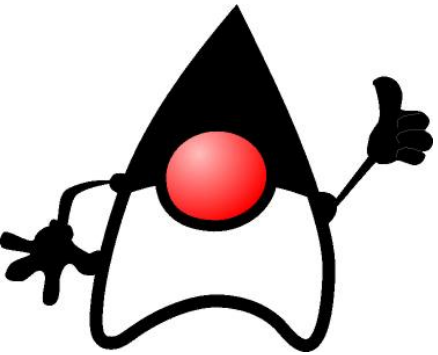
A Brief Introduction to Scala

Steven Reynolds

www.slreynolds.net

www.int.com



Dedicated to Dr. Augie Caponecchi

Scala

Functional programming on the Java Platform: immutability, functions are first class

You can choose: immutable or mutable

Why Does Functional Programming Matter?

Why Does Functional Programming Matter?

Map-Reduce

Why Does Functional Programming Matter?

Map-Reduce

Google

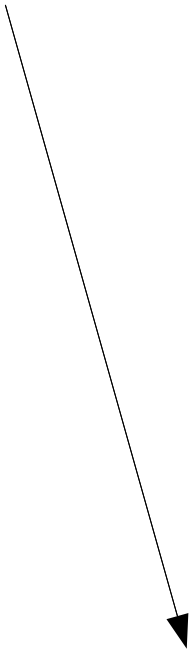
Why Does Functional Programming Matter?

Map-Reduce

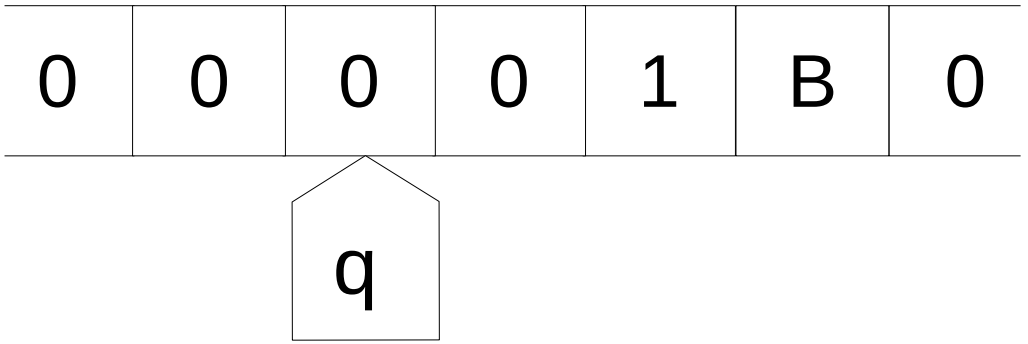
Google

$150 Billion

Functional Programming



$150 Billion


0
0
0
0
1
B
0
q

?

Scala design goals

- Combine functional with object oriented
- Pratical
- Can call from Scala to Java, and from Java to Scala
- Powerful language
- Powerful static type system that's easy to use
- Works on .NET (*not used much recently)*

Functional Programming

Functional Programming

- Function is a first class citizen
- Compact syntax to declare function literal
- Pass functions as arguments and as return values
- Functions are closures too
- Immutability: values don't change

History

- 1932/1936 – Alonzo Church – Lambda Calculus

Lambda Notation	Conventional Math Notation
λx.x + 2	f(x) = x + 2
(λx.x + 2) 3	f(3)

http://en.wikipedia.org/wiki/Lambda_calculus

Alonzo Church

Church-Turing Thesis

Lambda Calculus ◄——— equivalent ———► **Turing Machines**

functions — tape machine

Lambda Calculus is equivalent to any means of computation

Functional Programming

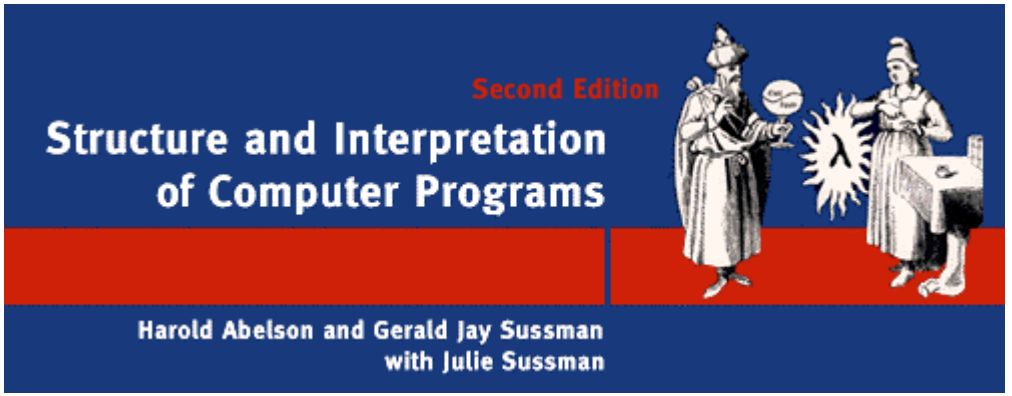
Advantages

- Powerful. Equivalent to any computational mechnism.
- What was once true will ever be true - like math
- Reasoning and testing are simpler
- Nice for concurrency and distributed systems

