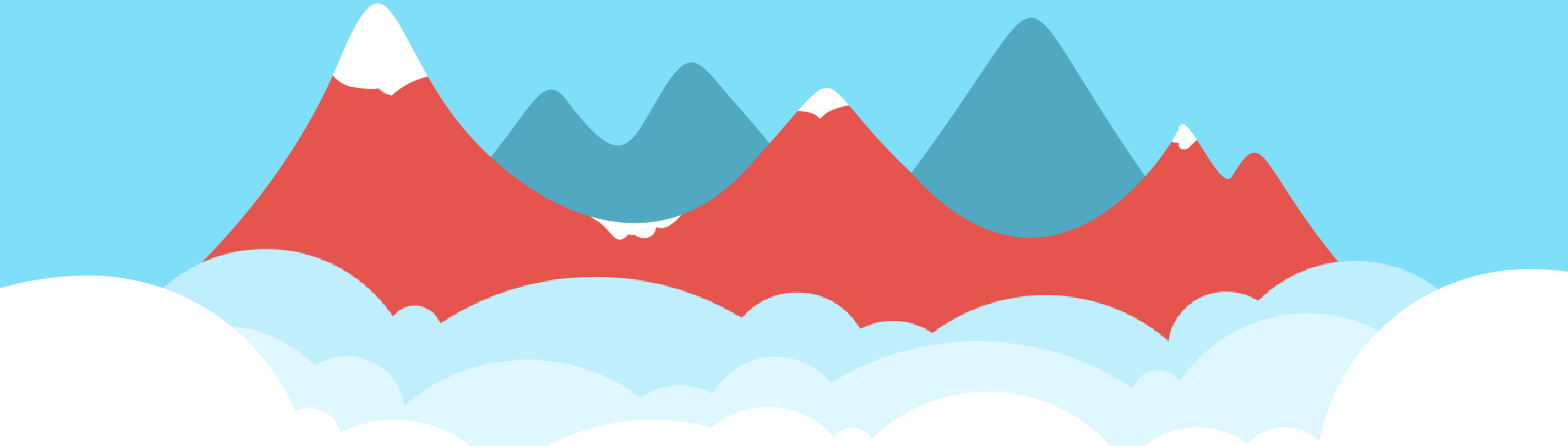
Scala Enthusiasts BS

Simon Barthel

Scala for Java Programmers



Project size

Extensibility

Scala = „scalable language“

Programming style

distributed computing

Martin Odersky

- Computer Scientist and Professor of programming methods
- Important Projects:
  - Modula2, Pizza, Generic Java, current version of javac, and of course **Scala**
- 2001: Development of Scala
- 2004: First version
- 2011: Foundation of **Typesafe**

Scala

- Combining worlds of OO and functional paradigms
- Strongly typed
- Running in the JVM

Yet another language?

New Syntax

New way of thinking

New Frameworks

New Libraries

New Environment

Dos and don'ts

Scala is a JVM Language

- For the start just use your **Java experience!**
- E.g. use **Maven-Scala-plugin** and start using Scala right now!
  - Learn some awesome new Scala features when you have time
  - Just switch back to Java when you have to be productive
  - Smoothly learn Scala over time

Variables

- Introduce new field/variable with **var**
- **Type** and **identifier** switch positions
- For generic types put type in **[squared brackets]**

```java
int i = 5;
String s = "Hello World";
Collection<Double> l = new ArrayList<Double>();
```

```scala
var i: Int = 5;
var s: String = "Hello World";
var l: Collection[Double] = new ArrayList[Double]();
```

Functions

- Introduce new method/function with **`def`**
- **Type** comes after the **parameter list**
- Add an ‘=‘ before the curly braces

```java
public String firstNChars(String s, int n) {
    return s.substring(0, n);
}
```

```scala
def firstNChars(s: String, n: Int): String = {
    return s.substring(0, n);
}
```

For-loop

- For-loops only iterate over **iterable objects**
  - Like Javas extended for-loop
- `0 to 100` creates a `Range` from 0 to 100

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

```
for(i <- 0 to 100) {
    System.out.println(i);
}
```

Try-Catch

- Catch-Block is now a `Partial Function`
  - To be introduced later

```
try { ... }
catch(IOException ioe) { ... }
catch(SQLException se) { ... }
```

```
try { ... } catch {
    case ioe: IOException => { ... }
    case se: SQLException => { ... }
}
```

Hello World!

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

```
object Main {
    def main(args: Array[String]): Unit = {
        System.out.println("Hello World");
    }
}
```

Classes

```
abstract class Person {
  public static int
    numArms = 2;
  public String name;
  public Person(String name) {
    this.name = name;
  }
  public abstract void
    eatBreakfest();
}
```

```
object Person {
  var numArms: Int = 2;
}

abstract class Person(
    var name: String
) {
  def eatBreakfest(): Unit;
}
```

Interfaces

- Interfaces are now called **traits**
- Apart from that the aforementioned rules apply

```
interface AcademicPerson {
    public String getDegree();
}

trait AcademicPerson {
    def getDegree(): String;
}
```

Inheritance

```java
class Bachelor extends Person implements AcademicPerson {
    public Bachelor(String name) {
        super(name);
    }
    @Override public void eatBreakfest() {
        System.out.println("nomnomnom");
    }
    @Override public String getDegree() {
        return "graduate";
    }
}
```

Inheritance

```
class Bachelor(name: String) extends Person(name)
with AcademicPerson {
  override def eatBreakfest(): Unit = {
    System.out.println("nomnomnom")
  }
  override def getDegree(): String = {
    return "graduate";
  }
}
```

Scala-Maven-Plugin

- Start coding Scala in your current Java Project!
  - **I show you how**

- Look up instructions at:
  - **https://github.com/scala-bs/meeting-1-MavenWithScalaAndJavaSources**