C# Language & .NET Platform
9th Lecture:
C# Type System

http://d3s.mff.cuni.cz/~jezek

Pavel Ježek

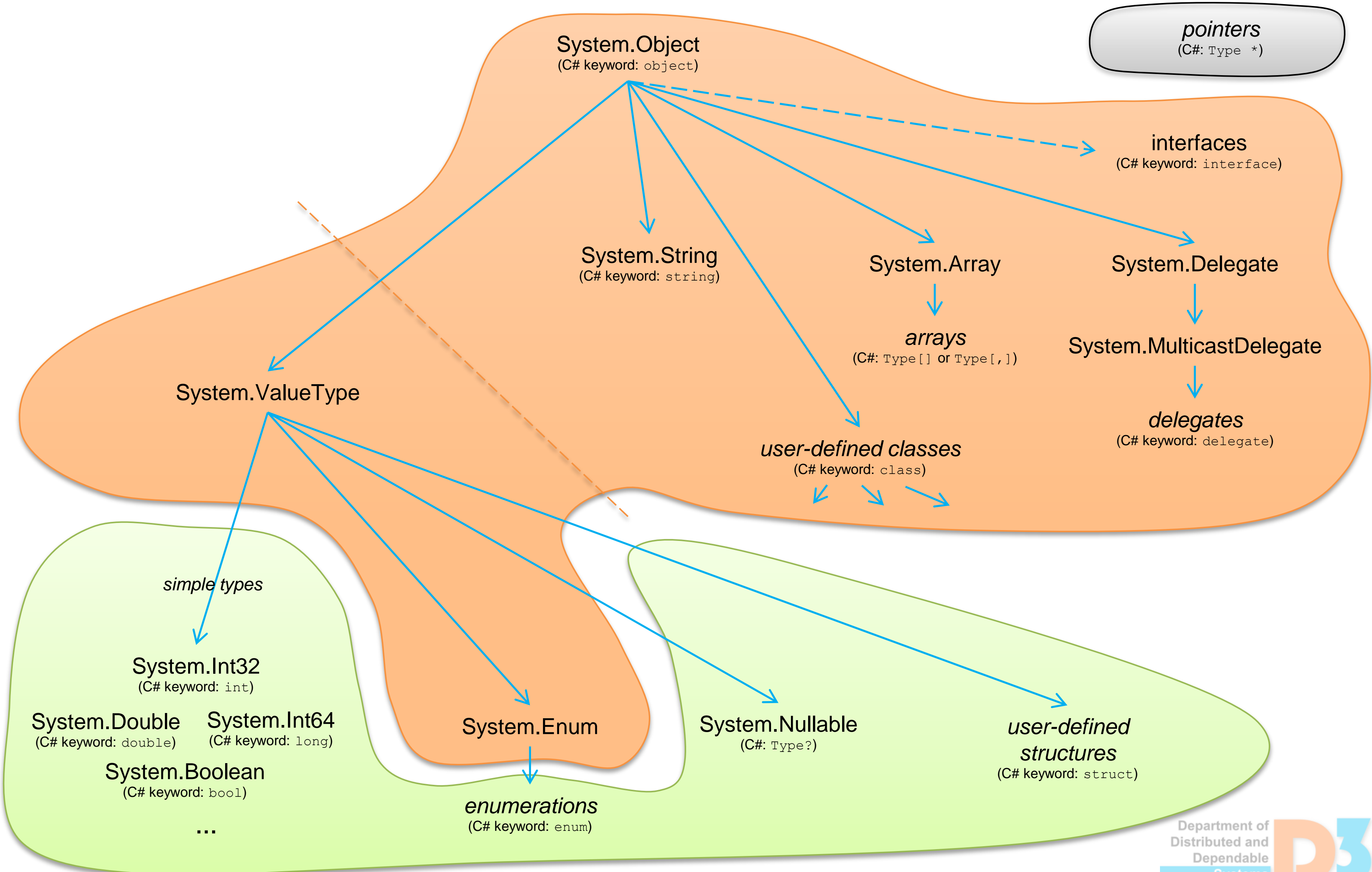
pavel.jezek@d3s.mff.cuni.cz

CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

Some of the slides are based on University of Linz .NET presentations.
© University of Linz, Institute for System Software, 2004
published under the Microsoft Curriculum License
(http://www.msdnaa.net/curriculum/license_curriculum.aspx)

