OBJECT ORIENTED ANALYSIS & DESIGN

For

Final year Students

20, Feb 2005 – ISO 9000 Tech Document

© Author JNEC, Aurangabad
FORWARD

It is my great pleasure to present this laboratory manual for final year engineering students for the subject of Object Oriented analysis & Design keeping in view the vast coverage required for presentation using Unified Modeling Language.

As a student, many of you may be wondering with some of the questions in your mind regarding the subject and exactly what has been tried is to answer through this manual.

As you may be aware that MGM has already been awarded with ISO 9000 certification and it is our endure to technically equip our students taking the advantage of the procedural aspects of ISO 9000 Certification.

Faculty members are also advised that covering these aspects in initial stage itself, will greatly relived them in future as much of the load will be taken care by the enthusiasm energies of the students once they are conceptually clear.

Prof. S.D.Deshmukh.
Principal
Vision of JNEC

College seeks to be the engineering college of choice in Maharashtra that can provide the best learning experience, the most productive learning community, and the most creative learning environment in Engineering Education and will be recognized as one of the best Engineering Colleges in India.

Mission of JNEC

To develop innovative engineers with human values, well equipped to solve complex technical problems, address the needs of modern society and pursue lifelong learning, by providing them competent, caring and committed faculty.

IT Vision:

IT department is committed to ensure the quality education to students’ by providing innovative resources & continuous up-gradation of the department. To achieve “Heights of Excellence” in the world we strive to organize regular interaction with Industry and Alumni.

IT Mission:

To impart core technical competency & knowledge in students through curriculum and certification programs to fulfill the industry requirements which ultimately benefits society at large.

Program Educational Objectives:

I. Preparation: To prepare students to excel in PG program or to succeed in Industry /Technical profession through global, rigorous education.

II. Core Competence: To provide students with a solid foundation in mathematical, scientific and engineering fundamentals required to solve engineering problems and also to pursue higher studies.

III. Breadth: To train students with good scientific and engineering breadth so as to comprehend, analyze, design and create novel product and solution for the real life problems.

IV. Professionalism: To inculcate in students’ professional and ethical attitude, effective communication skills, team work skills, multi-disciplinary approach and an ability to relate engineering issues to broader social context.

V. Learning Environment: To provide students with academic environment aware of excellence, leadership, written ethical codes and guidelines and lifelong learning needed for successful professional career.
This manual is intended for the final year students of Information Technology in the subject of Object Oriented analysis & Design. This manual typically contains practical/Lab Sessions related Object Oriented analysis & Design covering various aspects related to enhanced understanding.

Although, as per the syllabus, only study of presentations are prescribed, we have made the efforts to cover various aspects of Object Oriented analysis & Design for designing ATM

Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.

Good Luck for your Enjoyable Laboratory Sessions

S.N.Bhasme
SUBJECT INDEX

1. Case Study on  
   **ATM Simulation**

2. Quiz on Subject

3. Conduction of Viva-Voce Examinations

4. Submission

5. Evaluation and marking system
DOs and DON'T DOs in Laboratory:

1. Do not handle any equipment before reading the instructions/Instruction manuals

2. Read carefully the power ratings of the equipment before it is switched on whether ratings 230 V/50 Hz or 115V/60 Hz. For Indian equipments, the power ratings are normally 230V/50Hz. If you have equipment with 115/60 Hz ratings, do not insert power plug, as our normal supply is 230V/50 Hz, which will damage the equipment.

3. Observe type of sockets of equipment power to avoid mechanical damage

4. Do not forcefully place connectors to avoid the damage

5. Strictly observe the instructions given by the teacher/Lab Instructor

Instruction for Laboratory Teachers:

1. Submission related to whatever lab work has been completed should be done during the next lab session. The immediate arrangements for printouts related to submission on the day of practical assignments.

2. Students should be taught for taking the printouts under the observation of lab teacher.

3. The promptness of submission should be encouraged by way of marking and evaluation patterns that will benefit the sincere students.
ATM Simulation

Introduction

Requirements

Analysis

[ Use Cases ]
[ Initial Functional Tests ]
[ Analysis Classes ]

Design

[ CRC Cards ]
[ Class Diagram ]
[ State Charts ]
[ Interaction Diagrams ]

Detailed Design and Implementation

[ Detailed Design ]
[ Package Diagram ]
[ Code ]

Testing

[ Executable Applet ]

Maintenance

[ Maintenance Ideas ]
Introduction

What you will Find Here

This page is the starting point into a series of pages that attempt to give a complete example of object-oriented analysis, design, and programming applied to a moderate size problem: the simulation of an Automated Teller Machine. I developed these pages in the belief that students would benefit from seeing a complete example of OO methodology applied to a single problem. Since then, I have developed a similar-style solution to an even simpler problem: maintaining a very simple address book.

Beginning with a statement of requirements, the process proceeds through analysis, overall design, and detailed design and implementation, culminating with some suggestions for maintenance. (Testing is left as an exercise to the reader :-).)

Analysis is done by identifying the use cases and detailing a flow of events for each. Also, an initial set of functional test cases is specified, to serve as a vehicle for checking that the implementation is complete and basically correct. Analysis culminates in identifying classes implied by the use cases, and documenting them using an Analysis Class Diagram. (The Statechart diagrams done under design are also, in part, analysis tasks. In OO, the line between analysis and design is not always a sharp one.)

Overall design begins by using CRC cards to assign responsibilities to the various classes. The static structure of the design is summarized by means of an overall Class Diagram. Then the dynamic aspects of the design are developed, using State Charts for the major controller classes, plus an Interaction Diagram for each of the main use cases.

The detailed design is developed by spelling out the attributes and methods for each class, using a class diagram for each class with all three "compartments" fully filled in. A package diagram is used to show how the various classes are grouped into packages. Each class is then implemented in Java. The code page contains links both to Javadoc documentation for each class, and to the complete source code. Also included are a main class, which allows the simulation to run as an application, and an Applet class, which allows it to be run as an Applet. (This illustrates how the same basic code can be designed to be used either way; the application was used for most of the initial development work, but the Applet is accessible to anyone over the web).

The Executable Applet version can be loaded and executed from within any web browser that supports at least JDK 1.1.x.

The Maintenance page discusses ideas for possible changes to the system, each of which would require changes to multiple documents, not just the code.

Background
This is the second significant revision of this series of pages.

1. I originally developed this series of pages for use in a course which was taught to junior CS majors at Gordon College from 1996 to 1999, using C++. (That version is available here - from which you can follow a link to an even older version if you wish.)

2. The second version represents a major revision (done during 2000 and 2001) for use in "Object-Oriented Software Development", which is taught to sophomore CS majors and uses Java. The most significant change in the second and subsequent versions is the use of UML notation throughout, together with a reorganization of the process to more closely follow the UML approach. Moreover, the original version was implemented in both C++ and Java; but subsequent versions uses only Java, so as to be able to utilize some features of that language (e.g. threads) which have no direct analog in the C++ language per se. That version is available here.

3. I more recently (in 2002) revised the sequencing of the pages to reflect a slightly different ordering of the steps in the software development process as I am now teaching it in 2002. (The content of the current set of pages is almost exactly the same as the 2000-2001 version, but the order is different.)

The original version followed the design methodology taught in the textbook I used in the earlier course - [Horstmann, 1997], and some aspects of that approach remain. For second and subsequent versions, I have used a simplified version of the process outlined in [Jacobson, Booch, and Rumbaugh, 1999].

It may be argued that more diagrams have been used than are really necessary for a project of this size. I wanted to give as complete an example of using the various UML diagrams as possible. Even so, this example does not include any activity, component, or deployment diagrams.

