JUnit and TDD HansOn

Gian Enrico Conti / Niccolò Izzo

Prova Finale - Ingegneria del Software - AA 2017/18

Unit testing

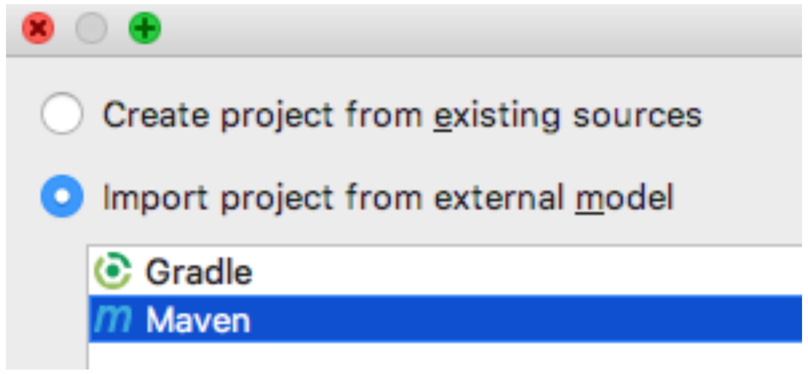
Partiamo da un project su repo:

(es. LMxx, voi lo avete gia..)

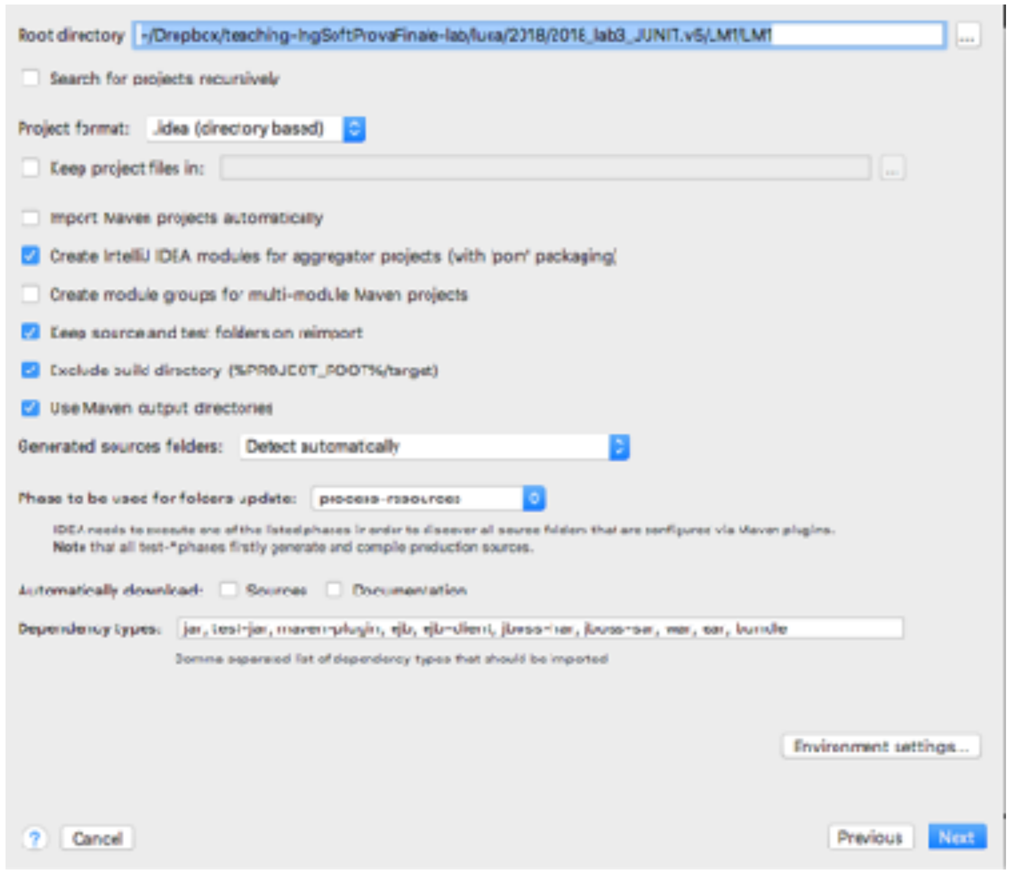
```
git clone https://github.com/AlessiaAcc/LM1.git
Cloning into 'LM1'...
remote: Counting objects: 92, done.
```

Unit testing

Import..


