Testing Report for Java Project 1

1. Trader

> Trader t = new Trader("John")

This tests the constructor and whether it takes the trader name without any errors. It should set our trader name to John. It does set the name to John and work.

> t.getName()

"John"

GetName should get the name from the constructor initialization, which in this case is John. We did get John from the getName method, which extracted it from the constructor.

> t.setName("Isaac")

> t.getName()

"Isaac"

This should change the name of the trader to whatever string we specified it to. We changed it to Isaac. Then, we check the result by getting the name and as we can see, it has changed it Isaac.

> t.getBalance()

1299.0

This should retrieve the value of the trader’s balance from the field balance which is initialized to 1299.0. As we can see, it does indeed give us 1299.0, which is the balance stored.

> t.withdraw(1200)

> t.getBalance()

99.0

This method should withdraw 1200 from the balance field and leave 99 dollars in the balance. We check this with getBalance and we see that our new balance is now 99 dollars, all in accordance.

> t.deposit(100)

> t.getBalance()

199.0

Next, we test the deposit method which should add money to the balance of the trader. We deposit a 100, which should give us 199 dollars in the balance. Indeed, we see that the getBalance method does indeed give us 199 dollars in the trader’s balance.

1. Order

> Order o = new Order('\*', 200, 3.5, true, new Trader("John"))

We test the constructor by initializing all the inputs and pray for no error when we hit enter. We also create Trader John.

> o.getStockSymbol()

'\*'

We test the getStockSymbol method, which should return the ‘\*’ that we initialized it to in the constructor. Indeed, the getStockSymbol gives us the asterisk we initialized it to.

> o.getNumberShares()

200

Next, we test the getNumberShares method, which also should return the 200 we initialized it to. And yes, it does work as it gives us 200.

> o.setNumberShares(300)

> o.getNumberShares()

300

We test the setNumberShares by setting the number of shares equal to 300. Then, we use the working getNumberShares to retrieve the value which should change to 300. In this case, it does change to 300.

> o.getPrice()

3.5

In order to obtain the price of the shares, we use the getPrice method, which should return 3.5, the value which we set it to in the constructor. We get 3.5, the method works.

> o.setPrice(4.5)

> o.getPrice()

4.5

Again, we test the setPrice method by setting the price of the shares to 4.5. Then, we use the getPrice method to see if the value changed to 4.5. Yes, it does change to 4.5, so it works.

> o.getAllOrNone()

True

We test whether the getAllOrNone method works, which should return a Boolean value of whatever we set it to. We set it to true in the beginning and when we use the getAllOrNone method, we see that it does return true.

> o.setAllOrNone(false)

> o.getAllOrNone()

False

We test the method setAllOrNone to see if we can alter the Boolean value that we initially set. This time, we change it to false from true. Then, we use the getAllOrNone method in order to see if the value has changed. It has changed to false; the method works.

> o.getTrader()

Trader@f94c40

We test whether the getTrader method works. It should return the location in memory of the field. It does return the location of Trader John.

>o.getTrader().getBalance()

1299.0

> o.getTrader().getName()

"John"

We should be able to use the getTrader to access Trader’s methods like getBalance and getName, which will return their respective values as they should as we tested them and they worked. In short, they return the right values and so it works.

1. BuyOrder

> BuyOrder b = new BuyOrder('a', 20, 2.5, true, new Trader("hi"))

> b.getTrader()

Trader@ef8fda

> b.getTrader().getName()

"hi"

> b.getTrader().getBalance()

1299.0

> b.getStockSymbol()

'a'

> b.getAllOrNone()

True

> b.getNumberShares()

20

> b.getPrice()

2.5

> b.setPrice(4.5)

> b.getPrice()

4.5

This section really doesn’t deserve own line by line write-ups. Basically, BuyOrder extends order which means that when we create a BuyOrder, it can do all the cool things order can. We initialize the constructor and this should allow us to get the number of shares, the stock symbol, the price of the shares and all the things order can as well as do the things trader can do by using the getTrader method. As tested before, all the methods work and we can indeed alter and get the price, get the stock symbol, get the name, and get the Boolean value for the allornone method. We can also access the balance of the trader and his name along with all the other relevant methods of trader if we so which to do so.

