Systems Analysis and Design

Chapter 5

System Modeling

The slide is mainly adopted from:

- I. Sommerville. Software Engineering. 10th Edition, Pearson, 2016
- J. S. Valacich, J. George, Modern Systems Analysis and Design. 8th Edition, Pearson 2017.

Last update: Oct. 29, 2018

Topics covered

- Context models
- Interaction models
- Structural models
- Behavioral models
- Model-driven engineering

System modeling

- System modeling is the process of developing abstract models of a system, with each model presenting a **different view** or perspective of that system.
- System modeling has now come to mean representing a system using some kind of **graphical notation**, which is now almost always based on notations in the **Unified Modeling Language (UML)**.
- System modelling helps the **analyst** to understand the **functionality** of the system and models are used to communicate with customers.

Existing and planned system models

- Models of the existing system are used during requirements engineering. They help **clarify** what the existing system does and can be used as a basis for discussing its **strengths** and **weaknesses**. These then lead to requirements for the **new system**.
- Models of the new system are used during requirements engineering to help explain the **proposed requirements** to other system stakeholders. Engineers use these models to discuss design proposals and to document the system for implementation.
- In a **model-driven** engineering process, it is possible to generate a complete or partial system implementation from the system model.

System perspectives

- An **external** perspective, where you model the context or environment of the system.
- An **interaction** perspective, where you model the interactions between a system and its environment, or between the components of a system.
- A **structural** perspective, where you model the **organization** of a system or the **structure** of the data that is processed by the system.
- A **behavioral** perspective, where you model the dynamic behavior of the system and how it **responds** to events.

UML diagram types

- **Activity diagrams**, which show the activities involved in a process or in data processing .
- **Use case diagrams**, which show the interactions between a system and its environment.
- **Sequence diagrams**, which show interactions between actors and the system and between system components.
- **Class diagrams**, which show the object classes in the system and the associations between these classes.
- **State diagrams**, which show how the system reacts to internal and external events.

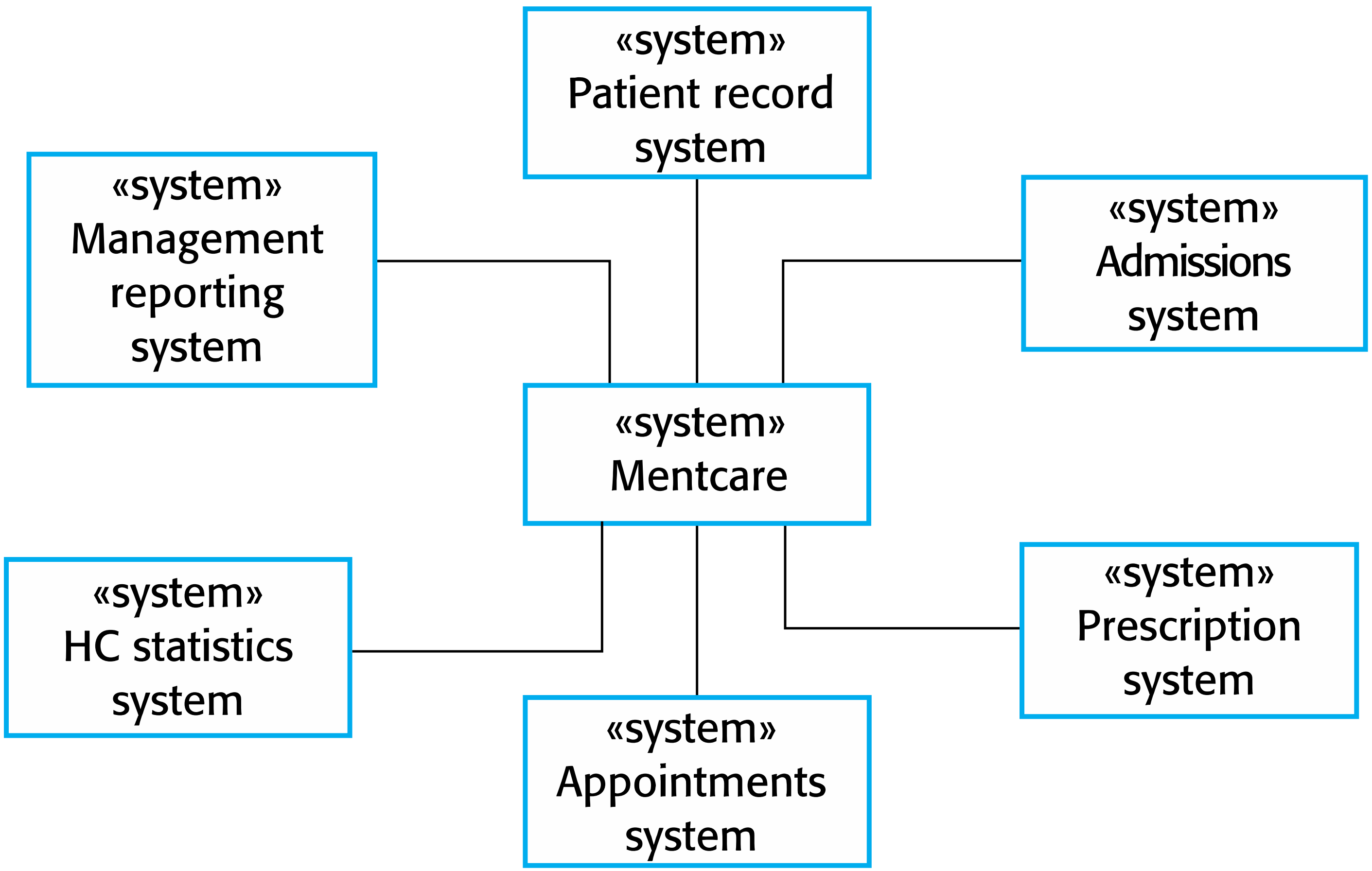
Use of graphical models

- As a means of **facilitating** discussion about an existing or proposed system
  - Incomplete and incorrect models are OK as their role is to support discussion.
- As a way of **documenting** an existing system
  - Models should be an accurate representation of the system but need not be complete.
- As a **detailed system description** that can be used to generate a system implementation
  - Models have to be both correct and complete.

Context models

- Context models are used to **illustrate** the **operational** context of a system - they show what lies outside the system **boundaries**.
- Social and organisational concerns may affect the decision on where to position system boundaries.
- Architectural models show the system and its relationship with other systems.
- System boundaries are established to define what is **inside** and what is **outside** the system.
  - They show other systems that are **used** or **depend** on the system being developed.
- The position of the system boundary has a profound effect on the system requirements.
- Defining a system boundary is a political judgment
  - There may be pressures to develop system boundaries that increase / decrease the influence or workload of different parts of an organization.