CLI Type Inheritance



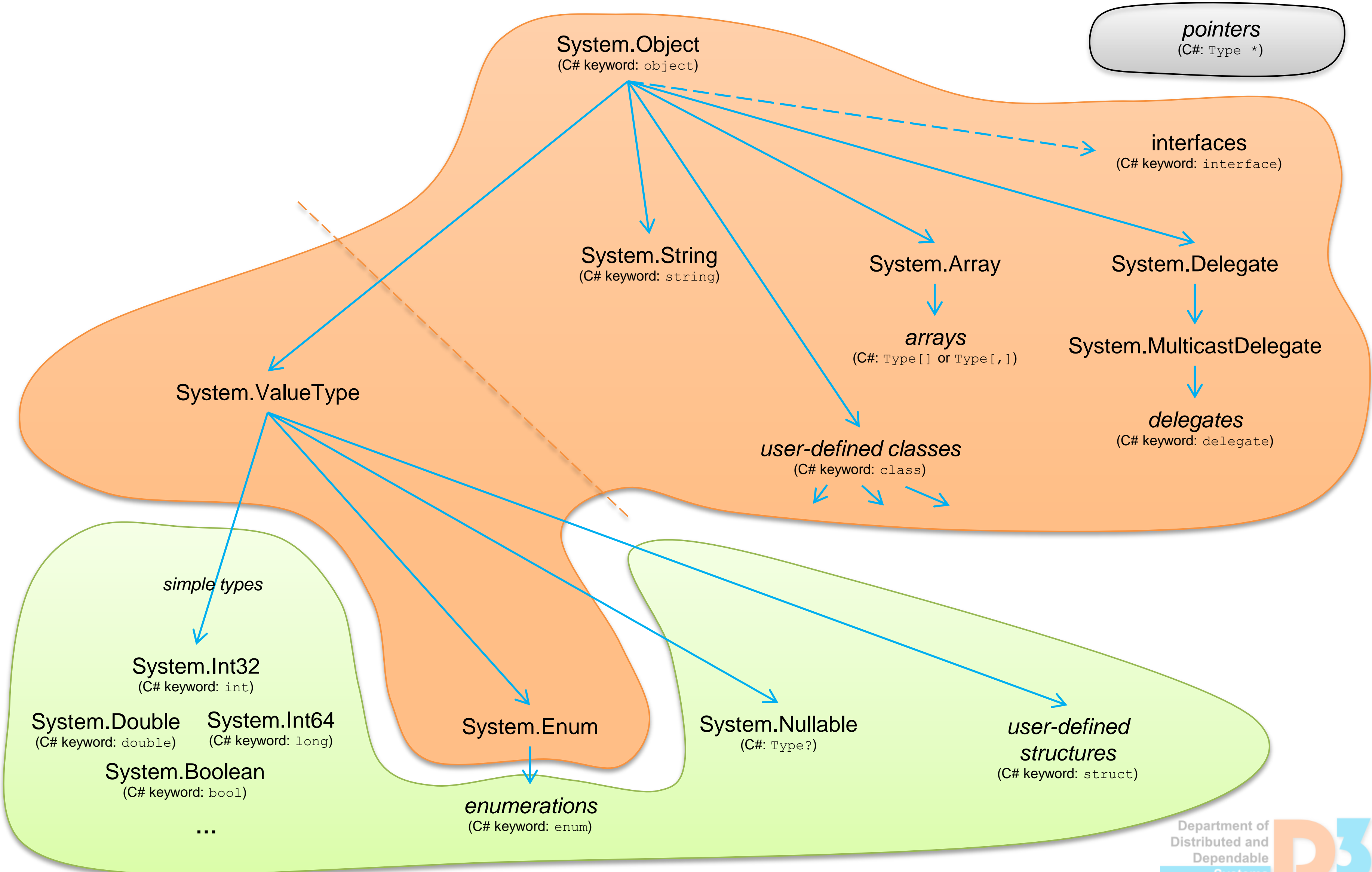
Virtual and Non-virtual Methods

Method Type	C++	Java	C#
virtual (new VMT entry)	`virtual`	*nothing*	`virtual`
virtual (reuse existing VMT entry)	`virtual`	*nothing* (optional @`override` annotation)	`override`
non-virtual	*nothing*	not supported/`final` (is similar to non-virtual in some scenarios)	*nothing*

Java `final` = C# `sealed`

Department of Distributed and Dependable Systems

CLI Type Inheritance



Arrays

One-dimensional arrays

```
int[] a = new int[3];
int[] b = new int[] {3, 4, 5};
int[] c = {3, 4, 5};
SomeClass[] d = new SomeClass[10];       // array of references
SomeStruct[] e = new SomeStruct[10];     // array of values (directly in the array)
```

Multidimensional arrays (jagged)

```
int[][] a = new int[2][];          // array of references to other arrays
a[0] = new int[] {1, 2, 3};        // cannot be initialized directly
a[1] = new int[] {4, 5, 6};
```

