UNIVERSITAS INDONESIA

Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

TOPIC 5

STRUCTURAL MODELLING

ANALISIS DAN PERANCANGAN SISTEM INFORMASI

CSIM603183

Learning Objectives

1. Able to explain the **rules and components** of **class diagram, CRC Cards, object diagrams, and other techniques**.
2. Able to explain the **process to create class diagram, CRC Cards, object diagrams, and other techniques**.
3. Able to explain the **relationship between structural and functional** model (usecase diagram)
4. Able to validate and verify the structural model.

Session Outline

1. Structural Models
2. Object Identification
3. CRC Cards
4. Class Diagrams
5. Object Diagrams
6. Verifying & Validating the Structural Model

Where are we?

- During **analysis**, analysts create business process and functional models to represent how the business system will **behave** externally (last week – use cases modeling)
- At the same time, analysts need to understand the **information** that is used and created by the business system.
  - We discuss **now** the **structural modeling** of how the objects underlying the behavior modeled in the business process and functional models are organized and presented.

Overview

- Requirements gathering and analysis (Week #4)
  - Deliver System Proposal
    - It's a proposed logical information systems:
      - functional requirements: relates to a process or data
      - non-functional requirements: relates to performance or usability
      - Nonfunctional requirements can **influence functional, structural, and behavioral model**
  - **Deliver precise list of requirements** that can be used as inputs to the rest of analysis for creating **functional, structural, and behavioral model**.

Overview

- List of functional requirements are input to Functional Model with **Use Cases as core building blocks—focus on process** (Week #5)
- A structural or conceptual model **describes the structure of the data that supports the business processes in an organization—focus on data** (Week #6)

1.1 STRUCTURAL MODELS (CONCEPT)

Introduction

- A structural or conceptual model describes the structure of the data that supports the business processes in an organization
- The structure of data used in the system is represented through *CRD cards*, *class diagrams*, and *object diagrams*
- Constructing the structural model is an iterative process involving: *textual analysis, brainstorming objects, role playing, creating the diagrams*, and *incorporating useful patterns*
- *Verifying & Validating the Structural Model* ensure the consistency of the Structural Model
- *Simplifying the class diagram* is often necessary (view)

Introduction

- All object-oriented systems development approaches are use-case driven, architecture-centric, and iterative and incremental. Use cases form the foundation on which the business information system is created.
- From an architecture-centric perspective, structural modeling supports the creation of an internal structural or static view of a business information system in that it shows how the system is structured to support the underlying business processes.

Introduction

- Finally, as with business process and functional modeling, you will find that you will need to not only iterate across the structural models (described in this chapter), but you will also have to iterate across all three architectural views (functional, structural, and behavioral) **to fully capture and represent the requirements for a business information system**.

What is a Structural Model?

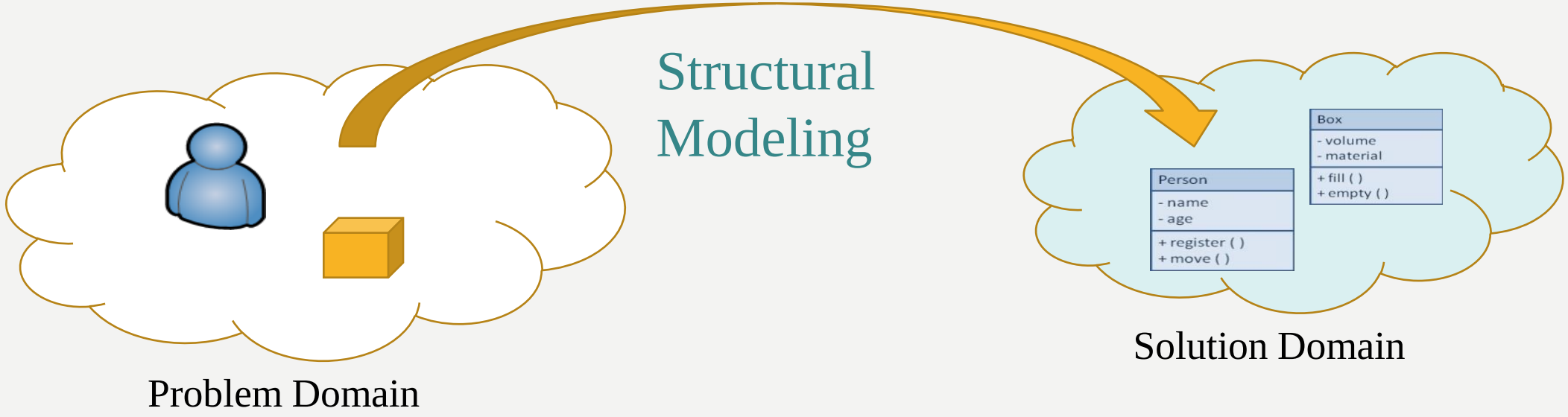
- A ***structural model*** is a formal way of representing the objects that are used and created by a business system**. It illustrates people, places, or things** about which information is captured and how they are related to one another.
- The structural model is drawn using an iterative process in which the model becomes more detailed and less conceptual over time.

What is a Structural Model?

- In **analysis**, analysts draw a ***conceptual model***, which shows the logical organization of the objects without indicating how the objects are stored, created, or manipulated.
- In **design**, analysts evolve the conceptual structural model into a design model that reflects how the objects will be organized in databases and software.
- At this point, the model is checked for redundancy, and the analysts investigate ways to make the objects easy to retrieve.

Structural Model

Main goal: to discover the key data contained in the problem domain and to build a structural model of the objects (static view).



Purpose of Structural Models

- Every time a systems analyst encounters a new problem to solve, the analyst must learn the underlying problem domain. The goal of the analyst is to discover the key objects contained in the problem domain and to build a structural model.
- One of the primary purposes of the structural model is to create a vocabulary that can be used by the analyst and the users to communicate effectively.
  - Structural models represent the things, ideas, or concepts contained in the domain of the problem. They also allow the representation of the relationships among the things, ideas, or concepts.

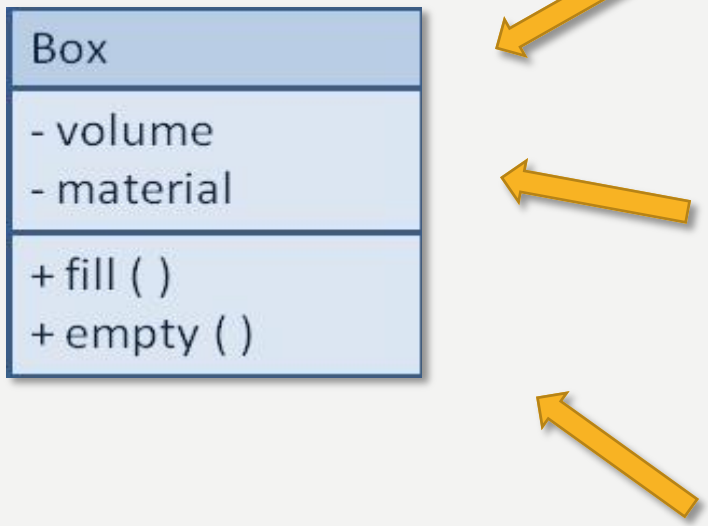
Purpose of Structural Models

- It is **important** to remember that at this stage of development, the structural model **does not represent software components** or **classes in an object-oriented programming language**, even though the structural model does contain analysis **classes**, **attributes**, **operations**, and the **relationships** among the analysis classes.
- The refinement of these initial classes into programming-level objects comes later.
- The structural model at this point should represent the responsibilities of each class and the collaborations among the classes.

Purpose of Structural Models

- Typically, structural models are depicted using **CRC cards**, **class diagrams**, and, in some cases, object diagrams. However, before describing CRC cards, class diagrams, and object diagrams, we describe **the basic elements of structural models: (1) classes, (2) attributes, (3) operations, and (4) relationships.**

Classes, Attributes, & Operations



- **Classes**
  - Templates for instances of people, places, or things
- **Attributes**
  - Properties that describe the state of an instance of a class (an object)
- **Operations**
  - Actions or functions that a class can perform

Classes

- A *class* is a general template that we use to create specific instances, or *objects,* in the problem domain. All objects of a given class are identical in structure and behavior but contain different data in their attributes.
- There are two general kinds of classes of interest during analysis: **concrete** and **abstract**.
  - Normally, when an analyst describes the application domain classes, he or she is referring to concrete classes; that is, *concrete classes* are used to create objects.
  - *Abstract classes* do not actually exist in the real world; they are simply useful abstractions.

Classes

- For example, from an *employee* class and a *customer* class, we may identify a generalization of the two classes and name the abstract class *person*.
- We might not actually instantiate the person class in the system itself, instead creating and using only employees and customers.

Classes

- A second classification of classes is the type of real-world thing that a class represents.
- There are domain classes, user-interface classes, data structure classes, file structure classes, operating environment classes, document classes, and various types of multimedia classes.
- At this point in the development of our evolving system, we are interested only in **domain classes**.
- Later in design and implementation, the other types of classes become more relevant.

Attributes

- An *attribute* of an analysis class represents a piece of information that is relevant to the description of the class within the application domain of the problem being investigated. An attribute contains information the analyst or user feels the system should keep track of.
- For example, a possible relevant attribute of an *employee* class is *employee name*, whereas one that might not be as relevant is *hair color*. Both describe something about an employee, but hair color is probably not all that useful for most business applications.

Attributes

- Finally, only attributes that are primitive or atomic types (i.e., integers, strings, doubles, date, time, Boolean, etc.) should be added. Most complex or compound attributes are really placeholders for relationships between classes (see next slides).

Operations

- The behavior of an analysis class is defined in an ***operation*** or service. **In later phases, the operations are converted to *methods*.**
- Like attributes, only problem domain–specific operations that are relevant to the problem being investigated should be considered.
- For example, it is normally required that classes provide means of creating instances, deleting instances, accessing individual attribute values, setting individual attribute values, accessing individual relationship values, and removing individual relationship values. However, at this point in the development of the evolving system, the analyst should avoid cluttering up the definition of the class with these basic types of operations and focus only on relevant problem domain–specific operations.
- Action that instances/objects can take focus on relevant problem-specific operations (at this point)

Relationships

- Describe how classes relate to one another
- Three basic types in UML
  - Generalization
    - Enables inheritance of attributes and operations
    - Represents relationships that are “a-kind-of”
  - Aggregation
    - Relates parts to wholes or assemblies
    - Represents relationships that are “a-part-of” or “has-parts”
  - Association
    - Miscellaneous relationships between classes
    - Usually a weaker form of aggregation

Generalization (a-kind-of relationship)

- The generalization abstraction enables the analyst to create classes that **inherit** attributes and operations of other classes.
  - A *superclass* contains basic attributes and operations that will be used in several *subclasses*.
  - The subclasses inherit the attributes and operations of their superclass and can also contain attributes and operations that are unique just to them.
  - In this way, the analyst can reduce the redundancy in the class definitions
  - *For example*, a *customer* class and an *employee* class can be generalized into a *person* class by extracting the attributes and operations both have in common and placing them into the new superclass, *person*. Thus, an employee is *a-kind-of* person.

Generalization (specialization & hierarchy)

- The analyst also can use the opposite of generalization: *Specialization* uncovers additional classes by allowing new subclasses to be created from an existing class.
  - For example, an *employee* class can be specialized into a *secretary* class and an *engineer* class.
- Furthermore, generalization relationships between classes can be combined to form generalization *hierarchies*.
  - Based on the previous examples, a *secretary* class and an *engineer* class can be subclasses of an *employee* class, which in turn could be a subclass of a *person* class.
  - This would be read as a secretary and an engineer are a-kind-of employee and a customer and an employee are a-kind-of person.

Generalization (substitutability)

- To ensure that the semantics of the subclasses are maintained, the analyst should apply the principle of *substitutability*.
- By this we mean that the subclass should be capable of substituting for the superclass anywhere that uses the superclass (e.g., anywhere we use the employee superclass, we could also logically use its secretary subclass).
- By focusing on the a-kind-of interpretation of the generalization relationship, the principle of substitutability is applied.

Aggregation (a-part-of / has-parts)

- Generally speaking, all aggregation relationships relate *parts* to *wholes* or *assemblies*.
  - For example, a door is a-part-of a car, an employee is a-part-of a department, or a department is a-part-of an organization.
- Like the generalization relationship, aggregation relationships can be combined into aggregation hierarchies.
  - For example, a piston is a-part-of an engine, and an engine is a-part-of a car.

