EECE 514 Assignment 5

Xi He (ID: 81319162)

Exercise 5.1

- *Pick an open source application with existing unit tests in one of the following languages: Java, C/C++, JavaScript or Python.*
- *The application must be on GitHub. Include the URL of the application in your report.*
- *One group member must fork the repository. Please refer to https://help.github.com/articles/fork-a-repo/ for learning about forking a repository. Include the URL of the forked repository in your report.*
- *Every extension should happen in the forked repository (which can remain public). Create a folder on the forked repository and name it as "unit-testcases" for assignment 5. Your test cases should be added to this directory. The folder should also contain a Readme file explaining how the TA can set up and run the application and its test cases.*

Answer:

- In this assignment, I choose a different app because the project "Prestamos" for assignment 4 does not provide any test case. Therefore, I tried to find a new Java project with test suite for assignment 5. Please open this URL to check the GitHub project "Prestamos" for assignment 4.
  https://github.com/cpenarrieta/prestamos-web

- Please open this URL to check the Amazon application for this assignment:
  https://github.com/amzn/amazon-instant-access-sdk-java

- Please open this URL to check the forked repository:
  https://github.com/Martin-Ho/amazon-instant-access-sdk-java

Exercise 5.2

- *In your report, explain the framework/technique they are using for testing the application. Discuss the pros and cons of the technique for the application under assessment. Discuss an alternative testing approach/framework that may fit the application and compare it with the existing approach.*
- *Document the unit level features (in the code) that are important to test.*
- *Measure the coverage of the existing test suite. Report the coverage and discuss the results.*
- *Identify the features that are not properly tested.*

Answer:

- Firstly, they are using the unit testing with assertions and several test suites including "connection", "serialization.messages", "serialization.serializer", "servlets" and "signature". Secondly, they are using "EasyMock" to mock objects that mimic the behavior of real objects in controlled ways. Thirdly, they are using "javax.servlet" to do web UI testing. Its main function is to interactively browse and modify the data to generate dynamic Web content.

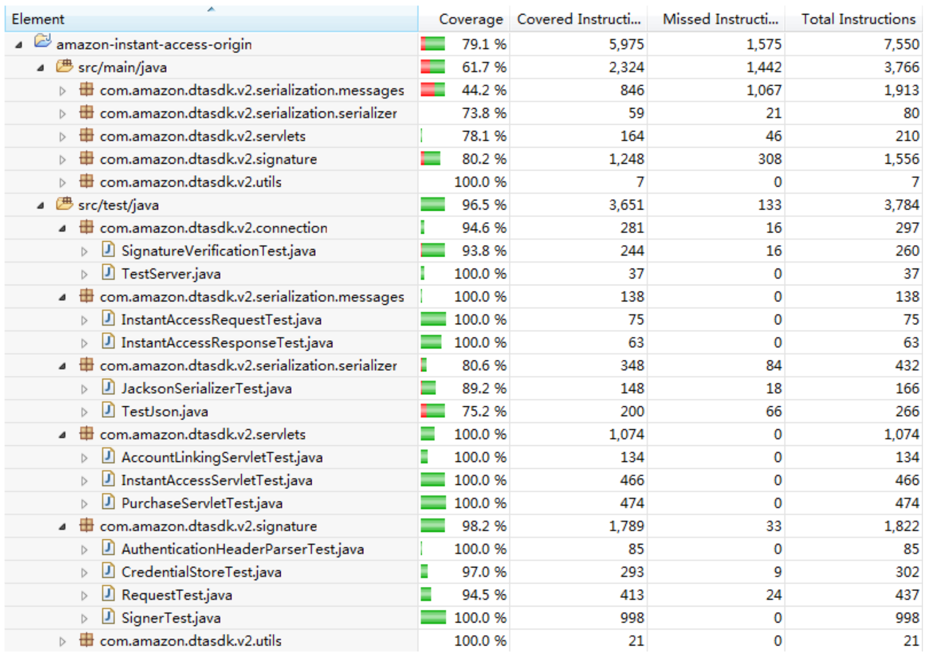
- With unit testing, the scripting can be executed multiple times, and the production codes can be measured in adequacy and effectiveness. However, unit testing is not guarantee for quality, and it only helps to find bugs. Mock objects can improve the controllability, improve observability and break dependencies. However, mock-based testing does not find errors resulting from the interplay of several components, and changes to the interface of the real implementation require changes to the dummy object. Servlet provides high execution efficiency, high platform independence and high security.

- Maybe it can use Selenium as a GUI testing to capture and replay the test cases. Selenium scripts are created by recording actions using the web application under test running in a browser. Therefore, Selenium testing can simulate the operations of the web application in the user side, and it finds wide usage for UI, regression, unit and acceptance testing. However, Selenium is not a complete and comprehensive solution to fully automate the testing of web applications. It requires third-party frameworks, language bindings and so on to be truly effective.

