Scala A

A primer presentation by Abhi Vempati & Kevin Williams

About Scala

- Statically typed general purpose programming language that supports OOP and functional programming
- Most of its design was aimed to address the criticism of Java
- Open-source programming language

Initiation

- Creator: Martin Oderskey (R)
- Prof. at EPFL (École Polytechnique Fédérale de Lausanne)
- Led development team for lang. Internal release for 2003.
- Public release in 2004

The Uprise of Scala

- Following its public release, Oderskey's team won a research grant of over €2.3 million
- Scala continued to gain popularity since its release
- Since then, Oderskey made a commercial company, Lightbend, that offers support for Scala.

Highlights

Seamless Java Interop

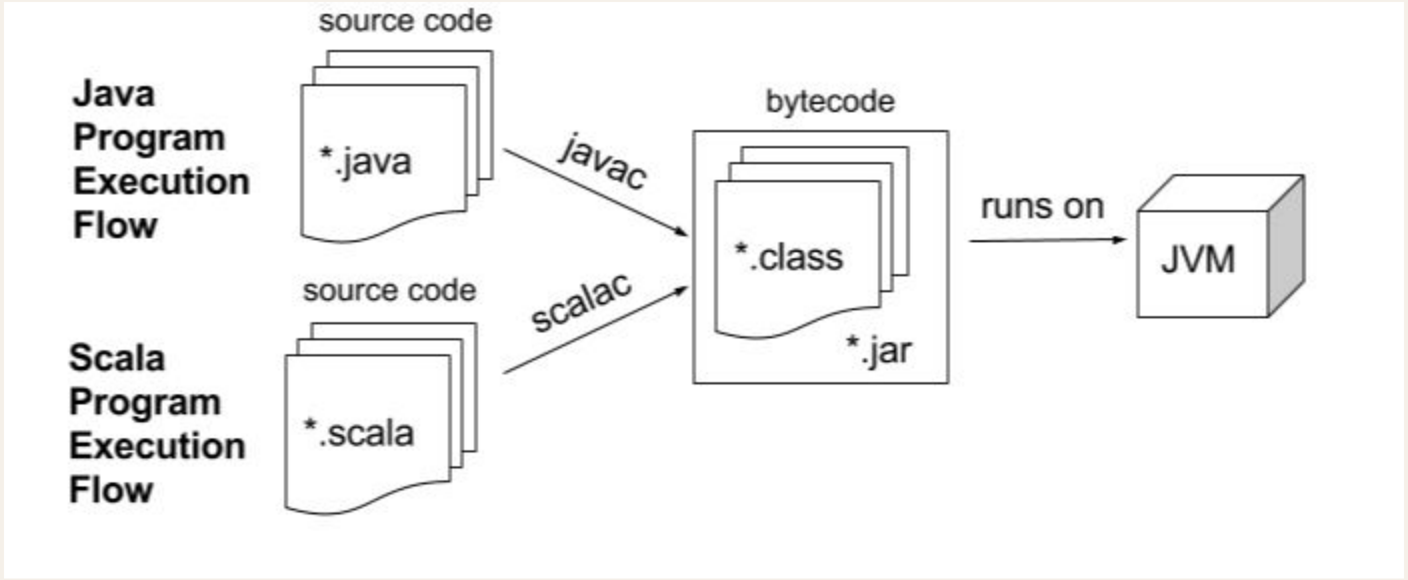
Type Inference

Traits

Pattern Matching

Higher-Order Functions

Concurrency & Distribution



Under the hood

Usage

- Big Data, Hadoop, Apache Spark
- Uprising support with Blockchain technologies
- Community growth for language
- Rise in web development with Finatra and Play
- Easy to pick up (really)! A great combination of Java and Python

Scala is Cool because...

SO on most loved languages

Link @ 53.2% (Higher than Java, R, C++, Ruby)

SO on languages associated with pay

Link -- Median pay is $150K, (higher than Python, Java, C,C++)

How to Install

How to install Scala on your machine

Don’t

Just Kidding

Scala3 installation is *slightly* tedious. We've written installation guides for all OS. Please check canvas

Scala Files and Layout

- *.scala file extension
- Object name you define will be the file name of the executable

```
object sampleClass {
    def main(args: Array[String]) =
        //code
}
```

Compile and Run

In your terminal :

```
sbt run
```

Builds the scala3 project through sbt

Variables

Declaration

```
// immutable
val a = 0

// mutable
var b = 1
```

Declaration (cont.)

```
val x: Int = 1   // explicit
val x = 1        // implicit; the compiler infers the type
```

- Implicit, compiler will infer (Type inference)
- Explicit is nice (readable)
- Both accepted but better practice is to explicitly state data type

Data Types

```
val b: Byte = 1
val i: Int = 1
val l: Long = 1
val s: Short = 1
val d: Double = 2.0
val f: Float = 3.0
```

