COMP241
Software Engineering Development
Lecture 3: Comparing Java and C#

- Important features
- Same
- Slightly different
- Different

Mark Hall

THE UNIVERSITY OF WAIKATO
DEPARTMENT OF COMPUTER SCIENCE
TARI ROROHIKO

Important Features

Feature	C#	Java
Inheritance	Single class inheritance, multiple interface implementation	Single class inheritance, multiple interface implementation
The notion of interface	Through the "interface" keyword	Through the "interface" keyword
Memory management	Managed, using a garbage collector	Managed, using a garbage collector
Pointers	Yes, but only in the rarely-used "unsafe" mode. References are used, instead.	Not at all. References are used, instead.
Form of compiled source code	.NET intermediate language (IL)	Byte code
One common base class	Yes	Yes

Same

- Virtual machine/language runtime
  - Java -> byte code -> JVM
  - C# -> intermediate lang. (IL) -> CLR
  - Both do native compilation via JITs
- Jagged arrays

```
int [][]myArray = new int[2][];
myArray[0] = new int[3]; myArray[1] = new int[9];
```

- No global methods
  - Belong to a class either as instance or static methods

Same

- Strings are immutable

```
String s = "Grapes";
s.ToLower(); /* C#. Does not modify string,
      returns lowercase copy */
s.toLowerCase(); // Java. Same as above
```

  - Use `java.lang.StringBuilder` for modifiable strings
- Constructor chaining and calling base class constructors
  - Both C# and Java automatically call base class constructors
  - Both provide a way to call the constructor of the base class with specific parameters

Same

- Unextendable classes
  - C#: `sealed`, Java: `final`
- Exceptions
  - `try, catch, finally`
  - Inheritance hierarchy for exceptions
  - Method for obtaining a stack trace
- Runtime type identification
  - C#: `is`, Java: `instanceof`

Slightly Different

- Inheritance syntax
  - C# uses C++ syntax for inheritance (class inheritance and interface implementation)
    - `class B:A, Icomparable {…`
  - Java uses `extends` and `implements`
    - `class B extends A implements Comparable {…`

Slightly Different

- Grouping collections of classes
  - C#: `namespace`, Java: `package`
  - In Java, package names dictate the directory structure for source files
  - C# allows nested namespaces
- Reflection
  - In C#, reflection is done at the *assembly* level
    - Need the DLL containing the targeted class to be available
  - In Java, reflection is done at the *class* level
    - Need to be able to load the class file for the targeted class

Slightly Different

- Declaring constants
  - Java: `final` for either compile time or run time constants
  - C#: `const` for compile time constants and `readonly` for runtime constants

```
// C# ...
// compile time
const int i1 = 10; // static

// run time
public static readonly unint i1 =
   (uint)DateTime.Now.Ticks;

// uninitialized readonly
readonly float f; /* init in
                     constructor */
```

```
// Java ...
// compile time
final int i1 = 10; // instance
static final int i2 = 20; // class

// run time
public static final long l1 = new
   Date().getTime();

// uninitialized final
final float f; /* init in
                  constructor */
```


Slightly Different

- Primitive types
  - Every Java primitive type has a corresponding C# type with the same name
    - Exception: `byte` in Java is signed and is analogous to `sbyte` in C#
  - C# has unsigned versions of some primitives:
    - `ulong`, `uint`, `ushort` and `byte`
  - C# has the `decimal` type
    - Stores decimal numbers without rounding errors (at the cost of space and speed)

Somewhat Different

- Nested classes
  - C#: static nested classes
    - Have access to static member variables and methods of the enclosing class
  - Java: static and non-static nested (inner) classes
    - Instance of inner class is tied to instance of outer class & has access to its member variables and methods

Somewhat Different

- C# assemblies
  - Similar to Java's JAR files
  - Fundamental unit of code packaging in the .NET environment
  - Contain intermediate code from compiling, metadata etc.
  - Actions related to interacting with types are done at the assembly level- e.g. granting security permissions, code deployment, versioning etc.
    - Java typically does these at the class level
  - Assemblies are stored as EXEs or DLLs, JAR files are stored in the ZIP file format

