# **Object Meets Function**

Let's Mutate The State - Imperative

Steven Lolong

steven.lolong(at)uni-tuebingen.de

Programming Language Research Group
Tuebingen University

Oct. 30, 2023



Steven Lolong UT INF3213 Oct. 30, 2023

# Summary - Imperative Programming

- Variable
- Type
- Input and Output
  - Operator
- Control Structure
  - if or if-else

- while-do
- for-do
- for-do with Guards
- for-yield
- match
- try-catch-finally



Steven Lolong UT Oct. 30, 2023 2/39

## Variable

Variable

Mutable and Immutable

- Mutable = define once and change the value anytime.var name: type = value
- Immutable = define once and bind the value once.
  val name: type = value

var / val keyword

name variable name, ex: valA, heightOfSquare, etc.

type variable type, ex. String, Int, Boolean, etc.

value the content (expression) of variable



Steven Lolong UT INF3213 Oct. 30, 2023

## Variable

#### Example

```
// Mutable
var vLength : Int = 5
vLength = 7 // re-assign
// type inference
var vName = "Hayabusa"
```

#### Listing: Mutable

```
// Immutable
val vWidth: Int = 80
// type inference
val vCenter = vWidth / 2
// error when re-assign
// vWidth = 100
```

#### Listing: Immutable



Steven Lolong UT INF3213 Oct. 30, 2023 4 / 39

# Type

Primitive (Built-in Types)

Boolean true or false

Byte 8-bit signed integer

Short 16-bit signed integer

Int 32-bit integer

Long 64-bit integer

Fload 32-bit float

Double 64-bit float

Char 16-bit unsigned Unicode character

String a sequence of Char

Unit a single value ()

Any Top-level type

Nothing Bottom-level type of value

Null Bottom-level type of reference

EREPHARD KARIS Universität TÜBINGEN

Steven Lolong UT Oct. 30, 2023

# **Build-in Types**

Hierarchy

see: https://docs.scala-lang.org/tour/unified-types.html

```
var tInt : Int = 8
// error
// tInt = "eight"
var tString : String = "eight"
//error
// tString = 8
var tAny : Any = "eight"
tAny = 8
```

Listing: Type Hierarchy



Steven Lolong UT Oct. 30, 2023 6/39

## Input and Output (I/O)

```
// put to stdout (standard output)
 println(vLength)
 // import library of standard input
 import scala.io.StdIn._
 // get from stdin (standard input)
 val fName = readLine()
8 println("Name: " + fName)
```

Listing: Input and Output



Steven Lolong UT Oct. 30, 2023

```
+ | - | * | / % arithmetic: add, subtract, multiply, divide, modules

==! =><>=<= relational:
    && || ! logical: and, or, not
    & | bitwise: and, or

<<>>>>> bitwise: left shift, right shift,

= + = - = * = / = % =>>=<<== = | = 32-bit IEEE 754 single-precision float
```



if or if-else

#### if-else

if expression1: Boolean then expression2: A else expression3: A

#### if

if expression1: Boolean then expression2: A

expression1 evaluate to Boolean

expression2 and expression3 sequence of expression that can evaluate to any value



if or if-else

```
import scala.io.StdIn._
 def numberChecking() : Unit =
     val inputNum = readLine("Input a number: ")
     if inputNum.toInt % 2 == 0 then
5
          println("Your number is even")
6
     else
          println("Your number is odd")
8
```

Listing: example of if-else



Steven Lolong UT Oct. 30, 2023 10 / 39

while-do

```
while
    expression1 : Boolean
 do
3
    expression2 : A
```

Listing: while-do

expression1 evaluate to Boolean expression2 sequence of expression that can evaluate to any value



Steven Lolong UT Oct. 30, 2023

while-do

```
def print1to20() : Unit =
    var number = 1
    while
3
      number \leq 20
    do
5
      println(number)
6
      number += 1
```

Listing: example of while-do



Steven Lolong UT Oct. 30, 2023 12/39

for-do

```
for expression1 do
    expression2
3
    expressionN
```

Listing: for-do

expression1 evaluate to Boolean expression2 sequence of expression that can evaluate to any value



Steven Lolong UT Oct. 30, 2023 13/39

for-do

```
def triangleStar(): Unit =
   for(a <- 1 to 5) do
      for(b <- 1 to 5) do
3
        if(b \le a) then
4
          print("*")
5
      println("")
6
```

Listing: example for-do



Steven Lolong UT Oct. 30, 2023 14/39

for-do with Guards

#### Usually in programming language to filter in iteration, it will do like

```
for(i <- 1 to 10) do
   if(i % 2 == 0) then
   println(i)</pre>
```

Listing: for-do with filter

Is any other way in Scala?



