

SCALA 101 - INTRODUCTION TRAINING

SESSION 1

WELCOME

- Experience and expectations
- Objectives
 - Scala syntax
 - Object-oriented concepts in Scala
 - Basics of functional programming
 - Using Scala's Collections
 - Working with SBT and ScalaTest

FORMAT OF THE TRAINING

```
repeat {  
    I.explain(Some("Scala features"))  
    you.do(exercises)  
    we.interact()  
}
```

SESSION 1

- Cleaner Java
- Object-oriented programming
- Tools
 - SBT
 - ScalaTest
- Scala Collections

WHAT IS SCALA

Scala combines object-oriented and functional programming in one concise, high-level language. Scala's static types help avoid bugs in complex applications, and its JVM and JavaScript runtimes let you build high-performance systems with easy access to huge ecosystems of libraries.

Source: <https://www.scala-lang.org/>

WHAT IS SCALA

- Functional *and* Object-Oriented
- Fully compatible with Java
 - But more concise
- Strongly Typed
 - With type inference
- Proven in production

SCALA'S STRONG POINTS

- Concurrency
- Programs correct by definition
- Good support for Domain Specific Languages (DSLs)
- All the things that Java is good at

SCALA'S CHALLENGES

- Complexity
 - Syntax
 - Type system
 - Operator overloading
- Paradigm shift
 - Object-Oriented vs Functional
- Concurrency
 - Mutability is still possible

SCALA 101 - MODULE 1

A CLEANER JAVA

LOOK MAMA, NO SEMICOLONS

```
println("Hello World!")
```

TYPE INFERENCE

Types are given *after* the identifier (as opposed to Java where they are placed in front of the identifier)

```
val str: String = "Hello World!"
```

is equivalent to

```
val str = "Hello World!"
```

The compiler infers the type of **str** from the type of the assignment expression

STRING INTERPOLATION

Given

```
val hello = "Hello"  
val world = "world"
```

This assignment

```
val greeting = hello + ", " + world + "!"
```

is equivalent to

```
val greeting = s"$hello, $world!"
```

The string interpolation is compiled as a sequence of operations on a **StringBuffer**

STRING ESCAPING

Triple quotes allow you to use quote literals in your string expressions without escaping them:

```
val str = "Hello, \"world\"!"
```

is equivalent to

```
val str = """Hello, "world"!"""
```

The triple-quote strings also support string interpolation (**s""""Hello \$name""""**)

You can use the **.stripMargin** operator to create a long string with embedded newlines (**\n**):

```
val longString = """  
| This is a pretty  
| long string that spans multiple  
| lines""".stripMargin
```

is equivalent to

```
val longString = "This is a pretty\nlong string that spans multiple\nlines"
```

JAVA INTEROPERABILITY

What differences do you spot compared to Java?

```
import java.util.ArrayList  
  
val dates = new ArrayList[LocalDate]  
  
dates.add( new LocalDate(2022, 10, 25) )  
dates.add( new LocalDate(1999, 12, 31) )  
  
println(dates.size)
```

A NOTE ABOUT STYLE



- Parentheses `()` can often be omitted
- Convention is to
 - keep them if the call has side effects
 - `dates.clear()`
 - `control.fireTheMissiles()`
 - omit them if the call has NO side effects
 - `room.getTemperature`

INFIX NOTATION

- Functions that take exactly 1 parameter can be called in different ways
- Depending on the context this can greatly improve readability

```
val str = "hello"

str.contains("e")    // normal function call
s contains "e"       // infix notation of same call
```

A NOTE ABOUT STYLE



- Never use infix notation for side-effecting calls

```
dates.add(new LocalDate(2002, 9, 11)) // DON'T DO THIS
```

```
dates.add(new LocalDate(2022, 10, 25)) // Correct
```



INFIX NOTATION

```
// method call on Integer  
1.to(3)  
  
// can be written as  
1 to 3
```

SYMBOLIC METHODS

Symbols can be powerful conveyors of meaning when used appropriately.

```
val s = "Hello"  
val t = "world"  
  
// Do not use this Java style  
val same = s.equals(t)  
val hash = s.hashCode  
  
// Use this Scala style  
val same = s == t  
val hash = s.##
```

SYMBOLIC METHODS

Scala allows you to (re)define methods with symbols in their name

```
// + is just a function name  
1.+(3)
```

```
// can be written as  
1 + 3
```

SYMBOLIC METHODS

Some more examples:

```
// operators on Types  
bigDecimal1 + bigDecimal2  
  
// send a message to an actor  
actor ! "message"  
  
val todo = ???
```

???

- The **???** can be used to tell the scala compiler that this part is not yet written
- Compiler will treat this as valid in any context where it is used
- Throws a **scala.NotImplementedError** exception when code is reached at runtime
- From the scala source code:

```
/** `???` can be used for marking methods that remain to be implemented.  
 * @throws NotImplementedError when `???` is invoked.  
 * @group utilities  
 */  
def ??? : Nothing = throw new NotImplementedError
```

A NOTE ABOUT STYLE



- Do not go overboard with inventing symbolic operators. It poses a strain on developers that need to learn the meaning
- $x : \dashv y$

Periodic Table of Dispatch Operators

All operators of Scala's marvelous Dispatch library on a single page

Start building a Request from scratch

<code>^</code>	<code><<?</code> (values)	<code>POST</code>	<code>>></code> ((in, charset) => result)	<code>as_source</code>
<code>:/ (host, port)</code>	<code>/ (path)</code>	<code>PUT</code>	<code>>></code> ((in) => result)	<code>as_str</code>
<code>:/ (host)</code>	<code><< (text)</code>	<code>DELETE</code>	<code>>~</code> ((source) => result)	<code>>>></code> (out)
<code>/ (path)</code>	<code><<< (file, content_type)</code>	<code>HEAD</code>	<code>>-</code> ((text) => result)	<code>>>> ((map) => result)</code>
<code>url (url)</code>	<code><<< (values)</code>	<code>secure</code>	<code>>>~</code> ((reader) => result)	<code>>+ (block)</code>
	<code><< (text)</code>	<code><& (request)</code>	<code><></code> ((elem) => result)	<code>~></code> ((conversion) => result)
	<code><< (values)</code>	<code>>\ (charset)</code>	<code></></code> ((nodeseq) => result)	<code>>+> (block)</code>
	<code><< (text, content_type)</code>	<code>to_uri</code>	<code>>#</code> ((json) => result)	<code>>! (listener)</code>
	<code><< (bytes)</code>		<code>> </code>	
	<code><:< (map)</code>			
	<code>as (user, passwd)</code>			
	<code>as_! (user, passwd)</code>			
	<code>gzip</code>			

EXCERCISE 1: THE BASICS



- Open a new Scala worksheet
 - File->New->Scala Worksheet
- Experiment!
- Press the green arrow to "run" the worksheet
- Observe the results



EXTRA

- Open <https://scala-lang.org/api/current>
- Find **RichInt.to**
- What parameters does it accept?
- What does it return?
- What about **RichInt.until**?

SCALA 101 - MODULE 2

OBJECT-ORIENTED PROGRAMMING

CLASSES

Initialiser values are private by default

```
class Shape(area: Double) {  
    // area is private  
}  
  
val shape = new Shape(1.0)
```

CLASSES

Public members can be added

```
class Shape(a: Double) {  
    // a is private, area is public  
    val area = a  
}  
  
val shape = new Shape(1.0)  
println(shape.area)
```

CLASSES

Public members can be added

```
class Shape(a: Double) {  
    // a is private, area is public  
    val area = a  
}  
  
val shape = new Shape(1.0)  
println(shape.area)
```

The **val** can also be used in the class definition:

CLASSES

Public members can be added

```
class Shape(a: Double) {  
    // a is private, area is public  
    val area = a  
}  
  
val shape = new Shape(1.0)  
println(shape.area)
```

The **val** can also be used in the class definition:

```
class Shape(val area: Double) {  
    // area is public  
}
```

SUB CLASSES

Scala has support for inheritance, allows you to create specialisations of a class

SUB CLASSES

Scala has support for inheritance, allows you to create specialisations of a class

```
1 class Shape(val area: Double) {  
2     // area is public  
3 }  
4  
5 class Circle(val r: Double) extends Shape(Math.PI * r * r) {  
6     // ...  
7 }  
8  
9 val circle = new Circle(1.0)  
10 println(circle.area)
```

SUB CLASSES

Body of the class definition is the constructor

```
class Circle(val r: Double) extends Shape(Math.PI * r * r) {  
    require(r > 0)  
}  
val circle = new Circle(-1.0) // throws!
```

METHODS

You can use imperative style and functional style

```
class Circle(val r: Double) {  
    // imperative style  
    def circumference: Double = {  
        return 2 * Math.PI * r  
    }  
  
    // is equivalent to functional style  
    def circumference: Double = 2 * Math.PI * r  
}
```

METHODS

```
class Circle(val r: Double) {  
  
    def draw(c: Canvas): Unit = {  
        val box = determineBoundingBox // draw using box  
  
        private def determineBoundingBox = new Rectangle(2 * r, 2 * r)  
    }  
}
```

