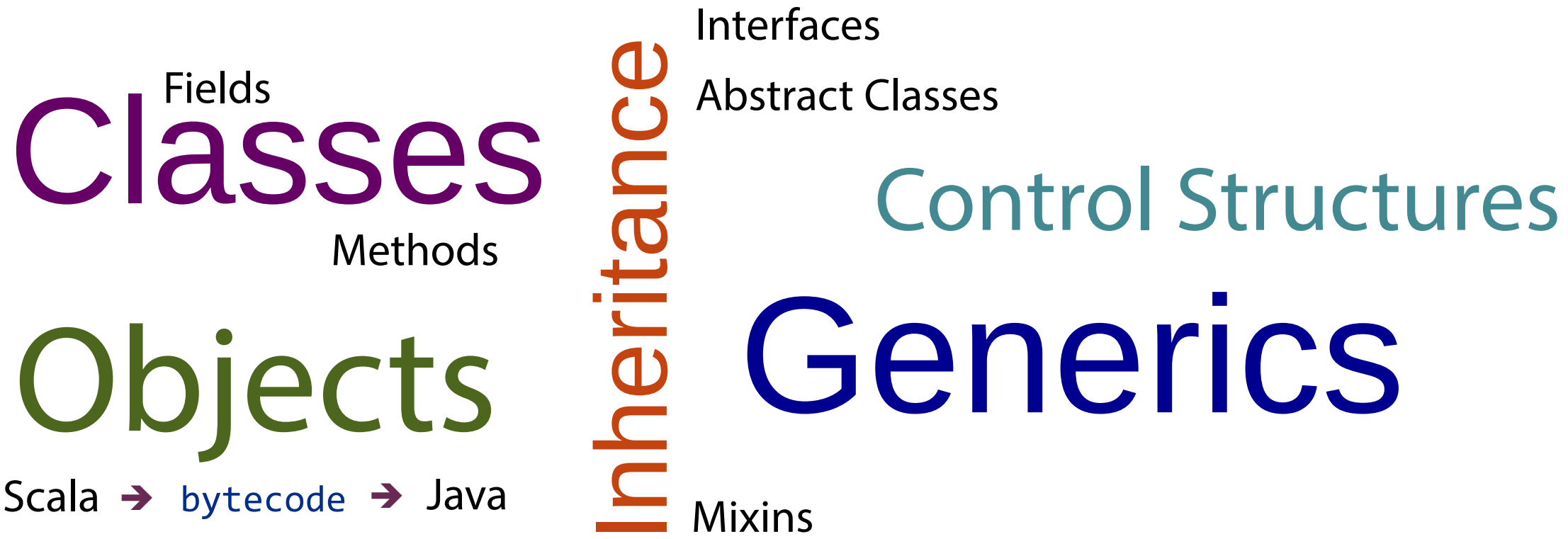
Key Syntactical Differences

Toby Weston
http://baddotrobot.com
@jamanifin

pluralsight
hardcore dev and IT training

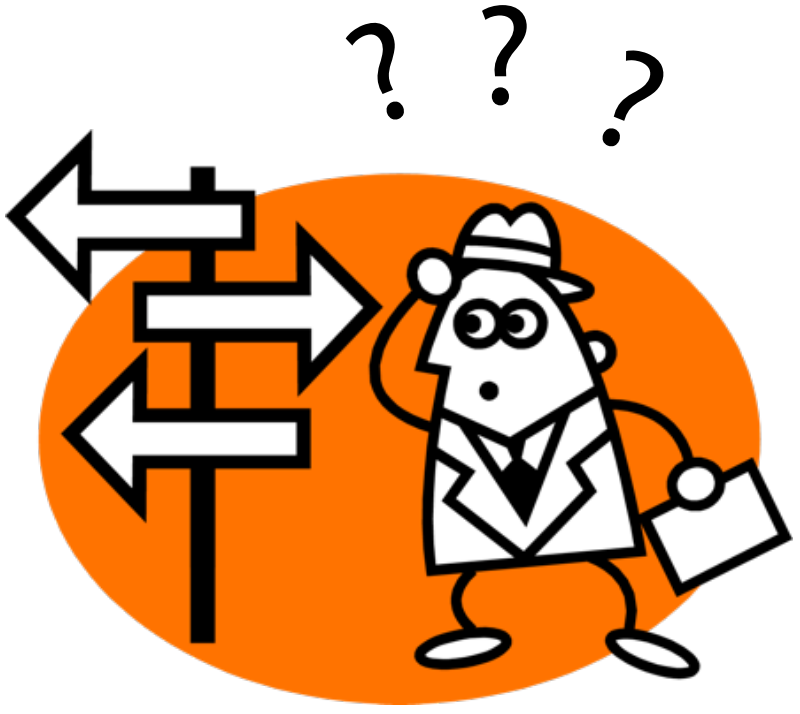
Key Syntactical Differences

Module Introduction


Fields
Classes
Methods
Objects
Scala ➔ bytecode ➔ Java
Inheritance
Interfaces
Abstract Classes
Mixins
Control Structures
Generics

very

Scala is Flexible

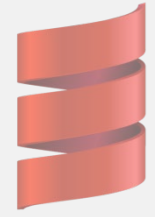


Scala Favors Immutability

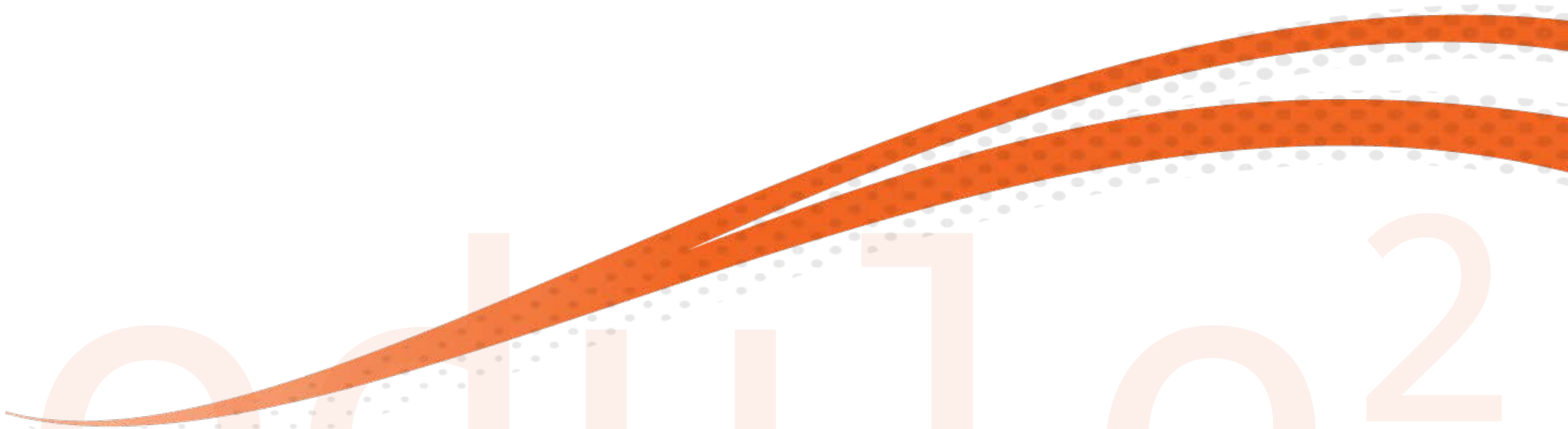
and being declarative

```java
for (int count = 0; count < 100; count++) {
    System.out.println(count);
}
```

```
(0 to 100).foreach(println(_))
```



Let’s Get Started…



Key Syntactical Differences

Classes and Objects

- Creating classes
- Defining fields
- Overriding Getter and Setters

New Topic

Creating Classes

01-classes-java/scala.demo

DEMO

```
class Customer(val name: String, val address: String) {
  // body
}
```

New Term

Primary constructor

```
class Customer(val name: String, val address: String) {
  // body
}
```

02-cfr-customer-decompile.demo

DEMO

http://www.benf.org/other/cfr/

New Topic

Defining Fields

New Topic

Overriding Setters and Getters

New Topic

Summary

```
class Customer(val name: String, val address: String) {
  // body
}
```

```
public class Customer {
    private String name;
    private String address;

    public Customer(String name, String address) {
        this.name = name;
        this.address = address;
    }
}
```

val

```
class Example(val x: Int)

public def x(): Int

public def x_=(y: Int)

// generated constructor
public Example(int x)
```

var

```
class Example(var x: Int)

public def x(): Int

public def x_=(y: Int)

// generated constructor
public Example(int x)
```

none

```
class Example(x: Int)
```

~~`public def x(): Int`~~

~~`public def x_=(y: Int)`~~

```
// generated constructor
public Example(int x) {
    // x only visible here
}
```

private

val

```
class Example(private val x: Int)

private def x(): Int

public def x_=(y: Int)

// generated constructor
public Example(int x)
```

private var

```
class Example(private var x: Int)

private def x(): Int

private def x_=(y: Int)

// generated constructor
public Example(int x)
```

Creating Fields Within a Class Definition

class Foo(? x)	val x	var x	x	private val x	private var x
Getter created x()	✓ public	✓ public	✗	✓ private	✓ private
Setter created x_=(y)	✗	✓ public	✗	✗	✓ private
Generated constructor includes x	✓	✓	✗	✓	✓