The usual stuff. Some Java like stuff you can do (below):

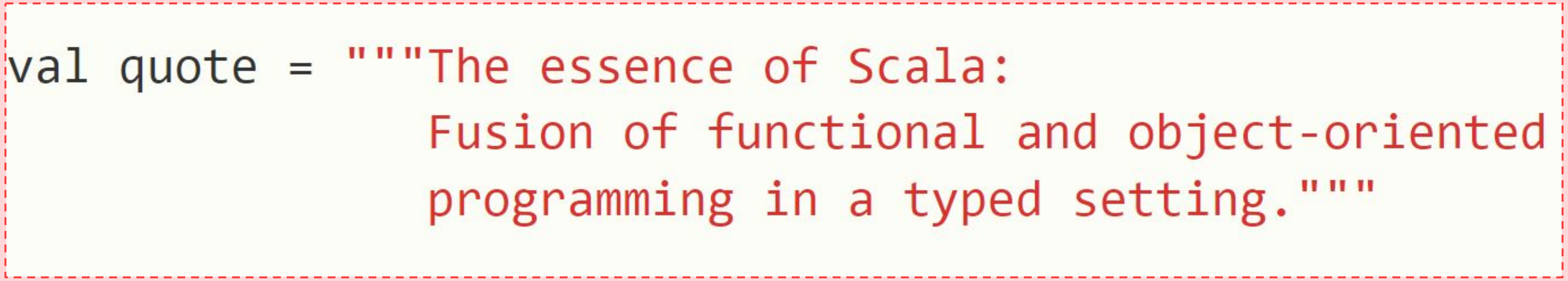
```
val x = 1_000L    // val x: Long = 1000
val y = 2.2D      // val y: Double = 2.2
val z = 3.3F      // val z: Float = 3.3
```

Mention what kind of variable it is

Strings

- Basically Java Strings. Has two additional features
  - Multiline Strings
  - String interpolation

Strings//Multiline



```
val quote = """The essence of Scala:
               Fusion of functional and object-oriented
               programming in a typed setting."""
```

Strings//Interpolation

Readable capability of using variables inside strings

```
val firstName = "John"
val mi = 'C'
val lastName = "Doe"
```

```
println(s"Name: $firstName $mi $lastName")    // "Name: John C Doe"
```

Expresion capabilities

```
println(s"2 + 2 = ${2 + 2}")    // prints "2 + 2 = 4"
```

Data Structures

Classified Into 3 Types

Sequences

Sequential collection of elements that may be indexed (array) or linear (Linked list)

Examples:
(mutable data structures)
Indexed Seq (queues)
Buffers like ArrayBuffer
ListBuffer

Maps

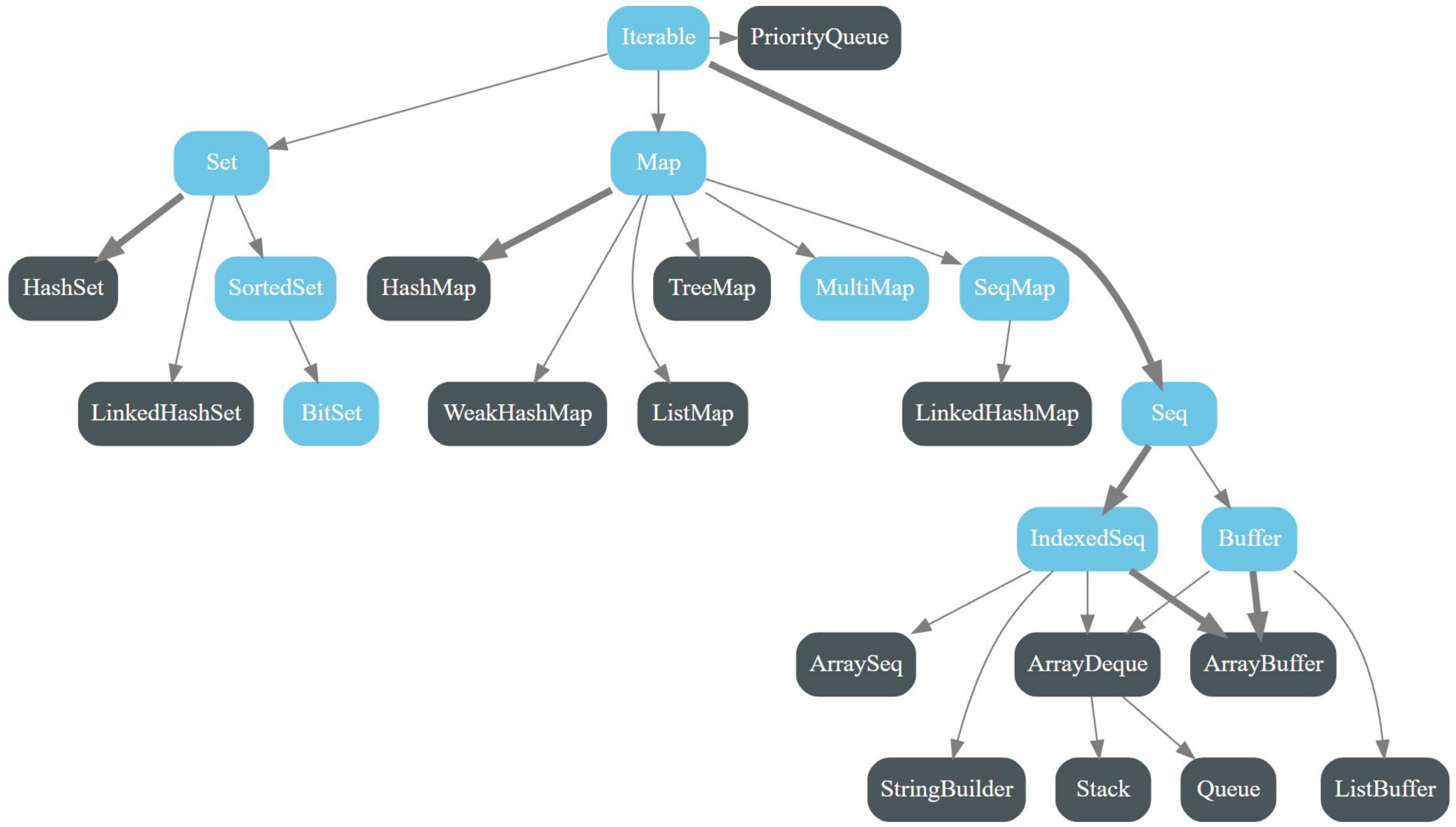
Contains a collection of key/value pairs like Java's Map or Python's Dict.