A NOTE ABOUT STYLE

- Avoid using explicit return statements
- Always provide a return type for public **defs**



NAMED PARAMETERS

- When calling methods or creating objects, you can explicitly assign to the parameters
- By using the name of the parameter, you are no longer restricted in the order of the arguments
- Very useful for methods with a lot of parameters (that have default values)

NAMED PARAMETERS

- When calling methods or creating objects, you can explicitly assign to the parameters
- By using the name of the parameter, you are no longer restricted in the order of the arguments
- Very useful for methods with a lot of parameters (that have default values)

```
def createQueue( durable: Boolean, exclusive: Boolean, autoAcknowledge: Boolean) = ???  
  
val queue = createQueue(true, false, false) // what do these mean  
  
val queue2 = createQueue( autoAcknowledge = false, exclusive = false, durable = true) // now it is clear
```

NAMED PARAMETERS

- When calling methods or creating objects, you can explicitly assign to the parameters
- By using the name of the parameter, you are no longer restricted in the order of the arguments
- Very useful for methods with a lot of parameters (that have default values)

```
def createQueue( durable: Boolean, exclusive: Boolean, autoAcknowledge: Boolean) = ???  
  
val queue = createQueue(true, false, false) // what do these mean  
  
val queue2 = createQueue( autoAcknowledge = false, exclusive = false, durable = true) // now it is clear
```

IDE's will help you these days (with parameter hints), but we should not rely on those features too much

DEFAULT PARAMETERS

```
def createQueue( durable: Boolean, exclusive: Boolean, autoAcknowledge: Boolean = true) = ???  
  
val queue = createQueue(true, false) // uses default value for autoAcknowledge
```

CASE CLASSES

JAVA

```
public class Point {  
    private final int x;  
    private final int y;  
  
    public Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public int getX() {  
        return x;  
    }  
  
    public int getY() {  
        return y;  
    }  
  
    @Override  
    public String toString() {  
        return "Point[x=" + x + ", y=" + y + "]";  
    }  
}
```

```
@Override  
public int hashCode() {  
    final int prime = 31;  
    int result = 1;  
    result = prime * result + x;  
    result = prime * results + y;  
    return result;  
}  
  
@Override  
public boolean equals( Object obj) {  
    if (this == obj)  
        return true;  
    if (obj == null)  
        return false;  
    if (getClass() != obj.getClass())  
        return false;  
    Point other = (Point) obj;  
    if (x != other.x)  
        return false;  
    if (y != other.y)  
        return false;  
    return true;  
}
```

CASE CLASSES

SCALA

```
case class Point(x: Int, y: Int)
```

CASE CLASSES

Advantages of case classes

CASE CLASSES

Advantages of case classes

- Constructor, **equals**, **hashCode** (also **.##** alias in Scala)

CASE CLASSES

Advantages of case classes

- Constructor, **equals**, **hashCode** (also **.##** alias in Scala)
- Immutable

CASE CLASSES

Advantages of case classes

- Constructor, **equals**, **hashCode** (also **.##** alias in Scala)
- Immutable
- **copy** method

CASE CLASSES

Advantages of case classes

- Constructor, **equals**, **hashCode** (also **.##** alias in Scala)
- Immutable
- **copy** method
- Easy serialisation with the **apply** and **unapply** methods

CASE CLASSES

Advantages of case classes

- Constructor, **equals**, **hashCode** (also **.##** alias in Scala)
- Immutable
- **copy** method
- Easy serialisation with the **apply** and **unapply** methods
- Put your whole domain in a single short file

CASE CLASSES

Improve safety of your application by reducing the need for Stringly typed code

```
// Stringly typed
def execute(sql: String): Result
```

CASE CLASSES

Improve safety of your application by reducing the need for Stringly typed code

```
// Stringly typed  
def execute(sql: String): Result
```

Using (case) classes makes the intent more visible and prevents mixups:

CASE CLASSES

Improve safety of your application by reducing the need for Stringly typed code

```
// Stringly typed
def execute(sql: String): Result
```

Using (case) classes makes the intent more visible and prevents mixups:

```
case class Sql(s: String)
def execute(sql: Sql): Result
```

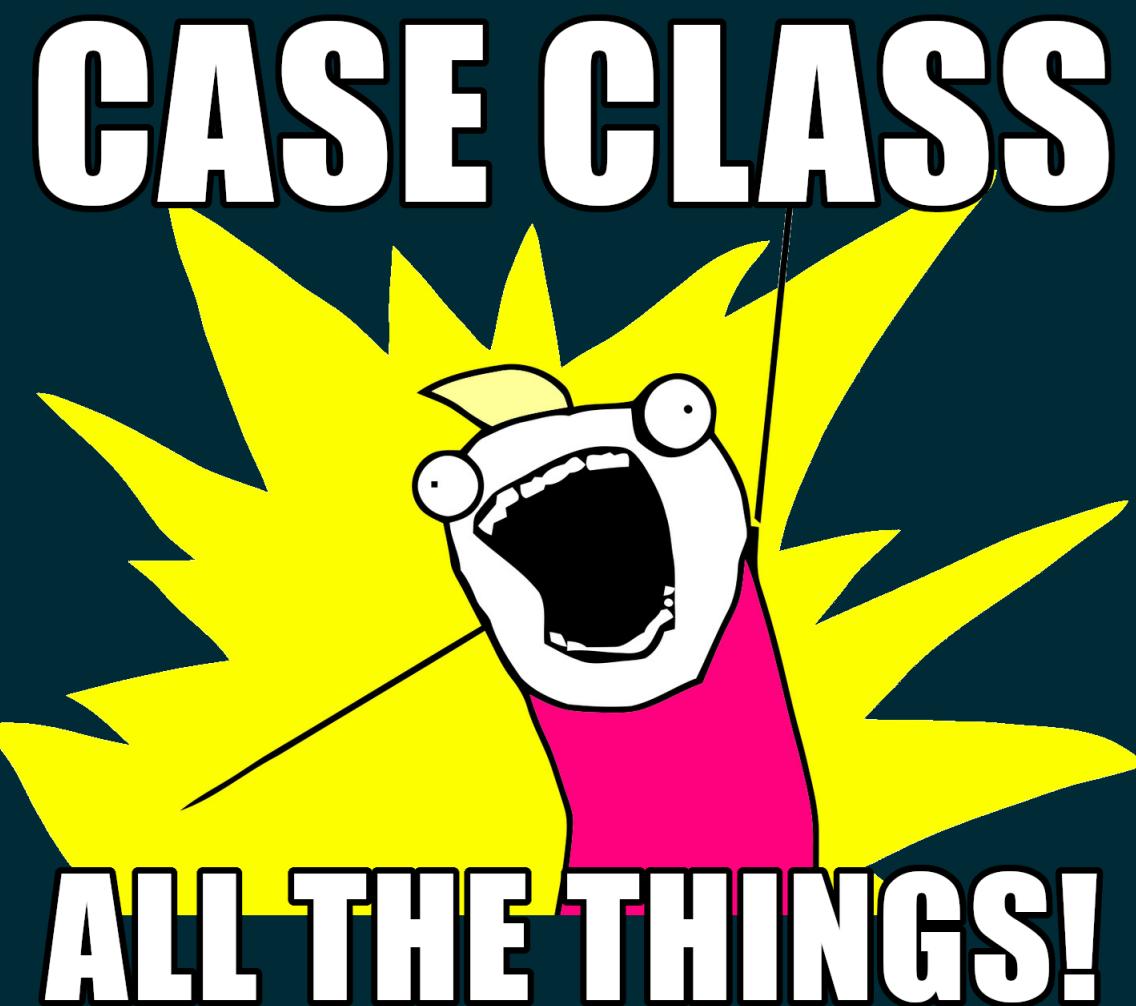
CASE CLASSES

Improve safety of your application by reducing the need for Stringly typed code

```
// Stringly typed  
def execute(sql: String): Result
```

Using (case) classes makes the intent more visible and prevents mixups:

```
case class Sql(s: String)  
def execute(sql: Sql): Result
```



SINGLETON OBJECT

- Similar to Java static methods

```
object Main {  
    def main(args: Array[String]): Unit = {  
        println("Hello world!")  
    }  
}
```

SINGLETON OBJECT

- You could write Java-like code in Scala:

```
class Circle(val r: Double)

object CircleFactory {
  def create(r: Double): Circle = new Circle(r)
}

val c = CircleFactory.create(1.0)
```

COMPANION OBJECT

- In Scala we do not like factory classes
- Companion objects offer easy ways to create instances of a specific class
- Companion objects *must* be in the same file as their class

```
1 class Circle(val r: Double)
2
3 object Circle { // notice that object uses the same name as the class
4     def create(r: Double): Circle = new Circle(r)
5 }
6
7 val c = Circle.create(1.0)
```

APPLY METHOD

- An even more Scala-esque way is to use the **apply** method for object creation

```
1 class Circle(val r: Double)
2
3 object Circle {
4   def apply(r: Double): Circle = new Circle(r)
5 }
6
7 val c = Circle(1.0)
```

