CS427 MP5: Reverse Engineering & Refactoring

September 2016

Context

In this MP, you will be working to improve existing software by finding code smells and removing them through refactoring and reverse engineering the code.

The provided software system is some simulation of a local area network, and contains all the functionality for a previous milestone. Your task is to look over this code and improve its design so that new functionality can be more easily added in the future. You are also provided with regression tests.

***** Note that Questions in italics are for you to think about. Questions in light magenta are for you to answer and submit.

Getting Started

You will first need to prepare the environment for subsequent tasks. Do a bare import of MP5 into your Intellij setting, instructions are on the MP5 webpage:

https://wiki.illinois.edu/wiki/display/cs427fa16/MP5

Skim the Documentation

First, take a look at the documentation that comes with the system.

Expand the javadoc folder and double-click on the index.html file. Read through the documentation.

What are your first impressions about the system? Where would you focus your refactoring efforts? Discuss with your group partner.

Read all the Code in 10 minutes

Next, confirm some of your initial impressions by reading the source code.

From within Intellij, read the code. Use the features of Intellij to help you navigate the code quickly. For instance, if you are in the middle of a portion of code and want to see where a class/method is defined, hold on ctrl (or the apple key) and click on the word.

What are your first impressions about the system? Where would you focus your refactoring efforts? Discuss with your group partner.

Do a Mock Installation

Finally, try to run the code and the regression tests that come along with it.

1. Try to compile and run the LANSimulation.java file.

    (a) In the Project structure window, expand the lanSimulation package

    (b) Right click on the LANSimulation.java file > Run 'LanSimulation.main()'

    (c) The first run should fail because you don't have any parameter for this class. You should be able to open the configuration setting on the right top.

    (d) You need to pass the character "s" as a program argument and turn on assertions on the JVM using the "-ea" switch. Please see Figure 1 (how to find "edit configuration") and Figure 2(pass parameters) in next page.

