



# REQUIREMENTS MODELING

## — CLASS-BASED METHODS

### (LESSON 05)

PHÂN TÍCH YÊU CẦU PHẦN MỀM (SOFTWARE REQUIREMENTS)



# CONTENTS

- IDENTIFYING ANALYSIS CLASSES
- SPECIFYING ATTRIBUTES
- DEFINING OPERATIONS
- CLASS-RESPONSIBILITY-COLLABORATOR MODELING
- ASSOCIATIONS AND DEPENDENCIES
- ANALYSIS PACKAGES



# CONCEPTS

- **Class-based modeling** represents the 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, and the **collaborations** that occur between the classes that are defined.
- Elements of a class-based model:
  - classes and objects
  - attributes
  - operations
  - class-responsibility-collaborator (CRC) models
  - collaboration diagrams, and
  - packages.

# CONTEXTS

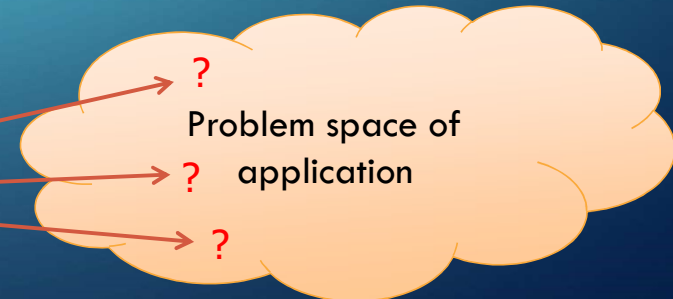
- **What is it?** Software problems can almost always be characterized in terms of a set of interacting objects each representing something of interest within a system. Each object becomes a member of a class of objects. Each object is described by **its state** – the data attributes that describe the object. All of this can be represented using class-based requirements modeling methods.
- **Why is it important?** A class-based requirements model makes use of objects drawn from the customer's view of an application or system. **The model depicts a view of the system that is common to the customer.** Therefore, it can be readily evaluated by the customer, resulting in useful feedback at the earliest possible time. **Later, as the model is refined, it becomes the basis for software design.**

# IDENTIFYING ANALYSIS CLASSES

- If you look **around a room**, there is a set of **physical objects** that can be **easily** identified, classified, and defined (in terms of attributes and operations).



- But when you **"look around" the problem space of a software application**, the classes (and objects) may be **more difficult to comprehend**.



# IDENTIFYING ANALYSIS CLASSES

- Identify classes by examining the usage scenarios developed as part of the requirements model and performing a **“grammatical parse”** [Abb83] on the use cases developed for the system to be built.
  - Classes are determined by **underlining each noun or noun phrase** and entering it into a simple table.
  - **Synonyms should be noted.** If the class (noun) is required to implement a solution, then it is part of **the solution space**;
  - otherwise, if a class is **necessary only to describe a solution**, it is part of the **problem space**.

# CATEGORIZATION OF ANALYSIS CLASSES

- Analysis classes manifest themselves in one of the following ways:
  - **External entities** (e.g., other systems, devices, people) that **produce or consume information** to be used by a computer-based system.
  - **Things** (e.g., reports, displays, letters, signals) that are **part of the information domain** for the problem.
  - **Occurrences or events** (e.g., a property transfer or the completion of a series of robot movements) that **occur within the context of system operation**.
  - **Roles** (e.g., manager, engineer, salesperson) **played by people who interact** with the system.
  - **Organizational units** (e.g., division, group, team) that are relevant to an application.
  - **Places** (e.g., manufacturing floor or loading dock) that establish the **context of the problem** and the overall function of the system.
  - **Structures** (e.g., sensors, four-wheeled vehicles, or computers) that **define a class of objects or related classes of objects**.

**BUT**, this categorization is one of many that have been proposed in the literature. For example, Budd [Bud96] suggests a taxonomy of classes that includes producers (sources) and consumers (sinks) of data, data managers, view or observer classes, and helper classes.

