02Lab
Advanced testing, mocking, integration

Mirko Viroli
mirko.viroli@unibo.it

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

a.a. 2024/2025

One slide sum-up on Acceptance Testing

Unit testing

- clean and simple tests for isolated units of code
- trade-off between completeness and cost of maintenance
- coverage could be a measure, but it has limits
- dependencies must be finely-mocked: use `Mockito`

Integration testing

- when we test collaborations, with are quitting unit testing
- mocking still needs to be used to isolate few collaborations at a time

Starting point and goals

References

- Mockito: `https://site.mockito.org/`
- 02-repo-testing: `https://github.com/mviroli/asmd24-public-02-testing`
- LLMs:
  - **ChatGPT**: `https://chatgpt.com/`
  - **Perplexity**: `https://www.perplexity.ai/`
  - **Claude**: `https://claude.ai`
  - **Gemini**: `https://gemini.google.com/app?hl=it`
- Slide 02 provide few examples of mockito/Junit

General goals

- be operative with Mockito and JUnit
- understand mocking in its details
- exercise with integration testing
- pre-check ability of ChatGPT to help you in unit and integration testing

Operational Steps: Play with Mockito

Step 1 – Get Ready!!

- Clone the repo (at `https://github.com/mviroli/asmd24-public-02-testing`)
- Open it in IntelliJ
- Run the tests
  - If everything is fine, you are ready to go :)

Step 2 – Reorganise the Device Example

- **Goal**: Create technically excellent, isolated tests for the device and its failing policy
- First, explore the existing code:
  - Review `test/java/devices/StandardDeviceTest.java` and `test/java/devices/AlternateStandardDeviceTest.java`
  - Identify different mocking techniques and patterns used in these files
  - Understand how Mockito is being applied in different scenarios
- Then, create proper isolated unit tests:
  - Develop a new unit test class for the device, completely isolated from the failing policy
  - Create a separate unit test class focused solely on testing the failing policy
  - Ensure both test classes follow best practices for unit testing
- Finally, build integration tests:
  - Develop integration tests that verify the proper collaboration between the device and failing policy
  - Compare the differences between your unit and integration tests

Tasks

TOOLING

Experiment with installing/using Mockito with Scala and/or in VSCode. Is VSCode better at all here? What's the state of mocking technologies for Scala?

REENGINEER

Take an existing implemented small app with GUI, e.g. an OOP exam. Add a requirement that it outputs to console some relevant messages, through a log class. Now you have an App with at least 3 classes (GUI, Model, Log). How would you write integration tests for it? Search here: `https://bitbucket.org/mviroli/oop2023-esami` (2023, 2022,...)

GUI-TESTER

Generally, GUIs are a problem with testing. How do we test them? How do we automatise as most as possible testing of an app with a GUI? Play with a simple example and derive some useful consideration.

TESTING-LLM

LLMs/ChatGPT can arguably help in write/improve/complete/implement/reverse-engineer a JUnit test, either unit or integration test. Experiment with this, based on the above tasks or in other cases. Is ChatGPT useful for all that?