SWE 621:
Software Modeling and Architectural Design

Lecture Notes on Software Design

Lecture 10 - Class Design

Hassan Gomaa

Dept of Computer Science
George Mason University
Fairfax, VA

Steps in Using COMET/UML

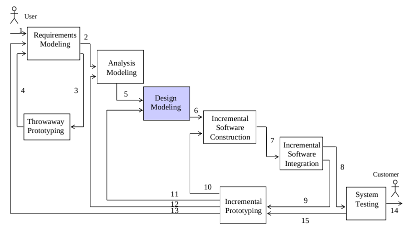
1 Develop Software Requirements Model

2 Develop Software Analysis Model

3 **Develop Software Design Model**

- Design Overall Software Architecture (Chapter 12, 13)
- Design Distributed Component-based Subsystems (Chapter 12-13,15)
- Structure Subsystems into Concurrent Tasks (Chapter 18)
- **Design Information Hiding Classes (Chapter 14)**
- Develop Detailed Software Design

Figure 6.1 COMET object-oriented software life cycle model



Copyright 2006 H. Gomaa

15

SWE 621:
Software Modeling and Architectural Design

Lecture 10 - Class Design

Hassan Gomaa

Reference: H. Gomaa, Chapters 14 - *Software Modeling and Design,* Cambridge University Press, February 2011

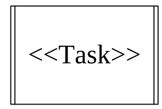
Reference: H. Gomaa, Chapter 15 - *Designing Concurrent, Distributed, and Real-Time Applications with UML,* Addison Wesley Object Technology Series, July, 2000

Design Information Hiding Classes

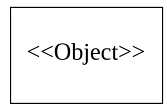
- Design of passive classes
  - Initially determined from Analysis Model
  - Each class hides design decision
  - Encapsulates information
  - Accessed by operations
- Design class operations
  - Primarily from Communication Model
  - Shows direction of message from sender object to receiver object
- Develop class hierarchies using inheritance
  - Subclass inherits attributes & operations from superclass
  - Subclass may add attributes and operations, redefine operations

Active and Passive Objects

- Objects may be **active** or **passive**
- **Active object**
  - **Concurrent Task**
  - Has thread of control



- **Passive object**
  - a.ka. **Information Hiding Object**
  - Has no thread of control
  - Operations of passive object are executed by task
  - Operations execute in task’s thread of control
    - Directly or indirectly



- Software Design terminology
  - **Task** refers to active object
  - **Object** refers to passive object

Example of Information Hiding

- Example of Stack object
- Information hiding solution
  - Hide stack data structure and internal linkage
  - Specify operations on stack data structure
  - Access to stack only via operations
    - push (x), pop (x), empty, full
- Consider
  - Array implementation changed to
  - Linked list implementation

