**Tung Ho, Lin txl429**

**Project 1 Testing Report**

**To Graders:**

**-because this testing report has been written in separate periods of time, some values of the variables that have the same name might not equal to each other because even though each time I executed my previous interactions, a new value is generated. I ensured every value of the same variable, although different, is identical to the value of where that particular variable originated from.**

**-I apologize for this format of my testing report. This is the only format I can think of and I will make sure I will find a better format next time.**

**Class Trader**

Testing: Comments:

Trader t1=new Trader("Mary") //testing if the constructor initializes the instance of the class and create an instance with the entered parameters//

> t1.getName() //testing the getName method, it should return the name entered//

"Mary" //it returns the name I entered//

> t1.setName("Lori") //testing the setName method//

> t1.getName()

"Lori" //it returns the new name I entered//

> t1.getBalance() //testing the getBalance method, it should return the balance//

0.0 //it returns the balance stored in default, which is $0.0//

> t1.deposit(100.0) //testing the deposit method, it should deposit the money//

> t1.getBalance()

100.0 //the balance increases by the deposit//

> t1.withdraw(100.0) //testing the withdraw method, it should withdraw the money//

> t1.getBalance()

0.0 //the balance decreases by the withdrawal//

//everything is correct in the class Trader//

**Class Order**

Testing: Comments:

> Order o1=new Order('c',10,10.0,true,t1) //testing if the constructor initializes the instance of the class and create an instance with the entered parameters//

> o1.getStockSymbol() //testing the getStockSymbol method, it should return the stock symbol entered//

'c' //it returns the stock symbol entered//

> o1.getNumberShares() //testing the getNumberShares method, it should return the no. of shares//

10 //it returns the no. of shares//

> o1.setNumberShares(20) //testing the setNumberShares method//

> o1.getNumberShares()

20 //it returns the new number of shares//

> o1.getPrice() //testing the getPrice method, it should return the price entered//

10.0 //it returns the price entered//

> o1.setPrice(100.0) //testing the setPrice method//

> o1.getPrice()

100.0 //it returns the new price//

> o1.getAllOrNone() //testing the getAllOrNone method , it should return whether it is all or none//

true //it returns the all or none value entered//

> o1.setAllOrNone(false) //testing the setAllOrNone method//

> o1.getAllOrNone()

false //it returns the new all or none value//

> o1.getTrader() //testing if it returns the trader value//

Trader@2aa937cd //it returns the trader value//

//Everything is correct in the class Order//

**Class BuyOrder**

Testing: Comments:

> BuyOrder b1=new BuyOrder('c',10,10.0,true,t1) //testing if the constructor initializes the instance of the class and create an instance with the entered parameters//

> b1.getStockSymbol() //testing the getStockSymbol method , it should return the stock symbol entered//

'c' //it returns the stock symbol entered//

> b1.getNumberShares() //testing the getNumberShares method , it should return the no. of shares//

10 //it returns the no. of shares//

> b1.setNumberShares(20) //testing the setNumberShares method//

> b1.getNumberShares()

20 //it returns the new number of shares//

> b1.getPrice() //testing the getPrice method, it should return the price entered//

10.0 //it returns the price entered//

> b1.setPrice(70.0) //testing the setPrice method//

> b1.getPrice()

70.0 //it returns the new price//

> b1.getAllOrNone() //testing the getAllOrNone method, it should return whether it is all or none//

true //it returns the all or none value entered//

> b1.setAllOrNone(false) //testing the setAllOrNone method//

> b1.getAllOrNone()

false //it returns the new all or none value//

> b1.getTrader() //testing if it returns the trader value//

Trader@2aa937cd //it returns the trader value//

//Everything is correct in the class BuyOrder//

**Class SellOrder**

Testing: Comments:

> SellOrder s1=new SellOrder('c',10,30.0,true,t1) //testing if the constructor initializes the instance of the class and create an instance with the entered parameters//

> s1.getStockSymbol() //testing the getStockSymbol method , it should return the stock symbol entered//

'c' //it returns the stock symbol entered//

> s1.getNumberShares() //testing the getNumberShares method, it should return the no. of shares//

10 //it returns the no. of shares//

> s1.setNumberShares(40) //testing the setNumberShares method//

> s1.getNumberShares()

