ATM Case Study

OBJECTIVES

- .......
- .......
2.9 (Optional) Software Engineering Case Study: Examining the Requirements Document

- Object-oriented design (OOD) process using UML
  - Chapters 3 to 8, 10
- Object-oriented programming (OOP) implementation
  - Appendix J
2.9 (Optional) Software Engineering Case Study (Cont.)

- **Requirements Document**
  - New automated teller machine (ATM)
  - Allows basic financial transaction
    - View balance, withdraw cash, deposit funds
  - User interface
    - Display screen, keypad, cash dispenser, deposit slot
  - ATM session
    - Authenticate user, execute financial transaction

---

**Fig. 2.17** | Automated teller machine user interface.

© 2005 Pearson Education, Inc. All rights reserved.
Fig. 2.18 | ATM main menu.

Fig. 2.19 | ATM withdrawal menu.
2.9 (Optional) Software Engineering Case Study (Cont.)

- Analyzing the ATM System
  - Requirements gathering
  - Software life cycle
    - Waterfall model
    - Interactive model
  - Use case modeling

- Use case Diagram
  - Model the interactions between clients and its use cases
  - Actor
    - External entity

Fig. 2.20 | Use case diagram for the ATM system from the user’s perspective.
Fig. 2.21 | Use case diagram for a modified version of our ATM system that also allows users to transfer money between accounts.

© 2005 Pearson Education, Inc. All rights reserved.

2.9 (Optional) Software Engineering Case Study (Cont.)

- UML diagram types
  - Model system structure
  - Class diagram
    - Models classes, or “building blocks” of a system
    - screen, keypad, cash dispenser, deposit slot.
2.9 (Optional) Software Engineering Case Study (Cont.)

- Model system behavior
  - Use case diagrams
    - Model interactions between user and a system
  - State machine diagrams
    - Model the ways in which an object changes state
  - Activity diagrams
    - Models an object’s activity during program execution
  - Communication diagrams (collaboration diagrams)
    - Models the interactions among objects
    - Emphasize what interactions occur
  - Sequence diagrams
    - Models the interactions among objects
    - Emphasize when interactions occur
3.10 (Optional) Software Engineering Case Study: Identifying the Classes in a Requirements Document

- Begin designing the ATM system
  - Analyze the nouns and noun phrases
  - Introduce UML class diagrams

Identifying the Classes in a System

- Key nouns and noun phrases in requirements document
  - Some are attributes of other classes
  - Some do not correspond to parts of the system
  - Some are classes
    - To be represented by UML class diagrams
Fig. 3.19 Nouns and noun phrases in the requirements document.

© 2005 Pearson Education, Inc. All rights reserved.

Modeling Classes

- **UML class diagrams**
  - Top compartment contains name of the class
  - Middle compartment contains class's attributes or instance variables
  - Bottom compartment contains class's operations or methods

© 2005 Pearson Education, Inc. All rights reserved.
Fig. 3.20 | Representing a class in the UML using a class diagram.

© 2005 Pearson Education, Inc. All rights reserved.

Modeling Classes

• UML class diagrams
  – Allows suppression of class attributes and operations
    • Called an elided diagram
  – Solid line that connects two classes represents an association
    • Numbers near end of each line are multiplicity values

© 2005 Pearson Education, Inc. All rights reserved.
Fig. 3.21 | Class diagram showing an association among classes.

© 2005 Pearson Education, Inc. All rights reserved.

<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 non-negative 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>

Fig. 3.22 | Multiplicity types.

© 2005 Pearson Education, Inc. All rights reserved.
Modeling Classes

- **UML class diagrams**
  - Solid diamonds attached to association lines indicate a composition relationship
  - Hollow diamonds indicate aggregation – a weaker form of composition

![Class diagram showing composition relationships.](image)
Fig. 3.24 | Class diagram for the ATM system model.

© 2005 Pearson Education, Inc. All rights reserved.

Fig. 3.25 | Class diagram showing composition relationships of a class Car.

© 2005 Pearson Education, Inc. All rights reserved.
Fig. 3.26 | Class diagram for the ATM system model including class Deposit.
4.15 Identifying Class Attributes

- **Identifying attributes**
  - Look for descriptive words and phrases in the requirements document
  - Create attributes and assign them to classes
  - Each attribute is given an attribute type
  - Some attributes may have an initial value
  - Some classes may end up without any attributes
    - Additional attributes may be added later on as the design and implementation process continues
  - Reference-type attributes are modeled more clearly as associations

---

**Fig. 4.23** | Descriptive words and phrases from the ATM requirements.
At early stages in the design process, classes often lack attributes (and operations). Such classes should not be eliminated, however, because attributes (and operations) may become evident in the later phases of design and implementation.

