TOPIC 8.

CONTROL ABSTRACTION
CURRYING
BY-NAME PARAMETERS

BJÖRN A. JOHNSSON

CONTROL STRUCTURES

- SCALA ONLY HAS 5 BUILT-IN CONTROL STRUCTURES:
  - `if`, `while`, `for`, `try`, `match` AND FUNCTION CALLS.
- INSTEAD OF PREDEFINING MANY CONTROL STRUCTURES, SCALA GIVES THE ABILITY TO CREATE NEW ONES.
- MADE POSSIBLE BY FUNCTION LITERALS.

RECAP

```
scala> val printer = (toPrint: String) => println(toPrint)
printer: String => Unit = <function1>

scala> printer(”Hello, world!”)
Hello, world!
```

CONTROL ABSTRACTION

- HIGHER-ORDER-FUNCTIONS – FUNCTIONS THAT TAKE FUNCTIONS AS PARAMETERS
- ENABLES THE CREATION OF CONTROL ABSTRACTIONS
- BENEFITS:
  - REDUCES CODE DUPLICATION
  - SIMPLIFIES CLIENT CODE

CONTROL ABSTRACTION

EXAMPLE

```
object FileMatcher {
  private def filesHere = (new java.io.File(".")).listFiles

  def filesMatching(query: String, matcher: (String, String) => Boolean) =
    for (file <- filesHere; if matcher(file.getName, query))
      yield file
}
```

CONTROL ABSTRACTION

EXAMPLE

```
object FileMatcher {
  private def filesHere = (new java.io.File(".")).listFiles

  def filesMatching(query: String, matcher: (String, String) => Boolean) =
    for (file <- filesHere; if matcher(file.getName, query))
      yield file

  def filesEnding(query: String) =
    filesMatching(query, _.endsWith(_))
}
```

CONTROL ABSTRACTION

EXAMPLE

```
object FileMatcher {
  private def filesHere = (new java.io.File(".")).listFiles

  def filesMatching(query: String, matcher: (String, String) => Boolean) =
    for (file <- filesHere; if matcher(file.getName, query))
      yield file

  def filesEnding(query: String) =
    filesMatching(query, _.endsWith(_))

  def filesContaining(query: String) =
    filesMatching(query, _.contains(_))
}
```

CONTROL ABSTRACTION

EXAMPLE

```
object FileMatcher {
  private def filesHere = (new java.io.File(".")).listFiles

  def filesMatching(query: String, matcher: (String, String) => Boolean) =
    for (file <- filesHere; if matcher(file.getName, query))
      yield file

  def filesEnding(query: String) =
    filesMatching(query, _.endsWith(_))

  def filesContaining(query: String) =
    filesMatching(query, _.contains(_))

  // ...more methods with omitted...
}
```

CURRYING

- A FUNCTIONAL PROGRAMMING TECHINQUE
- "A CURRIED FUNCTION IS APPLIED TO MULTIPLE ARGUMENT LISTS, INSTEAD OF JUST ONE."

"PLAIN OLD" FUNCTION

```
scala> def plainOldSum(x: Int, y: Int) = x + y
plainOldSum: (x: Int,y: Int)Int
```

CURRYING

- A FUNCTIONAL PROGRAMMING TECHINQUE
- "A CURRIED FUNCTION IS APPLIED TO MULTIPLE ARGUMENT LISTS, INSTEAD OF JUST ONE."

"PLAIN OLD" FUNCTION

```
scala> def plainOldSum(x: Int, y: Int) = x + y
plainOldSum: (x: Int,y: Int)Int

scala> plainOldSum(1, 2)
res1: Int = 3
```

CURRYING

- A FUNCTIONAL PROGRAMMING TECHINQUE
- "A CURRIED FUNCTION IS APPLIED TO MULTIPLE ARGUMENT LISTS, INSTEAD OF JUST ONE."

"PLAIN OLD" FUNCTION

```
scala> def plainOldSum(x: Int, y: Int) = x + y
plainOldSum: (x: Int,y: Int)Int

scala> plainOldSum(1, 2)
res1: Int = 3
```

CURRIED FUNCTION

```
scala> def curriedSum(x: Int)(y: Int) = x + y
curriedSum: (x: Int)(y: Int)Int
```

CURRYING

- A FUNCTIONAL PROGRAMMING TECHINQUE
- "A CURRIED FUNCTION IS APPLIED TO MULTIPLE ARGUMENT LISTS, INSTEAD OF JUST ONE."