**Using these Pages**

Probably the best way to start using these pages is to first read the requirements, and then work through the entire analysis, design, and implementation process.
1. Begin with the Statement of Requirements and then view the Use Cases. The Use Case document has a Use Case Diagram and a series of flows of events, one for each use case. The analysis phase can be studied both in terms of the "big picture" and in terms of the details of the use cases.
   - Before studying the flows of events in detail, it might be helpful to look at the diagram showing the Analysis classes to get an overall view of the system implicit in the use cases. The Analysis Class Diagram captures the basic class structure implied by the use case flows of events.
   - Then it is time to study the flows of events in detail.

2. Having looked at the requirements/analysis documents, you can then study the design phase by viewing the CRC cards, the overall Class Diagram, and the Statechart and Interaction Diagrams.
   - There is one CRC card for each class, incorporating all responsibilities of the class arising from all of the use cases, linked to from the icon for the class in either Class Diagram. The CRC cards were created by "walking through" each use case, assigning the responsibility for each task to some class. Note that there are CRC cards for classes that do not appear in the Analysis Class Diagram; the need for these classes became apparent as the other cards were being created. (This is typical; Booch suggests that the ratio of classes in a complete application to analysis classes may be as much as 5:1; in this particular case, it is about 2:1.)
   - The Class Diagram pulls together all of the structural information from the CRC Cards and shows the needed links between classes. Each class icon is linked to other relevant design documents, including a detailed design for that class.
   - For the major controller classes, there is also a Statechart diagram depicting the various states the corresponding object can be in. This, too, is accessible from the class's icon in either Class Diagram.
   - Finally, there is one Interaction diagram for each use case, linked to from the use case flow of events. This shows the objects that work together to realize the use case, and the flow of messages between them. This records the same assignment of responsibilities as in the CRC cards, but in a different way (organized by use case, rather than by class), with additional detail. (These, rather than the CRC cards, were actually used when creating the detailed design. CRC cards are not actually a UML diagram, but are a useful tool to help in discovering the design that does get recorded in UML Interaction diagrams.)

   In terms of the order in which these would be created, the Analysis Class Diagram and the CRC cards would be done first, then the overall Class Diagram, and finally the Statechart and Interaction Diagrams. Thus, to understand the design process, these could be read in the order they were created. However, in terms of understanding the design itself, it may be desirable to read the Class Diagram first.

3. The detailed design document was developed directly from the Interaction diagrams and Class Diagram. The former determined what methods the class would need, and both the former and the latter determined what attributes were needed.
In terms of actually writing the code, the skeletons for the various classes were created directly from the detailed design. The individual method bodies were fleshed out by using the Statechart and Interaction diagrams. Each message on the Interaction diagram becomes a line or so of code in the implementation. The controller classes for which a statechart was done have a method that is structured as a finite state machine based on the statechart (run() in ATM; performSession() in Session; performTransaction() in Transaction.)

Although this web site shows a finished product in which the various design documents and the code are in agreement, the reality is that the coding process demonstrated the need for slight changes to the various design documents, so in some cases the design documents were modified to conform to the code, rather than vice versa. (However, the bulk of the design did exist before the code was written! The idea is to use the design documents to guide coding, not to produce design documents after the fact!)

4. Under the Quality Assurance heading, there is a link to an executable (applet) form of the Java implementation, which you can run if you have a JDK 1.1.x (or later) compliant browser. Enjoy!

5. Finally, under the Maintenance heading, there is a link to a page of suggested changes. One such change deals with implementing a stated requirement that is missing from the design and implementation presented here. (This would never happen in real life, of course :-)) Can you find the missed requirement? Other changes have to do with adding additional features to the system. These make interesting exercises for the reader.

**Requirements Statement for Example ATM System**

The software to be designed will control a simulated automated teller machine (ATM) having a magnetic stripe reader for reading an ATM card, a customer console (keyboard and display) for interaction with the customer, a slot for depositing envelopes, a dispenser for cash (in multiples of $20), a printer for printing customer receipts, and a key-operated switch to allow an operator to start or stop the machine. The ATM will communicate with the bank's computer over an appropriate communication link. (The software on the latter is not part of the requirements for this problem.)
The ATM will service one customer at a time. A customer will be required to insert an ATM card and enter a personal identification number (PIN) - both of which will be sent to the bank for validation as part of each transaction. The customer will then be able to perform one or more transactions. The card will be retained in the machine until the customer indicates that he/she desires no further transactions, at which point it will be returned - except as noted below.

The ATM must be able to provide the following services to the customer:

1. A customer must be able to make a cash withdrawal from any suitable account linked to the card, in multiples of $20.00. Approval must be obtained from the bank before cash is dispensed.
2. A customer must be able to make a deposit to any account linked to the card, consisting of cash and/or checks in an envelope. The customer will enter the amount of the deposit into the ATM, subject to manual verification when the envelope is removed from the machine by an operator. Approval must be obtained from the bank before physically accepting the envelope.
3. A customer must be able to make a transfer of money between any two accounts linked to the card.
4. A customer must be able to make a balance inquiry of any account linked to the card.

A customer must be able to abort a transaction in progress by pressing the Cancel key instead of responding to a request from the machine.

The ATM will communicate each transaction to the bank and obtain verification that it was allowed by the bank. Ordinarily, a transaction will be considered complete by the bank once it has been approved. In the case of a deposit, a second message will be sent to the bank indicating that the customer has deposited the envelope. (If the customer fails to deposit the envelope within the timeout period, or presses cancel instead, no second message will be sent to the bank and the deposit will not be credited to the customer.)

If the bank determines that the customer's PIN is invalid, the customer will be required to re-enter the PIN before a transaction can proceed. If the customer is unable to successfully enter the PIN after three tries, the card will be permanently retained by the machine, and the customer will have to contact the bank to get it back.

If a transaction fails for any reason other than an invalid PIN, the ATM will display an explanation of the problem, and will then ask the customer whether he/she wants to do another transaction.

The ATM will provide the customer with a printed receipt for each successful transaction, showing the date, time, machine location, type of transaction, account(s), amount, and ending and available balance(s) of the affected account ("to" account for transfers).

The ATM will have a key-operated switch that will allow an operator to start and stop the servicing of customers. After turning the switch to the "on" position, the operator will be required to verify and enter the total cash on hand. The machine can only be turned off when it is not servicing a customer. When the switch is moved to the "off" position, the machine will shut down, so that the operator may remove deposit envelopes and reload the machine with cash, blank receipts, etc.
The ATM will also maintain an internal log of transactions to facilitate resolving ambiguities arising from a hardware failure in the middle of a transaction. Entries will be made in the log when the ATM is started up and shut down, for each message sent to the Bank (along with the response back, if one is expected), for the dispensing of cash, and for the receiving of an envelope. Log entries may contain card numbers and dollar amounts, but for security will *never* contain a PIN.

**Analysis**

*Use Cases for Example ATM System*
Flows of Events for Individual Use Cases

**System Startup Use Case**

The system is started up when the operator turns the operator switch to the "on" position. The operator will be asked to enter the amount of money currently in the cash dispenser, and a connection to the bank will be established. Then the servicing of customers can begin.
System Shutdown Use Case

The system is shut down when the operator makes sure that no customer is using the machine, and then turns the operator switch to the "off" position. The connection to the bank will be shut down. Then the operator is free to remove deposited envelopes, replenish cash and paper, etc.
**Session Use Case**

A session is started when a customer inserts an ATM card into the card reader slot of the machine. The ATM pulls the card into the machine and reads it. (If the reader cannot read the card due to improper insertion or a damaged stripe, the card is ejected, an error screen is displayed, and the session is aborted.) The customer is asked to enter his/her PIN, and is then allowed to perform one or more transactions, choosing from a menu of possible types of transaction in each case. After each transaction, the customer is asked whether he/she would like to perform another. When the customer is through performing transactions, the card is ejected.
from the machine and the session ends. If a transaction is aborted due to too many invalid PIN entries, the session is also aborted, with the card being retained in the machine.