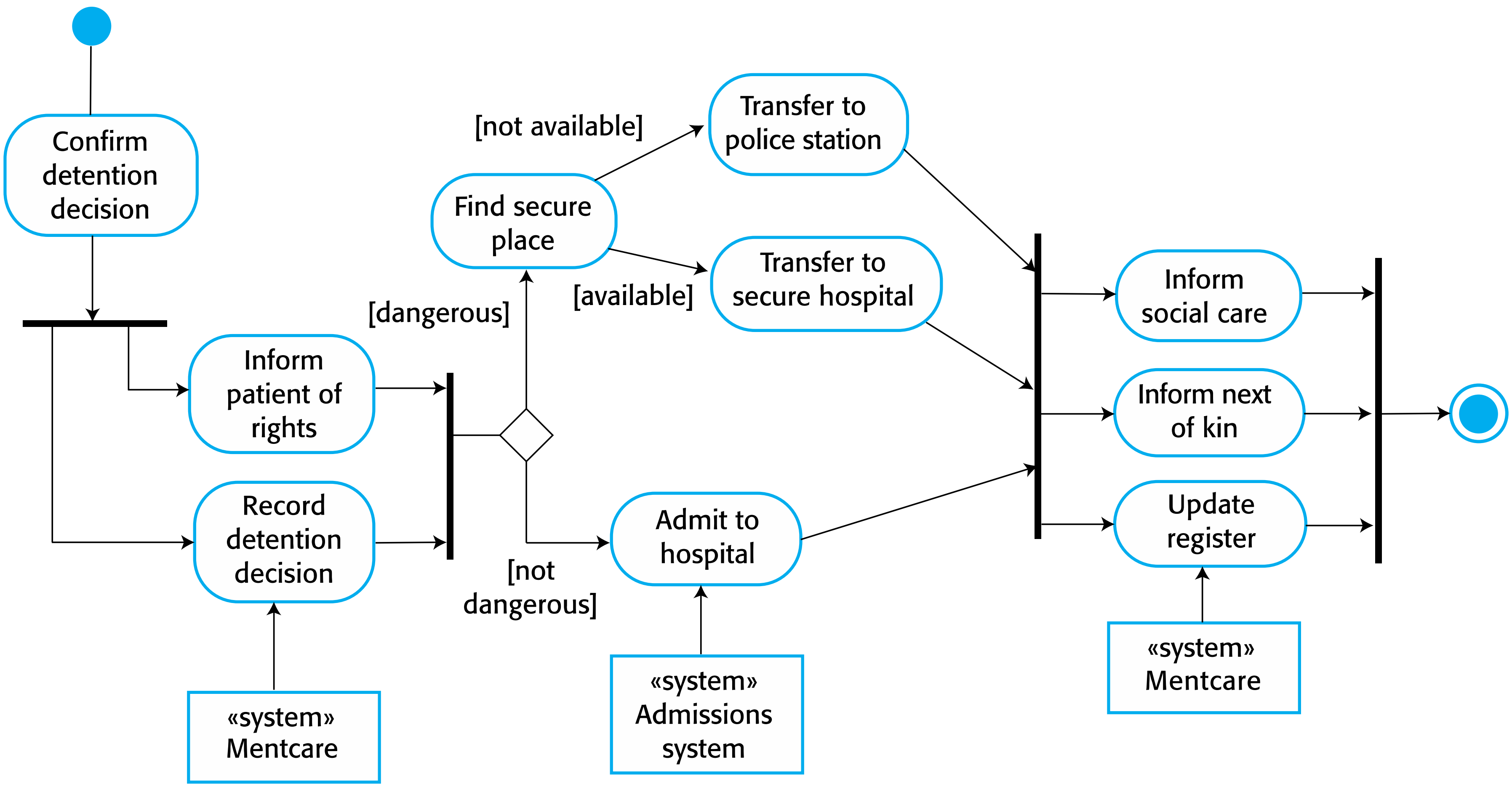
The context of the Mentcare system



Process perspective

- **Context models** simply show the other systems in the environment, not how the system being developed is used in that environment.
- **Process models** reveal how the system being developed is used in broader business processes.
- UML activity diagrams may be used to define business process models.

Process model of involuntary detention



Interaction models

- Modeling user interaction is important as it helps to identify user requirements.
- Modeling system-to-system interaction highlights the communication problems that may arise.
- Modeling component interaction helps us understand if a proposed system structure is likely to deliver the required system performance and dependability.
- Use case diagrams and sequence diagrams may be used for interaction modelling.

Use case modeling

- Use cases were developed originally to support requirements elicitation and now incorporated into the UML.
- Each use case represents a discrete task that involves external interaction with a system.
- Actors in a use case may be people or other systems.
- Represented diagrammatically to provide an overview of the use case and in a more detailed textual form.

Use Case Diagram

- Used for describing a set of user **scenarios**
- Mainly used for capturing user requirements
- Work like a **contract** between end user and software developers
- Steps
  1. Identify, define, and document new actors.
  2. Identify, define, and document new use cases.
  3. Identify any reuse possibilities.
  4. Refine the use-case model diagram (if necessary).
  5. Document system analysis use-case narratives.

Use Case Diagram (core components)

Actors: A role that a user plays with respect to the system,including human users and other systems. e.g.,inanimate physical objects (e.g. robot); an external system that needs some information from the current system.

Use case: A set of scenarios that describing an interaction between a user and a system, including alternatives.

System boundary: rectangle diagram representing the boundary between the actors and the system.

Use Case Diagram(core relationship)

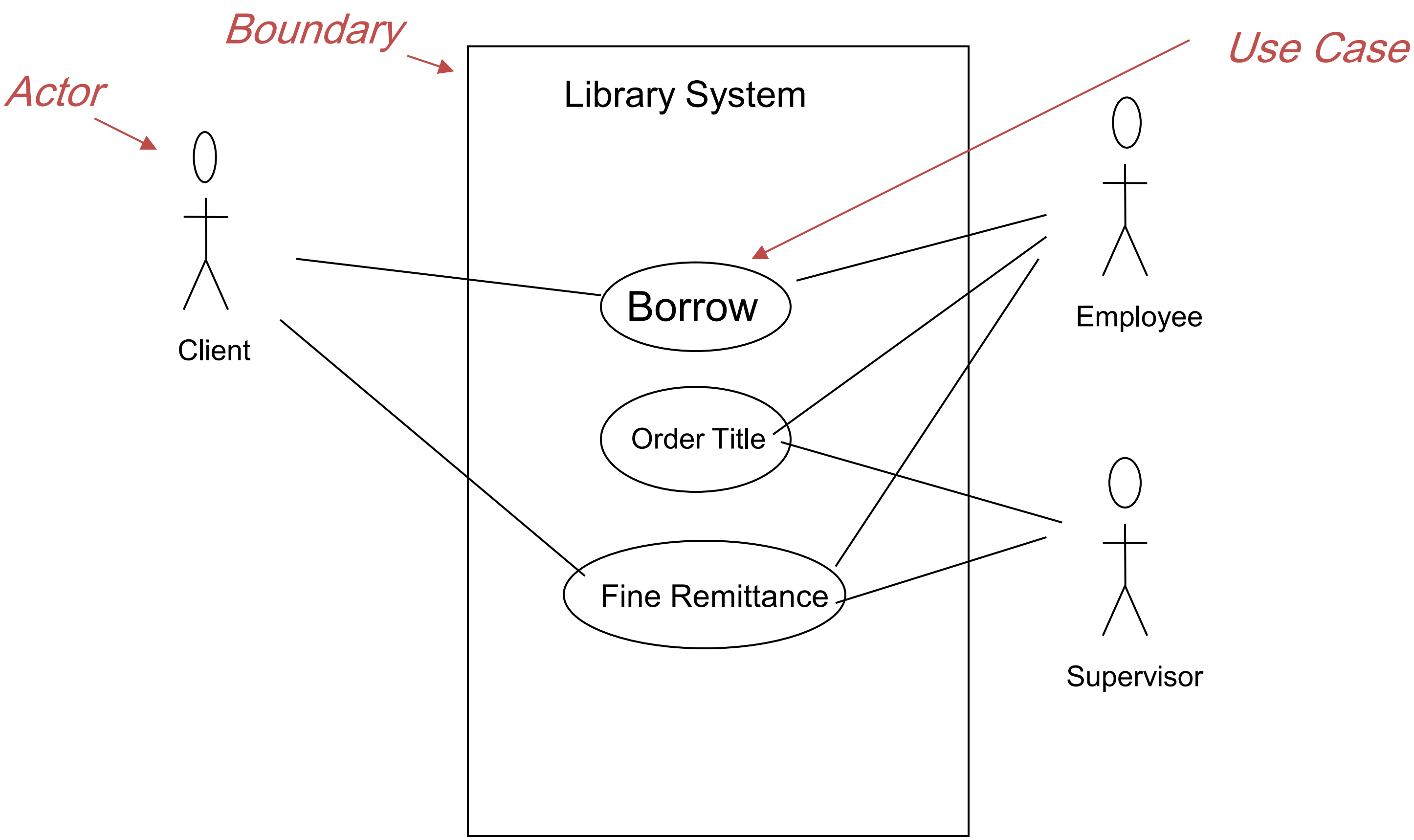
- **<u>Association</u>**: communication between an actor and a use case; Represented by a solid line.
- **<u>Generalization</u>**: relationship between one general use case and a special use case (used for defining special alternatives); Represented by a line with a triangular arrow head toward the parent use case.
- **<u>Include</u>**: a dotted line labeled <<include>> beginning at base use case and ending with an arrows pointing to the include use case.
  The insertion of additional behavior into a base use case that explicitly describes the insertion

  <<include>>

- **<u>Extend</u>**: a dotted line labeled <<extend>> with an arrow toward the base case. The insertion of **additional behavior** into a base use case that does not know about it. The base class declares “extension points”.

