Case Study: ATM machine I

Amalia Foka
CEID - University of Patras
Object Oriented Programming II (C++)
Fall 2010-2011
Requirements Document

- An ATM allows users to perform basic financial transactions
  - view their account balance
  - withdraw cash
  - deposit funds
- Each user can have only one account at the bank
User Interface

- a screen to display messages
- a keypad for numeric input
- a cash dispenser
- a deposit slot
ATM session

- The cash dispenser begins each day loaded with 500 $20 bills.
- Authenticating a user
  - based on account number and (PIN)
  - bank's account information database
    - stores an account number, a PIN and a balance
ATM Session

- Display a welcome message and prompt the user to enter an account number.
- The user enters a five-digit account number, using the keypad.
- The screen prompts the user to enter the PIN.
- The user enters a five-digit PIN, using the keypad.
- If the user enters a valid account number and the correct PIN for that account, the screen displays the main menu.
- If the user enters an invalid account number or an incorrect PIN, the screen displays an appropriate message, then the ATM returns to Step 1 to restart the authentication process.
ATM Main menu

When an invalid option is entered, display an error message, then redisplay the main menu.
1 – View My Balance

- displays the user's account balance.
  - the ATM retrieves the balance from the bank's database
2 – Withdraw Cash

- Display withdrawal menu
2 – Withdraw Cash

- withdrawal amount greater than balance
  - Display message, ask for smaller amount

- Choose option to exit
  - Display main menu
2 – Withdraw Cash

- Valid withdrawal amount is valid
  - Authenticate user – ask for pin
  - If amount > than ATM money in dispenser
    - Display message to ask for smaller amount
    - Display Withdrawal menu
  - Debit the withdraw amount for balance in database
  - Dispense the money
  - Display a message reminding the user to take the money
3 – Deposit Funds

- Prompts user to enter a deposit amount or 0 (zero) to cancel the transaction.
- User enters a deposit amount or 0
  - the amount is entered as a number of cents (e.g., 125).
  - The ATM divides this number by 100 to obtain a dollar amount (e.g., 125 ÷ 100 = 1.25).
- Displays a message telling the user to insert a deposit envelope into the deposit slot
- If the slot receives a deposit envelope within two minutes, the ATM credits the amount to the balance in the bank's database.
- This money is not immediately available for withdrawal.
  - First the bank verifies the amount and then updates the balance stored in the database.
- If the slot does not receive an envelope within two minutes
  - display a message that the transaction is cancelled due to inactivity.
  - The ATM then displays the main menu and waits for user input.
ATM Session

- When a transaction is executed
  - redisplay the main menu
- If the user chooses to exit the system
  - display a thank you message
  - then display the welcome message for the next user.
Design Stage I

- **Use case diagrams**
  - model the interactions between a system and its external entities (actors) in terms of use cases
    - system capabilities, such as
      - "View Account Balance,"
      - "Withdraw Cash" and
      - "Deposit Funds"
Use Case Diagrams

- Use Case Diagram
  - model the interactions between
    - a system's clients (bank customers) and
    - the system
  - produced during the analysis stage of the software life cycle
Use Case Diagrams

- **Actor**
  - defines the roles that an external entity such as a person or another system plays when interacting with the system
Design Stage II

- **Class diagrams**
  - model the classes, or "building blocks," used in a system.
  - Each noun or "thing" described in the requirements document is a candidate to be a class in the system (e.g., "account," "keypad").
  - Class diagrams help specify the structural relationships between parts of the system.
Identifying the Classes

- find key nouns and noun phrases to help us identify classes that comprise the ATM system

<table>
<thead>
<tr>
<th>Nouns and noun phrases in the requirements document</th>
</tr>
</thead>
<tbody>
<tr>
<td>bank</td>
</tr>
<tr>
<td>ATM</td>
</tr>
<tr>
<td>user</td>
</tr>
<tr>
<td>customer</td>
</tr>
<tr>
<td>transaction</td>
</tr>
<tr>
<td>account</td>
</tr>
<tr>
<td>balance</td>
</tr>
</tbody>
</table>
Identifying the Classes

- Create classes only for the nouns and noun phrases that have significance in the ATM system
- We do not need to model "bank" as a class
  - the bank is not a part of the ATM system
- "Customer" and "user" also represent entities outside of the system
  - they are important because they interact with our ATM system, but we do not need to model them as classes in the ATM software.
  - We modeled an ATM user (i.e., a bank customer) as the actor in the use case diagram.
Identifying the Classes

- Create classes only for the nouns and noun phrases that have significance in the ATM system.
- We do not need to model "bank" as a class:
  - the bank is not a part of the ATM system.
- "Customer" and "user" also represent entities outside of the system:
  - they are important because they interact with our ATM system, but we do not need to model them as classes in the ATM software.
  - We modeled an ATM user (i.e., a bank customer) as the actor in the use case diagram.
Identifying the Classes

