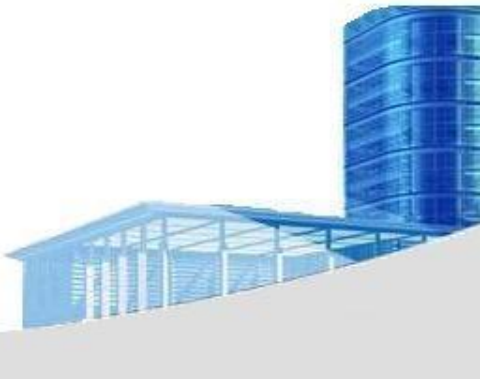
Ch.10 Requirements Modeling: Class-Based Methods

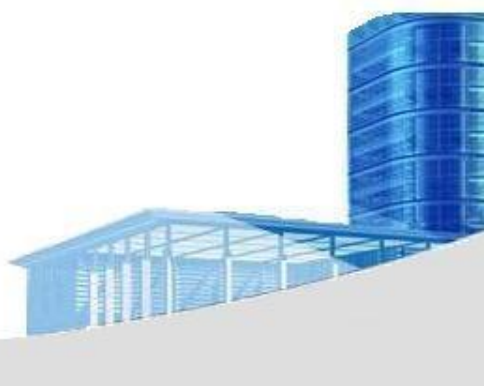


Requirements Modeling Strategies

- 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 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 operat
  - Encapsulation and instan
  - 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

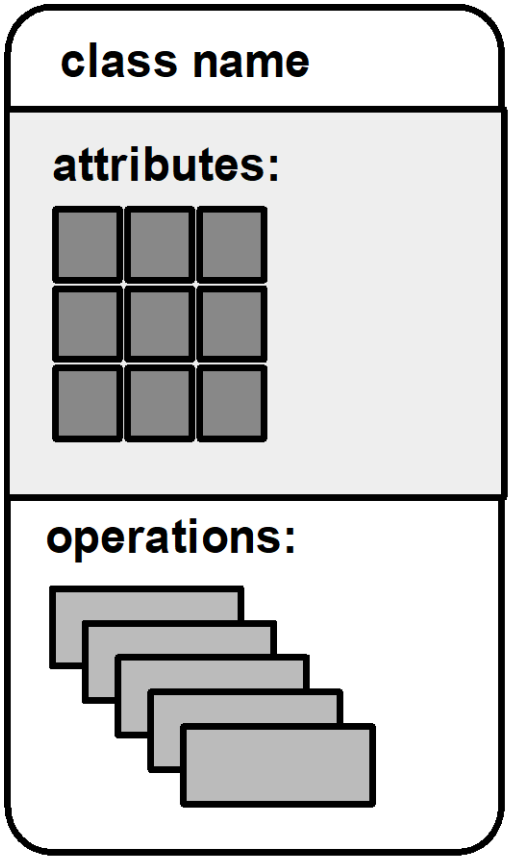


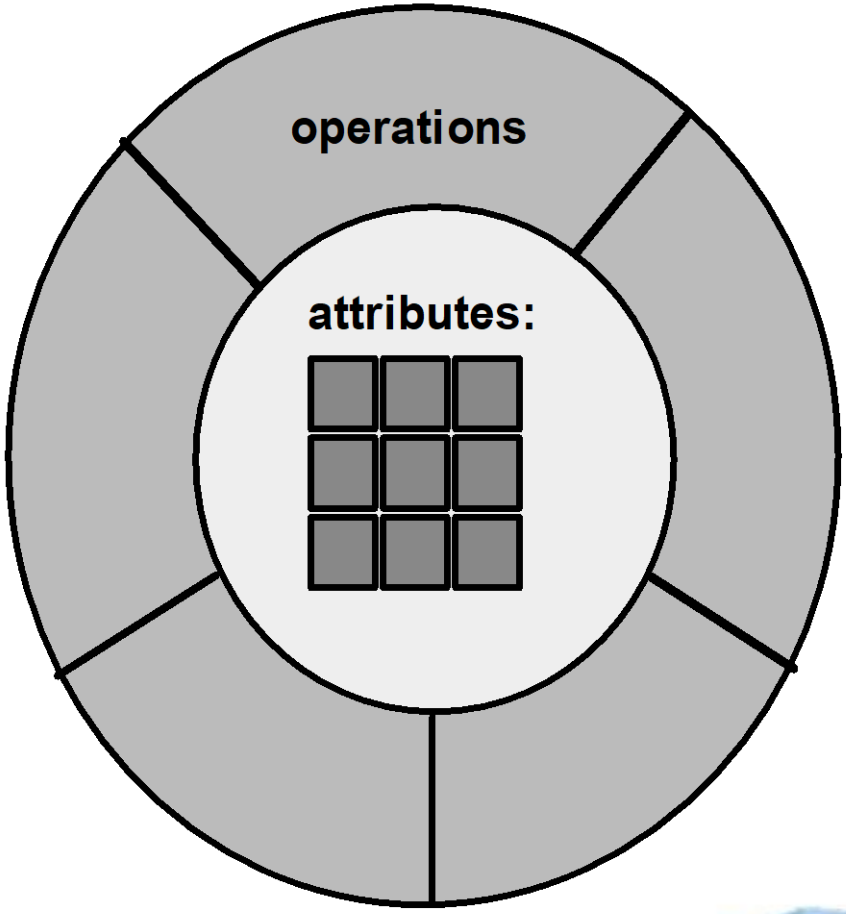
Why encapsulation?

