Object-oriented design

Part 2: OO tools & UML

CRC cards

- Design tool & method for discovering classes, responsibilities, & relationships
- Record on note card:
  - class name & purpose
  - general responsibilities
  - name(s) of class(es) this class depends on to fulfill its responsibilities

Why use cards?

- Could record this information using paper, whiteboard, etc.
- Advantages of cards:
  - portable: can easily group & rearrange cards to illustrate/discover relationships between classes
  - disposable: easily modified or discarded as design changes

Example CRC card for ATM

Class: ATM (performs financial services for a bank customer)

Responsibilities	**Collaborations**
• create & initialize transactions	Transaction
• display greeting	User Message
• display main menu	Menu
• tell cancel key to reset	Cancel Key
• check for a cancel	Cancel Key
• eject receipt	Receipt Printer
• eject bank card	Bank Card Reader

- Don’t have to list collaborators on same line as responsibilities - but doesn’t hurt to do so
- This class is unusual for two reasons:
  - large # of responsibilities
  - fulfills all responsibilities via collaboration

More ATM examples

Class: Account (represents account in bank database)

Responsibilities	**Collaborations**
• Know account balance	
• Accept deposits	
• Accept withdrawals	

Class: Transaction (performs financial service & updates account)

Responsibilities	**Collaborations**
• Execute financial transaction	
• Gather information	Menu, Form, User Message
• Remember data relevant to transaction	
• Commit transaction to database	Account
• Check to see if cancel key has been pressed	Cancel Key

Some notes on CRC cards

- Cards are meant to be transitory tools for proposing designs
- Meant as discovery tool, not archival information
- For design documentation, use UML diagrams accompanied by explanatory text

Intro to UML

- UML: unified modeling language
- System for graphically representing & manipulating an object-oriented software system
- Both a representation of design & tool to assist in design process

Class Diagram

- 3 parts: class name, attribute, methods - generally listed in that order
- Don’t have to list all attributes or methods - usually just the most important
- For some diagrams, especially those depicting relationships among classes, can omit all but the class name

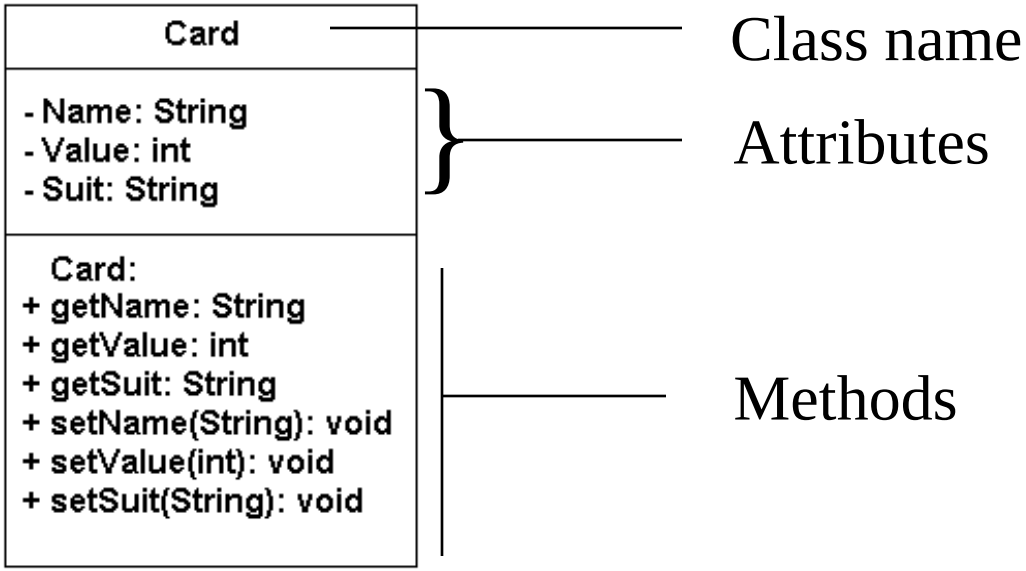
Attributes

- Attributes generally correspond to data members of a class
- Can include the following information in a class diagram:
  - access designation:
    + means public
    - means private
    # means protected
  - name
  - data type