**Fig. 4.24 | Classes with attributes.**
5.11 (Optional) Software Engineering Case Study: Identifying Object’s State and Activities

- **State Machine Diagrams**
  - Commonly called state diagram
  - Model several states of an object
  - Show under what circumstances the object changes state
  - Focus on system behavior
- **UML representation**
  - **State**
    - Rounded rectangle
  - **Initial state**
    - Solid circle
  - **Transitions**
    - Arrows with stick arrowheads
Software Engineering Observation 5.5

Software designers do not generally create state diagrams showing every possible state and state transition for all attributes—there are simply too many of them. State diagrams typically show only key states and state transitions.
• Activity Diagrams
  – Focus on system behavior
  – Model an object’s workflow during program execution
  – Model the actions the object will perform and in what order
  – UML representation
    • Action state (rectangle with its left and right sides replaced by arcs curving outwards)
    • Action order (arrow with a stick arrowhead)
    • Initial state (solid circle)
    • Final state (solid circle enclosed in an open circle)

Fig. 5.30 | Activity diagram for a BalancelInquiry object.
Fig. 5.31 | Activity diagram for a withdrawal transaction.

Fig. 5.32 | Activity diagram for a deposit transaction.
6.14 (Optional) Identifying Class Operations

- Identifying operations
  - Examine key verbs and verb phrases in the requirements document

- Modeling operations in UML
  - Each operation is given an operation name, a parameter list and a return type:
    - `operationName ( parameter1 , parameter2 , ... , parameterN ) : returnType`
    - Each parameter has a parameter name and a parameter type
      - `parameterName : parameterType`
6.14 (Optional) Identifying Class Operations (Cont.)

- Some operations may not have return types yet
  - Remaining return types will be added as design and implementation proceed

- Identifying and modeling operation parameters
  - Examine what data the operation requires to perform its assigned task
  - Additional parameters may be added later on

---

<table>
<thead>
<tr>
<th>Class</th>
<th>Verbs and verb phrases</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATM</td>
<td>executes financial transactions</td>
</tr>
<tr>
<td>BalanceInquiry</td>
<td>[none in the requirements document]</td>
</tr>
<tr>
<td>Withdrawal</td>
<td>[none in the requirements document]</td>
</tr>
<tr>
<td>Deposit</td>
<td>[none in the requirements document]</td>
</tr>
<tr>
<td>BankDatabase</td>
<td>authenticates a user, retrieves an account balance, credits a deposit amount to an account, debits a withdrawal amount from an account</td>
</tr>
<tr>
<td>Account</td>
<td>retrieves an account balance, credits a deposit amount to an account, debits a withdrawal amount from an account</td>
</tr>
<tr>
<td>Screen</td>
<td>displays a message to the user</td>
</tr>
<tr>
<td>Keypad</td>
<td>receives numeric input from the user</td>
</tr>
<tr>
<td>CashDispenser</td>
<td>dispenses cash, indicates whether it contains enough cash to satisfy a withdrawal request</td>
</tr>
<tr>
<td>DepositSlot</td>
<td>receives a deposit envelope</td>
</tr>
</tbody>
</table>

**Fig. 6.20** | Verbs and verb phrases for each class in the ATM system.
Fig. 6.21 | Classes in the ATM system with attributes and operations.

© 2005 Pearson Education, Inc.  All rights reserved.

Fig. 6.22 | Class BankDatabase with operation parameters.

© 2005 Pearson Education, Inc.  All rights reserved.
Fig. 6.23 | **Class Account** with operation parameters.

© 2005 Pearson Education, Inc. All rights reserved.

Fig. 6.24 | **Class Screen** with operation parameters.

© 2005 Pearson Education, Inc. All rights reserved.
**Fig. 6.25** | **Class CashDispenser with operation parameters.**

© 2005 Pearson Education, Inc. All rights reserved.
7.14 (Optional) Software Engineering Case Study: Collaboration Among Objects

• Collaborations
  – When objects communicate to accomplish task
    • Accomplished by invoking operations (methods)
  – One object sends a message to another object

7.14 (Optional) Software Engineering Case Study (Cont.)

