

StockMaster: A Java-Based Stock Portfolio Manager

Bhagyashri Bhagvat, Mark J. Boyd, James Freeby and Edward Parrish

Jack Baskin School of Engineering
University of California
Santa Cruz, CA 95064
{bhagvat, mjboyd, jimfree, eparrish}@cse.ucsc.edu

Abstract

This paper introduces StockMaster, a stock portfolio manager which uses Java applets and applications to provide portfolio information for users through a web interface. StockMaster uses a three-tier architecture. The tiers include an applet which interfaces with a server-side application, which in turn accesses a database/quote server. Use of design patterns, extensible object-oriented programming methods, and database design are discussed in the context of designing the project.

1 Introduction

Object-oriented programming (OOP) for web based electronic commerce applications is an increasingly important area of computer science [2, 4, 5]. Java (and its OOP structure) is widely used in *e-business*, business conducted through the internet. Java's popularity in this area is due to its extensive libraries, rapid Graphic User Interface (GUI) development, easy network interfaces, platform independence, and support for internet browser environments.

This paper describes StockMaster¹, a web-

based object-oriented system design, written mostly in Java. StockMaster provides tracking and summary statistics for an individual's stock portfolio. This three week e-business project was a product of the efforts of a team of four engineers. OOP and high-level modeling enabled the group to successfully partition and interface the work in the very short time period, a critical feature for most e-business applications.

Section 2 reviews necessary definitions, and Section 3 describes how the system relates to e-business. Section 4 presents an outline of StockMaster and its significant components, and Section 5 presents an analysis of the system design. Section 6 presents test results, Section 7 discusses future extensions, and Section 8 concludes the paper.

2 Definitions

2.1 OOP

Object-Oriented Programming (OOP) is organized around objects rather than actions, data rather than logic. OOP takes the view that our focus should be on the objects we want to manipulate rather than the methods

¹Resources for the project are found at <http://www.cse.ucsc.edu/~eparrish/research/portfolio/portfolio.html>

used to manipulate them. Examples of objects range from human beings (described by name, address, and so forth) to buildings and floors (whose properties can be described and managed) down to the little widgets on your computer desktop (such as buttons and scroll bars). [1]

Classes describe a generalized class of objects, defining the data contained and any logical sequences that can manipulate the data. These manipulations are called methods [8].

2.2 UML

Universal Modeling Language (UML) [6] is a product of the process of object-oriented design (OOD). The graphical object-relationship models produced by UML clearly show the relationships between objects, rather than data or functions.

2.3 Java

Java is an object-oriented language without all of the complexity of C++. Features like strong typing, excellent compiler error checking, platform independence, and automatic garbage collection support rapid application development. One disadvantage of Java, in general, is an order of magnitude slower code execution.

2.4 Applet

A small application, usually downloaded from a web page link, which executes on the client computer.

3 E-business Applicability

Our application demonstrates the feasibility of an online portfolio. This portfolio can be supported by two types of revenue. The first type is advertisements, currently pervasive on the

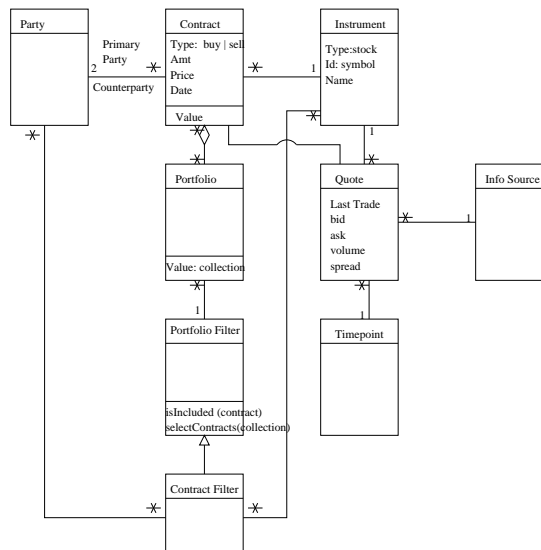


Figure 1: Portfolio Design Pattern

web. The second possible revenue source is interaction with brokerage and other trading firms. By adding secure transaction capabilities to our model, we can allow our on-line customers to buy and sell securities at many different brokerages. These brokerages can pay us a small finder's fee for bringing them this business. Thus we have two possible methods of conducting e-business on the Internet.