Aggregation (a-part-of / has-parts)

- Aggregation relationships are *bidirectional*. The flip side of aggregation is *decomposition*. The analyst can use decomposition to uncover parts of a class that should be modeled separately.
  - For example, if a door and an engine are a-part-of a car, then a car has-parts door and engine. The analyst can bounce around between the various parts to uncover new parts. For example, the analyst can ask, What other parts are there to a car? or To which other assemblies can a door belong?

Association (miscellaneous)

- There are other types of relationships that do not fit neatly into a generalization (a-kind-of) or aggregation (a-part-of) framework.
- Technically speaking, these relationships are usually a weaker form of the aggregation relationship.
  - For example, a patient schedules an appointment. It could be argued that a patient is a-part-of an appointment.
- However, there is a clear semantic difference between this type of relationship and one that models the relationship between doors and cars or even workers and unions. Thus, they are simply considered to be *associations* between instances of classes.

1.1 OBJECT IDENTIFICATION

Introduction

- The 4 (four) most common approaches for object identification are
    1. textual analysis,
    2. brainstorming,
    3. common object lists, and
    4. patterns.
- Most analysts use a combination of these techniques to **make sure that no important objects and object attributes, operations, and relationships have been overlooked.**

Textual Analysis

- **Review the use-case diagrams** and examining the text in the use-case descriptions to identify potential objects, attributes, operations, and relationships.
- The **nouns** in the use case suggest possible classes, and the verbs suggest possible operations.
- This figure presents a summary of useful guidelines.
- The textual analysis of use-case descriptions has been criticized as being too simple, but because its primary purpose is to create an initial rough-cut structural model, its simplicity is a major advantage.

Textual Analysis

- A common or improper noun implies a class of objects.
- A proper noun or direct reference implies an instance of a class.
- A collective noun implies a class of objects made up of groups of instances of another class.
- An adjective implies an attribute of an object.
- A doing verb implies an operation.
- A being verb implies a classification relationship between an object and its class.
- A having verb implies an aggregation or association relationship.
- A transitive verb implies an operation.
- An intransitive verb implies an exception.
- A predicate or descriptive verb phrase implies an operation.
- An adverb implies an attribute of a relationship or an operation.

Adapted from: These guidelines are based on Russell J. Abbott, "Program Design by Informal English Descriptions," *Communications of the ACM* 26, no. 11 (1983): 882–894; Peter P-S Chen, "English Sentence Structure and Entity-Relationship Diagrams," *Information Sciences: An International Journal* 29, no. 2–3 (1983): 127–149; Ian Graham, *Migrating to Object Technology* (Reading, MA: Addison Wesley Longman, 1995).

Textual Analysis

- For example, if we applied these rules to the Make Old Patient Appt use case:
  - We can easily identify potential objects for an old patient, doctor, appointment, patient, office, receptionist, name, address, patient information, payment, date, and time.
  - We also can easily identify potential operations, for example, patient contacts office, makes a new appointment, cancels an existing appointment, changes an existing appointment, matches requested appointment times and dates with requested times and dates, and finds current appointment.

Textual Analysis

Use Case Name: Make Old Patient Appt	ID: 2	Importance Level: Low
Primary Actor: Old Patient	Use Case Type: Detail, Essential	

Stakeholders and Interests:
Old Patient – wants to make, change, or cancel an appointment
Doctor – wants to ensure patient's needs are met in a timely manner

Brief Description: This use case describes how we make an appointment as well as changing or canceling an appointment for a previously seen patient.

Trigger: Patient calls and asks for a new appointment or asks to cancel or change an existing appointment.

Type: External

Relationships:

Association: Old Patient
Include:
Extend: Update Patient Information
Generalization: Manage Appointments

Normal Flow of Events:

1. The Patient contacts the office regarding an appointment.
2. The Patient provides the Receptionist with his or her name and address.
3. If the Patient's information has changed
   Execute the Update Patient Information use case.
4. If the Patient's payment arrangements has changed
   Execute the Make Payments Arrangements use case.
5. The Receptionist asks Patient if he or she would like to make a new appointment, cancel an existing appointment, or change an existing appointment.
   If the patient wants to make a new appointment,
   the S-1: new appointment subflow is performed.
   If the patient wants to cancel an existing appointment,
   the S-2: cancel appointment subflow is performed.
   If the patient wants to change an existing appointment,
   the S-3: change appointment subflow is performed.
6. The Receptionist provides the results of the transaction to the Patient.

SubFlows:

S-1: New Appointment
1. The Receptionist asks the Patient for possible appointment times.
2. The Receptionist matches the Patient's desired appointment times with available dates and times and schedules the new appointment.

S-2: Cancel Appointment
1. The Receptionist asks the Patient for the old appointment time.
2. The Receptionist finds the current appointment in the appointment file and cancels it.

S-3: Change Appointment
1. The Receptionist performs the S-2: cancel appointment subflow.
2. The Receptionist performs the S-1: new appointment subflow.

Alternate/Exceptional Flows:

S-1, 2a1: The Receptionist proposes some alternative appointment times based on what is available in the appointment schedule.
S-1, 2a2: The Patient chooses one of the proposed times or decides not to make an appointment.

Brainstorming

- Essentially, in this context, brainstorming is a process that a set of individuals sitting around a table suggest potential classes that could be useful for the problem under consideration.
- Typically, a brainstorming session is kicked off by a facilitator who asks the set of individuals to address a specific question or statement that frames the session.
  - For example, using the appointment problem described previously, the facilitator could ask the development team and users to think about their experiences of making appointments and to identify candidate classes based on their past experiences.

Brainstorming

- Notice that this approach **does not** use the functional models developed earlier.
  - It simply asks the participants to identify the objects with which they have interacted. For example, a potential set of objects that come to mind are doctors, nurses, receptionists, appointment, illness, treatment, prescriptions, insurance card, and medical records.
  - Once a sufficient number of candidate objects have been identified, the participants should discuss and select which of the candidate objects should be considered further.
  - **Once these have been identified, further brainstorming can take place to identify potential attributes, operations, and relationships for each of the identified objects**.

Principles to Guide a Brainstorming Session by Bellin and Simone

- First, all suggestions should be taken seriously. At this point in the development of the system, it is much better to have to delete something later than to accidentally leave something critical out.
- Second, all participants should begin thinking fast and furiously. After all ideas are out on the proverbial table, then the participants can be encouraged to ponder the candidate classes they have identified.

Principles to Guide a Brainstorming Session by Bellin and Simone

- Third, the facilitator must manage the fast and furious thinking process. Otherwise, the process will be chaotic. Furthermore, the facilitator should ensure that all participants are involved and that a few participants do not dominate the process. To get the most complete view of the problem, we suggest using a round-robin approach wherein participants take turns suggesting candidate classes. Another approach is to use an electronic brainstorming tool that supports anonymity.
- Fourth, the facilitator can use humor to break the ice so that all participants can feel comfortable in making suggestions
- Seems having similarity with JAD?

Common Object Lists

- A *common object list* is simply a list of objects common to the business domain of the system.
- Several categories of objects have been found to help the analyst in creating the list, such as physical or tangible things, incidents, roles, and interactions.
  - Analysts should first look for *physical*, or *tangible, things* in the business domain. These could include books, desks, chairs, and office equipment. Normally, these types of objects are the easiest to identify.
  - *Incidents* are events that occur in the business domain, such as meetings, flights, performances, or accidents.

Common Object Lists

  - Reviewing the use cases can readily identify the *roles* that the people play in the problem, such as doctor, nurse, patient, or receptionist.
  - Typically, an *interaction* is a transaction that takes place in the business domain, such as a sales transaction.
- Other types of objects that can be identified including places, containers, organizations, business records, catalogs, and policies

Patterns

- The idea of using patterns is a relatively new area in object-oriented systems development. There have been many definitions of exactly what a pattern is.
- From our perspective, a *pattern* is simply a useful group of collaborating classes that provide a solution to a commonly occurring problem. Because patterns provide a solution to commonly occurring problems, they are reusable.
- According to Alexander and his colleagues, it is possible to make very sophisticated buildings by stringing together commonly found patterns, rather than creating entirely *new* concepts and designs.
- In a similar manner, it is possible to put together commonly found object-oriented patterns to form *elegant* object-oriented information systems.

Patterns

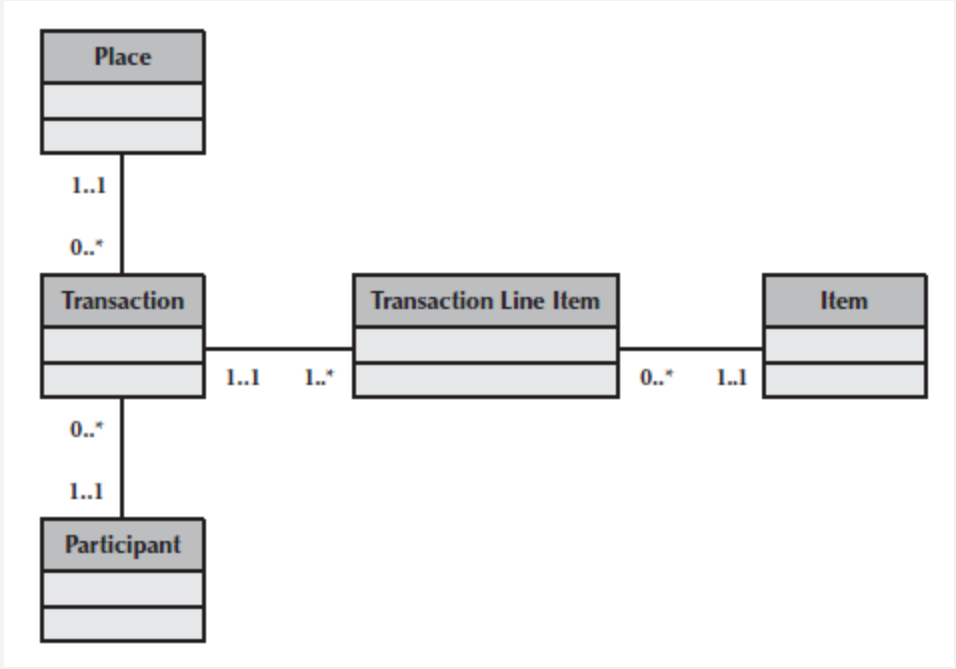
- For example, many business transactions involve the same types of objects and interactions. Virtually all transactions would require a transaction class, a transaction line item class, an item class, a location class, and a participant class. By reusing these existing patterns of classes, we can more quickly and more completely define the system than if we start with a blank piece of paper.
- If we are developing a business information system in one of these business domains, then the patterns developed for that domain may be a very useful starting point in identifying needed classes and their attributes, operations, and relationships.

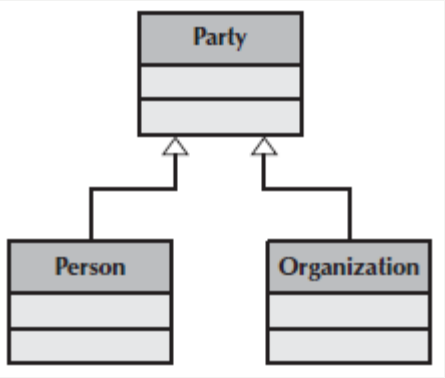
Useful Patterns

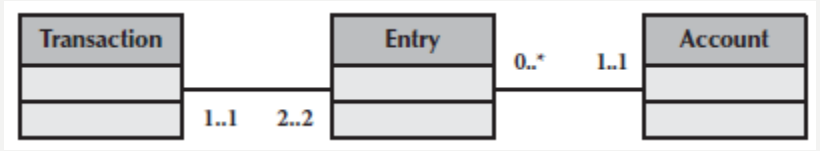
Business Domains	Sources of Patterns
Accounting	3, 4
Actor-Role	2
Assembly-Part	1
Container-Content	1
Contract	2, 4
Document	2, 4
Employment	2, 4
Financial Derivative Contracts	3
Geographic Location	2, 4
Group-Member	1
Interaction	1
Material Requirements Planning	4
Organization and Party	2, 3
Plan	1, 3
Process Manufacturing	4
Trading	3
Transactions	1, 4

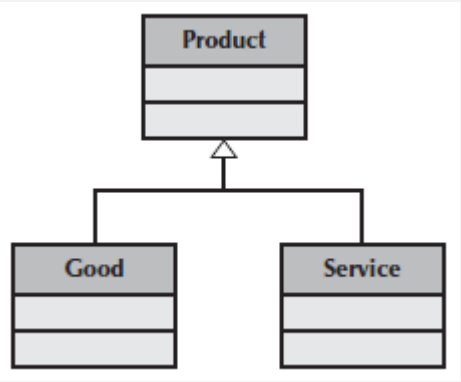
1. Peter Coad, David North, and Mark Mayfield, *Object Models: Strategies, Patterns, and Applications*, 2nd Ed. (Englewood Cliffs, NJ: Prentice Hall, 1997).
2. Hans-Erik Eriksson and Magnus Penker, *Business Modeling with UML: Business Patterns at Work* (New York: Wiley, 2000).
3. Martin Fowler, *Analysis Patterns: Reusable Object Models* (Reading, MA: Addison-Wesley, 1997).
4. David C. Hay, *Data Model Patterns: Conventions of Thought* (New York, NY, Dorset House, 1996).

