Design and Simulation of an Agent-based Stock Trader

by

Xin Feng

Master of Computer Science, University of North Dakota, 2002

A Project

Submitted to the Graduate Faculty

of the

University of North Dakota

in partial fulfillment of the requirements

for the degree of

Master of Science

Grand Forks, North Dakota

May 2002

This report, submitted by Xin Feng in partial fulfillment of the requirements for the Degree of Master of Science from the University of North Dakota, has been read by the Faculty Advisor under whom the work has been done and is hereby approved.

[illegible]

Permission

Title: Design and Simulation of an Agent-based Stock Trader

Department: Department of Computer Science

Degree: Master of Science

Signature Xin Feng

Date 05/02/2002

Table of Contents

List of Figures

List of Tables

ABSTRACT

In this report, we introduce a noble implementation scheme of the Belief-Desire-Intention (BDI) model to be used in an agent-based application using Java. The example prototype system is the Agent-based Stock Trader (AST) that is a stock-trading expert based on intelligent agents. Agents in AST are based on the Belief-Desire-Intention (BDI) model in artificial intelligence.

This report proposes how to program the BDI-based agents using the Java programming language, and how to make an agent-based application more intelligent and flexible. This study contributes new implementation scheme of the BDI agents in the Java programming language useful on many applications. This work also shows how nicely implement the BDI agents with Java while manipulating BDIs intelligently and dynamically at runtime. Using our concepts and implementation scheme, the internet-based application like stock trading can be more intelligent and flexible.

1. INTRODUCTION

With the introduction of Internet, online investment comes true and lots of online brokers appear. But compared with traditional brokerages, most of online brokerages will not call clients with a hot stock tip or track clients down to tell clients, “one of your securities is in trouble and you should sell”.

So we propose to create an Agent-based Stock Trader (AST) that is intelligent to facilitate the service of recommendation. It means the AST can give clients some useful recommendations and help clients to make right decisions, just like traditional agents. The AST has its own knowledge base. It can continue learning new knowledge at run time and keep updating its knowledge base. This online agent also provides a set of basic services such as new clients’ registration, open an account, deposit/withdraw money, buy/sell stocks, check stock prices, check account status and trading history. Since we cannot access to the real stock market, we just simply simulate one by building the stock database. However, the concept and implementation scheme used in this project can be applicable to the real world problem.

The procedure and methodology is based on object-oriented analysis and design to develop this agent-oriented program. The theoretical Belief Desire Intention (BDI) model of artificial intelligence is used to implement the online agent’s intelligence. Three-tiered design is considered. HTML, Java Servlet, JSWDK, JDBC and Access are used. In this report, we emphasize how to develop the BDI-based agents in the project using the Java programming language, and how to make an agent-based application more intelligent and flexible.

The next section, we introduce some background about agent-oriented programming with BDI concepts, explains some basic concepts used in our work, and compare our work with the related works.

In the Section 3, we describe the architecture of our prototype application, Agent-based Stock Trader (AST), constructed based on the BDI-agent concepts. In the Section 4, we show our implementation scheme of AST with the Java codes. The AST is a just small example program by which we describe our nice implementation scheme of the BDI-agent concept using Java with the help of database. Therefore, we describe how we can nicely implement the BDI-agents while showing how we can implement a BDI-agent-based application like AST.

The Section 5 shows an example session of AST. With a nice user interface, each session just shows how much the AST system is applicable while using the BDI agents described technically in the previous sections. Finally we conclude with future work and references.

2. BACKGROUND OF BDI-AGENT

An agent is a software entity that has some degree of intelligence and autonomy. It is a high-level system component, which is capable of having goals that it needs to be accomplished. From the traditional definitions, agents have the following properties: autonomous, perceptive, pro-active and cooperative [DeLoach 99]. Agents may have their autonomy and are not controlled directly by the others. The perception of agents allows the communication between the agents and their environments. Agents can cooperate with other agents to achieve the common goal. Intelligent agents have learning ability, so that those agents can learn and adapt to new environment to achieve their goals in the better way while learning. Agent computing is a new and active research area today [Franklin & Graesser 96] [Petrie 96].

To realize the intelligent agent computing effectively, a model based on Belief, Desire, and Intention (BDI) has been proved as a powerful technique [Bratman 87]. The "Belief" in BDI of an agent represents the knowledge about itself and the world (outside environment) of the BDI agent. The "Desire" in BDI represents a goal that the agent likes to achieve. The "Intention" describes a set of plans to achieve the predefined goal or to react to a specific situation.

There are many possible ways to support agents based on the BDI model. Multi-paradigm mixed with object-oriented programming, BDI concepts, and logic programming can be a way to support it. Another possibility we found was knowledge base manipulation, which database supports specific knowledge domain BDI. An embedded language such as a query language embedded in the existing languages is another possible solution to support the BDI-agent model. Other solution may include distributed computing language models similarly found in distributed computing systems such as JavaSpaces and JINI.

In this work, we have tried to program BDI-based agents for an example stocking trading application using the Java programming language. Since Java does not support agent programming, there was no proper language constructs to program agents. To make it worse, Java does not have any constructs that support the BDI concept at all, and there is no way to program it. Therefore we have tried to program BDI-based agents using Java classes with database support. We show here how to program agents, belief, desire and intention by using class constructs. Stock trading agents create objects for belief, desire and intention from the corresponding classes implemented already. The desire object finds appropriate an intention object to achieve its goal based on the information of the belief object.

We have studied not only how to use the Java programming language nicely to build the BDI-agent in our online stock trader application, but also how to manipulate runtime knowledge dynamically to make agents intelligent using database for BDI knowledge.

There have been several research and experimental works based on the BDI model extending the Java programming language. BDIM Agent Toolkit is implemented as a Java package to provide a prototype of runtime architecture [Busetta & R. 97, 98].

Developing a BDIM agent needs to derive classes for each of agent's plan and belief from the relevant BDIM base classes.

JAM Agents are composed of five primary components such as a world model, a plan library, an interpreter, an intention structure, and an observer [Huber 99]. JAM uses text-based files to specify the agent's beliefs, goals and plans. To implement a JAM agent, a user has to write appropriate primitive functions in Java and specify the agent's BDI in JAM files.

JACK offers Class, Interface, Method, Syntactic and Semantic extensions of Java implemented as Java plug-ins to support an agent-oriented development environment [JACK 99]. Especially, JACK uses the Database class as its data storage device to describe an agent's beliefs. Then the Database class is fully integrated with other JACK classes. However, JACK has its own extended syntax different from Java. To build a JACK agent, a user has to understand the JACK structure very well, but it is not easy.

All of these works are based on Java. JACK uses DB nicely for belief; however, it does not support dynamic manipulation of desire and intention, while our work does. Even though JAM provides a way to manipulate BDIs, neither JAM nor BDIM supports well dynamic manipulation of BDIs not like ours. In their works, once agents are built by users, then they are fixed during runtime. In other words, users cannot modify them dynamically at runtime. Users have to redefine and recompile them if a modification is needed. In our work, we have built a BDI agent directly in Java with the help of database, so that a user can handle the BDI agent dynamically by manipulating the relations among the agent's belief, desire, and intention defined in the BDI knowledge base at runtime.

3. DESIGN OF AST

3.1 The Architecture of AST

The following figure shows the conceptual overview of the AST system. Three-tiered design is considered to make the AST system components portable and flexible [Figure 1].

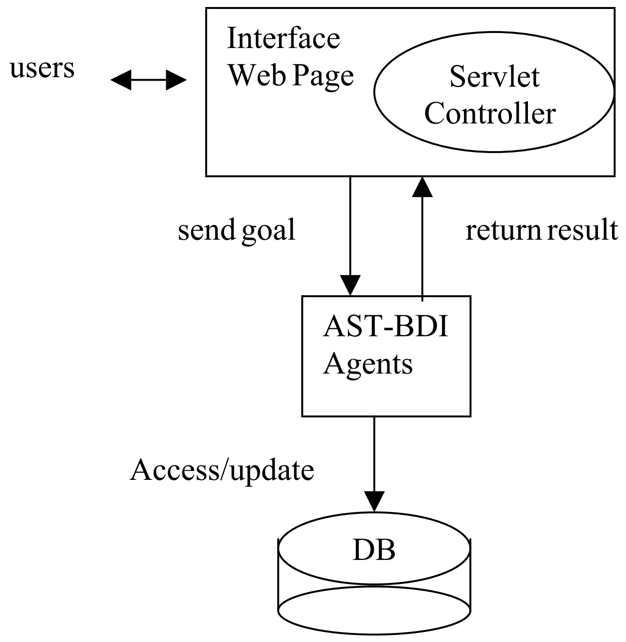


Figure 1. The Architecture of AST

The whole system consists three tiers. The first tier is the Servlet Controller, which is a link between users and BDI agents. It provides user interface and dynamically generates corresponding web pages. It includes all kinds of objective servlets that perform different functions such as registration, deposit/withdraw money, buy/sell stocks check stock prices, check account status and trading history, or give recommendations through BDI-agent. The second tier consists of the BDI agents, which are the core part of AST business. The third tier is a data access layer where all information about the AST system such as the BDI agent's beliefs and the knowledge base are stored. In AST, we use relational database to represent the beliefs and the knowledge base.

When a user asks to find the recommendation of a specific stock, the interface agent will send this goal to the agent through the servlet program. Then the corresponding agent will check whether it can fulfill this goal. If so, it will choose proper plans to achieve the goal through its control structure and return the result. If not, it may directly ask other agents for help, or send a failure message back. Finally the interface agent generates the web page from which the user can get the recommendation as a result.

3.2 The BDI Concepts in AST

Agent-based Stock Trader (AST) is a stock-trading expert based on intelligent agents, which uses BDI model in artificial intelligence. Beliefs in AST specify all kind of stocks information related to that agents know. Agents have explicit goals to achieve or events (Desires) to handle. The stock names to get recommendation from expert agents in AST can be goals. A set of plans (Intentions) is applied to describe how agents achieve their goals based on certain beliefs. Each plan elucidates how to achieve a goal under varying environments. In this report, we define a belief as a set of states representing environments, a desire as a set of goals, and an intention as a set of plans.

The examples shown in the next sections describe how BDI-agents work within AST. According to the technical analysis of the history and current information of some stocks, one plan will be chosen to give its suggestion to help users. However, another plan may be used if the decision consideration based on history, information, or stock is different. Agent itself can autonomously decide which plan will be executed according its current situation at runtime. The BDI knowledgebase helps this dynamic decision and manipulation.

4. HOW TO IMPLEMENT BDI-AGENTS?

In this section we describe how to implement agents based on the BDI-agent concept using Java.

4.1 Agent-Based Programming Language

Agent-based programming is a new programming paradigm that has been arisen from research in distributed artificial intelligence. Unfortunately we have no proper agent-based programming language to implement the BDI agent concept well. We have decided to use the Java programming language to implement encapsulation of agents. However, Java does not support the BDI agent concept. Therefore we have to devise some mechanism to support the BDI concept on Java. We discuss here how to implement the BDI agents using Java with knowledge base.

4.2 How To Implement BDI-Agents

Java is a fully object-oriented platform-independent language and has a sufficient standard class library, including network-programming facilities. It provides a natural way to develop internet-based or web-based applications. Agent-based applications can be also developed in Java. Maybe it is not natural, but it seems to be suitable.

In AST, we use a relational database to represent an agent's belief, including the agent's knowledge base and the environment states.

We have designed the following BDI mapping table [Table 1] that includes current states of belief, desires to achieve, and its corresponding intentions. By using this BDI mapping table, we can manipulate dynamically belief, desire, and intention at runtime. We can also show mapping among them, and we can change their mapping dynamically at runtime.

Table 1. BDI Mapping Table

Belief	**Desire**	**Intention**
Bi	Di	Ii
…	…	…
Bk	Dk	Ik

A proactive BDI agent acts based on a goal, scans beliefs, finds an intention based on them, and achieves the goal by executing the intention [Figure 2a]. A reactive BDI agent reacts based on belief, finds a goal and its intention based on it, can also affect its belief [Figure 2b]. DeLoach also mentioned about a goal-directed behavior [DeLoach 99] for proactive agents.

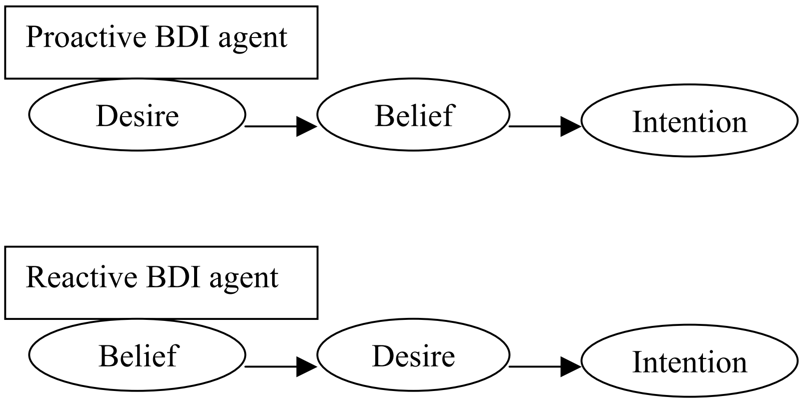


Figure 2 (2a & 2b). Proactive/Reactive BDI Agents

The AST application starts by initiating a certain goal defined in the desire definition. Based on specific belief an intention is chosen to achieve the goal based on current states of belief. There may be several sophisticated plans for each intention. Each plan triggers the event handler to achieve the goal based on behavioral description in each plan. An intention consists of a combination of one or more plans. In our current implementation, we use only one plan for each intention. To implement the concept of the BDI mapping table, we use several database tables with Java classes like the following. The related Java codes are shown in the next section.