# REMARKS

REMARKS: It is also important to note **what classes or objects ARE NOT**.

A class should never have an “imperative procedural name” [Cas89].

Example:

If the developers of software for a medical imaging system defined an object with the name **InvertImage** or even **ImageInversion**, they would be making a **SUBTLE MISTAKE**.



The **Image** obtained from the software **could be a class (of course)** (it is a thing that is part of the information domain).

**Inversion of the image** is an operation that is applied to the object. It is likely that **inversion would be defined as an operation for the object Image**, but it **WOULD NOT BE** defined as a separate class to connote “image inversion.”

As Cashman [Cas89] states, “The intent of object-orientation is to encapsulate, but still keep separate, data and operations on the data.”

# PROJECT: SAFEHOME

To illustrate how **analysis classes** might be defined during the early stages of modeling, consider a grammatical parse (nouns are underlined, verbs italicized) for a processing narrative for the SafeHome security function.

Extracting the nouns, we can propose a number of potential classes:

Potential Class	General Classification
Homeowner	Role or external entity
Sensor	External entity
control panel	External entity
Installation	Occurrence
system (alias security system)	Thing
number, type	<b>not objects</b> , attributes of sensor
master password	Thing
telephone number	Thing
sensor event	Occurrence
audible alarm	External entity
monitoring service	Organizational unit or External entity

The SafeHome security function enables the **homeowner** to configure the security system when it is installed, monitors all **sensors** connected to the **security system**, and interacts with the homeowner through the Internet, a PC or a **control panel**. During **installation**, the SafeHome PC is used to program and configure the **system**. Each sensor is assigned a **number and type**, a **master password** is programmed for arming and disarming the system, and **telephone number(s)** are input for dialing when a **sensor event** occurs.

When a sensor event is recognized, the software invokes an **audible alarm** attached to the system. After a delay time that is specified by the homeowner during system configuration activities, the software dials a telephone number of a **monitoring service**, provides information about the location, reporting the nature of the event that has been detected. The telephone number will be redialed every 20 seconds until telephone connection is obtained.

The homeowner receives security information via a control panel, the PC, or a browser, collectively called an interface. The interface displays prompting messages and system status information on the control panel, the PC, or the browser window. Homeowner interaction takes the following form . . .



# SELECTION CHARACTERISTICS

**06 selection characteristics** that should be used in the analysis model:

- 1) **Retained information:** The potential class will be useful during analysis only if information about it must be remembered so that the system can function.
- 2) **Needed services:** The potential class must have a set of identifiable operations that can change the value of its attributes in some way.
- 3) **Multiple attributes:** During requirement analysis, the focus should be on “major” information; a class with a single attribute may, in fact, be useful during design, but is probably better represented as an attribute of another class.
- 4) **Common attributes:** A set of attributes can be defined for the potential class and these attributes apply to all instances of the class.
- 5) **Common operations:** A set of operations can be defined for the potential class and these operations apply to all instances of the class.
- 6) **Essential requirements:** External entities that appear in the problem space and produce or consume information essential to the operation of any solution for the system will almost always be defined as classes in the requirements model.

A potential object should satisfy all (or almost all) of these characteristics.

# SELECTION CHARACTERISTICS

- However, the first step of class-based modeling is the definition of classes, and decisions (even subjective ones) must be made. With this in mind, you should apply the selection characteristics to the list of potential SafeHome classes:

Potential Class	Characteristic Number That Applies
Homeowner	rejected: 1, 2 fail even though 6 applies
Sensor	accepted: all apply
control panel	accepted: all apply
Installation	rejected
system (alias security function)	accepted: all apply
number, type	rejected: 3 fails, attributes of sensor
master password	rejected: 3 fails
telephone number	rejected: 3 fails
sensor event	accepted: all apply
audible alarm	accepted: 2, 3, 4, 5, 6 apply
monitoring service	rejected: 1, 2 fail even though 6 applies

It should be noted that (1) the preceding list is not all inclusive, additional classes would have to be added to complete the model; (2) some of the rejected potential classes will become attributes for those classes that were accepted (e.g., number and type are attributes of Sensor, and master password and telephone number may become attributes of System); (3) different statements of the problem might cause different "accept or reject" decisions to be made (e.g., if each homeowner had an individual password or was identified by voice print, the Homeowner class would satisfy characteristics 1 and 2 and would have been accepted).

