CPSC 2151

Lab 7

Due: Friday October 19th at 11:59 pm

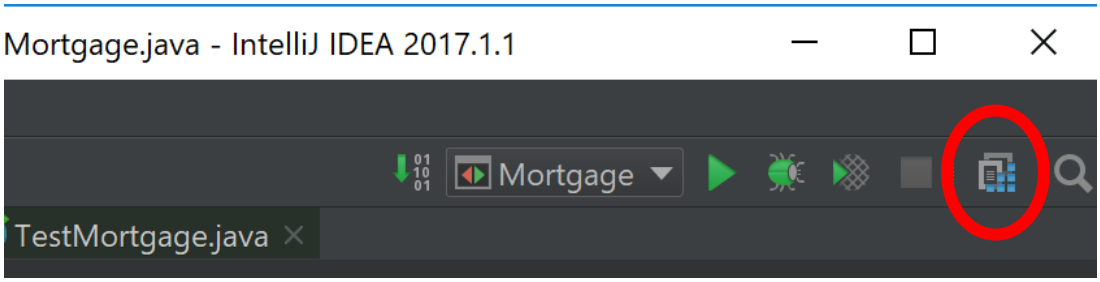
In this lab you will be working with Test Driven Development to complete the Mortgages project we worked on in Lab 3. You are provided with a TestMortgage class, which will provide several test JUnit test cases for the Mortgage class. We do not have test cases for each method, just the "difficult" methods. We will discuss JUnit and how to create JUnit test cases in more detail in class, but this will be a good introduction. Every test case in JUnit is it's own function. When you run the JUnit class, it will run all the test cases, and report which one failed. If you have any failed test cases, you can go to that specific function, and see what input was used to see why your program failed.

This is what's called Test Driven Development because we developed our test cases first, and we can tell when we have completed the code correctly by whether or not all of our test cases pass. As you work on the code, if you think you have the function correctly completed, you can run the test suite and see if the test cases for that function pass. Note: in order to test some of these functions, it is assumed that you have completed simpler functions such as the constructor and the get functions. Those functions will be called as well as the function being tested. Also, JUnit does not allow us to directly call private functions, so we can only test those indirectly through publicly available functions. Luckily all the test cases have been provided for you in this lab.

Please follow the set up instructions carefully to ensure that Junit is set up correctly.

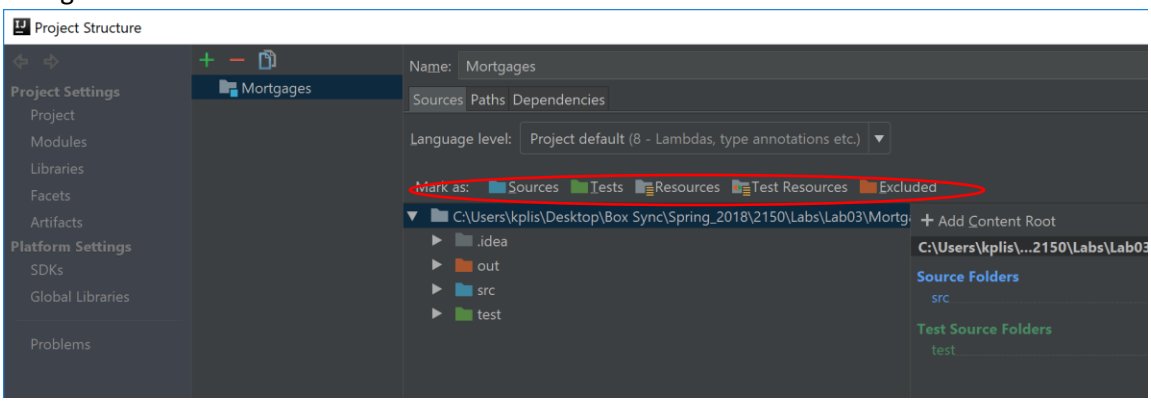
You will need to perform some setup to use JUnit in your intelliJ project:

1. Start by opening up your completed project from lab 3. You don't have to worry about changing the package name. We'll leave it as cpsc2150.labs.lab2.
2. In your navigation panel on the left side of the screen, right click on the project folder. This should be the highest level folder you can see, and it should be called Mortgages. Right click on it and select new->directory. Name your new directory "test". This directory should appear in your navigation pane below your source folder.
3. Now click on the project structure button on the top right corner of the intellij window. It should be right next to your search icon.