- The unit level features for testing are listed in the table below.

Package	Class	Features
serialization.messages	FulfillPurchaseRequest FulfillPurchaseResponse GetUserIdSerializableRequest GetUserIdSerializableResponse InstantAccessRequest InstantAccessResponse RevokePurchaseRequest RevokePurchaseResponse SubscriptionActivateRequest SubscriptionDeactivateRequest SubscriptionRequest SubscriptionResponse	We need to test different request messages of the web pages of Amazon, such as purchase request, access request, subscription activate request, subscription deactivate request and so on. Moreover, we also need to test whether the correct responses of the web pages can be generated after the request messages are sent and received.
serialization.serializer	JacksonSerializer SerializationException	
servlets	AccountLinkingServlet InstantAccessServlet PurchaseServlet	Test whether the data and contents can be correctly generated, modified and generated in web UI.
signature	AuthenticationHeaderParser CredentialStore Request Signer	Test the processes of signature on the web pages of Amazon, such as the signature request, validation, authentication and so on.
utils	Clock	Test the functions of clock.

- The coverage of the existing test suite is shown in the figure below. The coverage percentage for the whole project, for the production code and for the test suite is 79.1%, 61.7% and 96.5% respectively. We can notice that the coverage of the production code is much less than the coverage of the test suite. Especially, the coverage of the production code package "serialization.messages" is only 44.2%, which indicates that the functions of the request messages and the request response in the web pages have not been fully and properly tested.



Element	Coverage	Covered Instructi...	Missed Instructi...	Total Instructions
amazon-instant-access-origin	79.1 %	5,975	1,575	7,550
src/main/java	61.7 %	2,324	1,442	3,766
com.amazon.dtasdk.v2.serialization.messages	44.2 %	846	1,067	1,913
com.amazon.dtasdk.v2.serialization.serializer	73.8 %	59	21	80
com.amazon.dtasdk.v2.servlets	78.1 %	164	46	210
com.amazon.dtasdk.v2.signature	80.2 %	1,248	308	1,556
com.amazon.dtasdk.v2.utils	100.0 %	7	0	7
src/test/java	96.5 %	3,651	133	3,784
com.amazon.dtasdk.v2.connection	94.6 %	281	16	297
SignatureVerificationTest.java	93.8 %	244	16	260
TestServer.java	100.0 %	37	0	37
com.amazon.dtasdk.v2.serialization.messages	100.0 %	138	0	138
InstantAccessRequestTest.java	100.0 %	75	0	75
InstantAccessResponseTest.java	100.0 %	63	0	63
com.amazon.dtasdk.v2.serialization.serializer	80.6 %	348	84	432
JacksonSerializerTest.java	89.2 %	148	18	166
TestJson.java	75.2 %	200	66	266
com.amazon.dtasdk.v2.servlets	100.0 %	1,074	0	1,074
AccountLinkingServletTest.java	100.0 %	134	0	134
InstantAccessServletTest.java	100.0 %	466	0	466
PurchaseServletTest.java	100.0 %	474	0	474
com.amazon.dtasdk.v2.signature	98.2 %	1,789	33	1,822
AuthenticationHeaderParserTest.java	100.0 %	85	0	85
CredentialStoreTest.java	97.0 %	293	9	302
RequestTest.java	94.5 %	413	24	437
SignerTest.java	100.0 %	998	0	998
com.amazon.dtasdk.v2.utils	100.0 %	21	0	21

