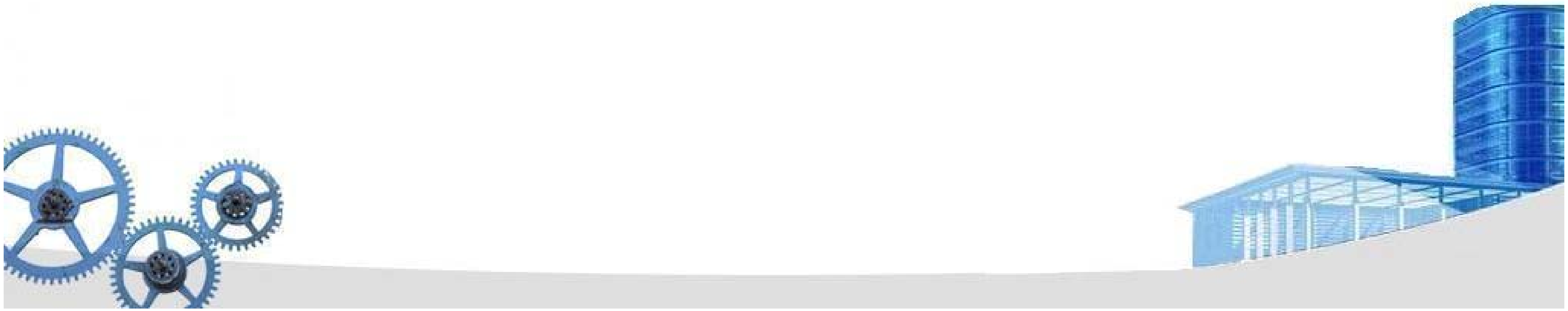
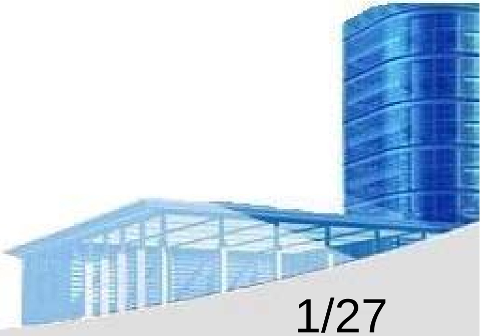
Ch.10 Requirements Modeling: Class-Based Methods

April 20, 2015



Requirements Modeling Strategies

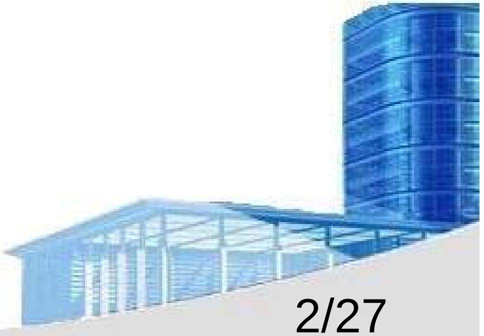
- One view of requirements modeling, called structured analysis, considers data and the processes that transform the data as separate entities.
  - Data objects are modeled in a way that defines their attributes and relationships.
  - Processes that manipulate data objects are modeled in a manner that shows how they transform data as data objects flow through the system.
- A second approach to analysis modeled, called object-oriented 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
  - 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**

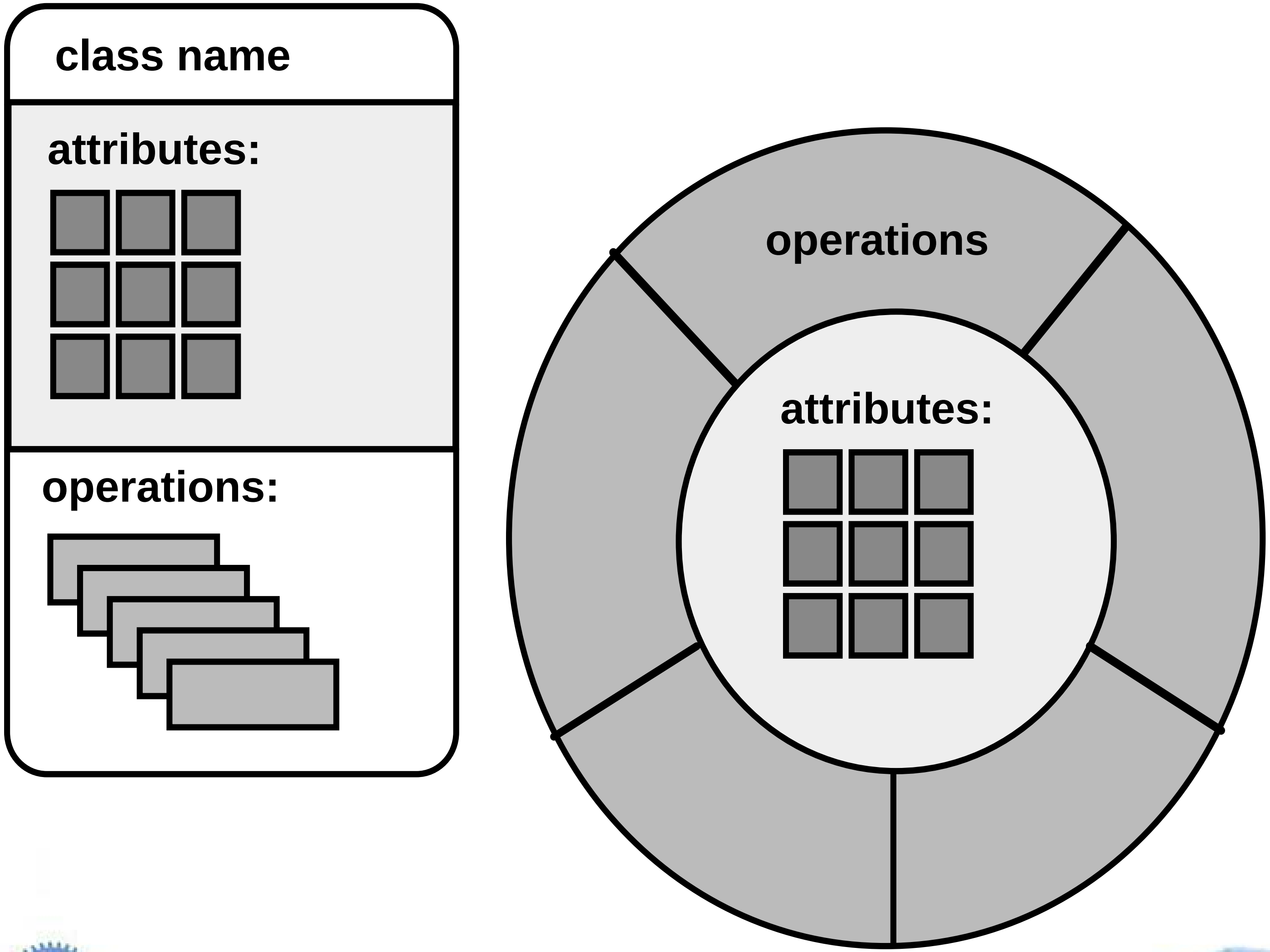
Why encapsulation?



Classes

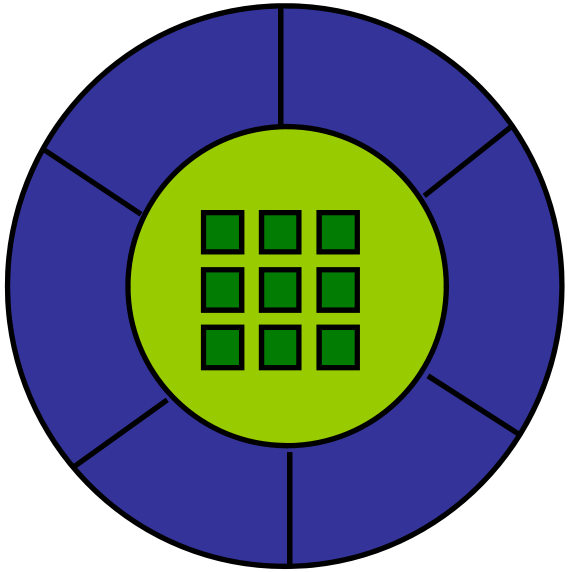
- object-oriented thinking begins with the definition of a class, often defined as:
  - template
  - generalized description
  - describing a collection of similar items
- a metaclass (also called a superclass) establishes a hierarchy of classes
- once a class of items is defined, a specific instance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