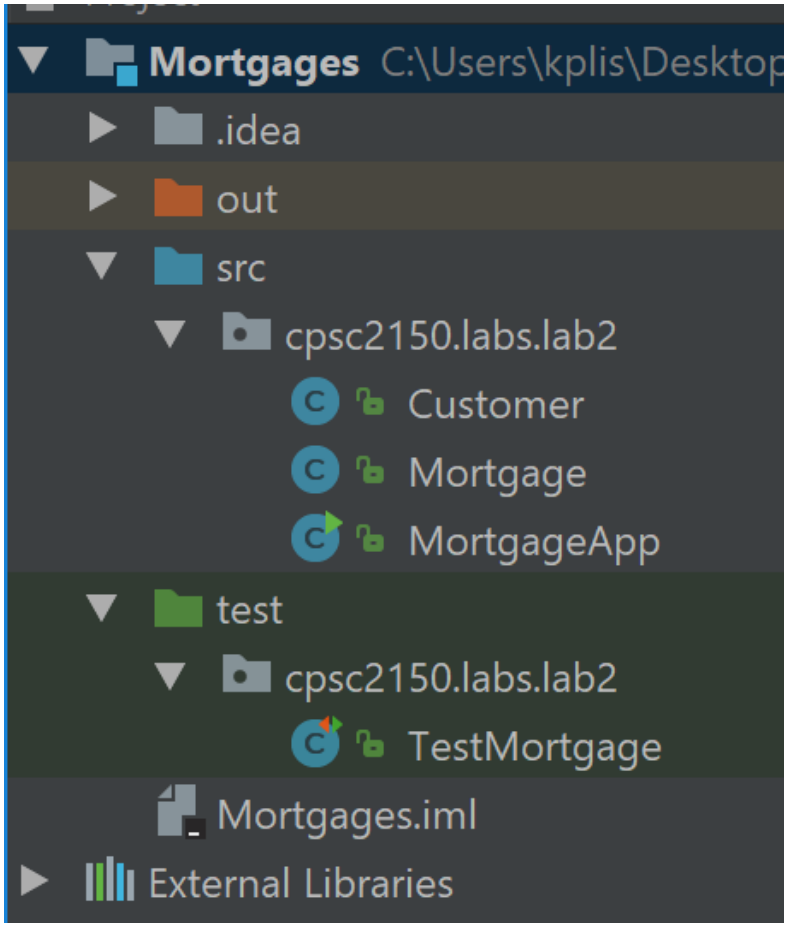
   a. On the left side of the new window, select Modules
   b. If it is not already on the "Sources" tab, select that tab. You should now see a list of your folders, such as src and test
   c. Select the test folder. Above the list of folders is a "Mark Source As" option. Select the "Tests" option to mark your test folder as a repository for tests. Your test folder should

turn green.



