BESTWAY STOCK TRADER

Always The Best Way to Buy and Sell Stocks

CS 4310 Software Engineering I: Requirements

Team Members:

Instructor: Prof. Ted Billard
Stock Trader: System Narrative

Why Bestway Stock Trader

For intelligent investors who need to quickly take advantage of market conditions with proven trading strategies and capable tools at their fingertips, the Bestway Stock Trader is a web-based securities management tool that enables investors to maximize their assets and minimize their risk. Unlike other online trading tools, our product appeals to new and seasoned investors alike by providing an intuitive but powerful interface to a suite of market analytics combined with advanced trading logic for smarter and safer trades.

Create New Investor

New investors are welcomed into the system with an easy interface to provide investor identification, password, name, phone, and other personal data. A first account is then opened for the investor.

Create New Account

Besides the original account, say for growth, the investor may open additional accounts for IRA, college funds, etc. Investors can move between accounts and keep investment activities associated with specific accounts.

Login

Registered investors log into their accounts with an investor ID and password.

Display Account Information

Investors have access to various displays, including pending orders, a history of completed transactions, and a portfolio of stocks and cash. The portfolio computes the market value of all stocks based upon current market conditions.

Trade Stock

Investors can buy and sell stocks in a real-time environment. Before trading, the investor probably wants to examine current market conditions. Quotes are available for individual stocks, including recent trade prices. An investor interested in buying a stock can see the minimum sell price offered. Likewise, an investor interested in selling a stock can see the maximum buy price offered.

After making decisions, an investor can place buy or sell orders for specific stocks, including the number of shares and the desired price. The Bestway Stock Trader will execute these trades within one second and provide confirmation. The investor’s portfolio is updated and a transaction is generated. Sometimes the price does not match current market conditions with the result that a pending order is generated for future markets. If the price is good but the number of shares is high, then the system transacts with as many other offers as possible, with the remaining quantity posted as a pending order.
Stock Trader: System Narrative

Investors can view their portfolio, pending orders, and transactions to monitor their trading activities. Investors can change pending orders for price and share, or delete the order.

Investors also have access to real-time graphs which trend the market conditions. Other advanced features include watchlists, stock filters, trading wizards, alert notification, and stock price streaming.

Cash Management

A sale of a stock results in increased cash value within the investor’s account. Of course, stock purchases require cash but investors can trade on margin, using their stocks as security.

Investors can deposit and withdraw cash into a specific account at any time.

Logout

Investors can exit the system at any time.

Conclusions

Our system provides all the functionality, with an easy-to-use interface, that the investor needs for successful stock management.

Open an account today and start trading!

The Bestway Stock Trader is always the best way to buy and sell stocks.
<table>
<thead>
<tr>
<th>FIELD</th>
<th>VALUE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Investor ID</td>
<td>jones</td>
<td>Investor’s unique ID</td>
</tr>
<tr>
<td>Password</td>
<td>jones</td>
<td>Secret password to login</td>
</tr>
<tr>
<td>First</td>
<td>John</td>
<td>Investor’s first name</td>
</tr>
<tr>
<td>Last</td>
<td>Jones</td>
<td>Investor’s last name</td>
</tr>
<tr>
<td>Phone</td>
<td>510-111-1111</td>
<td>Investor’s home phone number</td>
</tr>
<tr>
<td>Account#</td>
<td>1000</td>
<td>Investor currently logged into this account</td>
</tr>
<tr>
<td>Cash</td>
<td>1000.00</td>
<td>used to deposit/withdraw money from account</td>
</tr>
<tr>
<td>Stock</td>
<td>IBM</td>
<td>used for stock quote or to buy/sell</td>
</tr>
<tr>
<td>Shares</td>
<td>10</td>
<td>how many shares to buy/sell</td>
</tr>
<tr>
<td>Price</td>
<td>$82.50</td>
<td>how much is investor willing to pay, or must receive</td>
</tr>
</tbody>
</table>
Stock Trader: Output Reports

Stock Quote:
Quote: IBM Minimum SELL: $86.00 Maximum BUY: $82.50 Last Sale: $84.00

Transactions:

<table>
<thead>
<tr>
<th>Trans#</th>
<th>Date</th>
<th>Account</th>
<th>Stock</th>
<th>Type</th>
<th>Price</th>
<th>Shares</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Fri May 05 15:49:33 PDT 2006</td>
<td>1001</td>
<td>CASH DEPOSIT</td>
<td>BUY</td>
<td>$1000.00</td>
<td>1</td>
</tr>
<tr>
<td>1003</td>
<td>Fri May 05 15:49:33 PDT 2006</td>
<td>1001</td>
<td>IBM</td>
<td>BUY</td>
<td>$80.00</td>
<td>5</td>
</tr>
<tr>
<td>1004</td>
<td>Fri May 05 15:49:33 PDT 2006</td>
<td>1001</td>
<td>IBM</td>
<td>SOLD</td>
<td>$82.50</td>
<td>5</td>
</tr>
<tr>
<td>1005</td>
<td>Fri May 05 15:49:33 PDT 2006</td>
<td>1001</td>
<td>CSCO</td>
<td>BUY</td>
<td>$20.00</td>
<td>100</td>
</tr>
</tbody>
</table>

Pending Orders:

<table>
<thead>
<tr>
<th>Order#</th>
<th>Date</th>
<th>Account</th>
<th>Stock</th>
<th>Type</th>
<th>Price</th>
<th>Shares</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>Fri May 05 15:49:33 PDT 2006</td>
<td>1000</td>
<td>INTC</td>
<td>BUY</td>
<td>$19.0</td>
<td>50</td>
</tr>
<tr>
<td>1005</td>
<td>Fri May 05 15:49:33 PDT 2006</td>
<td>1000</td>
<td>IBM</td>
<td>SELL</td>
<td>$86.0</td>
<td>5</td>
</tr>
</tbody>
</table>

