Software Requirements Analysis - Conceptual Model

Zhiming Liu
zhimingliu88@swu.edu.cn
http://computer.swu.edu.cn
Centre for Research and Innovation in Software Engineering (RISE)
School of Computer and Information Science
Southwest University
Chongqing, China

Conceptual Models

Objectives

- understand the concepts of **classes**, **associations** and **attributes**

Conceptual Models

Objectives

- understand the concepts of **classes**, **associations** and **attributes**
- identify classes, associations and attributes in the problem domain

Conceptual Models

Objectives

- understand the concepts of **classes**, **associations** and **attributes**
- identify classes, associations and attributes in the problem domain
- understand characteristics of **objects**

Conceptual Models

Objectives

- understand the concepts of **classes**, **associations** and **attributes**
- identify classes, associations and attributes in the problem domain
- understand characteristics of **objects**
- define **class diagrams** and their use for describing a conceptual model

Conceptual Models

Objectives

- understand the concepts of **classes**, **associations** and **attributes**
- identify classes, associations and attributes in the problem domain
- understand characteristics of **objects**
- define **class diagrams** and their use for describing a conceptual model
- define **object diagrams**

Conceptual Models

Objectives

- understand the concepts of **classes**, **associations** and **attributes**
- identify classes, associations and attributes in the problem domain
- understand characteristics of **objects**
- define **class diagrams** and their use for describing a conceptual model
- define **object diagrams**

Artifact: **Conceptual Class Diagram**

- Models the concepts, objects, their relations and attributes
- Models the structure in the problem domain
- Relevant to the use cases

Conceptual Models

Objectives

- understand the concepts of **classes**, **associations** and **attributes**
- identify classes, associations and attributes in the problem domain
- understand characteristics of **objects**
- define **class diagrams** and their use for describing a conceptual model
- define **object diagrams**

Artifact: Conceptual Class Diagram

- Models the concepts, objects, their relations and attributes
- Models the structure in the problem domain
- Relevant to the use cases

Modelling real world concepts, objects, their properties and relations

Classes Model Concepts

- Identification of classes in the domain and creation of a conceptual model are the most typical activities in OOA
  - A **conceptual model** illustrates meaningful **concepts** in the problem domain

Classes Model Concepts

- Identification of classes in the domain and creation of a conceptual model are the most typical activities in OOA
  - A **conceptual model** illustrates meaningful **concepts** in the problem domain
- Informally,
  - a **concept** is an idea, thing(s), or a **class** of **objects**
  - a **class** represents a **concept**

Classes Model Concepts

- Identification of classes in the domain and creation of a conceptual model are the most typical activities in OOA
  - A **conceptual model** illustrates meaningful **concepts** in the problem domain
- Informally,
  - a **concept** is an idea, thing(s), or a **class** of **objects**
  - a **class** represents a **concept**
- **Formally**, a concept is represented in terms of its **symbol**, **intension** and **extension**:
  - **symbol**: words representing a concept, used when talking about the concept
  - **intension**: the definition of a concept
  - **extension**: the set of **instances** to which the concept applies

Example 1: Module

symbol:

Example 1: Module

symbol: Module

Example 1: Module

symbol: Module

intention:

Example 1: Module

symbol: Module

intention: a course offered as a component of a degree,

Example 1: Module

symbol: Module

intention: a course offered as a component of a degree,

extension:

Example 1: Module

symbol: Module

intention: a course offered as a component of a degree,

extension: e.g. the module with code CMP5215 ...

Example 2: Student

symbol:

Example 2: Student

symbol: Student

Example 2: Student

symbol: Student

intention:

Example 2: Student

symbol: Student

intention: a person who takes modules at a university

Example 2: Student

symbol: Student

intention: a person who takes modules at a university

extension:

Example 2: Student

symbol: Student

intention: a person who takes modules at a university

extension: e.g. students with names John Smith, James Brown, Peter Butcher ...

Example in POST: Sale

symbol: Sale

Example in POST: Sale

symbol: Sale

intention: represents the event of a purchase

Example in POST: Sale

symbol: Sale

intention: represents the event of a purchase

extension: sale1, sale2 ...

Attributes of Concepts

An **attribute** of a concept (class) is the name representing a property of its instances

1. **Student**: name, age, student number ...
2. **Module**: title, code, credit ...
3. **Sale**: date, time, total ...

Object-Oriented Terminology

- A **class** represents a **concept**
- A class has **attributes** representing properties of its objects
- An **object** of a class is an **instance** of the corresponding concept
- Each object has a **value** for each **attribute**
- A **class** defines a set of **objects** which have **common types of properties**

Object-Oriented Terminology

- A **class** represents a **concept**
- A class has **attributes** representing properties of its objects
- An **object** of a class is an **instance** of the corresponding concept
- Each object has a **value** for each **attribute**
- A **class** defines a set of **objects** which have **common types of properties**

Examples

- an instance of Student: name = John Smith, age = 21 ..