Override generated methods

Override generated methods

1. **Rename**

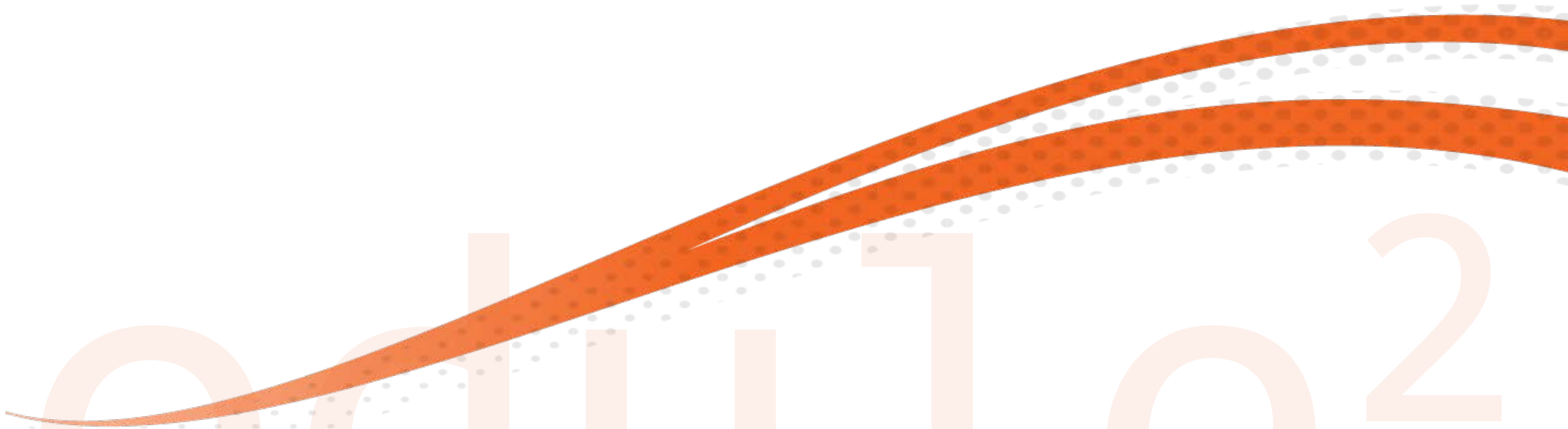
Override generated methods

1. Rename
2. **Mark as private**

Override generated methods

1. Rename
2. Mark as private
3. **Recreate setter and getter**

Next up…



Key Syntactical Differences

Classes and Objects

- Fields within classes
- Additional constructors
- Singleton objects
- Companion objects

New Topic

Fields Within Classes

Creating Fields Within a Primary Constructor

class Foo()	val x	var x	private val x	private var x
Getter created x()	✓ public	✓ public	✓ private	✓ private
Setter created x_=(y)	✗	✓ public	✗	✓ private

New Topic

Additional Constructors

New Term

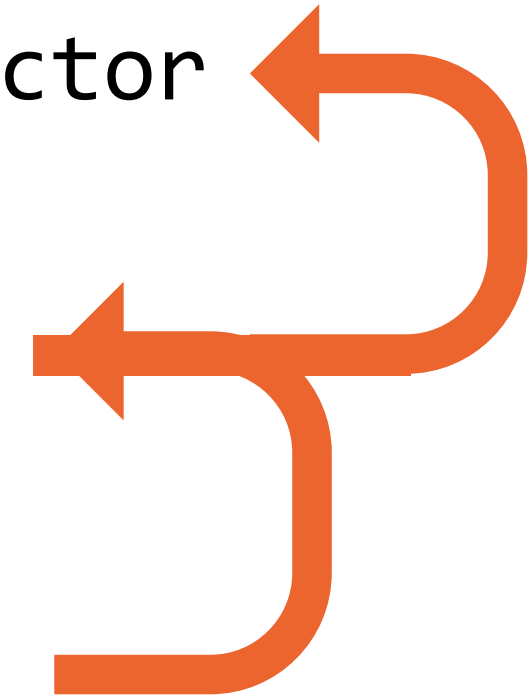
Auxiliary constructors

```
def this(…)
```

```
default constructor
```

```
def this(…)
```

```
def this(…)
```



New Topic

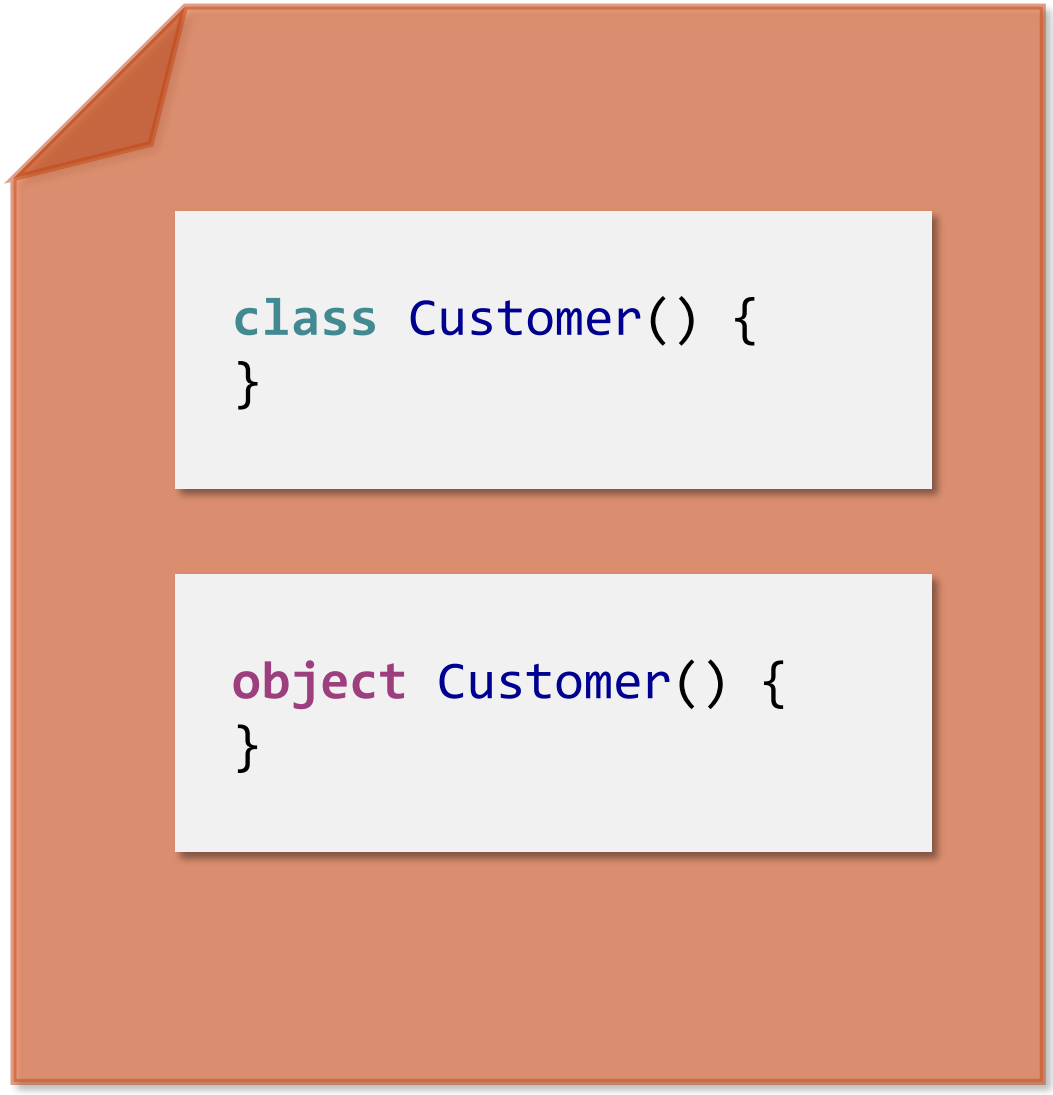
Singleton Objects

ShoppingCart

```
Logger.getLogger(“example”).log(INFO, “Everything is fine.”);
```

New Topic

Companion Objects


class Customer() {
}
object Customer() {
}


static

Why Use Companion Objects?

- **Functions vs. methods**
- **Factory methods**

New Term

The "apply" method

New Term

Case classes

Next up…

Key Syntactical Differences

Inheritance

- Sub-type Inheritance
- Interfaces
- Abstract Classes
- “Mixin” Behaviour

New Topic

Sub-type Inheritance

extends

final

07

DEMO

New Topic

Anonymous Sub-Classes

New Topic

Interfaces

Interfaces ≈ Traits

08

DEMO

Classes : Abstract Classes : Traits

New Topic

Methods on Traits

10/11

DEMO

New Topic

Convert Anonymous Types to Lambdas

11

DEMO

New Topic

Concrete Fields on Traits

12

DEMO

New Topic

Abstract Fields on Traits

12

DEMO

Summary

Trait

- Default Implementations
- Fields
- Reuse State
- Implement *n* Traits
- “Mixins”

Interface

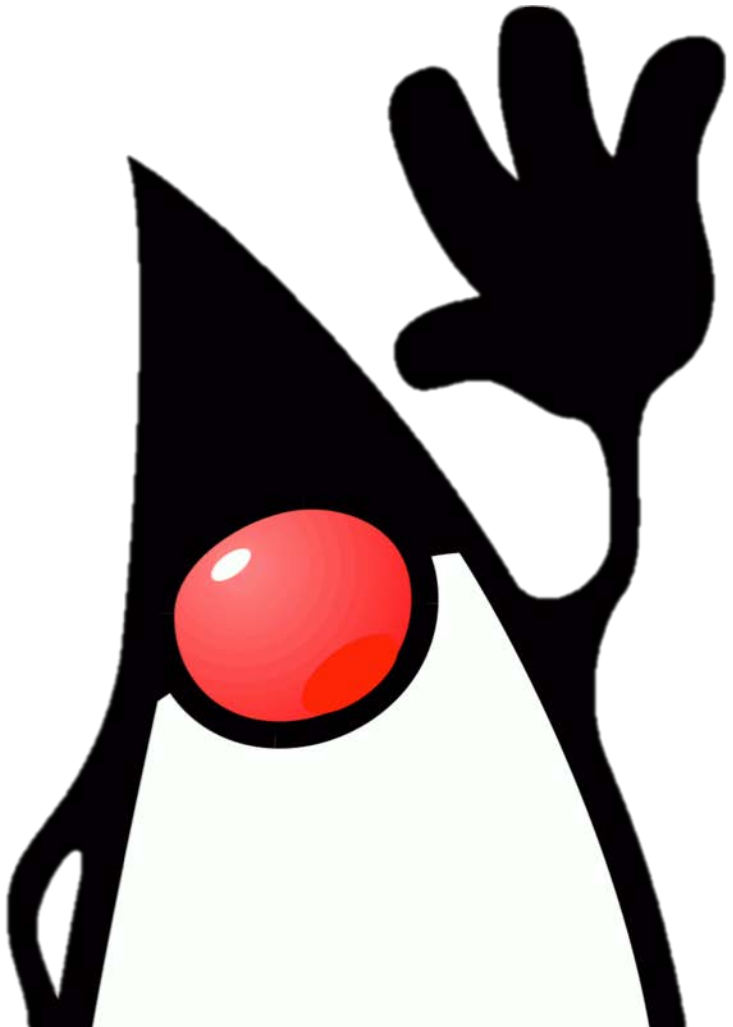
- Virtual Extension Methods in Java 8
- No Fields
- No State
- Implement *n* Interfaces
- “Mixins” (sort of)