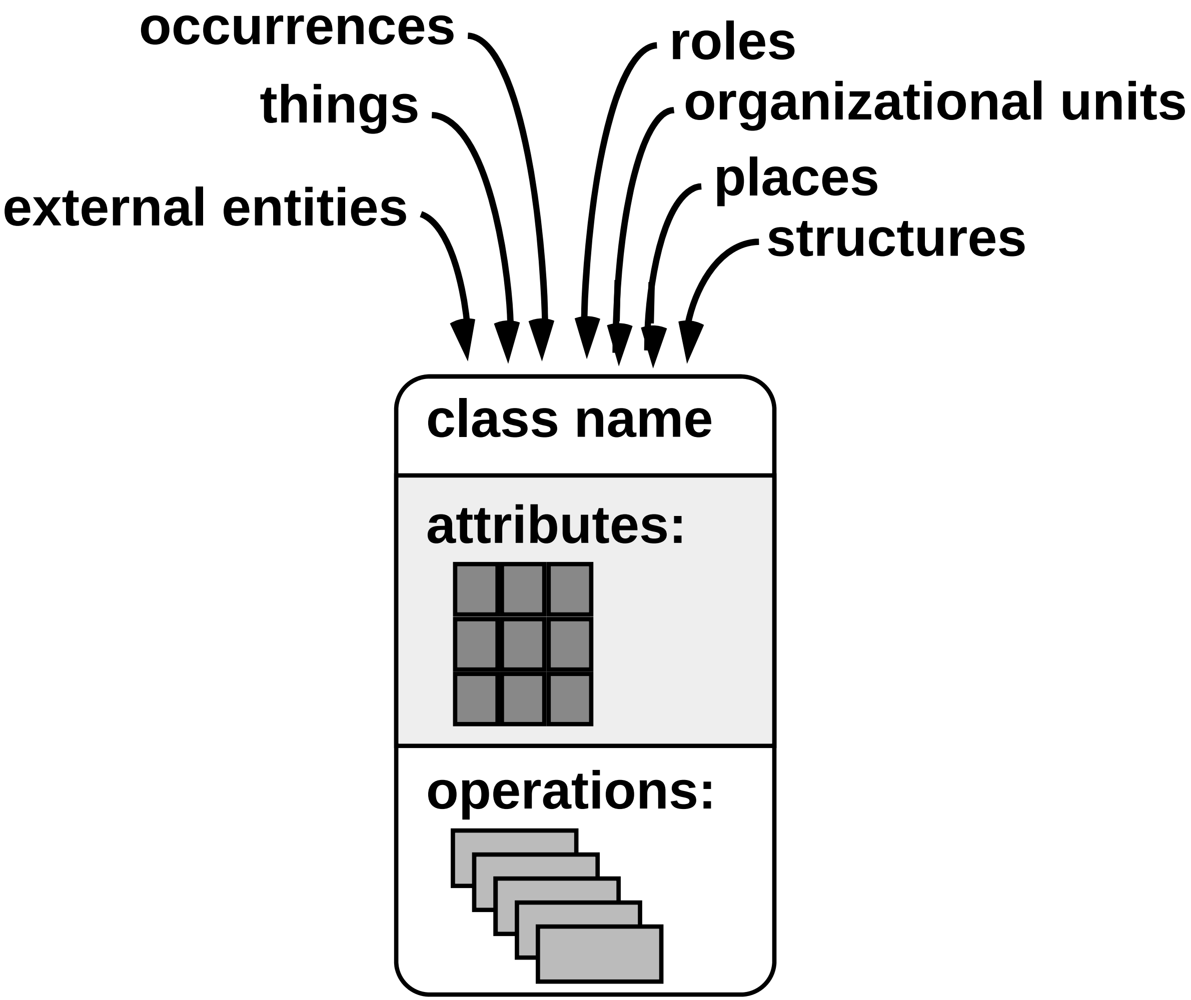
What are Data Attributes?

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

object: automobile

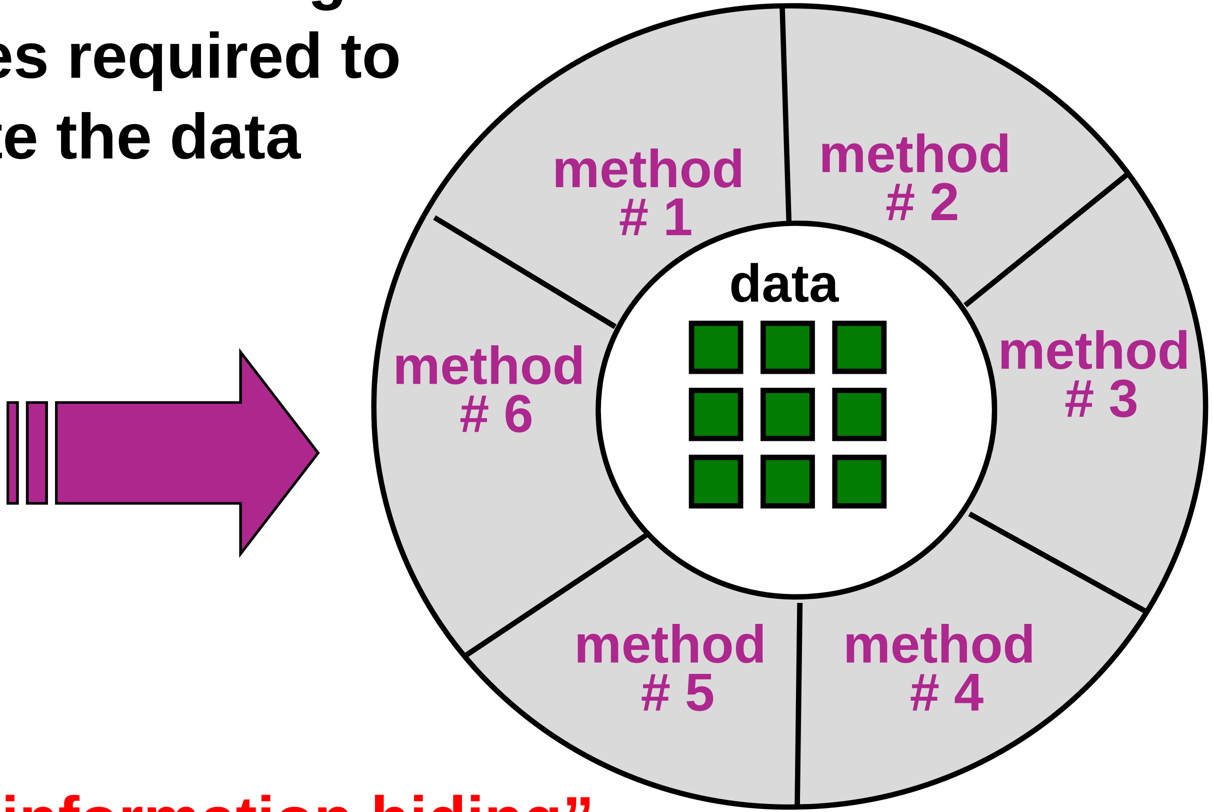
attributes:
- **make**
- **model**
- **body type**
- **price**
- **options code**

What is a Class?



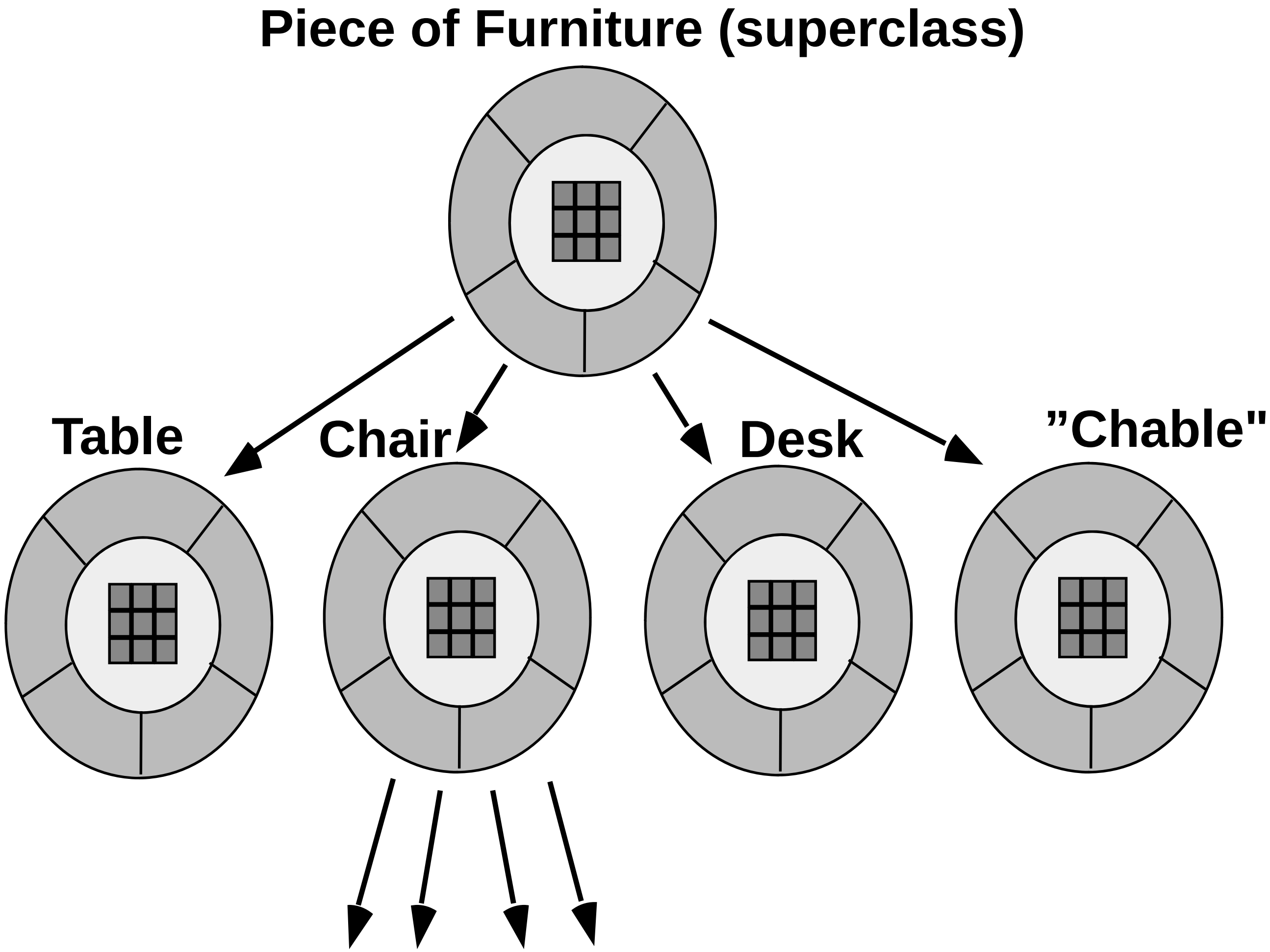
Encapsulation/Hiding

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.

