

# Are you ready?

☐ A Yes

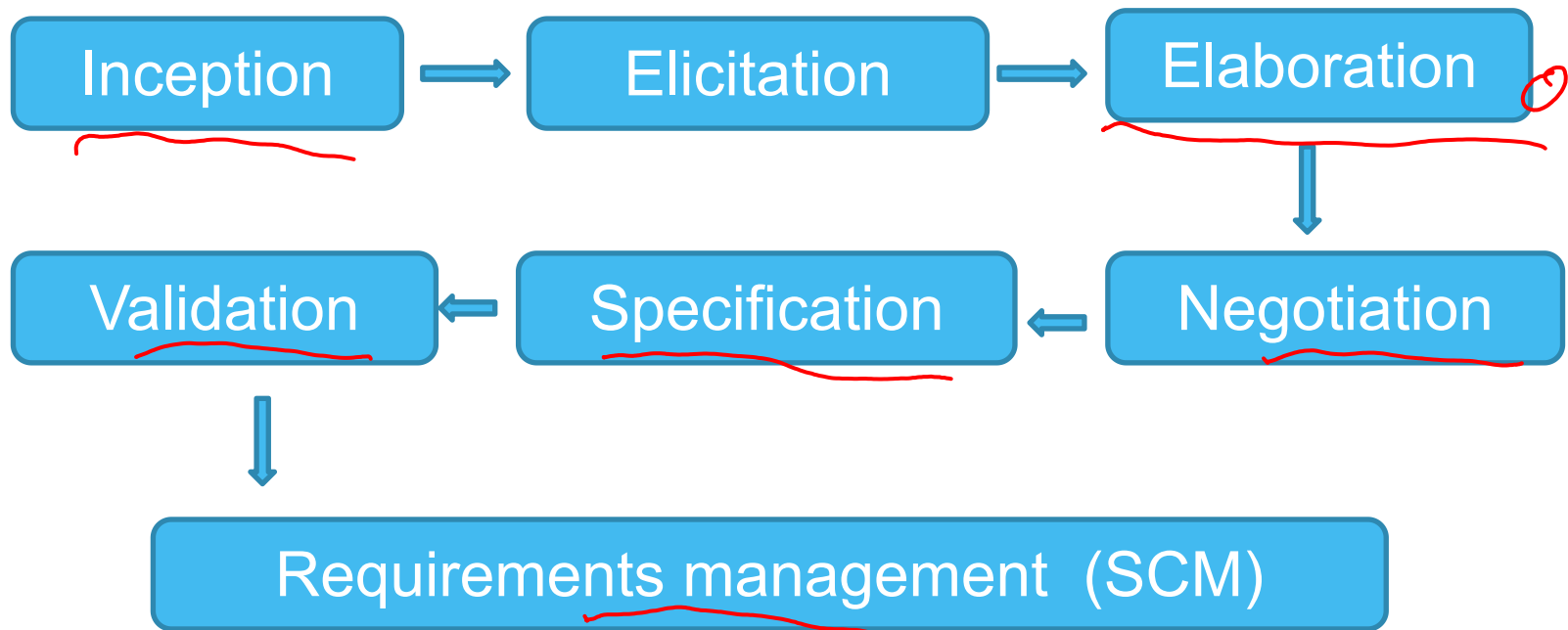
☐ B No



提交

# Review – Ch8

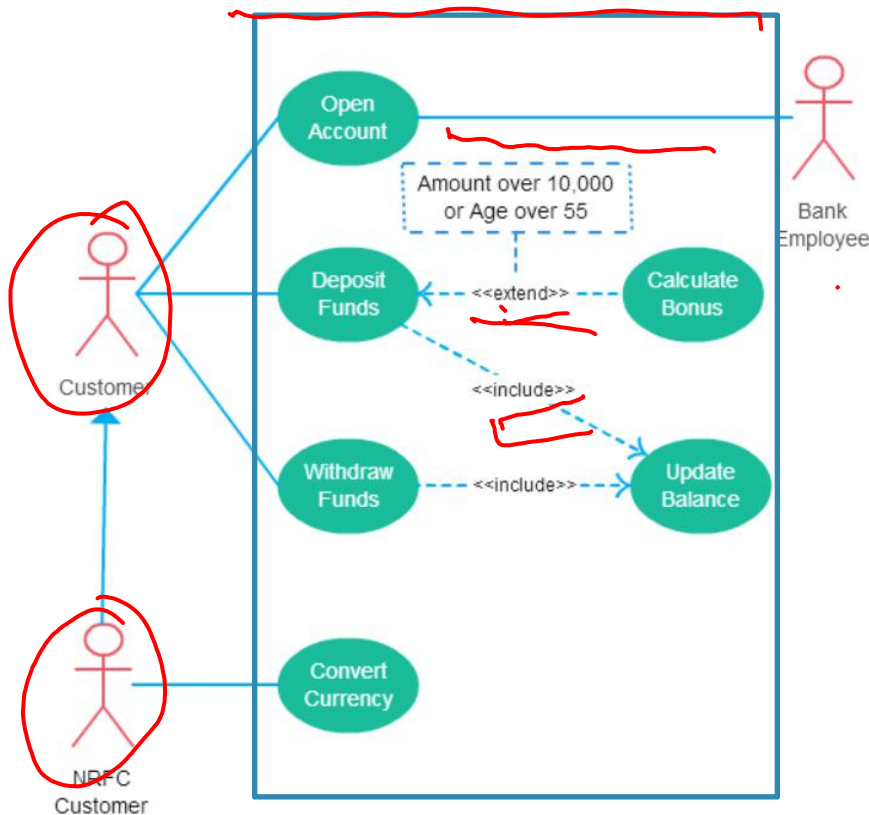
- Requirements Engineering



# Review – Ch9 Use case Diagram

- scenario-based model

UML



*Includes is usually used to model common behavior*

## Relationship in use-case diagram

Type	Description
<u>Association</u>	between actor and use case
Generalization ( <u>Inheritance</u> )	between actor or between use case
Include (Every time)	between <u>use case</u>
Extend (Some case)	between use case

# Review – Ch9 Use case Diagram

- Quiz: An Old Examination Question

## The Pizza Ordering System

*The Pizza Ordering System allows the user of a web browser to order pizza for home delivery. To place an order, a shopper searches to find items to purchase, adds items one at a time to a shopping cart, and possibly searches again for more items.*

*When all items have been chosen, the shopper provides a delivery address. If not paying with cash, the shopper also provides credit card information.*

