06 Lab
OOP/FP and Collections

Mirko Viroli, Roberto Casadei, Gianluca Aguzzi
{mirko.viroli,roby.casadei,gianluca.aguzzi}@unibo.it

C.D.L. Magistrale in Ingegneria e Scienze Informatiche
Alma Mater Studiorum—Università di Bologna, Cesena

a.a. 2024/2025

Lab 06: Outline

Outline

- Exercise with Scala's combined OOP/FP programming model
- Exercise with Scala's collections

Getting started

- Fork/clone repository `https://github.com/unibo-pps/pps-lab06`
- Then, follow the instructions in the following slides

Exercise 1: OO/FP lists

1) Implement `zipWithValue` that creates a list of pairs, with the second element being the value passed as parameter
2) Implement `def length: Int` using `foldLeft`
3) Implement `zipWithIndex`: it creates a list of pairs, with the second element being l-1,l-2,l-3,..,0 where l is the length of the list.
   - **Hint**: Directly call foldRight to implement this function.
   - Generalize the generation of indexes and also try to implement using the right index order (0,1,2,3,..)
4) Implement `partition`: it partitions the list into the two lists of elements that do and do not satisfy the given predicate
5) Implement `span`: similar to `partition`, but here the predicate creates a split point
6) Implement `takeRight`: returns a list with the last k elements of the original list
7) Extend `List` with a `collect` function that accepts a `PartialFunction[A,B]` and performs map & filter in one shot

- Try to implement these functions firstly with recursions then using `foldLeft`, `foldRight`, `map`, `flatMap`, and `filter`
- **NB!** if some methods cannot be implemented with `fold*` or `map/flatMap`, try to generalize the operation in a new method.

Exercise 2: mini management application

Practice implementing a management application using Scala collections and OOP/FP concepts

- Reimplement in Scala the system described in `https://bitbucket.org/mviroli/oop2018-esami/src/master/a03b/e1/Test.java`
- Notes
  - This time you have to rewrite the java interface + enum in Scala
  - Follow the line guides described in class (prefer immutable var over mutable val, ...)
  - You can look to the solution (`https://bitbucket.org/mviroli/oop2018-esami/src/master/a03b/sol1/ConferenceReviewingImpl.java`)

Exercise 3: collections (Optional)

- Take a look at the examples in Lecture slides
- For each kind of collection (sequence, set, map) and mutable/immutable version:
  - Create a collection
  - Read (i.e., query) the collection (e.g., for size or specific elements)
  - Update the collection
  - Delete elements from the collection

Evaluate the performance of collections

- Write a program or tests showing the efficiency or inefficiency of collection types of your choice. Think about an effective organisation of such an evaluation program.
  - You are given a `PerformanceUtils` module with helper functions
  - Note: this is a *naive* form of "microbenchmarking" (an effective approach for measuring performance should take into account several issues and work statistically)
- Please take a look at the performance profiles of the collection:
  - `https://docs.scala-lang.org/overviews/collections-2.13/performance-characteristics.html`
- Share your results e.g., with your peers in the forum of the course
- Consider also to be inspired by similar explorations:
  - `https://www.lihaoyi.com/post/BenchmarkingScalaCollections.html`
  - `https://github.com/Ehyaei/scala-collection-benchmark?tab=readme-ov-file`
- Try to find a scenario where mutable list outperforms immutable list:
  - Describe a trait which may use both mutable and immutable data structures
  - Implement the trait using mutable (e.g., List)
  - Implement the trait using immutable (e..g, MutableList)
  - Note! Compare with the corresponding collection (e.g., List vs. MutableList, HashSet vs. Set)
  - Compare the performance of the two implementations