Portfolio:

<table>
<thead>
<tr>
<th>Stock</th>
<th>Shares</th>
<th>Market</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM</td>
<td>10</td>
<td>$840.00</td>
</tr>
<tr>
<td>CSCO</td>
<td>100</td>
<td>$2200.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$3040.00</td>
</tr>
<tr>
<td>Cash</td>
<td></td>
<td>$1000.00</td>
</tr>
<tr>
<td>Net Worth</td>
<td></td>
<td>$4040.00</td>
</tr>
</tbody>
</table>
# BESTWAY STOCK TRADER
## User’s Manual

<table>
<thead>
<tr>
<th><strong>Function:</strong></th>
<th>Buy Stock</th>
<th><strong>Synopsis:</strong></th>
<th>Use cash in investor’s account to purchase new shares of stock.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inputs:</strong></td>
<td>Stock</td>
<td>symbol of stock to purchase</td>
<td>Shares: number of shares to purchase</td>
</tr>
<tr>
<td><strong>Actions:</strong></td>
<td>Move cursor to Buy button, click on button.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Outputs:</strong></td>
<td>Confirmation: symbol, shares, price, remaining pending order.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td>Stock: INTC</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Shares: 100</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Price: 22.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sale completed: 75 shares of INTC at $22.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Current outstanding order: BUY 25 shares of INTC at $22.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| **Function:** | Sell Stock |
## Bestway Stock Trader

<table>
<thead>
<tr>
<th>File</th>
<th>Investor ID: jones</th>
<th>Password: *****</th>
<th>Account: 1000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>First: John</td>
<td>Last: Jones</td>
<td>Phone: 510-111-1111</td>
</tr>
<tr>
<td></td>
<td>Cash: _________</td>
<td>Shares: ________</td>
<td>Price: ________</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Login</th>
<th>New Investor</th>
<th>Deposit</th>
<th>Buy</th>
<th>Pending Orders</th>
<th>Quote</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chg Acct</td>
<td>New Account</td>
<td>Withdraw</td>
<td>Sell</td>
<td>Transactions</td>
<td>Portfolio</td>
</tr>
</tbody>
</table>

### PORTFOLIO:

<table>
<thead>
<tr>
<th>Stock</th>
<th>Shares</th>
<th>Market</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM</td>
<td>10</td>
<td>640.00</td>
</tr>
<tr>
<td>CSCO</td>
<td>100</td>
<td>2200.00</td>
</tr>
</tbody>
</table>

$3040.00

Cash $1000.00

Net Worth $4040.00
Stock Trader: Database Design

Database Schema:

Investors([INVESTOR_ID, PASSWORD, FIRST, LAST, PHONE])

Accounts([ACCOUNT_NO, INVESTOR_ID, CASH])

Holdings([ACCOUNT_NO, SYM, QTY])

Orders([ORDER_NO, DATE, ACCOUNT, SYM, TYPE, PRICE, QTY])

Transactions([TRANS_NO, DATE, ACCOUNT, SYM, TYPE, PRICE, QTY])

Stocks([SYM, PRICE])
Investor → Stock Trading System

buy, account, stock, price, qty

confirmation
Stock Trader Object-Oriented Design: Domain Objects
<table>
<thead>
<tr>
<th>DATA STRUCTURE</th>
<th>OPERATION</th>
<th>AVERAGE-CASE</th>
<th>WORST-CASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stock Hasetable</td>
<td>Find</td>
<td>O(1)</td>
<td>O(x)</td>
</tr>
<tr>
<td></td>
<td>Insert</td>
<td>O(1)</td>
<td>O(x)</td>
</tr>
<tr>
<td></td>
<td>Delete</td>
<td>O(1)</td>
<td>O(x)</td>
</tr>
<tr>
<td>Min/Max Heap</td>
<td>Findmin/max</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td></td>
<td>Insert</td>
<td>O(log m)</td>
<td>O(log n)</td>
</tr>
<tr>
<td></td>
<td>Delete</td>
<td>O(log m)</td>
<td>O(log n)</td>
</tr>
<tr>
<td>Match Overall</td>
<td></td>
<td>O(log m)</td>
<td>O(x + log n)</td>
</tr>
</tbody>
</table>

n = total # of pending orders
x = # of stock symbols
m = average # of pending/orders per stock = n/x
n = worst-case # of pending orders per stock (all orders are for just one stock)
scanOrders(account, stock, TYPE, QTY, PRICE) {
    while not done do {
        if this order’s QTY = 0 then done is TRUE // QTY consumed
        else {
            if TYPE is BUY then { // and similar for SELL
                if SELL minheap is empty then done is TRUE
                access order1 in the root of the SELL minheap
                if new order’s PRICE < order1’s PRICE then done is TRUE // no more possible matches
                else {
                    q = min of new order’s QTY and order1’s QTY1
                    this order’s QTY -= q
                    order1’s QTY1 -= q
                    update cash and generate transactions for both accounts
                    update stock market value
                    adjust holdings for both accounts
                    - if new stock, insert
                    - if old stock, update
                    - if selling all quantity, delete
                    if order1’s QTY1 = 0 then delete from minheap // QTY1 consumed
                }
            }
        }
    }
    if this order’s QTY > 0 then { // no more matches, QTY not consumed
        generate order number and insert this order into
        - BUY maxheap
        - account’s vector of orders
    }
} // end if statement
} // end if statement
} // end while loop
} // end constructor
/* FUNCTION: ReceiveBuyOrder  
PURPOSE : To process a new buy order  
INPUTS :  
    account purchaser  
    stock to buy  
    price offered  
    qty of shares to buy  
RETURN : confirmation or error  
ALGORITHM: Given a purchase request, locate the best sell, adjust holdings,  
generate transactions, update the market value  
*/
int ReceiveBuyOrder(int account, char *stock, double price, int qty);