Create project from existing sources
[✓ Import project from external model]

Gradle
[✓ Maven]


Root directory: ~/Dropbox/teaching-IngSoftProvaFinale-lab/luca/2018/2018_lab3_JUNIT.v5/LM1/LM1

- [ ] Search for projects recursively

Project format: .idea (directory based)

- [ ] Keep project files in:
- [ ] Import Maven projects automatically
- [x] Create IntelliJ IDEA modules for aggregator projects (with 'pom' packaging)
- [ ] Create module groups for multi-module Maven projects
- [x] Keep source and test folders on reimport
- [x] Exclude build directory (%PROJECT_ROOT%/target)
- [x] Use Maven output directories

Generated sources folders: Detect automatically

Phase to be used for folders update: process-resources

IDEA needs to execute one of the listed phases in order to discover all source folders that are configured via Maven plugins.
Note that all test-*phases firstly generate and compile production sources.

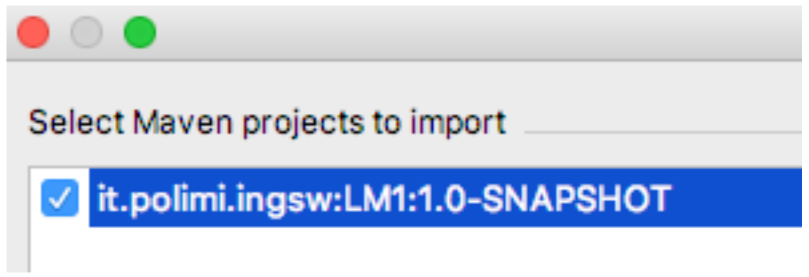
Automatically download: [ ] Sources [ ] Documentation

Dependency types: jar, test-jar, maven-plugin, ejb, ejb-client, jboss-har, jboss-sar, war, ear, bundle

Comma separated list of dependency types that should be imported

Unit testing

Import (cont)..

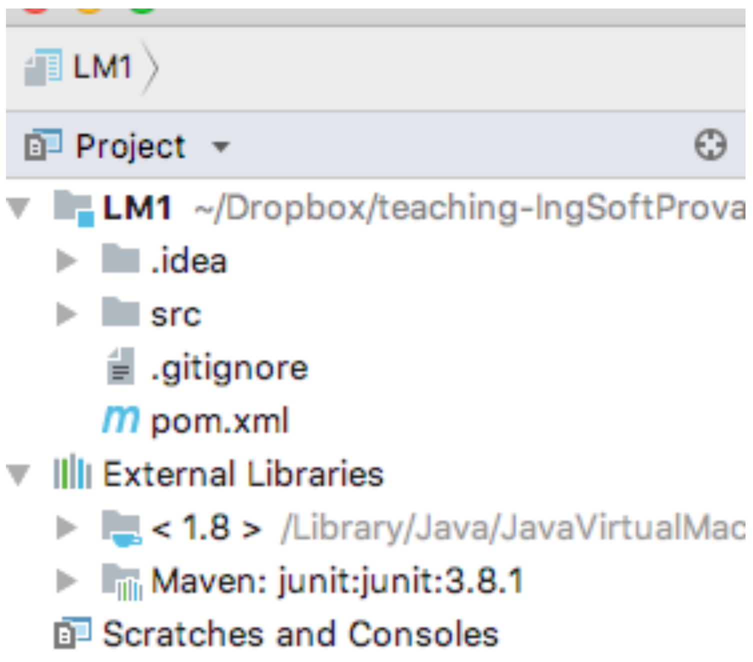


Select Maven projects to import

✓ it.polimi.ingsw:LM1:1.0-SNAPSHOT

…

Alla fine:



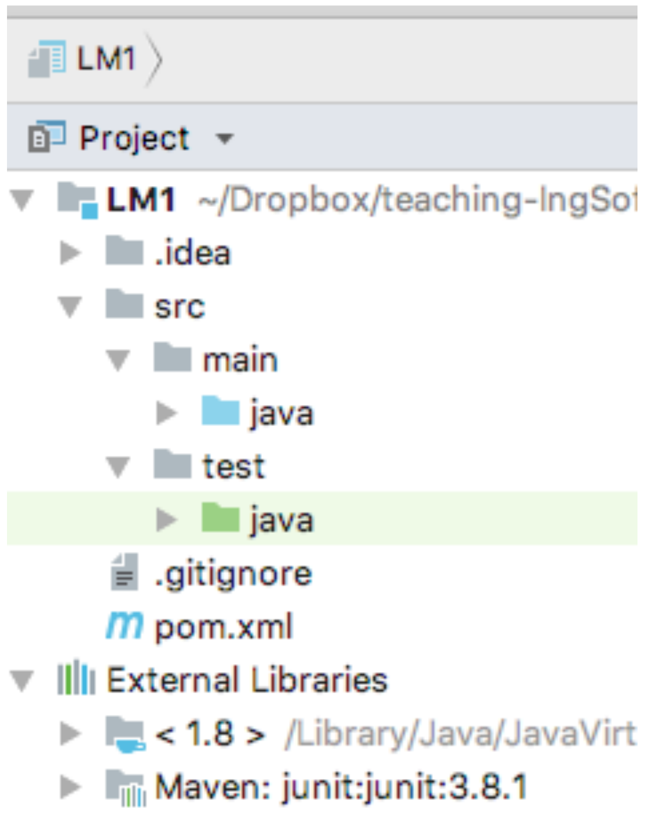
```
LM1 〉
Project ▾
▼ LM1  ~/Dropbox/teaching-IngSoftProva
    ▶ .idea
    ▶ src
      .gitignore
      pom.xml
▼ External Libraries
    ▶ < 1.8 >  /Library/Java/JavaVirtualMac
    ▶ Maven: junit:junit:3.8.1
  Scratches and Consoles
```

Unit testing

2 cartelle:

main

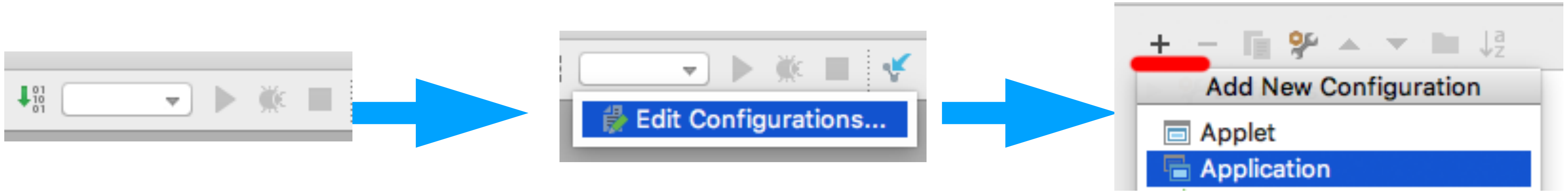
test



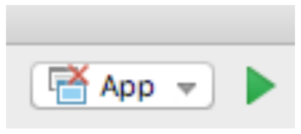
Unit testing

Creiamo velocemente le config:

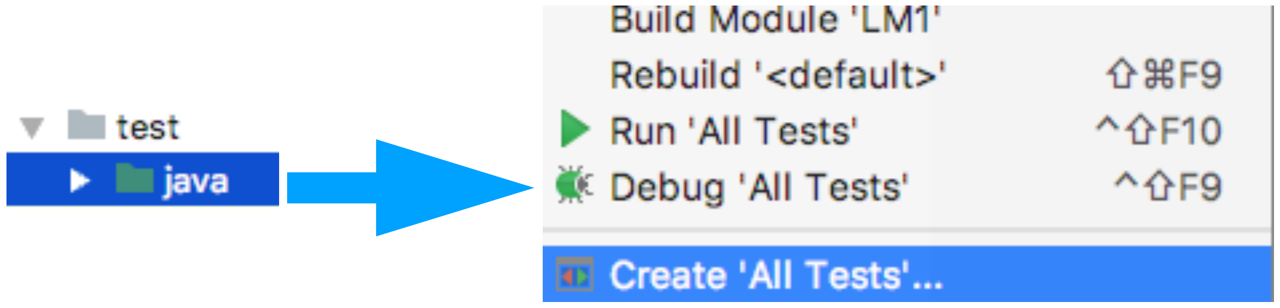
- App:



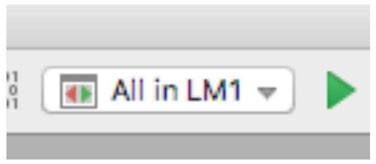
run…



- Test: right click:



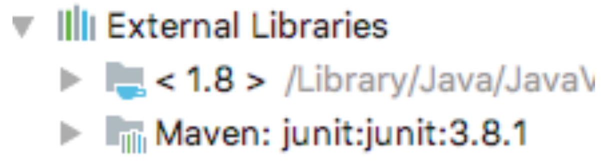
run…



Unit testing

So far so good..

abbiamo v.3..



```
▼ External Libraries
  ▶ < 1.8 > /Library/Java/JavaV
  ▶ Maven: junit:junit:3.8.1
```

passiamo a JUnit 5…

Unit testing

da sito: https://junit.org/junit5/

A first test case

```
import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

class FirstJUnit5Tests {

    @Test
    void myFirstTest() {
        assertEquals(2, 1 + 1);
    }
}
```

Mettiamolo nel file AppTest.java..

Unit testing

```
package it.polimi.ingsw;


import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

class FirstJUnit5Tests {

    @Test
    void myFirstTest() {
        assertEquals(2, 1 + 1);
    }

}
```

Manca junit 5..

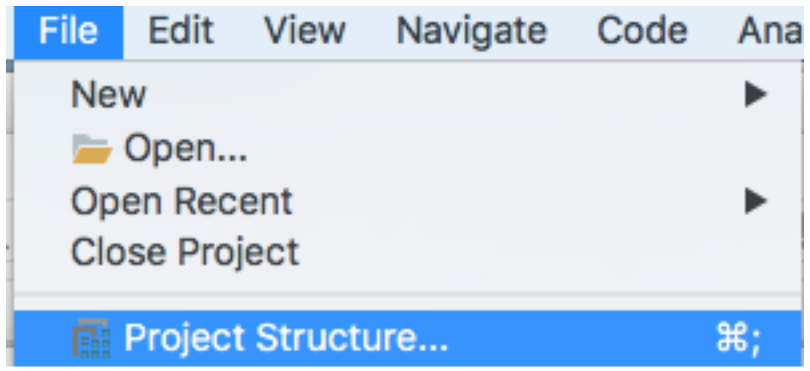
Alt Invio..