Figure 3.5 Example of Information Hiding

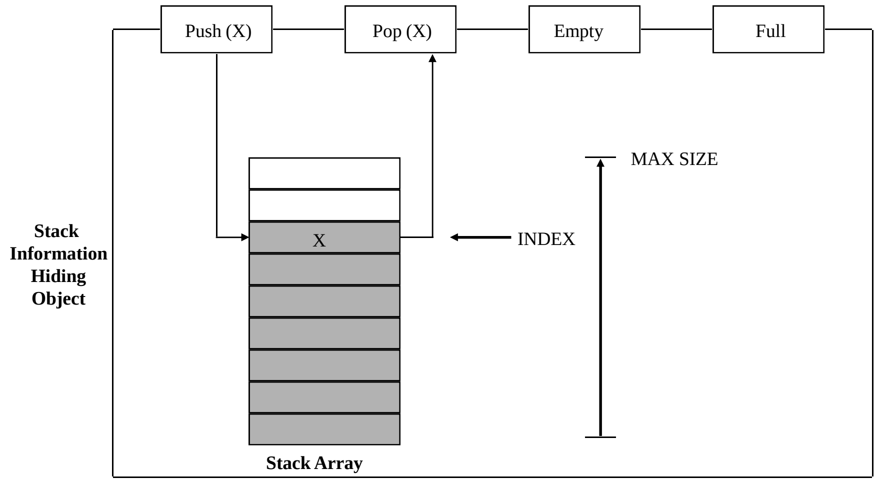
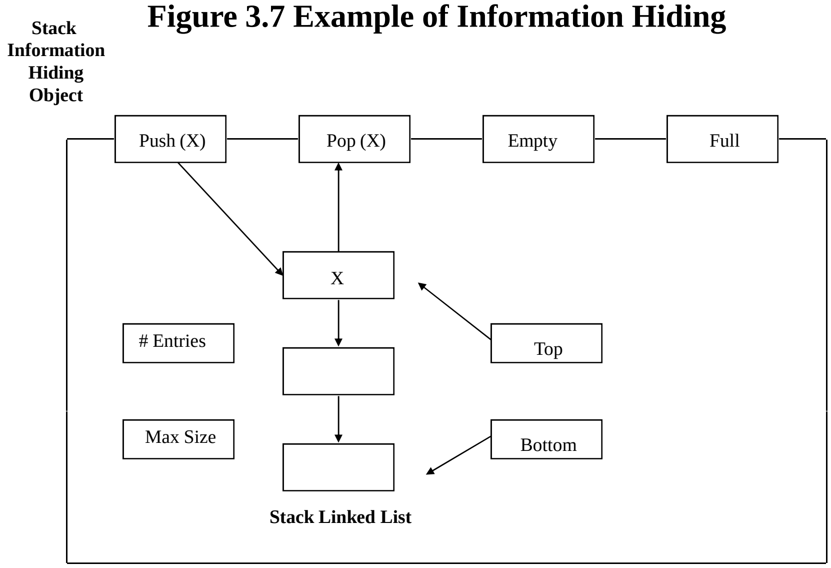


Figure 3.7 Example of Information Hiding



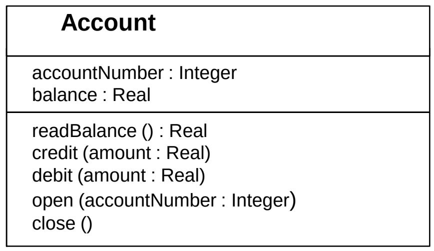
Example of Information Hiding

- Example of Stack object
  - Information hiding solution
- Consider
  - Array implementation changed to
  - Linked list implementation
- Change to stack only impacts Stack object
  - Interface unchanged
    - push (x), pop (x), empty, full
  - Implementation (internals) modified

Classes and Operations

- Class
  - Represents a collection of identical objects (instances)
  - Described by means of attributes (data items)
  - Has one or more operations to access internal data
  - Each object instance can be uniquely identified
- Operation (also known as method)
  - Function or procedure that manipulates values of attributes maintained by object
  - All objects in class have same operations

Figure 3.3 Class with attributes and operations



Design Class Operations

- Design Class Operations from Communication Model
  - Shows direction of message from sender object to receiver object
- Design Class Operations from Finite State Machine Model
  - Statechart actions are mapped to operations
- Design Class Operations from Static Model
  - May be used for entity classes
  - Standard operations
    - Create, Read, Update, Delete
  - Specific operations
    - Based on services provided by class

Information Hiding Class Structuring

- Class Design
  - Initially determined from Analysis Model
  - Each class hides design decision
- Design of Information Hiding Classes
  - Entity classes are categorized further
    - Data abstraction classes
    - Database wrapper classes
- Design class operations
  - Primarily from communication Model
  - Shows direction of message from sender object to receiver object

Data Abstraction Class

- Encapsulates data structure
  - Hides internal structure and content of data structure
  - Attributes provided by static model (class diagram)
- Design Class interface
  - Data accessed indirectly via operations
  - Consider services required by client objects that interact with data abstraction object
  - Consider communication model

