Scala Lang training

Gopalakrishnan Subramani

Scala

- **Scala**ble Language
- **Pure Objected Oriented** and **Functional** Language
- Runs on Java Virtual Machine
- Java Interoperability
- Designed by Martin Odersky

Martin Odersky

The design of Scala started in 2001 at the (EPFL) by Martin Odersky. Odersky formerly worked on Generic Java, and javac, Sun's Java compiler

Odersky and collaborators launched Typesafe Inc. (renamed Lightbend Inc., February 2016)

Odersky focus is on development of language and next generation Scala compiler [Dotty]

Agile, lightweight Syntax

Object-Oriented

Functional

Safe, Performant, Strong Typing

JavaC vs ScalaC

ScalaC - Smart Compiler

Type Inference:
Compiler derive types from inference

```
val  i =10
println(i)
```

JavaC - Compiler

```
int  i =10;
system.out.println(i);
```

Use Cases

- Data Analytics
- Machine Learning
- Concurrent Programming
- Tasks
- Big Data Streaming

Frameworks

Many popular frameworks built on Scala

- Apache Spark
- Apache Kafka
- Akka Framework
- Play Framework
- Flink Framework

Compiled Language

- Scala compiled to JVM byte-code
- Produces .class files
- JVM compatible

Scala Interpretter

- Download Scala lang
- Set path to scala/bin to PATH
- Run scala in terminal/command prompt
- REPL [READ, Evaluate, Print Loop]

```
scala > println("hello")
```

Scala Advantages

- Reduce Code size Approx. 1-2 times compared to Java
- Type Inference [Drive data type automatically*]
- Smart Compiler
- Use all existing Java libraries

Scala Cons

- Steep Learning curve
- Overloaded signs, symbols [not that you have to use them all]
- Slightly painful to use inside Java
- Compiler bugs
- No binary compatibility between minor version 2.11 binary != 2.12 binary

Strongly Typed

- Strongly Typed Language
- Compile Time and Runtime type checking

Setup

- JAVA JDK 1.8
- Scala 2.13
- SBT

IDE

- IntelliJ Community Edition [Free]
- IntelliJ Commercial Edition [$$$$]
- ~~SCALA IDE~~
- Eclipse IDE

SBT

- Simple Build Tool
- Also Known as Scala Build Tool
- Apache IVY for dependency management

Simple Build Tool (SBT)

- Manage project dependencies
- Incremental compiler
- compile, run, package Scala projects
- Generate eclipse project

Scala SDK

- Includes Eclipse
- Scala Compiler
- Debugger for Scala
- Scala project creation

Null, Nil, Nothing

- Null– Its a Trait.
- null– Its an instance of Null- Similar to Java null.
- Nil– an empty List of anything of zero length.
- Nothing is a Trait. Its a subtype of everything. There are no instances of Nothing.
- None – Used to represent a sensible return value to avoid null pointer exception, used with Option
- Unit– Type of method that doesn’t return a value of anys sort, but doesn’t mean it is nothing

Data Types

- Boolean (true/false)
- Byte (8 bits), Short (16 bits), Int (32 bits), Long (64 bits)
- Float (32 bits), Double (64 bits)
- Char (16 bits), Unicode
- String
- Data Types are class types
- At JVM, Byte, Short, Int, Long, Float, Double, Char are mapped to Java native Data types

Immutable

- Immutable is core of language
- Suitable concurrent data processing
- Values can’t be changed once assigned
- Useful in functional programming
- Library immutable collection
- Threadsafe by nature

```
val PI: Double = 3.14;
//ERROR
PI = 2.14;
```

Mutable

- Mutable variable (var)
- Values can be changed
- Used inside class and object members
- Try not using var

```
var name: String = "Scala 2.11";
//OK
name = "Scala 2.12";
```

Operators

- Arithmetic +, -, *, %, /
- Logical &&, ||, !
- Relational >, <, >=, <=, !=, ==
- bitwise &, |
- Assignment =, +=, -=

No Self Increment

- No Self increment/decrement operators
- No ++ self increment
- No –- self decrement

Java vs Scala Operators

- Scala Support Operator Overloading
- Java Operators are functions

```
var i = 10;

i = i + 1
// i: Int = 11

i = i.+(1)
i: Int = 12
```

No Ternary (?)

