Scala, a Scalable Language

Alan Dipert

September 3, 2009

What Scala Is

- Multi-paradigm: purely object oriented, but with functional capabilities

What Scala Is

- Multi-paradigm: purely object oriented, but with functional capabilities
- Statically typed

What Scala Is

- Multi-paradigm: purely object oriented, but with functional capabilities
- Statically typed
- Seamless Java interoperability

What Scala Is

- Multi-paradigm: purely object oriented, but with functional capabilities
- Statically typed
- Seamless Java interoperability
- Open source (BSD License)

What Scala Is

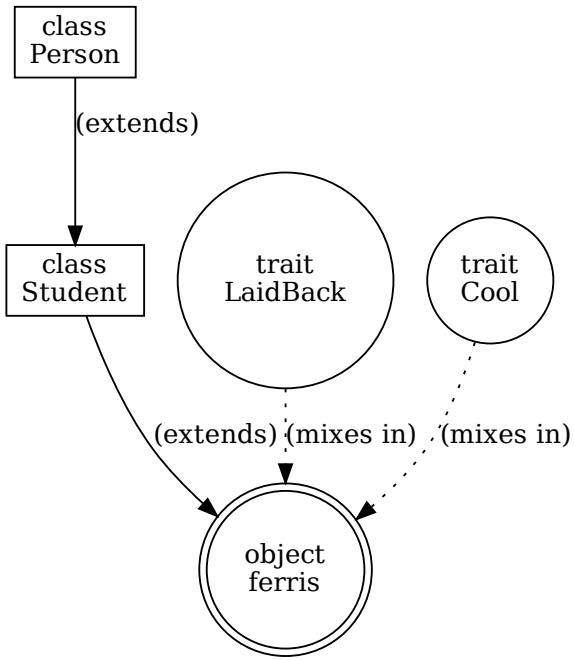
- Multi-paradigm: purely object oriented, but with functional capabilities
- Statically typed
- Seamless Java interoperability
- Open source (BSD License)
- Latest of several JVM languages from creator Martin Odersky and team at EPFL

What Scala Is

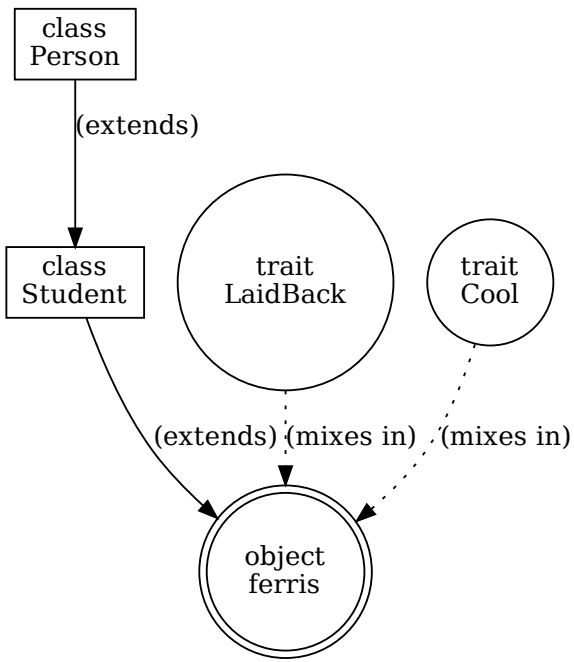
- Multi-paradigm: purely object oriented, but with functional capabilities
- Statically typed
- Seamless Java interoperability
- Open source (BSD License)
- Latest of several JVM languages from creator Martin Odersky and team at EPFL
- Cool: `def fact(x:BigInt) = (BigInt(1) to x).reduceLeft(_*_)`

Object Oriented Programming in Scala

- Classes and objects similar to Java...

