PROGRAMMING IN PAIN

ENNO RUNNE

transmode

WHAT MAKES ME FEEL PAIN

Coding Scala made me value new things.

I reconsidered old Java truths about what is good code, and I try to take some of Scala's goodness home to Java – at the price of more noise.

Some concepts can't be translated easily.

MANY METHOD ARGUMENTS

MANY METHOD ARGUMENTS

```java
new Person("Enno", "Runne")
new Person("Enno", "Runne", "Sweden")
new Person("Enno", "Runne", "Sweden", "Stockholm")
```

MANY METHOD ARGUMENTS

```java
new Person("Enno", "Runne")
new Person("Enno", "Runne", "Sweden")
new Person("Enno", "Runne", "Sweden", "Stockholm")
new Person("Enno", "Runne", null, "Stockholm")
```

MANY METHOD ARGUMENTS

```java
new Person("Enno", "Runne")
new Person("Enno", "Runne", "Sweden")
new Person("Enno", "Runne", "Sweden", "Stockholm")
new Person("Enno", "Runne", null, "Stockholm")
```

Java

```scala
p = new Person("Enno", "Runne", city = "Stockholm")
```

Scala

MANY METHOD ARGUMENTS

```java
new Person("Enno", "Runne")
new Person("Enno", "Runne", "Sweden")
new Person("Enno", "Runne", "Sweden", "Stockholm")
new Person("Enno", "Runne", null, "Stockholm")
```

Java

```scala
p = new Person("Enno", "Runne", city = "Stockholm")
p = new Builder("Enno", "Runne").city("Stockholm").build();
```

Scala

MANY METHOD ARGUMENTS

```java
new Person("Enno", "Runne")
new Person("Enno", "Runne", "Sweden")
new Person("Enno", "Runne", "Sweden", "Stockholm")
new Person("Enno", "Runne", null, "Stockholm")
```

Java

```scala
p = new Person("Enno", "Runne", city = "Stockholm")
p = new Builder("Enno", "Runne").city("Stockholm").build();
```

Scala

builders relieve my pain

MANY METHOD ARGUMENTS

```java
new Person("Enno", "Runne")
new Person("Enno", "Runne", "Sweden")
new Person("Enno", "Runne", "Sweden", "Stockholm")
new Person("Enno", "Runne", null, "Stockholm")
```

Java

```scala
p = new Person("Enno", "Runne", city = "Stockholm")
p = new Builder("Enno", "Runne").city("Stockholm").build();
```

Scala

FLUENT INTERFACE

Java

```
class Builder {
   String fn, ln, co, ci;

   Builder(String fn, String ln) {
      this.fn = fn;
      this.ln = ln;
   }

   Person build() {
      return new Person(fn, ln, co, ci);
   }

   Builder city(String ci) {
      this.ci = ci;
      return this;
   }

   Builder country(String co) {
      this.co = co;
      return this;
   }
```

Scala

```
class Person(
  val fn: String,
  val ln: String,
  val country: String = null,
  val city: String = null)
```

FLUENT INTERFACE

Java

```
class Builder {
   String fn, ln, co, ci;

   Builder(String fn, String ln) {
      this.fn = fn;
      this.ln = ln;
   }

   Person build() {
      return new Person(fn, ln, co, ci);
   }

   Builder city(String ci) {
      this.ci = ci;
      return this;
   }

   Builder country(String co) {
      this.co = co;
      return this;
   }
```

Scala

```
class Person(
  val fn: String,
  val ln: String,
  val country: String = null,
  val city: String = null)
```

VARIABLES AND VALUES

VARIABLES AND VALUES

```
var count = 6
```
variable, multiple assignment **Scala**

```
val count = 6
```
value, single assignment **Scala**

VARIABLES AND VALUES

```
var count = 6
```
variable, multiple assignment **Scala**

```
val count = 6
```
value, single assignment **Scala**

VARIABLES AND VALUES

```
var count = 6
```
variable, multiple assignment — Scala

```
int count = 6;
```
Java

```
val count = 6
```
value, single assignment — Scala

```
final int count = 6;
```
Java

VARIABLES AND VALUES

```scala
var count = 6
```
variable, multiple assignment — Scala

```java
int count = 6;
```
Java

```scala
val count = 6
```
value, single assignment — Scala

```java
final int count = 6;
```
Java

final keyword can mimic Scala's val

VARIABLES AND VALUES

```
var count = 6
```
variable, multiple assignment — Scala

```
int count = 6;
```
Java

```
val count = 6
```
value, single assignment — Scala

```
final int count = 6;
```
Java