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

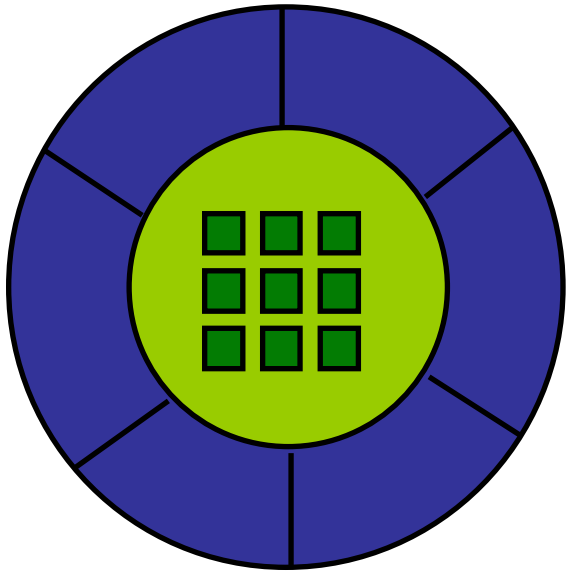
Building a Class



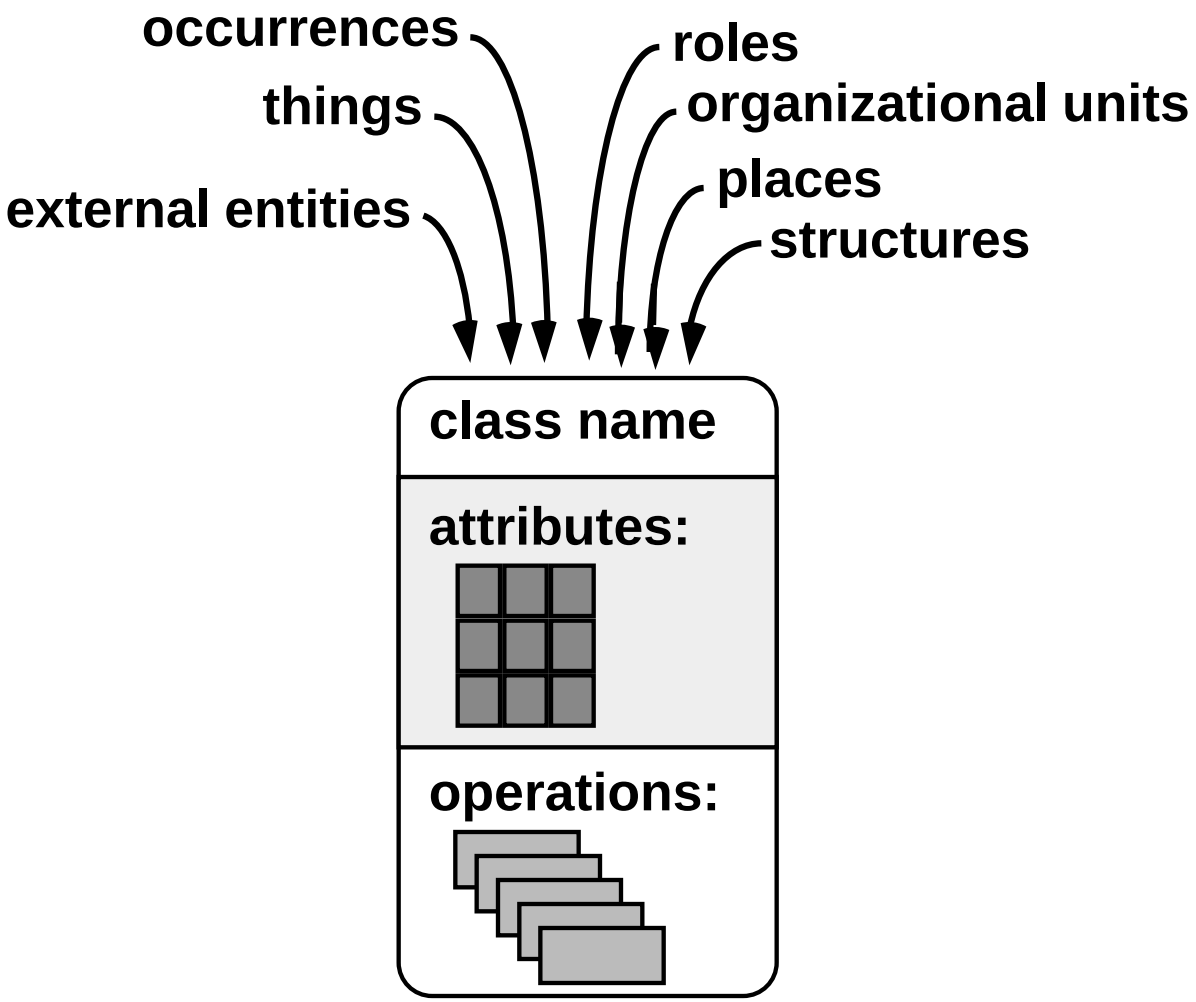


Methods

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.