40 //it returns the new number of shares//

> s1.getPrice() //testing the getPrice method, it should return the price entered//

30.0 //it returns the price entered//

> s1.setPrice(50.0) //testing the setPrice method//

> s1.getPrice()

50.0 //it returns the new price//

> s1.getAllOrNone() //testing the getAllOrNone method, it should return whether it is all or none//

true //it returns the all or none value entered//

> s1.setAllOrNone(false) //testing the setAllOrNone method//

> s1.getAllOrNone()

false //it returns the new all or none value//

> s1.getTrader() //testing if it returns the trader value//

Trader@2aa937cd //it returns the trader value//

//Everything is correct in the class SellOrder//

\*An instance of Market named market1 has been created in order to create the MarketBuyOrder instance and the MarketSellOrder instance.

Market market1=new Market("market1", 'c', 10.0, 10.0, 200, 50.0);

**Class MarketBuyOrder**

Testing: Comments:

> MarketBuyOrder mmbuyorder1=new MarketBuyOrder('c', 10, 100.0, true, m1) //testing if the constructor initializes the instance of the class and create an instance with the entered parameters//

> mmbuyorder1.getStockSymbol() / /testing the getStockSymbol method , it should return the stock symbol entered//

'c' //it returns the stock symbol entered//

> mmbuyorder1.getNumberShares() //testing the getNumberShares method, it should return the no. of shares//

10 //it returns the no. of shares//

> mmbuyorder1.setNumberShares(20) //testing the setNumberShares method//

> mmbuyorder1.getNumberShares()

20 //it returns the new number of shares//

> mmbuyorder1.getPrice() //testing the getPrice method, it should return the price entered//

100.0 //it returns the price entered//

> mmbuyorder1.setPrice(500.0) //testing the setPrice method//

> mmbuyorder1.getPrice()

500.0 / /it returns the new price//

> mmbuyorder1.getAllOrNone() //testing the getAllOrNone method, it should always return false//

false //it returns false//

> mmbuyorder1.getTrader() //testing if it returns the market trader value//

Market@102b1682 //it returns the market trader value//

//Everything is correct in the class MarketBuyOrder//

**Class MarketSellOrder**

Testing: Comments:

MarketSellOrder mmsellorder1=new MarketSellOrder('c', 8, 80.0, true, m1) //testing if the constructor initializes the instance of the class and create an instance with the entered parameters//

> mmsellorder1.getStockSymbol() //testing the getStockSymbol method , it should return the stock symbol entered//

'c' //it returns the stock symbol entered//

> mmsellorder1.getNumberShares() //testing the getNumberShares method, it should return the no. of shares//

8 //it returns the no. of shares//

> mmsellorder1.setNumberShares(10) //testing the setNumberShares method//

> mmsellorder1.getNumberShares()

10 //it returns the new number of shares//

> mmsellorder1.getPrice() //testing the getPrice method, it should return the price entered//

80.0 //it returns the price entered//

> mmsellorder1.setPrice(120.0) //testing the setPrice method//

> mmsellorder1.getPrice()

120.0 //it returns the new price//

> mmsellorder1.getAllOrNone() //testing the getAllOrNone method, it should always return false//

false //it returns false//

> mmsellorder1.getTrader() //testing if it returns the market trader value//

Market@102b1682 //it returns the market trader value//

//Everything is correct in the class MarketSellOrder//

**Class Transaction**

Testing: Comments:

Trader t2=new Trader("Susan") //a new instance of Trader t2 is created

> t1.deposit(10000.0) //10000 is deposited into t1

> t2.deposit(10000.0) //10000 is deposited into t2

> Transaction transaction1=new Transaction('c', 5, 80.0, t1, t2, market1) //testing if the constructor initializes the instance of the class. and create an instance with the entered parameters/

> transaction1.getStockSymbol() /testing the getStockSymbol method , it should return the stock symbol entered//

'c' //it returns the stock symbol entered//

> transaction1.getNumberShares() //testing the getNumberShares method, it should return the no. of shares//

5 //it returns the no. of shares//

> transaction1.getPrice() //testing the getPrice method, it should return the price entered//

80.0 //it returns the new price//

> transaction1.getMarketMaker() //testing the getMarketMaker method, it should return the market value entered//