Alternatively:

int ReceiveBuyOrder( // process a purchase order, return confirmation  
    int account, // purchaser  
    char *stock, // to buy  
    double price, // offered  
    int qty // shares to buy  
);

order_ptr RetrieveBestSell(char *stock);

void AdjustHoldings(int account, char *stock, int qty, int type);

void GenerateTransaction(int account, char *stock, int type, price, qty);

void UpdateMarketValue(char *stock, price);
doSale(a1,a2,s,p,q)

:TradeExecution

1: CASH+ = q*p

2: CASH- = q*p

3: remove()

4: create(a2,s,q)

5: create(a1,s,SELL,p,q)

6: create(a2,s,BUY,p,q)

7: PRICE:=p

a1:Account

a2:Account

h1:Holding

h2:Holding

t1:Transaction

t2:Transaction

s:Stock
doSale(a₁,a₂,s,p,q)

:TradeExecution → a₁:Account → a₂:Account → h₁:Holding → h₂:Holding

\[ \text{CASH}^+ = q^*p \]

\[ \text{CASH}^- = q^*p \]

\[ \text{remove()} \]

\[ \text{create}(a₂,s,q) \]

\[ \text{create}(a₁,s,\text{SELL},p,q) \]

\[ \text{create}(a₂,s,\text{BUY},p,q) \]

\[ \text{PRICE} := p \]
Assuming TYPE= “BUY” stock:

<table>
<thead>
<tr>
<th>#</th>
<th>PATH</th>
<th>PRICE</th>
<th>PRICE1</th>
<th>QTY</th>
<th>QTY1</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>0-1-10-11-12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>infeasible, done=T initially</td>
</tr>
<tr>
<td>2.</td>
<td>0-1-10-12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>infeasible, QTY &gt;0 initially</td>
</tr>
<tr>
<td>3.</td>
<td>0-1-2-3-1-10-12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>infeasible, QTY &gt;0 initially</td>
</tr>
<tr>
<td>4.</td>
<td>0-1-2-4-5-6-1-10-11-12</td>
<td>$48</td>
<td>$50</td>
<td>*</td>
<td>*</td>
<td>no match at all</td>
</tr>
<tr>
<td>5.</td>
<td>0-1-2-4-5-7-8-1-2-3-1-10-12</td>
<td>$50</td>
<td>$48</td>
<td>100sh</td>
<td>110sh</td>
<td>only QTY consumed</td>
</tr>
<tr>
<td>6.</td>
<td>0-1-2-4-5-7-8-9-1-2-4-5-6-1-10-11-12</td>
<td>$50</td>
<td>$48</td>
<td>100sh</td>
<td>90sh</td>
<td>only QTY1 consumed</td>
</tr>
<tr>
<td>TEST</td>
<td>INPUTS</td>
<td>EXPECTED OUTPUTS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>--------</td>
<td>------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 1.   | Investor ID: jones  
Password: jones  
push Login | First: John  
Last: Jones  
Account: 1000  
Phone: 510-111-1111 |
| 2.   | push Pending Orders  
push Transactions  
push Portfolio | Pending Orders: 2  
Transactions: 4  
Cash: 1000.00  
Net Worth: 4040.00 |
| 3.   | Cash: 1000  
push Deposit  
Cash: 500  
push Withdraw  
push Portfolio | Cash: 1500.00 |
| 4.   | Stock: INTC  
push Quote  
Cash: 1000 | Min SELL: 21.00  
Max BUY: 19.00  
Last Sale: 20.00 |
| 5.   | Stock: INTC  
Shares: 100  
Price: 19.50  
push Buy | Outstanding Order: 100 sh |
| 6.   | Price: 22  
push Buy | Sale completed: 75 sh  
Outstanding Order: 25 sh |
| 7.   | Shares: 25  
Price: 23  
push Buy | Sale completed: 25 sh |
| 8.   | Stock: IBM  
Shares: 10  
Price: 82 | Sale completed: 5 sh  
Sale completed: 5 sh |
| 9.   | push Pending Orders  
push Transactions  
push Portfolio | Pending Orders: 4  
Transactions: 10  
Cash: 95.00  
Net Worth: 4595.00 |
| 10.  | push New Account  
Account: 1003  
push Chg Acct  
push Portfolio | Account: 1003  
Cash: 0.00  
Net Worth: 0.00 |
| 11.  | Investor ID: james  
First: Henry  
Last: James  
Phone: 510-444-4444  
push New Investor  
push Portfolio | Account: 1004  
Cash: 0.00  
Net Worth: 0.00 |
import java.awt.*;
import java.applet.***;

public class StockTest {

    public StockTest( // called by StockApplet when Test button is pushed
        // must have access to applet to set fields and textarea to print
            StockApplet applet, TextArea ta, TextField acctField, TextField firstField,
        TextField lastField, TextField idField, TextField phoneField, TextField passField,
        TextField cashField, TextField stockField, TextField sharesField, TextField priceField) {

            ta.append("-----------------------------------------\n");
            ta.append("BEGIN: AUTOMATED TEST \n");
            ta.append("-----------------------------------------\n");

            ta.append("-----------------------------------------\n");
            ta.append("TEST: 1 | Investor ID: jones | First: John \n");
            ta.append(" | Password: jones | Last: Jones \n");
            ta.append(" | push Login | Account: 1000 \n");
            ta.append(" | | Phone: 510-111-1111 \n");
            ta.append("-----------------------------------------\n");