VALUES AT LARGE

Scala

- stresses immutability
- can't accidentally change the wrong object
- no mutable state, convenient in multi-threaded apps
- just as String class in Java, value objects in DDD

Java

- JavaBean standard: get and set methods for all fields
- easy:
  - remove all setters
  - make fields final

VALUES AT LARGE

Scala

- stresses immutability
- can't accidentally change the wrong object
- no mutable state, convenient in multi-threaded apps
- just as String class in Java, value objects in DDD

Java

- JavaBean standard: get and set methods for all fields
- easy:
  - remove all setters
  - make fields final

VALUES AT LARGE

Scala

- stresses immutability
- can't accidentally change the wrong object
- no mutable state, convenient in multi-threaded apps
- just as String class in Java, value objects in DDD

Java

- JavaBean standard: get and set methods for all fields
- easy:
  - remove all setters
  - make fields final

VALUES AT LARGE

Scala

- stresses immutability
- can't accidentally change the wrong object
- no mutable state, convenient in multi-threaded apps
- just as String class in Java, value objects in DDD

Java

- JavaBean standard: get and set methods for all fields
- easy:
  - remove all setters
  - make fields final

different conviction

VALUES AT LARGE

Scala

- stresses immutability
- can't accidentally change the wrong object
- no mutable state, convenient in multi-threaded apps
- just as String class in Java, value objects in DDD

Java

- JavaBean standard: get and set methods for all fields
- easy:
  - remove all setters
  - make fields final

SMALL CLASSES

SMALL CLASSES

Scala

```
class Xyz(val s: String)
```

- very little code required
- no requirement on matching file names
- even several packages in one file

Java

```
public class Xyz {
   public final String s;
   public Xyz(String s) {
      this.s = s;
   }
}
```

- quite a bit of text
- one public class per file

SMALL CLASSES

Scala

```
class Xyz(val s: String)
```

- very little code required
- no requirement on matching file names
- even several packages in one file

Java

```
public class Xyz {
   public final String s;
   public Xyz(String s) {
      this.s = s;
   }
}
```

- quite a bit of text
- one public class per file

more work to split code into smaller classes

SMALL CLASSES

Scala

```
class Xyz(val s: String)
```

- very little code required
- no requirement on matching file names
- even several packages in one file

Java

```
public class Xyz {
   public final String s;
   public Xyz(String s) {
      this.s = s;
   }
}
```

- quite a bit of text
- one public class per file

COLLECTIONS

COLLECTIONS

Scala
```
val words = List("one", "two", "three")
val wordsWithTs = words.filter(elem => elem.contains("t"))
```

COLLECTIONS

Scala

```
val words = List("one", "two", "three")
val wordsWithTs = words.filter(elem => elem.contains("t"))
```

Java

```
List<String> words = Arrays.asList("one", "two", "three");
List<String> wordsWithTs = new ArrayList<String>();
for (String elem : words) {
   if (elem.contains("t")) {
      wordsWithTs.add(elem);
   }
}
```

COLLECTIONS

Scala
```
val words = List("one", "two", "three")
val wordsWithTs = words.filter(elem => elem.contains("t"))
```

COLLECTIONS

Scala

```
val words = List("one", "two", "three")
val wordsWithTs = words.filter(elem => elem.contains("t"))
```

Java (with Google Guava)

```
List<String> words = Arrays.asList("one", "two", "three");
Collection<String> wordsWithTs = filter(words,
   new Predicate<String>() {
      public boolean apply(String elem) {
         return elem.contains("t");
      }
   });
```

COLLECTIONS

Scala

```
val words = List("one", "two", "three")
val wordsWithTs = words.filter(elem => elem.contains("t"))
```

Java (with Google Guava)

```
List<String> words = Arrays.asList("one", "two", "three");
Collection<String> wordsWithTs = filter(words,
  new Predicate<String>() {
     public boolean apply(String elem) {
        return elem.contains("t");
     }
  });
```

COLLECTIONS

Scala

```
val words = List("one", "two", "three")
val wordsWithTs = words.filter(elem => elem.contains("t"))
```

Java (with Google Guava)

```
List<String> words = Arrays.asList("one", "two", "three");
Collection<String> wordsWithTs = filter(words,
  new Predicate<String>() {
     public boolean apply(String elem) {
        return elem.contains("t");
     }
  });
```