Examples:
(mutable data structures)
HashMap
TreeMap

Sets

Unordered collection of unique elements like Python

Examples:
(mutable data structures)
Hashset
Linkedset


Iterable
PriorityQueue
Set
Map
Seq
HashSet
SortedSet
HashMap
TreeMap
MultiMap
SeqMap
LinkedHashSet
BitSet
WeakHashMap
ListMap
LinkedHashMap
IndexedSeq
Buffer
ArraySeq
ArrayDeque
ArrayBuffer
StringBuilder
Stack
Queue
ListBuffer

Major Focus on ArrayBuffer

ArrayBuffer is an indexable data structure that is mutable, allowing to grow and shrink. This is like an arraylist in Java or list in Python.

ArrayBuffer

- Populate your ArrayBuffer with **fill()**:
  - Sample
  - Sample with 2D array
- Empty an ArrayBuffer with **empty()** returns the empty Array Buffer:

  sample_arrbuffer.empty()
- Access elements like:

  sample_arrbuffer(1) // returns content from index 1
- Delete elementslike:
- Sample_arrbuffer -= element

Control Structures

Content

- Conditional (if 'then' else) literally.
- For loops and expressions
- Match expressions
- While loops
- Exception handling with try/catch

if ‘then’ else (literally)

Skeleton Code:

```
if(/* condition */) then
    // code
else if(/* condition */) then
    // more cool code
else
    // final code
```

On steroids (if/else as expressions)

These are really expressions and not statements.

This can return a value.

All control structures can be used as expressions

```
val x = if a < b then a else b
```

For loops

Skeleton code:

```
for
    Generator_List
    Guards
do
    Code
```

Guards

A Scala specific feature:

Allows developers to add one or more if expressions inside the **for** loop

```
for
  i <- list_of_stuff
  if i > 2
do
  //code
  // like println(i)
```

Guard

Yield

This keyword is what makes for loops expressions.

Notice how there is no **do** keyword
Think: What language does this remind you of?

```
val x =
    for
        range of i
    yield
        //code
```

while loops

Looks similar to other languages.

Watch out for **do**

Parenthesis are optional with the **while** condition

```
while /*condition*/ do
    //code
```

match expressions

Think switch case from Java!

However, much more powerful!

The '_' serves as the default case

```scala
import scala.annotation.switch

// `i` is an integer
val day = i match
  case 0 => "Sunday"
  case 1 => "Monday"
  case 2 => "Tuesday"
  case 3 => "Wednesday"
  case 4 => "Thursday"
  case 5 => "Friday"
  case 6 => "Saturday"
  case _ => "invalid day"   // the default, catch-all
```

try/catch/finally

Like Java and other languages, allows you to catch and manage exceptions.

Scala uses same syntax as **match** expressions and supports pattern matching on the different possible exceptions that can occur.

```
var text = ""
try
  text = openAndReadAFile(filename)
catch
  case fnf: FileNotFoundException => fnf.printStackTrace()
  case ioe: IOException => ioe.printStackTrace()
finally
  // close your resources here
  println("Came to the 'finally' clause.")
```

Sample Program

A detailed walk through of the ArrayBuffer and for loop generators