Class Based Modeling

Approaches for Identifying Classes

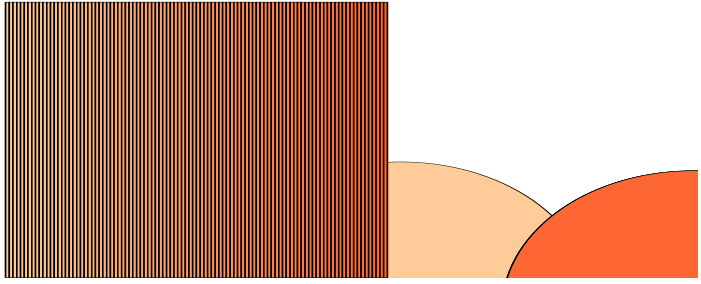
- The noun phrase approach.
- The common class patterns approach.
- The class responsibilities collaboration (CRC) approach.
- The use-case driven approach.

1. Noun-phrase approach

- Initial list of noun phrases
- Eliminate irrelevant classes
- Reviewing redundant classes
- Review adjective classes
- Review possible attributes
- Review the class purpose

Noun Phrase Strategy (Contd…)

- Change all plurals to singular and make a list, which can then be divided into three categories.



1. **Relevant classes**
2. **Fuzzy classes(class that are not sure about)**
3. **Irrelevant classes**

1.1 Initial list of Noun phrases

- Guidelines for identifying *tentative* classes
  - Look for nouns in the System Requirements Specification (SRS)
  - Some classes are implicit or taken from general knowledge
  - All classes are to be taken from problem domain
  - Carefully choose and define class names

1.2 Eliminate Irrelevant classes

- Noun-phrase are put into 3 categories
  - Relevant classes
  - Fuzzy classes
  - Irrelevant classes - unnecessary, so omit

1.3 Reviewing Redundant classes

- Redundant classes from Relevant and Fuzzy classes
  - Do not keep two classes that express the same information
  - In case of more than one to describe the same idea, choose the word used by user of the system

1.4 Review Adjective classes

- Adjectives are used in many ways
  - Adjectives can suggest a different kind of object
  - Different use of the same object

- Find whether the object represented by noun behave differently when the adjective is applied to it.

- Make a new class only if the behavior is different

1.5 Review possible Attributes

- Attribute classes
  - Classes defined only as values should be restated as attributes and not classes

1.6 Review the class purpose

- Each class must have a purpose and every class should be clearly defined.
- Formulate a statement of purpose for each candidate class.
- Otherwise, eliminate the candidate class

Guidelines For Identifying Classes

- The followings are guidelines for selecting classes in your application:
  - Look for nouns and noun phrases in the problem statement.
  - Some classes are implicit or taken from general knowledge.
  - All classes must make sense in the application domain.
  - Avoid computer implementation classes, defer it to the design stage.
  - Carefully choose and define class names.

Guidelines For Identifying Classes (Contd…)

- External Entities (e.g. other systems, devices, people)
- Things (e.g. reports, displays, letters, signals, results)
- Occurrences or Events (e.g. property transfer, fund transfer, robot movements)
- Roles (e.g. manager, engineer, salesperson)
- Organizational Units (e.g. division, group, team)
- Places (e.g. manufacturing floor, loading dock, back door)
- Structures (e.g. sensors, vehicles, computers)
- Not Objects/Attributes (e.g. number, type)

1. Identify potential classes and define above attributes based on above mentioned points.
2. Then apply selection characteristics to retain with potential classes.

Selection characteristics of Potential Classes

- Retained Information: information about class must be remembered so that the system should function.
- Needed Services: must have set of identifiable operations that can change the value of attributes in some way.
- Multiple Attributes: multiple attributes should be represented in the class
- Common Attributes: set of common attributes can be defined or applied to all instances of that class.
- Common Operations: set of common operations can be defined or applied to all instances of that class.
- Essential Requirements: information producing or consuming by external entities should be considered as classes

Guidelines For Refining Classes

Redundant Classes:

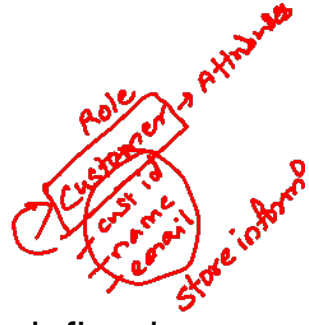
- Do not keep two classes that express the same information.
- If more than one word is being used to describe the same idea, select the one that is the most meaningful in the context of the system.

Guidelines For Refining Classes (Contd..)

Adjective Classes:

- Does the object represented by the noun behave differently when the adjective is applied to it?
- If the use of the adjective signals that the behavior of the object is different, then make a new class.
- For example, If *Adult Membership* and *Youth Membership* behave differently, than they should be classified as different classes.

