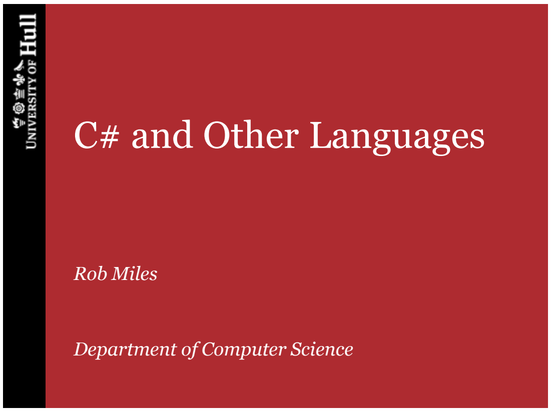


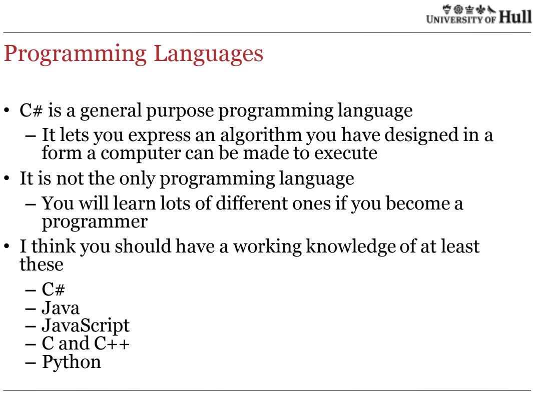
C# and Other Languages

Rob Miles

Department of Computer Science

Why do we have lots of Programming Languages?

- Different developer audiences
- Different application areas/target platforms
  – Graphics, AI, List Processing…
- Different priorities
  – Fast, small, portable, bomb proof…
- Marketing
  – Get developers onto your platform by supporting a good language



Programming Languages

- C# is a general purpose programming language
  – It lets you express an algorithm you have designed in a form a computer can be made to execute
- It is not the only programming language
  – You will learn lots of different ones if you become a programmer
- I think you should have a working knowledge of at least these
  – C#
  – Java
  – JavaScript
  – C and C++
  – Python

HEALTH WARNING

- The content here is a bit subjective, as it is impossible to talk about this kind of thing without letting your preferences show through
- If you ask other people about these issues you will get slightly different answers from the ones that I'm going to give
- However, of course, everyone else is wrong…….

Java Origins

- Invented by Sun Microsystems
  - (who have been bought by Oracle)
- Originally intended for use in "Set Top Boxes"
- Needed a language that was portable across a wide range of devices
- Also needed a way to ensure that programs did not "crash" the hardware
- Uses a "Virtual Machine" to execute code

Computer Hardware

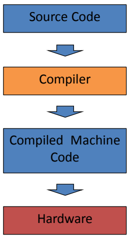
- Programs are executed by hardware
- This provides storage, input/output and a processor (cpu)
- The processor will have a particular design (Pentium, ARM, etc)
  - A certain arrangement of internal registers
  - A certain set of physical instructions
- A particular compiled "binary" program will work on a particular processor

Virtual Machine

- Rather than target a specific platform (Pentium, ARM, PowerPC) you design a “Virtual Machine”
- This has an arrangement of registers and memory, like a real processor, but it is implemented in software
- Any platform that has a program that implements the Virtual Machine can run programs written for it

Conventional Compiler Model

- Source code is compiled to produce an executable file which contains machine code instructions for the target hardware
- The hardware then obeys these instructions to execute the program

