## Project Stage 6: Implementing passage tests and dealing with multiple versions

For this project you ***must*** work in *groups of 3 (or 2 in a few cases)*. You must not partition the work, but work together (using "pair" programming) as you implement the tests (Part 1) and answer the questions (Part 2). Include the names of all group members in the comments, and one person per group should submit their file. Each team member is responsible for the entire test suite, including ensuring that the tests are all correct and that it is handed in (Part 1). Similarly, create a single write-up for your group for the multiple version testing plan (Part 2).

### Part 1

Test the `getReferencesForPassage` method of ***BibleReaderModel*** by implementing a test suite based on the references listed below. Place your tests in class ***TestGetReferencesForPassage*** in your **bibleReader.tests** package (You must name it exactly this!). To do the tests, you will call `getReferencesForPassage` to get a list of the proper references and then call `getVerses` to get the verses from the KJV bible that have those references. Then you will test those against the expected results. We test the results of the `VerseLists` because it turns out to be easier to write the tests this way (See the hints/tips).

The tests are grouped according to how a reference will be parsed and some of the expected areas of difficulty. Some of them include a comment about why they were chosen. Create a test method with a sensible name for each group of tests. Your tests will need to compare the result from ***BibleReaderModel*** with the actual expected results, so you will need to determine what the actual results are.

#### Valid references

- The following specify a single verse and are correct.
  - John 3 : 16 (spaces around the colon)
  - Gen 1:1 (first verse in Bible)
  - Revelation 22:21 (last verse in Bible)
- The following specify verses from a single chapter
  - Ecclesiastes 3 : 1 - 8 (spaces everywhere, including one at beginning and end)
  - Joshua 24:28-33 (last chapter of a book)
  - Psalm 23:1-6 (whole chapter)
- The following specify one or more whole chapters
  - Song of Solomon 3
  - Revelation 22 (the last chapter of the Bible)
  - 1 Tim 2-4 (contains a space in the book name)
  - 1 John 2-3
- The following specify verses from multiple chapters
  - Isa 52:13 - 53:12 (space around the "-")
  - Mal 3:6-4:6 (to the end of the book)
- The following specify a whole book
  - 1 Kings
  - Philemon

- The following specify multiple verses from multiple chapters using the oddest syntax
    - Ephesians 5-6:9
    - Hebrews 11-12:2

**Invalid references**

- The following contain an invalid book, chapter, and/or verse.
    - Jude 2
    - Herman 2
    - John 3:163
    - Mal 13:6-24:7
- The following contain chapters/verses out of order or invalid syntax (e.g. missing numbers).
    - 1Tim 3-2
    - Deut :2-3
    - Josh 6:4- :6
    - Ruth : - :
    - 2 Sam : 4-7 :
    - Ephesians 5:2,4 (someone might want to enter a comma)
    - John 3;16 (semicolon instead of colon)

One member from your group should submit ***TestGetReferencesForPassage.java*** using Handin under assignment **235-P6**. The autograder will run your tests against my (correct) code and give you the results. Grades for this part will be based on the correctness and implementation details of the tests (thus, your tests passing is not enough to earn a good grade since your tests may or may not be testing the correct things).

**Hints/Tips for Part 1**

- The most important tip: If you read all of these hints/tips and put in the effort to understand what I am suggesting, you will save a lot of time and effort both now and in the future.
- Make sure ***everyone*** looks at the final test suite to verify that all of the tests look correct.
- The easiest way to start is to copy **Stage05BibleReaderModelTest**. Remove all of the methods with the @Test annotation, but keep the rest. Refer to the test methods from the original file to get ideas about how to write your tests.
- *Make sure your tests call methods on the **model** object, not the Bible object.*
- We will assume that BibleIO works correctly. Thus, you can use the *versesFromFile* ArrayList in your tests to get your "expected" results.
- Your tests should use verses from **kjv.atv**. In light of this, the following "helper" might be useful in your test class:

```
public VerseList getVersesForReference(String reference) {
      ArrayList<Reference> list =
                              model.getReferencesForPassage(reference);
      VerseList results = model.getVerses("KJV", list);
      return results;
}
```

***Do not*** include an `@Test` annotation for this method! You do not want it to run as a test.

- You can open the **kjv.atv** file in Eclipse. Since there is one verse per line and they start at line 2, the index of a verse in *versesFromFile* is 2 less than the line number. Thus, you should be able to easily write tests to determine whether a whole passage was correctly returned by comparing them with a range of verses from *versesFromFile*.
- Writing your tests will be *really easy* if you consider how the following methods will help: **subList** and **toArray** on ***ArrayList*** and **assertArrayEquals** from JUnit.
- *Do NOT* use assertEquals on two arrays. It might work, but ***assertArrayEquals*** is more appropriate and WILL work.
- If you think before you start writing your tests, it will save you a lot of work!
- If your test code is really long for each test, do more thinking! Each passage should only take 2-5 lines of code to test. (Hint: Code reuse!)

**Part 2**

We need to eventually write tests for the **BibleModel** class for when we have multiple versions. But first we need to figure out how the model should behave. *As a group, provide* ***thoughtful answers*** *to the following questions.*

1. Sometimes a word will be in one version and not another. So if I do a word search with multiple versions, there will be different results for each version. In the model, should we store the actual results for each version or a merged list? If not, how should we store the results? In your answer, discuss the benefits and problems with each choice.
2. It would seem that passage lookup should be easier than searching since every version should have all of the verses in a given passage, right? Unfortunately, that is not the case. There are some instances when one or more verses does not appear in a translation. How should passage lookup be handled in this case? Is it a valid lookup if it is valid in any of the translations or does it have to be in *all* of them? If only one of them has to be valid, what should the result be for the versions that don't contain all of the verses—no result, or the portion of the passage that is in the given translation?
3. The last question overlooked a subtle problem: What if the missing verse is in the middle of a passage? Then each version will return a result, but they will not necessarily "line up". Do we need to deal with this in the model, or let the outside world (e.g. the GUI) deal with it?
4. Overarching all of these questions is this: When there are differences between the results from different versions, who should handle the details of combining the results in a coherent way? Should this be all up to the model, partly up to the model, or all up to whoever wants the information (e.g. probably the **ResultView** in our case)? Do we need to add some methods to the model to facilitate this? Should we remove or change any methods that are currently in the model? Should we create a new class to deal with the combined results?
5. Are there other possible complications we will run into when we have to deal with multiple versions? Discuss them.

Include in your write-up an expected grade and indicate how many hours your group spent on Part 1 and on Part 2 (give them separately). *This should be handed in on paper at the beginning of class on the due date.* Make sure the names of all group members are on this.