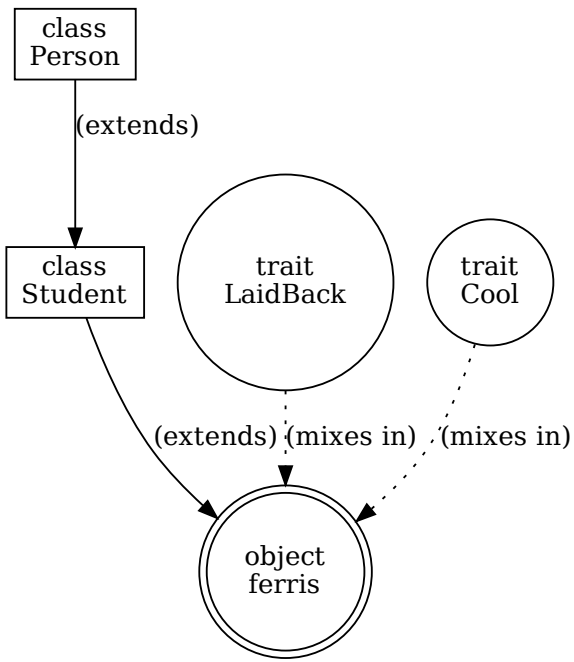


Object Oriented Programming in Scala



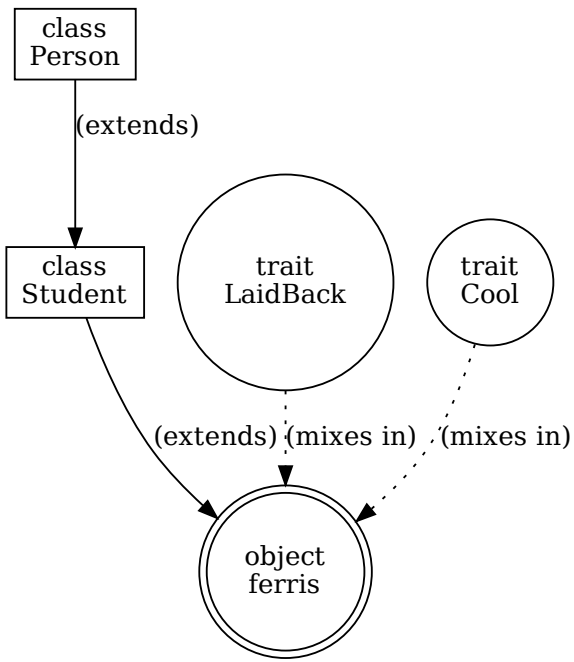
- Classes and objects similar to Java...
- ...except "everything is an object." - no primitive types like Java

Object Oriented Programming in Scala



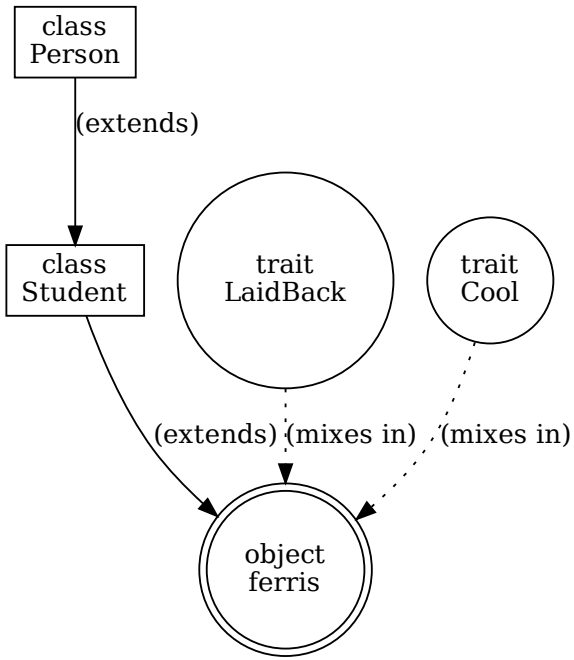
- Classes and objects similar to Java...
- ...except "everything is an object." - no primitive types like Java
- Automatic getters/setters (like :attr_accessor in Ruby)

Object Oriented Programming in Scala



- Classes and objects similar to Java...
- ...except "everything is an object." - no primitive types like Java
- Automatic getters/setters (like :attr_accessor in Ruby)
- "Singleton objects" instead of static methods in classes

Object Oriented Programming in Scala



- Classes and objects similar to Java...
- ...except "everything is an object." - no primitive types like Java
- Automatic getters/setters (like :attr_accessor in Ruby)
- "Singleton objects" instead of static methods in classes
- "Traits," behaviors mixed in with classes and objects, simplify multiple inheritance

Class Definition in Scala

```
class Person(val name:String, val age:Int) {
  override def toString = "My name is %s and I'm %d years
old.".format(name,age)
}

val thePrez = new Person("Barack Obama", 48)

println(thePrez.toString)

"My name is Barack Obama and I'm 48 years old."
```

Inheritance in Scala

```
class Person(val name:String, val age:Int) {
  override def toString = "My name is %s and I'm %d years old.".format(name,age)
}

class Student(val school:String, name:String, age:Int) extends Person(name,age) {
  override def toString = super.toString+" I go to %s.".format(school)
}

val alan = new Student("RIT", "Alan Dipert", 25)

println(alan toString)

"My name is Alan Dipert and I'm 25 years old. I go to RIT."
```