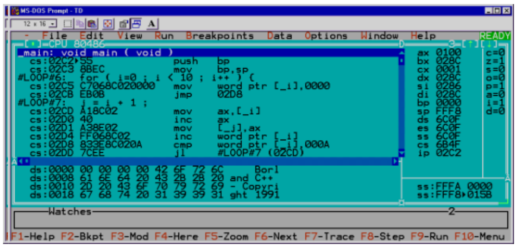

Source Code

Compiler

Compiled Machine Code

Hardware

Compiled Code

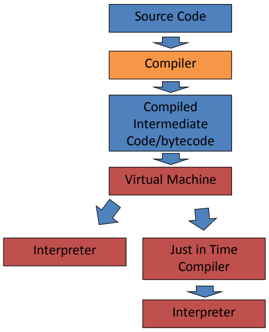

MS-DOS Prompt - TD
12 x 16
A
- File Edit View Run Breakpoints Data Options Window Help READY
[■]-CPU 80486
Watches
2
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

```
_main: void main ( void )                                 ax 0100   c=0
  cs:02C2▸55              push   bp                       bx 028C   z=1
  cs:02C3 8BEC            mov    bp,sp                    cx 0001   s=0
#LOOP#6:   for ( i=0 ; i < 10 ; i++ ) {                   dx 028C   o=0
  cs:02C5 C7068C020000    mov    word ptr [_i],0000       si 0286   p=1
  cs:02CB EB0B            jmp    02D8                     di 028C   a=0
#LOOP#7:   j = i + 1 ;                                    bp 0000   i=1
  cs:02CD A18C02          mov    ax,[_i]                  sp FFF8   d=0
  cs:02D0 40              inc    ax                       ds 6C0F
  cs:02D1 A38E02          mov    [_j],ax                  es 6C0F
  cs:02D4 FF068C02        inc    word ptr [_i]            ss 6C0F
  cs:02D8 833E8C020A      cmp    word ptr [_i],000A       cs 6B4F
  cs:02DD 7CEE            jl     #LOOP#7 (02CD)           ip 02C2

  ds:0000 00 00 00 00 42 6F 72 6C     Borl
  ds:0008 61 6E 64 20 43 2B 2B 20 and C++
  ds:0010 2D 20 43 6F 70 79 72 69 - Copyri               ss:FFFA 0000
  ds:0018 67 68 74 20 31 39 39 31 ght 1991               ss:FFF8▸015B
```

Virtual Machine Model

- Source code is compiled to an “intermediate code” for a “virtual machine”
- When the program runs this is either interpreted or compiled again by a “Just In Time” compiler
- The code runs in a “Managed” environment



Source Code

Compiler

Compiled Intermediate Code/bytecode

Virtual Machine

Interpreter

Just in Time Compiler

Interpreter

Interpreting a Program

- An interpreter decodes each step of a intermediate code, performs the requested action and then moves on to the next step
- The steps of the program are never converted into machine code, they are just executed by the interpreter program
- The interpreter itself is not tied to the underling hardware
- Languages that run this way are sometimes called “scripting” languages
  - Python runs in this way

Interpreters

- Good because:
  - Easy to write
  - Very easy to move from one platform to another
  - Very safe, the program never gets control of the hardware
- Bad because:
  - Slow
  - Can’t take advantage of hardware features

Just in Time Compilation

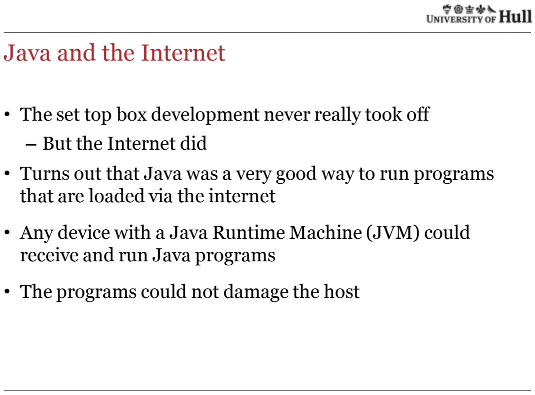
- The other way to make a Virtual Machine run programs is to compile the intermediate code into machine code just before it executed
- This is called “Just In Time” compilation
- When you run your program it is compiled into machine code just before it is run
- This is performed a method at a time
- Methods that are never called are never compiled

