

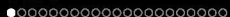
PV178: Programming for the CLI Environment

Seminar: Week 2

Tomáš Pochop, Michal Ordelt

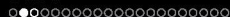
Institute of Computer Science and Faculty of Informatics
Masaryk University

March 11, 2007



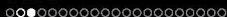
Seminar outline

- 1 Value vs reference types
- 2 Boxing and unboxing of value types
- 3 Method parameters in C#
- 4 C# and arrays



Value types

- 1 Value type objects direct contain the actual data in a variable
- 2 The variable each have their own copy of the data
- 3 It is not possible for operations on one to affect the other
- 4 Data are stored on stack, or on heap (if the value type variable is a field of a reference type)
- 5 `int`, `char`, `bool`, `struct`, `enum`, etc
- 6 example: `int i;`



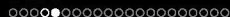
Reference types

- 1 Reference type variables store the reference of the actual data
- 2 Data for reference types is stored on the heap and a pointer (which points to the data on the heap) is created on the stack
- 3 When an instance of reference type is created, the pointer is returned back and is used to manipulate the data on the heap
- 4 It is possible for two variables to reference the same object, and thus possible for operations on one variable to affect the object referenced by the other variable
- 5 object, class, interface, delegate, array types
- 6 example: `MyClass cl = new MyClass();`
- 7 See the ValueRefType example



Class definition

```
1 class SimpleClass {
2     //some data field
3     double d;
4     //some constructor
5     SimpleClass()
6     {
7         //do something
8     }
9     //some method
10    void MyMethod()
11    {
12        //do something
13    }
14 }
```



Using a class

```
1 //declaration of a class
2 SimpleClass cl;
3 //instantiation of a class
4 cl = new SimpleClass();
5 //the MyMethod method call
6 cl.MyMethod();
```



Structure definition

```
1 struct SimpleStruct {
2     //some data field
3     double d;
4     //some constructor
5     SimpleStruct(double d)
6     {
7         //do something
8     }
9     //some method
10    void MyMethod(){
11        //do something
12    }
13 }
```



Using a structure

```
1 //declaration of a structure
2 SimpleStruct st;
3 //set some value to the d field
4 st.d = 5.4;
5 //the MyMethod method call
6 st.MyMethod();
```


Boxing

- 1 When a value type needs to behave like a reference type (value type is converted to the base object type or to an interface it implements)
- 2 CLR allocates memory from the managed heap to hold a copy of the value type instance
- 3 CLR then copies the value type instance to the newly allocated area in the heap

Unboxing

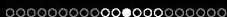
- 1 When an instance of an object type or interface, created as a result of boxing, is explicitly converted back to its true value type
- 2 CLR returns a pointer to the value type instance contained within the reference object
- 3 The unboxed instance is typically copied to a stack-based instance through an assignment operation
- 4 Returns value-type copy, not the heap-based instance
- 5 The boxed value still exists on the heap (after no further references to the object exist, Garbage collector reclaims the space)
- 6 See the `BoxUnbox` example

Method parameters in C#

- 1 Value parameters
- 2 Reference parameters
- 3 Output parameters
- 4 Parameter arrays

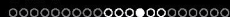
Value parameters

- 1 Correspond to a local variable that gets its initial value from the corresponding argument supplied in the method invocation
- 2 Any changes in method have no effect on the actual argument given in the method invocation
- 3 A new storage location is created and a value is copied into it
- 4 example: `void MyMethod(int onlyIn);`



Reference parameters

- 1 Represents the same storage location as the variable given as the argument in the method invocation
- 2 Any changes in method affects the actual argument given in the method invocation
- 3 The `ref` keyword must be used in both method declaration and method call
- 4 example: `void MyRefMethod(ref int inOut);`



Output parameters

- 1 Similar to a reference parameter, it represents the same storage location as the variable given as the argument in the method invocation
- 2 Every output parameter of a method must be assigned before the method ends
- 3 The `out` keyword must be used in both method declaration and method call
- 4 example: `void MyOutMethod(out int onlyOut);`

