

C# Overview

Part II

Tomáš Matoušek
&
Ladislav Prošek

tmd.havit.cz/teaching/csharp.htm

Types

- type of each managed data must be known
- common supertype: `System.Object`
 - `ToString`, `GetType`, `Equals`, `GetHashCode`, `Finalize`, `MemberwiseClone`
- fundamental division
 - reference types
 - allocated on managed heap
 - accessed via a reference
 - value types
 - allocated on the stack or embedded to an object or array
 - simple data or compound structures
 - boxing converts a value to an object
 - sealed subtypes of `System.ValueType`
- primitive types
 - types known to a particular language compiler
 - C#
 - `System.Boolean` (bool), `System.Int32` (int), `System.Double` (double), `System.Char` (char), `System.String` (string), `System.Object` (object), ...

Boxing & Unboxing

- value storages
 - values allocated on stack
 - not managed by GC (however, GC knows about them)
 - lifetime during a method execution
 - freed when a stack frame is released
 - finalizer is not called
 - values embedded into objects or arrays
 - value lifetime corresponds to the object/array lifetime
 - not managed by GC, finalizer not called on release
 - boxed values on the managed heap
 - value is wrapped to an object allocated on heap
 - finalizer called if defined (which is not the case in C#)
 - implicit boxing when value is used as reference in C#

```
object o = 1;
```

- explicit unboxing

```
int a = (int)o;
```

Type Casting

- explicit cast
 - may lead to a compile-time or a run-time error
 - throws an `InvalidCastException` on run-time failure

```
string str = (string) obj;
```

- is operator
 - returns whether an object is of a specified type

```
bool b = obj is string;
```

- as operator
 - casts an object to a specified reference type
 - cannot be used on value types
 - returns a null reference if on failure

```
string str = obj as string;  
if (str != null) Console.WriteLine(str.ToLower());
```

Nested Types

- declared inside another type declaration
 - influences visibility only
 - no other relationship between enclosing and nested classes

```
public class A
{
    private static int x;
    private int m() { return 1; }

    private class B // nested class (a "friend" of A)
    {
        public f(A a) { return a.m() + x; }
    }
}
```

- Java inner classes emulation
 - inner class = nested class + field *outer* initialized in ctor

```
public class Outer
{
    private class Inner
    {
        private readonly Outer outer;
        public Inner(Outer outer) { this.outer = outer; }
    }
}
```

Interfaces

- contracts
- inheritance in CLR
 - single implementation inheritance
 - method implementations are inherited
 - multiple interface inheritance
 - methods have to be implemented unless type is abstract
- declaring an interface
 - interface keyword
 - contains
 - public virtual instance methods
 - CLR, not C#: static fields, methods, ctors, constants
 - can implement other interfaces
 - inherits a contract

Implementing Interfaces

- interface method implementation
 - same name & signature (param. names may differ)
 - has to be public (keyword required)
 - can be virtual or sealed (if virtual keyword missing)
- explicit interface method implementation
 - inherent problem of multiple inheritance
 - more methods having the same name and signature
 - implementations prefixed with interfaces
 - no access modifiers
 - not accessible via implementing class references (\Rightarrow private)
 - accessible only using reference typed to the interface (\Rightarrow public)
 - only interfaces directly implemented by the class are allowed here
 - another use:
 - hide implementation of an internal interface