New Topic

Abstract Classes

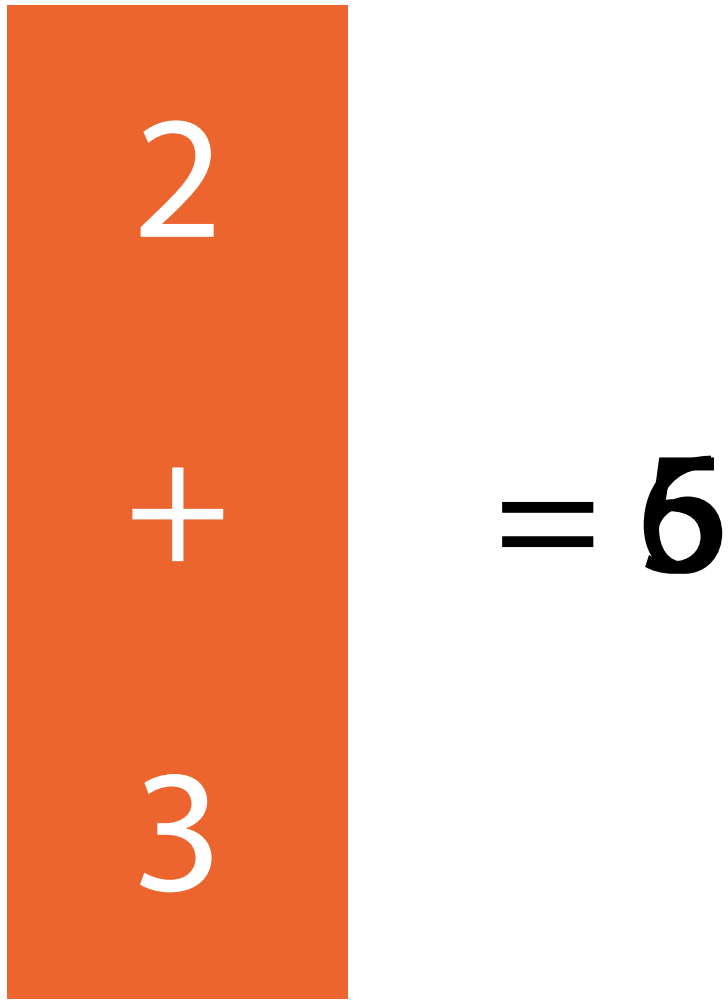
New Topic

Polymorphism and Traits

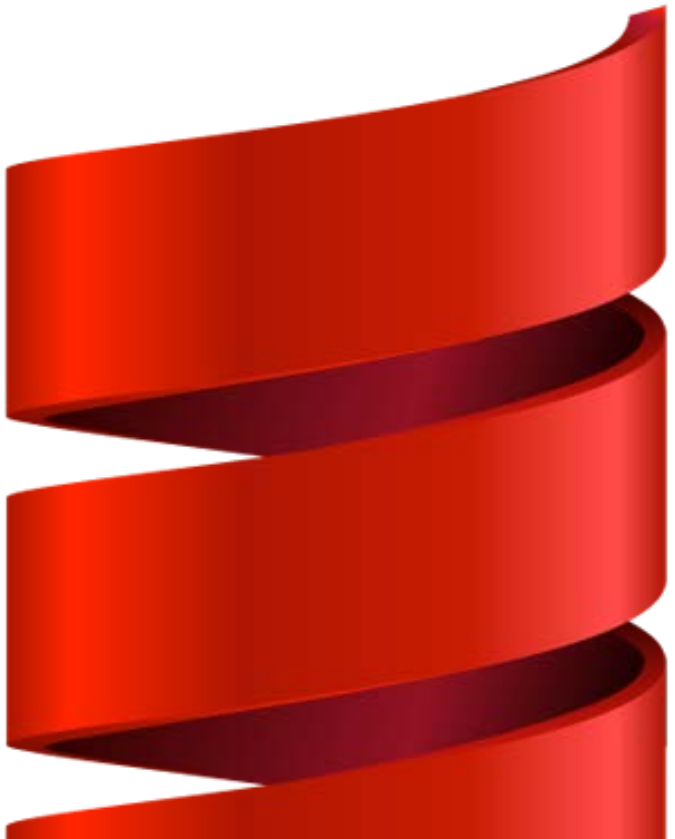


- **Sub-type Class**
  - For behaviour and state
- **Sub-type Interfaces**
  - For behaviour

Substitutability


2
+
3
= 5

Inclusion Polymorphism



- **Traits**
  - Without default implementations
  - With implementations
- **Abstract Classes**
  - With and without fields
- **Classes Extension**
- **Structural Types**

New Topic

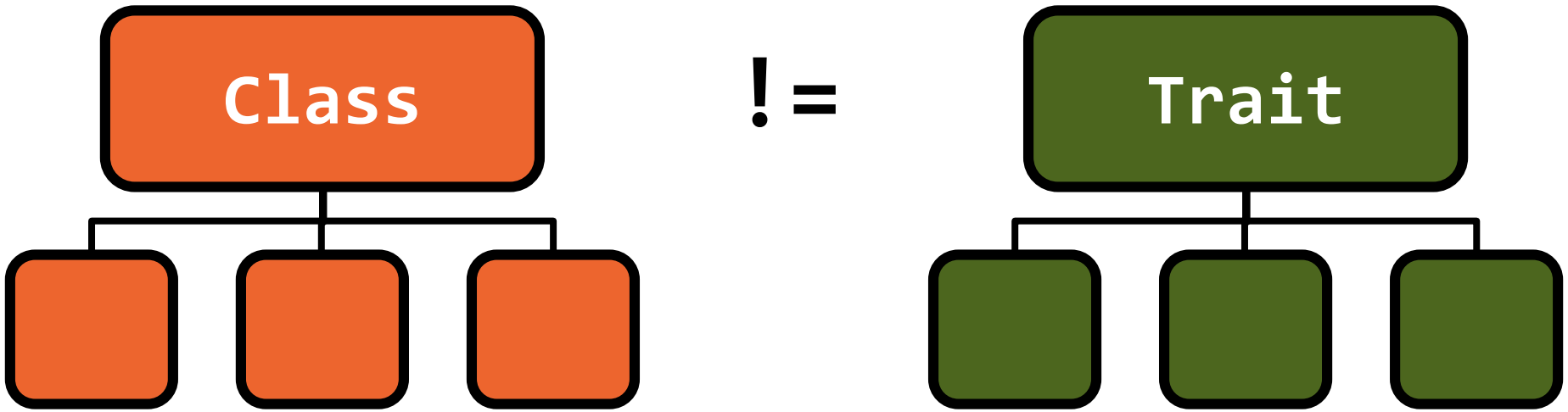
Traits vs. Abstract Classes

Traits

- No constructor arguments
- Provide for multiple-inheritance
- Classes can mix in *n* traits

Abstract Classes

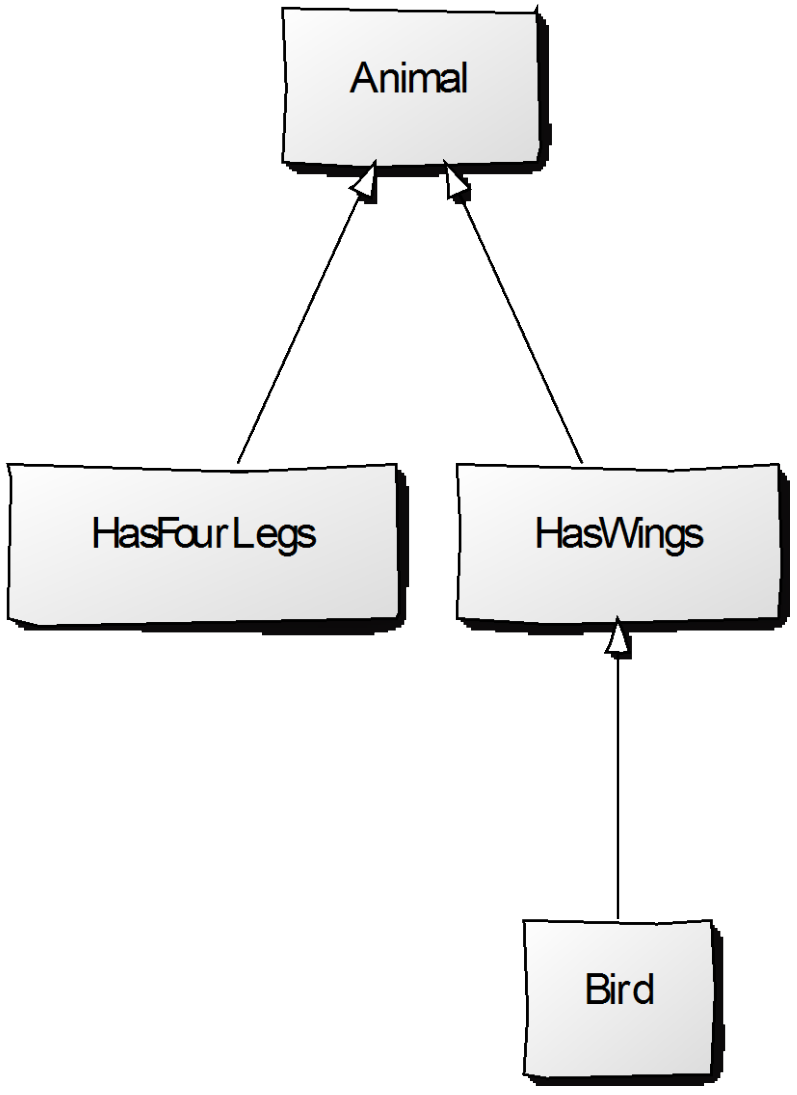
- Constructor arguments
- Cause problems for multiple-inheritance
- Classes can extend only 1 super class


