Advanced Models and Programs:

C# 2.0
C# versus Java

Peter Sestoft

Mon 2011-01-31*

Agenda

- Overview of the course
- Crash course in C#
- Features C# and Java have in common
- Features in C# not in Java
- Features in Java not in C#

Some themes of the course

- Concepts of programming languages
  - Illustrated by differences between Java and C#
- C# and how it differs from Java
- Lexing, parsing and compilation
  - From source code to abstract syntax to stack machine code
- Functional programming in C#, Scala, Scheme
- Program generation and program specialization
- Extended static checking of programs
  - Like type systems but much stronger
- Static analysis and data-flow analysis

Course requirements

- Hand in weekly assignments, 10 in total
- At least 8 must be approved
- Do a project in groups of 2-3 students
- Oral exam in the project and some other course theme, week of June 20-24

- Some project examples:
  - Change the MicroC compiler to generate x86 code
  - Write interpreter for an Icon language subset
  - Implement data-flow analysis for a subset of some familiar language (e.g., Java, C#, ...)
  - Use JML on a realistic piece of software
  - Investigate Eclipse API tools
  - … many more possibilities

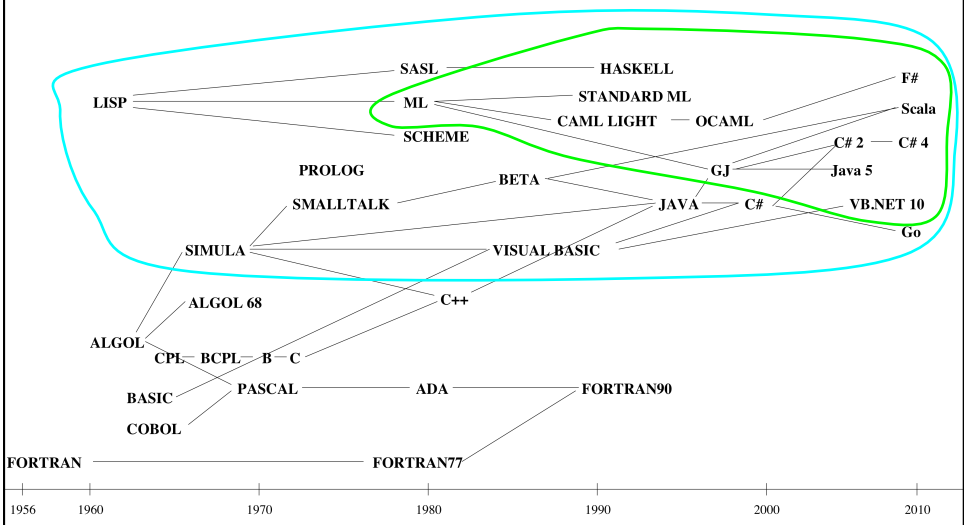
C# basic concepts

- Object-oriented, class-based, single-inheritance, virtual machine, garbage collector
- Very much like Java … even syntactically:

```
using System;
class Hello {
  static void Main(String[] args) {
    Console.WriteLine("Hello, " + args[0]);
  }
}
```

- But
  - Much faster language evolution, new cool features
  - Standardized (up to 2.0) by Ecma and ISO
  - But largely controlled by Microsoft
  - Implemented only by Microsoft and Mono/Novell

Programming language genealogy



Features in C# not in Java 5.0/6.0

- Properties and indexers
- Operator overloading and conversions
- Non-virtual method calls, ref. parameters
- Struct types, stack-allocated "objects"
- Nullable value types
- Iterator blocks (yield return and yield break)
- Delegate types and anonymous methods
- More powerful generic types and methods
- Possibility of unsafe code (disrecommended)

Features in Java not in C#

- Inner classes
- Checked exceptions
- Wildcards in generic types, use-side variance

- Java community:
  - Not much language evolution
  - Large framework development community