The customer may abort the session by pressing the Cancel key when entering a PIN or choosing a transaction type.

Transaction Use Case

Note: Transaction is an abstract generalization. Each specific concrete type of transaction implements certain operations in the appropriate way. The flow of events given here describes the behavior common to all types of transaction. The flows of events for the individual types of transaction (withdrawal, deposit, transfer, inquiry) give the features that are specific to that type of transaction.
A transaction use case is started within a session when the customer chooses a transaction type from a menu of options. The customer will be asked to furnish appropriate details (e.g. account(s) involved, amount). The transaction will then be sent to the bank, along with information from the customer’s card and the PIN the customer entered.

If the bank approves the transaction, any steps needed to complete the transaction (e.g. dispensing cash or accepting an envelope) will be performed, and then a receipt will be printed. Then the customer will be asked whether he/she wishes to do another transaction.

If the bank reports that the customer’s PIN is invalid, the Invalid PIN extension will be performed and then an attempt will be made to continue the transaction. If the customer’s card is retained due to too many invalid PINs, the transaction will be aborted, and the customer will not be offered the option of doing another.

If a transaction is cancelled by the customer, or fails for any reason other than repeated entries of an invalid PIN, a screen will be displayed informing the customer of the reason for the failure of the transaction, and then the customer will be offered the opportunity to do another.

The customer may cancel a transaction by pressing the Cancel key as described for each individual type of transaction below.

All messages to the bank and responses back are recorded in the ATM’s log.

[ Interaction Diagram ]

Transaction Sequence Diagram
Withdrawal Transaction Use Case

A withdrawal transaction asks the customer to choose a type of account to withdraw from (e.g. checking) from a menu of possible accounts, and to choose a dollar amount from a menu of possible amounts. The system verifies that it has sufficient money on hand to satisfy the request before sending the transaction to the bank. (If not, the customer is informed and asked to enter a different amount.) If the transaction is approved by the bank, the appropriate amount of cash is dispensed by the machine before it issues a receipt. (The dispensing of cash is also recorded in the ATM's log.)

A withdrawal transaction can be cancelled by the customer pressing the Cancel key any time prior to choosing the dollar amount.
Deposit Transaction Use Case

A deposit transaction asks the customer to choose a type of account to deposit to (e.g. checking) from a menu of possible accounts, and to type in a dollar amount on the keyboard. The transaction is initially sent to the bank to verify that the ATM can accept a deposit from this customer to this account. If the transaction is approved, the machine accepts an envelope from the customer containing cash and/or checks before it issues a receipt. Once the envelope has been received, a second message is sent to the bank, to confirm that the bank can credit the customer's account - contingent on manual verification of the deposit envelope contents by an operator later. (The receipt of an envelope is also recorded in the ATM's log.)

A deposit transaction can be cancelled by the customer pressing the Cancel key any time prior to inserting the envelope containing the deposit. The transaction is automatically cancelled if the customer fails to insert the envelope containing the deposit within a reasonable period of time.
After being asked to do so.

**Deposit Transaction Collaboration**

1.1: \( \tau : \text{readMenuChoice(} \text{"Account to deposit to"}, \text{availableAccounts menu)} \)

1.2: \( \text{amount = readMenuChoice(} \text{"Amount to deposit"}) \)

1: \( \text{message = getSpecificationsFromCustomer()} \)

2: \( \text{receipt = completeTransaction()} \)

- «self»

- 2.1 acceptEnvelope()

- 1.3 «create»

- 2.2 «create»

- 2.3 send(message, balance)

- 2.4 «create»

**Diagram:**

- CustomerConsole
- EnvelopeAccepter
- Deposit
- Message
- NetworkToBank
- Receipt
Transfer Transaction Use Case

A transfer transaction asks the customer to choose a type of account to transfer from (e.g. checking) from a menu of possible accounts, to choose a different account to transfer to, and to type in a dollar amount on the keyboard. No further action is required once the transaction is approved by the bank before printing the receipt.

A transfer transaction can be cancelled by the customer pressing the Cancel key any time prior to entering a dollar amount.

Transfer Transaction Collaboration

1. message = getSpecificationsFromCustomer()
2. receipt = completeTransaction()

[ Interaction Diagram ]
**Inquiry Transaction Use Case**

An inquiry transaction asks the customer to choose a type of account to inquire about from a menu of possible accounts. No further action is required once the transaction is approved by the bank before printing the receipt.

An inquiry transaction can be cancelled by the customer pressing the Cancel key any time prior to choosing the account to inquire about.

**Inquiry Transaction Collaboration**

```
1: message = getSpecificsFromCustomer()
2: receipt = completeTransaction()
```

[Interaction Diagram]
Invalid PIN Extension

An invalid PIN extension is started from within a transaction when the bank reports that the customer's transaction is disapproved due to an invalid PIN. The customer is required to re-enter the PIN and the original request is sent to the bank again. If the bank now approves the transaction, or disapproves it for some other reason, the original use case is continued; otherwise the process of re-entering the PIN is repeated. Once the PIN is successfully re-entered, it is used for both the current transaction and all subsequent transactions in the session. If the customer fails three times to enter the correct PIN, the card is permanently retained, a screen is displayed informing the customer of this and suggesting he/she contact the bank, and the entire customer session is aborted. If the customer presses Cancel instead of re-entering a PIN, the original transaction is cancelled.

[ Interaction Diagram ]
Initial Functional Test Cases for Example ATM System

The following initial test cases can be identified early in the design process as a vehicle for checking that the implementation is basically correct. *No attempt has been made at this point to do thorough testing, including all possible errors and boundary cases.* That needs to come later. These cases represent an initial check that the functionality specified by the use cases is present.

Some writers would argue for developing test cases like these *in place of* use cases. Here, they are presented as a vehicle for "fleshing out" the use cases, not as a substitute for them.

