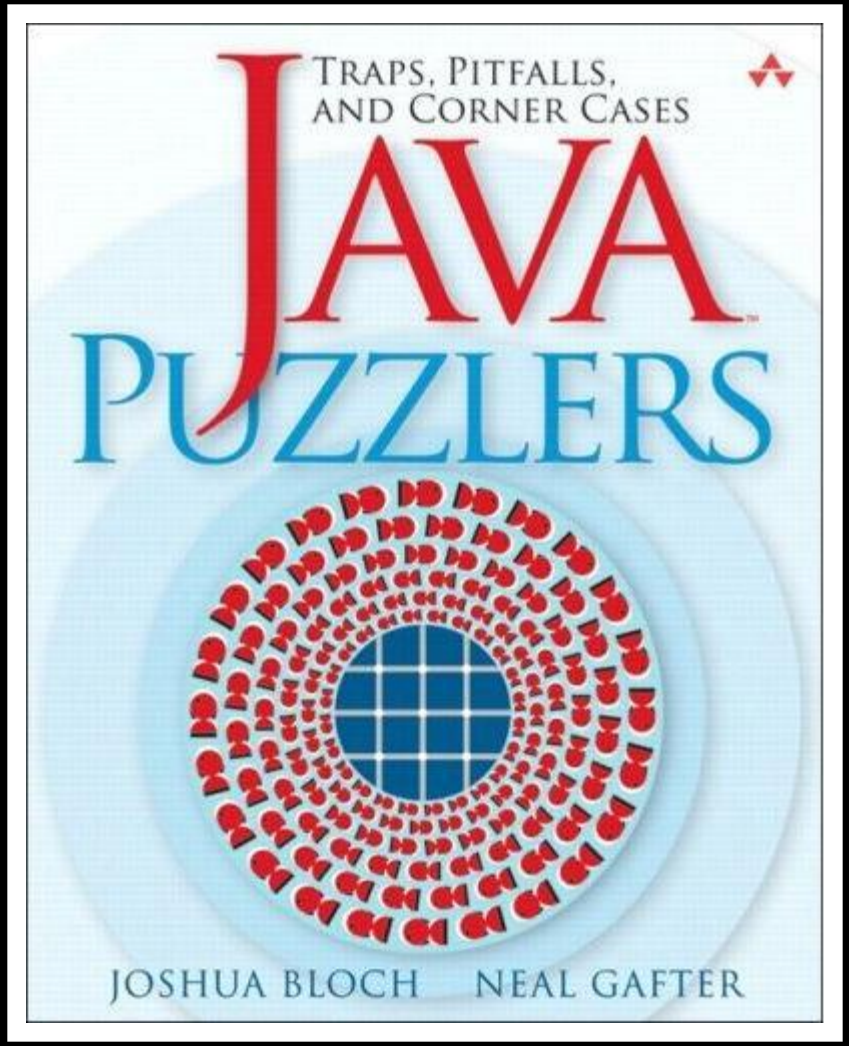
<Подводные камни C#>

Гайдар Магдануров
Microsoft


TRAPS, PITFALLS,
AND CORNER CASES
JAVA
PUZZLERS
JOSHUA BLOCH
NEAL GAFTER

Microsoft®
Visual C#® 2008
Express Edition

http://www.microsoft.com/express/ru/

За длинным числом погонишься…

```
static void Main(string[] args)
{
    Console.WriteLine(GetSomeResult(10000));
}

static long GetSomeResult(long someValue)
{
    long value1 = 10 * 1000 * 10000 * someValue;
    long value2 = 10 * 1000 * 10000 * 100000;
    return value2 / value1;
}
```

Не компилируется.
Переполнение.

За длинным числом погонишься…

```
static void Main(string[] args)
{
    Console.WriteLine(GetSomeResult(10000));
}

static long GetSomeResult(long someValue)
{
    long value1 = 10 * 1000 * 10000 * someValue;
    long value2 = 10 * 1000 * 10000 * 100000L;
    return value2 / value1;
}
```

Я – не я?

```
float x = float.NaN;
Console.Write(x == x)
```

Выводит False.

Как это может быть?

Он или не он, вот в чем вопрос...

```
public static void Main()
   {
      Test t = new Test();
      Console.WriteLine(t.Equals(t));
   }
```

Выводит False.

Метод Equals, наследуемый от Object, сравнивает ссылки. А тут, что за дела?

```
public class Test
   {
      public bool Equals(Test t) { return false; }
   }
```

Тернарный условный оператор

```
static void Main(string[] args)
{
    char a = 'a';
    int b = 0;
    Console.WriteLine(true ? a : b);
}
```

97

Типы операндов не совпадают и компилятор приводит оба типа к общему совместимому Int32.

