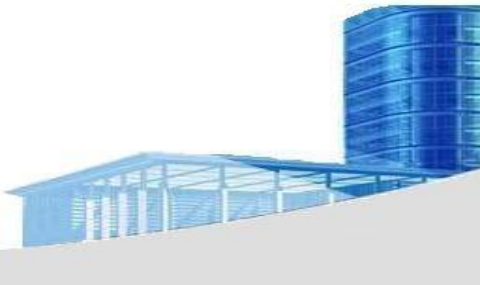
Ch.9 Requirements Modeling：

Scenario-Based Methods

March 25, 2024

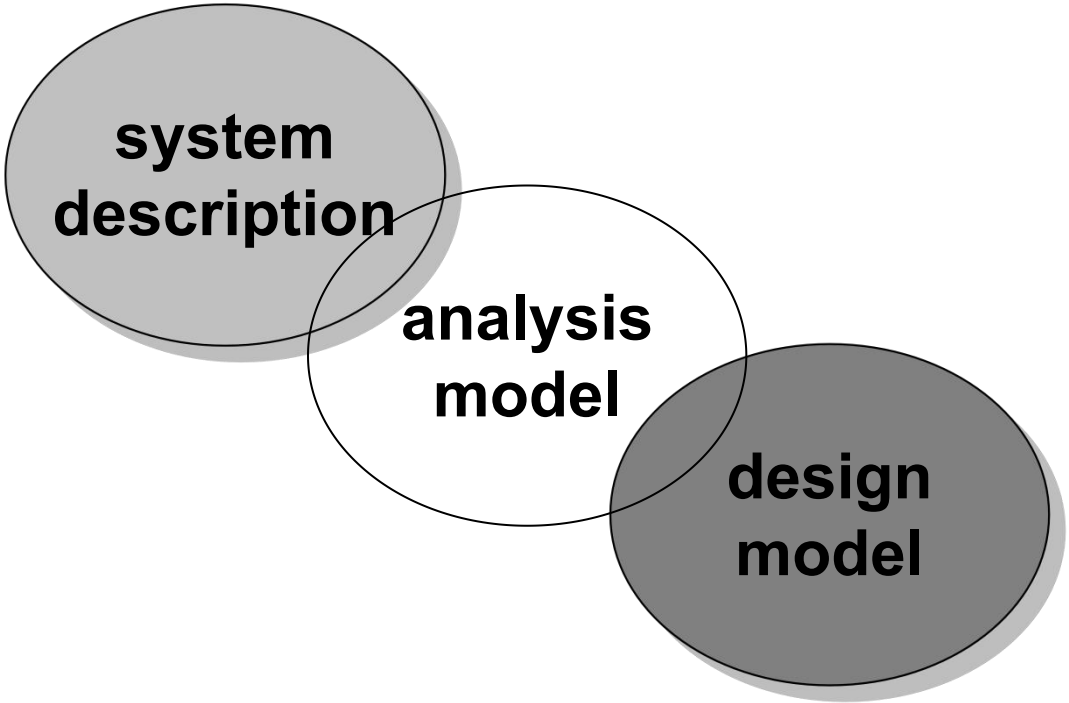


Requirements Analysis

- **Objectives**
  - **Describe what the customer requires**
  - **Establish a basis for the creation of a software design**
  - **Define a set of requirements that can be validated**

- **Requirements analysis allows the software engineer (called an *analyst* or *modeler* in this role) to:**
  - **elaborate on basic requirements established during earlier requirement engineering tasks**
  - **build models that depict user scenarios, functional activities, problem classes and their relationships, system and class behavior, the flow of data as it is transformed, constraints that software must meet.**

A Bridge

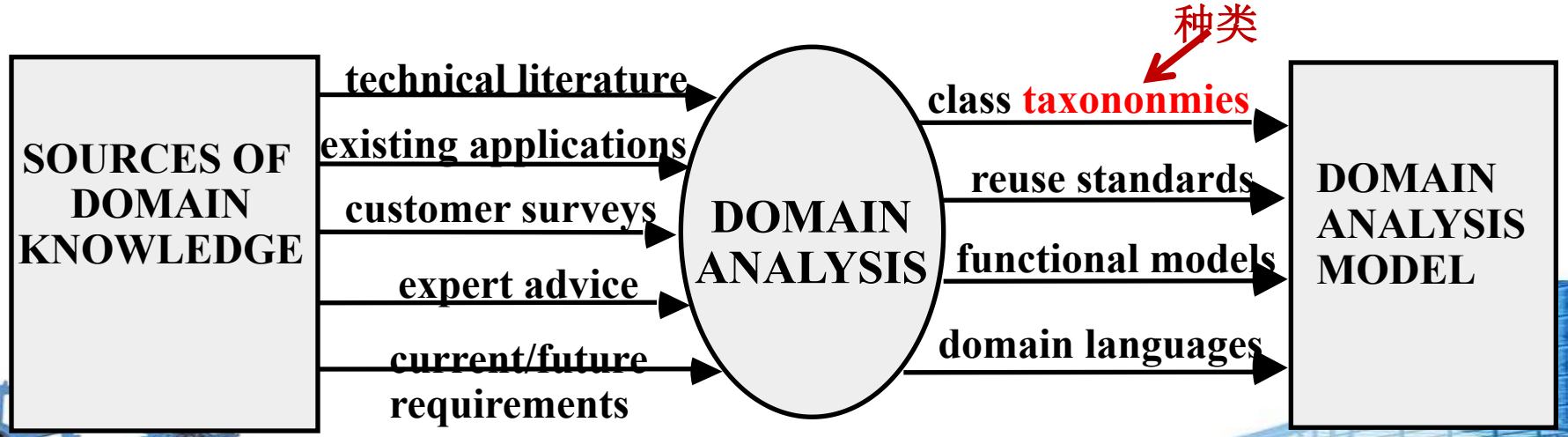


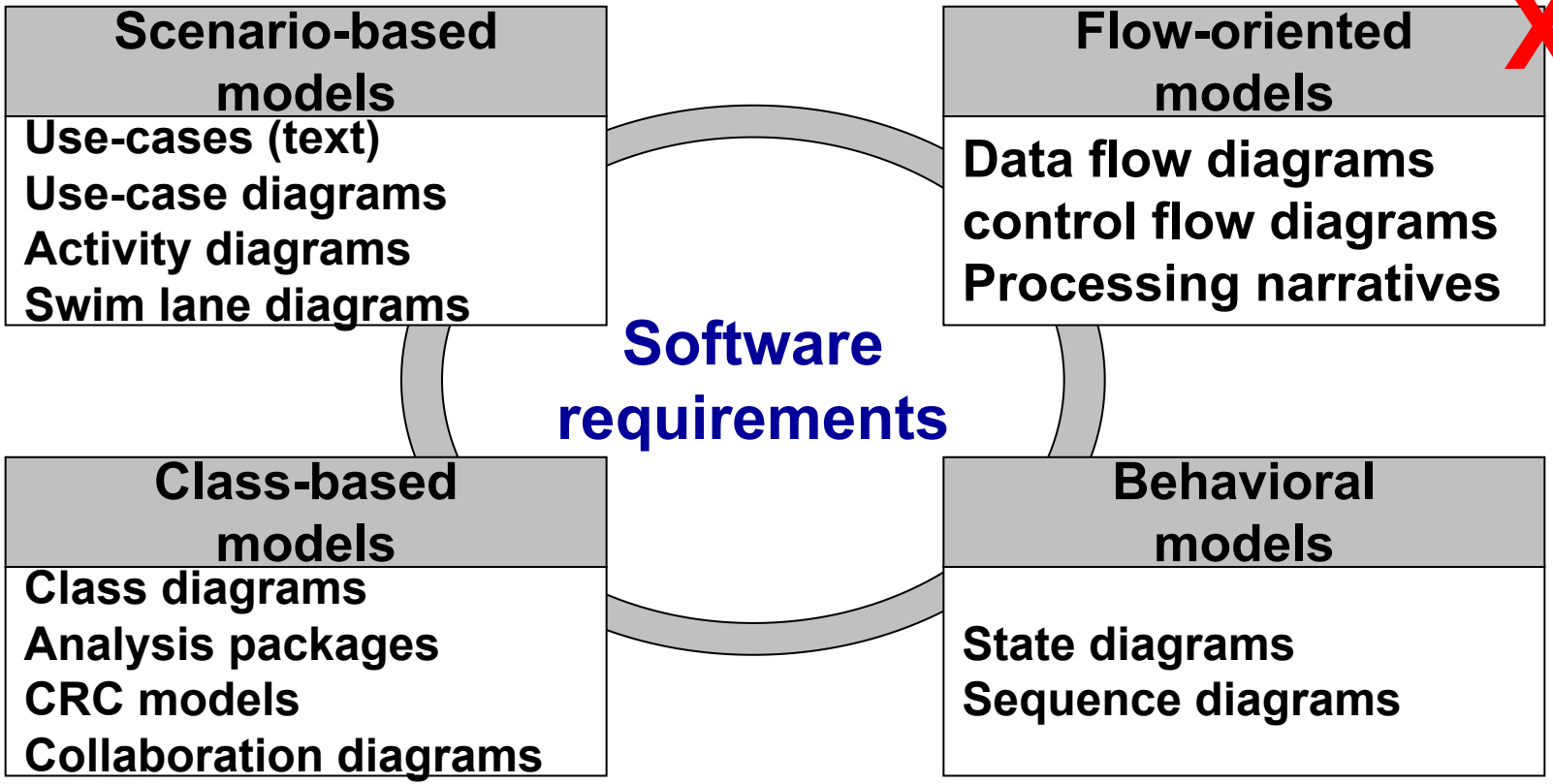
Rules of Thumb ←（经验法则）

- The model should focus on requirements that are visible within the problem or business domain. The level of abstraction should be relatively high.
- Each element of the analysis model should add to an overall understanding of software requirements and provide insight into the information domain, function and behavior of the system.
- Delay consideration of infrastructure（基础设施） and other non-functional models until design (手机 or 头盔 or 眼镜？).
- Minimize coupling throughout the system.
- Be certain that the analysis model provides value to all stakeholders.
- Keep the model as simple as it can be.

Domain（领域） Analysis

Goal: Software domain analysis is the identification, analysis, and specification of common requirements from a specific application domain, typically for reuse on multiple projects within that application domain . . .




Scenario-based models
Use-cases (text)
Use-case diagrams
Activity diagrams
Swim lane diagrams
Flow-oriented models
Data flow diagrams
control flow diagrams
Processing narratives
X?
Software requirements
Class-based models
Class diagrams
Analysis packages
CRC models
Collaboration diagrams
Behavioral models
State diagrams
Sequence diagrams

Scenario-Based Modeling

Use-cases are simply an aid to defining what exists outside the system (**actors**) and what should be performed by the system

(1) What should we write about?

(2) How much should we write about it?

(3) How detailed should we make our description?

(4) How should we organize the description?

Use-Cases

- a scenario that describes a “thread of usage” for a system
- *actors* represent roles people or devices play as the system functions
- *users* can play a number of different roles for a given scenario

Developing a Use-Case

- What are the main tasks or functions that are performed by the actor?
- What system information will the actor acquire, produce or change?
- Will the actor have to inform the system about changes in the external environment?
- What information does the actor desire from the system?
- Does the actor wish to be informed about unexpected changes?