Disadvantages

- Modular programming is harder/impossible
- Sometimes performance issues

These issues explored thoroughly in



http://mitpress.mit.edu/sicp/

Scala Approach

- Functions are first class citizens
- Collections have immutable and mutable versions
- `val` - immutable
- `var` - mutable
- Steer to immuable, but make mutable equally easy

it's your choice

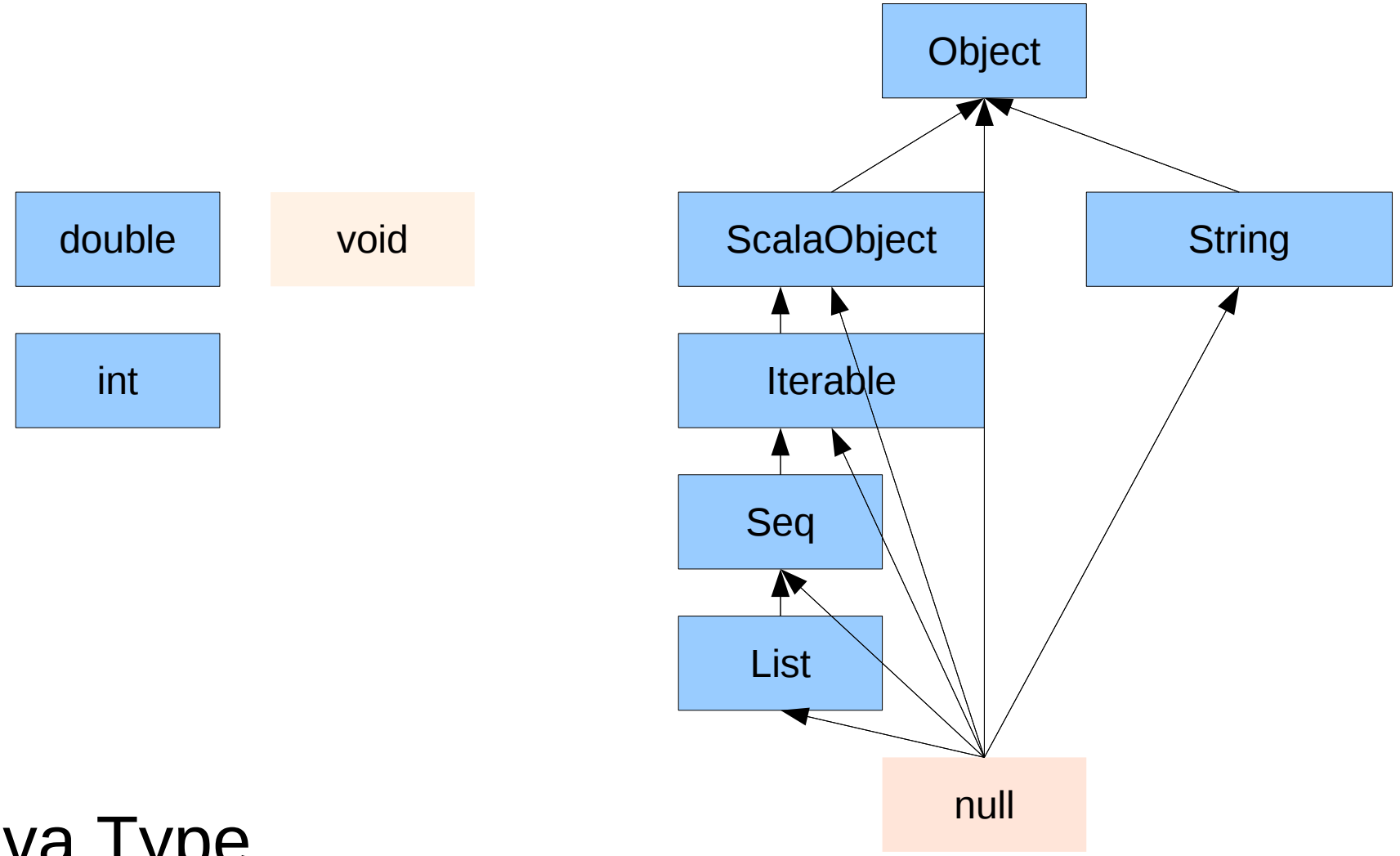
Scala Type System

Scala Type system

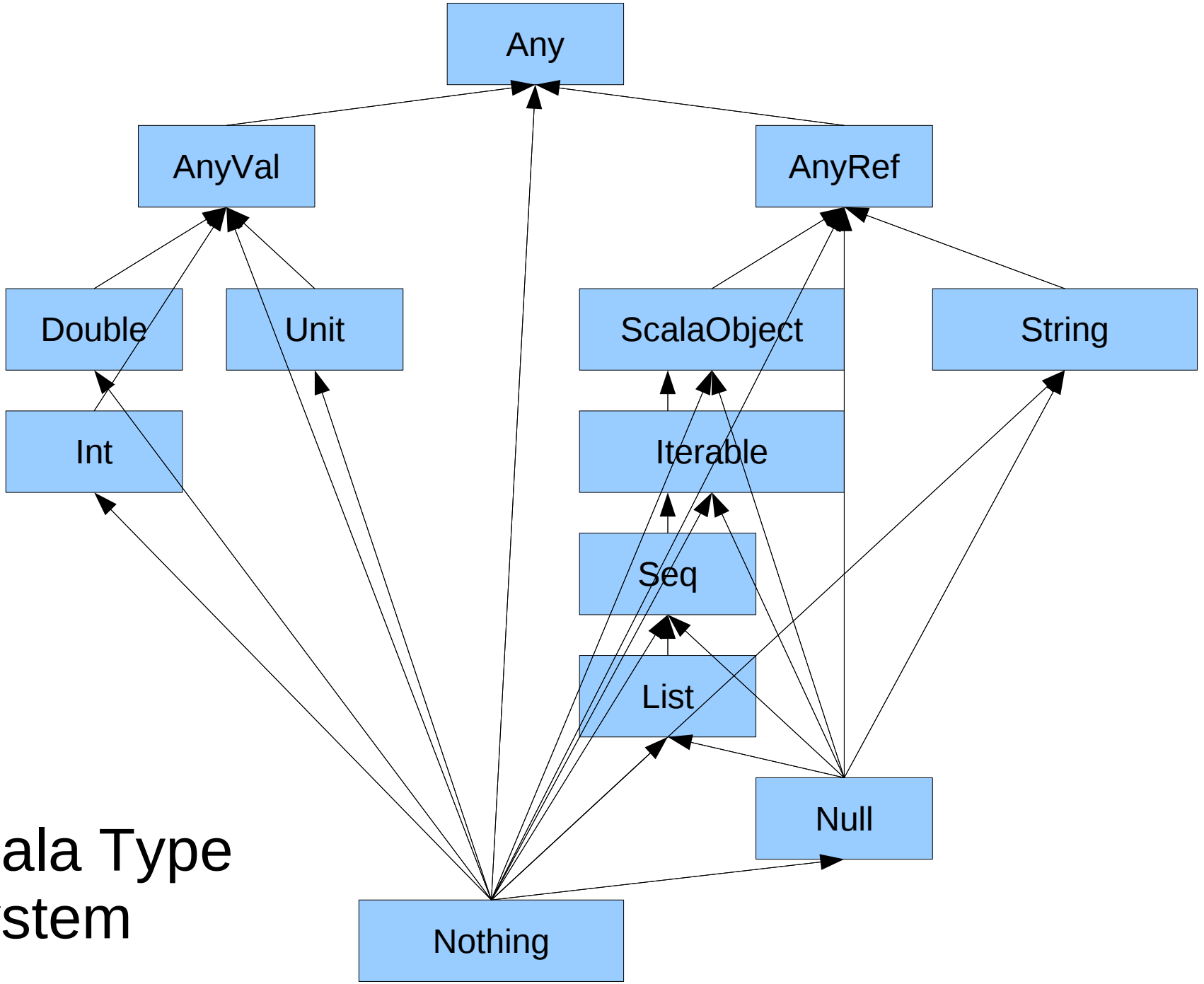
- Every statement has a value
- Type inferencing
  - Mostly, the compiler figures out the type
  - Inferencing is limited in scope
- Type system is a lattice
- Generics with *all* types supported
- Can use Option[A] for a value that might exist or not

Option

- Represents value that might be present or not
- Explicit type means compiler helps you avoid nullptr exceptions
- For example, map returns an Option



Java Type System



Scala Type System

a lattice is

a partially ordered set in which any two elements have

- a unique least upper bound (join) and
- a unique greatest lower bound (meet).

Source: Wikipedia

Great. What can I do with a it?

```
def error(message: String): Nothing =
  throw new RuntimeException(message)
```

You can use error anywhere

```
def roots(a: Double, b: Double, c:
Double):(Double,Double) {

  dscr = b*b – 4*a*c
  if (dscr < 0 )
    error(“no complex math yet”)
  else
    ((-b – sqrt(dscr))/2*a,
     (-b + sqrt(dscr))/2*a)
}
```