1. SellOrder

SellOrder s = new SellOrder ('a', 20, 2.5, true, new Trader("hi"))

> s.getTrader()

Trader@c7cdb7

> s.getTrader().getName()

"hi"

> s.getTrader().getBalance()

1299.0

> s.getStockSymbol()

'a'

> s.getAllOrNone()

True

> s.getNumberShares()

20

> s.getPrice()

2.5

> s.setPrice(4.5)

> s.getPrice()

4.5

SellOrder is another extension of order and basically functions the same as BuyOrder, so there should be relatively little explaining. Because I am lazy, I even did the same inputs and same testing process. When we initialize SellOrder, it should be able to do all the things Order can and Trader can via the getTrader method. We initialize the constructor using the inputs and voila, we are able to access the name of the trader, the price per share, the number of shares, the stock symbol, the balance, and pretty much every method that Trader and Order has in the bank. Since we know Trader and Order methods work very well, then we know that the extensions of them should work well too, which they do.

1. MarketBuyOrder

The constructor spits out an error if the last is not a market type.

> MarketBuyOrder m = new MarketBuyOrder('\*', 20, 2.5, true, new Market("marketname", 'b', 3.5, 4.5, 6, 10.5))

> m.getNumberShares()

20

> m.getAllOrNone()

False

> m.setAllOrNone(true)

> m.getAllOrNone()

false

> m.getStockSymbol()

'\*'

> m.setNumberShares(49)

> m.getNumberShares()

49

> m.getPrice()

2.5

> m.setPrice(3.4)

> m.getPrice()

3.4

The first part of MarketBuyOrder deals with the regular methods of Order. Again, by initializing everything in the constructor, we should be able to get the number of shares, the stock symbol, the number of shares, and the price. We should also be able to set the price and numbershares. However, we should always have AllOrNone be false no matter if we set it or not. As we see above, we are able to retrieve all of the relevant information stored in the constructor as well as set price and numbershares. However, no matter what, AllOrNone remains constant at false like it should.

> ((Market)m.getTrader()).getStockSymbol()

'b'

> ((Market)m.getTrader()).getCommision()

4.5

> ((Market)m.getTrader()).getTradeFee()

3.5

> ((Market)m.getTrader()).setTradeFee(5.5)

> ((Market)m.getTrader()).getTradeFee()

5.5

> ((Market)m.getTrader()).getMarketOrderSize()

6

> ((Market)m.getTrader()).setMarketOrderSize(10)

> ((Market)m.getTrader()).getMarketOrderSize()

10

> ((Market)m.getTrader()).getPriceOffset()

10.5

> ((Market)m.getTrader()).setPriceOffset(11.5)

> ((Market)m.getTrader()).getPriceOffset()

11.5

> ((Market)m.getTrader()).getName()

"marketname"

This part of the code allows us to see that we can access Market’s methods using the constructor that we set up and the code that we used in MarketBuyOrder. Using this, we should be able to get and set the trade fee, the order size, the price offset, and simply retrieve the symbol, commission and name. We typecast the trader to market in order to be able to use its methods. As we see, we indeed can access all the info stored in the market part of the constructor as well as alter it if need be.

1. Market Sell Order

This is much the same as the Market Buy Order; I even use the same variables XD.

MarketSellOrder s = new MarketSellOrder('\*', 20, 2.5, true, new Market("marketname", 'b', 3.5, 4.5, 6, 10.5))

> s.getNumberShares()

20

> s.getAllOrNone()

False

> s.setAllOrNone(true)

> s.getAllOrNone()

false

> s.getStockSymbol()

'\*'

> s.setNumberShares(30)

> s.getNumberShares()

30

> s.getPrice()

2.5

> s.setPrice(3.1)

> s.getPrice()

3.4

> ((Market)s.getTrader()).getStockSymbol()

'b'

> ((Market)s.getTrader()).getCommision()

4.5

> ((Market)s.getTrader()).getTradeFee()

3.5

> ((Market)s.getTrader()).setTradeFee(6.5)

> ((Market)s.getTrader()).getTradeFee()

6.5

> ((Market)s.getTrader()).getMarketOrderSize()

6

> ((Market)s.getTrader()).setMarketOrderSize(12)

> ((Market)s.getTrader()).getMarketOrderSize()

12

> ((Market)s.getTrader()).getPriceOffset()

10.5

> ((Market)s.getTrader()).setPriceOffset(11.5)

> ((Market)s.getTrader()).getPriceOffset()

11.5

> ((Market)s.getTrader()).getName()

"marketname"

We don’t really need to explain this one too in depth as it basically is a repeat of the Market Buy Order program. Using this constructor that we initialized at the very top, we are able to access both the Order’s method to fetch the data in the constructor as well as do the same for the Market. Again, we see that we can access all of the data in the constructor using Order’s methods as well as noticing that it always stays false in the AllOrNone method despite what we set it too. Then, we also see we can retrieve all the market data in the constructor using the Market method too.