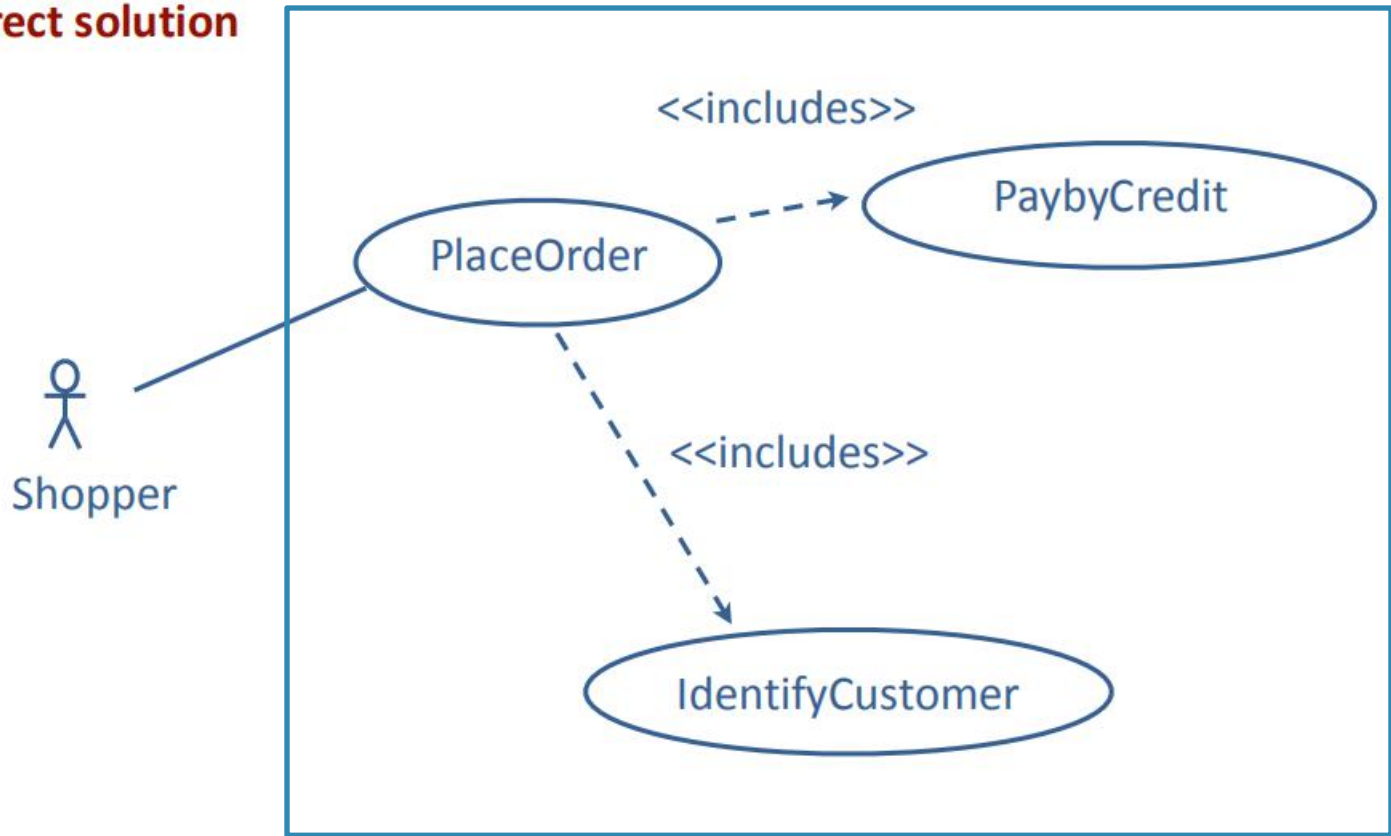
*The system has an option for shoppers to register with the pizza shop. They can then save their name and address information, so that they do not have to enter this information every time that they place an order.*

Develop a use case diagram, for a use case for placing an order, PlaceOrder. The use case should show a relationship to two previously specified use cases, IdentifyCustomer, which allows a user to register and log in, and PaybyCredit, which models credit card payments.

