Reflection

- The objective of the course is to prepare you for programming related job positions by introducing to you different programming paradigms, e.g. imperative, functional, object-oriented and the combination of the three.
- Discuss which of the 4 languages you like the most and the language you like the least and why.

Test IV Review

- Scala programming language
- Lecture Notes
- Assignment 7 and 8

For-comprehension

for(i <- 1 to n) { ... }
// i traverses from 1 to n
for(i <- 0 until n) { ... }
// i traverses from 0 to n-1
for(i <- 1 to 3; j <- 1 to 3) println (i+"","+j)
//?
for(i <- 1 to 3; jfrom = 4-i; j <- jfrom to 3) println (i+"","+j)
//?
For-Comprehension - Continue

• `yield` generates a sequence.

```scala
scala> for (i <- 1 to 5) yield i % 3
res1: Vector(1, 2, 0, 1, 2)
```

```scala
scala> val a = Array(1, 2, 3, 4)
scala> for (e <- a if e > 2) yield e
res2: Array[Int] = Array(3, 4)
```

```scala
scala> val b = Array(10, 11, 12, 13)
scala> for ((i, j) <- a; j <- b; if j % i != 0) yield (i, j)
res3: Array[(Int, Int)] = Array((2, 11), (2, 13), (3, 10), (3, 11), (3, 13), (4, 10), (4, 11), (4, 13))
```

Higher-order Function

• Functions which take other functions as parameters or return them as results are called `higher-order functions`.

```scala
scala> Array(1, 2, 3, 4).map((x: Int) => x + 1)
res10: Array[Int] = Array(2, 3, 4, 5)
```

```scala
scala> def mulBy(factor: Double) = (x: Double) => factor * x
mulBy: (factor: Double)Double => Double
scala> val triple = mulBy(3)
triple: Double => Double = <function1>
```

Pattern Matching for Case Class

• Case classes are a special kind of classes that are defined to use in pattern matching.

```scala
abstract class Expr // Expr is an abstract class
case class Number(n: Double) extends Expr // Number and Sum are case classes.
val e1 = Number(10)
```

```scala
def eval(e: Expr): Double = e match{
  case Number(n) => n
  case Sum(e1, e2) => eval(e1) + eval(e2)
}
```

Val e = Sum(Number(1), Number(2), Number(3))
eval(e)//?
Mix-in Traits

- **Reuse:** Extend a class and extend a trait make the methods and instance variables of the class and trait available for the sub-class.
- **Extend a Class:** all methods and instance variables of the extended class are **inherited** by the class.
- **Extend a Trait:** all methods and instance variables of the extended Trait are “mixed in” with the class, i.e. the compiler **adds** the mixed-in methods/instance variables to the class.

```javascript
var alice=new WorkerStudent(12);
alice.parkingFee;//?
alice.itemDueDate;//?
var bob=new StudentWorker(21);
bob.parkingFee;//?
bob.itemDueDate;//?
```

Simulate Multiple Inheritance

```
abstract class Person{
parkingFee()
itemDueDate()
}

trait Student
parkingFee()
itemDueDate()
}

trait Worker
parkingFee()
itemDueDate()
}

class StudentWorker extends Worker with Student

class WorkerStudent extends Student with Worker
```