Scala for Java Programmers

An Introduction

Dianne Marsh
Emerging Technologies for the Enterprise
dmarsh@srtsolutions.com
3/27/2008

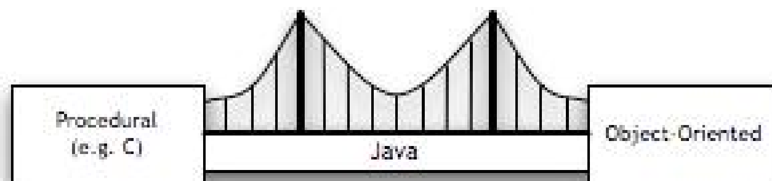
What is Scala?

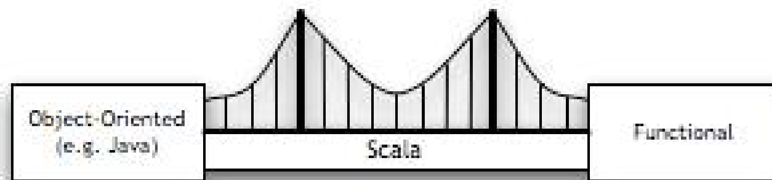
- Object-oriented/functional hybrid
- Designed by Martin Odersky
- Download at www.scala-lang.org
- Runs on JVM

SRT SOLUTIONS
We Develop Software Developers

What's different about Scala?

- Hybrid
- Interoperable with Java
- "Feel" of a scripting language





SRT SOLUTIONS
We Develop Software Developers

Small language

- No statics, operators, primitives
- Few control structures
- Building blocks
- Libraries

SRT SOLUTIONS
We Develop Software Developers

What is Functional Programming?

- Historically: mathematical, research
- Intent: improve everyday programming
  - Emphasis on functions (simplicity)
  - No side effects (pure, ease of debugging)
  - Good for concurrency
  - Minimized dependencies

SRT SOLUTIONS
We Develop Software Developers

Outline for Discussion

- Show a bit of Java
- Show a corresponding bit of Scala
- Compare/contrast
- Get your feedback

SRT SOLUTIONS
We Develop Software Developers

The problem

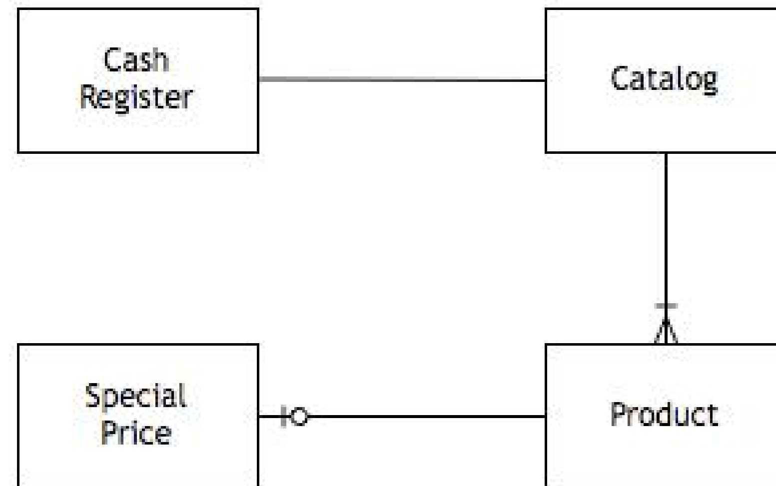
- Supermarket checkout,
  with special pricing on some units
- Exercise from codekata.pragprog.com

```
Item    Unit       Special
        Price      Price
--------------------------
   A      50       3 for 130
   B      30       2 for 45
   C      20
   D      15
```