Samples of Pattern

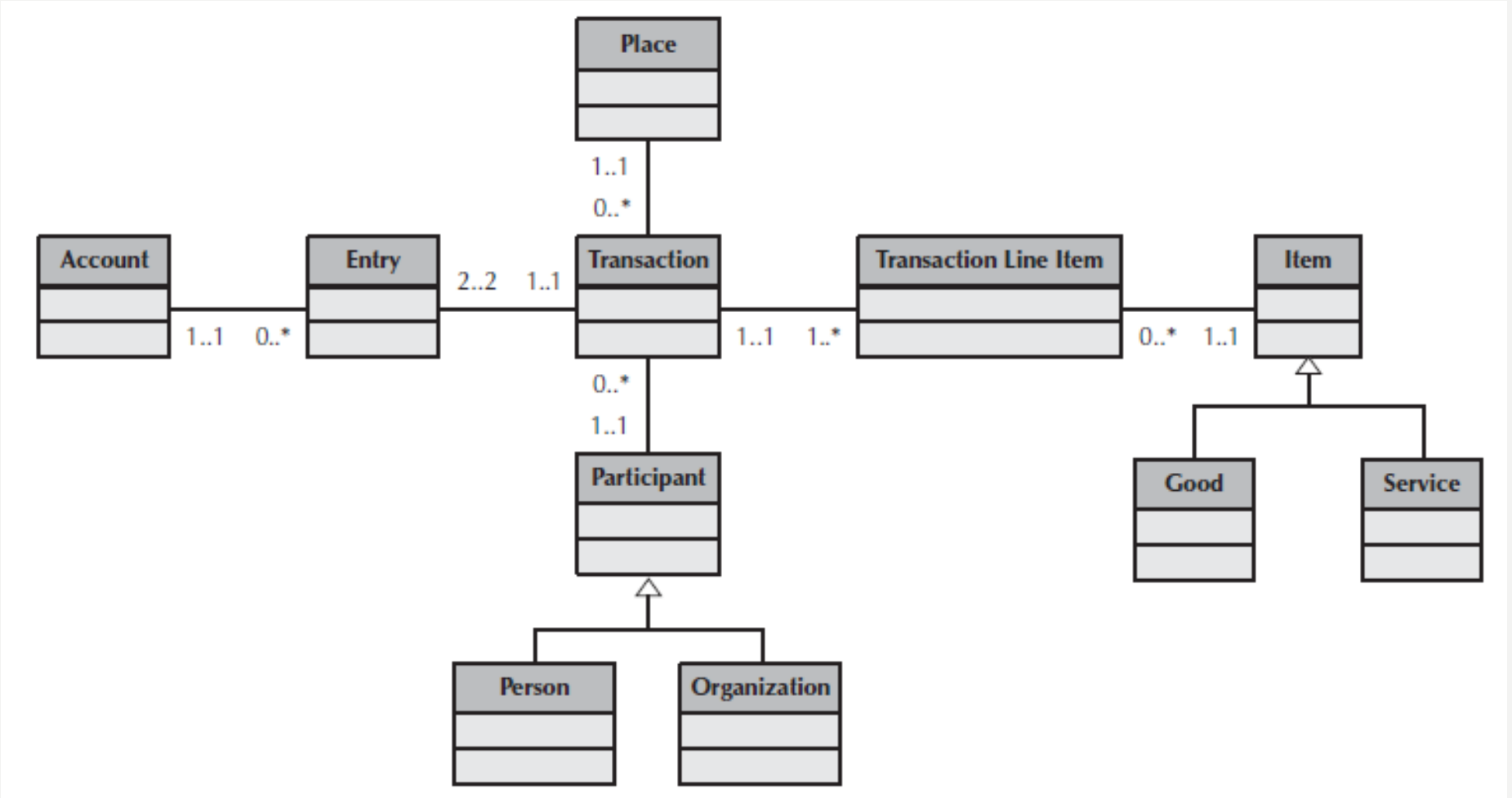








Samples of Pattern (Integration)



1.2 CLASS-RESPONSIBILITY-COLLABORATION (CRC) CARDS

Introduction

- In addition to the object identification approaches described earlier (textual analysis, brainstorming, common object lists, and patterns), CRC cards can be used in a role-playing exercise that has been shown to be useful in discovering additional objects, attributes, relationships, and operations.

Responsibilities

- *Responsibilities* of a class can be broken into two separate types: *knowing* and *doing*.
  - *Knowing responsibilities* are those things that an instance of a class must be capable of knowing.
    - An instance of a class typically knows the values of its attributes and its relationships.
  - *Doing responsibilities* are those things that an instance of a class must be capable of doing.
    - In this case, an instance of a class can execute its operations or it can request a second instance, which it knows about, to execute one of its operations on behalf of the first instance

Collaborations

- The structural model describes the objects necessary to support the business processes modeled by the use cases. Most use cases involve a set of several classes, not just one class.
- These classes form *collaborations*. Collaborations allow the analyst to think in terms of clients, servers, and contracts.
  - A *client* object is an instance of a class that sends a request to an instance of another class for an operation to be executed.
  - A *server* object is the instance that receives the request from the client object.
  - A *contract* formalizes the interactions between the client and server objects.
- Collaboration
  - Objects working together to service a request

CRC (Class-Responsibility- Collaboration)

- The idea of class responsibilities and client–server–contract collaborations can be used to help identify the **classes**, along with the attributes, operations, and relationships, involved with a use case.
- *Anthropomorphism—pretending that the classes have human characteristics.*
- Members of the development team can either ask questions of themselves or be asked questions by other members of the team. Typically the questions asked are of the form:
  - Who or what are you?
  - What do you know?
  - What can you do?
- The answers to the questions are then used to add detail to the evolving CRC cards.

Example of Anthropomorphism

- For example, in the appointment problem, a member of the team can pretend that he or she is an appointment. In this case, the appointment would answer that he or she knows about the doctor and patient who participate in the appointment and they would know the date and time of the appointment.
- Furthermore, an appointment would have to know how to create itself, delete itself, and to possibly change different aspects of itself. In some cases, this approach will uncover additional objects that have to be added to the evolving structural model.

A CRC Card

Front:

Class Name: Old Patient	**ID:** 3	**Type:** Concrete, Domain
Description: An individual who needs to receive or has received medical attention		**Associated Use Cases:** 2

Responsibilities	**Collaborators**
Make appointment	Appointment
Calculate last visit	
Change status	
Provide medical history	Medical history

Back:

Attributes:

Amount (double)	
Insurance carrier (text)	

Relationships:

Generalization (a-kind-of): Person

Aggregation (has-parts): Medical History

Other Associations: Appointment

Role-Playing CRC Cards with Use Cases

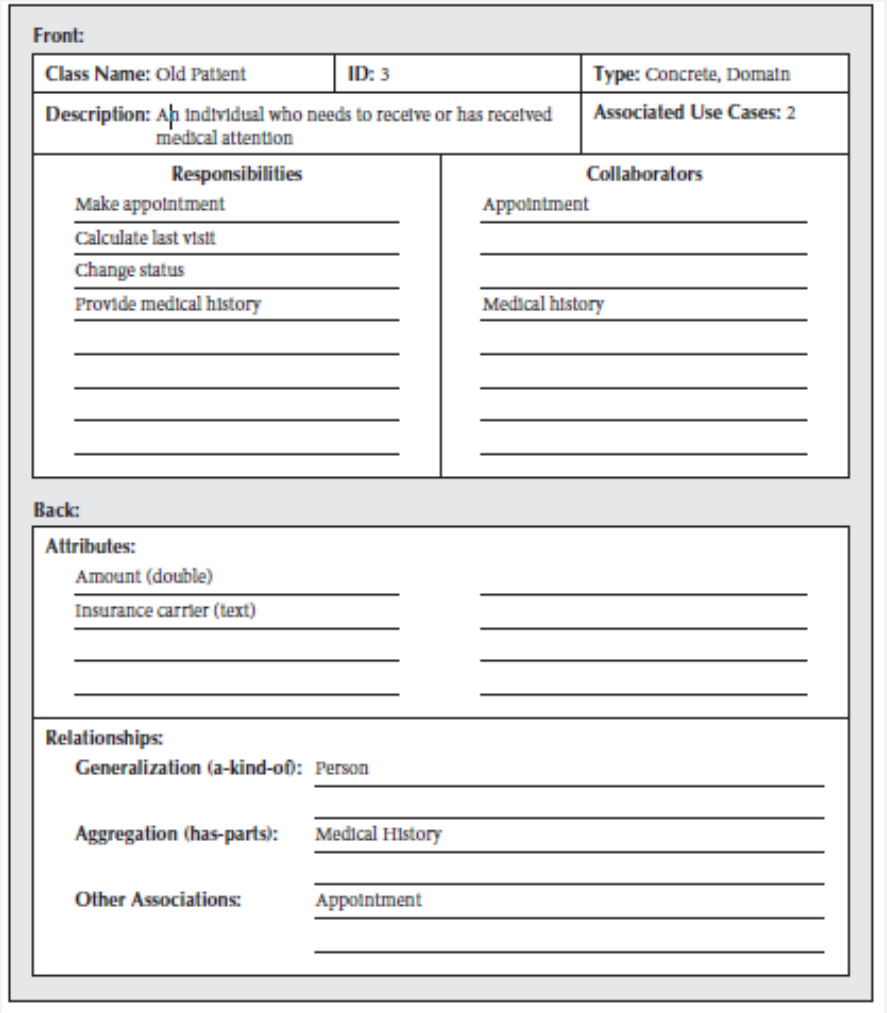
- Role-playing is very useful in testing the fidelity of the evolving structural model
- Technically speaking, the members of the team perform the different steps associated with *a specific scenario of a use case*.
- Consists of 4 steps:
  1. Review Use Cases
  2. Identify Relevant Actors and Objects
  3. Role-Play Scenario
  4. Repeat Steps 1 to 3

Role-Playing CRC Cards with Use Cases

- This allows the team to pick a specific use case to role-play.
- Even though it is tempting to try to complete as many use cases as possible in a short time, the team should not choose the easiest use cases first.
- Instead, at this point in the development of the system, the team should choose the use case that is the *most important*, the *most complex*, or the *least understood*.

Role-Playing CRC Cards with Use Cases – Second Step – Identify Relevant Actors and Objects

- Each role is associated with either an actor or an object. To choose the relevant objects, the team *reviews each of the CRC cards and picks the ones that are associated with the chosen use case*.
- For example, we see that the CRC card that represents the **Old Patient class** is associated with Use Case number 2.