# SPECIFYING ATTRIBUTES

- Attributes describe a class that **has been selected for inclusion in the analysis model**. In essence, it is the attributes that define the class—that **clarify what is meant by the class in the context of the problem space**.

Example:

(1) Building a system that tracks **baseball statistics for professional baseball players**

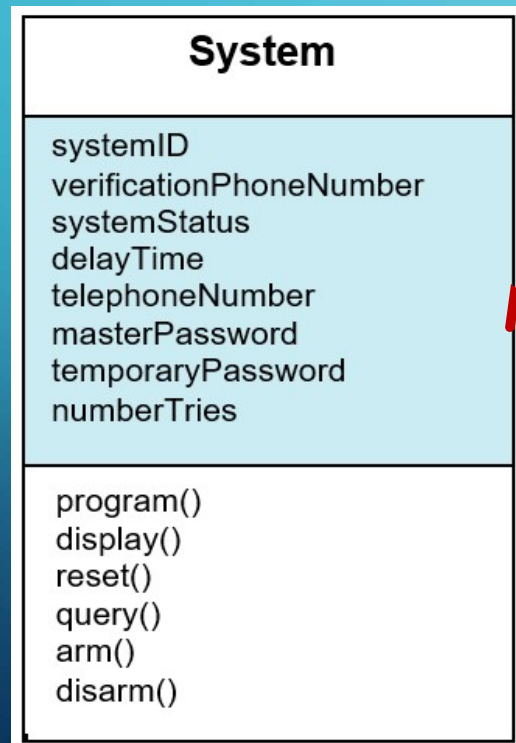
=> Attributes of the class Player (attributes such as name, position, batting average, fielding percentage, years played, and games played).

(2) Building a **professional baseball pension system**:

=> Some of above attributes of the class Player would be meaningful, but others would be replaced (or augmented) by attributes like average salary, credit toward full vesting, pension plan options chosen, mailing address, and the like.

- To develop a **meaningful set of attributes** for an analysis class, you should **study each use case and select those “things” that reasonably “belong” to the class**.
- Following question should be answered for each class: **What data items (composite and/or elementary) fully define this class in the context of the problem at hand?**

# EXAMPLE OF “SYSTEM” CLASS



A homeowner can configure the security function to reflect sensor information, **alarm response information**, **activation/deactivation information**, **identification information**, and so forth. We can represent these composite data items in the following manner:

**identification information** = system ID + verification phone number + system status

**alarm response information** = delay time + telephone number

**activation/deactivation information** = master password + number of allowable tries + temporary password

# DEFINING OPERATIONS

- Operations define the behavior of an object.
- Four broad categories:
  - (1) operations that **manipulate** data in some way (e.g., adding, deleting, reformatting, selecting),
  - (2) operations that **perform** a computation,
  - (3) operations that **inquire** about the state of an object, and
  - (4) operations that **monitor** an object for the occurrence of a controlling event.
- Functions are accomplished by **operating on attributes** and/or **associations**.  
Therefore, **an operation must have “knowledge” of the nature of the class attributes and associations.**

# EXAMPLE OF “SYSTEM” CLASS

System
systemID verificationPhoneNumber systemStatus delayTime telephoneNumber masterPassword temporaryPassword numberTries
program() display() reset() query() arm() disarm()

- As a first iteration at deriving a set of operations for an analysis class, you can again study a processing narrative (or use case) and select those operations that reasonably belong to the class. To accomplish this, the grammatical parse is again studied and verbs are isolated. Some of these verbs will be legitimate operations and can be easily connected to a specific class. For example, from the SafeHome processing narrative presented earlier in this chapter, we see that “sensor is assigned a number and type” or “a master password is programmed for arming and disarming the system.”
- These phrases indicate a number of things:
  - That an **assign()** operation is relevant for the Sensor class.
  - That a **program()** operation will be applied to the System class.
  - That **arm()** and **disarm()** are operations that apply to System class.



