Background:

Please download,extract, and import the Dictionary project on Blackboard (under the Projects tab). Remember you must import pom.xml, not the entire extracted folder. A simple demo can be found (and executed) in Main.java. This program allows a user to add words/definitions to a dictionary. A user can also find definitions of words that have already been added to the dictionary. A user is also able to save the state of the dictionary into a file and subsequently load it from a file. The current code will reject any **word** that is NOT alphabetic (i.e., no symbols, numbers, or spaces). The **definition** of a word must also be alphabetic but is allowed to include spaces.

To better understand the Dictionary implementation, it may help to examine the JUnit tests written to verify the correctness of the Dictionary, which can be found in DictionaryTest.java. Additionally, try to read through Main.java, which is the demo application.

Questions:

1) The Dictionary allows you to save the current state to a file. What is the name of the file that this program saves to your hard drive?

2) What is the file extension of the file that is saved?

3) Where is it located on your hard drive? **Hint:** There is a special name for the folder we are saving to.

4) Run the program and save a few word/definitions pairs to file. Open the file with a development-friendly editor (e.g., Notepad++). Technically, you can use IntelliJ by going to File > Open. Take a screenshot of the opened file and paste it along with your solutions to the questions above.

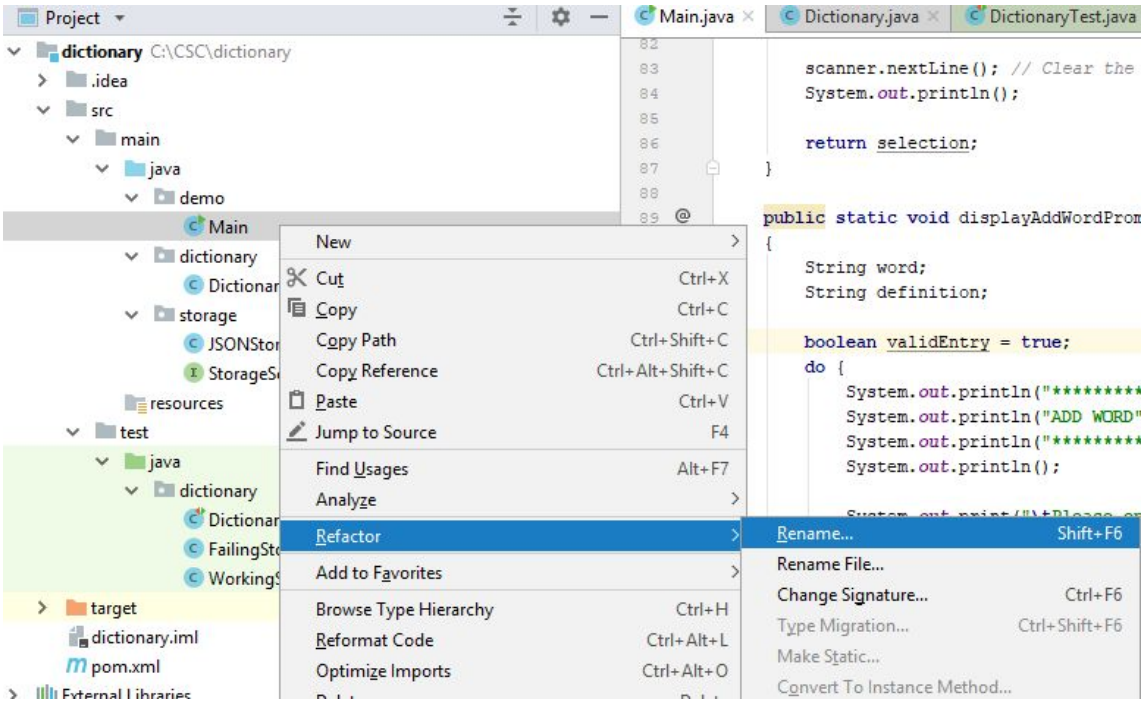
Submission Details:

Please **submit the questions along with your answers as a Microsoft Word document or a PDF** and **place this inside the Maven resources folder** of your project. Remember that this is `src/main/resources`. We will be placing other answers in this same document.