Front:

Class Name: Old Patient	**ID:** 3	**Type:** Concrete, Domain
Description: An individual who needs to receive or has received medical attention		**Associated Use Cases:** 2

Responsibilities	**Collaborators**
Make appointment	Appointment
Calculate last visit	
Change status	
Provide medical history	Medical history

Back:

Attributes:

Amount (double)	
Insurance carrier (text)	

Relationships:

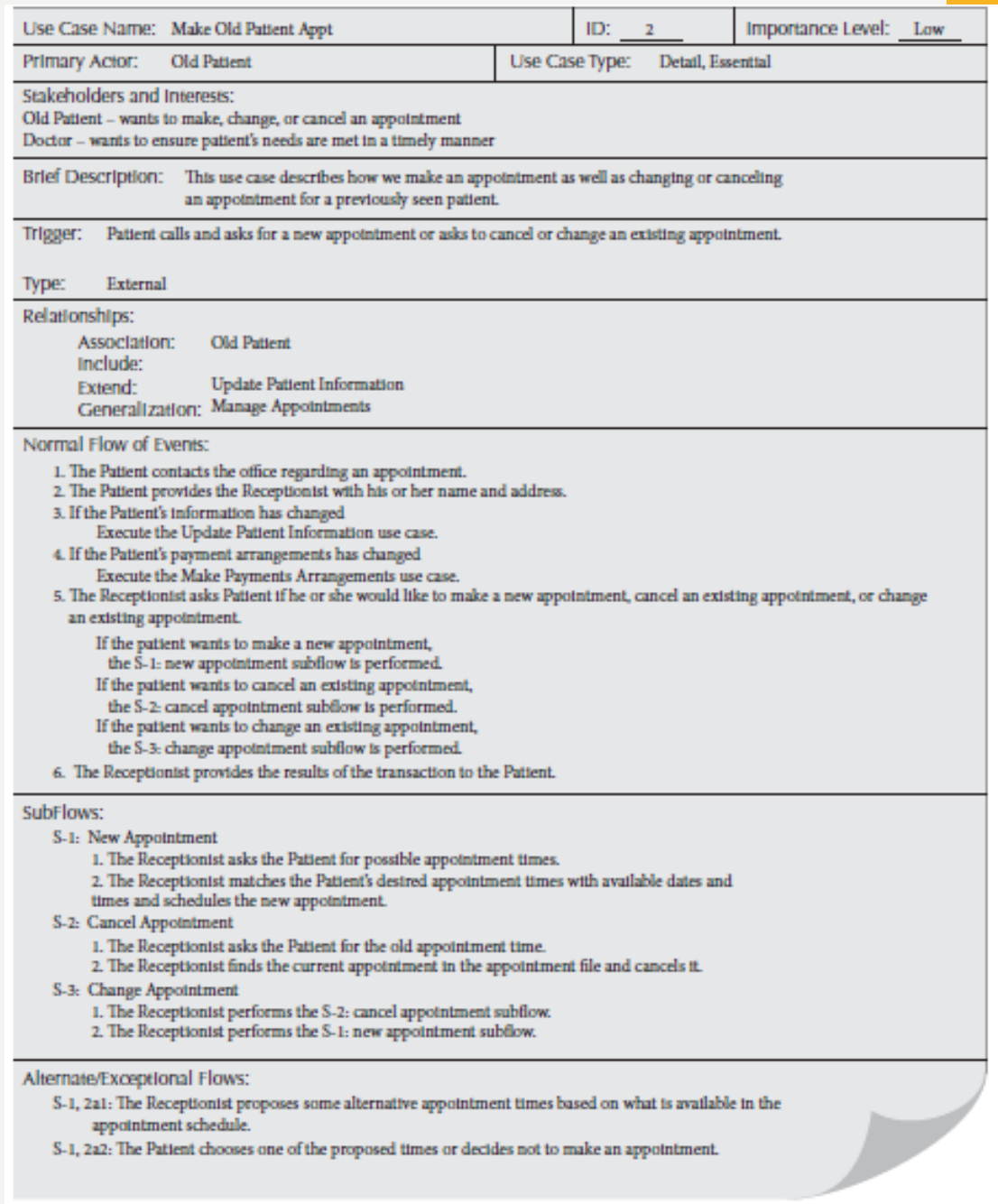
Generalization (a-kind-of): Person

Aggregation (has-parts): Medical History

Other Associations: Appointment

Role-Playing CRC Cards with Use Cases -- Second Step – Identify Relevant Actors and Objects

- So if we were going to role-play the **Make Old Patient Appt use case**, we would need to include the Old Patient CRC card.
- By reviewing the use-case description, we can easily identify the Old Patient and Doctor actors (see Primary Actor and Stakeholders section of the use case description).
- By reading the event section of the use-case description, we identify the internal actor role of *Receptionist*.
- After identifying all of the relevant roles, we assign each one to a different member of the team.



Use Case Name: Make Old Patient Appt | ID: 2 | Importance Level: Low

Primary Actor: Old Patient | Use Case Type: Detail, Essential

Stakeholders and Interests:
Old Patient – wants to make, change, or cancel an appointment
Doctor – wants to ensure patient's needs are met in a timely manner

Brief Description: This use case describes how we make an appointment as well as changing or canceling an appointment for a previously seen patient.

Trigger: Patient calls and asks for a new appointment or asks to cancel or change an existing appointment.

Type: External

Relationships:
Association: Old Patient
Include:
Extend: Update Patient Information
Generalization: Manage Appointments

Normal Flow of Events:
1. The Patient contacts the office regarding an appointment.
2. The Patient provides the Receptionist with his or her name and address.
3. If the Patient's information has changed
   Execute the Update Patient Information use case.
4. If the Patient's payment arrangements has changed
   Execute the Make Payments Arrangements use case.
5. The Receptionist asks Patient if he or she would like to make a new appointment, cancel an existing appointment, or change an existing appointment.
   If the patient wants to make a new appointment,
   the S-1: new appointment subflow is performed.
   If the patient wants to cancel an existing appointment,
   the S-2: cancel appointment subflow is performed.
   If the patient wants to change an existing appointment,
   the S-3: change appointment subflow is performed.
6. The Receptionist provides the results of the transaction to the Patient.

SubFlows:
S-1: New Appointment
1. The Receptionist asks the Patient for possible appointment times.
2. The Receptionist matches the Patient's desired appointment times with available dates and times and schedules the new appointment.

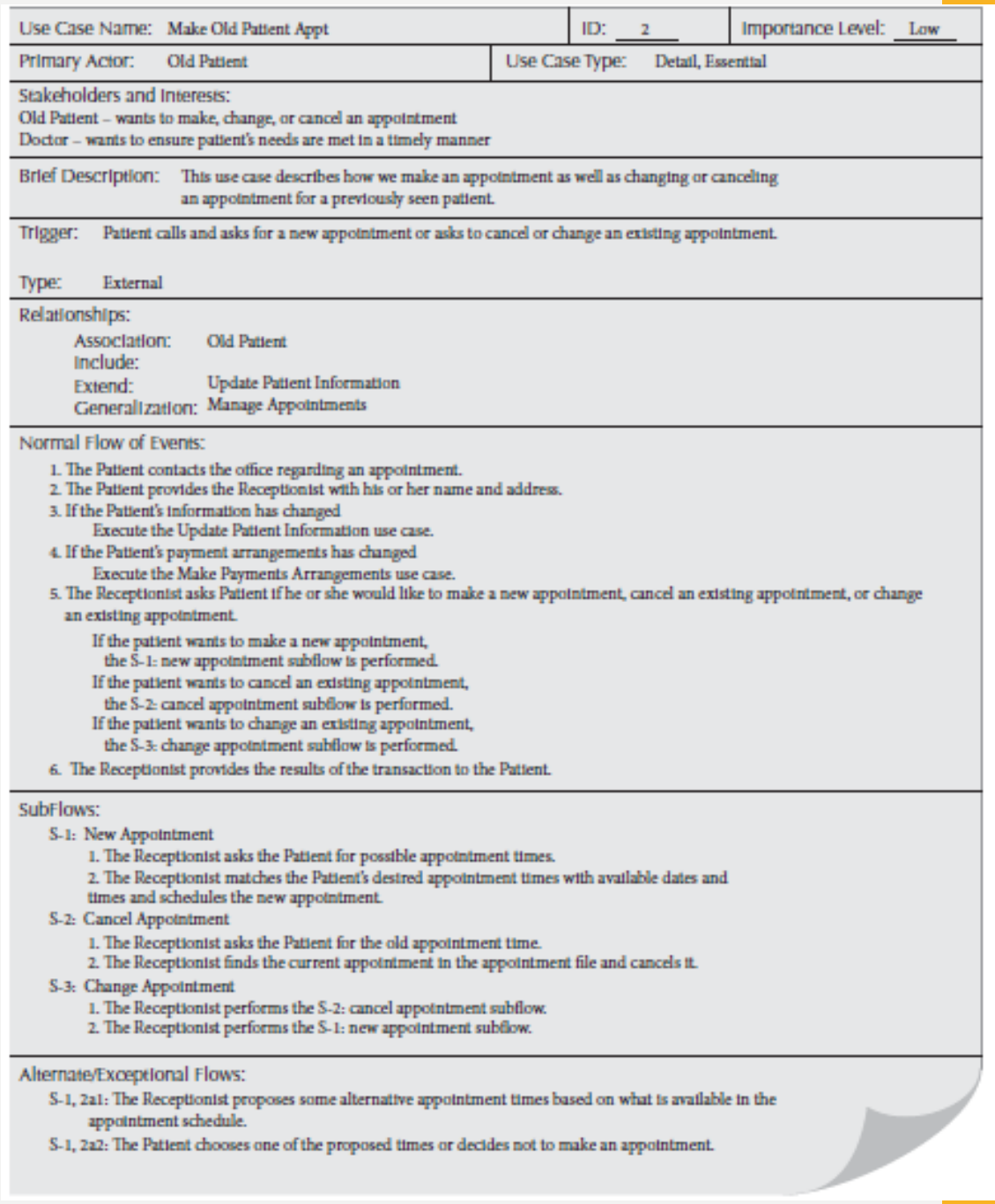
S-2: Cancel Appointment
1. The Receptionist asks the Patient for the old appointment time.
2. The Receptionist finds the current appointment in the appointment file and cancels it.

S-3: Change Appointment
1. The Receptionist performs the S-2: cancel appointment subflow.
2. The Receptionist performs the S-1: new appointment subflow.

Alternate/Exceptional Flows:
S-1, 2a1: The Receptionist proposes some alternative appointment times based on what is available in the appointment schedule.
S-1, 2a2: The Patient chooses one of the proposed times or decides not to make an appointment.

Role-Playing CRC Cards with Use Cases -- Third Step – Role-Play Scenario

- Each team member must pretend that he or she is an instance of the role assigned to him or her.
- For example, if a team member was assigned the role of the Receptionist, then he or she would have to be able to perform the different steps in the scenario associated with the Receptionist.
- In the case of the change appointment scenario, this would include steps 2, 5, 6, S-3, S-1, and S-2.
- However, when this scenario is performed (role-played), it would be discovered that steps 1, 3, and 4 were incomplete… *continued in next slides*



Use Case Name: Make Old Patient Appt | ID: 2 | Importance Level: Low

Primary Actor: Old Patient | Use Case Type: Detail, Essential

Stakeholders and Interests:
Old Patient – wants to make, change, or cancel an appointment
Doctor – wants to ensure patient's needs are met in a timely manner

Brief Description: This use case describes how we make an appointment as well as changing or canceling an appointment for a previously seen patient.

Trigger: Patient calls and asks for a new appointment or asks to cancel or change an existing appointment.

Type: External

Relationships:
Association: Old Patient
Include:
Extend: Update Patient Information
Generalization: Manage Appointments

Normal Flow of Events:
1. The Patient contacts the office regarding an appointment.
2. The Patient provides the Receptionist with his or her name and address.
3. If the Patient's information has changed
   Execute the Update Patient Information use case.
4. If the Patient's payment arrangements has changed
   Execute the Make Payments Arrangements use case.
5. The Receptionist asks Patient if he or she would like to make a new appointment, cancel an existing appointment, or change an existing appointment.
   If the patient wants to make a new appointment, the S-1: new appointment subflow is performed.
   If the patient wants to cancel an existing appointment, the S-2: cancel appointment subflow is performed.
   If the patient wants to change an existing appointment, the S-3: change appointment subflow is performed.
6. The Receptionist provides the results of the transaction to the Patient.

SubFlows:
S-1: New Appointment
1. The Receptionist asks the Patient for possible appointment times.
2. The Receptionist matches the Patient's desired appointment times with available dates and times and schedules the new appointment.

S-2: Cancel Appointment
1. The Receptionist asks the Patient for the old appointment time.
2. The Receptionist finds the current appointment in the appointment file and cancels it.

S-3: Change Appointment
1. The Receptionist performs the S-2: cancel appointment subflow.
2. The Receptionist performs the S-1: new appointment subflow.

Alternate/Exceptional Flows:
S-1, 2a1: The Receptionist proposes some alternative appointment times based on what is available in the appointment schedule.
S-1, 2a2: The Patient chooses one of the proposed times or decides not to make an appointment.

Role-Playing CRC Cards with Use Cases -- Third Step – Role-Play Scenario

- For example, in Step 1, what actually occurs? Does the Patient make a phone call? If so, who answers the phone?
- In other words, a lot of information contained in the use-case description is only identified in an **implicit**, not explicit, manner.
- When the information is not identified explicitly, there is a lot of room for interpretation, which requires the team members to make **assumptions**.
- It is much better to **remove the need to make an assumption by making each step explicit.**

Role-Playing CRC Cards with Use Cases -- Third Step – Role-Play Scenario

- In this case, Step 1 of the Normal Flow of Events should be modified. **Once the step has been fixed, the scenario is tried again.** This process is repeated until the scenario can be executed to a successful conclusion. Once the scenario has successfully concluded, the next scenario is performed. This is repeated until all of the scenarios of the use case can be performed successfully.