The following table [Table 2] shows the AST database contents.

Table 2. AST Database Contents

AST Database	
Table Name	**Content**
Client	Clients' personal information and account balance.
Holding	What kinds and how many stocks clients own.
Orders	All orders clients posted and the agent processed.
Knowledge	Agent's knowledge. Desire, belief values and a synthesis value of belief values.
BPmap	Shows what plan will be chosen according the certain condition.

It is good to be able to manipulate BDIs dynamically for the BDI-based agent system. Belief can be dynamically changed to represent current environment. Desires can be set and changed to new ones at run time based on current situations such as belief. Intentions can be changed, updated, or newly added to achieve a current desire. However, it is hard to manipulate BDIs directly and dynamically well using Java. To solve this, the belief values can be stored in a file or database table, and can be manipulated dynamically at run time. For this AST application, belief was stored in a database table with which stock agents can consult and update dynamically. In the AST, the desire is also stored and manipulated dynamically in a database table. Both desire and belief are stored at the “Knowledge” table in the AST database. To map the corresponding plans in the intention dynamically based on both the current goal defined in the desire and the current environment based on the belief, another table “BPmap” has been used in the AST system. The table “Client” holds clients’ account information. The table “Holding” keeps the information of stocks owned by clients. The table “Orders” keeps the information of orders which clients post and the AST agent should process. Among contents in the AST database, the table “Knowledge” is the most important table to manipulate the BDI information dynamically.

The following table “Knowledge” shows a knowledge base for desire and belief [Table 3].

Table3. AST Database Knowledge Table Definition

AST BDI Knowledge		
Field Name	**Data Type**	**Description**
Symbol	Text	Stock name
A	Number	World economy
B	Number	US economy
C	Number	Financial markets and institutions
E	Number	Others
F	Number	Buy or sell market
Belief	Number	The result of analysis of A~F

The stock name “Symbol” is a kind of desire to get any recommendation on its stocks through the stock market for stock exchange (either selling or buying). For example, the “Symbol” may hold desire that represents a certain stock of company such as ORCL (Oracle), YHOO (Yahoo), IBM (IBM), MS (Microsoft), and etc. The knowledge A through F represents its belief that represents the current environment surrounding its stock. Its belief holds the values that represent the current status of environment such as world economy, US economy, financial markets and institutions, other factors, and buy/sell market. The final field “Belief” holds the value that we can get from the analysis of the belief factors, A~F, by applying certain mathematical and stochastic equations. In summary, the “Symbol” keeps the value of current goal/desire, the values from A to F keep the current environment/belief, and the value of “Belief” is synthesized from the values of A through F. The following table shows possible values for the table “Knowledge” [Table 4].

Table 4. Sample Values in Knowledge Table

Values in Knowledge							
Symbol	**A**	**B**	**C**	**D**	**E**	**F**	**Belief**
YHOO	-1	0	1	1	1	1	3
…	…	…	…	…	…	…	…

The following table shows the table “BPmap” that describes the mapping to the corresponding plans defined in the intention, from the information based on the current environment defined in the belief [Table 5]. Therefore, it is a kind of mapping table from belief to intention.

Table 5. BPmap Table Definition

Bpmap		
Field Name	**Data Type**	**Description**
LowLimit	Number	Lower limit of belief
UpLimit	Number	Upper limit of belief
Plan	Text	Corresponding plan
PlanChoice	Text	Choice of a plan

The condition fields, LowLimt and UpLimit, represent the lower limit and upper limit of the value of belief, from which the corresponding plans are selected. Both “Plan” and “PlanChoice” are to choose plans defined in the AST intention, which can be selected by a certain rules defined in the AST system based on the current goal/desire and the current situation/belief.

The following table shows some example values for each column in “BPmap” table [Table 6].

Table 6. Sample Values in BPmap Table

Values in Bpmap			
LowLimit	**UpLimit**	**Plan**	**PlanChoice**
-3	3	Hold	4
…	…	…	…

The “Plan” may have recommendation values for stock exchange such as “Strongly Sell”, “Moderate Sell”, “Hold”, “Moderate Buy”, or “Strongly Buy”. The “PlanChoice” holds the identification number for each plan defined in the AST intention definition. Theoretically, we may have one or a combination of plans. However, to make this example simple, we use only one plan as a recommendation value.

4.3 How To Program BDI-Agents

The stock trading agent can be created by defining its own Belief, Desire, and Intention classes. The agent has its own main controller whose control structure is shown in Figure 3.

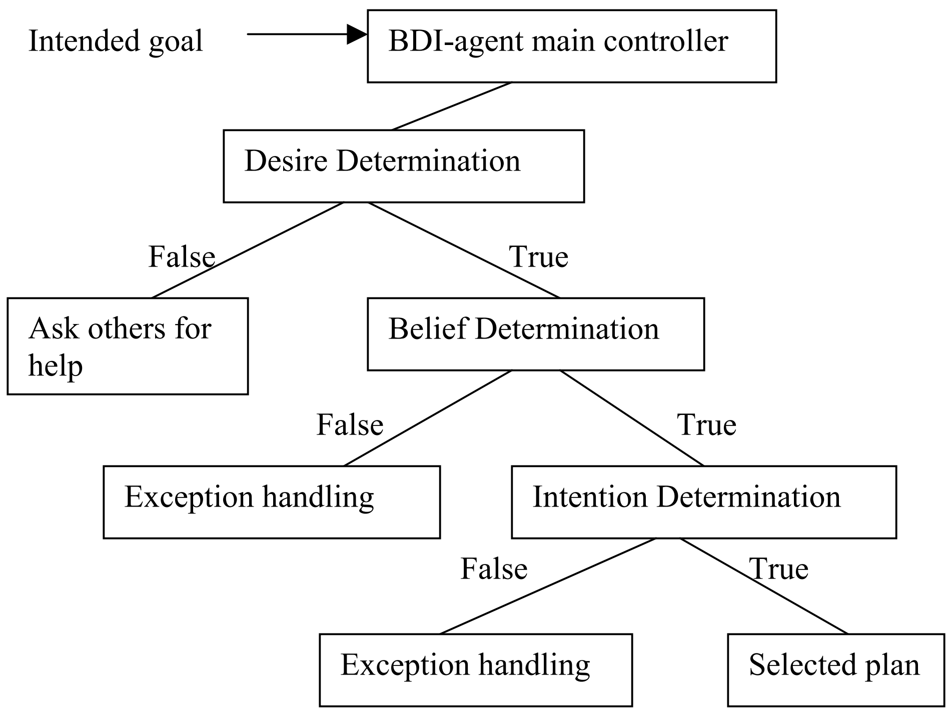


Figure 3: BDI-Agent Runtime Control

The following code is the Belief class to access and update the knowledge base for its belief. The “getBelief(String goal)” function accesses the belief knowledge base based on the current goal (“getBelief(String goal)”), and performs some mathematical and stochastic equations to calculate the exact current environment surrounding the goal. (Even though in this example application we use a just simple summation formula, in the real application, any heuristic formula can be used to diagnose the exact case and stock market.).

```
// Belief.java

package agent;
import java.util.*;
import java.sql.*;

public class Belief {
  private Connection connection;

  public Belief (Connection x)  {   this.connection = x;   }

  public Connection getConnect() {  return connection;  }

  //Get believes according to the desire
  public int getBelief (String goal)
  {
    try {
      Statement st = connection.createStatement();
      String sql = "SELECT * FROM knowledge WHERE Symbol='" + goal + "'";
      ResultSet rs = st.executeQuery(sql);
      boolean more = rs.next();
      if (more) {
        int sum = 0;
        ResultSetMetaData rsmd = rs.getMetaData();
        for (int i=2; i<rsmd.getColumnCount(); ++i)
              //1st element: Symbol
              sum += rs.getInt(i);
        System.out.println("sum="+sum);
        return sum;     // A value of belief synthesis
      }
      else
         return 11111; // 11111 is  the false flag
    }
   catch (SQLException sqlex) {
      sqlex.printStackTrace();
      System.out.println("Desire SQL error\n");
      return 11111;     // A false flag
   }
  }

  // update believes of the desire
  // … …
}
```

The Desire class accesses to its knowledge base and finds whether a certain goal can be achieved or not (by "achievable(String goal)"). If the goal is achievable the main agent program will select proper plans under proper situation based on belief. The following program shows the Desire class for the AST application.

```
// Desire.java

package agent;
import java.util.*;
import java.sql.*;

public class Desire {
    private Connection connection;

    public Desire (Connection x)  // constructor
    {   this.connection = x;  }

   public boolean achievable(String goal)
   {
       try {
            Statement st = connection.createStatement();
           String sql = "SELECT Symbol FROM knowledge WHERE Symbol='"
                        + goal + "'";
            ResultSet rs = st.executeQuery(sql);
            if (rs.next()){
                System.out.println(rs.getString(1));
               return true;
            }
            else
                return false;
       }
       catch (SQLException sqlex) {
            sqlex.printStackTrace();
            System.out.println("Desire SQL error\n");
            return false;
       }
  }

  // add a new goal
  // … …
  // delete a goal
  // … …
}
```

Theoretically, we can build an intention class to list a set of plans the agent prepares for possible goals. Each plan is represented as a method in the intention class, which is a behavioral representation to describe how to achieve a goal under a particular situation represented by a set of states in the belief representation. However, Java does not support dynamic configuration (mapping) of a combination of plans, which are suitable for the BDI agent concept. Therefore, in our current implementation, we use an intention-mapping table ("BPmap") in the database to store a set of rules that explain how to dynamically select a suitable plan under a particular situation (performed by the "selectPlan" function). Then the agent can dynamically access and monitor its belief through JDBC and SQL statements. The following program shows its Intention class.

```
// Intention.java

package agent;
import java.util.*;
import java.sql.*;

public class Intention {
    private Connection connection;

    public Intention(Connection x)
    {   this.connection = x;        }

    public String selectPlan(int bresult) {
        try {
             Statement st = connection.createStatement();
             String sql = "SELECT Plan FROM BPmap WHERE UpLimit>" + bresult
                          + " and LowLimit<=" + bresult;
             ResultSet rs = st.executeQuery(sql);
            if (rs.next()){
                String s = rs.getString(1);
                System.out.println("Intention plan = "+s);
                return s;
             }else
                return "fail";
        }
        catch (SQLException sqlex) {
             sqlex.printStackTrace();
             return "fail";
        }
    }
```

```
  // update the mapping table
  // … …
}
```

We may also have methods to update plans by which we can update the mapping values in the mapping table (BPmap table). It is a good idea to have a proper programming language to update plans dynamically and more naturally.

Finally, we can create an agent class by declaring its own belief, desire, and intention from their definitions. When a goal is sent to an agent through the desire, the agent will check whether it can handle. If so, its corresponding desire will be achieved by executing proper plans defined in its intention. If not, it may delegate it to other agents for help. It can be implemented by message exchange among multi-cooperative agents, even though our current system is not implemented in this way. The following code shows the Agent class for the AST application.

```
// Agent.java

package agent;
import java.util.*;
import java.sql.*;

public class Agent {
   public Belief B;
   public Desire D;
   public Intention I;

   // constructor
   public Agent (Connection x)
   {
      D = new Desire(x);
      B = new Belief(x);
      I = new Intention(x);
   }

   // agent performance controller
   public String perform(String goal)
   {
        if ( D.achievable(goal) ) {
           int b = B.getBelief(goal);
           if (b==11111)  // 1111 means False flag
              return "Belief sensor error";
           else {
              String p = I.selectPlan(b);
              return p;
           }
        }
        else
           return "Unachievable goal";
    }

}
```

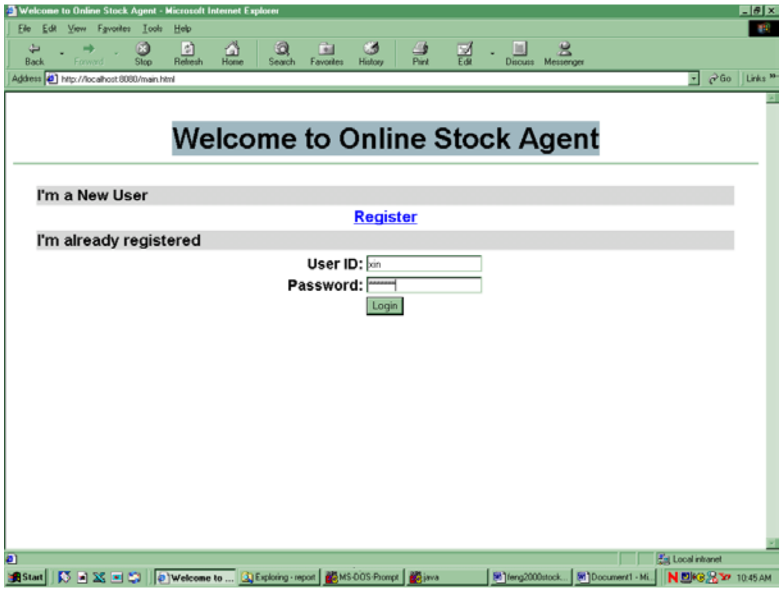
An agent always starts from the behavioral description named “perform” in this application (“perform(String gaol)”). A desire is passed to the application through the string “goal” (“String goal”). The desire class checks whether the current goal is achievable or not (by “D.achievable(goal)”). If it is achievable, the system gets the current environment through the belief (by “B.getBelief(goal)”). Unless the current belief is not available, its plan is chosen to achieve the given goal using the information of belief (“I.selectPlan(b)”).