Class
!=
Trait

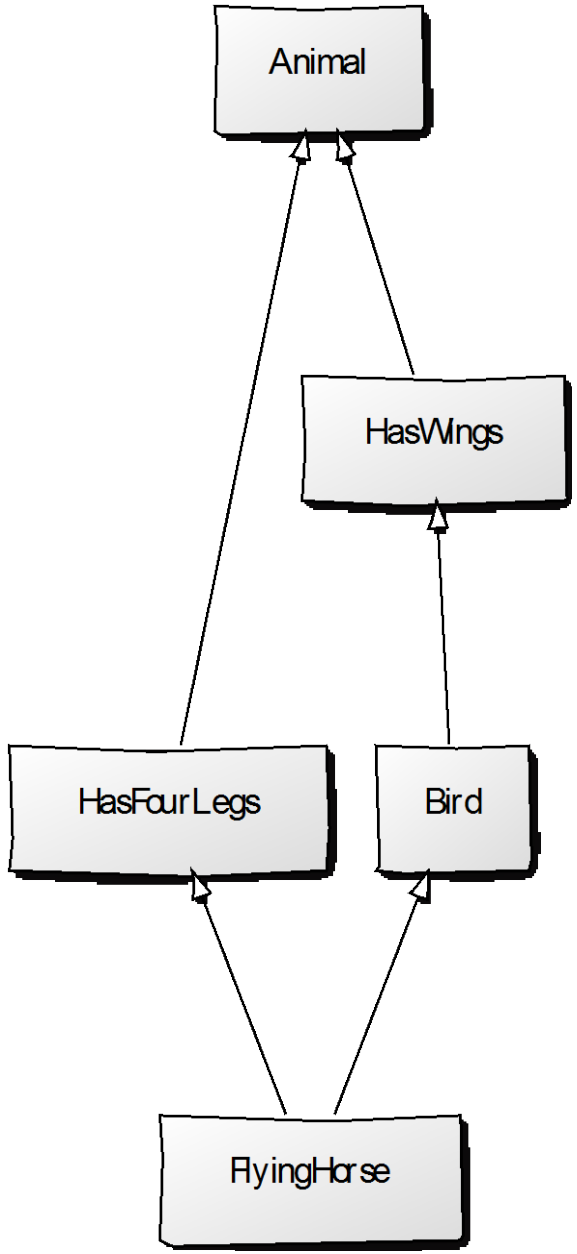
New Term

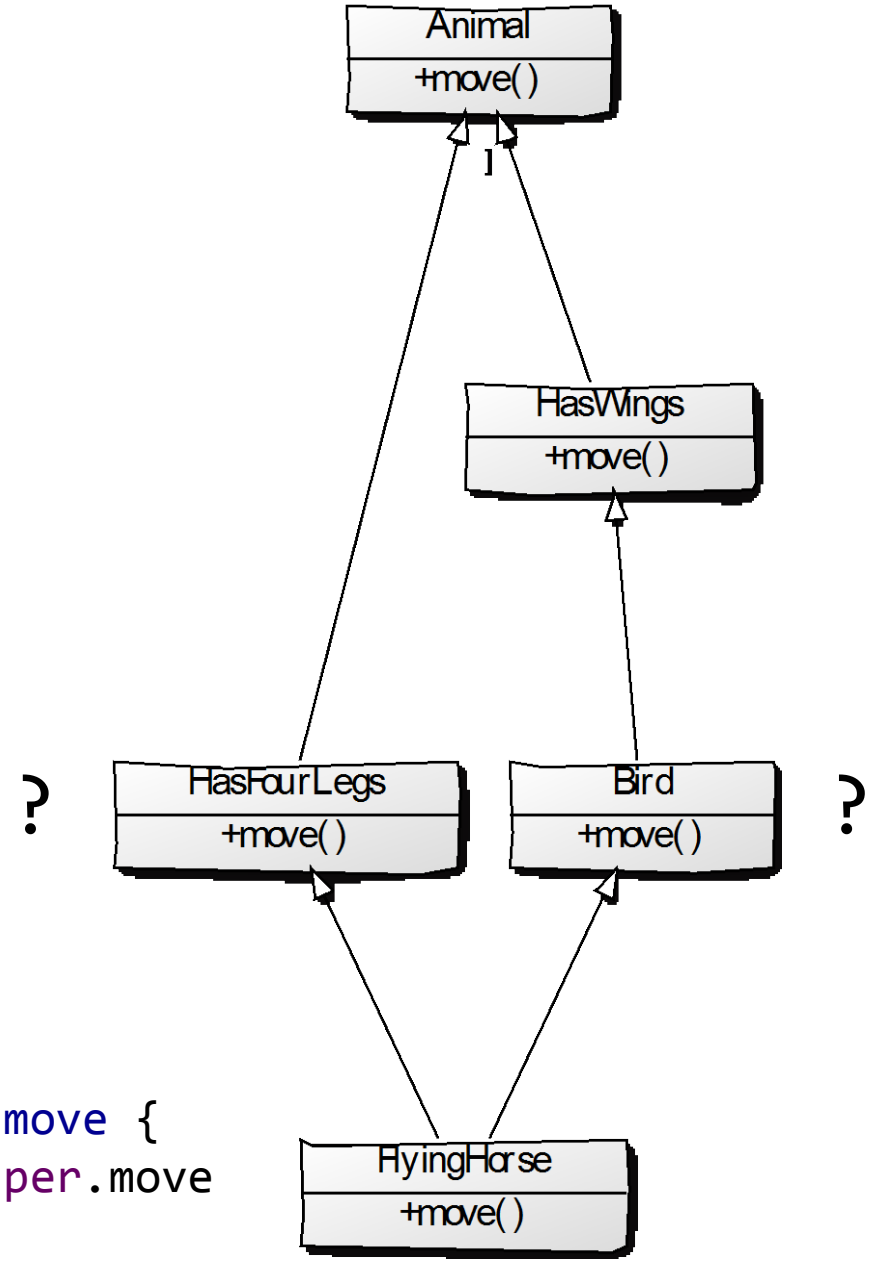
Linearization

```
class Animal

trait HasWings

trait Bird extends HasWings

trait HasFourLegs extends Animal
```