Figure 14.2 Example of data abstraction class

Figure 14.2a Analysis model – communication diagram

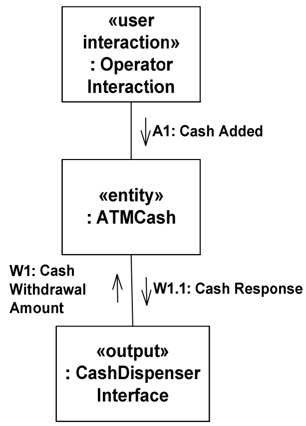


Figure 14.2b Design model – communication diagram

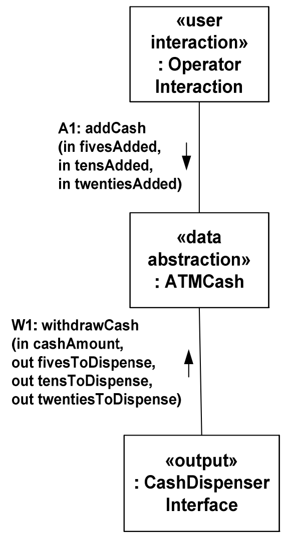


Figure 14.2 Example of data abstraction class

Figure 14.2c Design model - class diagram

«data abstraction» ATMCash
- cashAvailable: Integer = 0 - fives: Integer = 0 - tens: Integer = 0 - twenties: Integer = 0
+ addCash (**in** fivesAdded, **in** tensAdded, **in** twentiesAdded) + withdrawCash (**in** cashAmount, **out** fivesToDispense, **out** tensToDispense, **out** twentiesToDispense)

State Machine Class

- Hides contents of statechart / state transition table
  - Maintains current state of object
- Process Event Operation
  - Called to process input event
  - Depending on current state and conditions
    - Might change state of object
    - Might return action(s) to be performed
- Current State Operation
    - Returns the state stored in state transition table
- If state transition table changes
- Only this class is impacted

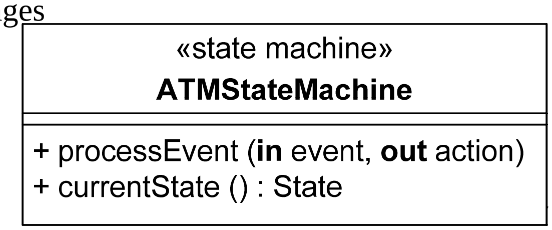


Figure 14.2 Example of State Machine class

Business Logic Class

- Hides business application logic
  - Encapsulate business rules
- Business rules could change
  - Independently of other business logic classes
  - Independently of entity classes
- E.g., Bank Withdrawal Transaction Manager business rules
  - Account must have positive (or zero) balance after withdrawal
  - Maximum daily withdrawal limit is $300

Figure 14.5: Example of business logic class

Figure 14.5a: Analysis model - communication diagram

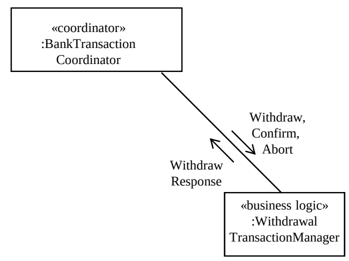


Figure 14.5b: Design model - communication diagram

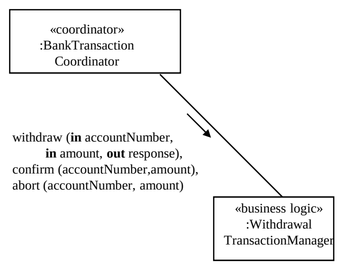
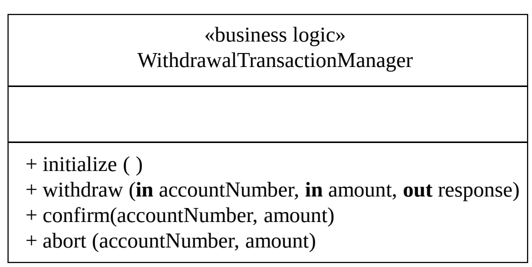