5. EXAMPLE SESSIONS

Here we show some example sessions using AST.

5.1 Login Entry

The AST system starts with the entry form of user login. The registered users may login with their user Ids and passwords. The new user may register into the system. This session is related to the table "Client" in the database [Table 2]. Here is a snapshot of the entry login form.


Welcome to Online Stock Agent - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Print Edit Discuss Messenger
Address http://localhost:8080/main.html
Go Links
Local intranet
Start
10:45 AM

Welcome to Online Stock Agent

I'm a New User

Register

I'm already registered

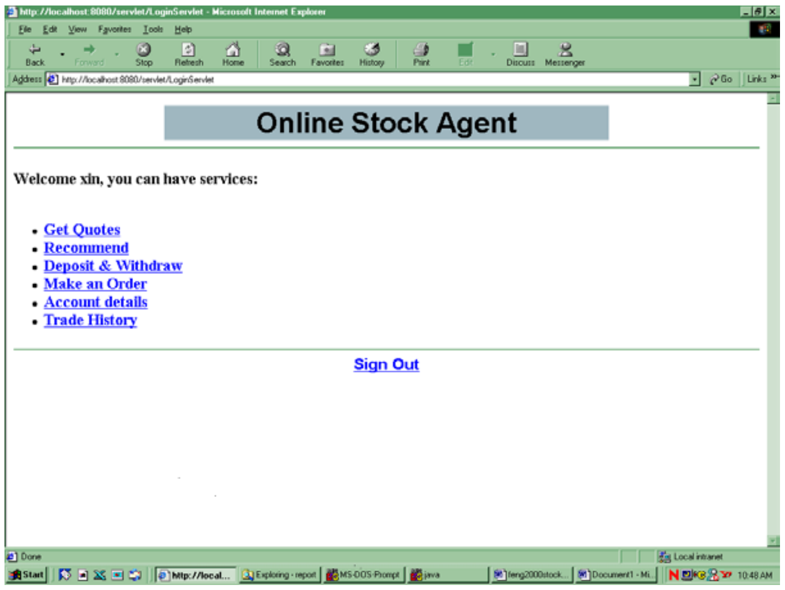
User ID: xin

Password: ********

Login

5.2 Basic Services

After successful login, a list of services regarding online stock trading that AST can provide appears. Currently the prototype system provides six services such as getting quotes, recommendation for stock trading, deposit/withdraw, order placement, account details, and trading history.


Online Stock Agent

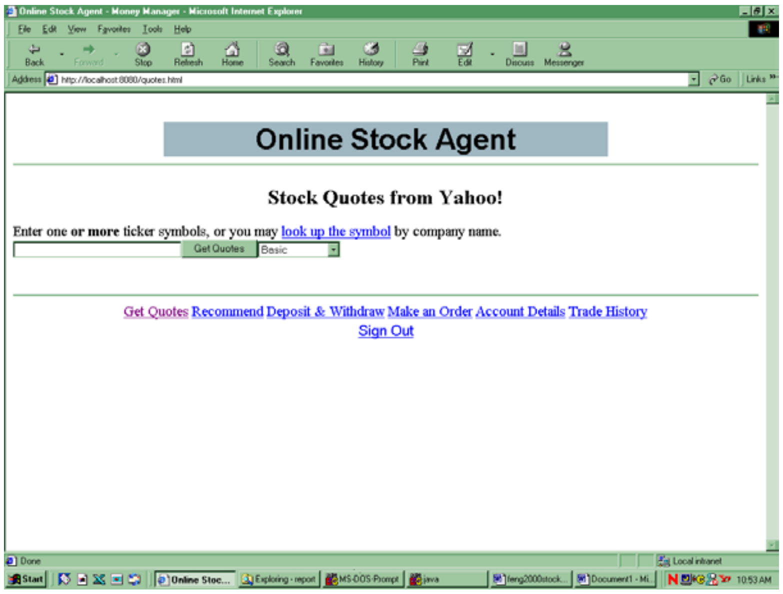
Welcome xin, you can have services:

- Get Quotes
- Recommend
- Deposit & Withdraw
- Make an Order
- Account details
- Trade History

Sign Out

5.3 Stock Quotation

The selection of the stock quote option brings the user to the stock quotation menu. We borrow the Stock Quotes from Yahoo [Yahoo 00], due to the limitation of accessing to a real stock market database.


Online Stock Agent - Money Manager - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Print Edit Discuss Messenger
Address http://localhost:8080/quotes.html
Go Links
Get Quotes
Basic
Done
Local intranet
Start
10:53 AM

Online Stock Agent

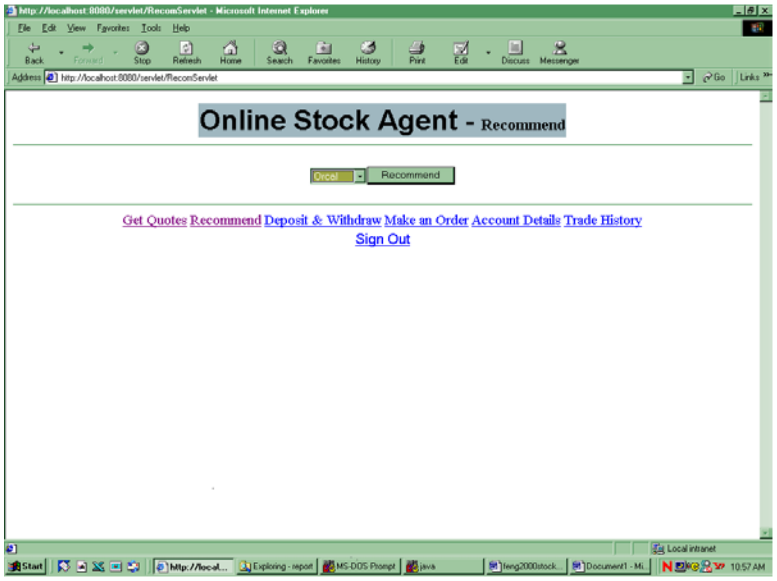
Stock Quotes from Yahoo!

Enter one or more ticker symbols, or you may look up the symbol by company name.

Get Quotes Recommend Deposit & Withdraw Make an Order Account Details Trade History
Sign Out

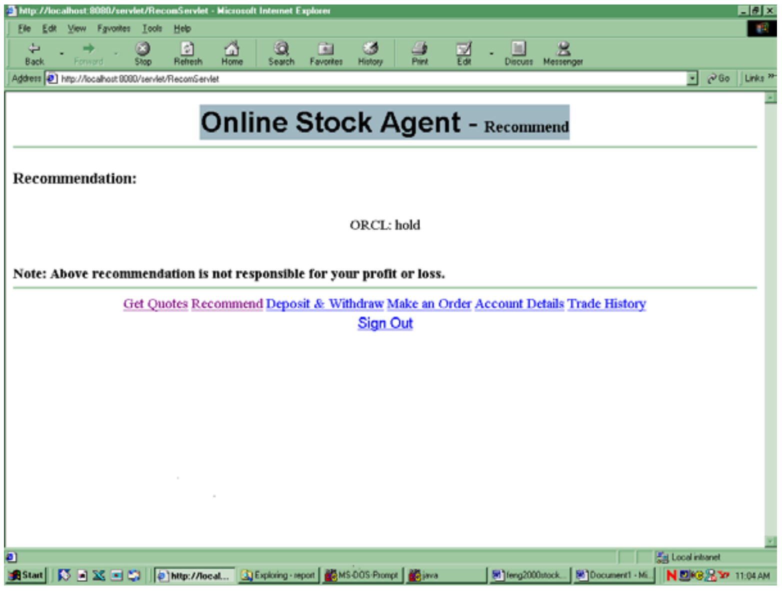
5.4 Stock Trading Recommendation

The recommendation service provides a recommendation to buy, hold, or sell for a certain stock. Actually the expert agent does the background work. Once a user selects a stock and clicks on the "Recommend" button, this stock name will be sent to the expert BDI-Agent as a goal through the servlet program. Then the expert BDI-Agent achieves the goal (by "perform (String goal)") just as we described in the previous sections. It checks its desire and current belief to select and execute a suitable plan. Finally the expert BDI-Agent sends back its achievement, the recommendation of the stock, to the interface through the servlet program. Both "Knowledge" and "BPmap" tables [Table 2] are related to this session.



http://localhost:8080/servlet/RecomServlet - Microsoft Internet Explorer
Online Stock Agent - Recommend
Orcal
Recommend
Get Quotes Recommend Deposit & Withdraw Make an Order Account Details Trade History
Sign Out

Once the agent performs a certain function needed to get an appropriate recommendation, the system shows the result. This is the corresponding answer from the export agent. With this particular example, the AST system recommends the user to hold the ORCL stock continuously.



Online Stock Agent - Recommend

Recommendation:

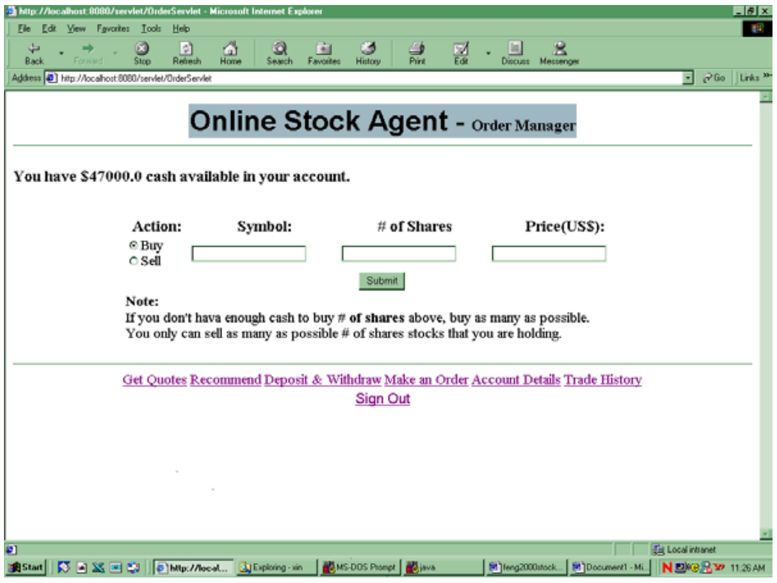
ORCL: hold

Note: Above recommendation is not responsible for your profit or loss.

http://localhost:8080/servlet/RecomServlet - Microsoft Internet Explorer
Get Quotes Recommend Deposit & Withdraw Make an Order Account Details Trade History
Sign Out

5.5 Posting An Order

Here the user can post his stock order, buy or sell stocks, and the order management agent will process it for the user. The order will be either successful or fail based on the several reasons such as the availability of the stock in the market to buy, and the availability of the funds to buy the stock, and etc. In our system, we simply use the random function to decide an order successful or fail since we cannot access to the real stock market. The related tables in the database are “Client”, “Holding” and “Orders” [Table 2].


http://localhost:8080/servlet/OrderServlet - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Print Edit Discuss Messenger
Address http://localhost:8080/servlet/OrderServlet
Go Links
Local intranet
Start
11:26 AM

Online Stock Agent - Order Manager

You have $47000.0 cash available in your account.

Action: Symbol: # of Shares Price(US$):

[✓ Buy] Sell

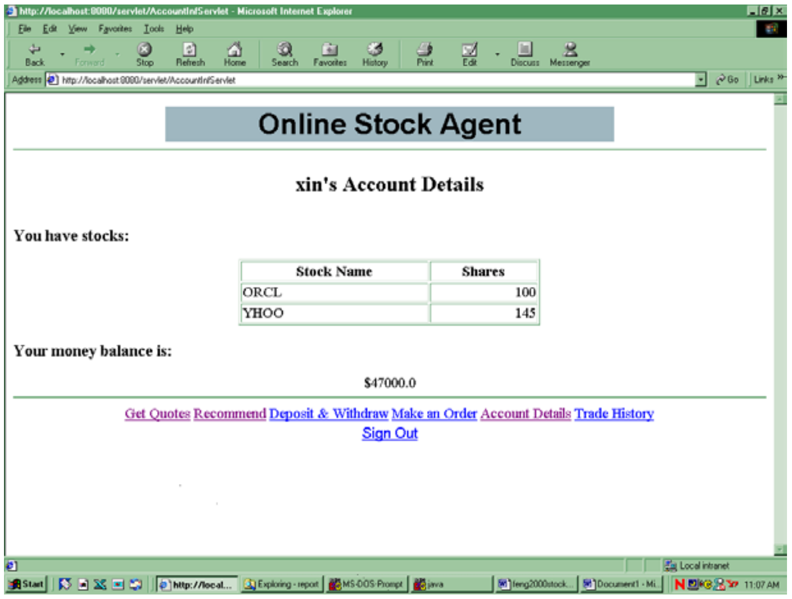
Submit

Note:
If you don't hava enough cash to buy # **of shares** above, buy as many as possible.
You only can sell as many as possible # of shares stocks that you are holding.

Get Quotes Recommend Deposit & Withdraw Make an Order Account Details Trade History
Sign Out

5.6 Account Details and Trading History

The account detail option in the main menu can be selected to see the user’s account in detail. It shows what stocks and how much money the user owns. The tables “Client” and “Holding” [Table 2] relate to this session.