Different conventions

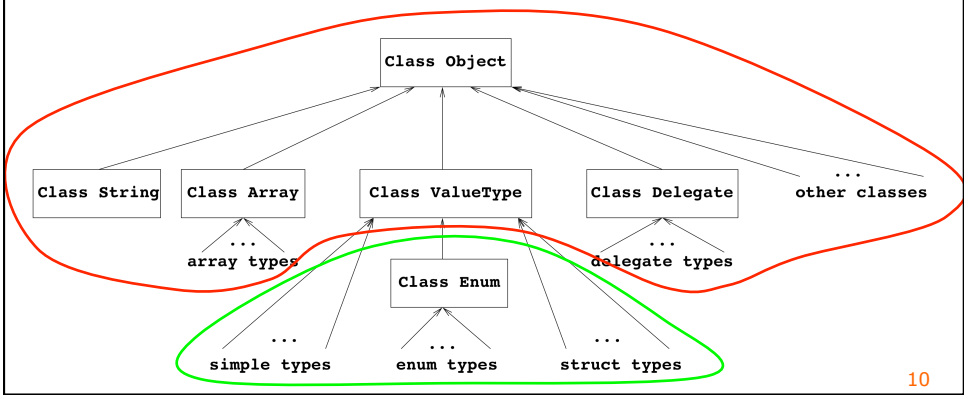
	Java	C#
Class	MyClass	MyClass
Interface	MyInterface	IMyInterface
Method	myMethod	MyMethod
Fields	myField	myField
Local variables	myVariable	myVariable
Package/namespace	java.lang.Reflect	System.Reflection

- For detailed C#/.NET conventions, see K. Cwalina and B. Abrams: Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2005.

9

C# value types and reference types

- Value types are int, long, double, … as in Java
  - plus user-defined struct types
- Reference types are classes, arrays, String as Java
  - plus delegate (function) types



10

A struct type is a value type

- A struct type is similar to a class
  - But is a value type, faster to allocate, copied on assignment, has no base class, no virtual method
  - Good for small values with *readonly* fields
  - Good for values with overloaded operators
  - Such as fractions, complex numbers, pairs, …

```
struct Frac {
  public readonly long n, d;        // Represents n/d
  public Frac(long n, long d) {
    long f = Gcd(n, d); this.n = n/f; this.d = d/f;
  }
  ...
  private static long Gcd(long m, long n) { ... }
}
Frac x = new Frac(15, 6);           // Represents 5/2
```

11

Example with structs and objects

```
public class Point {
  public int x, y;
}
```

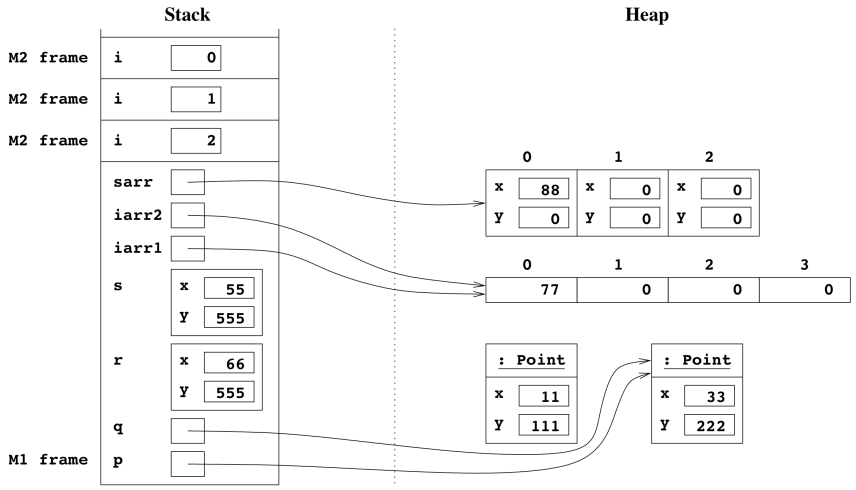
```
struct SPoint {
  public int x, y;
}
```

```
public static void M1() {
  Point p = new Point(11, 111), q = new Point(22, 222);
  p = q;
  p.x = 33;
  SPoint r = new SPoint(44, 444), s = new SPoint(55, 555);
  r = s;
  r.x = 66;
  int[] iarr1 = new int[4];
  int[] iarr2 = iarr1;
  iarr1[0] = 77;
  SPoint[] sarr = new SPoint[3];
  sarr[0].x = 88;
  M2(2);
}
public static void M2(int i) {
  if (i > 0)
    M2(i-1);
}
```