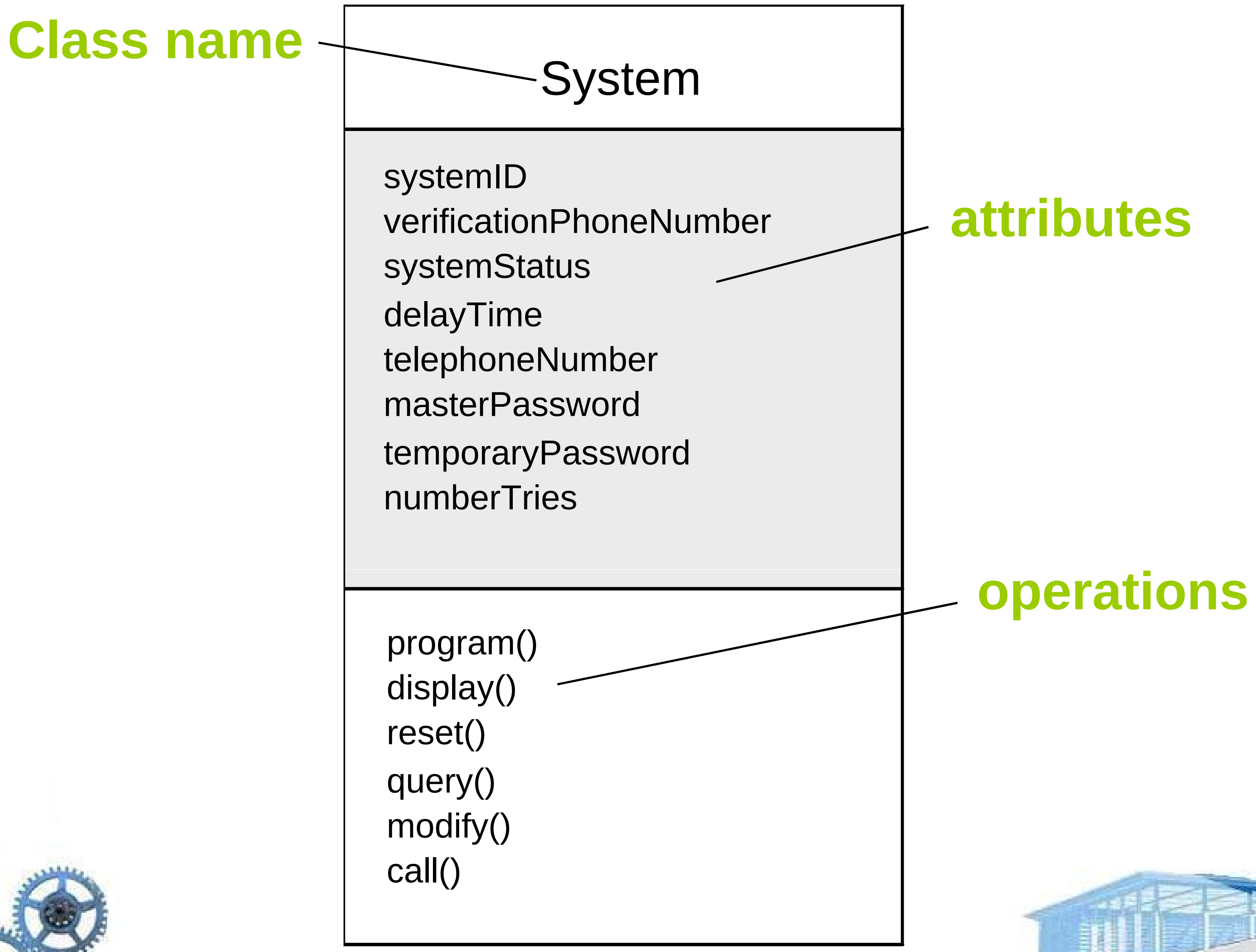
Class-Based Modeling

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

Potential Classes

- ☑ retained information
- ☑ needed services
- ☑ multiple attributes
- ☑ common attributes
- ☑ common operations
- ☑ essential requirements

Class Diagram



Class name

System

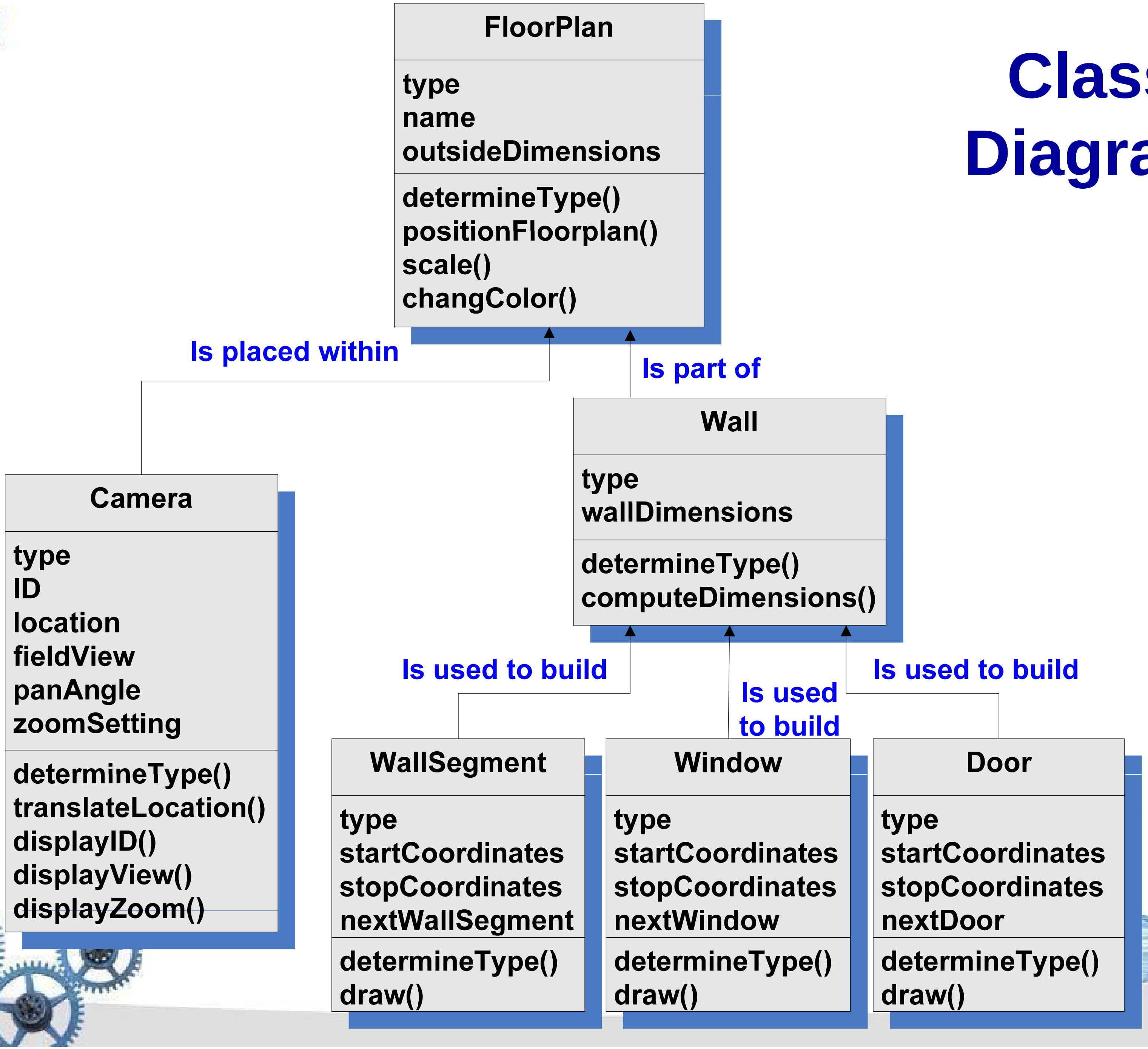
systemID
verificationPhoneNumber
systemStatus
delayTime
telephoneNumber
masterPassword
temporaryPassword
numberTries

attributes

program()
display()
reset()
query()
modify()
call()

operations

Class Diagram



FloorPlan

type
name
outsideDimensions

determineType()
positionFloorplan()
scale()
changColor()

Is placed within

Is part of

Camera

type
ID
location
fieldView
panAngle
zoomSetting

determineType()
translateLocation()
displayID()
displayView()
displayZoom()

Wall

type
wallDimensions

determineType()
computeDimensions()

Is used to build

Is used to build

Is used to build

WallSegment

type
startCoordinates
stopCoordinates
nextWallSegment

determineType()
draw()

Window

type
startCoordinates
stopCoordinates
nextWindow

determineType()
draw()