- ? has special meaning, discussed later
- ? Is not a ternary like other languages
- Use if else for ternary
- Scala if is an expression that returns value

```
val n  = 100;
val result:String = if (n % 2 == 0) "even"
                                    else "odd"
```

Loops

- For loop
- For Yield [returns results]
- While loop
- Do while loop

Break

For Loop with break Must include breakable, from ‘Breaks’ package

```
import util.control.Breaks._
.....
breakable {
    for (i <- 1 to 10) {

    if (i == 5)  break;

    println(i);
    }
}
```

Prints

1
2
3
4

Breaks the loop at 5

Continue

- No direct support for continue
- Use break inside loop

```
import util.control.Breaks._;

for (i <- 1 to 10) {
    breakable {
        if (i % 2 == 0)  break;
        println(i);
    }
}
```

Prints only odd numbers

Nested Loop

- Nested loops can be flatted into single line

```
for ( i: Int <- 1 to 3; j: Int <- 4 to 6) {
    println(i, j)
}
```

Prints

(1, 4), (1, 5), (1, 6)
(2, 4), (2, 5), (2, 6)

For loop over list iteration

```
var numList = List(1, 2, 3, 4, 5, 6);
for ( x <- numList) {
    println(x)
}
```

Prints 1, 2,3,4,5,6

For To

- Scala has variances of for loop The simple one

```
for ( i: Int <- 0 to 5) {
    println(i)
}
```

this prints 0, 1, 2, 3, 4, 5

For Until

```
for ( i: Int <- 0 until 5) {
    println(i)
}
```

prints 0, 1, 2, 3, 4

For loop with if guard

```
var numList = List(1, 2, 3, 4, 5, 6);
for ( x <- numList if x %2 == 0) {
    println(x)
}
```

prints 2, 4, 6

For Yield

- Using for yield returns iteratable collection
- for..yield makes for loop as an expression, that returns values

```
var itr = for (i <- 0 until 5) yield i;

for (k <- itr) {
 println(k);
}
```

While Loop

```
var i = 0

while (i < 5) {
    println(i)
    i += 1
}
```

Do while

```
var i = 0

do {
    println(i)
    i += 1
} while  (i < 5)
```

No Semi-colon rule

- No compulsory semi colon for end of the statement
- No harm in using semi color if you prefer

Import a package

- import a package
- Import classes, objects from other packages

```
import scala.util

var rnd = new util.Random
println(rnd.nextInt())
```

import a class/object

- Import classes, objects from other packages

```
import scala.util.Random

var rnd = new Random
println(rnd.nextInt(1000))
```

import some

- use specific classes/objects

```
import scala.util.{Random, Success}
```

```
var rnd = new Random
println(rnd.nextInt(1000))
```

import all

- use _ for importing all

```
import scala.util._
var rnd = new Random
println(rnd.nextInt(1000))
```

import with alias

- Import with alias name

```
import scala.util.{Random => RandomX}
var rnd = new RandomX
println(rnd.nextInt(1000))
```

import Hide

- Hide random, import everything
- Random is not imported

```
import scala.util.{Random => _, _}

//below lines throw error
var rnd = new Random
println(rnd.nextInt(1000))
```