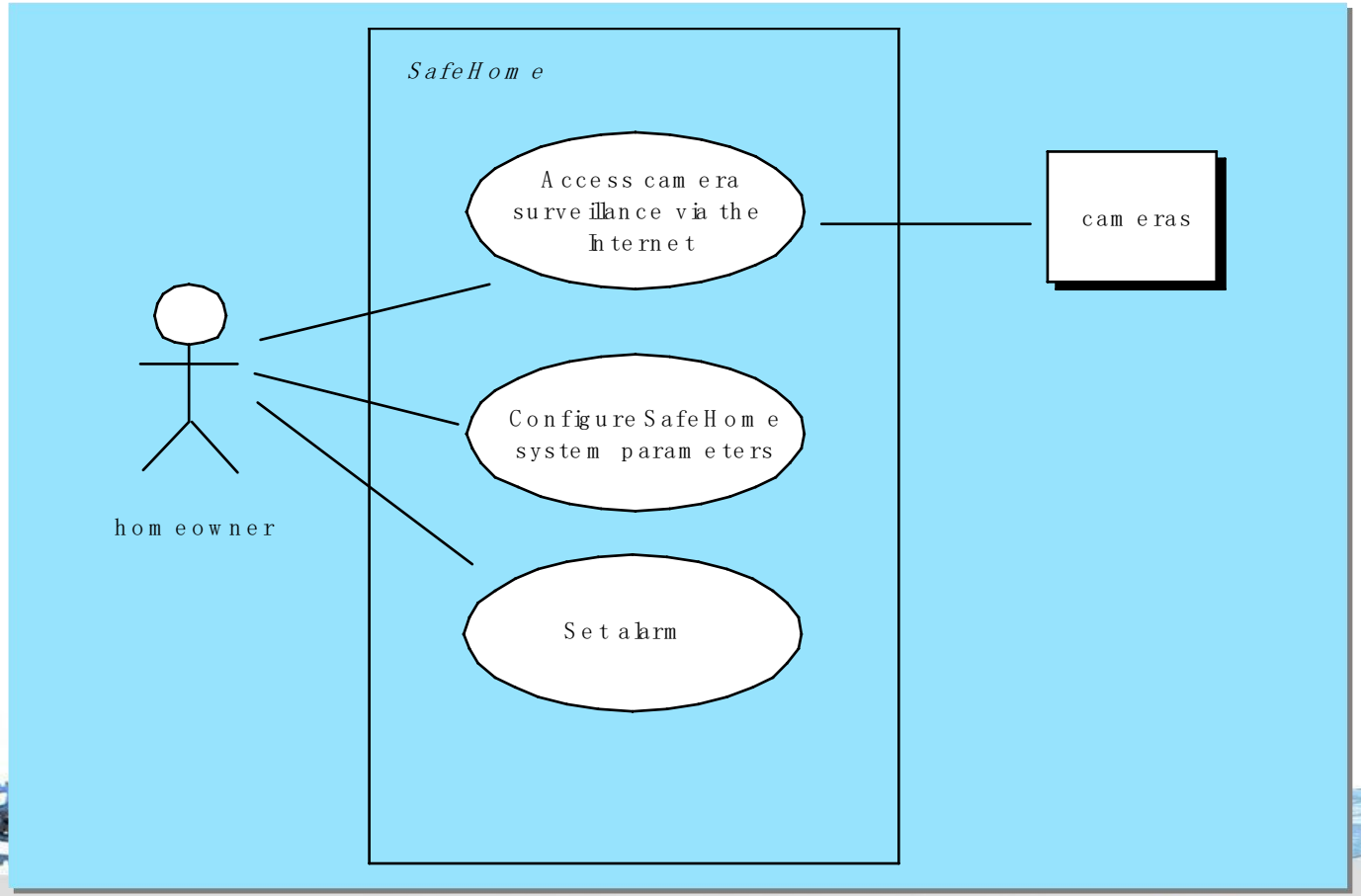
Reviewing a Use-Case

- Use-cases are written first in narrative（叙述的） form and mapped to a template if formality is needed

- Each primary scenario should be reviewed and refined to see if alternative interactions are possible

  - Can the actor take some other action at this point? e.g. SafeHome
  - Is it possible that the actor will encounter an error condition at some point? If so, what?
  - Is it possible that the actor will encounter some other behavior at some point? If so, what?

Use-Case Diagram

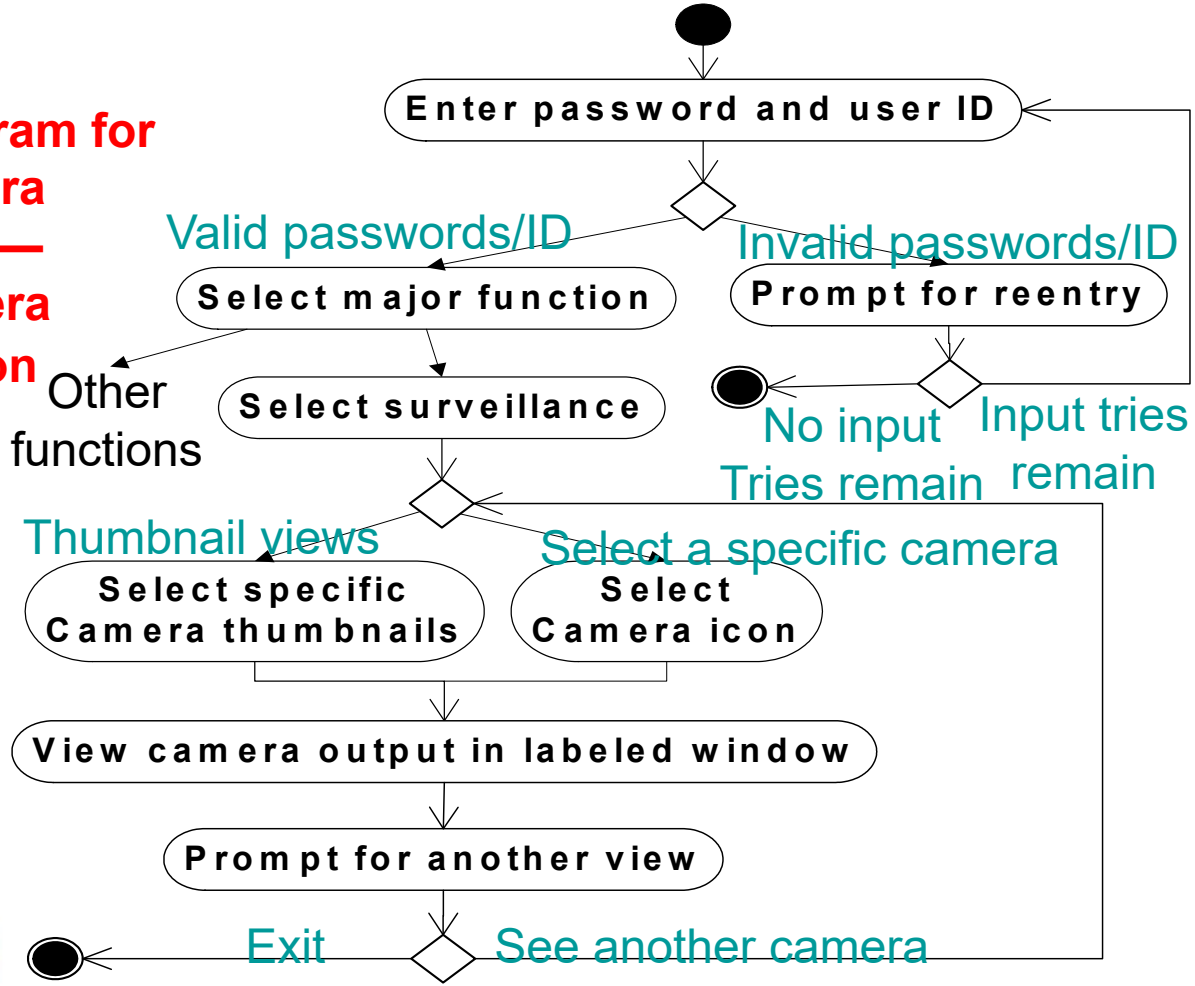


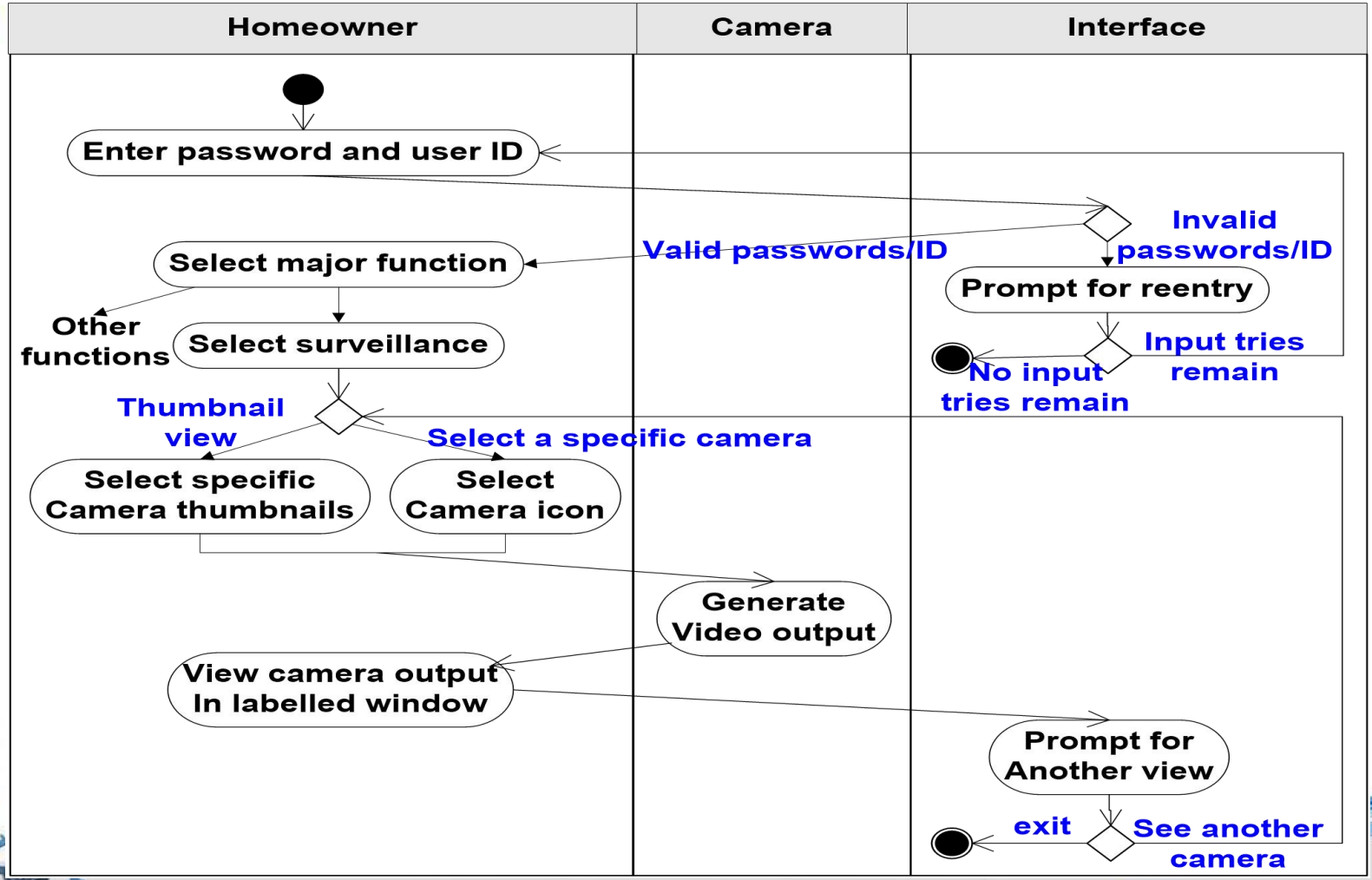
Activity and Swim Lane Diagrams

- **Activity diagram** supplements the use-case by providing a diagrammatic representation of procedural flow.