Door

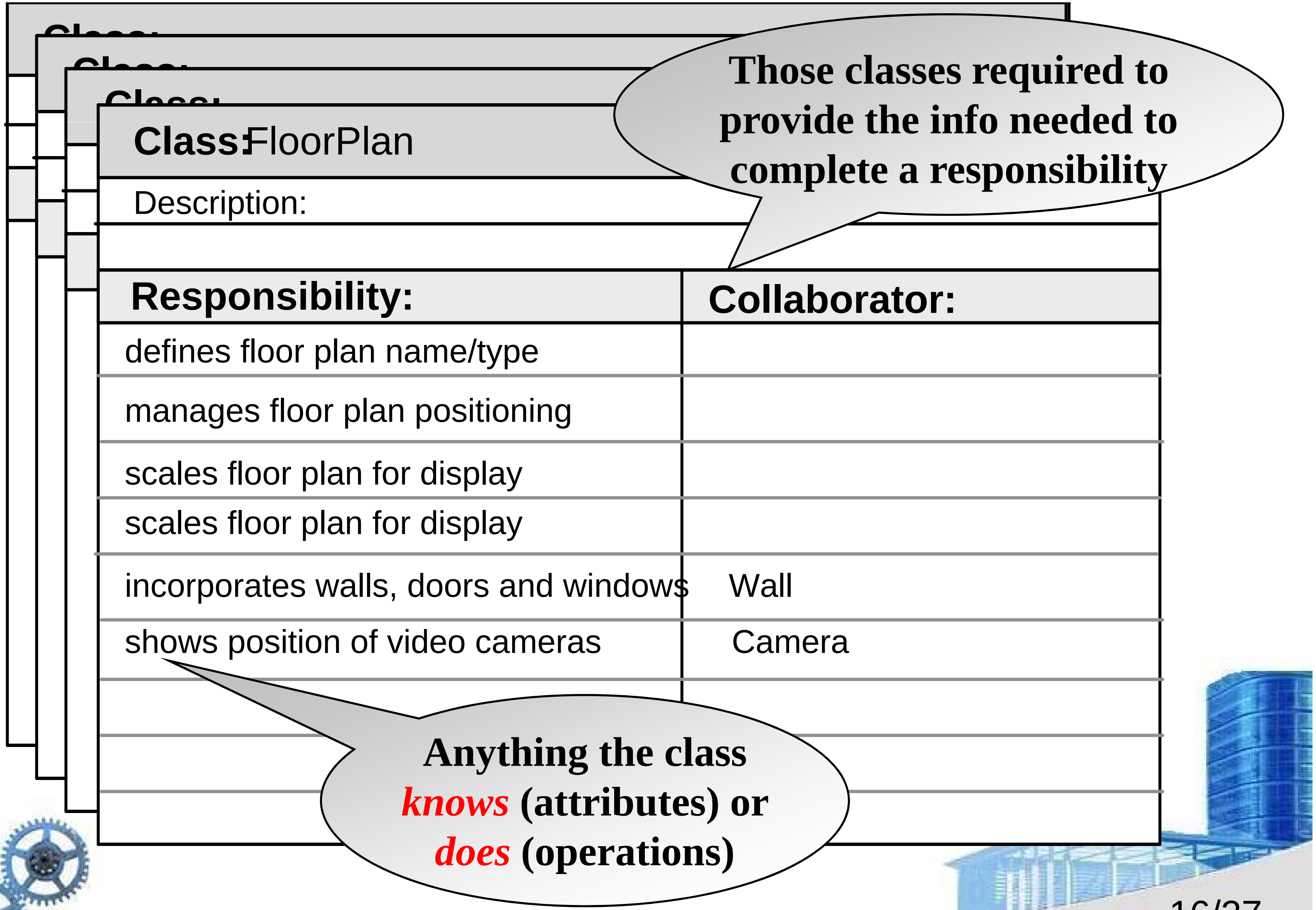
type
startCoordinates
stopCoordinates
nextDoor

determineType()
draw()

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: FloorPlan

Description:

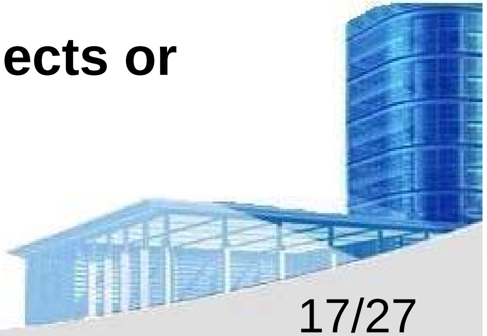
Responsibility:	**Collaborator:**
defines floor plan name/type	
manages floor plan positioning	
scales floor plan for display	
scales floor plan for display	
incorporates walls, doors and windows	Wall
shows position of video cameras	Camera

Those classes required to provide the info needed to complete a responsibility

Anything the class *knows* (attributes) or *does* (operations)

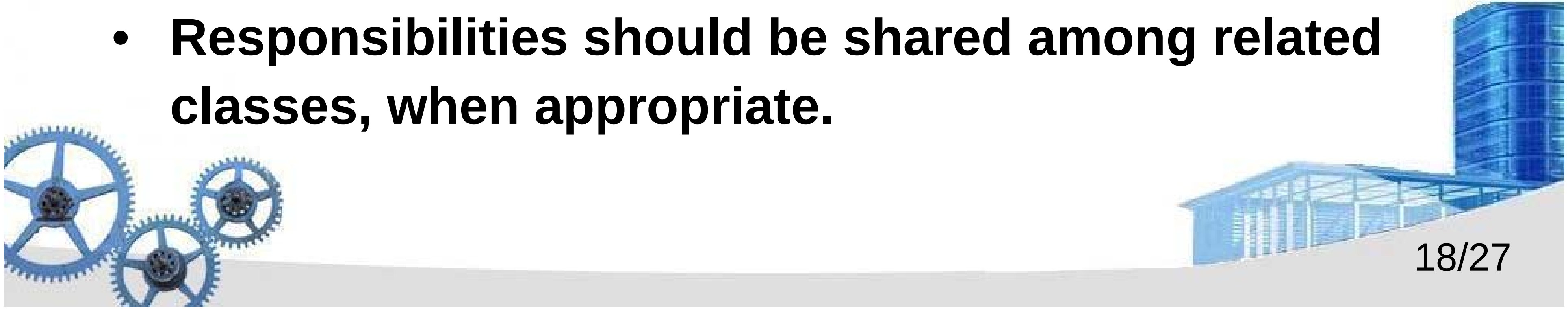
Class Types

- ***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.**



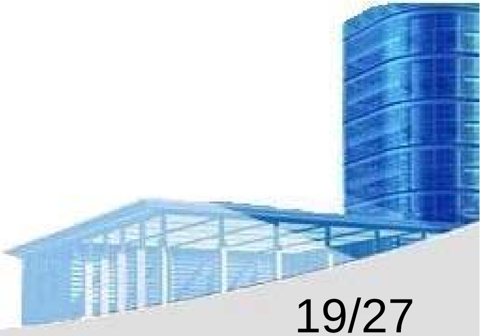
Guidelines for Allocating Responsibilities

- System intelligence should be distributed across classes to best address 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.

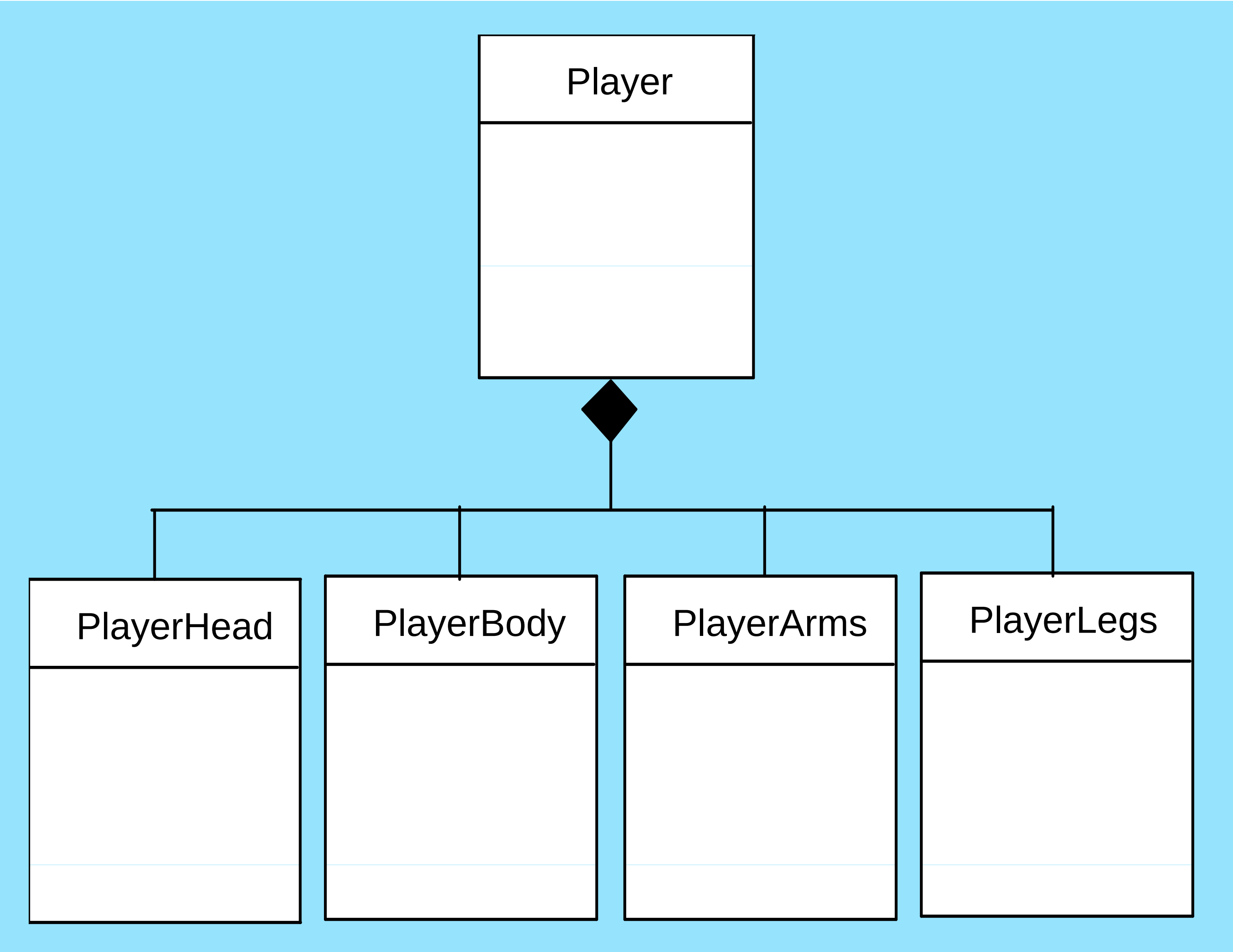


Collaborations

- 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* relationship
  - the *depends-upon* relationship



Composite Aggregate Class



Reviewing the CRC Model

- All participants in the review (of the CRC model) are given a subset 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.
  - As the review leader comes to a named object, she passes a token to the person holding the corresponding class index card.