Figure 14.5: Example of business logic class

Figure 14.5c: Design model - class diagram

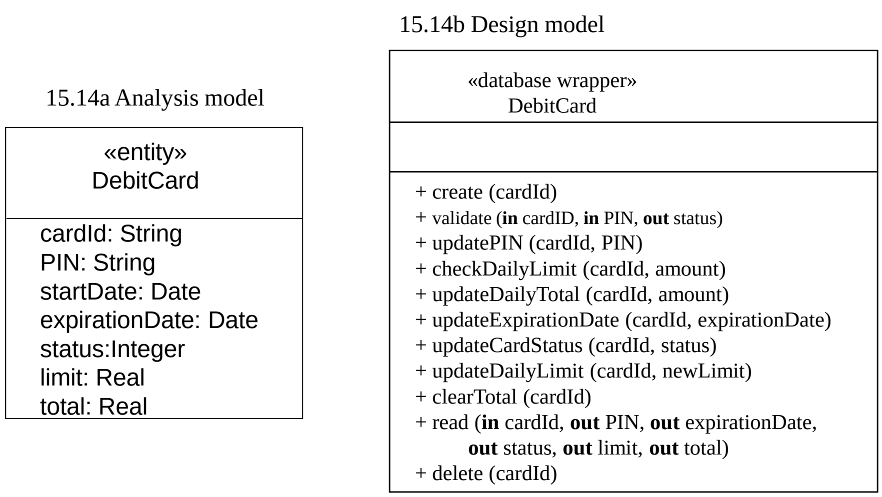


Copyright 2011 H. Gomaa

Database Wrapper Class

- Entity class in Analysis Model
  - Encapsulated data is actually stored in database
- Analysis Model class mapped to
  - Database Wrapper Class
    - Hides interface to database (e.g., relational)
  - Attributes of class mapped to
    - Relation (flat file) stored in database
- Database Wrapper Class
  - Provides OO interface to database
  - Hides details of how to access data in database
    - Hides SQL statements
  - May hide details of one relation or
    - Database view (join of two or more relations)

Figure 15.14: Example of database wrapper class



Relation in relational database :
DebitCard (<u>cardId</u>, PIN, startDate, expirationDate, status, limit, total, *customerId*)

(underline = <u>primary key</u>, italic = *foreign key*)

Inheritance in Design

- Subclass inherits generalized properties from superclass
    - Property is Attribute or Operation
- Inheritance
    – Allows sharing of properties between classes
    – Allows adaptation of parent class (superclass) to form child class (subclass)
- Subclass inherits attributes & operations from superclass
    – May add attributes
    – May add operations
    – May redefine operations

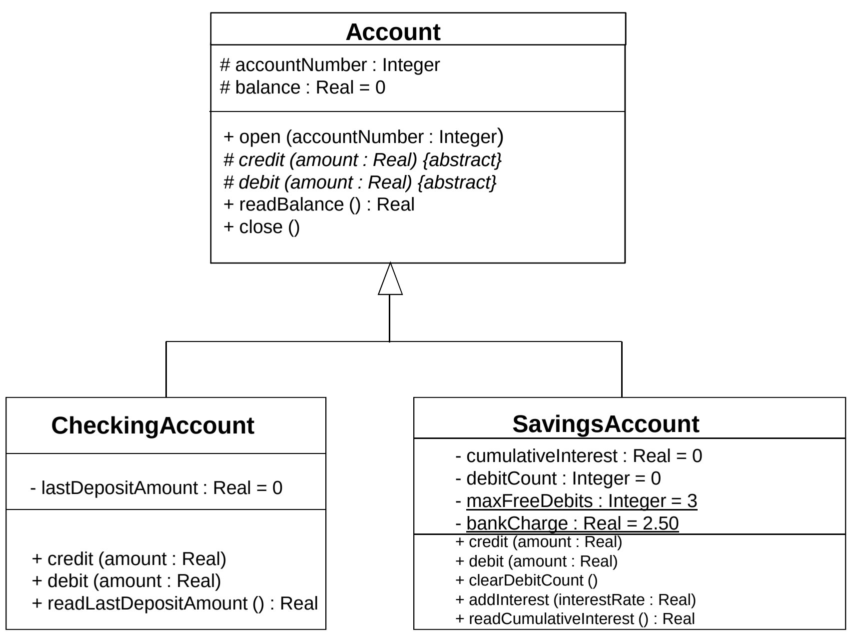
Abstract Class

- Abstract Class
    – Template for creating subclasses
    – Has no instances
    – Only used as superclass
    – Defines common interface for subclasses
- Abstract operation
    – Operation declared in abstract class but not implemented
- Abstract Class defers implementation of some or all of its operations to subclasses
- Different subclasses can define different implementations of same abstract operation

Example of Inheritance

- Attributes of *Account* Superclass
  - *accountNumber*, *balance*
- Operations of *Account* Superclass
  - *open (accountNumber : Integer)*
  - *close ()*
  - *readBalance () :Real*
  - *credit (amount :Real) {abstract}*
  - *debit (amount :Real) {abstract}*