- **Swim lane diagram** allows the modeler to represent the flow of activities described by the use-case and at the same time indicate which actor (if there are **multiple actors** involved in a specific use-case) or analysis class has **responsibility** for the action described by an activity rectangle.

Activity diagram for access camera surveillance — display camera views function




Homeowner
Camera
Interface
Enter password and user ID
Invalid passwords/ID
Valid passwords/ID
Select major function
Prompt for reentry
Other functions
Select surveillance
Input tries remain
No input tries remain
Thumbnail view
Select a specific camera
Select specific Camera thumbnails
Select Camera icon
Generate Video output
View camera output In labelled window
Prompt for Another view
exit
See another camera

Ch.10 Requirements Modeling: Class-Based Methods

Requirements Modeling Strategies

- One view of requirements modeling, called **structured analysis**, considers data and the processes that **transform** the data as separate entities.
  - **Data objects** are modeled in a way that defines their attributes and relationships.
  - **Processes** that manipulate data objects are modeled in a manner that shows how they transform data as data objects flow through the system.
- A second approach to analysis modeled, called **object-oriented (O-O)analysis**, focuses on
  - the **definition of classes** and
  - the **manner** in which they **collaborate** with one another to **effect** customer requirements.

Object-Oriented Concepts

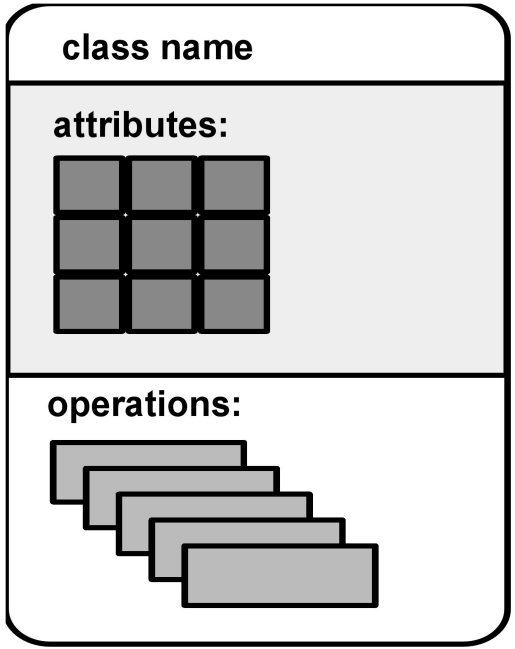
- Key concepts:
  - Classes and objects
  - Attributes and operations
  - Encapsulation and instantiation (实例)
  - Inheritance
- Tasks
  - Classes (attribute and method) must be identified
  - A class hierarchy is defined
  - Object relationship should be represented
  - Object behavior must be modeled
  - Above tasks are reapplied iteratively

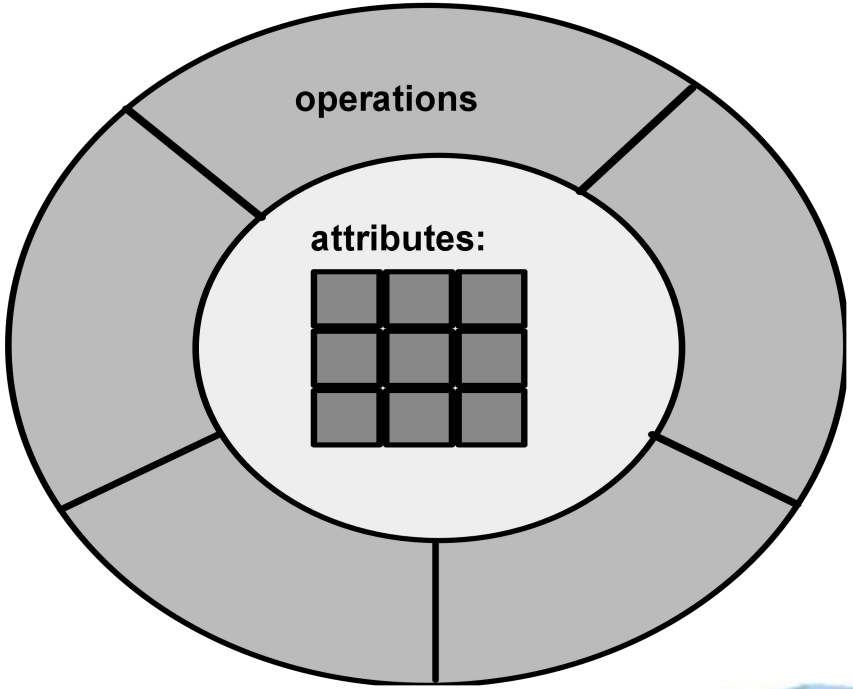
Why encapsulation?

Classes

- object-oriented thinking begins with the definition of a class, often defined as:
  - template
  - generalized description
  - describing a collection of similar items
- a metaclass (also called a superclass) establishes a hierarchy of classes
- once a class of items is defined, a specific instance of the class can be identified

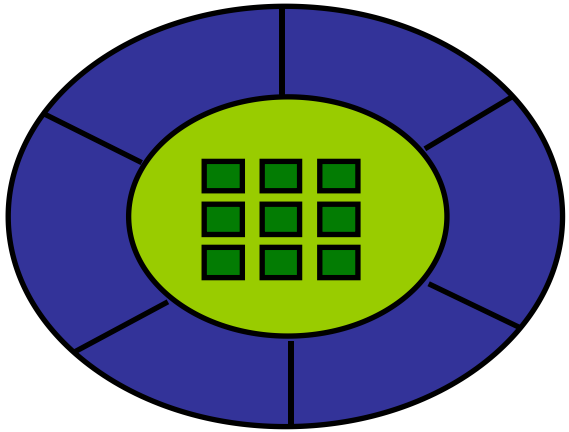
Building a Class





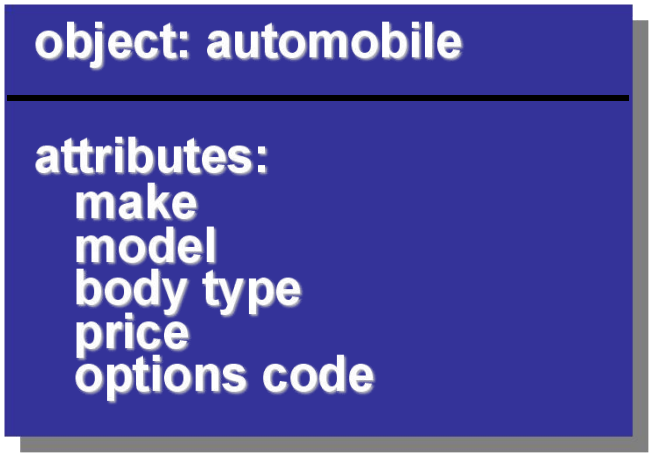
Methods

Also called operations or services. An executable procedure that is encapsulated in a class and is designed to operate on one or more data attributes that are defined as part of the class. A method is invoked via message passing.

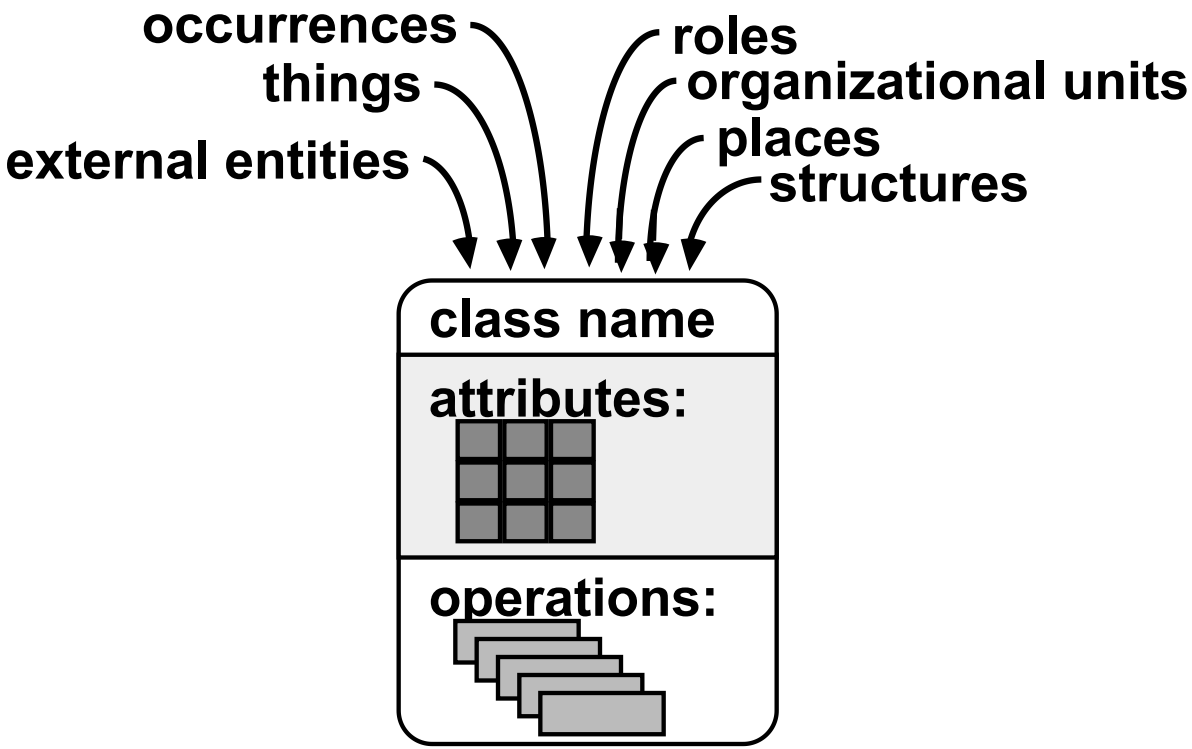


What are Data Attributes?

