SYSTEM TEST PLAN PROJECT 2 PT 1: Design Proposal

Document Author(s): Abby Finan

Date: Oct 21, 2021

Design Rationale

The proposed design for the SystemTestPlan application consists of the following classes: TestPlan, TestPlanList, TestCase, TestCaseList, SystemManager, SystemTestReader, SystemTestWriter, SystemTestGUI, PassState, FailState, and the ResultState interface.
The design is meant to be centered around the SystemManager class, where users can load, save, clear, and quit the Test System application - users can also remove, select, add, and edit test plans, as well as list, add, remove, and order a specific test case.

The TestCase class implements the ResultState interface so the finite state machine can update results of test cases - either pass or fail. TestCase implements the ResultState interface as it has two classes for the two states for a result of a TestCase. There are two constructors for TestCase - one being a null constructor and one with all the parameters, testId, description, expectedResults, and actualResults. It also has a testType field defined, which determines the type of each test case. The TestCase class is used when a user wants to add a new test case and it contains getters and setters for each of its fields. The getters create the field and the setters check to make sure the field fits all requirements - if these requirements are met, the test case will be added, if they are not met, an IllegalArgumentException is thrown.

The ResultState interface is the main center point of the finite state machine and is able to update the result changes made to each class. This interface is used with the Pass and Fail classes which each have the same methods in order to update TestCase. The interface has the same methods for each test case and these are called whenever a test case is loaded from a file or is changed. This allows for the system test plan manager to update a test case if a user is editing it by updating the field, which is then called in the method in the ResultState interface and then is passed down to each of the classes to handle the result state change. The Pass and Fail state classes are called in the interface and they show whether the results of the TestCases are valid or not.

The TestPlan class implements and calls on the TestCase class for the given fields and is used for editing, adding, and deleting a test case within a test plan. The TestPlan class has two constructors, one for null, and one with the String parameter for the test plan name. The SystemManager class calls on this TestPlan class for multiple methods referring to TestCase and TestPlan.

The SystemManager class handles all methods related to the user interface and ability to use the SystemTestGUI. The SystemManager constructor creates a TestPlan with TestCases within it. The getDisplay method returns a string array with testid, description, expected, and actual results for each test case in a given file. This class is also meant to handle main menu options for the user to navigate such as load the application, save the application, and clear the application using the methods loadSystemManager, saveSystemManager, and clearSystemManager. The SystemManager class adds, edits, selects and removes TestPlans as the user chooses. On the other hand, the addTestCase method for this class handles assigning the fields to the test case, the listTestCase provides a list of all of the test cases within a test plan, the removeTestCase method removes the test case from the display, and the

orderTestCase calls on the linked list and can order the test cases. The quitSystemManager method exits the application.