Object-Oriented Terminology

- A **class** represents a **concept**
- A class has **attributes** representing properties of its objects
- An **object** of a class is an **instance** of the corresponding concept
- Each object has a **value** for each **attribute**
- A **class** defines a set of **objects** which have **common types of properties**

Examples

- an instance of Student: name = John Smith, age = 21 ..
- an instance of Module: title = Software Design, code = CMPS5212, credit = 15

Object-Oriented Terminology

- A **class** represents a **concept**
- A class has **attributes** representing properties of its objects
- An **object** of a class is an **instance** of the corresponding concept
- Each object has a **value** for each **attribute**
- A **class** defines a set of **objects** which have **common types of properties**

Examples

- an instance of Student: name = John Smith, age = 21 ..
- an instance of Module: title = Software Design, code = CMPS5212, credit = 15
- an instance of Sale: date = 20.03.2008, time = 1500, total = 103

Object-Oriented Terminology

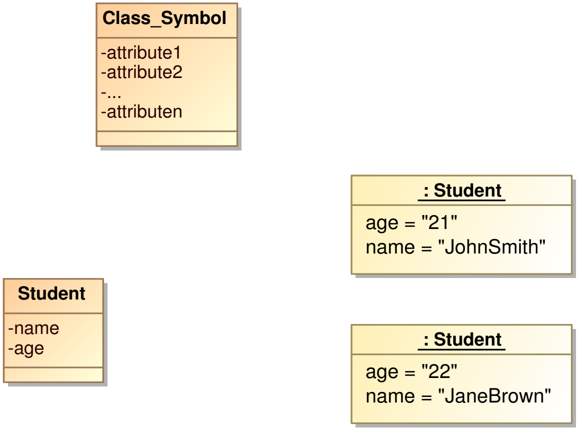
- A **class** represents a **concept**
- A class has **attributes** representing properties of its objects
- An **object** of a class is an **instance** of the corresponding concept
- Each object has a **value** for each **attribute**
- A **class** defines a set of **objects** which have **common types of properties**

Examples

- an instance of Student: name = John Smith, age = 21 ..
- an instance of Module: title = Software Design, code = CMPS5212, credit = 15
- an instance of Sale: date = 20.03.2008, time = 1500, total = 103

This textual representation is not easy to read or understand

Modelling Notation in UML



Focus on Definite Objects and Properties

We do not consider "fuzzy" concepts or models, such as

- King CAO Rui of Kingdom Wei in the Three Kingdom Time ask best painters to produce a portray of his mother
- He described the lady as

  > *She looked as if a Dan in her mouth. Her beautiful teeth were as white as pearl buttons. Her hands looked like young weeds. Her fingers looked like spring onions.*
  > *Her smile looked like morning glow, and her worries looked waves of lotus flower buds.*

- The painters were "modellers", the portrays were models, and the King was the client and person to validate.
- Can CAO Rui validate the models?

口若含丹，齿若编贝：
手如柔荑，指如青葱。

一笑如太阳升朝霞：
一愁如芙蕖染绿波。

Four Defining Features of Objects

1. **object's identity**: every object is distinguishable from every other, even if two objects have exactly the same properties

Four Defining Features of Objects

1. **object's identity**: every object is distinguishable from every other, even if two objects have exactly the same properties
2. **object's states**: every object has some property at any moment of time

Four Defining Features of Objects

1. **object's identity**: every object is distinguishable from every other, even if two objects have exactly the same properties
2. **object's states**: every object has some property at any moment of time
3. **object's persistence**: every object has a life time (**static nature of the system**)

Four Defining Features of Objects

1. **object's identity**: every object is distinguishable from every other, even if two objects have exactly the same properties
2. **object's states**: every object has some property at any moment of time
3. **object's persistence**: every object has a life time (**static nature of the system**)
4. **object's behaviour**: an object may act on other objects and/or be acted on by other objects (**dynamic nature of the system**)

Four Defining Features of Objects

1. **object's identity**: every object is distinguishable from every other, even if two objects have exactly the same properties
2. **object's states**: every object has some property at any moment of time
3. **object's persistence**: every object has a life time (**static nature of the system**)
4. **object's behaviour**: an object may act on other objects and/or be acted on by other objects (**dynamic nature of the system**)

An object may be in different states at different moments of time, and may behave differently in different states

Characterisation of Objects and Classes

- An **object** is an entity with **identity**, **state** and **behaviour**
- A **class** is a description of a collection of objects, **sharing structure**, **behavioural pattern**, and **attributes**

Characterisation of Objects and Classes

- An **object** is an entity with **identity**, **state** and **behaviour**
- A **class** is a description of a collection of objects, **sharing structure**, **behavioural pattern**, and **attributes**

Example

- A **car** is an entity. It has an identity and a life time. It can be repaired, owned, driven by a **person**. It can transport other objects from one place to another
- All **cars** share structure, behavioural pattern, and attributes
- **Car** is the class defines all the car objects

Capture Classes

There are usually two strategies to identify concepts

1. Identify the **noun phrases** in the textual descriptions of a problem domain
   - the client's requirements descriptions, the initial investigation reports
   - The **expanded use cases**

Capture Classes

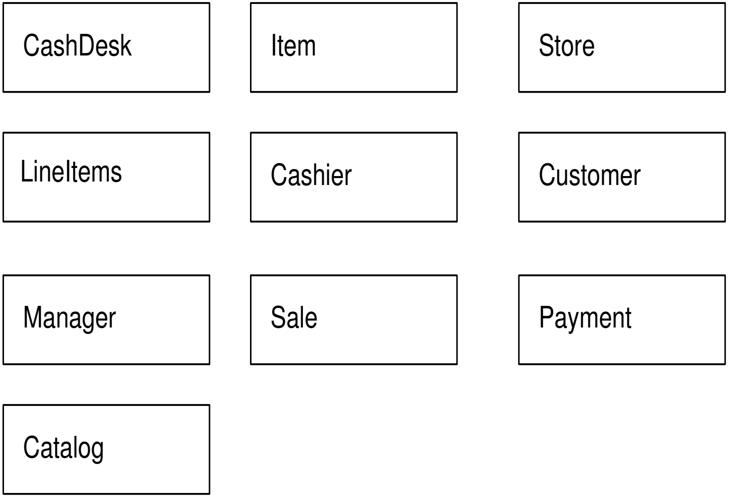
There are usually two strategies to identify concepts

1. Identify the **noun phrases** in the textual descriptions of a problem domain
   - the client's requirements descriptions, the initial investigation reports
   - The **expanded use cases**
2. Find concepts with **concept category lists**
   - **physical or tangible objects**, such as cash desk, airplane, car, house, sheep, etc.
   - **places** such as store, office, airport, etc
   - **roles** that a person in an organisation, such as cashier, pilot, professor, student, etc.