Market@60dc4c81 //it returns the market value entered//

> transaction1.getBuyer() //testing the getBuyer method, it should return the Trader buyer value//

Trader@45b3278a //it returns the Trader buyer value//

> transaction1.getSeller() //testing the getSeller method, it should return the Trader seller value//

Trader@1a2d502d //it returns the Trader seller value//

> transaction1.IsClosed() //testing the isClosed method, it should return whether the transaction is closed or not//

false //it returns false, which means the transaction is still open//

> transaction1.processTransaction() //testing the processTransaction method, it should evaluate the value of isClosed and if it is true, then the method does nothing; and if it is false, it will withdraw the (price times shares) from the buyer and deposit it into the seller. It will also withdraw ((commission times trade fee)/2) from both the buyer and the seller and finally set the value of isClosed to true//

> t1.getBalance() //getting the balance of t1 to see if the correct amount of money is deposited/withdrawn/

9550.0 //(price times share)=5\*80.0=400.0, ((commission times trade fee)/2)=10.0\*10.0/2=50.0, 10000.0-(400.0+50.0)=9550.0; the method processTransaction produced the correct final balance for t1//

> t2.getBalance()

10350.0

//(price times share)=5\*80.0=400.0, ((commission times trade fee)/2)=10.0\*10.0/2=50.0, 10000.0+400.0-50.0=10350.0; the method processTransaction produced the correct final balance for t2//

> transaction1.IsClosed() //the method processTransaction should change the value of isClosed to true after processing the transaction//

true //it did change the value from false to true, which means the transaction is closed//

> transaction1.processTransaction() //testing the method when isClosed is true//

> t1.getBalance()

9450.0

> t2.getBalance()

10450.0

//the method processTransaction does nothing to t1 and t2//

//Everything in class Transaction is correct//

**Class Market**

Testing: Comments:

>Market market1=new Market("market1", 'c', 10.0, 10.0, 200, 50.0); //testing if the constructor initializes the instance of the class and create an instance with the entered parameters//

> market1.getStockSymbol() //testing the getStockSymbol method , it should return the stock symbol entered//

'c' //it returns the stock symbol entered//

> market1.getCommission() //testing the getCommission method, it should return the commission entered//

10.0 //it returns the commission entered//

> market1.setCommission(12.0) //testing the setCommission method//

> market1.getCommission()

12.0 //it returns the new commission//

> market1.getTradeFee() //testing the getTradeFee method, it should return the trade fee entered//

10.0 //it returns the trade fee entered//

> market1.setTradeFee(15.0) //testing the setTradeFee method//

> market1.getTradeFee()

15.0 //it returns the new trade fee//

> market1.getMarketOrderSize() //testing the getTradeFee method, it should return the market order size entered//

100 //it returns the market order size entered//

> market1.setMarketOrderSize(70) //testing the setMarketOrderSize method//

> market1.getMarketOrderSize()

70 //it returns the new market order size//

> market1.getPriceOffset() //testing the getPriceOffset method, it should return the price offset entered//

30.0 //it returns the price offset entered//

> market1.setPriceOffset(50.0) //testing the setPriceOffset method//

> market1.getPriceOffset()

50.0 //it returns the new price offset//

>market1.isOpen() //testing the isOpen method, it should return true only if market buy order and market sell order exist and the price of buy order is larger than that of sell order

false //returns false because there is no market buy order or market sell order

market1.setMarketBuyOrder(mmbuyorder1) //testing the setMarketBuyOrder method//

> market1.getMarketBuyOrder()

MarketBuyOrder@7463e563 //it returns the new MarketBuyOrder//

> market1.setMarketSellOrder(mmsellorder1) //testing the setMarketSellOrder method//

> market1.getMarketSellOrder()

MarketSellOrder@76d78df0 //it returns the new MarketSellOrder//

>market1.isOpen()

false //returns false because the price of market buy order is not larger than that of market sell order//

> mmsellorder1.setPrice(120.0)

> market1.isOpen()

true //returns true because all conditions are satisfied//

b1

BuyOrder@7c6c2896 //value of b1

market1.addOrderToMarket(b1) //testing the method addOrderToMarket, it should add the buy order to the market; and if it has the highest price, it will become the best buy order, otherwise it will become the second best buy order.