• Identifying the collaborations in a system
  – Read requirements document to find
    • What ATM should do to authenticate a use
    • What ATM should do to perform transactions
  – For each action, decide
    • Which objects must interact
      – Sending object
      – Receiving object
An object of class... | sends the message... | to an object of class...
---|---|---
ATM | displayMessage | Screen
getInput | Keypad
authenticateUser | BankDatabase
execute | BalanceInquiry
execute | Deposit
Withdrawal | displayMessage | Screen
getInput | Keypad
getAvailableBalance | BankDatabase
isSufficientCashAvailable | CashDispenser
debit | BankDatabase
dispenseCash | BankDispenser
Deposit | displayMessage | Screen
getInput | Keypad
isEnvelopeReceived | DepositSlot
Credit | BankDatabase
BankDatabase | validatePIN | Account
getAvailableBalance | Account
getTotalBalance | Account
debit | Account
Credit | Account

Fig. 7.25 | Collaborations in the ATM system.

7.14 (Optional) Software Engineering Case Study (Cont.)

- **Interaction Diagrams**
  - Model interactions use UML
  - Communication diagrams
    - Also called collaboration diagrams
    - Emphasize which objects participate in collaborations
  - Sequence diagrams
    - Emphasize when messages are sent between objects
7.14 (Optional) Software Engineering Case Study (Cont.)

- **Communication diagrams**
  - Objects
    - Modeled as rectangles
    - Contain names in the form `objectName:className`
  - Objects are connected with solid lines
  - Messages are passed along these lines in the direction shown by arrows
  - Name of message appears next to the arrow

---

![Diagram of ATM and BalanceInquiry](image)

**Fig. 7.26** Communication diagram of the ATM executing a balance inquiry.
7.14 (Optional) Software Engineering Case Study (Cont.)

- **Sequence of messages in a communication diagram**
  - Appear to the left of a message name
  - Indicate the order in which the message is passed
  - Process in numerical order from least to greatest

---

**Fig. 7.27** | Communication diagram for executing a balance inquiry.
7.14 (Optional) Software Engineering Case Study (Cont.)

- **Sequence diagrams**
  - Help model the timing of collaborations
  - **Lifeline**
    - Dotted line extending down from an object’s rectangle
      - Represents the progression of time
  - **Activation**
    - Thin vertical rectangle
      - Indicates that an object is executing

---

**Fig. 7.28** | Sequence diagram that models a Withdrawal executing.
Fig. 7.29 | Sequence diagram that models a Deposit executing.
8.18 (Optional) GUI and Graphics Case Study: Using Objects with Graphics

- To create a consistent drawing that remains the same each time it is drawn
  - Store information about the displayed shapes so that they can be reproduced exactly the same way each time `paintComponent` is called

8.19 Starting to Program the Classes of the ATM System

- **Visibility**
  - Attributes normally should be private, methods invoked by clients should be public
  - Visibility markers in UML
    - A plus sign (+) indicates public visibility
    - A minus sign (-) indicates private visibility

- **Navigability**
  - Navigability arrows indicate in which direction an association can be traversed
  - Bidirectional navigability
    - Associations with navigability arrows at both ends or no navigability arrows at all can be traversed in either direction
8.19 Starting to Program the Classes of the ATM System (Cont.)

- Implementing the ATM system from its UML design (for each class)
  - Declare a public class with the name in the first compartment and an empty no-argument constructor
  - Declare instance variables based on attributes in the second compartment
  - Declare references to other objects based on associations described in the class diagram
  - Declare the shells of the methods based on the operations in the third compartment
    - Use the return type void if no return type has been specified

© 2005 Pearson Education, Inc. All rights reserved.

Fig. 8.24 | Class diagram with visibility markers.

© 2005 Pearson Education, Inc. All rights reserved.
Fig. 8.25 | Class diagram with navigability arrows.

```java
1 // Class Withdrawal represents an ATM withdrawal transaction
2 public class Withdrawal
3 {
4     // no-argument constructor
5     public Withdrawal()
6     {
7         // end no-argument Withdrawal constructor
8     } // end class Withdrawal
```

Outline

withdrawal.java
1 // Class Withdrawal represents an ATM withdrawal transaction
2 public class Withdrawal
3 {
4     // attributes
5     private int accountNumber; // account to withdraw funds from
6     private double amount; // amount to withdraw
7     // no-argument constructor
8     public Withdrawal()
9     {
10    } // end no-argument Withdrawal constructor
11 } // end class Withdrawal

// attributes
private int accountNumber; // account to withdraw funds from
private double amount; // amount to withdraw