# PROJECT: **SAFEHOME** CLASS MODEL

**These functions are accomplished by operating on attributes and/or associations. Therefore, an operation must have “knowledge” of the nature of the class attributes and associations.**

**Ed:** So when the homeowner wants to pick a camera, he or she has to pick it from a floor plan. I’ve defined a FloorPlan class. Here’s the diagram. (They look at Figure 10.2.)

**Jamie:** So FloorPlan is an object that is put together with walls, doors, windows, and cameras. That’s what those labeled lines mean, right?

**Ed:** Yeah, they’re called “associations.” One class is associated with another according to the associations I’ve shown.

**Vinod:** So the actual floor plan is made up of walls and contains cameras and sensors that are placed within those walls. How does the floor plan know where to put those objects?

**Ed:** It doesn’t, but the other classes do. See the attributes under, say, WallSegment, which is used to build a wall. The wall segment has start and stop coordinates and the draw() operation does the rest.

**Jamie:** And the same goes for windows and doors. Looks like camera has a few extra attributes.

## Class Models

The scene: Ed’s cubicle, as analysis modeling begins.

The players: Jamie, Vinod, and Ed — all members of the SafeHome software engineering team.

**Ed:** Yeah, I need them to provide pan and zoom info.

**Vinod:** I have a question. Why does the camera have an ID but the others don’t? I notice you have an attribute called nextWall. How will WallSegment know what the next wall will be?

**Ed:** Good question, but as they say, that’s a design decision, so I’m going to delay that until . . .

**Jamie:** Give me a break . . . I’ll bet you’ve already figured it out.

**Ed (smiling sheepishly):** True, I’m gonna use a list structure which I’ll model when we get to design. If you get religious about separating analysis and design, the level of detail I have right here could be suspect.

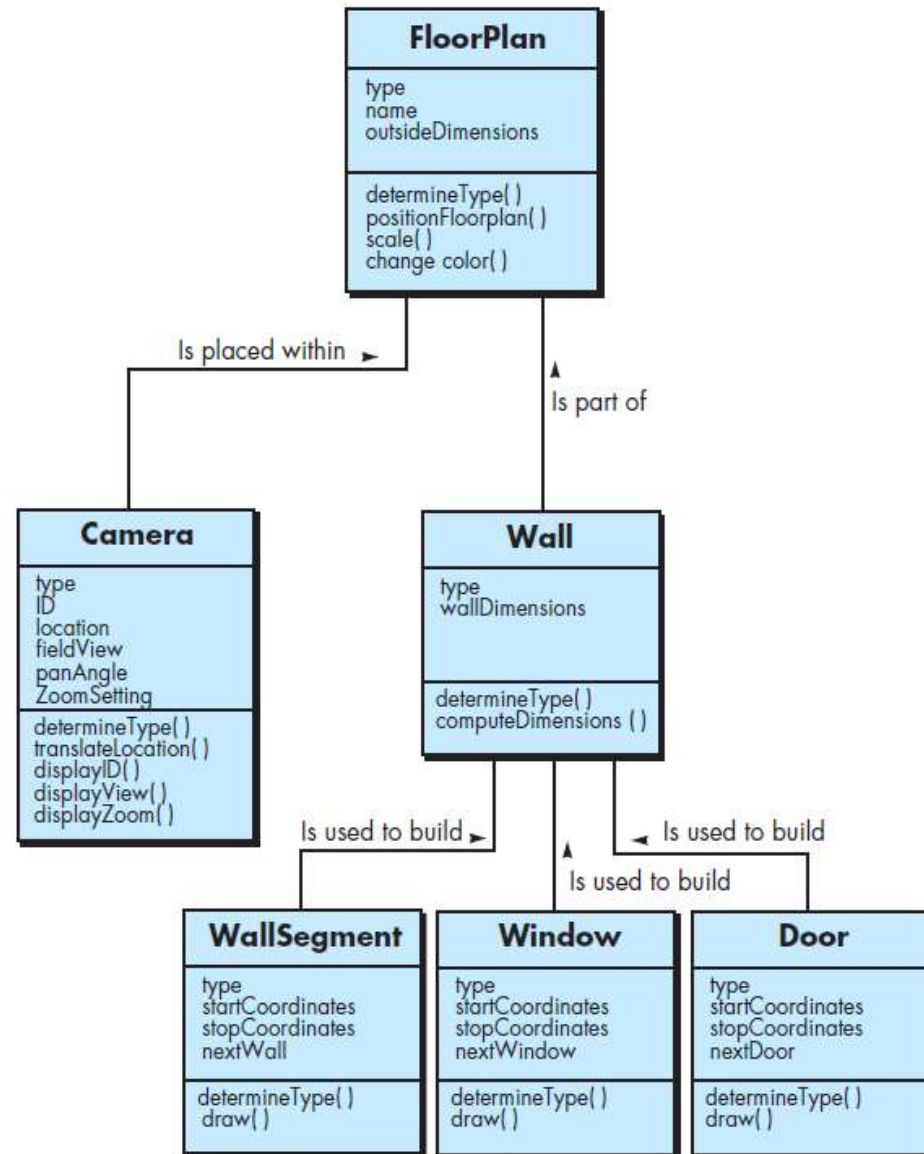
**Jamie:** Looks pretty good to me, but I have a few more questions. (Jamie asks questions which result in minor modifications.)

