**ECE155: Engineering Design with Embedded Systems** Winter 2013

Lecture 3 — January 11, 2013

*Patrick Lam* *version 1.1*

## Java for C# programmers

The labs this semester will require you to write Java code for the Android platform, yet you learned C# in ECE150. Fortunately, there are a lot of similarities between Java and C#, so you should have a smooth transition.

**What's the same.** All of the basic imperative constructs from ECE150 work the same way:

- `x = y;`, plus expression syntax is the same.
- `if(cond) { ... } else { ... }`
- `for (init; cond; expr2) { ... }`
- `while (cond) { ... }`
- `do { ... } while (cond);`
- `switch (v) { case N: ...; break; case M: ...; break; default: ...; break; }`

However, C#'s `foreach (type t in c)` is instead `for (Type t : c)` in Java.

Methods work the same way also:

```
modifiers rtype methodName(param-list) {
  T1 t; T2 r;
  ...
  return r;
}
```

**Caveat.** Follow conventions. In C#, the method name convention is `UppercaseFirstLetter()`, while in Java, it is `lowercaseFirstLetter()`.

**Example.** A simple unit converter:

```
using System;

class FootConverter {
  static double ConvertFeetToMeters(double feet) {
    return feet * 3.28;
  }
  static void Main(string[] s) {
    Console.WriteLine("{0} ft is {1} m.", s[0],
      ConvertFeetToMeters(Convert.ToDouble(s[0])));
  }
}
```

```
class FootConverter {
  static double convertFeetToMeters(double feet) {
    return feet * 3.28;
  }
  public static void main(String[] s) {
    System.out.printf("%s ft is %.2f m.\n", s[0],
      convertFeetToMeters(Double.parseDouble(s[0])));
  }
}
```

| **C#** | **Java** |
|---|---|
| Previous example: | |
| `UppercaseFirst()` | `lowercaseFirst()` |
| Main(...) | main(String[] argv) |
| `Console.WriteLine("{0}", s);` | `System.out.printf("%s", s);` |
| `s = String.format("{0:.##}", f);` | `s = String.format("%.2f", f);` |
| `string` | `String` |
| `Convert.ToDouble` | `Double.parseDouble()` |
| Other fundamental language features: | |
| `const` | `final` |
| `bool` | `boolean` |
| both rectangular and jagged arrays | jagged arrays only |
| `array .Length` | `array .length` |
| `ref`, `out` (for method parameters) | no equivalent |
| pointers, `unsafe`, `fixed` | no equivalent |
| Object-oriented Features | |
| `class C : Parent, I` | `class A extends Parent implements I` |
| `struct` | `class`es only |
| `class C { public C(...) : base(...) {} }` | `class C { public C(...) { super(...); } }` |
| default visibility private | default visibility package |
| `x.GetType()` | `x.getClass()` |
| `is` | `instanceof` |
| `C cc = x as C` | `C cc = null;`<br>`if (x instanceof C) cc = (C)x;` |
| `virtual` | (is mandatory default) |
| ‘‘`new`’’ modifier | no equivalent |
| `override` | `@Override` |
| `IComparable` | `Comparable` |
| properties | manual getters and setters[1] |
| no equivalent | checked exceptions |
| using | import static (but don't use it) |
| namespaces | packages |

File I/O is slightly different, but we won't need to use that in this course.

## Logging for Android Development

We'll finish with an Android development tip. `System.out.println()` is great for debugging console applications, but doesn't work on Android. Instead, use:

```
Log.d("tag", "i = "+i);
```

This writes out a debug (d) logging message, which appears e.g. in your Eclipse `LogCat` window. Instead of `d`, you can write `Log.d`, `.i`, `.v`, `.w` or `.wtf`. You can then filter out logging messages by level or tag, so that you only see the ones you're interested in.

[1] `http://stackoverflow.com/questions/565095/are-getters-and-setters-evil`