Somewhat Different

- C#: `goto` no longer considered harmful
  - Jump directly from a point in the code to a label
  - Cannot jump into a statement block using `goto`

Somewhat Different

- Virtual methods
  - All methods in Java are virtual
  - In C#, by default all methods are non-virtual
    - Use `virtual` keyword
    - Subclass can choose to override the virtual method by using `override` or not by using `new`
- Final methods
  - Methods can be marked as final in Java
    - Cannot be overriden in base class
  - In C#, this can be done by not marking the method as virtual
    - Subclass can still define the method but the base class version will be called if the object is used via a base class reference

Somewhat Different

- Multiple classes in a single file
  - Java: one class per source file that has public access
    - Must have the same name as the source file (minus the extension)
  - C#: no restriction on number of public classes in a given source file
    - No requirement for the name of any class in the file to match that of the source file
- Importing libraries
  - Referencing source code
    - Java: `import`; C#: `using`
  - Tell the compiler where to find the library
    - Java: `CLASSPATH` env. variable; C#: `/r` compiler switch

Different

- C#: Delegates
  - Mechanism for providing *callback* functions
  - Similar to function pointers in C
  - Same functionality can be achieved in Java by defining interfaces that specify the signature of the callback function


```
public class HasDelegates { // C#
   public delegate bool CallbackFunc(string a, int b);
   public bool execCallback(CallbackFunc doCallBack, string x, int y) {
        Console.WriteLine("Excecuting callback…");
        return doCallback(x, y);
   }
}

Public class FunctionDelegates {
   public static bool FunctionFoo(string a, int b) {
        Console.WriteLine("Foo: {0} {1}", a, b); return true;
   }
}

Public class DelegateTest {
   public static void Main(string [] args) {
        HasDelegates MyDel = new HasDelegates();
        HasDelegates.CallbackFunc myCallBack =
                new HasDelegates.CallBackFunc(FunctionDelegates.FunctionFoo);
        MyDel.executeCallback(myCallBack, "Twenty", 20);
   }
}
```



```
// Java

public interface HasCallback {
   boolean callbackFunc(String a, int b);
}

class UsesCallback {
   public boolean doStuff(HasCallback h, String a, int b) {
        return h.callbackFunc(a, b);
   }
}

public class CallbackTest {
   public static void main(String [] args) {
        UsesCallback u = new UsesCallback();
        u.doStuff(new HasCallback() {
                        public boolean callbackFunc(String a, int b) {
                                System.out.println("a & b " + a + " " + b);
                        }
                }, "Twenty", 20);
   }
}
```


Different

- C#: Value types (Structs)
  - Specify that objects of a certain class should be stack-based (rather than live on the heap)
  - Stack memory is faster to allocate and is automatically reclaimed by the system when no longer needed
  - Avoid garbage collection overhead

Different

- Runtime type identification
  - C#: as operator
  - Performs an explicit cast
    - Returns null if unsuccessful

```
MyClass mc = o as MyClass;
```

- Multidimensional arrays
  - C# has multidimensional arrays that are a contiguous block containing members of the same type
  - Java only has jagged arrays (arrays of arrays)

Different

- Pass by reference
  - In Java the arguments to a method are passed by *value*
    - Method operates on copies of the items passed to it
  - In C# it is possible to specify that the args to a method actually be *references* to the items being passed
    - Keywords **`ref`** and **`out`**

Different

- Pointers and unsafe code
  - In C# it is possible to have pointer types similar to those in C/C++
    - Can only be used in an `unsafe` context
  - Useful in certain situations
    - Eg. When interfacing with the underlying operating system, accessing a memory-mapped device, times when performance is critical, etc.

Different

- Cross platform portability
  - Java byte code can be run on any platform with a JVM
- Dynamic class loading
  - Java can dynamically load classes at runtime
  - Powerful when combined with remote procedure call
    - Java apps can download class files that do not exist on the target machine