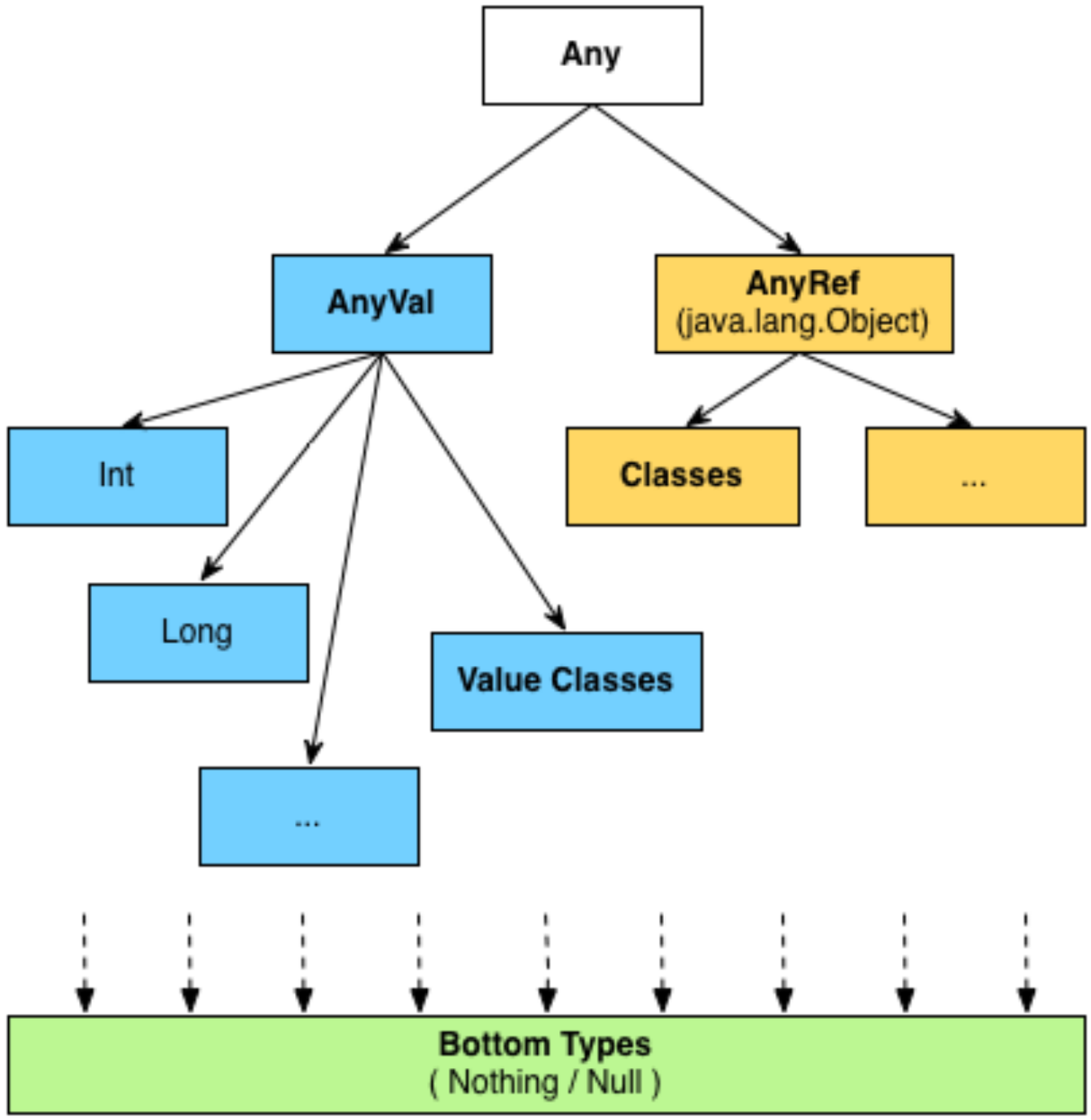
Import Scoped

- import scope

- import in scope bountry, – Random can’t be accessed outside scope

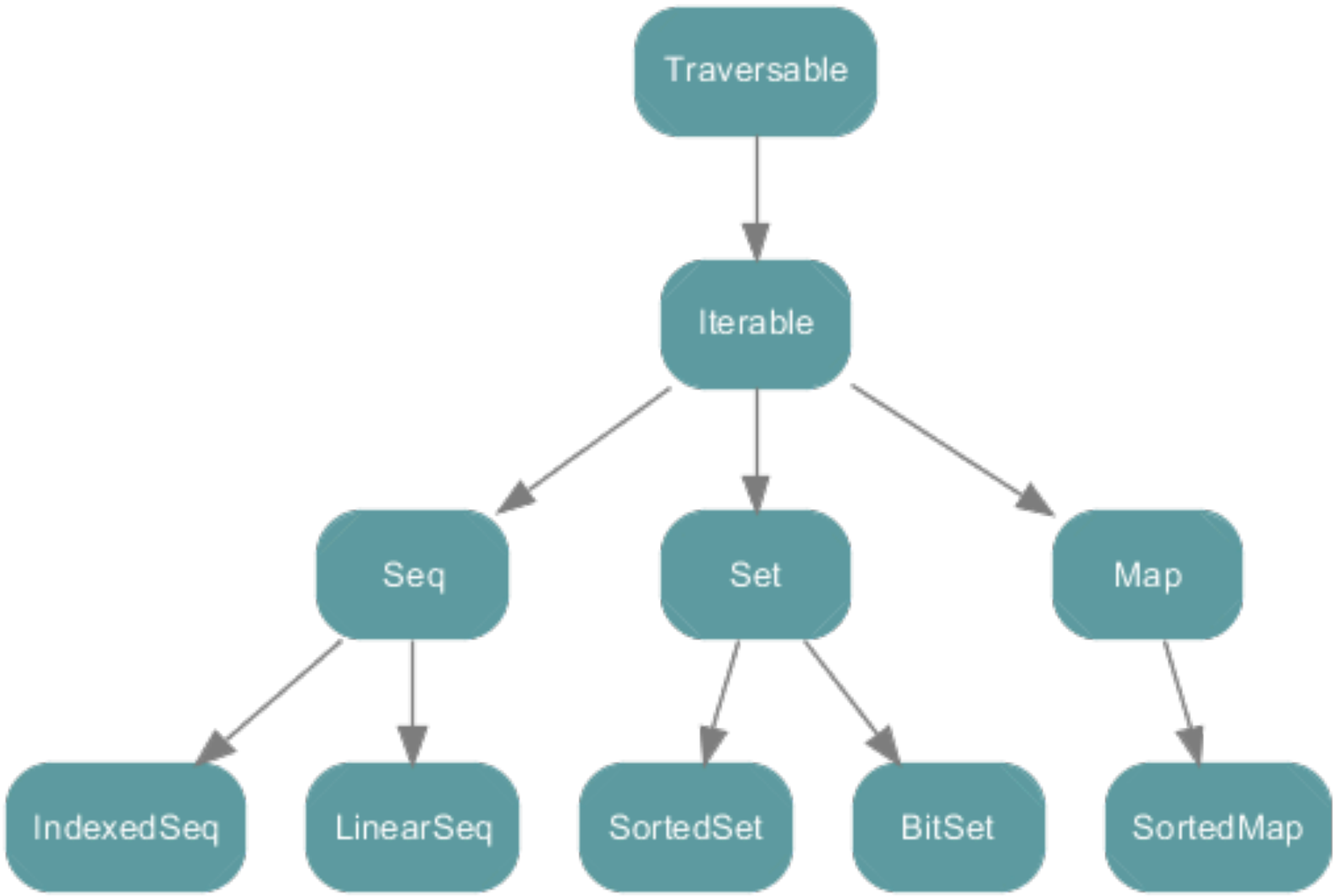
```
 def generateRandom : Int = {
    import scala.util.{Random}
    var rnd = new Random
    var n = rnd.nextInt()
    return n;
}
```