12

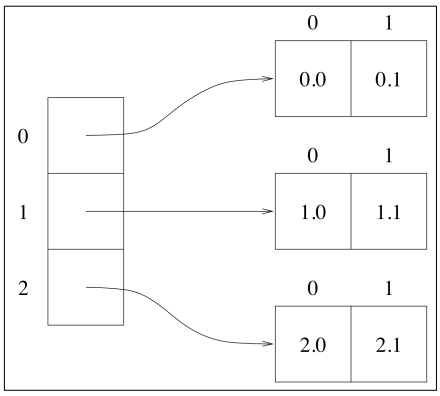
Machine model

- Objects in heap (like Java), structs on stack

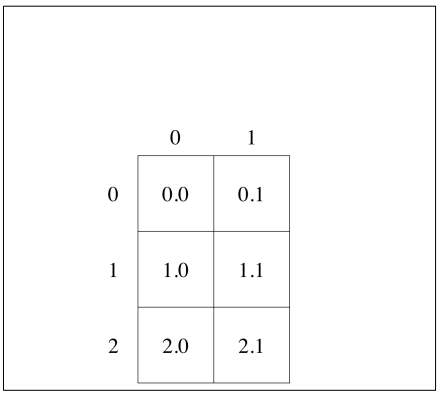


Rectangular arrays

- A Java "2D array" is an array of arrays
- C# also has 2D rectangular arrays, like C



Java/C# **`double[][]`**
an array of arrays



C# **`double[,]`**
a rectangular array

Using rectangular arrays

- Rectangular arrays
  - may use less space
  - may be faster because better storage locality
  - may be needed to interface to C/C++ code
- Two ways to allocate and initialize 2D array

```
double[,] r1 =
   { { 0.0, 0.1 }, { 1.0, 1.1 }, { 2.0, 2.1 } };

double[,] r2 = new double[3,2];
for (int i=0; i<3; i++)
  for (int j=0; j<2; j++)
    r2[i,j] = i + 0.1 * j;
```

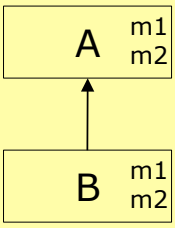

C# non-virtual instance methods

- In Java a method is either
  - static, or
  - a virtual instance method (non-static)
- C# furthermore has
  - non-virtual instance methods
  - (and other non-virtual methods, see later)

- Consider method call a.m()
- The called method is determined by
  - the *compile-time type* of a, if m is non-virtual
  - the *run-time class* of the value of a, if m is virtual

Examples: virtual and non-virtual

```
class A {
  public virtual void m1() { Console.WriteLine("A.m1()"); }
  public void m2() { Console.WriteLine("A.m2()"); }
}
class B : A {
  public override void m1() { Console.WriteLine("B.m1()");
  public new void m2() { Console.WriteLine("B.m2()"); }
}
class Override {
  static void Main(string[] args) {
    B b = new B();
    A a = b;
    a.m1();              // B.m1()
    b.m1();              // B.m1()
    a.m2();              // A.m2()
    b.m2();              // B.m2()
 }
}
```



The new and override modifiers are used to avoid accidental overriding and hiding

C# reference parameters

- In Java, a method argument is passed *by value*, similar to an assignment: **`x = a;`**

```
void m(int x) {
  ...
}
```

```
int a = ...
m(a);
```

- C# can also pass arguments *by reference*
  - Argument **`a`** must be variable, field, array access
  - The parameter **`x`** becomes *an alias* for **`a`**
- A reference parameter may be **`ref`** or **`out`**
  - **`ref`** is used for input and/or output
  - **`out`** is used for output only
  - Both are passed in exactly the same way

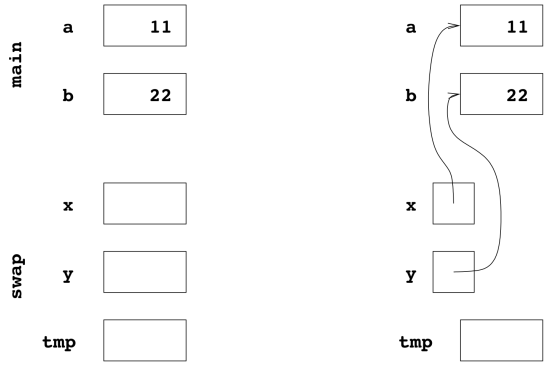
By-value versus by-reference

```
void swapV(int x, int y) {
  int tmp = x; x = y; y = tmp;
}

a = 11; b = 22;
swapV(a, b);
```

```
void swapR(ref int x, ref int y) {
  int tmp = x; x = y; y = tmp;
}

a = 11; b = 22;
swapR(ref a, ref b);
```