SRT SOLUTIONS
We Develop Software Developers

Class Overview

- CashRegister
- Catalog
- Product
- SpecialPrice



SRT SOLUTIONS
We Develop Software Developers

“SpecialPrice”

```
// ******************** Java ************************
public class SpecialPrice {
    private int quantity;
    private int price;
    public SpecialPrice(int quantity, int price) {
        this.quantity = quantity;
        this.price = price;
    }
    public int getQuantity() {
        return quantity;
    }
    public int getPrice() {
        return price;
    }
}
```

SRT SOLUTIONS
We Develop Software Developers

“SpecialPrice”

```
// ******************** Java ************************
public class SpecialPrice {
    private int quantity;
    private int price;
    public SpecialPrice(int quantity, int price) {
        this.quantity = quantity;
        this.price = price;
    }
    public int getQuantity() {
        return quantity;
    }
    public int getPrice() {
        return price;
    }
}


// ******************* Scala *************************
class SpecialPrice(val quantity: Int, val price: Int) {}
```

SRT SOLUTIONS
We Develop Software Developers

SpecialPrice Takeaway

- Scala does the obvious things by default
- Minimizes boilerplate
- Example: Classes can take parameters

SRT SOLUTIONS
We Develop Software Developers

“Product” class

```
// ******************* Java *************************
public class Product {
    private String productCode;
    private int unitPrice;
    private SpecialPrice specialPrice;
    public Product(String productCode, int unitPrice, SpecialPrice specialPrice) {
        this.productCode = productCode;
        this.unitPrice = unitPrice;
        this.specialPrice = specialPrice;
    }
    public String getProductCode() {return productCode;}
    public int getUnitPrice() {return unitPrice;}
    public SpecialPrice getSpecialPrice() {return specialPrice;}
}
```

SRT SOLUTIONS
We Develop Software Developers

“Product” class

```
// ******************* Java *************************
public class Product {
    private String productCode;
    private int unitPrice;
    private SpecialPrice specialPrice;
    public Product(String productCode, int unitPrice, SpecialPrice specialPrice) {
        this.productCode = productCode;
        this.unitPrice = unitPrice;
        this.specialPrice = specialPrice;
    }
    public String getProductCode() {return productCode;}
    public int getUnitPrice() {return unitPrice;}
    public SpecialPrice getSpecialPrice() {return specialPrice;}
}

// ****************** Scala  ************************
class Product(val productCode: String, val unitPrice: Int,
    val specialPrice: Option[SpecialPrice]) { }
```

SRT SOLUTIONS
We Develop Software Developers

Product Takeaway

- Simple
- Option[T] gives an explicit way to say “we may or may not have a T instance”
- (Hang on … more about that in a moment)

SRT SOLUTIONS
We Develop Software Developers

Catalog Overview

Catalog
-products : Map<String, Product>
+Catalog() +addProduct(Product) +getProduct(String) : Product +getcodes() : Collection<String>

SRT SOLUTIONS
We Develop Software Developers

Catalog in Detail – 1 of 2

```
// ******************** Java ************************
public class Catalog {
    private Map<String, Product> products;
    public Catalog() {
        products = new HashMap<String, Product>();
    }
    void addProduct(Product p) { … }
    public Product getProduct(String code) { … }
    public Collection<String> getCodes() { … }
}
// ******************** Scala ************************
class  Catalog  {
    val products = new HashMap[String, Product]()
    def addProduct(p: Product)  { … }
    def getProduct(code: String) { … }
    def elements { … }
    def codes  { …}
}
```

SRT SOLUTIONS
We Develop Software Developers

Catalog in Detail – 2 of 2

```
// ******************** Java ************************
public class Catalog {
    …
    void addProduct(Product p) {
        products.put(p.getProductCode(), p);
    }
    public Product getProduct(String code) {
        return products.get(code);
    }
    public Collection<String> getCodes() {
        return Collections.unmodifiableCollection(products.keySet());
    }
}
// ******************** Scala ************************
class  Catalog  {
    …
    def addProduct(p: Product) = products += (p.productCode -> p)
    def getProduct(code: String) =  products(code)
    def elements = products.elements
    def codes = products.keys
}
```