Figure 14.7: Example of superclass and subclass



Example of Inheritance

- Attributes of *Checking Account* Subclass
  - Inherits *accountNumber, balance*
  - Adds *lastDepositAmount*
- Operations of *Checking Account* Subclass
  - Inherits specification and implementation of *open, readBalance, close*
  - Inherits specification of *credit* and *debit* but defines implementation
  - Adds *readLastDepositAmount () : Real*

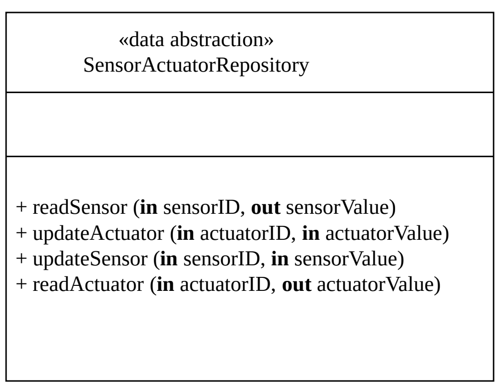
Example of Inheritance

- Attributes of *Savings Account* Subclass
  - Inherits *accountNumber, balance*
  - Adds instance attributes *cumulativeInterest, debitCount*
  - Adds static class attributes *maxFreeDebits, bankCharge*
- Operations of *Savings Account* Subclass
  - Inherits specification & implementation of *open, readBalance, close*
  - Inherits specification of *credit* and *debit* but defines implementation
  - *debit*
    - Debit balance
    - Deduct *bank Charge* if *debit Count* > *max Free Debits*
  - Adds Operations
    - *addInterest (interestRate)* Add daily interest
    - *readCumulativeInterest () :Real*
    - *clearDebitCount ()* Reinitialize debit Count to zero

Class Interface Specification

- Information hidden by class
- Class structuring criterion
- Assumptions made in specifying class
- Anticipated changes
- Superclass (if applicable)
- Inherited Operations (if applicable)
- Operations provided by class
  - Function performed
  - Precondition
  - Postcondition
  - Invariant
  - Input parameters
  - Output parameters
  - Operations used by class (provided by other classes)

Example of class defined by class interface specification



Example of Class Interface Specification

Information Hiding Class: Sensor Actuator Repository

Information Hidden: Encapsulates sensor/actuator data structure. Stores current values of sensors and actuators.

Class structuring criterion: Data abstraction class.

Assumptions: Operations may be concurrently accessed by more than one task.

Anticipated changes: Currently supports Boolean sensors and actuators only. Possible extension to support analog sensors and actuators.

Superclass: None

Inherited operations: None

Operations provided:

1) readSensor (**in** sensorID, **out** sensorValue)

Function: Given the sensor id, returns the current value of the sensor

Precondition: Sensor value has previously been updated.

Invariant: Sensor value remains unchanged.

Postcondition: Sensor value has been read.

Input parameters: sensorID

Output parameters: sensorValue

Operations used: None

2) updateActuator (**in** actuatorID, **in** actuatorValue)

Function: Used to update the value of the actuator in preparation for output

Precondition: Actuator exists.

Postcondition: Actuator value has been updated.

Input parameters: actuatorID, actuatorValue

Output parameters: None

Operations used: None

Example of Class Interface Specification

3) updateSensor (**in** sensorID, **in** sensorValue)

Function: Used to update sensor value with new reading from the external environment

Precondition: Sensor exists.

Postcondition: Sensor value has been updated.

Input parameter: sensorID, sensorValue

Output parameters: None

Operations used: None

4) readActuator (**in** actuatorID, **out** actuatorValue)

Function: Used to read the new value of the actuator to output to the external environment

Precondition: Actuator value has previously been updated.

Invariant: Actuator value remains unchanged.

Postcondition: Actuator value has been read.