Remarks

1. Mechanical noun-to-concept mapping is not possible,
2. Words in natural languages are ambiguous, and
3. Nouns may include concepts which are about either *attributes*, *events*, *operations* which should **not** be modelled as *classes*.

Initial Class Diagram for POST



The Need-to-Know Principle

There are many noun phrases, such as **receipt** and **price**, that are not included, as

1. there is not indication from the use case there is a need to store, manipulate or use a **receipt**
2. the defining features of objects, such as **identity** do not apply to **price**

The first is about the need-to-know policy and the second is about features of objects and attributes

Example: Buyer-Seller System

- **Order:** the request from a buyer for a number of items. It has a date and an address for delivery
- **Item:** each item “takes a line” in an order
- **Invoice:** a request for payment sent from the seller to a buyer in response to an order
- **Account:** An invoice is paid by transferring money from the buyers account to the seller’s account

Identify and write a number of use cases

Example: Library System

- **Member:** one who has registered to use the library services
- **Publications:** the library has publications on different subjects
- **Copy:** the library may have more than one copy of a publication
- **Loan:** the record/contract about that a member has borrowed a copy of a book
- **Invoice:** a request for payment sent to a member for an over due item
- **Reservation:** When all copies of a publication are on loan, a member can reserve it

Consider classes in the context of use cases

Example: Management System for Hair Salon

- **Customer:** someone who receives treatment
- **Appointment:** agreed time of a treatment for a customer
- **Appointment Book:** contains some appointments
- **Employee:** someone working in the salon
- **Treatment:** to a customer, such as "Orange curlers"
- **Work Schedule:** agreement between the boss and the employees about their schedules
- **Planned Vacation:** important for appointments and work schedule
- **Boss, Assistant, and Receptionist:**

Identify and write a number of use cases

Associations

Aims

1. Understand the concept of “associations”
2. Understand the structure of the problem domain in terms of associations
3. Understand class structure in terms of generalization
4. Understand object structure in terms of aggregation and general association
5. Identifying associations between classes
6. Notation for associations

Associations

Aims

1. Understand the concept of “associations”
2. Understand the structure of the problem domain in terms of associations
3. Understand class structure in terms of generalization
4. Understand object structure in terms of aggregation and general association
5. Identifying associations between classes
6. Notation for associations

Output: Creation of a conceptual model in the form of a set of class diagrams with **classes and object structures**

Overview

- Component/systems with totally independent classes and objects do not have meaningful behaviour
- Objects need to be related so that they can interact and work with each other to provide useful services
- The structure of a component is determined by the relationships between objects and classes
- There are two kinds of structure: **structures between classes** and **structures between objects**.

Class Structure – Generalization

- **generalization** structures gather the common properties and behavioural patterns of different classes into more general classes
- **specializations** partition a classes into subclasses which share some common properties or behavioural patterns
- a **generalization structure** is a relation between two or more specialized classes and one general class
- **Generalization:** a general class (**called super class**) describes properties common to a group of specialized classes (**subclasses**)

Generalisation/specialisation: *reuse, abstraction, refinement, polymorphism*

Characterisation of Generalisation

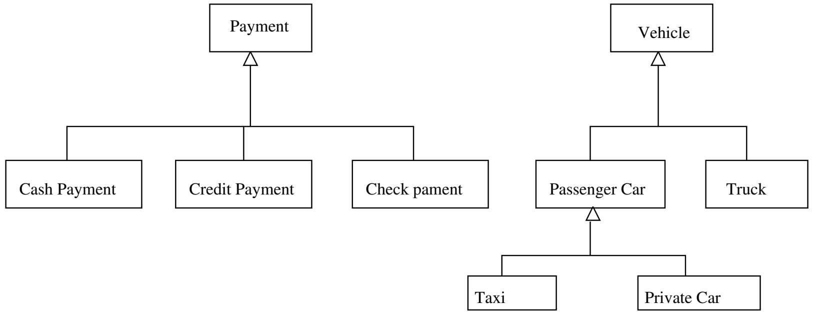
A class defines a set of objects and

all members of a subclass are members of its super class

$$Subclass \subseteq Superclass$$

This is also termed as the **Is-a-Rule** for testing a correct subtype

Examples and UML Notation



Super-sub classes in a Bank System?

Remarks

- Any object in a subclass is an object of the super class
- NOT required that every object in the super class is an object of a subclass class
- Subclasses are not necessarily mutually exclusive
- Everything that holds for the super class also holds for the subclasses
- Subclasses inherit the properties and behavioural pattern of the general class – *inheritance*.

Remarks

- Any object in a subclass is an object of the super class
- NOT required that every object in the super class is an object of a subclass class
- Subclasses are not necessarily mutually exclusive
- Everything that holds for the super class also holds for the subclasses
- Subclasses inherit the properties and behavioural pattern of the general class – *inheritance*.

All these properties are useful for identification of Generalisation-Specialisation relations in the domain.