— A data object contains a set of attributes that act as an aspect, quality, characteristic, or descriptor of the object

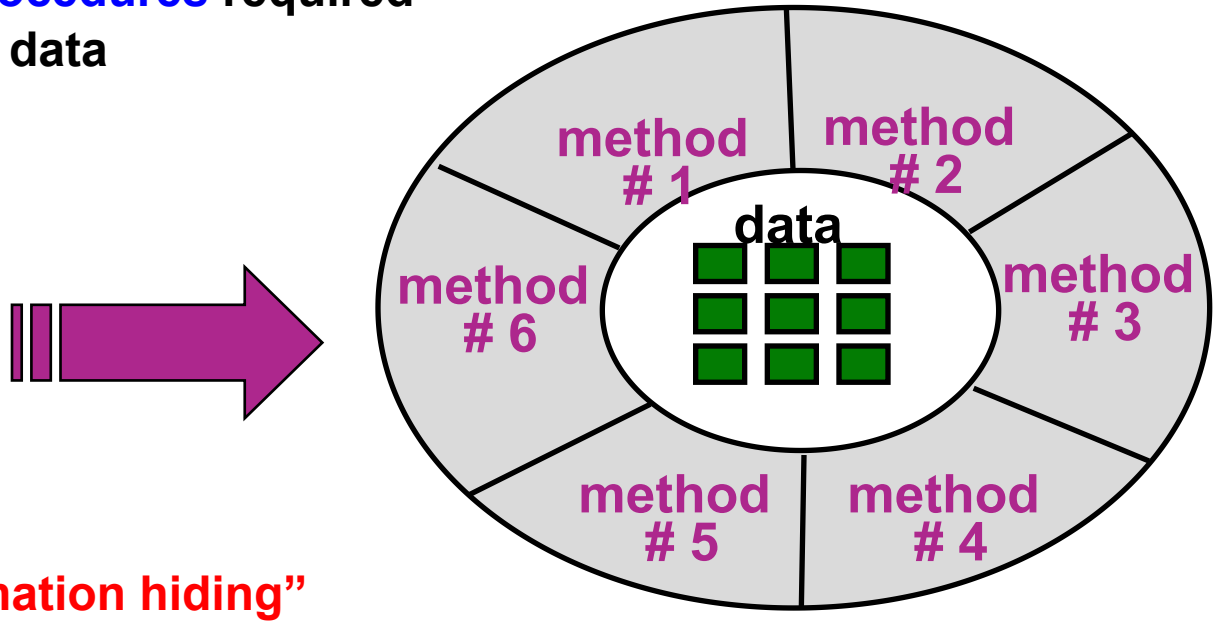


What is a Class?



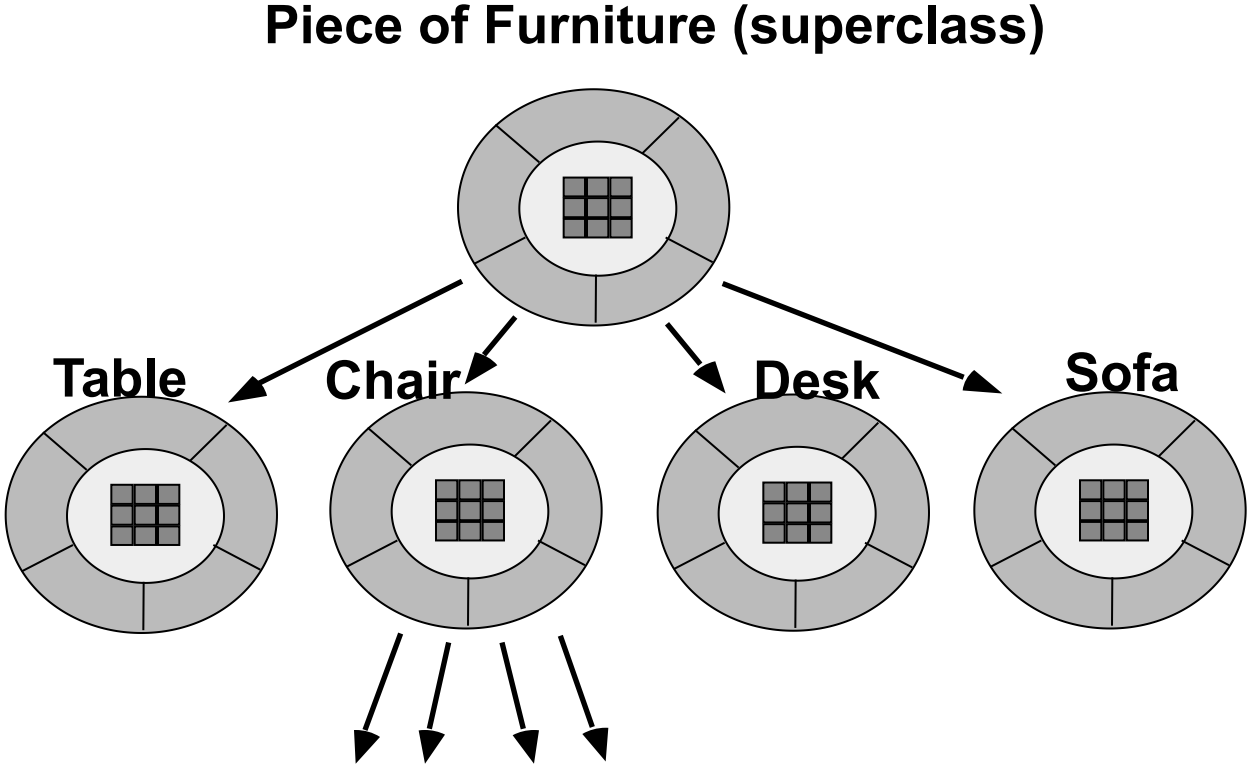
Encapsulation/Hiding

The object encapsulates both data and the logical procedures required to manipulate the data



Achieves "information hiding"

Class Hierarchy



Class-Based Modeling

- **Class-based modeling represents:**
  - **objects that the system will manipulate**
  - **operations (also called methods or services) that will be applied to the objects to effect the manipulation**
  - **relationships (some hierarchical) between the objects**
  - **collaborations that occur between the classes that are defined.**

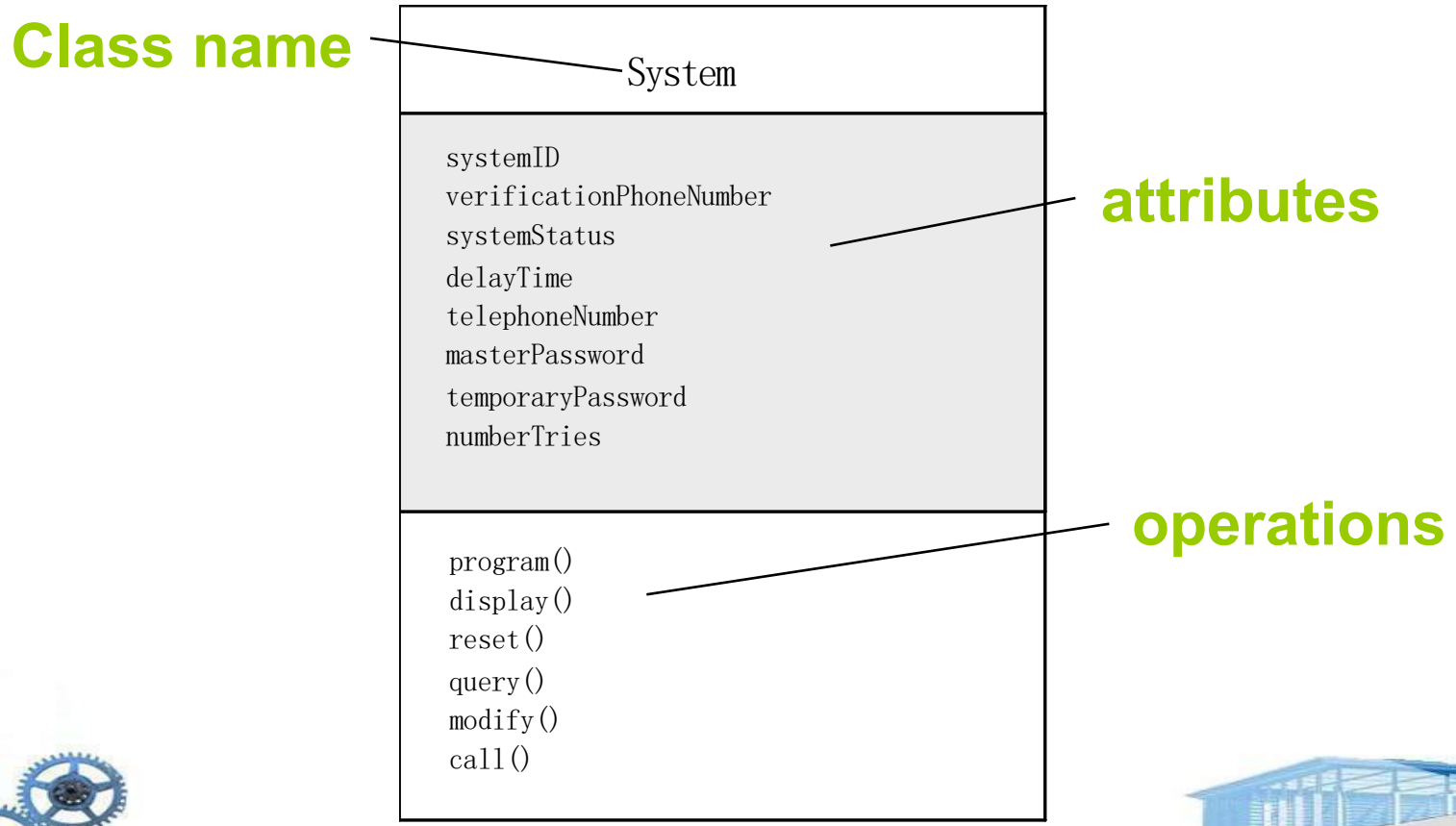
Class-Based Modeling

- Identify analysis classes by examining the problem statement
- Use a “grammatical parse(语法分析)” to isolate potential classes
- Identify the attributes of each class
- Identify operations that manipulate the attributes

