Introduction to Class Modeling

Fall 2018

1

Overview

- Purpose of object analysis
- Revisit object orientation concepts.
- What are analysis classes?
- Identify and classify important objects
- Connect objects using relationships

2

Activities in Class Modeling

1. Identify and define conceptual or analysis classes
   a. Identify classes identification (textual analysis, domain experts).
   b. Identify attributes and operations
   c. Identify associations between classes
   d. Identify multiplicities
   e. Identify roles
   f. Identify constraints

3

Purpose of Object Analysis

- To produce an Analysis Model of the system's desired outcome:
  - This model should be a statement of **what** the system does **not how** it does it.
  - We can think of the analysis model as a "first-cut" or "high level" design model.
  - **It is in the language of the business.**
- Based on our object analysis, we identify **analysis classes**.

4

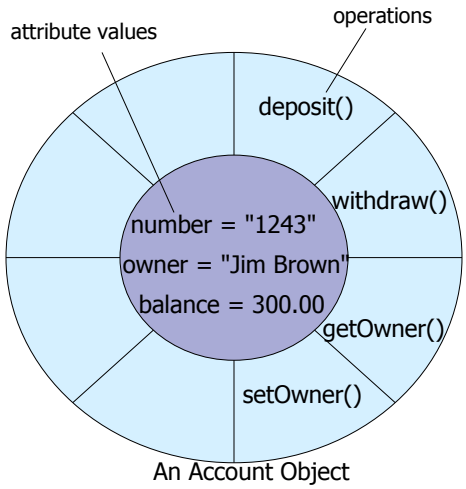
Review Object Orientation Concepts

- **Objects** consist of data (attributes) and function (methods) packaged together in a reusable unit.
- Every object is an instance of some *class* which defines the common set of *features* (attributes and operations) shared by all of its instances.
- All objects have:
  - ***Identity***: Each object has its own unique identity and can be accessed by a unique handle.
  - ***State***: This is the actual data values stored in an object at any point in time.
  - ***Behavior***: The set of operations that an object can perform.

5

Review Object Orientation Concepts

- Data is hidden inside the object. The only way to access the data is via one of the operations.
- This is *encapsulation* or *data hiding* and it is a very powerful idea. It leads to more robust software and reusable code.



6

Messaging

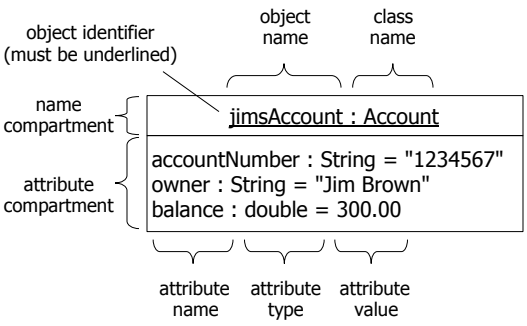
- In OO systems, objects send messages to each other over links.
- These messages cause an object to invoke an operation.

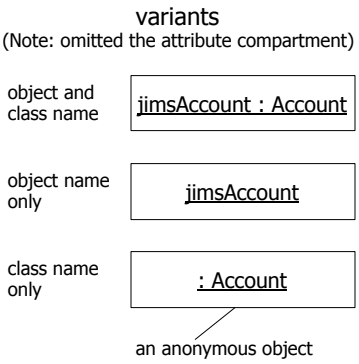
The Bank object sends the message "withdraw 150.00" to an Account object.

The Account object responds by invoking its withdraw operation. This operation decrements the account balance by 150.00.

7

UML Object Syntax

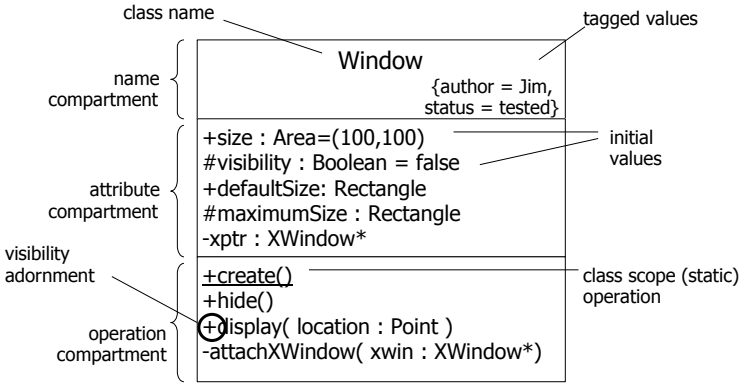




- All objects of a particular class have the same set of operations. They are not shown on the object diagram, they are shown on the class diagram (see later).
- Attribute types are often omitted to simplify the diagram.
- Naming convention:
  - object and attribute names in lowerCamelCase
  - class names in UpperCamelCase

8

UML Class Notation

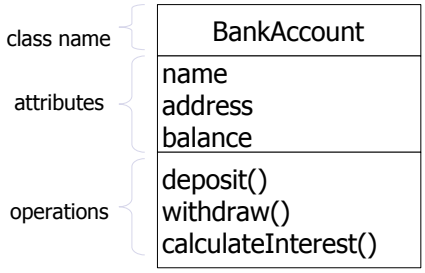


- Classes are named in UpperCamelCase
- Use descriptive names that are nouns or noun phrases
- Avoid abbreviations!