Guidelines For Refining Classes (Con't)



Attribute Classes:

- Tentative objects which are used only as values should be defined or restated as attributes and not as a class.
- For example the demographics of Membership are not classes but attributes of the Membership class.

Guidelines For Refining Classes (Con't)

Irrelevant Classes:

- Each class must have a purpose and every class should be clearly defined and necessary.
- If you cannot come up with a statement of purpose, simply eliminate the candidate class.

Identifying a list of candidate classes: Example 1

Books and journals The library contains books and journals. It may have several copies of a given book. Some of the books are for short term loans only. All other books may be borrowed by any library member for three weeks. Members of the library can normally borrow up to six items at a time, but members of staff may borrow up to 12 items at one time. Only members of staff may borrow journals.

Borrowing The system must keep track of when books and journals are borrowed and returned, enforcing the rules described above.

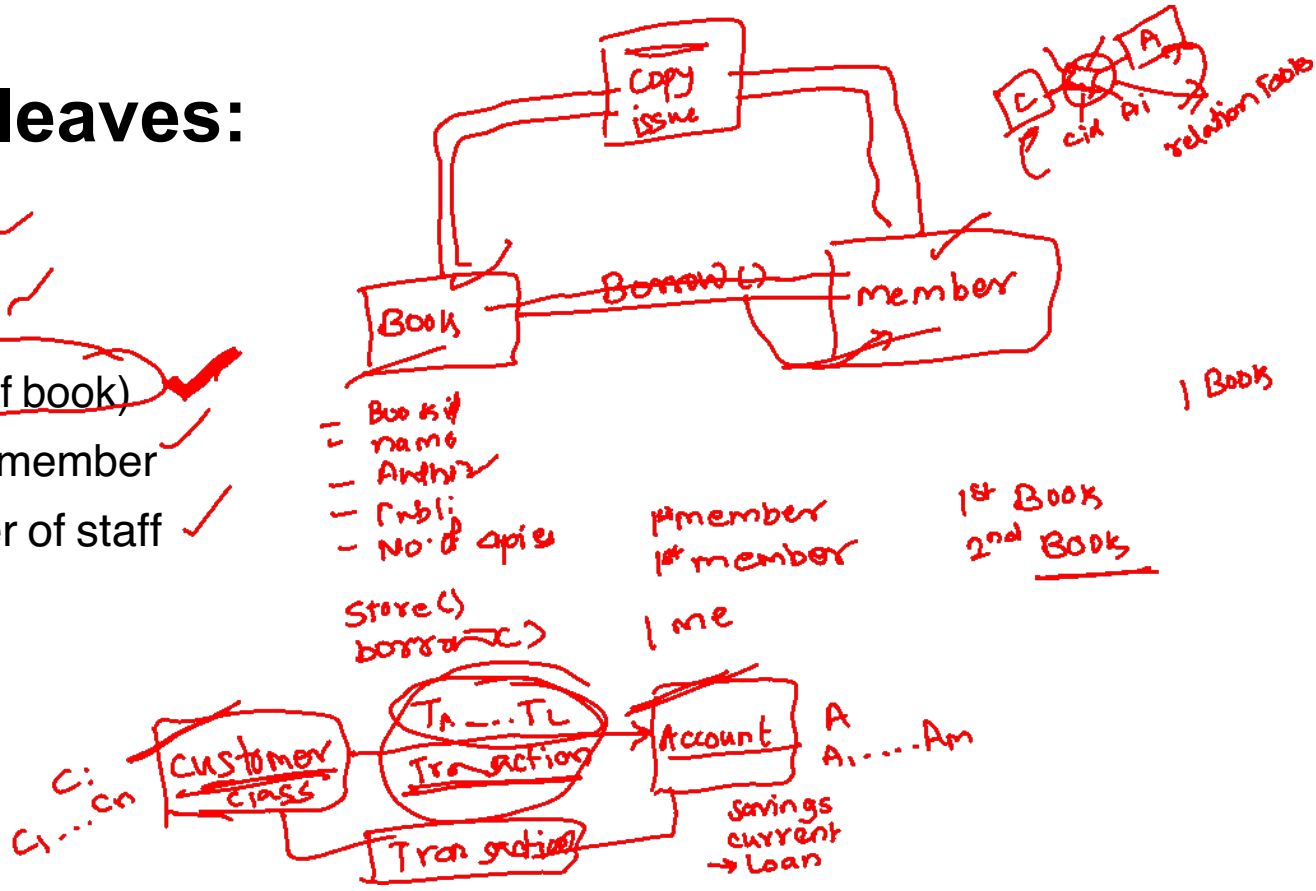
- Take a coherent, concise statement of the requirement of the system
- Underline its noun and noun phrases, that is, identify the words and phases the denote things
- This gives a list of candidate classes, which we can then whittle down and modify to get an initial class list for the system