Multidimensional arrays (rectangular)

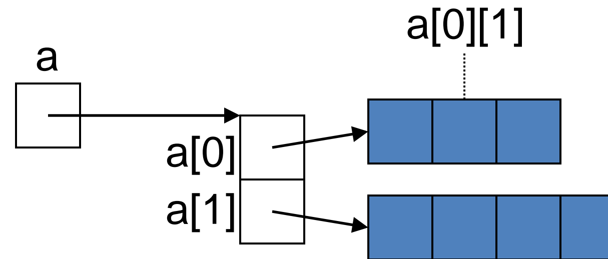
```
int[,] a = new int[2, 3];          // block matrix
int[,] b = {{1, 2, 3}, {4, 5, 6}}; // can be initialized directly
int[,,] c = new int[2, 4, 2];
```

Department of Distributed and Dependable Systems
D3S

Multidimensional Arrays

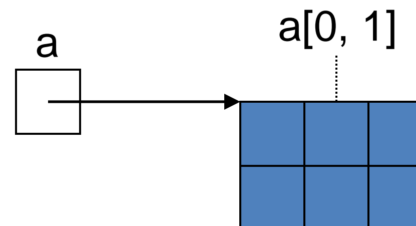
Jagged (like in Java)

```
int[][] a = new int[2][];
a[0] = new int[3];
a[1] = new int[4];

int x = a[0][1];
```



Rectangular (like in C/C++)

```
int[,] a = new int[2, 3];

int x = a[0, 1];
```

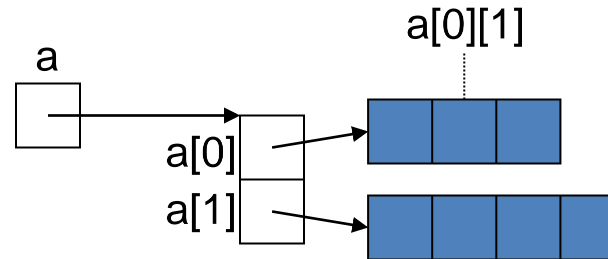


Department of Distributed and Dependable Systems
D3S

Multidimensional Arrays

Jagged (like in Java)

```
int[][] a = new int[2][];
a[0] = new int[3];
a[1] = new int[4];

int x = a[0][1];
```



Rectangular (like in C/C++)

```
int[,] a = new int[2, 3];

int x = a[0, 1];
```

Antipattern in current .NET (slower access)

Department of Distributed and Dependable Systems

Other Array Properties

Indexes start at 0

Array length

```
int[] a = new int[3];
Console.WriteLine(a.Length);  // 3

int[][] b = new int[3][];
b[0] = new int[4];
Console.WriteLine("{0}, {1}", b.Length, b[0].Length);   // 3, 4

int[,] c = new int[3, 4];
Console.WriteLine(c.Length);   // 12
Console.WriteLine("{0}, {1}", c.GetLength(0), c.GetLength(1));  // 3, 4
```

System.Array provides some useful array operations

```
int[] a = {7, 2, 5};
int[] b = new int[2];
Array.Copy(a, b, 2);                // copies a[0..1] to b
Array.Sort(b);
...
```

Department of Distributed and Dependable Systems
D3S

Problems Without Generic Types

Assume we need a class that can work with arbitrary objects

```
class Buffer {
    private object[] data;
    public void Put(object x) {...}
    public object Get() {...}
}
```

Problems

- Type casts needed

```
buffer.Put(3);                // boxing imposes run-time costs
int x = (int) buffer.Get();   // type cast (unboxing) imposes run-time costs
```

- One cannot statically enforce homogeneous data structures

```
buffer.Put(3); buffer.Put(new Rectangle());
Rectangle r = (Rectangle) buffer.Get();  // can result in a run-time error!
```

- Special types IntBuffer, RectangleBuffer, ... introduce redundancy

Department of Distributed and Dependable Systems
D3S

Generic Class Buffer

generic type | placeholder type

```
class Buffer<Element> {
    private Element[] data;
    public Buffer(int size) {...}
    public void Put(Element x) {...}
    public Element Get() {...}
}
```

- works also for structs and interfaces
- placeholder type *Element* can be used like a normal type

Usage

```
Buffer<int> a = new Buffer<int>(100);
a.Put(3);                    // accepts only int parameters; no boxing
int i = a.Get();             // no type cast needed!
```

```
Buffer<Rectangle> b = new Buffer<Rectangle>(100);
b.Put(new Rectangle());      // accepts only Rectangle parameters
Rectangle r = b.Get();       // no typ cast needed!
```

Benefits

- homogeneous data structure with compile-time type checking
- efficient (no boxing, no type casts)