**Vinod:** Do you have CRC cards for each of the objects? If so, we ought to role-play through them, just to make sure nothing has been omitted.

**Ed:** I’m not quite sure how to do them.

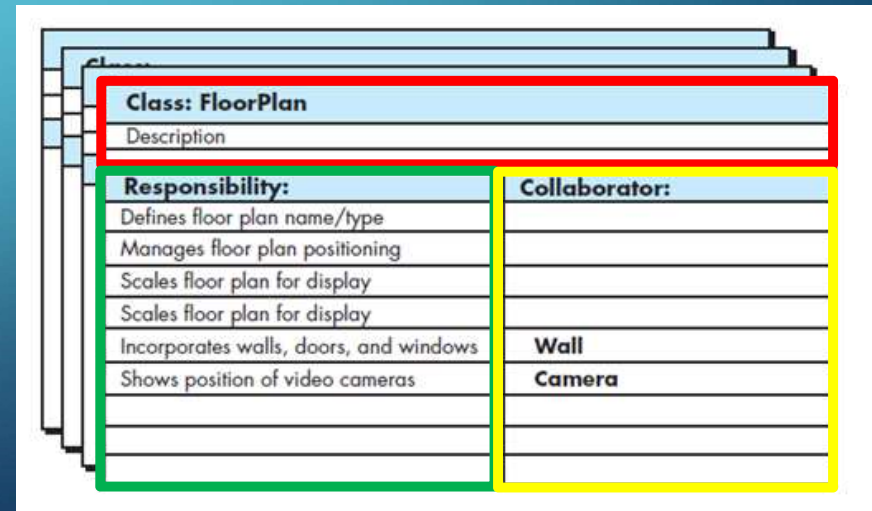
**Vinod:** It’s not hard and they really pay off. I’ll show you.

# CLASS DIAGRAM FOR FLOOR-PLAN (SEE SIDEBAR DISCUSSION)



# CLASS-RESPONSIBILITY-COLLABORATOR MODELING

- **Class-responsibility-collaborator (CRC) modeling** [Wir90] provides a simple means for **identifying** and **organizing** the classes that are **relevant to system or product requirements**.
- Ambler [Amb95] describes **CRC modeling** in the following way:
  - A CRC model is really a collection of standard index cards that represent classes.
  - Cards are divided into three sections:
    - Top: Writing the name of the class.
    - Left body: Listing the class responsibilities.
    - Right body: Listing the collaborators.