SRT SOLUTIONS
We Develop Software Developers

Catalog Takeaway (1)

- Clean, simple syntax.
  - NOT special-cased Scala syntax for maps.
  - operators are really methods
  - += is just a lib-defined method (on Map)
  - -> is just a method that creates a Tuple
    - p.productCode -> p is really just:

      (p.productCode).->(p) which returns a Tuple2

SRT SOLUTIONS
We Develop Software Developers

Catalog Takeaway (2)

- Clean, simple syntax.
  - less “boilerplate” (types not repeated, but inferred, including return types)
  - no need for braces around method bodies if definition is a single expression
  - note that “val products …” is implicitly invoked whenever an instance is created

SRT SOLUTIONS
We Develop Software Developers

Catalog Takeaway (3)

- Option[T]: optional value
- Preferred to Java "object reference or null" idiom
  - Option[T] gives you Some(T) and None
    - Some(T) means "I have this T"
    - None means "I don't have anything"
  - Case classes can be used in explicit matches to drive logic (more in a moment)
  - FREE out-of-band value over any base type

SRT SOLUTIONS
We Develop Software Developers

CashRegister Overview

CashRegister
-ourCat : Catalog -quantities : Map<String, Integer>
+CashRegister() +scan(String) +total() : int

SRT SOLUTIONS
We Develop Software Developers

Comparing Init Code

```
// *************** Java ***************
public class CashRegister {
    private Catalog ourCat;
    private Map<String, Integer> quantities;

    public CashRegister(Catalog catalog) {
        ourCat = catalog;
        quantities = new HashMap<String, Integer>();
        for (String code : catalog.getCodes()) {
            quantities.put(code, 0);
        }
    }

// ************** Scala ***************
class CashRegister(catalog: Catalog) {

    val quantities = new HashMap[String, Int]()
    for (code <- catalog codes) quantities += code -> 0

    …
}
```

SRT SOLUTIONS
We Develop Software Developers

Comparing Scan Code

```
// ************* Java *************************
public class CashRegister {
    …
    public void scan(String code) {
        quantities.put(code, quantities.get(code) + 1);
    }

    public int total() { … }
}

// ************* Scala *************************
class CashRegister(catalog : Catalog) {
    …
    def scan(code: String) =
        quantities(code) = quantities(code) + 1
    def total: Int = { … }
}
```

SRT SOLUTIONS
We Develop Software Developers

Java CashRegister (total)

```
public class CashRegister {
    …
    public int total() {
        int sum = 0;
        for (String code : quantities.keySet()) {
            int count = quantities.get(code);
            if (count > 0) {
                Product p = ourCat.getProduct(code);
                SpecialPrice special = p.getSpecialPrice();
                if (special != null) {
                    int specialCount = count / special.getQuantity();
                    int remainCount = count % special.getQuantity();
                    sum += (specialCount * special.getPrice() + remainCount
                            * p.getUnitPrice());
                } else {
                    sum += count * p.getUnitPrice();
                }
            }
        }
        return sum;
    }
}
```

SRT SOLUTIONS
We Develop Software Developers

Scala CashRegister (total)

```
class CashRegister(catalog : Catalog) {
    …
    def total : Int = {
        var sum = 0
        for {
            (code, count) <- quantities.elements
            if count > 0
        } {
            val p = catalog getProduct code
            sum += (p.specialPrice match {
                case Some(s) =>
                    val specialCount = count / s.quantity
                    val remainCount = count % s.quantity
                    specialCount * s.price + remainCount * p.unitPrice
                case None => count * p.unitPrice
            })
        }
        sum
    }
}
```