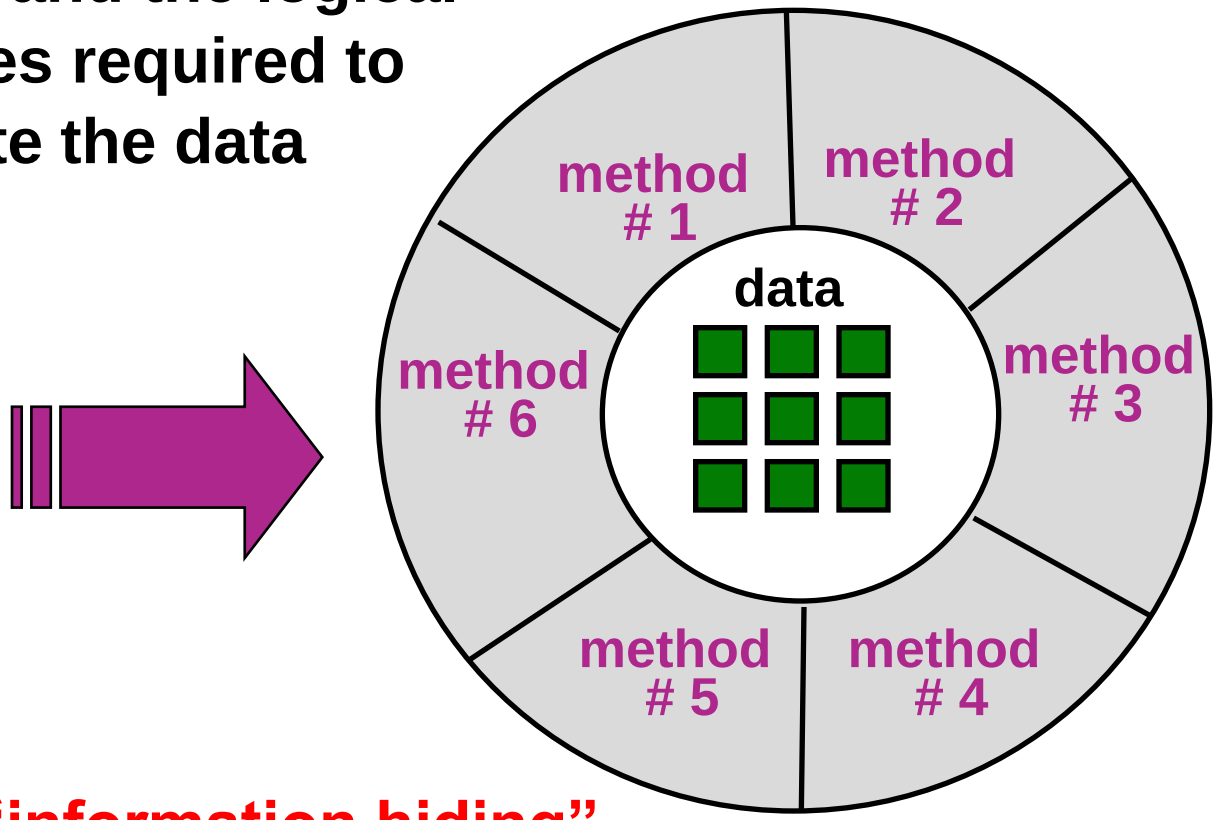


What is a Class?