Traits in Scala

```
class Person(val name:String, val age:Int) {
  override def toString = "My name is %s and I'm %d years old.".format(name,age)
}

class Student(val school:String, name:String, age:Int) extends Person(name,age) {
  override def toString = super.toString+" I go to %s.".format(school)
}

trait Cool {
  override def toString = super.toString+" And I'm cool."
}

trait LaidBack {
  override def toString = super.toString+" And I'm laid back."
}

val ferris = new Student("Shermer High", "Ferris Bueller", 17) with LaidBack with Cool

println(ferris toString)

"My name is Ferris Bueller and I'm 17 years old. I go to Shermer High. And I'm laid back. And
I'm cool."
```

Object Oriented Programming Conclusions

- Scala's object system gives you a lot of control

Object Oriented Programming Conclusions

- Scala's object system gives you a lot of control
- It feels much more like Ruby or Python than Java or C++

Object Oriented Programming Conclusions

- Scala's object system gives you a lot of control
- It feels much more like Ruby or Python than Java or C++
- It's a carefully designed fusion of useful features and Java compatibility

Object Oriented Programming Conclusions

- Scala's object system gives you a lot of control
- It feels much more like Ruby or Python than Java or C++
- It's a carefully designed fusion of useful features and Java compatibility
- Like a lot of Scala, you can start by doing things the Java way

Object Oriented Programming Conclusions

- Scala's object system gives you a lot of control
- It feels much more like Ruby or Python than Java or C++
- It's a carefully designed fusion of useful features and Java compatibility
- Like a lot of Scala, you can start by doing things the Java way
- Don't worry, it's still possible to write crappy and overly complex code

Functional Programming Introduction

- FP is the paradigm of avoiding state and mutability in code, "functions and data the same"

Functional Programming Introduction

- FP is the paradigm of avoiding state and mutability in code, "functions and data the same"
- Around in practical form since Lisp (1958), popular lately because of renewed interest in concurrency

Functional Programming Introduction

- FP is the paradigm of avoiding state and mutability in code, "functions and data the same"
- Around in practical form since Lisp (1958), popular lately because of renewed interest in concurrency
- Side-effect free functions and immutable data structures have definite advantages in many domains

Functional Programming Introduction

- FP is the paradigm of avoiding state and mutability in code, "functions and data the same"
- Around in practical form since Lisp (1958), popular lately because of renewed interest in concurrency
- Side-effect free functions and immutable data structures have definite advantages in many domains
- On the JVM, FP alternatives are Clojure, CAL

Functional Programming Introduction

- FP is the paradigm of avoiding state and mutability in code, "functions and data the same"
- Around in practical form since Lisp (1958), popular lately because of renewed interest in concurrency
- Side-effect free functions and immutable data structures have definite advantages in many domains
- On the JVM, FP alternatives are Clojure, CAL
- In Java terms, anonymous inner classes on crack

Functional Programming in Scala

- Scala has first-class functions

Functional Programming in Scala

- Scala has first-class functions
- Many Collection classes provide convenience higher order functions: `reduceLeft`, `foldLeft`, `filter`, `forall`

Functional Programming in Scala

- Scala has first-class functions
- Many Collection classes provide convenience higher order functions: `reduceLeft`, `foldLeft`, `filter`, `forall`
- Lots of functional stand-bys: `head`, `tail`

Functional Programming in Scala

- Scala has first-class functions
- Many Collection classes provide convenience higher order functions: `reduceLeft`, `foldLeft`, `filter`, `forall`
- Lots of functional stand-bys: `head`, `tail`
- Functional techniques help you avoid off-by-one errors, temporary variables, and explicit loops

Functional Programming in Scala

- Scala has first-class functions
- Many Collection classes provide convenience higher order functions: `reduceLeft`, `foldLeft`, `filter`, `forall`
- Lots of functional stand-bys: `head`, `tail`
- Functional techniques help you avoid off-by-one errors, temporary variables, and explicit loops
- Chained higher-order functions let you write common imperative for/if patterns concisely

Functional Programming in Scala Examples

- Sum a list of Integers:
- `List(1,5,3,2).reduceLeft((x,y) => x+y) (Int = 11)`

Functional Programming in Scala Examples

- Sum a list of Integers:
- `List(1,5,3,2).reduceLeft((x,y) => x+y) (Int = 11)`
- More concisely:
- `List(1,5,3,2).reduceLeft(_+_) (Int = 11)`

Functional Programming in Scala Examples

- Sum a list of Integers:
- `List(1,5,3,2).reduceLeft((x,y) => x+y) (Int = 11)`
- More concisely:
- `List(1,5,3,2).reduceLeft(_+_) (Int = 11)`
- Prune odd numbers:
- `List(1,2,3,4).filter(x => x % 2 == 0) (List[Int] = List(2,4))`

