CoffeeMaker.java

Evan Alexander & George Totolos

CS 1632 - FINAL DELIVERABLE

Further Exhaustive Testing of CoffeeMaker

https://github.com/get13/CS1632/tree/master/Deliverable%206/src

For the final deliverable, we decided to test the original CoffeeMaker file by using a decompilier with automatic JUnit tests as opposed to manual unit testing like we did in the first deliverable. We felt this was appropriate because although manual unit tests are an efficient way to test software, it does not require any additional coding and therefore falls short of being exhaustive. Furthermore, we wanted to avoid altering the original code at all costs, because we wanted to simulate testing a program that was not written with development in mind (TDD or BDD). By revisiting an old program that we previously tested in a less technical manner, we are learning to improve the quality and technique of our tests. This presented a challenge to us because of the obscurity of automatically testing void methods. Therefore, we had to be creative in our techniques to adequately test each requirement.

There were many difficulties we faced when developing the testing plan for this file. Firstly, not every method is set up to be easily automatically tested. For instance, testing that a character will produce an output is simple because it only requires an assert call in JUnit, but when the complexity of the method increases, the difficulty of developing an exhaustive test increases as well. In order to test these methods without writing automatic unit tests, we revised some of our manual unit tests.

Although this software's core functionality exists, there are still minor insignificant defects. I would still recommend this software to be deployable despite these defects due to their low severity and the fact that they do not drastically alter the program. In general, you want your software to have no defects, but under certain deadlines it is important to be able to make sacrifices and changes and decide which defects are insignificant enough to deploy anyway.

One particular requirement that was difficult to test was to ensure that the input of the “H” key produces a help output. If this method existed, the JUnit test to verify it would be easy to write. However, once again, we did not want to alter any of the original code to simulate a real software testing scenario. Therefore, it was difficult to test a non-existent method, but we did write a manual test just to be complete instead, ensuring that H does not in fact produce a help message and is instead interpreted as an invalid entry. In addition to the non-existing function that was difficult to test, functions that required I/O were actually extremely difficult to automatically test. Upon Googling around about how to adequately automatically test mock console input output, it became apparent to me that this is commonly regarded as a very obscure, complex and usually difficult thing to do. Therefore, we used manual unit tests for various methods.

IDENTIFIER: FUN-HELP
TEST CASE: Ensure that entering ‘H’ will display a list of possible commands and what their effects are
PRECONDITIONS: Program has just been started executing and is at the very first iteration (Small room)
INPUT VALUES: ‘H’
EXECUTION STEPS: User inputs the letter ‘H’ and presses enter
OUTPUT VALUES: ‘What?’
POSTCONDITIONS: The list of possible commands and their effects was not displayed

IDENTIFIER: FUN-ITERATION-1
TEST CASE: Ensure that the ‘N’ command at the Small room brings us to a different room
PRECONDITIONS: Program has just been started executing and is at the very first iteration (Small room)
INPUT VALUES: ‘N’
EXECUTION STEPS: User typed in the letter ‘N’ and pressed Enter on the keyboard
OUTPUT VALUES:
‘You see a Funny room.
It has a Sad record player.
A Beige door leads North.
A Massive door leads South.’
POSTCONDITIONS: Output value was displayed, user is now prompted to enter the same allowed values (N,S,L,I, or D) once again.

IDENTIFIER: FUN-ITERATION-2
TEST CASE: Ensure that the ‘S’ command at the Small room brings us to a different room
PRECONDITIONS: Program has just been started executing and is at the very first iteration (Small room)
INPUT VALUES: ‘S’
EXECUTION STEPS: User typed in the letter ‘S’ and pressed Enter on the keyboard
OUTPUT VALUES:
‘You are in a magical land! But you are returned to the beginning!
POSTCONDITIONS: Output value was displayed, user is now prompted to enter the same allowed values (N,S,L,I, or D) once again.