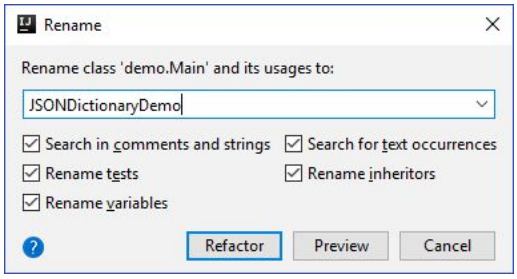
Refactoring

Refactoring is a term often used when developers wish to clean up certain code. For the Dictionary project, there is a demo that lives in the **demo** package. The demo class is currently named Main.java. **This isn't a descriptive name for this class.** We would like to **change the class name to JSONDictionaryDemo.java**. We wish to rename this class using special features built into IntelliJ. **Please note that if you don't use IntelliJ to do this, things WILL break.**

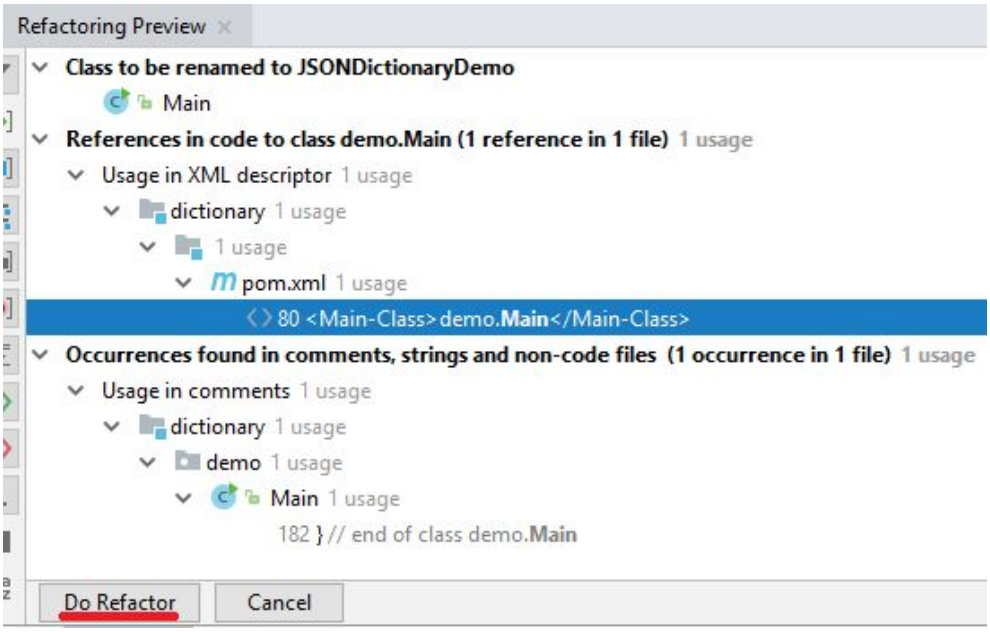
Right-Click on Main.java in the Project View and go to Refactor > Rename



When you are asked to choose a new name for the class, please use JSONDictionaryDemo.java and then, click Refactor.



IntelliJ understands that the original name of Main.java **may need to be changed in a variety of places.** Please see the following 'Refactoring Preview' that opens up in the bottom left of your screen after you pressed Refactor on the prior screen.



Click 'Do Refactor' to fix/rename all occurrences of the old Main class that IntelliJ is aware of.

Submission Details:

You will eventually zip up your entire project and upload it as a submission. If you did not rename the original Main class to JSONDictionaryDemo as described above, points will be deducted accordingly.

Bug 1

There is a bug in this project that should be fixed. In order to replicate the issue, please launch the original demo (which should now be named JSONDictionaryDemo.java) **without** loading an existing dictionary file. If you then attempt to search for a word that doesn't exist, an exception is thrown and not caught, ending your program; this is bad. There is a cause and there is an effect. The effect is that an exception is thrown; **this is not where your investigative efforts should go**. You should figure out what the underlying **cause** is and write the **code to remedy the issue.**

Tests Affected/Needed:

This will require writing a new test. This new test should **prove** via an assert that an exception is no longer thrown when Dictionary.findDefinition() is called for a word that is not in the Dictionary. You should consider using one of the existing Mock classes. Choose an adequate name for this test.

Submission Details:

In your Word Doc / PDF that will go in `src/main/resources`, please briefly describe how you fixed Bug 1 and please mention the name of the JUnit you've added to DictionaryTest.java.

Bug 2

There is a bug in our original demo, now called JSONDictionaryDemo.java. You will need to write code to fix it.

To replicate the problem, try to add an invalid word (e.g., a word with a number in it like apple2). You will find that the program will **never** go back to the main menu after entering an invalid word, **even if** you enter valid words/definitions immediate after an invalid word/definition.

Classes Affected:

JSONDictionaryDemo

Submission Details:

After you fix the bug, please include a **plain-english description of how you fixed Bug 2 in the Word Doc / PDF that you are creating**. You do not need to write any tests for this solution. If there is no description of how you solved the problem, **you will receive no credit.** The description does not need to be long, but it should be enough for me to find the code you wrote to solve the issue.