4 Portfolio Manager Design

The system model uses the *Portfolio* design pattern [2]. This pattern has four parts for the pattern: portfolios, contracts, quotes, and scenarios. The scenario object, although extensible from our design, was not implemented in the demonstration version.

Using a previously outlined design pattern significantly reduces design time. Designing the system from an object-oriented view-

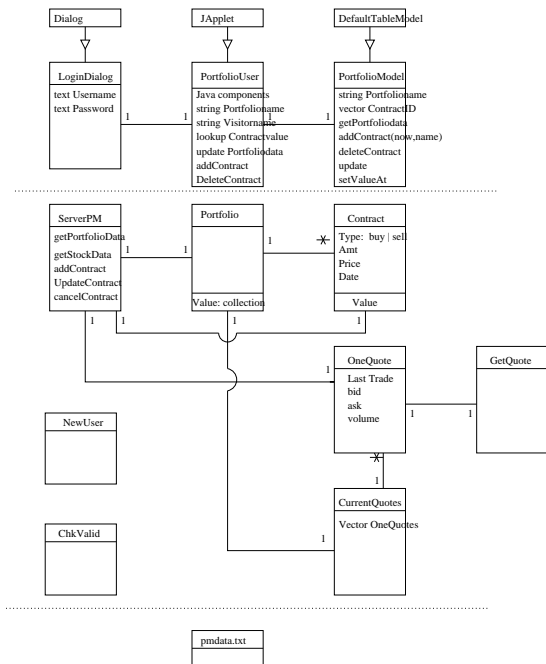


Figure 2: StockMaster Three Tier Design

point allows an easy partitioning of the project among team members. These techniques together support very rapid project development. Figure 1 shows the Portfolio design pattern.

The architecture of the portfolio system is a three-tier design, originally proposed by Tsichiritzis and Klug [7]. The three tiers are: (1) the user interface (UI) communicating with (2) a server program which communicates with (3) the databases and data sources. The advantage of this design is that it frees the UI to concentrate on the user rather than the physical location and structure of the data. Thus we can later change applications or database structure without breaking other parts of the design or existing applications. Figure 2 shows the Portfolio design pattern.

The UI tier contains two primary applications: an identification and security GUI (sub-

section 4.1) and a portfolio GUI (subsection 4.2). The portfolio GUI is implemented as a series of web pages, one of which contains a Java applet. The Java applet handles the user interface for user identification and portfolio display and management. The current portfolio contents are obtained from the server. Any changes made by the user are stored on the server. Other web pages contain help information and a new user registration form.

The design of the server tier (subsection 4.3) was adapted from a design pattern by Martin Fowler [2]. This tier presents a simplified interface to the UI tier using the Facade design pattern [3].

Using this pattern as the basis of our design provided us with a flexible and extensible design that has been proven in a trading system maintained by a bank. We kept the overall architecture of the design but only implemented those parts required for our project.

The database tier contains two sections: a contract database (subsection 4.4) and a network quote server link to existing databases at Yahoo (subsection 4.5). The contract database contains the user's portfolio data. Current trading data are obtained from Yahoo. Thus we are able to keep the user's portfolio data secure on our system and get current trading values.

4.1 Identification and Security

The website displays a web page that prompts a new user to register and if already registered, the user is required to enter the username and password to log on and view the portfolio. To register, the new user needs to provide username and password. After the user supplies the information, the applet calls a Perl script which in turn calls a Java program for adding the new user to the registered users' list. If the username and password are unique, the user is added to the registered users' database. If the

username is not unique, the user is prompted to enter another username, until a unique username is entered. The user can then use the login dialogue. The username and password entered in the dialogue box are checked for validity by the Java program which again is invoked by a Perl script. If valid, the applet loads the user's portfolio.

4.2 GUI

The Graphic User Interface (GUI) uses the login and password protection gatekeeper, and then presents a table of securities and valuations for the portfolio.