// references to associated objects
private Screen screen; // ATM’s screen
private Keypad keypad; // ATM’s keypad
private CashDispenser cashDispenser; // ATM’s cash dispenser
private BankDatabase bankDatabase; // account info database

// no-argument constructor
public Withdrawal()
{
} // end no-argument Withdrawal constructor

Declare instance variables

Declare references to other objects
// Class Withdrawal represents an ATM withdrawal transaction
public class Withdrawal
{
  // attributes
  private int accountNumber; // account to withdraw funds from
  private double amount; // amount to withdraw

  // references to associated objects
  private Screen screen; // ATM's screen
  private Keypad keypad; // ATM's keypad
  private CashDispenser cashDispenser; // ATM's cash dispenser
  private BankDatabase bankDatabase; // account info database

  // no-argument constructor
  public Withdrawal()
  {
  } // end no-argument Withdrawal constructor

  // operations
  public void execute()
  {
  } // end method execute
} // end class Withdrawal

// Class Keypad represents an ATM's keypad
public class Keypad
{
  // no attributes have been specified yet

  // no-argument constructor
  public Keypad()
  {
  } // end no-argument Keypad constructor

  // operations
  public int getInput()
  {
  } // end method getInput
} // end class Keypad

Declare shell of a method with return type void
10.9 (Optional) Software Engineering Case Study: Incorporating Inheritance into the ATM System

- **UML model for inheritance**
  - The generalization relationship
    - The superclass is a generalization of the subclasses
    - The subclasses are specializations of the superclass

- **Transaction superclass**
  - Contains the methods and fields `BalanceInquiry`, `Withdrawal`, and `Deposit` have in common
    - `execute` method
    - `accountNumber` field
**Fig. 10.19** Attributes and operations of classes `BalanceInquiry`, `Withdrawal` and `Deposit`.

**Fig. 10.20** Class diagram modeling generalization of superclass `Transaction` and subclasses `BalanceInquiry`, `Withdrawal` and `Deposit`. Note that abstract class names (e.g., `Transaction`) and method names (e.g., `execute in class Transaction`) appear in italics.
Fig. 10.21 | Class diagram of the ATM system (incorporating inheritance). Note that abstract class names (e.g., Transaction) appear in italics.

Software Engineering Observation 10.12

A complete class diagram shows all the associations among classes and all the attributes and operations for each class. When the number of class attributes, methods and associations is substantial (as in Fig. 10.21 and Fig. 10.22), a good practice that promotes readability is to divide this information between two class diagrams—one focusing on associations and the other on attributes and methods.
• Incorporating inheritance into the ATM system design
  – If class A is a generalization of class B, then class B extends class A
  – If class A is an abstract class and class B is a subclass of class A, then class B must implement the abstract methods of class A if class B is to be a concrete class

---

Fig. 10.22 | Class diagram with attributes and operations (incorporating inheritance). Note that abstract class names (e.g., Transaction) and method names (e.g., execute in class Transaction) appear in italic
// Class Withdrawal represents an ATM withdrawal transaction
public class Withdrawal extends Transaction
{

} // end class Withdrawal

Outline
Subclass Withdrawal extends superclass Transaction
Withrawl.java

Outline
Subclass Withdrawal extends superclass Transaction
Withdrawal.java

Outline
Subclass Withdrawal extends superclass Transaction
Withdrawal.java
Several UML modeling tools convert UML-based designs into Java code and can speed the implementation process considerably. For more information on these tools, refer to the Internet and Web Resources listed at the end of Section 2.9.

```
1 // Abstract class Transaction represents an ATM transaction
2 public abstract class Transaction
3 {
4     // attributes
5     private int accountNumber; // indicates account involved
6     private Screen screen; // ATM's screen
7     private BankDatabase bankDatabase; // account info database
8     // no-argument constructor invoked by subclasses using super()
9     public Transaction() {
10     } // end no-argument Transaction constructor
11     // return account number
12     public int getAccountNumber() {
13     } // end method getAccountNumber
14 }
```

© 2005 Pearson Education, Inc. All rights reserved.
// return reference to screen
public Screen getScreen() {
} // end method getScreen

// return reference to bank database
public BankDatabase getBankDatabase() {
} // end method getBankDatabase

// abstract method overridden by subclasses
public abstract void execute();

} // end class Transaction

Outline

Transaction.java

Declare abstract method execute

© 2006 Pearson Education, Inc. All rights reserved.