APPLY METHOD

- An even more Scala-esque way is to use the **apply** method for object creation

```
1 class Circle(val r: Double)
2
3 object Circle {
4   def apply(r: Double): Circle = new Circle(r)
5 }
6
7 val c = Circle(1.0)
```

This **apply** method is automatically generated for case classes

A NOTE ABOUT STYLE



- Do not overuse the **apply** method
- Use it for factory methods in companion objects
 - Provides a nice way to convert objects into other types
- Use it when creating Domain Specific Languages

TRAITS

```
trait Ordered[T] {  
    def compareTo(other: T): Int  
}  
  
case class Player(skill: Int) extends Ordered[Player] {  
    override def compareTo(other: Player): Int =  
        skill.compareTo(other.skill)  
}  
  
player1.compareTo(player2) < 0
```

Traits are like Java's interfaces

TRAITS

```
trait Ordered[T] {  
    def compareTo(other: T): Int  
  
    def <(other: T): Boolean = this.compareTo(other) < 0  
    def >(other: T): Boolean = this.compareTo(other) > 0  
}  
  
case class Player(skill: Int) extends Ordered[Player] {  
    override def compareTo(other: Player): Int =  
        skill.compareTo(other.skill)  
}  
  
player1 < player2
```

- Traits allow you to add implementations
- Traits can be combined **... extends Trait1 with Trait2 with ...**

ENUM

```
sealed trait Color

object Color {
  case object Red extends Color
  case object Green extends Color
  case object Blue extends Color
}

val color = Color.Blue
```

- Typed enums (no risk of mixing up different enums)
- Pattern matching

OPTION

- Scala tries to avoid using **null** explicitly
- **Option** is a class that models the presence or absence of a value explicitly

```
def findUser(id: Int): Option[Person] = {  
    val result = doQuery(id)  
  
    if (result == null)  
        None  
    else  
        Some(result)  
  
    // shorthand for this if statement:  
    // Option(result)  
}
```

OPTION

```
val user: Option[Person] = findUser(42)

if (user.isDefined) {
  val u = user.get
  // do something
} else {
  // do something else
}
```

OPTION

```
val user: Option[Person] = findUser(42)

val u = user.getOrElse(defaultUser)
val a: Option[Address] = user.map(_.address)

Option(null) == None
```

IMPLICIT CLASSES

```
implicit class RichInt(val i: Int) extends AnyVal {  
    def twice = i * 2  
}  
  
println(2.twice)
```

- Allows you to add functionality to existing classes
 - very powerful, so keep in mind: W.G.P.C.G.R :)

A NOTE ABOUT STYLE



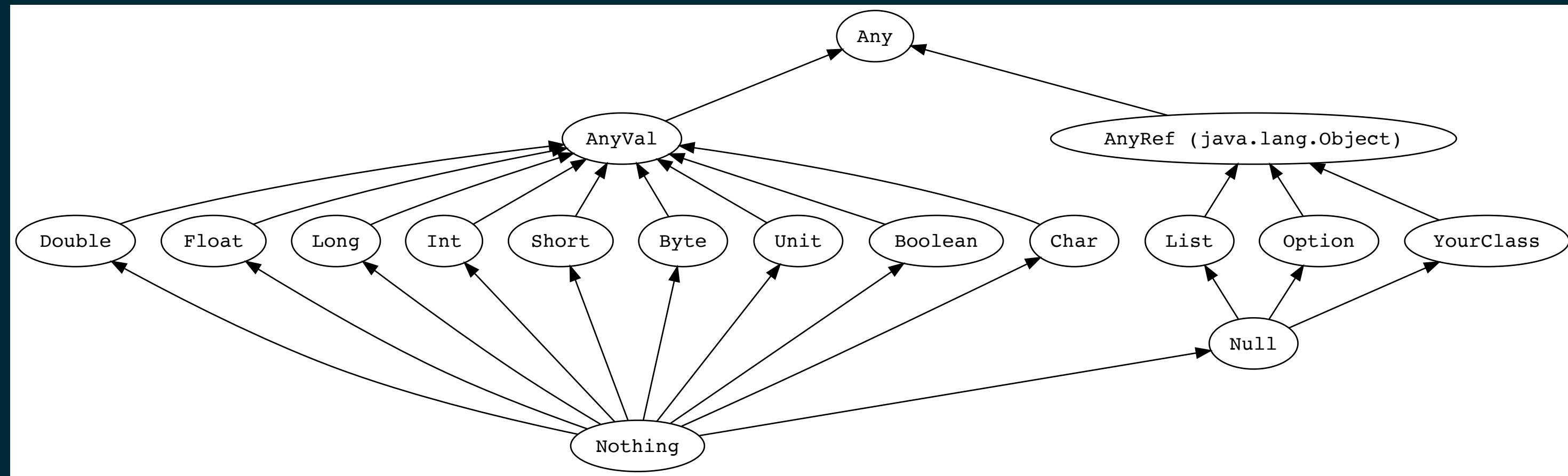
- Implicits are a controversial Scala feature
- Implicits are an advanced Scala feature
 - API / DSL designers
 - You must be aware that they exist
- Often, implicits make code harder to understand
 - **implicit def** is even more magical, and almost always a bad idea

RICH WRAPPERS

Standard Java types are enhanced by Scala's standard library:

```
1.to(3)                                // val res0: scala.collection.immutable.Range.Inclusive = Range 1 to 3
42.toHexString                           // val res1: String = 2a
"42".toInt                             // val res2: Int = 42
"2" * 3                                // val res3: String = 222
"[x=%d, y=%d]".format(2, 3)           // val res4: String = [x=2, y=3]
val regex1 = ".*@.*\\.com".r            // val regex1: scala.util.matching.Regex = .*@.*\\.com
val regex2 = """.*@.*\\.com""".r         // val regex2: scala.util.matching.Regex = .*@.*\\.com
```

TYPE HIERARCHY





EXCERCISE 2: CLASSES AND OBJECTS

Given this code

```
trait PrettyPrintable {  
    def prettyPrint: String  
}  
  
def toScreen(p: PrettyPrintable): Unit = ???  
  
val shakespeare = Author("William", "Shakespeare")  
val hamlet = Book("Hamlet", shakespeare, 1603)  
  
toScreen(hamlet)  
// must print "Hamlet, by William Shakespeare, as published in 1603"
```

How would you implement classes **Author** and **Book**

Open **exercises/src/main/scala/exercise2/Hamlet.scala** and add/update the code to make the program work as expected.

Exercises available at <https://github.com/code-star/scala101-full-course.git>

SCALA 101 - MODULE 3

TOOLS

SBT

- <https://www.scala-sbt.org/>
- Scala Build Tool
- Simple Build Tool
- Interactive Build Tool

SBT

- You can also use ant, gradle, make, maven, ...
- Most Scala projects use SBT
 - It's faster for Scala code
 - It has an interactive mode
 - It has good support for cross-compiling

BUILD.SBT

```
name := "My Cool Project"
version := "1.0"
scalaversion := "2.13.10"

libraryDependencies ++= Seq(
  "com.typesafe.akka" %% "akka-actor" % "2.5.6",
  "org.scalatest" %% "scalatest" % "3.2.14" % Test
)
```

- **% Test** is the scope in which you want this dependency to appear

BUILD.SBT

- Some dependencies are available for different scala versions
 - org.scalatest:scalatest_2.12:3.2.14
 - org.scalatest:scalatest_2.13:3.2.14
- Dependency in build.sbt
 - Use `%%` to make SBT choose the appropriate minor version based on `scalaVersion`
 - `"org.scalatest" % "scalatest_2.13" % "3.2.14" % Test`
 - `"org.scalatest" %% "scalatest" % "3.2.14" % Test`

BUILD.SBT

- Some dependencies are available for different scala versions
 - org.scalatest:scalatest_2.12:3.2.14
 - org.scalatest:scalatest_2.13:3.2.14
- Dependency in build.sbt
 - Use `%%` to make SBT choose the appropriate minor version based on `scalaVersion`
 - `"org.scalatest" % "scalatest_2.13" % "3.2.14" % Test`
 - `"org.scalatest" %% "scalatest" % "3.2.14" % Test`

BUILD.SBT

- Some dependencies are available for different scala versions
 - org.scalatest:scalatest_2.12:3.2.14
 - org.scalatest:scalatest_2.13:3.2.14
- Dependency in build.sbt
 - Use `%%` to make SBT choose the appropriate minor version based on `scalaVersion`
 - `"org.scalatest" % "scalatest_2.13" % "3.2.14" % Test`
 - `"org.scalatest" %% "scalatest" % "3.2.14" % Test`

BUILD.SBT

- Some dependencies are available for different scala versions
 - org.scalatest:scalatest_2.12:3.2.14
 - org.scalatest:scalatest_2.13:3.2.14
- Dependency in build.sbt
 - Use `%%` to make SBT choose the appropriate minor version based on `scalaVersion`
 - `"org.scalatest" % "scalatest_2.13" % "3.2.14" % Test`
 - `"org.scalatest" %% "scalatest" % "3.2.14" % Test`

