Nathan Sellers

Test Document

* StockItemUnitTests
  + UnitTest\_StockItem()
    - Tests that a stock item is instantiated correctly, and that the attributes are populated. This test passes.
* VendingMachineUnitTests
  + UnitTest\_Start\_Restock()
    - Tests that when a user selects 'r' when a vending machine is started, the state switches to STOCK. There is also a timeout of 1000ms. This test passes.
  + UnitTest\_Start\_Exit()
    - Tests that when a user selects 'e' when a vending machine is started, the state switches to OFF. There is also a timeout of 1000ms. This test passes.
  + UnitTest\_Start\_AnythingElse()
    - Tests that when a user does any input besides ‘e’ or ‘r’ when a vending machine is started, the state switches to INSERT. There is also a timeout of 1000ms. This test passes.
  + UnitTest\_LoadStock()
    - Tests that when a vending machine is instantiated, LoadStock() is called by checking if the MaxValue attribute was updated. This test passes.
  + UnitTest\_GetFileName()
    - Tests that when a path is passed into GetFileName(), the result is equal to the initial path that was passed in. This test passes.
  + UnitTest\_ReadFile()
    - Tests that when a file is read, the list of strings that are returned are equal to the values that we would expect it to be. This test passes.
  + UnitTest\_InsertMoney\_MultipleInserts()
    - Tests what happens when InsertMoney() is called multiple times, and checks to see if the vending machine’s Balance attribute is being updated correctly. This test passes.
  + UnitTest\_InsertMoney\_Parameterized()
    - Tests the InsertMoney() method in the VendingMachine class, and that Balance is updated correctly. This test was parameterized and all the test cases pass.
  + UnitTest\_InsertMoney\_MaxCostExceeded()
    - Tests the InsertMoney method when the amount of money inserted exceeds MaxValue. It tests that when it is exceeded, the state changes to SELECT. This test is parameterized and all the test cases pass.
  + UnitTest\_DispenseChange()
    - Tests the DispenseChange method, and makes sure that the Balance is set back down to zero and that the state is switched to START. This test passes.
  + UnitTest\_DispenseSelection\_NoChange()
    - Tests the DispenseSelection method, and checks to see if the state is switched to START and there is no Balance left in the vending machine. This test fails because the state switches to CHANGE instead of START.
  + UnitTest\_DispenseSelection\_NoChange()
    - Tests the DispenseSelection method, and checks to see if the state is switched to CHANGE and there is a Balance left in the vending machine. This test fails because the state switches to START instead of CHANGE.
  + UnitTest\_Select\_Refund()
    - Tests the Select method, and makes sure that the state is switched to CHANGE when the input received is “r”. This test passes.
  + UnitTest\_Select\_Valid\_Number()
    - Tests the Select method when a valid number input is received. It makes sure that the state is switched to DISPENSE, and that the selection is equal to the input passed in. This test passes.
  + UnitTest\_Select\_Invalid\_Number()
    - Tests the Select method when an invalid number input is received. It makes sure that the state is not switched to DISPENSE, and that the selection is equal to -1. This test passes.
* IntegrationTests
  + IntegTest\_CreateVendingMachine\_StockItemCreatedCorrectly\_OneItem()
    - Tests that when a vending machine object is created with only one stock item, the stock item is created correctly within the VendingMachine class. This test passes.
  + IntegTest\_CreateVendingMachine\_NoStockItems\_EmptyStock()
    - Tests that when a vending machine object is created with no stock items, the stock items dictionary in the VendingMachine class remains empty. This test passes.
  + IntegTest\_CreateVendingMachine\_MultipleItems()
    - Tests that when a vending machine object is created with multiple stocks, the stock item dictionary is updated with the correct number of stocks. This test passes.
  + IntegTest\_VendingMachine\_Relationship\_States\_Start\_Off()
    - Tests that when a vending machine object is created, and then started with an input of ‘e’, the state switched from START to OFF. This test passes.
  + IntegTest\_VendingMachine\_Relationship\_States\_Start\_Insert()
    - Tests that when a vending machine object is created, and then started with an input of anything besides ‘e’ or ‘r’, the state switched from START to INSERT. This test passes.
  + IntegTest\_VendingMachine\_Relationship\_States\_Start\_Stock()
    - Tests that when a vending machine object is created, and then started with an input of ‘r’, the state switched from START to STOCK. This test passes.
  + IntegTest\_VendingMachine\_Relationship\_States\_Stock\_Start()
    - Tests that when a vending machine object is started, then restocked, the state is switched from STOCK back to START. This test passes.
  + IntegTest\_VendingMachine\_Relationship\_FullSim1()
    - Tests the whole lifecycle of a vending machine interaction and makes sure that the states are switched correctly in between each transition. The user purchases an item and does not use exact change. This test fails because the vending machine does not properly switch states from DISPENSE to CHANGE in order to dispense the user their change.
  + IntegTest\_VendingMachine\_Relationship\_FullSim2()
    - Tests the whole lifecycle of a vending machine interaction and makes sure that the states are switched correctly in between each transition. The user purchases an item and uses exact change. This test fails because the vending machine does not properly switch states from DISPENSE to START in order end the lifecycle properly.

\* I was not able to download the enterprise version of Visual Studio in time, so I was not able to record my code coverage. With that being said, I know that it is not quite 100%, and there were other tests I could have potentially made.