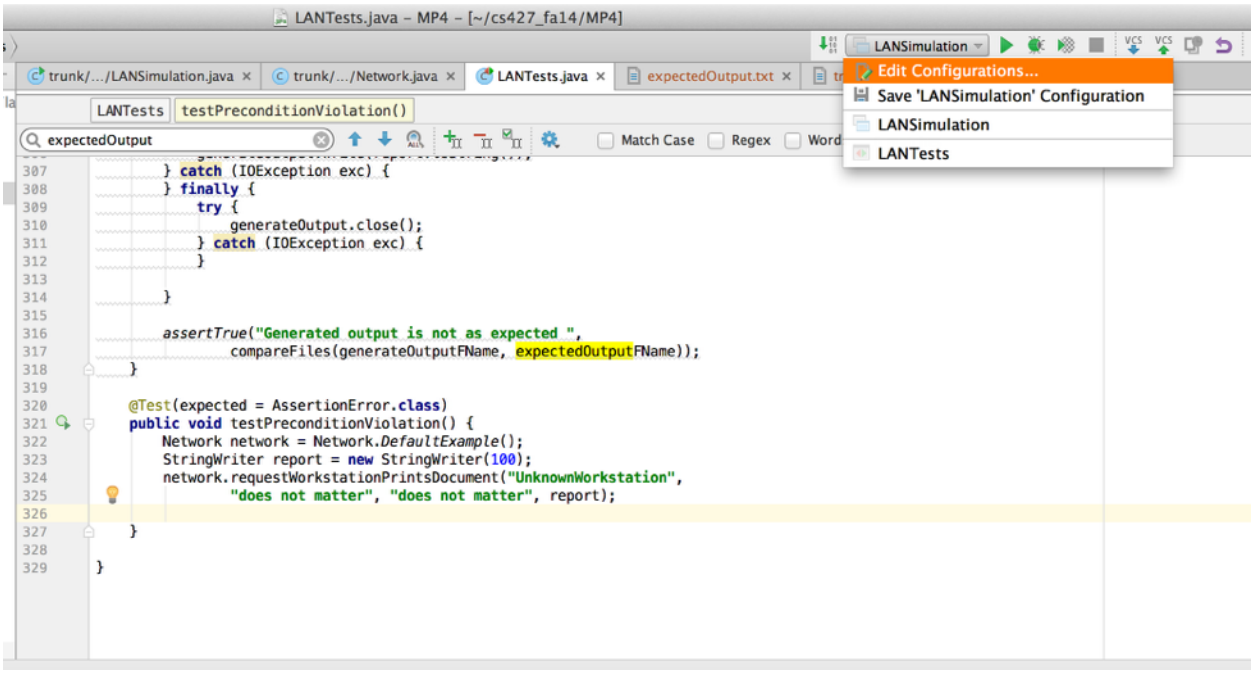

Figure 1

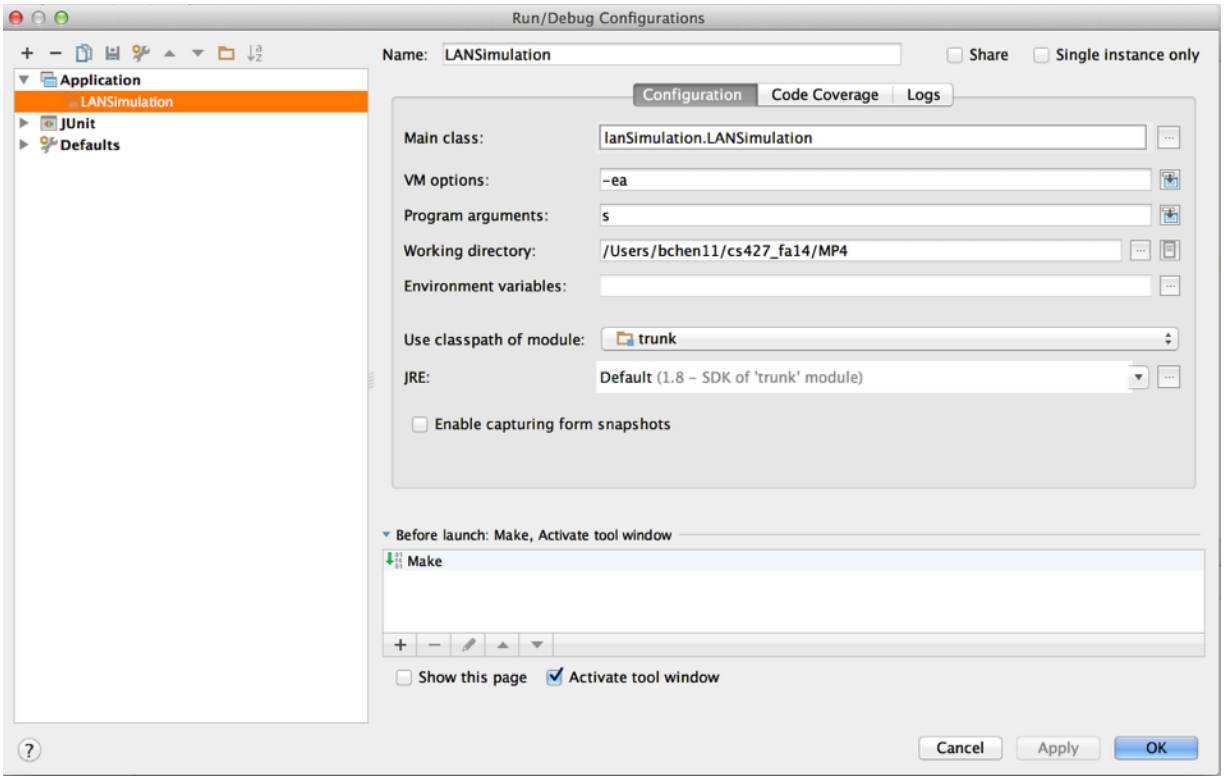

Figure 2

2. The first time you run the tests, one of them will fail. You need to perform an additional step manually to get it to pass. You MUST only do this once throughout this MP. You should not need to repeat this process throughout this MP unless change your workspace or Intellij installation. Here's what you need to do:

   (a) After the first run of tests in the LANTests.java file, a file called useOutput.txt should appear in the Package Explorer view. Rename it to expectedOutput.txt

   (b) Run the tests again now. They should all pass.

   (c) Commit the expectedOutput.txt file to your git repository.

   (d) IMPORTANT: Do NOT change this file again in the rest of this MP.

3. Change a few lines here and there in the code to verify whether the regression tests do test the code you are looking at. This step is important so that you can verify if that part of the code indeed does what you think it should.

Do you feel that the code base is ready to be refactored? How about the quality of the regression tests: can you safely start to refactor? Discuss with your group partner.

For the next few sections, use the automated refactoring tools in Intellij to perform the refactorings whenever possible, although there are some refactorings that you still have to perform by hand. To use Intellij support when in a Java file, you can [a](left-)click on the Refactor menu and select the appropriate refactoring, [b](right-)click on an appropriate program element and select the appropriate refactoring, or [c] learn the keyboard shortcut (e.g., the one for rename is quite useful).