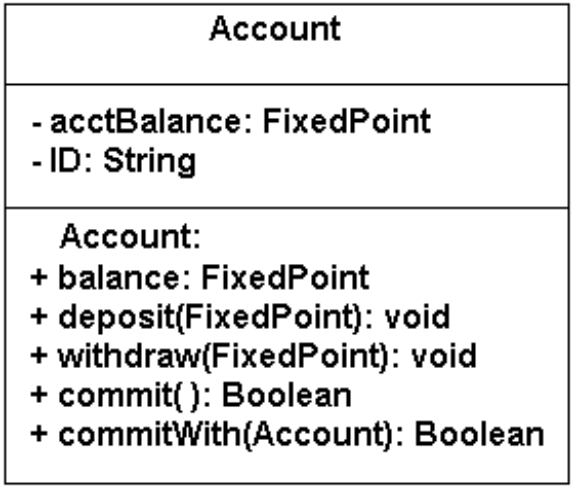
Methods in Class Diagram

- Constructor
  - special method with same name as class
  - has no return type & no access designation
- Other methods
  - access designation (+ or -)
  - method name
  - parameter list, if needed - in parentheses
  - return type after colon

Example: playing card class



Example from ATM



Depicting class relationships

- Relationships between classes are represented by lines between (abbreviated) class diagrams
- Line type (solid vs. dotted) and arrowhead type distinguish between various kinds of relationships

Aggregation

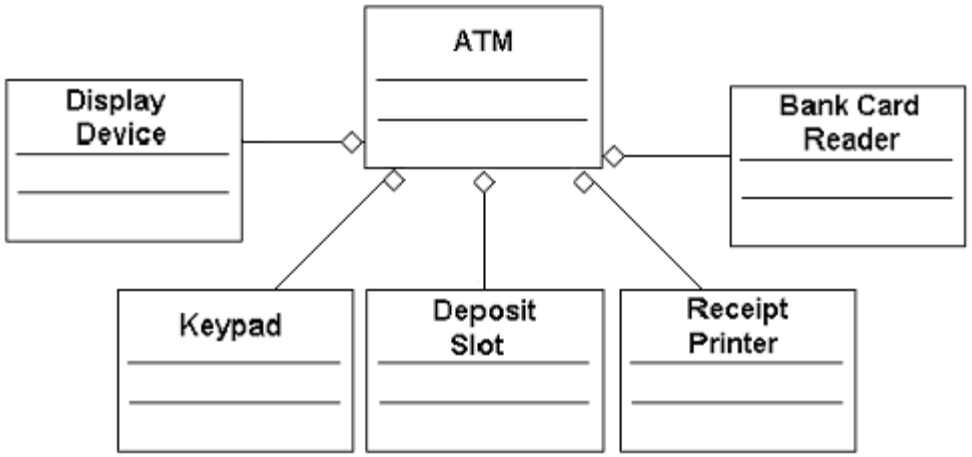
- Aggregation is used when a class is made up of class components
  - For example, a car has an engine, an electrical system, etc.
  - A stereo has a receiver, a CD player, speakers, etc.

Aggregation & UML

- Aggregations are represented with diamond-headed lines connecting an aggregate class to its components
- The diamond is at the aggregation end
- Can use multiplicity notation to depict the number of instances of each component

ATM Example

Although we didn't choose to model it this way, an ATM can be thought of as an aggregate of several things, for example, display device, keypad, deposit slot, receipt printer and bank card reader

