Uvod u C#

Programiranje korisničkih interfejsa

Bojan Furlan

Hello, World

```
using System;

class Hello
{
  public static void Main()
  {
    Console.WriteLine("Hello, World");
  }
}
```

Exception Handling

```
using System;
public class Hello
{
  public static void Main(string[ ] args)
  {
     try{
         Console.WriteLine(args[0]);
        }
     catch (Exception e) {
         Console.WriteLine("Exception at
        ➥{0}", e.StackTrace);
     }
  }
}
```

Generisanje XML Dokumentacije

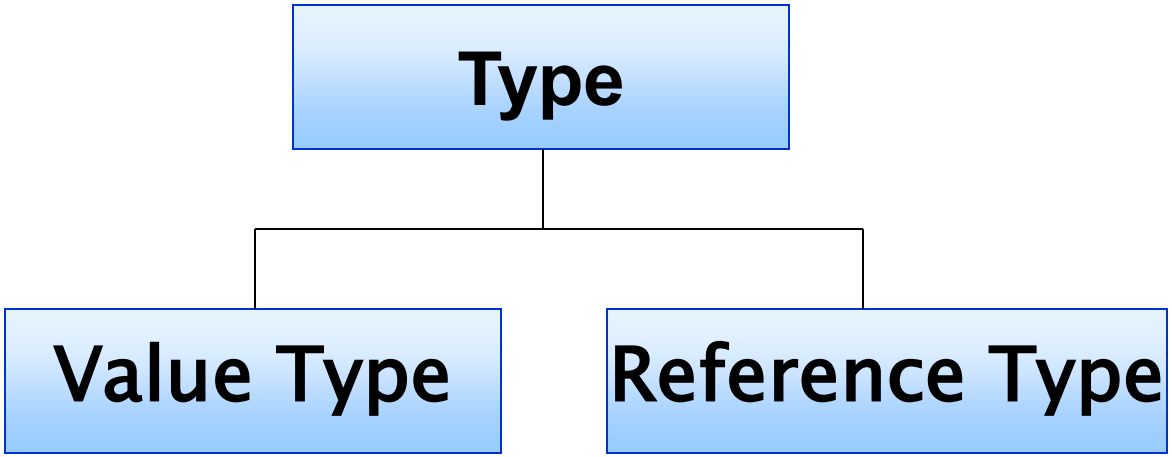
```
/// <summary>
/// ovo je test metod
/// </summary>
///<param name="id"> id poziva </param>
private void test(int id)
{
    //standardni komentar
    System.Console.WriteLine("test {0}",id);
}
```

Debugging

- Exceptions and JIT Debugging
- The Visual Studio Debugger
  - Setting breakpoints and watches
  - Stepping through code
  - Examining and modifying variables

Tipovi

Common Type System



Value Type (čuvaju se na steku):

struct, enum, numeric types (integer, floating-point, decimal), bool, ...

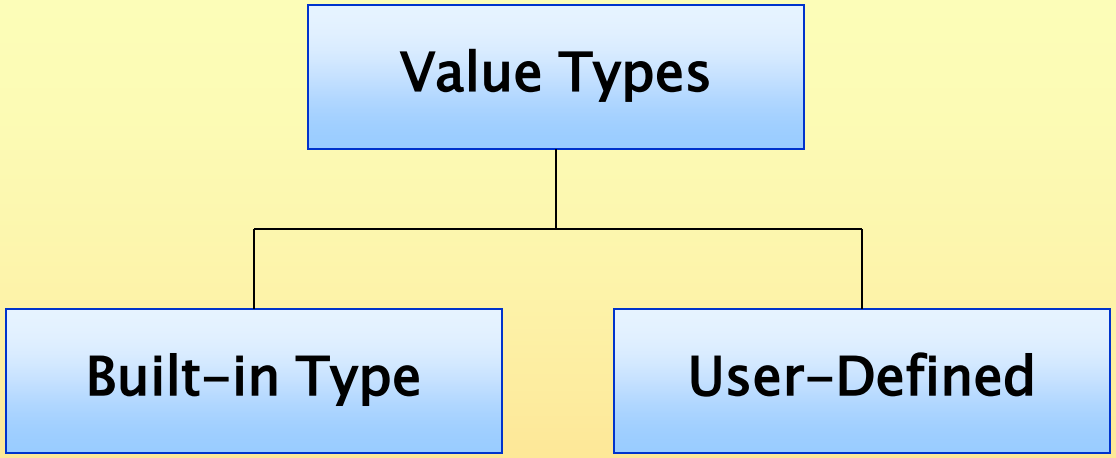
a=b kopira vrednost

Reference Type (čuvaju se na hipu):

