2.10 Scala, O-O and Java

Scala and functional programming

- There are five key features of Scala:
  - Functional Programming;
  - Object-oriented;
  - Strict types (with type inference);
  - Java virtual machine;
  - Implicits.

- We already discussed Functional Programming.

Object-oriented

- Type hierarchy—basically the same as for Java;
- Polymorphism, encapsulation, all that good stuff;
- Classes, traits, objects.
  - traits instead of Interface (traits can have default implementations, similar to what's now in Java8);
  - "objects" in Scala are singleton classes (similar to marking stuff static in Java);
  - Also "case" classes:
    - Provide all of the standard boiler-plate code, including *unapply* for pattern-matching;
- There are no "primitives" in Scala (at least *you* don't have to be aware of them);
- The Class Loader (doesn't distinguish between Scala and Java code);
- The ubiquitous "dot" operator.

Static Types

- Parametric (as opposed to generic) types
  - Liskov substitution principle (defines what sub- and super-types mean);
  - Full treatment of variance (where *S* is a sub-type of *T*):
    - Invariant: *Array[T]* is neither subtype nor supertype of *Array[S]*
    - Covariant: *List[S]* is a subtype of *List[T]*
    - Contravariant: *T => U* is a subtype of *S => U*
  - This allows us always to be very strict about types (unlike Java or Python, for example)
- Type classes and other type constructors
  - T[X]
- **Once you have succeeded in compiling a Scala program, it usually does what you want (run-time errors are less common than with Python or Java).**

Java Virtual Machine

- Scala wouldn’t be a serious contender in the language space if it didn’t run on the JVM.
- Running on the JVM gives access to thousands of Java libraries, especially important in the early days of Scala.
- The JVM, lazy evaluation, functions, class loader, etc. are what powers Spark. There’s nothing magic in Spark.

Implicits

- One of the big problem areas in any language is dealing with (configuring, searching) “global” variables;
- Implicits not only provide a solution to these global variables, but they give us many other benefits (e.g. “extension” methods);
- Type classes;
- Implicit classes and conversions (instead of just “widening” as in Java).

What's different from other FP languages?

- Scala does allow *var* (mutable variable) and mutable collections:
  - but you are discouraged from using them! And you really don't need them!
- Scala has similar ways of defining new types (*Option*, *Either*, etc.) as well as "type constructors" and lots of arcane stuff that you will never actually come across–however,
  - Scala also allows you to define new types by inheritance (just like O-O)
- Scala doesn't actually have a monadic type—
  - if you really want that, you have to use one of the libraries such as ScalaZ

What's different from Java?

- Scala avoids *nulls* and exceptions via built-in types:
  - *Option[X]* which has two cases: *Some(x)* and *None*;
  - *Try[X]* which has two cases: *Success(x)* and *Failure(e)*;
- Other "union" types like *Either[L,R];*
- *Future[X]* is used to handle asynchronous results
  - different from Java's *Future*.
- Scala doesn't have primitives like *int*, *double*;
- Collections are different;
- Function types are much more general than Java8's function types:
  - multiple parameters easy to define;
  - "curried" functions for example;
- Other types, e.g. *String, Array, MyClass*, are the same;
- Traits can have default methods (Java8 introduced this);
- Type *inference* (now available in Java9).

What's different from Java (continued)?

- You can define related types (traits, classes, etc.) in one module
  - i.e. without having to make private inner classes;
- There are no static ("class") methods: all singletons* are "objects";
- In Java, you can't return more than one value from a method unless you go to the trouble of defining a class—in Scala you just return a tuple.
- In Java, generics are kind of a mess (they were an after-thought).
  - You can cheat, or you can just make everything a "?".
  - In Scala, parametric types are very strictly enforced. So, you can't be surprised at run-time by a value that doesn't conform to the proper type.

* Have we talked about singletons?