IDENTIFIER: FUN-ITERATION-3
TEST CASE: Ensure that the ‘L’ command at the Small room finds cream
PRECONDITIONS: Program has just been started executing and is at the very first iteration (Small room)
INPUT VALUES: ‘L’
EXECUTION STEPS: User typed in the letter ‘L’ and pressed Enter on the keyboard
OUTPUT VALUES:
There might be something here...
You found some creamy cream!
POSTCONDITIONS: Output value was displayed, user is now prompted to enter the same

allowed values (N,S,L,I, or D) once again.
IDENTIFIER: FUN-INPUT-CAPS
TEST CASE: Ensure inputting ‘l’ at the small room will have the same result as inputting ‘L’ at the small room
PRECONDITIONS: Program has just been started executing and is at the very first iteration (Small room)
INPUT VALUES: ‘l’
EXECUTION STEPS: User typed in the letter ‘l’ and pressed Enter on the keyboard
OUTPUT VALUES: ‘What?’
POSTCONDITIONS: Output value is displayed along with repeated prompt for the Small Room

IDENTIFIER: FUN-UNIQ-ROOM-FURNISHING
TEST CASE: Ensure that the initial room has exactly one unique furnishing visible
PRECONDITIONS: Program has just been started executing and is at the very first iteration (Small room)
INPUT VALUES: *None*
EXECUTION STEPS: Run game and view initial message
OUTPUT VALUES:
‘You see a Small room.
It has a Quaint sofa.
A Magenta door leads North.’
POSTCONDITIONS: Description of Small room is displayed, containing only one unique furnishing

IDENTIFIER: FUN-UNKNOWN-COMMAND
TEST CASE: Ensure that inputting 5 into the prompt returns “What?”
PRECONDITIONS: Program has just been started executing and is at the very first iteration (Small room)
INPUT VALUES: ‘5’
EXECUTION STEPS: User typed in the letter ‘5’ and pressed Enter on the keyboard
OUTPUT VALUES: ‘What?’
POSTCONDITIONS: Correct output value is displayed along with the initial Small room description

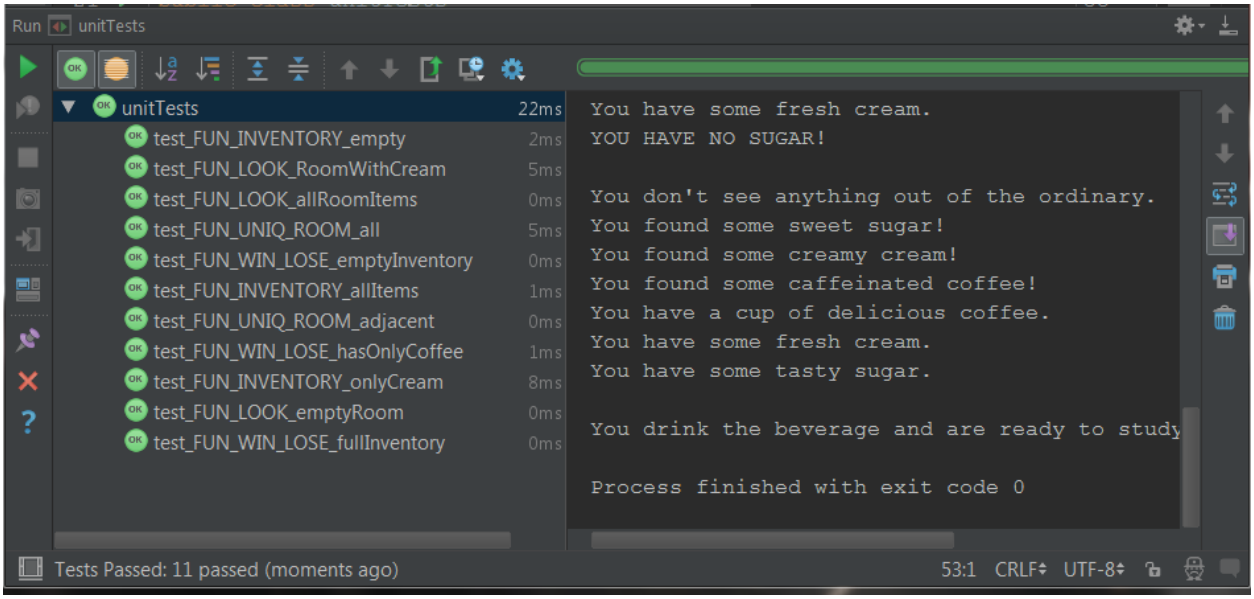
IDENTIFIER: FUN-MOVE
TEST CASE: Ensure that moving south in the Small room (that has no existing south door) will not cause the player to move
PRECONDITIONS: Program has just been started executing and is at the very first iteration (Small room)
INPUT VALUES: ‘S’
EXECUTION STEPS: User typed in the letter ‘S’ and pressed Enter on the keyboard
OUTPUT VALUES: ‘You are in a magical land! But you are returned to the beginning!’
POSTCONDITIONS: Output prompt is displayed along with the Instructions line