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

# CLASS-RESPONSIBILITY-COLLABORATOR MODELING

- **Classes.** Basic guidelines for identifying classes and objects were presented earlier in this chapter. The taxonomy of class types can be extended by considering the following categories:
  - **Entity classes**, also called **model or business classes**, are extracted directly from the statement of the problem (e.g., FloorPlan and Sensor).
  - **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 (e.g., CameraWindow).
  - **Controller classes** manage a “unit of work” from start to finish. That is, controller classes can be designed to manage (1) the creation or update of entity objects, (2) the instantiation of boundary objects as they obtain information from entity objects, (3) complex communication between sets of objects, (4) validation of data communicated between objects or between the user and the application. In general, controller classes are not considered until the design activity has begun.

# CLASS-RESPONSIBILITY-COLLABORATOR MODELING

- **Responsibilities.** Wirfs-Brock and her colleagues [Wir90] suggest **five guidelines** for allocating responsibilities to classes:
  - (1) System intelligence should be distributed across classes to best address the needs of the problem.
  - (2) Each responsibility should be stated as generally as possible.
  - (3) Information and the behavior related to it should reside within the same class.
  - (4) Information about one thing should be localized with a single class, not distributed across multiple classes.
  - (5) Responsibilities should be shared among related classes, when appropriate.

# CLASS-RESPONSIBILITY-COLLABORATOR MODELING

- **Collaborations.** Classes fulfill their responsibilities in one of **two ways**:
  - (1) A class can **use its own operations to manipulate its own attributes**, thereby **fulfilling a particular responsibility**, or
  - (2) a class can **collaborate with other classes**. Wirfs-Brock and her colleagues [Wir90] define collaborations in the following way:

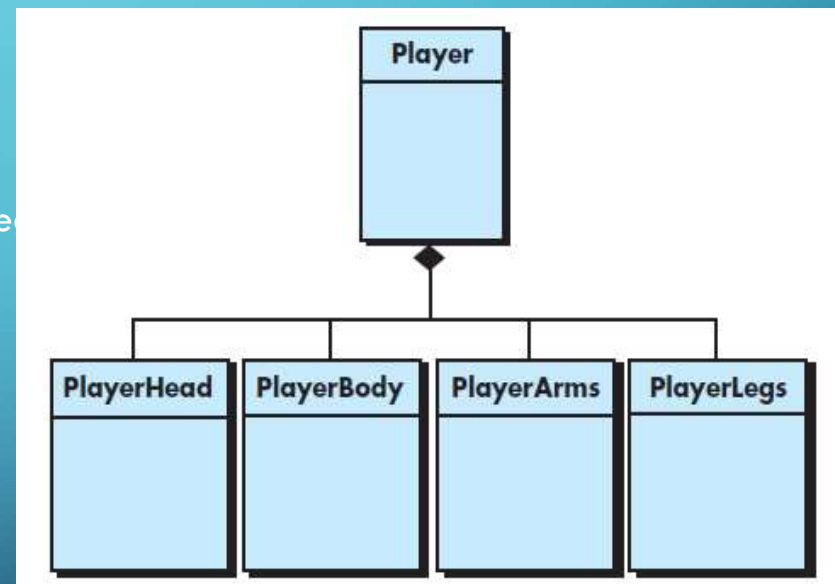
Collaborations represent **requests from a client to a server** in fulfillment of a client responsibility. A collaboration is **the embodiment of the contract between the client and the server . . . .** We say that an object collaborates with another object if, to fulfill a responsibility, **it needs to send the other object any messages**. A **single collaboration flows in one direction** — representing a request from the client to the server. From the client's point of view, **each of its collaborations is associated with a particular responsibility implemented by the server**.