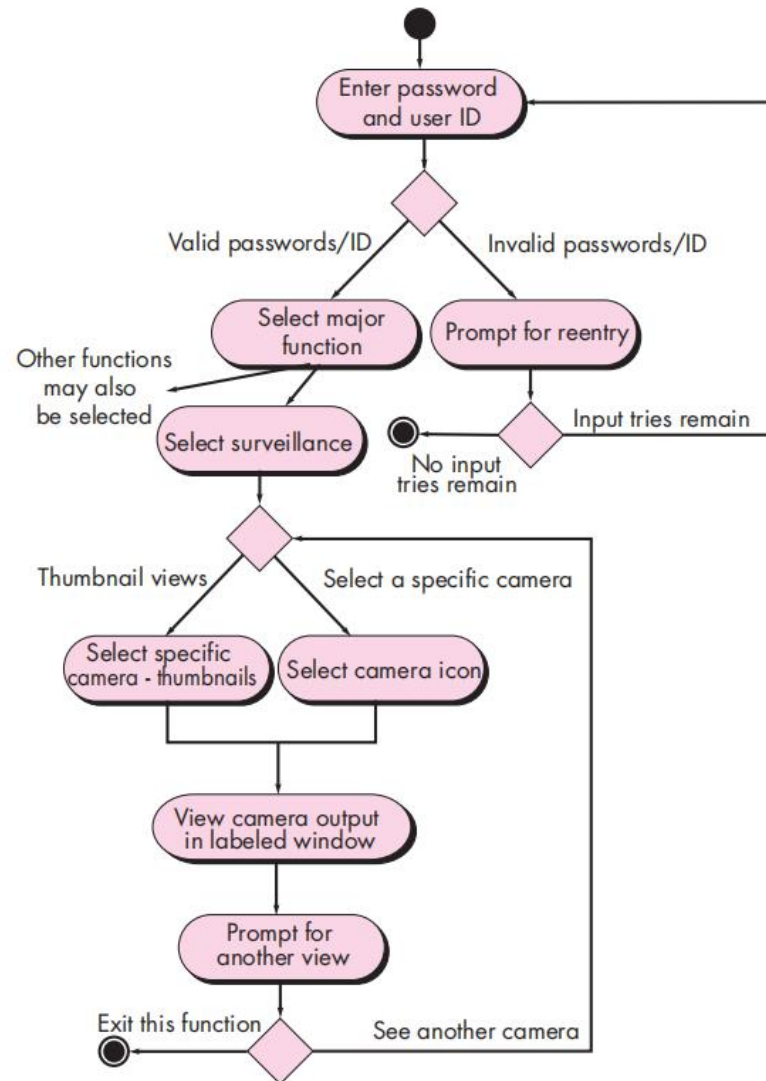
# Review – Ch9 Use case Diagram

- An Old Examination Question

Correct solution

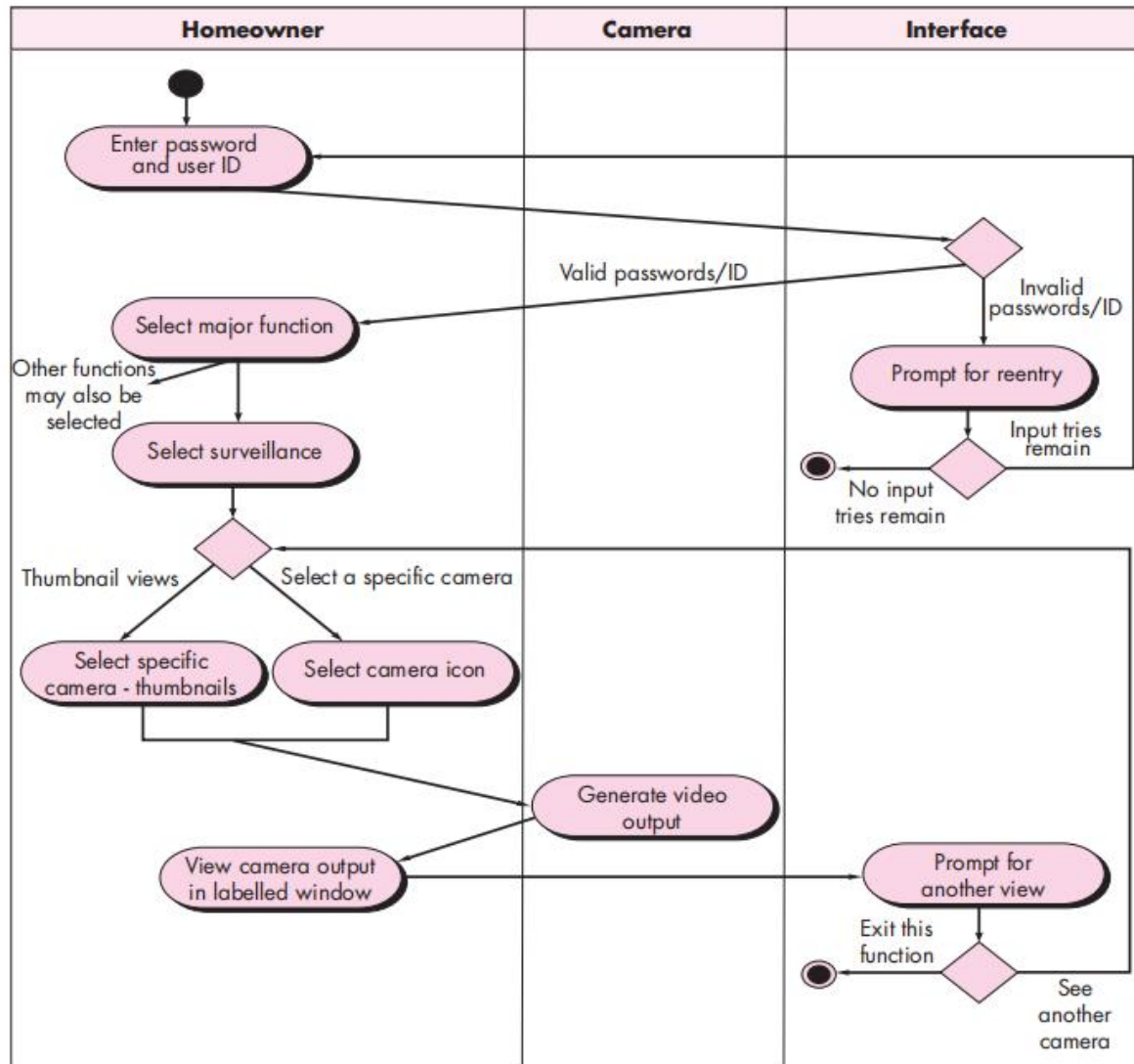


# Review – Ch9 Activity Diagram





# Review – Ch9 Swimlane Diagrams





# Software Engineering

## Part 2 Modeling

### Chapter 10

## Requirements Modeling: Class-Based Methods



# Contents

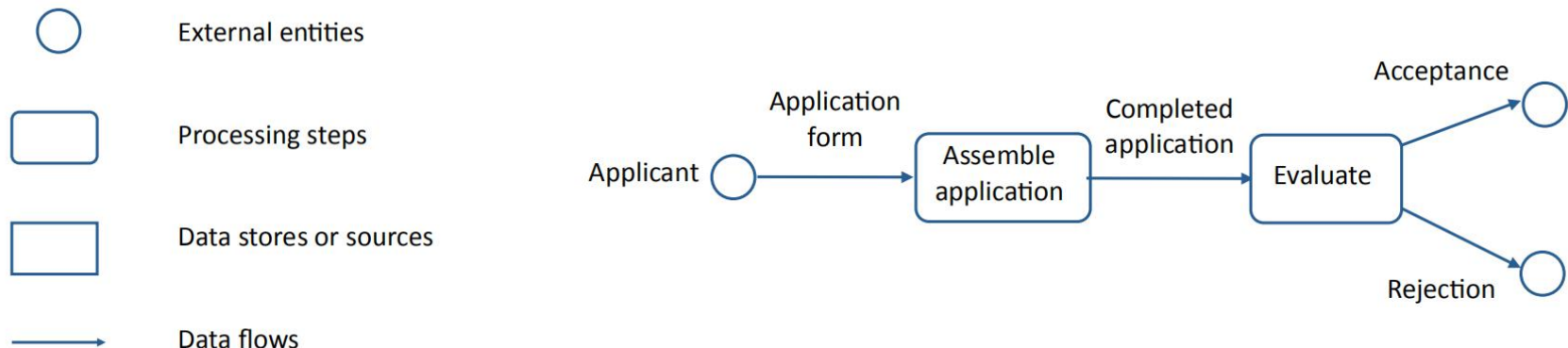
- 10.1 Requirements Modeling: Class-based Methods
- 10.2 Specifying Attributes
- 10.3 Defining Operations
- 10.4 Class-Responsibility-Collaborator Modeling
- 10.5 Associations and Dependencies
- 10.6 Analysis Packages

# Requirements Modeling Strategies

- *structured analysis*

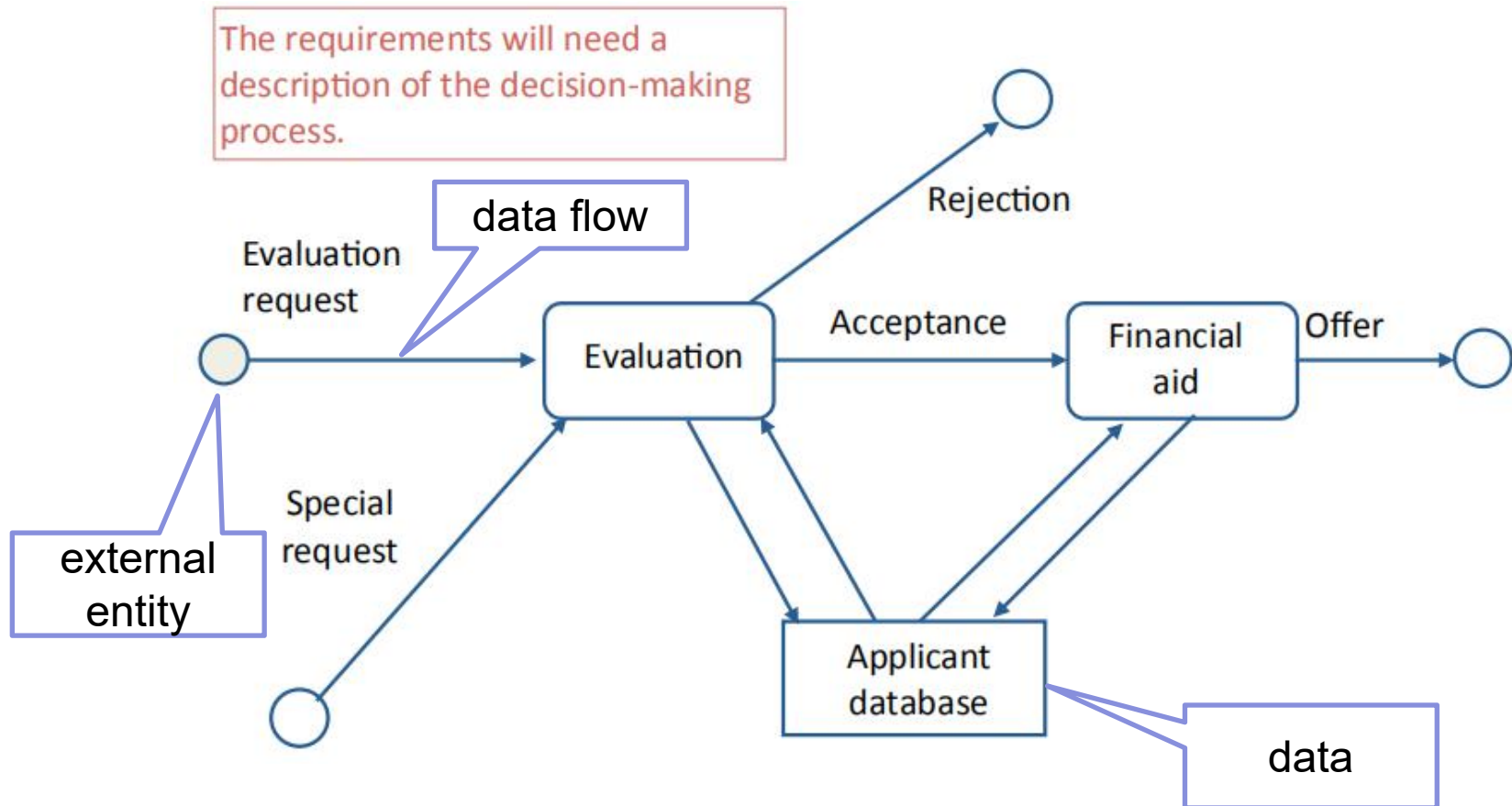
considers data and the processes that transform the data as separate entities.