B
K
C

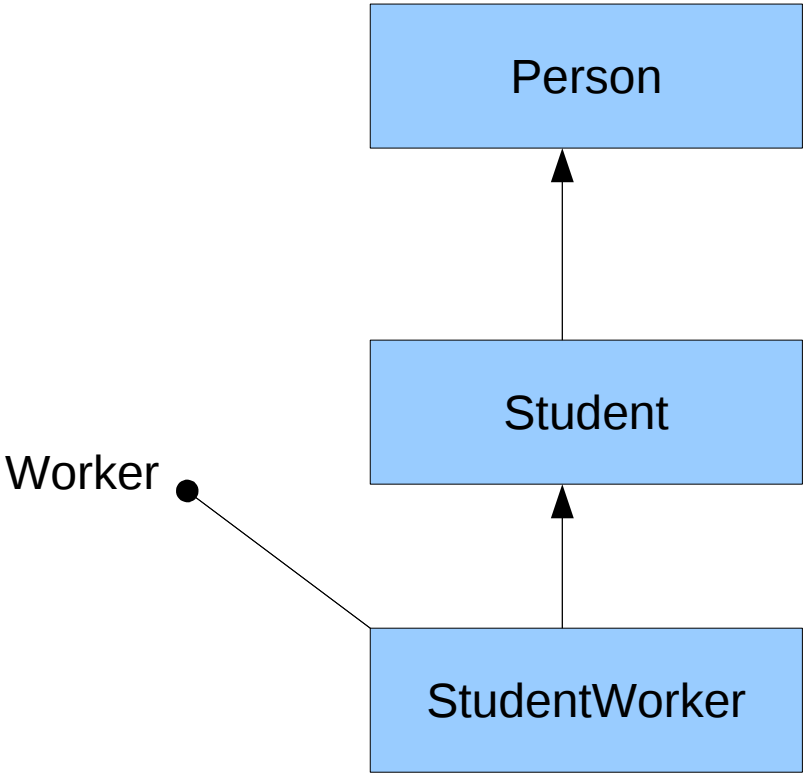
Inheritance in Scala

Inheritance

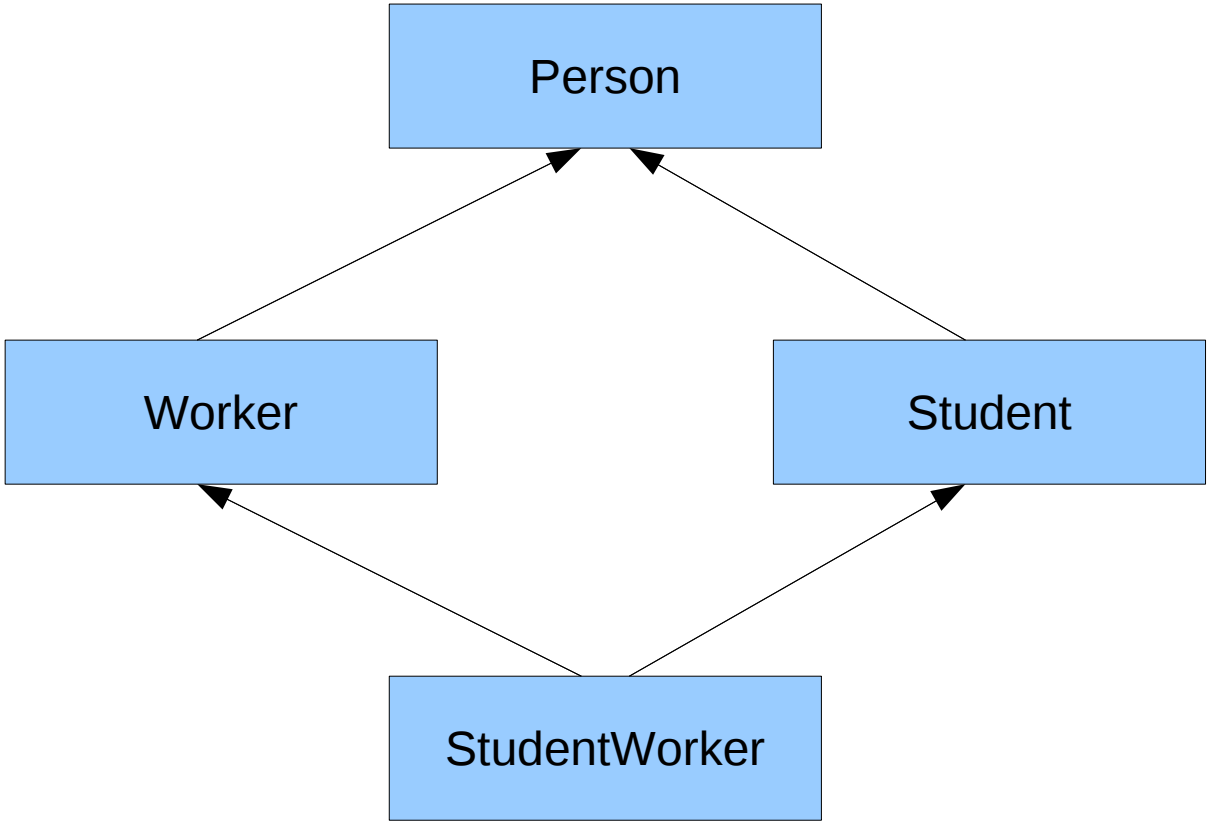
- C++ has multiple inheritance
- Java has single inheritance
- Scala has single inheritance with mixins (traits)

Origin of mixins?