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Function Being Tested</th>
<th>Initial System State</th>
<th>Input</th>
<th>Expected Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Startup</td>
<td>System is started when the switch is turned &quot;on&quot;</td>
<td>System is off</td>
<td>Activate the &quot;on&quot; switch</td>
<td>System requests initial cash amount</td>
</tr>
<tr>
<td>System Startup</td>
<td>System accepts initial cash amount</td>
<td>System is requesting cash amount</td>
<td>Enter a legitimate amount</td>
<td>System is on</td>
</tr>
<tr>
<td>System Startup</td>
<td>Connection to the bank is established</td>
<td>System has just been turned on</td>
<td>Perform a legitimate inquiry transaction</td>
<td>System output should demonstrate that a connection has been established to the Bank</td>
</tr>
<tr>
<td>System Shutdown</td>
<td>System is shut down when the switch is turned &quot;off&quot;</td>
<td>System is on and not servicing a customer</td>
<td>Activate the &quot;off&quot; switch</td>
<td>System is off</td>
</tr>
<tr>
<td>System Shutdown</td>
<td>Connection to the Bank is terminated when the system is shut down</td>
<td>System has just been shut down</td>
<td>Verify from the bank side that a connection to the ATM no longer exists</td>
<td></td>
</tr>
<tr>
<td>Session</td>
<td>System reads a customer's ATM</td>
<td>System is on and not servicing a customer</td>
<td>Insert a readable card</td>
<td>Card is accepted; System asks for entry</td>
</tr>
<tr>
<td>Use Case</td>
<td>Function Being Tested</td>
<td>Initial System State</td>
<td>Input</td>
<td>Expected Output</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------</td>
<td>---------------------------------------</td>
<td>------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>System Startup</td>
<td>System is started when the switch is turned &quot;on&quot;</td>
<td>System is off</td>
<td>Activate the &quot;on&quot; switch</td>
<td>System requests initial cash amount</td>
</tr>
<tr>
<td>Session</td>
<td>System rejects an unreadable card</td>
<td>System is on and not servicing a customer</td>
<td>Insert an unreadable card</td>
<td>Card is ejected; System displays an error screen; System is ready to start a new session</td>
</tr>
<tr>
<td>Session</td>
<td>System accepts customer's PIN</td>
<td>System is asking for entry of PIN</td>
<td>Enter a PIN</td>
<td>System displays a menu of transaction types</td>
</tr>
<tr>
<td>Session</td>
<td>System allows customer to perform a transaction</td>
<td>System is displaying menu of transaction types</td>
<td>Perform a transaction</td>
<td>System asks whether customer wants another transaction</td>
</tr>
<tr>
<td>Session</td>
<td>System allows multiple transactions in one session</td>
<td>System is asking whether customer wants another transaction</td>
<td>Answer yes</td>
<td>System displays a menu of transaction types</td>
</tr>
<tr>
<td>Session</td>
<td>Session ends when customer chooses not to do another transaction</td>
<td>System is asking whether customer wants another transaction</td>
<td>Answer no</td>
<td>System ejects card and is ready to start a new session</td>
</tr>
<tr>
<td>Transaction</td>
<td>Individual types of transaction will be tested below</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transaction</td>
<td>System handles an invalid PIN properly</td>
<td>A readable card has been entered</td>
<td>Enter an incorrect PIN and then attempt a transaction</td>
<td>The Invalid PIN Extension is performed</td>
</tr>
<tr>
<td>Withdrawal</td>
<td>System asks customer to choose an account to withdraw from</td>
<td>Menu of transaction types is being displayed</td>
<td>Choose Withdrawal transaction</td>
<td>System displays a menu of account types</td>
</tr>
<tr>
<td>Withdrawal</td>
<td>System asks customer to choose a dollar amount to withdraw</td>
<td>Menu of account types is being displayed</td>
<td>Choose checking account</td>
<td>System displays a menu of possible withdrawal amounts</td>
</tr>
<tr>
<td>Withdrawal</td>
<td>System performs a legitimate withdrawal</td>
<td>System is displaying the menu of</td>
<td>Choose an amount that the system currently has and</td>
<td>System dispenses this amount of cash; System prints a</td>
</tr>
<tr>
<td>Use Case</td>
<td>Function Being Tested</td>
<td>Initial System State</td>
<td>Input</td>
<td>Expected Output</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------</td>
<td>----------------------------</td>
<td>--------------------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>System Startup</td>
<td>System is started when the switch is turned &quot;on&quot;</td>
<td>System is off</td>
<td>Activate the &quot;on&quot; switch</td>
<td>System requests initial cash amount</td>
</tr>
<tr>
<td></td>
<td>transaction properly</td>
<td>withdrawal amounts</td>
<td>which is not greater than the account balance</td>
<td>correct receipt showing amount and correct updated balance; System records transaction correctly in the log (showing both message to the bank and approval back)</td>
</tr>
<tr>
<td>Withdrawal</td>
<td>System verifies that it has sufficient cash on hand to fulfill the request</td>
<td>System has been started up with less than the maximum withdrawal amount in cash on hand; System is requesting a withdrawal amount</td>
<td>Choose an amount greater than what the system currently has</td>
<td>System displays an appropriate message and asks customer to choose a different amount</td>
</tr>
<tr>
<td>Withdrawal</td>
<td>System verifies that customer's balance is sufficient to fulfill the request</td>
<td>System is requesting a withdrawal amount</td>
<td>Choose an amount that the system currently has but which is greater than the account balance</td>
<td>System displays an appropriate message and offers customer the option of choosing to do another transaction or not.</td>
</tr>
<tr>
<td>Withdrawal</td>
<td>A withdrawal transaction can be cancelled by the customer any time prior to choosing the dollar amount</td>
<td>System is displaying menu of account types</td>
<td>Press &quot;Cancel&quot; key</td>
<td>System displays an appropriate message and offers customer the option of choosing to do another transaction or not.</td>
</tr>
<tr>
<td>Withdrawal</td>
<td>A withdrawal transaction can be cancelled by the customer any time prior to choosing the dollar amount</td>
<td>System is displaying menu of dollar amounts</td>
<td>Press &quot;Cancel&quot; key</td>
<td>System displays an appropriate message and offers customer the option of choosing to do another transaction or not.</td>
</tr>
<tr>
<td>Use Case</td>
<td><strong>Function Being Tested</strong></td>
<td><strong>Initial System State</strong></td>
<td><strong>Input</strong></td>
<td><strong>Expected Output</strong></td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------------------------</td>
<td>--------------------------</td>
<td>---------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>System Startup</td>
<td>System is started when the switch is turned &quot;on&quot;</td>
<td>System is off</td>
<td>Activate the &quot;on&quot; switch</td>
<td>System requests initial cash amount not.</td>
</tr>
<tr>
<td>Deposit</td>
<td>System asks customer to choose an account to deposit to</td>
<td>Menu of transaction types is being displayed</td>
<td>Choose Deposit transaction</td>
<td>System displays a menu of account types</td>
</tr>
<tr>
<td>Deposit</td>
<td>System asks customer to enter a dollar amount to deposit</td>
<td>Menu of account types is being displayed</td>
<td>Choose checking account</td>
<td>System displays a request for the customer to type a dollar amount</td>
</tr>
<tr>
<td>Deposit</td>
<td>System asks customer to insert an envelope</td>
<td>System is displaying a request for the customer to type a dollar amount</td>
<td>Enter a legitimate dollar amount</td>
<td>System requests that customer insert an envelope</td>
</tr>
<tr>
<td>Deposit</td>
<td>System performs a legitimate deposit transaction properly</td>
<td>System is requesting that customer insert an envelope</td>
<td>Insert an envelope</td>
<td>System accepts envelope; System prints a correct receipt showing amount and correct updated balance; System records transaction correctly in the log (showing message to the bank, approval back, and acceptance of the envelope)</td>