9

What are analysis classes?

- Analysis classes represent a clear abstraction in the problem domain.
- They may ultimately be refined into one or more design classes.



10

Analysis Classes

- Analysis classes have:
  - A very “high level” set of attributes.
  - They *indicate* the attributes that the design classes ***might*** have.
  - Operations that specify (at a high level) the key services that the class must offer.
- Analysis classes must map onto real-world business concepts. Very important!

11

What makes a good analysis class?

- Its name reflects its intent.
- It is a *clear abstraction* that models **one specific** element of the problem domain.
  - It maps onto a clearly identifiable feature of the problem domain.

12

Identify Important Objects

- **Boundary objects** – these are actors of our system
  - **A person with a specific role.**
  - **A specific external system that our system interfaces with.**
  - **A specific time that triggers a use case.**
- **Business objects** – represent “real world” objects and without them, we have no “business!” and there is no reason to develop the application system.
- **Container objects – contain other object and it has “Part Of” relationship to the other objects.**
  - Example: A purchase order is a container object because it must contain a vendor object and **one or more items (item otject).**

13

Important Objects

- **Utility objects** – these are “helper" objects, help business objects communicate and perform tasks.
  - Examples: we need utitlity objects to help validate object data, display information on screen, send data to a printer, etc.
- **Persistent objects** – they “live” after our application ends. Usually, they are business objects that we create with a set of attributes.

14

How do you find classes?

- Review your use case model and use case specifications
- Perform noun/verb analysis:
  - Nouns are candidate classes.
  - Verbs are candidate *responsibilities.*
- Perform CRC card analysis
  - A brainstorming technique using sticky notes.
  - Useful for brainstorming, Joint Application Development (JAD) and Rapid Application development (RAD).

15

Finding Relationships

- What is a relationship?
- What is a link?
- What is an association?
- Association syntax
- Multiplicity
- Reflexive associations
- Navigability
- Summary

16

What is a relationship?

- A *relationship* is a connection between modelling elements.
- We will look at:
  - *Links* between **objects**
  - *Associations* between **classes**

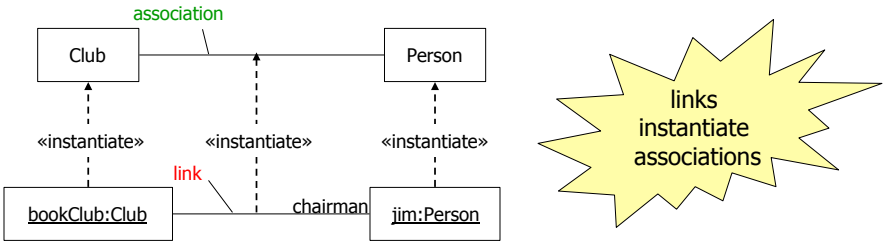
17

What is a link?

- Links are connections between objects
  - Think of a link as a telephone line connecting you and a friend.
  - You can send messages back and forth using this link.
- Links allow objects to communicate
  - Objects send messages to each other via links.
  - Messages invoke operations.

18

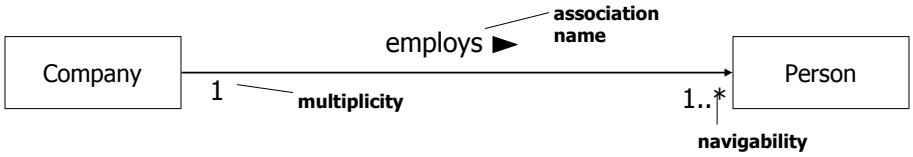
What is an association?

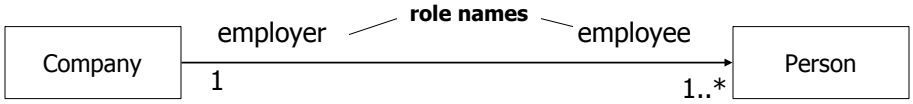


- Associations are relationships between classes.
- Associations between classes indicate that there are links between objects of those classes.
- A link is an instantiation of an association just as an object is an instantiation of a class.

19

Association syntax

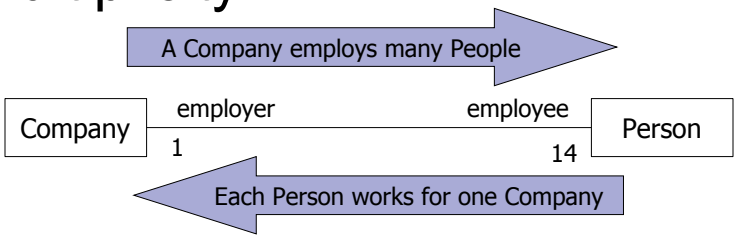




- An association can have role names *or* an association name.
- The black triangle indicates the direction in which the association name is read: "Company employs many Person(s)"
- Association names are in lowerCamelCase.