Types



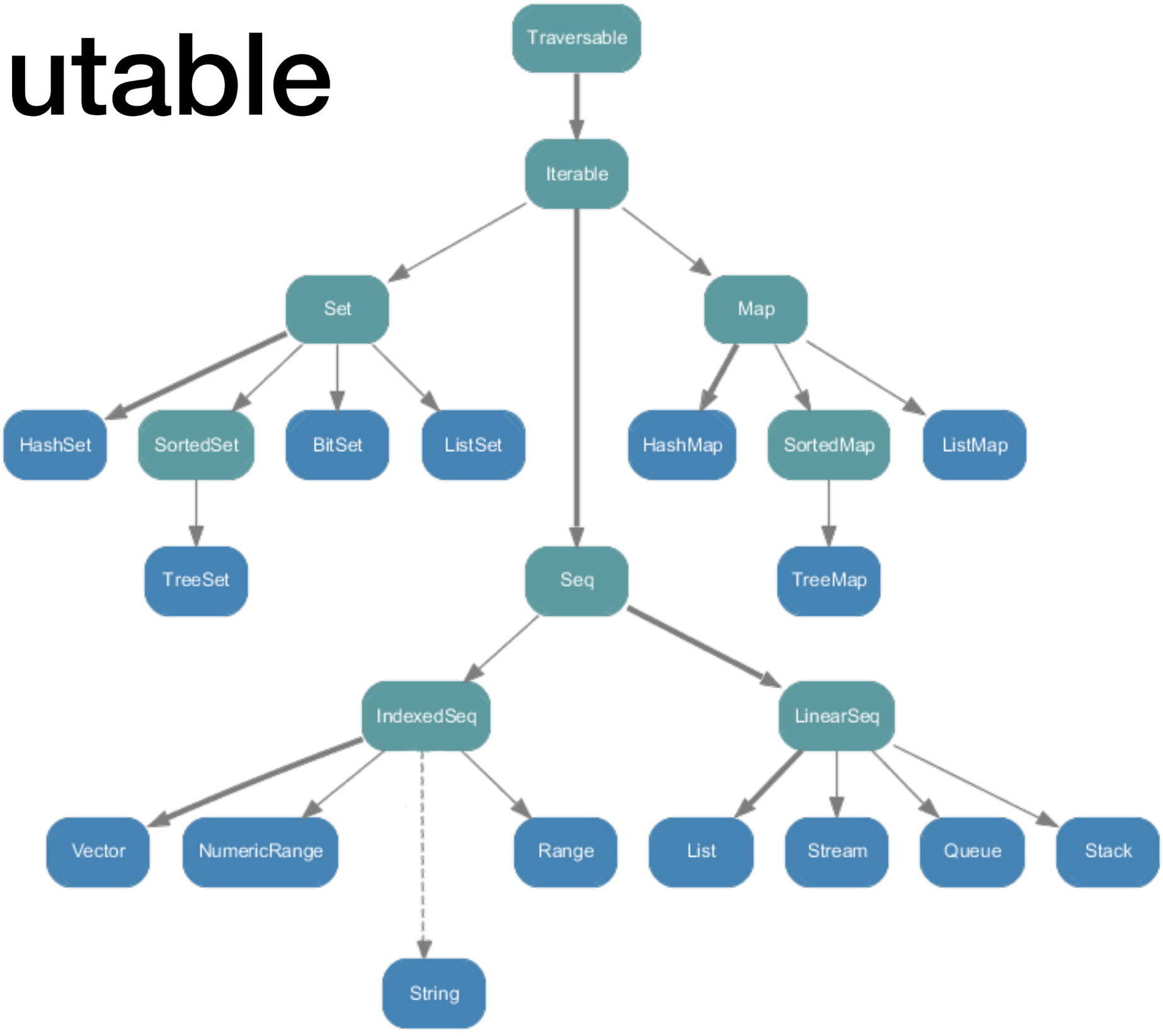
source: http://ktoso.github.io/scala-types-of-types/