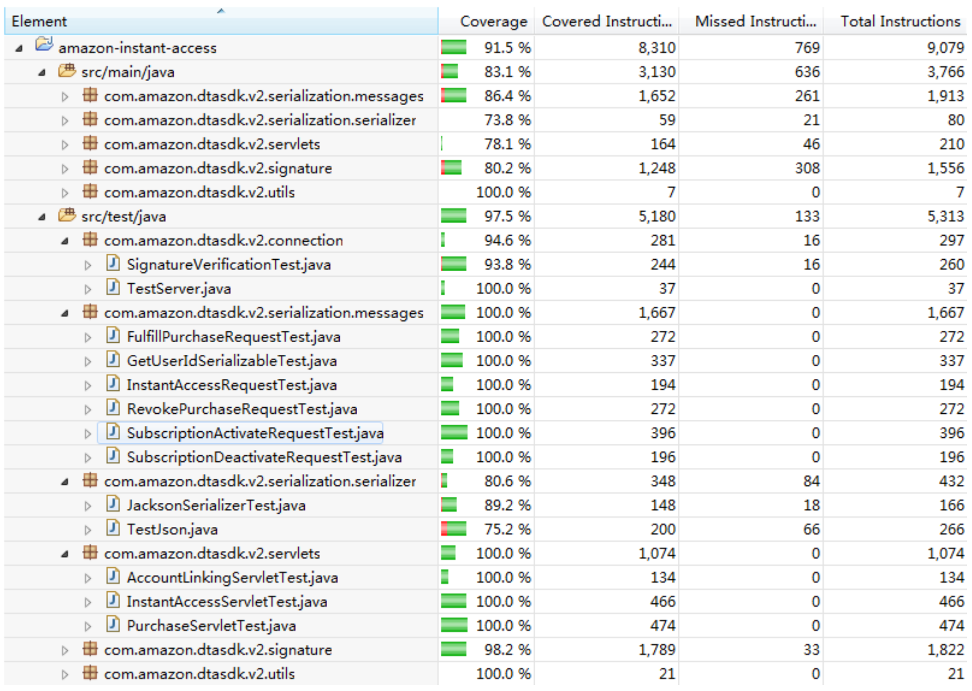
Exercise 5.3

- *Extend the test suite and write unit level test cases for uncovered features.*
- *Measure, report and analyze the new code coverage results of the extended test suite.*
- *Pay attention to the quality of your test cases and assertions. Document the ways in which you have tried to improve the quality of your test cases.*

Answer:

- Please check the new unit test cases in the file "unit-testcases" in GitHub: https://github.com/Martin-Ho/amazon-instant-access-sdk-java/tree/master/unit-testcases

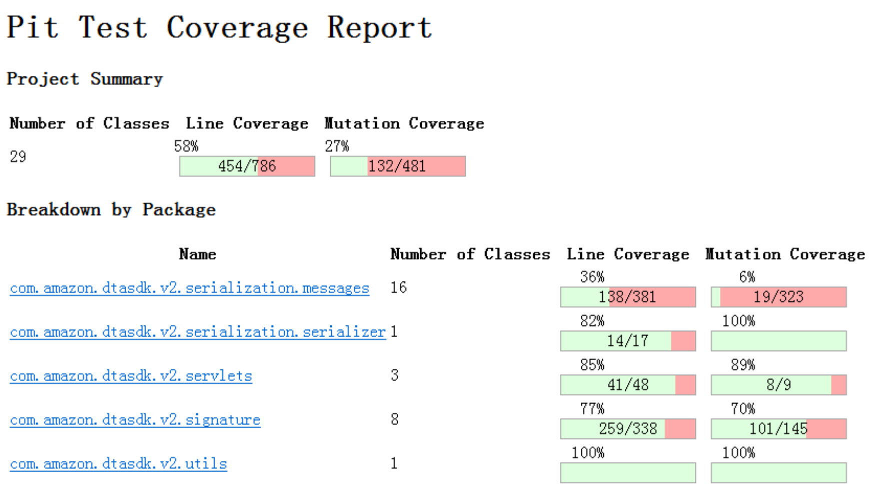
- The new code coverage results of the extended test suite is shown in the figure below. The new coverage for the whole project, for the production codes and for the test suite is increased to 91.5%, 83.1% and 97.5% respectively. And the new coverage of the production code package "serialization.messages" is largely improved to 86.4%, which means that most features in the request messages and the response messages have been covered.



Element	Coverage	Covered Instructi...	Missed Instructi...	Total Instructions
amazon-instant-access	91.5 %	8,310	769	9,079
src/main/java	83.1 %	3,130	636	3,766
com.amazon.dtasdk.v2.serialization.messages	86.4 %	1,652	261	1,913
com.amazon.dtasdk.v2.serialization.serializer	73.8 %	59	21	80
com.amazon.dtasdk.v2.servlets	78.1 %	164	46	210
com.amazon.dtasdk.v2.signature	80.2 %	1,248	308	1,556
com.amazon.dtasdk.v2.utils	100.0 %	7	0	7
src/test/java	97.5 %	5,180	133	5,313
com.amazon.dtasdk.v2.connection	94.6 %	281	16	297
SignatureVerificationTest.java	93.8 %	244	16	260
TestServer.java	100.0 %	37	0	37
com.amazon.dtasdk.v2.serialization.messages	100.0 %	1,667	0	1,667
FulfillPurchaseRequestTest.java	100.0 %	272	0	272
GetUserIdSerializableTest.java	100.0 %	337	0	337
InstantAccessRequestTest.java	100.0 %	194	0	194
RevokePurchaseRequestTest.java	100.0 %	272	0	272
SubscriptionActivateRequestTest.java	100.0 %	396	0	396
SubscriptionDeactivateRequestTest.java	100.0 %	196	0	196
com.amazon.dtasdk.v2.serialization.serializer	80.6 %	348	84	432
JacksonSerializerTest.java	89.2 %	148	18	166
TestJson.java	75.2 %	200	66	266
com.amazon.dtasdk.v2.servlets	100.0 %	1,074	0	1,074
AccountLinkingServletTest.java	100.0 %	134	0	134
InstantAccessServletTest.java	100.0 %	466	0	466
PurchaseServletTest.java	100.0 %	474	0	474
com.amazon.dtasdk.v2.signature	98.2 %	1,789	33	1,822
com.amazon.dtasdk.v2.utils	100.0 %	21	0	21

- I tried to improve the quality of my test cases in several ways.