- **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.



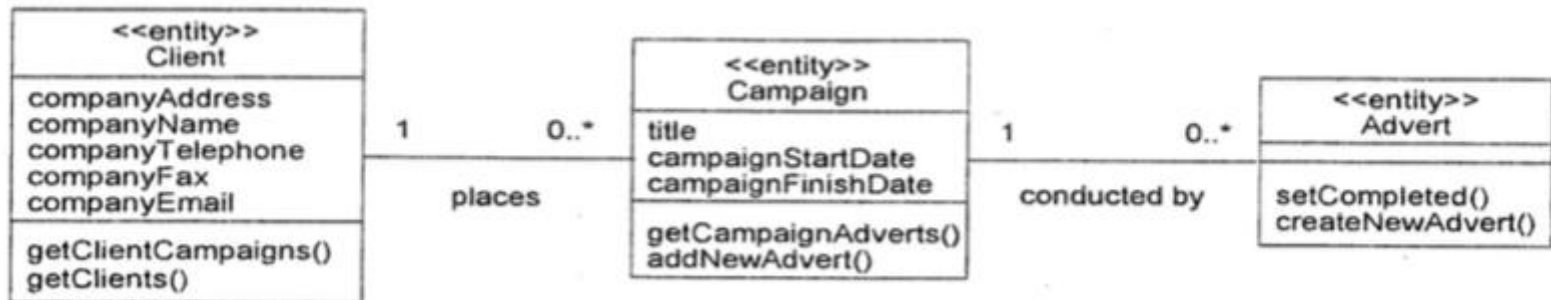
Graph are cited from: CS5150 Cornell University slide

# Requirements Modeling Strategies



# Requirements Modeling Strategies

- *object-oriented analysis*, focuses on
  - the definition of **classes** and
  - the manner in which they collaborate with one another to effect customer requirements.



# 10.1 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.
- The elements of a class-based model include classes and objects, attributes, operations, CRC models, collaboration diagrams and packages.

# 10.1 Identifying Analysis Classes

- Examining the usage scenarios developed as part of the requirements model and perform a "grammatical parse" [Abb83]
  - **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**.



# 10.1 Analysis Classes

- *Analysis classes* manifest themselves in the following ways:
  - *External entities* (e.g., other systems, devices, people) that produce or consume information
  - *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
  - *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
  - *Structures* (e.g., sensors, or computers) that define a class of objects or related classes of objects

# 10.1 Analysis Classes - noun and verb

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 promoting messages and system status information on the control panel, the PC, or the browser window. Homeowner interaction takes the following form . . .



# 10.1 Analysis Classes - Extracting the nouns

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 . . .

# 10.1 Analysis Classes - Extracting the nouns

A number of potential classes:

---

## Potential Class

homeowner

sensor

control panel

installation

system (alias security system)

number, type

master password

telephone number

sensor event

audible alarm

monitoring service

## General Classification

role or external entity

external entity

external entity

occurrence

thing

not objects, attributes of sensor

thing

thing

occurrence

external entity

organizational unit or external entity

---

# 10.1 Potential Classes Selection

1. *Retained information.* The potential class will be useful during analysis only if information 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 during the analysis activity.
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.

# 10.1 Potential Classes Selection

## Selection

---

### Potential Class

homeowner

sensor

control panel

installation

system (alias security function)

number, type

master password

telephone number

sensor event

audible alarm

monitoring service

### Characteristic Number That Applies

rejected: 1, 2 fail even though 6 applies

accepted: all apply

accepted: all apply

rejected

accepted: all apply

rejected: 3 fails, attributes of sensor

rejected: 3 fails

rejected: 3 fails

accepted: all apply

accepted: 2, 3, 4, 5, 6 apply

rejected: 1, 2 fail even though 6 applies

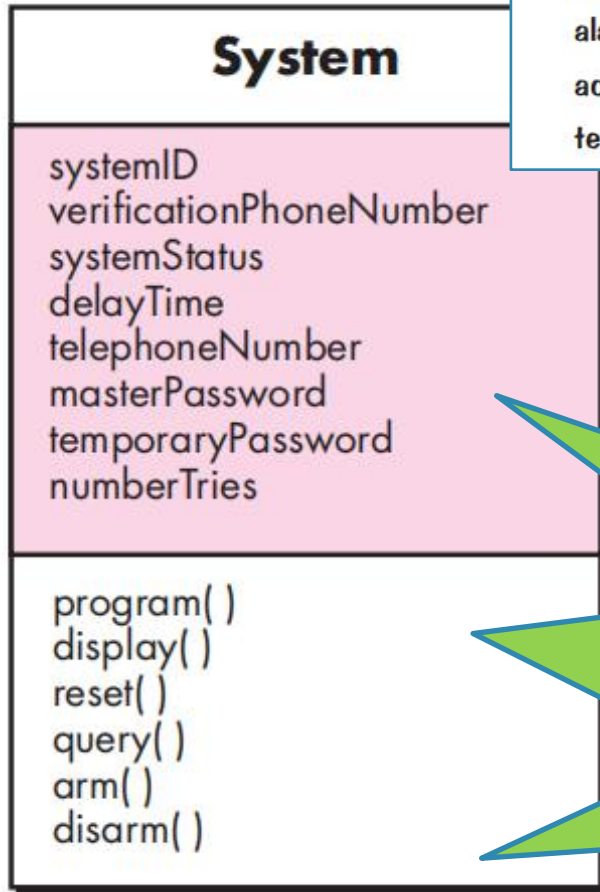
---



## 10.2 Defining Attributes

- *Attributes* describe a class that has been selected for inclusion in the analysis model.
  - two different classes for professional baseball players
    - **For Playing Statistics software:** name, position, batting average, fielding percentage, years played, and games played might be relevant.
    - **For Pension Fund software:** average salary, credit toward full vesting, pension plan options chosen, mailing address, and the like.

## 10.2 Defining Attributes



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

Sensor ?

↑  
composite  
data items

avoid defining an item as  
an attribute if more than  
one of the items is to be  
associated with the class

## 10.2 Defining Attributes

NoticeAction	
-	notice: Notice
-	noticeService: NoticeService
+	saveNotice(Notice) : void
+	dispatchNotice() : Notice
+	findNotice() : List
+	replyNotice(Reply) : void

- Public (+)
- Private (-)
- Protected (#)
- Package (~)
- Derived (/)
- Static (underlined)

## 10.3 Defining Operations

- Do a grammatical parse of a processing narrative and look at the **verbs**
- Operations can be divided into four 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
  4. operations that **monitor an object** for the occurrence of a controlling event.

