CSC 300 Software Engineering I 2018 Spring

Project 01

Black Box Testing - Test Scenario Design

Project Objective(s): to gain experience in

* designing a set of test scenarios based on a set of requirements
* JUnit test framework generation

You are supplied with Java interface files named StudentSpec and StudentCollectionSpec. StudentCollectionSpec is designed to specify the behavior of an implementing class intended to allow a client to store a number of StudentSpec objects in a StudentCollectionSpec collection object. The goal for this project: to design a set of test scenarios intended to demonstrate (provide documentation) that an implementation of **StudentCollectionSpec** is a "correct" implementation of all specified requirements, and prepare for implementation of (but not actually implement) the test scenarios. Note that:

* There are two specific objectives for this project:
  + - * Design a comprehensive set of test scenarios for an implementation of StudentCollectionSpec based on a supplied set of StudentCollectionSpec requirements.
      * Create a properly configured NetBeans project that contains a *framework* for a JUnit class that will provide for automated testing of an implementation of StudentCollectionSpec. You will *not* be *implementing* any testing *logic* in this project, *just* creating the framework.

Project resources:

* This project requirements document
* TestScenariosTemplate.docx
* TestScenariosTemplate With Examples.docx
* Project01InterfacesOnly.zip

**Project activities and expected outcomes**

Note: it is expected that you will have already read the preceding functional and non-functional requirements **as well as *all* of the following project activities *before* you begin project activities**.

Steps P1-P8 (creating a JUnit test skeleton) need to be executed in the sequence listed; D1 (designing and documenting a set of test scenarios) can be worked on independent of steps P1-P8, except that by the end of the project the results of D1 must be stored inside the project folder created in in P1.

Steps P1-P8 are each designed to achieve a specific objective, and in total designed to work together *in a specified sequence*. Depending on your prior experiences, you may be tempted to rearrange the order in which these activities are performed – please don’t, as the odds are high that changing the order will have a significant effect on what you can and cannot do in subsequent steps. *Pay attention to the details*, do what is specified, do *not* do things that are explicitly stated you *shouldn’t* be doing, and *don’t make assumptions*.

Project activities:

1. You have been provided with a zipped NetBeans project (Project01InterfacesOnly.zip). Extract the contents of the zip archive onto your hard drive (H:…, or C:… if using your own computer, but not in your user profile!).
2. Launch NetBeans and open the project that you just extracted.
   * Change the name of the project: right-click on the project name, choose "Rename", in the resulting wizard replace the existing name with Project01Project\_***LastNameInitialOfFirstName*** (camel case, no spaces, no hyphens – Donald Knuth 🡪KnuthD) and check the "Also Rename Project Folder" option, then click Rename.
   * Open and view the supplied source files (there should be two, StudentSpec and StudentCollectionSpec). For StudentCollectionSpec, expand the Javadoc comments (right-click in the code pane, Code Folds🡪Expand All Javadoc) and read the javadocs.
   * Right-click the project name in the project pane and click Generate Javadoc. The generated HTML-formatted documentation will automatically open in a browser - use the left navigation pane of the documentation to browse through all of the documented classes, and explicitly compare what you see for StudentCollectionSpec *n the browser* to the javadocs embedded in the StudentCollectionSpec *java code*.
     + - Note that once you have generated the javadocs:
     + Javadocs for a specific method or class attribute can be displayed in a popup by hovering over an entry in NetBeans’ Navigator pane;
     + Javadocs for the class can be accessed by placing the cursor on a class element (class header, attribute, method) and typing Alt+F1.
     + Note for future reference – if javadoc comments *in the code* are altered, the HTML version is *not* automatically updated: to update, perform a “clean and build”, then Generate Javadocs as described above.
3. Examine the StudentCollectionSpec interface and associated documentation, *paying close attention to the Javadocs* to determine the functional requirements to be addressed by an implementing class. The functional requirements will be important when you get to step **P8**.
4. Create a Java class named Student that implements StudentSpec, placing Student in a package named *student*.
   * + New🡪Java Class, *fill in the name of the class and the package*, and make sure that the class will be placed in the current project folder (…Project01Project\_*LastNameInitialOfFirstName* - use the browse button to verify location, changing the location if necessary).
     + Delete all of the "boilerplate" Javadoc comments that are automatically supplied in the generated Student file, then add the appropriate "implements" clause. You'll see a syntax error:
       - Right-click in the code pane, use "Fix Imports". Note that an "import" statement appears, and that the syntax error is replaced by a different syntax error – ignore the new error and move on to the next bullet.
     + Use the NetBeans hint feature that will automagically “implement” any contractual obligations.
       - Click on the lightbulb that appears to the left of the class header, choose “Implement all abstract methods”. Note the implementations provided for the methods…
     + The Student class will eventually need two constructor methods, with headers as follows:

*public Student( String theSID, String theFirstName, String theMiddleInitial, String theLastName,*

*String theMajor, double theTotalDegreeCredits, double theTotalQualityPoints )*

and

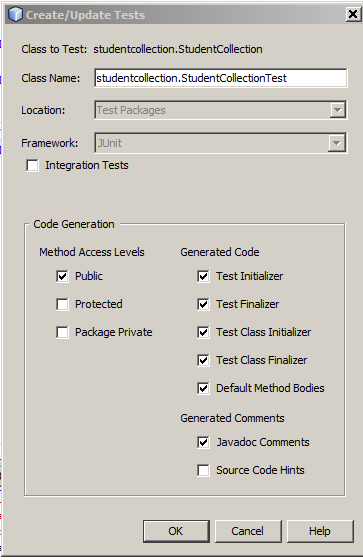
*public Student( String theSID, String theFirstName, String theMiddleInitial,*

*String theLastName, String theMajor )*

Create these two methods (you will need to type them – automagic only goes so far); fill in the bodies of the two constructor methods with a “throws UnsupportedOperationException()” statement.