Collections

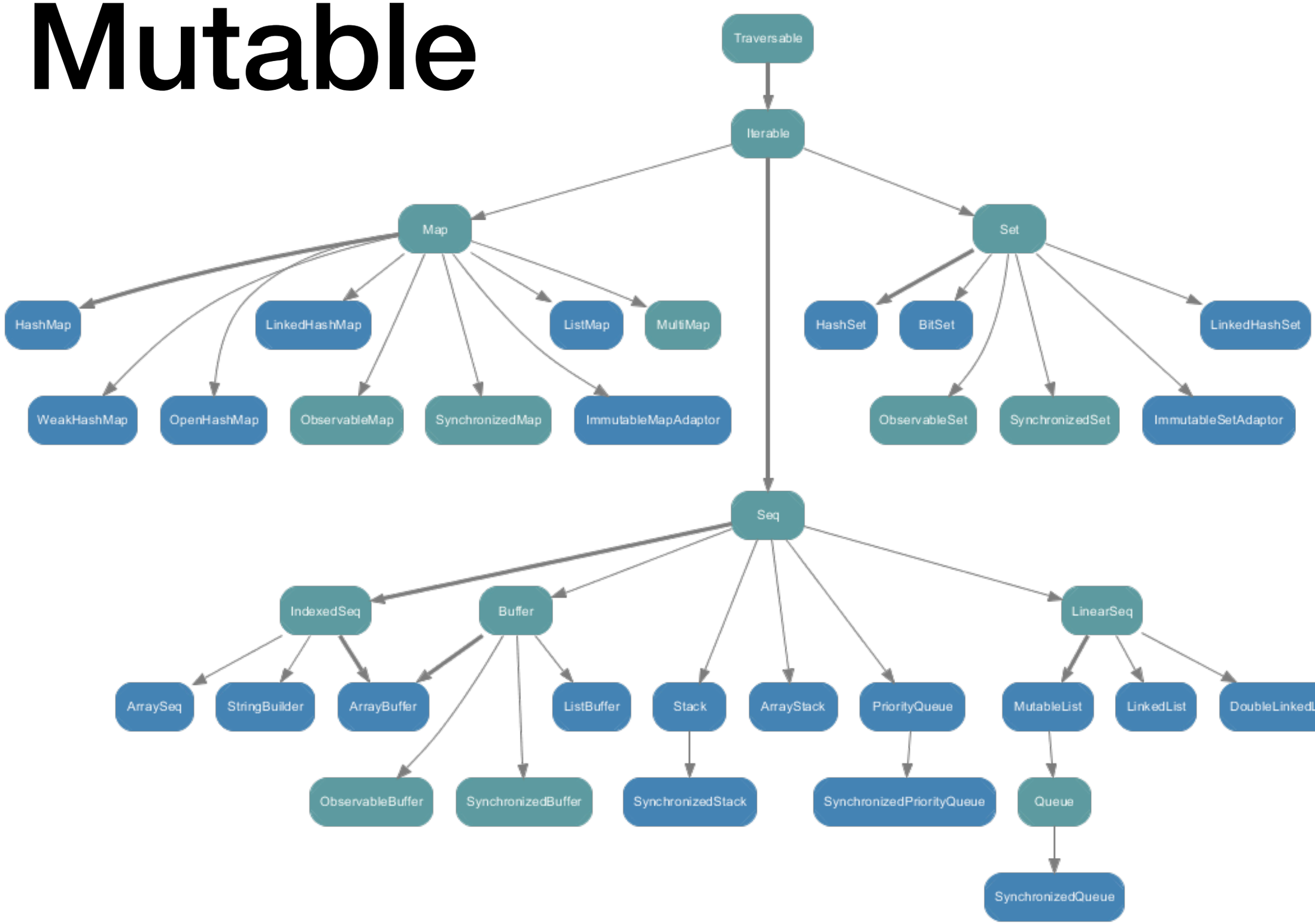


https://docs.scala-lang.org/overviews/collections/overview.html

Immutable


Traversable
Iterable
Set
Map
HashSet
SortedSet
BitSet
ListSet
HashMap
SortedMap
ListMap
TreeSet
Seq
TreeMap
IndexedSeq
LinearSeq
Vector
NumericRange
Range
List
Stream
Queue
Stack
String

