



Programming Methodology

Lecture 02 – Source Control and Scala





Objectives

- To understand source control
- To learn and apply git basics
- To learn about Java Collections
- To understand the use of Scala
- To understand Java vs Scala
- To learn the basics of Scala
- To learn about the REPL



From last time...

You will be graded on a number of aspects including functional (did it pass tests), your own tests, code style, following instructions, git activity, comments and documentation, and any other criteria designated by the assignment documentations

Grades will be on a letter scale: A-F

This course does not give 0's on assignments. **Why?**

Because: $100+0/2 = 50$ (an F), $100+50/2 = 75$ (a C)

But: $A+F/2 \neq F$, so it makes sense for $A+F/2 = C$ (in my book!)



VirtualBox/Vagrant

- Why use a virtual machine?
- How is it configured?
- How do we boot up?
- How do we connect?



Basic Command Line

- Why do we use the command line?
- What are some simple commands?



Compiling Java Programs

- IDE is not necessary!
- We will use **javac** to compile Java programs
- Need a good structure for our source code
 - src/main/java, bin
- Other things to consider?
 - packages, classpath



Creating a Project Directory

- Very simple from the command line:

```
$ mkdir -p myproj/src/main/java
```

```
$ mkdir myproj/bin
```

```
$ cd myproj
```

```
$ touch README.md
```



Quick Git Introduction

- Let us go to the command line for this!



Example: Compiling Java

- How to compile Java's hello world

```
$ javac -d bin src/main/java/Hello.java
```

- This will generate class files into the bin directory.
- What is a class file?



Example: Running Java

- How to run Java's hello world

```
$ java -cp bin Hello
```

- This designates that the “classpath” will be the bin directory and the class to run is Hello.



Example: Compiling Packages

- What if we have our code in a package?

```
$ javac -d bin src/main/java/cs220/Hello.java
```

- This will generate class files into the bin directory.
- What is different about the bin directory now?
Is there anything wrong?

Example: Running Packaged Java



- How to run Java's hello world

```
$ java -cp bin cs220.Hello
```

- This designates that the “classpath” will be the bin directory and the class to run is cs220.Hello.



Java and Data Structures

- Java has a vast library that comes with many useful data structures.
- You can find these in `java.util`
- It helps to understand these data structures a little so that we can compare them to what Scala and other languages provide.



Java's Data Structures

- List
 - ArrayList, LinkedList, Stack, Vector, ...
- Map
 - HashMap, Hashtable, TreeMap, ...
- Set
 - HashSet, TreeSet, ...
- Queue
 - LinkedList, PriorityQueue, ...



ArrayList<T>

- How is an array list implemented?
- What does the <T> mean?
- Here is how you create one:

```
List<String> xs = new ArrayList<String>();
```

- What is interesting about the line above?



Elements in a List

- How do you search a list to determine if an element exists?



Elements in a List

- How do you search a list to determine if an element exists?

`xs.contains("Hello")`

- What does this return?



Lists of Other Things

- Imagine this class:

```
class Student {  
    private String sid, fname, lname;  
}
```

- And creating a new list:

```
List<Person> xs = new ArrayList<Person>();
```

Elements in a List of other things



- Assume we have a Person p , then what does this mean?

`xs.contains(p)`

- What does this return?
- Is this ok for lists?



What about sets?

- Here is a set in Java

```
Set<Person> s = new HashSet<Person>();
```

- Then, what does `s.contains()` mean?



Object Equality

- Very important to understand!
- Java's data structures depend on it!
- A set contains 1 and only 1 element as defined by what it means to be "equal" to other elements in the set.
- So, how do we make this work right?



Overriding equals

- We need to define what it means for objects to be “equal” to each other.

How might we define this for the Student class?

Let us look at an example...

Is this all we need?



Object hashCode

- Very important to understand!
- Java's data structures depend on it!
- A set contains 1 and only 1 element as defined by what it means to be "equal" to other elements in the set.
- So, how do we make this work right?



Overriding hashCode

- We need to define what it means for objects to be “unique”.

How might we define this for the Student class?

Let us look at an example...

Is this all we need?



What if we use TreeSet?

- Here is a tree set in Java

```
Set<Person> s = new TreeSet<Person>();
```

- Are we still required to implement equals and hashCode?
- No, instead we must implement Comparable<T>!



Maps

- Maps are an extremely useful data structure
 - Often referred to as a Hashtable or Dictionary
 - Java provides multiple implementations
 - Offer $O(1)$ (constant time) insertion and lookup

```
Map<String, Integer> m = new HashMap<String,Integer>();  
m.put("banana", 24);  
m.put("pear", 43);  
m.get("banana") == 24; // evaluates to true
```



What do we override?