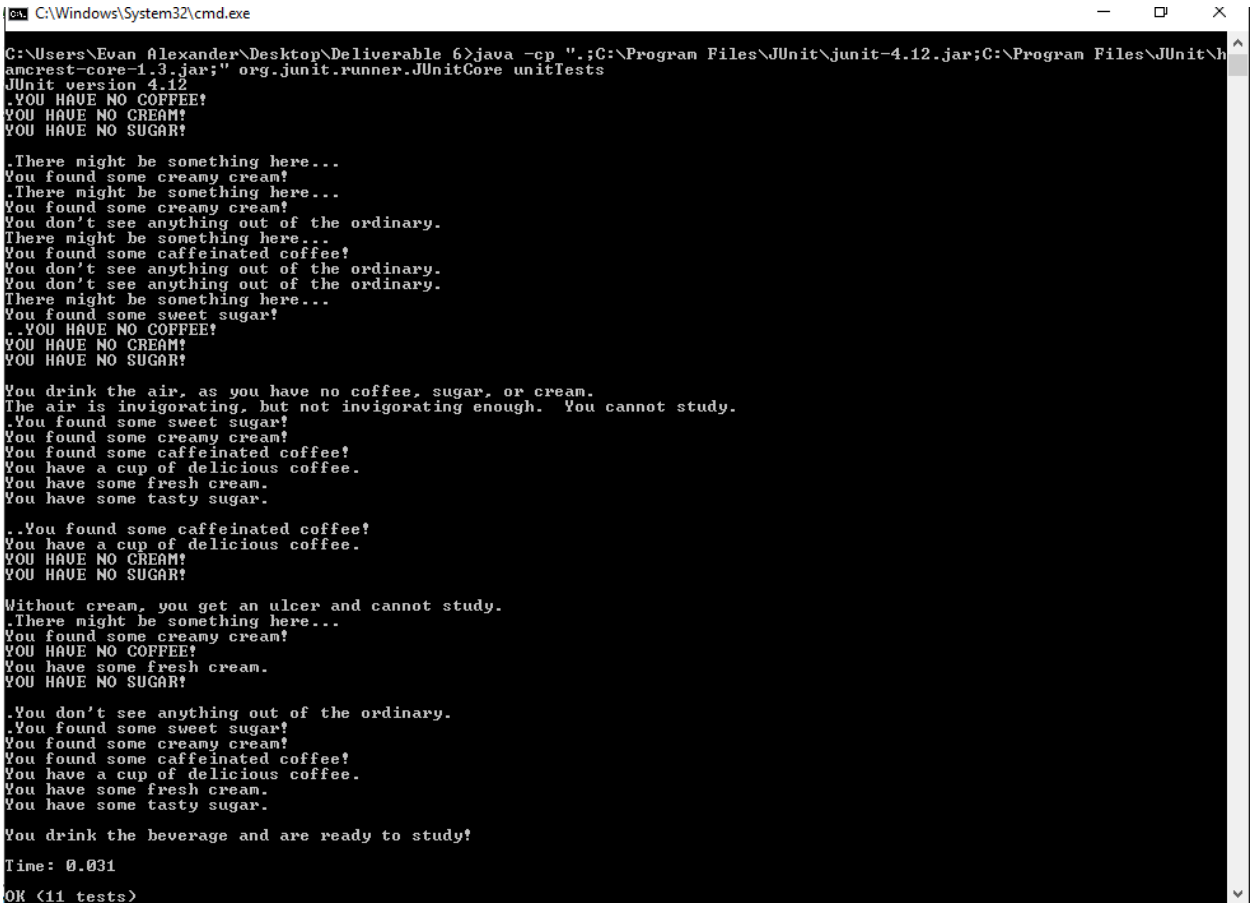
FUN-HELP	**FAILED**
FUN-ITERATION (3 tests)	**PASSED**
FUN-INPUT-CAPS	**FAILED**
FUN-UNIQUE-ROOM-FURNISHING	**PASSED**
FUN-UNKNOWN-COMMAND	**PASSED**
FUN-MOVE	**FAILED**

DESCRIPTION: User entered 'H' and the program did not display a list of possible commands and their effects.
SUMMARY: Program interprets the letter 'H' as an invalid character.
REPRODUCTION STEPS: Run program and press 'H' and press enter.
EXPECTED BEHAVIOR: A help message should display with a list of possible commands and their effects.
OBSERVED BEHAVIOR: The message "What?" was displayed.