class, interface, delegate, array...

a=b kopira referencu

Ugrađeni i korisnički def. tipovi



- int
- float
- enum
- struct

Nabrojivi tip – enum

- Definicija

```
enum Color { Red, Green, Blue }
```

- Upotreba

```
Color colorPalette = Color.Red;
```

- Prikaz

```
Console.WriteLine("{0}", colorPalette); // Prikazuje Red
```

Struktura

- Definicija

```
public struct Employee
{
     public string firstName;
     public int age;
    public int id()
    {
        …
    }
}
```

- Upotreba

```
Employee companyEmployee;
companyEmployee.firstName = "Joe";
companyEmployee.age = 23;
```

Konverzija vrednostnih tipova

Implicitna konverzija

- int to long:

```
using System;
class Test
{
    static void Main( )
    {
        int intValue = 123;
        long longValue = intValue;
        Console.WriteLine("(long) {0} = {1}", intValue,
    ➥longValue);
    }
}
```

Eksplicitna konverzija

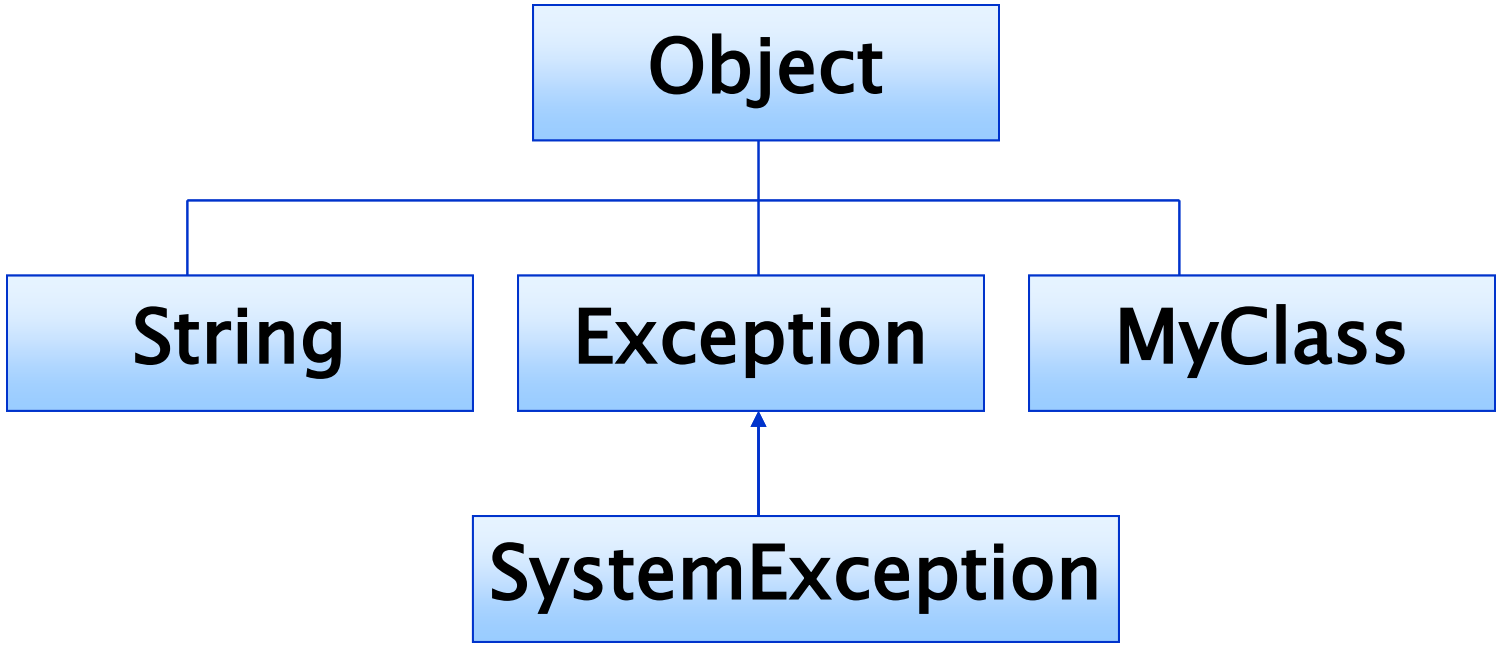
- cast operator:

```
using System;
class Test
{
    static void Main( )
    {
        long longValue = Int64.MaxValue;
        int intValue = (int) longValue;
        Console.WriteLine("(int) {0} = {1}", longValue,
    ➥intValue);
    }
}
```