            // set the fields in the applet, call doAction just like a button push
        // let applet do the usual calls to StockReport
            idField.setText("jones");
            passField.setText("jones");
            applet.doAction("Login");

            ta.append("-----------------------------------------\n");
            ta.append("TEST: 2 | push Pending Orders | Pending Orders: 2 \n");
            ta.append(" | push Transactions | Transactions: 4 \n");
            ta.append(" | push Portfolio | Cash: 1000.00 \n");
            ta.append(" | | Net Worth: 4040.00 \n");
            ta.append("-----------------------------------------\n");
            applet.doAction("Pending Orders");
            applet.doAction("Transactions");
            applet.doAction("Portfolio");

            // ETC.

            ta.append("-----------------------------------------\n");
            ta.append("END: AUTOMATED TEST \n");
            ta.append("-----------------------------------------\n");
    }
}
BEGIN: AUTOMATED TEST

TEST: 1 | Investor ID: jones | First: John
       | Password: jones    | Last: Jones
       | push Login         | Account: 1000
       |                   | Phone: 510-111-1111

Investor Login: jones

TEST: 2 | push Pending Orders | Pending Orders: 2
       | push Transactions  | Transactions: 4
       | push Portfolio     | Cash: 1000.00
       |                   | Net Worth: 4040.00

ORDERS:
1000 Fri May 12 08:30:08 PDT 2006 1000 INTC BUY 19.050
1005 Fri May 12 08:30:08 PDT 2006 1000 IBM SELL 86.05

TRANSACTIONS:
1000 Fri May 12 08:30:08 PDT 2006 1000 CASH DEPOSIT 1000.01
1003 Fri May 12 08:30:08 PDT 2006 1000 IBM BUY 80.05
1004 Fri May 12 08:30:08 PDT 2006 1000 IBM SELL 82.05
1005 Fri May 12 08:30:08 PDT 2006 1000 CSC0 BUY 20.0100

PORTFOLIO:
Stock   Shares Market
------- ------ --------
IBM     10    840.0
CSC0    100   2200.0

$3040.0
Cash $1000.0
Net Worth $4040.0

ETC.

END: AUTOMATED TEST
import java.util.Vector; // Java container class
import java.util.Hashtable; // Java lookup mechanism based on key

public class Investor {
    public String INVESTOR_ID; // primary key
    public String PASSWORD; // public visibility: same as a record struct
    public String FIRST;
    public String LAST;
    public String PHONE;
    public Vector accounts = new Vector(); // list of all account children: 1 to many
    // children add themselves to this container
    public static Hashtable investors = new Hashtable(); // find investor keyed on INVESTOR_ID
    // static: just one hashtable

    public Investor(String INVESTOR_ID, String PASSWORD, String FIRST, String LAST, String PHONE){
        this.INVESTOR_ID = INVESTOR_ID; this.PASSWORD = PASSWORD; this.FIRST = FIRST;
        this.LAST = LAST; this.PHONE = PHONE;
        investors.put(INVESTOR_ID, this); // put itself into the hashtable using primary key
    }
    public String toString() {
        return (INVESTOR_ID + " " + LAST + " " + FIRST + " " + PHONE);
    }
}

public class Account {
    public int ACCOUNT_NO; // primary key
    public Investor investor; // reference to Investor parent: many to 1
    public double CASH = 0; // public visibility: same as record struct
    public Vector holdings = new Vector(); // list of all holding children : 1 to many
    public Vector orders = new Vector(); // list of all order children : 1 to many
    public Vector transactions = new Vector(); // list of all transaction children: 1 to many
    private static int base_acct = 1000; // static var to auto generate account numbers

    public Account(Investor investor, double CASH) {
        this.investor = investor;
        ACCOUNT_NO = base_acct++;
        this.CASH = CASH;
        investor.accounts.addElement(this); // add itself to the parent’s list of children
    }
    public String toString() {
        return (ACCOUNT_NO + " " + investor.FIRST + " " + investor.LAST + " " + CASH);
    }
}
public class Holding {
    public Account account;  // part of primary key, but not a string
    public Stock stock;       // part of primary key, reference to parent
    public int QTY;

    public Holding(Account account, Stock stock, int QTY) {
        this.account = account; this.stock = stock; this.QTY = QTY;
        account.holdings.addElement(this);  // add itself to the parent’s list of children
    }
    public String toString() {
        return (account.ACCOUNT_NO + " " + stock.SYM + " " + QTY);  // access primary key string
    }
    public void remove() {
        account.holdings.removeElement(this); // remove itself when QTY is sold off
    }
}

import java.util.Vector;        // Java container class
import java.util.Date;           // auto generate a date, convert to String

public class Order {
    public int ORDER_NO;          // primary key
    public String DATE;           // date as a String
    public Account account;       // reference to parent
    public Stock stock;           // reference to parent
    public String TYPE;           // public visibility: same as a record struct
    public double PRICE;
    public int QTY;
    public static int base_order = 1000; // static var to auto generate order numbers

    public Order(Account account, Stock stock, String TYPE, double PRICE, int QTY) {
        this.account = account; this.stock = stock; this.TYPE = TYPE; this.PRICE = PRICE;
        this.QTY = QTY;
        ORDER_NO = base_order++;    // auto generate
        Date date = new Date();     // get date/time right now
        DATE = date.toString();     // store as a String
        account.orders.addElement(this);  // add itself to the parent’s list of children
        if (TYPE.equals("SELL"))
            stock.sellheap.put(this); // add itself to it’s parent’s heap of orders
        else
            stock.buyheap.put(this);
    }
    public String toString() {
        return (ORDER_NO + " " + DATE + " " + account.ACCOUNT_NO + " " + stock.SYM + " " + TYPE + 
            " " + PRICE + " " + QTY);
    }
    public void remove() {
        account.orders.removeElement(this); // remove itself when the QTY is consumed
    }
}
import java.util.Hashtable; // Java lookup mechanism based on key

public class Stock {
    public String SYM;       // primary key
    public double PRICE;     // public visibility: same as record struct
    public Heap buyheap = new Heap(true);
    public Heap sellheap = new Heap(false);
    public static Hashtable stocks = new Hashtable(); // find stock keyed on SYM
               // static: just one hashtable

