An introduction to C#

oxygen

12. september 2017

Demo: Hello World

oxygen

Really simple. But you know that!

Design Goals

The big ideas

oxygen

- The first Component-oriented language in C-family
- Everything in C# is an object
- Designed to make robust and maintainable software

Component-oriented

oxygen

- First component-oriented language in the C-family
- Component conepts are first-class citizens
  - Properties, methods and events
  - Design-time and run-time attributes
  - Integrated documentation via XML
- Enables one-stop programming
  - No external files
    - Like: header files, IDL etc
  - Can be embedded in webpages

Everything is an object

oxygen

- Traditionally
  - C++ and Java:
    - Primitive types are ”magic” and are not considered objects
    - Do not interoperate with objects.
  - SmallTalk and Lisp:
    - Primitive types are object
    - Suffers a huge performance impact.
- C# unifies with little to no performance impact
  - Simplicity is the key
- Improved extensibility and reusability
  - New primitive types
    - Decimal, SQL, …
  - Collections work for all types

Robust and durable

oxygen

- Garbage collection
  - No memory leaks
  - No stray pointers
- Exceptions
  - Error handling is not an afterthought
- Type-safety
  - No uninitialized variables or unsafe casts
- Versioning

New and old…

oxygen

- C++ Heritage
  - Namespaces, enums, pointers (in unsafe code), unsigned types, etc.
  - No unnecessary sacrifices
- Real-world useful constructs
  - foreach, using, switch on string
  - decimal type for financial applications
  - ref and out parameters
- Millions of lines of C# code in .NET
  - Short learning curve
  - Increased productivity

Language Features

Structure

oxygen

- Namespaces
  - Contain types and other namespaces
- Type declarations
  - Classes, structs, interfaces, enums and delegates
- Members
  - Constants, fields, methods, properties, indexers, events, operators, constructors and destructors
- Organization
  - No header files
    - Code is written ”in-line”
  - No declaration or order dependence

Type system

oxygen

- Value types
  - Directly contains data
  - Cannot be null
- Reference types
  - Contains references to objects
  - Is nullable

Quiz: Which is is which

```
int i = 27;
string s = ”Nicolai is an awesome teacher”;
```

Type system (cont.)

oxygen

Value types:

- Primitives
  - int i;
- Enums
  - enum State { On, Off }
- Structs
  - struct Point { int x, y; }

Reference types:

- Classes
  - class Foo : Bar, IFoo {…}
- Interfaces
  - interface Ifoo : Ibar {…}
- Arrays
  - string[] a = new string[10];
- Delegates
  - delegate void Empty();

Predifined types

Reference

- object, string

Signed

- sbyte, short, int, long

Unsigned

- byte, ushort, uint, ulong

Character

- char

Floating point

- float, double, decimal

Logical

- bool

Predefined types are simply aliases for system-provided types

Classes

oxygen

- Single inheretance
- Multiple interface implementations
- Class members
  - Constants, fields, methods, properties, etc.
  - Static and instance members
  - Nested types
- Member access
  - Public, protected, internal and private