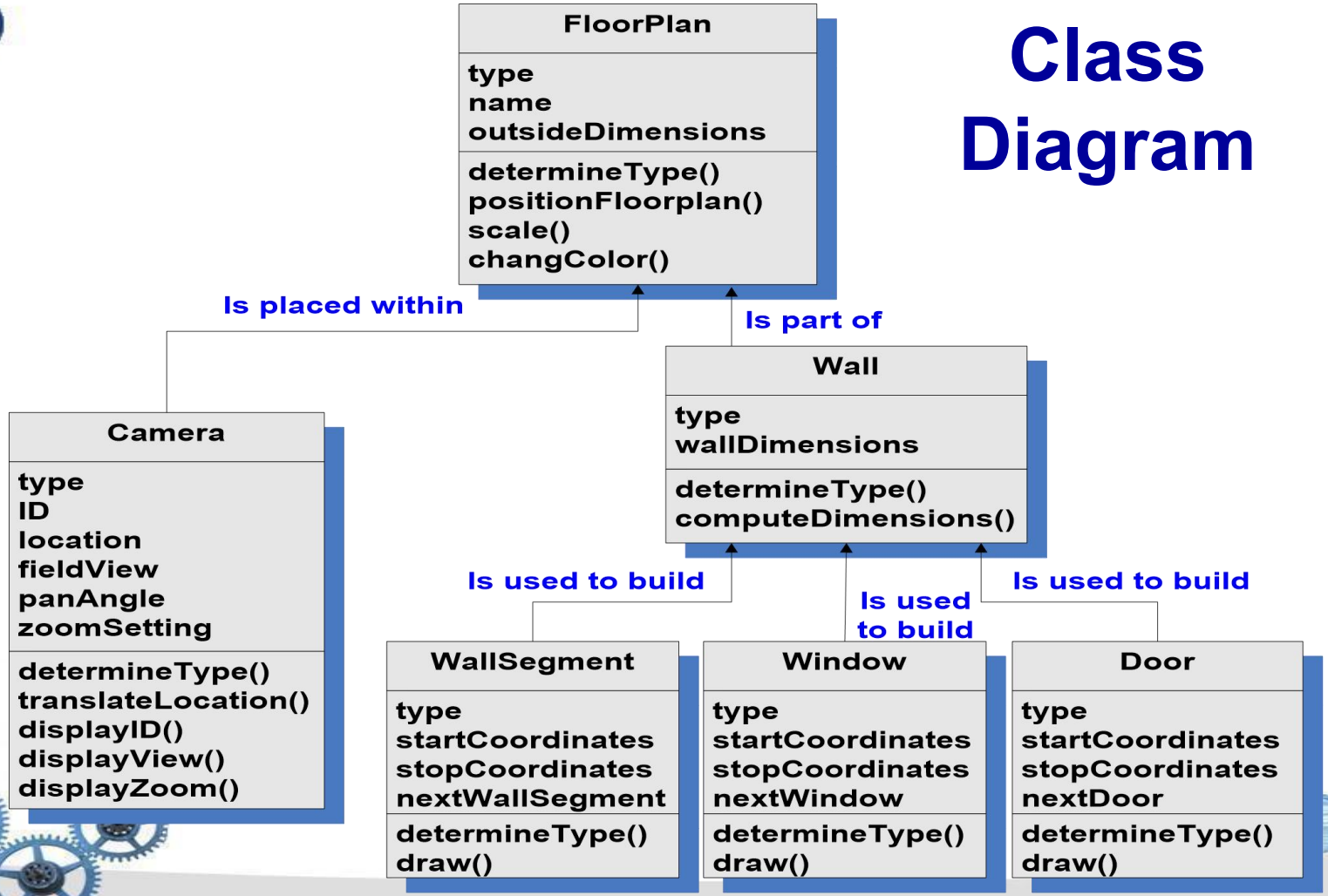
Potential Classes

- Retained(保持) information
- needed services
- multiple attributes
- common attributes
- common operations
- essential requirements

Class Diagram



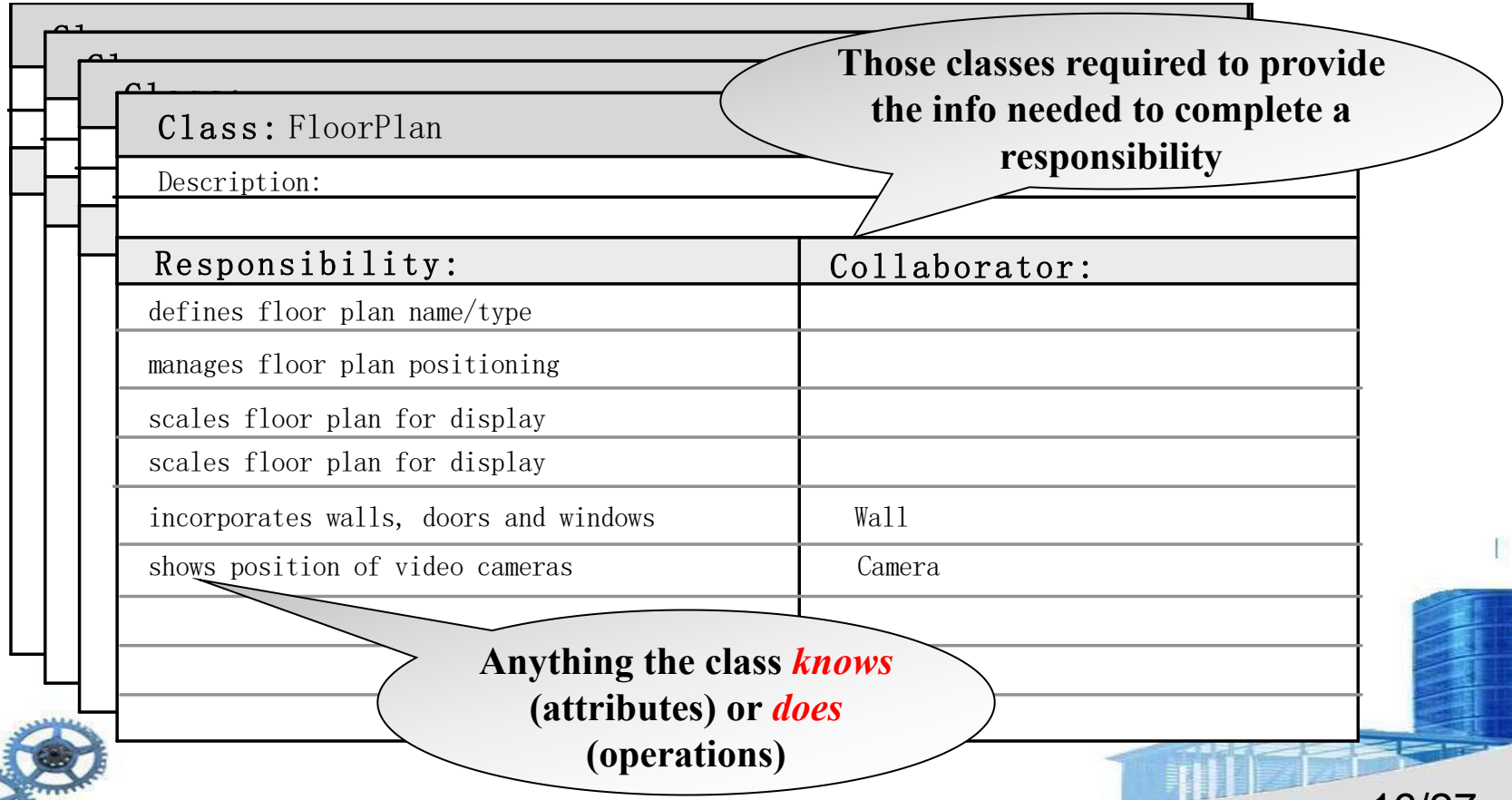
Class Diagram



CRC Modeling

- Analysis classes have “responsibilities”
  - *Responsibilities* are the attributes and operations encapsulated by the class
- Analysis classes collaborate with one another
  - *Collaborators* are those classes that are required to provide a class with the information needed to complete a responsibility.
  - In general, a *collaboration* implies either a request for information or a request for some action.

CRC Modeling



Class Types

- ***Entity classes***, also called *model* or ***business*** **classes**, are extracted directly from the statement of the problem
- ***Boundary classes*** are used to create the interface (e.g., **interactive screen** or **printed reports**) that the user sees and interacts with as the software is used.
- ***Controller classes*** manage a "unit of work" from start to finish. That is, controller classes can be designed to manage
  - the **creation or update** of entity objects;
  - the **instantiation** of boundary objects as they obtain information from entity objects;
  - **complex communication** between sets of objects;
  - **validation** of data communicated between objects or between the user and the application.

Guidelines for Allocating Responsibilities

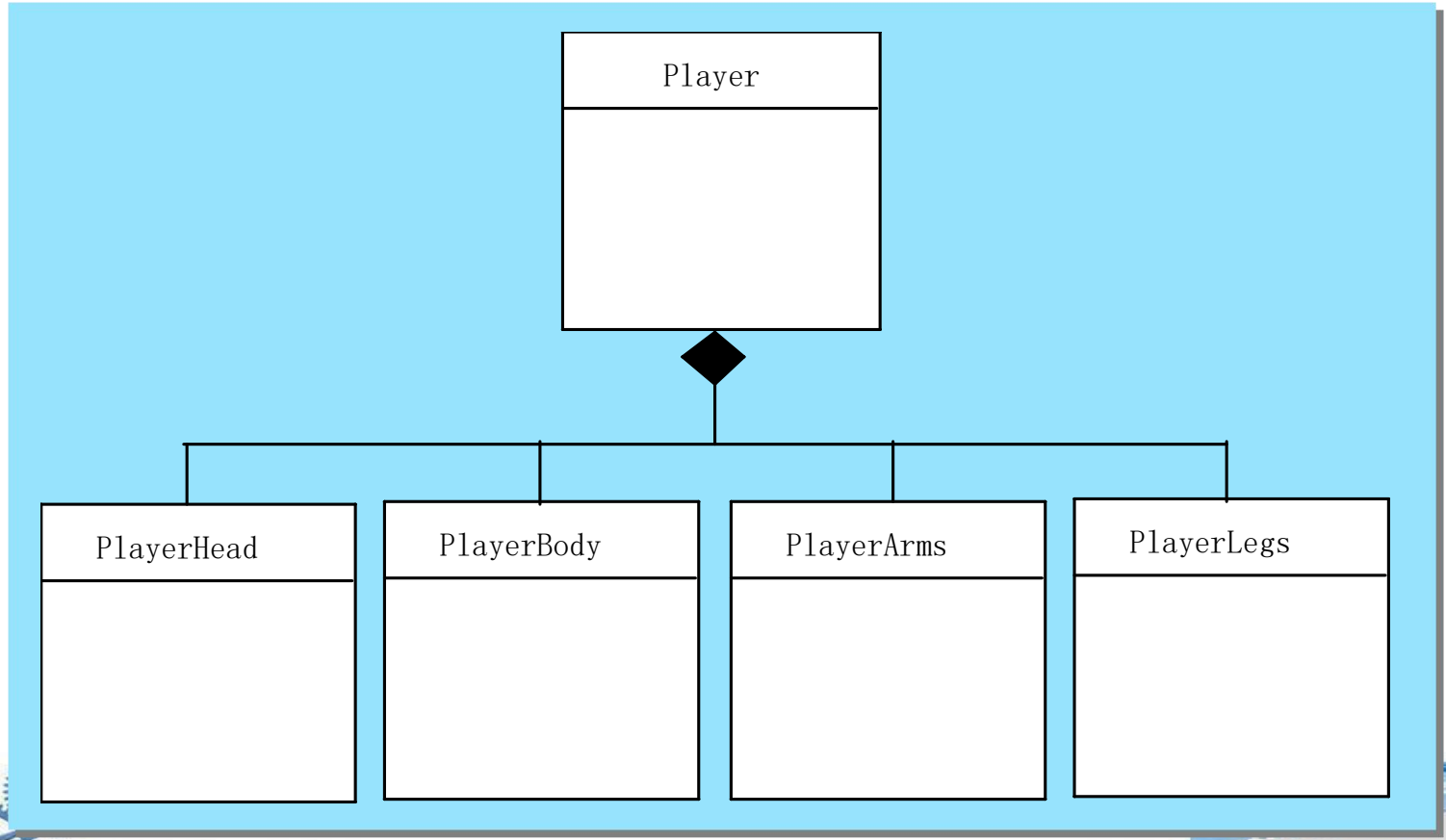
- System intelligence should be distributed across classes to best address(处理) the needs of the problem
- Each responsibility should be stated as generally as possible
- Information and the behavior related to it should reside within the same class
- Information about one thing should be localized with a single class, not distributed across multiple classes.
- Responsibilities should be shared among related classes, when appropriate.