    public Stock(String SYM, double PRICE) {
        this.SYM = SYM; this.PRICE = PRICE;
        stocks.put(SYM,this);  // put itself into the hashtable using primary key
    }
    public String toString() {
        return (SYM + " " + PRICE);
    }
}

import java.util.Vector;       // Java container class
import java.util.Date;         // auto generate a date, convert to String

public class Transaction {
    public int TRANS_NO;   // primary key
    public String DATE;    // date as a String
    public Account account;   // reference to parent
    public Stock stock;    // reference to parent
    public String TYPE;    // public visibility: same as record struct
    public double PRICE;
    public int QTY;
    public static int base_trans = 1000; // static var to auto generate tranaction numbers

    public Transaction(Account account, Stock stock, String TYPE, double PRICE, int QTY) {
        this.account = account; this.stock = stock; this.TYPE = TYPE; this.PRICE = PRICE;
        this.QTY = QTY;
        TRANS_NO = base_trans++;  // auto generate
        Date date = new Date();  // get date/time right now
        DATE = date.toString();  // store as a String
        account.transactions.addElement(this); // add itself to it's parent's list of children
    }
    public String toString() {
        return (TRANS_NO + " " + DATE + " " + account.ACCOUNT_NO + " " + stock.SYM + " " + TYPE + " " + PRICE + " " + QTY);
    }
}
Boot Sequence

main() → Stockmain

1:create()
2:init()

:StockApplet → 2.1:create()

2.2:create()

:AccountManagement :ReportGenerator :TradeExecution

2.1.1:create() → :Investor

:Account :Stock :Holding

:Transaction :Order
import java.awt.*; // Java GUI classes
import java.applet.*;
import java.awt.event.*;

public class Stockmain { // to run: java Stockmain
    public static void main(String args[]) {
        Applet applet = new StockApplet(); // create the applet and
        Frame frame = new StockFrame(applet); // put it into a frame
    }
}

class StockFrame extends Frame implements ActionListener {
    public StockFrame(Applet applet) {
        super("Bestway Stock Trader");
        MenuBar menubar = new MenuBar(); // make a menu with Quit
        Menu file = new Menu("File",true);
        menubar.add(file);
        file.add("Quit");
        setMenuBar(menubar);
        file.addActionListener(this);
        add("Center",applet);
        setSize(620,680);
        applet.init(); // start applet
        this.show();
    }
    public void actionPerformed (ActionEvent evt) {
        String arg = evt.getActionCommand();
        if (arg.equals("Quit")) { // if Quit, exit
            System.exit(0);
        }
    }
}
Boot Sequence: StockInit

public class StockInit {
    public StockInit() { // called by StockApplet
        // create investors
        Investor i1 = new Investor("jones","jones","John","Jones","510-111-1111");
        Investor i2 = new Investor("smith","smith","John","Smith","510-222-2222");
        Investor i3 = new Investor("adams","adams","John","Adams","510-333-3333");

        // create accounts for the investors above
        Account a1 = new Account(i1, 1000.0);
        Account a2 = new Account(i2, 1000.0);
        Account a3 = new Account(i3, 1000.0);

        // cash is used as the "stock" transaction for DEPOSIT/WITHDRAW
        Stock cash = new Stock("CASH",0.00);

        // create stocks with market price
        Stock s1 = new Stock("IBM",84.00);
        Stock s2 = new Stock("CSCO",22.00);
        Stock s3 = new Stock("INTC",20.00);

        // give some holdings to the accounts
        new Holding(a1,s1,10);
        new Holding(a1,s2,100);
        new Holding(a2,s3,100);
        new Holding(a3,s2,100);

        // generate some existing transactions
        new Transaction(a1,cash,"DEPOSIT",1000.00,1);
        new Transaction(a2,cash,"DEPOSIT",1000.00,1);
        new Transaction(a3,cash,"DEPOSIT",1000.00,1);
        new Transaction(a1,s1,"BUY",80.00,5);
        new Transaction(a1,s1,"SOLD",82.00,5);
        new Transaction(a1,s2,"BUY",20.00,100);
        new Transaction(a2,s3,"BUY",18.00,100);
        new Transaction(a3,s2,"BUY",19.00,100);

        // generate some outstanding pending orders
        new Order(a1,s3,"BUY",19.00,50);
        new Order(a2,s3,"SELL",23.00,50);
        new Order(a3,s3,"SELL",21.00,75);
        new Order(a3,s1,"BUY",82.00,5);
        new Order(a3,s1,"BUY",82.50,5);
        new Order(a1,s1,"SELL",86.00,5);
    }
}
import java.awt.*;       // Java GUI classes
import java.applet.*;
import java.awt.event.*;
// Presentation Layer
public class StockApplet extends Applet implements ActionListener {
    // buttons
    private Button newInvestor, newacct, login, deposit, withdraw,
        portfolio, orders, trans, quote, buy, sell, test, chgacct;
    // labels for fields
    private Label acctLabel, firstLabel, lastLabel, idLabel, phoneLabel,
        passLabel, cashLabel, stockLabel, sharesLabel, priceLabel;
    // fields
    private TextField acctField, firstField, lastField, idField, phoneField,
        passField, cashField, stockField, sharesField, priceField;
    // print to textField
    private TextArea textField;
    // references to Domain Logic
    private AccountManagement management;
    private ReportGenerator report;
    private TradeExecution trade;
    public void init() {
        super.init();
        setLayout(null);
        // create a label and field, place on the screen
        idLabel = new Label("Investor ID:");
        idLabel.setBounds(20,10,70,30);
        add(idLabel);
        idField = new TextField(130);
        idField.setBounds(100,10,80,30);
        add(idField);
        // ETC. for other labels and fields
        // create a TextField for printing
        textField = new TextField();
        textField.setBounds(20,270,580,350);
        Font font = new Font("Courier",Font.PLAIN,10);
        textField.setFont(font);
        add(textField);
        // create a button, place on the screen, applet "listens" for button push
        login = new Button("Login");
        login.setBounds(20,210,60,25);
        add(login);
        login.addActionListener(this);
        // ETC. for other buttons
        // pre-populate some Domain Objects
        new StockInit();
        // create the Domain Logic classes
        // hand over textField so class can print to it (in this case, same TextField)
        management = new AccountManagement(textField);
        report = new ReportGenerator(textField);
        trade = new TradeExecution(textField);
    }
Presentation Layer: StockApplet