Online Stock Agent

xin's Account Details

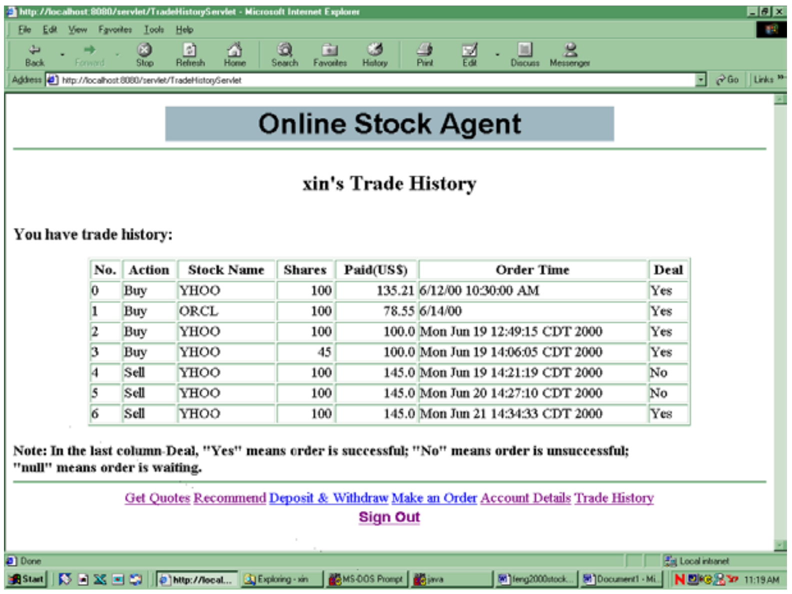
You have stocks:

Stock Name	Shares
ORCL	100
YHOO	145

Your money balance is:

$47000.0

A user may select the trading history option to see his all trade history in the current past, which includes successful deals, unsuccessful deals and deals in processing.



Online Stock Agent

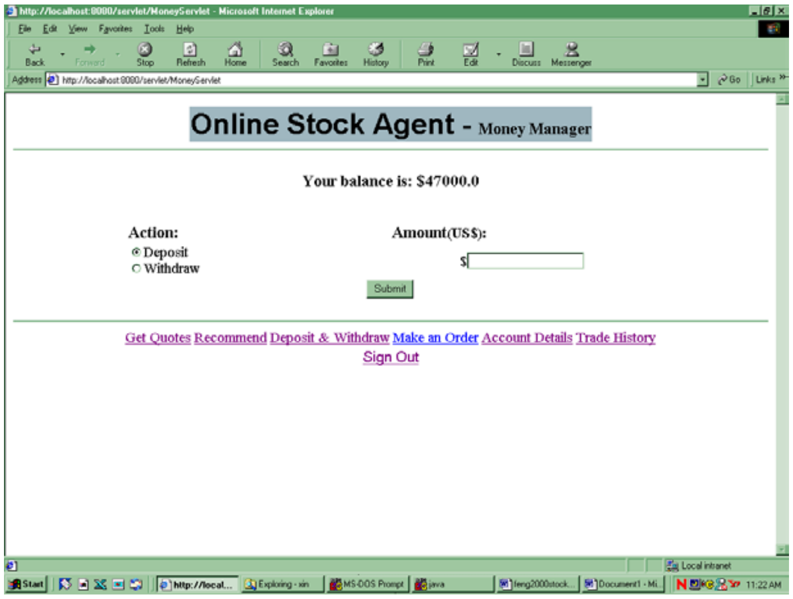
xin's Trade History

You have trade history:

No.	Action	Stock Name	Shares	Paid(US$)	Order Time	Deal
0	Buy	YHOO	100	135.21	6/12/00 10:30:00 AM	Yes
1	Buy	ORCL	100	78.55	6/14/00	Yes
2	Buy	YHOO	100	100.0	Mon Jun 19 12:49:15 CDT 2000	Yes
3	Buy	YHOO	45	100.0	Mon Jun 19 14:06:05 CDT 2000	Yes
4	Sell	YHOO	100	145.0	Mon Jun 19 14:21:19 CDT 2000	No
5	Sell	YHOO	100	145.0	Mon Jun 20 14:27:10 CDT 2000	No
6	Sell	YHOO	100	145.0	Mon Jun 21 14:34:33 CDT 2000	Yes

Note: In the last column-Deal, "Yes" means order is successful; "No" means order is unsuccessful; "null" means order is waiting.

The fund manager handles both deposit and withdraw for the user.



Online Stock Agent - Money Manager

Your balance is: $47000.0

Action: [✓ Deposit] Withdraw

Amount(US$): $

http://localhost:8080/servlet/MoneyServlet - Microsoft Internet Explorer
Submit
Get Quotes Recommend Deposit & Withdraw Make an Order Account Details Trade History
Sign Out

6. CONCLUSIONS

Agent-based computing is emerged as a new computing paradigm. The BDI model is one of the powerful techniques to describe autonomous intelligent agents. In this report we have presented a stock trading application based on intelligent agents using the BDI model.

One of the merits of this work is to show how nicely we can use the object-oriented language, Java, to implement the BDI-agent-based application. The Java programming language does not support any construct for the BDI-agent concepts. However, in our work, the agent and its belief, desire, and intention are programmed as Java classes. Its information of desire, belief, and intention are stored in a database and updated dynamically at runtime as the environment changes. Therefore, in our work, we show how nicely we can use the Java programming language to program the BDI-agent-based application by using a database to implement the knowledge base for BDIs. We also show how we can manipulate BDIs dynamically at runtime without having any trouble while Java does not support any runtime knowledge management. Our work also shows how we can implement a real world application like stock trading using the BDI-agent model to represent the real world problem more naturally in a better way. The Java/Servlet programming technique on the Internet has been used to implement our prototype system.

7. ACKNOWLEDGMENTS

This work has been partially supported by the EPSCoR IIP Seed Grant (4709-0701) in summer 2001.

8. REFERENCES

[1] Agent Oriented Software Pty. Ltd., JACK Intelligent Agents User Guide, http://www.agent-software.com.au, 1999.

[2] Bratman, Michael E., Intention, Plans, and Practical Reason, Harvard Univ. Press, 1987 (also by CSLI Publication, 1999).

[3] Busetta, Paolo and Ramamohanarao, Kotagiri. The BDIM Agent Toolkit Design, Technical Report 97/15, Department of computer Science, The University of Melbourne, Australia, 1997. http://www.cs.mu.oz.au/publications/tr_db/TR.html.

[4] Busetta, Paolo and Ramamohanarao, Kotagiri. An architecture for Mobile BDI Agent, Mobile Computing Track, ACM SAC '98, 7-8, 1998.

[5] DeLoach, Scott A. Multiagent Systems Engineering: A Methodology and Languages for Designing Agent Systems, http://en.afit.af.mil/ai/publications/Conference/aois-99/MaSE-AOIS99.htm, 1999.

[6] Franklin, Stan and Graesser, Art. Is it an Agent, or just a Program? : A Taxonomy for Autonomous Agents, http://www.msci.memphis.edu/~franklin/AgentProg.html, Also in the Proc. of the 3rd International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 2-7, 1996.

[7] Huber, Marcus J. JAM: A BDI-theoretic Mobile Agent Architecture, Proc. Of the Autonomous Agents '99, Seattle, USA, 236-243, 1999.

[8] Jo, Chang-Hyun, Agent-based Programming Language: APL, ACM SAC 2002 Conference, Madrid, Spain, 27-31, March 2002.

[9] Petrie, Charles J. Agent-Based Engineering, the Web, and Intelligence, http://www-cdr.stanford.edu/NextLink/Expert.html, also appeared in the IEEE Expert, (December 1996).

[10] Yahoo Stock Quotes, http://finance.yahoo.com, 2000.

APPENDICES

APPENDIX 1: Agent-based Stock Trader (AST) User Manual (Readme.doc)

1. Introduction

Agent-Based Stock Trader (AST) is an Internet-based application project using BDI agent concepts in Java programming language. AST provides a set of basic services. For example, a new client can register and open an account, after logging in, the client can get a set of services, such as deposit/withdraw money, buy/sell stocks, check stock prices, check his account detail and trading history. AST also facilitates the service of recommendation based on its knowledge base. Since we cannot access to the real stock market, we just simply simulate one by building an AST database. Figure 4 presents the structure of AST package and contents.

2. Requirements

- **Operating System**

  This release has been tested on the following platforms: Windows 95/98.

- **JDK**

  Install JDK1.1.8 or up version. http://java.sun.com/products/jdk/1.2/install-windows.html

- **JSWDK1.0.1**

  Download and install JSWDK1.0.1 at http://java.sun.com/

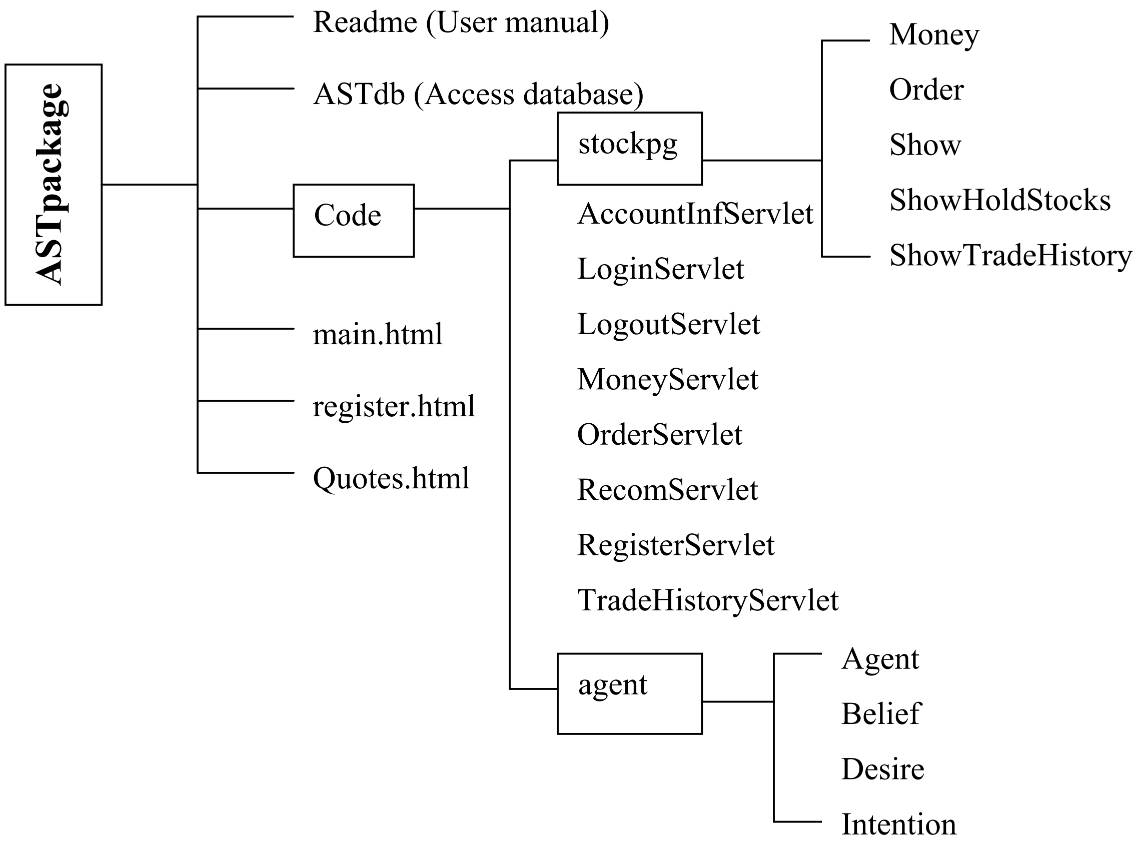


Figure 4: ASTpackage Structure

- **Environment variables**

1. Add the full path of the JDK bin directory (where the java command is located) to the PATH variable. To do so, click Start-> Run and enter Sysedit. Open and edit the AUTOEXEC.BAT file and add or change the PATH statement.
2. Add the tools.jar (from the JDK lib directory) file to your CLASSPATH variable. Save AUTOEXEC.BAT and close it. Restart your computer.

3. Copy servlet.jar (from JSWDK1.0.1 lib directory) file to your JDK extensions directory (the directory c:\jdk1.2.2\jre\lib\ext on windows).

- **Microsoft Access**

  To execute this application, you need to have a database system like Microsoft Access. ODBC for MS Access database is provided with windows system. You need to set up the AST database file as described in the following section.

3. Install AST

- **Register AST database**

  AST database named ASTdb.mdb contains five tables. To connect to the database, an ***ODBC Data Source*** must be registered with the system through the ***ODBC Data Sources*** option in the Windows **Control Panel**.

1. Double-click this option to display the ***ODBC Data Source Administrator*** dialog.
2. Select the ***User DSN*** tab is selected, then click ***ADD***… to display ***the Create New Data Source*** dialog.
3. Select ***Microsoft Access Driver*** and click ***Finish***. Then ***the ODBC Microsoft Access 97 Setup*** dialog appears.
4. Click on **Select…** button to display the **Select Database** dialog. Use this dialog to locate and select the **ASTdb.mdb** file on your system. Click **OK** to return to the ***ODBC Microsoft Access 97 Setup*** dialog.

5. Enter the name (**ASTdb**) in the ***Data Source Name*** text field. Enter a description.
6. Click **OK** to dismiss ***ODBC Microsoft Access 97 Setup*** dialog.
7. Notice that the ***ODBC Data Source Administrator*** dialog now contains the data source **ASTdb.** Click **OK** to dismiss the dialog.

- **Copy HTML files**

  Copy main.html, quotes.html and register.html (from the Appendix 3) to the directory jswdk-1.0.1/webpages/

- **Install servlet classes**

  Copy all contents in ASTpackage/code directory to the directory jswdk-1.0.1/webpages/WEB-inf/servlet

4. Operation procedure

- **Start server**

  To start the server, open the directory with Windows Explorer and double-click on `STARTSERVER.BAT` in JSWDK1.0.1 directory (If this does not work, you can try opening a DOS window and running it from there.)