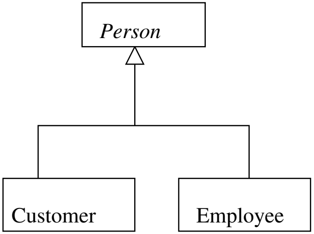
Abstract Classes

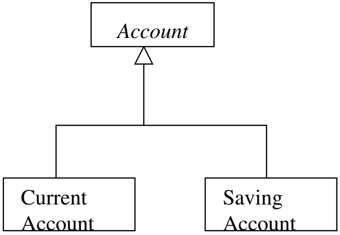
- When every object of the super class is an object of one of the mutually exclusive subclasses, the super class is also said abstract
- Engineers and programmers usually define a class to be abstract if it contains no concrete objects
- This in fact means that all the objects of the general level are only the those at the specialized level

Abstract Classes

- When every object of the super class is an object of one of the mutually exclusive subclasses, the super class is also said abstract
- Engineers and programmers usually define a class to be abstract if it contains no concrete objects
- This in fact means that all the objects of the general level are only the those at the specialized level

Examples





Identify Generalisation

Three approaches:

1. Take every pair of selected classes and determine whether one is a generalisation of the other.
2. Determine if a relevant generalisation class exists for pairs of selected classes,

Identify Generalisation

Three approaches:

1. Take every pair of selected classes and determine whether one is a generalisation of the other.
2. Determine if a relevant generalisation class exists for pairs of selected classes,

   E.G. “Business Partner” could be a superclass of Customer and Supplier.

Identify Generalisation

Three approaches:

1. Take every pair of selected classes and determine whether one is a generalisation of the other.
2. Determine if a relevant generalisation class exists for pairs of selected classes,

   E.G. “Business Partner” could be a superclass of Customer and Supplier.
3. Take a selected classes and attempt to define a relevant generalisation or specialisation.

Identify Generalisation

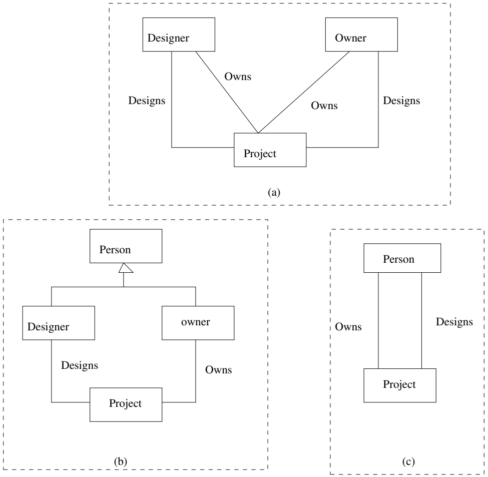
Three approaches:

1. Take every pair of selected classes and determine whether one is a generalisation of the other.
2. Determine if a relevant generalisation class exists for pairs of selected classes,

   E.G. “Business Partner” could be a superclass of Customer and Supplier.
3. Take a selected classes and attempt to define a relevant generalisation or specialisation.

   E.G. model may contain different types of Employees

Examples

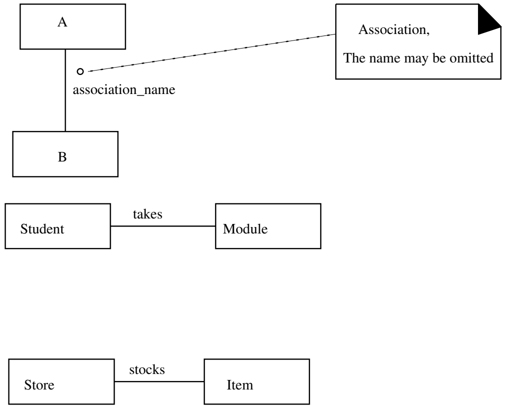


Which model is better?

Define Associations

- **Objects** must be related to each other to interact and collaborate with each other
- Two objects can interact only when they are **“related”** or **“linked”** at the time of the interaction
- An **association** between two classes is a **type** of **links** between some objects of the classes
- An **association** between two classes is a temporal relation between the objects of the two classes

Example and UML Notation

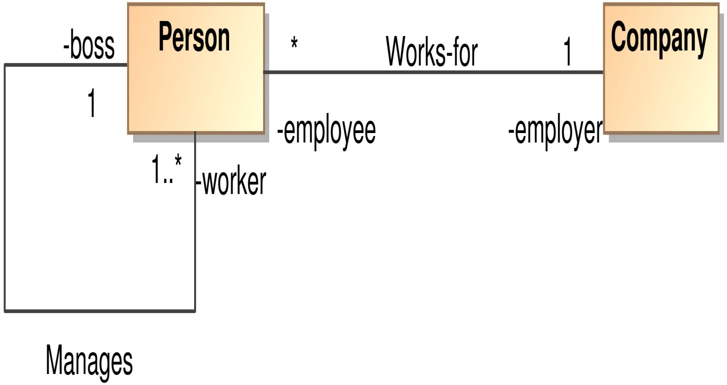


Need-to-Know Policy: a relationship that needs to be maintained, manipulated, needed for interaction

Roles of Associations

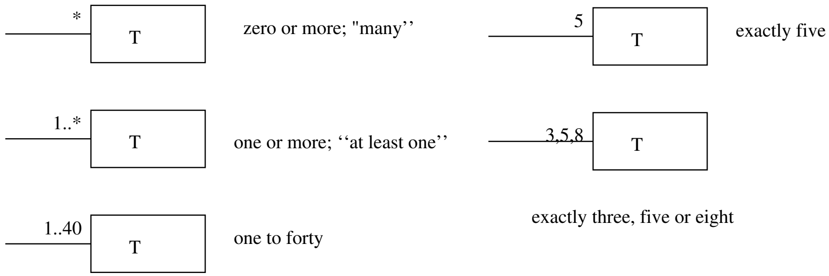
- Each end of an association is called a **role** of the association.
- A role optionally have
  - a name
  - a multiplicity
- role names are not very much significant at RA,
- but useful for an association which relates objects in the same class

