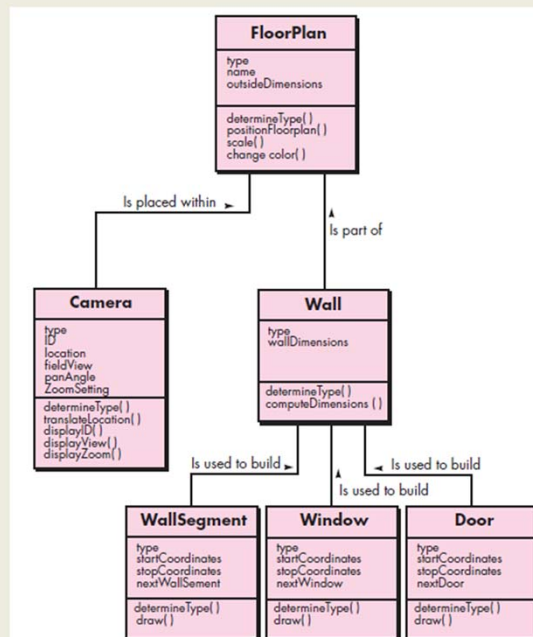


Requirements Modeling:
Scenarios, information, and analysis classes
(V)

Example: SafeHome (FloorPlan class)

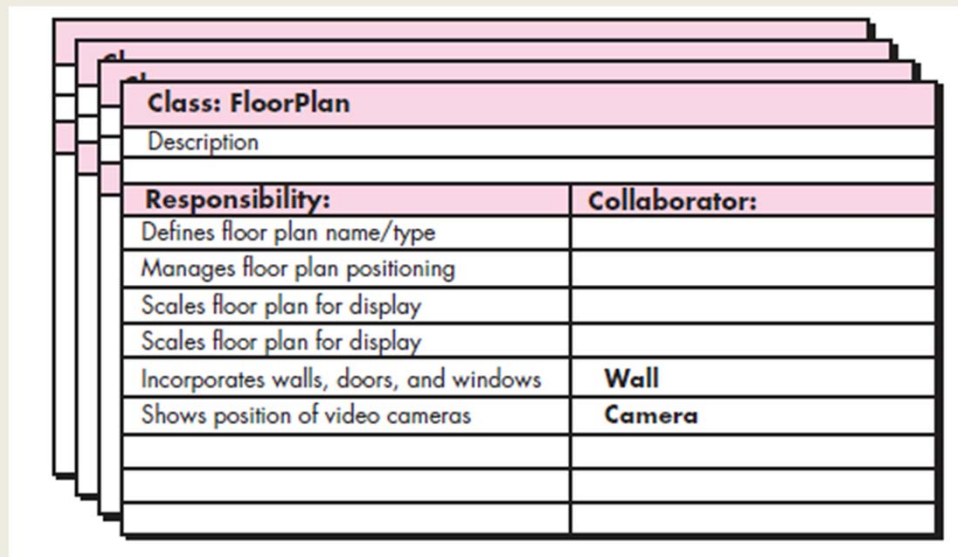


In the SafeHome, ACS-DCV use case.

As you can see, there are not only classes but also the communication between them. This communication is realized through passing messages

Between related classes. This communication can be modeled using the CRC model.

CRC Modeling



The image shows a stack of index cards. The top card is titled 'Class: FloorPlan' and contains a table with responsibilities and 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

A simple CRC index card for the **FloorPlan** class is illustrated

Example: SafeHome (security)

- the **ControlPanel** class must determine whether any sensors are open.
 - A responsibility named *determine-sensor-status()* is identified for this class
- If sensors are open, **ControlPanel** must set a status attribute to “not ready.”
- Sensor information can be acquired from each **Sensor** object.

the responsibility *determine-sensor-status()* can be fulfilled only if **ControlPanel** works in collaboration with **Sensor**.

