Lecture 2 — Introduction to Java

Patrick Lam & Jeff Zarnett
p.lam@ece.uwaterloo.ca & jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

November 14, 2015

Java for C# Programmers

The labs this semester will require you to write Java code for the Android platform, yet you learned C# in ECE 150.

Fortunately, there are a lot of similarities between Java and C#, so you should have a smooth transition.

As it says in the syllabus, please take some time at the beginning of the term to get caught up on Java.

Java: Object Oriented

Java is an object-oriented programming language:

- Every piece of data is encapsulated in some object.
- Every executable statement is in some method.
- Every object is an instance of a class (or is an array).

Java Types

Java has eight primitive (basic) types. Every variable will be one of the primitive types or a reference to a Java object.

A reference may be `null` or contain the address of an instance of an object.

The Eight Basic Types

The eight basic types are:

1. `boolean`
2. `char`
3. `byte`
4. `short`
5. `int`
6. `long`
7. `float`
8. `double`

Floating Point Special Values

Note that in addition to their normal values, the floating point types have some extra weird values:

`NEGATIVE_INFINITY`, `POSITIVE_INFINITY`, and NaN.

These special values result from operations that go out of range or make no mathematical sense.

Java Types

The common types you're likely to use are `boolean`, `int`, and `double` (in the labs, `float` gets a fair amount of use).

You have no doubt noticed that `String` does not appear.

It's not a simple type; it's a reference to an array of characters.

Strings

Declare a `String` as `null`, like this: `String s = null;`.

Strings are immutable: once created, they don't change.

If you add to a String or replace characters in it, you get a new String back and not the old one.

Example: `string1.replaceAll('?', '.')`,
$\rightarrow$ assign the result
```
string1 = string1.replaceAll('?', '.')
```

Strings

The equality operator == can behave strangely on Strings.

Use the `equals` method:

```
  if (string1.equals(string2)) { ... } .
```

Wrapper Classes

Java also provides wrapper classes for the primitive types.

Work with them as if they were regular objects.

The wrapper class for `int`: `Integer`

(Mostly, just capitalize the first letter).

Complex Objects

Just like in C# you create them in classes.

To create an instance of that class you use the new keyword.

Example: `Integer example = new Integer(20);`.

The new keyword invokes the class Constructor.

Java Semantics

Java does not have `structs`; classes only.

Frequently asked questions: there are no pointers or delegates.

Java uses *Garbage Collection*. No freeing up or de-allocating objects that are no longer needed.

This prevents the extremely common error of trying to use some object/memory that has been released.

Imperative Constructs

- Assignment: `x = y;`
- Math: `i = j + k`. (This includes the operators like += etc.)
- Expressions: `z > 10 || ((c == 0) && (a == b))`
- If-Statements: `if(cond) { ... } else if (cond2) { ... } else { ... }`
- For Loops: `for (init; cond; expr2) { ... }`
- While Loops: `while (cond) { ... }`
- Do-While Loops: `do { ... } while (cond);`
- Switch-Case Statements: `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

Methods work the same way also:

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

When you call a method on an object, you still use the `.` between the object and the method, such as `t.toString()`.

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

Simple Unit Converter: C#

```
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])));
  }
}
```

Simple Unit Converter: Java

```
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])));
  }
}
```

Arrays & Collections

The simple array is created with the `[]` square brackets.

Example: `int[] numbers = new int[10];`

You can have null elements in an array (say, of Strings) without this affecting the array length.

An array is technically an object so you can assign it where a generic `Object` is expected.

Arrays

Multidimensional arrays are also allowed, but only for primitive types `int[][][] coordinates = new int[5][10][2];`.

This is not a big restriction because you can just have an array of arrays.

Unlike C# though, you can’t specify a rectangular array.

Arrays

This is great, but like the String the explicit array is of fixed size.

Create a new, bigger one if you needed it and copy all the data to the bigger one...?

Allocate an array of capacity 999 when we aren't sure how many we'll need?

Wouldn't it be nice if we had a dynamic collection?

Java Collections

In Java, we do, and they're called, `Collections`.

The most common one: the `List`.

The type List takes a parameter in angle brackets to tell you what this is a list of.

Example: `List<String>`.

Lists

Note that you can't call `new List<String>()` because no constructor exists for just plain `List`.

Be specific about what kind of list you want to have, such as `ArrayList` (a very common one) or `LinkedList`.

Three Basic Collections

Three basic collections exist:

1. List
2. Map (in a later lecture)
3. Set (like a list, but no duplicates)

Your First Java Program: C# Equivalent

```
using System;

class C {
  static void Main() {
    Console.WriteLine("Hello, world!");
  }
}
```

Your First Java Program

```
class C {
  public static void main(String[] argv) {
    System.out.println("Hello, world!");
  }
}
```

Logging for Android

Android tip: `System.out.println()` is great for debugging console applications, but doesn't work on Android.

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

This writes out a debug (d) logging message, which appears e.g. in your Eclipse `LogCat` window.

You can then filter out logging messages by level or tag, so that you only see the ones you're interested in.

Eclipse Demo

Now we'll open up Eclipse and create some basic Java programs. We can take a look at the syntax of Java in simple situations.