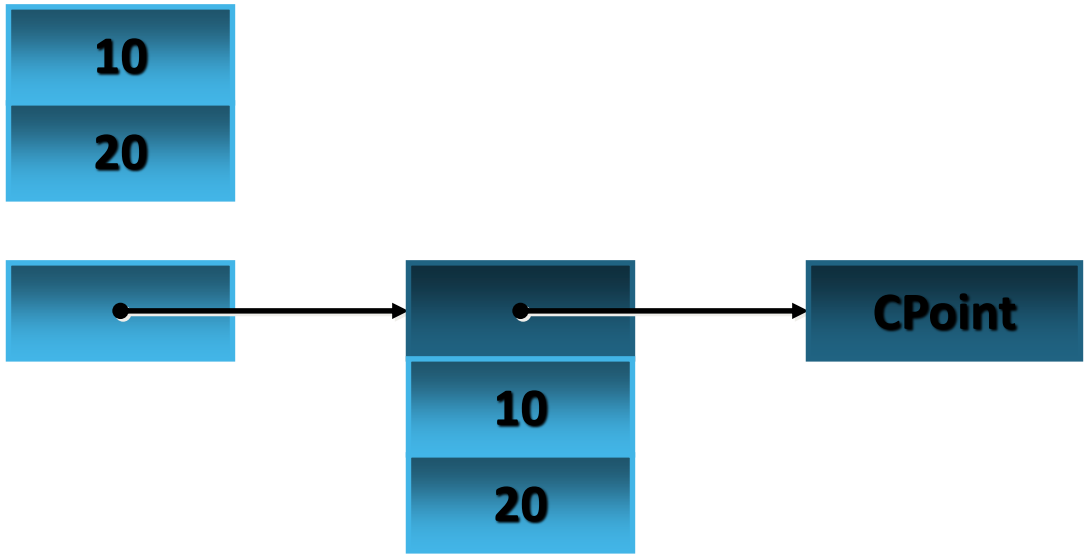
Structs

oxygen

- Like classes, except:
  - Stored inline, not heap-allocated
  - Assignment copies data (not referenced)
  - No inheretance
- Ideal for lightweight objects
  - Complex, Point, Rectangle, color, etc.
  - Int, float, double, etc. are all structs
- Benefits
  - No heap allocation
  - Less pressure on GC
  - More efficient use of memory

Classes and structs

oxygen



Interfaces

oxygen

- Multiple inheritance
- Can contain methods, properties, indexers and events
- Allows for private interface implementation

Enums

oxygen

- Strongly typed
  - No implicit conversions to/from int
- Can specify underlying type
  - Byte, short, int and long

Delegates

oxygen

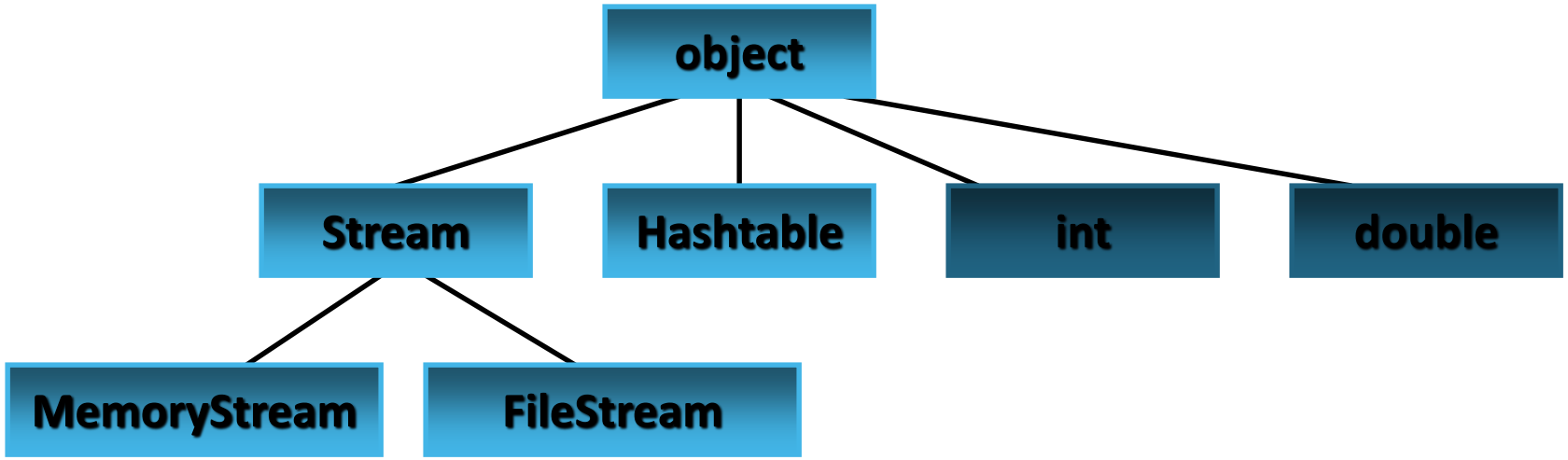
- Object oriented function pointers
- Multiple receivers
  - Each delegate has an invocation list
  - Thread-safe + and – operations
- Foundation for framework events

The unified type system!