# CLASS-RESPONSIBILITY-COLLABORATOR MODELING

- Collaborations are identified by determining whether a class can fulfill each responsibility itself. **If it cannot, then it needs to interact with another class. Hence, a collaboration.**
- To help in the identification of collaborators, you can **examine three different generic relationships between classes** [Wir90]:
  - (1) the is-part-of relationship,
  - (2) the has-knowledge-of relationship, and
  - (3) the depends-upon relationship.

# CLASS-RESPONSIBILITY-COLLABORATOR MODELING

- All classes that are part of **an aggregate class** are connected to the aggregate class via an is-part-of relationship.
- Consider the classes defined for the video game noted earlier, the class PlayerBody **is-part-of** Player, as are PlayerArms, PlayerLegs, and PlayerHead.
- In UML, these relationships are represented as the **aggregation**.



Aggregation relationship

# ASSOCIATIONS AND DEPENDENCIES

- In many instances, two analysis classes are related to one another in some fashion.
- In UML these relationships are called associations.
- Example: FloorPlan class.
  - The FloorPlan class is defined by identifying a set of associations between FloorPlan and two other classes, Camera and Wall.
  - The class Wall **is associated with three classes** that allow a wall to be constructed, WallSegment, Window, and Door.

# ASSOCIATIONS AND DEPENDENCIES

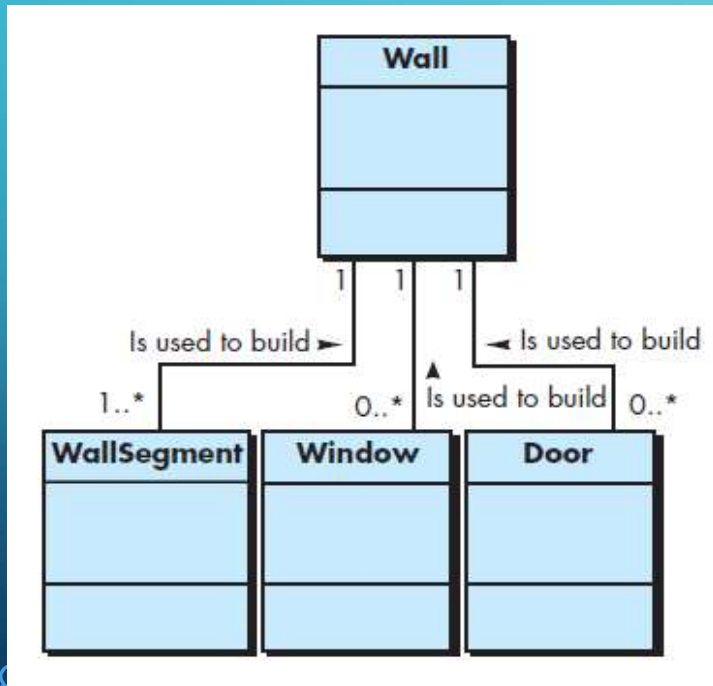


Figure 5.5. Multiplicity.

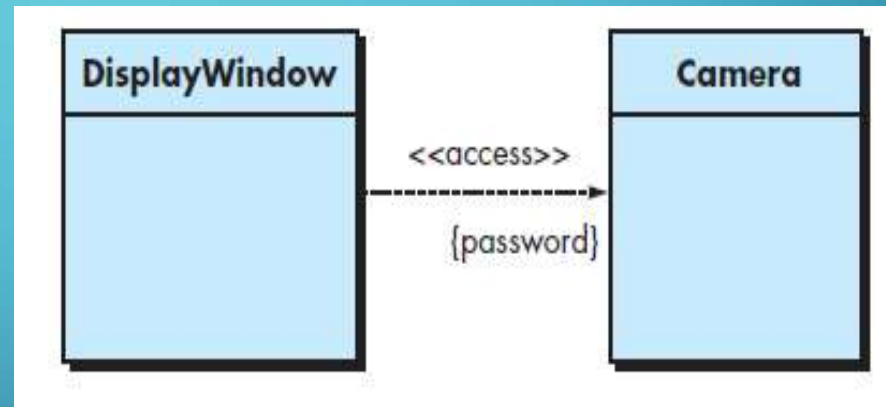


Figure 5.6. Dependencies.