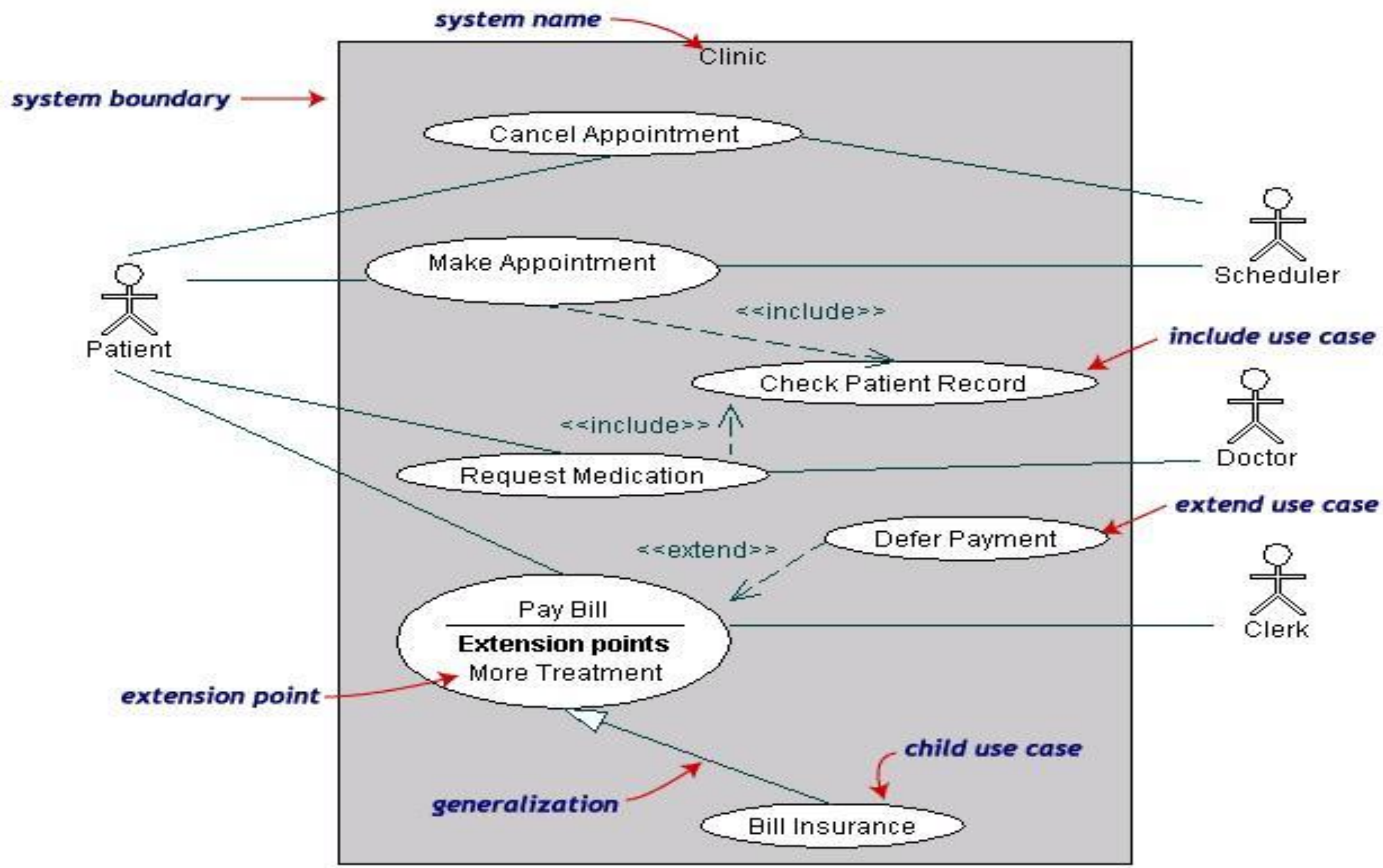
  <<extend>>

Use Case Diagrams



- A generalized description of how a system will be used.
- Provides an overview of the intended functionality of the system

Use Case Diagrams(cont.)



(TogetherSoft, Inc)

Use Case Diagrams(cont.)

- **Pay Bill** is a parent use case and **Bill Insurance** is the child use case. (generalization)
- Both **Make Appointment and Request Medication** include **Check Patient Record** as a subtask.(include)
- The **extension point** is written inside the base case **Pay bill**; the extending class **Defer payment** adds the behavior of this extension point. (extend)

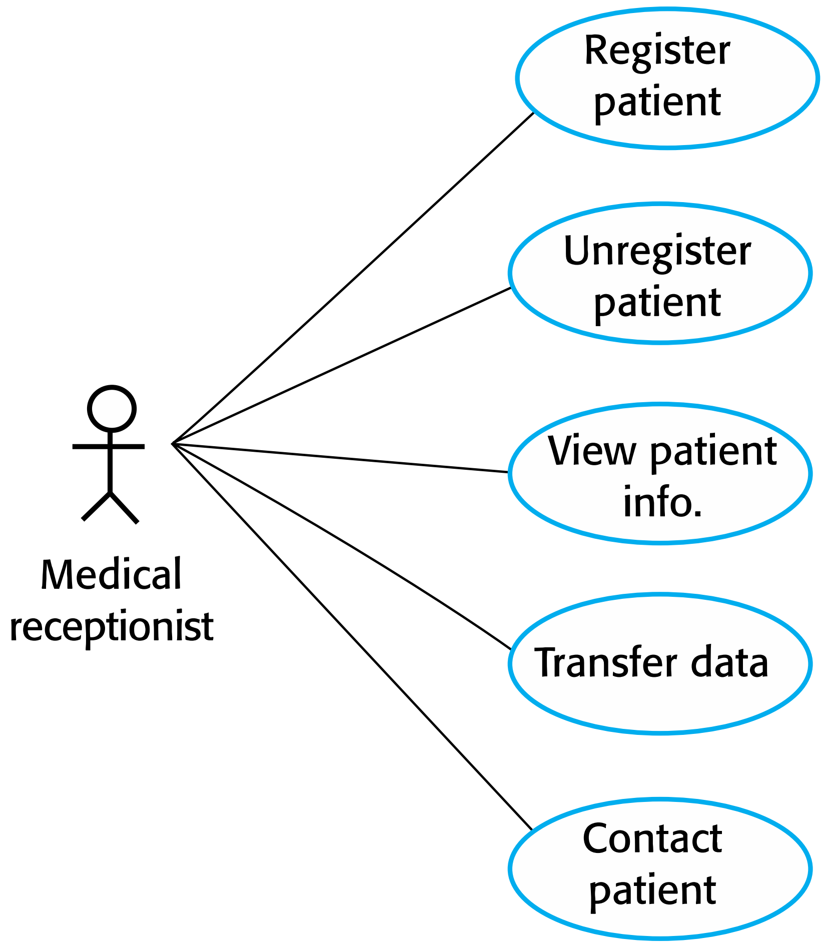
Transfer-data use case

- A use case in the Mentcare system

Tabular description of the ‘Transfer data’ use-case (Use case Narrative)

MHC-PMS: Transfer data	
Actors	Medical receptionist, patient records system (PRS)
Description	A receptionist may **transfer** data from the <u>Mentcase system</u> to a <u>general patient record database</u> that is maintained by a health authority. The information transferred may either be updated personal information (address, phone number, etc.) or a summary of the patient’s diagnosis and treatment.
Data	Patient’s personal information, treatment summary
Stimulus	User command issued by medical receptionist
Response	Confirmation that PRS has been updated
Comments	The receptionist must have appropriate security permissions to access the patient information and the PRS.

