Why you want to learn

What is Scala?

- Programming language for the JVM
  - 99%* Compatible with Java
- Blends Functional programming (Erlang, Haskell) with Object-Oriented (Java, C#)
- Created by Martin Odersky (EPFL)

Why Should I care?

- 99% Java Compatibility
- Static Typing + Type inference
- Pattern Matching
- Powerful collections library
- For-expressions
- Closures
- Mixin (Multiple) Inheritance
- Amazing community

Does Anyone use Scala?

Syntax Overview

- Support a few high-level language features in consistent ways
- The Power Triads
  - Members - defs, vars and vals
  - Types - classes, objects and traits
  - Control - pattern matching, closures, for expressions

Members Triad

- **val**
  - Immutable (assign once) variables
- **var**
  - Mutable variables
- **def**
  - Methods

Members - vals

- Similar to public final fields in java

```
val x: String = "Hello World"
```

- Type-inference

```
val x = "Hello World"
```

- The "type" of x is still a String, only inferred
- Immutable! (well, sort of)

Members - defs

```
public String sayHelloTo(String name) {
  return “HI, ” + name;
}

def sayHello(name: String): String =
  "HI, " + name

def sayHello(name : String) = "HI, " + name
```

Members - defs (cont'd)

- Multi-line Syntax

```
def doStuff(arg : Type) = {
...
}
```

- Returning void

```
def doStuff() : Unit = { … }
def doStuff() {}
```

Members vars

- Similar to regular java fields

```
var x : Int = _
x = 5
System.out.println(x);
```

Members - Abstract + Uninitialized

- Abstract - leave off the right-hand-side (=…)

```
def abstractMethod() : Unit
val abstractValue : Int
```

- Uninitialized? Use the _
  - `var x : Int = _`
  - Only viable for var members

Uniform Access Principle

- To the greatest extent possible, allow var/val/def to be interchangeable
  - HOW???
  - vars => Getter/Setter methods
  - val => Getter methods
- Scala allows you to override a no-argument “def” with a “val” of the def’s return type.

Laziness

- Scala also allows “lazy” vals
- The val is not evaluated until first use

```
lazy val someThing =
someExpensiveComputation()
```

- Can improve efficiency of algorithms
- Can be used to side-step initialization order issues.

```
lazy val hashCode = super.hashCode
```

Scala Types

- Classes
  - Concrete types
- Objects
  - "instance" types
- Traits
  - "mixin" abstract types

Scala Types - Class

- Classes are most similar to java

```
class MyClass {}
```

- Classes may extend one other class

```
class MySubClass extends MyClass {}
```

- Classes may “mixin” one or more trait/interface

```
class MyOtherSubClass extends MyClass
with SomeInterface {}
```

Classes + Members

- Classes may contain any number of members

```
class XYZ {
   def doSomething() = x * x
   val x = 5
}
```

- All members turn into methods with optionally field backing
  - Only one namespace per class!

Class Constructors

- Classes have one mandatory entry point

```
class MyClass(x : Int) {
  def doubled = x * 2
}
```

- Constructor arguments may optionally be promoted as vars/vals

```
class MyClass(val x : Int) {
  def doubled = x *2
}
val tmp = new MyClass(5)
assert(tmp.x*2 == tmp.doubled)
```

Scala Types - Object

- class + singleton mixed

```
object MyObject extends MyClass with
SomeInterface {
  val SOME_CONSTANT = 5
  def doSomethingFun() { … }
}
```

- No statics in scala! (unless objects count)

Scala Types - The Trait

- Like Java Interfaces

```
trait MyService {
  def doSomeBusinessLogic() : Unit
}
```

- May contain implementations

```
trait MyOtherService {
  def double(x : Int) = 2 * x
}
```

- Must *not* take any constructor parameters

Traits (con't)

- Traits can extend other traits

```
trait A {}
trait B extends A {}
trait C extends A with B {}
```

- Trait can extend classes (as long as the class has a no-argument constructor)

```
trait A extends JComponent {}
```

Avoiding the Deadly Diamond

traits override their parents in a right-to-left fashion

```
trait A {
  def service(){ Console.print("A") }
}
trait B extends A {
  def service(){ Console.print("B"); super.
service() }
}
trait C extends A {
def service() { Console.print("C"); super.
service() }
}
val x = new A with B with C
x.service() //Output CBA
```

Control Structures

- Pattern Matching*
  - Choosing behavior based on input
- For Expressions*
  - Iteration/looping/streaming like operations
- Functions as Objects (Closures/Lambdas)*
  - Passing behavior between objects
- The Familiar
  - if-else
  - try-catch

Control Structures (the familiar)

- ifs and whiles similar to java

```
if( x < 5) { doSomething() } else {
doFailure() }
var x = 2
while(x > 0) { println(x) x = x - 1; x-=1; }
```

- “Expression-Oriented” programming
  - if-expressions return values
  - No need for ?:

```
val y = if ( x != null ) x else 6
```

Control Structures (Pattern Matching)

- Similar to Switch
- Very robust in ‘case’ syntax

```
x match {
case 42 => “But what is the question?”
case “Hello” => println(“Why, Hello!”)
case y : String => doSomethingWithAString(y)
case null => doSomethingWithNull()
}
```

- Also “expression-oriented”

Control Structures (try-catch)

- Somewhat similar to java

```
try {
  liveDangerously();
} catch {
  case x : SomeException => handle(x)
  case _ => defaultHandle()
}
```

- Catch block is a pattern-match!

