WTF Is An Implicit?!?

Brendan W. McAdams

10gen, Inc.

NY Scala Enthusiasts

Types of Implicits
Major Coverages

- Implicit Parameters
- Implicit Methods
- Pimp My Library
- Type Classes

mongoDB

Implicit Parameters
What Are Implicit Parameters?

- 'Explicit' parameter indicates a method argument you pass, well *explicitly*.
- 'Implicit' indicates a method argument which is. . . *implied*. (But you can pass them explicitly too!)
- The value of an implicit argument is inferred from the scope by default; it can be used to define an environmental context.
- Useful for:
  - Values which are initialized once in scope and you don't want to keep passing explicitly.
  - Passing utility libraries such as conversion methods to conform to a protocol.

mongoDB

Implicit Parameters
Demonstrated. . .

- Implicit parameters are passed as an additional argument list:

```
import com.mongodb._
import org.bson.types.ObjectId

def query(id: ObjectId)(implicit coll: DBCollection) = coll.findOne(id)

val conn = new Mongo()
val db = conn.getDB("test")
implicit val coll = db.getCollection("testData")

// coll is passed implicitly
query(new ObjectId())

// or we can override the argument
query(new ObjectId())(db.getCollection("testDataExplicit"))

// You can accept multiple implicits also
def query(id: ObjectId)(implicit conn: Mongo, coll: DBCollection) = {
  conn.slaveOk()
  coll.findOne(id)
}
// Scala prints this for the above method in console:
/* query: (id: ObjectId)(implicit conn: Mongo,implicit coll: DBCollection)DBObject */
```

- How does this differ from default arguments?

Implicit Parameters
Demonstrated. . .

- Implicit parameters are passed as an additional argument list:

```
import com.mongodb._
import org.bson.types.ObjectId

def query(id: ObjectId)(implicit coll: DBCollection) = coll.findOne(id)

val conn = new Mongo()
val db = conn.getDB("test")
implicit val coll = db.getCollection("testData")

// coll is passed implicitly
query(new ObjectId())

// or we can override the argument
query(new ObjectId())(db.getCollection("testDataExplicit"))

// You can accept multiple implicits also
def query(id: ObjectId)(implicit conn: Mongo, coll: DBCollection) = {
  conn.slaveOk()
  coll.findOne(id)
}
// Scala prints this for the above method in console:
/* query: (id: ObjectId)(implicit conn: Mongo,implicit coll: DBCollection)DBObject */
```

- How does this differ from default arguments?

Caveats of Implicits

- Caveat #1: Scala will *not* try to second guess you with ambiguity. It *will* fail to compile.

```
scala> implicit val x = 5
x: Int = 5

scala> implicit val y = 15
y: Int = 15

scala> implicit val y = 15.5
y: Double = 15.5

scala> def test(implicit value: AnyVal) = println("Test Value: %s".format(value))
test: (implicit value: AnyVal)Unit

scala> test
<console>:18: error: ambiguous implicit values:
 both value y in object $iw of type => Double
 and value x in object $iw of type => Int
 match expected type AnyVal
       test
       ^
```

mongoDB

Fitting A Square Peg Into a Round Hole

In Strongly Typed languages (e.g. Java), square pegs do *not* fit into round holes:

mongoDB

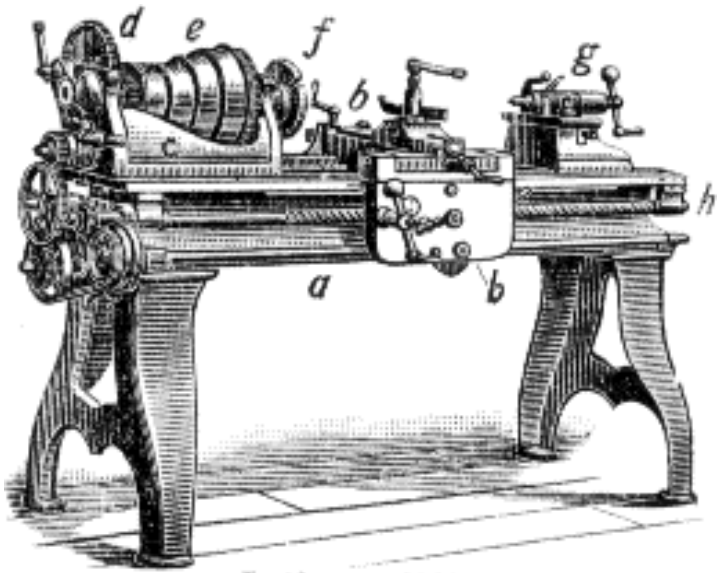
Fitting A Square Peg Into a Round Hole

In Loosely Typed languages (e.g. Perl), round holes can be convinced to accept square pegs:

mongoDB

Fitting A Square Peg Into a Round Hole

With Scala, you get a lathe instead:



Lathe, p. 1218.

mongoDB

Implicit Conversions I

- You can declare *methods* to be implicit as well as *values*
- When Scala's compiler encounters an invalid argument type being passed to a method:

```
def printNumber(x: Int) = println(x)

printNumber(5)
printNumber("212") // won't compile
```

mongoDB

Implicit Conversions II

- If it finds a matching implicit method (takes bad type and returns good type) in scope, and no ambiguity. . . Square Pegs can become Round Pegs:

```
implicit def strToNum(x: String) = x.toInt
def printNumber(x: Int) = println(x)

printNumber(5)
printNumber("212")
```

- In a dynamic language, this may be called “monkey patching”. Unlike Perl, Python, etc. Scala resolves implicits at compile time.

mongoDB

Caveats of Implicits

- Caveat #2: Normal Type Boundaries
  `def foo[A <: SomeComplexType]` will *not* allow implicit conversions.
- Use View Boundaries `def foo[A <% SomeComplexType]` instead!

