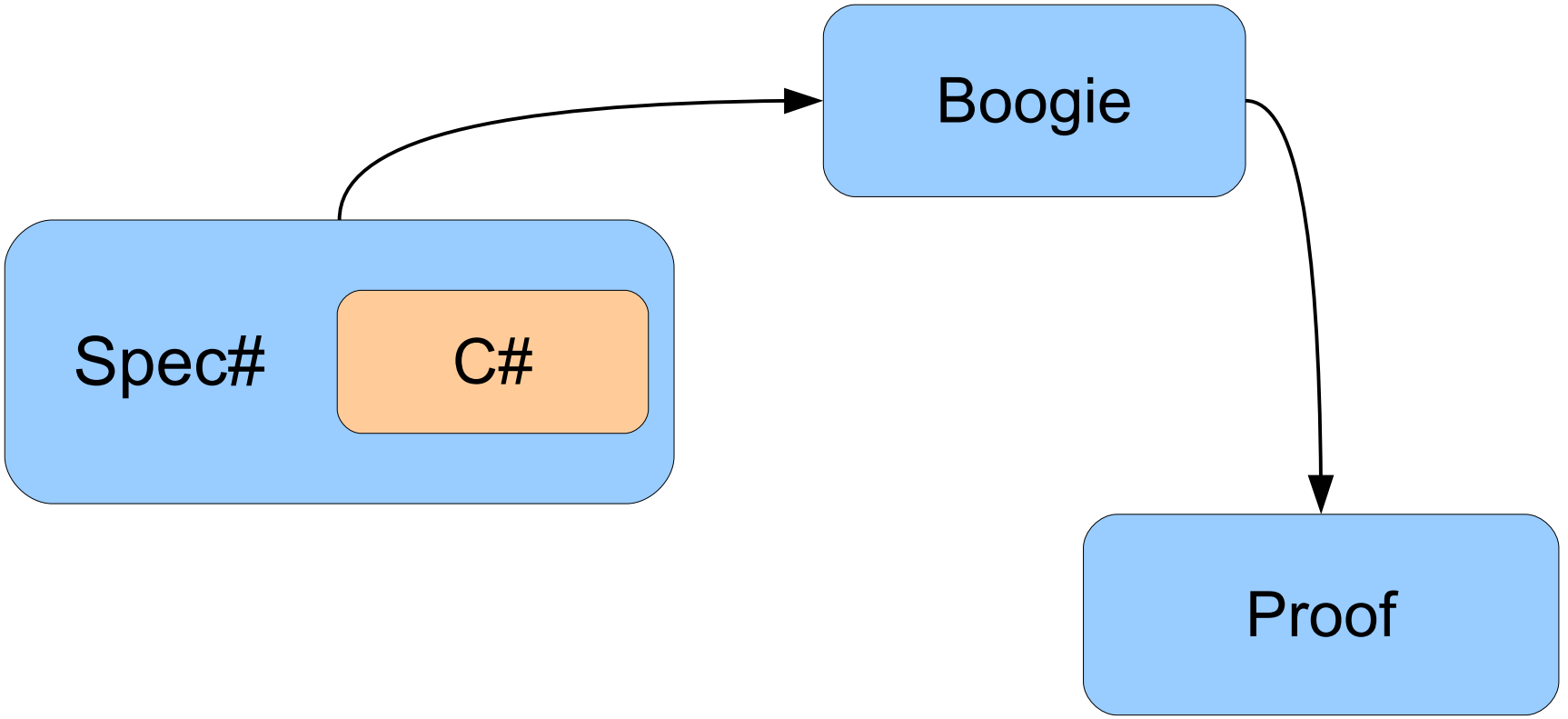
# Program verification using Spec#

Boriss Shelajev
Institute of Cybernetics

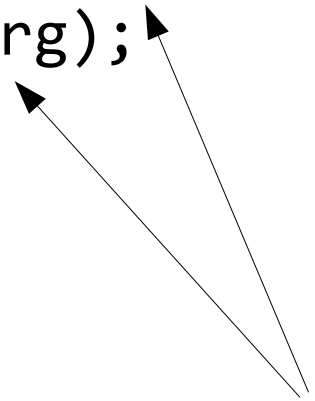
Theory Days at Otepää, 2013

# Introduction



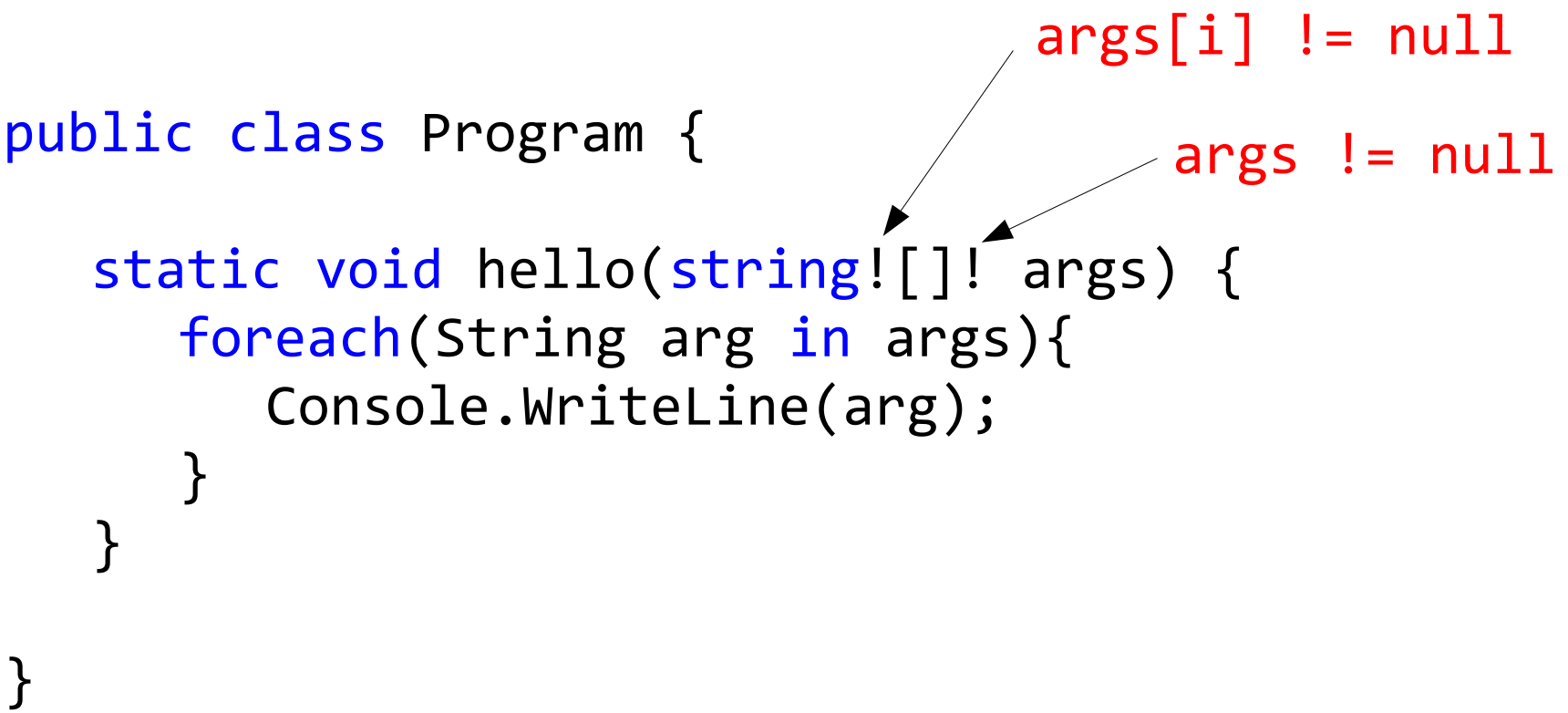
# Non-Null types

```
public class Program {

   static void Main(string[] args) {
      foreach(String arg in args){
         Console.WriteLine(arg);
      }
   }

}
```



Possible null dereference

# Non-Null types



# Initializing Non-Null types

```
public class A {

   T! x;

   public A() {
      x = new T();
      x.Call();
   }

}
```

Cannot call this.x until this is fully initialized

# Initializing Non-Null types

```
public class A {

   T! x;
   [NotDelayed]
   public A() {
      x = new T();
      base();
      x.Call();
   }

}
```

Allows fields of the receiver be read

All non-null fields must be initialized before calling base()

# Method contracts

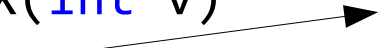
```
int addIncrementally(int a, int b)
   requires b >= 0;
   ensures result == old(a) + old(b);
{
   int ghostVar = a + b;
   while(b>0)
      invariant b >= 0;
      invariant ghostVar == a + b;
   {
      a = a + 1;
      b = b - 1;
   }

   return a;
}
```

# Quantifiers and comprehensions

```
forall{int i in (0: a.Length); 0 <= a[i]}

exists{int i in (0: a.Lenght); 0 <= a[i]}


sum{int i in (0: a.Length); 0 <= a[i]}

and product, min, max, count
```

# Object invariants. Why?

```
public class A {
    private int x;
    public int divideByX(int v)
        requires x != 0;
    {
        return v / x;
    }

}
```



```
public class A {