PROJECT/BUILD.PROPERTIES

- It must contain the version of SBT you want to use in this project

```
sbt.version=1.7.2
```

- This *must* be present, or weird errors may happen
- It helps keeping your build stable across different installations

RUNNING SBT

- sbt clean
- sbt compile
- sbt test
- sbt
 - starts the interactive prompt

RUNNING SBT INTERACTIVELY

```
$ sbt
[info] welcome to sbt 1.7.2 (Oracle Corporation Java 11.0.2)
[info] loading global plugins from /Users/jml/.sbt/1.0/plugins
[info] loading project definition from /Users/jml/work/ordina/training/scala101-full-course/project
[info] loading settings for project root from build.sbt ...
[info] set current project to Scala 101 Full Course (in build file:/Users/jml/work/ordina/training/scala101-full-course/)
[info] sbt server started at local:///Users/jml/.sbt/1.0/server/1222e06f84b0a561587e/sock
[info] started sbt server
sbt:Scala 101 Full Course> _
```

RUNNING SBT INTERACTIVELY

```
1 sbt:Scala 101 Full Course> compile
2 [info] compiling 4 Scala sources to /Users/jml/work/ordina/training/scala101-full-course/exercises/target/scala-2.13/classes ..
3 [error] /Users/jml/work/ordina/training/scala101-full-course/exercises/src/main/scala/exercise2/Hamlet.scala:13:21: not found: \
4 [error]   val shakespeare = Author("William", "Shakespeare")
5 [error]           ^
6 [error] /Users/jml/work/ordina/training/scala101-full-course/exercises/src/main/scala/exercise2/Hamlet.scala:14:16: not found: \
7 [error]   val hamlet = Book("Hamlet", shakespeare, 1603)
8 [error]           ^
9 [error] two errors found
10 [error] (exercises / Compile / compileIncremental) Compilation failed
11 [error] Total time: 2 s, completed 29 Oct 2022, 11:56:56
12 sbt:Scala 101 Full Course> _
```

RUNNING SBT INTERACTIVELY

```
1 sbt:Scala 101 Full Course> ~compile
2 [info] compiling 4 Scala sources to /Users/jml/work/ordina/training/scala101-full-course/exercises/target/scala-2.13/classes ..
3 [error] /Users/jml/work/ordina/training/scala101-full-course/exercises/src/main/scala/exercise2/Hamlet.scala:13:21: not found: \
4 [error]   val shakespeare = Author("William", "Shakespeare")
5 [error]           ^
6 [error] /Users/jml/work/ordina/training/scala101-full-course/exercises/src/main/scala/exercise2/Hamlet.scala:14:16: not found: \
7 [error]   val hamlet = Book("Hamlet", shakespeare, 1603)
8 [error]           ^
9 [error] two errors found
10 [error] (exercises / Compile / compileIncremental) Compilation failed
11 [error] Total time: 0 s, completed 29 Oct 2022, 11:58:59
12 [info] 1. Monitoring source files for root/compile...
13 [info]     Press <enter> to interrupt or '?' for more options.
```

RUNNING SBT INTERACTIVELY

```
1 sbt:Scala 101 Full Course> ~compile
2 [info] compiling 4 Scala sources to /Users/jml/work/ordina/training/scala101-full-course/exercises/target/scala-2.13/classes ..
3 [error] /Users/jml/work/ordina/training/scala101-full-course/exercises/src/main/scala/exercise2/Hamlet.scala:13:21: not found: \
4 [error]   val shakespeare = Author("William", "Shakespeare")
5 [error]           ^
6 [error] /Users/jml/work/ordina/training/scala101-full-course/exercises/src/main/scala/exercise2/Hamlet.scala:14:16: not found: \
7 [error]   val hamlet = Book("Hamlet", shakespeare, 1603)
8 [error]           ^
9 [error] two errors found
10 [error] (exercises / Compile / compileIncremental) Compilation failed
11 [error] Total time: 0 s, completed 29 Oct 2022, 11:58:59
12 [info] 1. Monitoring source files for root/compile...
13 [info]     Press <enter> to interrupt or '?' for more options.
```

SBT WITH INTELLIJ

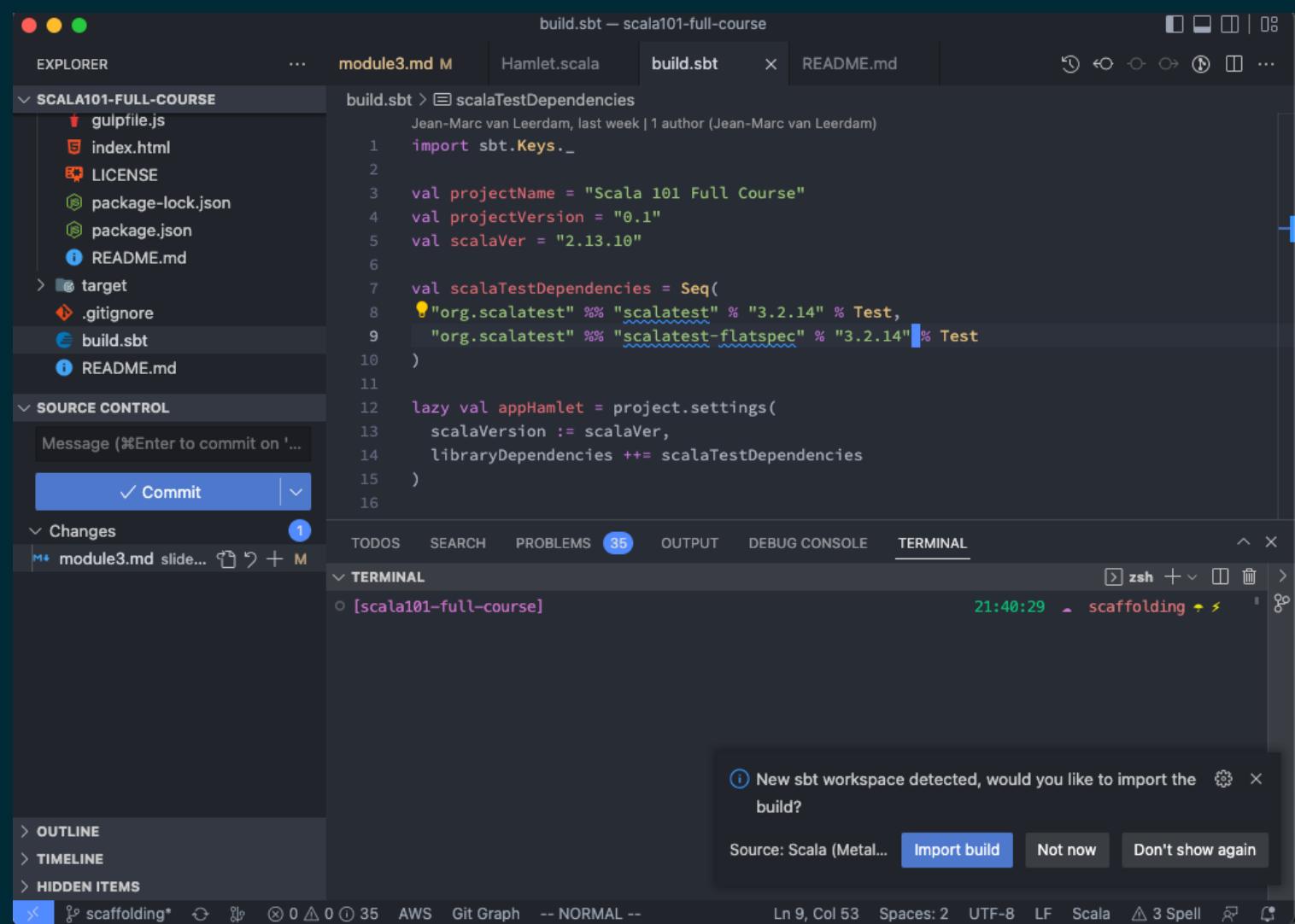
- IntelliJ has support for scala and sbt through the Scala plugin
 - provided by JetBrains
- Open project by opening the **build.sbt**

SBT WITH VS CODE

- VSCode supports scala via an extension from Scalameta, called scalameta language server, or Metals
 - Rich IDE features (completions, GOTO, import build, etc.)
 - No sbt built-in
 - sbt commands are run from Terminal instead

IMPORT A BUILD

- The first time you open Metals in a new workspace, it prompts you to import the build.
 - This is "vanilla" indexing like IntelliJ
 - Click "Import build" to start the installation step.