</tr>
<tr>
<td>Deposit</td>
<td>A deposit transaction can be cancelled by the customer any time prior to inserting an envelope</td>
<td>System is displaying menu of account types</td>
<td>Press &quot;Cancel&quot; key</td>
<td>System displays an appropriate message and offers customer the option of choosing to do another transaction or not.</td>
</tr>
<tr>
<td>Deposit</td>
<td>A deposit transaction can be cancelled by the customer any time prior to inserting an envelope</td>
<td>System is requesting customer to enter a dollar amount</td>
<td>Press &quot;Cancel&quot; key</td>
<td>System displays an appropriate message and offers customer the option of choosing to do another transaction or not.</td>
</tr>
<tr>
<td>Use Case</td>
<td>Function Being Tested</td>
<td>Initial System State</td>
<td>Input</td>
<td>Expected Output</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------------------------------------------</td>
<td>---------------------------------------------</td>
<td>--------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>System Startup</td>
<td>System is started when the switch is turned &quot;on&quot;</td>
<td>System is off</td>
<td>Activate the &quot;on&quot; switch</td>
<td>System requests initial cash amount not.</td>
</tr>
<tr>
<td>Deposit</td>
<td>A deposit transaction can be cancelled by the customer any time prior to inserting an envelope</td>
<td>System is requesting customer to insert an envelope</td>
<td>Press &quot;Cancel&quot; key</td>
<td>System displays an appropriate message and offers customer the option of choosing to do another transaction or not.</td>
</tr>
<tr>
<td>Deposit</td>
<td>A deposit transaction is cancelled automatically if an envelope is not inserted within a reasonable time</td>
<td>System is requesting customer to insert an envelope</td>
<td>Wait for the request to time out</td>
<td>System displays an appropriate message and offers customer the option of choosing to do another transaction or not.</td>
</tr>
<tr>
<td>Transfer</td>
<td>System asks customer to choose an account to transfer from</td>
<td>Menu of transaction types is being displayed</td>
<td>Choose Transfer transaction</td>
<td>System displays a menu of account types specifying transfer from</td>
</tr>
<tr>
<td>Transfer</td>
<td>System asks customer to choose an account to transfer to</td>
<td>Menu of account types to transfer from is being displayed</td>
<td>Choose checking account</td>
<td>System displays a menu of account types specifying transfer to</td>
</tr>
<tr>
<td>Transfer</td>
<td>System asks customer to enter a dollar amount to transfer</td>
<td>Menu of account types to transfer to is being displayed</td>
<td>Choose savings account</td>
<td>System displays a request for the customer to type a dollar amount</td>
</tr>
<tr>
<td>Transfer</td>
<td>System performs a legitimate transfer transaction properly</td>
<td>System is displaying a request for the customer to type a dollar amount</td>
<td>Enter a legitimate dollar amount</td>
<td>System prints a correct receipt showing amount and correct updated balance; System records transaction correctly in the log (showing both message to the bank and approval back)</td>
</tr>
<tr>
<td>Transfer</td>
<td>A transfer transaction can be</td>
<td>System is displaying menu</td>
<td>Press &quot;Cancel&quot; key</td>
<td>System displays an appropriate message</td>
</tr>
<tr>
<td>Use Case</td>
<td>Function Being Tested</td>
<td>Initial System State</td>
<td>Input</td>
<td>Expected Output</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------------------------------------------------------</td>
<td>---------------------------------------</td>
<td>--------------------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>System Startup</td>
<td>System is started when the switch is turned &quot;on&quot;</td>
<td>System is off</td>
<td>Activate the &quot;on&quot; switch</td>
<td>System requests initial cash amount</td>
</tr>
<tr>
<td></td>
<td>cancelled by the customer any time prior to entering dollar amount</td>
<td>of account types specifying transfer from</td>
<td></td>
<td>and offers customer the option of choosing to do another transaction or not.</td>
</tr>
<tr>
<td>Transfer</td>
<td>A transfer transaction can be cancelled by the customer any time prior to entering dollar amount</td>
<td>System is displaying menu of account types specifying transfer to</td>
<td>Press &quot;Cancel&quot; key</td>
<td>System displays an appropriate message and offers customer the option of choosing to do another transaction or not.</td>
</tr>
<tr>
<td>Transfer</td>
<td>A transfer transaction can be cancelled by the customer any time prior to entering dollar amount</td>
<td>System is requesting customer to enter a dollar amount</td>
<td>Press &quot;Cancel&quot; key</td>
<td>System displays an appropriate message and offers customer the option of choosing to do another transaction or not.</td>
</tr>
<tr>
<td>Inquiry</td>
<td>System asks customer to choose an account to inquire about</td>
<td>Menu of transaction types is being displayed</td>
<td>Choose Inquiry transaction</td>
<td>System displays a menu of account types</td>
</tr>
<tr>
<td>Inquiry</td>
<td>System performs a legitimate inquiry transaction properly</td>
<td>System is displaying menu of account types</td>
<td>Choose checking account</td>
<td>System prints a correct receipt showing correct balance; System records transaction correctly in the log (showing both message to the bank and approval back)</td>
</tr>
<tr>
<td>Inquiry</td>
<td>An inquiry transaction can be cancelled by the customer any time prior to choosing an account</td>
<td>System is displaying menu of account types</td>
<td>Press &quot;Cancel&quot; key</td>
<td>System displays an appropriate message and offers customer the option of choosing to do another transaction or not.</td>
</tr>
<tr>
<td>Invalid PIN</td>
<td>Customer is asked to</td>
<td>Enter an incorrect</td>
<td>Customer is asked to</td>
<td></td>
</tr>
<tr>
<td>Use Case</td>
<td>Function Being Tested</td>
<td>Initial System State</td>
<td>Input</td>
<td>Expected Output</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------</td>
<td>----------------------</td>
<td>----------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>System Startup</td>
<td>System is started when the switch is turned &quot;on&quot;</td>
<td>System is off</td>
<td>Activate the &quot;on&quot; switch</td>
<td>System requests initial cash amount</td>
</tr>
<tr>
<td>Extension</td>
<td>reenter PIN</td>
<td></td>
<td>PIN; Attempt an inquiry transaction on the customer's checking account</td>
<td>re-enter PIN</td>
</tr>
<tr>
<td>Invalid PIN</td>
<td>Correct re-entry of PIN is accepted</td>
<td>Request to re-enter PIN is being displayed</td>
<td>Enter correct PIN</td>
<td>Original transaction completes successfully</td>
</tr>
<tr>
<td>Extension</td>
<td>A correctly re-entered PIN is used for subsequent transactions</td>
<td>An incorrect PIN has been re-entered and transaction completed normally</td>
<td>Perform another transaction</td>
<td>This transaction completes successfully as well</td>
</tr>
<tr>
<td>Invalid PIN</td>
<td>Incorrect re-entry of PIN is not accepted</td>
<td>Request to re-enter PIN is being displayed</td>
<td>Enter incorrect PIN</td>
<td>An appropriate message is displayed and re-entry of the PIN is again requested</td>
</tr>
<tr>
<td>Extension</td>
<td>Correct re-entry of PIN on the second try is accepted</td>
<td>Request to re-enter PIN is being displayed</td>
<td>Enter incorrect PIN the first time, then correct PIN the second time</td>
<td>Original transaction completes successfully</td>
</tr>
<tr>
<td>Invalid PIN</td>
<td>Correct re-entry of PIN on the third try is accepted</td>
<td>Request to re-enter PIN is being displayed</td>
<td>Enter incorrect PIN the first time and second times, then correct PIN the third time</td>
<td>Original transaction completes successfully</td>
</tr>
<tr>
<td>Extension</td>
<td>Three incorrect re-entries of PIN result retaining card and aborting transaction</td>
<td>Request to re-enter PIN is being displayed</td>
<td>Enter incorrect PIN three times</td>
<td>An appropriate message is displayed; Card is retained by machine; Session is terminated</td>
</tr>
</tbody>
</table>