Example and UML Notation

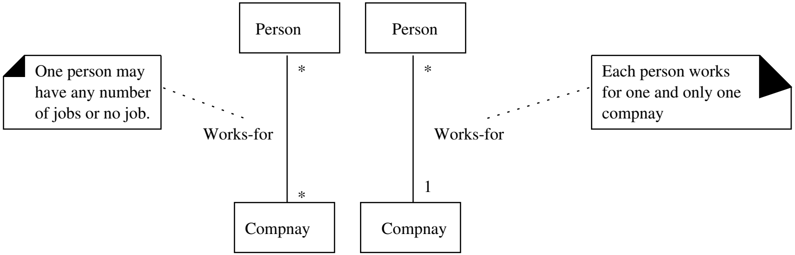


Revise this model using generalization-specialization?

Multiplicities in UML

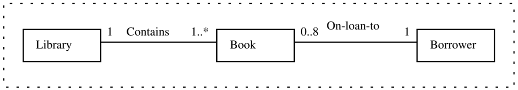


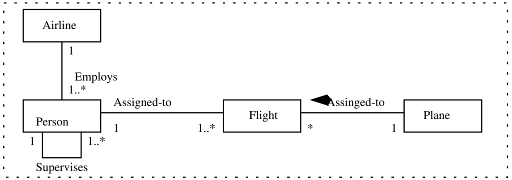
Example

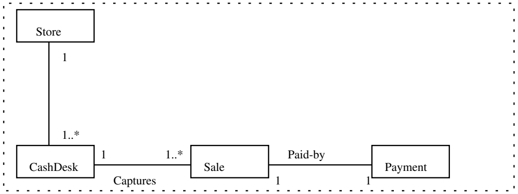

Person
Person
One person may have any number of jobs or no job.
*
*
Works-for
Works-for
Each person works for one and only one compnay
*
1
Compnay
Compnay

Determining multiplicity often exposes hidden assumptions

More Examples







Exercises

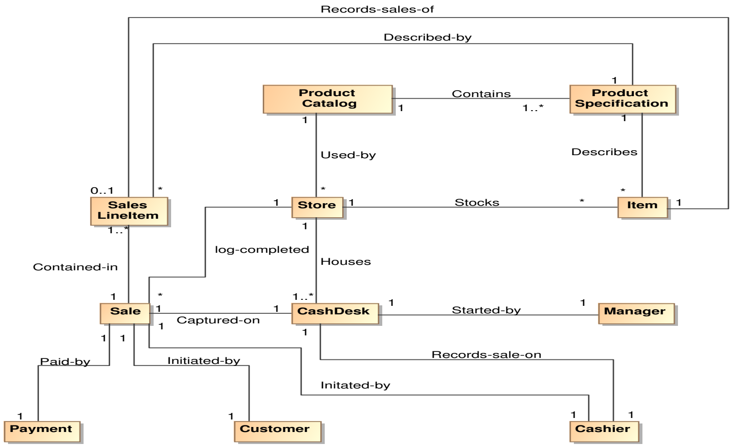
1. In the domain of the airline, model two relationships between a **Flight** and **Airport**, **Flies-to** and **Flies-from**:
2. Model the facts that a car is owned by one owner, and a car can be used by any number of people.

Capture Associations

1. Capture a relation that implies
   - a knowledge of a relationship that needs to be preserved for some duration ("need-to-know" association)
   - a link between two objects fulfilling the role to provide a mean for meaningful interaction
   - a **physical** or **conceptual** connections between objects of the classes
2. Verb phrases
3. Useful categories of associations:
   - A **is a physical part of** B
   - A **is a logical part of** B
   - A **contains** B
   - A **uses** B, A **manages** B

   Experienced requirements analyst/consultant keep their own categories for special supplication domains.

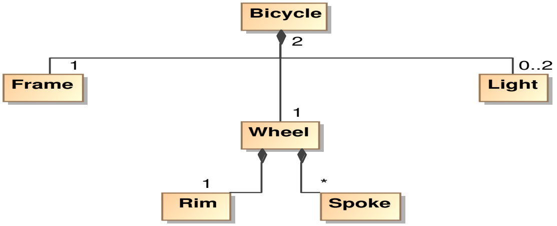
Conceptual Model for POST

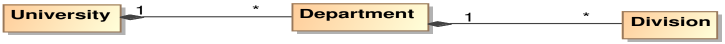


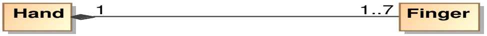
Aggregation

- An **aggregation** an association used to model a whole-part relationship between objects
- An aggregation is a **composition** if the multiplicity at the **composite** end is at most one
- An aggregation is a **shared aggregation** if the multiplicity at the composite end is more than one

