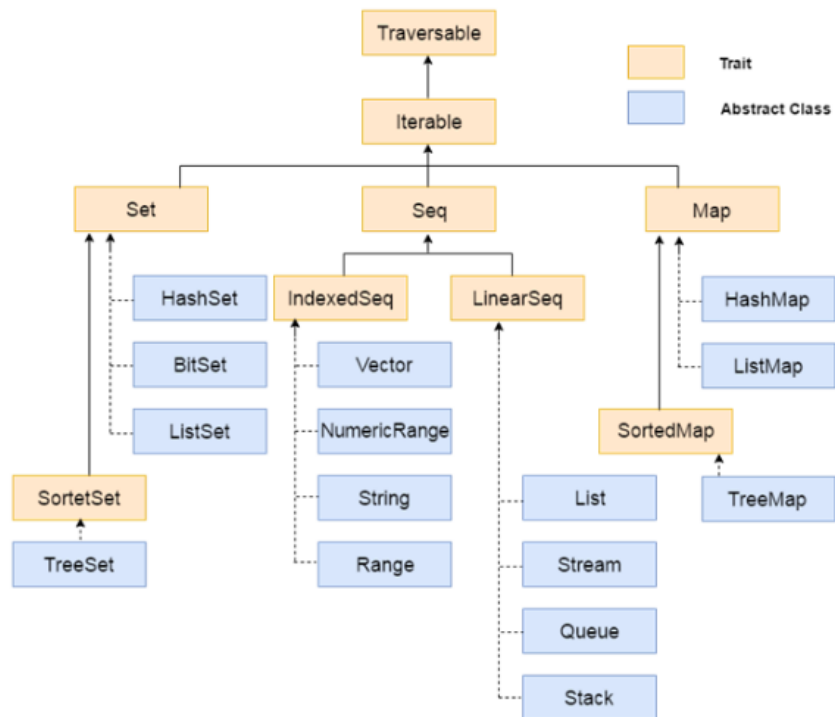


<b>Scala Features</b>	<b>1</b>
<b>Hello world</b>	<b>2</b>
<b>Values</b>	<b>3</b>
<b>Variables</b>	<b>3</b>
<b>Blocks</b>	<b>3</b>
<b>Basic Types</b>	<b>4</b>
<b>String Interpolation</b>	<b>4</b>
<b>Array</b>	<b>5</b>
<b>List</b>	<b>5</b>
<b>Map</b>	<b>6</b>
<b>Methods</b>	<b>7</b>
<b>Nested Methods</b>	<b>8</b>
<b>Function Literals</b>	<b>9</b>
<b>Method Overload</b>	<b>9</b>
<b>Class</b>	<b>10</b>
<b>Trait</b>	<b>11</b>
<b>Protected/Private</b>	<b>11</b>
<b>Generic Classes</b>	<b>12</b>
<b>If/Else</b>	<b>12</b>
<b>Loop</b>	<b>12</b>
<b>Pattern matching</b>	<b>14</b>
<b>Exceptions</b>	<b>14</b>

## Scala Features

- Object-oriented
- Created for high performance
- You don't need to specify the types of variables

- You don't need to specify the return types for methods
- Run on the JVM
- Can execute Java code
- Lazy evaluation (call-by-need)
- Has interfaces like Java (trait keyword)
- Rich collection library



## Hello world

```
object HelloWorld {
  def main(args: Array[String]) {
    println("Hello, World!")
  }
}
```

- The method name is the main
- Takes a String array as a parameter
- println is used to print the content

# Values

```
val v1: String = "foo"  
val v2 = "bar"
```

- Values are immutable, can't be changed!
- The type of v1 is String
- The type of v2 is String (type inference)
- Recommended if you don't change the value during execution.

# Variables

```
var counter = 0  
counter = counter + 5
```

- Variables are mutable, can be changed
- You can adjust the variables within the Scala code blocks
- The type of counter is integer.

```
var color = "red"  
color = 5 // Invalid
```

- You can not assign a invalid type to the variable.