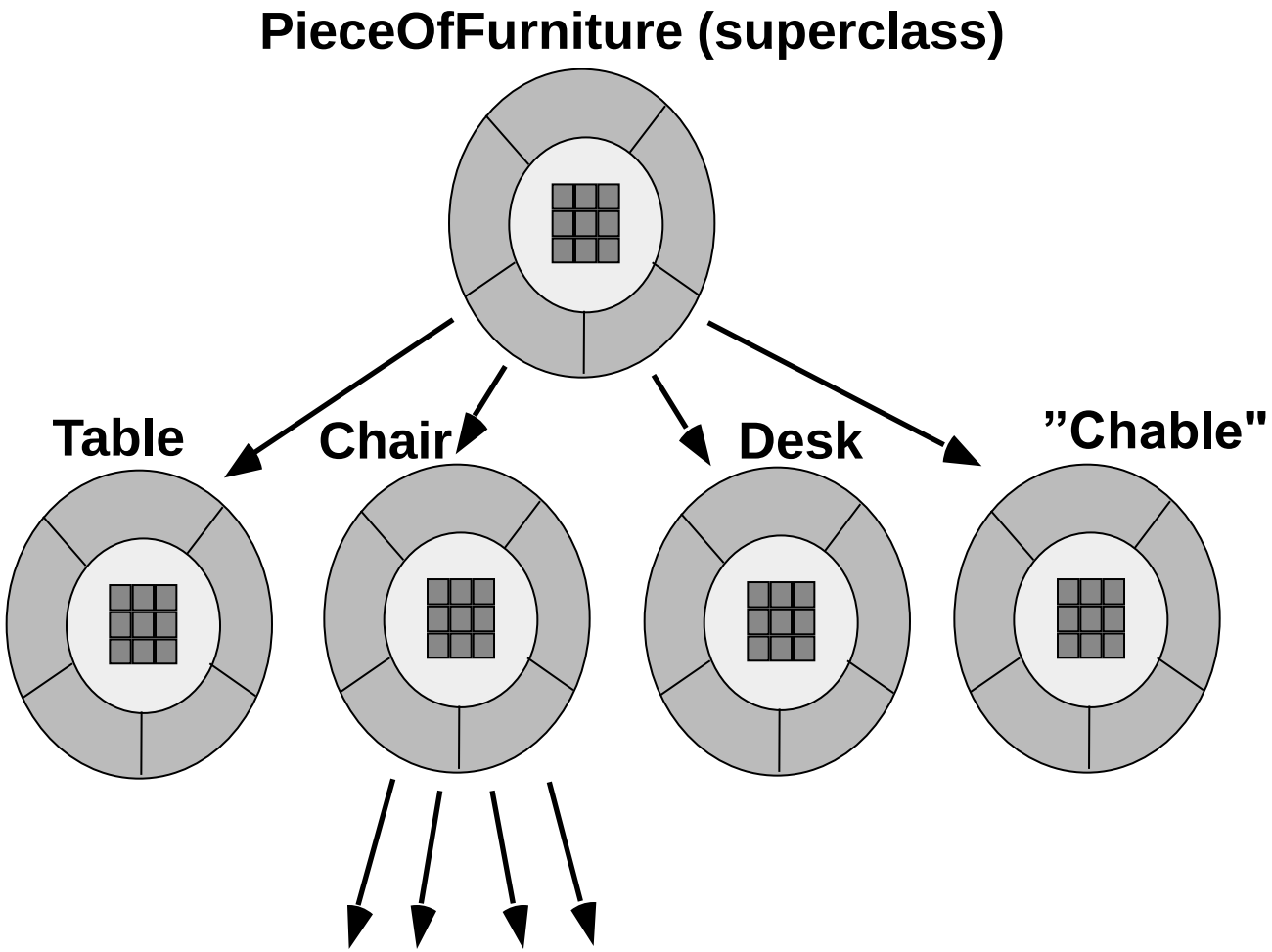
Encapsulation/Hiding

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.**

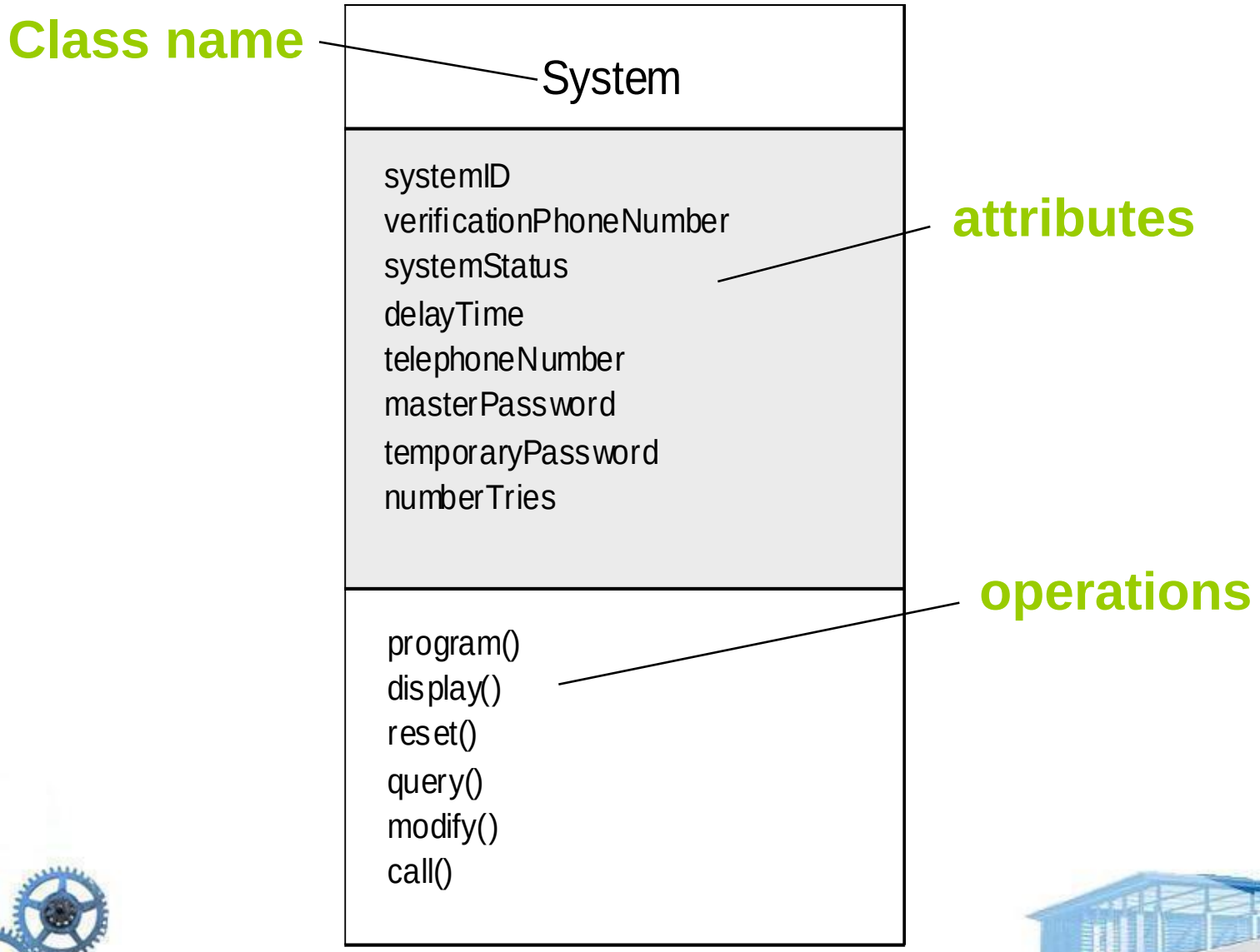