- **Start user-interface**

  To start the user-interface, open your web browser. Either Internet Explorer or Netscape Navigator will be fine. Use your browser to access the following address: http://localhost:8080/main.html. Then the main user-interface, which is login entry, will be brought to user. You can start to test AST and monitor the server window.

If you test AST from a remote computer, you have to find out the IP address or domain name (DN) of the server computer that you have installed AST application in. Then you access to the IP address or DN of the server instead of localhost.

- **Stop server**

  To stop the server, open Windows Explorer and double-click on `STOPSERVER.BAT` in JSWDK1.0.1 directory (If this doesn't work, you can try opening a DOS window and running it from there.)

5. Features and Deficiencies

The AST system starts with the entry form of user login. The registered users may login with their user Ids and passwords. The new user may register into the system with a unique id. After successful login, a list of services regarding online stock trading that AST can provide appears. Currently the prototype system provides six services such as getting quotes, recommendation for stock trading, deposit/withdraw, order placement, account details, and trading history. Getting Quotes service is borrowed from Yahoo, due to the limitation of accessing to a real stock market database. The same reason for order service, we use a random function to process an order during a time period. Recommendation service is based on the BDI agent, which runs dynamically according to the manipulation of its knowledge base and the BDI mapping table.

APPENDIX 2: ASTdb Access Database

ASTdb Access Database contains 5 tables. The details are shown at Table 2 [p.11]. The definitions of Knowledge Table and Bpmap Table are shown at Table 3 [p.12] and Table 5 [p.14]. The other tables' definitions are shown at the following tables.

Table 7. Client Table Definition

Field Name	Data Type	Description
UserID	Text	User Id (primary key)
Password	Text	Password
Lname	Text	last name
Fname	Text	First name
Email	Text	email address
Other	Text	other information
Cash	Number (Double)	money on the account

Table 8. Orders Table Definition

Field Name	Data Type	Description
OrdNum	Number (int)	Order number (primary key)
UserID	Text	User id (primary, foreign key)
Action	Text	Sell or Buy
Symbol	Text	Stock name
Shrs	Number (int)	The shares of the stock
Paid	Number (Double)	The price of each share
OrdTime	Text	The time of an order pasted
Deal	Text	Wait or Yes or No

Table 9. Holding Table Definition

Field Name	Data Type	Description
UserID	Text	User Id (primary key, foreign key)
Symbol	Text	stock name
Shrs	Number (int)	Shares

We give some sample data in AST access database, which are shown in the following table.

Table 10. Sample Data in ASTdb.mdb for MS Access

Client

userID	password	LName	fName	Email	other	cash
xin	xintest	Feng	Xin	xin@cs.und.edu		36500

Holding

userID	Symbol	Shrs
xin	ORCL	100
xin	YHOO	200
xin	IBM	100

Orders

OrdNum	userID	Action	Symbol	Shrs	Paid	OrdTime	Deal
1	xin	buy	YHOO	100	10	Thu Sep 06 14:35:11 PDT 2001	No
2	xin	buy	YHOO	300	10	Thu Sep 06 14:43:25 PDT 2001	Yes
3	xin	sell	YHOO	100	20	Wed Jan 30 11:32:55 PST 2002	No
4	xin	sell	YHOO	100	20	Wed Jan 30 11:39:10 PST 2002	Yes
5	xin	buy	orcl	100	10	Wed Jan 30 11:45:34 PST 2002	No
6	xin	buy	orcl	200	10	Wed Jan 30 11:52:42 PST 2002	Yes
7	xin	sell	orcl	100	15	Wed Jan 30 15:16:14 PST 2002	Yes
8	xin	buy	IBM	100	100	Wed Jan 30 15:20:47 PST 2002	Yes

BPmap

LowLimit	UpLimit	Plan	PlanChoice
-1000	-10	Strong Sell	1
-10	-3	Miderate Sell	2
-3	3	hold	3
3	10	Miderate Buy	4
10	1000	Strong Buy	5