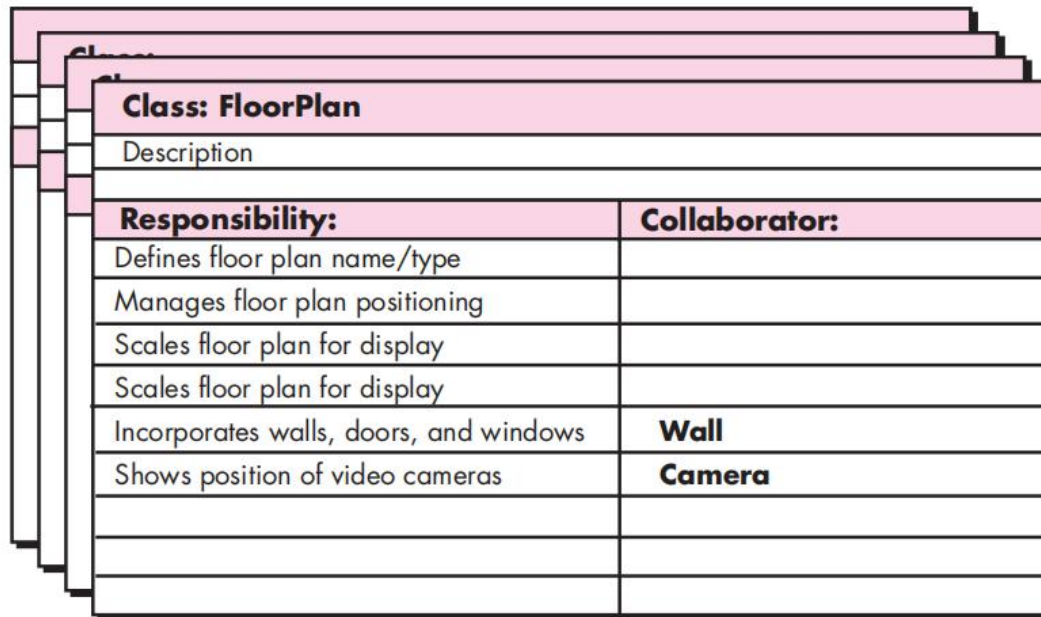
## 10.3 Defining Operations

- “sensor is assigned a number and type.”
  - `assign()` operation for **Sensor** class
- “a master password is programmed for arming and disarming the system.”
  - `program()` operation for **System** class
  - `arm()` and `disarm()` for **System** class

# 10.4 CRC Models

*to develop an organized representation of classes:*

- *Class-Responsibility-Collaborator (CRC) modeling* [Wir90] provides a simple means for **identifying and organizing** the classes that are relevant to system or product requirements.
  - A CRC model is really a collection of standard index cards that represent classes.



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	<b>Wall</b>
Shows position of video cameras	<b>Camera</b>



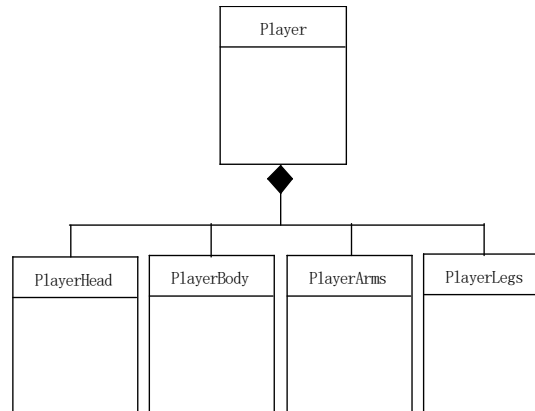
# 10.4 Class Types

- *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
  - 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

# 10.4 Responsibilities

## ● Guidelines for allocating responsibilities to classes

1. **System intelligence** should be distributed across classes to best address the needs of the problem. (abstraction)
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.



**Function !**

# 10.4 Collaborations

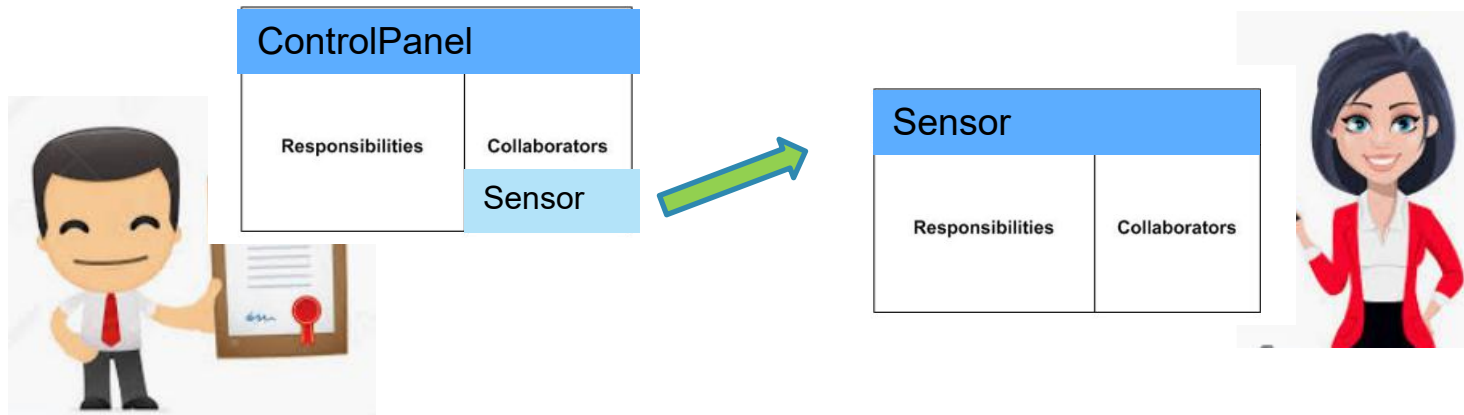
- Classes fulfill their responsibilities in one of two ways:
  - A class can use its own operations to manipulate its own attributes, thereby fulfilling a particular responsibility, or
  - a class can collaborate with other classes.
- Collaborations identify relationships between classes
- three different generic relationships between classes:
  - the *is-part-of* relationship (PlayerBody is-part-of Player )
  - the *has-knowledge-of* relationship ( determine-sensor-status(): Control panel and Sensor )
  - the *depends-upon* relationship ( PlayerHead depends-upon PlayerBody)

# 10.4 CRC - Review

Class Name	
Responsibilities	Collaborators

When the review leader comes to “control panel,” in the use case narrative, the token is passed to the person. holding the **ControlPanel** index card. The phrase “implies that a sensor is open” requires that the index card contains a responsibility that will validate this implication (the responsibility *determine-sensor-status()* accomplishes this). Next to the responsibility on the index card is the collaborator **Sensor**. The token is then passed to the **Sensor** object.

Textbook: P196





# 10.4 CRC - Role-play

Class Name

Responsibilities

Collaborators

## SAFEHOME



### CRC Models

**The scene:** Ed's cubicle, as requirements modeling begins.

**The players:** Vinod and Ed—members of the SafeHome software engineering team.

#### The conversation:

[Vinod has decided to show Ed how to develop CRC cards by showing him an example.]

**Vinod:** While you've been working on surveillance and Jamie has been tied up with security, I've been working on the home management function.

**Ed:** What's the status of that? Marketing kept changing its mind.

**Vinod:** Here's the first-cut use case for the whole function . . . we've refined it a bit, but it should give you an overall view . . .

**Use case:** SafeHome home management function.

**Narrative:** We want to use the home management interface on a PC or an Internet connection to control electronic devices that have wireless interface controllers. The system should allow me to turn specific lights on and off, to control appliances that are connected to a wireless interface, to set my heating and air-conditioning system to temperatures that I define. To do this, I want to select the devices from a floor plan of the house. Each device must be identified on the floor plan. As an optional feature, I want to control all audiovisual devices—audio, television, DVD, digital recorders, and so forth.

With a single selection, I want to be able to set the entire house for various situations. One is home, another is away, a third is overnight travel, and a fourth is extended travel. All of these situations will have settings

that will be applied to all devices. In the overnight travel and extended travel states, the system should turn lights on and off at random intervals (to make it look like someone is home) and control the heating and air-conditioning system. I should be able to override these settings via the Internet.

**Ed:** The facing fig

**Vinod (:** problem. home ma Let's use

**Ed:** Oka attributes rations a

**Vinod:**

**Ed:** May

**Vinod:**

**HomeM**

#### Attributes:

optionsPanel—contains info on buttons that enable user to select functionality.

situationPanel—contains info on buttons that enable user to select situation.

floorplan—same as surveillance object but this one displays devices.

deviceIcons—info on icons representing lights, appliances, HVAC, etc.

devicePanels—simulation of appliance or device control panel; allows control.