C# properties

- A *property* is neat syntax for getFoo, setFoo

```
class Lifo {
  private double[] stack = ...;
  private int sp = ...;
  public int Count {
    get { return sp+1; }
  }
  public double Top {
    get { return stack[sp]; }
    set { stack[sp] = value; }
  }
}
```

get_Count

get_Top

set_Top

```
Console.WriteLine(lifo.Count);
Console.WriteLine(lifo.Top);
lifo.Top = 6.7;
```

C# indexers

- An indexer is an array-like property

```
class Lifo {
  private double[] stack = ...;
  private int sp = ...;
  public double this[int i] {
    get { return stack[sp-i]; }
    set { stack[sp-i] = value; }
  }
}
```

get_Item(int i)

set_Item(...)

```
Console.WriteLine(lifo[0]);
lifo[1] = 7.8;
```


C# operator overloading

- Operators ==, !=, +, *, … can be overloaded
- An operator is a public static method
- Works well with struct types (never null)

```
struct Frac {
  public readonly long n, d;      // Represents n/d
  public static bool operator==(Frac r1, Frac r2) {
    return r1.n==r2.n && r1.d==r2.d;
  }
  public static Frac operator+(Frac r1, Frac r2) {
    return new Frac(r1.n*r2.d+r2.n*r1.d, r1.d*r2.d);
  }
  public static Frac operator*(Frac r1, Frac r2) {
    return new Frac(r1.n*r2.n, r1.d*r2.d);
  }
}
```

```
Frac x, y, z;       ... x+y*z==x+z*y ...
```

C# user-defined conversions

- A special method that transforms values
- Implicit (automatic) or explicit (by cast)

```
struct Frac {
  public readonly long n, d;
  public Frac(long n, long d) { ... }
  public static implicit operator Frac(int n) {
    return new Frac(n, 1);
  }
  public static explicit operator double(Frac r) {
    return ((double)r.n)/r.d;
  }
}
```

```
Frac f1 = 42;                // Impl: int-->Frac
double d2 = (double)f2;      // Expl: Frac-->double
Frac f2 = f1 + 27;           // Impl: int-->Frac
```


C# delegate types and delegates

- A delegate type is a method (function) type

```
delegate bool IntPredicate(int x);
```

- A delegate value is a method (static or not)

```
static bool Even(int x)
  { return x%2 == 0; }
static bool PrintBig(int x)
  { Console.WriteLine(x); return x > 100; }
```

```
IntPredicate m = Even;
Console.WriteLine(m(4));
```

- Multicast: Delegate may hold multiple meth.
- This is used for event handling in GUIs

```
m += PrintBig;
m += PrintBig;
Console.WriteLine(m(7));     // Prints: 7 7 False
```

C# enum types

- An enum is an integer value (unlike in Java)

```
public enum Month {
  Jan=1, Feb, Mar, Apr, May, Jun,
    Jul, Aug, Sep, Oct, Nov, Dec
}
```

- You can do arithmetics on enums
- May produce a value not corresponding to a declared enum

```
Month m = Month.Sep;
m += 42;       // Perfectly legal
```

yrk!

C# foreach statement

- Much like Java "enhanced for-statement"
- Works over types that implement IEnumerable<T>, including arrays

```
double[] arr = { 9.213, 91.345, 410.0, 323.5, 930.25 };
int sum = 0;
foreach (int v in arr)
  sum += v;
```

Cast from double to int

IDisposable and the using statement

```
interface IDisposable {
  void Dispose();
}
```

- Using statement: neat pattern for cleanup

```
using (t x = e)
  body
```

=

```
t x = e;
try {
  body
} finally {
   if (x != null)
     ((IDisposable)x).Dispose();
}
```