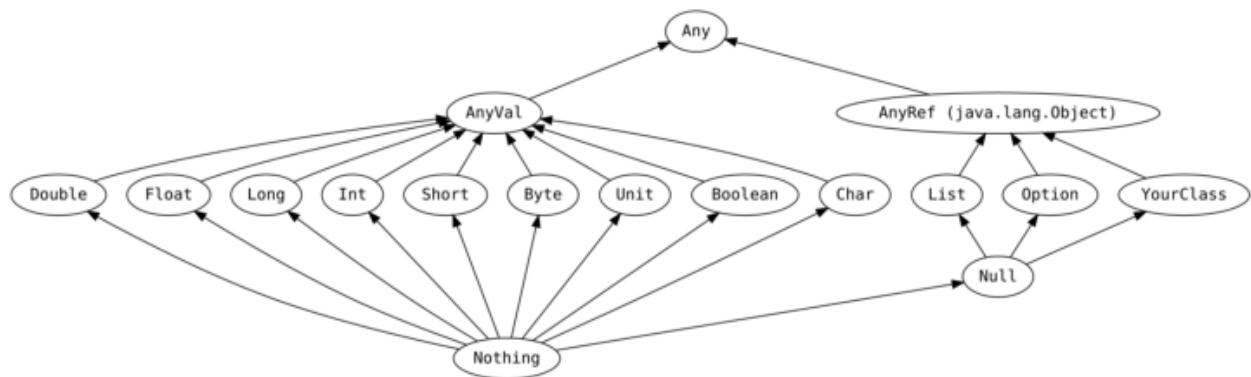
# Blocks

```
println(7) // Prints 7  
  
println {  
  val i = 5  
  i + 2  
} // Prints 7
```

- prints the latest statement in the block

# Basic Types

- Byte
- Short
- Int
- Long
- Float
- Double
- Char
- Boolean



- Basic types inherits from AnyVal
- Objects inherits from AnyRef

## String Interpolation

```
val name = "Serkan"
println(s"Name $name")
```

- You can embed the String variable for logging reasons

# Array

```
val a = new Array[Int](2)
a(0) = 3
a(1) = 5

println(s"Length a : ${a.length}")
println(s"First : ${a(0)}")
println(s"Second : ${a(1)}")

val b = new Array[String](2)
b(0) = "foo"
b(1) = "foo1"

println(s"Length b:${b.length}")
println(s"First : ${b(0)}")
```

- Array is defined with this format `val a = new Array[Type](length)`
- You can assign values to each index `a(index) = value`
- Even if the array is defined as a val, you can update the value of indexes

```
var array1 = Array(1,2,3)
println(array1(0))

var array2 = Array.ofDim[Int](3, 3)
array2(0)(0) = 4

println(array2(0)(0))
```

- You can define multidimensional array : `Array.ofDim[Type](Dimension 1, Dimension2)`
- Array can be init in the beginning of definition : `Array(val1, val2, val3)`

# List

```
val list = List(5, 2)
list(0) = 5 // Compilation error
```

- List is immutable. In case you need to change, it gives a compile error

# Map

```
val map = Map("06"->"Ankara", "34"->"Istanbul")
println(map("06"))

val map1 = map + ("35"->"Izmir")
println(s"35 : ${map1("35")}")
```

- You can use Map as a key value collection
- The elements can not be modified in map

```
map1("35") = "izmir"
```

- Error in this blog
- You need to define like this

```
val map = scala.collection.mutable.Map("06"->"Ankara", "34"->"Istanbul")
map("06") = "ankara"

println(map("06"))
```

- The type should be `scala.collection.mutable.Map`

## Methods

```
def add(x: Int, y: Int): Int = {  
  x + y  
}
```

- The method name is add
- The parameters are x,y
- Return type is Int
- The last parameter is the return type

```
def increment(x: Int, y: Int): (Int, Int) = {  
  (x + 1, y + 1)  
}
```

- You can return multiple values

```
def variablesArguments(args: Int*): Int = {  
  var n = 0  
  for (arg <- args) {  
    n += arg  
  }  
  n  
}
```

- You can add multiple parameters that can be repeated.

```
def main(args: Array[String]): Unit = {  
  multiply(y=3)  
}  
  
def multiply(x: Int = 1, y: Int = 2): Int = {
```

```
println(x*y)
x
}
```

- You can define default value

## Nested Methods

```
def main(args: Array[String]): Unit = {
  println(addThreeNumber(1,2,3))
}

def addThreeNumber(first: Int, second: Int, third: Int): Int = {
  first + addTwoNumber(second,third)
}

def addTwoNumber(second: Int, third: Int): Int = {
  second + third
}
```

- You can call another method within the method

```
def main(args: Array[String]): Unit = {
  println(addThreeNumber(1,2,3))
}

def addThreeNumber(first: Int, second: Int, third: Int): Int = {

  def addTwoNumber(second: Int, third: Int): Int = {
    second + third
  }
}
```



```
}  
first + addTwoNumber(second,third)  
}
```

- Definition can also be nested

## Function Literals

```
val increment = (x: Int) => x + 1  
println(increment(5))
```

- `x+1` is a function that will be executed
- `(x: Int)` represents the parameter

## Method Overload

```
def printVal() = {  
    println("Print without value")  
}  
  
def printVal(i: Int) = {  
    println("Print with value : " + i)  
}
```