Knowledge

Symbol	A	B	C	D	E	F	Belief
ORCL	-1	-1	1	1	0	1	
YHOO	-1	0	1	1	1	1	

APPENDIX 3: Source Codes

```
<!--main.html -->

<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1">
   <meta name="Author" content="XinFeng">
   <meta name="GENERATOR" content="Mozilla/4.73 [en] (Win98; I)
[Netscape]"><title>Welcome to Online Stock Agent</title>
</head>

<body>
<center><table>
<tr><td BGCOLOR="#A0B8C8"><b><font face="Arial"><font size=+3>Welcome
to Online Stock Agent</font></font></b></td></tr></table></center>

<hr><p>
<form action=http://localhost:8080/servlet/LoginServlet method=post>
<center><table WIDTH="94%" >
<tr><td BGCOLOR="#DCDCDC"><b><font face="Arial"><font size=+1>I'm a New
User</font></font></b></td></tr>

<tr><td ALIGN=CENTER><b><font face="Arial"><font size=+1><a
href="http://localhost:8080/register.html">Register</a></font></font></
b></td></tr>
<tr><td></td></tr>
<tr><td BGCOLOR="#DCDCDC"><b><font face="Arial"><font size=+1>I'm
already registered</font></font></b></td></tr>

<tr><td><center><table><tr>
<td ALIGN=RIGHT NOWRAP><b><font face="Arial">User ID:</font></b></td>
<td><input name=login size=20 maxlength=32></td></tr>
<tr>
<td ALIGN=RIGHT NOWRAP><b><font face="Arial">Password:</font></b></td>
<td><input name=password type=password size=20 maxlength=32></td></tr>
<tr><td ALIGN=CENTER COLSPAN="2" NOWRAP><input type=submit
value="Login"></td></tr>
</table></center></td></tr>

</table></center>
</form>
</body>
</html>
```

```
<!--register.html -->

<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1">
   <meta name="Author" content="XinFeng">
   <meta name="GENERATOR" content="Mozilla/4.73 [en] (Win98; I)
[Netscape]"><title>Register Form</title>
</head>

<body>
<h1>Registration Form</h1>
<hr><form action=http://localhost:8080/servlet/RegisterServlet
method=post>
<center><table WIDTH="94%" >
<tr><td><b><font size=+2>Create Your Account</font></b></td>
<td><font size=+1>(Already Have One? <a
href="main.html">Login</a>)</font></td></tr>

<tr><td ALIGN=RIGHT><b>User ID:</b></td>
<td><input type=text name="login" value="" autocomplte=off size=20
maxlength=32></td></tr>

<tr><td ALIGN=RIGHT><b>Passward:</b></td>
<td><input type=password name="pwd1" value="" autocomplete=off size=20
maxlength=32></td></tr>

<tr><td ALIGN=RIGHT><b>Retype passward:</b></td>
<td><input type=password name="pwd2" value="" autocomplete=off size=20
maxlength=32></td></tr>
</table></center>

<center><table WIDTH="94%" >
<tr><td><b><font size=+2>Personal Information</font></b></td></tr>

<tr><td><b>* First Name:</b><input type=text name=FirstName></td></tr>

<tr><td><b>* Last Name:</b><input type=text name=LastName></td></tr>

<tr><td><b>* Email address:&nbsp;</b><input type=text name=Email
size=50></td></tr>

<tr><td><b>Other Information:</b><input type=text name=other
size=50></td></tr>

<tr><td ALIGN=CENTER>* fields are required</td></tr>

<tr><td ALIGN=CENTER><input type=submit value="Submit"></td></tr>
</table></center>
</from>
</body>
</html>
```

```
<!--quotes.html -->

<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1">
   <meta name="Author" content="xinfeng">
   <meta name="GENERATOR" content="Mozilla/4.73 [en] (Win98; I)
[Netscape]">
   <title>Online Stock Agent - Quotes</title>
</head>

<body>

<center><table WIDTH="60%" >
<tr BGCOLOR="#A0B8C8">
<td ALIGN=CENTER><b><font face="Arial"><font size=+3>Online Stock
Agent</font></font></b></td>
</tr>
</table></center>

<hr>
<center><h2>Stock Quotes from Yahoo!</h2></center>

<form method=get action="http://finance.yahoo.com/q"
target=_blank>Enter one <b>or more </b>ticker symbols, or you may
<a href="http://finance.yahoo.com/l" target=_blank>look up the
symbol</a> by company name.<br>
<input type=text name=s size=30><input type=submit value="Get
Quotes"><select name=d><option value=v1 selected>Basic<option
value=v2>DayWatch<option value=v4>Fundamentals<option
value=t>Detailed<option value=1y>Chart<option
value=r>Research</select></form>

<br><hr>

<center><table>
<tr>
<td><a href="http://localhost:8080/quotes.html">Get Quotes</a></td>
<td><a href="/servlet/RecomServlet">Recommend</a></td>
<td><a href="/servlet/MoneyServlet">Deposit &amp; Withdraw</a></td>
<td><a href="/servlet/OrderServlet">Make an Order</a></td>
<td><a href="/servlet/AccountInfServlet">Account Details</a></td>
<td><a href="/servlet/TradeHistoryServlet">Trade History</a></td>
</tr>
</table></center>

<center><font face="Arial"><a href="/servlet/LogoutServlet">Sign
Out</a></font></center>

</body>
</html>
```

```
// AccountInfServlet.java
// Shows client's account information (stocks, shares, money)

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.sql.*;

import stockpg.*;          // my package

public class AccountInfServlet extends HttpServlet {
   private Statement statement = null;
   private Connection connection = null;
   private String URL = "jdbc:odbc:ASTdb";

   public void init( ServletConfig config )
      throws ServletException
   {
      super.init( config );

      try {
         Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
         connection =
         DriverManager.getConnection( URL, "", "" );
      }
      catch ( Exception e ) {
         e.printStackTrace();
         connection = null;
      }
   }

public void doGet( HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException
   {
      HttpSession session = req.getSession(false);
      String userID = (String)session.getValue(session.getId());

      if (userID != null) {
         Show show = new Show(connection, userID);
         Vector holdstocks = show.holdstocks();
         double holdmoney = show.holdmoney();

         PrintWriter out = res.getWriter();
         res.setContentType( "text/html" );

         out.println("<html><body>" +
            "<center><table width=\"60%\">
            <tr BGCOLOR=#A0B8C8><td align=center>" +
            "<b><font face=Arial size=+3>Online Stock Agent</font></b>"
            + "</td></tr></table></center><hr><br>" + "<center><h2>" +
            userID + "'s Account Details</h2></center><br>");

         if (holdstocks.isEmpty())
```

```
            out.println("<h3>You do not have any stocks.</h3>");
         else {
            ShowHoldStocks s= new ShowHoldStocks(holdstocks);
            out.println("<h3>You have stocks: </h3>" + "<center><table
                 border width=\"40%\">" + "<tr><b><th>Stock
                 Name</th><th>Shares</th></b></tr>");
            for(int i=0; i<s.stockName.length; i++)
               out.println("<tr><td>" + s.stockName[i] + "</td><td
                    ALIGN=right>" + s.shares[i] + "</td></tr>");
            out.println("</table></center><br>");
         }
         out.println("<h3>Your money balance is: </h3>" +
              "<center><b>$</b>" + holdmoney + "</center><hr>");
         // Add other links
         out.println("<center><table><tr>" + "<td><a
              href=\"http://localhost:8080/quotes.html\">Get
              Quotes</a></td>" + "<td><a
              href=\"/servlet/RecomServlet\">Recommend</a></td>" +
              "<td><a href=\"/servlet/MoneyServlet\">Deposit &
              Withdraw</a></td>" + "<td><a
              href=\"/servlet/OrderServlet\">Make an Order</a></td>" +
              "<td><a href=\"/servlet/AccountInfServlet\">Account
              Details</a></td>" + "<td><a
              href=\"/servlet/TradeHistoryServlet\">Trade
              History</a></td>" + "</tr></table><font face=Arial>" + "<a
              href=\"/servlet/LogoutServlet\">Sign
              Out</a></font></center>" );
         out.println("</body></html>");
         out.close();
      }
   }

   public void destroy()
   {
      try {
         connection.close();
      }
      catch( Exception e ) {
         System.err.println( "Problem closing the database" );
      }
   }

   public String getServletInfo()
   {
      return "The AccountInfo Servlet shows a client's account
              details.";
   }

}
```

```
// LoginServlet.java

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.sql.*;

public class LoginServlet extends HttpServlet {
   private Statement statement = null;
   private Connection connection = null;
   private String URL = "jdbc:odbc:ASTdb";

   public void init( ServletConfig config ) throws ServletException
   {
      super.init( config );
      try {
         Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
         connection =
         DriverManager.getConnection( URL, "", "" );
      }
      catch ( Exception e ) {
         e.printStackTrace();
         connection = null;
      }
   }

   public void doPost(HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException
   {
      String user, pswd;
      user = req.getParameter( "login" );
      pswd = req.getParameter( "password" );
      PrintWriter output = res.getWriter();
      res.setContentType( "text/html" );

      if ( user.equals( "" ) || pswd.equals( "" ) ) {
         output.println( "<H3> Please click the back " + "button and
               fill in User ID " + "and Password.</H3>" );
         output.close();
         return;
      }

      try {
         String query = "SELECT * FROM client where userID = '" + user
                  + "' AND password = '" + pswd + "'";
         statement = connection.createStatement();
         ResultSet rs = statement.executeQuery( query );
         boolean success = rs.next();
         if ( success ) {
            HttpSession session = req.getSession(true);
            session.putValue(session.getId(), user);
            output.println("<html><body>");
            output.println("<center><table width=\"60%\">
                  <tr BGCOLOR=#A0B8C8><td align=center>" +
                  "<b><font face=Arial size=+3>Online Stock Agent
                  </font></b>" +"</td></tr></table></center><hr><br>");
            output.println("<h3>Welcome " + user +
                  ", you can have services:<h3><br>");
            output.println("<UL>" +
                  "<LI><a href=\"http://localhost:8080/quotes.html\">
                  Get Quotes</a>" + "<LI><a
                  href=\"/servlet/RecomServlet\">Recommend</a>" +
                  "<LI><a href=\"/servlet/MoneyServlet\">
                  Deposit & Withdraw</a>" +
                  "<LI><a href=\"/servlet/OrderServlet\">
                  Make an Order</a>" +
                  "<LI><a href=\"/servlet/AccountInfServlet\">
                  Account details</a>" +
                  "<LI><a href=\"/servlet/TradeHistoryServlet\">
                  Trade History</a>" +
                  "</UL>" + "<hr><center><font face=Arial>" +
                  "<a href=\"/servlet/LogoutServlet\">
                  Sign Out</a></font></center>" );
         }
         else
            output.println("<H2>Incorrect Login, Please try again.</h2>
                  " + "<center><font size=+1><a
                  href=\"http://localhost:8080/main.html\">Login</a></f
                  ont></center><br>" + "<H2>If you are a New User,
                  Please" + "<a
                  href=\"http://localhost:8080/register.html\">REGISTER
                  </a> first.</H2>" );

         output.println("</body></html>");
         output.close();
         statement.close();
      }
      catch ( SQLException sqlex ) {
         System.err.println("ERROR: Problems with finding an entry" );
         sqlex.printStackTrace();
         return;
      }
   }

   public void destroy()
   {
      try {
         connection.close();
      }catch( Exception e )
         System.err.println( "Problem closing the database" );
   }

   public String getServletInfo() {
      return "The Login Servlet gives a login client services.";
   }

}
```

```
// LogoutServlet.java

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.sql.*;

public class LogoutServlet extends HttpServlet {

   public void doGet( HttpServletRequest req, HttpServletResponse res )
      throws ServletException, IOException
   {
      HttpSession session = req.getSession(false);
      session.invalidate();

      PrintWriter output = res.getWriter();
      res.setContentType( "text/html" );

      output.println("<html><head><title>Logout</title></head><body>" +
            "<center><table width=\"60%\">" +
            "<tr bgcolor=a0b8c8><td align=center><b><font
            face=\"Arial\" size=+3>" +
            "Online Stock Agent</font></b></td></tr></table>" +
            "<hr><br><table width=\"90%\">" +
            "<tr><td><b><I><font face=\"Arial\"size=+1>" +
            "Thank you for using Online Stock Agent!
            </font></I></b></td></tr>" +
            "<tr><td>You have been successfully logged out</td></tr>" +
            "<tr><td align=center><font face=Arial size=+1><b>" +
            "<a href=\"http://localhost:8080/main.html\">" +
            "Return to Online Stock Agent</a></b></font>" +
            "<br>You may neet to log in again.
            </td></tr></table></center>" +
            "</body></html>");
      output.close();
   }

   public String getServletInfo()
   {
      return "The Logout Servlet lets a client logout.";
   }

}
```

```
// MoneyServlet.java

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.sql.*;

import stockpg.*;

public class MoneyServlet extends HttpServlet {
   private Statement statement = null;
   private Connection connection = null;
   private String URL = "jdbc:odbc:ASTdb";

   public void init( ServletConfig config ) throws ServletException
   {
      super.init( config );

      try {
         Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
         connection = DriverManager.getConnection( URL, "", "" );
      }
      catch ( Exception e ) {
         e.printStackTrace();
         connection = null;
      }
   }

   public void doGet( HttpServletRequest req, HttpServletResponse res )
      throws ServletException, IOException
   {
      HttpSession session = req.getSession(false);
      String userID = (String)session.getValue(session.getId());
      //System.out.println("userID=" + userID);

      Show sb = new Show(connection, userID);
      double b = sb.holdmoney();

      PrintWriter out = res.getWriter();
      res.setContentType( "text/html" );

      out.println("<html><body>" + "<center><table><tr><td
           BGCOLOR=#A0B8C8><b><font face=Arial size=+3>" + "Online Stock
           Agent - </font><font size=+1>Money Manager</font></b>" +
           "</td></tr></table></center><hr><br>");

      out.println("<center><h3>Your balance is: $" + b +
           "</h3></center></br>");

      out.println("<form method=post action=\"/servlet/MoneyServlet\">"
           + "<center><table width=\"70%\">" + "<tr><td
           width=\"50%\"><b><font size=+1>Action:</font></b></td>" +
           "<td width=\"50%\"><b><font
```

```
        size=+1>Amount</font>(US$):</b></td></tr>" + "<tr><td><input
        type=\"radio\" name=\"action\" value=\"deposit\"
        checked>Deposit" + "<br><input type=\"radio\" name=\"action\"
        value=\"withdraw\">Withdraw</td>" + "<td align=center>$<input
        name=\"amount\" size=\"20\"></td></tr>" + "<tr><td
        align=center colspan=\"2\"><input type=\"submit\"
        value=\"Submit\"></td></tr>" + "</table></center></form><hr>"
        + "<center><table><tr>" + "<td><a
        href=\"http://localhost:8080/quotes.html\">Get
        Quotes</a></td>" + "<td><a
        href=\"/servlet/RecomServlet\">Recommend</a></td>" + "<td><a
        href=\"/servlet/MoneyServlet\">Deposit & Withdraw</a></td>" +
        "<td><a href=\"/servlet/OrderServlet\">Make an
        Order</a></td>" + "<td><a
        href=\"/servlet/AccountInfServlet\">Account Details</a></td>"
        + "<td><a href=\"/servlet/TradeHistoryServlet\">Trade
        History</a></td>" + "</tr></table></center>" + "<center><font
        face=Arial>" + "<a href=\"/servlet/LogoutServlet\">Sign
        Out</a></font></center>");

   out.println("</body></html>" );
   out.close();

}

public void doPost(HttpServletRequest req, HttpServletResponse res )
   throws ServletException, IOException
{
   HttpSession session = req.getSession(false);
   String userID = (String)session.getValue(session.getId());

   String action = req.getParameter("action");
   //double amount = (double)req.getParameter("amount");
   String money = req.getParameter("amount");
   double amount = Double.parseDouble(money);

   Money m = new Money(connection, userID);
   int mr = m.manager(action, amount);

   PrintWriter output = res.getWriter();
   res.setContentType( "text/html" );

   output.println("<html><body>");
   output.println("<center><table><tr><td BGCOLOR=#A0B8C8><b><font
          face=Arial size=+3>" + "Online Stock Agent - </font><font
          size=+1>Money Manager</font></b>" +
          "</td></tr></table></center><hr><br>");
   if (mr==1)
      output.println("<h3>You just " + action + " $" + amount +
            "</h3><br>");
   else
      output.println("<h3>Error happenned, Would you please try it
            later.</h3><br>");
      Show show = new Show(connection, userID);
      double balance = show.holdmoney();

      output.println("<center><h3>Your total balance is: $" + balance +
              "</h3></center><br>");

      output.println("<hr><center><table><tr>" + "<td><a
            href=\"http://localhost:8080/quotes.html\">Get
            Quotes</a></td>" + "<td><a
            href=\"/servlet/RecomServlet\">Recommend</a></td>" +
            "<td><a href=\"/servlet/MoneyServlet\">Deposit &
            Withdraw</a></td>" + "<td><a
            href=\"/servlet/OrderServlet\">Make an Order</a></td>" +