**Analysis Classes**

An initial reading of the use cases suggests that the following will be part of the system.
- A controller object representing the ATM itself (managing the boundary objects listed below.)

- Boundary objects representing the individual component parts of the ATM:
  - Operator panel.
  - Card reader.
  - Customer console, consisting of a display and keyboard.
  - Network connection to the bank.
  - Cash dispenser.
  - Envelope acceptor.
  - Receipt printer.

- Controller objects corresponding to use cases. (Note: class ATM can handle the Startup and Shutdown use cases itself, so these do not give rise to separate objects here.)
  - Session
  - Transaction (abstract generalization, responsible for common features, with concrete specializations responsible for type-specific portions)

- An entity object representing the information encoded on the ATM card inserted by customer.

- An entity object representing the log of transactions maintained by the machine.

This leads to the following diagram of analysis classes:

Click on a class icon to go to links to various kinds of information about it
Using CRC cards to assign responsibilities to various classes for the tasks required by the various use cases leads to the creation of the following cards.
The following links can be used to go directly to the CRC cards for the various classes:

- **Class ATM**
- **Boundary/entity objects - component parts of the ATM**
  - Class CardReader
  - Class CashDispenser
  - Class CustomerConsole
  - Class EnvelopeAccepter
  - Class Log
  - Class NetworkToBank
  - Class OperatorPanel
  - Class ReceiptPrinter

- **Controller objects corresponding to the various use cases**
  - Class Session
  - Class Transaction
  - Class Withdrawal
  - Class Deposit
  - Class Transfer
  - Class Inquiry

- **Entity objects found necessary when assigning responsibilities to other objects**
  - Class Balances
  - Class Card
  - Class Message
  - Class Receipt
  - Class Status

---

Class ATM

<table>
<thead>
<tr>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start up when switch is turned on</td>
<td>OperatorPanel</td>
</tr>
</tbody>
</table>
Shut down when switch is turned off
Start a new session when card is inserted by customer
Provide access to component parts for sessions and transactions

---

**Class CardReader**

**Responsibilities**
Tell ATM when card is inserted
Read information from card
Eject card
Retain card

**Collaborators**
ATM
Card

---

**Class CashDispenser**

**Responsibilities**
Keep track of cash on hand, starting with initial amount
Report whether enough cash is available
Dispense cash

**Collaborators**
Log

---

**Class CustomerConsole**

**Responsibilities**
Display a message
Display a prompt, accept a PIN from keyboard
Display a prompt and menu, accept a choice from keyboard
Display a prompt, accept a dollar amount from keyboard
Respond to cancel key being pressed by customer

---

**Class EnvelopeAcceptor**

**Responsibilities**
Accept envelope from customer; report if timed out or cancelled

**Collaborators**
Log

---

**Class Log**
Class NetworkToBank

Responsibilities
- Initiate connection to bank at startup
- Send message to bank and wait for response

Collaborators
- Message
- Log
- Balances
- Status

Class OperatorPanel

Responsibilities
- Inform ATM of changes to state of switch
- Allow operator to specify amount of initial cash

Collaborators
- ATM

Class ReceiptPrinter

Responsibilities
- Print receipt

Collaborators
- Receipt

Class Session

Responsibilities
- Perform session use case

Collaborators
- ATM
- CardReader
- Card
- CustomerConsole
- Transaction

Update PIN value if customer has to re-enter it

Abstract Class Transaction
### Responsibilities

**Perform Transaction Use Case**
- Allow customer to choose a type of transaction

**Perform invalid PIN extension**

### Collaborators

- ATM
- CustomerConsole
- Withdrawal
- Deposit
- Transfer
- Inquiry
- ATM
- CustomerConsole
- Withdrawal
- Deposit
- Transfer
- Inquiry
- Message
- NetworkToBank
- Receipt
- ReceiptPrinter
- CustomerConsole
- Session
- CardReader

### Class Withdrawal

**Responsibilities**
- Perform operations peculiar to withdrawal transaction use case

**Collaborators**
- CustomerConsole
- CashDispenser
- Message
- Receipt

### Class Deposit

**Responsibilities**
- Perform operations peculiar to deposit transaction use case

**Collaborators**
- CustomerConsole
- Message
- EnvelopeAcceptor
- Receipt

### Class Transfer

**Responsibilities**
- Perform operations peculiar to transfer transaction use case

**Collaborators**
- CustomerConsole
- Message
<table>
<thead>
<tr>
<th>Class</th>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inquiry</td>
<td>Perform operations peculiar to inquiry transaction use case</td>
<td>CustomerConsole, Message, Receipt</td>
</tr>
<tr>
<td>Balances</td>
<td>Represent account balance information returned by bank</td>
<td></td>
</tr>
<tr>
<td>Card</td>
<td>Represent information encoded on customer's ATM card</td>
<td></td>
</tr>
<tr>
<td>Message</td>
<td>Represent information to be sent over network to bank</td>
<td></td>
</tr>
<tr>
<td>Receipt</td>
<td>Represent information to be printed on a receipt</td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td>Represent transaction status information returned by bank</td>
<td></td>
</tr>
</tbody>
</table>
Showed below is the class diagram for the ATM system. The basic structure of the class diagram arises from the responsibilities and relationships discovered when doing the CRC cards and Interaction Diagrams. (If a class uses another class as a collaborator, or sends a message to an object of that class during an Interaction, then there must either be an association linking objects of those classes, or linking the "sending" class to an object which provides access to an object of the "receiving" class.)

In the case of the ATM system, one of the responsibilities of the ATM is to provide access to its component parts for Session and Transaction objects; thus, Session and Transaction have associations to ATM, which in turn has associations to the classes representing the individual component parts. (Explicit "uses" links between Session and Transaction, on the one hand, and the component parts of the ATM, on the other hand, have been omitted from the diagram to avoid making it excessively cluttered.)

The need for the various classes in the diagram was discovered at various points in the design process.
Some classes were discovered when doing analysis (see the Analysis Class Diagram developed earlier.)

- Some classes were discovered when doing CRC cards
  - Message - used to represent a message to the bank.
  - Receipt - used to encapsulate information to be printed on a receipt.
  - Status - used to represent return value from message to the bank.
  - Balances - used to record balance information returned by the bank.

Some classes were discovered when doing detailed design or writing code

- Money - used to represent money amounts, in numerous places.
- AccountInformation - contains names of various types of accounts customer can choose from

That is, OO design is not a "waterfall" process - discoveries made when doing detailed design and coding can impact overall system design.

To prevent the diagram from becoming overly large, only the name of each class is shown - the attribute and behavior "compartments" are shown in the detailed design, but are omitted here.
State Charts for Example ATM System
Three of the objects we have identified have behavior that is sufficiently complex to warrant developing a State Chart for them. (These are the objects that were identified as the major controller objects.)

- The object representing the machine itself (responsible for the System Startup and Shutdown use cases)
- Objects representing a customer session (one per session) (responsible for the Session use case)
- Objects representing an individual transaction (one per transaction) (responsible for the Transaction use case, use cases for the specific types of transaction, and Invalid PIN extension).

**State-Chart for Overall ATM (includes System Startup and System Shutdown Use Cases)**

[State Chart Diagram]

- **OFF**
  - Entry: Display "Not available"
  - Transition: Switch turned on / perform startup
  - Transition: Switch turned off / perform shutdown

- **SERVING CUSTOMER**
  - Transition: Card inserted / create session
  - Transition: Session completed or aborted

- **IDLE**
  - Entry: Display "Please insert card"
State-Chart for One Transaction
(italicized operations are unique to each particular type of transaction)

Detailed Design
A major task of detailed design is to spell out, in detail, the attributes and methods needed by each class (the second and third "compartments" of the representation for each class in a class diagram.)

The methods needed by each class are implicit in the responsibilities assigned to the class in the CRC cards, and become explicit in the Interaction Diagrams. A responsibility listed for a class on its CRC card generally maps into a method or methods in the detailed design. Likewise, any time an object belonging to a given class is shown as the recipient of a message in either a Sequence or Collaboration Diagram, the class needs a corresponding method. Many of the needed attributes are also either explicitly or implicitly present in the diagrams; the need for others becomes evident as the code for the class is being written. (Thus detailed design and coding are a "round trip" process - detailed design dictates coding, and coding leads to elaboration of the detailed design.)