Mutable


Traversable
Iterable
Map
Set
HashMap
LinkedHashMap
ListMap
MultiMap
HashSet
BitSet
LinkedHashSet
WeakHashMap
OpenHashMap
ObservableMap
SynchronizedMap
ImmutableMapAdaptor
ObservableSet
SynchronizedSet
ImmutableSetAdaptor
Seq
IndexedSeq
Buffer
LinearSeq
ArraySeq
StringBuilder
ArrayBuffer
ListBuffer
Stack
ArrayStack
PriorityQueue
MutableList
LinkedList
DoubleLinkedList
ObservableBuffer
SynchronizedBuffer
SynchronizedStack
SynchronizedPriorityQueue
Queue
SynchronizedQueue

Usage Pattern

```
Traversable(1, 2, 3)
Iterable("x", "y", "z")
Map("x" -> 24, "y" -> 25, "z" -> 26)
Set(Color.red, Color.green, Color.blue)
SortedSet("hello", "world")
Buffer(x, y, z)
IndexedSeq(1.0, 2.0)
LinearSeq(a, b, c)
```

Classes

Classes

- Scala support class concept
- Class with instance variables, methods, properties
- Java style getter/setters (get/set)
- Single Inheritance
- Polymorphism

No Static

- No Static Functions
- No Static class members
- Instead use object

Constructor

- One Primary constructor
- Multiple Auxiliary constructors
- **this** keyword for access current class members
- **super** keyword for base class

Primary Constructor

- Primary constructor defined in class definition
- Params in constructor become member variable by default
- Constructor params must have **var** or **val** to be member variable

```
class Appliance(var name: String, var price: Int) {
}

var a = new Appliance("Light", 100);
println(a.name, a.price)
```

Access specifier

- Public, by default
- No public keyword
- private keyword for inside class access
- protected keyword for child classes

Constructor with Access Specifier

- name is public member variable
- price is private member variable
- Discount is protected member variable

```
class Appliance(var name: String,
                private var price: Int,
                protected val discount: Int) {
}
```

Access Specifier within Class Definition

```
class Counter {
    private var count = 0;

    //private methods
    private def reset() {
        count = 0
    }

    //public methods
    def increment() { count += 1; }
    def current() = count
}

val counter = new Counter
counter.increment();
println(counter.current())
```

Auxiliary Constructor

- Secondary constructors
- defined with this keyword inside class
- Auxiliary constructor must start with call to previous constructor/default constructor

Auxiliary Constructor

```
class Appliance {
    private var name = "";
    private var price = 0;

    def this(name: String) {
        this(); //calls default cons
        this.name = name;
    }

    def this(name: String, price: Int) {
        this(name); //previous cons
        this.price = price;
    }
}

new Appliance() // default primary cons
new Appliance(“TV") // aux cons
new Appliance("TV", 20000) //aux cons
```

Properties

- Getter, Setter similar to Java
- def to define get property
- def with suffix _ for setter
- Not compatible with Java (get/set)Name pattern

Method vs Property

```
class Counter {
    private val count = 0;
    // () for method
    def increment() { count += 1 }
    //note, no () for property
    def current = count
}

//usage
val counter = new Counter()
counter.increment()  // () for method call
//note, no () for property
println(counter.count)
```

Getter/Setter

```
class Appliance {
    private var itemPrice = 0

    //public getter
    def price = itemPrice

    //public setter (property name with _=)
    def price_= (newPrice: Int) {
        if (newPrice >= 0)
            itemPrice = newPrice
    }
}

let appliance = new Appliance()
//calls getter
println(appliance.price)
//calls setter
appliance.price = 100
//do not set value -10 due to condition
appliance.price = -10
```

Java compatible getter/ setters

- per JavaBean specification
- to use @BeanProperty

```
import scala.reflect.BeanProperty
```

```
class Applicance {
    @BeanProperty var name: String = _
}
//@BeanProperty Generate 4 methods automatically
//Scala compatible getter/setter
name: String
name_ = (newValue: String) :Unit
//Java compatible
getName() : String
setName(newValue: String) : Unit
```

Inheritance

- Scala support inheritance
- Uses extends keyword
- override to override base class methods
- super to call base class cons/methods

Inheritance

```
class Appliance {}
class Light extends Appliance {}

var appliance:Appliance = new Light;
```

