

# What's new in Scala and the Scala IDE for Eclipse for 2.8.0

Miles Sabin, Chuusai Ltd.  
<http://www.chuusai.com/>

# Vastly Improved REPL

- Read Eval Print Loop: Interactive interpreter
  - Unusual for a statically typed JVM language
  - New maintainer: Paul Phillips
- New for 2.8.0
  - Completion (packages, classes, members)
  - Interactive debugging

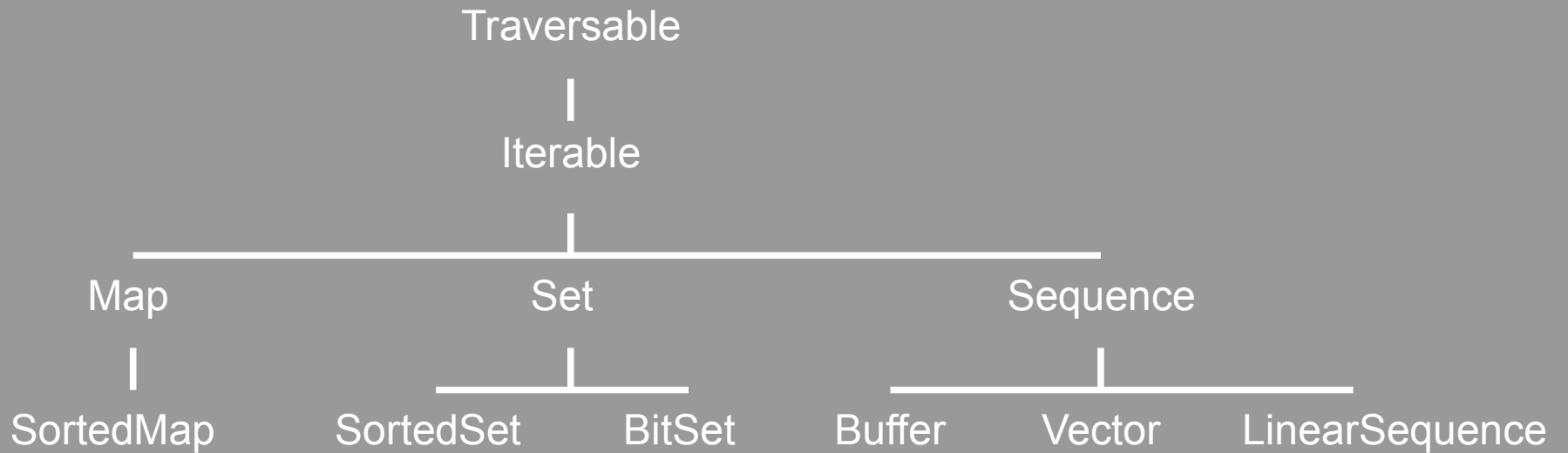
# New Collection Libraries

- Collections in  $\leq 2.7.5$ 
  - Evolved haphazardly
  - Hard work for implementors
  - Code duplication to avoid unnatural types
  - Unpredictable performance
  - Java interoperability incomplete

# New Collection Libraries

- Collections in 2.8.0
  - Coherent design in terms of traversals and builders
  - Builders support natural types
  - Systematic sharing reduces duplication
  - Lazy views now explicit
  - Java interoperability conversion bidirectional
  - Minimal impact on clients

# New Collection Libraries



# Package Objects

- How to support clients expecting old package layout?
- Package objects support additional members not on any class, trait, object

```
package object scala {  
    type List[+A] = scala.collection.immutable.List[A]  
    val List = scala.collection.immutable.List  
    ...  
}
```

- Can be used by any library provider

# Break (but not continue)

- Scala promotes functional styles — this can trip up imperative programmers

```
for(e <- l) if(p(e)) f(p) else ??? // How to bail out?
```

- Traditional break behaviour now provided via a library

```
import scala.util.control.Breaks._  
breakable {  
    for(e <- l) if(p(e)) f(p) else break // Continue from *  
}  
// *
```

# Type Constructor Inference

- Type constructor polymorphism originally introduced in 2.5.0.
- Intention was to provide natural types for collection methods
- Lack of type inference made this difficult in practice — new collections use alternative
- Patch being reviewed for 2.8.0, but not yet clear if it will be included

# Named and Default Arguments

- Method arguments can be supplied by name instead of position

```
def resize(width: Int, height: Int) = { ... }  
resize(width = 120, height = 42)
```

- Method definitions can provide default arguments as an alternative to overloading

```
def encode(s : String, encoding : String = "UTF-8")  
encode("Hello world") // Uses default encoding  
encode("Hello world", "ISO-8859-1") // Uses supplied
```

- Allows support for nested annotations

# Functional Objects

- An application of named and default arguments
- Functional update for case classes

```
case class Complex(val re : Double, val im : Double)
val c1 = Complex(1.0, -1.0)
val c2 = c1.copy(re = 2.0) // c2 == Complex(2.0, -1.0)
```

- Tuples are case classes so they inherit this behaviour

```
val p1 = (23, "foo")
val p2 = p1.copy(_2 = "bar") // p2 = (23, "bar")
```

# Delimited Continuations

- Powerful control abstraction capturing the concept of “the rest of the computation”
- Can be used to build coroutines, generators and more exotic things (type-safe printf)
- Enable event-driven I/O models without requiring explicit state machines
- Initially being provided via a compiler plugin

# Delimited Continuations

- call/cc (Scheme etc.) capture the entire subsequent control flow
- Scala continuations capture control flow within a delimited region (shift/reset)

```
import scala.continuations._
import scala.continuations.ControlContext._
val result =
  reset {
    shift {
      s : (String => String) => s("Hello")
    } + " world"
  } // result == "Hello world"
```

# Delimited Continuations

- Can be stored for later execution

```
var stored : (String => String) = _
reset {
  shift {
    s : (String => String) => stored = s
  } + " world"
}
stored("Hello") // == "Hello world"
```

This enables complex control flow behaviours

- Can be serialized for remote execution

# Type Specialization

- Adding `@specialize` annotation to generic type parameter — compiler generates specialized versions of
- Can eliminate expensive boxing and unboxing with higher order functions and arrays.
- Significantly reduce the cost of abstraction
- Nb. this doesn't eliminate erasure in general

# The Scala IDE for Eclipse

- Integration with the JDT continues
- Much functionality migrated to scalac
  - New interface for extracting semantic information for hyperlinks, hovers, completion
  - New build manager
- More robust basic functionality
- Dramatic simplification of the codebase
  - More maintainable
  - Lower barrier to contributions

# What's new in Scala and the Scala IDE for Eclipse for 2.8.0

Miles Sabin, Chuusai Ltd.  
<http://www.chuusai.com/>