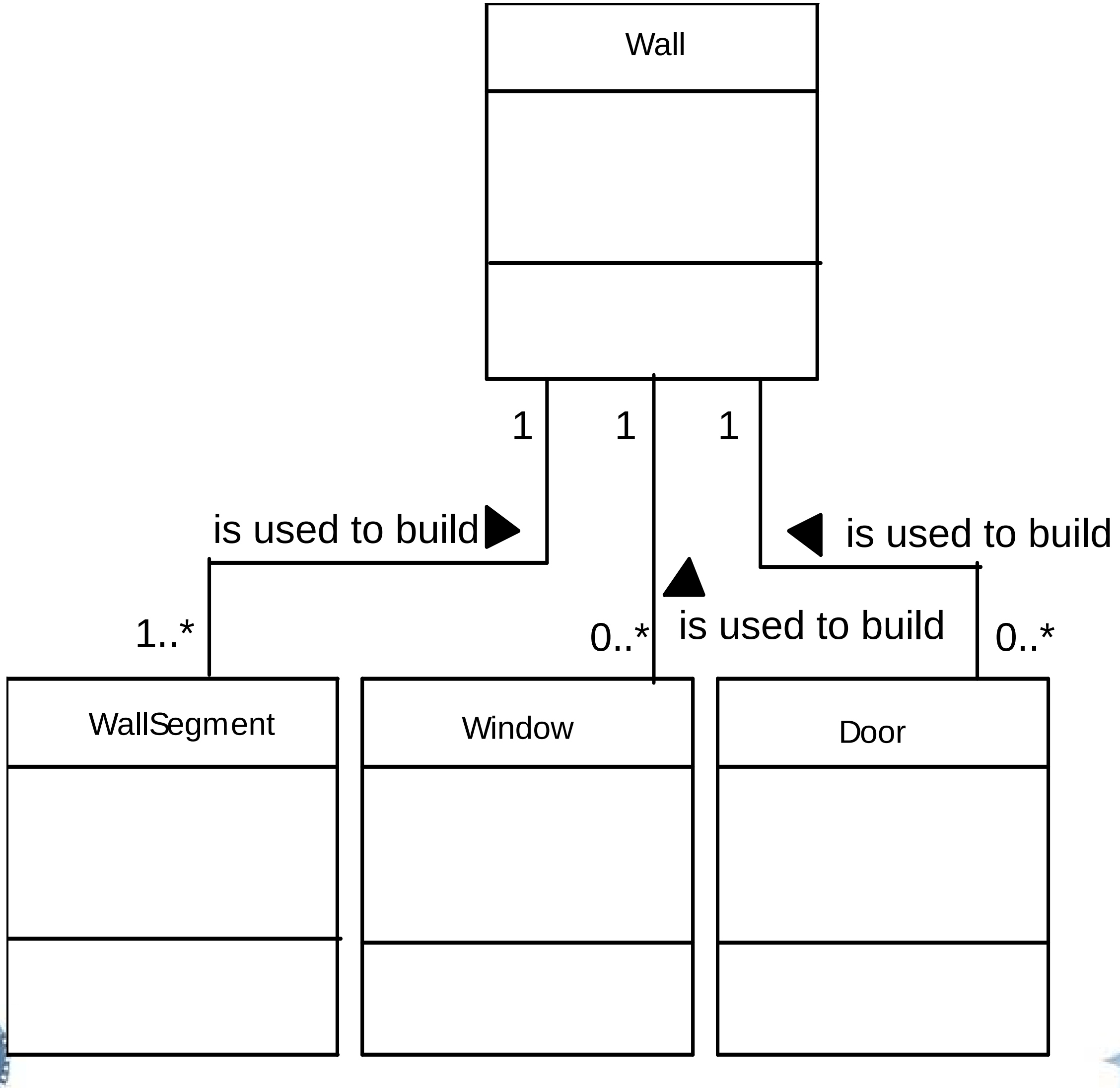
Reviewing the CRC Model (cont.)

- 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.

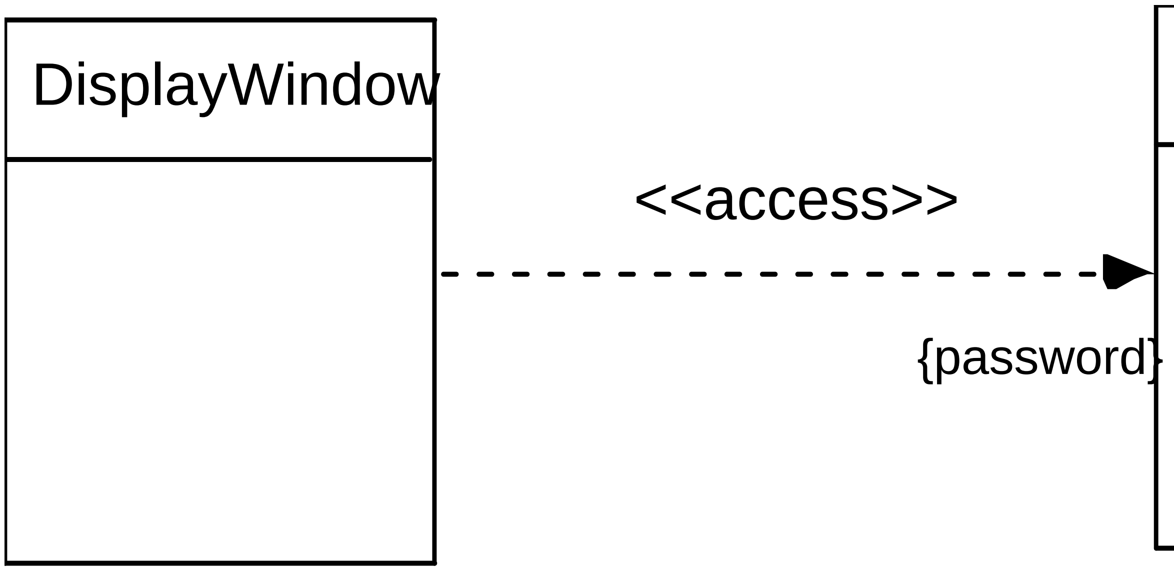
Associations and Dependencies

- 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

服务器类

Analysis Packages

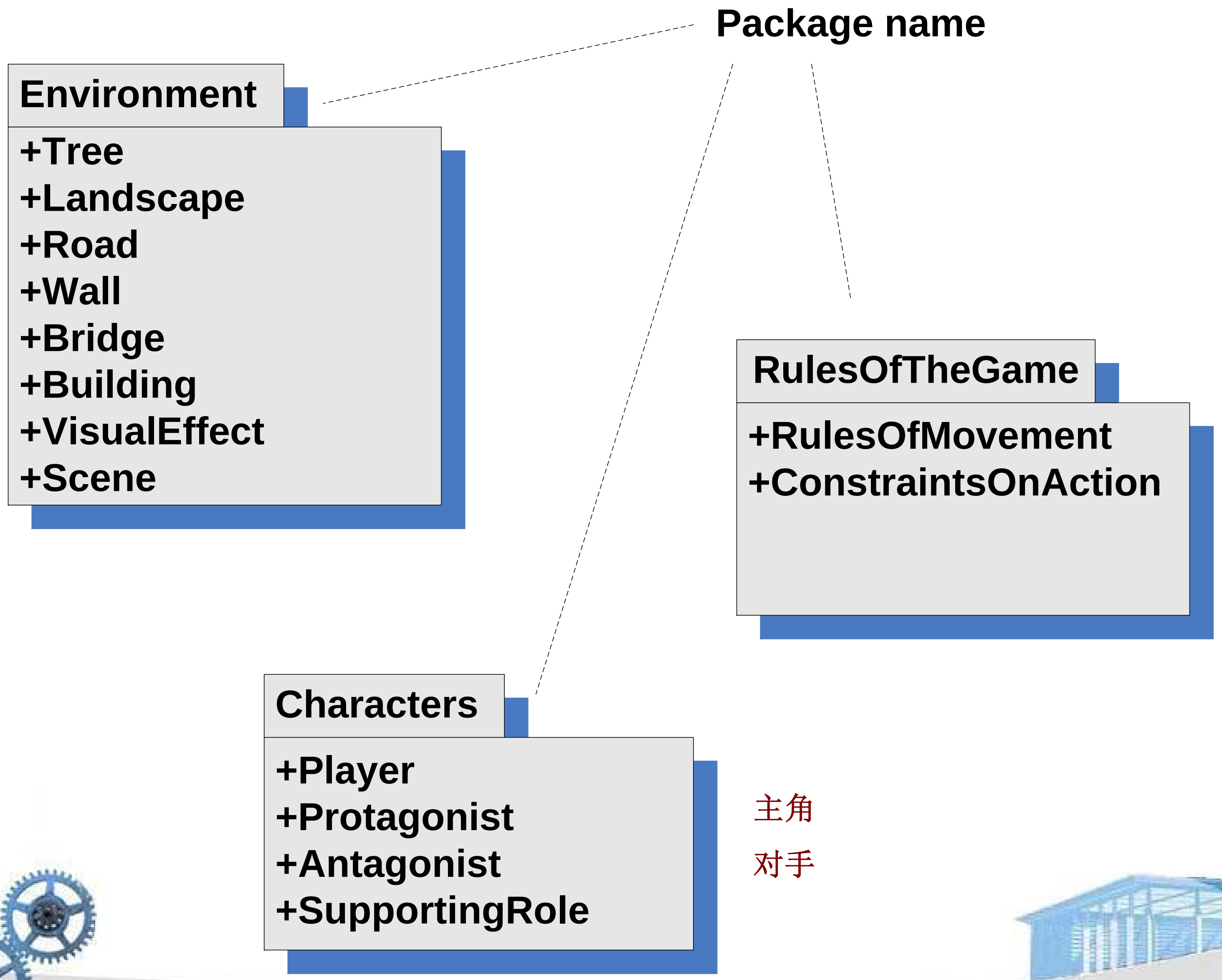
- Various elements of the analysis model (e.g., use-cases, analysis classes) are categorized in a manner that packages them as a grouping