Use cases in the Mentcare system involving the role 'Medical Receptionist'



Behavioral models

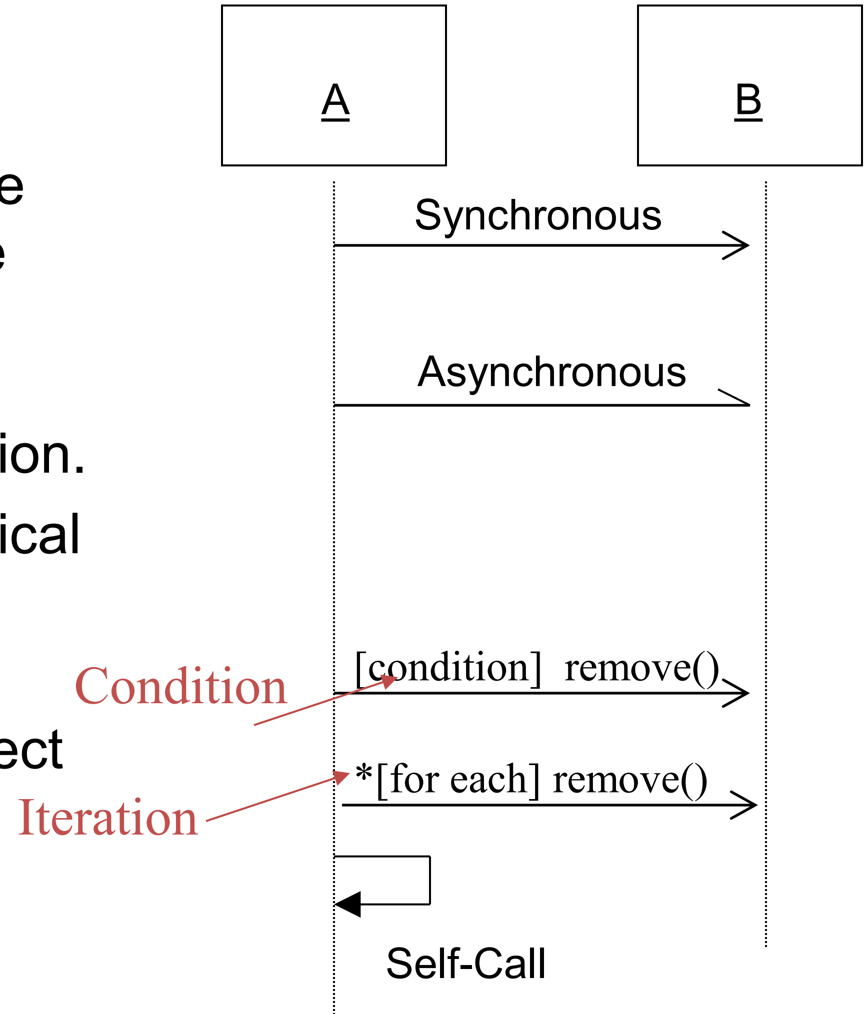
- Behavioral models are models of the **dynamic** behavior of a system as it is executing. They show what happens or what is supposed to happen when a system responds to a stimulus from its environment.
- You can think of these stimuli as being of two types:
  - Data. Some data arrives that has to be processed by the system.
  - Events. Some event happens that triggers system processing. Events may have associated data, although this is not always the case.

Sequence diagrams

- Sequence diagrams are part of the UML and are used to model the interactions between the **actors** and the **objects** within a system.
- A sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance.
- The **objects** and **actors** involved are listed along the top of the diagram, with a dotted line drawn vertically from these.
- Interactions between objects are indicated by annotated arrows.

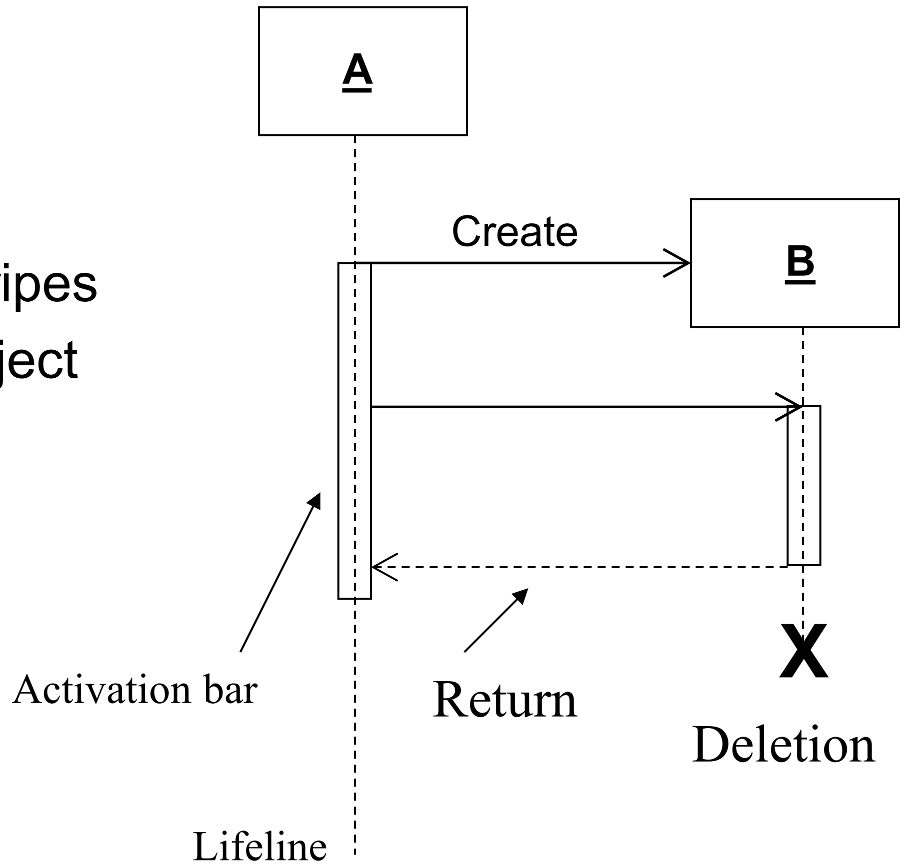
Sequence Diagram: Object interaction

- A **sequence diagram** displays an interaction as a two-dimensional chart. The vertical dimension is the time axis; time proceeds down the page. The horizontal dimension shows the roles that represent individual objects in the collaboration.
- Each role is represented by a vertical column containing a head symbol and a vertical line—a lifeline.
- **Self-Call:** A message that an Object sends to itself.
- **Condition**: indicates when a message is sent. The message is sent only if the condition is true.