1. Transaction

> Transaction t = new Transaction('\*', 2, 2.5, new Trader("Isaac"), new Trader ("Tyler"), new Market("marketname", 'b', 3.5, 4.5, 6, 10.5))

> t.getStockSymbol()

'\*'

> t.getNumberShares()

2

> t.getPrice()

2.5

> t.getSeller()

Trader@118b755

> t.getSeller().getName()

"Tyler"

> t.getBuyer()

Trader@15f8908

> t.getBuyer().getBalance()

1299.0

> t.isClosed()

False

After initializing the constructor, we should be able to access all the methods stored within Trader. I only did a couple as we know that the methods work within Trader and it’s a question of whether our constructor was designed well enough to return the values in the constructor. As we can see, I can get the name of the seller using Trader methods as well as get the balance of the buyer after finding the address of the buyer and seller I also test whether the basic methods in the program work like the isClosed, getPrices, getNumberShares, and obtaining the address of the buyer and seller. It does work as we see we can get all of these without error.

> t.getBuyer().getBalance()

1299.0

> t.getSeller().getBalance()

1299.0

> t.processTransaction()

> t.getBuyer().getBalance()

1279.25

> t.getSeller().getBalance()

1289.25

> t.processTransaction()

> t.getSeller().getBalance()

1289.25

> t.getBuyer().getBalance()

1279.25

Now we can test the last method. However, we do add the cost to the seller’s account as profit and subtract cost from the buyer’s account. We should be subtracting both the commission/2 and the transaction fee from the buyer and seller’s account. Then, we add that to the market’s accounts. Finally, we can check if the Balance has changed. The balance of the original 2 was set to 1299 and when we go through the process, we see that the buyer and seller balance has changed accordingly to the code. We can run through the transaction again and see that since it is closed, the balance of the buyer and seller no longer change. This is all correct and in line with the code.

1. Market

Let’s start with the simple ones:

> Market m = new Market ("marketname", 'a', 2.5, 4.5, 10, 20.5)

> m.getStockSymbol()

'a'

> m.getCommision()

4.5

> m.setCommision(5)

> m.getCommision()

5.0

> m.getTradeFee()

2.5

> m.setTradeFee(3.0)

> m.getTradeFee()

3.0

> m.getMarketOrderSize()

10

> m.setMarketOrderSize(20)

> m.getMarketOrderSize()

20

> m.getPriceOffset()

20.5

> m.setPriceOffset(20.3)

> m.getPriceOffset()

20.3

These are all the basic methods that allow us to get the various data that we set forth in the constructor and even change it. As we can see, they all work very well and we can set and get the required variables.

> Market somemarket = new Market("Isaac", 'd', 40, 0.3, 6000, 50000)

> MarketSellOrder mso = new MarketSellOrder('a', 51, 23, false, somemarket)

> MarketBuyOrder msb = new MarketBuyOrder('a', 51, 26, false, somemarket)

> somemarket.setMarketSellOrder(mso)

> somemarket.setMarketBuyOrder(msb)

> BuyOrder bo = new BuyOrder('a', 4, 23, false, new Trader("Isaac"))

> SellOrder bs = new SellOrder('a', 4, 21, false, new Trader("Isaac"))

> somemarket.addOrderToMarket(bo)

> somemarket.addOrderToMarket(bs)

> SellOrder secondbs = new SellOrder('a', 4, 20, false, new Trader("Isaac"))

> somemarket.addOrderToMarket(secondbs)

> somemarket.currentMarketBuyPrice()

26.0

> somemarket.currentMarketSellPrice()

23.0

We initialize the values of the Market and run through the process. We also add the order to the market and it works and we also set both the buy orders and sell orders for the markets. We thus see that the set and get methods work. We see that this should have returned the best buy price of the market and it does, returning us the 26. It also should return the best sell price of the market and it replies with the 23. We see that the currentmarketbuyprice and currentmarketsellprice also work as they return the values.

> somemarket.isOpen()

False

This method should turn false because of how the buy price is higher than the sell price.

The rest of the code I can’t figure out how to work and this is due to the fact that no matter what I do, the market never opens and is always closed. I’ve looked over my code and never been able to figure out why this is happening. I feel confident in the rest of my code but simply can’t proceed due to this roadblock.

> BuyOrder bo = new BuyOrder('a', 4, 21, false, new Trader("Isaac"))

> SellOrder bs = new SellOrder('a', 4, 23, false, new Trader("Isaac"))

> somemarket.isOpen()

false

> somemarket.isOpen()

false

> double z = bo.getPrice()

> z

23.0

> double x = bs.getPrice()

> x

21.0

> BuyOrder bo = new BuyOrder('a', 4, 21, false, new Trader("Isaac"))

> SellOrder bs = new SellOrder('a', 4, 23, false, new Trader("Isaac"))

> somemarket.isOpen()

false

>