05 Lab
OOP/FP and Collections

Mirko Viroli, Roberto Casadei
{mirko.viroli,roby.casadei}@unibo.it

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

a.a. 2020/2021

Lab 05: 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-20-21-lab05`
- Then, follow the instructions at the following slides

Exercise 1: OO/FP lists

1) Implement `zipRight`: it creates a list of pairs, with the second element being 0,1,2,3,..
   - May need an internal recursion
2) Implement `partition`: it partitions the list into the two lists of elements that do and do not satisfy the given predicate
3) Implement `span`: similar to `partition`, but here the predicate creates a split point
4) Implement `reduce`: it is similar to `fold` but does not take an initial value (raises an exception on empty lists; returns the head on single-element lists)
5) Implement `takeRight`: returns a list with the last k elements of the original list
6) Extend `List` with a `collect` function that accepts a `PartialFunction[A,B]` and performs map & filter in one shot

Exercise 2: mini management application

Make practice with collections by implementing a management application

- Reimplement in Scala the system described in `https://bitbucket.org/mviroli/oop2018-esami/src/master/a01b/e1/`
- Notes
  - Enums can be implemented in Scala with usual sum/product types:
    - a base trait `trait MyEnum`
    - a memberless `case class A() extends MyEnum` (or a `case object A extends MyEnum`) for each element, to be put in a companion `object MyEnum`;
  - Alternatively, enums can be expressed as follows: `object MyEnum extends Enumeration { type MyEnum = Value; val A,B = Value }`[1] where values are also mapped to `int`s as usual for enums

[1] For details, read: `https://underscore.io/blog/posts/2014/09/03/enumerations.html` or consult the Scala API

Exercise 3: collections

Exercise with collections

- 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 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)
- Share your results e.g. with your peers in the forum of the course

Exercise 4 (optional)

- Implement a `sequence` function with the following signature:

  ```
  def sequence[A](a: List[Option[A]]): Option[List[A]]
  ```

  It combines a list of `Options` into one `Option` that contains the list of all the `Some` values in the original list. If the original list contains `None` even once, the result should be `None`, otherwise `Some` with a list of all the values.
  - Examples:

    ```
    sequence(List(Some(1),Some(2),Some(3))) // Some(List(1, 2, 3))
    sequence(List(Some(1),None,Some(3))) // None
    ```
  - Hint: consider using a `fold`
    - Question: what `fold` (left or right) does support processing a list while preserving its order?