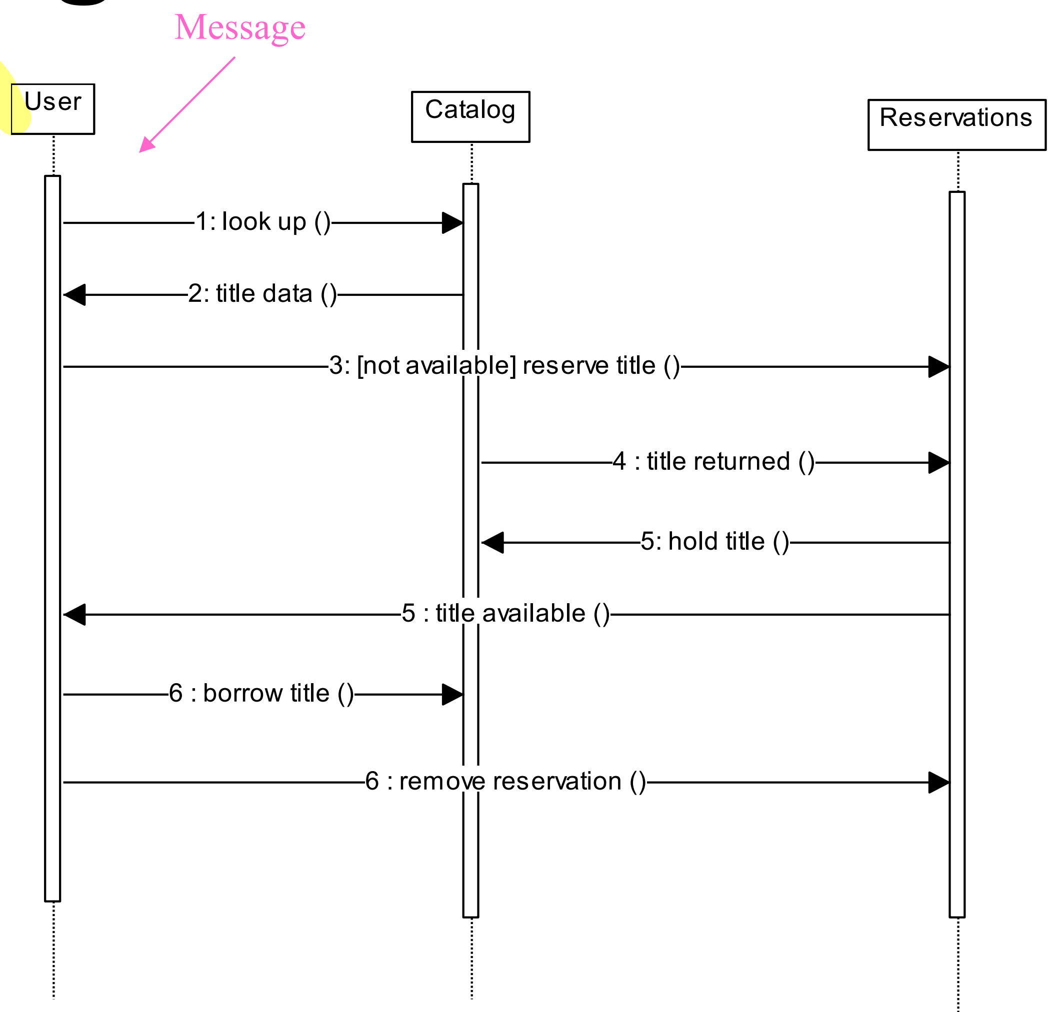
Sequence Diagrams – Object Life Spans

- Creation
  - Create message
  - Object life starts at that point
- Activation
  - Symbolized by rectangular stripes
  - Place on the lifeline where object is activated.
  - Rectangle also denotes when object is deactivated.
- Deletion
  - Placing an ‘X’ on lifeline
  - Object’s life ends at that point



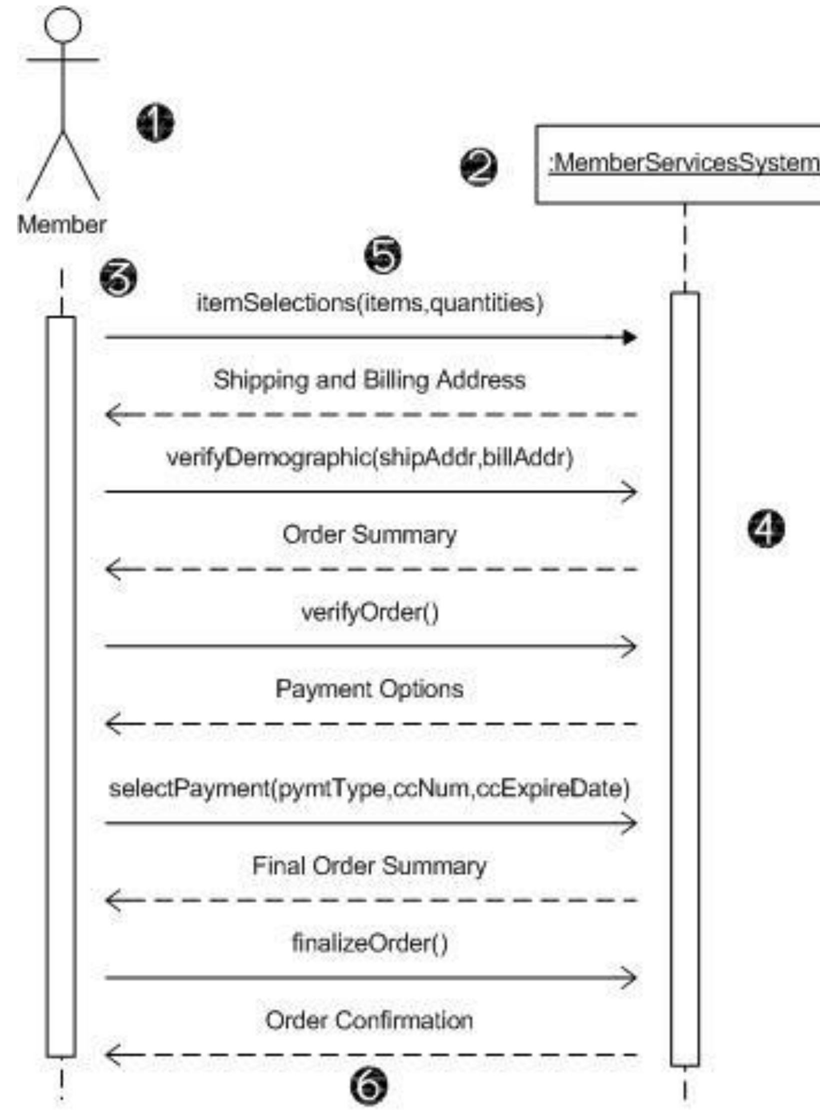
Sequence Diagram

- Sequence diagrams demonstrate the behavior of objects in a use case
- by describing the objects and the messages they pass.
- The horizontal dimension shows the objects participating in the interaction.
- The vertical arrangement of messages indicates their order.
- The labels may contain the seq. # to indicate concurrency.



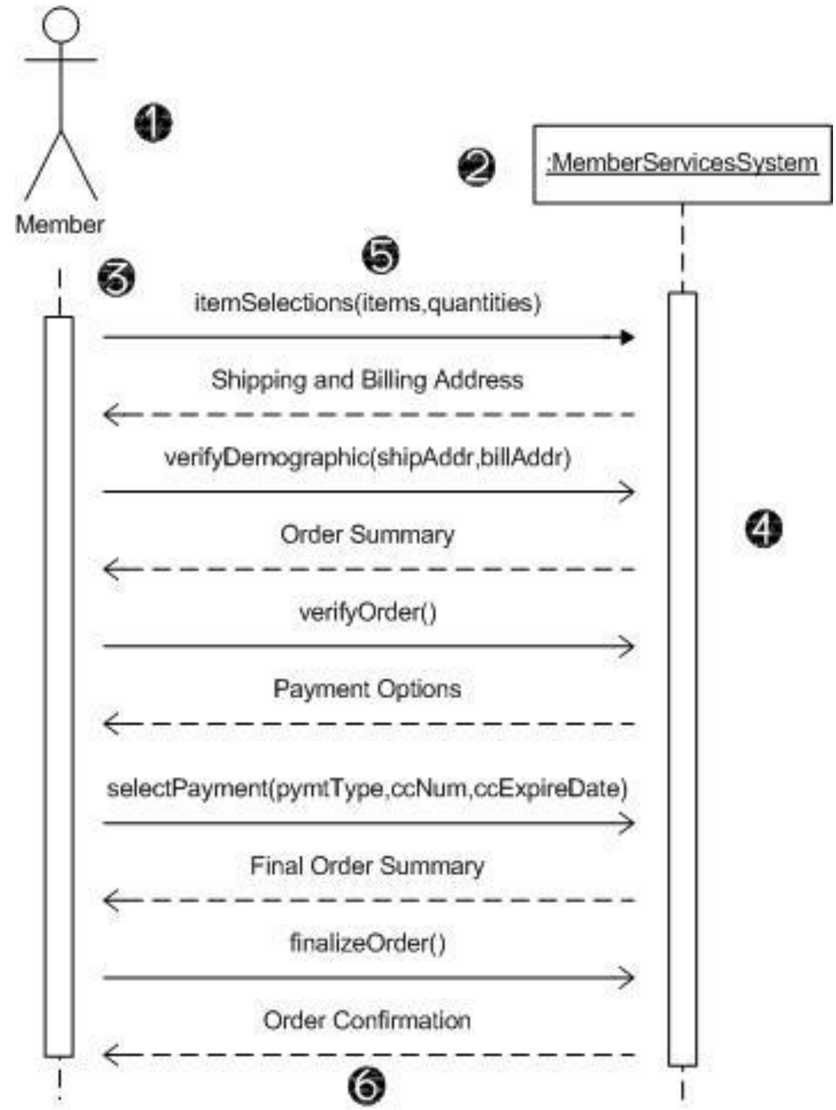
System Sequence Diagram Notations

1. **Actor** - the initiating actor of the use case is shown with the use case actor symbol.
2. **System** – the box indicates the system as a "black box" or as a whole. The colon (:) is standard sequence diagram notation to indicate a running "instance" of the system.
3. **Lifelines** – the dashed vertical lines extending downward from the actor and system symbols, which indicate the life of the sequence.
4. **Activation bars** – the bars set over the lifelines indicate period of time when participant is active in the interaction.

