

Test Driven Development with Ruby Mock Objects

Exercises

- Download the exercise source code from <https://github.com/royosherove/rubydrills>
- Open the "mocks1" folder

1. Feature: String Calculator Logging

In order to easily debug the results the a user is getting as developers

we want to have logging enabled for the string calculator

we will only use HAND WRITTEN mocks (no framework)

Scenario: Adding a number

Given a string calculator

And a slow logger instance

When the string calculator add method is called with input "1,2"

Then slow logger should be notified with "got 3"

Scenario: Failed Logging

Given a string calculator

And a slow logger instance

And a web service instance

And calling logger results in an error 'fake error'

When the string calculator add method is called with input "1,2"

Then the web service should be notified with "got 'fake error'"

2. Repeat feature 1 using **Bogus** instead of hand written mocks
refer to the docs here <https://www.relishapp.com/bogus/bogus/docs>

3. Feature: Login Manager Logging

In order to learn how to use mock objects and stubs effectively as developers

we want to make logging enabled for the LoginManager class

we will bogus stubs and mocks

Scenario: Adding a user

Given a `LoginManager`

And a `SlowLogger` instance

When calling `LoginManager.add_user` with username "a" and password "pass"

Then the slow logger is notified with the string "logon by user 'a' and password 'pass'"

Scenario: Failed Logging

Given a Login Manager

And a slow logger instance

And a web service instance

And slow logger is configured to raise a "fake error" error

When calling `add_user` with username "a" and password "pass"

Then the web service is notified with the string "logger threw up 'fake error'"

Scenario: Changing Passwords

Given a Login Manager

With an existing user "a" with password "pass1"

And slow logger instance

When calling `change_password` with username "a" and existing password "pass1" and new password "pass2"

Then the slow logger should be notified with "user changed passwords:a"

4. Feature: Refactoring to trace message object

In order to make our code more maintainable and readable
as developers

we want to use an object to hold logging information and pass it to the logger
we will bogus stubs and mocks

Scenario: Adding `TraceMessage`

Given a Login Manager

And a slow logger instance

When calling `add_user` with username "a" and password "pass"

Then the slow logger is notified with an object of type `TraceMessage` with
attributes `text` with the value we used in other scenarios and `severity` with
value of 100

5. Feature: error on bad pass change

In order to make our code more expressive
as developers

we want to to have more meaningful errors when using our API

Scenario: Changing Passwords

Given a Login Manager

With an existing user "a" with password "pass1"

And slow logger instance

When calling `change_password` with username "a" and wrong password
"badpass" and new password "pass2"

Then the `LoginManager` should raise error "password not changed"

6. Feature: calling a class method

In order to add logging
as developers

we want `LoginManagerWithShared` to use `SharedLogger` for logging

Scenario: adding a user

Given a `LoginManagerWithShared` instance

With no users

When calling `add_user` with username "a" and password "pass"

Then the `SharedLogger.write(text)` method should be called with
input "user added: a, pass"

Scenario: changing password badly for user

Given a `LoginManagerWithShared` instance

With an existing user "a" with password "pass1"

When calling `change_password` with username "a" and wrong password "badpass" and new password "pass2"

Then the `SharedLogger.write(text)` method should be called with input "tried to change password. password not changed"

7. Feature: Manipulating Time and Hardware

In order to know when and where logging happened as developers

we want to add the current time and machine name to our logs

and we will use the Timecop gem to do it:

<https://github.com/travisjeffery/timecop>

Scenario: Formatting the string with a date

Given a `LogStringFormatter` instance

And the current time is 01/01/2000 11:11

When calling `format` with input "text"

Then the formatter should return "01/01/2000 11:11 text"

Scenario: Formatting the string with a machine name

Given a `LogStringFormatter` instance

And the current time is 01/01/2000 11:11

And the current machine name "sOMEMACHINE"

When calling `format` with input "text"

Then the method should return "01/01/2000 - sOMEMACHINE 11:11 text"

8. Feature: Manipulating Future Instances

In order to get logging in StringCalculatorWithFuture as developers we want to use an instance of SlowLogger in it and instantiated it directly

Scenario: Adding numbers

Given a StringCalculatorWithFuture instance

When calling add with input "1"

Then the calculator will instantiate **a new instance** of SlowLogger and call its write method with input "got 1"