

# Stubs, Mocks, and Architecture

# OBJECTIVES

- Writing a stub (temporary code)
- Writing a mock (faking code)
- Architecting to not require stubs or mocks
- Continue practice of Git/GitHub

# USING SEG3103\_PLAYGROUND

- Create **/lab05** directory
  - Extract **grades.zip** and **twitter.zip**
- Make sure you can
  - compile code
  - Run tests

## Grades

Homework #1	Labs #1	<b>Final Grade</b>  Letter Grade --  Numeric Grade --  Percent --  <b>Calculate</b>  <b>mix phx.server</b> <b><a href="http://localhost:4000">http://localhost:4000</a></b>
Homework #2	Labs #2	
Homework #3	Labs #3	
Homework #4	Labs #4	
Midterm	Labs #5	
Final	Labs #6	

The grades project will be similar to assignment 2. You will want to get the code up and running **mix phx.server**. Then visit <http://localhost:4000> and you should see a page similar to this one.

Please read the errors!!!

Before continuing, please read the error messages, really read the error messages and spend at least 5 minutes attempting to resolve the issue before continuing.

```
Unchecked dependencies for environment dev:  
* telemetry_metrics (Hex package)  
  the dependency is not available, run "mix deps.get"  
* phoenix_live_view (Hex package)  
  the dependency is not available, run "mix deps.get"  
* telemetry_poller (Hex package)  
  the dependency is not available, run "mix deps.get"  
* phoenix_live_reload (Hex package)  
  the dependency is not available, run "mix deps.get"  
* jason (Hex package)  
  the dependency is not available, run "mix deps.get"  
* phoenix_html (Hex package)  
  the dependency is not available, run "mix deps.get"  
* phoenix (Hex package)  
  the dependency is not available, run "mix deps.get"  
* plug_cowboy (Hex package)  
  the dependency is not available, run "mix deps.get"  
** (Mix) Can't continue due to errors on dependencies
```

`mix deps.get`

```
[info] Running gradesWebEndpoint with console 2.0.0 at 0.0.0.0:3000 (http://localhost:3000)  
[error] Could not start Node.js watcher because script "/Users/aforward/sin/courses/_/seg3  
x03_internal/lab05_solution/grades/assets/node_modules/webpack/bin/webpack.js" does not ex  
ist. Your Phoenix application is still running, however assets won't be compiled. You may  
fix this by running "npm install" inside the "assets" directory.  
[info] Running gradesWebEndpoint with console 2.0.0 at 0.0.0.0:3000 (http://localhost:3000)
```

```
cd assets && npm install
```

In an older version, npm was needed and to fix you had to run "cd assets && npm install"

## Grades

Homework #1	Labs #1	<b>Final Grade</b>  Letter Grade --  Numeric Grade --  Percent --  <b>Calculate</b>
Homework #2	Labs #2	
Homework #3	Labs #3	
Homework #4	Labs #4	
Midterm	Labs #5	
Final	Labs #6	

Button doesn't work

Now if you click the calculate button... it pauses and then restarted. Go read the error messages :-)



```
[error] GenServer #PID<0.2239.0> terminating
** (UndefinedFunctionError) function Grades.Calculator.letter_grade/1 is undefined
(module Grades.Calculator is not available)
    Grades.Calculator.letter_grade(%{final: "", homework: ["", "", "", ""], labs:
["", "", "", "", "", ""], midterm: ""})
    (grades 0.1.0) lib/grades_web/live/page_live.ex:23:
GradesWeb.PageLive.handle_event/3
```

### Stub Grades.Calculator

- percentage\_grade
- letter\_grade
- numeric\_grade

You are the front-end team hoping to finish up the user interface for the grades calculator. You are waiting on your team-mates to finish the calculate function, so in the interim write a STUB so that you can test your user interface.

```
socket
  > assign(:grades, grades)
  ... > assign(:letter_grade, Calculator.letter_grade(grades))
  ... > assign(:percentage_grade, Calculator.percentage_grade(grades))
  ... > assign(:numeric_grade, Calculator.numeric_grade(grades))
  > my_reply(:noreply)
end
```