DESCRIPTION: User entered 'n' and the program did not behave the same as when user entered 'N'
SUMMARY: Program is casesensitive;
it differentiates between lowercase and uppercase
characters and behaves differently depending on the input.
REPRODUCTION STEPS: Run program, press 'n', then press enter.
EXPECTED BEHAVIOR: Program will advance into the North door and behave just as it would if the capital letter 'N' was entered
OBSERVED BEHAVIOR: The message "What?" was displayed.

DESCRIPTION: Pressing 'S' when there is no south door present will still move the player to a different room, having them start in a magical land (at the beginning)
SUMMARY: Program allows user to move either South or North no matter where player is in the game.
REPRODUCTION STEPS: Run program, press 'S', then press enter
EXPECTED BEHAVIOR: Player will not move, and will be in the same room as before.
OBSERVED BEHAVIOR: Program displays "You are in a magical land! But you are returned to the beginning!"


Run unitTests
unitTests 22ms
test_FUN_INVENTORY_empty 2ms
test_FUN_LOOK_RoomWithCream 5ms
test_FUN_LOOK_allRoomItems 0ms
test_FUN_UNIQ_ROOM_all 5ms
test_FUN_WIN_LOSE_emptyInventory 0ms
test_FUN_INVENTORY_allItems 1ms
test_FUN_UNIQ_ROOM_adjacent 0ms
test_FUN_WIN_LOSE_hasOnlyCoffee 1ms
test_FUN_INVENTORY_onlyCream 8ms
test_FUN_LOOK_emptyRoom 0ms
test_FUN_WIN_LOSE_fullInventory 0ms
You have some fresh cream.
YOU HAVE NO SUGAR!
You don't see anything out of the ordinary.
You found some sweet sugar!
You found some creamy cream!
You found some caffeinated coffee!
You have a cup of delicious coffee.
You have some fresh cream.
You have some tasty sugar.
You drink the beverage and are ready to study
Process finished with exit code 0
Tests Passed: 11 passed (moments ago)
53:1 CRLF UTF-8


C:\Windows\System32\cmd.exe
C:\Users\Evan Alexander\Desktop\Deliverable 6>java -cp ".;C:\Program Files\JUnit\junit-4.12.jar;C:\Program Files\JUnit\hamcrest-core-1.3.jar;" org.junit.runner.JUnitCore unitTests
JUnit version 4.12
.YOU HAVE NO COFFEE!
YOU HAVE NO CREAM!
YOU HAVE NO SUGAR!
.There might be something here...
You found some creamy cream!
.There might be something here...
You found some creamy cream!
You don't see anything out of the ordinary.
There might be something here...
You found some caffeinated coffee!
You don't see anything out of the ordinary.
You don't see anything out of the ordinary.
There might be something here...
You found some sweet sugar!
..YOU HAVE NO COFFEE!
YOU HAVE NO CREAM!
YOU HAVE NO SUGAR!
You drink the air, as you have no coffee, sugar, or cream.
The air is invigorating, but not invigorating enough. You cannot study.
.You found some sweet sugar!
You found some creamy cream!
You found some caffeinated coffee!
You have a cup of delicious coffee.
You have some fresh cream.
You have some tasty sugar.
..You found some caffeinated coffee!
You have a cup of delicious coffee.
YOU HAVE NO CREAM!
YOU HAVE NO SUGAR!
Without cream, you get an ulcer and cannot study.
.There might be something here...
You found some creamy cream!
YOU HAVE NO COFFEE!
You have some fresh cream.
YOU HAVE NO SUGAR!
.You don't see anything out of the ordinary.
.You found some sweet sugar!
You found some creamy cream!
You found some caffeinated coffee!
You have a cup of delicious coffee.
You have some fresh cream.
You have some tasty sugar.
You drink the beverage and are ready to study!
Time: 0.031
OK (11 tests)