- 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

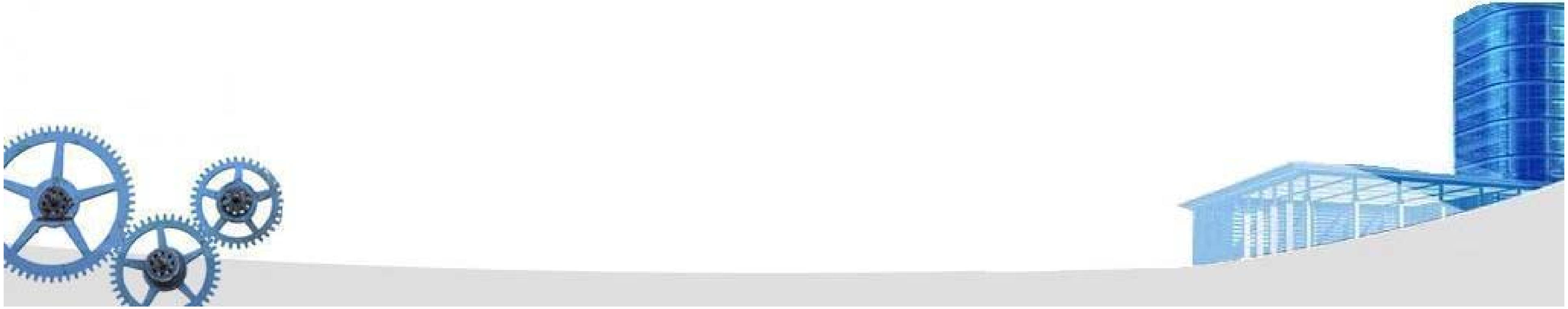
Characters

+Player
+Protagonist
+Antagonist
+SupportingRole

主角

对手

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



Behavioral Modeling

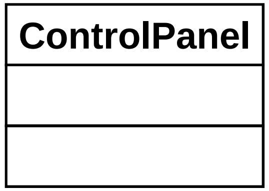
- The behavioral model indicates how software will respond to external events or stimuli. To create the model, the analyst must perform the following steps:
  1. Evaluate all **use-cases** to fully understand the sequence of interaction within the system.
  2. Identify **events** that drive the interaction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

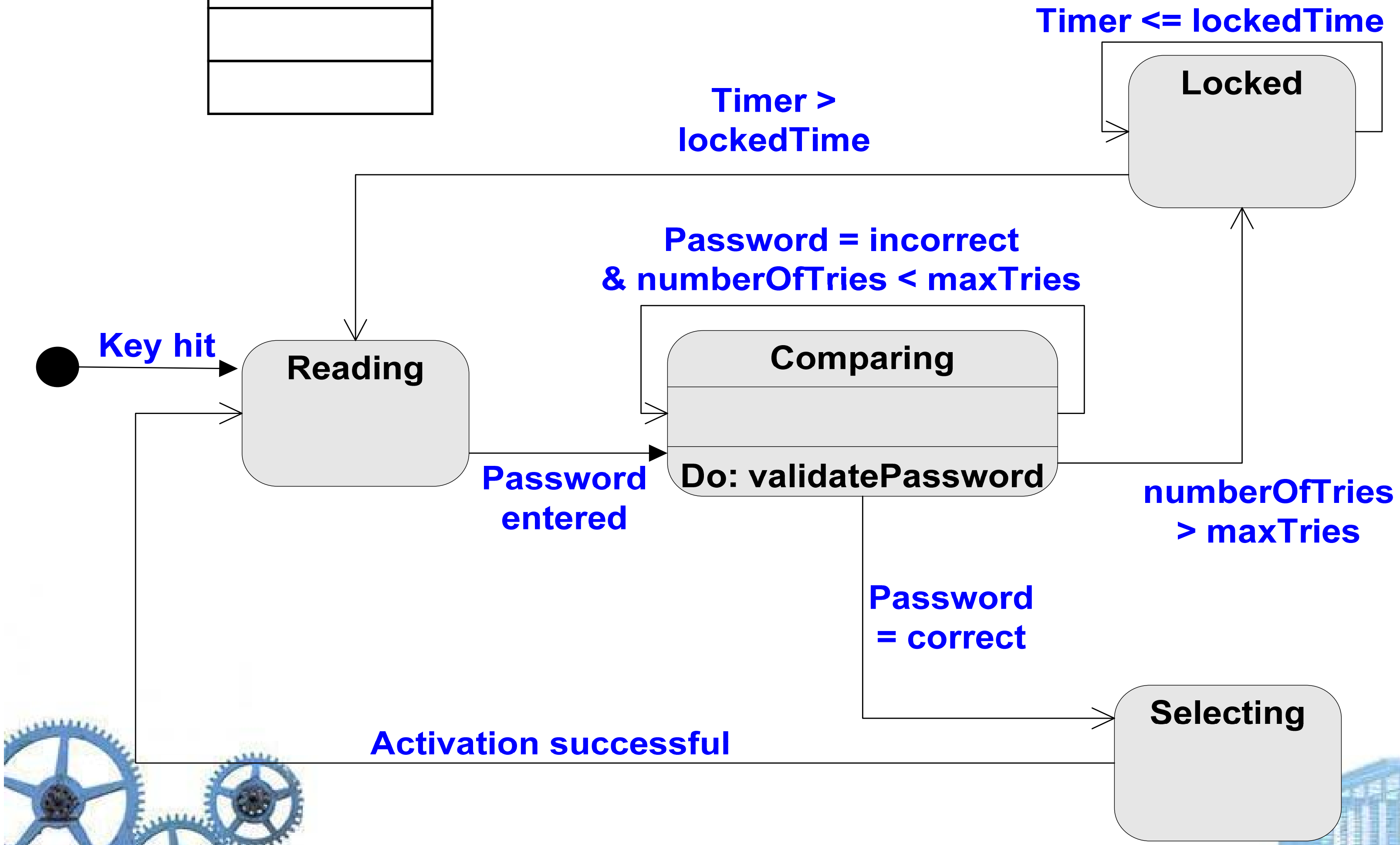
Behavioral Modeling

- 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 and
  - ***the state of the system*** as observed from the outside as the system performs its function