#### Operations:

*displayControl()*, *selectControl()*, *displaySituation()*, *select situation()*, *accessFloorplan()*, *selectDeviceIcon()*, *displayDevicePanel()*, *accessDevicePanel()*, . . .

**Class:** HomeManagementInterface

#### Responsibility

*displayControl()*  
*selectControl()*  
*displaySituation()*  
*selectSituation()*  
*accessFloorplan()*  
. . .

#### Collaborator

**OptionsPanel** (class)  
**OptionsPanel** (class)  
**SituationPanel** (class)  
**SituationPanel** (class)  
**FloorPlan** (class) . . .

**Ed:** So when the operation *accessFloorplan()* is invoked, it collaborates with the **FloorPlan** object just like the one we developed for surveillance. Wait, I have a description of it here. (They look at Figure 10.2.)

**Vinod:** Exactly. And if we wanted to review the entire class model, we could start with this index card, then go to the collaborator's index card, and from there to one of the collaborator's collaborators, and so on.

**Ed:** Good way to find omissions or errors.

**Vinod:** Yep.

Textbook: P197-P198

# 10.4 CRC

- CRC cards example:

<http://www.math-cs.gordon.edu/courses/cs211/ATMExample/CRCCards.html#ATM>

Class Name	
Responsibilities	Collaborators

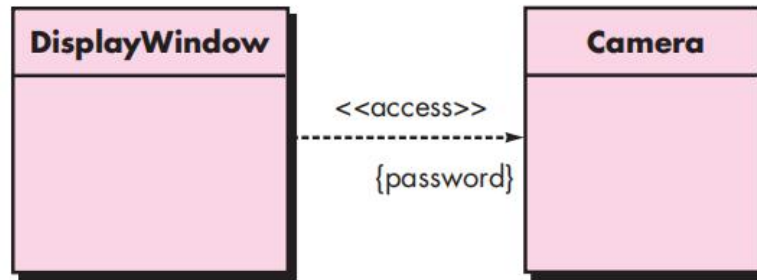
Class ATM	
<b><u>Responsibilities</u></b>  Start up when switch is turned on  Shut down when switch is turned off Start a new session when card is inserted by customer  Provide access to component parts for sessions and transactions	<b><u>Collaborators</u></b>  <a href="#"><u>OperatorPanel</u></a> <a href="#"><u>CashDispenser</u></a> <a href="#"><u>NetworkToBank</u></a> <a href="#"><u>NetworkToBank</u></a> <a href="#"><u>CustomerConsole</u></a> <a href="#"><u>Session</u></a>



## 10.5 Associations and Dependencies

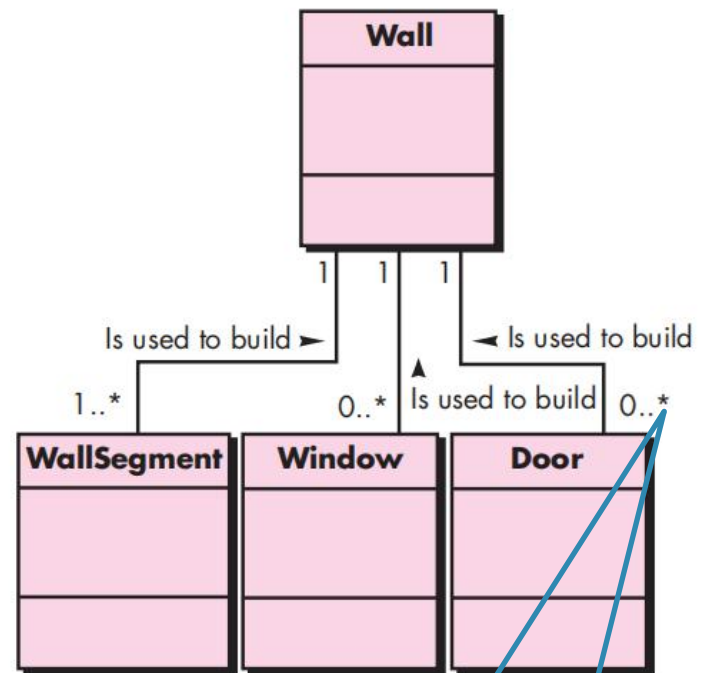
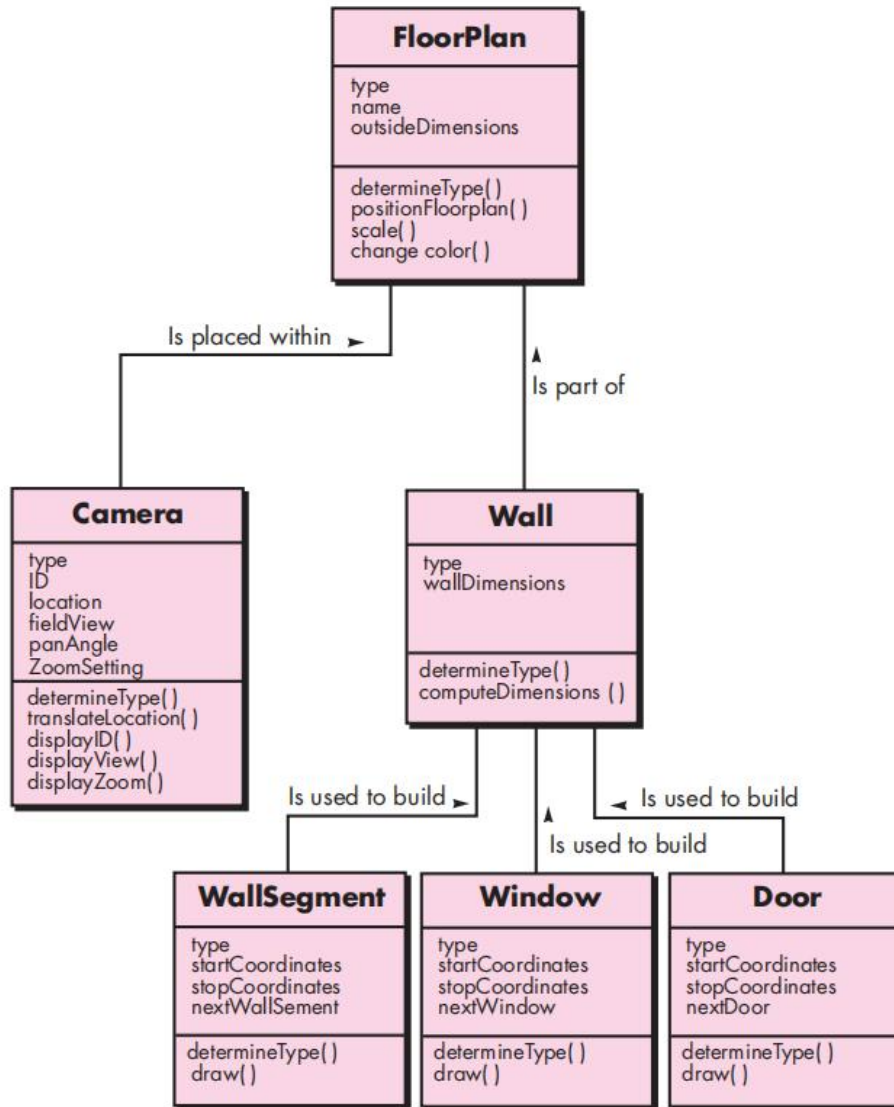
- Two analysis classes are often related to one another in some fashion
  - In UML these relationships are called *associations*
  - Associations can be refined by indicating *multiplicity* (the term *cardinality* is used in data modeling)
- In many instances, a client-server relationship exists between two analysis classes.
  - In such cases, a client-class depends on the server-class in some way and a *dependency relationship* is established

# 10.5 Dependencies



use case: surveillance  
a special password must be provided in  
order to view specific camera locations.