    private int x;
    invariant x != 0;
    public int divideByX(int v)
    {
        return v / x;
    }

}
```

# Object invariants. Why?

- Specifing method contracts one specifies how method should be used, spells out what is expected from caller and what caller should expect back

- To specify the design implementation one uses assertion involving object invariants

Each object data field must satisfy object invariant whenever object is in a valid state

# Object invariants. Counter example

```
public class Counter{
    int c; bool even;
    invariant 0 <= c;
    invariant even <==> c%2 == 0;

    public Counter(){
        c = 0;
        even = true;
    }

    public void inc()
        modifies c, even;
        ensures c == old(c) + 1;
    {
        c++;
        even = !even;
    }
}
```

# Object invariants. Counter example

```
public class Counter{
    int c; bool even;
    invariant 0 <= c;
    invariant even <==> c%2 == 0;

    public Counter(){
        c = 0;
        even = true;
    }

    public void inc()
        modifies c, even;
        ensures c == old(c) + 1;
    {
        expose(this){
            c++;
            even = !even;
        }
    }
}
```

Invariant may be broken in the constructor

Invariant must be established and checked exiting mutable state

Invariant may be broken within the expose block

# Object states

Every object has a special field State, which is either Valid or Mutable

- Mutable state
  - Object invariant might be violated
  - Field updates are allowed
- Valid state
  - Object invariant holds
  - Field updates are allowed only if they maintain invariant