20

Multiplicity



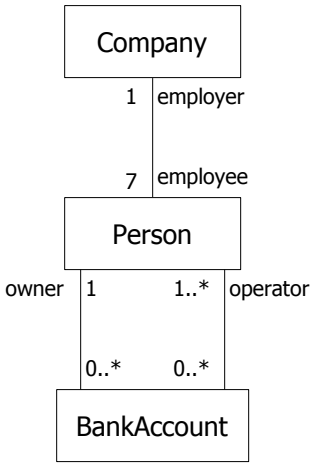
- Multiplicity is a constraint that specifies the number of objects that can participate in a relationship at *any point in time.*
- If multiplicity is not explicitly stated in the model then it is undecided – *there is no default multiplicity.*

multiplicity syntax: minimum..maximum	
0..1	zero or 1
1	exactly 1
0..*	zero or more
*	zero or more
1..*	1 or more
1..6	1 to 6

21

Multiplicity exercise

- How many
  - Employees can a Company have?
  - Employers can a Person have?
  - Owners can a BankAccount have?
  - Operators can a BankAccount have?
  - BankAccounts can a Person have?
  - BankAccounts can a Person operate?



22

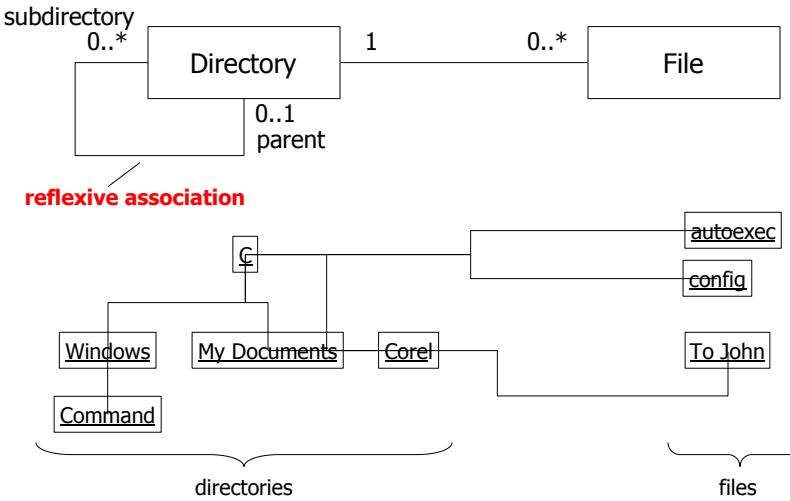
Exercise

- Model a computer file system. Here are the minimal facts you need:
  - The basic unit of storage is the file
  - Files live in directories
  - Directories can contain other directories
- Use your own knowledge of a specific file system (e.g. Windows XP or UNIX) to build a model

Hint: a class can have an association to itself!

23

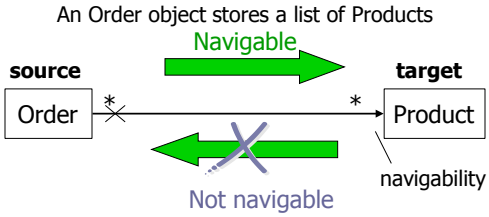
Reflexive Associations

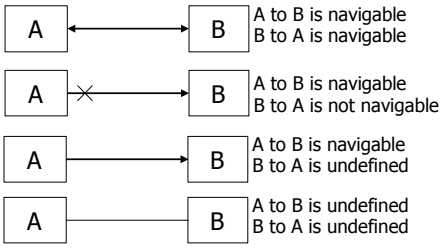


24

Navigability

- Navigability indicates that it is possible to traverse from an object of the *source* class to objects of the *target* class.
- Even if there is *no* navigability, it might still be possible to traverse the relationship via some indirect means.

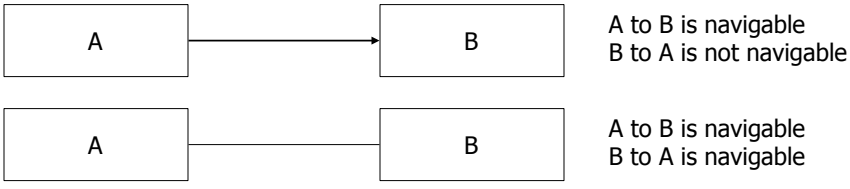




25

Navigability - standard practice

- Strict UML 2 navigability can clutter diagrams so the UML standard suggests three possible modeling idioms:
  - Show navigability explicitly on diagrams
  - Omit all navigability from diagrams
  - Omit crosses from diagrams
    - bi-directional associations have no arrows
    - unidirectional associations have a single arrow
    - you can't show associations that are not navigable in either direction (not useful anyway!)



26

What we have learned so far…

- In this section we have looked at:
  - Links – relationships between objects
  - Associations – relationships between classes
    - association names
    - role names
    - multiplicity
    - navigability

27

To Learn More…

Object-Oriented Software Engineering, Second Edition –
Bernd Bruegge, Allen H. Dutoit

UML 2 and The Unified Process, Second Edition –
Jim Arlow and Ila Newstadt

The Unified Modeling Language User Guide, *Second Edition – Grady Booch, James Rumbaugh, and Ivar Jacobson*

28