Input parameters: actuatorID

Output parameters: actuatorValue

Operations used: None

ATM Client Subsystem - Information Hiding Class Categorization

- Data Abstraction Classes
  - ATM Card
  - ATM Transaction
  - ATM Cash
- State Machine Class
  - ATM Control
- Reference: Chapter 21

Figure 18.13 Task architecture – initial concurrent communication diagram for ATM Client (after task structuring)

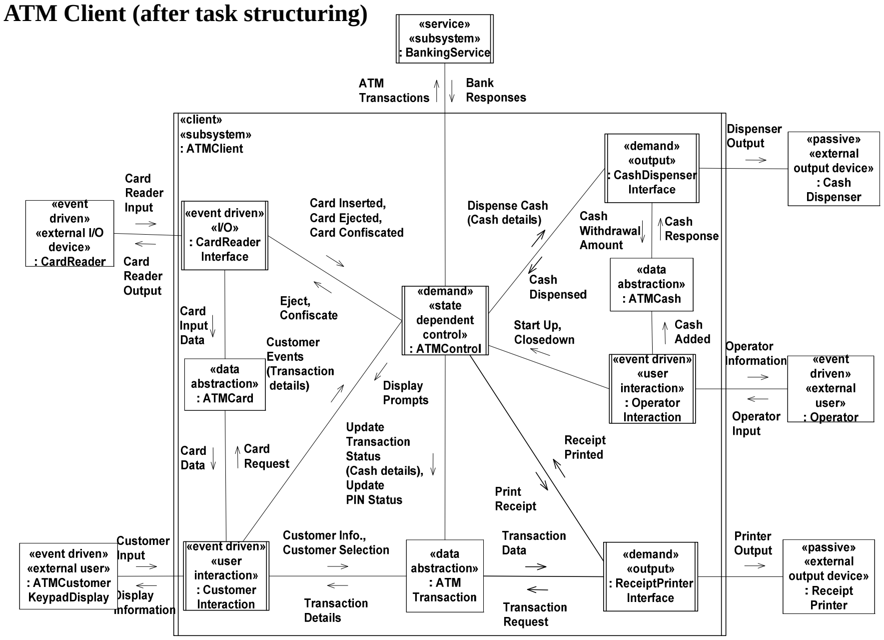


Figure 21.31 Design of ATM Client information hiding classes

«data abstraction»
ATMCard

- cardNumber: String
- startDate: Date
- expirationDate: Date

+ write (in cardId, in startDate, in expirationDate)
+ read (out cardId, out startDate, out expirationDate)

«data abstraction»
ATMCash

- cashAvailable: Integer = 0
- fives: Integer = 0
- tens: Integer = 0
- twenties: Integer = 0

+ addCash (in fivesAdded, in tensAdded, in twentiesAdded)
+ withdrawCash (in cashAmount, out fivesToDispense, out tensToDispense, out twentiesToDispense)

«data abstraction»
ATMTransaction

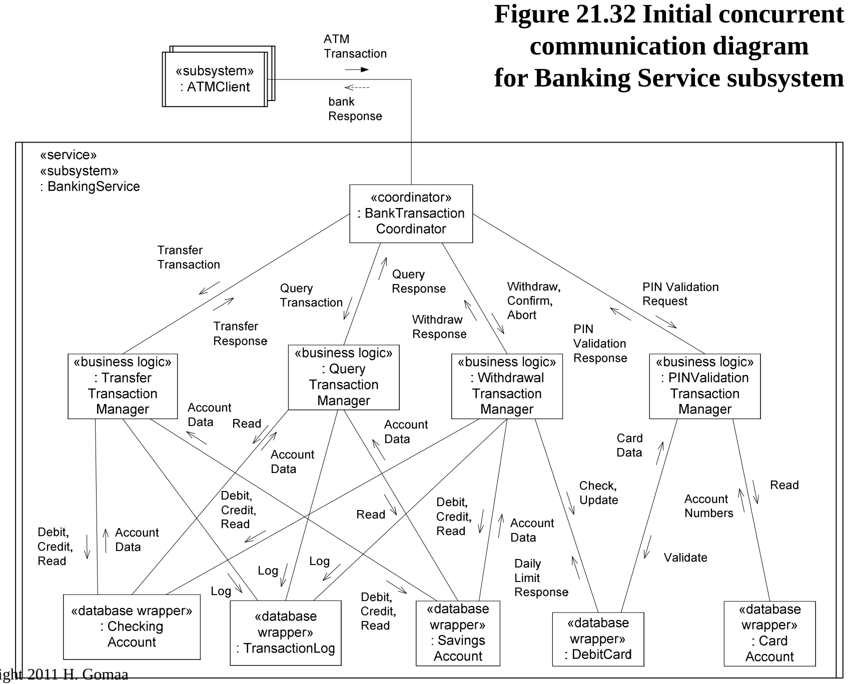
- transactionId: String
- cardId: String
- PIN: String
- date: Date
- time: Time
- amount: Real
- balance: Real
- PINCount: Integer
- status: Integer