- We do not model "$20 bill" or "deposit envelope" as classes.
  - These are physical objects in the real world, but they are not part of what is being automated.
  - We can adequately represent the presence of bills in the system using an attribute of the class that models the cash dispenser.
    - For example, the cash dispenser maintains a count of the number of bills it contains.
  - The requirements document does not say anything about what the system should do with deposit envelopes after it receives them.
    - We can assume that simply acknowledging the receipt of an envelope an operation performed by the class that models the deposit slot is sufficient to represent the presence of an envelope in the system.
Identifying the Classes

- "money", "balance"
  - attributes of other classes seems most appropriate

- "account number", "PIN"
  - important attributes of a bank account. They do not exhibit behaviors. Model them as attributes of an account class.

- "transaction"
  - We do not model the broad notion of a financial transaction at this time. Instead, we model the three types of transactions as individual classes.
    - "balance inquiry,"
    - "withdrawal" and
    - "deposit"
  - Later we "factor out" common features of all transactions into a general "transaction" class using the object-oriented concepts of abstract classes and inheritance
Candidate Classes

- ATM
- screen
- keypad
- cash dispenser
- deposit slot
- account
- bank database
- balance inquiry
- withdrawal
- deposit
Candidate Classes

- ATM
- Screen
- Keypad
- CashDispenser
- DepositSlot
- Account
- BankDatabase
- BalanceInquiry
- Withdrawal
- Deposit
UML Class Diagram

- Each class modeled as a rectangle with three compartments.
- The top compartment contains the name of the class, centered horizontally and in boldface.
- The middle compartment contains the class's attributes.
- The bottom compartment contains the class's operations.
the solid line that connects the two classes represents an **association** relationship between classes.

The numbers near each end of the line are **multiplicity** values, which indicate how many objects of each class participate in the association.

- at any given moment, one ATM object participates in an association with either zero or one Withdrawal objects.

An **association** can be named.

association names are directional, as indicated by the filled arrow-head

- "one object of class ATM executes zero or one objects of class Withdrawal."
Role name – currentTransaction

- indicates that the Withdrawal object participating in the Executes association with an object of class ATM represents the transaction currently being processed by the ATM.
- In other contexts, a Withdrawal object may take on other roles (e.g., the previous transaction).
- we do not specify a role name for the ATM end of the Executes association.
- Role names in class diagrams are often omitted when the meaning of an association is clear without them.
### Multiplicity Types

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>One</td>
</tr>
<tr>
<td>$m$</td>
<td>An integer value</td>
</tr>
<tr>
<td>0..1</td>
<td>Zero or one</td>
</tr>
<tr>
<td>$m, n$</td>
<td>$m$ or $n$</td>
</tr>
<tr>
<td>$m..n$</td>
<td>At least $m$, but not more than $n$</td>
</tr>
<tr>
<td>*</td>
<td>Any nonnegative integer (zero or more)</td>
</tr>
<tr>
<td>0..*</td>
<td>Zero or more (identical to *)</td>
</tr>
<tr>
<td>1..*</td>
<td>One or more</td>
</tr>
</tbody>
</table>
UML – composition relationship
UML – composition relationship

- **solid diamonds** attached to the association lines of class ATM indicate that class ATM has a **composition** relationship with classes Screen, Keypad, CashDispenser and DepositSlot.

- Composition implies a whole/part relationship.
  - The class that has the composition symbol (the solid diamond) on its end of the association line is the whole (in this case, ATM),
  - the classes on the other end of the association lines are the parts in this case, classes Screen, Keypad, CashDispenser and DepositSlot.

- An object of class ATM is formed from one object of class Screen, one object of class CashDispenser, one object of class Keypad and one object of class DepositSlot.

- The ATM "has a" screen, a keypad, a cash dispenser and a deposit slot.

- The **"has-a" relationship** defines composition.
UML – composition relationship properties

- Only one class in the relationship can represent the whole
  - (i.e., the diamond can be placed on only one end of the association line).
  - For example, either the screen is part of the ATM or the ATM is part of the screen, but the screen and the ATM cannot both represent the whole in the relationship.

- The parts in the composition relationship exist only as long as the whole, and the whole is responsible for the creation and destruction of its parts.
  - For example, the act of constructing an ATM includes manufacturing its parts. Furthermore, if the ATM is destroyed, its screen, keypad, cash dispenser and deposit slot are also destroyed.

- A part may belong to only one whole at a time, although the part may be removed and attached to another whole, which then assumes responsibility for the part.
UML – composition relationship properties

- **hollow diamonds**
  - can be attached to the ends of association lines to indicate

- **Aggregation**
  - a weaker form of composition

- For example, a personal computer and a computer monitor participate in an aggregation relationship. The computer "has a" monitor, but the two parts can exist independently, and the same monitor can be attached to multiple computers at once, thus violating the second and third properties of composition.
Class diagram for the ATM system model