Конкатенируя конкатенируй

```
Console.WriteLine("A" + "B" + "C");
Console.WriteLine('A' + 'B' + 'C');
```

Оператор «+» не определен для char, поэтому выполняется приведение к Int32.

Циклическая инициализация

```
public class A { public static int x = B.y + 1; }
public class B { public static int y = A.x + 1; }

    static void Main(string[] args)
    {
        Console.WriteLine("A.x = " + A.x);
        Console.WriteLine("B.y = " + B.y);
    }
```

A.x = 2
B.y = 1

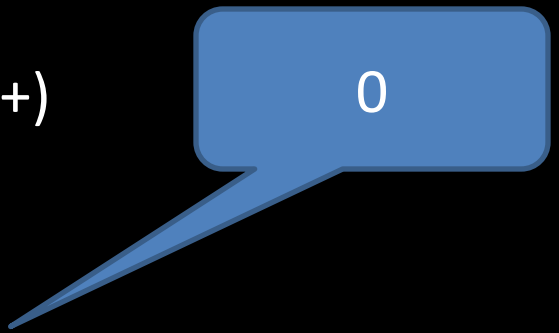
Вызывается конструктор A, затем конструктор B, но т.к. не определено значение A.x, то для x используется 0 в конструкторе B.

Инкремент инкременту волк

```
int j = 0;

    for (int i = 0; i < 10; i++)
  j = j++;

    Console.WriteLine(j);
```



Оператор ++ возвращает значение до инкрементации, поэтому j сохраняет исходное значение.

На пределе возможностей

```
int end = int.MaxValue;
int begin = end - 100;
int counter = 0;

for (int i = begin; i <= end; i++)
{
        counter++;
        Console.WriteLine(counter);
 }
```

Бесконечный цикл

Все переменные типа Int32 меньше или равны Int32.MaxValue.

Большие числа, большие проблемы

```
float begin = 1000000000;
 int counter = 0;

 for (float i = begin; i < (begin + 10); i++)
     counter++;

 Console.WriteLine(counter);
```

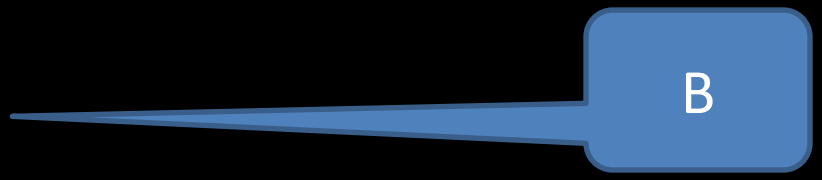
Бесконечный цикл

Для таких больших значений float нет разницы между begin и begin + 10, но компилятор что-то подозревает. Использовать int/long счетчик цикла.

Кто компилятору милее

```
class A { public void Test(int n) { Console.WriteLine("A"); } }
class B : A { public void Test(double n) { Console.WriteLine("B"); } }

static void Main(string[] args)
{
    B b = new B();
    b.Test(5);
}
```



Приоритет отдается методу класса, по типу ссылки, если в нем определен метод с типами аргументов позволяющих выполнить преобразование без потерь.

Кто компилятору милее

```
class A { public void Test(double n) { Console.WriteLine("A"); } }
class B : A { public void Test(int n) { Console.WriteLine("B"); } }

static void Main(string[] args)
{
    B b = new B();
    b.Test(5.0);
}
```

Кто компилятору милее 2

```
public class Test
  {
     public Test(object obj) { Console.WriteLine("object"); }
     public Test(int[] obj) { Console.WriteLine("int[]"); }
     public Test(float[] obj) { Console.WriteLine("float[]"); }
  }
  public static void Main()  { Test t = new Test(null); }
```

Не компилируется. Неоднозначность конструкторов.

Кто компилятору милее 3

```
public class Test
  {
     public Test(object obj) { Console.WriteLine("object"); }
     public Test(int[] obj) { Console.WriteLine("int[]"); }
  }
  public static void Main()  { Test t = new Test(null); }
```

Компилятор при определении вызова не использует текущее значение, а выбирает наиболее «специфический» конструктор. Стоит явно указать тип: new Test((object)null);

Анонимные типы в C#

Синтаксис:

```
var anonType = new {
        FirstName = "John",
        LastName = "Lennon"
        };
```

Компилятор создает некий базымянный тип, область видимости которого ограничена текущей областью видимости в которой объявлен анонимный тип.

C# Общие типы (Generics) в C#

Синтаксис:

```
public class Stack<T>
 {
        private T[] items;
        public void Push(T item) { }
        public T Pop() { }
}
```

Возможность использования одного класса для работы с разными типами a la шаблоны C++.

