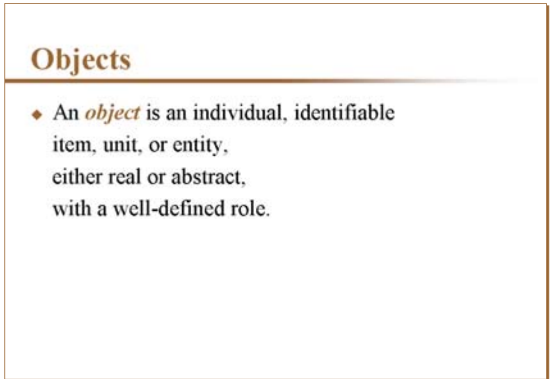
Practical Guidelines

for Identifying Classes and Relationships

Prof. T.H. Tse
Department of Computer Science
Email: thtse@cs.hku.hk
Web: hku.hk/thtse.

Identifying Objects and Classes

Recall:

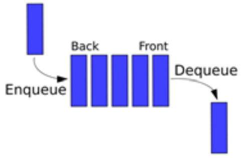


Objects

- An *object* is an individual, identifiable item, unit, or entity, either real or abstract, with a well-defined role.

Identifying Objects and Classes

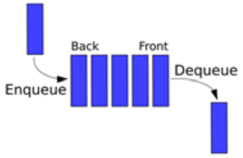
- Objects must either be
  - physical entities (such as persons), or
  - conceptual entities on their own (such as accounts)
- Must be meaningful in the application domain (not just the target system)
  - *Example:* queue ***??*** ...



or

Identifying Objects and Classes

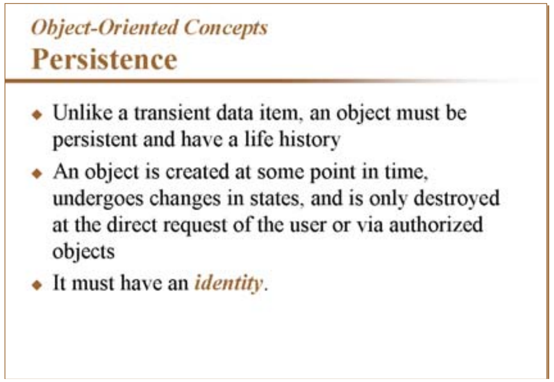
- Objects must either be
  - physical entities (such as persons), or
  - conceptual entities on their own (such as accounts)
- Must be meaningful in the application domain (not just the target system)
  - *Example:* queue ***??*** ...



or

Identifying Objects and Classes

Recall:



Object-Oriented Concepts

Persistence

- Unlike a transient data item, an object must be persistent and have a life history
- An object is created at some point in time, undergoes changes in states, and is only destroyed at the direct request of the user or via authorized objects
- It must have an *identity*.

Identifying Objects and Classes

- An object has ***observable attributes***, which can be changed using its ***encapsulated methods***
- ***But*** an object should not simply be a convenient collection of attributes and methods .

Identifying Objects and Classes

- Classes are collections of related objects
- They are usually described by
  - nouns (such as Account), or
  - noun phrases (such as Cheque Account)
- During ***Systems Analysis***, a class should not be considered for ***normalization*** or ***implementation*** .

Example: ATM Classes

Access | Account | Account data | ATM | Bank

Bank computer | Banking network | Cash | Cash card | Cashier

Cashier station | Central computer | Communications line

Consortium | Cost | Customer | Receipt

Recordkeeping provision | Security provision | Software | System

Transaction | Transaction data | Transaction log | User .

Example: ATM Classes

Find Problematic Classes

Access · Account · Account data · ATM · Bank

Bank computer · Banking network · Cash · Cash card · Cashier

Cashier station · Central computer · Communications line

Consortium · Cost · Customer · Receipt

Recordkeeping provision · Security provision · Software · System

Transaction · Transaction data · Transaction log · User .

Example: ATM Classes

Keep the Right Classes

Account · ATM · Bank

Bank computer · Cash card · Cashier

Cashier station · Central computer

Consortium · Customer

Transaction .

Keeping the Right Classes

- ***Redundant Classes***
  - If 2 classes express very similar information, select only the more descriptive one

    User ***versus*** ***Customer***
- ***Irrelevant Classes***
  - Eliminate classes having little to do with the problem ***Cost .***

Keeping the Right Classes

- ***Vague Classes***
  - Reconsider ill-defined boundaries

    System

    Security provision

    Recordkeeping provision

    Banking network .

Keeping the Right Classes

- ***Attributes***
  - Names that describe properties of objects should be restated as attributes

    Account data ***Cash*** ***Receipt*** ***Transaction data***
- ***Operations***
  - Reconsider classes whose names describe operations ***Telephone call*** .