- If we use a `HashMap<F,T>`
 - We must override `equals` and `hashCode`
- If we use a `TreeMap<F,T>`
 - We must implement `Comparable<T>`
 - And override `compareTo`
- **How do we know?**
 - Read the documentation of the class in the Java API carefully to understand how to use the data structure properly!



Activity

- Take 5 minutes to write a class called Student that can be used correctly with both a HashMap and a TreeMap.
- In addition, given a Person P with fname = “Jane”, lname = “Doe”, and sid = “1234”, the following should produce the proper output:

```
System.out.println(P);  
// evaluates to: Jane Doe (1234)
```



i-clicker

How many methods does your Person class contain?

- a) 1
- b) 2
- c) 3
- d) 4
- e) 5



Take Away Point

- In Java, always **override** the equals and hashCode methods!
- In Java, always implement Comparable<T>
 - And implement the compareTo method!
- If you do not do this your classes will not be safe to use with the java.util Collections!



Switch Gears

- Ok, enough about Java for the moment...



 **Scala**

The Scala logo, which consists of a red, stylized, wavy graphic element to the left of the word "Scala" in a bold, black, sans-serif font.



Scala

- **A Language that Grows on You**

Scala is a blend of two important programming paradigms: *functional* and *object-oriented*.

Scala is useful both in the small (scripting) as well as in the large to build robust and reusable frameworks and libraries.

Scala compiles to the JVM and can easily interoperate with Java and its API making it a good next language to learn and transition to.



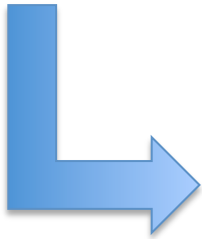
Concise

```
var capital = Map("US" -> "Washington", "France" -> "Paris")  
capital += ("Japan" -> "Tokyo")  
println(capital("France"))
```



Growing New Types

```
def factorial(x: BigInt): BigInt =  
  if (x == 0) 1 else x * factorial(x - 1)
```



As compared to Java's

```
import java.math.BigInteger  
  
def factorial(x: BigInteger): BigInteger =  
  if (x == BigInteger.ZERO)  
    BigInteger.ONE  
  else  
    x.multiply(factorial(x.subtract(BigInteger.ONE)))
```



Scala – Is Scalable

- Object-Oriented
 - Provides classes, interfaces (traits), inheritance, ...
 - Goes further than Java, everything is an Object
- Functional
 - Functions are *first-class values*
 - Lisp, Scheme, SML, Erlang, Haskell, OCaml, F#
 - Emphasizes *immutability*, functions map input values to output values providing *referential transparency*




Scala – Why?

- Simpler, more concise code. Take a look:

```
// this is Java
class MyClass {
    private int index;
    private String name;
    public MyClass(int index, String name) {
        this.index = index;
        this.name = name;
    }
}
```

Java

A red arrow pointing from the word "Java" towards the code block.



Scala – Why?

- Simpler, more concise code. Take a look:

// this is Java

```
class MyClass {  
    private int index;  
    private String name;  
    public MyClass(int index, String name) {  
        this.index = index;  
        this.name = name;  
    }  
}
```

Java



Scala



```
class MyClass(index: Int, name: String)
```



Scala – Why?

- High level. Take a look:

Java



```
// this is Java
boolean nameHasUpperCase = false;
for (int i = 0; i < name.length(); ++i) {
    if (Character.isUpperCase(name.charAt(i))) {
        nameHasUpperCase = true;
        break;
    }
}
```



Scala – Why?

- High level. Take a look:

Java



```
// this is Java
boolean nameHasUpperCase = false;
for (int i = 0; i < name.length(); ++i) {
    if (Character.isUpperCase(name.charAt(i))) {
        nameHasUpperCase = true;
        break;
    }
}
```

Scala



```
val nameHasUpperCase = name.exists(_.isUpper)
```



Scala – Type Inference

- Scala provides more type inference than Java which leads to more concise code:

```
val x = new HashMap[Int, String]()  
val x: Map[Int, String] = new HashMap()
```




Scala – First Steps

- The Read-Eval-Print Loop!
 - Will become your best friend!
 - A wonderful way to experiment and learn!

```
$ scala
```

```
Welcome to Scala version 2.11.5 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0
```

```
Type in expressions to have them evaluated.
```

```
Type :help for more information.
```

```
scala>
```

Scala – Simple Evaluation



```
scala> 3+4  
res0: Int = 7
```

```
scala> res0 * 3  
res1: Int = 21
```

```
scala> println("Hello, World")  
Hello, World
```