# 10.5 Multiplicity

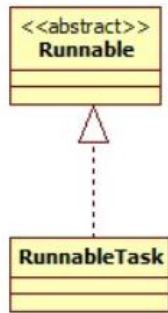


\* (asterisk) : an unlimited upper bound on the range.

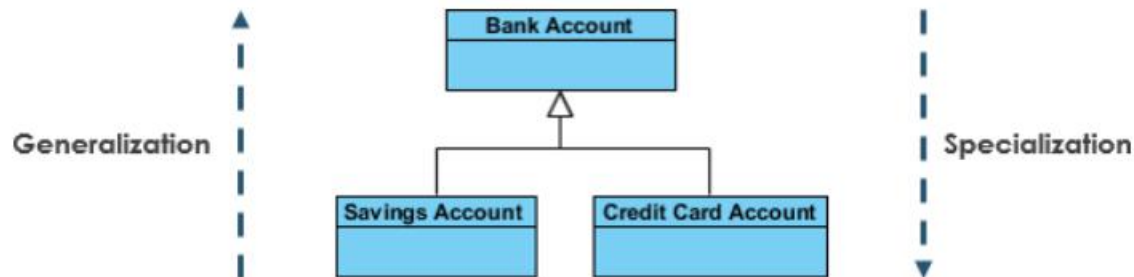
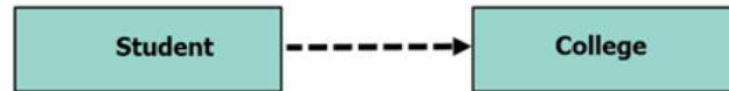
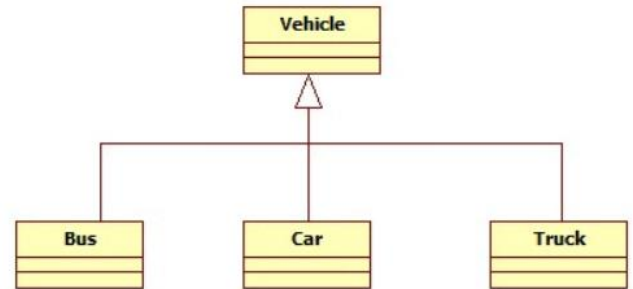
*cardinality*

# 10.5 UML Class diagram

## ● Relation between class



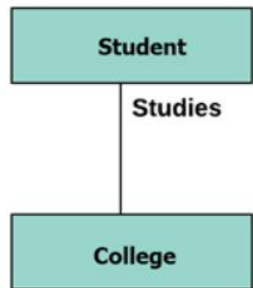
- ❑ Generalization (is-a): Inheritance
- ❑ Realization (Interface)
- ❑ Dependency (call)
- ❑ Association



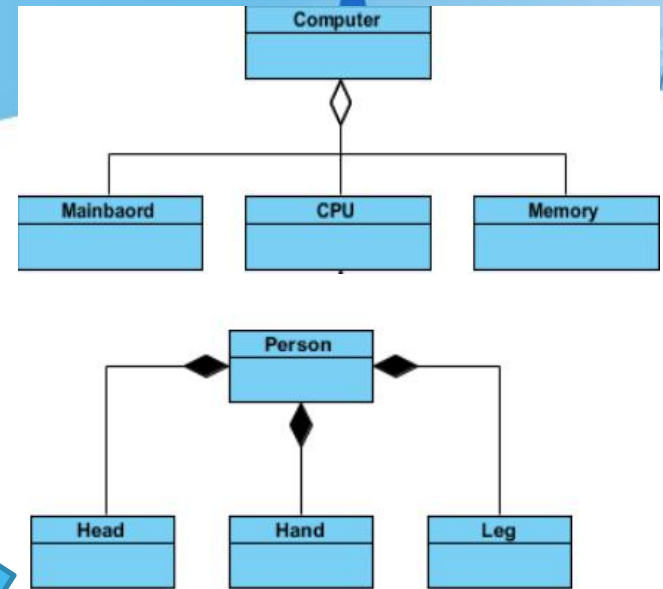
# 10.5 UML Class diagram

## ● Relation between class

### □ Association

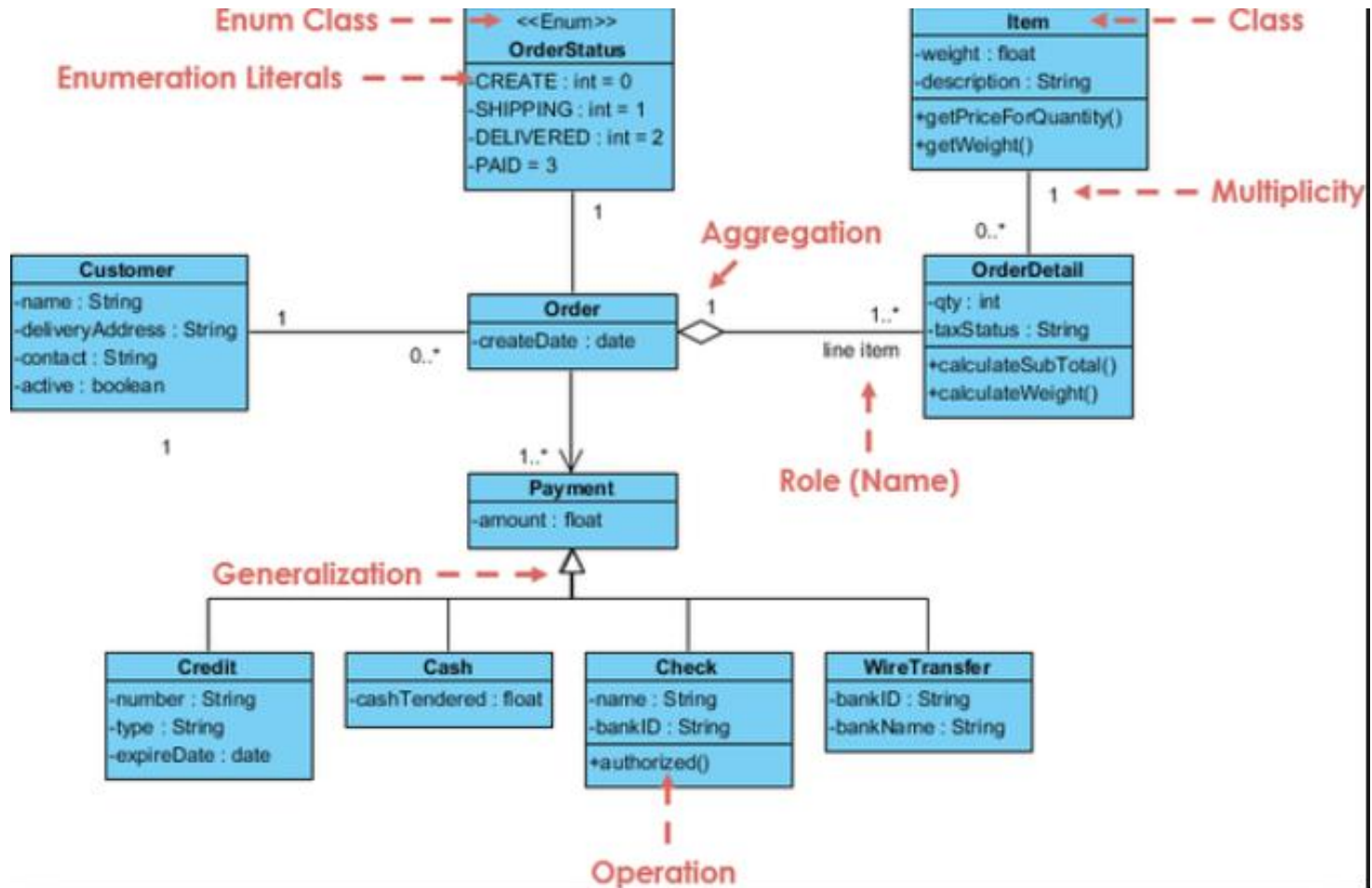


- Common (reference)
- Aggregation (has-a)
- Composition (contains-a, special case of aggregation)