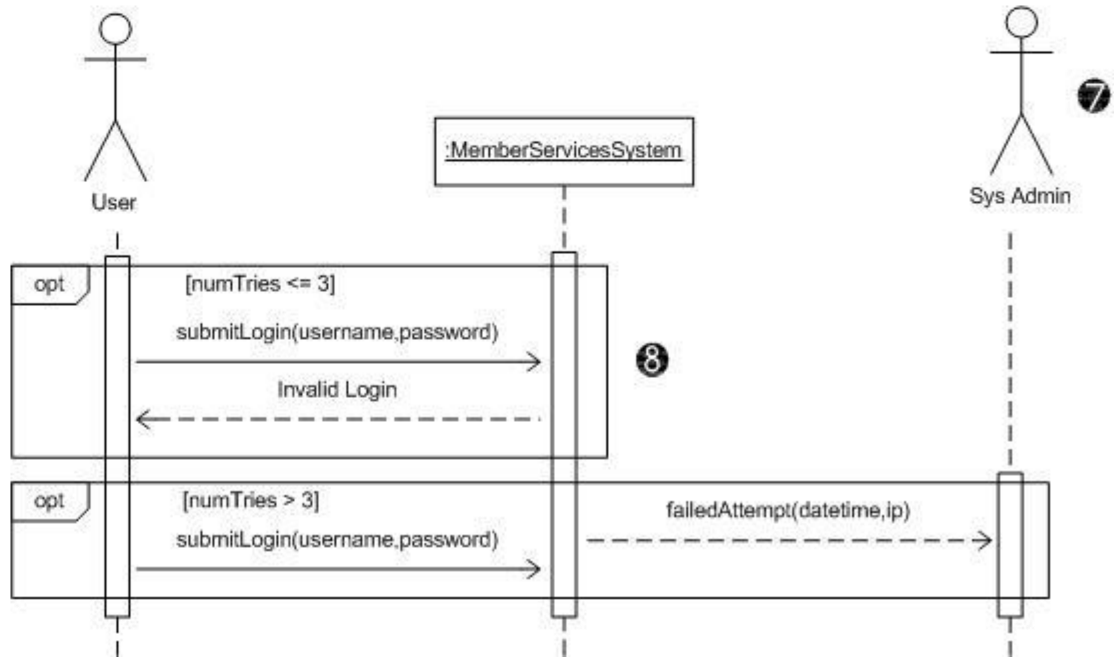


System Sequence Diagram Notations (cont.)

5. **Input messages** - horizontal arrows from actor to system indicate the message inputs. UML convention for messages is to begin the first word with a lowercase letter and add additional words with initial uppercase letter and no space. In parentheses include parameters, following same naming convention and separated with commas.
6. **Output messages** – horizontal arrows from system to actor shown as dashed lines. Since they are web forms, reports, e-mails, etc. these messages do not need to use the standard notation.

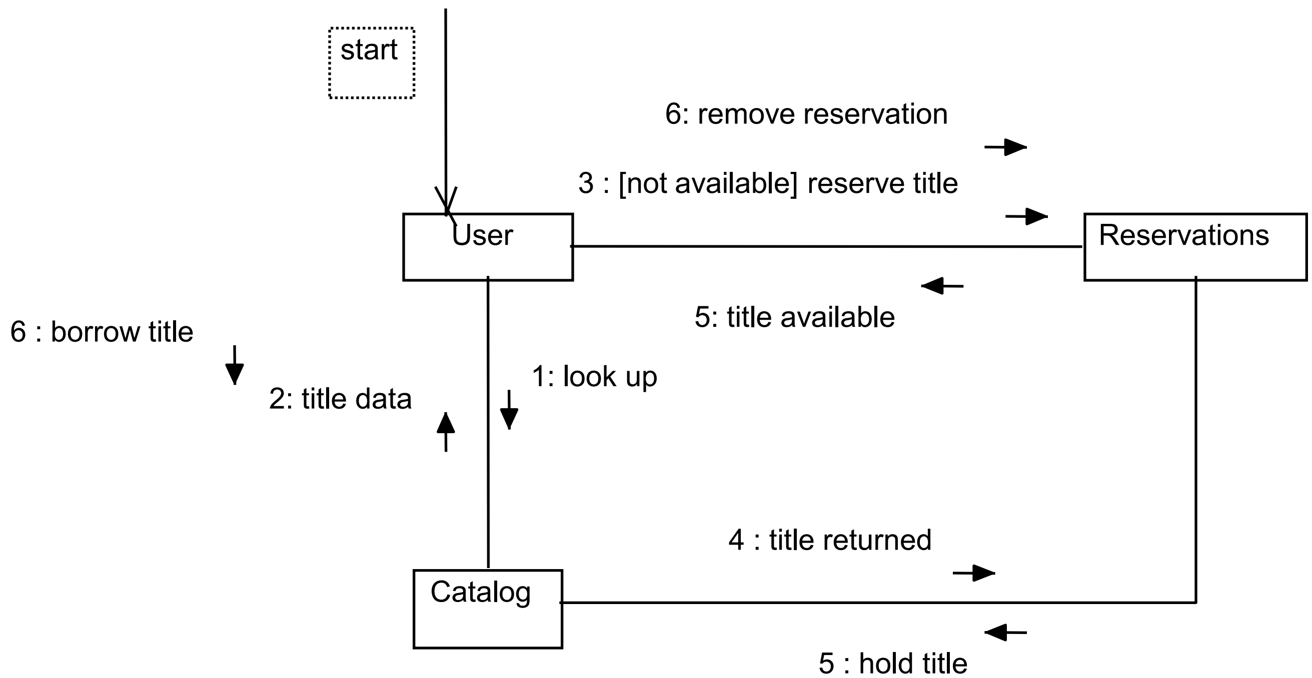


System Sequence Diagram Notations (cont.)



7. **Receiver Actor** – other actors or external systems that receive messages from the system can be included.
8. **Frame** – a box can enclose one or more messages to divide off a fragment of the sequence. These can show loops, alternate fragments, or optional (opt) steps. For an optional fragment the condition shown in square brackets indicates the conditions under which the steps will be performed.

Interaction Diagrams: Collaboration diagrams



- Shows the relationship between objects and the order of messages passed between them.
- The objects are listed as rectangles and arrows indicate the messages being passed
- The numbers next to the messages are called sequence numbers. They show the sequence of the messages as they are passed between the objects.
- convey the same information as sequence diagrams, but focus on object roles instead of the time sequence.

Event-driven modeling

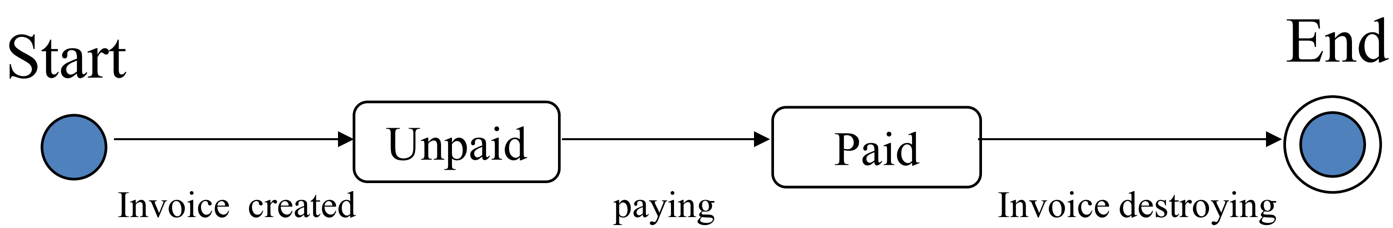
- Real-time systems are often event-driven, with minimal data processing. For example, a landline phone switching system responds to events such as ‘receiver off hook’ by generating a dial tone.
- Event-driven modeling shows how a system responds to external and internal events.
- It is based on the assumption that a system has a finite number of states and that events (stimuli) may cause a transition from one state to another.