d. Hit OK to exit the project Structure window

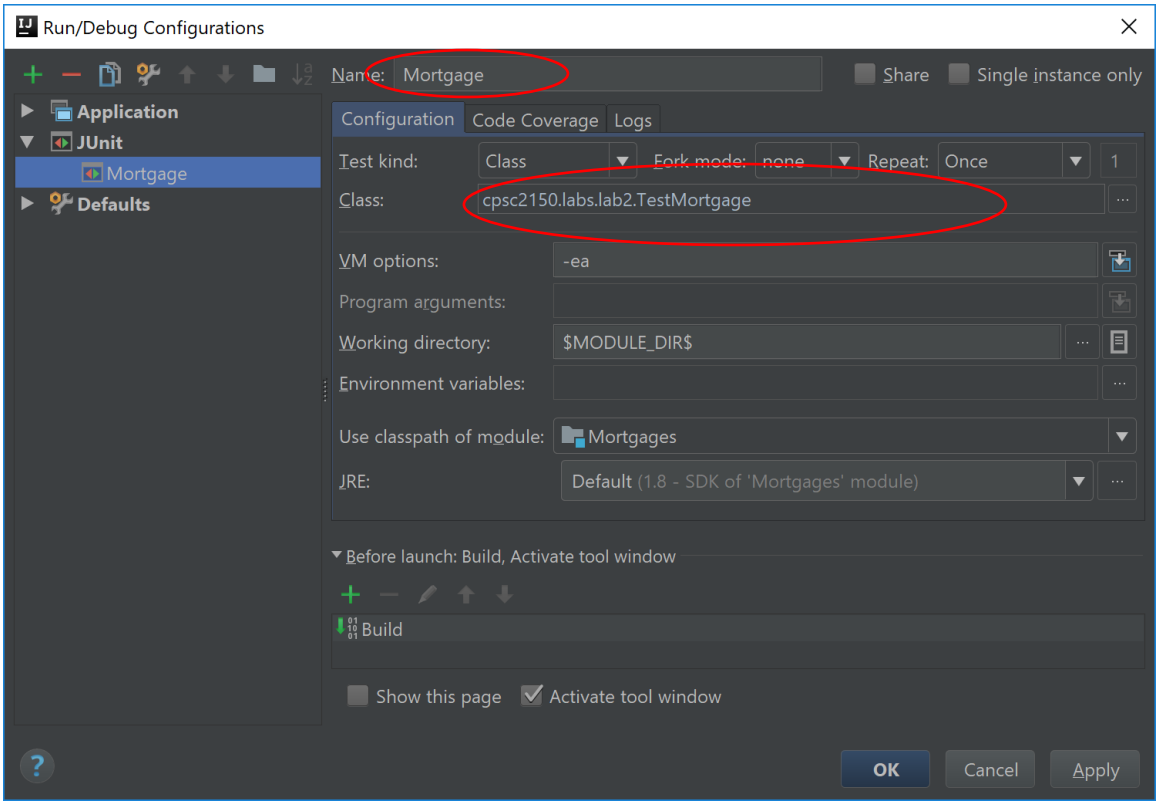
4. In the navigation pane, right click on your test folder, and add a package to it. Use the same package name as the src folder (cpsc2150.labs.lab2)
5. Add a class called “TestMortgage” to the new package in the test folder. It’s a good practice to name your JUnit classes Test<CLASSNAME> to keep our naming consistent. Your file structure should look like:



6. Copy the code I provided into the TestMortgage class.
7. Most likely IntelliJ will have an issue with the @Test annotation (among other things). This is because it doesn’t know we are trying to use JUnit. Click on the @Test annotation and hit alt +

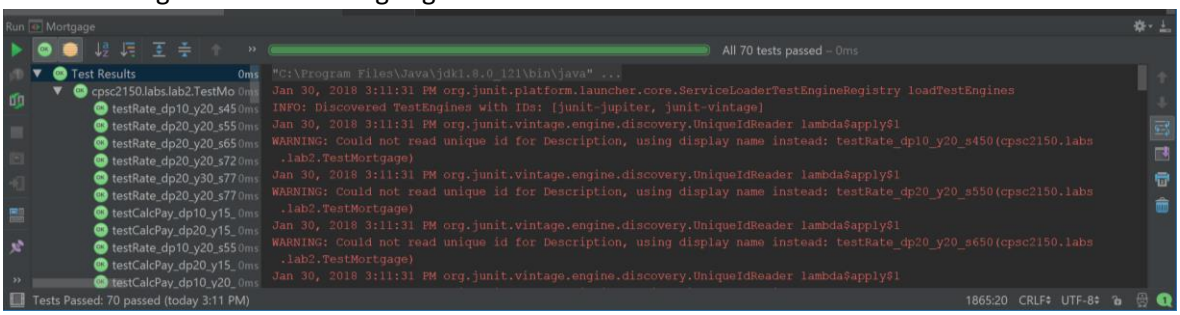
enter. If one of the options is to add JUnit4 to the classpath, congratulations! You can select that option and skip ahead. To step 9. I suspect that won't happen

8. Assuming you were not able to add JUnit in the previous step follow these instruction
    a. Open the project structure window again
    b. Select "libraries" on the left
    c. Click on the green plus sign to add a new library, select "from maven"
    d. A search box will appear. Type in "junit:junit:4" and hit search
    e. One of the options that pops up should be junit:junit:4.10. Select that one and hit ok
    f. Select to add it to the current project
    g. Hit ok at the bottom of the project structure window
    h. It should now recognize your junit statements
    i. If it does not, select @test with your cursor and alt + enter should now add the junit statements.
9. We should now have our test cases set up, but we still need to add a configuration to run the JUnit tests
10. Click on your configuration selector (top right corner of the screen, next to the run button) and select Edit Configuration
11. Click on the plus icon to add a new configuration, and select the type as JUnit
12. In the menu that pops up, calls your new configuration TestMortgage
13. In the class option, use the selector (button with 3 dots) to select the TestMortgage class



14. Hit OK to save this new configuration. You can now run your configuration. You hope to see a green bar saying all of your tests have passed. If not, it will tell you the name of the tests that

failed. You'll get a lot or warnings. Ignore those.