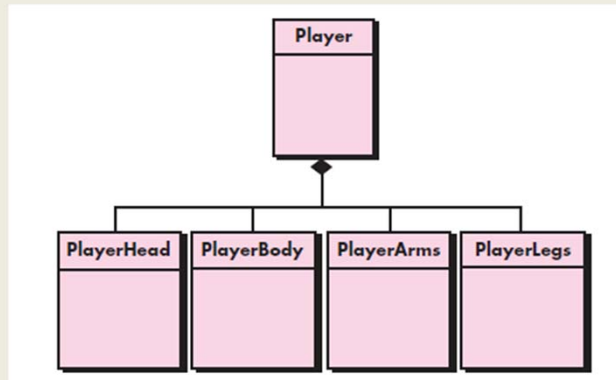
CRC: Collaboration

- 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*
- Collaborations are identified by *determining whether a class can fulfill each responsibility itself*
- three different generic relationships between classes:
 - the *is-part-of* relationship
 - the *has-knowledge-of* relationship
 - the *depends-upon* relationship

Examining these different generic relationships between classes helps with the identification of collaborators

Is-part-of relationship

- All classes that are part of an **aggregate class** are connected to the aggregate class
- Example



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 shown in Figure 6.12.

Has-knowledge-of relationship

- One class must acquire information from another class.
- Example
 - The *determine-sensor-status()* responsibility in ControlPanel class, which has knowledge of the Sensor class

Depends-upon relationship

- Two classes have a dependency that is not achieved by *has-knowledge-of* or *is-part-of*.
- Example
 - **PlayerHead** *depends-upon* **PlayerBody**
 - **PlayerHead** must always be connected to **PlayerBody** yet each object could exist without direct knowledge of the other
 - An attribute of the **PlayerHead** object called center-position is determined from the center position of **PlayerBody**. This information is obtained via a third object, **Player**, that acquires it from **PlayerBody**.

Is-part-of and has-knowledge-of also imply 'dependency'; other kinds of dependency are categorized as 'depends-upon'

Review a CRC model

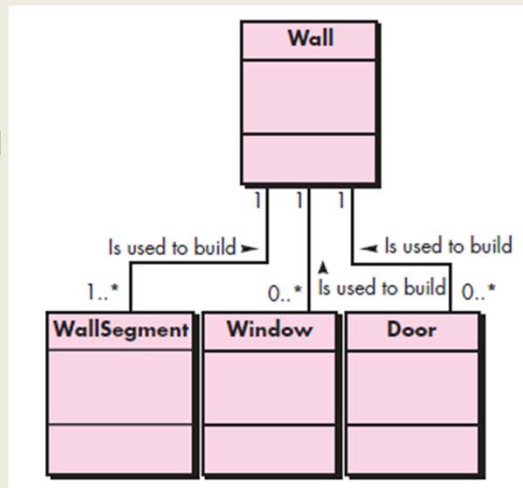
1. participants are given a subset of the CRC model **index cards**.
2. All **use-case** scenarios (and corresponding use-case diagrams) are organized into categories.
3. Review leader reads the use-case deliberately, passing a token to the person holding the corresponding class index card when coming to a named object
4. the holder of the class card is asked to describe the **responsibilities** noted on the card, and the group determines whether one (or more) of the responsibilities satisfies the use-case requirement.
5. If the responsibilities and collaborations noted on the index cards cannot accommodate the use-case, **modifications** are made to the cards.

This is how stakeholders would review the CRC model once developed.

The index card contains a list of responsibilities and the corresponding collaborations that enable the responsibilities to be fulfilled; the collaborator class name is recorded on the CRC model index card next to the responsibility.

Class-based modeling: analysis-class relationships

- Associations
 - Two analysis classes are related to one another in some fashion: represented as *associations* in UML
 - Can be further refined by indicating *multiplicity*



An *association* between two classes means that there is a structural relationship between them.

The class **Wall** is associated with three classes that allow a wall to be constructed, **WallSegment**, **Window**, and **Door**.

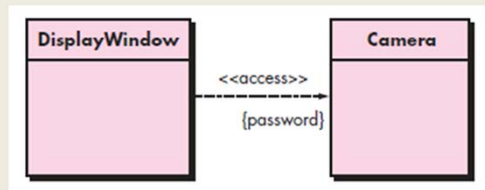
In some cases, an association may be further defined by indicating *multiplicity*.

a **Wall** object is constructed from one or more **WallSegment** objects. In addition, the **Wall** object may contain 0 or more **Window** objects and 0

or more **Door** objects. These multiplicity constraints are illustrated in Figure 6.13, where "one or more" is represented using 1..*, and "0 or more" by 0..*. In UML, the asterisk indicates an unlimited upper bound on the range.

