# 05 Lab
# OOP in Scala

Mirko Viroli, Gianluca Aguzzi
{mirko.viroli,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 05: Outline

## Outline

- Exercises on OOP in Scala

# Getting started

- Fork/clone repository `https://github.com/unibo-pps/pps-lab05`
- Then, follow the instructions in the following slides

# Exercise 1: Scala OOP

- Consider the classes described in the package `ex`:
  - `Vector2D` - a 2D vector implementation
  - `OnlineCoursePlatform` - a platform for managing online courses
- First implement the `Vector2D` class, then the `OnlineCoursePlatform`
- Ideally, use private nested classes (possibly case classes?) inside the companion object
- Follow the guidelines provided in the comments at the top of each main class
- Review the code inside the main methods to understand the expected behavior of your implementations

# Exercise 2: more oop + java

- Consider the Java code located in `java/polyglot/a05b`
- It contains the interface `Logics` and a GUI which uses this `Logics` interface
  - This follows the standard structure of Prof. Viroli's OOP exams
- **Goal**: Implement the application logic in idiomatic Scala code based on the `Logics` Java interface
  - Keep the GUI as it is, and implement the Logics interface with a Scala class
  - Guidelines: Write compact code contained in a single Scala source file
  - Specifically, implement it in a Scala `LogicsImpl` class inside `scala/polyglot/a05b`
- **Note:** You must only use the data structures covered in class (i.e., `Sequence`, `Optional`, etc.). Do not use Scala's standard collections
- You may use `scala.util.Random` (similar to `java.util.Random`):

```
1  val random = scala.util.Random(42)
2  random.nextInt() // unbounded next int
3  random.nextInt(10) // random number between 0 and 10
4
```

# Exercise 3: more oop and Java

- Do the same of Exercise 2, but with another OOP exam of your choice, or that in folders `java/polyglot/01a` or `01b`
- This time, implement `Logics` interface as a Scala `trait` first, and try to call it from the GUI.
- Implement the `trait` in Scala, without looking at the OOP solution.

# Exercise 4: more oop

- Implement again exercise 2 of previous Lab (`SchoolModel`) using OOP in Scala, so that it passes the same test

# Exercise 5: unapply

- Implement an extractor `sameCategory(cat)` on a list of `Courses` that extracts the category `cat` in common to **all** courses (if any)

```
1 val courses = Sequence(course1, course2, course3)
2 courses match
3   case sameCategory(cat) => println(s"$courses have same
      category $cat")
4   case _ => println(s"$courses have different categories
    ")
```

- NB: extractors (`unapply`) must use `scala.Option`, `scala.Some`... (it is easy to write a method to convert "our" `Option` to Scala's one)