Collaborations

- Classes fulfill their responsibilities in one of two ways:
  - A class can use its own operations to manipulate its own attributes, thereby fulfilling a particular responsibility, or
  - a class can collaborate with other classes.
- Collaborations identify relationships between classes
- three different generic relationships between classes
  - the *is-part-of* relationship
  - the *has-knowledge-of*（有…的知识）relationship Ex. ControlPanel vs Sensor
  - the *depends-upon* relationship

Ex. PlayerHead depends-upon Playerbody

Composite Aggregate(聚合的) Class



Reviewing the CRC Model

- All participants in the review (of the CRC model) are given a subset（子集） of the CRC model index cards.
  - Cards that collaborate should be separated (i.e., no reviewer should have two cards that collaborate).

- All use-case scenarios (and corresponding use-case diagrams) should be organized into categories.

- The review leader reads the use-case deliberately (细致地).
  - As the review leader comes to a named object, she passes a token（令牌） to the person holding the corresponding class index card.

Reviewing the CRC Model (cont.)

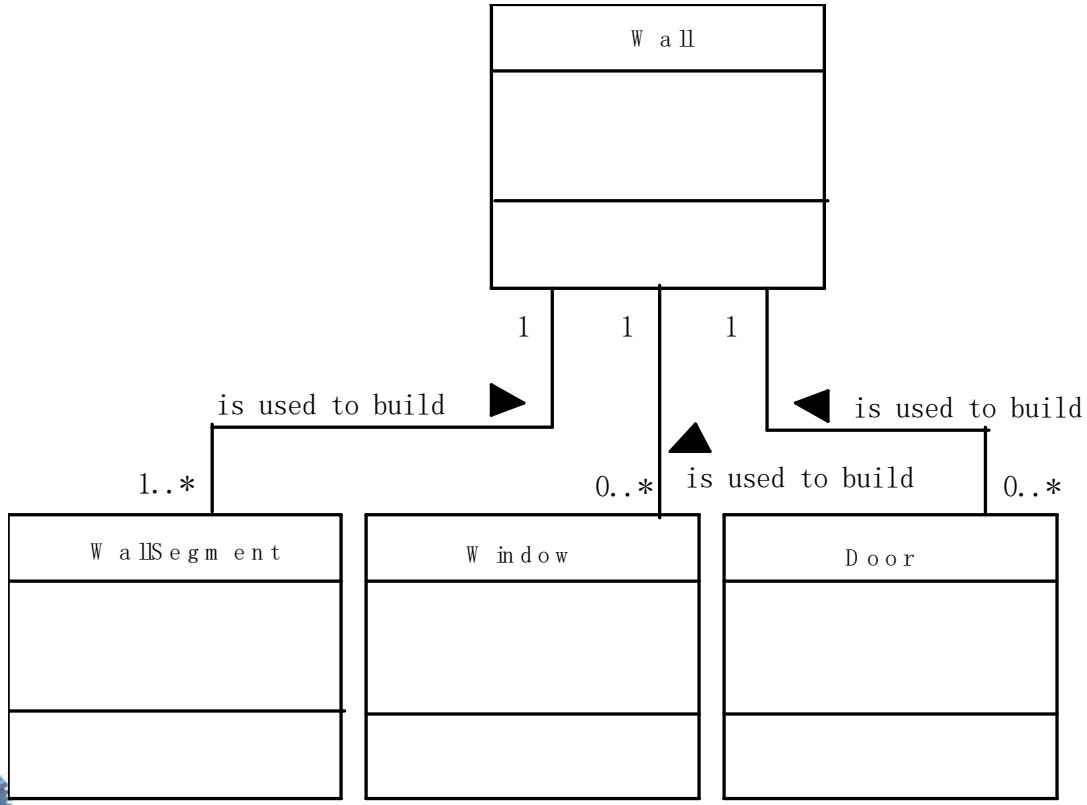
- When the token is passed, the holder of the class card is asked to describe the responsibilities noted on the card.
  - The group determines whether one (or more) of the responsibilities satisfies the use-case requirement.
- If the responsibilities and collaborations noted on the index cards cannot accommodate the use-case, modifications are made to the cards.
  - This may include the definition of new classes (and corresponding CRC index cards) or the specification of new or revised responsibilities or collaborations on existing cards.

Associations and Dependencies

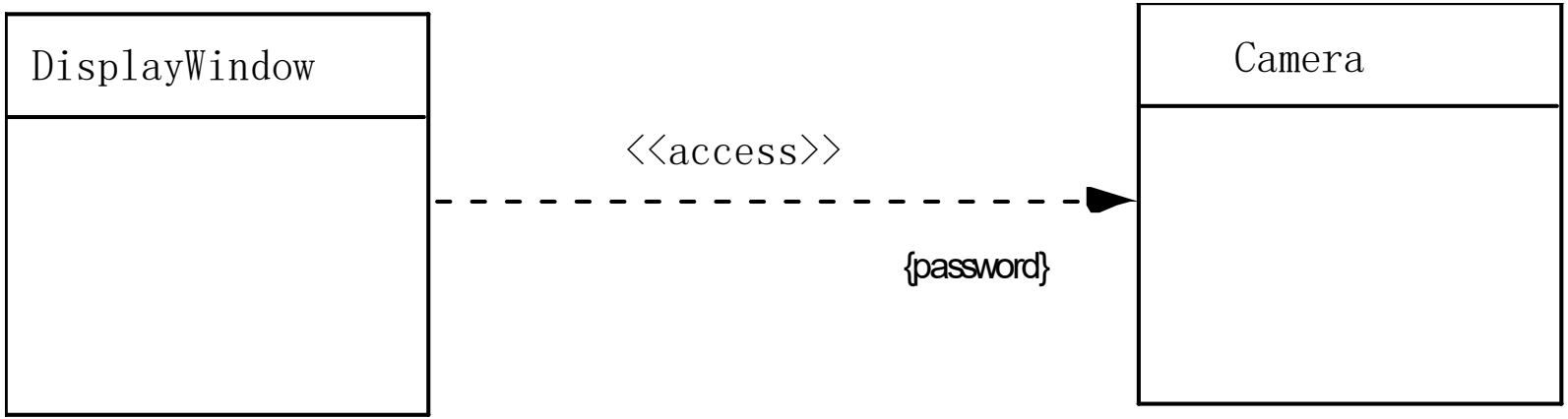
- Two analysis classes are often related to one another in some fashion
  - In UML these relationships are called associations
  - Associations can be refined by indicating multiplicity (the term cardinality(基数) is used in data modeling

- In many instances, a client-server relationship exists between two analysis classes.
  - In such cases, a client-class depends on the server-class in some way and a dependency relationship is established

Multiplicity



Dependencies



客户类,对象

服务器类

Analysis Packages

- Various elements of the analysis model (e.g., use-cases, analysis classes) are categorized in a manner that packages them as a grouping
- The plus sign（+） preceding the analysis class name in each package indicates that the classes have public visibility and are therefore accessible from other packages.
- Other symbols can precede an element within a package. A minus sign（-） indicates that an element is hidden from all other packages。
- A # symbol indicates that an element is accessible only to classes contained within a given package.

Analysis Packages

Package name

Environment
- +Tree
- +Landscape
- +Road
- +Wall
- +Bridge
- +Building
- +VisualEffect
- +Scene

RulesOfTheGame
- +RulesOfMovement
- +ConstraintsOnAction

Characters
- +Player
- +Protagonist
- +Antagonist
- +SupportingRole

主角

对手

2019.3.11---腾讯游戏学院访问浙大CAD&CG国家重点实验室

Ch.11 Requirements Modeling: Behavior, Patterns, and Web/Mobile Apps

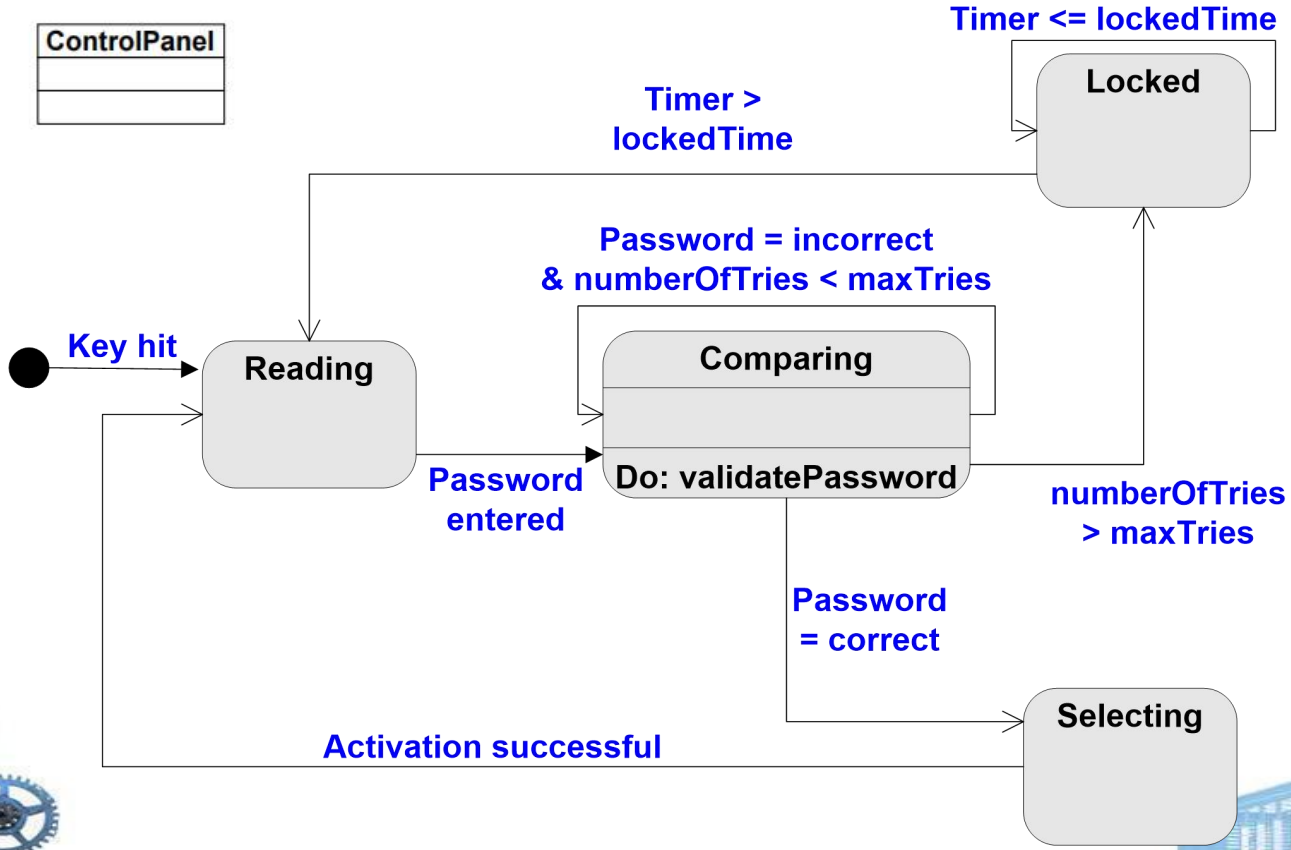
Behavioral Modeling

- The behavioral model indicates how software will respond to external events or stimuli. To create the model, the analyst must perform the following steps:
  - **Evaluate** all use-cases to fully understand the sequence of interaction within the system.
  - **Identify** events that drive the interaction sequence and understand how these events relate to specific objects.
  - **Create** a sequence for each use-case.
  - **Build** a state diagram for the system.
  - **Review** the behavioral model to verify accuracy and consistency.