Konverzija ref. tipova

Tip object

- Sinonim za System.Object
- Bazna klasa za sve klase



Parent/Child Conversions

- Conversion to parent class reference
  - Implicit or explicit
  - Always succeeds
  - Can always assign to object
- Conversion to child class reference
  - Explicit casting required
  - Will check that the reference is of the correct type
  - Will raise **InvalidCastException** if not

is Operator

- Kao u Javi *instanceof*

```
Bird b;
if (a is Bird)
    b = (Bird) a; // Safe
else
    Console.WriteLine("Not a Bird");
```

NE TREBA ZLOUPOTREBITI! – Koristiti polimorfizam!

```
if (a is ClassA)
    a.metod1(); //
else
    if(a is ClassB) a.metod2(); …
```

as Operator

- Kao *cast* operator
- U slucaju greške ne baca izuzetak već vraća *null*

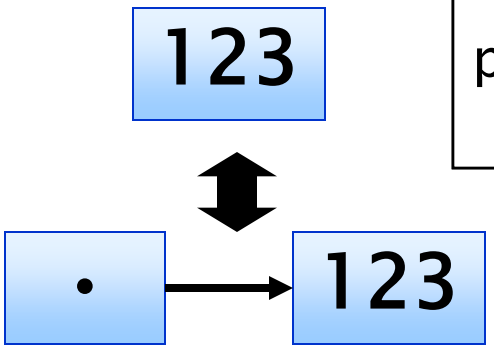
```
Bird b = a as Bird; // Convert

if (b == null)
      Console.WriteLine("Not a bird");
```

Boxing and Unboxing

- Unified type system
- Boxing
- Unboxing
- Calling object methods on value types

```
int p = 123;
object box;
box = p;
```




```
p = (int)box;
```

Boxing and Unboxing (Primer)

```
class List
 {
        public void add(object x);
        public object remove();
        ...
}

List a = new List();
a.add(1);
...
int x = (int)a.remove();
```

Generic Class

```
public class NonGeneric
{
   object item;
   public object getItem()
   {
        return item;
   }
}
```

```
public class Generic<T>
{
    T item;
    public T getItem()
    {
        return item;
    }
}
```

```
public class MyClass()
{
    NonGeneric ng = new NonGeneric ();
    Generic<int> g = new Generic<int>();
}
```

Generic Class

- Prednosti
  - Type checking, no boxing, no downcasts
  - Povećana reupotrebljivost (tipizirane kolekcije)
- Kako su implementirane generičke klase?
  - Provera pri deklaraciji, ne instanciranju
  - Instanciraju se u vreme izvršavanja,
  - ne kompajliranja
  - Precizna run-time type information
  - Mogu se koristiti i za reference i za value tipove

Generic Methods

```
static void AddMultiple<T>(List<T> list, params T[]
  values)
{
  foreach (T value in values)
      list.Add(value);
}

void MyMethod()
{
  List<int> list = new List<int>();
  AddMultiple<int>(list, 2, 4, 6);
}
```