In this particular case we discard

- **Library**, because it is outside the scope of our system
- **Short term loan**, because a loan is really an event, which so far as we know is not a useful object in this system
- **Member of the library**, which is redundant
- **Week**, because it is a measure, not a thing
- **Item**, because it is vague (we need to clarify it)
- **Time**, because it is outside the scope of the system
- **System**, because it is part of the meta-language of requirements description, not a part of domain
- **Rule**, for the same reason

This leaves:

- Book
- Journal
- Copy (of book)
- Library member
- Member of staff



Example 2:

System domain

In the hospital, there are a number of wards, each of which may be empty or have on it one or more patients. There are two types of ward, male wards and female wards. A ward can only have patients of the specified sex on it. Every ward has a fixed capacity, which is the maximum number of patients that can be on it at one time (i.e. the capacity is the number of beds in the ward). Different wards may have different capacities. Each ward has a unique name. The hospital has an Administration department that is responsible for recording information about the hospital's wards and the patients that are on each ward.

The doctors in the hospital are organised into teams, each of which has a unique team code (such as Orthopaedics A, or Paediatrics). Each team is headed by a consultant doctor who is the only consultant doctor in the team; the rest of the team are all junior doctors, at least one of whom must be at grade 1. Each doctor is in exactly one team. The Administration department keeps a record of these teams and the doctors allocated to each team.

Each patient is on a single ward and is under the care of a single team of doctors; the consultant who heads that team is responsible for the patient. A patient may be treated by any number of doctors but they must all be in the team that cares for the patient. A doctor can treat any number of patients.

hospital
number of wards
patient
type of ward
male ward
female ward
ward
sex
capacity (maximum number of patients, number of beds)
name of ward
Administration department
information

doctor
team
team code
Orthopaedics A
Paediatrics
grade 1
consultant doctor (consultant)
junior doctor
record of teams and doctors
care
number of doctors
number of patients

Potential classes

Eliminated Classes

- hospital, Administration department (outside the scope of the system)
- number of wards (language idiom – the requirements document could equally well have referred to ‘wards’ or ‘some wards’)
- type, capacity, name (attributes of ward)
- sex (attribute of patient)
- information (generic term referring to the behaviour of the system)
- team code (attribute of team)
- record of teams and doctors (part of the behaviour required of the system)
- number of doctors, number of patients (language idiom)

You may have noticed that there are three nouns,

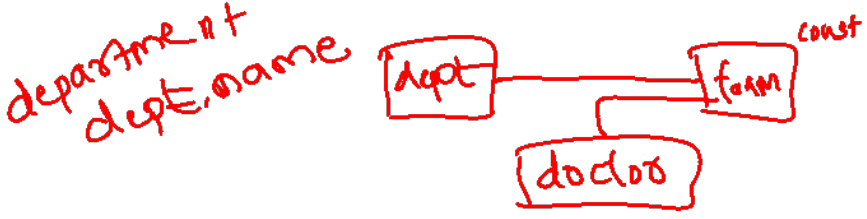
‘Orthopaedics A,

‘Paediatrics’ and

‘grade 1’,

which have not been eliminated using the guidelines for rejection, yet which common sense suggests should not be modelled by classes.

‘Orthopaedics A and ‘Paediatrics’ are in fact given in the requirements document as example values of ‘team code’, which was identified above as an attribute of team, and ‘grade 1’ is a value of an attribute, ‘grade’, of junior doctor.