Class-based modeling: analysis-class relationships

- Dependencies
 - a **client-server relationship** between two analysis classes
 - **client-class** depends on the **server-class** in some way
 - defined by a **stereotype**
 - an “extensibility mechanism” within UML that allows you to *define a special modeling element* whose semantics are custom defined.



- <<access>> implies that the use of the camera output is controlled by a special password.

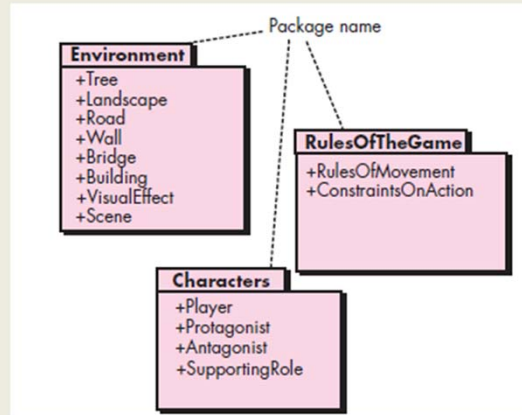
In a use case written for surveillance (not shown), you learn that a special password must be provided in order to view specific camera locations. One way to achieve this is to have **Camera** request a password and then grant permission to the **DisplayWindow** to produce the video display.

Class-based modeling

- Represent objects, operations, relationships, and collaborations
- Elements
 - Classes / Objects
 - Attributes
 - Operations
 - CRC model
 - Collaboration diagrams
 - packages

Analysis package

- A **package** is used to assemble a collection of related classes
 - Various elements of the analysis model (e.g., use-cases, analysis classes) are categorized in a manner that packages them as a **grouping**



Symbols: **+** (visible from external packages), **-** (hidden to outside) , **#** (accessible for packages within a specific package)

In a video game, Some focus on the game environment—the visual scenes that the user sees as the game is played;

Others focus on the characters within the game, describing their physical features, actions, and constraints.

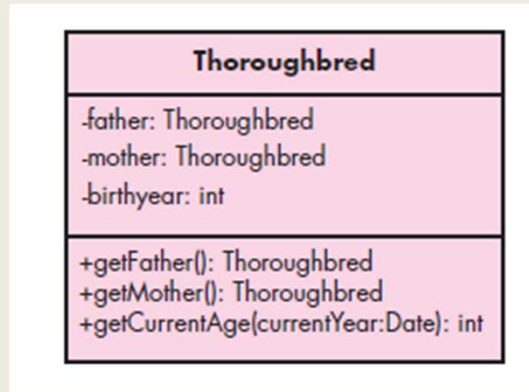
Still others describe the rules of the game—how a player navigates through the environment

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 packages contained within a given package.

Class-based modeling in UML

- **Class diagram**: representing analysis classes in UML
 - Elements: attribute, operation
 - Visibility: public(+), package(~), private(-), protected(#).



Shows the structure of your software

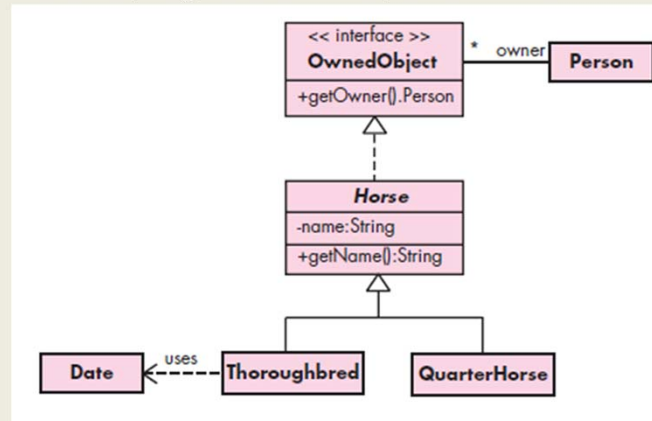
References: <http://msdn.microsoft.com/en-us/library/dd409437.aspx>

a simple example of a **Thoroughbred** class that models thoroughbred horses. Each attribute can have a name, a type, and a level of visibility. The type and visibility are **optional**

can also specify that an attribute is a static or class attribute by underlining it. Each operation can also be displayed with a level of visibility, parameters with names and types, and a return type.