  Firstly, I use PIT to do mutation testing to ensure that the quality of the test suite will not be worse when more test cases are added. The original PIT test coverage report is shown in the figure below. The report shows that the line coverage is 58%, and the mutation coverage is 27%. Especially, the line coverage and the mutation coverage of the package "serialization.messages" is only 36% and 6% respectively, which means that not only the number of the test cases are not enough, but also the quality of the test cases is not good enough.



Pit Test Coverage Report

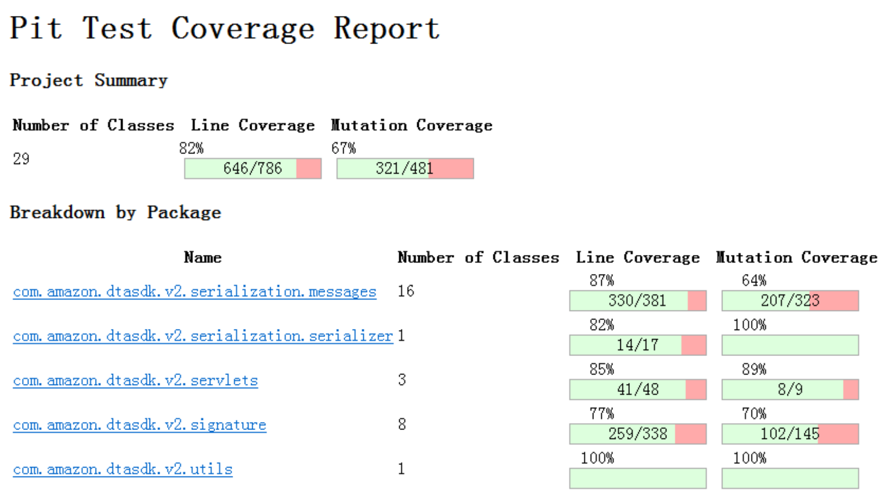
Project Summary

Number of Classes	Line Coverage	Mutation Coverage
29	58% 454/786	27% 132/481

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage
com.amazon.dtasdk.v2.serialization.messages	16	36% 138/381	6% 19/323
com.amazon.dtasdk.v2.serialization.serializer	1	82% 14/17	100%
com.amazon.dtasdk.v2.servlets	3	85% 41/48	89% 8/9
com.amazon.dtasdk.v2.signature	8	77% 259/338	70% 101/145
com.amazon.dtasdk.v2.utils	1	100%	100%

After the extension of the test suite, the new PIT test coverage report is shown in the figure below. The line coverage of the project is increased to 82%, and the mutation coverage is also increased to 67%. Most importantly, the line coverage and the mutation coverage of the package "serialization.messages" is largely improved to 87% and 64% respectively, which indicates that the quality of the test cases can reach to a certain level.



Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage
29	82% 646/786	67% 321/481

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage
com.amazon.dtasdk.v2.serialization.messages	16	87% 330/381	64% 207/323
com.amazon.dtasdk.v2.serialization.serializer	1	82% 14/17	100%
com.amazon.dtasdk.v2.servlets	3	85% 41/48	89% 8/9
com.amazon.dtasdk.v2.signature	8	77% 259/338	70% 102/145
com.amazon.dtasdk.v2.utils	1	100%	100%

Secondly, I used Delta Debugging and Spectrum-Based Fault Localization when I wrote the test cases. After running coverage testing in JUnit, I tried to find the differences between the passing test case and the failing test case. I also compared executed statements in each run and look for anomalies to infer the causes of failures.

Thirdly, I tried to detect the fine-grained redundancies after I finished writing my test cases. I removed some redundant and useless lines in code, in order to reduce the scale of the test suite and improve the efficiency of the codes. Last but not least, I check the test suite again to prevent the new test cases affecting the original test cases.

Exercise 5.4

- *Optionally, you can send a pull request to the original repository after extending the test suite. If your pull request is accepted, you will receive up to 5 extra marks. Please check https://help.github.com/ articles/creating-a-pull-request/ for learning about creating pull requests.*

Answer:

I have sent a pull request of the new test cases. Please open this URL to check the pull request:
https://github.com/amzn/amazon-instant-access-sdk-java/pulls