Class-Based Modeling

- 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

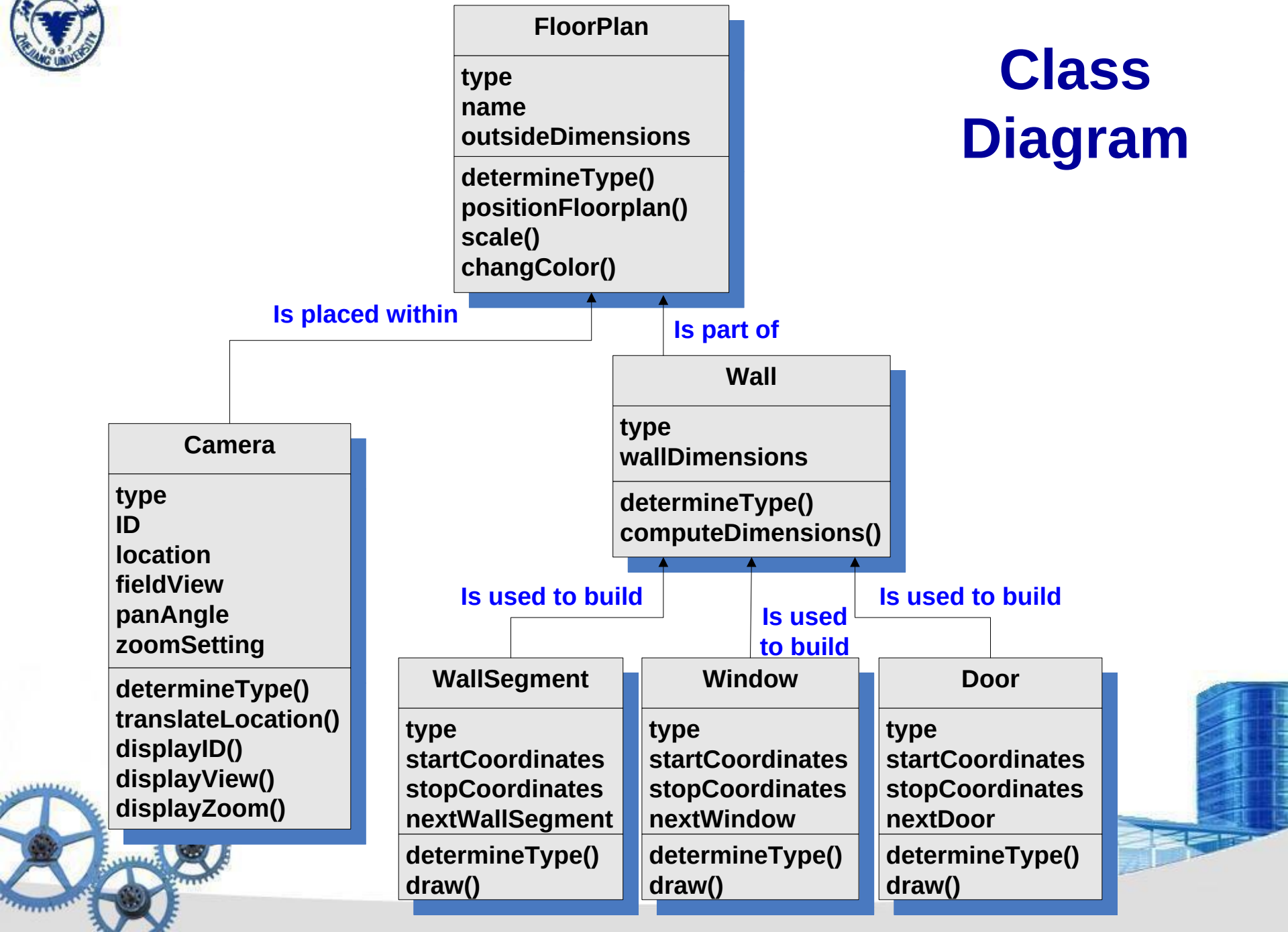