// called by actionPerformed whenever a button is pushed
// also called by StockTest to do Automated Test
public void doAction(String arg) {

    // depending on the button, call the appropriate method in Domain Logic
    if (arg.equals("Login"))
        management.doLogin(idField.getText(), passField.getText(), acctField, firstField, lastField, phoneField);
    else if (arg.equals("Chg Acct"))
        management.doChgAcct(acctField.getText());
    if (arg.equals("New Investor"))
        management.doNewInvestor(idField.getText(), passField.getText(), firstField.getText(),
                                lastField.getText(), phoneField.getText(), acctField);
    else if (arg.equals("New Account"))
        management.doNewAccount(acctField);
    else if (arg.equals("Deposit"))
        management.doCash("DEPOSIT", cashField.getText());
    else if (arg.equals("Withdraw"))
        management.doCash("WITHDRAW", cashField.getText());
    else if (arg.equals("Portfolio"))
        report.doPortfolio();
    else if (arg.equals("Pending Orders"))
        report.showOrders();
    else if (arg.equals("Transactions"))
        report.showTrans();
    else if (arg.equals("Quote"))
        report.showQuote(stockField.getText());
    else if (arg.equals("Buy"))
        trade.doOrder("BUY", stockField.getText(), sharesField.getText(), priceField.getText());
    else if (arg.equals("Sell"))
        trade.doOrder("SELL", stockField.getText(), sharesField.getText(), priceField.getText());
    else if (arg.equals("Test"))
        new StockTest (this, textarea1, acctField, firstField, lastField, idField, phoneField, passField, cashField, stockField, sharesField, priceField);
}

// called by the window manager whenever a button is pushed
public void actionPerformed(ActionEvent event) {
    String arg = event.getActionCommand();
    doAction(arg);
}
} // class StockApplet
import java.awt.*; // Java GUI classes
import java.applet.*;

public abstract class DomainLogic { // super class, abstract => must be subclassed

    // protected variables so subclasses can access
    protected TextArea textarea;       // print to textarea
    protected static Account account = null; // current account, static to share among subclasses
    protected static Investor investor = null; // current investor, static also

    public DomainLogic(TextArea textarea) {
        this.textarea = textarea;
    }

    // protected methods so subclasses can access
    protected boolean chkLogin() { // reports and trading can only occur after a login
        boolean OK = false;
        if (investor==null)
            textarea.append("Must login first.\n");
        else
            if (account==null)
                textarea.append("Must have an account.\n");
            else
                OK = true;
        return OK;
    }

    // helper methods for text processing
    protected double toDouble(String s) {
        Double d = new Double(0.0);
        try {
            d = Double.valueOf(s);
        } catch(Exception e) {return 0.0;}
        return d.doubleValue();
    }

    protected int toInt(String s) {
        Integer i = new Integer(0);
        try {
            i = Integer.valueOf(s);
        } catch(Exception e) {return 0;}
        return i.intValue();
    }

    protected String pad(String s, int width, String padChar, boolean padLeft) {
        for (int i=s.length()+1; i<=width; i++)
            if (padLeft)
                s = padChar + s;
        else
            s = s + padChar;
        return s;
    }
}
Domain Logic: AccountManagement

import java.awt.*;
import java.applet.*;
public class AccountManagement extends DomainLogic { // inherits from DomainLogic
    public AccountManagement(TextArea textarea1) {
        super(textarea1);
    }
    public void doChgAcct(String acct) { // change to a different account
        if (acct==null)
            textarea1.append("Must login first.\n");
        else {
            int acctno = toInt(acct);
            account = null;
            // iterate through the list of this investor's accounts
            for (int i=0; i<investor.accounts.size(); i++) {
                Account acct1 = (Account)(investor.accounts.elementAt(i));
                if (acct1.ACCOUNT_NO == acctno) {
                    account = acct1;
                    textarea1.append("Change Account: " + acctno +"\n");
                    return;
                }
            }
            textarea1.append("No Account #: " + acctno +"\n");
        }
    }
    public void doNewAccount(TextField acctField) { // generate a new account
        if (acct==null)
            textarea1.append("Must login first.\n");
        else {
            account = new Account(investor,0.0);
            acctField.setText(account.ACCOUNT_NO+" ");
            textarea1.append("New Account #: " + account.ACCOUNT_NO+"\n");
        }
    }
    public void doNewInvestor(String id, String password, String first, // generate new investor
        String last, String phone, TextField acctField) {
        investor = null;
        if (id.equals("") || password.equals("") || first.equals("") ||
            last.equals("") || phone.equals(""))
            textarea1.append("Must provide Investor ID, Password, First, Last, Phone.\n");
        else
            if ((Investor)Investor.investors.get(id) != null)
                textarea1.append("Investor ID already exists: "+id+"\n");
            else {
                investor = new Investor(id, password, first, last, phone);
                textarea1.append("New Investor Complete\n");
                account = new Account(investor,0.0);
                textarea1.append("New account number: "+account.ACCOUNT_NO+"\n");
                acctField.setText(account.ACCOUNT_NO+" ");
            }
    }
}
public void doLogin(String id, String passGiven, TextField acctField, // validate login
    TextField firstField, TextField lastField, TextField phoneField) {
    investor = null;
    account = null;
    if ((id.equals("")) || (passGiven.equals("")))
        textArea1.append("Must enter Investor ID and Password.\n");
    else {
        investor = (Investor) Investor.investors.get(id);
        if (investor == null)
            textArea1.append("No such Investor ID: "+id+"\n");
        else
            if (!investor.PASSWORD.equals(passGiven))
                textArea1.append("Incorrect password.\n");
            else {
                if (investor.accounts.size() > 0) {
                    account = (Account) investor.accounts.elementAt(0);
                    acctField.setText(account.ACCOUNT_NO+"");
                }
                else
textArea1.append("No open accounts for: "+id+"\n");
                firstField.setText(investor.FIRST);
                lastField.setText(investor.LAST);
                phoneField.setText(investor.PHONE);
                textArea1.append("Investor Login: "+id+"\n");
            }
        }
}