Do not implement the methods, just  
provided *good enough* replies (aka stubs)  
for those methods

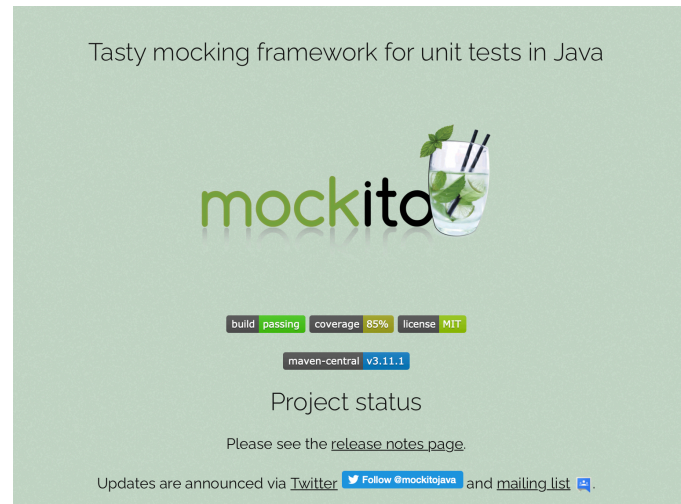
The calculator module provides three methods, we don't want to change this code, we just want an implementation that is good enough so that we can do our front end changes while we wait on the backend to provide the real implementation.

```
# alias Grades.Calclator

defmodule Calculator do
  def letter_grade(_input) do
    # implement me...
  end
end
```

Make the stub code easy to throw out.

Instead of calling the real `Grades.Calculator`, we will implement an inline one so the rest of our code need not change when the *real Calculator* is available.



<https://site.mockito.org>

TBD

Unable to get Mockito to work as expected using JUnit5.

As an aside you can play with Mockito yourself and push back sample code if you get it working.

# EASYMOCK

Easy mocking. Better testing.

[Getting started](#)

[Download \(v4.3\)](#)

<http://easymock.org/>

<https://easymock.org/user-guide.html>

```
twitter (main)$ ./bin/run
Twitter Text Feed
Hello to @you

twitter (main)$ ./bin/run
Twitter Text Feed

twitter (main)$ ./bin/run
Twitter Text Feed
Hello to @you

twitter (main)$ ./bin/run
Twitter Text Feed
I am tweet that likes to talk about @me
```

# Compile the application (just the application)

```
javac -encoding UTF-8 --source-path src -d dist src/*.java
```

# Run the code

```
java -cp ./dist Main
```

```
├─ JUnit Jupiter ✓  
  └─ TwitterTest ✓  
    ├── mock_full_object() ✓  
    ├── mock_partial_object() ✓  
    └─ actual_call() ✓  
└─ JUnit Vintage ✓
```