oxygen

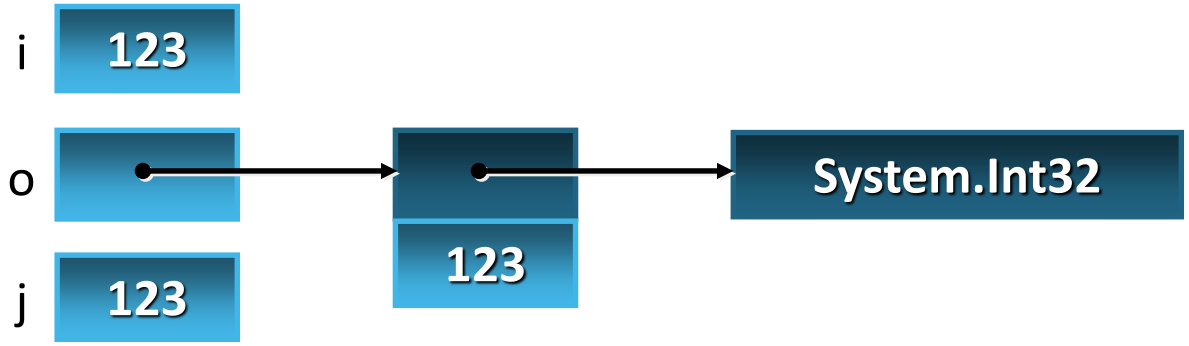
Everything is an object!!

- All types ultimately inherit from object



Boxing and unboxing

```
int i = 123;
object o = i;
int j = (int)o;
```



Benefits of the unified type system

oxygen

- Effectively eliminates wrapper classes
- Collection classes works with all types

Component development

oxygen

- What defines a component?
  - Properties, methods and events
  - Integrated help and documentation
  - Design-time information
- C# has first-class support
- Components are easy to build and consume

Properties

oxygen

Properties are smart fields

They have a natural syntax

They are easily accessed

Demo time? ☺

Events

oxygen

Define event signature

- public delegate void EventHandler(object sender, EventArgs e);

Define the Event and firing logic

- ```
  public class Button
  {
     public event EventHandler Click;
     protected void OnClick(EventArgs e) {
             if (Click != null) Click(this, e);
     }
  }
  ```
  ```

Events (cont.)

oxygen

Handling an event

```
public class MyForm: Form
{
   Button okButton;

   public MyForm() {
      okButton = new Button(...);
      okButton.Caption = "OK";
      okButton.Click += new EventHandler(OkButtonClick);
   }

   void OkButtonClick(object sender, EventArgs e)
      ShowMessage("You pressed the OK button");
   }
}
```

Attributes

oxygen

- Associate information with types and members
  - Documentation URL for a class
  - Transaction context for a method
  - XML persistence mapping
- Traditional solutions
  - Add keywords to language
  - Use external files
- Solution in C#
  - Attributes

XML Comments

oxygen

```
class XmlElement
{
   /// <summary>
   ///    Returns the attribute with the given name and
   ///    namespace</summary>
   /// <param name="name">
   ///    The name of the attribute</param>
   /// <param name="ns">
   ///    The namespace of the attribute, or null if
   ///    the attribute has no namespace</param>
   /// <return>
   ///    The attribute value, or null if the attribute
   ///    does not exist</return>
   /// <seealso cref="GetAttr(string)"/>
   public string GetAttr(string name, string ns) {
      ...
   }
}
```

Statements and expressions

oxygen

- if and while require a bool condition
- Switch
  - No fall-through, “goto case” or “goto default”
- Goto can’t jump into blocks
- For and Foreach

Operator overloading

oxygen

- First class user-defined data types
- Used in base class library
  - Decimal, DateTime, TimeSpan
- Used in the framework
  - Unit, point, rectangle
- Used in SQL integration

Operator overloading (cont.)

oxygen

```
public struct DBInt
{
   public static readonly DBInt Null = new DBInt();

   private int value;
   private bool defined;

   public bool IsNull { get { return !defined; } }

   public static DBInt operator +(DBInt x, DBInt y) {...}

   public static implicit operator DBInt(int x) {...}
   public static explicit operator int(DBInt x) {...}
}
```

```
DBInt x = 123;
DBInt y = DBInt.Null;
DBInt z = x + y;
```

Conditional compilation

oxygen

- #define, #undef
- #if, #elif, #else, #endif
  - Simple boolean logic
- Conditional methods

```
public class Debug
{
   [Conditional("Debug")]
   public static void Assert(bool cond, String s) {
      if (!cond) {
         throw new AssertionException(s);
      }
   }
}
```

oxygen

Questions?