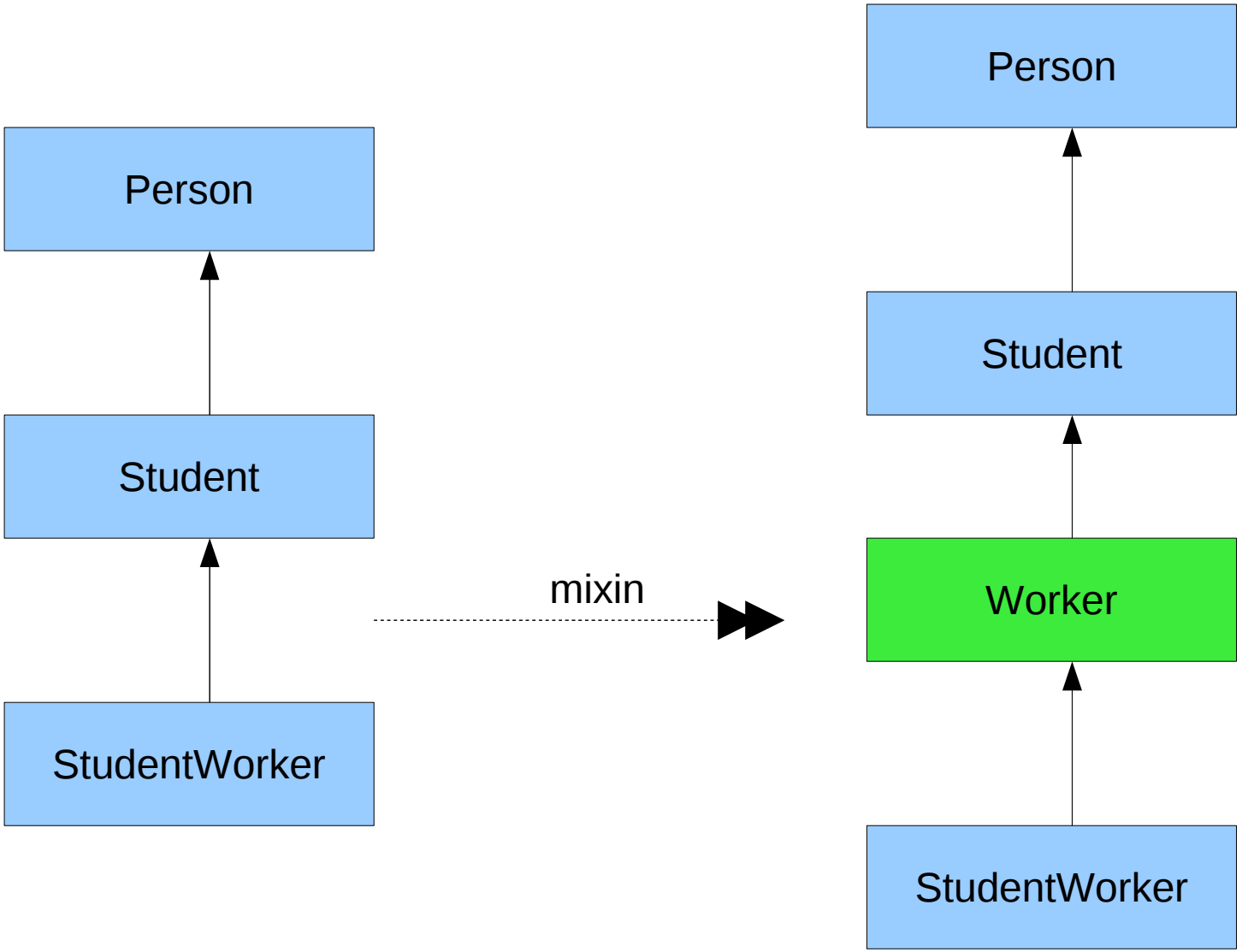
- Lisp flavor's
- Steve's ice cream



Single Inheritance



Multiple Inheritance


Person
Student
StudentWorker
mixin
Person
Student
Worker
StudentWorker

Mixin

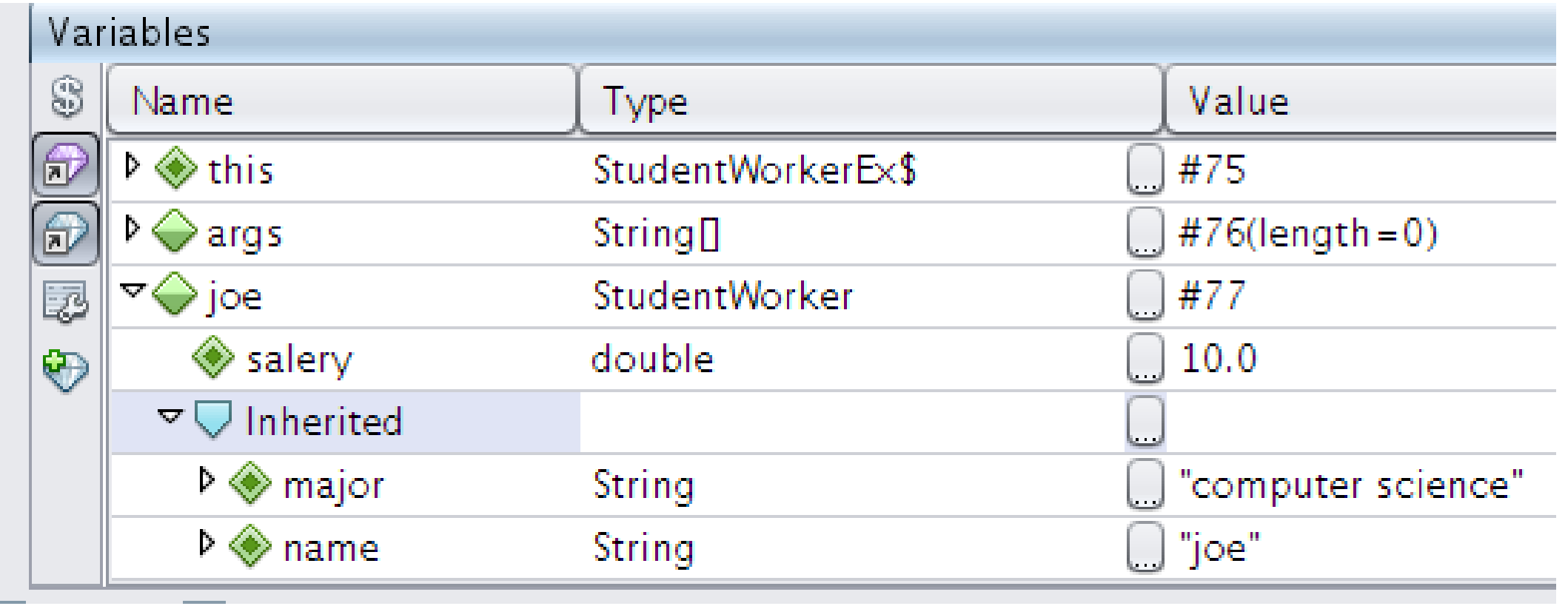
Mixins

- defines a delta relative to the hierarchy that it is inserted
- does define behavior, and state
- cannot be understood until you know where its inserted

Class

- is completely understood when you write the class
- is absolute

StudentWorker



Worker.class

```
/* Worker - Decompiled by JODE
 * Visit http://jode.sourceforge.net/ */

import scala.ScalaObject;

public interface Worker extends ScalaObject {
  public double getSalery();

  public void salery_$eq(double d);

  public double salery();
}
```

StudentWorker.class

```
import scala.ScalaObject;
import scala.runtime.BoxesRunTime;

public class StudentWorker extends Student
implements Worker, ScalaObject
{
  private double salery;

  public StudentWorker(String n, String m,
                       double s) {
    super(n, m);
    Worker$class.$init$(this);
    this.salery_$eq(s);
  }
```

```
  public String to_string() {
    return new
      scala.StringBuilder().append((Object)
      super.to_string())
      .append
      ((Object) " ").append

(BoxesRunTime.boxToDouble(this.getSalery())).
toString();
    }