SRT SOLUTIONS
We Develop Software Developers

Testing the Scala Version

```
object CashRegisterTest extends Application {

  val cat = new Catalog()
  cat.addProduct(new Product("A", 50, Some(new SpecialPrice(3, 130))))
  cat.addProduct(new Product("B", 30, Some(new SpecialPrice(2, 45))))
  cat.addProduct(new Product("C", 20, None))
  cat.addProduct(new Product("D", 15, None))

  val reg = new CashRegister(cat)

  reg scan "A"
  reg scan "B"
  reg scan "A"
  reg scan "B"
  reg scan "B"
  reg scan "C"

}
```

SRT SOLUTIONS
We Develop Software Developers

CashRegister Takeaway(1)

- Notice again the natural syntax (not special-case!) for accessing and mutating a map's value.
- For-comprehension with multiple clauses (think query in SQL) with no control structure nesting
- Supports mid-stream assignment

SRT SOLUTIONS
We Develop Software Developers

CashRegister Takeaway(2)

- Match is far more powerful than simple switch!
  - Plays well with Option[T]
  - Implicit creation of vals bound to matched data
  - Can have multiple clauses per case
  - As is usual in Scala, can be used “expressionally”

SRT SOLUTIONS
We Develop Software Developers

CashRegister Takeaway(3)

- More match goodness!
  - No fall-throughs between cases
  - Some matching case is required (Scala wildcard _ can be used as “default”)
- No “return” needed at end of block

SRT SOLUTIONS
We Develop Software Developers

Migration to functional …

- Vals vs. vars
- Immutable types
- Use pattern matching
- Option type vs. null
- Use functional constructs (map, filter, sum, etc.)

SRT SOLUTIONS
We Develop Software Developers

Programmer Support

- Early adopter stage
- Book in PDF (Dec 2007)
- IDE: plugins in progress
- ScalaTest released (Jan 2008)

SRT SOLUTIONS
We Develop Software Developers

For more information …

- Scala book by Odersky, Spoon, Venners on Artima (PDF)
- scala-lang.org examples, etc.
- Scala event after Java One
- Java Posse podcast (Roundup ’08 recording: “What does Scala Need”)

SRT SOLUTIONS
We Develop Software Developers

Scala for Java Programmers

An Introduction

Dianne Marsh
Emerging Technologies for the Enterprise
dmarsh@srtsolutions.com
3/27/2008

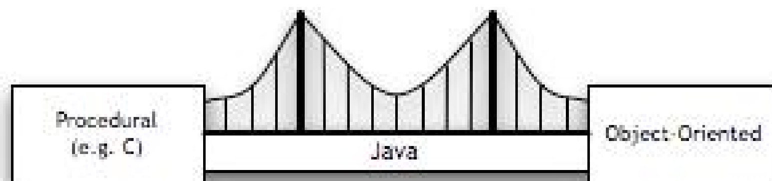
What is Scala?

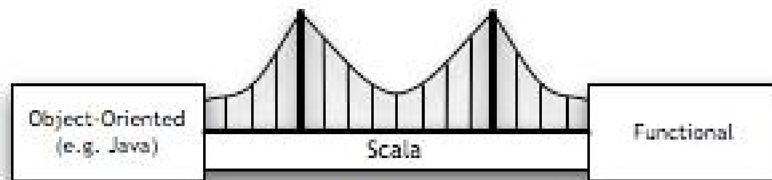
- Object-oriented/functional hybrid
- Designed by Martin Odersky
- Download at www.scala-lang.org
- Runs on JVM

SRT SOLUTIONS
We Develop Software Developers

What's different about Scala?

- Hybrid
- Interoperable with Java
- "Feel" of a scripting language





SRT SOLUTIONS
We Develop Software Developers

Small language

- No statics, operators, primitives
- Few control structures
- Building blocks
- Libraries

SRT SOLUTIONS
We Develop Software Developers

What is Functional Programming?