public void doCash(String type, String amount) {  // DEPOSIT or WITHDRAW cash
    if (chkLogin())
        if (amount.equals(""))
            textArea1.append("Empty Cash field.\n");
        else {
            double dollars = toDouble(amount);
            if (type.equals("DEPOSIT"))
                account.CASH = account.CASH + dollars;
            else
                account.CASH = account.CASH - dollars;
                Stock cash = (Stock) Stock.stocks.get("CASH");
                new Transaction(account.cash, type, dollars, 1);
                textArea1.append(type+ " completed.\n");
        }
    } // AccountManagement class
import java.awt.*;
import java.applet.*;

// Domain Logic
public class ReportGenerator extends DomainLogic { // inherits from DomainLogic
    public ReportGenerator(TextArea textArea1) {
        super(textArea1);
    }
    public void doPortfolio() { // display portfolio
        if (chkLogin()) {
            textArea1.append("PORTFOLIO:\n");
            double sum = 0.0;
            textArea1.append("Stock Shares Market\n");
            textArea1.append("----- ------ --------\n");
            for (int i=0; i<account.holdings.size(); i++) {
                Holding holding = (Holding) account.holdings.elementAt(i);
                String stockStr = pad(holding.stock.SYM,7," ",false);
                String qty = pad(String.valueOf(holding.QTY),7," ",true);
                double market = holding.QTY * holding.stock.PRICE;
                String marketStr = String.valueOf(market);
                marketStr = pad(marketStr,12," ",true);
                textArea1.append(stockStr+qty+marketStr+"\n");
                sum += market;
            }
            String sumStr = Double.toString(sum);
            sumStr = pad(sumStr,9," ",true);
            textArea1.append(" 
");
            textArea1.append("$\n"");
            String cashStr= String.valueOf(account.CASH);
            cashStr = pad(cashStr,9," ",true);
            textArea1.append("Cash $\n");
            textArea1.append("\n");
            sum = sum + account.CASH;
            sumStr = Double.toString(sum);
            sumStr = pad(sumStr,9," ",true);
            textArea1.append("Net Worth $\n");
            textArea1.append("\n");
        }
    }
    public void showOrders() { // display orders
        if (chkLogin()) {
            textArea1.append("ORDERS:\n");
            for (int i=0; i<account.orders.size(); i++) {
                Order order = (Order)account.orders.elementAt(i);
                textArea1.append(order.toString()+"\n");
            }
            textArea1.append("\n");
        }
    }
}
public void showTrans() { // display transactions
    if (chkLogin()) {
        textarea1.append("TRANSACTIONS:\n");
        for (int i=0; i<account.transactions.size(); i++) {
            Transaction trans = (Transaction)account.transactions.elementAt(i);
            textarea1.append(trans.toString()+"\n");
        }
        textarea1.append("\n");
    }
}

private String lastPrice(Stock stock) { // last traded price is stored in stock class
    return "$"+stock.PRICE;
}

private String maxPrice(Stock stock) { // maximum offered price is root of buyheap
    String price = "NONE";
    if (stock.buyheap.size() > 0) {
        Order order = stock.buyheap.testRoot();
        price = "$"+order.PRICE;
    }
    return price;
}

private String minPrice(Stock stock) { // minimum offered price is root of sellheap
    String price = "NONE";
    if (stock.sellheap.size() > 0) {
        Order order = stock.sellheap.testRoot();
        price = "$"+order.PRICE;
    }
    return price;
}

public void showQuote(String sym) { // lookup the stock in the hashtable, and display prices
    sym = sym.toUpperCase();
    if (sym.equals(""))
        textarea1.append("Empty stock field.\n");
    else {
        Stock stock = (Stock) Stock.stocks.get(sym);
        if (stock == null)
            textarea1.append("Invalid stock symbol: "+ sym +"\n");
        else
            textarea1.append("Quote: "+ sym +" Minimum SELL: "+minPrice(stock)+
              " Maximum BUY: "+maxPrice(stock)+
              " Last Sale: "+lastPrice(stock)+"\n\n");
    }
} // class ReportGenerator
public class TradeExecution extends DomainLogic {
    // inherits from DomainLogic
    public TradeExecution(TextArea textArea1) {
        super(textArea1);
    }
    void chgCash(Account account, double amount) {
        // adjust cash after BUY/SELL
        account.CASH = account.CASH + amount;
    }
    private Holding getHolding(Account account, Stock stock) {
        // find the holding record
        Holding holding = null;
        for (int i = 0; i < account.holdings.size(); i++) {
            Holding holding1 = (Holding) account.holdings.elementAt(i);
            if (holding1.stock == stock) {
                holding = holding1;
                break;
            }
        }
        return holding;
    }
    // adjust QTY of holding, or remove
    private void chgHolding(Account account, Stock stock, int qty) {
        Holding holding = getHolding(account, stock);
        if (holding == null)
            new Holding(account, stock, qty);
        else {
            holding.QTY += qty;
            if (holding.QTY <= 0)
                holding.remove();
        }
    }
    private void chkOrder(Order order, int qty) {
        // adjust QTY of order, or remove
        if (qty <= 0)
            order.remove();
        else
            order.QTY = qty;
    }
    // make sure a seller actually has the stock in the correct amount
    private boolean validOrder(Account account, Stock stock, String type,
                              String price, String shares) {
        boolean OK = true;
        if (type.equals("SELL")) {
            OK = false;
            Holding holding = getHolding(account, stock);
            if (holding == null)
                textArea1.append("Account does not have any shares of " + stock.SYM + "\n");
            else
                if (toInt(shares) > holding.QTY)
                    textArea1.append("Account only has " + holding.QTY + " shares of "+ stock.SYM + "\n");
                else
                    OK = true;
        }
        return OK;
    }
}
// new buyer has to pay at least as much as pending seller wants, vice versa for a new seller
private boolean chkPrices(String type, double p1, double p2) {
    boolean OK = false;
    if (type.equals("BUY")) {
        if (p1 >= p2)
            OK = true;
    }
    else // SELL
    if (p1 <= p2)
        OK = true;
    return OK;
}