Just in Time Compilation

- Good Because
  - Should get the same performance as a “properly” compiled program
  - Can make a compiler for each platform
- Bad Because
  - Slows down your program starting up as it has to compile your program before it can do anything

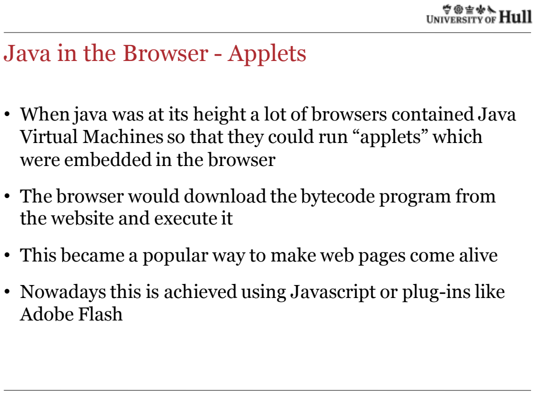
Managed Code

- One of the other reasons for creating a virtual machine is that it allows you to run a “managed code” environment
  - Programs that run directly on the hardware can contain instructions that may break the underlying system
- Managed code provides a wrapper around the program that stops it doing bad things
- Both C# and Java run programs in a managed environment



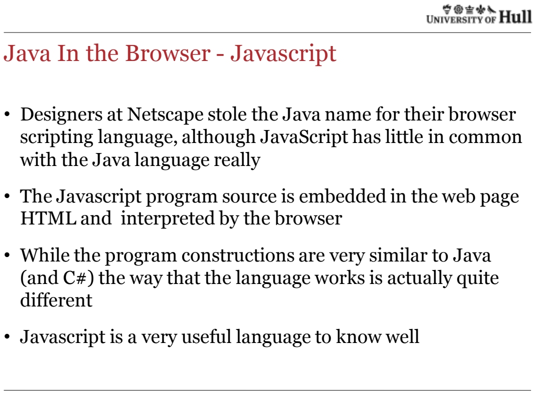
Java and the Internet

- The set top box development never really took off
  - But the Internet did
- Turns out that Java was a very good way to run programs that are loaded via the internet
- Any device with a Java Runtime Machine (JVM) could receive and run Java programs
- The programs could not damage the host



Java in the Browser - Applets

- When java was at its height a lot of browsers contained Java Virtual Machines so that they could run “applets” which were embedded in the browser
- The browser would download the bytecode program from the website and execute it
- This became a popular way to make web pages come alive
- Nowadays this is achieved using Javascript or plug-ins like Adobe Flash



Java In the Browser - Javascript

- Designers at Netscape stole the Java name for their browser scripting language, although JavaScript has little in common with the Java language really
- The Javascript program source is embedded in the web page HTML and interpreted by the browser
- While the program constructions are very similar to Java (and C#) the way that the language works is actually quite different
- Javascript is a very useful language to know well

Java Code

```
/**
 * The HelloWorldApp class implements an application
 * that prints "Hello World!" to standard output.
 */
class HelloWorldApp {
    public static void main(String[] args) {
        // Display the string.
        System.out.println("Hello World!");
    }
}
```

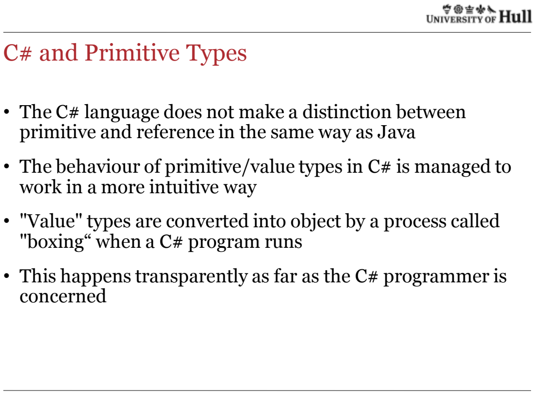
- Java looks incredibly like C#
- This is because both languages are based on the syntax of C++
- There are some differences when using class hierarchies, but the principles are the same