Once you have your JUnit class set up, you can begin working on your code. If you did not complete the contracts in Lab 3 (or had issues with them) you will want to do that now. Once your Contracts are ready, you can start coding in the Mortgage class. Start with the constructor and the get functions, as they should be straight forward. Then work on your calcRate, calcPayment, and loanApproved in that order. As you complete one function, run your TestMortgage class and see if the test cases for that function pass. Once you have a fully green bar, you should be done working on the Mortgage class.

After you finish the Mortgage class, complete the Customer class, using the mortgage class. These functions should all be relatively straight forward, as most of the decision making is handled in the Mortgage class. There are no provided test cases for the Customer class.

Once both classes have finished, change the Main function in MortgageApp to prompt the user for a Customer's information (including input validation) and the loan application information (also with input validation. Remember at this point you will be the client of the Customer and Mortgage classes, so remember your responsibilities as laid out in the contracts you wrote. Once you have the information from the user, print off the customer and loan information using the Customer.toString() method. Make sure to change configurations to run the MortgageApp.java code. Your program does not need to keep asking for more customer and mortgage information, just one time is enough.

You should not make any changes to the code in TestMortgage.java.

Customer.toString()

In the original starter file, there was an error in toString that could cause it to fail if you called toString without applying for a loan. Please use this correction, which checks to see if the loan is null.

```
*/
    @Override
    public String toString()
    {
        String str = "";
        str += "Name: " + name + "\n";
        str += "Income: $" + income + "\n";
        str += "Credit Score: " + creditScore + "\n";
        str += "Monthly Debt: $" + monthlyDebtPayments + "\n";
        str += "Mortgage info: \n";
        if( loan != null) {
            str += loan.toString();
        }
```

```
        return str;

    }
```

General requirements and tips

- Remember our best practices we've discussed in class
- No magic numbers, use static final variables
- Not all of our constants need to be publicly available, some of them are just for internal use in the class
- Make sure you follow the contracts you have provided
- Make sure you update your contracts if you add in public static final variables
- Remember to comment your code. Javadoc comments and contracts are a good start, and may be enough for simpler functions, but also comment your code as well when needed.
- You must provide a make file for your code as well. Your make file does not need to run the Junit code.

Running Junit on Unix Machines

While your program code needs to run on unix, we have not covered Junit in enough detail to require running that on unix. We will cover that next week in lab.

Groups

You may, but are not required to, work with a partner on this lab. Your partner must be in the same lab section as you, not just the same lecture section. If you work with a partner, only one person should submit the assignment. You should put the names of both partners in a comment at the top of the file in order for both partners to get credit. This assignment may take more than just the lab time. Make sure you are able to meet outside of class to work on the assignment before you decide to work with someone else. Remember to actively collaborate and communicate with your partner. Trying to just divide up the work evenly will be problematic.

If you worked with a partner in lab 3, you do not have to work with the same partner. If you did not work with a partner in lab 3, you can choose to work with a partner on this lab. In the event that you are partnering with a new person for this lab and you both have different submissions for lab 3, choose the best submission to start with.

Before Submitting

You should make sure your MortgageApp program will run on Unix before you submit. Make sure you correctly set up the directory structure to match the package name. You should provide a makefile for your code as well

Submitting your file

You will submit your files using handin in the lab section you are enrolled in. You need to submit a zipped directory with your package directory and your make file in it. Include the TestMortgage.Java file in your package directory with your Mortgage.java, Customer.java, and MortgageApp.java code files.

If you are unfamiliar with handin, more information is available at https://handin.cs.clemson.edu/help/students/

Checklist

- Does my code pass all of the provide JUnit Tests?
- Are my contracts provided, complete and correct?
- Did I update my Mortgage App class to get input from the user?
- Did I validate my user input?
- Did I follow Design by Contract?
- Did I follow best practices?
- Did I avoid magic numbers?
- Is my code well commented?
- Is my code well formatted?
- Did I provide a make file?
- Does my code run on Unix?
- Is my directory structure correct?
- Can the TA unzip my submission, and then run my program using the make file?