- Historically: mathematical, research
- Intent: improve everyday programming
  - Emphasis on functions (simplicity)
  - No side effects (pure, ease of debugging)
  - Good for concurrency
  - Minimized dependencies

SRT SOLUTIONS
We Develop Software Developers

Outline for Discussion

- Show a bit of Java
- Show a corresponding bit of Scala
- Compare/contrast
- Get your feedback

SRT SOLUTIONS
We Develop Software Developers

The problem

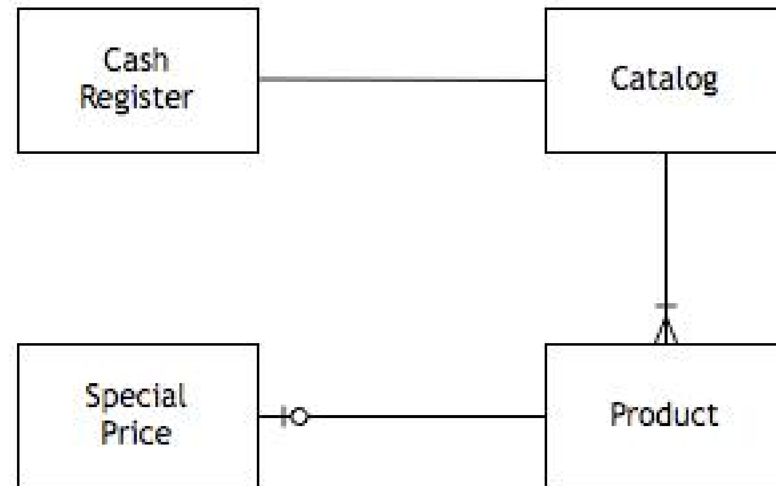
- Supermarket checkout,
  with special pricing on some units
- Exercise from codekata.pragprog.com

```
Item    Unit        Special
        Price       Price
--------------------------
   A      50        3 for 130
   B      30        2 for 45
   C      20
   D      15
```

SRT SOLUTIONS
We Develop Software Developers

Class Overview

- CashRegister
- Catalog
- Product
- SpecialPrice



SRT SOLUTIONS
We Develop Software Developers

“SpecialPrice”

```java
// ******************** Java ************************
public class SpecialPrice {
    private int quantity;
    private int price;
    public SpecialPrice(int quantity, int price) {
        this.quantity = quantity;
        this.price = price;
    }
    public int getQuantity() {
        return quantity;
    }
    public int getPrice() {
        return price;
    }
}
```

SRT SOLUTIONS
We Develop Software Developers

“SpecialPrice”

```
// ******************** Java ************************
public class SpecialPrice {
    private int quantity;
    private int price;
    public SpecialPrice(int quantity, int price) {
        this.quantity = quantity;
        this.price = price;
    }
    public int getQuantity() {
        return quantity;
    }
    public int getPrice() {
        return price;
    }
}


// ****************** Scala *************************
class SpecialPrice(val quantity: Int, val price: Int) {}
```

SRT SOLUTIONS
We Develop Software Developers

SpecialPrice Takeaway

- Scala does the obvious things by default
- Minimizes boilerplate
- Example: Classes can take parameters

SRT SOLUTIONS
We Develop Software Developers

“Product” class

```
// ******************* Java *************************
public class Product {
    private String productCode;
    private int unitPrice;
    private SpecialPrice specialPrice;
    public Product(String productCode, int unitPrice, SpecialPrice specialPrice) {
        this.productCode = productCode;
        this.unitPrice = unitPrice;
        this.specialPrice = specialPrice;
    }
    public String getProductCode() {return productCode;}
    public int getUnitPrice() {return unitPrice;}
    public SpecialPrice getSpecialPrice() {return specialPrice;}
}
```

SRT SOLUTIONS
We Develop Software Developers

“Product” class

```
// ******************* Java *************************
public class Product {
    private String productCode;
    private int unitPrice;
    private SpecialPrice specialPrice;
    public Product(String productCode, int unitPrice, SpecialPrice specialPrice) {
        this.productCode = productCode;
        this.unitPrice = unitPrice;
        this.specialPrice = specialPrice;
    }
    public String getProductCode() {return productCode;}
    public int getUnitPrice() {return unitPrice;}
    public SpecialPrice getSpecialPrice() {return specialPrice;}
}

// ****************** Scala  ************************
class Product(val productCode: String, val unitPrice: Int,
    val specialPrice: Option[SpecialPrice]) { }
```