Inheritance

```
//Inheritance

class Base(var name: String) {
}

class Derived(name: String) extends Base(name) {
}

var d = new Derived("Scala")
println(d.name)
```

Type Checking

- Scala support run time type checking
- Throws exception on invalid casting
- keywords: isInstanceOf, classOf
- keywords: asInstanceOf for casting

Casting

```
class Appliance {}
class Light extends Appliance {}

var appliance:Appliance = new Light;
```

```
//doesn't work, downward casting
//var light:Light = appliance;
```

Class Type Check

- Helps to check exact concrete class used to create object

```
class Appliance {}
class Light extends Appliance {}
val applianceAppliance = new Light;

// prints true
println(appliance.getClass == classOf[Light])
// prints false
println(appliance.getClass == classOf[Appliance])
```

Instance Check

```
class Appliance {}
class Light extends Appliance {}
val appliance: Appliance = new Light;

// prints true for both
println(appliance.isInstanceOf[Light])
println(appliance.isInstanceOf[Appliance])
```

object

- Singleton instance by default

- To be used as alternative to Static members/functions in Java Classes

object constructor

- **object can have body constructor, which is executed very first use of object.**
- **Good for initialising values.**

```
object NumberFactory {
    // object constructor
    var id: Int = 0;
    id = 1000;
}
```

Object with members

```
object NumberFactory {
    var id: Int = 0;

    def next: Int = {
        id += 1;
        return id
    }

    id = 1000;
    println("inside factory", next);
}
```

when we use NumberFactory first time, it prints inside factory 1001

Companion Object

- A companion object is an object with the same name as a class or trait.

```
class Product (var name: String) {
    private var year = 0
    private var id = Product.seq;
    ...
}

object Product {
    var uniId: Int = 0

    def seq(): Int = {
        uniId += 1;
        uniId; //return uniId
    }
}
```

Companion Object

- Companion object can access Companion class private member

- Companion class can access companion object private members

- Companion object can create companion object working as factory

Companion Object apply()

- apply method of object useful to create objects of companion classes

```
object Product {
    def apply(name : String, year: Int): Product = {
        var product : Product = new Product(name)
        product.year = year
        return product
    }
}
To create object, call like

Product("Phone 1", 2017)
```

Trait

- is an interface [similar to Java + more]
- Trait can have default implementation
- Useful to make mixins, add additional functionalities
- Multiple traits implementation for class
- Order matters with multiple traits

Functions

- Function Multiple Args
- A function can accept variable number of arguments by using * in the declaration

Functions

```
def sumAll(name: String, args:Int*) : Int  =  {
      var count: Int = 0
      for {arg: Int <- args}
      count += arg;
      count
}
sumAll(“add”)

prints 0

sumAll(“add”, 10, 20, 30)

print 60
```

Function

- Below function sums number between range a and b recursively

```
def sum(a: Int, b: Int): Int {
    if (a > b) 0 else a + sum(a + b, b)
}
Call it like,

sum(1, 5)

prints 15
```

```
var sum2 = sum _

sum2(1, 5)
```

returns 15

but we can convert to lambda below code retuns the lambda

assign function
below statement throws exception, that wraps the function sum

When we assign function to variable, it automatically converts to lambda

Now defined a funciton square,

```
def square(a: Int): Int= a * a
```

Higher order function that square numbers and sum a to b

```
def sumSquare(f: Int => Int, a: Int, b: Int): Int{
    if (a > b) 0 else f(a) + sumSquare(f, a+1, b)
}
sumSquare(square, 1, 5) res27: Int = 55
```

Functions Inside functions

```
def sum(f: Int => Int, a: Int, b: Int): Int =
      | if (a > b) 0 else f(a) + sum(f, a + 1, b)
```

```
def sumInts(a: Int, b: Int): Int =
                        sum((x: Int) => x, a, b)

def sumSquareInts(a: Int, b: Int): Int =
              sum((x: Int) => x * x, a, b)
```

Default values

```
def printName(firstName:String = “Unknown”) {
      println(firstName)
}

printName()
```

Named arguments

```
def printName(firstName:String, lastName:String) {
    println(firstName, lastName)
}
```

```
printName(firstName="Krish", lastName="S")
printName( lastName="S", firstName="Krish")
```

Collection

```
var list: List[Int]= List(10,20,30,40,50)

returns values from 0th index

> list(0)
```

```
throws out of bound exception

> list(10)

> var option: Option[Int] = list.lift(20)
 => returns Option[Int] doesn't crash
 option.isDefined
 option.get
```

Scala Collection

- Mutable Collection
- Immutable Collection
- Generic Traits(/Interfaces)