            "<td><a href=\"/servlet/AccountInfServlet\">Account
            Details</a></td>" + "<td><a
            href=\"/servlet/TradeHistoryServlet\">Trade
            History</a></td>" + "</tr></table><font face=Arial>" + "<a
            href=\"/servlet/LogoutServlet\">Sign
            Out</a></font></center>" );
      output.println("</body></html>");
      output.close();
   }

   public void destroy()
   {
      try {
         connection.close();
      }
      catch( Exception e ) {
         System.err.println( "Problem closing the database" );
      }
   }

   public String getServletInfo()
   {
      return "The Money Servlet manages money deposit and withdraw.";
   }

}
```

```
// OrderServlet.java

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.sql.*;

import stockpg.*;

public class OrderServlet extends HttpServlet {
   private Statement statement = null;
   private Connection connection = null;
   private String URL = "jdbc:odbc:ASTdb";

   public void init( ServletConfig config ) throws ServletException
   {
      super.init( config );

      try {
         Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
         connection = DriverManager.getConnection( URL, "", "" );
      }
      catch ( Exception e ) {
         e.printStackTrace();
         connection = null;
      }
   }

   public void doGet( HttpServletRequest req, HttpServletResponse res )
      throws ServletException, IOException
   {
      HttpSession session = req.getSession(false);
      String userID = (String)session.getValue(session.getId());

      Show sb = new Show(connection, userID);
      double b = sb.holdmoney();

      PrintWriter out = res.getWriter();
      res.setContentType( "text/html" );

      out.println("<html><body>" + "<center><table><tr><td
           BGCOLOR=#A0B8C8><b><font face=Arial size=+3>" + "Online Stock
           Agent - </font><font size=+1>Order Manager</font></b>" +
           "</td></tr></table></center><hr><br>");

      out.println("<h3>You have $" + b + " cash available in your
           account.</h3></br>");

      out.println("<form method=post action=\"/servlet/OrderServlet\">"
           + "<center><table width=\"70%\">" + "<tr><th><b><font
           size=+1>Action:</font></b></th>" + "<th><b><font
           size=+1>Symbol:</font></b></th>" + "<th><b><font size=+1># of
           Shares</font></b></th>" + "<th><b><font
           size=+1>Price(US$):</font></b></th></tr>" + "<tr><td><input
           type=\"radio\" name=\"action\" value=\"buy\" checked>Buy" +
           "<br><input type=\"radio\" name=\"action\"
           value=\"sell\">Sell</td>" + "<td><input name=\"symbol\"
           size=\"20\"></td>" + "<td><input name=\"shares\"
           size=\"20\"></td>" + "<td><input name=\"price\"
           size=\"20\"></td></tr>" + "<tr><td align=center
           colspan=\"4\"><input type=\"submit\"
           value=\"Submit\"></td></tr>" + "<tr><td
           colspan=\"4\"><b>Note:</b><br>If you don't hava enough cash
           to " + "buy <b># of shares </b>above, buy as many as
           possible.<br>" + "You only can sell as many as possible # of
           shares stocks that you are holding." +
           "</td></tr></table></center></form><hr>" +
           "<center><table><tr>" + "<td><a
           href=\"http://localhost:8080/quotes.html\">Get
           Quotes</a></td>" + "<td><a
           href=\"/servlet/RecomServlet\">Recommend</a></td>" + "<td><a
           href=\"/servlet/MoneyServlet\">Deposit & Withdraw</a></td>" +
           "<td><a href=\"/servlet/OrderServlet\">Make an
           Order</a></td>" + "<td><a
           href=\"/servlet/AccountInfServlet\">Account Details</a></td>"
           + "<td><a href=\"/servlet/TradeHistoryServlet\">Trade
           History</a></td>" + "</tr></table></center>" + "<center><font
           face=Arial>" + "<a href=\"/servlet/LogoutServlet\">Sign
           Out</a></font></center>");

      out.println("</body></html>" );
      out.close();
   }

   public void doPost(HttpServletRequest req, HttpServletResponse res )
      throws ServletException, IOException
   {
      HttpSession session = req.getSession(false);
      String userID = (String)session.getValue(session.getId());

      String action = req.getParameter("action");
      String symbol = req.getParameter("symbol");
      String ss = req.getParameter("shares");
      long shares = Long.parseLong(ss);
      String sprice = req.getParameter("price");
      double price = Double.parseDouble(sprice);

      Order o = new Order(connection, userID);
      String or = new String(o.orderpost(action, symbol, shares,
                             price));

      PrintWriter output = res.getWriter();
      res.setContentType( "text/html" );

      output.println("<html><body>");
      output.println("<center><table><tr><td BGCOLOR=#A0B8C8><b><font
             face=Arial size=+3>" + "Online Stock Agent - </font><font
```

```
            size=+1>Order Manager</font></b>" +
            "</td></tr></table></center><hr><br>");

      output.println("<h3>" + or + "</h3><br>");

      output.println("<hr><center><table><tr>" + "<td><a
            href=\"http://localhost:8080/quotes.html\">Get
            Quotes</a></td>" + "<td><a
            href=\"/servlet/RecomServlet\">Recommend</a></td>" +
            "<td><a href=\"/servlet/MoneyServlet\">Deposit &
            Withdraw</a></td>" + "<td><a
            href=\"/servlet/OrderServlet\">Make an Order</a></td>" +
            "<td><a href=\"/servlet/AccountInfServlet\">Account
            Details</a></td>" + "<td><a
            href=\"/servlet/TradeHistoryServlet\">Trade
            History</a></td>" + "</tr></table><font face=Arial>" + "<a
            href=\"/servlet/LogoutServlet\">Sign
            Out</a></font></center>" );
      output.println("</body></html>");
      output.close();

      if ( or.endsWith("processing.") ) {
         o.processOrder();
      }
   }

   public void destroy()
   {
      try {
         connection.close();
      }
      catch( Exception e ) {
         System.err.println( "Problem closing the database" );
      }
   }

   public String getServletInfo()
   {
         return "The Order Servlet manages buying or selling stocks.";
   }

}
```

```
// RecomServlet.java

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.sql.*;

import agent.*;

public class RecomServlet extends HttpServlet {
   private Statement statement = null;
   private Connection connection = null;
   private String URL = "jdbc:odbc:ASTdb";
   private Agent RecomAgent;

   public void init( ServletConfig config ) throws ServletException
   {
      super.init( config );

      try {
         Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
         connection = DriverManager.getConnection( URL, "", "" );
      }
      catch ( Exception e ) {
         e.printStackTrace();
         connection = null;
      }
   }

   public void doGet( HttpServletRequest req, HttpServletResponse res )
      throws ServletException, IOException
   {
      HttpSession session = req.getSession(false);
      String userID = (String)session.getValue(session.getId());

      PrintWriter out = res.getWriter();
      res.setContentType( "text/html" );

      out.println("<html><body>" + "<center><table><tr><td
           BGCOLOR=#A0B8C8><b><font face=Arial size=+3>" + "Online Stock
           Agent - </font><font size=+1>Recommend</font></b>" +
           "</td></tr></table></center><hr><br>");

      out.println("<form method=postaction=\"/servlet/RecomServlet\">"
           + "<center><SELECT NAME=\"sn\">" + "<OPTION
           VALUE=\"^DJI\">Dow" + "<OPTION VALUE=\"^IXIC\">Nasdaq" +
           "<OPTION VALUE=\"^SPC\">S&P 500"+ "<OPTION
           VALUE=\"ORCL\">Orcal" + "<OPTION
           VALUE=\"YHOO\">Yahoo</SELECT>" + "<INPUT TYPE=\"submit\"
           VALUE=\"Recommend\">" + "</center></form>" );

      out.println("<hr><center><table><tr>" + "<td><a
           href=\"http://localhost:8080/quotes.html\">Get
```

```
        Quotes</a></td>" + "<td><a
        href=\"/servlet/RecomServlet\">Recommend</a></td>" + "<td><a
        href=\"/servlet/MoneyServlet\">Deposit & Withdraw</a></td>" +
        "<td><a href=\"/servlet/OrderServlet\">Make an
        Order</a></td>" + "<td><a
        href=\"/servlet/AccountInfServlet\">Account Details</a></td>"
        + "<td><a href=\"/servlet/TradeHistoryServlet\">Trade
        History</a></td>" + "</tr></table></center>" + "<center><font
        face=Arial>" + "<a href=\"/servlet/LogoutServlet\">Sign
        Out</a></font></center>");

   out.println("</body></html>" );
   out.close();
}

public void doPost(HttpServletRequest req, HttpServletResponse res )
   throws ServletException, IOException
{
   HttpSession session = req.getSession(false);
   String userID = (String)session.getValue(session.getId());

   String symbol = req.getParameter("sn");

   RecomAgent = new Agent(connection);
   String rs = RecomAgent.perform(symbol);

   PrintWriter output = res.getWriter();
   res.setContentType( "text/html" );

   output.println("<html><body>");
   output.println("<center><table><tr><td BGCOLOR=#A0B8C8><b><font
          face=Arial size=+3>" + "Online Stock Agent - </font><font
          size=+1>Recommend</font></b>" +
          "</td></tr></table></center><hr><br>");

   output.println("<h3>Recommendation: </h3><br>" + "<center>" +
          symbol + ": " + rs + "</center><br>" + "<br><b>Note: Above
          recommendation is not responsible for your profit or
          loss.</b>");

   output.println("<hr><center><table><tr>" + "<td><a
          href=\"http://localhost:8080/quotes.html\">Get
          Quotes</a></td>" + "<td><a
          href=\"/servlet/RecomServlet\">Recommend</a></td>" +
          "<td><a href=\"/servlet/MoneyServlet\">Deposit &
          Withdraw</a></td>" + "<td><a
          href=\"/servlet/OrderServlet\">Make an Order</a></td>" +
          "<td><a href=\"/servlet/AccountInfServlet\">Account
          Details</a></td>" + "<td><a
          href=\"/servlet/TradeHistoryServlet\">Trade
          History</a></td>" + "</tr></table><font face=Arial>" + "<a
          href=\"/servlet/LogoutServlet\">Sign
          Out</a></font></center>" );
   output.println("</body></html>");
```

```
      output.close();
   }

   public void destroy()
   {
      try {
         connection.close();
      }
      catch( Exception e ) {
         System.err.println( "Problem closing the database" );
      }
   }

   public String getServletInfo()
   {
      return "The Order Servlet manages buying or selling stocks.";
   }

}
```

```
// RegisterServlet.java

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.sql.*;

public class RegisterServlet extends HttpServlet {
   private Statement statement = null;
   private Connection connection = null;
   private String URL = "jdbc:odbc:ASTdb";

   public void init( ServletConfig config ) throws ServletException
   {
      super.init( config );

      try {
         Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
         connection = DriverManager.getConnection( URL, "", "" );
      }
      catch ( Exception e ) {
         e.printStackTrace();
         connection = null;
      }
   }

   public void doPost(HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException
   {
      String userID, pswd1, pswd2, email, firstName, lastName, other;

      userID = req.getParameter( "login" );
      pswd1 = req.getParameter( "pwd1" );
      pswd2 = req.getParameter( "pwd2" );
      email = req.getParameter( "Email" );
      firstName = req.getParameter( "FirstName" );
      lastName = req.getParameter( "LastName" );
      other = req.getParameter( "other" );

      PrintWriter output = res.getWriter();
      res.setContentType( "text/html" );

      try {
         String query = "SELECT * FROM client where userID = '" +
                        userID + "'";

         statement = connection.createStatement();
         ResultSet rs = statement.executeQuery( query );
         boolean inuse = rs.next();

         if ( inuse ) {
            output.println( "<H3> User Id is already used, please reset
                   it.</H3>" );
            output.close();
            return;
         }
      }
      catch ( SQLException sqlex ) {
         System.err.println("ERROR: Problems with finding an entry" );
         sqlex.printStackTrace();
         return;
      }

      if ( ! pswd2.equals(pswd1) ) {
         output.println( "<H3> Please reset your password.</H3>" );
         output.close();
         return;
      }

      if ( email.equals( "" ) || firstName.equals( "" ) ||
           lastName.equals( "" ) ) {
         output.println( "<H3> Please click the back button and fill in
                 all fields.</H3>" );
         output.close();
         return;
      }

      boolean success = insertIntoDB( "'" + userID + "','" + pswd1 +
                        "','" + lastName + "','" + firstName + "','" +
                        email + "','" + other + "'" );

      if ( success )
         output.print( "<H2>Thank you " + firstName + " for
                 registering.</H2>" + "<br><center><font face=Arial
                 size=+1><b>" + "<a
                 href=\"http://localhost:8080/main.html\">" + "Login
                 Online Stock Agent</b></font></center>");
      else
         output.print( "<H2>An error occurred. " + "Please try again
                 later.</H2>" );

      output.close();
   }

   private boolean insertIntoDB( String stringtoinsert )
   {
      try {
         statement = connection.createStatement();
         statement.execute( "INSERT INTO client
                            (userID,password,lName,fName,email,other)
                            VALUES (" + stringtoinsert + ");" );
         statement.close();
      }
      catch ( Exception e ) {
         System.err.println( "ERROR: Problems with adding new entry" );
         e.printStackTrace();
         return false;
      }

      return true;
   }

   public void destroy()
   {
      try {
         connection.close();
      }
      catch( Exception e ) {
         System.err.println( "Problem closing the database" );
      }
   }
}
```

```
// TradeHistoryServlet.java
// show client's trade history (action,stocks,shares,paid,ordtime,deal)

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.sql.*;

import stockpg.*;          // my package

public class TradeHistoryServlet extends HttpServlet {
   private Statement statement = null;
   private Connection connection = null;
   private String URL = "jdbc:odbc:ASTdb";

   public void init( ServletConfig config ) throws ServletException
   {
      super.init( config );

      try {
         Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
         connection = DriverManager.getConnection( URL, "", "" );
      }
      catch ( Exception e ) {
         e.printStackTrace();
         connection = null;
      }
   }

   public void doGet( HttpServletRequest req, HttpServletResponse res )
      throws ServletException, IOException
   {
      HttpSession session = req.getSession(false);
      String userID = (String)session.getValue(session.getId());

      if (userID != null) {
         Show disp = new Show(connection, userID);
         Vector trade = disp.trade();

         PrintWriter out = res.getWriter();
         res.setContentType( "text/html" );
         out.println("<html><body>" + "<center><table width=\"60%\"><tr
              BGCOLOR=#A0B8C8><td align=center>" + "<b><font face=Arial
              size=+3>Online Stock Agent</font></b>" +
              "</td></tr></table></center><hr><br>" + "<center><h2>" +
              userID + "'s Trade History</h2></center><br>");

         if (trade.isEmpty())
            out.println("<h3>You do not have any trade history.</h3>");
         else {
            ShowTradeHistory s= new ShowTradeHistory(trade);
            out.println("<h3>You have trade history: </h3>" +
                 "<center><table border width=\"80%\">" +
                "<tr><b><th>No.</th><th>Action</th><th>Stock
                Name</th><th>Shares</th>" +
                "<th>Paid(US$)</th><th>Order
                Time</th><th>Deal</b></tr>");
            for (int i=0; i<s.stockName.length; i++)
               out.println("<tr><td>" + i + "</td><td>" + s.action[i] +
                   "</td><td>" + s.stockName[i] + "</td><td
                   ALIGN=right>" + s.shares[i] + "</td><td
                   ALIGN=right>" + s.paid[i] + "</td><td>" +
                   s.ordtime[i] + "</td><td>" + s.deal[i] +
                   "</td></tr>");

               out.println("</table></center><br><b>Note: </b>" + "In
                   the last column-Deal, \"Yes\" means order is
                   successful; " + "\"No\" means order is unsuccessful;
                   " + "<br>\"null\" means order is waiting." );
         }
         out.println("<hr><center><table><tr>" + "<td><a
              href=\"http://localhost:8080/quotes.html\">Get
              Quotes</a></td>" + "<td><a
              href=\"/servlet/RecomServlet\">Recommend</a></td>" +
              "<td><a href=\"/servlet/MoneyServlet\">Deposit &
              Withdraw</a></td>" + "<td><a
              href=\"/servlet/OrderServlet\">Make an Order</a></td>" +
              "<td><a href=\"/servlet/AccountInfServlet\">Account
              Details</a></td>" + "<td><a
              href=\"/servlet/TradeHistoryServlet\">Trade
              History</a></td>" + "</tr></table><font face=Arial>" + "<a
              href=\"/servlet/LogoutServlet\">Sign
              Out</a></font></center>" );