Role-Playing CRC Cards with Use Cases -- Fourth Step – Repeat Steps 1 to 3

- The fourth step is to simply repeat steps 1 through 3 for the remaining use cases.

Your Turn! A simple task

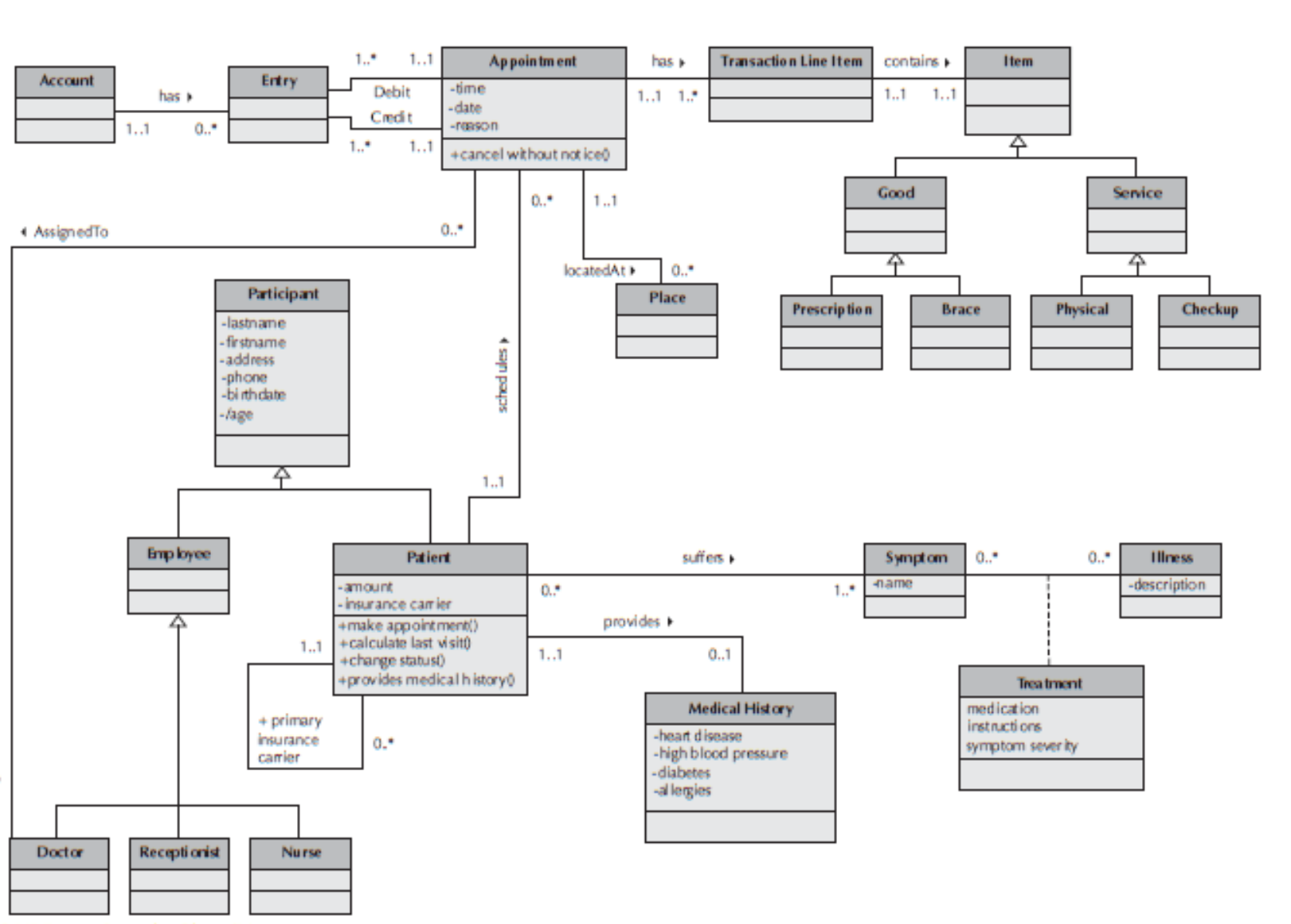
- Create a CRC card for each of the following classes:
  - Movie (title, producer, length, director, genre)
  - Ticket (price, adult or child, showtime, movie)
  - Patron (name, adult or child, age)

1.3 CLASS DIAGRAMS

Introduction

- A *class diagram* is a *static model* that shows the classes and the relationships among classes that remain constant in the system over time.
- The class diagram depicts classes, which include both behaviors and states, with the relationships between the classes.
  - The main building block of a class diagram is the class, which stores and manages information in the system.
- The following sections present the elements of the class diagram, different approaches that can be used to simplify a class diagram, and an alternative structure diagram: the object diagram.

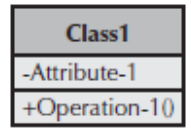
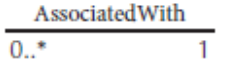
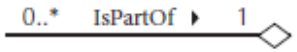
Example Class Diagram



Complete Class Syntax

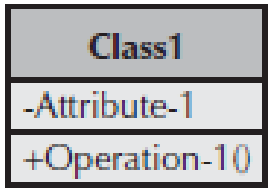
1. A class
2. An attribute
3. An operation
4. An association
5. A generalization
6. An aggregation
7. A composition

Each will be described in details in next slides …

A class: • Represents a kind of person, place, or thing about which the system will need to capture and store information. • Has a name typed in bold and centered in its top compartment. • Has a list of attributes in its middle compartment. • Has a list of operations in its bottom compartment. • Does not explicitly show operations that are available to all classes.	
An attribute: • Represents properties that describe the state of an object. • Can be derived from other attributes, shown by placing a slash before the attribute's name.	attribute name /derived attribute name
An operation: • Represents the actions or functions that a class can perform. • Can be classified as a constructor, query, or update operation. • Includes parentheses that may contain parameters or information needed to perform the operation.	operation name ()
An association: • Represents a relationship between multiple classes or a class and itself. • Is labeled using a verb phrase or a role name, whichever better represents the relationship. • Can exist between one or more classes. • Contains multiplicity symbols, which represent the minimum and maximum times a class instance can be associated with the related class instance.	
A generalization: • Represents a-kind-of relationship between multiple classes.	
An aggregation: • Represents a logical a-part-of relationship between multiple classes or a class and itself. • Is a special form of an association.	
A composition: • Represents a physical a-part-of relationship between multiple classes or a class and itself • Is a special form of an association.	

Classes

- During analysis, classes refer to the people, places, and things about which the system will capture information. Later, during design and implementation, classes can refer to implementation-specific artifacts such as windows, forms, and other objects used to build the system.
- We can see that the classes identified earlier, such as *Participant*, *Doctor*, *Patient*, *Receptionist*, *Medical History*, *Appointment*, and *Symptom*, are included in the previous figure.

A class:	
• Represents a kind of person, place, or thing about which the system will need to capture and store information. • Has a name typed in bold and centered in its top compartment. • Has a list of attributes in its middle compartment. • Has a list of operations in its bottom compartment. • Does not explicitly show operations that are available to all classes.	

Attributes (of Classes

- Attributes are properties of the class about which we want to capture information
- Notice that the Participant class in previous figure contains the attributes:
  - lastname, firstname, address, phone, and birthdate.
- At times, you might want to store *derived attributes*
  - attributes that can be calculated or derived;
  - these special attributes are denoted by placing a slash (/) before the attribute's name.
- E.g., the person class contains a derived attribute called /age, which can be derived by subtracting the patient's birth date from the current date.

An attribute: • Represents properties that describe the state of an object. • Can be derived from other attributes, shown by placing a slash before the attribute's name.	attribute name /derived attribute name

Attributes (of Classes

- Visibility relates to the level of information hiding to be enforced for the attribute. The visibility of an attribute can be public (+), protected (#), or private (–).
  - A *public* attribute is one that is not hidden from any other object. As such, other objects can modify its value.
  - A *protected* attribute is one that is hidden from all other classes except its immediate subclasses.
  - A *private* attribute is one that is hidden from all other classes.
- The default visibility for an attribute is normally *private*.

Operations (of Classes)

- *Operations* are actions or functions that a class can perform.
- The functions that *are available to all classes* (e.g., create a new instance, return a value for a particular attribute, set a value for a particular attribute, delete an instance) are ***not*** *explicitly shown within the class rectangle*.
- Instead, only operations **unique** to the class are included,
- E.g., the cancel without notice operation in the Appointment class and the calculate last visit operation in the Patient class in the previous figure.

An operation: • Represents the actions or functions that a class can perform. • Can be classified as a constructor, query, or update operation. • Includes parentheses that may contain parameters or information needed to perform the operation.	operation name ()

Operations (of Classes)

- Notice that both the operations are followed by parentheses, which contain the parameter(s) needed by the operation.
- If an operation has no parameters, the parentheses are still shown but are empty.
- As with attributes, the visibility of an operation can be designated public, protected, or private.
- The default visibility for an operation is normally public.

Operations (of Classes)

- There are four kinds of operations that a class can contain: *constructor*, *query*, *update*, and *destructor*.
  - A *constructor operation* creates a new instance of a class.
    - For example, the patient class may have a method called insert (), which creates a new patient instance as patients are entered into the system.
    - If an operation implements one of the basic functions (e.g., create a new instance), it is normally not explicitly shown on the class diagram, so *typically we do not see constructor methods explicitly on the class diagram*.

Operations (of Classes)

- A *query operation* makes information about the state of an object available to other objects, but it does not alter the object in any way.
  - For instance, the calculate last visit () operation that determines when a patient last visited the doctor's office will result in the object's being accessed by the system, but it will not make any change to its information.
  - If a query method *merely asks for information from attributes in the class* (e.g., a patient's name, address, phone), then it is *not shown* on the diagram because we assume that all objects have operations that produce the values of their attributes.

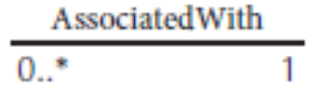
Operations (of Classes)

- An *update operation* changes the value of some or all the object's attributes, which may result in a change in the object's state.
  - Consider changing the status of a patient from new to current with a method called change status() or associating a patient with a particular appointment with make appointment (appointment).
  - If the result of the operation can change the state of the object, then the operation must be explicitly included on the class diagram. On the other hand*, if the update operation is a simple assignment operation, it can be omitted from the diagram*.

Operations (of Classes)

- A *destructor operation* simply deletes or removes the object from the system.
  - For example, if an employee object no longer represents an actual employee associated with the firm, the employee could need to be removed from the employee database, and a destructor operation would be used to implement this behavior.
  - However, deleting an object is one of the basic functions and therefore *would not be included on the class diagram*.

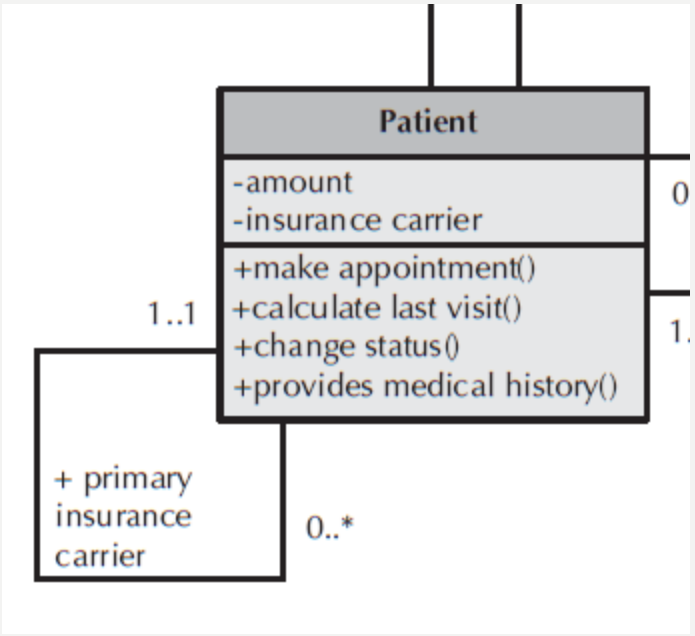
Associations (i.e., the Relationships)

An association: • Represents a relationship between multiple classes or a class and itself. • Is labeled using a verb phrase or a role name, whichever better represents the relationship. • Can exist between one or more classes. • Contains multiplicity symbols, which represent the minimum and maximum times a class instance can be associated with the related class instance.	