- You can manually import a build from Command Palette (**Cmd + Shift + P**)

>import

Metals: Import build

Bicep: Import Kubernetes Manifest

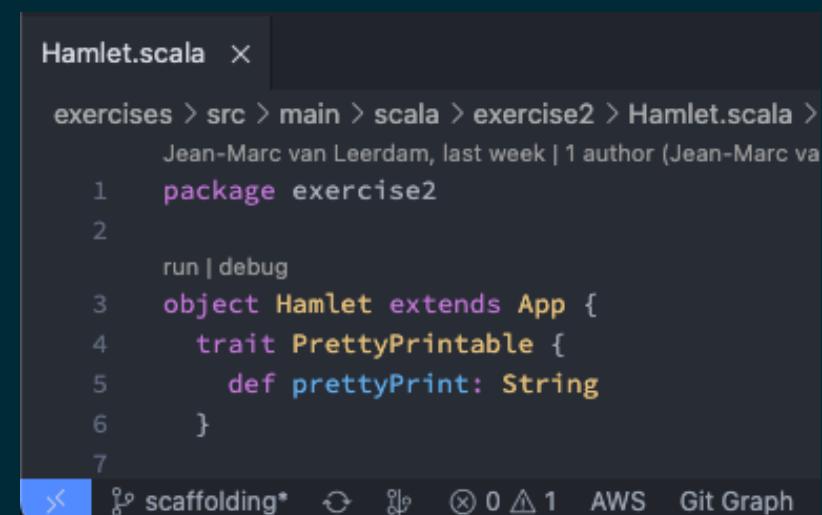
IntelliJ Keybindings: Import IntelliJ

Jupyter: Import Jupyter Notebook

Organize Imports

RUN CODE

- Once import successful, you can run the application via the UI.
- A launching point of the application must be either
 - an object that extends the **App** trait
 - an object with a properly defined **main** method



The screenshot shows a code editor window with the file 'Hamlet.scala' open. The code defines an object 'Hamlet' that extends the 'App' trait. The code is as follows:

```
Hamlet.scala  x
exercises > src > main > scala > exercise2 > Hamlet.scala >
Jean-Marc van Leerdam, last week | 1 author (Jean-Marc va
1 package exercise2
2
3 run | debug
4 object Hamlet extends App {
5   trait PrettyPrintable {
6     def prettyPrint: String
7   }
8 }
```

The code editor interface includes tabs for 'scaffolding*', 'AWS', and 'Git Graph' at the bottom.

- Otherwise, launch an sbt instance from Terminal and run commands from there

SCALATEST

- Most used testing framework for Scala
- Beautifully designed DSLs

SCALATEST

The problem with JUnit: everything has to be a method

```
@Test public void twoIntsAddedWithPlusShouldBeTheSumOfTwoInts(){ ... }  
  
@Test public void plus() { ... }
```

Test names typically are long and hard to read, or short and unclear

SCALATEST

The solution in ScalaTest:

```
// when using FunSuite
class exampleSpec extends AnyFunSuite with Matchers {
  test("2 ints added with + should be the sum of 2 ints") {
    2 + 2 shouldBe 4
  }
}

// or, when using AnyFlatSpec:
class exampleSpec extends AnyFlatSpec with Matchers {
  "2 ints added with +" should "be the sum of 2 ints" in {
    2 + 2 shouldBe 4
  }
}
```

This also works well with test tooling showing the results:

```
sbt:Scala 101 Full Course> test
...
[info] exampleSpec:
[info] 2 ints added with +
[info] - should be the sum of 2 ints
```

SCALATEST

The solution in ScalaTest:

```
// when using FunSuite
class exampleSpec extends AnyFunSuite with Matchers {
  test("2 ints added with + should be the sum of 2 ints") {
    2 + 2 shouldBe 4
  }
}

// or, when using AnyFlatSpec:
class exampleSpec extends AnyFlatSpec with Matchers {
  "2 ints added with +" should "be the sum of 2 ints" in {
    2 + 2 shouldBe 4
  }
}
```

This also works well with test tooling showing the results:

```
sbt:Scala 101 Full Course> test
...
[info] exampleSpec:
[info] 2 ints added with +
[info] - should be the sum of 2 ints
```

SCALATEST

- Assertions

```
class exampleSpec extends AnyFlatSpec with Matchers {
  "2 ints added with + " should "be the sum of 2 ints" in {
    2 + 2 should be (4)
    2 + 2 shouldBe 4
  }

  "my HashMap" should "contain 1 as a key" in {
    val map = HashMap((1 -> "a"), (2 -> "b"))

    map should contain key 1
  }
}
```

SCALATEST

- Testing Exceptions

```
class exampleSpec extends AnyFlatSpec with Matchers {

  "division by 0" should "throw an exception" in {
    intercept[ArithmeticalException] {
      1 / 0
    }
  }
}
```

SCALATEST

- There are many styles in which you can write tests
- ScalaTest lets you choose your favourite style
 - by mixing in Traits

TRAIT MIXINS

- Traits can be combined
- Similar to multiple inheritance

```
trait A {  
    def doA(): Unit = println("a")  
}  
  
trait B {  
    def doB(): Unit = println("b")  
}  
  
class X extends A with B  
  
val x = new X  
x.doA()  
x.doB()
```

CONFIGURE SCALATEST WITH TRAITS

- `org.scalatest.funsuite.AnyFunSuite` gives you the `test` method
- `org.scalatest.flatspec.AnyFlatSpec` gives you
 - `"something" should "behave like x" in`
- `org.scalatest.matchers.should.Matchers` gives you
 - `[expr] should be ([expectation])`
 - `[expr] shouldBe expectation`
- and many more, see https://www.scalatest.org/user_guide/selecting_a_style

FLATSPEC DETAILS

- Good fit to describe behavior of classes

```
class IntegerTest extends AnyFlatSpec with Matchers {  
    behavior of "Integer"  
  
    it should "add 2 ints with +" in {  
        1 + 1 shouldBe 2  
    }  
}
```

CONFIGURE SCALATEST WITH TRAITS

- ScalaTest also has **must** matchers that allow you to use **must** instead of **should**
- **OneInstancePerTest** - to create a new instance of the class for each test case (like JUnit)
- **BeforeAndAfter** - gives you methods to setup and tear down before each test
- **BeforeAndAfterAll** - gives you methods to setup and tear down before the test suite
- **MockitoSugar** - gives you syntactic sugar for calling Mockito

SCALA 101 - MODULE 4

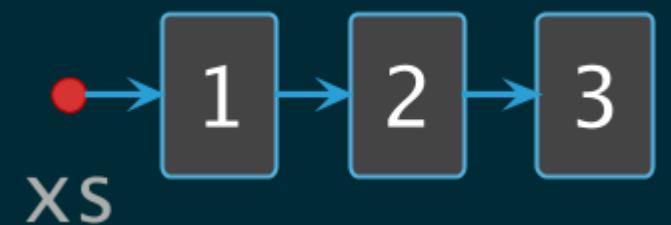
COLLECTIONS

COLLECTIONS

- Immutable by default
- Mutable if you need it

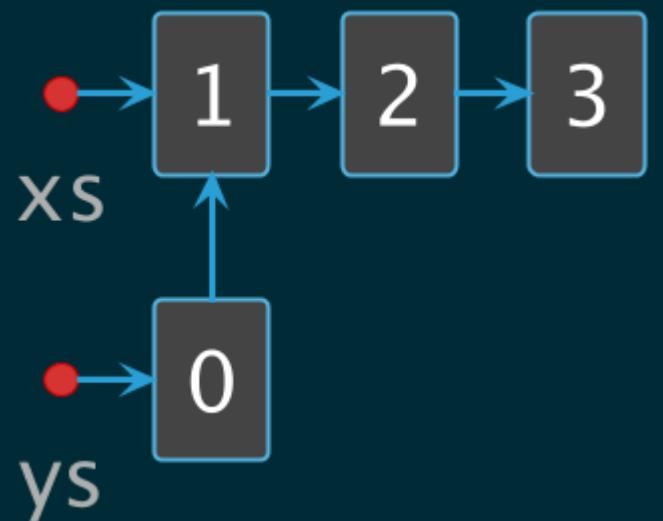
IMMUTABLE COLLECTIONS

```
val xs = List(1, 2, 3)
```



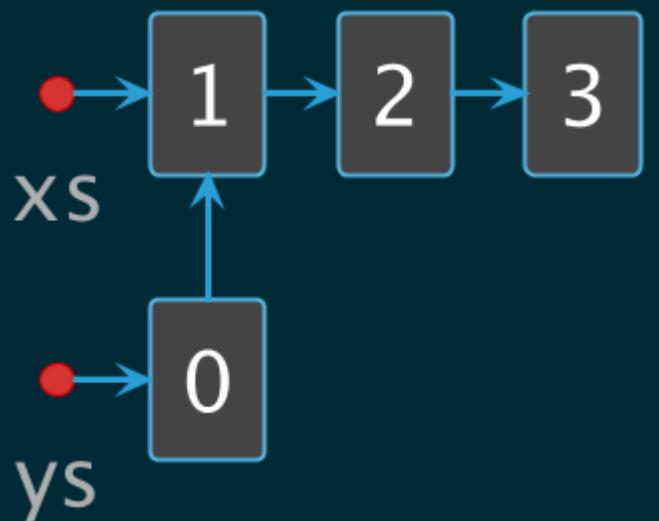
IMMUTABLE COLLECTIONS

```
val xs = List(1, 2, 3)  
val ys = 0 :: xs
```