In designing this system, a few key design decisions were made:

- The class ATM is made an active class - that is, the ATM object has its own thread. Using the Java thread facility leads to defining a run() method in this class whose body is executed by the ATM's thread. The fact that class ATM is active is indicated in class diagrams by enclosing it in a heavier outline.

- Certain signals initiate computation - e.g. the signals from the operator console when the state of the switch is changed, or from the card reader when a card is inserted. In the GUI simulation of the ATM, these signals are sent by the "actionPerformed()" method of the appropriate GUI button; in a real ATM they would be sent by the physical components themselves, which might then also need to be active classes. (Note: this forms an exception to the rule that a responsibility on a CRC card translates into a method in the design - in this case the class sends a signal, rather than receiving it, so it does not need a method directly corresponding to the responsibility.)

- The Transaction hierarchy consists of the abstract class Transaction and four concrete subclasses (Withdrawal, Deposit, Transfer and Inquiry). The class Transaction has a "virtual constructor" called makeTransaction() which asks the customer to choose a transaction type and then constructs and returns an object of the appropriate subclass. The Transaction class is made responsible for carrying out the Transaction use case and the Invalid PIN extension; for the former, it makes use of abstract methods getSpecificsFromCustomer() and completeTransaction() which are implemented concretely by each subclass.

- The class Receipt is abstract. The completeTransaction() method of each kind of transaction creates a concrete instance that contains the information relevant to that kind of transaction.

- The class Status is abstract. The send() method of NetworkToBank constructs a concrete instance that contains the information appropriate to the response received from the bank to a particular message.

In the design below, each class is developed in isolation. No attempt is made to connect each class to related classes as in the class diagram, because the resulting picture would not fit on a
displayable web page. Therefore, this detailed design should be viewed in conjunction with the [Class Diagram] developed earlier.

- Detailed design for class ATM
- Component parts of the ATM
  - Detailed design for class CardReader
  - Detailed design for class CashDispenser
  - Detailed design for class CustomerConsole
  - Detailed design for class EnvelopeAcceptor
  - Detailed design for class Log
  - Detailed design for class NetworkToBank
  - Detailed design for class OperatorPanel
  - Detailed design for class ReceiptPrinter

- Detailed design for class Session
- The Transaction class hierarchy
  - Detailed design for class Transaction
  - Detailed design for class Withdrawal
  - Detailed design for class Deposit
  - Detailed design for class Transfer
  - Detailed design for class Inquiry

- Classes representing banking concepts, used by the above
  - Detailed design for class AccountInformation
  - Detailed design for class Balances
  - Detailed design for class Card
  - Detailed design for class Message
  - Detailed design for class Money
  - Detailed design for class Receipt
  - Detailed design for class Status

ATM

| - id: int |
ATM(id: int, place: String, bankName: String, bankAddress: InetAddress)
run()
switchOn()
switchOff
cardInserted()
getID(): int
getPlace(): String
getBankName(): String
cardReader: CardReader
cashDispenser: CashDispenser
customerConsole: CustomerConsole
envelopeAcceptor: EnvelopeAcceptor
log: Log
networkToBank: NetworkToBank
operatorPanel: OperatorPanel
receiptPrinter: ReceiptPrinter
state: int
switchOn: boolean
cardInserted: boolean
OFF_STATE: final int
IDLE_STATE: final int
SERVING_CUSTOMER_STATE: final int
performStartup()
performShutdown()
**CardReader**

- atm: ATM

+ CardReader(atm: ATM)
+ readCard(): Card
+ ejectCard()
+ retainCard()

**CashDispenser**

- log: Log
- cashOnHand: Money

+ CashDispenser(log: Log)
+ setInitialCash(initialCash: Money)
+ checkCashOnHand(amount: Money): boolean
+ dispenseCash(amount: Money)

**CustomerConsole**

+ CustomerConsole()
+ display(message: String)
+ readPIN(prompt: String): int throws Cancelled
+ readMenuChoice(prompt: String, menu: String []): int throws Cancelled
+ readAmount(prompt: String): Money throws Cancelled
EnvelopeAcceptor

- log: Log

+ EnvelopeAcceptor(log: Log)
+ acceptEnvelope() throws Cancelled

Log

+ Log()
+ logSend(message: Message)
+ logResponse(status: Status)
+ logCashDispensed(amount: Money)
+ logEnvelopeAccepted()

NetworkToBank

- log: Log
- bankAddress: InetAddress

+ NetworkToBank(log: Log, bankAddress: InetAddress)
+ openConnection()
+ closeConnection()
+ sendMessage(message: Message, out balances: Balances): Status

OperatorPanel

- atm: ATM

+ OperatorPanel(atm: ATM)
+ getInitialCash(): Money
### ReceiptPrinter

```java
+ ReceiptPrinter()
+ printReceipt(receipt: Receipt)
```

### Session

```java
- atm: ATM
- pin: int
- state: int
- READING_CARD_STATE: final int
- READING_PIN_STATE: final int
- CHOOSING_TRANSACTION_STATE: final int
- PERFORMING_TRANSACTION_STATE: final int
- EJECTING_CARD_STATE: final int
- FINAL_STATE: final int

+ Session(atm: ATM)>
+ performSession()
+ setPIN(int pin)
```

### Transaction

```java
# atm: ATM
# session: Session
# card: Card
# pin: int
# serialNumber: int
# message: Message
# balances: Balances

- TRANSACTION_TYPES_MENU: final String []
```
- nextSerialNumber: int
- state: int
- GETTING_SPECIFICS_STATE: final int
- SENDING_TO_BANK_STATE: final int
- INVALID_PIN_STATE: final int
- COMPLETING_TRANSACTION_STATE: final int
- PRINTING_RECEIPT_STATE: final int
- ASKING_DO_ANOTHER_STATE: final int

# Transaction(atm: ATM, session: Session, card: Card, pin: int)
+ makeTransaction(atm: ATM, session: Session, card: Card, pin: int): Transaction throws Cancelled
+ performTransaction(): boolean throws CardRetained
+ performInvalidPinExtension(): Status throws Cancelled, CardRetained
+ getSerialNumber(): int

# getSpecificsFromCustomer(): Message throws Cancelled
# completeTransaction(): Receipt throws Cancelled

Withdrawal

- from: int
- amount: Money

+ Withdrawal(atm: ATM, session: Session, card: Card, pin: int)
# getSpecificsFromCustomer(): Message throws Cancelled
# completeTransaction(): Receipt

Deposit

- to: int
- amount: Money

+ Deposit(atm: ATM, session: Session, card: Card, pin: int)
# getSpecificsFromCustomer(): Message throws Cancelled
# completeTransaction(): Receipt throws Cancelled

## Transfer
- from: int
- to: int
- amount: Money

+ Transfer(atm: ATM, session: Session, card: Card, pin: int)

# getSpecificsFromCustomer(): Message throws Cancelled

# completeTransaction(): Receipt

## Inquiry
- from: int

+ Inquiry(atm: ATM, session: Session, card: Card, pin: int)

# getSpecificsFromCustomer(): Message throws Cancelled

# completeTransaction(): Receipt

## AccountInformation

+ ACCOUNT_NAMES: final String[]
+ ACCOUNT_ABBREVIATIONS: final String[]

## Balances
- total: Money
- available: Money

+ Balances()
+ setBalances(total: Money, available: Money)
+ getTotal(): Money
+ getAvailable(): Money

### Card

- number: int

+ Card(number: int)
+ getNumber(): int

### Message

+ WITHDRAWAL: final int
+ INITIATE_DEPOSIT: final int
+ COMPLETE_DEPOSIT: final int
+ TRANSFER: final int
+ INQUIRY: final int