Keeping the Right Classes

- ***Roles***
  - A class name should reflect its intrinsic nature and not the role of an association

    Owner

    Rename as ***Customer*** .

Keeping the Right Classes

- ***Implementation Constructs***
  - During Systems Analysis, eliminate constructs related to ***implementation***, rather than ***user requirements***

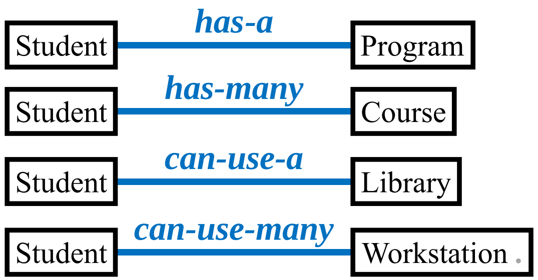
    Access ***Communication line***

    Software ***Transaction log***

    More about this later .

Identifying Associations

- Correspond to verbs or verb phrases connecting 2 or more classes
- Often related with ownership:



Student *has-a* Program

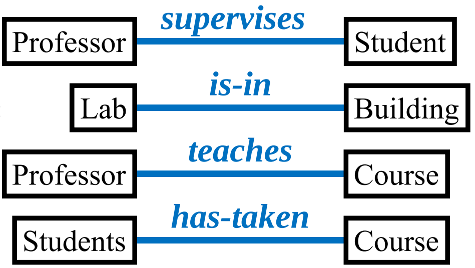
Student *has-many* Course

Student *can-use-a* Library

Student *can-use-many* Workstation .

Identifying Associations

- May also be related with
  - directed action:
  - physical location:
  - communication:
  - some condition:



Professor *supervises* Student

Lab *is-in* Building

Professor *teaches* Course

Students *has-taken* Course

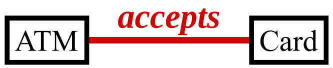
- Depends on ***user requirements*** .

Keeping the Right Associations

- ***Associations between Eliminated Classes***
  - If a class in the association has been eliminated, then eliminate the association or reinstate the class
- ***Irrelevant or Implementation Associations***
  - Eliminate associations dealing with ***implementation*** constructs unrelated to ***user requirements*** .

Keeping the Right Associations

- ***Actions***
  - An association describes a ***persistent*** property, not a ***transient*** event:



ATM *accepts* Card

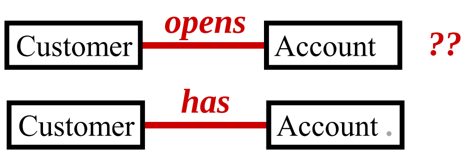
?? .

Keeping the Right Associations

- ***Derived Associations***
  - Cancel associations defined indirectly in terms of other associations
  - *Examples:*
    - Multiple paths
      - "Grandparent of "
    - Conditions on attributes
      - Define "younger than" using birth dates .

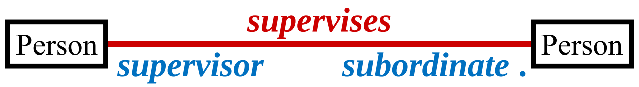
Keeping the Right Associations

- ***Misnamed Associations***
  - Avoid name that reflect historical event:



Keeping the Right Associations

- ***Role Names***
  - Add role name to clarify ambiguous situation:

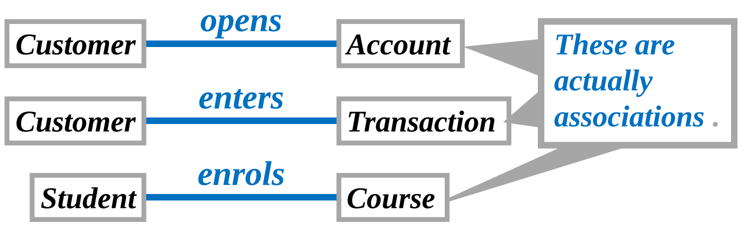


Keeping the Right Associations

- ***Multiplicity***
  - Specify multiplicity
  - Challenge 1:1 multiplicities
  - ***But*** do not put too much effort into getting multiplicities, since they often change during analysis
  - Ask whether the objects need to be ordered .