Tricks with Implicits

- Scala's Compiler lets you use Implicits to pull off a few tricks. . .

Pimpin' Your Library

mongoDB

Pimpin' Your Library

- Implicit methods allow for the the "Pimp My Library Pattern"
- Coined by Martin Odersky in a 2006 Blog post. Similar to C# extension methods, Ruby modules.
- Uses implicit conversions to tack on new methods at runtime.
- Either return a new "Rich_" or anonymous class. . .

```
import com.mongodb.gridfs.{GridFSInputFile => MongoGridFSInputFile}

class GridFSInputFile protected[mongodb](override val underlying:
      MongoGridFSInputFile) extends GridFSFile {
  def filename_=(name: String) = underlying.setFilename(name)
  def contentType_=(cT: String) = underlying.setContentType(cT)
}

object PimpMyMongo {
  implicit def mongoConnAsScala(conn: Mongo) = new {
    def asScala = new MongoConnection(conn)
  }

  implicit def enrichGridFSInput(in: MongoGridFSInputFile) =
    new GridFSInputFile(in)
}

import PimpMyMongo._
```

Type Classes I

- Finally, Implicit parameters allow for emulating Type Classes
- Lets you create a list of “Acceptable” divergent values which can pass a single type boundary
- Used heavily in Scala 2.8 for constructs such as Numeric, Ordering and CanBuildFrom

mongoDB

Type Classes II

- A Vending Machine could accept divergent funding:

```
abstract class ValidVendingFunds[T]

object ValidVendingFunds {
  implicit object VisaOk extends ValidVendingFunds[VisaCard]
  implicit object AmexOk extends ValidVendingFunds[AmexCard]
  implicit object MCOk extends ValidVendingFunds[MastercardCard]
  implicit object PaperDollarOk extends ValidVendingFunds[DollarBill]
  implicit object PaperFiverOk extends ValidVendingFunds[FiveDollarBill]
  implicit object NickelOk extends ValidVendingFunds[NickelCoin]
  implicit object DimeOk extends ValidVendingFunds[DimeCoin]
  implicit object QuarterOk extends ValidVendingFunds[QuarterCoin]
  implicit object DollarCoinOk extends ValidVendingFunds[DollarCoin]
}

def addMoney[T : ValidVendingFunds](funding: T, amount: Double) =
    println("Added $%2.2f funding from %s.".format(amount, funding))

import ValidVendingFunds._

scala> addMoney(citibankVisa, 5.0)
Added $5.00 funding from Citibank Visa Card #4...

scala> addMoney(wellsFargoMastercard, 25.0)
Added $5.00 funding from Wells Fargo Mastercard Card #5...

scala> addMoney(dollarCoin, 1.0)
Added $1.00 funding from A Dollar coin
```

Type Classes III

- No real leeway, must explicitly define **all** acceptable classes, won't pass a subclass of a declared type
- Use scala.math.Numeric with this boundary trick to accept any numeric type (Double, Int, Float, BigDecimal, Short, etc etc)
- Type Classes described in great detail by D.C. Sobral at `http://dcsobral.blogspot.com/2010/06/implicit-tricks-type-class-pattern.html`

mongoDB

Type Classes IV

- Bonus Feature: Variant of the Type Class boundary syntax lets you capture manifests in 2.8.x:

```
implicit def tupleToGeoCoords[T : Numeric : Manifest](coords: (T, T)) =
      GeoCoords(coords._1, coords._2)

trait PushAllOp extends BarewordQueryOperator {

   def $pushAll[A <: Any : Manifest](args: (String, A)*): DBObject =
    if (manifest[A] <:< manifest[Iterable[_]])
      apply("$pushAll")(args.map(z => z._1 -> z._2.asInstanceOf[Iterable[_]]):_*)
    else if (manifest[A] <:< manifest[Product])
      apply("$pushAll")(args.map(z => z._1 ->
      z._2.asInstanceOf[Product].productIterator.toIterable): _*)
    else if (manifest[A].erasure.isArray)
      apply("$pushAll")(args.map(z => z._1 -> z._2.asInstanceOf[Array[_]].toIterable):
      _*)
    else
      throw new IllegalArgumentException("$pushAll may only be invoked with a (String,
      A) where String is the field name and A is an Iterable or Product/Tuple of values
       (got %s).".format(manifest[A]))

}
```

mongoDB

Questions?
Contact Info

- twitter: **@rit**
- email: **brendan@10gen.com**
- 10gen is *hiring!* We need smart engineers in both NY and Bay Area: `http://10gen.com/jobs`

mongoDB