Potential Classes

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

Class Diagram



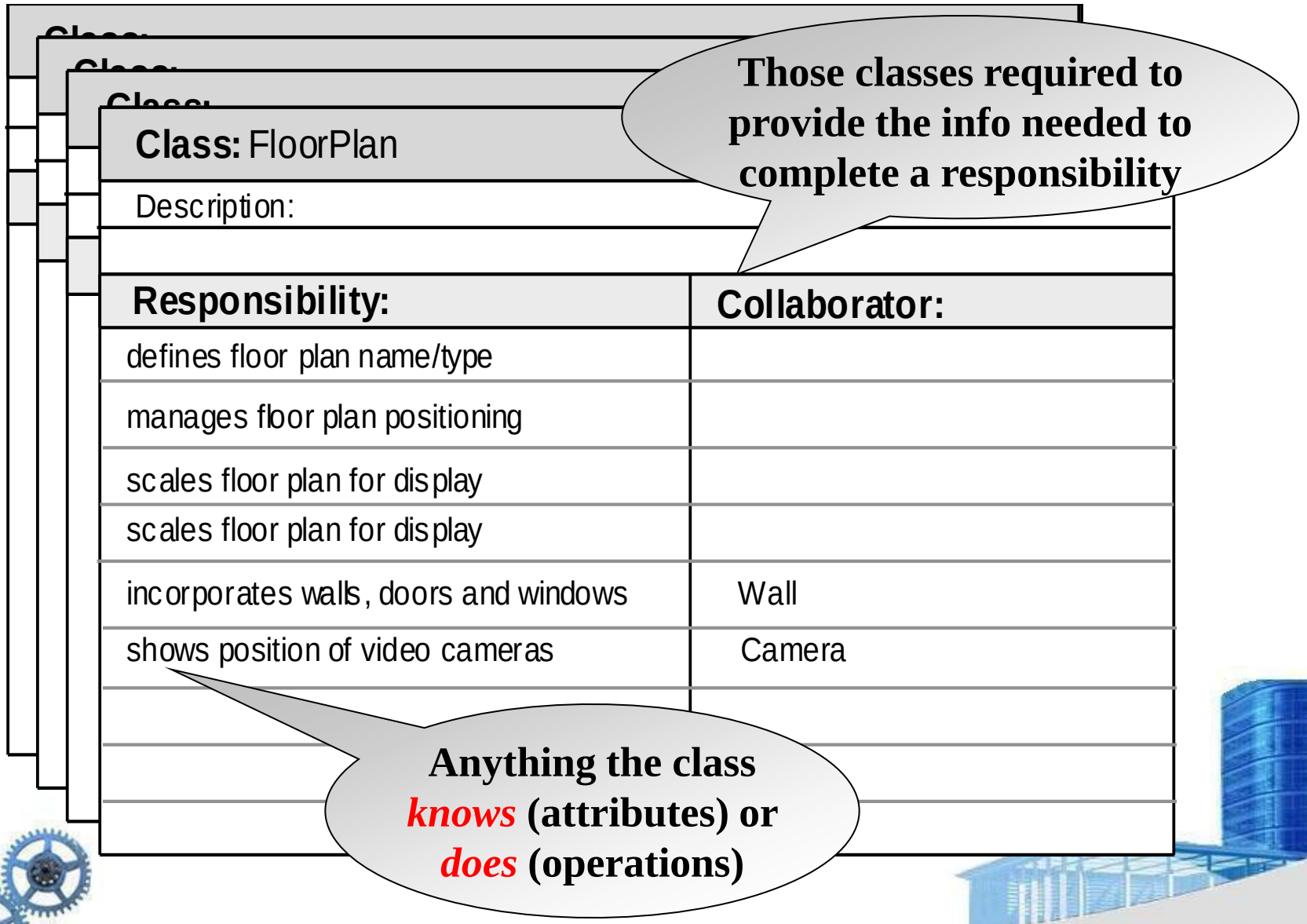
Class Diagram