Parameter arrays

- 1 For variable parameters count
- 2 It must be the last in the parameters list and it must be of a 1D array type
- 3 cannot be combined with the ref or out parameter type
- 4 Considered as an input-only parameter
- 5 declaration example: `void MyParamMethod(params int[] values);`
- 6 using example: `MyParamMethod(i1, i2, i3);` where `i1,i2,i3` are `int`
- 7 See the Params example

Reference types - by value vs by reference

- 1 When a reference type is passed by value, its pointer is duplicated and this copy points to the same memory on the heap
- 2 When a reference type is passed by reference, the original pointer itself is passed to the called method
- 3 Any manipulation done to the object in the called method will manipulate the same data to which the original pointer was pointing in both cases
- 4 But any changes done to the pointer in the by-value case are applied to the copy, whereas in the by-reference case the changes are applied to the original pointer itself

Parameters example

Implement a class `ParamClass` that contains one integer data field `n` and methods below. Value of `n` is set in constructor via parameter.

- 1 `int ValMethod(int par1)` returns the parameter value increased by the value of `n`, but the variable passed to this method is not changed
- 2 `int RefMethod(int par2)` increases the parameter value by the value of `n` (affects the variable passed to this method) and returns its original value
- 3 `void OutMethod(int par3, int par4)` adds to the first parameter the value of `n` and stores it to the variable passed as the second parameter, the first parameter is not changed
- 4 `int ParMethod(int value)` returns the sum of several parameters, the number of parameters can change
- 5 Use proper parameter passing (complete the parameter list where needed)



Single dimension arrays

- 1 Arrays are objects, declaring an array doesn't create an array, it must be instantiated by using the `new` operator
- 2 Array items are index from 0 for the first item and $n-1$ for the last item (where n is number of item that the array can hold)
- 3 Array declaration example: `int[] arrayOfInt;`
- 4 Array that can holds five integers: `arrayOfInt = new int[5];`
- 5 Initialization: `arrayOfInt = new int[] {1,2,3,4,5,6,7}`

Multi dimension arrays

- 1 Two-dimensional array declaration example: `int[,] matrix;`
- 2 A 5x5 matrix: `matrix = new int[5,5];`
- 3 A 2x3 matrix initialization: `matrix = new int[,]{{1,2,3},{4,5,6}};`

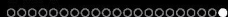


Array of arrays

- 1 Element of an array is an array
- 2 `int[] [] jagged = new int[2] [];`
- 3 `jagged[0] = new int[2]; jagged[1] = new int[3];`
- 4 `int[] [] jagged = new int[] [] {new int[] {1},
new int[] {2,3,}};`
- 5 See the Arrays example

Array example

Implement a structure `Book` that represents a book. `Book` contains two strings `author` and `title` for the books author and title, and one overridden method `ToString()`. This method returns the title and the author of this book as string. Next implement a class `Library` that represents a simple library. `Library` keeps an array of `Book` structures and contains two integer data fields, `capacity` for maximal books in the library and `actualBooks` for actual number of books in the library. `Library` has one constructor and two methods. The constructor has one integer parameter (through this parameter we can set `capacity`) and sets the array to keep `capacity` books and actual number of books sets to 0. The overridden method `ToString()` iterate through all books in library and returns information about them. If the library is not full, then in the `bool AddBook(Book)` method add a book to the end of the book list, update the count of books and return *true*. Otherwise perform nothing and return *false*.



Final Task

Write Console application that computes and displays rounded square root of a number. The number will be read from standard input. Check the user entered a number.

Useful methods :

```
public static double Math.Sqrt(double)
```

```
Boolean Char.IsDigit(Char)
```

```
Int64 Int64.Parse (String)
```

```
Int64 Convert.ToInt64(Double)
```

oooooooooooooooooooooooooooo