Keeping the Right Associations

- ***Missing Associations***
  - Add any missing association discovered:



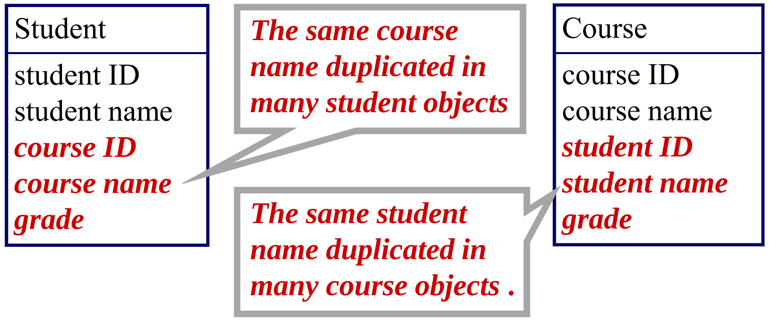
Identifying Attributes

- Attributes are observable properties of objects
- Usually corresponds to noun followed by preposition:
  - *colour of* Car — *white*
  - *state of* Button — *pressed* .
- Adjective may indicate attribute value

Keeping the Right Attributes

- ***Divergent Attributes***
  - A class with 2 sets of attributes unrelated to each other may indicate the need for splitting …

Divergent Attributes
Example



Student: student ID, student name, *course ID*, *course name*, *grade*

The same course name duplicated in many student objects

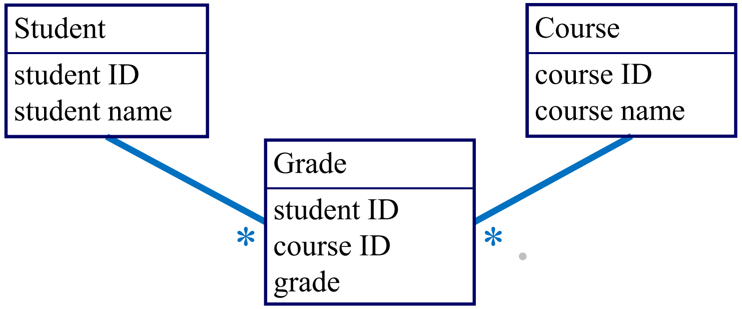
Course: course ID, course name, *student ID*, *student name*, *grade*

The same student name duplicated in many course objects .

Divergent Attributes
Example (Continued)

But not the result of database normalization

- Learn from database normalization



Keeping the Right Attributes

- ***Classes***
  - Entities that have features of their own within the given application constitute a class

    City in a mailing list is an attribute

    City in a census is a class .

Keeping the Right Attributes

- ***Classes* (*continued*)**
  - If the independent existence of an entity is important (rather than just the value), we should have a class

    Supervisor is a class

    Salary is an attribute .

Keeping the Right Attributes

- ***Identifiers***
  - Distinguish between
    - identifiers in the application domain
    - object identifiers for implementation
  - Should not specify pure object identifiers in the analysis model

    Account Code is an identifier used by the bank

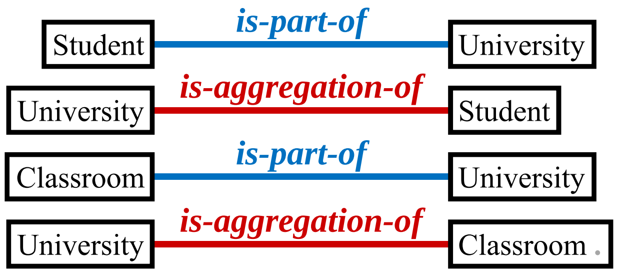
    Transaction ID may be an identifier in the implemented system **??** .

Keeping the Right Attributes

- ***Internal Values***
  - Eliminate any attribute which describes the internal state of an object and which is invisible outside the object .

Aggregations

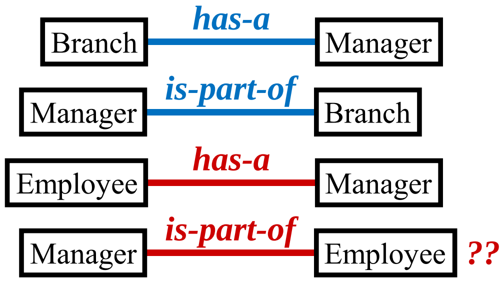
- A special type of association
- Class X is an ***aggregation*** of class Y if every object in Y ***is-part-of*** some object in X:



Aggregations

We Learn from Mistakes

- Be careful with
  - ***has-a***
  - ***has-many***



- However, do not spend too much time trying to distinguish between associations and aggregations .

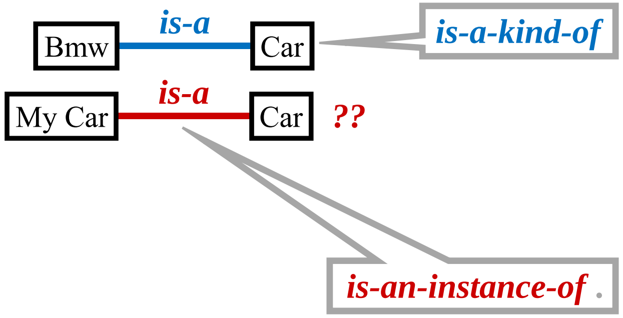
Identifying Inheritance

- Common descriptors to help to identify inheritance:
  - ***is-a-kind-of***
  - ***is-a*** .

