

TBB Authoring App Testing Manifest

Daniel Kozlovsky

Arshdeep Saini

Kirollos Kamel

Latest Revision: August. 21st 2018

Table of Contents

1.0 Tests	1
1.1– scenario.Scenario.java	1
1.1– scenario.ScenarioFormatter.java	2
1.1– scenario.ScenarioCommand.java	3
2.0 – Coverage	8

1.0 – Tests

This section outlines all tests undergone thus far. All test outlined in this section are facilitated by JUnit 5 in the Eclipse Oxygen IDE.

The test cases are named after the JUnit test name and are organized per class tested. The summary of the test case, its derivation and sufficiency is discussed.

1.1– scenario.Scenario.java

1.1.1 – testMoveCommand

Description

Two distinct *ScenarioCommand* objects are created and added into a *Scenario*. The *Scenario.moveCommand(ScenarioCommand, int)* is invoked to switch the two objects' places. Validates Success by testing position of second Object against created Object.

Derivation

Test confirms the basic function of *Scenario.moveCommand(ScenarioCommand, int)*.

Sufficiency

The tested method has basic functionality and implements the already widely tested *ArrayList.remove()* and *ArrayList.add()* methods.

1.1.2 – testDeepEquals

Description

Creates two separate *Scenario* objects with identical fields and data. Then invokes *Scenario.deepEquals(Scenario)* upon them.

Derivation

Identical fields were used to test basic functionality.

Sufficiency

Tests basic function and flow of only a conditional statement and a loop. Most of the method relies on *ScenarioCommand.deepEquals()*, which is extensively tested.

1.2 – scenario.ScenarioFormatter.java

1.2.1 – testExport_createsFile

Description

Invokes *ScenarioFormatter.export()* using a basic *Scenario* object to a specified path and test the existence of the created file.

Derivation

Needed to make sure a file is indeed created and in the proper directory.

Sufficiency

Tests the first step of *export()*, which is file creation and is the foundation for the rest of the method.

1.2.2 – testExportImport_allCommands

Description

Creates a basic *Scenario* object with a every possible *EnumPossibleCommands* value as population and is then used in invocation of *export()*. The resulting file is then read through *importParse()*. The resulting *Scenario* is compared against the original through *Scenario.deepEquals()*.

Derivation

It is needed to make sure *export()* and *importParse()* at least follow the same format rules. It is complicated to test them by themselves. Thus they are used in conjunction to test consistency of the client defined format rules. By using every possible *EnumPossibleCommands* value, it tests consistency of 100% of possibilities.

Sufficiency

Tests every possible *EnumPossibleCommands* value with both *export()* and *importParse()*. *export()* also invokes *format()*, so this code is reached.

1.2.3 – testImportParse_simpleCommands

Description

A *Scenario* is created to be identical to a pre-created, manually written scenario file. This file is imported using *importParse()* and the resulting *Scenario* is tested against the expected *Scenario* which mimics the manually written text file.

Derivation

This test was created as a way to test *importParse()* by itself and the basic functionality and flow of the method, without getting into specific boundary cases per *EnumPossibleCommands* value.

Sufficiency

Tests file reading and parsing for some arbitrary commands.

1.2.4 – testImportParse_emptyFile

Description

Reads an empty text file and tests the result of *importParse()* against null.

Derivation

Designed to make sure any improper formatting, such as an empty file, returns null for handling and not a false *Scenario*.

Sufficiency

Test one possible case of a mis-formatted file.

1.3 – scenario.ScenarioCommand.java

1.3.1 – testDeepEquals_sameObjSingle

Description

Creates one simple *ScenarioCommand* objects and assigns it to two *ScenarioCommand* references. Tests each reference against each other.

Derivation

Needed to make sure different references to the same object returned true, to save time checking all the fields.

Sufficiency

Test all possible cases of references pointing to the same object.

1.3.2 – testDeepEquals_sameObjAllCommands

Description

Creates two different *ScenarioCommand* objects with identical fields. Every possible *EnumPossibleCommands* value as population.

Derivation

Needed to make sure fields checks are consistent for each *EnumPossibleCommands* value. Mainly used to check basic functionality and flow of each branch in *deepEquals()*.

Sufficiency

Tests identical fields for every possible *EnumPossibleCommands* value.

1.3.3 – testDeepEquals_sameArgs

Description

Two basic *ScenarioCommand* objects are created with the same arguments field. They are tested against each other.

Derivation

Basic test for branch flow.

Sufficiency

Tests one possible type of similar fields.

1.3.4 – testDeepEquals_differentArgs

Two basic *ScenarioCommand* objects are created with different arguments field. They are tested against each other.

Derivation

Basic test for branch flow. Also makes sure the return false condition is reached.

Sufficiency

Tests one possible type of different fields.

1.3.5 – testDeepEqual_difObjAllCommands

Description

Creates two different *ScenarioCommand* objects with different fields. Every possible *EnumPossibleCommands* value as population.