Extract Method

One of the things you might have seen is that there is a considerable amount of duplicated code inside the class Network. In this section you will be getting rid of some of the duplicate code.

Q1. What code smell(s) do you detect? Mention these code smells by name and give concrete example(s) from the code.

1. The logging code occurs three times, twice inside requestWorkstationPrintsDocument and once inside requestBroadcast. Get rid of the clones by applying an Extract Method.

2. The accounting code occurs twice within printDocument. Get rid of the clones by applying an Extract Method.

3. Is there any other duplicated code representing important domain logic which should be refactored? Can you refactor it using Extract Method?

Are you confident that these refactorings did not break the code? Do you believe that these refactorings are worthwhile? Does the tool do a good job? Discuss with your group partner.

Before you go on, please stop and commit your work. Use the commit message, "Completed the Extract Method Section"

Move Behavior Close to Data

Having extracted the above methods, you note that none of them is referring to attributes defined on the class Network, the class these methods are defined upon. On the other hand, these methods do access public fields from the class Node and Packet.

Q2. What code smell(s) do you detect? Mention these code smells by name and give concrete example(s) from the code.

1. The logging method you just extracted does not belong in Network because most of the data it accesses belongs in another class. Apply a Move Method to define the behavior closer to the data it operates on.

2. Similarly, the printDocument method is also one that accesses attributes from two faraway classes, yet does not access its own attributes.

3. Are there any other methods that are better moved closer to the data they operate on? If so, apply Move Method until you're satisfied with the results.

Are you confident that these refactorings did not break the code? Do you believe that these refactorings are worthwhile? Does the tool do a good job? Discuss with your group partner.

Before you go on, please stop and commit your work. Use the commit message, "Completed the Move Behavior Close to Data Section"

Eliminate Navigation Code

There is still a piece of duplicated logic left in the code, namely the way we follow the nextNode pointers until we cycled trough the network; logic which is duplicated both in requestWorkstationPrintsDocument and requestBroadcast (and to a lesser degree in printOn, printHTMLOn, printXMLOn). This duplicated logic is quite vulnerable, because it accesses attributes defined on another class and in fact it represents a special kind of navigation code.

Q3. What code smell(s) do you detect? Mention these code smells by name and give concrete example(s) from the code.

Apply an Extract Method on the boolean expression defining the end of the loop, creating a predicate atDestination.

Are you confident that these refactorings did not break the code? Do you believe that these refactorings are worthwhile? Does the tool do a good job? Discuss with your group partner.

Before you go on, please stop and commit your work. Use the commit message, “Completed the Eliminate Navigation Code Section”

Transform Self Type Checks

Another striking piece of duplicated logic can be found in printOn, printHTMLOn, printXMLOn. However, this time it is a duplicated conditional. If extra functionality (such as a Router node) is added, this logic will need to be changed in multiple locations. Thus it is worthwhile to introduce new subclasses here.

Q4. What code smell(s) do you detect? Mention these code smells by name and give concrete example(s) from the code.

1. Normally, you should have noticed during Move Behavior Close to Data, that the switch statements inside printOn, printHTMLOn, printXMLOn should have been extracted and moved onto the class Node. If you haven’t done that, do it now; and name the new methods printOn, printHTMLOn, printXMLOn.

2. Create empty subclasses for the different types of Node that do exist (WorkStation, Printer).

3. Patch the constructor of the new subclasses so that they now create instances of the appropriate class.

4. Move the code from the legs of the conditional into the appropriate (sub)class, eventually removing the conditional.

5. Verify all accesses to the type attribute of Node. As long as you find any keep doing a Transform Self Type Checks or Transform Client Type Checks until you completely removed them all.

6. Remove the type attribute.

Are you confident that these refactorings did not break the code? Do you believe that these refactorings are worthwhile? Does the tool do a good job? Discuss with your group partner.

Please commit the final state of your project, including the useOutput.txt and expectedOutput.txt files. If you do not see these files at first, (right)-click on the MP5 project in the workspace and select Refreshing. Use the commit message, “Completed the Transform SelfType Checks Section”

IMPORTANT: Turn in your work according to the instructions given in the Submitting your code section of MP5 page of the course wiki. The URL for the wiki instructions for MP5 is repeated below, for your convenience.

https://wiki.cites.illinois.edu/wiki/display/cs427fa16/MP5