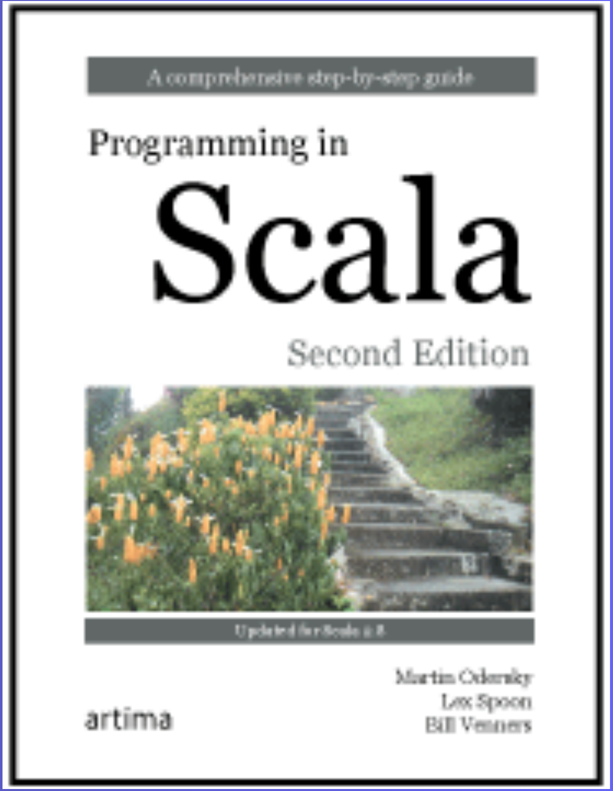
artima

Stairway to Scala - Flight 10

Packages and imports

Bill Venners
Dick Wall

www.artima.com



Flight 10 goal

Minimize coupling between modules using packages and imports. Understand Scala scoping and visibility rules.

Packages

- Modules minimize coupling
- Scala: multiple classes and packages in a single file
- The Java Way (perfectly valid in Scala):

```
package bobsrockets.navigation
class Navigator
```

artima

Another way of using packages

- An alternative way to the previous slide is:

```
package bobsrockets {
    package navigation {
        // In package bobsrockets.navigation
        class Navigator
        package tests {
            // In package bobsrockets.navigation.tests
            class NavigatorSuite
        }
    }
}
```

Slightly more compact

- A slightly more compact alternative:

```
package bobsrockets.navigation {
    // In package bobsrockets.navigation
    class Navigator
    package tests {
        // In package bobsrockets.navigation.tests
        class NavigatorSuite
    }
}
```

Nested packages

```
package bobsrockets {
    package navigation {
        class Navigator
    }
    package launch {
        class Booster {
            // No need to say bobsrockets.navigation.Navigator
            val nav = new navigation.Navigator
        }
    }
}
```

Going back to your roots

```
// In file launch.scala
package launch {
    class Booster3
}
// In file bobsrockets.scala
package bobsrockets {
    package navigation {
        package launch {
            class Booster1
        }
        class MissionControl {
            val booster1 = new launch.Booster1
            val booster2 = new bobsrockets.launch.Booster2
            val booster3 = new _root_.launch.Booster3
        }
    }
    package launch {
        class Booster2
    }
}
```

Imports

```
package bobsdelights
abstract class Fruit(
    val name: String,
    val color: String
)
object Fruits {
    object Apple extends Fruit("apple", "red")
    object Orange extends Fruit("orange", "orange")
    object Pear extends Fruit("pear", "yellowish")
    val menu = List(Apple, Orange, Pear)
}
```

Importing Bob's delights

```
// easy access to Fruit
import bobsdelights.Fruit

// easy access to all members of bobsdelights
import bobsdelights._

// easy access to all members of Fruits
import bobsdelights.Fruits._
```

Just in time imports

```
def showFruit(fruit: Fruit) {
    import fruit._
    println(name +"s are "+ color)
}
```

Scala's importing Kung Fu

- May appear anywhere
- May refer to objects (singleton or regular)
- May import packages themselves:

```
import java.util.regex
class AStarB {
    // Accesses java.util.regex.Pattern
    val pat = regex.Pattern.compile("a*b")
}
```

Selective, renaming, hiding

```
// import only Apple and Pear but not the others
import Fruits.{Apple, Pear}

// rename Apple on import
import Fruits.{Apple => McIntosh, Pear}

import java.sql.{Date => SDate}
import java.{sql => S}
val d = new S.Date

import Fruits.{_}  // equiv. to import Fruits._
import Fruits.{Apple => McIntosh, _}

import Notebooks._
import Fruits.{Apple => _, _}
```

Implicit imports

- Automatically available for every source file:

```
import java.lang._   // everything in the java.lang package
import scala._       // everything in the scala package
import Predef._      // everything in the Predef object
```

- java.lang - just like Java (System, Thread, etc.)
- scala - standard scala library (List, Map, etc.)
- Predef - types, methods, implicit conversions (assert, etc.)

Access modifiers

```
class Outer {
    class Inner {
        private def f() { println("f") }
        class InnerMost {
            f() // OK
        }
    }
    (new Inner).f() // error: f is not accessible
}
```

Protected members

```
package p {
    class Super {
        protected def f() { println("f") }
    }
    class Sub extends Super {
        f()
    }
    class Other {
        (new Super).f()
        // error: f is not accessible
    }
}
```

Public members and scope of protection

- Any member not private or protected is public
- No explicit modifier for public

- Scope of protection:

```
private[bobsrockets]

protected[navigation]
```

Scoped protection example

```
package bobsrockets {
    package navigation {
        private[bobsrockets] class Navigator {
            protected[navigation] def useStarChart() {}
            class LegOfJourney {
                private[Navigator] val distance = 100
            }
            private[this] var speed = 200
        }
    }
    package launch {
        import navigation._
        object Vehicle {
            private[launch] val guide = new Navigator
        }
    }
}
```

Effects of private modifiers

private[bobsrockets]	access within outer package
private[navigation]	same as package visibility in Java
private[Navigator]	same as private in Java
private[LegOfJourney]	same as private in Scala
private[this]	access only from same object

```
// effect of private[this]
val other = new Navigator
// this won't compile even if inside of class Navigator:
other.speed
```

Visibility and companion objects

```
class Rocket {
  import Rocket.fuel
  private def canGoHomeAgain = fuel > 20
}
object Rocket {
  private def fuel = 10
  def chooseStrategy(rocket: Rocket) {
    if (rocket.canGoHomeAgain)
      goHome()
    else
      pickAStar()
  }
  def goHome() {}
  def pickAStar() {}
}
```