Class-based modeling in UML

- **Class diagram**: representing analysis classes in UML
 - Special types: abstract class, interface
 - Relationships: generalization, realization



An abstract class or abstract method is indicated by the use of italics for the name in the class diagram.

An interface is indicated by adding the phrase “«interface»” (called a *stereotype*) above the name. An interface can also be represented graphically by a hollow circle.

The arrow points from the subclass to the superclass. In UML, such a relationship is called a **generalization**.

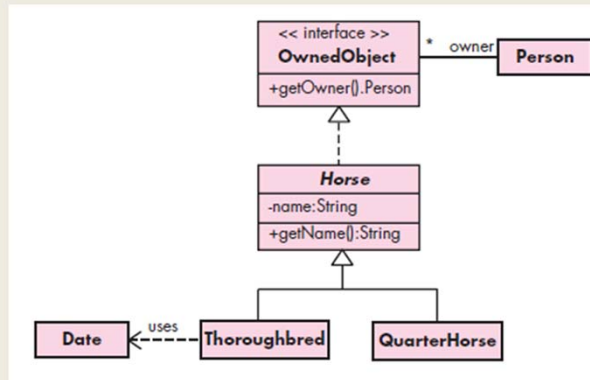
An arrow with a dashed line for the arrow shaft indicates implementation of an interface. In UML, such a relationship is called a **realization**.

a fourth section at the bottom of the class box can be used to list the responsibilities of the class. This section is particularly useful when transitioning from CRC cards (Chapter 6) to class diagrams

Class-based modeling in UML

- **Class diagram**: representing analysis classes in UML
 - Relationships
 - Association: unidirectional, bidirectional (**navigability**)

“An attribute of a class is very much the same thing as an association of the class with the class type of the attribute.”



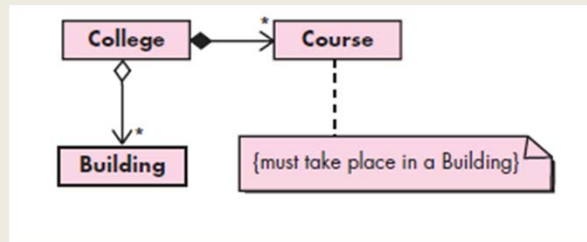
An *association* between two classes means that there is a structural relationship between them

Associations are represented by solid lines, there is an association between **OwnedObject** and **Person** in which the **Person** plays the role of owner. Arrows on either or both ends of an association line indicate **navigability**. An association with no arrows usually indicates a two-way association, but it could also just mean that the navigability is not important and so was left off.

Also, each end of the association line can have a multiplicity value displayed. The *multiplicity* of one end of an association means the number of objects of that class associated with the other class. A multiplicity is specified by a nonnegative integer or by a range of integers. A multiplicity specified by “0..1” means that there are 0 or 1 objects on that end of the association.

Class-based modeling in UML

- **Class diagram**: representing analysis classes in UML
 - Relationships
 - Association: unidirectional, bidirectional
 - Special type: Aggregation,
 - » Special type: composition



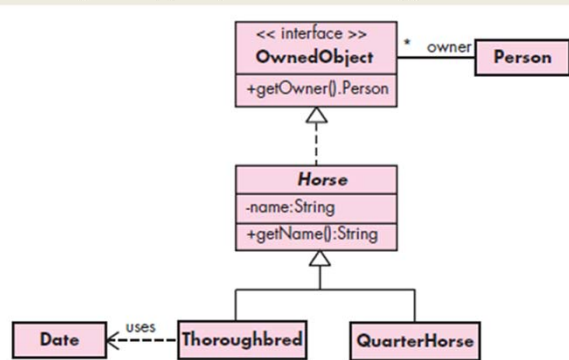
An *aggregation* is a special kind of association indicated by a hollow diamond, indicating a “whole/part” relationship.

A *composition* is an **aggregation** indicating strong ownership of the parts, in a composition, the parts live and die with the owner because they have no role in the software system independent of the owner.

Another common element of a class diagram is a **note**, which is represented by a box with a dog-eared corner and is connected to other icons by a dashed line. It can have arbitrary content (text and graphics) and is similar to comments in programming languages.