> market1.getBestBuyOrder() //testing the getBestBuyOrder method, it should return the best buy order in the market

BuyOrder@7c6c2896 //it returns the value of b1

> > BuyOrder b2=new BuyOrder('c', 10, 20.0, true, t1) //new instance b2 created with price higher than b1//

> b2

BuyOrder@5631573 //value of b2

> market1.addOrderToMarket(b2)

> market1.getBestBuyOrder()

BuyOrder@5631573 //now the best buy order is b2 because b2 has a higher price than b1//

> m1.get2ndBestBuyOrder() //now the second best buy order is b1//

BuyOrder@7c6c2896

> SellOrder s1=new SellOrder('c',10,50.0,true,t2)

> SellOrder s2=new SellOrder('c',10,60.0,true,t2) //two new instances of SellOrder is created//

> s1

SellOrder@4268cc6 //value of s1//

> s2

SellOrder@6460029d //value of s2//

> m1.addOrderToMarket(s1)

> m1.addOrderToMarket(s2)

> m1.getBestSellOrder()

SellOrder@6460029d //s2 has a higher price and therefore is the best sell order//

> m1.get2ndBestSellOrder()

SellOrder@4268cc6 //s1 becomes the second best buy order//

> BuyOrder b3=new BuyOrder('c', 10, 30.0, false, t1) //2 new instances of BuyOrder and SellOrder are created//

> SellOrder s3=new SellOrder('c', 20, 20.0, false, t1)

> BuyOrder b4=new BuyOrder('c', 10, 30.0, true, t1)

> SellOrder s4=new SellOrder('c', 5, 20.0, false, t1)

>m1.matchingOrders(b3,s3)

true //returns true because not any of b3 and s3 are all or none//

> m1.matchingOrders(b4,s4)

false //returns false because s4 does not have enough shares for b4//

> m1.matchingOrders(b1,s1)

false //returns false because s1 has a higher price than b1//

> b1.setPrice(70.0)

> m1.matchingOrders(b1,s1)

true //returns true now because b1 has a higher price than s1//

//since the testing report is written in different times, these MarketBuyOrder instances have been created again, and are different from those above//

> MarketBuyOrder mmbuyorder1=new MarketBuyOrder('c', 20, 20.0, false, m1)

> MarketSellOrder mmsellorder1=new MarketSellOrder('c', 20, 70.0, false, m1)

> m1.setMarketBuyOrder(mmbuyorder1)

> m1.setMarketSellOrder(mmsellorder1)

>m1.currentMarketBuyPrice() //testing the currentMarketBuyPrice method, it should return the highest buying price,//

70.0 //it returns the price of b1//

> m1.currentMarketSellPrice() //testing the currentMarketSellPrice method, it should return the highest selling price//

60.0 //it returns the price of s2//

> Order o1=new Order('c',5,5.0, false, t1)

> Order o2=new Order('c',5, 30.0, false, t1)

> Order o3=new Order('c',5, 100.0, false, t1) //new instances of Order are created//

> m1.isValidOrder(o1)

false //returns false because o3’s price is less than market buy order’s price//

> m1.isValidOrder(o2)

true //returns true because o3’s price is more than market buy order’s and less than market sell order’s price//

> m1.isValidOrder(o3)

false //returns false because o3’s price is more than market sell order’s price//

> m1.placeOrder(b1) //testing the placeOrder method, a transaction should occur if there is a matching order in the sell orders for b1, also when the market is open and the order is valid, and when the shares of the matching sell order’s shares is larger than b1. b1, which is the best buy order, matches with s2, which is the best sell order. A Transaction value should be returned//

Transaction@4959d87f //returns a Transaction value//

> m1.getBestSellOrder()

SellOrder@4268cc6 //the best buy order is now s1//

> s1

SellOrder@4268cc6

> m1.get2ndBestSellOrder()

null //the second best buy order is now nothing//

> m1.placeOrder(s1)

null //A transaction cannot occur because there are no matching orders for s1//

>m1.placeOrder(s3)

Transaction@5ef77511 //A transaction occurs because s3 matches with any of the buying orders//

//Everything in class Market is correct//

//Everything in Project 1 is correct//