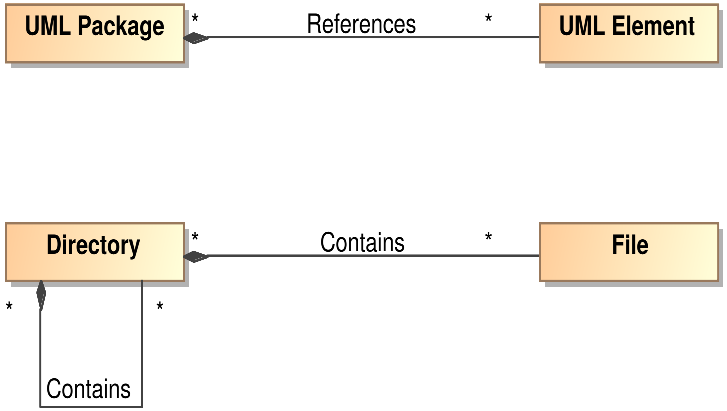
Examples of composition







Examples of Shared Aggregation



Common Properties of Aggregations

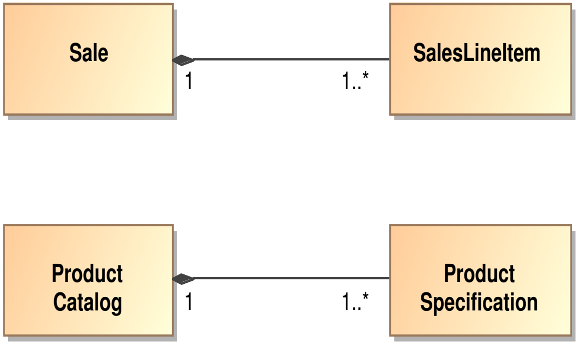
1. The lifetime of the part is bound within the lifetime of the composite, i.e. there is a create–delete dependency of the part on the whole.
2. Properties of the composite propagate to the parts, such as its location
3. Operations applied to the composite propagate to the parts, such as destruction, movement, recording.

Typical Cases of Aggregations

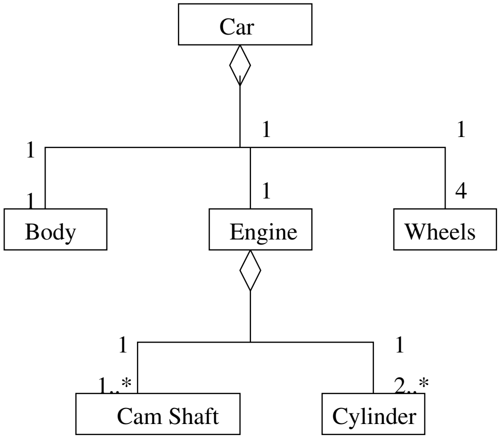
- **Whole-part:** the whole is the sum of the parts; adding or removing any part changes the whole fundamentally
- **Container-content:** the whole is a container for the parts; adding or removing any content will not change the fundamental properties of the whole.
- **Union-member:** the whole is an organisation union of members, the union is not changed fundamentally by adding or remove a few members, but there is a lower limit on the number of members