State machine models

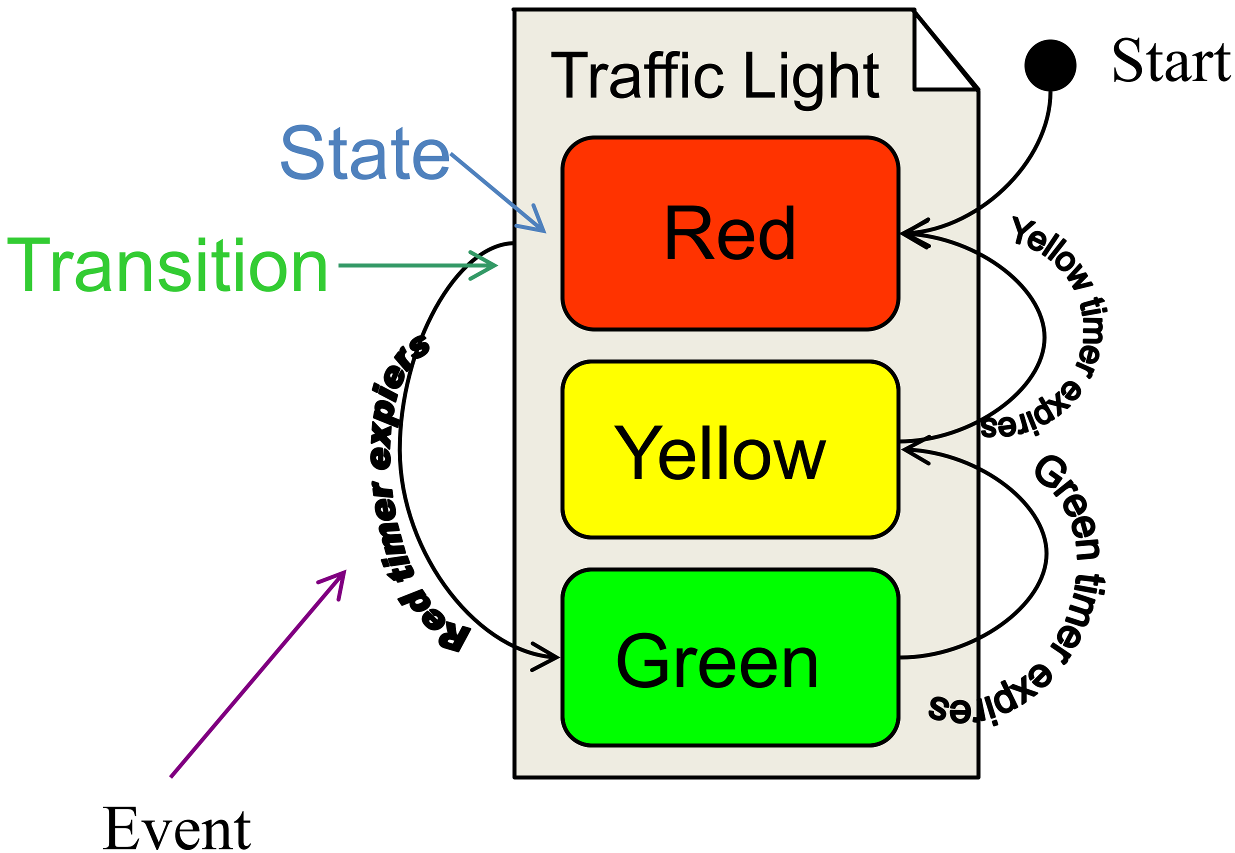
- These model the behaviour of the system in response to external and internal events.
- They show the system's responses to stimuli so are often used for modelling real-time systems.
- State machine models show system states as **nodes** and events as **arcs** between these nodes. When an event occurs, the system moves from one state to another.
- State-charts are an integral part of the UML and are used to represent state machine models.

State Diagrams (Billing Example)

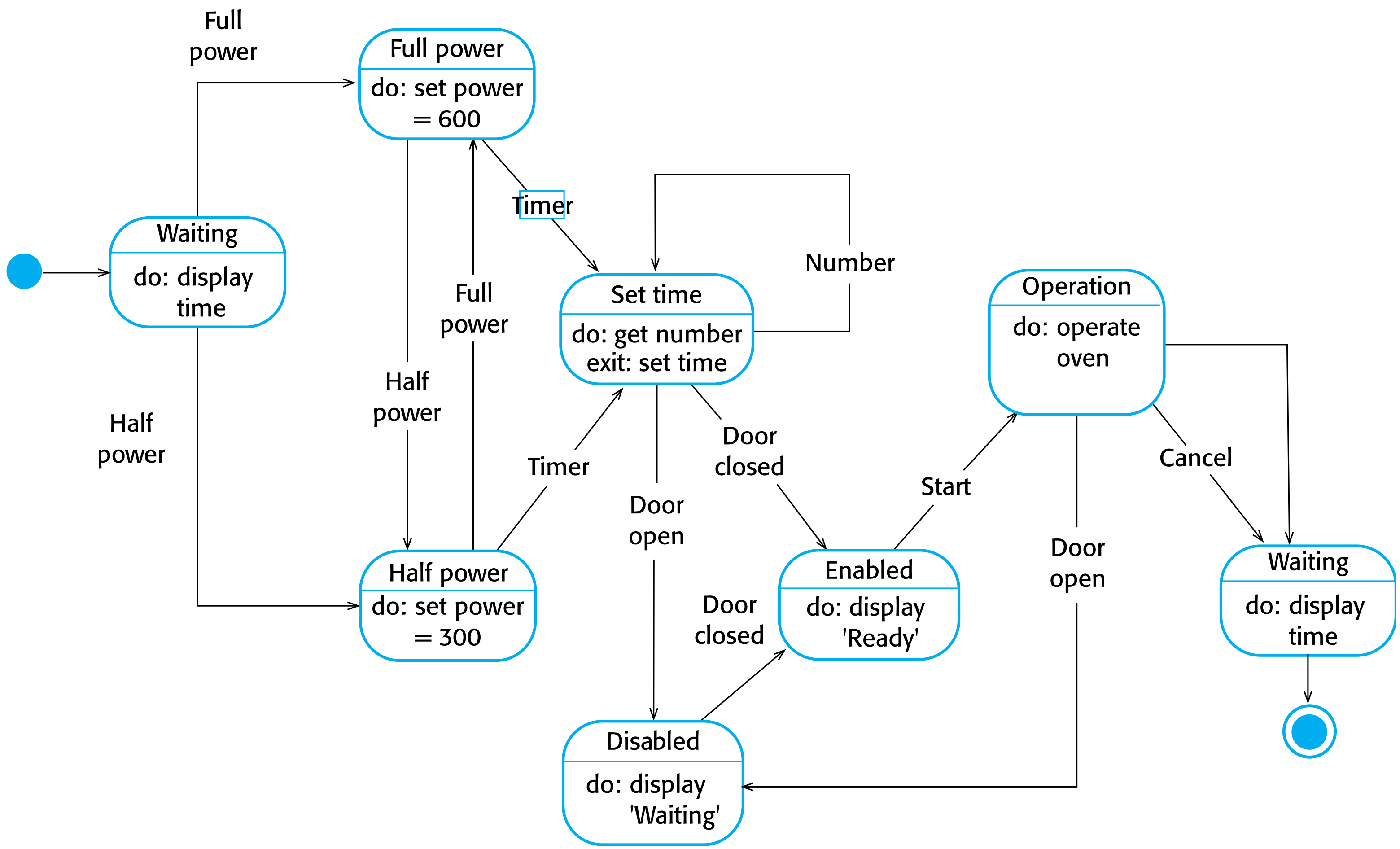
- State Diagrams show the sequences of states an object goes through during its life cycle in response to stimuli, together with its responses and actions; an abstraction of all possible behaviors.