IMMUTABLE COLLECTIONS

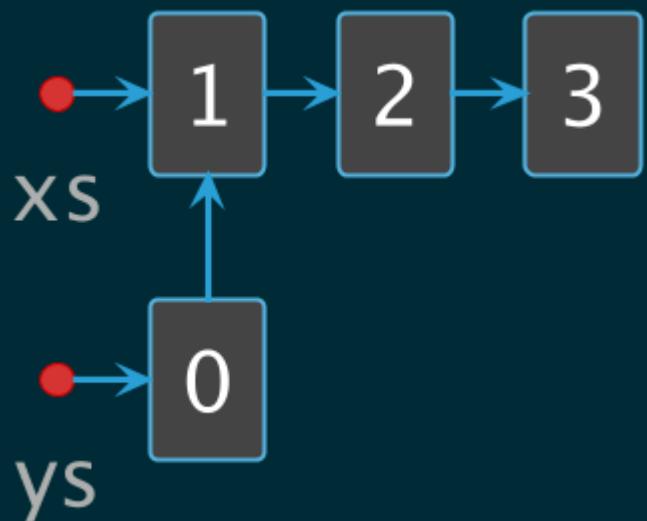
```
val xs = List(1, 2, 3)  
val ys = 0 :: xs
```



- Immutable collections can be implemented efficiently by re-using parts of existing collections

IMMUTABLE COLLECTIONS

```
val xs = List(1, 2, 3)  
val ys = 0 :: xs
```



- Immutable collections can be implemented efficiently by re-using parts of existing collections
- In a linked list, pre-pending simply re-uses the existing list, and the old pointer remains valid

IMMUTABLE COLLECTIONS

This technique is called *persistent collections*

IMMUTABLE COLLECTIONS

This technique is called *persistent collections*

- Has nothing to do with databases

IMMUTABLE COLLECTIONS

This technique is called *persistent collections*

- Has nothing to do with databases
- Demonstrated addition to a linked list

IMMUTABLE COLLECTIONS

This technique is called *persistent collections*

- Has nothing to do with databases
- Demonstrated addition to a linked list
- Other collections are similarly implemented

IMMUTABLE COLLECTIONS

This technique is called *persistent collections*

- Has nothing to do with databases
- Demonstrated addition to a linked list
- Other collections are similarly implemented
- Other operations (like remove) are also similarly implemented

IMMUTABLE COLLECTIONS

This technique is called *persistent collections*

- Has nothing to do with databases
- Demonstrated addition to a linked list
- Other collections are similarly implemented
- Other operations (like remove) are also similarly implemented
- Results in very performant operations

IMMUTABLE COLLECTIONS

This technique is called *persistent collections*

- Has nothing to do with databases
- Demonstrated addition to a linked list
- Other collections are similarly implemented
- Other operations (like remove) are also similarly implemented
- Results in very performant operations
 - But you must understand the characteristics

LISTS

```
val xs = List(1, 2, 3)
val ys = 4 :: 5 :: 6 :: Nil
val zs = xs ++ ys

zs.head
zs.tail

zs(2)
zs take 3
zs drop 2

zs.size
zs.isEmpty
```

LISTS

```
zs.mkString ", "
zs.reverse

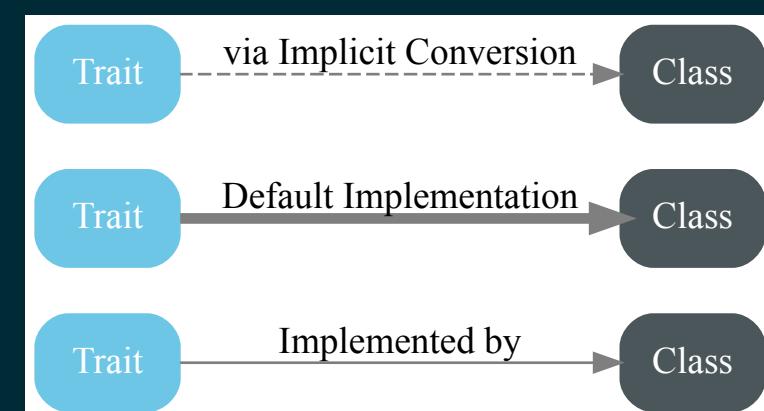
zs.min
zs.max
zs.sum

xs.zip ys

val rs = List(2, 4, 1, 3)
rs.zipWithIndex
rs.sorted
```

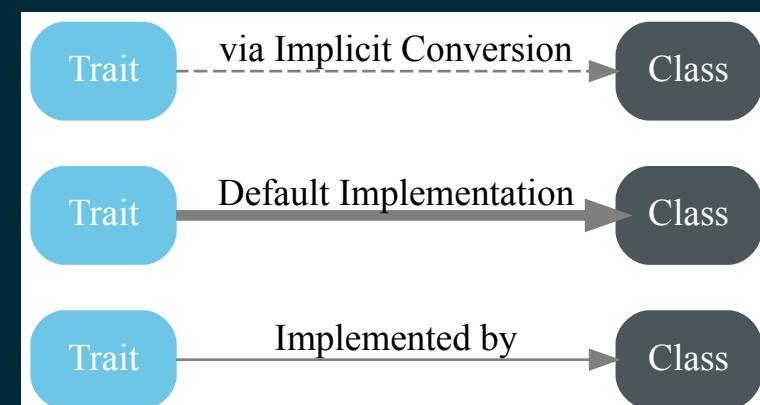
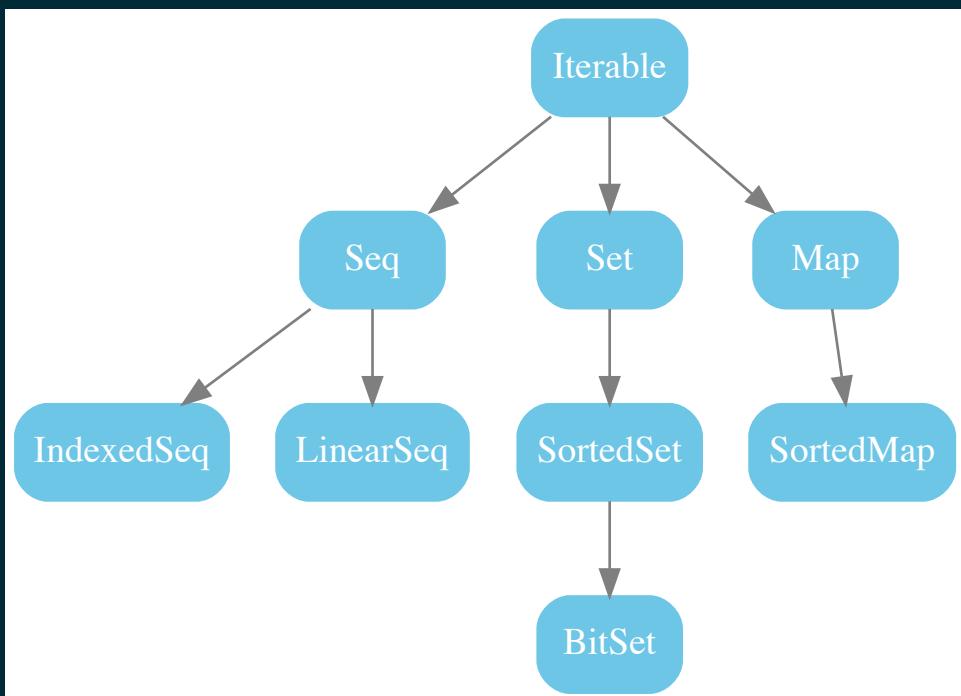
COLLECTIONS