Steven Lolong UT INF3213 Oct. 30, 2023 15 / 39

for-do with Guards

Scala's "for comprehension" is equivalent to Haskell's "do" notation, and it is nothing more than a syntactic sugar for composition of multiple monadic operations. ... [https://docs.scala-lang.org/tutorials/FAQ/yield.html]



Steven Lolong UT Oct. 30, 2023

for-do with Guards

```
for
forExp1
...
forExpN

forExp1:
Boolean

ifExpN: Boolean

do
expBody
```

Listing: for-do with guards format

forExp1 ... forExpN expressions for iteration, it can hold multiple expressions

ifExp1 ... ifExpN expression for filtering using *if*, it can hold multiple filtering

expBody body of expression



Steven Lolong UT INF3213 Oct. 30, 2023 17 / 39

for-do with Guards

```
for(i <- 1 to 10) do
   if(i % 2 == 0) then
   println(i)</pre>
```

Listing: filter inside the body

```
for

i <- 1 to 10

if i % 2 == 0

do

println(i)
```

Listing: filter inside the for-expression



18/39

Steven Lolong UT INF3213 Oct. 30, 2023

for-do with Guards

```
for

i <- 1 to 10

if i % 2 == 0

if i > 6

do

println(i)
```

Listing: for-do with guards example1

More than one filter expression execution of filter will do sequentially from the first to the last expression.

How about more than one iteration's expression?



 Steven Lolong UT
 INF3213
 Oct. 30, 2023
 1

for-do with Guards

#### How about this?

```
for
      <- 1 to 5
     <- i to 5
    if j > i
 do
5
    println(i)
6
```

Listing: for-do with guards example2



Steven Lolong UT Oct. 30, 2023 20/39

for-do comprehensions

#### for comprehensions

a syntactic sugar for composition of multiple monadic operation.

```
for
a <- exp1
b <- exp2
c <- exp3
do
expBody
```

Listing: for-do syntactic sugar

```
exp1.foreach(a => exp2.foreach(b => exp3.foreach(c => expBody)))
```

Listing: for-do comprehension



Steven Lolong UT INF3213 Oct. 30, 2023 21 / 39

#### for-do comprehensions

```
for
    x <- 1 to 10
do
    println(x)

for
    x <- List("Hello", "World")
do
    println(x)</pre>
```

Listing: for-do syntactic sugar - example

```
for
    x <- List("Hello", "World")

do
    println(x)

(1 to 10).foreach(x => println(x))
```

Listing: for-do comprehension - example

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

22/39

 Steven Lolong UT
 INF3213
 Oct. 30, 2023

#### for-do comprehensions

```
for
  x <- 1 to 10
do
  for
    v <- List("Hello", "World")</pre>
  do
    println(s"x = $x, y = $y")
```

#### Listing: for-do nested - example

```
for
  x <- 1 to 10
  y <- List("Hello", "World")
do
  println(s"x = $x, y = $y")
```

#### Listing: for-do nested - example

```
(1 to 10).foreach(x => List("Hello", "World").foreach(y =>
   println(s"x = $x, y = $y")))
```

UNIVERSITAT

23 / 39

Listing: for-do nested comprehension - example TÜBINGEN

for-do comprehensions

Is any task that can do using nested-for but can't use for-comprehension? Give a simple example!



Steven Lolong UT INF3213 Oct. 30, 2023 24 / 39

for-do comprehensions

```
for
    x <- 1 to 10
do
    println(x)
for
    y <- List("Hello", "World")
do
    println(y)</pre>
```

Listing: for-do nested with a specific task per iteration



Steven Lolong UT INF3213 Oct. 30, 2023 25 / 39

for-yield

Looping for-do with Output Buffer-yield



for-yield

For each iteration of your for loop, yield generates a value which will be remembered. It's like the for loop has a buffer you can't see, and for each iteration of your for-loop, another item is added to that buffer. When your for-loop finishes running, it will return this collection of all the yielded values. The type of the collection that is returned is the same type that you were iterating over, so a Map yields a Map, a List yields a List, and so on.