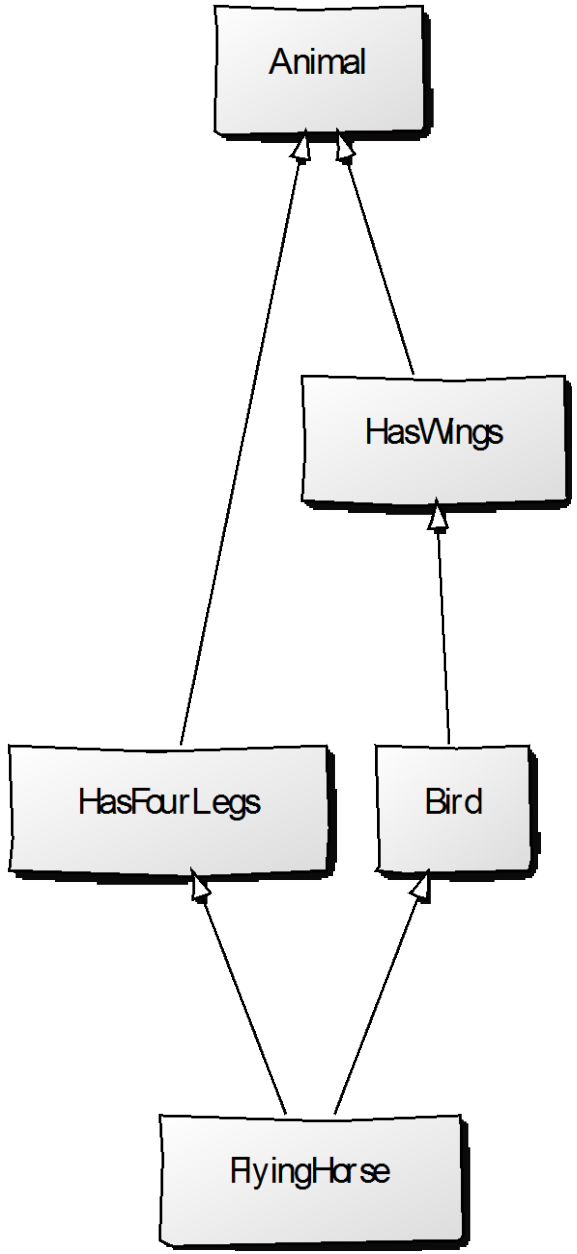

Animal
HasFourLegs
HasWings
Bird

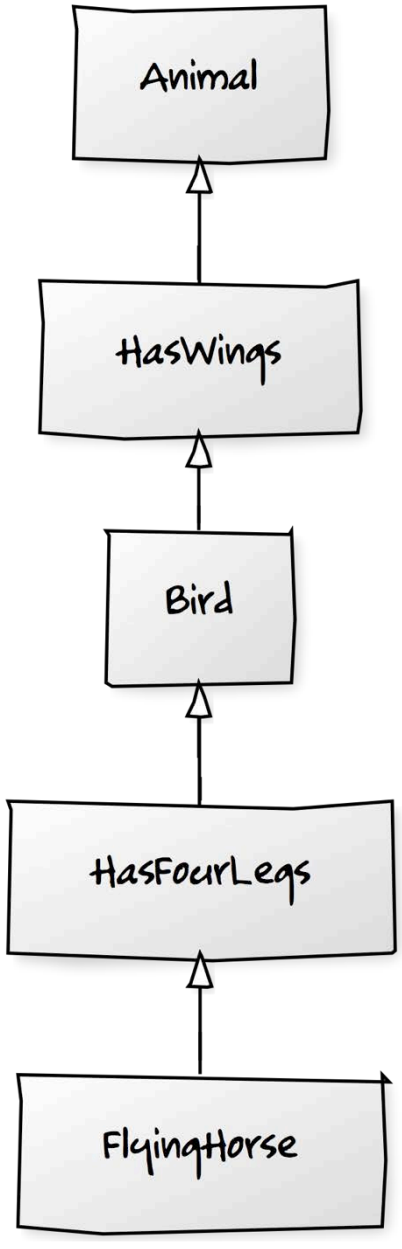
```scala
class Animal

trait HasWings

trait Bird extends HasWings

trait HasFourLegs extends Animal


class FlyingHorse extends Animal with Bird with HasFourLegs
```


Animal
HasWings
HasFourLegs
Bird
FlyingHorse


Animal
+move()
HasWings
+move()
?
HasFourLegs
+move()
Bird
+move()
?
def move {
super.move
}
FlyingHorse
+move()

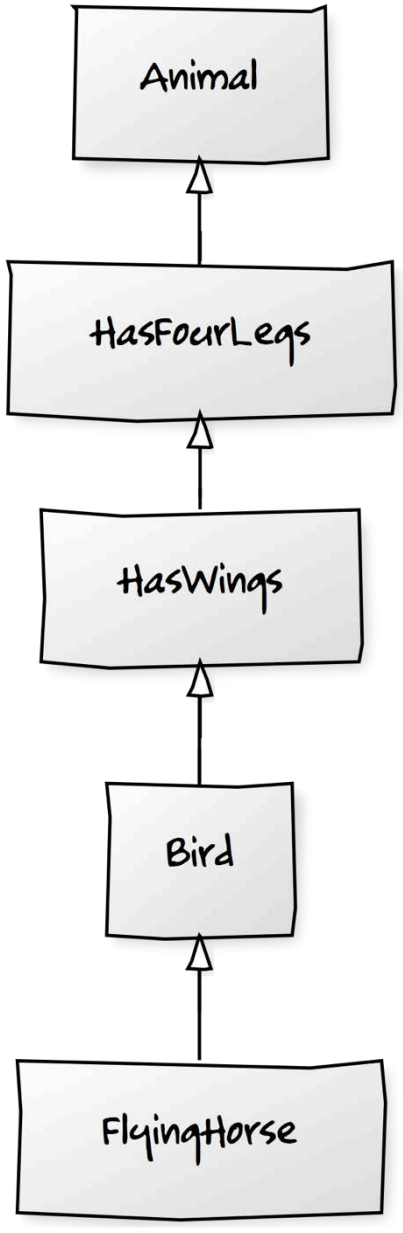
```
class Animal

trait HasWings

trait Bird extends HasWings

trait HasFourLegs extends Animal


class FlyingHorse extends Animal with Bird with HasFourLegs

// FlyingHorse, HasFourLegs, Bird, HasWings, Animal
```


Animal
HasWings
HasFourLegs
Bird
FlyingHorse

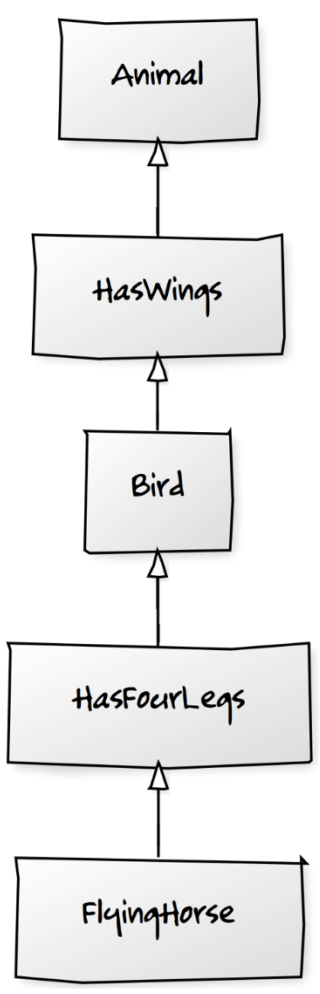

Animal
HasWings
Bird
HasFourLegs
FlyingHorse

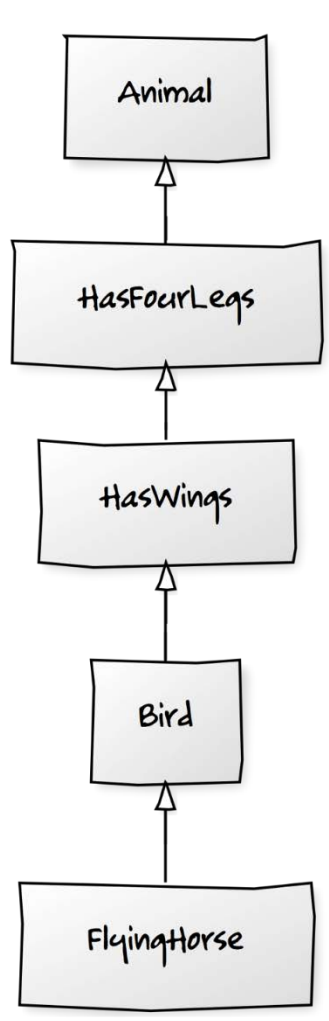
```scala
class Animal

trait HasWings

trait Bird extends HasWings

trait HasFourLegs extends Animal


class FlyingHorse extends Animal with Bird with HasFourLegs
```

```scala
class Animal

trait HasWings

trait Bird extends HasWings

trait HasFourLegs extends Animal


class FlyingHorse extends Animal with HasFourLegs with Bird
```

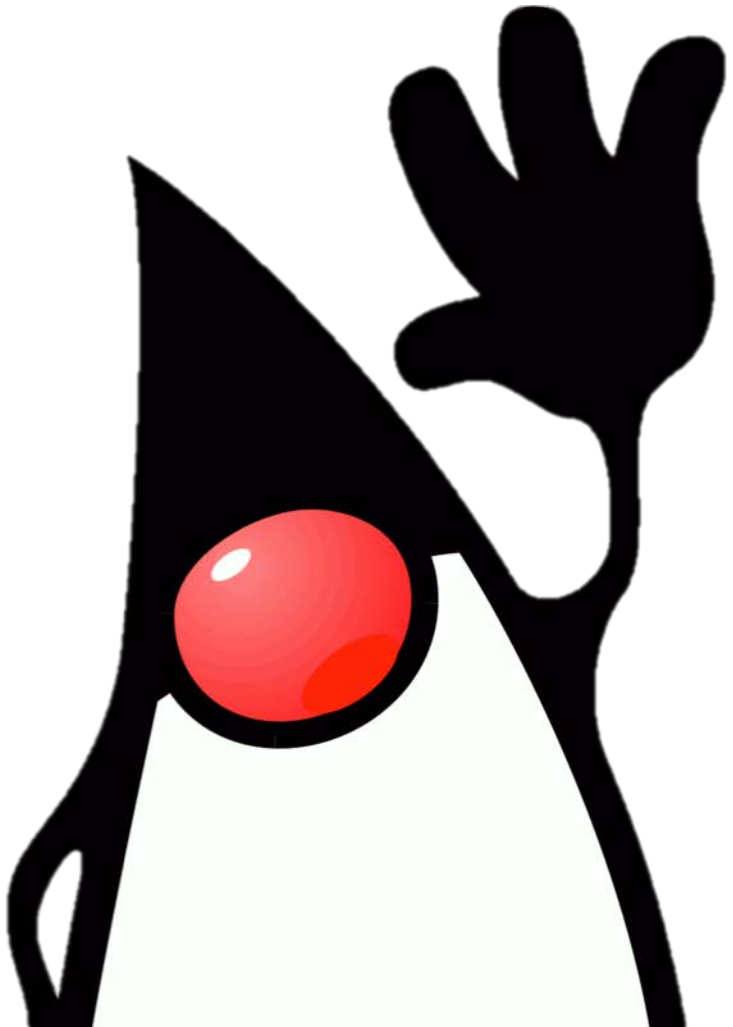

Animal
HasFourLegs
HasWings
Bird
FlyingHorse

```
class FlyingHorse extends Animal with HasFourLegs with Bird
```