Identifying Inheritance

We Learn from Mistakes

- Be careful with

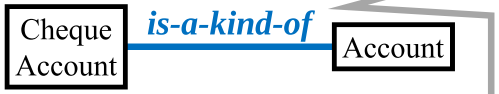


Identifying Inheritance

Two directions:

- *Top down*
  - Refine classes into specialized subclasses
  - More common in analysis

Look for extra behavioural constraints in the subclass that differentiates it from the superclass


Cheque Account

is-a-kind-of

Account

Inherits all methods in Account plus special operations like "honour cheque" .

Identifying Inheritance

Two directions:

- ***Bottom up***
  - Generalize classes into a superclass

  Example: Generalize stack and queue into a superclass "linked list"

  - May not reflect the real world, hence only recommended in design .

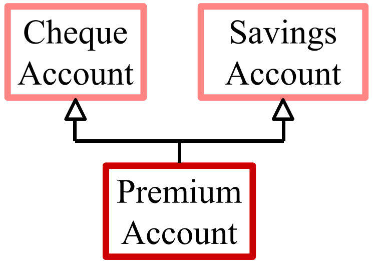
Look for classes with common attributes, associations, or methods

But is a queue a-kind-of linked list ??

Multiple Inheritance

- A class inherits from two superclasses
- May increase complexity

During analysis, let the user decide .


Cheque Account

Savings Account

Premium Account

Test the Access Paths

- Trace the access paths in a class diagram to see whether they give sensible results
- *Example:*
  - Unique result for 1-associations? .

Iterative Modelling

- The entire object-oriented development is a continual iterative process
- Different parts of a model may be at different stages of completion
- Refine the class diagram after dynamic modelling .

More Reading Materials at Student Request

Further Guidelines

(1) *Common Operations*

- The existence of operations common to 2 or more objects indicate a high probability of identifying an association, aggregation and/or inheritance .

Further Guidelines

(2) *Polymorphic methods*

- Polymorphic methods should not be considered as common methods when reviewing objects and relationships
  - *Examples:* "open" and "close"
- On the other hand, we should not only look at the name when deciding whether a method is polymorphic ...

Further Guidelines

(2) *Polymorphic methods*

- Check whether the method have common behaviour among related objects
- *Example:* To open a cheque account, we
  - create account object
  - copy information from customer object
  - set the transaction history to nil

  We do exactly the same things when opening a savings account or reserve account .

Further Guidelines

(3) *Normalization*

- The usual recommendations on the normalization of databases can be extended from associations to aggregations and inheritance, and from attributes to methods …

Further Guidelines

(3) *Normalization*

- *Example:*
  - Given 2 objects *X* and *Y*, an operation *Z* is ***transitively dependent*** on one of them if
    - *Z* is an operation of both *X* and *Y*
    - There exists a relationship between *X* and *Y*
  - The presence of transitive dependence indicates a high probability that *Z* is a redundant operation of either *X* or *Y* .

Further Guidelines

(4) *Meaningfulness*

- Look at the meaningfulness of the classes and their relationships
- Especially if we attempt to create new classes because of normalization
- Classes should not be factorized purely for the convenience of implementation, or to reduce the fan-in ratio
- Neither should new relationships be created for such purposes .

Further Guidelines

(5) *Reverse Associations*

- For every
  - has-a
  - has-many
  - uses-a
  - uses-many

  relationship between 2 objects, consider also the reverse association, resulting in complete multiplicities of the form
  - 1:1, 1:*M*, *M*:1, or *M*:*M* …

Further Guidelines

(5) *Reverse Associations*

- Reverse associations may be only for human consumption, to show users the full picture
- Not necessarily implemented in the final system because of efficiency considerations .

Further Guidelines

(6) *Resolving M:M Associations*

- Most methodologies recommend specifying *M*:*M* associations as two 1:*M* associations
  - *Example:* Since there is an *M*:*M* association between “student” and “teacher”, we create an artificial “student-teacher” object
- Classical example of allowing design issues to influence analysis ...

Further Guidelines

(6) *Resolving M:M Associations*

- Recommend retaining the *M*:*M* association unless the need for a middle man is a genuine user requirement (indicated by the presence of genuine operations at the intersection) .

Further Guidelines

(7) *Aggregations vs Inheritance*

- If an object *X* is made of one or pieces of object *Y*, together with other objects, then we have a candidate for an aggregation
- If an object *X* is made of exactly one piece of object *Y*, then we have a candidate for an inheritance
  - *Example:* “A keyboard is-part-of a computer”
  - “A notebook is-a-kind-of computer” .