SRT SOLUTIONS
We Develop Software Developers

Product Takeaway

- Simple
- Option[T] gives an explicit way to say “we may or may not have a T instance”
- (Hang on … more about that in a moment)

SRT SOLUTIONS
We Develop Software Developers

Catalog Overview

Catalog
-products : Map<String, Product>
+Catalog() +addProduct(Product) +getProduct(String) : Product +getcodes() : Collection<String>

SRT SOLUTIONS
We Develop Software Developers

Catalog in Detail – 1 of 2

```
// ******************** Java ************************
public class Catalog {
    private Map<String, Product> products;
    public Catalog() {
        products = new HashMap<String, Product>();
    }
    void addProduct(Product p) { … }
    public Product getProduct(String code) { … }
    public Collection<String> getCodes() { … }
}
// ******************** Scala ************************
class  Catalog  {
    val products = new HashMap[String, Product]()
    def addProduct(p: Product)  { … }
    def getProduct(code: String) { … }
    def elements { … }
    def codes  { …}
}
```

SRT SOLUTIONS
We Develop Software Developers

Catalog in Detail – 2 of 2

```
// ******************** Java ************************
public class Catalog {
    …
    void addProduct(Product p) {
        products.put(p.getProductCode(), p);
    }
    public Product getProduct(String code) {
        return products.get(code);
    }
    public Collection<String> getCodes() {
        return Collections.unmodifiableCollection(products.keySet());
    }
}
// ******************** Scala ************************
class  Catalog  {
    …
    def addProduct(p: Product) = products += (p.productCode -> p)
    def getProduct(code: String) =  products(code)
    def elements = products.elements
    def codes = products.keys
}
```

SRT SOLUTIONS
We Develop Software Developers

Catalog Takeaway (1)

- Clean, simple syntax.
  - NOT special-cased Scala syntax for maps.
  - operators are really methods
  - += is just a lib-defined method (on Map)
  - -> is just a method that creates a Tuple
    - p.productCode -> p is really just:

      (p.productCode).->(p) which returns a Tuple2

SRT SOLUTIONS
We Develop Software Developers

Catalog Takeaway (2)

- Clean, simple syntax.
  - less “boilerplate” (types not repeated, but inferred, including return types)
  - no need for braces around method bodies if definition is a single expression
  - note that “val products …” is implicitly invoked whenever an instance is created

SRT SOLUTIONS
We Develop Software Developers

Catalog Takeaway (3)

- Option[T]: optional value
- Preferred to Java “object reference or null” idiom
  - Option[T] gives you Some(T) and None
    - Some(T) means “I have this T”
    - None means “I don’t have anything”
  - Case classes can be used in explicit matches to drive logic (more in a moment)
  - FREE out-of-band value over any base type

SRT SOLUTIONS
We Develop Software Developers

CashRegister Overview

CashRegister

-ourCat : Catalog
-quantities : Map<String, Integer>

+CashRegister()
+scan(String)
+total() : int

SRT SOLUTIONS
We Develop Software Developers

Comparing Init Code

```
// *************** Java ***************
public class CashRegister {
    private Catalog ourCat;
    private Map<String, Integer> quantities;

    public CashRegister(Catalog catalog) {
        ourCat = catalog;
        quantities = new HashMap<String, Integer>();
        for (String code : catalog.getCodes()) {
            quantities.put(code, 0);
        }
    }

// ************** Scala ***************
class CashRegister(catalog: Catalog) {

    val quantities = new HashMap[String, Int]()
    for (code <- catalog codes) quantities += code -> 0

    …
}
```