Свободу попугаям!

- Возможно ли использовать общую (generic) коллекцию для анонимных типов? Например, ***List<?>***?

- Можно ли передать объект анонимного типа в другую область видимости?

Докатились List<?>…

```
var beatlesMember = new {
                        FirstName = "John",
                        LastName = "Lennon"
                    };

var beatlesList = (new[] { beatlesMember }).ToList();

beatlesList.Add(new {
                        FirstName = "Paul",
                        LastName = "McCartney"
                    });
```

Пример использования типа «по образцу».

Передача анонимного типа в другую область видимости? Час от часу …

```
static object GetBeatleName()
{ return new { First = “John", Last = “Lennon" }; }

static T CastType<T>(object obj, T type) { return (T)obj; }

static void Main()
{
  object o = GetBeatleName();
  var typed = CastType(o, new { First = "", Last = "" });
  Console.WriteLine(“First={0}, Last={1}",
          typed.First, typed.Last);
}
```

Пример использования типа «по образцу».

```
First=John
Last=Lennon
```

Лямбда-выражения в C#

Синтаксис для записи анонимного делегата:

```
item => item < 3
```

равнозначно

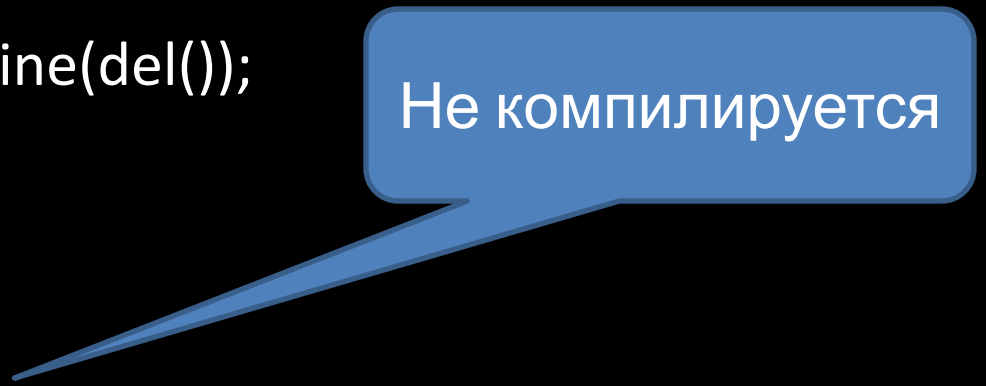
```
delegate(int item) { return item < 3; }
```

Пример использования:

```
List<int> list = new List<int>() { 1, 2, 3, 4, 5 };
var q = list.FindAll(item => item < 3);
```

Танец с лямбдами

```
delegate bool TestDelagate();
static void Test(TestDelagate del)
{
    Console.WriteLine(del());
}
static void Main()
{
    Test(p => true);
}
```



Делегат, не принимающий параметров, описывается **() => true**.

Простая считалочка

```
List<int> list = new List<int>() { 1, 2, 3, 4, 5 };
List<int> all = list.FindAll(
        i => { Console.Write(i); return i < 3; }
 );
```

12345

Делегат, переданный методу FindAll вызывается для каждого элемента.

Считалочка посложнее

```
List<int> list = new List<int>() { 1, 2, 3 };
var x = list.GroupBy(i => { Console.Write(i); return i; });
var y = list.ToLookup(i => { Console.Write(i); return i; });
```

123

Выполнение GroupBy отложено до обращения к результату. Вывод 123123 будет если дописать, например, строку

var z = x.ToArray();

И, наконец …

```
try {
    Console.WriteLine("Hello ");
    return;
}
finally { Console.WriteLine("Goodbye "); }

Console.WriteLine("world!");
```

Hello Goodbye

Finally выполняется даже если выполнение прервано по return.

И еще, наконец…

```
try {
    Console.WriteLine("Hello ");
    Thread.CurrentThread.Abort();
}
finally { Console.WriteLine("Goodbye "); }

Console.WriteLine("world!");
```

Abort выбрасывает исключение ThreadAbortException, которое обрабатывается finally, затем выполнение прерывается.

Все, совсем все!

```
try {
    Console.WriteLine("Hello ");
    System.Environment.Exit(0);
}
finally { Console.WriteLine("Goodbye "); }

Console.WriteLine("world!");
```

Выполнение программы прерывается в точке вызова System.Environment.Exit(0);

СПАСИБО ЗА ВНИМАНИЕ!

</Подводные камни C#>

Гайдар Магдануров
Microsoft

GAIDARMA@MICROSOFT.COM

BLOGS.MSDN.COM/GAIDAR