Behavioral Modeling

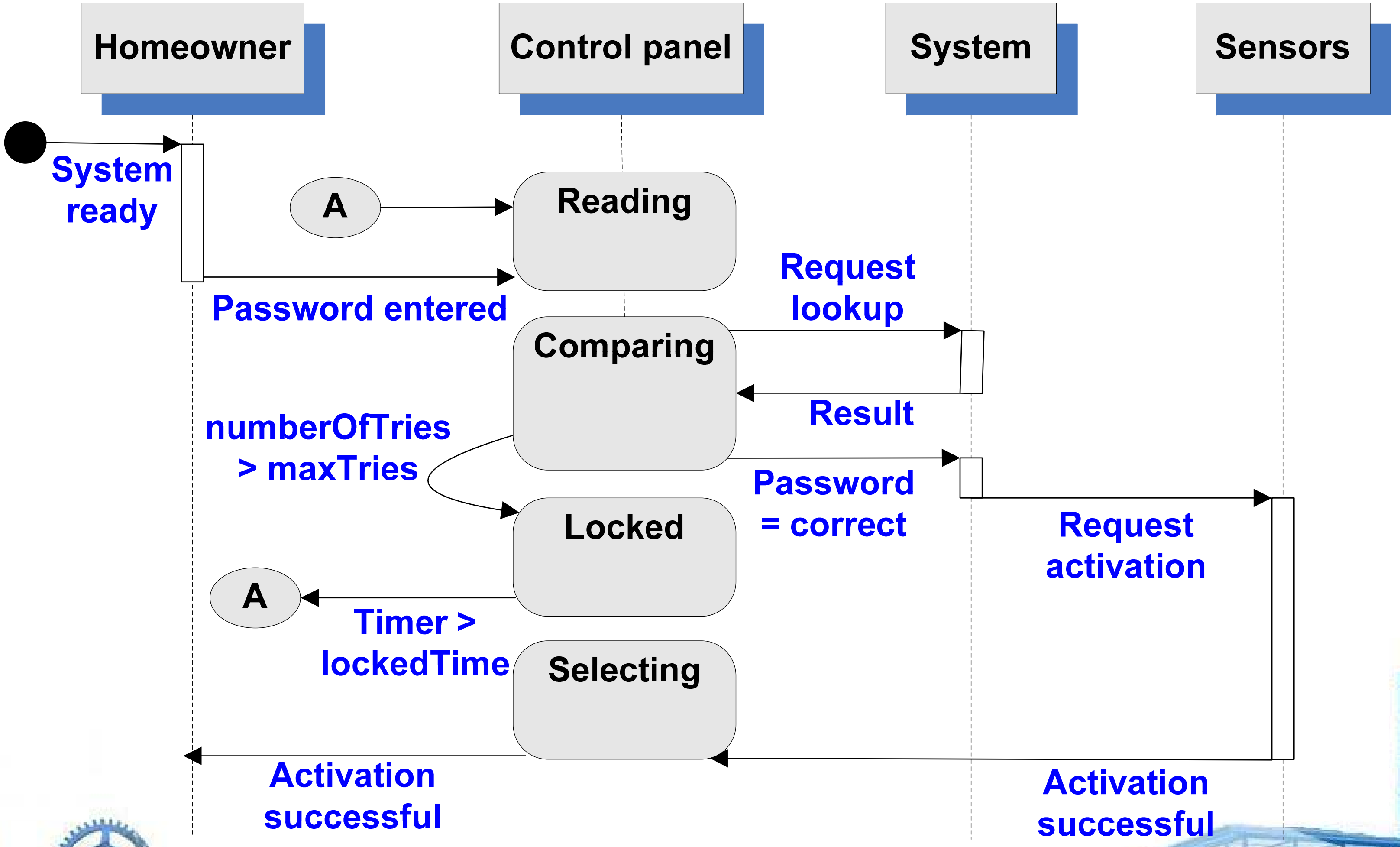
- State Diagram



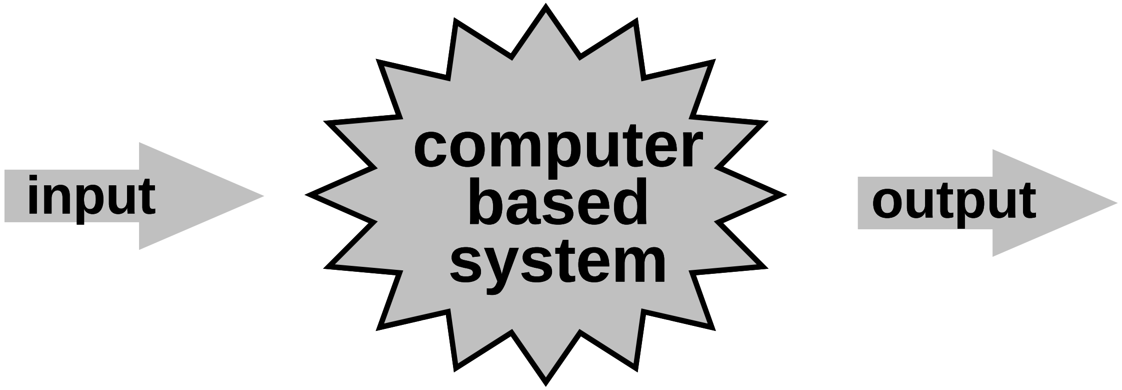


Behavioral Modeling

- **Sequence Diagram**

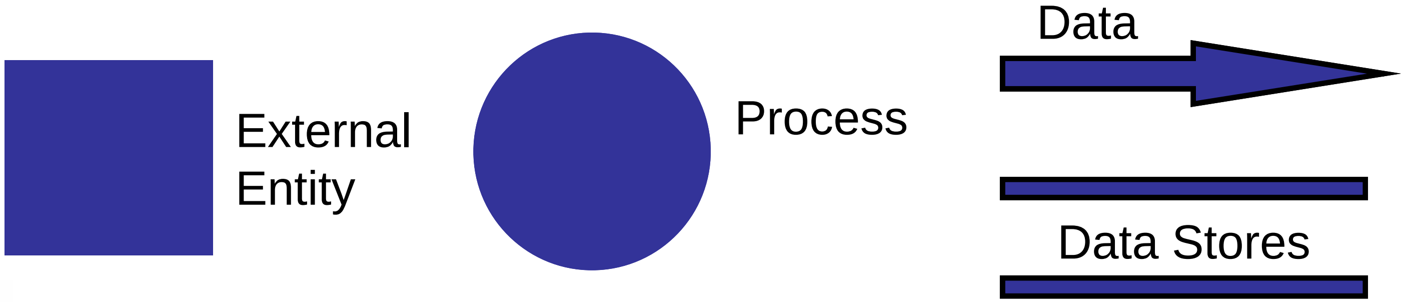


Flow-Oriented Modeling



System = data + function

- **Data Flow Diagram**



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** }

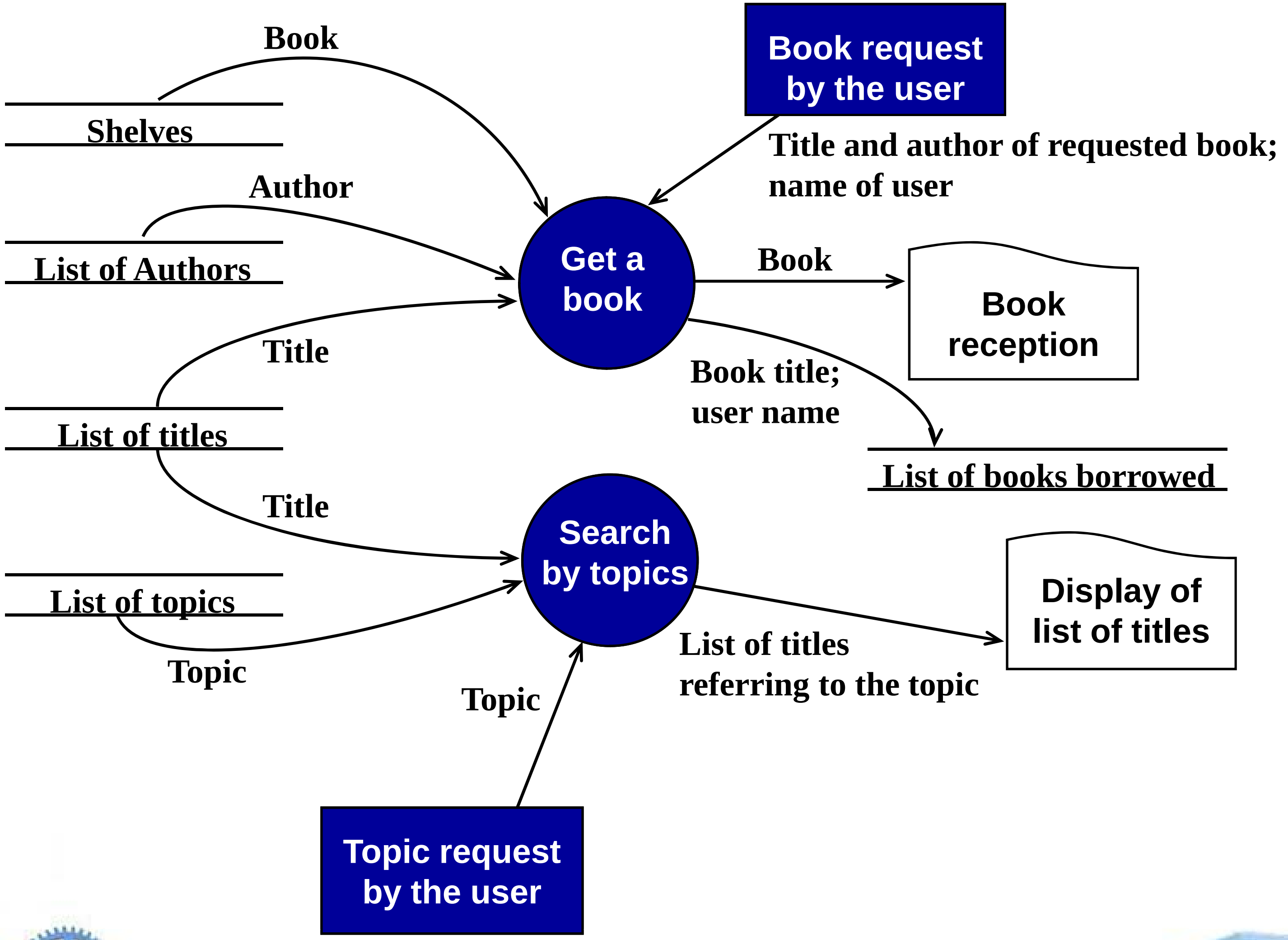
→ book, and user’s list of books borrowed;

if { user searches a book by topics }

{ **Search by topics** }

→ list of book titles referring to the topic.