+ updateCustomerInfo (cardData, PIN)
+ updateCustomerSelection (in selection, out transactionData)
+ updatePINStatus (status)
+ updateTransactionStatus (amount, balance)
+ read (out transactionData)

«state machine»
ATMStateMachine

+ processEvent (in event, out action)
+ currentState () : state

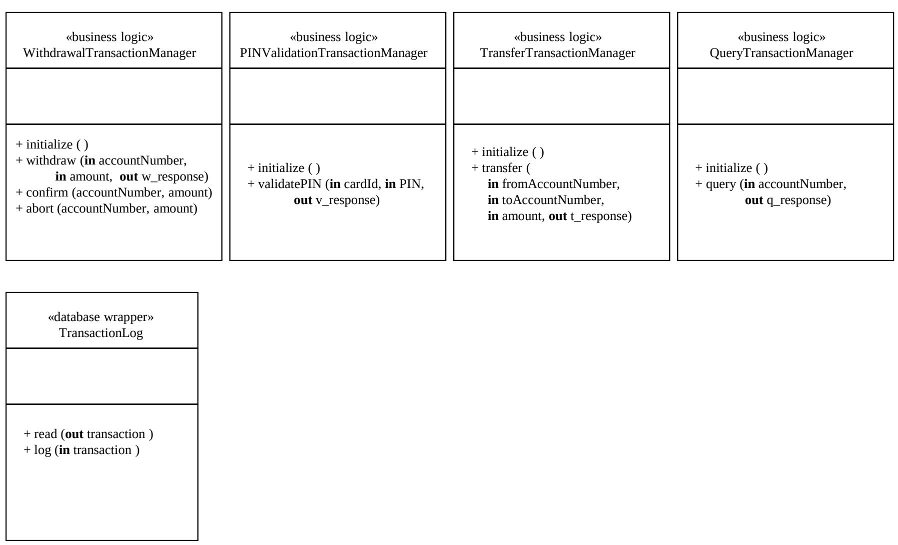
Figure 21.32 Initial concurrent communication diagram for Banking Service subsystem



Bank Server Subsystem - Information Hiding Class Categorization

- Business Logic Classes
  - PIN Validation Transaction Manager
  - Query Transaction Manager
  - Transfer Transaction Manager
  - Withdrawal Transaction Manager
- Database Wrapper Classes
  - Checking Account
  - Savings Account
  - Debit Card
  - Card Account
  - Transaction Log
- Reference: Chapter 21