Bug 3

In Dictionary.addEntry() and Dictionary.findDefinition(), we do a check for the validity of the inputs. As-is, the validity of input word is inconsistently checked between these two methods.

Since the same input validation should be done in all places, it is better to create private functions within the Dictionary class to do this.

Classes Affected:

Dictionary

Tests Affected:

There should be no tests affected. You do NOT need to write unit tests for private methods. None of the interface public methods should change to support this fix. We are just moving some of the implementation to separate functions such that this logic can be called in multiple places.

Submission Details:

In your Word Doc / PDF, wou must mention the name of the private function you created to validate the word for Bug 3.

Design Issue 1

In the JSONStorageService, we have two static methods. One static function throws an exception while the other doesn't. For consistency, you should make both throw exceptions. Specifically, convertMapToJsonString() shouldn't return "" if it couldn't convert, since this could lead to a silent failure.

Classes Affected:

JSONStorageService
Dictionary

Tests Affected:

None - we are not testing JSONStorageService

Submission Details:

In your final zip file, your class should be fixed as described above.

Design Issue 2

Consider that our Dictionary has a dependency on StorageService. The StorageService, helps us store dictionary information to our hard disk. Our JSONDictionaryDemo utilizes JSONStorageService, which is a derived class of the StorageService interface.

In Dictionary.loadFile(), notice that we are completely delegating responsibility to the underlying StorageService about how to convert a JSON data file into a Java Map. From the Dictionary's perspective, it has **no idea** that the StorageService "thinks" in JSON when it loads a file. This is a good thing - the less a class needs to assume/understand about its dependencies, the better.

With that said, notice that Dictionary.saveFile() is logically inconsistent with Dictionary.loadFile(). The Dictionary's saveFile() method is forced to convert a Map to a JSON string ***prior*** to calling the StorageService's underlying saveFile() method. In other words, the Dictionary class needs to know that the underlying StorageService is using JSON. Why? Because the JSONStorageService.saveFile() method accepts a precomputed json String as a parameter. **This is not a great design.** Why should our Dictionary class need to worry about converting maps to JSON (or any format like XML, CSV) before calling the StorageService? **StorageService should take care of this...**

When attempting to save a file, the Dictionary class should be able to just pass a Map of <String,String> to the StorageService and the StorageService should convert this to a JSON string behind the scenes. Please fix this inconsistency between the Dictionary's loadFile() and saveFile().

Note that the underlying issue is actually the StorageService interface's saveFile(), which should accept a Map<String,String>, not a precomputed json String. This will have a ripple effect that will force you to fix a few other classes, including Dictionary.java and the 2 mock classes.

Classes that will be affected:

StorageService and all Derived Classes.
Dictionary

Submission Details:

In your final zip file, your StorageService interface should be fixed, as described above.
Please explain your solution to 'Design Issue 2' in the Word Doc / PDF.

Feature 1

Currently, our JSONDictionaryDemo demonstrates how we can store our Dictionary as a readable/writeable JSON file. Imagine that a new feature is requested such that our project can also store the Dictionary as an XML file. The goal is to create a new demo that uses XML as the underlying storage format. Note that I am not asking the user to choose either JSON or XML; if the user wants to utilize XML, they will need to launch our separate XML-based demo. From the user's perspective, all of the prompts for both demos will be identical and they will not know the difference.

You will need to create **XMLDictionaryDemo.java** such that it is utilizing a new derived class (which derives from the StorageService interface) named **XMLStorageService.java**. Please note that the existing **JSONDictionaryDemo.java must still work** once you are done with the **XMLDictionaryDemo.java.** After you are done, all of the existing tests in DictionaryTest.java should still work, although you may need to refactor things.

You may need to generalize aspects of Dictionary.java. For example:

1. The Dictionary class currently assumes it will always save information to a file with a .json extension. This should be more flexible. Please solve this in the code and describe your solution in the Word Doc / PDF.

2. By the time you are finished, you will have two demos, one utilizing JSON and one utilizing XML. From the user's perspective, all of the prompts will be identical. You should avoid copying/pasting code. Note that there are many ways to avoid copying/pasting code, including implementation inheritance. If you are not sure you are on the right track, speak to your instructor.

Remember that currently, we are making heavy use of various libraries to (1) convert to/read from JSON and (2) save/load to file. It is worth reviewing how this works in the current project. In our Dictionary project, we are utilizing certain dependencies from the Jackson Open Source project. If you examine our current pom.xml, you will notice there are currently two libraries we are using for JSON capability. You should know that the Jackson project also provides XML capabilities, although you may need to do some Google searching for how to get this working. **Please speak to your instructor as soon as possible if you have questions.**

Submission Details:

In your Word Doc / PDF, explain your thought process/solution. It should be written to help me understand your decision making process.