Java and C# Differences

- Java has "primitive" data types as well as objects
  - Primitive types are a way of speeding up program execution
- C# is just one of several languages that run on top of the same Virtual Machine
  - This is all part of the .NET Framework
- C# programs cannot run as applets

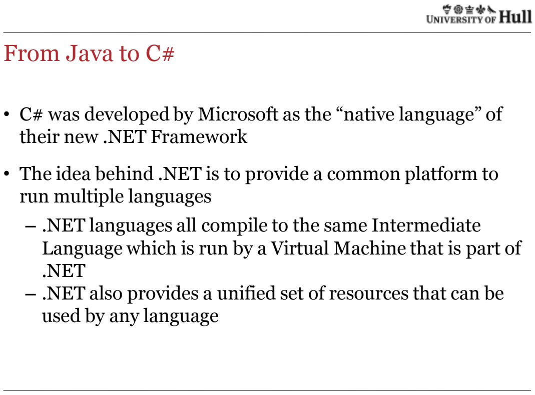
The Java primitive type

- A Java primitive type is **not** an object
- It cannot expose methods
- It is managed by value
- If you want primitive types which can do something these are provided in the form of "wrapper classes" which are object based implementations of the primitive types
- There are both int and Integer types in Java



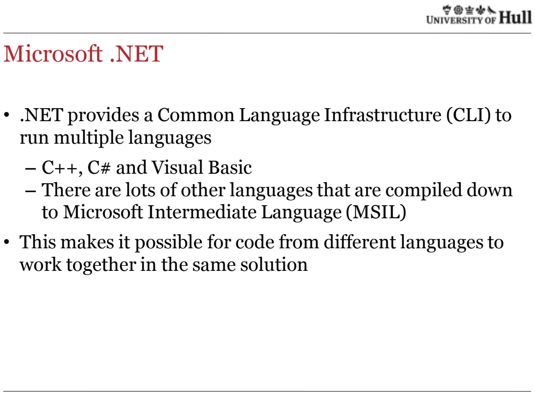
C# and Primitive Types

- The C# language does not make a distinction between primitive and reference in the same way as Java
- The behaviour of primitive/value types in C# is managed to work in a more intuitive way
- "Value" types are converted into object by a process called "boxing“ when a C# program runs
- This happens transparently as far as the C# programmer is concerned



From Java to C#

- C# was developed by Microsoft as the “native language” of their new .NET Framework
- The idea behind .NET is to provide a common platform to run multiple languages
  - .NET languages all compile to the same Intermediate Language which is run by a Virtual Machine that is part of .NET
  - .NET also provides a unified set of resources that can be used by any language



Microsoft .NET

- .NET provides a Common Language Infrastructure (CLI) to run multiple languages
  - C++, C# and Visual Basic
  - There are lots of other languages that are compiled down to Microsoft Intermediate Language (MSIL)
- This makes it possible for code from different languages to work together in the same solution

Common Language Infrastructure (CLI)

- This is the system which underpins the execution of the Intermediate Language code
- It is designed to be “language agnostic” and provide a platform capable of executing compiled code from a range of source languages
- It should also allow these components to interact in a useful manner

CLI Features

- The CLI must work as an operating system
  - Loads and executes components
  - Provides Memory Management and IO
- The CLI must work as a compiler/linker/loader
  - Place objects in memory
  - Compile code
  - Resolve references

CLI Concepts: Unified Types

- The CLI must provide a set of types which are used by compiled programs
- Types contain fields and properties which contain the data for that type
- The structure of a type is presented as *metadata*
- The CLI will load types as they are needed

