класса [: предки]
{
    [атрибуты]
    [спецификаторы] тип имя_поля [= значение];

    [атрибуты]
    [спецификаторы] тип имя_метода(параметры)
    { тело_метода }

    [атрибуты]
    [спецификаторы] тип имя_свойства
    {
        [[спецификаторы] get { тело }]
        [[спецификаторы] set { тело }]
    }
}
```

Пример

```
public class Circle
{
    public double x;
    public double y;
    private double r;

    public double R
    {
        get { return r; }
        set {
            if (value >= 0)
                r = value;
        }
    }

    public double GetLength()
    {
        return 2 * Math.PI * r;
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Circle c = new Circle();
        c.x = 5;
        c.y = 6;
        c.r = 7; // Ошибка
        c.R = 7; // Вызов метода set
        Console.WriteLine(c.GetLength());
    }
}
```

Поля и свойства

- **Поле** – переменная, хранящая значение.
- **Свойство** – пара методов, предназначенных для правильной инкапсуляции поля.
 - Getter (get) не должен выполнять длительных вычислений
 - Setter (set) должен проверять значения, передавать уведомления. Значение передаётся через ключевое слово value.

Свойства в С#

```
[атрибуты]  
[спецификаторы] тип имя_свойства  
{  
    get { тело }  
    set { тело }  
}
```

```
private double r;  
public double R  
{  
    get { return r; }  
    set { r = value; }  
}
```

Методы

- Конструкторы
- Статические методы
- Экземплярные методы

Конструкторы

Автоматический конструктор

```
Circle c = new Circle();
```

Явно заданный конструктор по умолчанию

```
public Circle()
```

```
{
```

```
    x = y = 0;
```

```
    r = 1;
```

```
}
```

Конструктор с параметрами

```
public Circle(double r)
```

```
{
```

```
    x = y = 0;
```

```
    this.r = r;
```

```
}
```

Приведение reference type

Неявное приведение

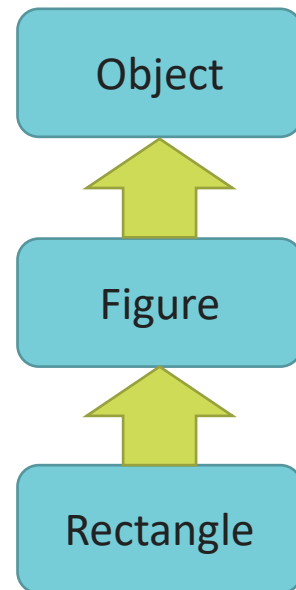
```
Figure f = new Rectangle();  
object o = f;
```

Явное приведение

```
// Существует два способа, отличающиеся  
// реакцией на ошибку приведения типов:
```

```
Figure f1 = (Figure)o;  
// возникает InvalidCastException
```

```
Rectangle r = f as Rectangle;  
// присваивается значение null
```



Проверка типа

ТИП (ЗНАЧЕНИЕ)

```
if (f is Rectangle)
{
    r = (Rectangle)f;
    // использование r
}
```

ЗНАЧЕНИЯ AS ТИП

```
r = f as Rectangle;
if (r != null)
{
    // использование r
}
```


Nullable типы

Объявление

```
Nullable<тип> имя = значение;  
тип? имя = значение;
```

Пример

```
Nullable<int> n = 20;  
int? n = 20;
```

Получение значения

```
int n1 = n.Value;  
int n1 = (int)n;
```

Определение наличия значения

```
if (n != null) { }  
if (n.HasValue) { }
```

Неявная типизация (синтаксический сахар)

Ключевое слово

var сообщает компилятору о необходимости определения типа переменной из выражения, находящегося справа от оператора присваивания.

Если тип результата определить невозможно, использование **var** недопустимо.

```
var i = 5;
```

```
struct System.Int32  
Represents a 32-bit signed integer.
```

```
var s = "Hello";
```

```
class System.String  
Represents text as a series of Unicode characters.
```

```
var a = new[] { 0, 1, 2 };
```

```
Int32[]
```

```
for (var x = 0; x < 10; x++)  
{ }
```

```
struct System.Int32  
Represents a 32-bit signed integer.
```

```
foreach (var x in a)  
{ }
```

```
struct System.Int32  
Represents a 32-bit signed integer.
```

[http://msdn.microsoft.com/ru-ru/library/bb384061\(v=vs.90\).aspx](http://msdn.microsoft.com/ru-ru/library/bb384061(v=vs.90).aspx)

Упаковка и распаков

Упаковка / Boxing

```
int x = 5;  
object obj = x;  
Type t = obj.GetType();  
if (obj.Equals(x)){ }
```

Распаковка / Unboxing

```
int y = (int)obj;
```

Действия при упаковке/распаковке

Упаковка

1. Выделяется память в управляемой куче
2. Поля значимого типа копируются в память
3. К объекту добавляется указатель на тип и SyncBlockIndex.
4. Возвращается адрес объекта

Распаковка

1. Возвращается указатель на упакованное значение.

Примечание: обычно распаковка происходит вместе с копированием

Способы передачи параметров


По значению (по умолчанию)

По ссылке (модификатор ref)

По ссылке для возврата (модификатор out)

Как массив (модификатор params)

Передача параметров по значению

- 
- В метод передаётся копия параметра
 - Любые совершаемые с формальным параметров действия не ведут к изменению фактического параметра
 - В качестве фактического параметра могут выступать выражения


Пример

```
static void Funct(int par)
{
    Console.WriteLine(par);
    par = 7;
    Console.WriteLine(par);
}

static void Main(string[] args)
{
    int x = 5;
    Console.WriteLine(x);
    Funct(x);
    Console.WriteLine(x);
    Console.Read();
}
```



Передача параметров по ссылке

- 
- В метод передаётся ссылка на параметр
 - Для обозначения способа передачи используется модификатор `ref`
 - Параметром может выступать только инициализированная переменная или поле объекта

Пример

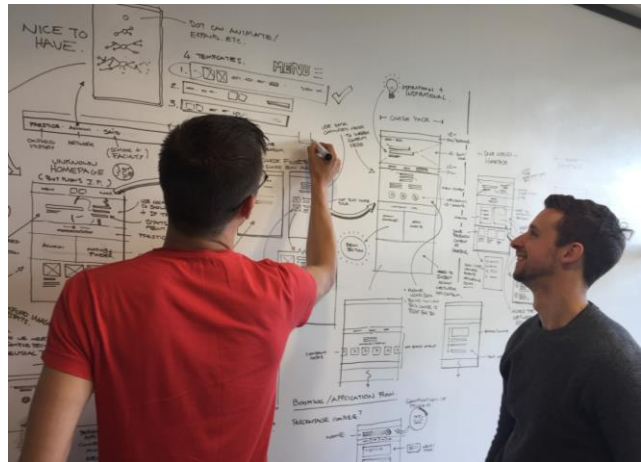
```
static void Funct(ref int par)
{
    Console.WriteLine(par);
    par = 7;
    Console.WriteLine(par);
}

static void Main(string[] args)
{
    int x = 5;
    Console.WriteLine(x);
    Funct(ref x);
    Console.WriteLine(x);
    Console.Read();
}
```



Объекты в качестве параметров

- При передаче по значению объекта передаётся ссылка на оригинал
- Массив – это объект. Следовательно, любые его изменения в методе приводят к модификации оригинала



СПАСИБО ЗА ВНИМАНИЕ