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.**

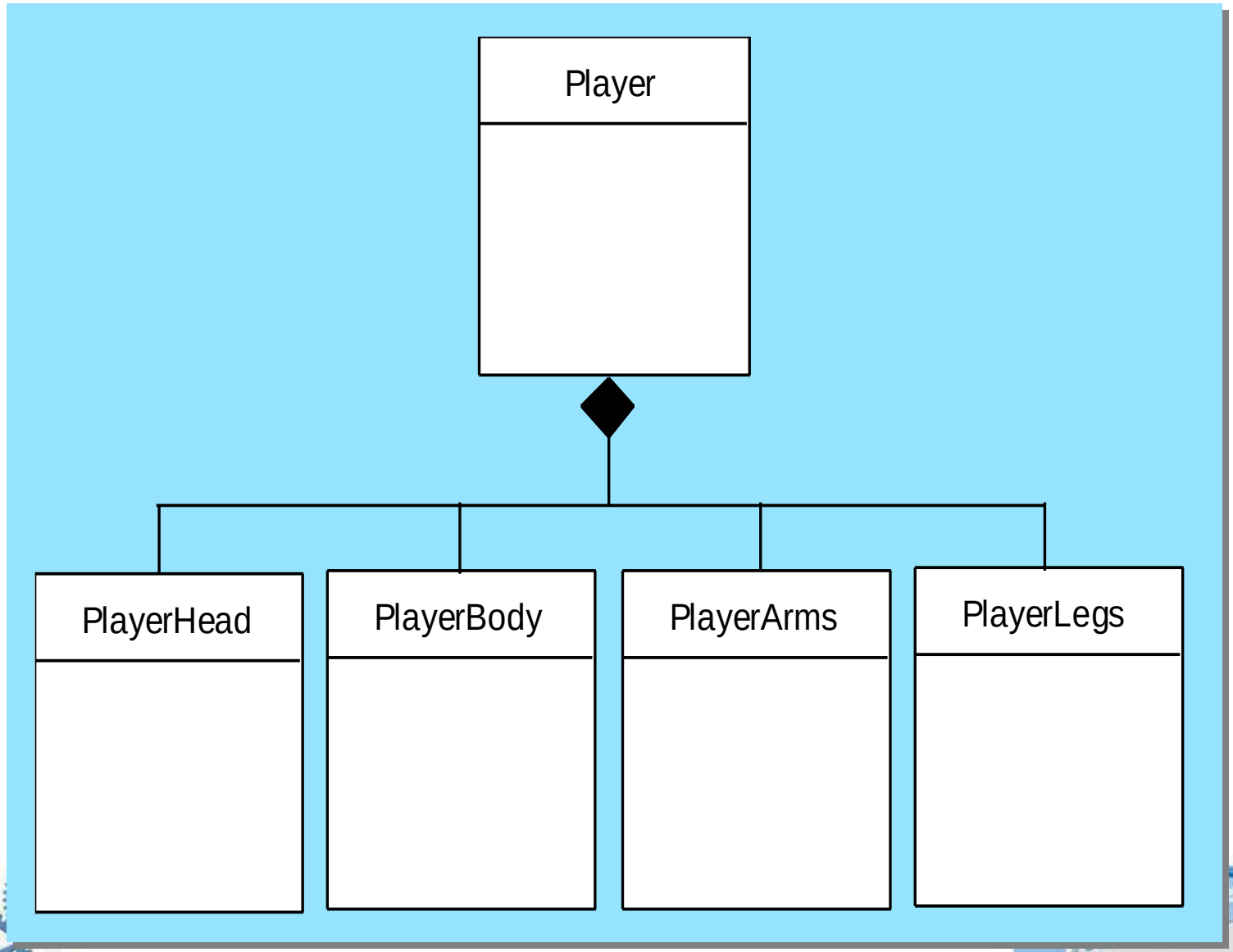
Guidelines for Allocating Responsibilities