* + - ***Do not implement any method body statements beyond what has been described in this step (P5)* – insert code to cause any syntax errors to disappear, *but no more than is necessary to make syntax errors disappear*!** Use the implementation code supplied by the "Implement all abstract methods" as an example of what to use for the method bodies of the two constructors.

1. Create a Java class named StudentCollection that implements StudentCollectionSpec, placing StudentCollection in a package named *studentcollection*.
   * + Proceed *using the same basic pattern as in the previous step*, adapting for the new package and class name.
     + The class must have two constructors, one accepting an int that will specify the capacity of a collection instance, the other parameterless.
     + ***Do not fully implement any method bodies –* as before, insert code to cause all syntax errors todisappear, *but no more than is necessary to make syntax errors disappear*!**
2. Clean and Build the project to verify that there are no syntax errors.
3. Give StudentCollection focus in the code pane, then use *Tools 🡪 Create/Update Tests* from either the menu bar or by right-clicking on StudentCollection.java in the project pane) to create a framework for JUnit testing.
   * + If the Create/Update Tests option does not appear:
       - Open the project's Properties wizard, select Sources
       - In the Test Package Folders pane, click Add Folder
       - In the resulting wizard, ***browse to the project folder****,* create a folder named *test* using the provided button, click "Open"
       - Change the label for the *test* folder by clicking the default label (*test*, on the right) and replacing the default with *Test Packages*.
       - Click OK.
       - Try *Tools 🡪 Create/Update Tests* again
     + Note – if the project is configured correctly, an alternative means of creating a testing framework would be to put the cursor on the StudentCollection class header, click on the lightbulb next to the class name and choose "Create Test Class [JUnit in test packages]". If this option is not available, proceed as described above.
     + You will be presented with a wizard that allows you to customize the framework for a JUnit class that is about to be generated:



* + - Do not alter the supplied class name – you can, but you will NOT appreciate the results.
    - Alter the other settings to match what you see above (particularly the “Method Access Levels” setting), then click OK.
      * If the “framework” specified is other than JUnit, use the drop-down to change it to JUnit
    - A new class will appear, named StudentCollectionTest, located in package "test".
      * *Do not alter any of the framework code that is supplied while working on this project* – one of the objectives of this project is to create the framework *and stop there*. To be explicit - do *not* fill in any JUnit implementation details.
      * Inspect the framework, noting in particular the number and name of the test methods (the methods preceded by @Test annotations); compare the number of test methods to the number of methods in StudentCollection.

1. Design and document a set of test scenarios that can be used to verify that StudentCollection implements all of the functional requirements specified in StudentCollectionSpec.
   * + The objective of this step is to determine what *logical* steps would be necessary to demonstrate that an implementation of StudentCollection is correct (addresses all of its functional requirements). *You are not expected to implement the logical steps (i.e., translate the logical steps into formal Java / JUnit statements) as part of this project,* ***and should not attempt to do so.***
     + Document your test scenarios using the supplied template file (TestScenariosTemplate.docx), changing the name to “TestScenarios Project01 *LastNameInitialOfFirstName*.docx”. **Use the supplied template** - test scenarios documented using a different template will not be accepted.
       - Place the resulting file inside the project folder – not in the *src* subfolder, not in the folder that contains the project folder, but *in* the project folder, at the same level as the *src* folder.
     + For each test scenario, be sure to state all setup work that will be necessary to allow the test scenario to be executed and the outcome properly assessed – this will be very important!
     + Note – as stated above, you are *not* to *implement* your test scenarios as JUnit code as part of this project - just *design* the test scenarios and *document* the test scenarios!!!
2. Create a zip archive of your project folder.
   * + Perform a "clean" of your project – this step is critical, otherwise your email submission may silently fail (the SSU email app will appear to send the email, but I will not receive it, and you will not receive a rejection notice).
       - Right-click on the project name in the project pane, choose "Clean".
     + The archive must be named "Project01Project\_*LastNameInitialOfFirstName***.zip**" (only .zip-formatted archive will be accepted).
3. To submit the results of this project, *one* member of the team must create an email with the following particulars:

TO: [jkasprzyk@salemstate.edu](mailto:jkasprzyk@salemstate.edu)

Subject: CSC300 Project 01 Submission *LastNameInitialOfFirstName*

(e.g. CSC300-01 Project 01 Submission KnuthD)

Attachments: (1) a "zipped" version of your NetBeans project

**Assessment criteria:**

DRAFT submission deadline: 23:59:59 2/4. Note: no resubmissions, no extensions. Your submission must include the results of step D1. **Grading of the draft submission is based SOLELY on the test scenarios document. 20% of project grade.**

FINAL submission deadline: 23:59:59 2/11. Note: no resubmissions, no extensions. 80% of project grade. Your final submission must include your NetBeans project and your test scenarios document, that is, the results of steps P1-P8 and D1.

The grade on this project will be determined by:

* + **the *thoroughness* of your test scenarios**
  + **the *quality* of your test scenarios (does each test scenario have a clear, specific purpose?)**
  + **the *lack* of *redundancy* in your test scenarios**
  + whether naming and submission conventions were followed