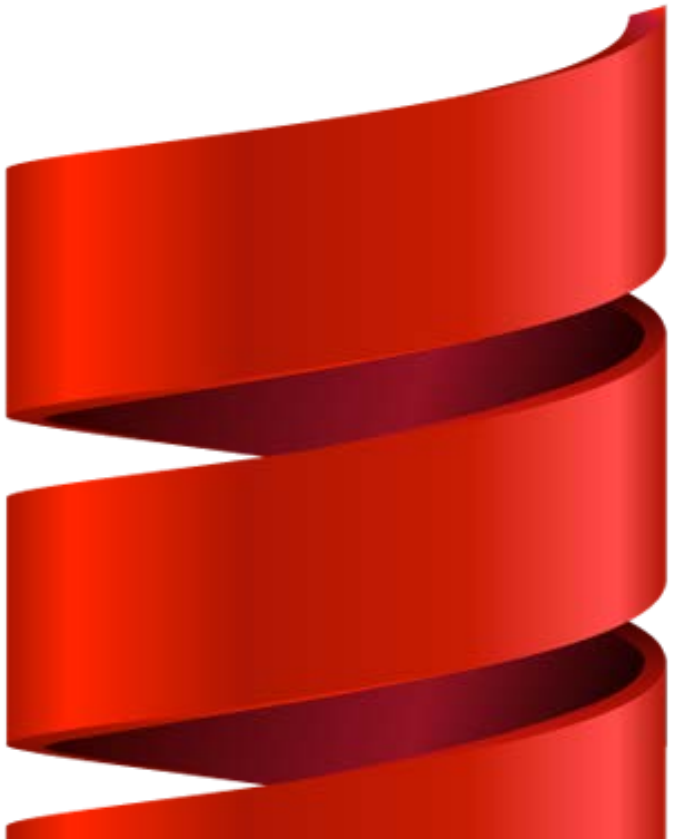





```
class FlyingHorse extends Animal with Bird with HasFourLegs
```

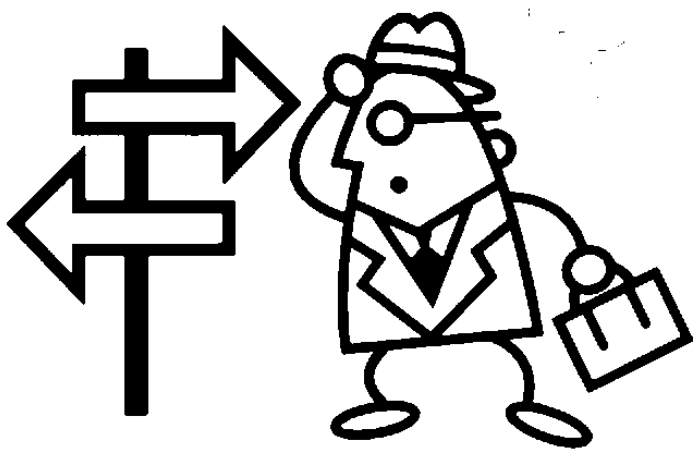


- **No Linearization in Java 8**
  - Clashes cause compiler errors



- **Stacked Traits**
  - Layered atop each other
  - Like AOP/decoration

Deciding Between the Options



Use a Trait

When “roles” can be substituted

Use a Class

When behaviour can be substituted

Use an Abstract Class

When the goal is implementation reuse for related classes

Use a Mix-in Trait

When goal is implementation reuse but for unrelated classes

A comprehensive step-by-step guide

Programming in Scala

Second Edition

Updated for Scala 2.8

Martin Odersky
Lex Spoon
Bill Venners

artima

Next up…

Key Syntactical Differences

Control Structures

- Conditionals
- Loops
- Breaks
- Exceptions

New Topic

Conditionals

if:then

01/02

DEMO

An expression *returns* a value whereas a statement *carries out an action*

An expression *returns* a value whereas a statement *carries out an action*

An expression *returns* a value whereas a statement *carries out an action*

An expression *returns* a value whereas a
statement *carries out an action*

An expression *returns* a value whereas a statement *carries out an action*

03/04

DEMO

New Topic

Ternaries

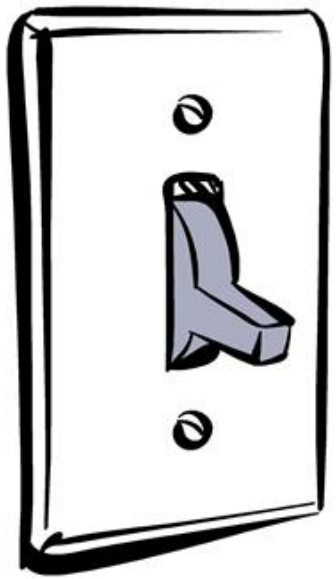
```
condition ? true : false
```

05

DEMO

- Java ternaries are *expressions*
- Java ifs are *statements*
- Scala has no ? :
- Scala ifs are *expressions*
- So Scala ifs are *equivalent* to ? :

New Topic




switch

New Term

Match Expressions

New Term

Pattern Matching

- Primitives
- Enums
- Strings

- Anything!
- Even Objects

06/07

DEMO

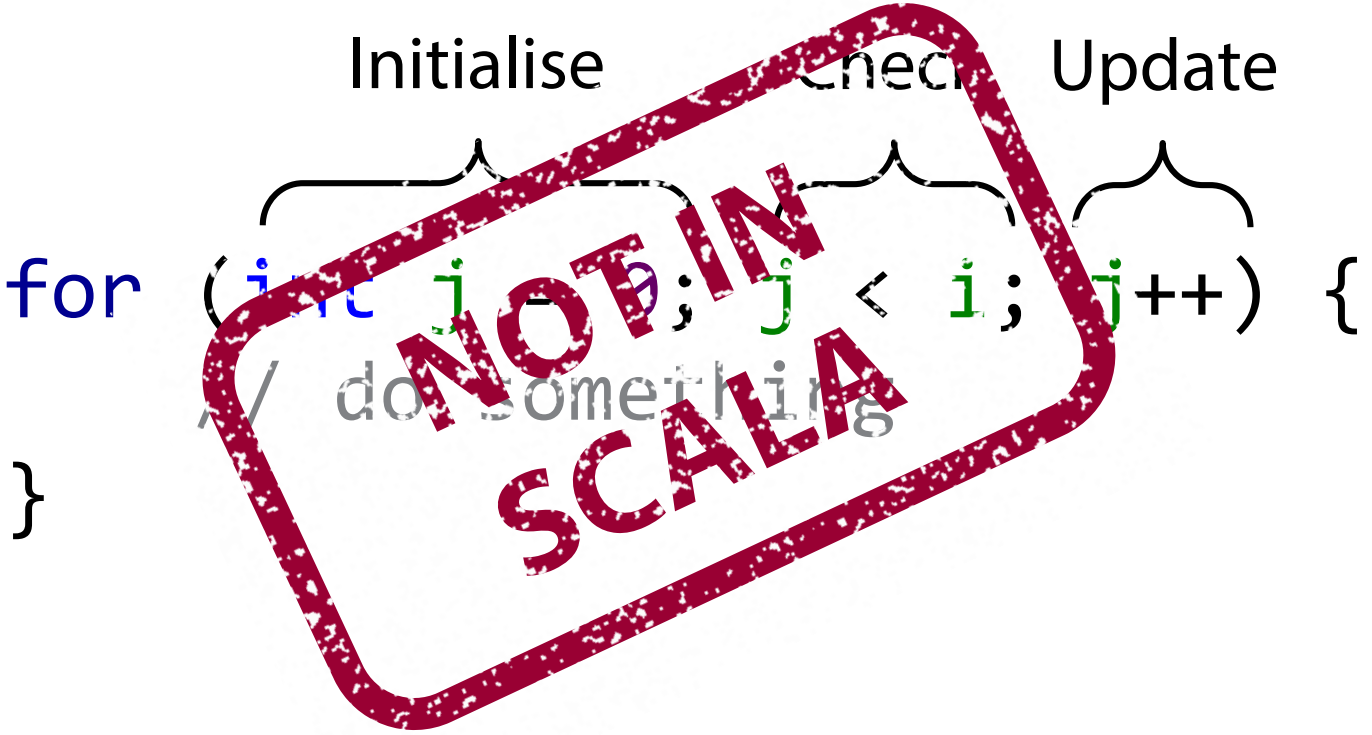
New Topic

Loops

For Loops

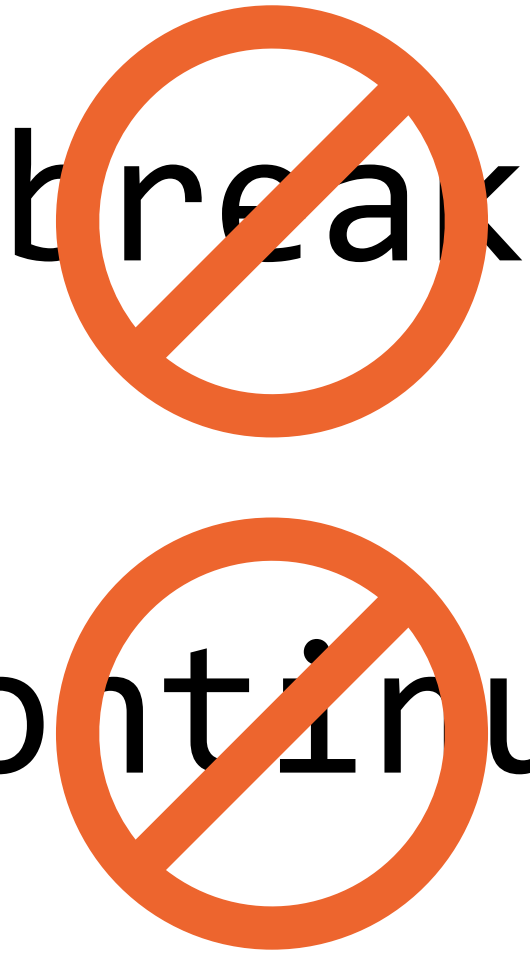
New Term

For comprehension


Initialise
Check
Update
for (int j = 0; j < i; j++) {
// do something
}
NOT IN SCALA