The GUI (Graphical User Interface) for the portfolio manager is implemented in the PortfolioUser class. This class accepts data from other system components and encapsulates the end user interface. The PortfolioUser class extends JApplet, thus inheriting all of the functionality of the JApplet class. PortfolioUser also implements ActionListener, which PortfolioUser uses to respond to GUI events to update the screen. In this manner, the GUI uses the advantages of inheritance and encapsulation espoused in OOP and OOD.

The PortfolioUser class uses Java's Swing [4] components and AWT to build the GUI. For example, buttons are created by instantiating the JButton class. The JButton class was developed with the Model-View-Controller design pattern in mind and as a result the PortfolioUser class easily implements this design pattern [4]. The Model is created when the button is declared. The View is instantiated when the applet begins and creates a window with the button. The View is dependent on how the Model was declared. The Controller responds to events. For example, if a user presses the button, the Controller calls a Model method, which updates the View.

By using the Java Swing components for this applet, the end user is required to install

a Java Plugin on his system. Although this causes some initial work by the end user, the GUI has a professional look and feel. Java Swing allows the GUI to show contract data in a grid layout, with rows and columns of data. The rows represent individual contracts, while the columns identify the data. For example, there is a column for "shares", which indicates the number of shares in the contract. Section 5.2 discusses this approach further.

Object	StockMaster	Lower Bound
Portfolio	$O(u)$	$\Theta(1)$
Contract	$O(cu)$	$\Theta(s)$
Quote	$O(su)$	$\Theta(s)$

Table 1: Scalability Analysis for Various Components (c=contracts per user, s=stocks per user, u=users)

4.3 Server-side Application

The server-side application acts to fetch the quote data and the database data for the portfolio of a given user. Such a feature is needed to access information securely, since a server-side database was used and some quote caching at the server speeds up performance. Table 1 shows, however, that the server must still divide tasks among the $O(u)$ users currently using the system.

4.4 Contracts Database

The contracts database maintains data on each users portfolios and contracts. Although currently implemented as a flat file, the object can easily be converted to a relational database without affecting any of the object interfaces to the other parts of the design.

4.5 Quote Server

The quote object queries a link (currently Yahoo!) for quotes on a security. A local cache of results ensures a quote is fetched for each stock, not each contract. However, if a new user logs in, the quote server is queried separately to ensure the latest current data is retrieved. In Table 1 this is shown by the $O(su)$ performance.

5 Design Decisions and Tradeoffs

Several important design decisions affected the complexity of the database design.

5.1 Integration

Using the Portfolio design pattern [2] and a clear UML design enabled a smooth integration of multiple portfolio manager components. The project was divided into four portions, with each team member working on an individual component of the system: User Login; GUI; Server Portfolio Data Retrieval; and Server Price Retrieval. Each part was designed to return data in a specified format. In this manner, the team members each worked on their respective components, knowing the specifications for returning data. For example, the user login portion returned a message indicating whether the login was correct or not. The GUI expected this message in a predefined format. As a result of the clear specifications, the integration of the four components went smoothly. For example, integrating the GUI with the Server Portfolio Data Retrieval took approximately 10 minutes. Table 2 clearly shows this relatively quick integration time when compared to the design, coding and testing of the individual components.

5.2 Java Plug-In

The use of the Java 2 plug-in was dictated by our decision to use a JTable in the Graphical User Interface (GUI). The JTable class provides a spreadsheet-like interface, which makes the presentation and entry of portfolio data familiar to the user. JTable is only available in the Swing version of the GUI and not the AWT version. Since most browsers do not support Swing at this time, we were able to obtain support by using the Java 2 plug-in.

The Java 2 plug-in does require download and setup by the user before they can use our application. This is acceptable for this project; if we were to implement this product commercially, we would implement a more automatic and shorter download of missing parts of the Swing set and thereby eliminate the need for the plug-in.