Google's Guava eases the pain

COLLECTIONS

Scala

```
val words = List("one", "two", "three")
val wordsWithTs = words.filter(elem => elem.contains("t"))
```

Java (with Google Guava)

```
List<String> words = Arrays.asList("one", "two", "three");
Collection<String> wordsWithTs = filter(words,
  new Predicate<String>() {
     public boolean apply(String elem) {
        return elem.contains("t");
     }
  });
```

COMPOSITION

COMPOSITION

Scala

```
class A
abstract class B
trait C
trait D

A extends B

A extends B with C with D
```

Java

```
class A
abstract class B
interface C
interface D

A extends B

A extends B implements C, D
```

COMPOSITION

Scala

```
trait HasName {
   var name: String = ""
   def getName() = name +
           getOther()
   def getOther(): String
}
```

COMPOSITION

Scala

```
trait HasName {
   var name: String = ""
   def getName() = name +
            getOther()
   def getOther(): String
}
```

COMPOSITION

Scala

```
trait HasName {
   var name: String = ""
   def getName() = name +
            getOther()
   def getOther(): String
}
```

Java

```
interface HasName {
   String getName();
   String getOther();
}

abstract class HasNameImpl
        implements HasName {
   String name;
   public String getName() {
      return name + getOther();
   }
}
```

COMPOSITION

Scala

```
class User
extends HasName {
   def getOther() = "!"
}

val u = new User
u.name = "name"
u.getName()
```

COMPOSITION

Scala

```
class User
extends HasName {
   def getOther() = "!"
}


val u = new User
u.name = "name"
u.getName()
```

Java

```
class User implements HasName {
   HasName outer = this;
   HasNameImpl impl = new HasNameImpl() {
      String getOther() {
         return outer.getOther();
      }};
   String getName() {
      return impl.getName();
   }
   String getOther() { return "!"; }
   void setName(String s) { impl.name = s; }
}

User uc = new User();
uc.setName("name");
uc.getName();
```

ADVANCED SWITCH

ADVANCED SWITCH

Java

```
if (o instanceof Xyz) {
   doThis();
}
else if (o instanceof Rst) {
   doThat();
}
```

ADVANCED SWITCH

Java

```
if (o instanceof Xyz) {
   doThis();
}
else if (o instanceof Rst) {
   doThat();
}
```

Java

```
catch (Xyz o) {
   doThis();
}
catch (Rst o) {
   doThat();
}
```

ADVANCED SWITCH

Java

```
catch (Xyz o) {
   doThis();
}
catch (Rst o) {
   doThat();
}
```

ADVANCED SWITCH

Scala

```
catch {
  case o: Xyz => {
    doThis();
  }
  case o: Rst => {
    doThat();
  }
}
```

Java

```
catch (Xyz o) {
   doThis();
}
catch (Rst o) {
   doThat();
}
```

ADVANCED SWITCH

Scala

```
o match {
  case o: Xyz => {
    doThis();
  }
  case o: Rst => {
    doThat();
  }
  case "hi" => {
    doSomething();
  }
}
```

ADVANCED SWITCH

Scala

```
o match {
  case o: Xyz => {
    doThis();
  }
  case o: Rst => {
    doThat();
  }
  case "hi" => {
    doSomething();
  }
}
```

ADVANCED SWITCH

Scala

```
o match {
  case o: Xyz => {
    doThis();
  }
  case o: Rst => {
    doThat();
  }
  case "hi" => {
    doSomething();
  }
}
```

Java

```
match(o,
  case_instanceOf(Xyz.class,
    new Code<String>() {
      public void run(String obj) {
        doThis(obj);
      }
    }),
  case_instanceOf(Rst.class,
    new Code<String>() {
      public void run(String obj) {
        doThat(obj);
      }
    }),
  case_eq("hi",
    new Code<String>() {
      public void run(String obj) {
        doSomething(obj);
```

ADVANCED SWITCH

Scala

```
o match {
  case o: Xyz => {
    doThis();
  }
  case o: Rst => {
    doThat();
  }
  case "hi" => {
    doSomething();
  }
}
```

Java

```
match(o,
  case_instanceOf(Xyz.class,
    new Code<String>() {
      public void run(String obj) {
        doThis(obj);
      }
    }),
  case_instanceOf(Rst.class,
    new Code<String>() {
      public void run(String obj) {
        doThat(obj);
      }
    }),
  case_eq("hi",
    new Code<String>() {
      public void run(String obj) {
        doSomething(obj);
```

Wednesday, February 16, 2011

Thank you for listening.

Enno Runne