Figure 21.34 Banking Service information hiding classes



Figures21.33 Banking Service information hiding classes

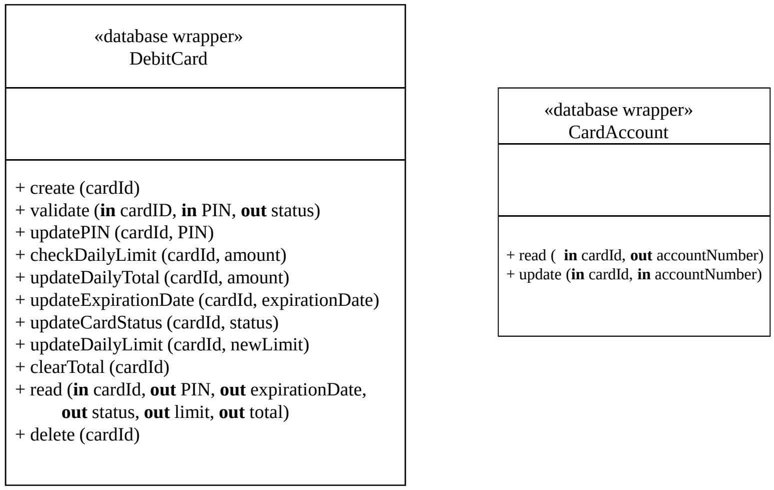


Figure 21.33 Banking Service information hiding classes

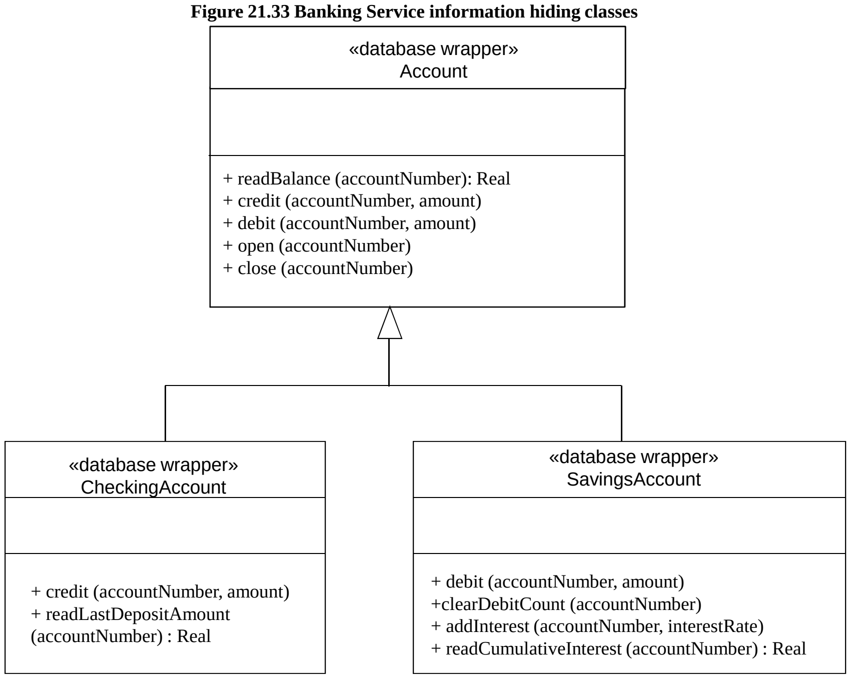
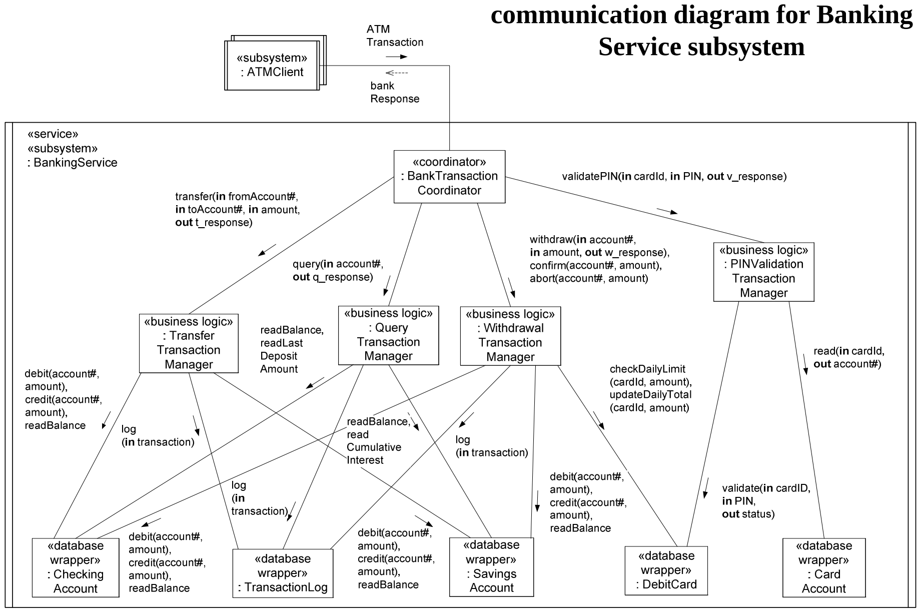


Figure 21.35 Revised concurrent communication diagram for Banking Service subsystem



Steps in Using COMET/UML

1 Develop Software Requirements Model
2 Develop Software Analysis Model
3 **Develop Software Design Model**
   – Design Overall Software Architecture (Chapter 12, 13)
   – Design Distributed Component-based Subsystems (Chapter 12-13,15)
   – Structure Subsystems into Concurrent Tasks (Chapter 18)
   – **Design Information Hiding Classes (Chapter 14)**
   – Develop Detailed Software Design

Figure 6.1 COMET object-oriented software life cycle model