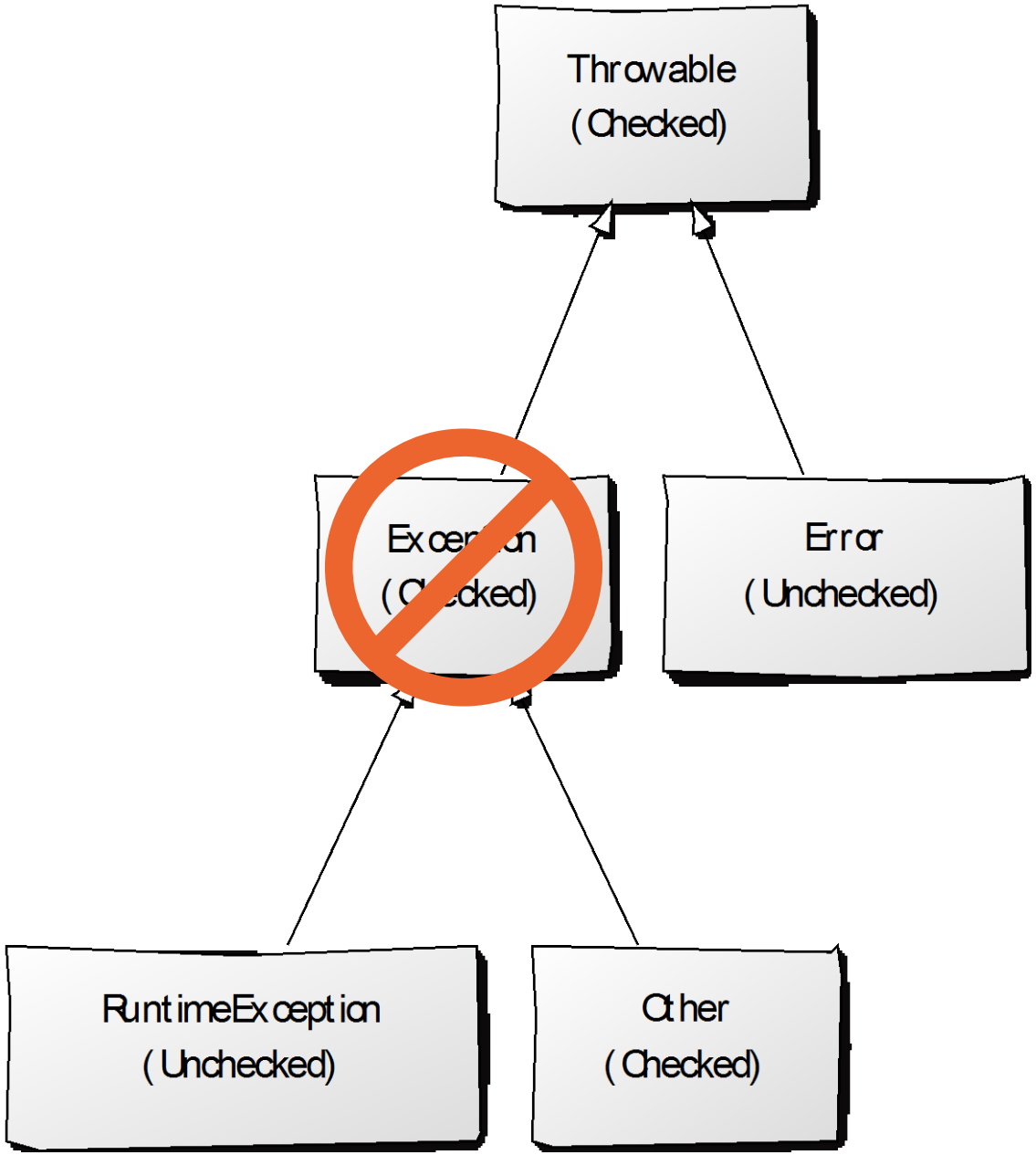
New Topic

Breaks


break
continue

New Topic

Exceptions


Throwable
(Checked)
Exception
(Checked)
Error
(Unchecked)
RuntimeException
(Unchecked)
Other
(Checked)

All

All Exceptions

All Exceptions in

All Exceptions in Scala

All Exceptions in Scala are

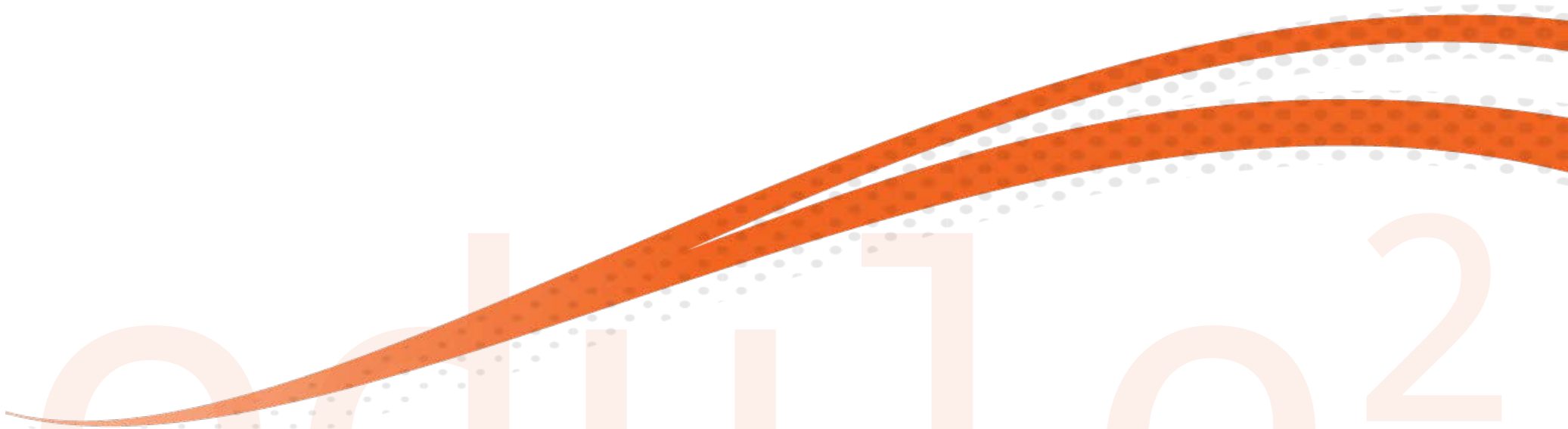
All Exceptions in Scala are `RuntimeExceptions`

Catching Exceptions

New Term

Pattern Matching

Next up…



Key Syntactical Differences

Generics

- Generic Types & Methods
- Bounded Types
- Wildcards
- Covariance / Contravariance

New Topic

Generic Programming

- ***Inclusion*** **polymorphism**
- ***Ad hoc*** **polymorphism**
- ***Parametric*** **polymorphism**

Parametric Polymorphism

```
List<Customer> customers = new ArrayList<>();
```

Parametric Polymorphism = Generics

```
List collection = new ArrayList();
```

```
List<Object> collection = new ArrayList<>();
```

New Topic

Class Generics

```
List<Customer> customers = new ArrayList<>();
customers.add(new HockeyPuck());
```

01

DEMO

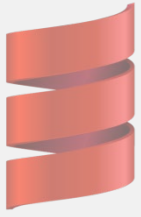
```
List<Customer> customers = new ArrayList<>();
val customers: List[Customer] = List();
```

New Topic

Method Generics

```java
public <A> void add(A a) {
    // do something
}
```

```scala
def add[A](a: A) {
    // do something
}
```



02-syntax-example-java/scala.demo

DEMO

New Topic

Bounded Types

```
List<Customer> customers = new ArrayList<>();
```

03-stack-java/scala.demo

DEMO

- **Bounded Types**
  - `extends`
  - `super`
- **Sub-type bounds** (upper bounds)
- **Super-type bounds** (lower bounds)
- **Classes or interfaces**

New Topic

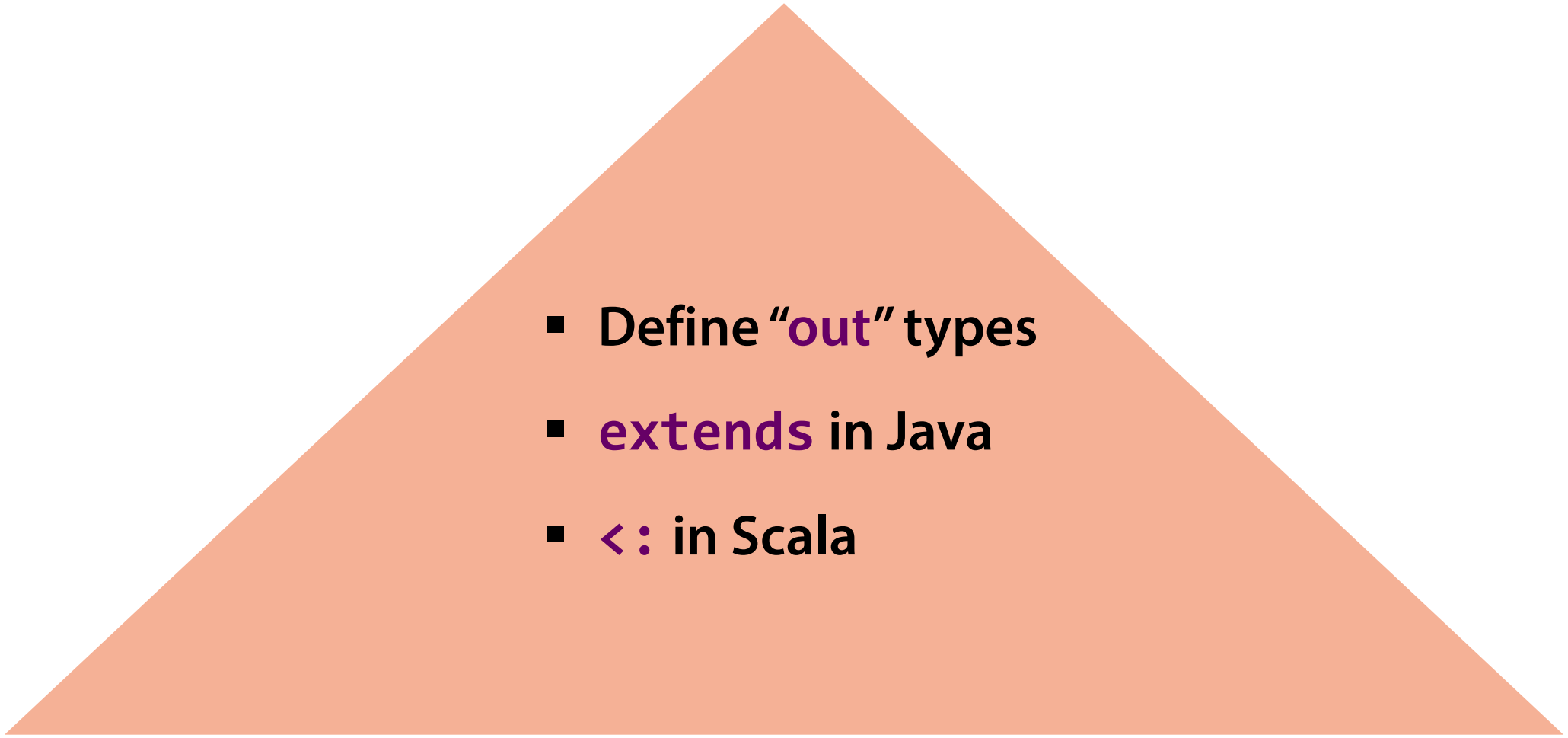
Upper Bounds

04-sortable-java/scala.demo

DEMO

```
class Customers implements Sortable<Customer> { ... }
```

```
class Customers extends Sortable[Customer] { ... }
```