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

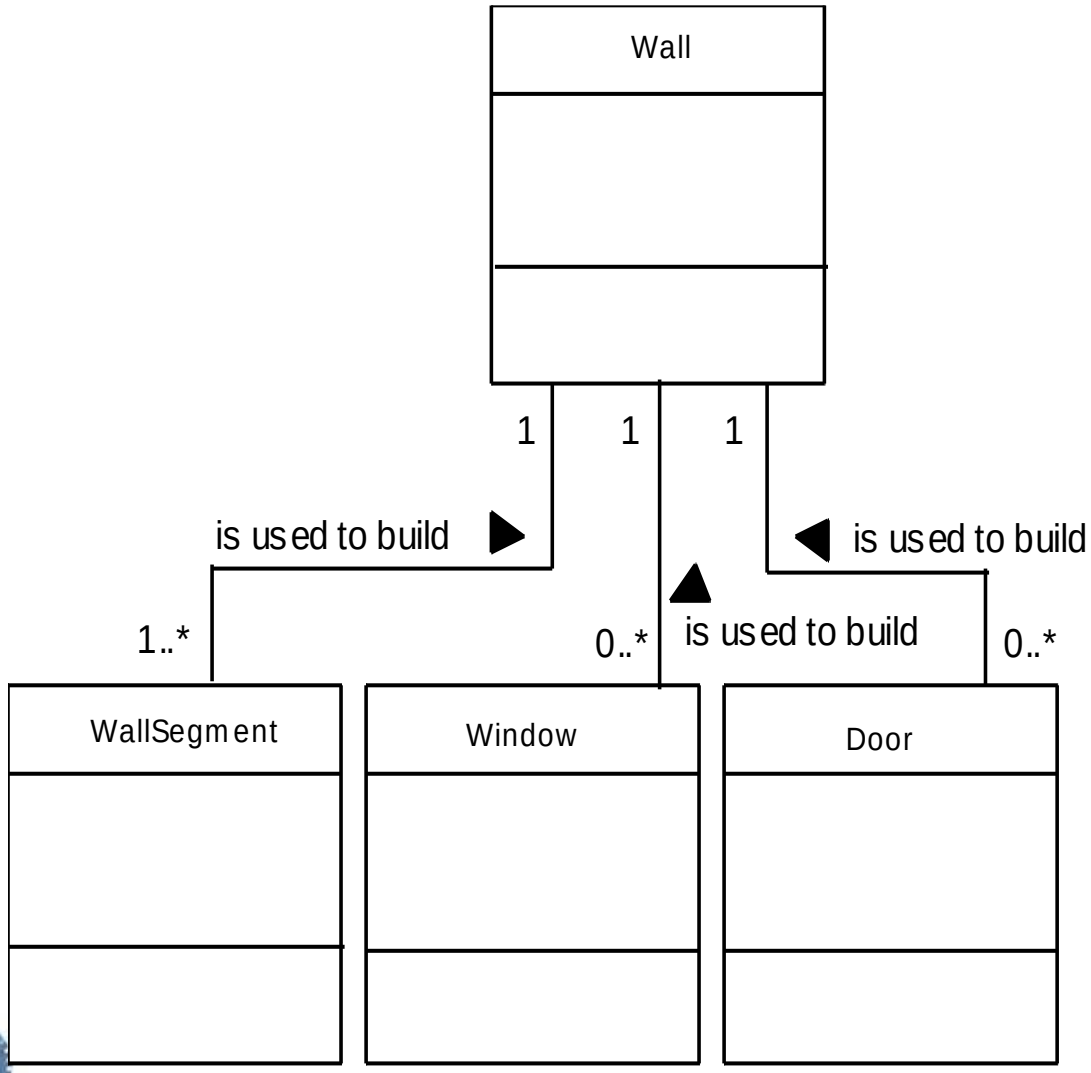
Reviewing the CRC Model (cont.)

- 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.

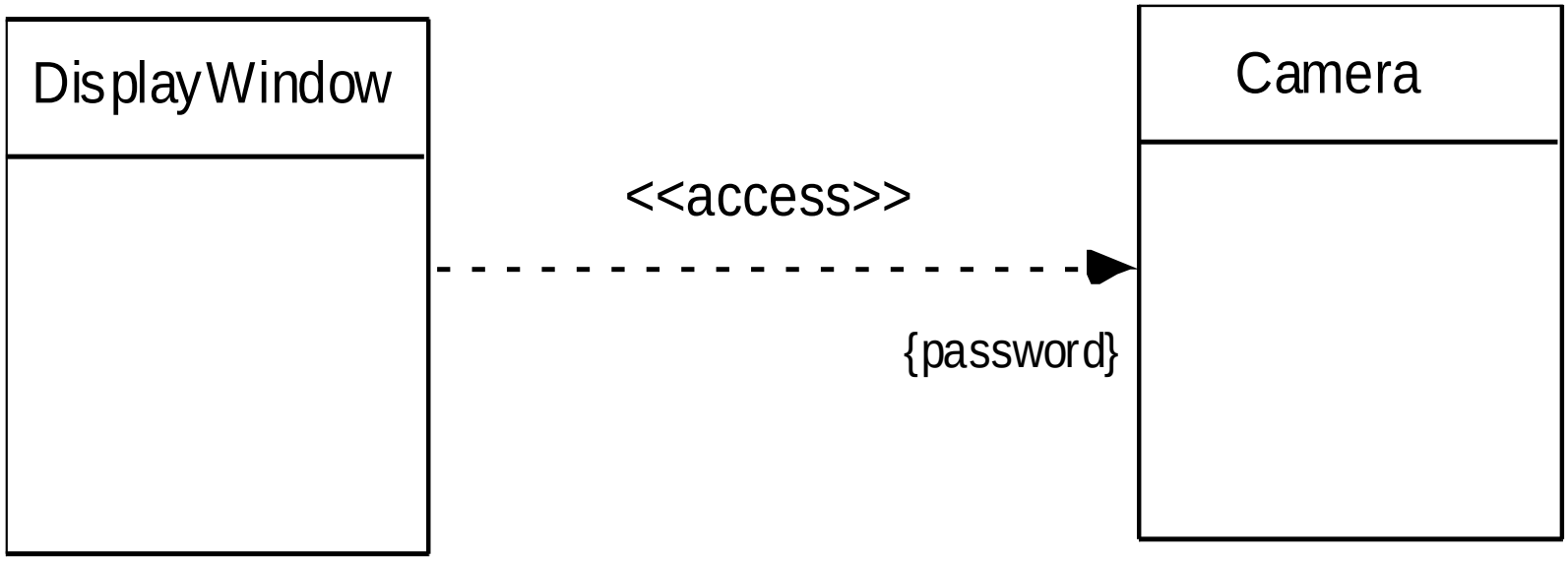
Associations and Dependencies

- 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 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



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 and a # symbol indicates that an element is accessible only to classes contained within a given package.

Analysis Packages