Pattern Matching + Case Classes

- Scala provides ‘sugar’ for matching against classes

```
case class DatabaseResponse(model : Model,
metaInfo : MetaModel)

makeQuery() match {
case DatabaseResponse(model, metaInfo) =>
//Use fully-typed Model + MetaModel
}
```

For Expressions

- Somewhat like for loops, but much more powerful
- Allows looping and expression style syntax

```
for(i <- 0 until 10) { println(i) }
val idx = for(i <- 0 until 10) yield i
```

- Supports multiple "ranges"

```
for( i <- 0 until 10; j <- 0 until 10 ) {
matrix.get(i,j).doCalc()
}
```

For Expressions (con't)

- Guards

```
val odds = for(i <- 1 until 100; if i % 2 == 1)
yield i
```

- Multi-line syntax

```
val evens = for {
i <- 1 until 100
if i % 2 == 0
} yield i
```

- Definitions

```
for (i <- 1 until col.size(); val item = col.
get(i) ) {
doSomethingWith(item)
}
```

Functions as Objects

- Scala allows you to treat functions as objects

```
scala> val x = { x : String => x + "HAI" }
x: (String) => java.lang.String = <function>

scala> x("O ")
res3: java.lang.String = O HAI
```

- Braces are optional (if function is a one-liner)

Functions as Objects (con't)

- Can use "lambda expressions" to define functions
  - _ means placeholder
  - can be typed

```
scala> val y = "Hello, " + (_ : String)
y: (String) => java.lang.String = <function>

scala> y("World")
res0: java.lang.String = Hello, World
```

Function Objects + Collections

- Can use "closures" to do interesting things with collections
  - map (Notice the type of _ is inferred)

```
scala> val x = 1 until 10
x: Range = Range(1, 2, 3, 4, 5, 6, 7, 8, 9)

scala> val y = x.map(_ * 2)
y: RandomAccessSeq.Projection[Int] = RangeM(2,
4, 6, 8, 10, 12, 14, 16, 18)
```

  - foreach (notice print is promoted to a closure)

```
scala> y.foreach(print)
24681012141618
```

LINQ 4j?

C#

```
string[] names = {"Burke","
Connor","Frank",
"Everett","Albert"};

IEnumerable<string> query =
from s in names
where s.Length == 5
orderby s
select s.ToUpper();
```

Scala

```
val names = List("Burke",
"Connor", "Frank",
"Everett", "Albert")

val query = for {
  s <- names.sort(_<_)
  if s.length() == 5
} yield s.toUpperCase()
```

Tool Support?

- IDEs
  - Netbeans
  - IntelliJ
  - Eclipse*
- Build tools
  - Ant
  - Maven*
  - Simple Build tool (SBT), Gradle, Rake

Questions?

Special Thanks to...

Alex Cruise, James Iry, Jorge Ortiz, Seth Tissue for QAing my slides!

Back-up Slides

Things I doubt we'll be able to fully cover, but in case there are questions...

Function Programming - Tail Recursion

```
@tailrec
def loop( f  : => Unit) : Unit = {
    f
    loop(f)
}

while(true) {
   f();
}
```

- @tailrec
  - Ensures tail recursion is used, or compiler fail

The Loaner Pattern

```
def ensureClose[A <: Closeable, B](resource : => A)(f : A
=> B) = {
   val r = resource
   try {
          f(r)
   } finally {
         r.close()
   }
}

//Exceptions are still thrown, but the resource is
guaranteed to be closed
ensureClose(new FileInputStream("in.txt")) { input =>
   //Do stuff with input (which has type FileInputStream)
}
```

Actor based concurrency

```
scala> import actors.Actor._
import actors.Actor._
scala> actor {
     |   loop {
     |      react {
     |           case msg : String => println(msg)
     |           case x : Int => println("NUMBER " + x)
     |      }
     |   }
     | }
res0: scala.actors.Actor = scala.actors.
Actor$$anon$1@106def2
scala> res0 ! "HELLO"
HELLO
scala> res0 ! 5
NUMBER 5
```

Property based Testing

```
import org.scalacheck._
val smallInteger = Gen.choose(0,100)
val propSmallInteger =
  Prop.forAll(smallInteger) {
     n => n >= 0 && n <= 100
  }
```

Delimited Continuations

```
import java.io._
import util.continuations._
import resource._
def each_line_from(r : BufferedReader) : String
@suspendable =
  shift { k =>
    var line = r.readLine
    while(line != null) {
      k(line)
      line = r.readLine
    }
  }
```

Delimited Continuations (Part 2)

```
reset {
  val server = managed(new ServerSocket(8007)) !
  while(true) {
    reset {
      val connection = managed(server.accept) !
      val output = managed(connection.
getOutputStream) !
      val input = managed(connection.getInputStream)
!
      val writer = new PrintWriter(new
BufferedWriter(new OutputStreamWriter(output)))
      val reader = new BufferedReader(new
InputStreamReader(input))
      writer.println(each_line_from(reader))
      writer.flush()
    }
  }
}
```

Ant tasks

- defined in scala-compiler.jar

```
<taskdef resource="scala/tools/ant/antlib.xml">
<classpath>
<pathelement location="${scala.home}/lib/scala-compiler.jar"
/>
<pathelement location="${scala-library.jar}" />
</classpath>
</taskdef>
<scalac srcdir="${sources.dir}" destdir="${build.dir}"
classpathref="build.classpath" force="changed">
<include name="compile/**/*.scala" />
<exclude name="forget/**/*.scala" />
</scalac>
```

- Complete Scala-project bootstrap (using maven-ant-tasks)
  - http://suereth.blogspot.com/2008/09/using-maven-ant-tasks-instead-of-ivy.html

Maven Plugin

```
<project>
...
<plugin>
  <groupId>org.scala-tools</groupId>
  <artifactId>maven-scala-plugin</artifactId>
  <executions>
    <execution>
      <goals>
        <goal>compile</goal>
        <goal>testCompile</goal>
      </goals>
    </execution>
  </executions>
</plugin>
...
</project>
```