Define "out" types
extends in Java
<: in Scala

New Topic

Lower Bounds

```
class Example<T, U super T> { ... }
```

- Define “in” types
- `super` in Java
- `>:` in Scala

New Topic

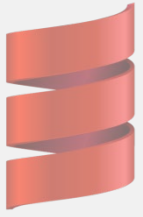
Wildcards

?

Upper Bounds

```java
void printAnimals(List<? extends Animal> animals) {
    for (Animal animal : animals) {
        System.out.println(animal);
    }
}
```

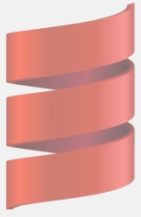
```scala
def printAnimals(animals: List[_ <: Animal]) {
  for (animal <- animals) {
    println(animal)
  }
}
```



Lower Bounds

```java
void addNumbers(List<? super Integer> numbers) {
    for (int i = 0; i < 100; i++) {
        numbers.add(i);
    }
}
```

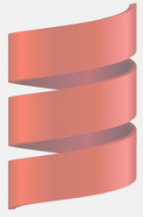
```scala
def addNumbers(numbers: List[_ >: Int]) {
  for (i <- 0 to 99) {
    // ...
  }
}
```



Unbounded

```
List<?> list
```

```
list: List[_]
```



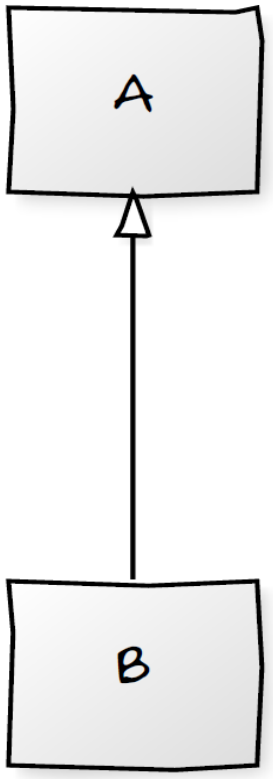
New Topic

Multiple Bounds

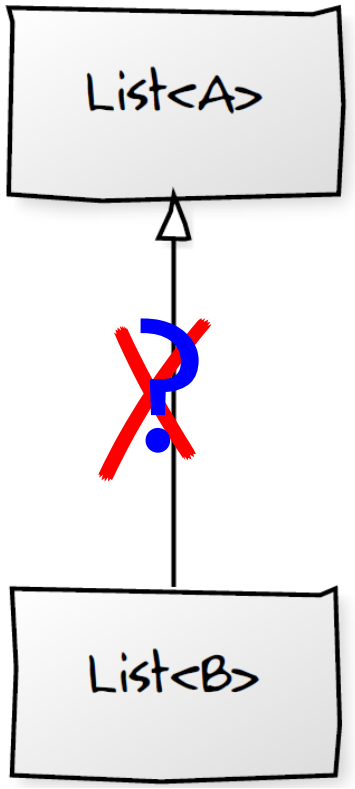
- **Java**
  - Multiple upper bounds
  - No lower **and** upper bounds
- **Scala**
  - Can set upper **and** lower bounds
  - No multiple upper bounds but…
  - Can constrain by multiple traits

New Topic

Variance

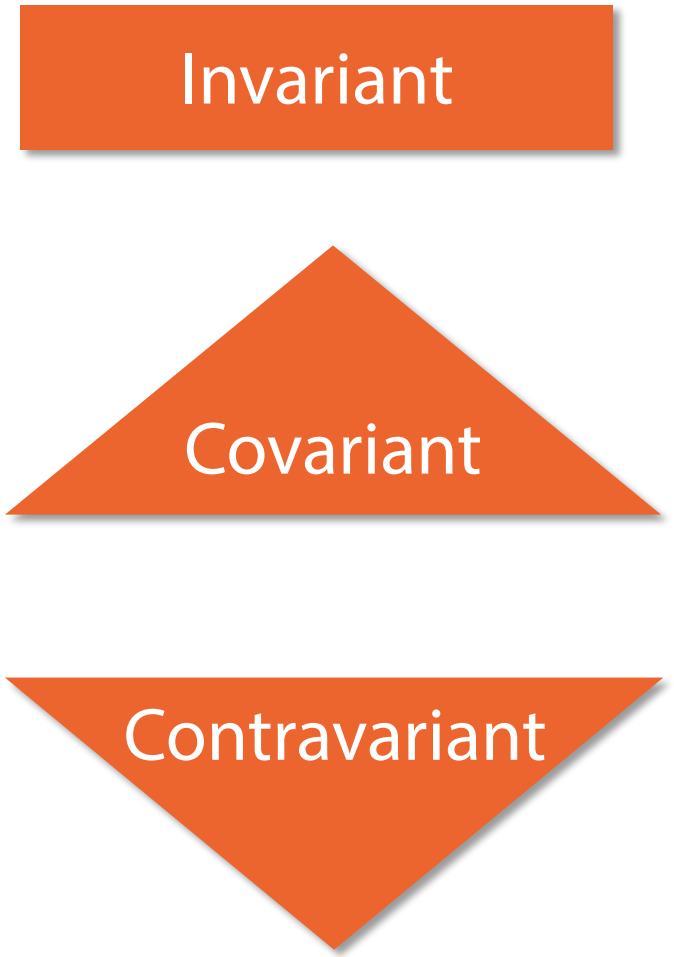


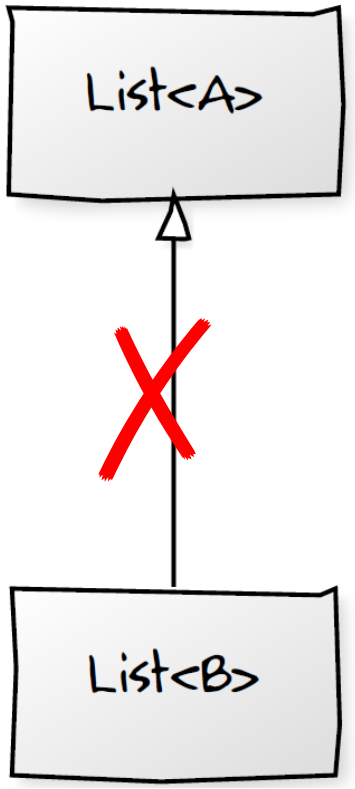

```
A a = new A();
B b = new B();

a = b;
b = a;    // compiler failure
```




```
List<A> a = new ArrayList<>();
List<B> b = new ArrayList<>();

a = b;    // compiler failure
```


Hello!
Variance

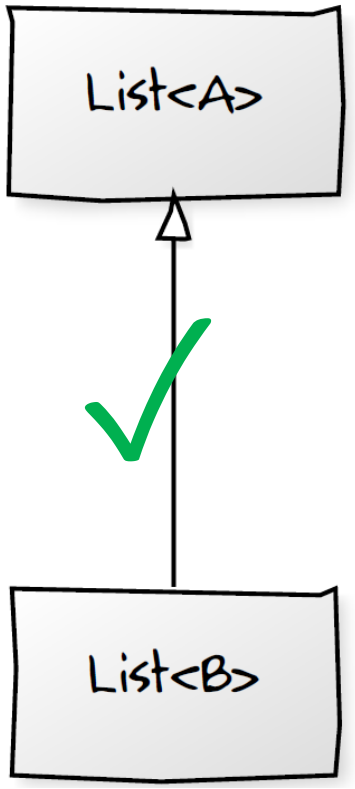

Invariant
Covariant
Contravariant



Invariant

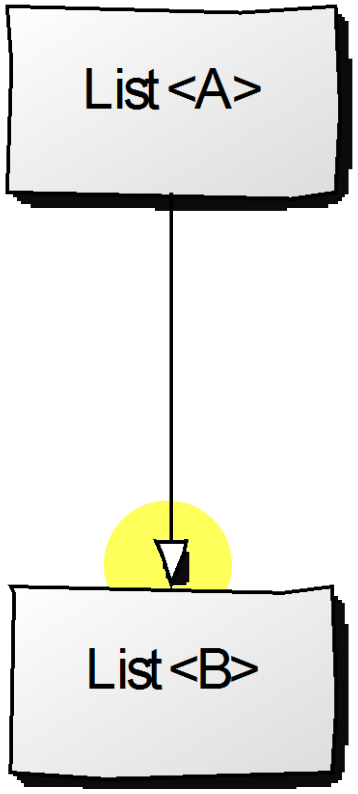
```
public class List<T> { }
```

```
class List[T] { }
```



Covariant

```
class List[+T] { }
```



Contravariant

```
class List[-T] { }
```

Summary

	Description	Scala Syntax
Invariant	List<A> and List<B> are not related	[T]

Next up…