  public double getSalery() {
    return Worker$class.getSalery(this);
  }

  public void salery_$eq(double x$1) {
    salery = x$1;
  }
```

```
  public double salery() {
    return salery;
  }
}
```

issues with multiple inheritance

- diamond pattern
- state copied?
- which method gets invoked?
- complex and/or fragile.
- possible solutions to fragility/complexity severely limit reuse

benefit of single inheritance

- simple to understand

issues with single inheritance

- interface behavior must be implemented over and over and over and ...
- number of classes/interfaces explode

Evaluation Order

Delayed Evaluation

By-name parameters

```
def byNameAssert(predicate: => Boolean) =
  if (assertionsEnabled && !predicate)
    throw new AssertionError

byNameAssert(5 > 3)
```

Lazy Initialization

- `vals` can be initialized lazily

```
lazy val x = { println(“initializing x”);
“done” }
```

- initialized at most once, when they are needed

Matching

Matching

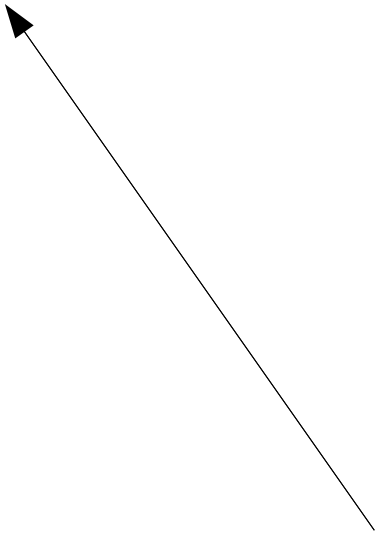
```
def feed(dog:Dog) = {
  dog match {
    case Mixed(n) =>
         print("fed " + n + " chicken\n")

    case AmericanEskimo("Samantha") =>
         print("fed Samantha peas and
carrots, and then she stole a loaf of
bread\n")

    case _ =>
         print("fed " + dog.toString + "
some dog food\n")
  }
}
```