- Alexander -



Steven Lolong UT INF3213 Oct. 30, 2023 27

#### for-yield

```
for for Exp1 ... for ExpN if Exp1 : Boolean ... if ExpN : Boolean yield yBody
```

Listing: for-yield

forExp1 ... forExpN expressions for iteration, it can hold multiple expressions.

ifExp1 ... ifExpN expression for filtering using *if*, it can hold multiple filtering.

yBody body of expression.



for-yield

```
var yA = for i <- 1 to 10 yield i
  def foo(n: Int, v: Int) =
    for i <- 0 until n
    j \leftarrow 0 until n \text{ if } i + j == v
6
    yield i+j
7
  var yB =
    for
9
       i < -1 to 10
       y <- List("Hello", "World")
    vield
       println("yield")
13
       (i,y)
```

Listing: for-yield example



match

Scala has a match expression, which in its most basic use is like a Java switch statement. -scala3 book

```
Matchable match
case val1 => exp1 : T
case val2 => exp2 : T
case valn => expn : T
case _ => exp_ : T
```

Listing: match



Steven Lolong UT Oct. 30, 2023 30 / 39

match

```
def singleDigitToText (d: Int) : String = d match
   case 1 => "one"
   case 2 => "two"
   case 3 => "three"
   // ... to 9
5
   case _ => "unknown"
6
```

Listing: matching on value - ex-1



Steven Lolong UT Oct. 30, 2023 31/39

match

```
def checkParamType (p: Any) : String = p match
     case (_: Int) => "Integer"
     case (_: Char) => "Character"
     case (_: String) => "String"
     case ( : Double) => "Double"
     // ... to all possible types
6
     case _ => "unknown"
```

Listing: matching on type - ex-2



Steven Lolong UT Oct. 30, 2023 32 / 39

match

```
def checkParamType2 (p: Any) : (Any, String) = p match
     case (v: Int) => (v, "Integer")
     case (v: Char) => (v, "Character")
     case (v: String) => (v, "String")
     case (v: Double) => (v, "Double")
     // ... to all possible types
6
     case _ => (p, "unknown")
```

Listing: matching on type - ex-3



33 / 39

Steven Lolong UT Oct. 30, 2023

for-yield

Listing: matching with guards - ex-4



match

Listing: matching on list constractor - ex-5



match

```
// matching on object
case class Person(id: Int)

def mapIdToName(p:Person) : String = p match
case Person(7) => "James Bond"
case Person(6) => "Hitman"
case _ => "Unknown id"

mapIdToName(new Person(7))
```

Listing: matching on object - ex-6



Steven Lolong UT INF3213 Oct. 30, 2023 36 / 39

match

```
// Mini Calculator Language
2 sealed trait Exp
case class Num(n: Int) extends Exp
| case class Add(lhs: Exp, rhs: Exp) extends Exp
case class Mul(lhs: Exp, rhs: Exp) extends Exp
6
 def eval (exp: Exp) : Int = exp match
    case Num(n) => n
    case Add(lhs, rhs) => eval(lhs)+ eval(rhs)
    case Mul(lhs, rhs) => eval(lhs) * eval(rhs)
10
    case _ => sys.error("Unknown expression")
 eval(Num(3))
 eval(Add(Num(7), Num(8)))
```

Listing: matching on object - ex-7



37/39

 Steven Lolong UT
 INF3213
 Oct. 30, 2023

try-catch-finally

Scala's try/catch/finally control structure lets you catch exceptions. It's similar to Java, but its syntax is consistent with match expressions.

—scala3 book

```
try
expression
catch
case throwVal1: Exception1 => expression1
case throwVal2: Exception2 => expression2
finally
expression4
```

Listing: try-catch-finally



Steven Lolong UT INF3213 Oct. 30, 2023 38 / 39

try-catch-finally

```
def divX(vD : Double, divD: Double) = divD match
    case 0 => throw new ArithmeticException("Error! divider
      is zero")
    case _ => vD / divD
  def testDiv =
    try
      divX(3,0)
    catch
8
      case ex: ArithmeticException => ex.printStackTrace()
9
    finally
10
      println("the last step")
```

Listing: try-catch-finally example



Steven Lolong UT Oct. 30, 2023 39 / 39