Derivation

Needed to make sure fields checks are consistent for each *EnumPossibleCommands* value. Mainly used to check basic functionality and flow of each branch in *deepEquals()* this time with the opposite case – different arguments.

Sufficiency

Tests different fields for every possible *EnumPossibleCommands* value.

1.3.6 – testSetArguments_nullArgs

Description

Two basic *ScenarioCommand* objects are tested, each with null arguments.

Derivation

Needed to make sure the null object relationship is retained.

Sufficiency

Tests the desired null argument case.

1.3.7 – testSetArguments_throwsIllegalArgumentExceptionWrongAmount

Description

Tries to create a *ScenarioCommand* object with the wrong argument amount. Tests for thrown exceptions

Derivation

Needed to make sure the proper Exceptions with proper messages are being thrown for stack tracing and debugging.

Sufficiency

Test basic functionality with one possible *EnumPossibleCommands* value.

1.3.8 – testSetArguments_throwsIllegalArgumentExceptionWrongType

Description

Tries to create a *ScenarioCommand* object with the wrong argument types. Tests for thrown exceptions

Derivation

Needed to make sure the proper Exceptions with proper messages are being thrown for stack tracing and debugging.

Sufficiency

Test basic functionality of second branch with one possible *EnumPossibleCommands* value.

1.3.9 – testSetArguments_allCommandsWrongArgumentAmount

Description

Tries to create a *ScenarioCommand* object with the wrong argument amount for each *EnumPossibleCommands* value. Tests for thrown exceptions

Derivation

Needed to make sure the proper Exceptions with proper messages are being thrown for stack tracing and debugging.

Sufficiency

Tests every possible *EnumPossibleCommands* value.

1.3.10 – testSetArguments_allCommandsWrongArgumentType

Description

Tries to create a *ScenarioCommand* object with the wrong argument type for each *EnumPossibleCommands* value. Tests for thrown exceptions

Derivation

Needed to make sure the proper Exceptions with proper messages are being thrown for stack tracing and debugging.

Sufficiency

Tests every possible *EnumPossibleCommands* value.

1.3.11 – testSetArguments_correctArgsReadText

Description

Tries to create a *ScenarioCommand* object with the correct arguments for *EnumPossibleCommands.READ_TEXT* value. Tests for thrown exceptions.

Derivation

Needed to make sure the basic structure and functionality works, so one command is used.

Sufficiency

Tests one possible *EnumPossibleCommands* value.

1.3.12 – testValidateArguments_correctArgsAllCommands

Description

Tries to create a *ScenarioCommand* object with the correct arguments for all *EnumPossibleCommands* value. Tests for thrown exceptions.

Derivation

Needed to make sure all possible *EnumPossibleCommands* values work and each argument is properly set.

Sufficiency

Tests every possible *EnumPossibleCommands* value.

1.3.13 – testSetArguments_wrongArgsAllCommands

Description

Tries to create a *ScenarioCommand* object with incorrect arguments for all *EnumPossibleCommands* value. Tests for thrown exceptions.

Derivation


















Needed to make sure all possible *EnumPossibleCommands* values are checked and the proper exception is thrown.

Sufficiency

Tests every possible *EnumPossibleCommands* value from a finite set of incorrect arguments. This test is repeated 10 times to get various combinations of arguments.

2.0 – Coverage

Coverage Results:

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
▲ TBB Authoring App	 45.3 %	2,319	2,795	5,114
▲ src	 45.3 %	2,319	2,795	5,114
▲ app	 0.0 %	0	2,335	2,335
▶ ScenarioCreator.java	 0.0 %	0	1,285	1,285
▶ addAudio.java	 0.0 %	0	423	423
▶ AudioRecorder.java	 0.0 %	0	391	391
▶ MainPanel.java	 0.0 %	0	147	147
▶ ApplicationWindow.java	 0.0 %	0	70	70
▶ Entry.java	 0.0 %	0	19	19
▲ scenario	 76.8 %	1,258	379	1,637
▶ ScenarioCommand.java	 55.5 %	399	320	719
▶ Scenario.java	 84.1 %	116	22	138
▶ ScenarioFormatter.java	 95.9 %	351	15	366
▶ EnumPossibleCommands.java	 98.0 %	392	8	400
▶ ImproperFormatException.java	 0.0 %	0	7	7
▶ MissingArgumentException.java	 0.0 %	0	7	7
▶ scenario.test	 92.9 %	1,061	81	1,142

(Eclipse representation of coverage for application tests)

Overall the coverage is fairly extensive. 45.3% of the whole project is reached, however 76.8% of the back-end code is covered with 0% of the UI tested.

The biggest loss of coverage is in *ScenarioCommand*. This is due to many branches in various switch cases being missed. The recommended course of action is to widen the incorrect argument set to try to access those missed branches.

Otherwise most functionality is tested as the UI is just an implementation of the *scenario* package.