./bin/test

./bin/clean

./bin/compile

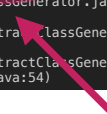
java --add-opens java.base/java.lang=ALL-UNNAMED -jar lib/junit-platform-console-standalone-1.7.1.jar -cp dist:lib/easymock-4.3.jar:lib/objenesis-3.2.jar --scan-class-path



```

JUnit Jupiter:TwitterTest:mock_full_object()
MethodSource [className = 'TwitterTest', methodName = 'mock_full_object', methodParameterTypes = '']
=> java.lang.ExceptionInInitializerError
    org.easymock.internal.ClassProxyFactory.createEnhancer(ClassProxyFactory.java:233)
    org.easymock.internal.ClassProxyFactory.createProxy(ClassProxyFactory.java:165)
    org.easymock.internal.MocksControl.createMock(MockControl.java:107)
    org.easymock.internal.MocksControl.createMock(MockControl.java:85)
    org.easymock.IMocksControl.mock(IMocksControl.java:67)
    [...]
Caused by: org.easymock.cglib.core.CodeGenerationException:
java.lang.reflect.InaccessibleObjectException-->Unable to make protected final java.lang.Class
java.lang.ClassLoader.defineClass(java.lang.String,byte[],int,int,java.security.ProtectionDomain) throws
java.lang.ClassFormatError accessible: module java.base does not "opens java.lang" to unnamed module
@480d3575
    org.easymock.cglib.core.ReflectUtils.defineClass(ReflectUtils.java:464)
    org.easymock.cglib.core.AbstractClassGenerator.generate(AbstractClassGenerator.java:339)
    org.easymock.cglib.core.AbstractClassGenerator$ClassLoaderData$3.apply(AbstractClassGenerator.java:96)
    org.easymock.cglib.core.AbstractClassGenerator$ClassLoaderData$3.apply(AbstractClassGenerator.java:94)
    org.easymock.cglib.core.internal.LoadingCache$2.call(LoadingCache.java:54)

```


**RTFE****--add-opens java.base/java.lang=ALL-UNNAMED**<https://github.com/easymock/easymock/issues/235>

```
|— JUnit Jupiter ✓  
  |— TwitterTest ✓  
    |— mock_full_object() ✓  
    |— mock_partial_object() ✓  
    |— actual_call() ✗ Cannot invoke "String.conf  
<local2>" is null  
  |— JUnit Vintage ✓
```

./bin/test 🥲

```
@Test
void mock_full_object() {

    Twitter twitter = createMock("twitter", Twitter.class);

    expect(twitter.loadTweet()).andReturn("hello @me");
    expect(twitter.loadTweet()).andReturn("hello @you");
    replay(twitter);

    String actual;

    actual = twitter.loadTweet();
    assertEquals("hello @me", actual);

    actual = twitter.loadTweet();
    assertEquals("hello @you", actual);
}
```

```
@Test
void mock_partial_object() {

    Twitter twitter = partialMockBuilder(Twitter.class)
        .addMockedMethod("loadTweet")
        .createMock();

    expect(twitter.loadTweet()).andReturn("hello @me").times(2);
    replay(twitter);

    boolean actual;

    actual = twitter.isMentionned("me");
    assertEquals(true, actual);

    actual = twitter.isMentionned("you");
    assertEquals(false, actual);
}
```

```
@Test
void isMentionned_lookForAtSymbol() {
    // Assuming a tweet like "hello @me"
    // isMentionned("me") should be true
    // isMentionned("you") should be false
}
```

```
@Test
void isMentionned_dontReturnSubstringMatches() {
    // Assuming a tweet like "hello @meat"
    // isMentionned("me") should be false
    // isMentionned("meat") should be true
}
```

```
@Test
void isMentionned_superStringNotFound() {
    // Assuming a tweet like "hello @me"
    // isMentionned("me") should be true
    // isMentionned("meat") should be false
}
```

```
@Test
void isMentionned_handleNull() {
    // Assuming no tweet is available (i.e. null)
    // isMentionned("me") should be false
    // isMentionned("meat") should be false
}
```

## Test isMentionned()

# SUBMISSION

- All work should be written under
  - **seg3103\_playground/lab05**
- Git commit your work at each step
  - When the application starts (BEFORE YOU MAKE ANY CHANGES)
  - With the stubbed value
  - With the value from assignment #2
- Create **README.md** to summarize your work
- Share your repository with the teacher and TA(s)
  - Submissions to BrightSpace should clearly reference your GitHub repository
- Grades project
  - Your stubbed code
  - Observations from the stub
- Twitter
  - Implement the 4 missing test cases using mock objects
  - Show the results of those tests
  - Analyze the results by looking at the code of isMentionned
  - If necessary fix the code based on your testing