5.3 Cookie vs. Database

We used a server-side database application instead of a user-side *cookie* for storing contract and portfolio data. Cookies are small data files which are stored on the user's local filesystem, instead of using the server's file resources. Using cookies instead of server disk space has several advantages. Since the server requires no storage space, the system is scalable (indifferent to the number of users) and the server doesn't allocate resources to each user. Using cookies alleviates security concerns (since the private data is kept and accessed locally, never transmitted). Finally, there is no file contention or organization required (which is an issue when a server-side database is maintained). Using a cookie would provide $O(c)$ instead of $O(cu)$ performance in Table 1, since only the data for that user would need to be searched or stored. The major disadvantage of cookies, and the reason they were not used, is their *physical filesystem dependence*. Since the

cookie is on a real physical filesystem, a user may only access portfolio data from a computer with access to that file. A user wishing to browse a portfolio from work would need to copy the cookie from a home filesystem to the work filesystem, or laboriously re-enter all the data.

5.4 Contracts Database

We used a contract-based model. Strictly speaking, the value of a portfolio has no relation to the purchase price of any security. Only the last trading price is important. Additionally, the current number of shares is more important than when and in what size blocks the shares were purchased. Using the contract based model adds significant complexity and size to the database. A single security, such as Intel stock (known by its symbol INTC) may have many contracts represented as database records. Instead of a single entry, such as [INTC, 100 shares], there may be many records for the contracts executed for INTC. To determine the total number of shares owned, the sum of all INTC records must be calculated. Additionally, users must enter each contract, instead of simply reducing or increasing the number of shares held. Using a non-contract approach makes the optimal performance (when using cookies) in Table 1 $O(s)$ instead of $O(cu)$. The major advantage of the contract based design is extensibility. A feature to calculate taxes for the year needs the additional contract information (purchase price and date, number of shares) to generate a tax report. Any number of other filters are also easy supported, especially if a relational database is used.

6 Test Results and Development Time

Our tests showed that StockMaster performs as expected during normal operation. Deficiencies in the demonstration version include slow response time (because a new UNIX process is started for each query) and inability to gracefully handle simultaneous multiple users (due to the minimal database implementation). Test results also found minor bugs not significant to this stage of development.

Task	Login	Portf	Contr	Quote
	Bhag.	Ed	Jim	Mark
design	3	20	8	2
code/test	10	48	16	4
integrate	4	2	2	2
meetings	10	10	10	10
report	1	2	2	18
present	2	2	2	2
totals	30	84	40	38

Table 2: Hours Required

7 Future Work

The system can be improved by making the database relational, using encryption for transmitted data, modifying the GUI to not require a plug-in, checking for user input errors, and correctly locking database records in use.

8 Conclusion

StockMaster provides a contract-based stock portfolio manager for use as a web application. Since it is an object-oriented Java design,

and used an established design pattern, development time was kept to a minimum, about three weeks. Performance improvements may be made for each object without requiring any changes to interfaces. Decisions about the design were not trivial; use of a server-side database and contract-based model had significant scalability concerns which were carefully considered during the initial design. StockMaster is a very good demonstration of the benefits and tradeoffs of Java and OOP applied to e-business.

References

- [1] Timothy Budd. *An Introduction to Object Oriented Programming (2nd Edition)*. Addison Wesley, Reading, MA, 1997.
- [2] Martin Fowler. *Analysis Patterns: Reusable Object Models*. Addison Wesley, Reading, MA, 1997.
- [3] Erich Gamma, Richard Helm, Ralph Johnson, and John Vissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.
- [4] Cay S. Horstmann and Gary Cornell. *Core Java 2*. Prentice-Hall, Upper Saddle River, NJ, 1999.
- [5] Cary A. Jardin. *Java Electronic Commerce Sourcebook : All the Software and Expert Advice You Need to Open Your Own Virtual Store*. John Wiley and Sons, New York, NY, 1997.
- [6] Kendall Scott and Martin Fowler. *UML Distilled (2nd Ed.): A Brief Guide to the Standard Object Modelling Language*. Addison Wesley, Reading, MA, 1999.
- [7] Dennis Tischritzis and Anthony Klug. The ANSI/X3/SPARC DBMS framework report of the study group on database management. In *Information Systems*, number 3, pages 173–191, 1979.
- [8] C. Thomas Wu. *An Introduction to Object Oriented Programming With Java*. McGraw-Hill College Division, New York, NY, 1998.