- A primary purpose of a class diagram is to show the relationships, or **associations**, that classes have with one another.
- When multiple classes share a relationship (or a class shares a relationship with itself), a line is drawn and labeled with either the name of the relationship or the roles that the classes play in the relationship.
- For example, the two classes namely patient and appointment are associated with one another whenever a patient schedules an appointment.
- Thus, a line labeled schedules connects patient and appointment, representing exactly how the two classes are related to each other.

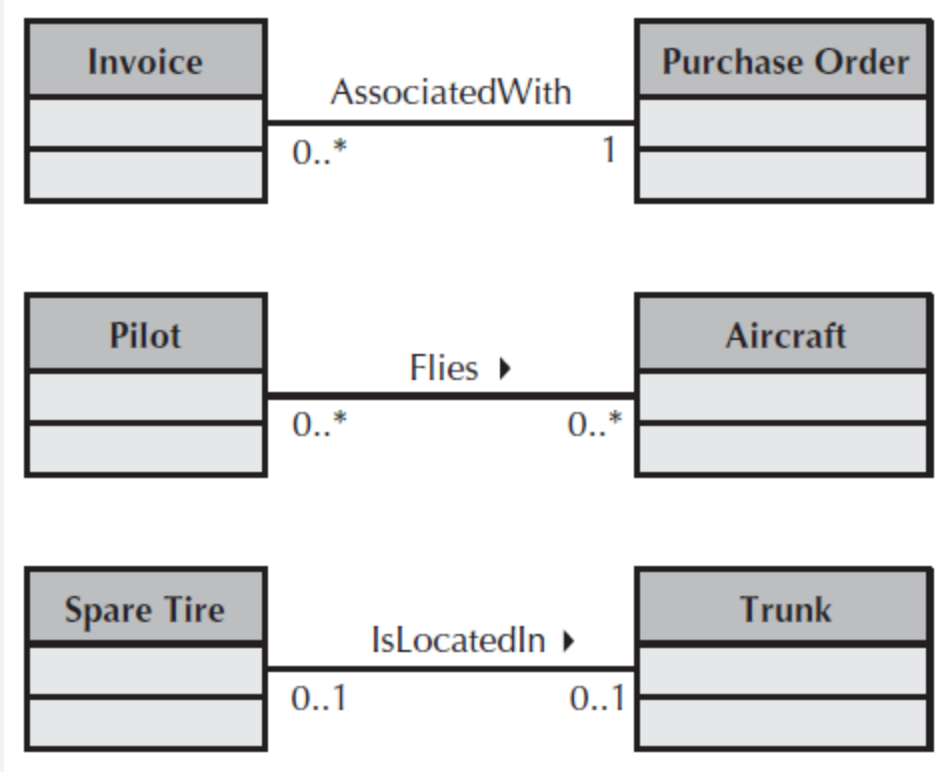
Associations

- Sometimes a class is related to itself, as in the case of a patient being the primary insurance carrier for other patients (e.g., spouse, children).
- Notice that a line was drawn between the patient class and itself and called *primary insurance carrier* to depict the role that the class plays in the relationship.
- Notice that a plus (+) sign is placed before the label to communicate that it is a role as opposed to the name of the relationship.
- When labeling an association, we use either a relationship name or a role name (not both), whichever communicates a more thorough understanding of the model.



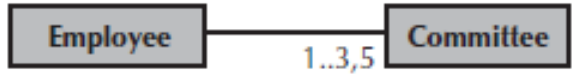
Associations

- Three examples of associations are portrayed:
  1. An Invoice is *AssociatedWith* a Purchase Order (and vice versa),
  2. a Pilot *Flies* an Aircraft , and
  3. a Spare Tire *IsLocatedIn* a Trunk.
- Also, notice that there is a **small solid triangle** beside the name of the relationship.
  - The triangle allows a direction to be associated with the name of the relationship.



Associations

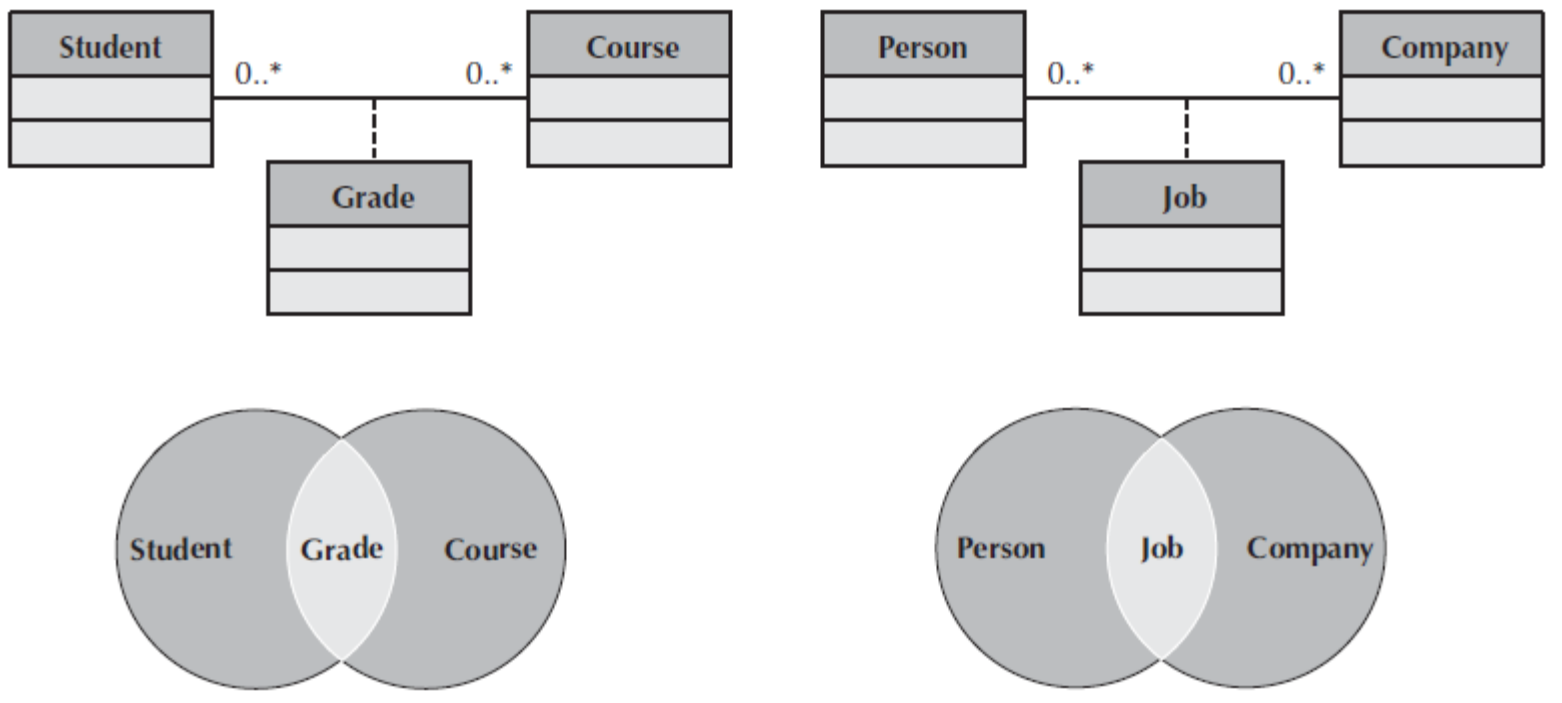
- Relationships also have *multiplicity*, which documents how an instance of an object can be associated with other instances.
- Numbers are placed on the association path to denote the **minimum** and **maximum** instances that can be related through the association in the format minimum number.. maximum number (see next Figure).

Exactly one	1		A department has one and only one boss.
Zero or more	0..*		An employee has zero to many children.
One or more	1..*		A boss is responsible for one or more employees.
Zero or one	0..1		An employee can be married to zero or one spouse.
Specified range	2..4		An employee can take from two to four vacations each year.
Multiple, disjoint ranges	1..3,5		An employee is a member of one to three or five committees.

Associations

- There are times when a relationship itself has associated properties, especially when its classes share a many-to-many relationship.
  - In these cases, a class called an *association class* is formed, which has its own attributes and operations.
  - It is shown as a rectangle attached by a dashed line to the association path, and the rectangle's name matches the label of the association. Think about the case of capturing information about illnesses and symptoms.
- Another way to decide when to use an association class is when attributes that belong to the intersection of the two classes involved in the association must be captured.

Associations



Associations

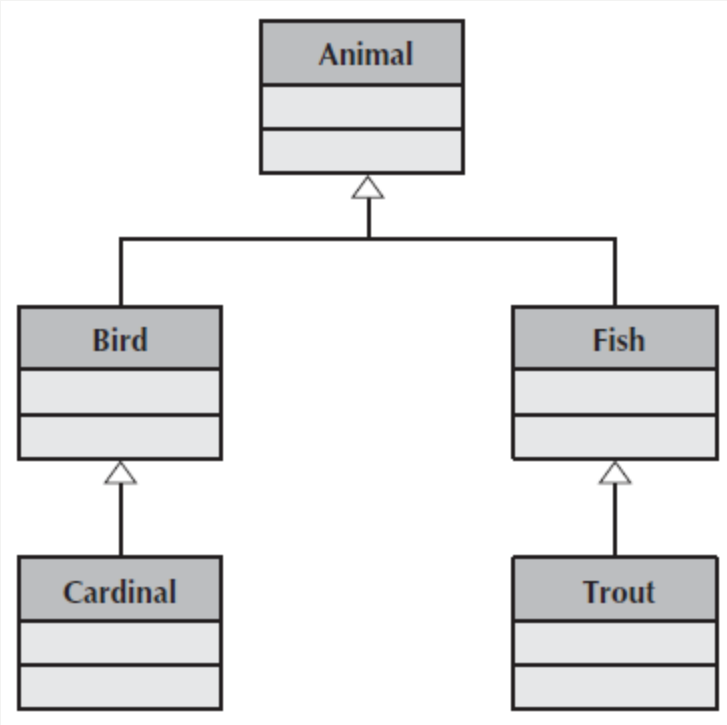
- We can visually think about an association class. For example, in the figure before, the Grade idea is really an intersection of the Student and Course classes, because a grade exists only at the intersection of these two ideas.
- Another example shown in the figure is that a job may be viewed as the intersection between a Person and a Company.
- Most often, classes are related through a normal association; however, there are two special cases of an association that you will see appear quite often: generalization and aggregation.

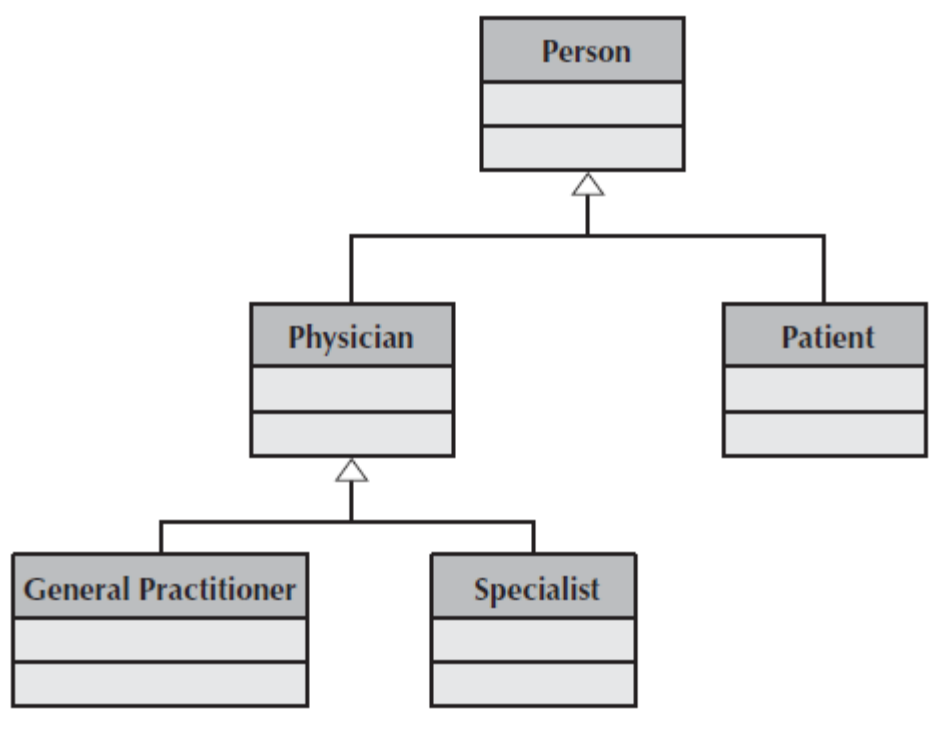
Generalization Associations

A generalization: • Represents a-kind-of relationship between multiple classes.	