- **List** is not the only collection



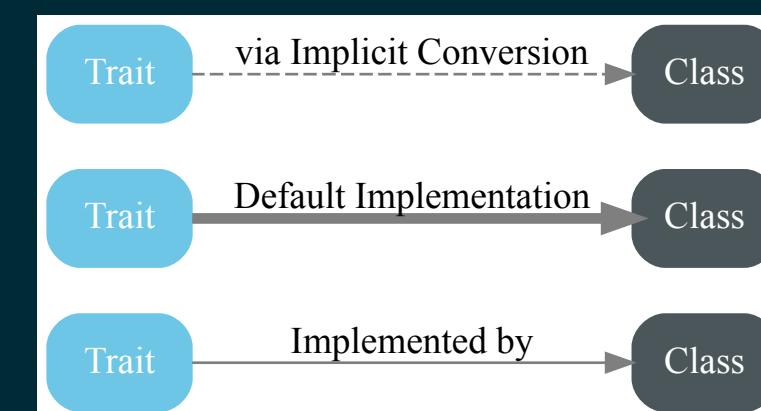
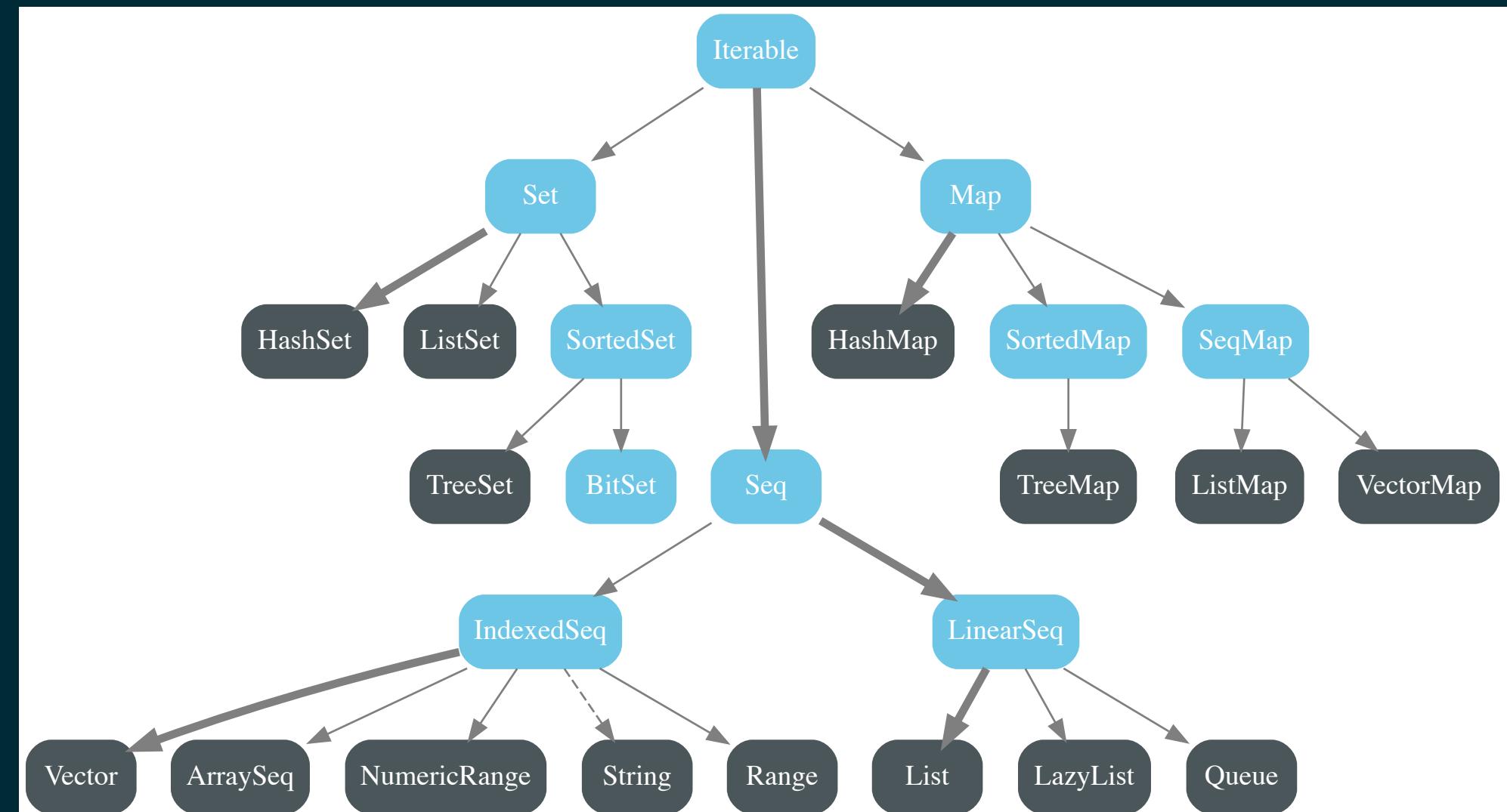
COLLECTIONS

- **List** is not the only collection



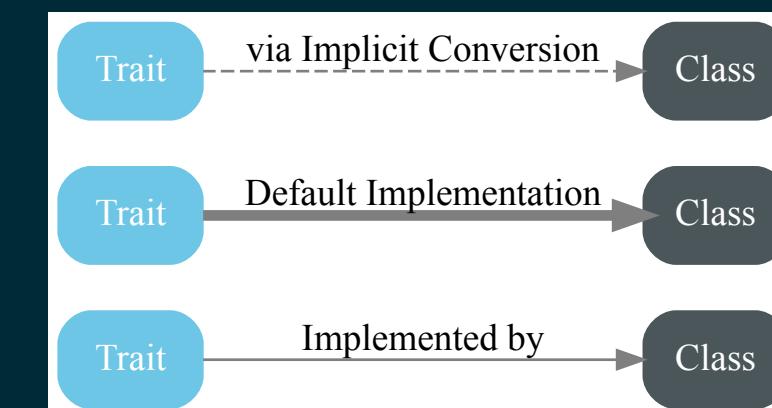
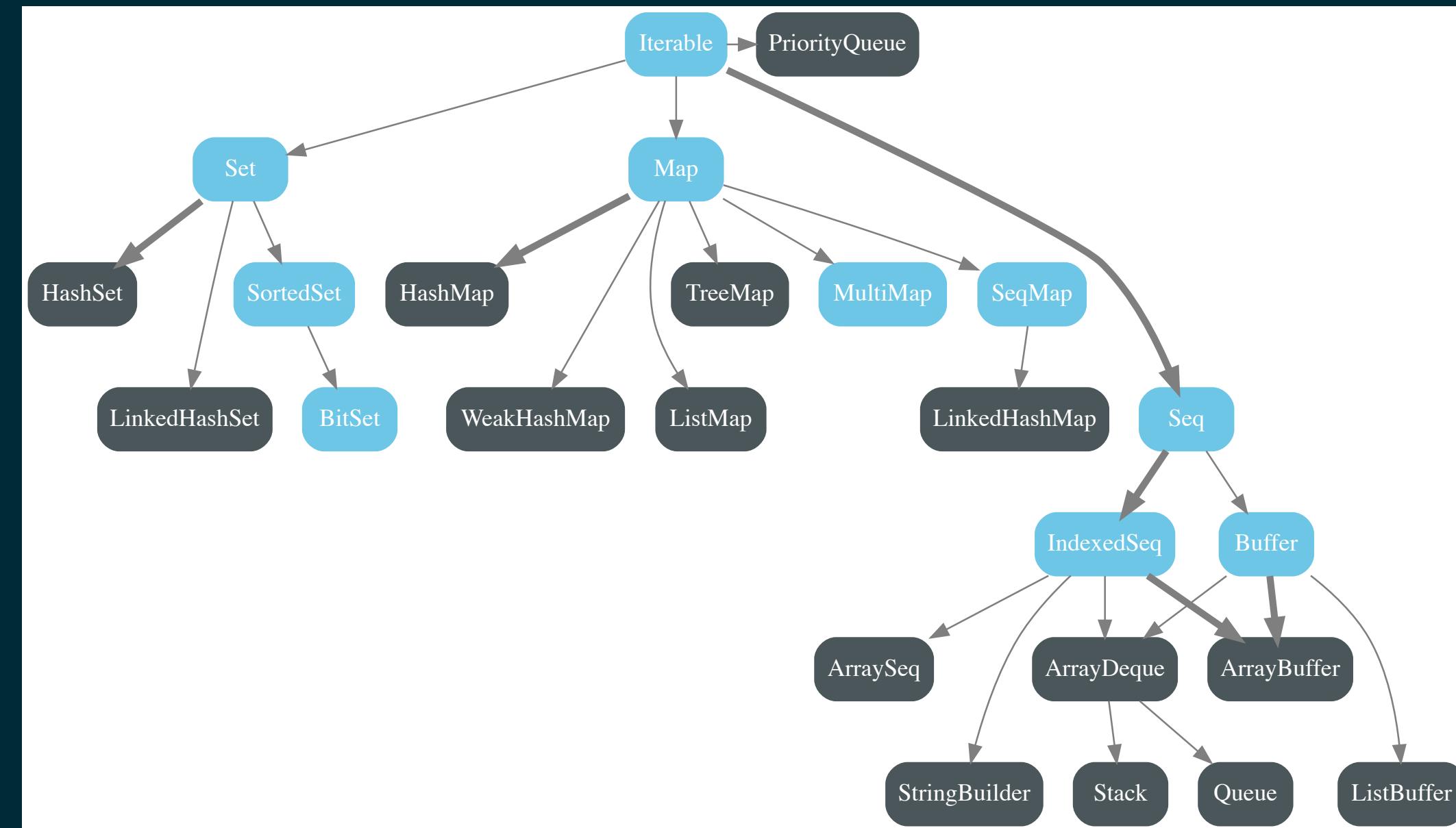
COLLECTIONS

- **List** is not the only collection



COLLECTIONS

- **List** is not the only collection



COLLECTIONS

Scala defines several collection classes:

- Base Traits
 - **Iterable** (collections you can iterate on)
 - **Seq** (ordered sequences)
 - **Set** (unordered without duplicates)
 - **Map** (lookup data structure)
- Immutable Collections
 - **List** (linked list, provides fast sequential access)
 - **Stream** (same as List, except that the tail is evaluated only on demand)
 - **Vector** (array-like type, implemented as tree of blocks, provides fast random access)
 - **Range** (ordered sequence of integers with equal spacing)
 - **String** (Java type, implicitly converted to a character sequence, so you can treat every string like a **Seq[Char]**)
 - **Map** (collection that maps keys to values)
 - **Set** (collection without duplicate elements)

COLLECTIONS

Scala defines several collection classes:

- Mutable Collections
 - **Array** (Scala arrays are native JVM arrays at runtime, therefore they are very performant)
 - Scala also has mutable maps and sets

Be aware: the *contents* of a Scala Array is mutable, even if you declare a **val ar: Array[Int] = ???**

COLLECTIONS

Scala defines several collection classes:

- Mutable Collections
 - **Array** (Scala arrays are native JVM arrays at runtime, therefore they are very performant)
 - Scala also has mutable maps and sets

Be aware: the *contents* of a Scala Array is mutable, even if you declare a **val ar: Array[Int] = ???**

A NOTE ABOUT STYLE



- Consider **List**
 - for pattern matching
 - for recursion
- Consider **Vector**
 - If you need to index elements
 - It is like Java's **ArrayList** (NOT like Java's **Vector**)

COLLECTION PERFORMANCE CHARACTERISTICS



- <https://docs.scala-lang.org/overviews/collections-2.13/performance-characteristics.html>

C	The operation takes (fast) constant time.
ec	The operation takes effectively constant time, but this might depend on some assumptions such as maximum length of a vector or distribution of hash keys.
ac	The operation takes amortized constant time. Some invocations of the operation might take longer, but if many operations are performed on average only constant time per operation is taken.
log	The operation takes time proportional to the logarithm of the collection size.
L	The operation is linear, that is it takes time proportional to the collection size. - The operation is not supported.

COLLECTION PERFORMANCE CHARACTERISTICS



- <https://docs.scala-lang.org/overviews/collections-2.13/performance-characteristics.html>

	head	tail	apply	update	prepend	append	insert
immutable							
<code>List</code>	C	C	L	L	C	L	-
<code>LazyList</code>	C	C	L	L	C	L	-
<code>ArraySeq</code>	C	L	C	L	L	L	-
<code>Vector</code>	eC	eC	eC	eC	eC	eC	-
<code>Queue</code>	aC	aC	L	L	L	C	-
<code>Range</code>	C	C	C	-	-	-	-
<code>String</code>	C	L	C	L	L	L	-
mutable							
<code>ArrayBuffer</code>	C	L	C	C	L	aC	L
<code>ListBuffer</code>	C	L	L	L	C	C	L
<code>StringBuilder</code>	C	L	C	C	L	aC	L
<code>Queue</code>	C	L	L	L	C	C	L
<code>ArraySeq</code>	C	L	C	C	-	-	-
<code>Stack</code>	C	L	L	L	C	L	L
<code>Array</code>	C	L	C	C	-	-	-
<code>ArrayDeque</code>	C	L	C	C	aC	aC	L

C The operation takes (fast) constant time.

eC The operation takes effectively constant time, but this might depend on some assumptions such as maximum length of a vector or distribution of hash keys.

aC The operation takes amortized constant time. Some invocations of the operation might take longer, but if many operations are performed on average only constant time per operation is taken.

Log The operation takes time proportional to the logarithm of the collection size.

L The operation is linear, that is it takes time proportional to the collection size.

- The operation is not supported.

COLLECTION PERFORMANCE CHARACTERISTICS



- <https://docs.scala-lang.org/overviews/collections-2.13/performance-characteristics.html>

	lookup	add	remove	min
immutable				
<code>HashSet</code> / <code>HashMap</code>	eC	eC	eC	L
<code>TreeSet</code> / <code>TreeMap</code>	Log	Log	Log	Log
<code>BitSet</code>	C	L	L	eC ¹
<code>VectorMap</code>	eC	eC	aC	L
<code>ListMap</code>	L	L	L	L
mutable				
<code>HashSet</code> / <code>HashMap</code>	eC	eC	eC	L
<code>WeakHashMap</code>	eC	eC	eC	L
<code>BitSet</code>	C	aC	C	eC ¹
<code>TreeSet</code>	Log	Log	Log	Log

Footnote: ¹ Assuming bits are densely packed.

C	The operation takes (fast) constant time.
eC	The operation takes effectively constant time, but this might depend on some assumptions such as maximum length of a vector or distribution of hash keys.
aC	The operation takes amortized constant time. Some invocations of the operation might take longer, but if many operations are performed on average only constant time per operation is taken.
Log	The operation takes time proportional to the logarithm of the collection size.
L	The operation is linear, that is it takes time proportional to the collection size.
-	The operation is not supported.

COLLECTION PERFORMANCE CHARACTERISTICS



- <https://docs.scala-lang.org/overviews/collections-2.13/performance-characteristics.html>

head	Selecting the first element of the sequence.
tail	Producing a new sequence that consists of all elements except the first one.
apply	Indexing.
update	Functional update (with <code>updated</code>) for immutable sequences, side-effecting update (with <code>update</code>) for mutable sequences).
prepend	Adding an element to the front of the sequence. For immutable sequences, this produces a new sequence. For mutable sequences it modifies the existing sequence.
append	Adding an element to the end of the sequence. For immutable sequences, this produces a new sequence. For mutable sequences it modifies the existing sequence.
insert	Inserting an element at an arbitrary position in the sequence. This is only supported directly for mutable sequences.

The second table treats mutable and immutable sets and maps with the following operations:

lookup	Testing whether an element is contained in set, or selecting a value associated with a key.
add	Adding a new element to a set or key/value pair to a map.
remove	Removing an element from a set or a key from a map.
min	The smallest element of the set, or the smallest key of a map.

C	The operation takes (fast) constant time.
ec	The operation takes effectively constant time, but this might depend on some assumptions such as maximum length of a vector or distribution of hash keys.
ac	The operation takes amortized constant time. Some invocations of the operation might take longer, but if many operations are performed on average only constant time per operation is taken.
Log	The operation takes time proportional to the logarithm of the collection size.
L	The operation is linear, that is it takes time proportional to the collection size.
-	The operation is not supported.

COLLECTION PERFORMANCE CHARACTERISTICS



- <https://docs.scala-lang.org/overviews/collections-2.13/performance-characteristics.html>

lookup	Testing whether an element is contained in set, or selecting a value associated with a key.
add	Adding a new element to a set or key/value pair to a map.
remove	Removing an element from a set or a key from a map.
min	The smallest element of the set, or the smallest key of a map.

C	The operation takes (fast) constant time.
ec	The operation takes effectively constant time, but this might depend on some assumptions such as maximum length of a vector or distribution of hash keys.
ac	The operation takes amortized constant time. Some invocations of the operation might take longer, but if many operations are performed on average only constant time per operation is taken.
log	The operation takes time proportional to the logarithm of the collection size.
L	The operation is linear, that is it takes time proportional to the collection size.
-	The operation is not supported.

SETS

- An unordered collection of items, without duplicates

```
import scala.collection.immutable.SortedSet

val s = Set(3, 1, 2, 1)
val t = SortedSet(3, 5, 4)

t + 42
t - 3

t.size
t.isEmpty
t contains 5

t intersect s
t union s
t diff s
```

MAPS

- Collection to store Key-Value pairs

```
val m = Map(1 -> "a", 2 -> "b", 3 -> "c")  
  
m + (4 -> "d")  
m - 1  
m.updated(3, "newValue")  
  
m(2)  
m(5)  
m.get(2)  
m.get(5)  
m.getOrElse(5, "WTF")  
  
m.isEmpty  
m.contains 5  
  
m.keys  
m.unzip
```



EXERCISE 3: SHAKESPEARE

Write code that takes the following input and produces the output listed below.

```
val phrase = "brevity is the soul of wit"
```

Target:

```
(brevity,0)
(is,1)
(of,4)
(soul,3)
(the,2)
(wit,5)
```

The words are sorted alphabetically, and show their position in the original string



EXERCISE 3: SHAKESPEARE

Write code that takes the following input and produces the output listed below.

```
val phrase = "brevity is the soul of wit"
```

Target:

```
(brevity,0)  
(is,1)  
(of,4)  
(soul,3)  
(the,2)  
(wit,5)
```

The words are sorted alphabetically, and show their position in the original string

Open **exercises/src/main/scala/exercise3/Brevity.scala** and add/update the code to make the program work as expected.

Exercises available at <https://github.com/code-star/scala101-full-course.git>



EXERCISE 4: TODO.TXT (HOMEWORK ASSIGNMENT)

- The assignment: write some unit tests for todo.txt
 - Source code available in module **appTodo**
 - The readme (**appTodo/README.md**) has a link to the format specification
- Hints
 - Add ScalaTest to SBT
 - Use the parser like this:

```
val input = "(A) Call mom"
val parser = TodoParser(input)
val task: Try[Task] = parser.task.run()
```

- Try is like Option:
 - **if (task.isSuccess) println(task.get) else println(":(")**

Exercises available at <https://github.com/code-star/scala101-full-course.git>

EXERCISE 4: TODO.TXT (HOMEWORK ASSIGNMENT)



- Note
 - For a single task: `parser.task.run()`
 - For a list of tasks, separated by newlines: `parser.tasks.run()`
- This is homework
 - We'll discuss it in the next session
- If you get stuck
 - Google is your friend
 - The instructor(s) - and your teammates - are your friend
 - you can contact them