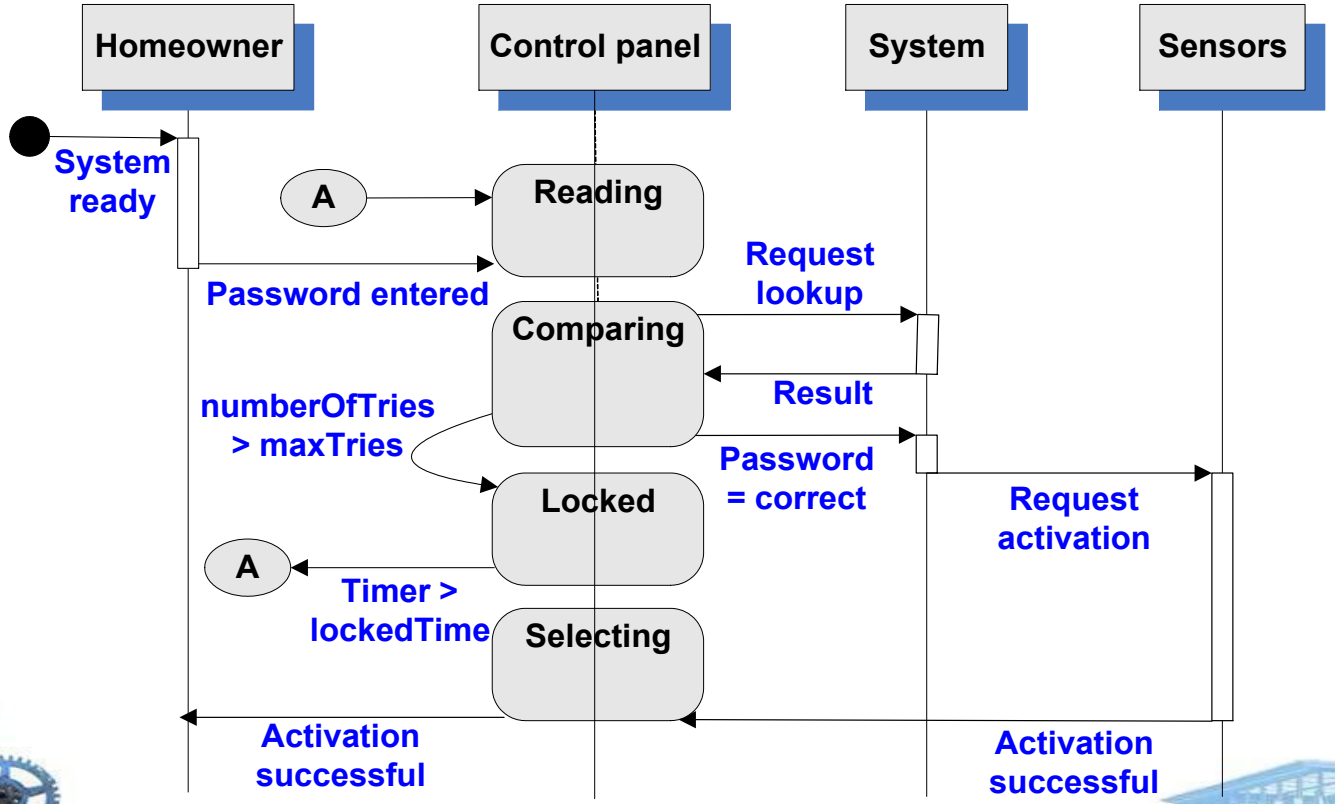
Behavioral Modeling

- In the context of behavioral modeling, two different characterizations of states must be considered:
  - the state of each class as the system performs its function
  - the state of the system as observed from the outside as the system performs its function

State Diagram



Sequence Diagram

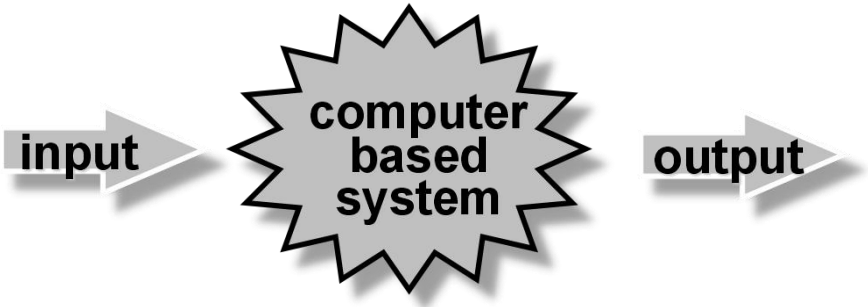


Flow-Oriented Modeling

- Represents how data objects are transformed at they move through the system
- A data flow diagram (DFD) is the diagrammatic form that is used
- Considered by many to be an ‘old school’ approach, flow-oriented modeling continues to provide a view of the system that is unique — it should be used to supplement other analysis model elements

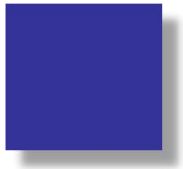
Data Flow Diagram (DFD)

- Every computer-based system is an information transform system
- Is a graphical representation that depicts information flow and the transforms that are applied as data move from input to output
- Be used to represent a system or software at any level of abstraction



Notation of DFD

External Entity



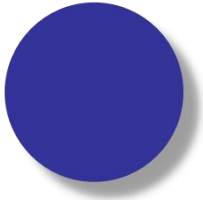
A producer or consumer of data

Examples: a person, a device, a sensor

Another example: computer-based system

Data must always originate somewhere and must always be sent to something

Process



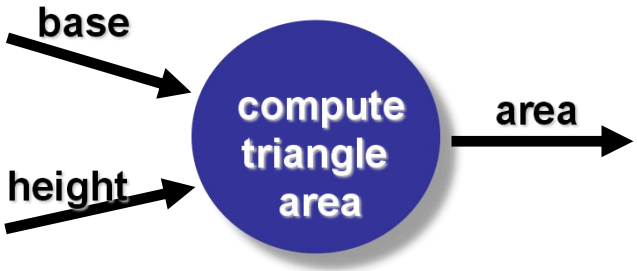
A data transformer (changes input to output)

Examples: compute taxes, determine area, format report, display graph

Data must always be processed in some way to achieve system function

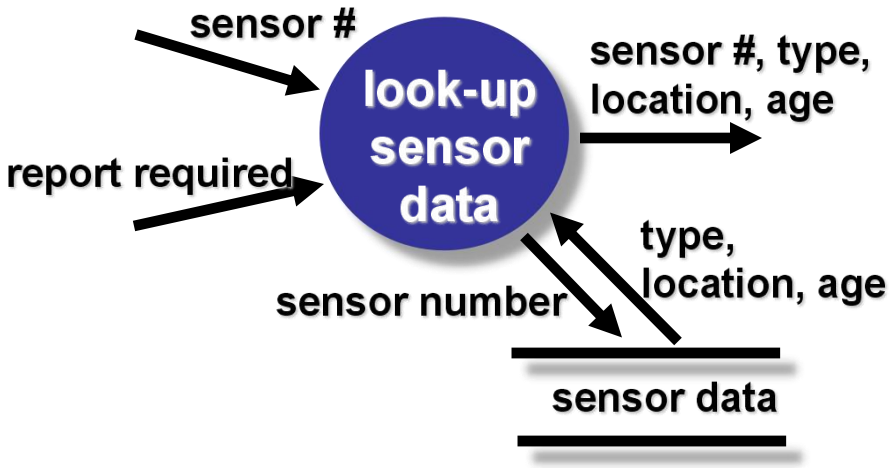
Data

Data flows through a system, beginning as input and be transformed into output.



Data Stores

Data is often stored for later use.



Flow-Oriented Modeling

- Example: [From《Fundamentals of Software Engineering》]

  Information System of a Public Library

  if { user requests a book (title, author, user’s name) }

  { Get a book }

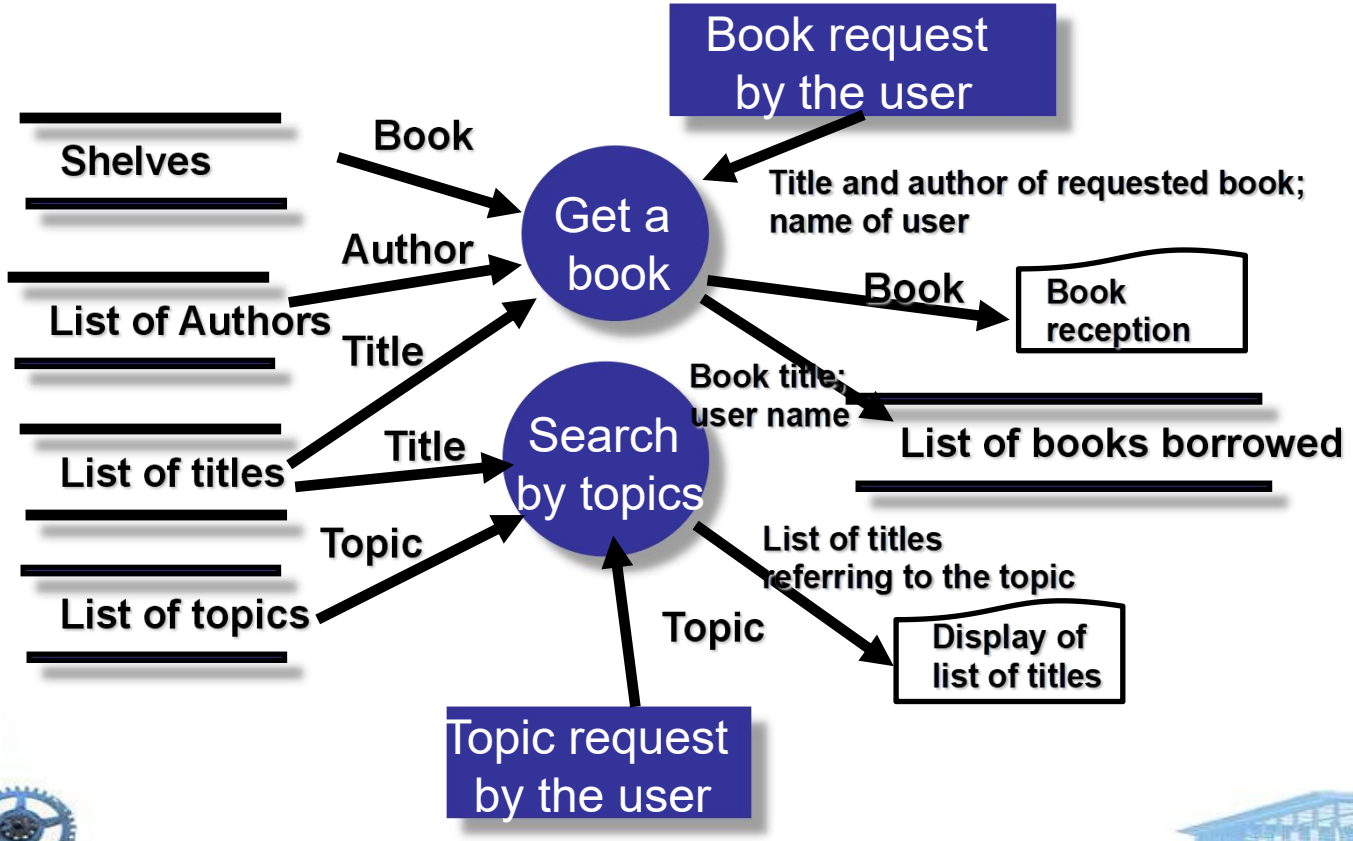
  → book, and user’s list of books borrowed;