Class-based modeling in UML

- **Class diagram**: representing analysis classes in UML
 - Relationships
 - Dependency
 - a dependency exists between two elements if changes to the definition of one element (the supplier) may cause changes to the other (the client)



A *dependency* relationship represents another connection between classes and is indicated by a dashed line (with optional arrows at the ends and with optional labels). One class depends on another if changes to the second class might require changes to the first class.

An association from one class to another automatically indicates a dependency.

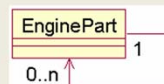
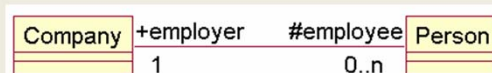
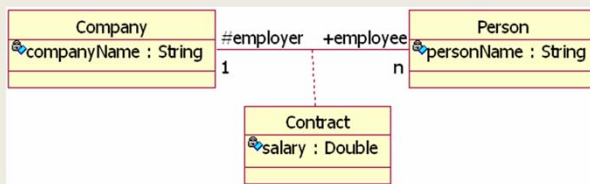
the **Thoroughbred** class uses the **Date** class whenever its `getCurrentAge()` method is invoked, and so the dependency is labeled "uses."

No dashed line is needed between classes if there is already an association between them.

An association almost always implies that one object has the other object as a field/property/attribute (terminology differs). A dependency typically (but not always) implies that an object accepts another object as a method parameter, instantiates, or uses another object. A dependency is very much implied by an association.

UML class diagram: more examples

- Association (uni/bi-directional)



```
Public class A
{
    public B b;

    public f1()
    {
    }
}
```

```
Public class B
{
    public f2()
    {
    }
}
```



No aggregation, composition for n-ary association

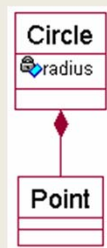
An association might also connect a class with itself, using a loop. Such an association indicates the connection of an object of the class with other objects of the same class.

UML class diagram: more examples

- Aggregation



- Composition



- Aggregation

- Is a form of association
- That specifies a whole-part relationship

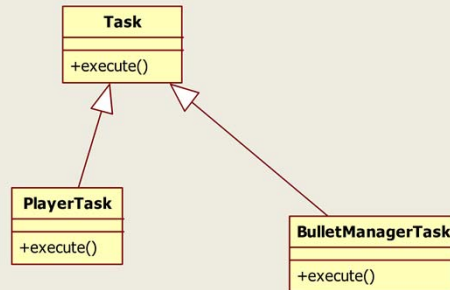
- Composition

- Is a form of aggregation
- With strong ownership and coincident lifetime of parts by the whole

Differences? Ownership; Lifetime.

UML class diagram: more examples

- Generalization



- Interface



- Realization

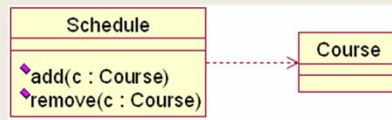


Generalization: "A-kind-of" relationship (A 'is a' B)

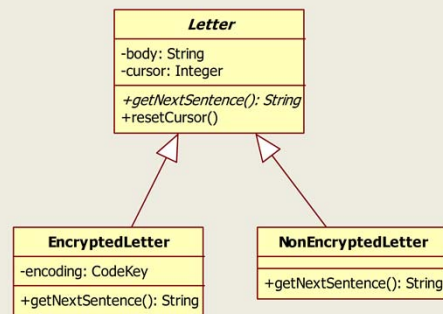
Interface: No attributes, only operations