- A *generalization association* shows that one class (subclass) inherits from another class (superclass), meaning that the properties and operations of the superclass are also valid for objects of the subclass.
- The generalization path is shown with a solid line from the subclass to the superclass and a hollow arrow pointing at the superclass.
- Remember that the generalization relationship occurs when you need to use words like “is a kind of” to describe the relationship.

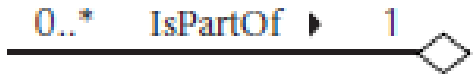
Generalization Associations





- The figures above state that Cardinal is a-kind-of Bird, which is a-kind-of Animal; a General Practitioner is a-kind-of Physician, which is a-kind-of Person

Aggregation Associations

An aggregation: • Represents a logical a-part-of relationship between multiple classes or a class and itself. • Is a special form of an association.	

- Aggregation is used to portray **logical** a-part-of relationships and is depicted on a UML class diagram by a hollow or **white** diamond.
- **Logical** implies that it is possible for a part to be associated with multiple wholes or that is relatively simple for the part to be removed from the whole.

Aggregation Associations

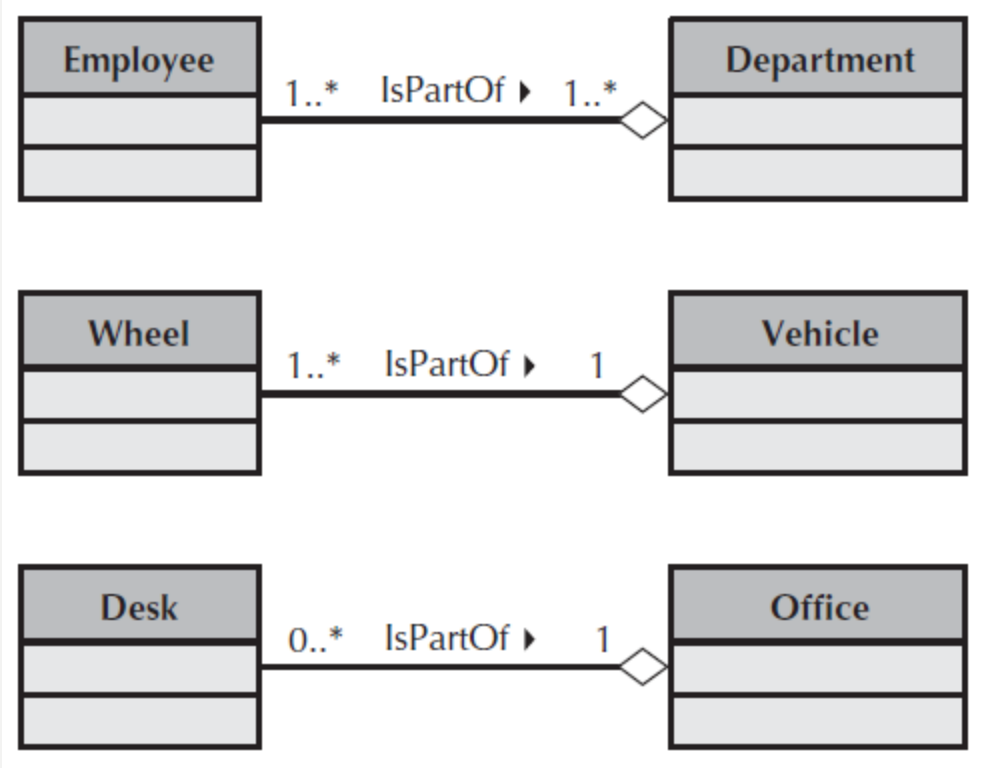
- For example, think about a doctor's office that has decided to create *health care teams* that include *doctors*, *nurses*, and *administrative personnel*.
- As patients enter the office, they are assigned to a health care team, which cares for their needs during their visits. We could include this new knowledge by adding two new classes (Administrative Personnel and Health Team) and aggregation relationships from the Doctor, the Nurse, and the new Administrative Personnel classes to the new Health Team class.
- A diamond is placed nearest the class representing the aggregation (health care team), and lines are drawn from the diamond to connect the classes that serve as its parts (doctors, nurses, and administrative personnel).

Aggregation Associations

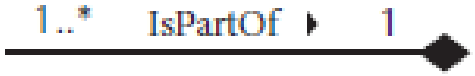
- Typically, you can identify these kinds of associations when you need to use words like “is a part of” or “is made up of” to describe the relationship.
- From a UML perspective, there are two types of aggregation associations: aggregation and composition.

Aggregation Associations

- For example:
  - an instance of the Employee class IsPartOf an instance of at least one instance of the Department class,
  - an instance of the Wheel class IsPartOf an instance of the Vehicle class, and
  - an instance of the Desk class IsPartOf an instance of the Office class.
- Obviously, in many cases an employee can be associated with more than one department, and it is relatively easy to remove a wheel from a vehicle or move a desk from an office.

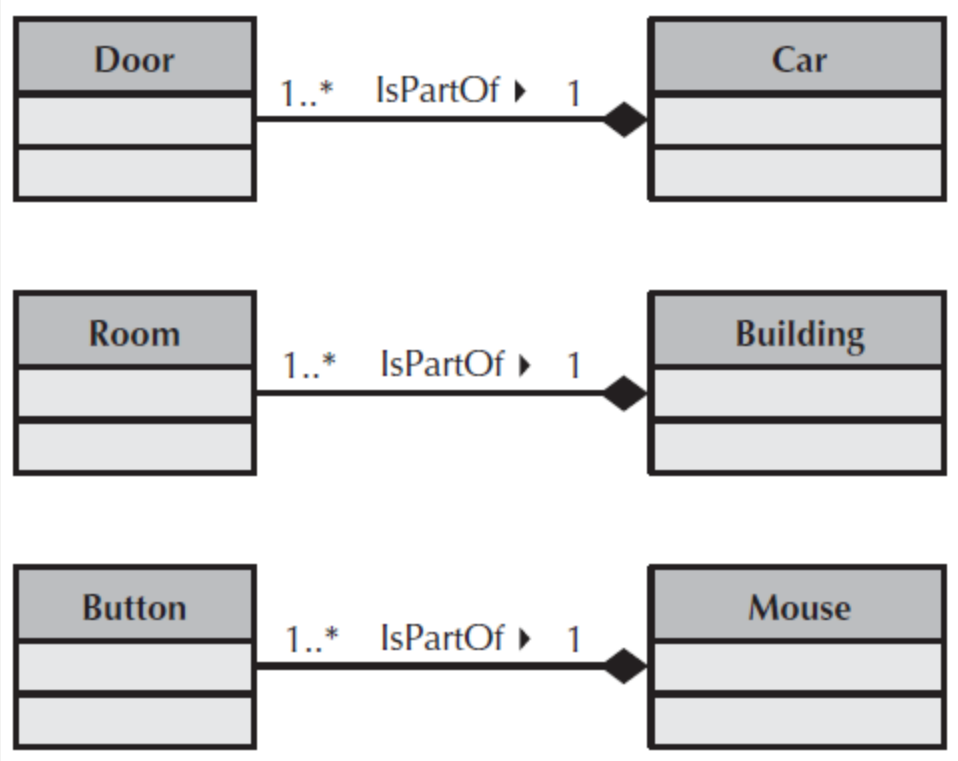


Composition Association

A composition: • Represents a physical a-part-of relationship between multiple classes or a class and itself • Is a special form of an association.	

- Composition is used to portray a physical part of relationships and is shown by a black diamond.
- **Physical** implies that the part can be associated with only a single whole.

Composition Association: Example



- For example in the next figure 3 physical compositions are illustrated: an instance of a door can be a part of only a single instance of a car, an instance of a room can be a part of an instance only of a single building, and an instance of a button can be a part of only a single mouse.

Composition Association: Example

- For example in the next figure 3 physical compositions are illustrated: an instance of a door can be a part of only a single instance of a car, an instance of a room can be a part of an instance only of a single building, and an instance of a button can be a part of only a single mouse.
- However, in many cases, the distinction that you can achieve by including aggregation (**white** diamonds) and composition (**black** diamonds) in a class diagram might not be worth the price of adding additional graphical notation for the client to learn.
- Therefore, many UML experts view the inclusion of aggregation and composition notation to the UML class diagram as simply "syntactic sugar" and not necessary because the same information can always be portrayed by simply using the association syntax.

Your turn! A simple task

- Create a class diagram based on the CRC cards you created for previous task!

wrapped up: the complete steps are …

Creating Structural Models Using CRC Cards and Class Diagrams (7-step)

1. Create CRC Cards
2. Review CRC Cards
   - review the CRC cards to determine if additional candidate objects, attributes, operations, and relationships are missing
3. Role-Play the CRC Cards
4. Create Class Diagram
5. Review Class Diagram
   - challenging the reasons for including the information contained in the model
6. Incorporate Patterns
7. Review the Model (see verifying & validating CRC cards and class diagram)

..... when we finish creating a class diagram, it can be the case that the class diagram is fully populated with all the classes and relationships for a real world system, thus the class diagram can become very difficult to interpret (i.e., can be very complex) .....

1.4 SIMPLIFYING CLASS DIAGRAM

Simplifying Class Diagrams

1. One way to simplify the class diagram is to show only concrete classes.
   - However, depending on the number of associations that are connected to abstract classes—and thus inherited down to the concrete classes—this particular suggestion could make the diagram more difficult to comprehend.

Simplifying Class Diagrams

2. A second way to simplify the class diagram is through the use of a *view* mechanism.
   - Views were developed originally with relational database management systems to show only a subset of the information contained in the database.
   - In this case, the view would be a useful subset of the class diagram, such as
     - A first view, i.e., a use-case view that shows only the classes and relationships relevant to a particular use case.
     - A second view could be to show only a particular type of relationship: aggregation, association, or generalization.
     - A third type of view is to restrict the information shown with each class, for example, show only the name of the class, the name and attributes, or the name and operations.

Simplifying Class Diagrams

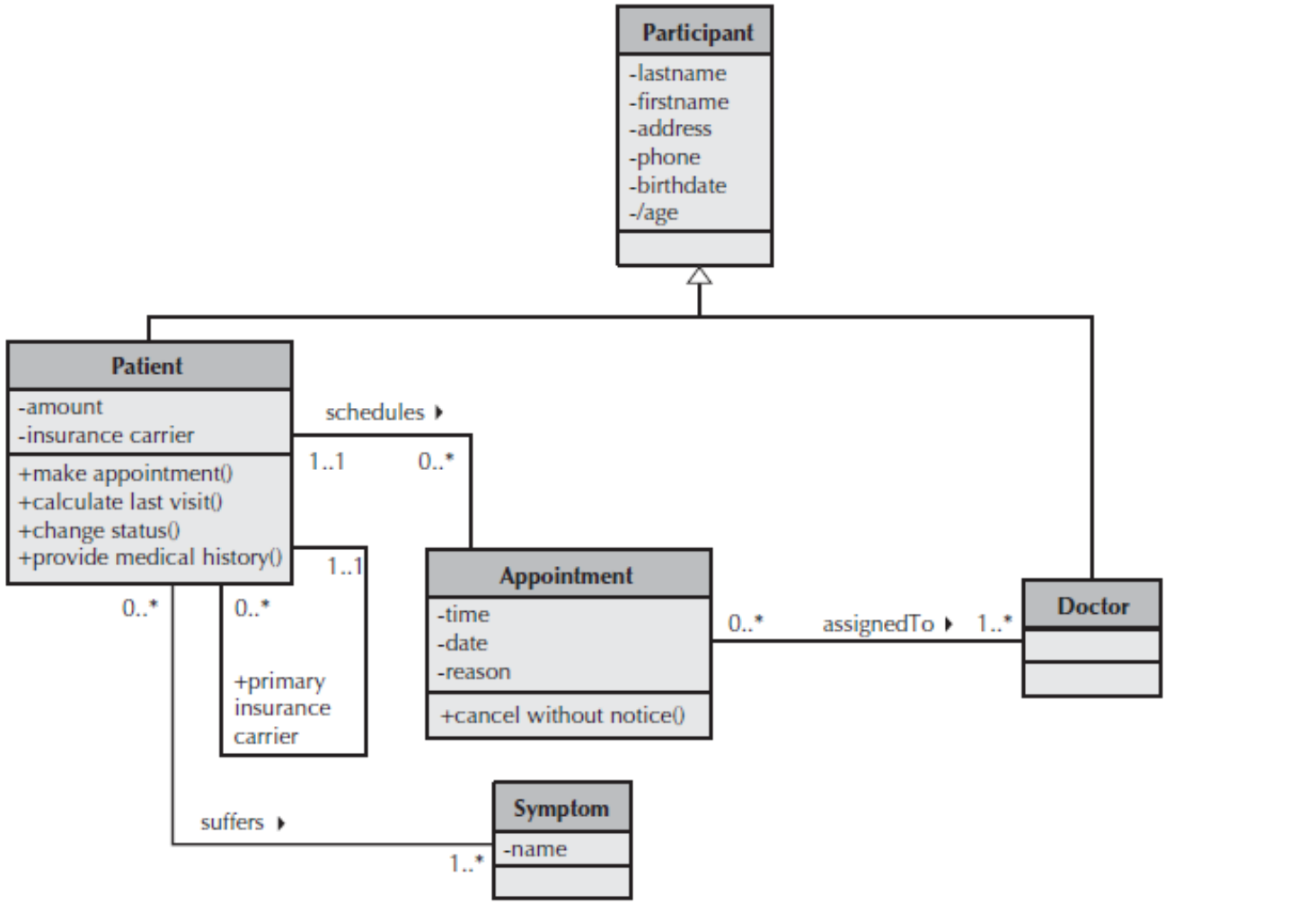
3. A third approach to simplifying a class diagram is through the use of *packages* (i.e., logical groups of classes).
   - To make the diagrams easier to read and keep the models at a reasonable level of complexity, the classes can be grouped together into packages.
   - In the case of class diagrams, it is simple to sort the classes into groups based on the relationships that they share.
   - These view mechanisms can be combined to further simplify the diagram.