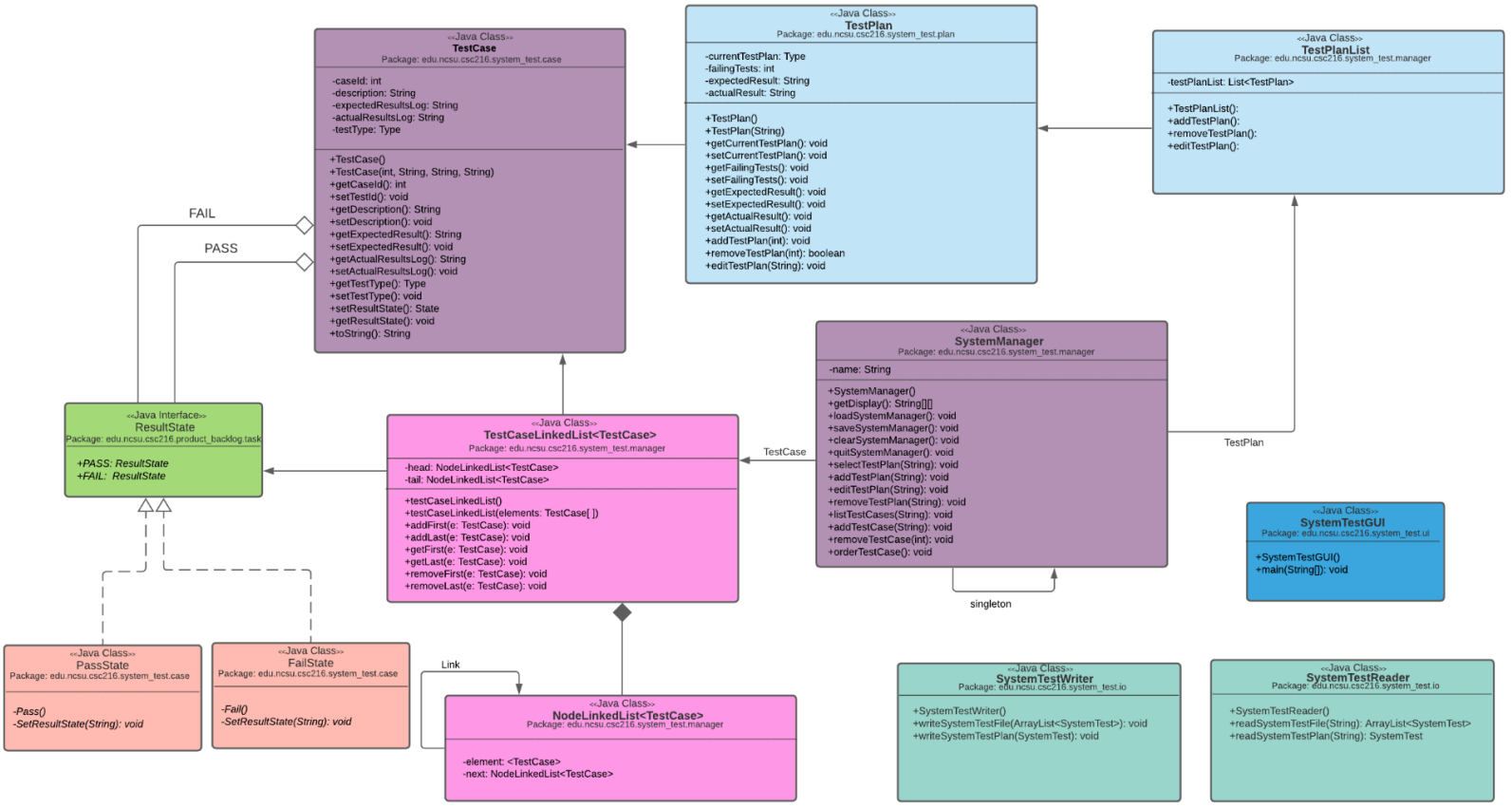
The SystemTestReader and SystemTestWriter classes take care of loading a file containing a product with tasks in it, processing this file using the readSystemTestFile method, and then writing the file to export it from the application using the writeSystemTestFile method. These IO classes are imported into the SystemManager class which manages the processing of the file by using the readSystemTestPlan method. The String being processed is the filename that is entered into the system by the user and then opens the file if there are no exceptions thrown and the file is valid. The readSystemTest method reads in a file with scanner and creates test cases with an arraylist with all the fields until it is able to be read and return a full list of test cases - the writeSystemTestPlan method takes a file and the array list of tasks and converts it to a string to export.

The TestCaseList class is a linked list - I chose to make the linked list pertain to the TestCase class, as it needed to be able to order different test cases and linked lists work efficiently for this. This list creates a default linked list, creates a linked list from the array of elements, and can add an element to the head of the list (beginning) or the tail of the list (end). This can also return the first or last element from the head or tail, or remove the first or last element from the head or tail of the list. This linked list will be used when the user wants to move the tests up and down in the list within the GUI so that they can prioritize them and order the test cases how they want. On the other hand, as well as the linked list, there is also an arraylist class - the TestPlanClass. I decided to make this an arraylist instead of a linked list because this list can be ordered alphabetically, not changing the order, so it can be used as a sorted list.

The Singleton pattern was implemented for the SystemManager because there can only be one instance of it and the way the manager is set up is that all of it can only be occurring at one time, meaning that instantiating it would not seem feasible. The TestPlan, TestCase, and SystemManager objects are the most important objects in this design as they are the building blocks for the rest of the design and are the basis of the project. Each of these objects contain and construct crucial information to manage the different information in a file. Without these objects, the rest of the design would not be able to run as efficiently and successfully. In order for this application to run, files and user input are required to implement the system. These data fields are the most important because they build the objects being used to run the program and give information that is needed to run each class. The file being read into this system is very important because it allows for the reader class to parse through it as well as create lists of to write so that users can add, remove, select these objects. The SystemManager is extremely important as it handles and calls on both the TestCase and TestPlan classes and the GUI will interact with it directly. The interface is an example of abstraction and the two list classes have a composition relationship with their objects, which allows for the lists to be created and for functions like adding, removing, etc.

A limitation of this design is that there is no way to see why a test case fails and no way to see the code that is failing it - this design only takes care of System Test Plans and does not organize source code or have any place to store a file of source code. I think a great addition to this application would be a place to store all classes of source code to see what the test case and test plan is referring to.

Figure 1: Class Diagram for <System Test Plan>



Document Revision History

Date	Author	Change Description
10/19/21	Abby Finan	Started document Drafted class diagram
10/21/21	Abby Finan	Finished class diagram Finished design rationale