Swing Library

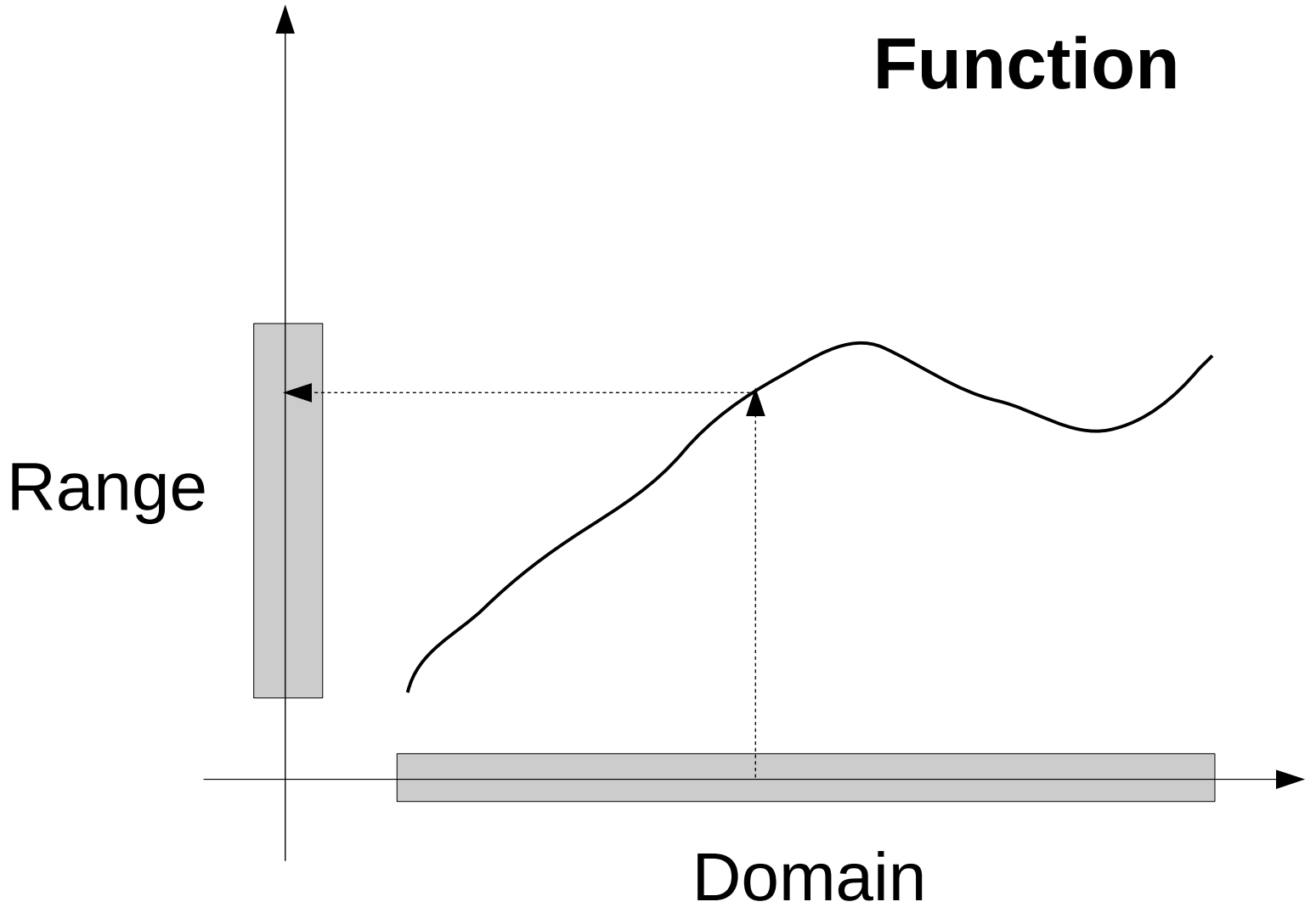
Swing

- Scala Swing library is beta
  -  debugging it with NetBeans
- Application inherits from `SimpleGUIApplication`
- Wraps swing components as you might expect
- Panels and layout managers are tied together
  - add method is now type-safe
- Install event handlers by adding them to `reactions` property