// process a sale for buyer/seller updating cash, holdings, transaction.
// Also, this sale price represents the stocks new "value", used in all portfolios
private void doSale(Account sellAccount, Account buyAccount, Stock stock, int qty, double price) {
    chgCash(sellAccount, +qty*price);
    chgCash(buyAccount, -qty*price);
    chgHolding(sellAccount, stock, -qty);
    chgHolding(buyAccount, stock, +qty);
    new Transaction(sellAccount, stock, "SELL", price, qty);
    new Transaction(buyAccount, stock, "BUY", price, qty);
    stock.PRICE = price;
}

// StockApplet calls doOrder when a BUY/SELL button is pushed
// Validate the input fields and lookup the stock in the hashtable
public void doOrder(String type, String sym, String shares, String price) {
    sym = sym.toUpperCase();
    if (chkLogin())
        if (((sym.equals("")) || (shares.equals("")) || (price.equals("")))
            textArea1.append("Must enter Stock, Shares, and Price.\n");
        else {
            Stock stock = (Stock)Stock.stocks.get(sym);
            if (stock == null)
                textArea1.append("Stock does not exist: " + sym);
            else
                if (validOrder(account, stock, type, price, shares))
                    scanOrders(account, stock, type, (int)shares, (double)price);
        }
}
// main algorithm (see pseudocode) for matching buys with sells
private void scanOrders(Account account, Stock stock, String type, int qty1, double price) {
    int qty2, qty;
    Heap heap = stock.buyheap;
    if (type.equals("BUY"))
        heap = stock.sellheap;
    while (true) {
        if (qty1 <= 0) break;
        Order order = (Order) heap.testRoot();
        if (order == null) break;
        if (!chkPrices(type, price, order.PRICE)) break;

        order = heap.getRoot();
        qty2 = order.QTY;
        Account acct2 = order.account;
        if (qty1 <= qty2)
            qty = qty1;
        else
            qty = qty2;
        Account seller;
        Account buyer;
        if (type.equals("BUY")) {
            buyer = account;
            seller = acct2;
        } else {
            seller = account;
            buyer = acct2;
        }
        qty1 -= qty;
        qty2 -= qty;
        doSale(seller, buyer, stock, qty, price);
        chkOrder(order, qty2);
        textArea1.append("Sale completed: " + qty + " shares of " + stock.SYM + ", at $" + price + " with account: " + acct2.ACCOUNT_NO + 
"n");
    } // while
    if (qty1 <= 0)
        textArea1.append(type + " completed.\n");
    else {
        new Order(account, stock, type, price, qty1);
        textArea1.append("Current outstanding order: " + qty + " shares of " + stock.SYM + ", at $" + price + "\n");
    }
}
} // class TradeExecution
public class Heap {
    private boolean maxheap; // true => maxheap, false => minheap
    private static final int MAXSIZE = 100;
    private Order[] array = new Order[MAXSIZE];
    private int size = 0;
    private int maxsize = MAXSIZE - 1;

    public Heap(boolean maxheap) {
        this.maxheap = maxheap;
    }
    public int size() {
        return size;
    }
    public void put(Order order) { // add new element at end, restore heap
        int i;
        if (size < maxsize) {
            i = ++size;
            if (maxheap)
                while ((i > 1) && (array[i/2].PRICE < order.PRICE)) {
                    array[i] = array[i/2];
                    i /= 2;
                } // i becomes the parent
            else
                while ((i > 1) && (array[i/2].PRICE > order.PRICE)) {
                    array[i] = array[i/2];
                    i /= 2;
                }
            array[i] = order;
        }
    }
    public Order testRoot() { // retrieve root
        if (size == 0)
            return null;
        else
            return (array[1]);
    }
}
public Order getRoot() {  // retrieve root, remove root, restore heap
    if (size == 0)
        return null;
    else {
        Order root = array[1];
        Order last_element = array[size--];
        int i = 1;
        while (2*i <= size) {  // 2*i is left child
            int child = 2*i;
            if (child != size)  // find smaller child
                if (maxheap) {
                    if (array[child+1].PRICE > array[child].PRICE)  // child+1 is right child
                        child++;
                }
                else
                    if (array[child+1].PRICE < array[child].PRICE)
                        child++;
            if (last_element.PRICE < array[child].PRICE)  // percolate one level
                array[i] = array[child];
            else
                break;
        } 
        else 
            if (last_element.PRICE > array[child].PRICE)
                array[i] = array[child];
            else
                break;
            i = child;
        }
        array[i] = last_element;
        return(root);
    }
}