- messageCode: int
- card: Card
- pin: int
- serialNumber: int
- fromAccount: int
- toAccount: int
- amount: int

+ Message(messageCode: int, cardNumber: Card, pin: int, serialNumber: int, fromAccount: int, toAccount: int, amount: Money)
+ toString(): String
+ setPIN(pin: int)
+ getMessageCode(): int
+ getCard(): Card
+ getPIN(): int
+ getSerialNumber(): int
+ getFromAccount(): int
+ getToAccount(): int
+ getAmount(): Money

**Money**

- cents: long

+ Money(dollars: int)
+ Money(dollars: int, cents: int)
+ Money(toCopy: Money)
+ toString(): String
+ add(amountToAdd: Money)
+ subtract(amountToSubtract: Money)
+ lessEqual(compareTo: Money): boolean

**Receipt**

- headingPortion: String []
# detailsPortion(): String []
- balancesPortion: String []

# Receipt(atm: ATM, card: Card, transaction: Transaction, balances: Balances)
+ getLines(): Enumeration

**Status**
The package diagram shows how the various classes are grouped into packages. There are two "top-level" classes - ATMMain and ATMApplet - which allow the system to be run (respectively) as an application or as an Applet. (Only one of the two would be instantiated in any particular use of the system.)

Each of these classes, in turn, depends on the package atm which contains the class ATM that represents the system as a whole, and the class Session that represents one session. ATM depends on Session, and vice versa - since the ATM creates Sessions, and each Session, in turn, uses the ATM to interact with the customer.

The subpackage transaction contains the classes used to represent individual transactions that a customer initiates. The class Session depends on the transaction package because it creates individual transaction objects. These, in turn, again depend on the ATM to interact with the customer.

The subpackage physical contains the classes that represent the various physical components of the ATM. For the purposes of this simulation, these are simulated by a GUI. A real ATM would have quite different classes in this package - classes that actually control its physical components. The class ATM makes use of these components, and Session and the various kinds of transaction gain access to them through ATM to actually perform the needed operations.

Finally, the package banking contains classes that represent the banking enterprise itself and the information communicated back and forth between the ATM and the bank - i.e. classes which might be the same in a totally different implementation of an ATM that interacts with the same bank.

This is, of course, a simulation. However, most of the code that is specific to the simulation resides in the package physical, plus the two top-level classes. Presumably, the other classes and packages might be similar in a real system.
Executable
(See below for directions)

To run the simulation above, you need to do the following:

1. Click on the "ON" button (lower right-hand corner) to turn the ATM on.
2. Enter the number of $20 bills you want to have be in the cash dispenser at the start of the simulation when you are prompted to do so, and press RETURN.
3. Perform any number of sessions, as follows:
   1. Click on the "Click to insert card" button to simulate inserting a card.
   2. Type in the card number when you are prompted to do so (see below), and press RETURN.
   3. Enter the PIN associated with the card (see below). Although you can use your regular keyboard, it's more fun to click on the keys on the simulated ATM keyboard as displayed.
   4. Perform any number of transactions, using your mouse to click the keys on the simulated ATM keyboard. Note that the machine will simulate ejecting your card when you indicate you do not wish to perform any more transactions (unless, of course, your card is retained due to too many invalid PINs).
4. Turn off the ATM by clicking on the "OFF" button (same position as the "ON" button). Note that you cannot turn the ATM off while in the middle of a customer session.
5. The entire simulation may be repeated as many times as you want, by turning the machine ON again.

For demonstration purposes, this program uses a very simple simulation of the bank, with hardwired card numbers and PIN's, etc. The following are the available cards:

<table>
<thead>
<tr>
<th>Card Number</th>
<th>PIN</th>
<th>Available accounts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>42</td>
<td>Checking account #1, Savings account #2</td>
</tr>
<tr>
<td>2</td>
<td>1234</td>
<td>Checking account #1, Money market account #3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Note that both cards link to the same checking account)</td>
</tr>
</tbody>
</table>

All the features of the bank work - both current and available balances (initially the same) are maintained for each account and a $300 daily withdrawal limit per card is enforced.

I would appreciate feedback about the operation and/or appearance of this applet on different platforms. (Feedback about other issues is welcome, too, of course.)

Maintenance
This page lists various changes that might be made to the system. Modifying the various documents to incorporate one or more of these changes would make an interesting exercise for the reader.

**Corrective Maintenance**

- The careful reader will note that the requirement that "Entries will be made in the log when the ATM is started up and shut down" has not been reflected in any of the diagrams or the code. What changes would be needed to implement this requirement?

- When the system asks the customer a question, it waits as long as necessary for the customer to answer. This could cause a severe security problem if the customer were to leave at this point without answering the question, since the ATM card would still be in the machine, generally with the customer's PIN having already been entered, thus allowing another user to "hijack" the session and perform operations like cash withdrawal from the customer's account.

While this seems unlikely, it is conceivably in some cases that a distracted customer might make this mistake. For example, after performing a transaction, the system asks the user "Would you like to do another transaction?". If the customer has just withdrawn cash, it is conceivable that the customer might place the cash in his/her wallet and accidentally forget about answering the question and retrieving the card.

A real ATM typically has a timeout mechanism that begins beeping and ultimately ejects the card and ends the session if the user fails to respond within a certain amount of time. While a customer who leaves before this happens might inadvertently leave a card behind, at least the customer would not be "logged in" with his/her PIN. (Thus, the security risk is comparable to that resulting from a lost card, with the customer's account still PIN protected.) What changes would be needed to the system requirements, design, and code to improve security this way?

**Adaptive Maintenance**

- Many ATM's have cash dispensers that can handle two different denominations of bills - often $20 and $5 - which allows a user to request cash in any amount that is a multiple of $5. Suppose the cash dispenser in the example (which only handles $20 bills) were replaced with such a dispenser. What changes would be needed to take advantage of the enhanced capabilities of such a dispenser?

**Perfective Maintenance**

- When the customer's card is retained, an entry should be made in the log to this effect.

- Many ATM's offer the customer additional options beyond the basic four listed here - e.g. making a payment on a loan the customer has from one of the accounts linked to his/her card. What changes would be needed to make it possible for a customer to make a loan payment at the ATM?
2. Quiz on the subject:

Quiz should be conducted on tips in the laboratory, recent trends and subject knowledge of the subject. The quiz questions should be formulated such that questions are normally are from the scope outside of the books. However twisted questions and self formulated questions by the faculty can be asked but correctness of it is necessarily to be thoroughly checked before the conduction of the quiz.
3. Conduction of Viva-Voce Examinations:

Teacher should oral exams of the students with full preparation. Normally, the objective questions with guess are to be avoided. To make it meaningful, the questions should be such that depth of the students in the subject is tested. Oral examinations are to be conducted in co-cordial environment amongst the teachers taking the examination. Teachers taking such examinations should not have ill thoughts about each other and courtesies should be offered to each other in case of difference of opinion, which should be critically suppressed in front of the students.

3. Submission:

Document Standard:
A] Page Size A4 Size
B] Running text Justified text
C] Spacing 1 Line
D] Page Layout and Margins (Dimensions in Cms)
14. Evaluation and marking system:

Basic honesty in the evaluation and marking system is absolutely essential and in the process impartial nature of the evaluator is required in the examination system to become popular amongst the students. It is a wrong approach or concept to award the students by way of easy marking to get cheap popularity among the students to which they do not deserve. It is a primary responsibility of the teacher that right students who are really putting up lot of hard work with right kind of intelligence are correctly awarded.
• The marking patterns should be justifiable to the students without any ambiguity and teacher should see that students are faced with unjust circumstances.