Advanced Software Engineering
Exception Handlers: Team Assignment 4
Gabrielle Taylor: gat2118
Jackie Lin: jl4162
Leon Song: ls3233
Aaron Ong: ao2591

Demo and Black Box Testing

https://github.com/gabriellet/thepriceisright
https://trello.com/b/J5WehrH2

Using Tests

To run general tests, run: `./manage.py test`

If you want to test something specific: `./manage.py test .tests.TestClass.TestMethod`

Our test cases have additional dependencies — Six 1.1.0 and Mock 2.0.0. To install these packages, use `pip install <package name>`

Test Code

Our test code is located mainly in stocks/tests.py. Descriptions of our equivalence partitions are located in our Wiki at https://github.com/gabriellet/thepriceisright/wiki/Black-Box-Testing.

In the code, we implement the tests we discussed on the Wiki. Our main strategy was to create a set of test objects and environments. These include objects we both expect to fail and expect to throw errors, and then we see if our system handles those errors correctly.

For example, we create a ParentOrder object with a negative order quantity number, then call our is_valid() function to check that it returns False. To test the API responses from the exchange simulator, we used a mock response to simulate server failure, then observe whether our trade() algorithm handles server failure by marking the object status as FAILED.

Results of Testing

We had one critical failure regarding our tests that stumped us for a long time. If an order fails to sell, the exchange server should respond with a JSON string with ‘avg_price’ equal to 0. Our code tried to handle this by decoding every server response and checking ‘avg_price’.

In reality, the exchange server does not send a JSON response when an order fails. This means that we were calling json.loads() on a null string, causing our code to break.

Our tests helped us to realize this, because our system worked when we mocked a JSON object with ‘avg_price’ equal to 0, but when our system was running, failed orders kept breaking it. This was thus crucial in helping us realize that maybe the exchange server wasn’t sending us that JSON response, or any JSON response at all.

CRC Cards

CRC Card: Class Parent Order	
Responsibilities: ● Accept trade from user (i.e. when a user places an order on our app, it creates a Parent Order) ● UI: Show prompt confirming the details of the parent order (e.g. **sell** 100k shares of ACME ETF) ● Execute the Time Weighted Average Price algorithm given by JP Morgan (essentially, split the order into even chunks and sell those periodically. We call these chunks “Child Orders". So a 100k Parent Order might split itself into 10x10k Child Orders.) ● Split self into child orders ● Attempt to sell each child order through calling the market simulator API ● When selling a Child Order fails, try to sell it again after waiting a few minutes. ● If we encounter repeated failures, split the Child Order into even smaller chunks (e.g. 10k Child Order might now become 2x5k Child Orders). ● Keeps track of order progress (e.g. how many Child Orders have sold successfully)	Collaborators: ● User ● Child Order

CRC Card: Class Child Order	
Responsibilities: ● Note down the results of the API call e.g. if succeeded, note the price each chunk successfully sold for. (Each child order needs to notify its Parent Order when it succeeds/fails)	Collaborators: ● Parent Order

CRC Card: Class User	
Responsibilities: ● Keep track of all the Parent Orders associated with the user ● Initiate new Parent Orders when a user wants to ● Handle authentication i.e. Our app will have User objects with username/password fields to authenticate with	Collaborators: ● Parent Order

UML Diagram