- Neat for closing files: `Dispose()` calls `Close()`

```
using (TextReader reader = new StreamReader("diskfile")) {
  ... double.Parse(reader.ReadLine()) ...
}
```


C# variable-arity methods

- `m(params int[] xr)` vs. Java `m(int... xr)`
- Example: maximum of 1 or more numbers

```
static int Max(int x1, params int[] xr) {
  int res = x1;
  foreach (int x in xr)
    if (x>res)
      res;
  return res;
}
```

```
... Max(69, 42) ...
... Max(2, 5, 7, 11, 3) ...
... Max(2, new int[] { 5, 7, 11, 3 })) ...
```

Namespaces – not packages

- Namespace contains types and namespaces
- Controls accessibility (=Java packages)
- Opened with **`using`** decl. (=Java **`import`**)
- Does not reflect source file hierarchy (≠Java)
- A namespace may have multiple chunks

```
using System;
namespace N1 {
  public class C11 { ... }
  namespace N2 {
    public class C121 { }
  }
}
class C1 { }                                    // Internal
namespace N1 { public struct S13 { } }
namespace N1.N2 { internal class C122 { } }
```

29

C# string formatting

- Rich facilities, different from Java/C style:

```
for (int c=2; c<=12; c++)
  Console.WriteLine("{0} came up {1} times", c, freq[c-1]);
```

```
2 came up 264 times
3 came up 572 times
4 came up 797 times
5 came up 1188 times
6 came up 1426 times
7 came up 1723 times
8 came up 1353 times
9 came up 1086 times
10 came up 767 times
11 came up 558 times
12 came up 266 times
```

Good

```
 2 came up  264 times
 3 came up  572 times
 4 came up  797 times
 5 came up 1188 times
 6 came up 1426 times
 7 came up 1723 times
 8 came up 1353 times
 9 came up 1086 times
10 came up  767 times
11 came up  558 times
12 came up  266 times
```

Better

```
for (int c=2; c<=12; c++)
  Console.WriteLine("{0,2} came up {1,4} times”,
                    c, freq[c-1]);
```

30

Compiling and running C# programs

- Visual Studio 2010 (C# 4.0) in classrooms
  - Start > MS Visual Studio 2010 > Visual Studio
  - Start > MS Visual Studio 2010 > Tools > Prompt
- Or download your own from MSDNAA
  - Contact IT dept (in 2C) about MSDNAA access
  - VS2010
- Or download Visual C# Express 2010
- Or use the Mono 2.8 implementation
  - From www.mono-project.com
  - For Windows, Linux, MacOS and more

Command line C# compiler

- Microsoft .NET on MS Windows:

```
Z:examples>csc Example001.cs
Microsoft C# 2010 Compiler version 4.0.30319.326

Z:examples>Example001 7 9 13
The sum is 29
```

- The ".exe" file loads the .NET runtime
- Mono on Linux, MacOSX, MS Windows:

```
sestoft@mac examples $ mcs Example001.cs
sestoft@mac examples $ mono Example001.exe 7 9 13
The sum is 29
```

Java inner classes, not in C#

- An instance inner of class Outer.Inner holds a reference to an instance of class Outer

```
class Outer {
  int f;
  static int sf;
  static class Nested {
    // Can refer sf not f
  }
  class Inner {
    int g = f + 2;
  }
}
```

Java

```
class Outer {
  int f;
  static int sf;
  class Nested {
    // Can refer sf not f
  }
}
```

C#

Nested class, no inner classes

Java checked exceptions, not in C#

- A Java method must declare the checked exceptions it can throw

```
static double[] readNumbers(String column)
  throws IOException, SQLException
  { ... }
```

- C# does not require nor allow **`throws`** clauses
- A big relief
- But hard to know in what ways a method may fail, and hard to handle all errors

Java wildcards in generics, not in C#

- Wildcards express use-site variance in generic types
- C# does not has use-site variance
- C# 4.0 has declaration-site variance (later)

Now what

- Wed 2 Feb: More advanced C# 3 features
- Mon 7 Feb: C# 4 features
- Wed 9 Feb: Scala = functional and object-oriented programming combined

- You should:
  - Get and install a C# implementation, either VS2010 or Visual C# Express or Mono 2.8
  - Start doing Exercise sheet 1
  - Hand in solutions no later than Friday 11 Feb