Unified Types: int32 in C#

```
public static int WorkOutFact ( int invalue ) {
  int result = 1;
  . . . . . .
```

```
.method public static int32  WorkOutFact(int32 invalue)
cil managed
{
  // Code size       24 (0x18)
  .maxstack  2
  .locals ([0] int32 i,
           [1] int32 result)
 IL_0000:  ldc.i4.1
 IL_0001:  stloc.1
. . . . . . .
```

Unified Types: int32 in VB

```
  Public Shared Function WorkOutFact
                  (ByVal invalue As Integer) As Integer

        Dim result As Integer
        Dim i As Integer
        result = 1
```

```
.method public static int32  WorkOutFact(int32 invalue) cil
managed
{
  // Code size       28 (0x1c)
  .maxstack  2
  .locals init ([0] int32 i,
           [1] int32 result,
           [2] int32 WorkOutFact,
           [3] int32 _Vb_t_i4_0)
  IL_0000:  nop
  IL_0001:  ldc.i4.1
  IL_0002:  stloc.1
. . . . . . .
```

Unified Types

- Each language implementation “agrees” on the size and orientation of the types within the program
- This makes it possible for the languages to interoperate in a useful way
- Types constructed in a given language are also described in meta-data which makes it possible for them to be linked with types from others
- The CLI should be unaware of the language origins of a program component

Common Conventions

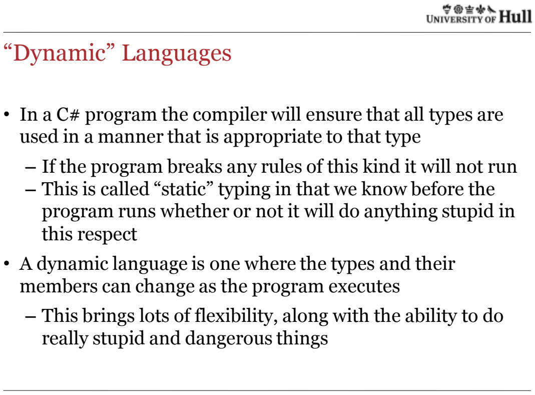
- In the case of .NET all the available languages must be forced to use the same convention, that of the CLI
- Note that this does not mean that the programs will necessarily execute this way
  - in some implementations the top few positions on the stack can be mapped onto processor registers
- This may impact on portability, but is only really an issue with un-managed code

Interacting with Native Code

- Native code is the machine code of the host processor
- The types used in the CLI are designed to be easily mapped onto "native" code
- This is reflected in the range of built in types supported in the CLI
- C# has this ability “built in”
- You can write C# programs that interact directly with the hardware
  - These must be flagged as “unsafe”

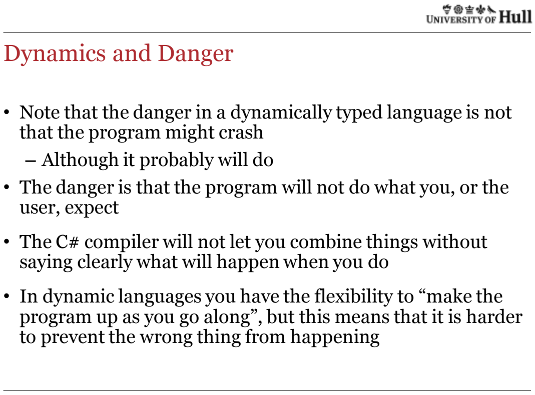
Java and C# Summary

- Both execute on Virtual Machines in a Managed Enviroment
- Both are based on C/C++ syntax
- Both are strongly typed
- Both are object oriented and provide inheritance and interfaces
- Both provide a managed code environment (although C# lets you turn this off)
- Both have a large support library