If not sure when to use it, ignore it and stick to plain association

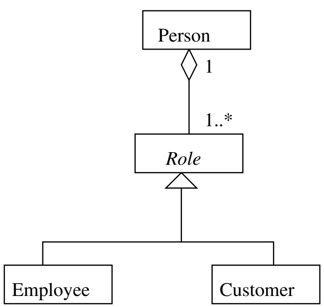
Aggregation in POST

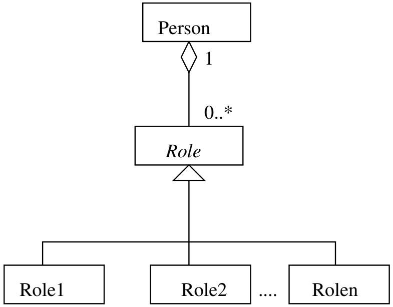


Example

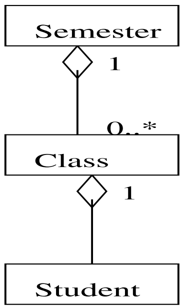


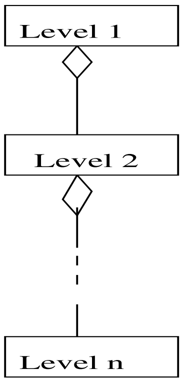
Modelling Patterns - Role Pattern





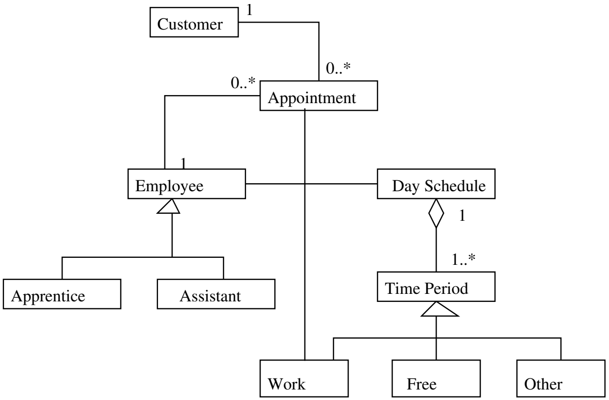
Hierarchy Pattern



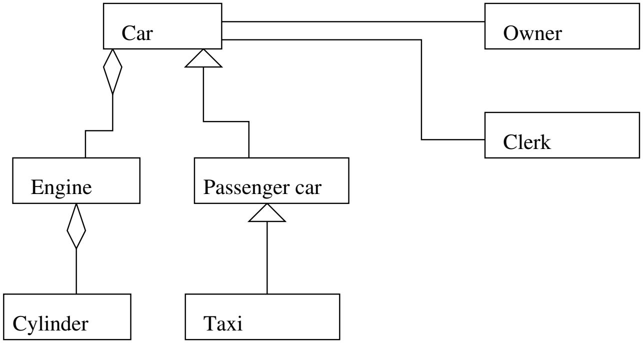


An object at one level can belong to several objects on the level above

More Conceptual Models



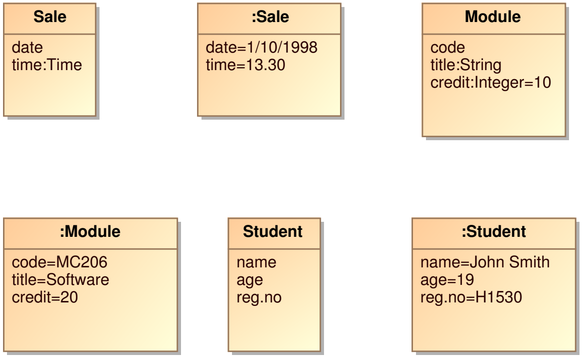
Another Example



Attributes of Classes

- An instance of a class has meaningful **properties**
- An **attribute** of a class is the abstraction of a single property of objects of the class
- At any time, an instance has a **logical value** of an attribute of its class, called the **attribute of the object** at that time
- One object has exactly one value for each attribute at **any given time, which can be recorded, modified, and passed among objects**

Examples and UML Notation



Attributes of Classes

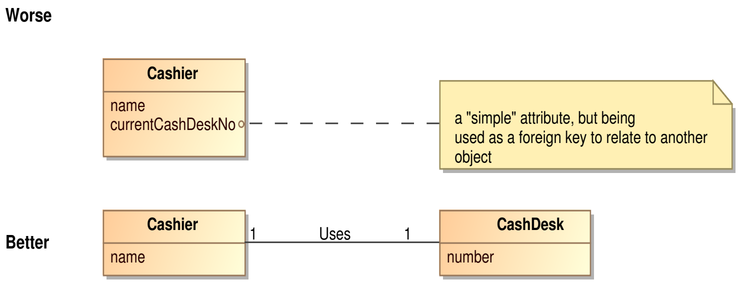
Desirable to be

- **Complete:** capture all relevant attributes about a class,
- **Fully factored:** each attribute captures a different property of an object,
- **Mutually independent:** the values of the attributes of an object are independent of each other, i.e. try to avoid derived attributes,
- **Relevant to the requirements and use cases:** focus on those attributes for which the requirements suggest or imply a need to remember information.

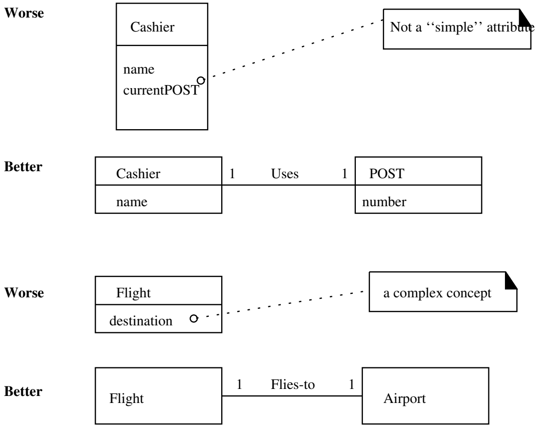
Attributes, Objects, Associations

Common mistake: represent something as an attribute when it should have been a concept or an association

- attributes are of **simple data**
- the defining features of objects do NOT apply to attributes
- attributes are NOT for inter-object communication or navigation

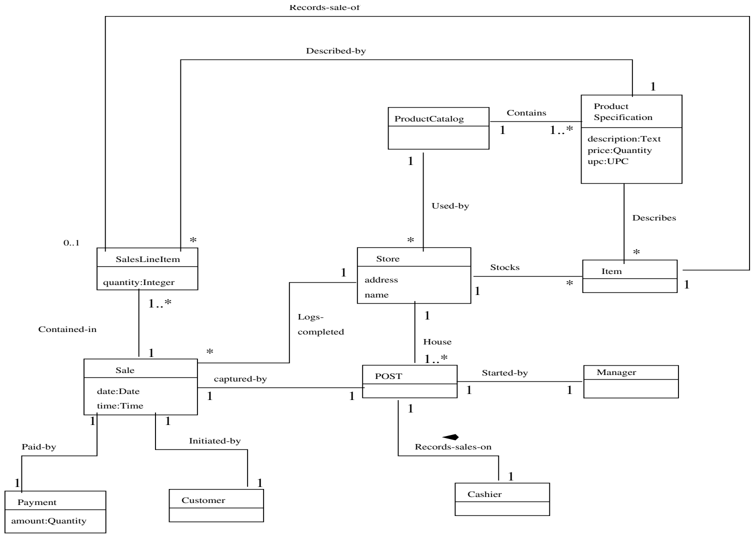

Worse
Cashier
name
currentCashDeskNo
a "simple" attribute, but being used as a foreign key to relate to another object
Better
Cashier
name
1
Uses
1
CashDesk
number

