CSC 260 final project test report:

- Testing Saving and Loading in the test class SaveLoaderTester.java:
    - These tests included both saving and loading formats the first being a text format and the second using the Serializable interface which saves classes in binary
    - We started by adding empty Blocks to appModels. When they saved and loaded properly we added different types of Lines and then DisplayText objects
    - When all these were able to be saved and loaded correctly in both formats we altered the DisplayObjects by changing locations, adding names, variables, and methods to the Blocks, changing the location of Lines, and adding text and changing the location of DisplayText objects. We made these changes step by step and refactored the code so that all the information would be saved and loaded properly.
    - We did the final testing by using AppModels toString to compare the Appmodel saved to the AppModel loaded.
- Testing Saving and Loading in the GUI:
    - We also had to test that the save and load aspects of the MenuDisplay which had save and load buttons and had methods for when they were pressed
    - We did this manually by running our program and manually adding the objects, moving them and adding text elements to them, and then saving and loading them.
    - For the Serializable save/load format the DisplayObjects had to be copied into a new appmodel because saving the instance variable saved the rest of the object as well
- Testing the AppModel in the AppModelTest.java file

- The test addBox() adds an empty Block to the model and checks to see if the Block was successfully added by using the containsObject() method this successfully tests both adding a block and the containsObject() method
- The moveblock method tests moving a Block object in the model by adding a Block to the model and then using the moveSelected() method to move the Blocks location. We then use assert methods to make sure that the Block had its X and Y location successfully changed. This successfully tests moving a selected object in the model.
- The connectLine() method tests connecting a line to two Blocks using the setLine() method which sets the instance variables for head and tail in the block and calls the updatePosition() function which moves each endpoint of the line so that it is at the middle of the closest block side that its head or tail is set to. It successfully tested this method's ability to do so and placed the lines in their proper position in the model.
- The moveLine() method tests that lines that are previously connected to blocks using connectLine() connectHead() and setTail() and updatePosition() will move around with the Blocks they are connected to. In this test one of the Blocks is moved and we test to see that the line moves with it. The line in this case did successfully move with the Block and continued to bisect its side
- The moveBlock() method tests that Blocks and Lines can be disconnected from each other. In the method we add a Block and Line to the model and connect one end of the line to the block using connectHead() which attaches it to the block. We then move this end point of the line using the setHead() method which moves the location of the first endpoint of the Line to new X and Y coordinates so that it

    is no longer connected to the block. We test that this happens successfully using the pointOneIsConnected() method that returns true when the first end point of the Line is connected to a Block and false when it does not

    - The testEquals method tests the equals method of the AppModel which returns true if the Appmodel has equal DisplayObjects regardless of the order. We add different Lines and Blocks that are equal to an AppModel and test to make sure that the two AppModels are equal
    - The testToString tested toString added Blocks, Lines, and DisplayText Objects to two AppModels and edited the contents of these objects before. The set of objects added to each AppModel were equal and added in the same order so the toStrings were equal and we used an assertEquals method to check this. As We added each object and made changes to it we would run the tests so we would easily be able to find out which part of the equals method had problems. This was effective
- Testing Things Manually
    - We had to do most of our testing for MenuDisplay, AppDisplay, and AppControl manually and we had to test a lot of different things.
        - In testing these things we started by just testing AppDisplay and doing all modification to the Display in main to make sure that our Render visitors that draw the objects correctly displayed them.
        - After we saw that things were displayed correctly we wanted to make sure that the user could properly interact with the interface and test AppControl which deals with actions by the user including clicking the mouse and typing things. We tested these by adding DisplayObjects to the model

running the program and we were able to move around Blocks Lines and DisplayText Objects all over the window that we created.

        - We also had to make sure that Lines could be permanently connected to Blocks by the user and that they would properly bisect the sides of the Blocks. We were able to connect Lines to Blocks and drag the Blocks around while the Lines stayed connected
        - We also had to test the menu and make sure that we could add the text features to the selected Objects and add them to the Model and Display by pressing buttons and after lots of manual testing on all the types of Objects we concluded that they worked successfully

- Testing Code Generation:
    - We wrote tests for each edge case that could break Code Gen - classes connected by an implementing line which would cause the class statement to change, same for an inheritance line connecting two classes.
    - We also wrote tests that checked when methods had multiple or single parameters to make sure the formatting was coming out correctly
    - We tested the interface case - when two classes are connected by an implementing line which makes the (head) block the interface.
    - We assumed that a user would not add an inheritance line or an implementation line without being connected to a head block and a tail block, so we did not test the case where an inheritance or implementation line are connected to only one block.