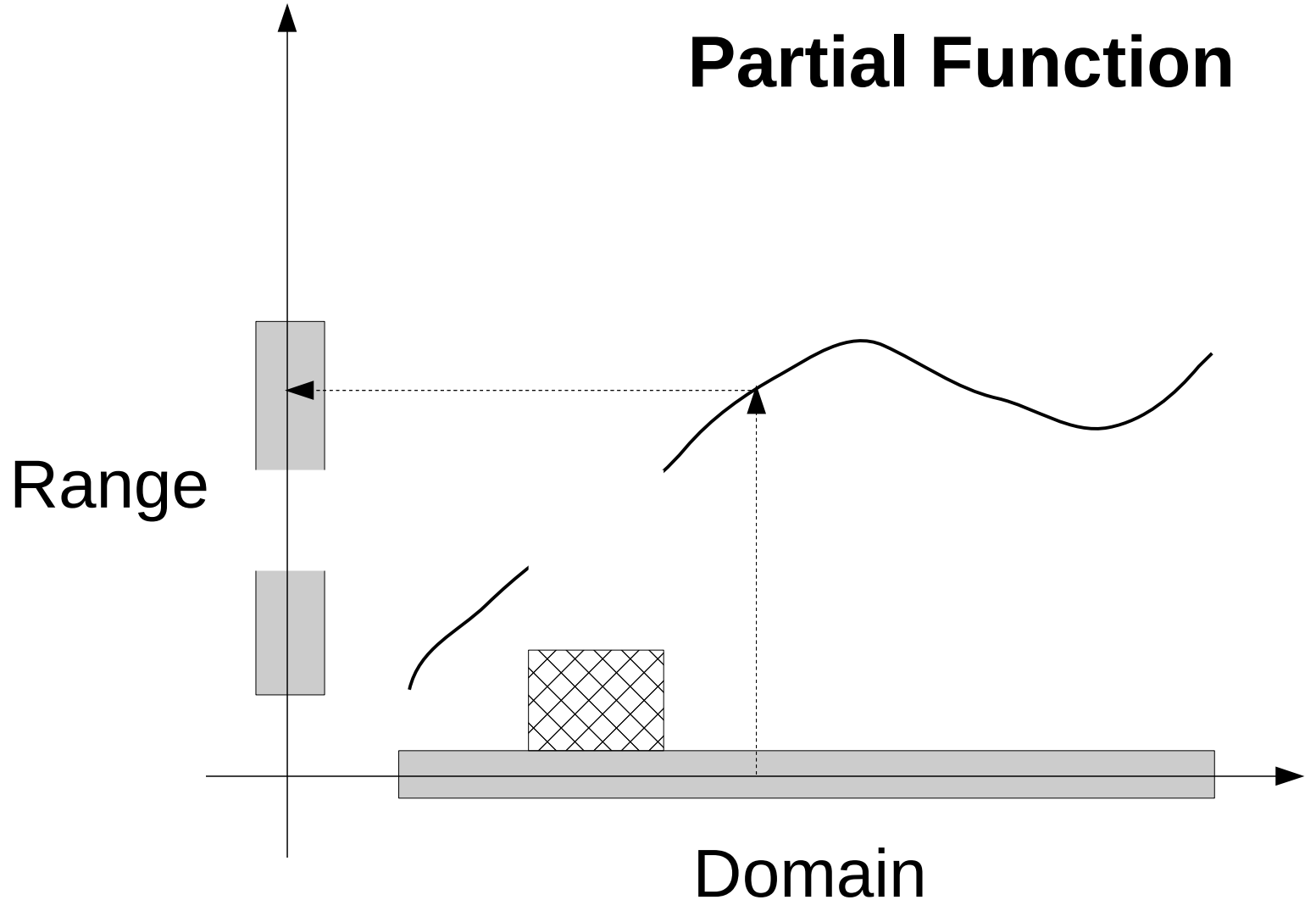
```
var nClicks = 0
reactions += {
  case ButtonClicked(b) =>
    nClicks+= 1
    label.text = “Num clicks: “ +
                                  nClicks
}
```



PartialFunction


Function
Range
Domain

Partial Function



Not defined on part of the domain

Function Examples

Function Example

Person Weight
people $\mapsto$ weight

Partial Function Example

2008 US Income tax due
people $\mapsto$ US dollars due

Scala `PartialFunction`

- A `Function` has an `apply()` method
- `PartialFunction` is a function with a `isDefinedAt()` method auto-generated

```
val second:
PartialFunction[List[Int],Int] = {
  case x:: y :: _ => y
}
```

```
new PartialFucntion[List[Int], Int] {
  def apply(xs: List[Int}) = xs match {
    case x :: y :: _ => y
  }
  def isDefinedAt(xs: List[Int]) =
  xs match {
    case x :: y :: _ +> true
    case _ => false
  }
}
```

```
reactions += {
   case SelectionChanged(_)  => updateEstimate()
}
```

```
listenTo(routeCombo.selection,
                                    departCombo.selection)
```

Scala Type System - Reprise

Generics

- More powerful than Java
- Writer of generic class annotates if the type argument is covariant, contravariant, or invariant
  - More precise control over types
  - Writer of generic class worries about it, not the user
  - Better type safety
  - More powerful
- A good story for another day...

Abstract Type Members

```
class Food
abstract class Animal {
    type SuitableFood <: Food
    def eat(food: SuitableFood)
}
```

abstract types – good for family polymorphism

Polymorphism

	type	value
field	abstract type member	normal object field
paramter	generic type	paramter to constructor

Other Type System Features

- self type annotation (rocket operator in traits)
- nested types like inner classes in Java Outer#Inner
- language support for singletons

Scala

Functional programming on the Java Platform: immutability, functions are first class

You can choose: immutable or mutable