"PLAIN OLD" FUNCTION

```
scala> def plainOldSum(x: Int, y: Int) = x + y
plainOldSum: (x: Int,y: Int)Int

scala> plainOldSum(1, 2)
res1: Int = 3
```

CURRIED FUNCTION

```
scala> def curriedSum(x: Int)(y: Int) = x + y
curriedSum: (x: Int)(y: Int)Int

scala> curriedSum(1)(2)
res1: Int = 3
```

TRICK: INCREASE ”BUILT-IN” FEEL

- CURLY BRACKETS CAN BE USED INSTEAD OF PARENTHESES WHEN INVOKING A METHOD THAT TAKES EXACLY ONE ARGUMENT.

```
scala> println (”Hello, world!”)
Hello, world!
```

```
scala> println { ”Hello, world!” }
Hello, world!
```

CURRYING USING CURLY BRACKETS

```
scala> def twice(x: Double, op: Double => Double) = op(op(x))
twice: (x: Double, op: Double => Double)Double
```

CURRYING USING CURLY BRACKETS

```
scala> def twice(x: Double, op: Double => Double) = op(op(x))
twice: (x: Double, op: Double => Double)Double

scala> twice(5, _ + 0.2)    // Looks like normal invokation
res1: Double = 5.4
```

CURRYING USING CURLY BRACKETS

```
scala> def twice(x: Double, op: Double => Double) = op(op(x))
twice: (x: Double, op: Double => Double)Double

scala> twice(5, _ + 0.2)    // Looks like normal invokation
res1: Double = 5.4
```

```
scala> def twice(x: Double)(op: Double => Double) = op(op(x))
twice: (x: Double)(op: Double => Double)Double
```

CURRYING USING CURLY BRACKETS

```
scala> def twice(x: Double, op: Double => Double) = op(op(x))
twice: (x: Double, op: Double => Double)Double

scala> twice(5, _ + 0.2)    // Looks like normal invokation
res1: Double = 5.4
```

```
scala> def twice(x: Double)(op: Double => Double) = op(op(x))
twice: (x: Double)(op: Double => Double)Double

scala> twice(5) {   // Looks more like a ”built-in” control structure
  _ + 0.2
}
res1: Double = 5.4
```

BY-NAME PARAMETERS

- CREATED BY GIVING THE PARAMETER A TYPE STARTING WITH => INSTEAD OF () =>

STANDARD PARAMATER

```
scala> def foo(predicate: () => Boolean) = predicate()
foo: (predicate: () => Boolean)Boolean
```

BY-NAME PARAMETERS

- CREATED BY GIVING THE PARAMETER A TYPE STARTING WITH => INSTEAD OF () =>

STANDARD PARAMATER

```
scala> def foo(predicate: () => Boolean) = predicate()
foo: (predicate: () => Boolean)Boolean

scala> foo(() => 3 > 4)    // A bit awkward...
res1: Boolean = false
```

BY-NAME PARAMETERS

- CREATED BY GIVING THE PARAMETER A TYPE STARTING WITH => INSTEAD OF () =>

STANDARD PARAMATER

```
scala> def foo(predicate: () => Boolean) = predicate()
foo: (predicate: () => Boolean)Boolean

scala> foo(() => 3 > 4)    // A bit awkward...
res1: Boolean = false
```

BY-NAME PARAMETER

```
scala> def foo(predicate: => Boolean) = predicate
foo: (predicate: => Boolean)Boolean
```

BY-NAME PARAMETERS

- CREATED BY GIVING THE PARAMETER A TYPE STARTING WITH => INSTEAD OF () =>

STANDARD PARAMATER

```
scala> def foo(predicate: () => Boolean) = predicate()
foo: (predicate: () => Boolean)Boolean

scala> foo(() => 3 > 4)    // A bit awkward...
res1: Boolean = false
```

BY-NAME PARAMETER

```
scala> def foo(predicate: => Boolean) = predicate
foo: (predicate: => Boolean)Boolean

scala> foo(3 > 4)    // Better!
res1: Boolean = false
```

TIME FOR DISCUSSION?