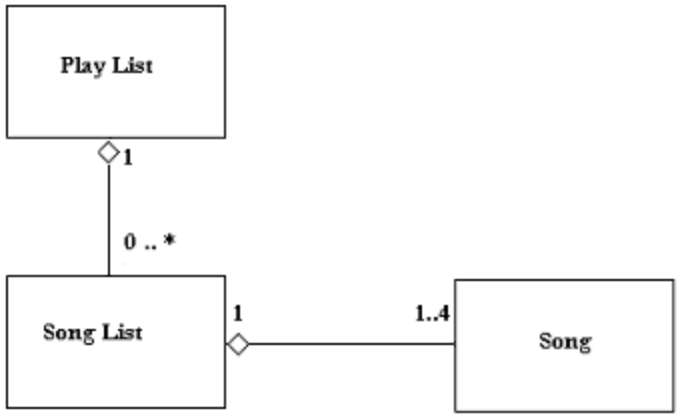


Multiplicity notation

- A number or symbol near either end point of a connecting line indicates multiplicity
- Common notations include:
  - 0 or more: *
  - 1 or more: 1 .. *
  - 0 or 1: 0 .. 1
  - exactly 1: 1

Multiplicity examples

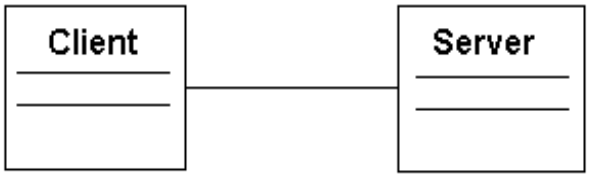
From voice mail system: a message queue can contain several messages



JukeBox:
A SongList (songs picked by an individual user) can have 1-4 songs, depending on the amount of money deposited; the PlayList consists of all the SongLists queued up as other users' songs are played

Associations

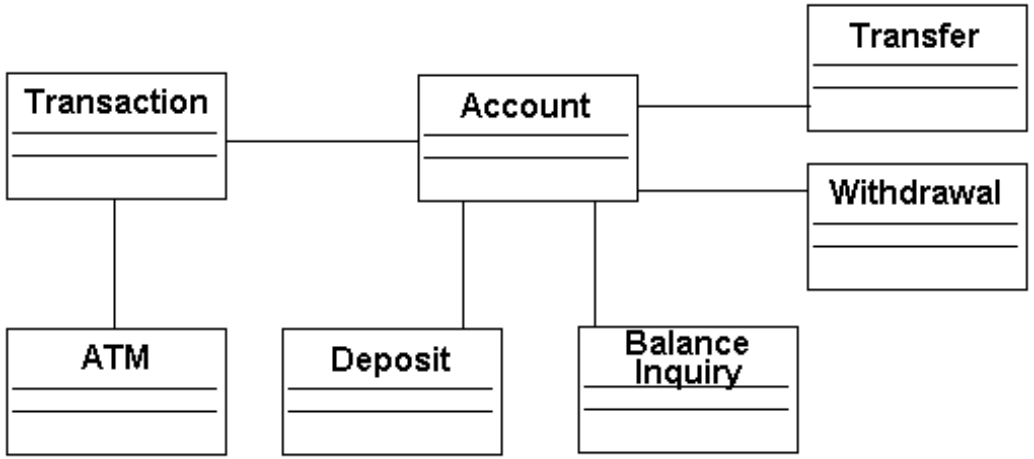
- Represent services provided between classes - a.k.a. collaborations
- Association is represented by a line between client & server class diagrams
- There is no indication of direction of flow of service



Associations

- Can add role designations to lines in association
- Helps clarify bidirectional relationship before a final decision is made about which class actually manages the pertinent information

Association Example from ATM



Contracts supported:

1. Access & modify account balance
2. Commit results to database
3. Execute a financial transaction

Each line represents a collaboration:

- ATM collaborates with transaction in fulfillment of ATM responsibility: create & initiate transactions
- Transaction collaborates with Account in fulfillment of transaction responsibility: commit transaction to database
- Various transaction types collaborate with Account to perform their responsibilities

Notes on Graph Complexity

- Diagram needs to depict system, but also needs to be readable
- All collaborations could be represented on a single association graph, but in a complex system the diagram would be so complicated as to be unusable

Notes on Graph Complexity

- Best approach is to create multiple diagrams that represent collaborations in fulfillment of a single contract or set of related contracts
- Previous example represented all collaborations in support of 3 contracts

Inheritance

- UML represents inheritance relationships between classes as a line beginning from a subclass diagram and ending in an arrow pointing to the superclass
- Abstract classes are represented using {} around their names

Example from ATM Design