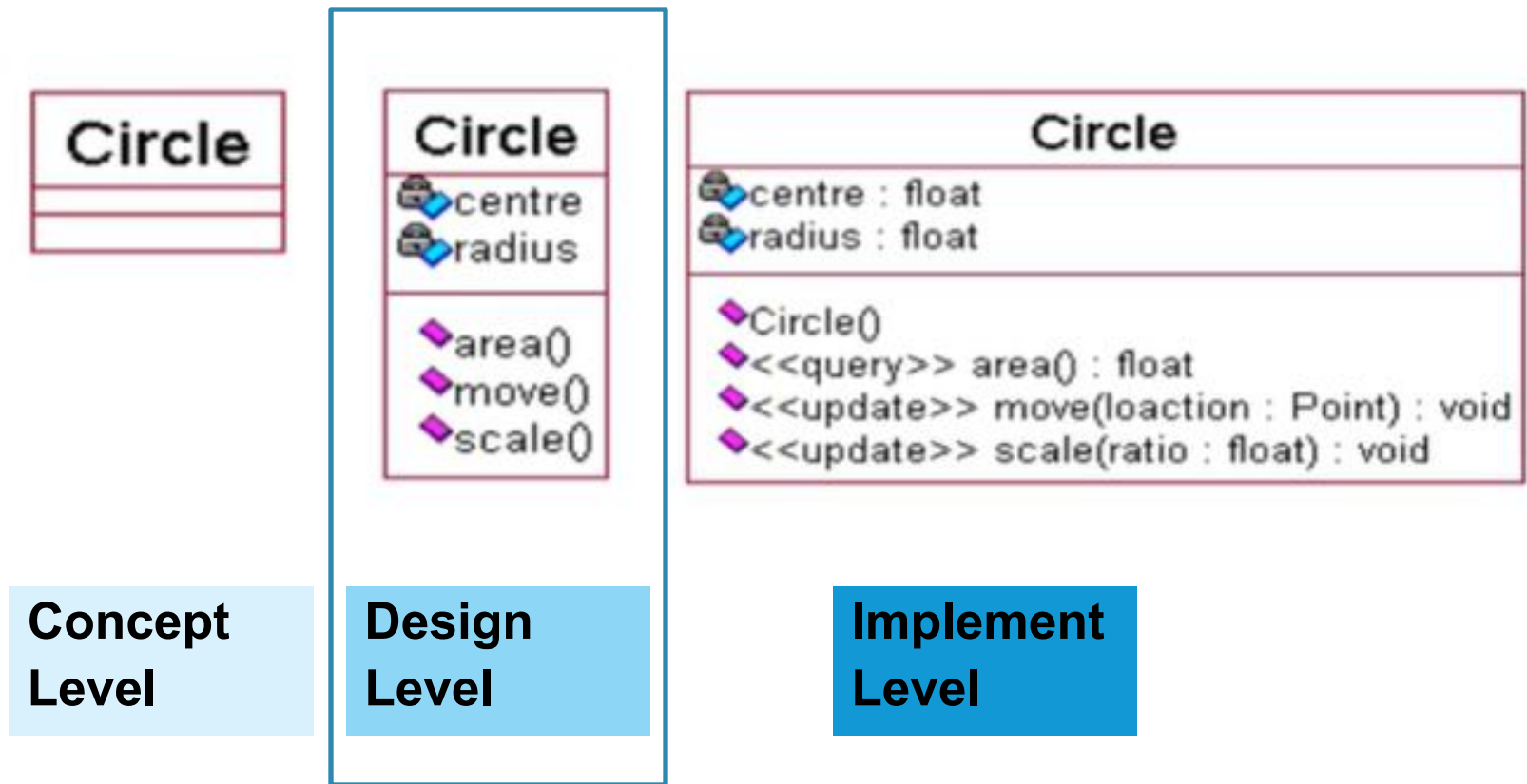
- **Association:** two classes in a model need to communicate with each other.
- **Aggregation** implies a relationship where **the child can exist independently of the parent**. Example: Class (parent) and Student (child). Delete the Class and the students still exist.
- **Composition** implies a relationship where **the child cannot exist independent of the parent**. Example: House (parent) and Room (child). Rooms don't exist separate to a House.

# 10.5 UML Class diagram



# 10.5 UML Class diagram

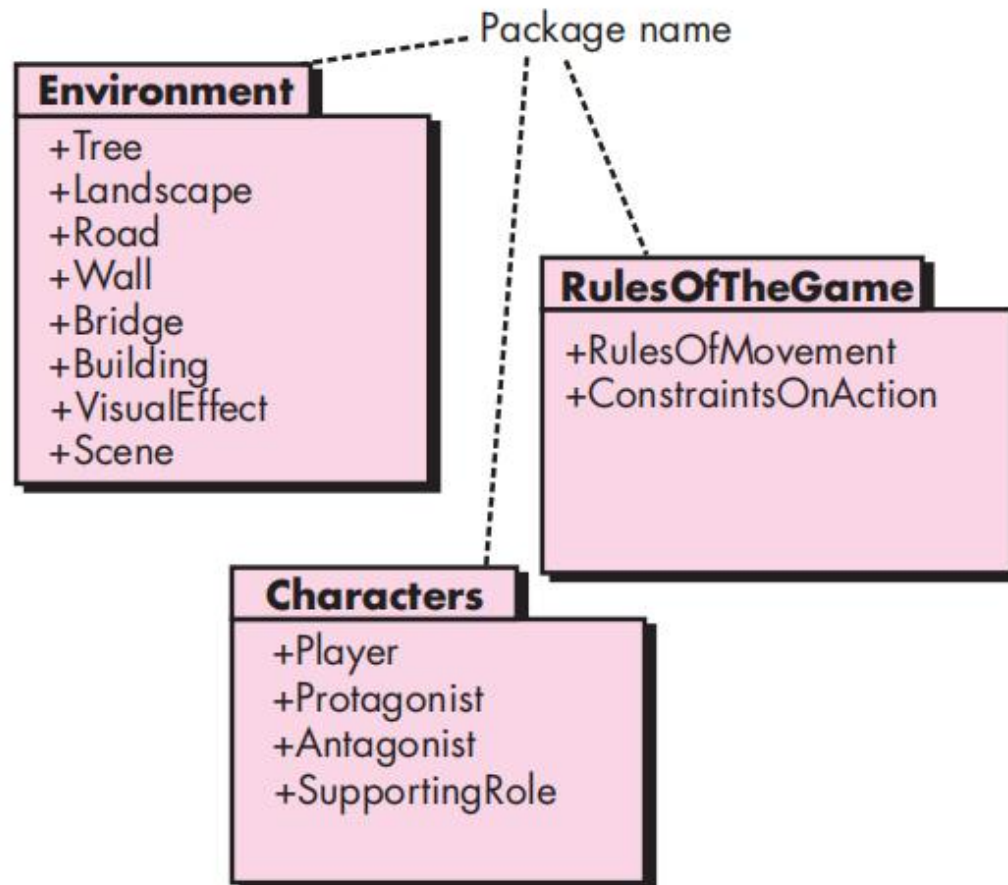
- Different level class diagram in development





# 10.6 Analysis Packages

- packages : class group



# Summary

- Class analysis (attribute + operation)
- UML class diagram

## Question

- **Tools: from source to class structure or diagram**



**THE END**