Functional Programming in Scala Examples

- Sum a list of Integers:
- `List(1,5,3,2).reduceLeft((x,y) => x+y) (Int = 11)`
- More concisely:
- `List(1,5,3,2).reduceLeft(_+_) (Int = 11)`
- Prune odd numbers:
- `List(1,2,3,4).filter(x => x % 2 == 0) (List[Int] = List(2,4))`
- Prune odd numbers, get sum:
- `List(1,2,3,4).filter(_ % 2 == 0).reduceLeft(_+_) (Int = 6)`

Functional Programming in Scala Examples

- Sum a list of Integers:
- `List(1,5,3,2).reduceLeft((x,y) => x+y) (Int = 11)`
- More concisely:
- `List(1,5,3,2).reduceLeft(_+_) (Int = 11)`
- Prune odd numbers:
- `List(1,2,3,4).filter(x => x % 2 == 0) (List[Int] = List(2,4))`
- Prune odd numbers, get sum:
- `List(1,2,3,4).filter(_ % 2 == 0).reduceLeft(_+_) (Int = 6)`
- Or, in Scala 2.8:
- `List(1,2,3,4).filter(_ % 2 == 0) sum (Int = 6)`

Putting it All Together: a Simple Application

- A program that adds a list of numbers taken as a command line argument, in three languages

Putting it All Together: a Simple Application

- A program that adds a list of numbers taken as a command line argument, in three languages
- Imperative (C), imperative/OO (Java), and OO/functional (Scala)

Putting it All Together: a Simple Application

- A program that adds a list of numbers taken as a command line argument, in three languages
- Imperative (C), imperative/OO (Java), and OO/functional (Scala)
- **Warning: always use the right tool for the job!**
- This is just a demonstration.

Putting it All Together: a Simple Application

- A program that adds a list of numbers taken as a command line argument, in three languages
- Imperative (C), imperative/OO (Java), and OO/functional (Scala)
- **Warning: always use the right tool for the job!**
- This is just a demonstration.
- Our program is supposed to take one argument, a comma-delimited list of integers to sum.
- Usage: `./summer 1,2,3 should print 6`

C

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct num_list {
  int size;
  int *nums;
} num_list;

int calc_sum(struct num_list *nlist) {
  int i, sum = 0;
  for(i = 0; i < nlist->size; i++) sum += nlist->nums[i];
  return sum;
}

struct num_list * convert_input(char *input) {
  struct num_list *nlist = (struct num_list*)malloc(sizeof(struct num_list));
  nlist->nums = (int*)malloc(100);
  int count = 0;
  char *token = strtok(input, ",");
  nlist->nums[count++] = atoi(token);
  while (token = strtok(NULL, ",")) nlist->nums[count++] = atoi(token);
  nlist->size = count;
  return nlist;
}

int main(int argc, char **argv) {
  if(argc > 1) {
    printf("%i\n", calc_sum(convert_input(argv[1])));
  } else {
    fprintf(stderr, "Please provide a list of integers in format 1,2,3\n");
  }
  return 0;
}
```

Java

```java
public class Summer {

  public static int calcSum(int [] nums) {
    int sum = 0;
    for(int num : nums) sum += num;
    return sum;
  }

  public static int[] convertInput(String arg) {
    String[] numStrings = arg.split(",");
    int[] numInts = new int[numStrings.length];
    for(int i = 0; i < numStrings.length; i++) {
      numInts[i] = Integer.parseInt(numStrings[i]);
    }
    return numInts;
  }

  public static void main (String [] args) {
    if(args.length > 0) {
      System.out.println(calcSum(convertInput(args[0])));
    } else {
      System.out.println("Please provide a list of integers in the format 1,2,3");
    }
  }
}
```

Scala

```scala
object Summer {

  def calcSum(n:Seq[Int]) = n.reduceLeft(_+_)

  def convertInput(a:String) = a.split(",").map(_.toInt)

  def main(args:Array[String]) {
    if(args.length > 0) {
      println(calcSum(convertInput(args.first)))
    } else {
      println("Please provide a list of integers in the format 1,2,3")
    }
  }
}
```

Bonus! Ruby

```ruby
class Summer

  def initialize(args)
    @args = args
  end

  def calcSum(arr)
    arr.inject { |x,y| x + y }
  end

  def convertInput(str)
    str.split(",").map { |x| x.to_i }
  end

  def result
    puts calcSum(convertInput(@args))
  end

end

if ARGV[0]
  Summer.new(ARGV[0]).result
else
  puts "Please provide a list of integers in the format 1,2,3"
end
```

Thank You!

- The Scala Homepage [http://www.scala-lang.org/]
- Programming in Scala, the definitive book [http://www.artima.com/shop/programming_in_scala]
- git repository for this presentation [http://github.com/alandipert/scala-presentation/tree/master]
- alan.dipert@gmail.com
- http://alan.dipert.org/