         out.println("</body></html>");
         out.close();
      }
   }

   public void destroy()
   {
      try {
         connection.close();
      }
      catch( Exception e )
         System.err.println( "Problem closing the database" );
   }

   public String getServletInfo()
   {
      return
         "The TradeHistory Servlet shows a client's trade history.";
   }
}
```

```
// Agent.java

package agent;

import java.util.*;
import java.sql.*;

public class Agent {
   public Belief B;
   public Desire D;
   public Intention I;

   public Agent(Connection x)
   {
      D = new Desire(x);
      B = new Belief(x);
      I = new Intention(x);
   }

   public String perform(String goal)
   {
      if ( D.achievable(goal) ) {
         int b = B.getBelief(goal);
         if (b==11111)
            return "Belief sensor error";
         else {
            String p = I.selectPlan(b);
            return p;
         }
      }
      else
         return "Unachievable goal";
   }

}
```

```
// Belief.java       Desire->Believes

package agent;

import java.util.*;
import java.sql.*;

public class Belief {
   private Connection connection;

   public Belief (Connection x)
   {
      this.connection = x;
   }

   public Connection getConnect() {return connection;}

   // Get believes according to the desire
   public int getBelief (String goal)
   {
      try {
         Statement st = connection.createStatement();
         String sql = "SELECT * FROM knowledge WHERE Symbol='" + goal +
                      "'";
         ResultSet rs = st.executeQuery(sql);
         boolean more = rs.next();
         if (more) {
            int sum = 0;
            ResultSetMetaData rsmd = rs.getMetaData();
            for (int i=2; i<rsmd.getColumnCount(); ++i)
            //1st element: Symbol
               sum += rs.getInt(i);

            System.out.println("sum="+sum);
            return sum;     //
         }
         else
            return 11111; //false flag
      }
      catch (SQLException sqlex) {
         sqlex.printStackTrace();
         System.out.println("Desire SQL error\n");
         return 11111; //false flag
      }
   }

}
```

```
// Desire.java

package agent;

import java.util.*;
import java.sql.*;

public class Desire {
   private Connection connection;

   public Desire (Connection x)
   {
      this.connection = x;
   }

   public boolean achievable(String goal)
   {
      try {
         Statement st = connection.createStatement();
         String sql = "SELECT Symbol FROM knowledge WHERE Symbol='" +
                      goal + "'";
         ResultSet rs = st.executeQuery(sql);
         if (rs.next()){
            System.out.println(rs.getString(1));
            return true;
         }
         else
            return false;
      }
      catch (SQLException sqlex) {
         sqlex.printStackTrace();
         System.out.println("Desire SQL error\n");
         return false;
      }
   }

}
```

```
// Intention.java

package agent;

import java.util.*;
import java.sql.*;

public class Intention {
   private Connection connection;

   public Intention(Connection x)
   {
      this.connection = x;
   }

   public String selectPlan(int bresult) {
      try {
         Statement st = connection.createStatement();
         String sql = "SELECT Plan FROM BPmap WHERE UpLimit>" + bresult
                      + " and LowLimit<=" + bresult;
         ResultSet rs = st.executeQuery(sql);

         if (rs.next()){
            String s = rs.getString(1);
            System.out.println("Intention plan1="+s);
            return s;
         }
         else
            return "fail";
      }
      catch (SQLException sqlex) {
         sqlex.printStackTrace();
         return "fail";
      }
   }

}
```

```
// Money.java

package stockpg;

import java.util.*;
import java.sql.*;

public class Money {
   private Connection connection;
   private Show show;
   private String ID;

   public Money(Connection x, String userID)
   {
      connection = x;
      ID = new String(userID);
      show = new Show(x, userID);
   }

   public int manager(String act, double bucks)
   {
      double amount = show.holdmoney();
      if ( act.equalsIgnoreCase("Deposit") )
         amount += bucks;
      else if ( act.equalsIgnoreCase("Withdraw") )
         amount -= bucks;  // who about amount<0
      else
         System.out.println("\nAction was:" + act);

      System.out.println("amount=" + amount);
      try {
         Statement st = connection.createStatement();
         String sql = "UPDATE client SET cash = " + amount + " WHERE
                       userID = '" + ID + "'";
         int rs = st.executeUpdate(sql);
         st.close();

         if (rs == 1) //System.out.println("\nUpdate successful\n");
            return 1;
         else         //System.out.println("\nUpdate failed\n");
            return -1;
      }
      catch (SQLException sqlex) {
         sqlex.printStackTrace();
         return -1;
      }
   }

}
```

```
// Order.java
// Post orders

package stockpg;
import java.util.*;
import java.sql.*;

public class Order {
   private Connection connection;
   private Show show;
   private String ID;
   private double money;
   private long num;
   private long OrdNo = 1;
   private String act;
   private String stock;
   private long shrs;
   private double price;
   private int hm;

   public Order(Connection x, String userID)
   {
      connection = x;
      ID = new String(userID);
      show = new Show(x, userID);
   }

   public String orderpost(String act, String stock, long shrs, double
price) {
      this.act = new String(act);
      this.stock = new String(stock);
      this.shrs = shrs;
      this.price = price;

      Statement st;
      money = show.holdmoney();
      num = show.checkhold(stock);

      if ( act.equalsIgnoreCase("Buy") ) {
         double total = shrs * price;
         if ( money == 0)              // no money, cannot buy
            return "No money available, can't buy any stocks.";
         if ( money < total )          // no enough money
            this.shrs = (int) ( money/price);
      }

      if ( act.equalsIgnoreCase("Sell") ) {
         if (num == -1)       // donot have this stock, cannot sell it
            return "You don't have this stock, can't sell it.";
         else
            if (num < shrs)   // donot have so many shares
               this.shrs = num;   // only can sell holding shares
      }
   try {
      st = connection.createStatement();
      String sql1 = "SELECT MAX(OrdNum) FROM orders";
      ResultSet rs1 = st.executeQuery(sql1);
      if (rs1.next())
         OrdNo = rs1.getLong(1) + 1;

      java.util.Date ud = new java.util.Date();
      hm = ud.getHours() * 60 + ud.getMinutes();

      String sql = "INSERT INTO orders (OrdNum, userID, Action,
                    Symbol, Shrs, Paid, OrdTime)" + "VALUES (" +
                    OrdNo + ",'" + ID + "', '" + act + "' ,'" + stock
                    + "', " + this.shrs + "," + price + ",'" +
                    ud.toString() + "')";
      int rs = st.executeUpdate(sql);
      if (rs == 1)
         return "Your order is posted successfully, and is
                 processing.";
      else
         return "Your order post is UNsuccessful.";
   }
   catch (SQLException sqlex) {
      sqlex.printStackTrace();
      return "Update datebase(SQL) fail.";
   }
}

public void processOrder ()
{
   int temphm;
   do {
      java.util.Date temp = new java.util.Date();
      temphm = temp.getHours() * 60 + temp.getMinutes();
   } while ((temphm-hm)< 3);      //deal time 3 minutes

   try {
      Statement st = connection.createStatement();
      String dord;
      Random r = new Random();
      if (Math.abs(r.nextInt())%2 == 0) {
         if (act.equalsIgnoreCase("Buy")) {
            String updtcash = "UPDATE client SET cash = " + (money-
                    shrs*price) + " WHERE userID='" + ID + "'";
            int rs11 = st.executeUpdate(updtcash);
            if (rs11==1)
               System.out.println("\nBuy cash update success");
            else
               System.out.println("\nBuy cash update fail");

            String updthold;
            if (num == -1)
               updthold = "INSERT INTO holding VALUES('" + ID +
                     "','"+stock+"',"+shrs+")";
            else
               updthold = "UPDATE holding SET Shrs = "+(num + shrs)+
                        " WHERE userID='" + ID + "' AND Symbol='" +
                        stock + "'";

            int rs12 = st.executeUpdate(updthold);
            if (rs12==1)
               System.out.println("\nBuy hold update success");
            else
               System.out.println("\nBuy hold update fail");
         }

         if (act.equalsIgnoreCase("Sell")) {
            String updtcash = "UPDATE client SET cash=" + (money +
                    shrs*price) + " WHERE userID='" + ID + "'";
            int rsupcash = st.executeUpdate(updtcash);
            if (rsupcash==1)
               System.out.println("\nSell cash update succecc");
            else
               System.out.println("\nSell cash update fail");

            String updthold;
            if ((num-shrs)==0)
               updthold = "DELETE FROM holding WHERE userID='" + ID
                         + "' AND Symbol='" + stock + "'";
            else
               updthold = "UPDATE holding SET Shrs=" + (num-shrs) +
                         " WHERE userID='"+ID+"' AND Symbol='" +
                         stock + "'";
            int rsuphold = st.executeUpdate(updthold);
            if (rsuphold==1)
               System.out.println("\nSell hold update success");
            else
               System.out.println("\nSell hold update fail");
         }

         dord="UPDATE orders SET Deal='Yes' WHERE OrdNum =" + OrdNo;
      }
      else
         dord="UPDATE orders SET Deal='No' WHERE OrdNum =" + OrdNo;

      int rsdord = st.executeUpdate(dord);
      if (rsdord==1)
         System.out.println("\nOrder processing successed");
      else
         System.out.println("\nOrder processing failed");
      st.close();
   }
   catch (SQLException sqlex) {
      sqlex.printStackTrace();
   }
   return ;
 }
}
```

```
// Show.java
// Show client's trade history and holding

package stockpg;
import java.util.*;
import java.sql.*;
import java.text.SimpleDateFormat;

public class Show {
   private Connection connection = null;
   private String ID = null;

   public Show(Connection x, String userID)
   {
      connection = x;
      ID = new String(userID);
   }

   // A client trade history
   public Vector trade()
   {
      Statement st;
      ResultSet rs;
      try {
         st = connection.createStatement();
         String sql = "SELECT Action, Symbol, Shrs, Paid, OrdTime, Deal
                       FROM orders WHERE userID ='" + ID + "'";
         rs = st.executeQuery(sql);
         return displayResult(rs);
      } catch (SQLException sqlex) {
         sqlex.printStackTrace();
         System.out.println("Show.trade() error"); //test error
         Vector dt = new Vector();
         return dt;
      }
   }

   // A client holds what and how many stocks
   public Vector holdstocks()
   {
      Statement st;
      ResultSet rs;
      try {
         st = connection.createStatement();
         String sql = "SELECT Symbol, Shrs FROM holding WHERE userID
                       ='" + ID + "'";
         rs = st.executeQuery(sql);
         return displayResult(rs);
      } catch (SQLException sqlex) {
         sqlex.printStackTrace();
         Vector dt = new Vector();
         return dt;
      }
   }
```

```
// Check a client has how many shares of a certain shock
public long checkhold(String stock)
{
   try {
      Statement st = connection.createStatement();
      String sql = "SELECT Shrs FROM holding WHERE userID='" + ID +
                   "' AND Symbol='" + stock + "'";
      ResultSet rs = st.executeQuery(sql);
      if (rs.next())
         return rs.getLong(1);
      else
         return -1;        // donot have this stock
   }
   catch (SQLException sqlex) {
      sqlex.printStackTrace();
      return -1;            // sql error
   }
}

// A client has how much money
public double holdmoney()
{
   Statement st;
   ResultSet rs;
   try {
      st = connection.createStatement();
      String sql="SELECT cash FROM client WHERE userID ='"+ID+"'";
      rs = st.executeQuery(sql);
      rs.next();
      return rs.getDouble(1);   // may return 0
   }
   catch (SQLException sqlex) {
      sqlex.printStackTrace();
      return -1;            // sql error
   }
}

private Vector displayResult( ResultSet rs) throws SQLException
{
   boolean more = rs.next();
   ResultSetMetaData rsmd = rs.getMetaData();
   Vector rows = new Vector();
   if (!more) {
      System.out.println("No trade history");
      return (rows);
   }

   do {
      rows.addElement( getNextRow(rs,rsmd) );
   } while ( rs.next() );

   return rows;
}
```

```
   private Vector getNextRow( ResultSet rs, ResultSetMetaData rsmd )
      throws SQLException
   {
      Vector curRow = new Vector();

      for (int i=1; i<=rsmd.getColumnCount(); ++i)
         switch( rsmd.getColumnType(i) ) {
            case Types.VARCHAR:
                 curRow.addElement(rs.getString(i));
                 break;
            case Types.INTEGER:
                 curRow.addElement( new Long(rs.getLong(i)) );
                 break;
            case Types.DOUBLE:
                 curRow.addElement( new Double(rs.getDouble(i)) );
                 break;
            default:       //DATETIME
                 System.out.println("Type was: " +
                                   rsmd.getColumnTypeName(i));

            curRow.addElement( rs.getDate(i));
         }

      return curRow;
   }

   //Show current stock price and other information
   public Vector getCurInf(String stock)
   {
      Statement st;
      try {
         st = connection.createStatement();
         java.util.Date ud = new java.util.Date();
         SimpleDateFormat formatter = new SimpleDateFormat ("M/dd/yy");
         // Here is a bug when month>9
         String dateString = formatter.format(ud);
         System.out.println("format time:" + dateString);

         String sql = "SELECT * FROM quotes WHERE Symbol='" + stock +
                      "' AND Date LIKE '" + dateString + "'";

         System.out.println("\nsql: " + sql);
         ResultSet rs = st.executeQuery(sql);
         return displayResult(rs);
      }
      catch (SQLException sqlex) {
         sqlex.printStackTrace();
         Vector dt = new Vector();
         return dt;
      }
   }

}
```

```
// ShowHoldStocks.java
// Shows holding stock record (stockName, shares)

package stockpg;
import java.util.*;

public class ShowHoldStocks {
   public String stockName[];
   public int shares[];

   public ShowHoldStocks(Vector v)
   {
      int SIZE = v.size();
      System.out.println("number of records is: " + SIZE);
      stockName = new String[SIZE];
      shares = new int[SIZE];

      for (int n=0; n<v.size(); ++n) {
         Vector r = new Vector();
         r.addElement(v.elementAt(n));
         Enumeration enum = r.elements();
         StringBuffer buf = new StringBuffer();

         while (enum.hasMoreElements()) {
            buf.append(enum.nextElement());
            //System.out.println("buf="+buf.toString());
            buf.deleteCharAt(0);
            buf.deleteCharAt(buf.length()-1);
            //System.out.println("bufnew="+buf.toString());
            String str = new String(buf);
            StringTokenizer strtok = new StringTokenizer(str, ",");

            stockName[n] = strtok.nextToken();
            StringBuffer temp = new StringBuffer();
            temp.append(strtok.nextToken());
            temp.delete(0,1);
            shares[n] = Integer.parseInt(temp.toString());
         }
      }
   }

}
```

```
// ShowTradeHistory.java
// Shows client's trade history (Action,Symbol,Shrs,Paid,OrdTime,Deal)

package stockpg;
import java.util.*;

public class ShowTradeHistory {
   public String action[];
   public String stockName[];
   public int shares[];
   public double paid[];
   public String ordtime[];
   public String deal[];

   public ShowTradeHistory(Vector v)
   {
      int SIZE = v.size();
      System.out.println("number of records is: " + SIZE);
      action = new String[SIZE];
      stockName = new String[SIZE];
      shares = new int[SIZE];
      paid = new double[SIZE];
      ordtime = new String[SIZE];
      deal = new String[SIZE];

      for (int n=0; n<v.size(); ++n) {
         Vector r = new Vector();
         r.addElement(v.elementAt(n));
         Enumeration enum = r.elements();
         StringBuffer buf = new StringBuffer();
         while (enum.hasMoreElements()) {
            buf.append(enum.nextElement());
            buf.deleteCharAt(0);
            buf.deleteCharAt(buf.length()-1);
            String str = new String(buf);
            StringTokenizer strtok = new StringTokenizer(str, ",");
            action[n] = strtok.nextToken();
            stockName[n] = strtok.nextToken();

            StringBuffer sb1 = new StringBuffer();
            sb1.append(strtok.nextToken());
            sb1.delete(0,1);
            shares[n] = Integer.parseInt(sb1.toString());
            StringBuffer sb2 = new StringBuffer();
            sb2.append(strtok.nextToken());
            sb2.delete(0,1);
            paid[n] = Double.parseDouble(sb2.toString());
            ordtime[n] = strtok.nextToken();
            deal[n] = strtok.nextToken();
         }
      }
   }

}
```