Due Saturday by 11:59pm **Points** 2 **Available** after Sep 11 at 10am

Lab 1— Writing New Classes & Methods

Home

Modules

Lab 1: Writing New Classes & Methods

1 Stocks

1.1 Introduction

We'll practice designing and implementing simple classes and unit tests this week by using the concept of an equity known a stock. Stocks are securities that represent fractional ownership of a corporation and can be purchased on exchanges like the NYSE, Nasdaq, FTSE and others.

1.2 Do this first

1. Set up your project for the lab. Be sure to have an src and test directory for your java source and test assets.

2. Create a package called stock for your source and test assets. For example, your src directory should resemble the following:

3. Write a class `Stock` that represents a stock to be traded. For this lab, we'll focus on selling stocks but our Stock class should be general enough to participate in any type of trade (buy/sell/etc.).

Our Stock class should contain:

- An Constructor that takes **THREE** (3) parameters: the symbol, name, and cost basis (in that order) as parameters and sets an instance values appropriately.
- "Getter methods" that return the appropriate instance data:
  - String getSymbol()
  - String getName()
  - double getCostBasis()

- double getCurrentPrice()

- TWO (2) "Setter methods" that set the value for the current price and cost basis.
  - setCostBasis(double)
  - setCurrentPrice(double)
- A calculation method that determines and returns the fractional change between the original cost basis and the current price. (NB: This method returns the fractional change. To get the actual percentage, a client would need to multiply the return value by 100).
  - double getChangePercent()
- A toString() method that returns a String representation of the Stock instance that shows the name of the company, current price and the gain/loss percentage. If the string were displayed on the console it would be rendered on 2 lines, as this example shows (note the 1-space indentation on the 2nd line):
  Apple Computer Current Price: $ 202.12
  Gain/Loss: 5.16%

For each method:

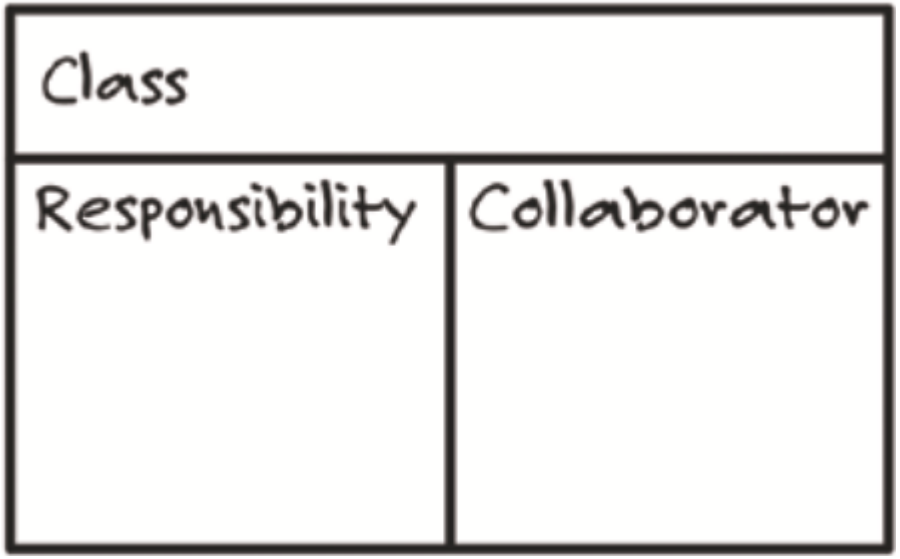
- Follow the signature of the method as specified above.
- Write Javadoc-style comments for that method.
- Write the body for the method.
- Write one or more tests that check that the method works as specified in all cases. Your test assets should be in the stock package as well, but in your test folder.

1.2 Do this next

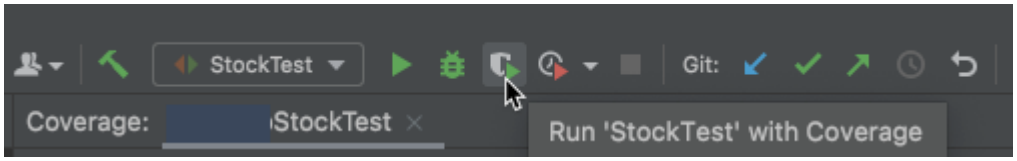
1. Download the **Transaction class** (https://northeastern.instructure.com/courses/123246/files/16688201/download?download_frd=1) we've created for this lab. The Transaction class is NOT functionally complete, but we're giving it to you so you can

get practice integrating some of your code with existing assets. Include the Transaction class in your stock package in your src directory and run it. Did your Stock class integrate well with the Transaction class? If not, write a 1-paragraph statement (plain-text file) indicating why you think things didn't work as anticipated.

2. Create two CRC cards that describe the two classes you've interacted with in this lab. You may use physical 3x5 index cards, or virtual CRC cards like the one below. Be sure to include all relevant details for the Stock and Transaction classes. When you submit your CRCs, jpg, png, pdf are all acceptable formats.



3. (Optional) Run your JUnit tests with code coverage. Take a screen capture of the results and save as a .png



4. **Submit your lab** to Gradescope. Be sure to include:

- your source code in the proper package structure (remember to REMOVE the Transaction class from your submission!),
- your test code in the proper package structure,
- your CRC card file(s) and your 1-paragraph write-up IF you did not get the Transaction class integrated.

- If you took a screen capture of your test coverage, you can submit that too. We'd love to see it.

Upload a zip (remember, the directory structure is not maintained via drag-n-drop of individual files) or submit via your GitHub account.

Note: This lab is giving you experience with Objects that "mutate" their values (e.g.: Stock objects can change their current price over the course of time). We'll talk more about the differences and trade-offs between mutable and non-mutable objects in future lectures.

Lab 1 - rubric

Full Credit - Pass	**Partial Credit - Pass**	**No Credit**
2 pts	**1 pt**	**0 pt**
Stock Solution passes all automated tests. Good OO theme implementation, documented (including CRC) and reasonable number of tests provided.	Stock Solution passes most automated tests. Good OO theme implementation. Some documentation (including CRC) and/or tests partially implemented	Stock solution passes few tests, problems with OO themes. Missing documentation, missing tests. Or assignment not submitted