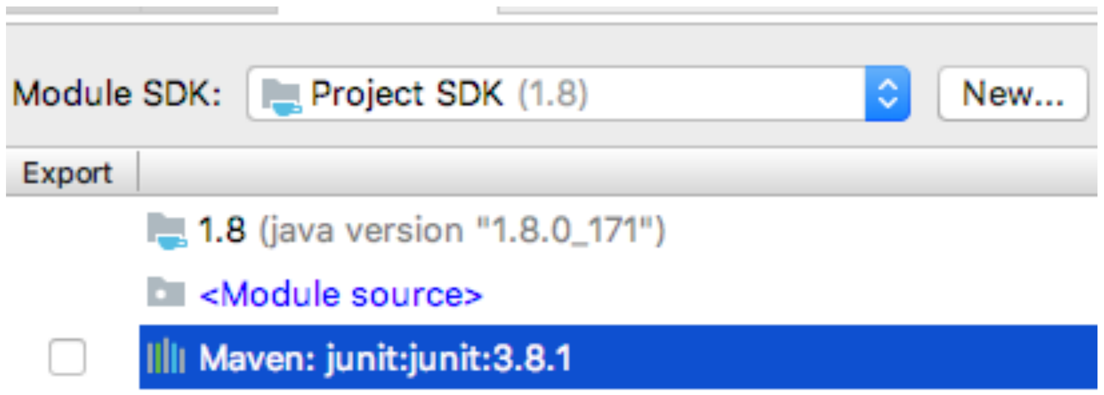
Unit testing

Build and run..

errore..

rimuovere Junit 3..





Module SDK: Project SDK (1.8)

Export

1.8 (java version "1.8.0_171")

<Module source>

Maven: junit:junit:3.8.1

e cambiare class, deve essere:

```
class AppTest{
```

run..

Unit testing

Ok.

Controprovova:

```
@Test
  void myFirstTest() {
      assertEquals(5, 1 + 1);
  }
```

```
org.opentest4j.AssertionFailedError: 
Expected :5
Actual   :2
 <Click to see difference>
```

test “veri”: validiamo il **login**.

Unit testing

Logica:

- classe LoginManager (tralasciamo x ora singleton etc..)
- Array di nick
- validiamo doppio inserimento (per ora solo nome..)

Unit testing

```
public class LoginManager {

}
```

- aggiungiamo:

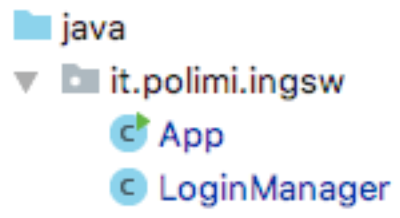
```
public class LoginManager {

    ArrayList<String> nicknames;

    void allocateLazy(){
        if (nicknames == null) {
            nicknames = new ArrayList<String>();
        }
    }

    Boolean login(String nick ) {
        allocateLazy();
        int count = nicknames.size();
        if (count >= 4)
            return false;

        nicknames.add(nick);
        return true;
    }
```



(serve `import java.util.ArrayList;`)

Unit testing

il test:

```
class AppTest{

   @Test
   void myFirstTest() {
       assertEquals(2, 1 + 1);
       }

       @Test
       void LoginTest() {
           System.out.println("testing");
           LoginManager lm = new LoginManager();
           assertEquals(true, lm.login("bob"));
           assertEquals(false, lm.login("bob"));
       }
```

il test: bob NON puo’ loggare 2 volte..

run..

Unit testing

```
org.opentest4j.AssertionFailedError:
Expected :false
Actual   :true
```

WHY?

Unit testing

TDD ha fatto suo dovere:

il metodo login e' errato:

```
Boolean login(String nick) {
    allocateLazy();
    int count = nicknames.size();
    if (count >= 4)
        return false;

    nicknames.add(nick);
    return true;
}
```

deve essere:

```
Boolean login(String nick) {
    allocateLazy();
    int count = nicknames.size();
    if (count >= 4)
        return false;

    if (nicknames.contains(nick))
        return false;

    nicknames.add(nick);
    return true;
}
```

Unit testing

```
testing
Process finished with exit code 0
```

Quindi:

- scrivete classse/metodo

- scrivete Test

from theory:

TDD:
1. Add a test
In test-driven development, each new feature begins with writing a test…