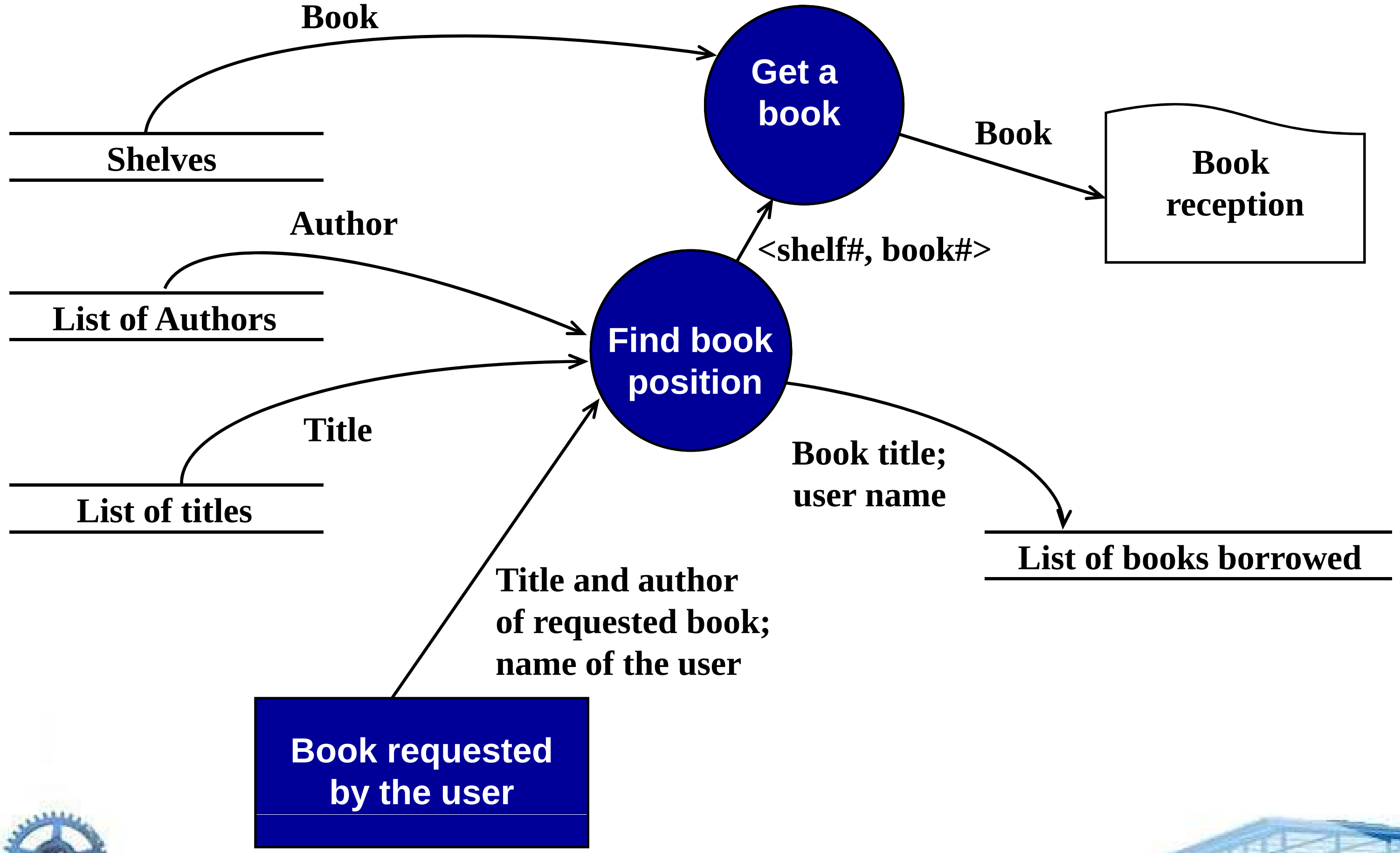
Flow-Oriented Modeling



Flow-Oriented Modeling

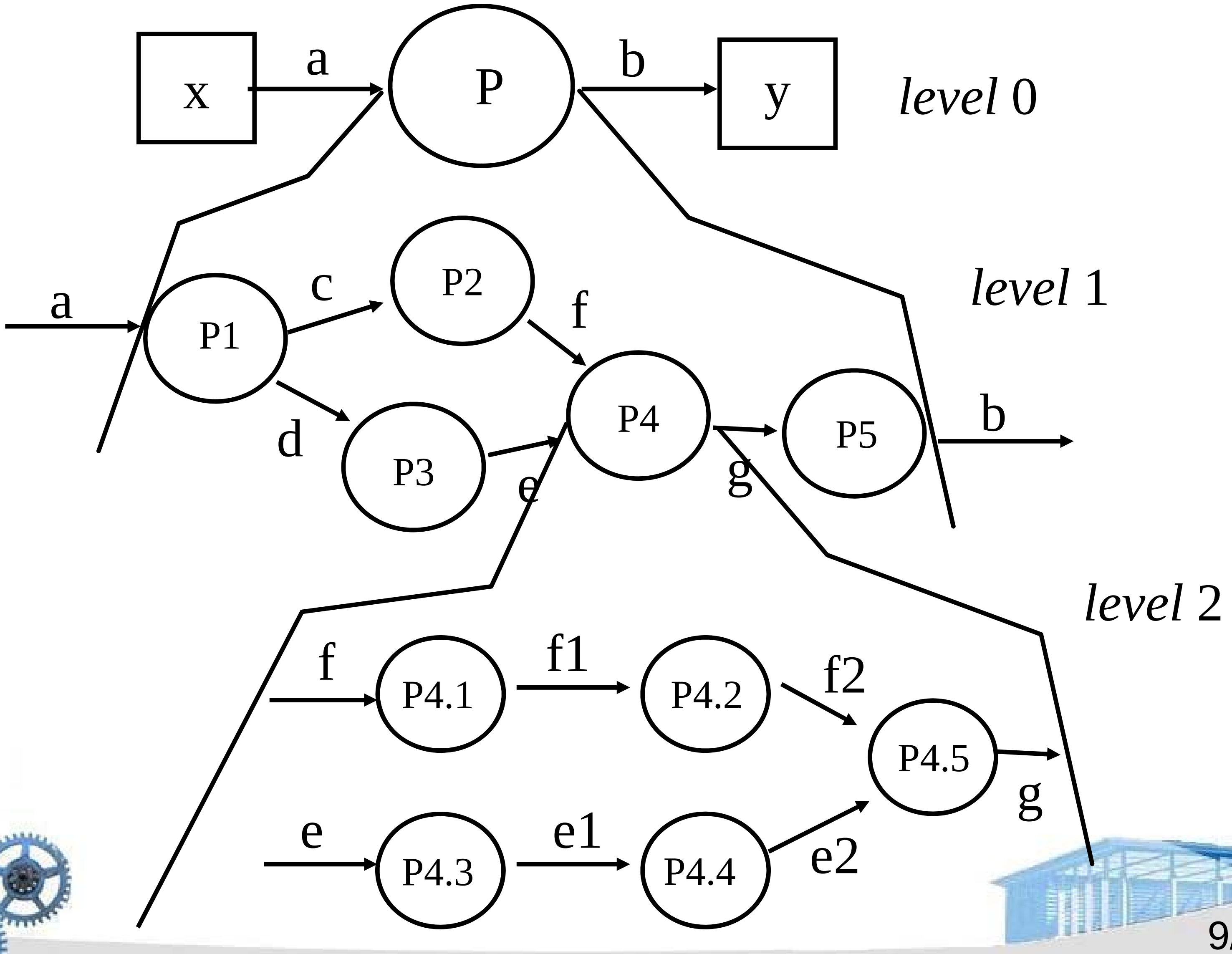
- **Refinement:**

Book request = Find book position + Get a book

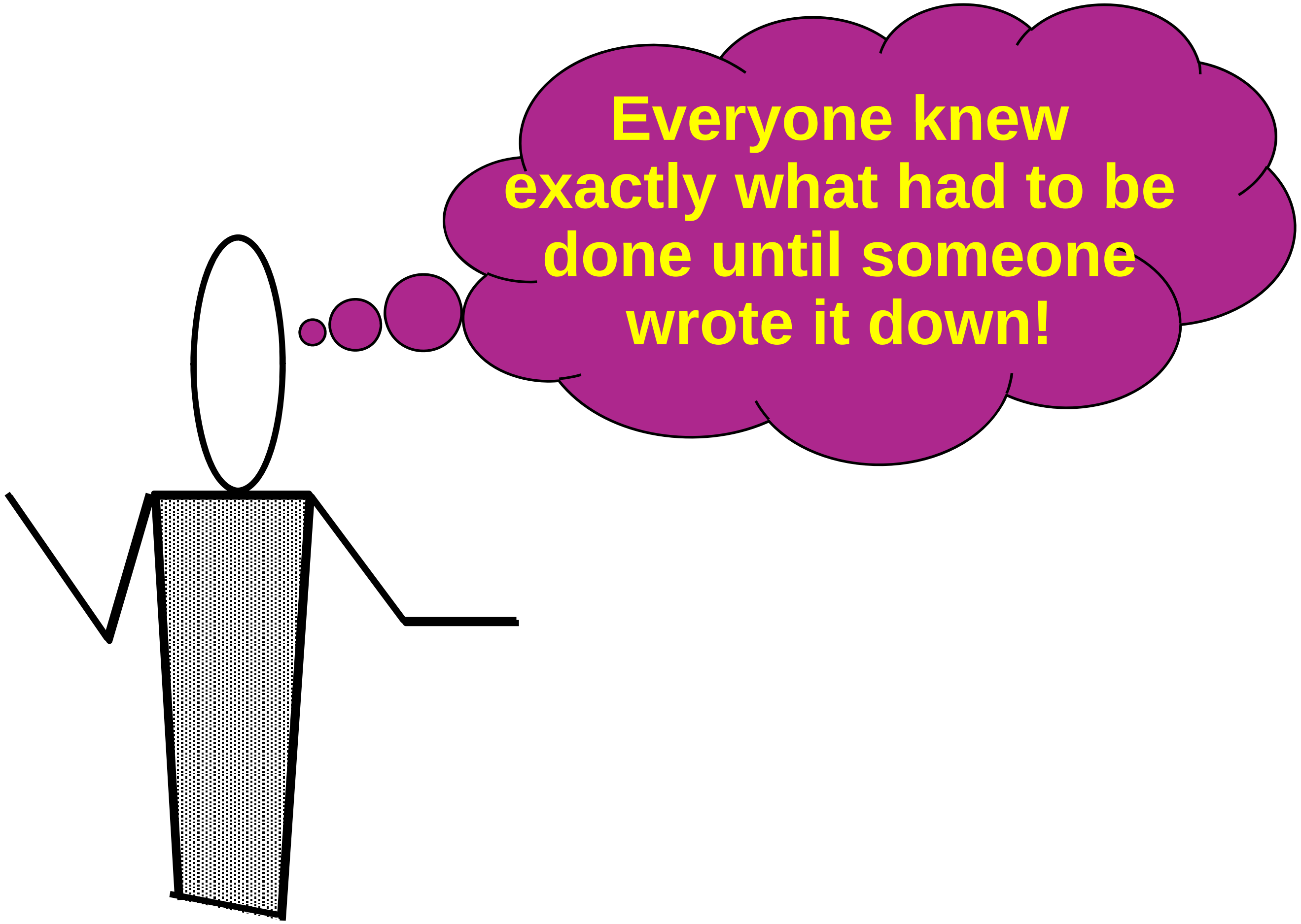


Flow-Oriented Modeling

- **The Data Flow Hierarchy**



Writing the Software Specification


Everyone knew exactly what had to be done until someone wrote it down!

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)
- ❑ 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?"

《Software Requirements Specification》

Due: 22:00 on April 27th, 2015

Minimum requirement of contents:

Introduction (2 points);
User Scenarios(8 points); Data Flow Diagram (7 points); State Diagrams(5 points); Class Diagrams(5 points) and CRC Cards (5 points);
Validation Criteria (15 points).

Concerned points:

The accuracy of the validation criteria: full marks can be obtained if more than 90% of the functions are covered. The acceptance testing of the subsystem version 1.0 will strictly go by the criteria. The language and style of the document must be uniformed (3 points).

Grading: The full mark = **50 points × number of participants**

Requirements Modeling for WebApps

- Why do we perform analysis?
  - the Web or Mobile App to be built is ***large*** and/or ***complex***
  - the number of ***stakeholders*** is large
  - 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

中国新女首富周群飞

Requirements Modeling for WebApps

- **Content Analysis** – describe
  - *text*
  - *graphics and images*
  - *video*
  - *audio*
- **Interaction Analysis** – use-cases
- **Functional Analysis** – use-cases that define
  - the operations that will be applied to WebApp content
  - imply other processing functions
- **Configuration Analysis** – environment and infrastructure
- **Navigation Analysis** – focus on overall requirements

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-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(加快进展) 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.10-11
- **Finish** “Problems and points to ponder” in Ch. 10-11
- **Preview** Ch 12, 19
- Please attend the course next Sunday morning, **8:30, Room 7-504，April 26!**
- **Submit Requirement Gathering Report due April 27 !**