UML class diagram: more examples

- Dependency

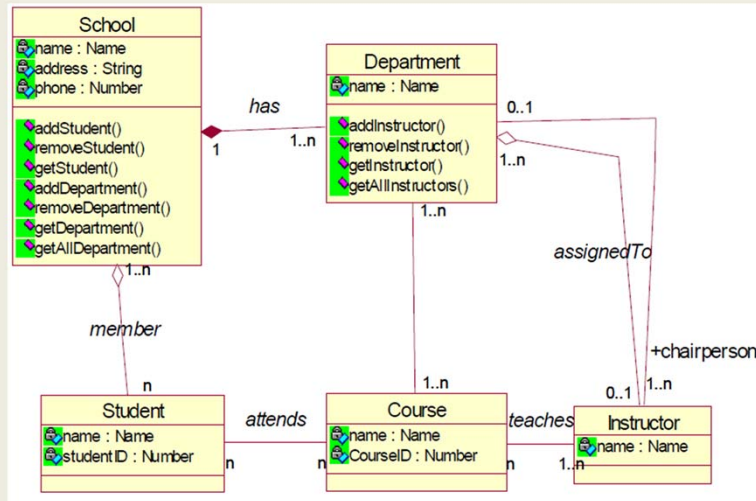


- Abstract class



UML class diagram: more examples

- Put together

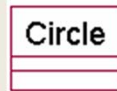


Association with different multiplicity indications;
Aggregation and composition

More on these relationships: https://vaughnvernon.co/?page_id=31

UML class diagram: various detail levels

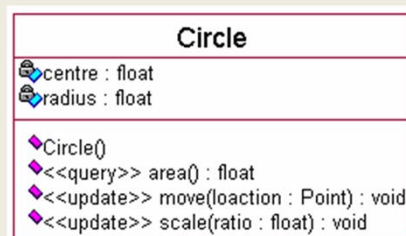
- Conceptual



- Specification



- Implementation



Implementation level class diagram are probably most commonly used. But sometimes specification level is adequate or even better for communication

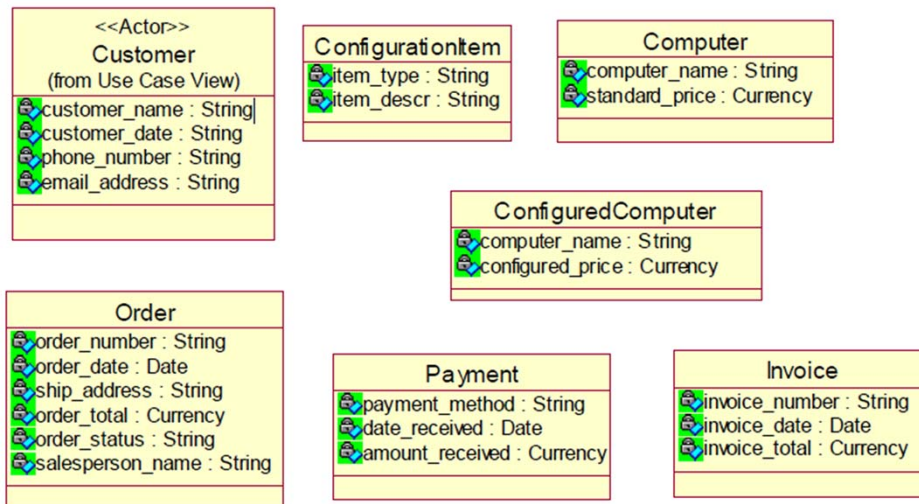
Exercise – requirements (narrative)

- A computer manufacturer provides opportunities for online purchasing through Internet.
- Customers can visit their website and select a computer to buy. Computers are defined in three categories: server, PC, laptop.
- Customers can choose default configurations, or customize their own by selecting configurable parts (memories, hard drives, etc.) from a list.
- For each configuration, the system calculates the price.
- To submit an order, the customer has to provide shipping and billing information, credit card or checks are both accepted.
- Once the order has been submitted, the system sends a confirmation email to the customer with details.
- Customer can check the status of an order before it is delivered.
- The sales team manages the orders that have been submitted. The process includes the following steps: verify the billing information, send the configuration to the warehouse, print receipt, and request shipment.