male ward

female ward

ward

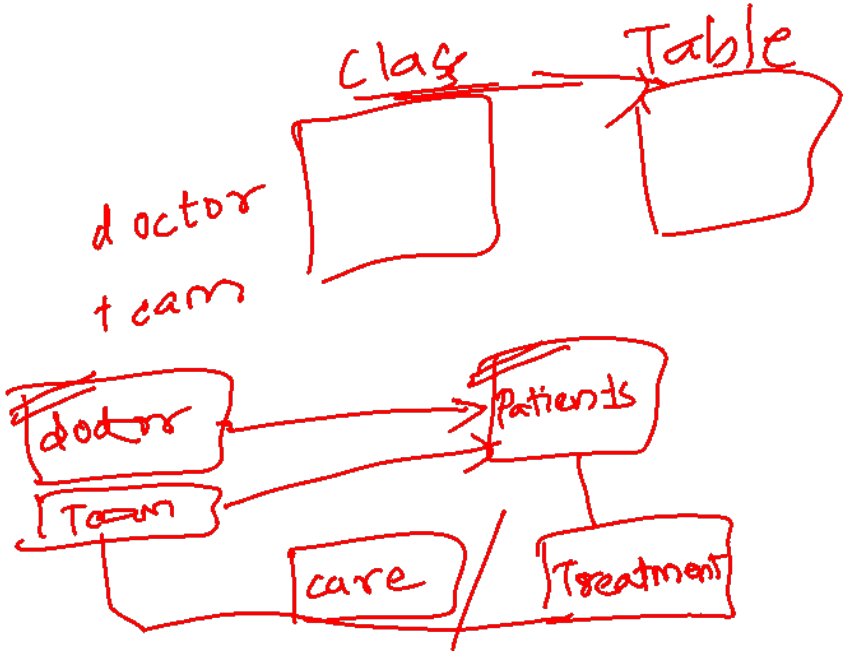
patient

doctor team

consultant doctor

junior doctor

care



Example 3:

A store wants to automate its inventory. It has point-of-sale terminals that can record all of the items and quantities that a customer purchases. Another terminal is also available for the customer service desk to handle returns. It has a similar terminal in the loading dock to handle arriving shipments from suppliers. The meat department and produce department have terminals to enter losses/discounts due to spoilage.

Identify nouns

Store
Inventory
Point-of-sale terminal
Terminals
Items
Quantity
Purchase
Customer
Customer service desk

Handle returns
Returns
Loading dock
Shipment
Handle shipment
Suppliers
Meat department
Produce department
Department
Enter losses
Enter discount
Spoilage

Eliminate irrelevant nouns

Store
Point-of-sale terminal
inventory
Item
Customer
Customer service desk
Handle returns
Returns
Handle shipment
Shipment
Meat department
Produce department
Department
Enter losses
Enter discount

Eliminate redundancies

Store
Point-of-sale terminal
Item
Customer service desk
Handle returns
Handle shipment
Meat department
Produce department
Enter losses
Enter discount

The final set of classes and objects after the elimination process .

Point-of-sale terminal
Item
Handling return
Handling shipment
Enter losses
Enter discount
Meat department
Produce department
Store

2. Common Class patterns approach

- Based on a knowledge base of the common classes proposed by various researchers
  - Concept class
  - Event class
  - Organization class
  - People class
  - Place class
  - Tangible things and device class
  - ***Review the class purpose***

2.1 Concept Class

- Idea or understanding of the problem
- Principles that are not tangible, and used to organize or keeping track of business activities

  E.g.: Performance, reservation

2.2 Event class

- Things happen at a given date and time
- Points in time that must be recorded

  E.g.: Request, order

2.3 Organization class

- A collection of people, resources, facilities, or group to which the users belong

  E.g.: Accounting department

2.4 People class (roles class)

- Different roles users play in interacting with application
- Guideline
  - Classes that represent users of the system

    E.g.: Operator, clerk
  - Classes representing people who do not use the system but about whom information is kept by the system

    E.g.: Employee, client, teacher, manager

2.5 Places class

- Areas set aside for people or things
- Physical locations that the system must keep information about

  E.g.: Building, store, site, travel office

2.6 Tangible things

- Physical objects or groups of objects that are tangible and devices with which application interacts

  E.g.: Car, ATM

ATM Case Study –System Requirements

- The bank client must be able to deposit an amount to and withdraw an amount from his or her accounts using the touch screen at the ATM kiosk. Each transaction must be recorded, and the client must be able to review all transactions performed against a given account. Recorded transactions must include the date, time, transaction type, amount and account balance after the transaction

- A bank client can have two types of accounts: a checking account and savings account. For each checking account, one related savings account can exist.

- Access to the bank accounts is provided by a PIN code consisting of four integer digits between 0 and 9

ATM Case Study –System Requirements

- One PIN code allows access to all accounts held by a bank client
- No receipts will be provided for any account transactions
- The bank application operates for a single banking institution only
- Neither a checking nor a savings account can have a negative balance. The system should automatically with draw money from a related savings account if the requested withdrawal amount on the checking account is more than its current balance. If the balance on a savings account is less than the withdrawal amount requested, the transaction will stop and the bank client will be notified.

Apply Common Class patterns to ATM Case study

- Concept class - Availability
- Events class – Account, Checking account, Savings account, Transaction class
- Organization class – Bank class
- People class – BankClient class
- Places class – Building
- Tangible things and devices class – ATM Kiosk

Guidelines for Naming Classes

- The class should describe a single object, so it should be the singular form of noun.
- Use names that the users are comfortable with.
- The name of a class should reflect its intrinsic nature.

Guidelines for Naming Classes (Con't)

- By the convention, the class name must begin with an upper case letter.
- For compound words, capitalize the first letter of each word - for example, LoanWindow.

Summary

- Finding classes is not easy.
- The more practice you have, the better you get at identifying classes.
- There is no such thing as the “right set of classes.”
- Finding classes is an incremental and iterative process.