- Method name is same, however you can use different parameters

# Class

```
class Point(var x: Int, var y: Int) {  
  def move(dx: Int, dy: Int): Unit = {  
    x += dx  
    y += dy  
    println(s"$x $y")  
  }  
}
```

- Defined with class keyword
- The first line shows the constructor like Java
- Create with new keyword

```
object CallPoint {  
  def main(args: Array[String]): Unit = {  
    val point = new Point(2,3);  
    point.move(1,1);  
  }  
}
```

- You can instantiate with `val point = new Point(2,3);`

However, an object is a **singleton** in Scala. (one instance in the memory)

# Trait

```
trait Car {  
  val color: String  
  def drive(): Unit  
}
```

- Used to define an interface

```
class CarClass extends Car {  
  val color: String = "red"  
  
  def drive(): Unit = {  
    println(s"Drive $color car")  
  }  
}
```

- **extends** is used to implements the method and values from interface

```
object MainCar {  
  def main(args: Array[String]): Unit = {  
  
    var car = new CarClass()  
    car.drive()  
  }  
}
```

# Protected/Private

- private : members are accessible in the current class
- protected : members are only accessible from sub-class

# Generic Classes

```
class GenericClass [A]{  
  def print(x: A, y: A): Unit = {  
    println(s"x=$x,y=$y")  
  }  
}
```

- You can use any type for A

```
val generics = new GenericClass[Int];  
generics.print(2,3)
```

# If/Else

```
val x = 2  
if (x == 1) {  
  println(s"x==1")  
} else if (x<1) {  
  println(s"x<1")  
} else {  
  println(s"x>1")  
}
```

# Loop

```
for (a <- 0 to 10) {  
  println(a)  
}
```

- Loops 0 to 10

```
for (a <- 0 until 10) {  
  println(a)  
}
```

- Loops 0 to 9

```
for (a <- 0 until 2; b <- 0 to 2) {  
  print(a,b )  
}
```

- Iterates all possible values

```
val list = List(5, 7, 3, 0, 10, 6, 1)  
for (elem <- list if elem % 2 == 0) {  
  println(elem)  
}
```

- Iterates over the list

```
val sub = for (elem <- list if elem % 2 == 0) yield elem
```

- yield creates a sub list

# Pattern matching

```
def matchA(i: Int): String = {  
  i match {  
    case 1 => return "one"  
    case 2 => return "two"  
    case _ => return "something else"  
  }  
}  
  
print(matchA(1))
```

- Same with switch-case in Java

# Exceptions

```
import java.io.FileReader  
  
object Exceptions {  
  
  def main(args: Array[String]): Unit = {  
  
    try {  
      val n = new FileReader("input.txt").read()  
      println(s"Success: $n")  
    } catch {  
      case e: Exception =>  
        e.printStackTrace  
    }  
  
  }  
  
}
```

# Samples

- Operand
- DecisionMaking
- NestedIf
- ForLoop
- WhileLoop
- DoWhile
- ArrayTest
- SmartPhone
- TraitMain
- Throw
- ReadFile

## Mutable and immutable variables

Variable Type	Description
<code>val</code>	Creates an <i>immutable</i> variable—like <code>final</code> in Java. You should always create a variable with <code>val</code> , unless there's a reason you need a mutable variable.
<code>var</code>	Creates a <i>mutable</i> variable, and should only be used when a variable's contents will change over time.

*// immutable*

**val** a = 0

*// mutable*

**var** b = 1

```
object Variables {  
  def main(args: Array[String]): Unit = {  
    // immutable
```

```
val a = 0
// mutable
var b = 1

println(a)
println(b)

a = 2
b = 3

println(a)
println(b)
}
```

---

[/Users/serkans/IdeaProjects/untitled3/src/main/scala/Variables.scala:11:7](#)  
reassignment to val  
a = 2

## Types of Variables

Explicit and implicit

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

Scala Numeric 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
```

Scala evaluates the type of value



## Scala 2 and 3

```
val i = 123    // defaults to Int
val j = 1.0    // defaults to Double
```

## Scala Big Values

```
var a = BigInt(1_234_567_890_987_654_321L)
var b = BigDecimal(123_456.789)
```

## Scala String and char values

```
val name = "Bill"    // String
val c = 'a'          // Char
```

Scala has a feature for interpolation

```
val firstName = "John"
val mi = 'C'
val lastName = "Doe"
println(s"Name: $firstName $mi $lastName")
```

# Control Structures

Scala has the control structures you find in other programming languages, and also has powerful **for** expressions and **match** expressions:

- if/else**
- for** loops and expressions
- match** expressions (similar with switch case)
- while** loops
- try/catch**

(Finish)