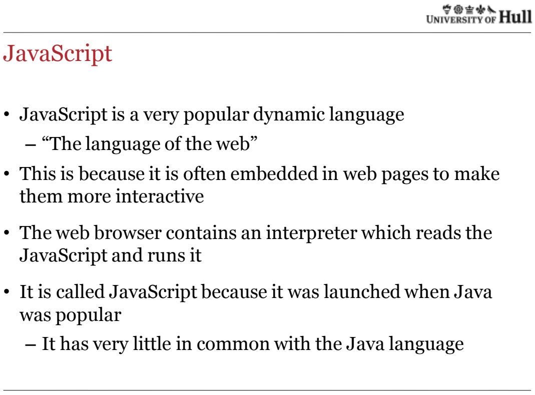
“Dynamic” Languages

- In a C# program the compiler will ensure that all types are used in a manner that is appropriate to that type
  - If the program breaks any rules of this kind it will not run
  - This is called “static” typing in that we know before the program runs whether or not it will do anything stupid in this respect
- A dynamic language is one where the types and their members can change as the program executes
  - This brings lots of flexibility, along with the ability to do really stupid and dangerous things



Dynamics and Danger

- Note that the danger in a dynamically typed language is not that the program might crash
  - Although it probably will do
- The danger is that the program will not do what you, or the user, expect
- The C# compiler will not let you combine things without saying clearly what will happen when you do
- In dynamic languages you have the flexibility to “make the program up as you go along”, but this means that it is harder to prevent the wrong thing from happening



JavaScript

- JavaScript is a very popular dynamic language
  - “The language of the web”
- This is because it is often embedded in web pages to make them more interactive
- The web browser contains an interpreter which reads the JavaScript and runs it
- It is called JavaScript because it was launched when Java was popular
  - It has very little in common with the Java language

Scary JavaScript

- JavaScript works with objects, although it is much more relaxed about how you can create and use them
- The C# compiler frets a lot about whether your program makes sense, and whether what the program does with things is valid
  - The program must always make sense
- JavaScript works on a different principle
  - The program must always do something
- This makes it easy to write (and run) broken code

JavaScript and the Future

- Because of the rise of HTML5 and web applications JavaScript is going to be with us for a long time
  - There are some very useful frameworks that work well with it – for example JQuery
- You therefore need to be familiar with it
- I recommend the book “JavaScript: The Good Parts” by Douglas Crockford
- And the website codeacademy.com

Python

- Python is a scripting language that is a bit like Java, JavaScript, C and C#
- It is becoming popular as one of the primary languages for the Raspberry Pi
- It is interesting because it also provides a “Python Shell” where you can write language statements which are obeyed immediately
  - A bit like the old versions of Basic
- It is also a very powerful and flexible language
  - If a bit scary…

C

- The C programming language was developed by Brian Kernighan and Dennis Ritchie of Bell Labs
- They used it to create an operating system they were developing
  - ..called UNIX
- C has the same language syntax as C#, Java and JavaScript
  - which is not that surprising, as they are based on it
- C is great for low level stuff, but it is very easy to write a C program that causes your process (and maybe even the computer) to crash

C and C++

- C and C++ are closely related
  - C is the original, C++ adds support for objects
- C is a great language for writing operating systems
  - And a rather dangerous language for writing pretty much anything else
- C++ is a very powerful general purpose language which combines the danger of C with support for Objects
  - But has no garbage collection

The Future and C++

- C++ is important because it runs really fast on the target hardware
- C++ is used to create Video Games
  - There are high performance C++ compilers for just about any platform
- You will be learning C++ next year

Final Important Point

- Just because there are all these languages out there you don’t need to “start from scratch” each time you have to learn a new one
  - They all have statements, variables, assignments, tests, loops, arrays and methods
- You get started in a new language by learning how these controls are used in it
  - A bit like changing from one car to another
- All procedural languages work in essentially the same way when they run

Review

- C# is not the only language you will ever use
  - Although it is one of the best ☺
- As a programmer you will have to learn many languages through your career – and this is not a problem
- They will all have their good parts and their bad parts
  - “You can write horrible code in any language”
  - “You can write great code in any language”