SRT SOLUTIONS
We Develop Software Developers

Comparing Scan Code

```
// ************* Java *************************
public class CashRegister {
    …
    public void scan(String code) {
        quantities.put(code, quantities.get(code) + 1);
    }

    public int total() { … }
}

// ************* Scala *************************
class CashRegister(catalog : Catalog) {
    …
    def scan(code: String) =
        quantities(code) = quantities(code) + 1
    def total: Int = { … }
}
```

SRT SOLUTIONS
We Develop Software Developers

Java CashRegister (total)

```
public class CashRegister {
    …
    public int total() {
        int sum = 0;
        for (String code : quantities.keySet()) {
            int count = quantities.get(code);
            if (count > 0) {
                Product p = ourCat.getProduct(code);
                SpecialPrice special = p.getSpecialPrice();
                if (special != null) {
                    int specialCount = count / special.getQuantity();
                    int remainCount = count % special.getQuantity();
                    sum += (specialCount * special.getPrice() + remainCount
                            * p.getUnitPrice());
                } else {
                    sum += count * p.getUnitPrice();
                }
            }
        }
        return sum;
    }
}
```

SRT SOLUTIONS
We Develop Software Developers

Scala CashRegister (total)

```
class CashRegister(catalog : Catalog) {
    …
    def total : Int = {
        var sum = 0
        for {
            (code, count) <- quantities.elements
            if count > 0
        } {
            val p = catalog getProduct code
            sum += (p.specialPrice match {
                case Some(s) =>
                    val specialCount = count / s.quantity
                    val remainCount = count % s.quantity
                    specialCount * s.price + remainCount * p.unitPrice
                case None => count * p.unitPrice
            })
        }
        sum
    }
}
```

SRT SOLUTIONS
We Develop Software Developers

Testing the Scala Version

```
object CashRegisterTest extends Application {

  val cat = new Catalog()
  cat.addProduct(new Product("A", 50, Some(new SpecialPrice(3, 130))))
  cat.addProduct(new Product("B", 30, Some(new SpecialPrice(2, 45))))
  cat.addProduct(new Product("C", 20, None))
  cat.addProduct(new Product("D", 15, None))

  val reg = new CashRegister(cat)

  reg scan "A"
  reg scan "B"
  reg scan "A"
  reg scan "B"
  reg scan "B"
  reg scan "C"

}
```

SRT SOLUTIONS
We Develop Software Developers

CashRegister Takeaway(1)

- Notice again the natural syntax (not special-case!) for accessing and mutating a map's value.
- For-comprehension with multiple clauses (think query in SQL) with no control structure nesting
- Supports mid-stream assignment

SRT SOLUTIONS
We Develop Software Developers

CashRegister Takeaway(2)

- Match is far more powerful than simple switch!
  - Plays well with Option[T]
  - Implicit creation of vals bound to matched data
  - Can have multiple clauses per case
  - As is usual in Scala, can be used “expressionally”

SRT SOLUTIONS
We Develop Software Developers

CashRegister Takeaway(3)

- More match goodness!
  - No fall-throughs between cases
  - Some matching case is required (Scala wildcard _ can be used as “default”)
- No “return” needed at end of block

SRT SOLUTIONS
We Develop Software Developers

Migration to functional …

- Vals vs. vars
- Immutable types
- Use pattern matching
- Option type vs. null
- Use functional constructs (map, filter, sum, etc.)

SRT SOLUTIONS
We Develop Software Developers

Programmer Support

- Early adopter stage
- Book in PDF (Dec 2007)
- IDE: plugins in progress
- ScalaTest released (Jan 2008)

SRT SOLUTIONS
We Develop Software Developers

For more information …

- Scala book by Odersky, Spoon, Venners on Artima (PDF)
- scala-lang.org examples, etc.
- Scala event after Java One
- Java Posse podcast (Roundup ’08 recording: “What does Scala Need”)

SRT SOLUTIONS
We Develop Software Developers