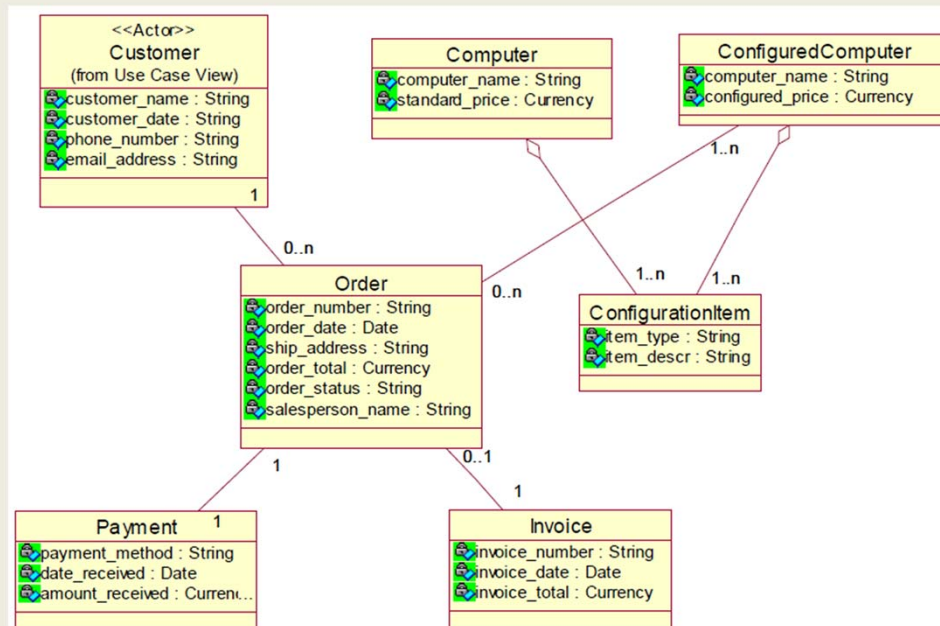
Exercise – requirements (narrative)

- A computer manufacturer provides opportunities for online purchasing through Internet.
- **Customers** can **visit their website and select a computer** to buy. Computers are defined in three categories: server, PC, laptop.
- **Customers** can **choose default configurations, or customize their own** by selecting configurable parts (memories, hard drives, etc.) from a list.
- For each configuration, **the system calculates the price**.
- To **submit an order**, **the customer** has to provide shipping and billing information, credit card or checks are both accepted.
- Once the order has been submitted, **the system sends a confirmation** email to the customer with details.
- **Customer** can **check the status of an order** before it is delivered.
- The **sales team manages the orders** that have been submitted. The process includes the following steps: verify the billing information, send the configuration to the warehouse, print receipt, and request shipment.

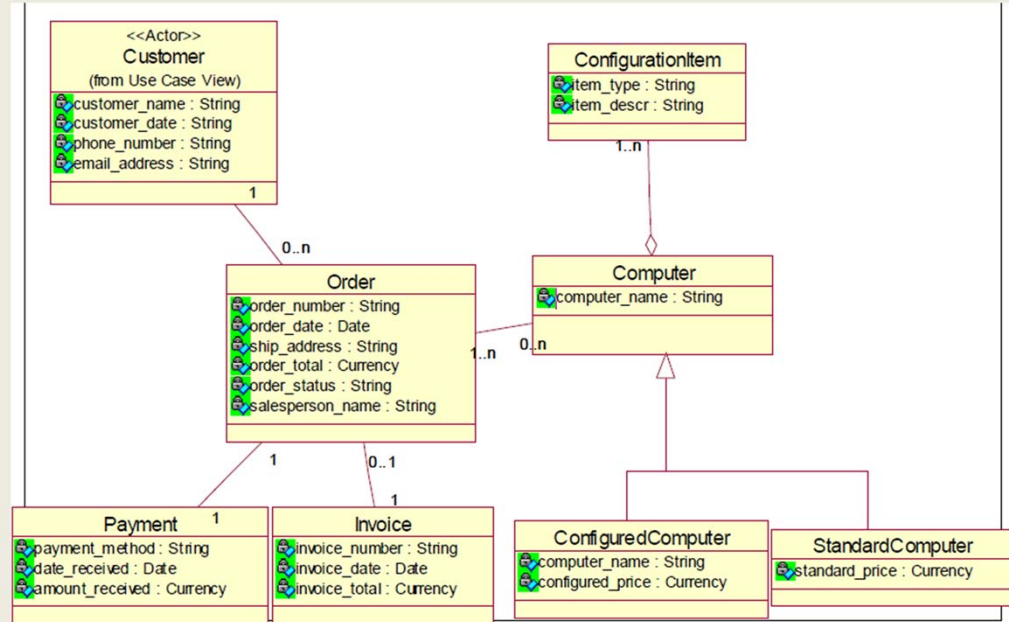
Exercise – analysis classes



Exercise – classes with relationships



Exercise – class relationship refined



Summary

- CRC modeling (II)
 - Class relationship
 - Model review procedure
- Class-based modeling
 - Analysis package
 - class diagrams
 - Class relationship
 - UML notations (see also <http://www.uml-diagrams.org/>)
 - Examples