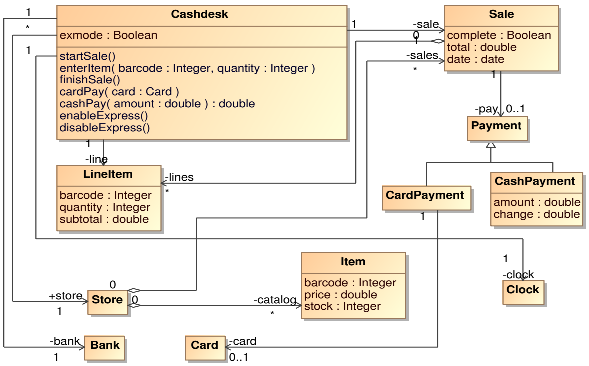
Examples



Relate concepts with an association, not with an attribute

Adding Attributes to Class Diagram

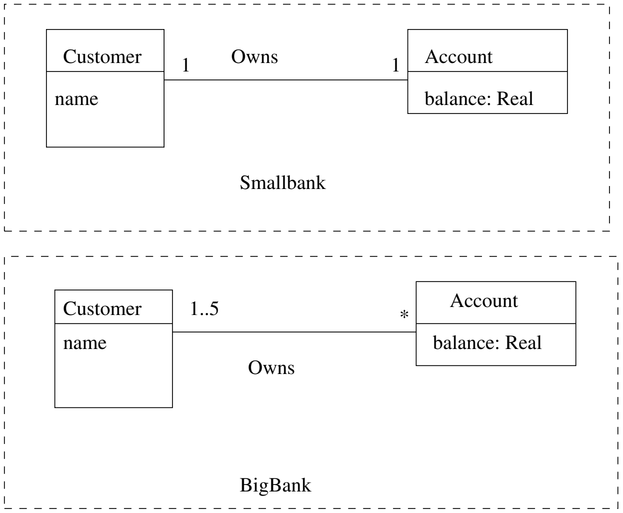



Cashdesk
exmode : Boolean
startSale()
enterItem(barcode : Integer, quantity : Integer)
finishSale()
cardPay(card : Card)
cashPay(amount : double) : double
enableExpress()
disableExpress()
Sale
complete : Boolean
total : double
date : date
Payment
CardPayment
CashPayment
amount : double
change : double
LineItem
barcode : Integer
quantity : Integer
subtotal : double
Item
barcode : Integer
price : double
stock : Integer
Store
Bank
Card
Clock
-sale
-sales
-pay
0..1
-line
-lines
+store
-catalog
-bank
-card
-clock
1
*
0

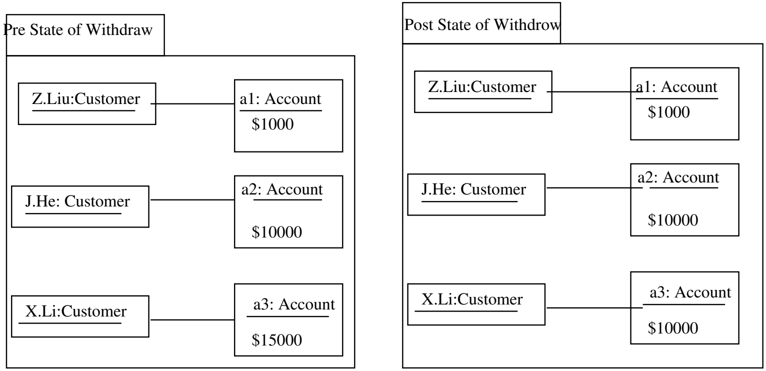
States of a Use Case (Component)

- a **state** of a component at a moment of time defines
  - the existing objects of each of the classes,
  - the values of the attributes of these objects,
  - the existing links between objects
- the class diagram defines the state space of the component
- a state is represented in UML by an **object diagram**

Example: Banks

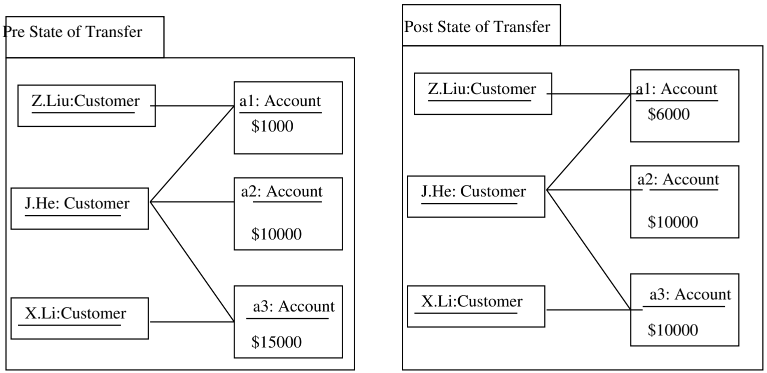


Object Diagrams



OBD of SmallBank or BigBank?

More Object Diagrams

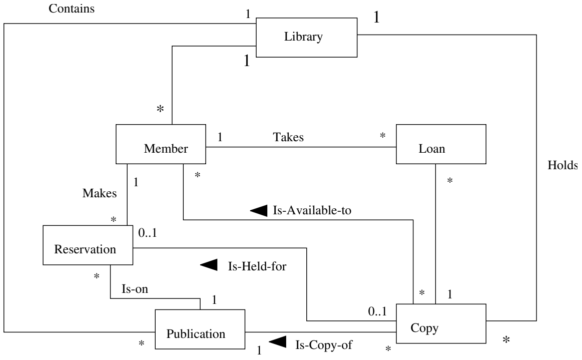


OBD of SmallBank or BigBank?

State Constraints

- an object diagram is a state of the component at a moment of time
- classes, associations, multiplicities and attributes in a conceptual model impose constraints on the set of allowable states
- can a conceptual model express all the possible constraints?

Consider the Library System



a copy *c* that Is-Held-for a reservation *r* must be a copy of the publication that is reserved by the reservation *r*?

Add State Constraints

- diagrams not expressive enough for all possible state constraints
- add informal textual or **formal specification**
  - $\forall c : \mathit{Copy}, r : \mathit{Reservation}, p : \mathit{Publication} \bullet$ c Is-Held-For $r \wedge r$ Is-on $p \Rightarrow c$ Is-Copy-of p
  - Is-Held-for $\circ$ Is-on $\subseteq$ Is-Copy-of
- state constraints are important for the system design and need to be checked for every step in every use case
- Object Constraint Language (OCL), as part of UML, is used to describe state constraints
- This module does not cover OCL