# ANALYSIS PACKAGES

- An important part of analysis modeling is **categorization**.
- That is, various elements of the requirements model (e.g., use cases, analysis classes) **are categorized in a manner that packages** them as a **grouping** — called an analysis package — that is given **a representative name**.

# ANALYSIS PACKAGES

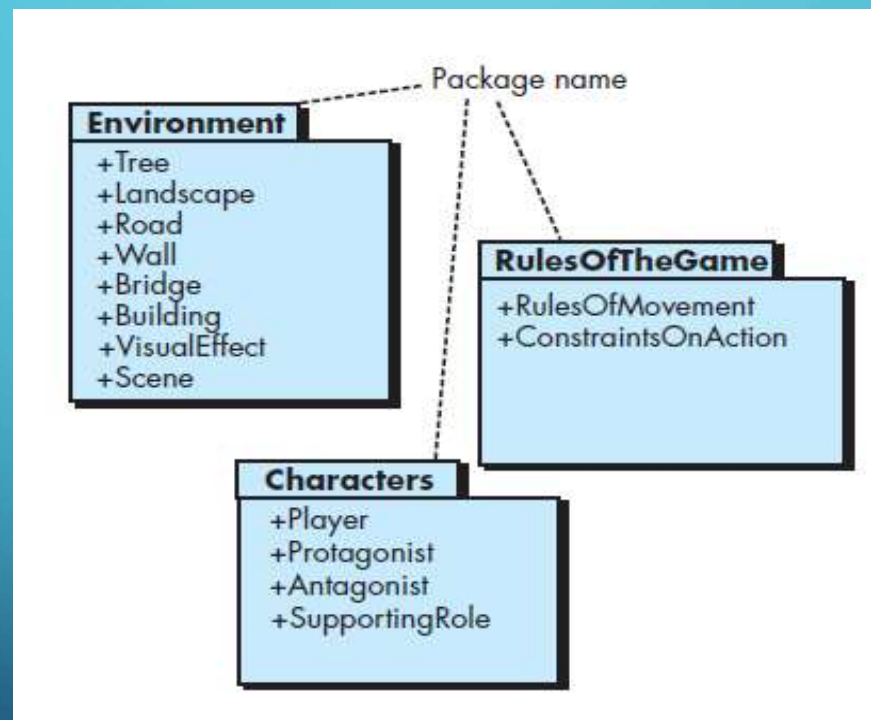


Figure 5.7. Packages.



# SUMMARY

- **Class-based modeling** uses information derived from use cases and other written application descriptions to identify analysis classes.
- A grammatical parse may be used to extract candidate classes, attributes, and operations from text-based narratives.
- A set of **class-responsibility-collaborator** index cards can be used to define relationships between classes.
- In addition, a variety of **UML modeling notation** can be applied to define **hierarchies, relationships, associations, aggregations, and dependencies** among classes.
- Analysis packages are used to **categorize and group classes** in a manner that makes them **more manageable for large systems**.



## Q & A

1. You have been asked to build one of the following systems: A network-based course registration system for your university; A Web-based order-processing system for a computer store; A simple invoicing system for a small business; An Internet-based cookbook that is built into an electric range or microwave. Select the system that is of interest to you and develop a processing narrative. Then use the grammatical parsing technique to identify candidate objects and classes.
2. Develop a set of operations that are used within the classes identified in Problem 5.1.
3. Write a template-based use case for the SafeHome home management system described informally in the sidebar following Section 5.4.
4. Develop a complete set of CRC model index cards on the product or system you chose as part of Problem 5.1.
5. Conduct a review of the CRC index cards with your colleagues. How many additional classes, responsibilities, and collaborators were added as a consequence of the review?
6. What is an analysis package and how might it be used?

(PROBLEMS AND POINTS TO PONDER)