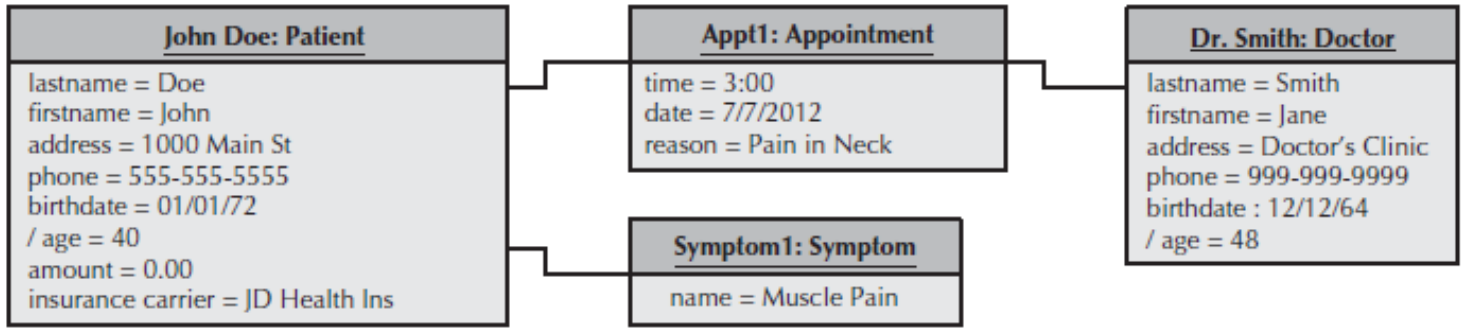
1.5 OBJECT DIAGRAMS (BRIEF)

Object Diagrams

- Although class diagrams are necessary to document the structure of the classes, a second type of *static structure diagram*, called an object diagram, can be useful in revealing additional information.
- An *object diagram* is essentially an instantiation of all or part of a class diagram.
- *Instantiation* means to create an instance of the class with a set of appropriate attribute values.
- Object diagrams can be very useful when trying to uncover details of a class.

Object Diagrams: example





1.6 VERIFYING & VALIDATING THE STRUCTURAL MODEL

Verifying & Validating the Structural Model (8-step)

- First, every CRC card should be associated with a class on the class diagram, and vice versa.
  - In the appointment example, the Old Patient class represented by the CRC card does not seem to be included on the class diagram.
  - However, there is a Patient class on the class diagram. The Old Patient CRC card most likely should be changed to simply Patient.
- Second, the responsibilities listed on the front of the CRC card must be included as operations in a class on a class diagram, and vice versa.
  - The make appointment responsibility on the new Patient CRC card also appears as the make appointment() operation in the Patient class on the class diagram.
  - Every responsibility and operation must be checked.

Verifying & Validating the Structural Model (8-step)

- Third, collaborators on the front of the CRC card imply some type of relationship on the back of the CRC card and some type of association that is connected to the associated class on the class diagram.
  - The appointment collaborator on the front of the CRC card also appears as another association on the back of the CRC card and as an association on the class diagram that connects the Patient class with the Appointment class.
- Fourth, attributes listed on the back of the CRC card must be included as attributes in a class on a class diagram, and vice versa.
  - For example, the amount attribute on the new Patient CRC card is included in the attribute list of the Patient class on the class diagram.

Verifying & Validating the Structural Model (8-step)

- Fifth, the object type of the attributes listed on the back of the CRC card and with the attributes in the attribute list of the class on a class diagram implies an association from the class to the class of the object type.
  - For example, technically speaking, the amount attribute implies an association with the double type.
  - However, simple types such as int and double are never shown on a class diagram.
  - Furthermore, depending on the problem domain, object types such as Person, Address, or Date might not be explicitly shown either.
  - However, if we know that messages are being sent to instances of those object types, we probably should include these implied associations as relationships.

Verifying & Validating the Structural Model (8-step)

- Sixth, the relationships included on the back of the CRC card must be portrayed using the appropriate notation on the class diagram.
  - For example, instances of the Patient class are *a-kind-of* Person, it has instances of the Medical History class as part of it, and it has an association with instances of the Appointment class. Thus, the association from the Patient class to the Person class should indicate that the Person class is a generalization of its subclasses, including the Patient class; the association from the Patient class to the Medical History class should be in the form of an aggregation association (a white diamond); and the association between instances of the Patient class and instances of the Appointment class should be a simple association.

Verifying & Validating the Structural Model (8-step)

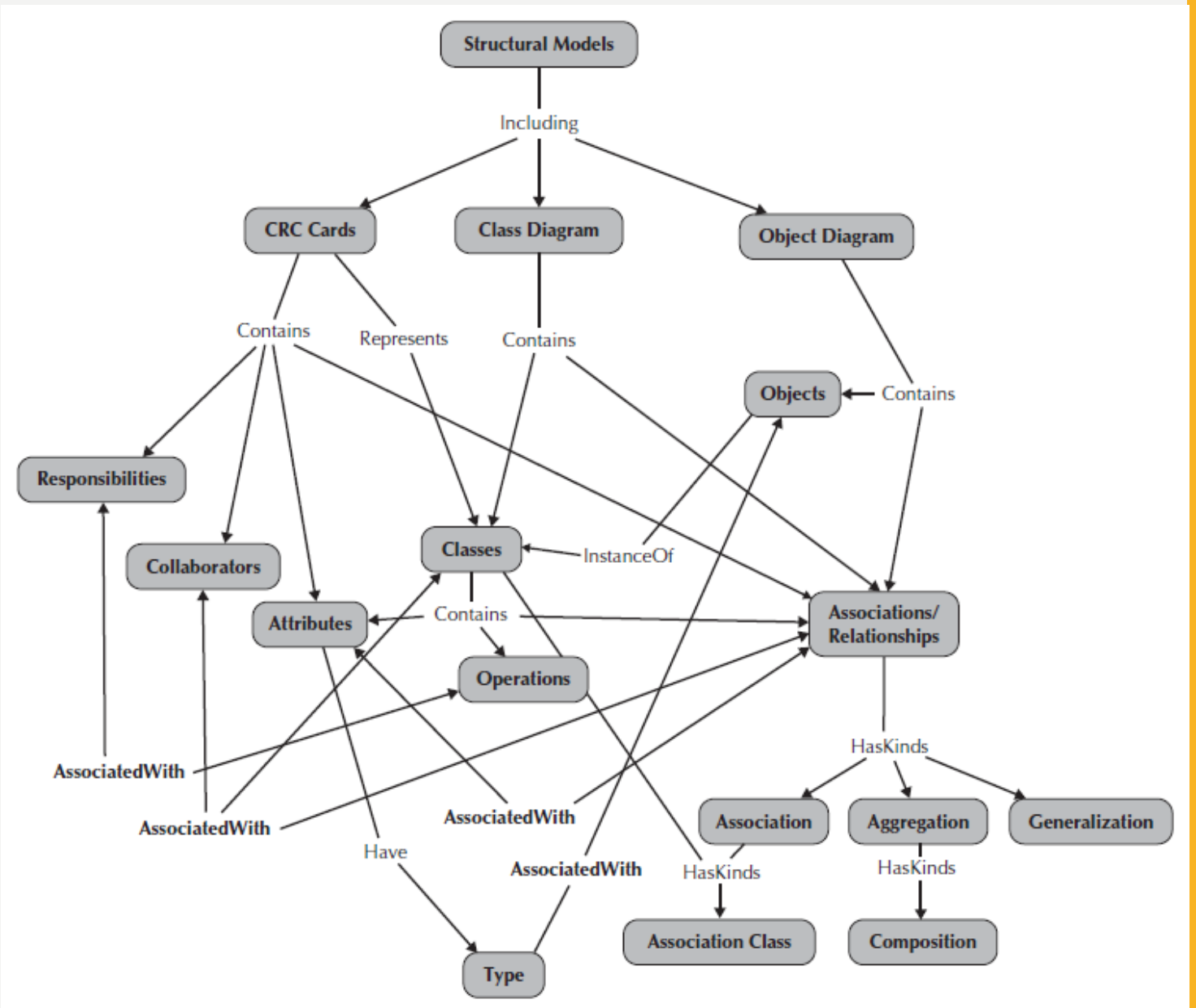
- Sixth, the relationships included on the back of the CRC card must be portrayed using the appropriate notation on the class diagram.
  - However, when we review the class diagram example, this is not what we find. If you recall, we included in the class diagram the transaction pattern. When we did this, many changes were made to the classes contained in the class diagram. All of these changes should have been cascaded back through all of the CRC cards. In this case, the CRC card for the Patient class should show that a Patient is a-kind-of Participant (not Person) and that the relationship from Patient to Medical History should be a simple association

Verifying & Validating the Structural Model (8-step)

- Seventh, an association class, such as the Treatment class, should be created only if there is indeed some unique characteristic (attribute, operation, or relationship) about the intersection of the connecting classes.
  - If no unique characteristic exists, then the association class should be removed and only an association between the two connecting classes should be displayed.
- Finally, as in the functional models, specific representation rules must be enforced.
  - For example, a class cannot be a subclass of itself. The Patient CRC card cannot list Patient with the generalization relationships on the back of the CRC card, nor can a generalization relationship be drawn from the Patient class to itself.

Verifying & Validating the Structural Model

This figure portrays the associations among the structural models



Summary

- *CRC cards* capture the essential elements of a class.
- *Class and object diagrams* show the underlying structure of an object-oriented system.
- Constructing the structural model is an iterative process involving: *textual analysis, brainstorming objects, role playing, creating the diagrams*, and *incorporating useful patterns.*
- *Object diagrams can be used to help identifying details of the class diagrams*
- *Verifying & Validating the Structural Model ensure the consistency of the Structural Model*

References

- Systems Analysis and Design: An Object Oriented Approach with UML 5th ed. Alan Dennis, Barbara Haley Wixom, and Roberta M. Roth © 2015

Your turn! Do it in pair

Draw a class diagram for each of following situations!

1. Whenever new patients are seen for the first time, they complete a patient information form that asks their name, address, phone number, and insurance carrier, which are stored in the patient information fi le. Patients can be signed up with only one carrier, but they must be signed up to be seen by the doctor. Each time a patient visits the doctor, an insurance claim is sent to the carrier for payment. The claim must contain information about the visit, such as the date, purpose, and cost. It would be possible for a patient to submit two claims on the same day.

Your turn! Do it in pair

Draw a class diagram for each of following situations!

2. The state of Georgia is interested in designing a system that will track its researchers. Information of interest includes researcher name, title, position, researcher's university name, university location, university enrollment, and researcher's research interests. Researchers are associated with one institution, and each researcher has several research interests.