  if { user searches a book by topics }

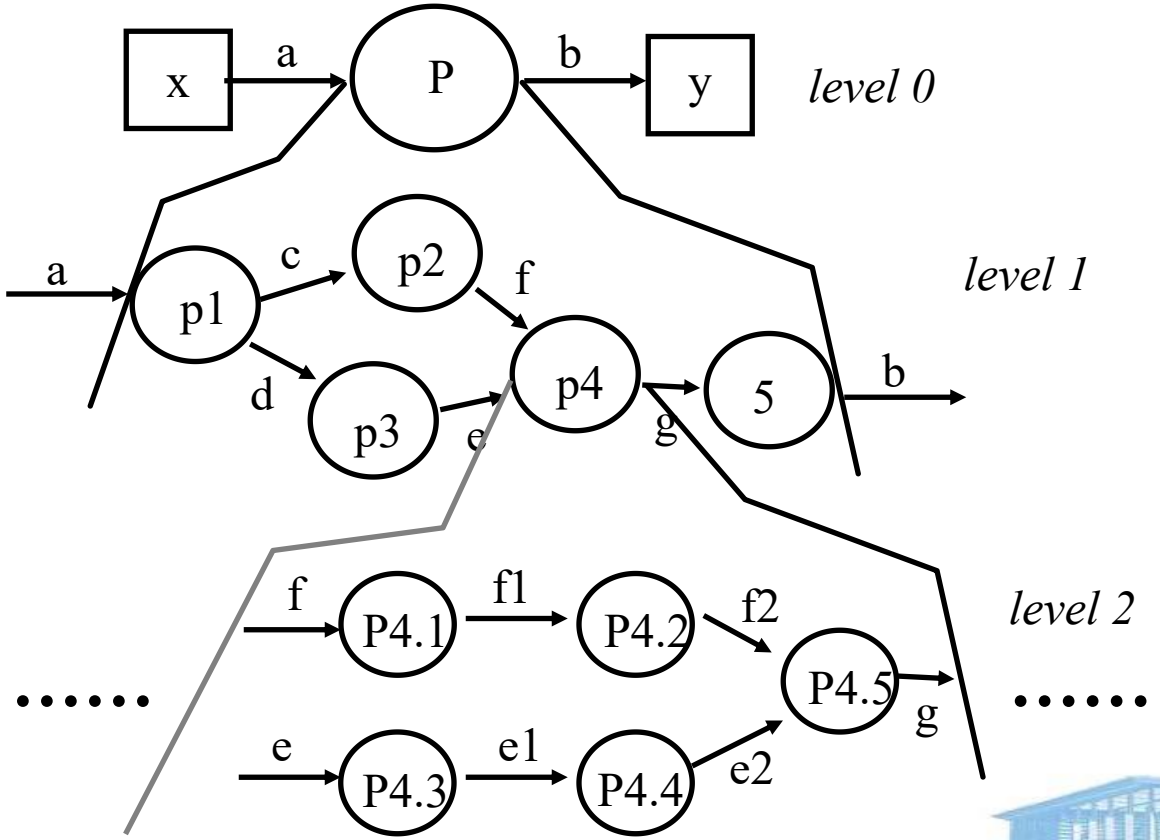
  { Search by topics }

  → list of book titles referring to the topic.

Flow-Oriented Modeling



The Data Flow Hierarchy



Writing the Software Specification

Specification Guidelines

- Use a **layered format** that provides increasing detail as the "layers" deepen
- Use **consistent graphical notation** and apply textual terms **consistently** (stay away from aliases（别名）)
- Be sure to define all **acronyms** 首字母缩写词，如：APEC
- Be sure to include a **table of contents**; ideally, include an **index** and/or a **glossary**
- Write in a **simple, unambiguous** style
- Always **put yourself in the reader's position**, "Would I be able to understand this if I wasn't intimately familiar with the system?"

Requirements Modeling for WebApps

In some Web/Mobile App situations, analysis and design merge.
However, an explicit analysis activity occurs when:

- The Web or Mobile App to be built is large and/or complex
- The number of stakeholders is large (Ex. 众筹)
- The number of developers is large
- The development team members have not worked together before
- the success of the app will have a **strong bearing**（关系）on the success of the business

Requirements Modeling for WebApps

- **Content Analysis.** The full spectrum of content to be provided by the WebApp is identified, including **text, graphics and images, video, and audio**. Data modeling can be used to identify and describe each of the data objects.
- **Interaction Analysis.** The manner in which the user interacts with the WebApp is described in detail. **Use-cases** can be developed to provide detailed descriptions of this interaction.
- **Functional Analysis.** The usage scenarios (use-cases) created as part of interaction analysis define **the operations that will be applied to WebApp content and imply other processing functions**. All operations and functions are described in detail.
- **Configuration Analysis.** The **environment and infrastructure** in which the WebApp resides are described in detail.
- **Navigation Analysis.** The organization of web page link.Focus on **overall requirements**.

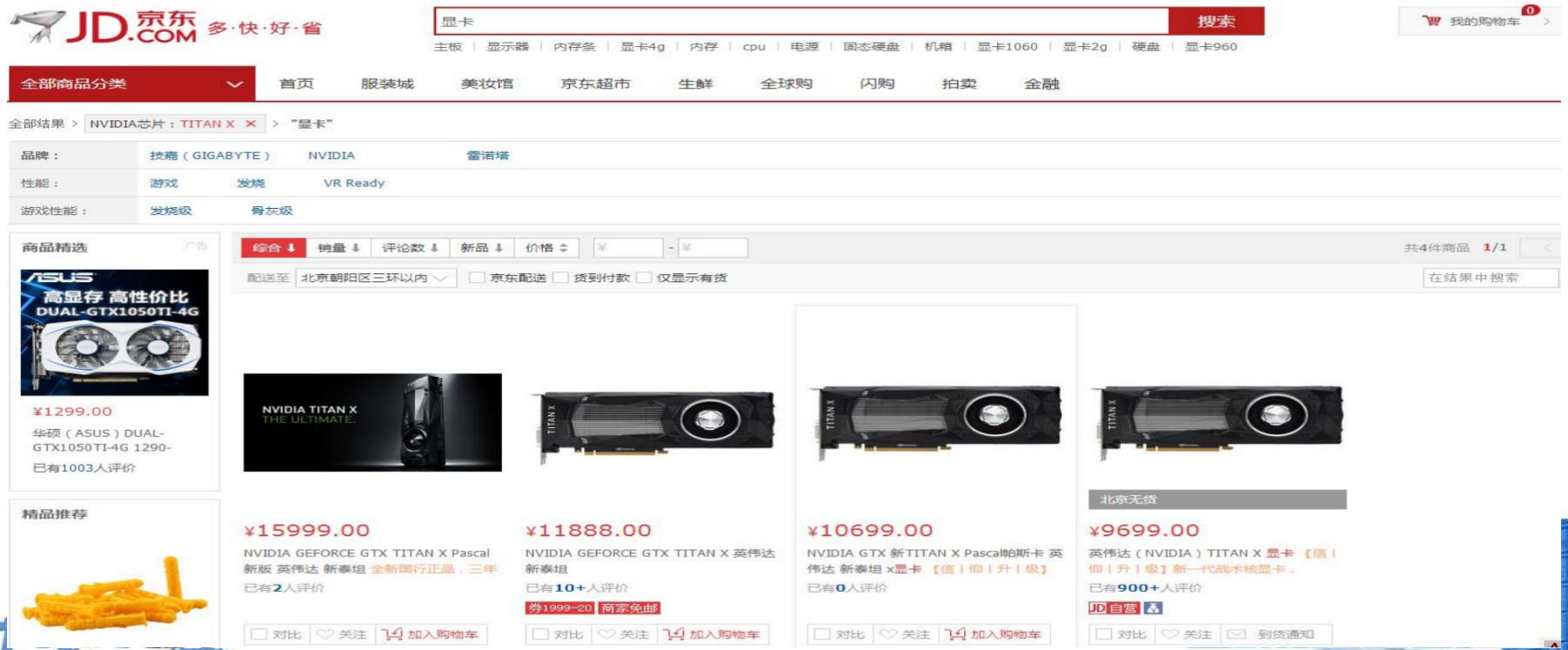
The Configuration Model

- **Server-side**
  - **Server hardware and operating system** environment must be specified
  - **Interoperability** considerations on the server-side must be considered
  - Appropriate **interfaces**, **communication protocols** and related **collaborative information** must be specified
- **Client-side**
  - Browser configuration issues must be identified
  - Testing requirements should be defined

Navigation Modeling-I

- Should certain elements be **easier to reach** (require fewer navigation steps) than others? What is the priority for presentation?
- Should certain elements be **emphasized** to force users to navigate in their direction?
- How should navigation errors be handled?
- Should navigation to **related groups of elements** be given priority over navigation to **a specific element**.
- Should navigation be accomplished via **links**, via **search-based access**, or by some other means?
- Should certain elements be presented to users based on the **context of previous navigation actions**?
- Should a **navigation log** be maintained for users?

Navigation Modeling-I



Navigation Modeling-II

- Should a full **navigation map or menu** (as opposed to a single “back” link or directed pointer) be available at every point in a user’s interaction?
- Should navigation design be driven by the **most commonly expected user behaviors** or by the **perceived importance** of the defined WebApp elements?
- Can a user “store” his previous navigation through the WebApp to **expedite future usage**?
- For which **user category** should optimal navigation be designed?
- How should **links external** to the WebApp be handled? **overlaying** the existing browser window? as a **new browser window**? as a **separate frame**?

Tasks

- **Review** Ch. 9-11
- **Finish** “Problems and points to ponder” in Ch. 9-11
- **Preview** Ch. 12, 19
- **Submit Software Requirements Specification due April 1!**

- Experimental time this afternoon(曹西503，4:15-5:50pm)

---1) Team(A/B) leader meeting(4:00-4:15pm, or 1:00-1:15pm)；

---2）Group meeting(4:15-4:30pm)； ---3) **Preview time** (4:30-5:30pm);

---4) Quiz 1(5:30-5:50pm)