List

- Immutable List
- List are seq
- Use ArrayBuffer/ListBuffer for Dynamic, mutable list

Map

- Key value pair
- Map is an interface – HashMap is a implementation

```
val states = Map(“AP” -> “Andra Pradesh”, “KA” ->
“Karnataka”, “TN” -> “Tamilnadu”)

//To get value

states(“AP”) //returns Andra Pradesh
states(“AK”) //throws exception NotSuchElement

// Fail safe get
states.get(“AP”) //returns Option[String]

// Fail safe get
states.get(“AK”) //returns Option[None]
```

```
// Iterates
for ((key,value) <- states)
    printf(“key: %s, value: %s\n”, key, value)

// Iterage using for Each (provides Tuple)
states.foreach ( state => state._1 + state._2)

// contains
states.contains(“AP”)
// returns all keys as set
states.keys
// returns values as collection
state.values
```

```
To Add item at index, returns new List

list = 60 :: list

Removing Items from list best done using filter

> val newList = list.filter(_ > 2)

Take first few Items

> list.take(5)
```

Mutable Map

```
import scala.collection.mutable.Map

var states =  Map(“AP” -> “Andra Pradesh”,
                  “KA” -> “Karnataka”,
                  “TN” -> “Tamilnadu”)

// add
states += (“MH” -> “Maharastra”)
states += (“ST1” -> “State 1”, “ST2” -> “State 2”)

// remove
states -= “ST1” states -= (“ST1”, “ST2”)

// update
states(“ST1”) = “State 2”
```

ListBuffer

```scala
import scala.collection.mutable.ListBuffer
val listBuffer = ListBuffer(10, 20, 30, 40, 50)

listBuffer -= 10
listBuffer += 60
listBuffer -= (20, 30)

//Remove By Index
listBuffer.remove(2)

//Use ++ for add/remove collections
listBuffer ++= Seq(10, 20, 30)

listBuffer ++= List(10, 20, 30)

//remove first matches
listBuffer --= Seq(10, 20, 30)
```

Tuples

- Immutable by default/always
- Special Type in Scala
- Created with parentheses (10, 20)
- Or using arrow
- val t = (10 -> 20)
- val numbers = (10, 20, 30)
- Note, 1 based index, not zero based accessible
- via numbers._1, numbers._2, & _3

Iterator

- Iterator walks through collection one after another
- Has two methods
- **hasNext** returns true if more element found
- **next** returns the actual value

```
var it: Iterator[Int] = Iterator.range(1, 10)

//below statement print true
println(it.hasNext)
//below statement print 1
it.next
//below statement print 2, the next element
it.next
//The below code prints rest of the items 3, to 99
while (it.hasNext) {
	println(it.next)
}
```

Closure keeps the value in scope after exit

```
var Seq = (start: Int) => {
    var id = start;
    var incr = () => {
      id += 1
      id
    }
    incr
}
```

```
var seq = Seq(100);
println(seq()); // 101
println(seq()); // 102

var seq2 = Seq(1000);
println(seq2()); // 1001
println(seq()); // 103
```

```
implicit def toInt(s: String): Int = {
  try {
    s.toInt
  } catch {
    case e: Exception => 0
  }
}
```

```
--
implicit def toInt(s: String): Option[Int] = {
  try {
    Some(s.toInt)
  } catch {
    case e: Exception => None
  }
}
```

```
import scala.util.{Try, Success, Failure}
def makeInt(s: String): Try[Int] = Try(s.trim.toInt)


//cange Option[Int] to zero
//To do: check value
implicit def optionToInt(v: Option[Int]): Int = if
(v.isDefined) v.get else 0
```

```
scala> var n1:Option[Int] = Some(20)
ll: Option[Int] = Some(20)

scala> var n2 : Option[Int] = None
oo: Option[Int] = None

scala> n1 + n2
res: Int = 20
```

Implicits

- Helps to do conversion to one to another type
- Compiler automatically lookup for converters
- To be enabled with
- import scala.language.implicitConversions

- Or with compiler option
- implicit def stringToInt(str: String) = str.toInt
- implicit def doubleToInt(d: Double) = d.toInt
- val k:Int = “100” calls stringToInt
- val x:Int = 3.14 calls doubleToInt
- Having two implicits of given paramter would produce error
-

Option

- Represent valid instance of Some scala class
- Represent instance of None

```
val opt:Option[Int] = Some(10)
if (opt.isDefined && opt.get == 10)
    ...
```

Default Values

- default initial value
- Useful to initialize default values inside class
- Rule: Cannot be used with in functions

```
class Product {
    var id: Int = _ //initialize with 0
    var name: String = _ //initialize with null
}
```