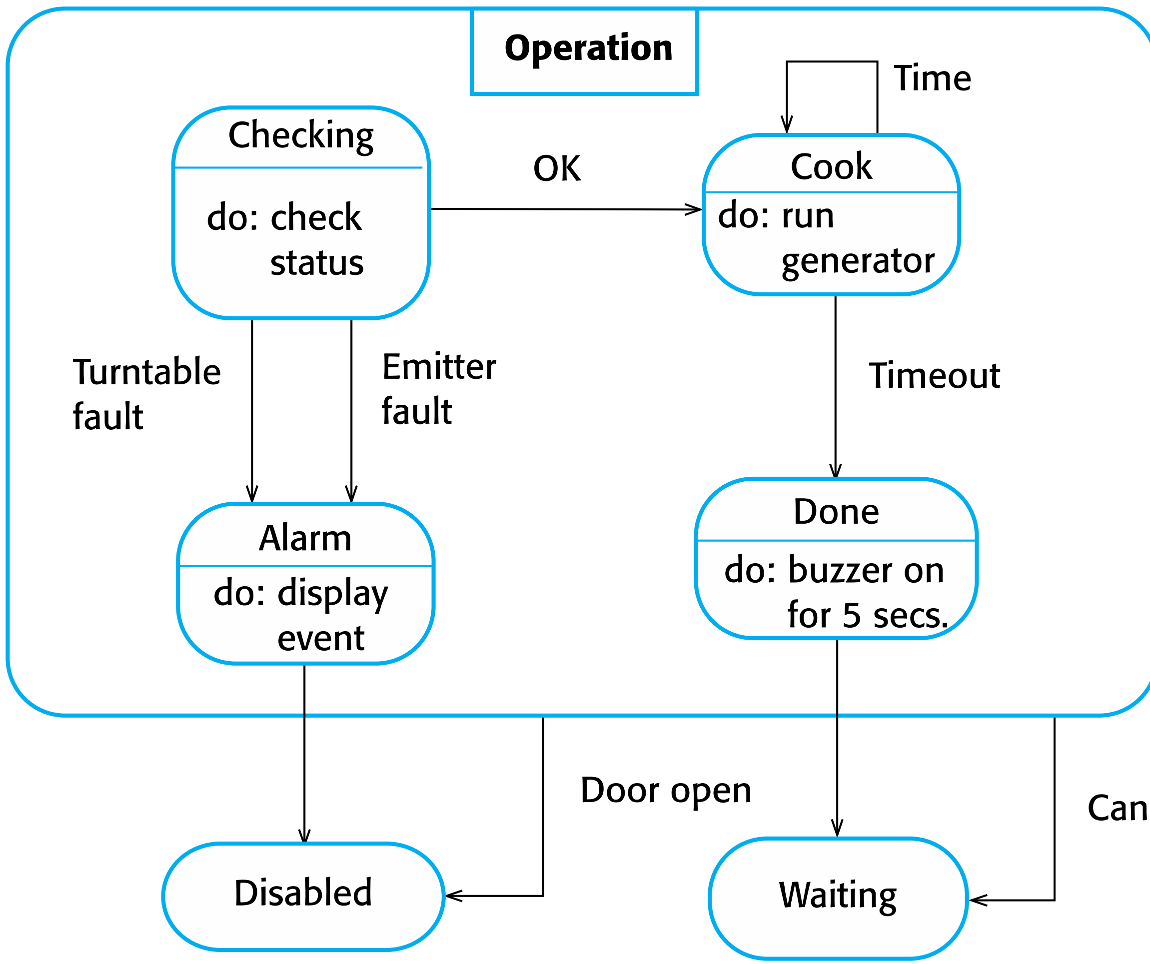
State Diagrams (Traffic light example)



State diagram of a microwave oven



Microwave oven operation



States and stimuli for the microwave oven (a)

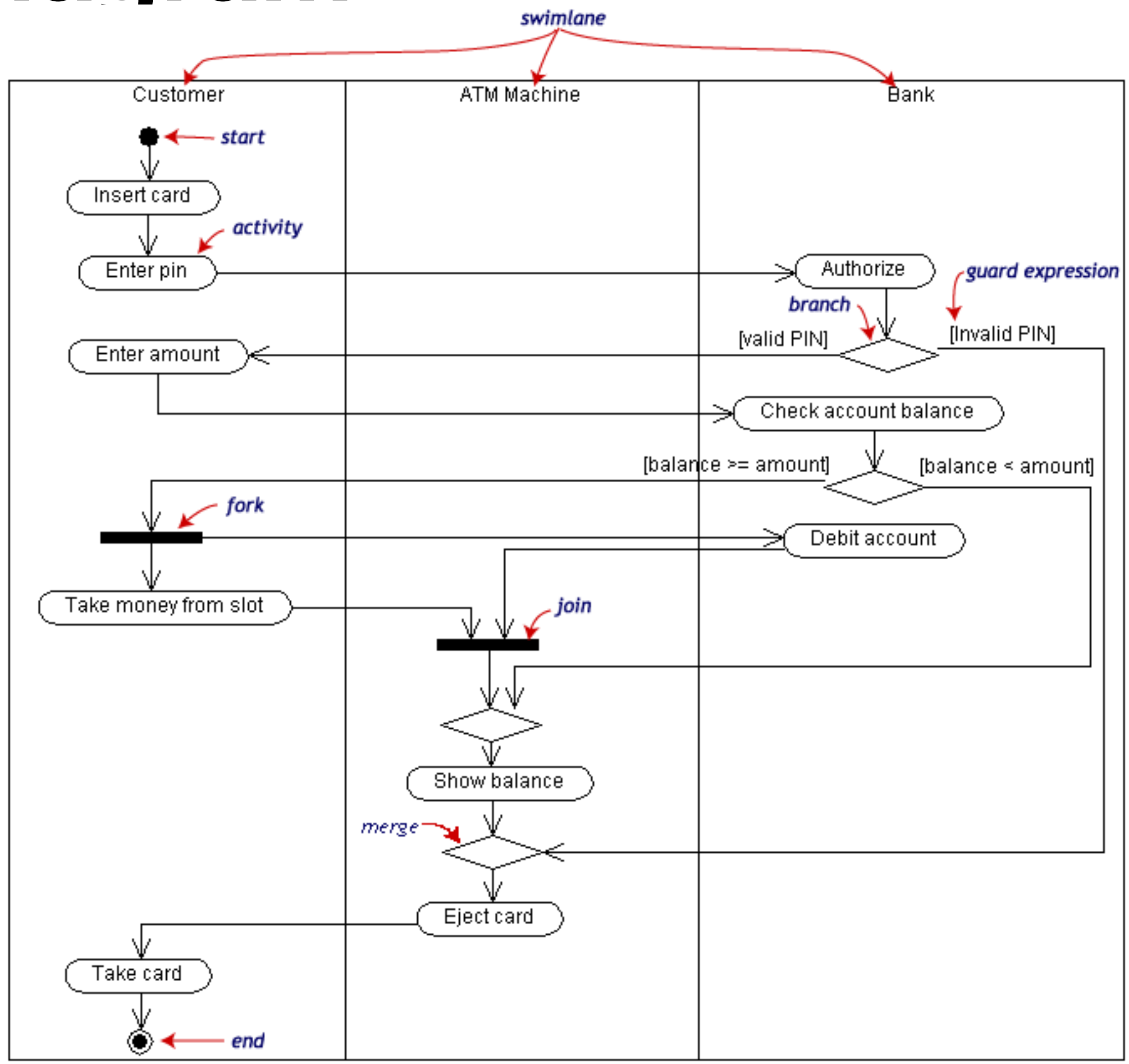
State	Description
Waiting	The oven is waiting for input. The display shows the current time.
Half power	The oven power is set to 300 watts. The display shows ‘Half power’.
Full power	The oven power is set to 600 watts. The display shows ‘Full power’.
Set time	The cooking time is set to the user’s input value. The display shows the cooking time selected and is updated as the time is set.
Disabled	Oven operation is disabled for safety. Interior oven light is on. Display shows ‘Not ready’.
Enabled	Oven operation is enabled. Interior oven light is off. Display shows ‘Ready to cook’.
Operation	Oven in operation. Interior oven light is on. Display shows the timer countdown. On completion of cooking, the buzzer is sounded for five seconds. Oven light is on. Display shows ‘Cooking complete’ while buzzer is sounding.

States and stimuli for the microwave oven (b)

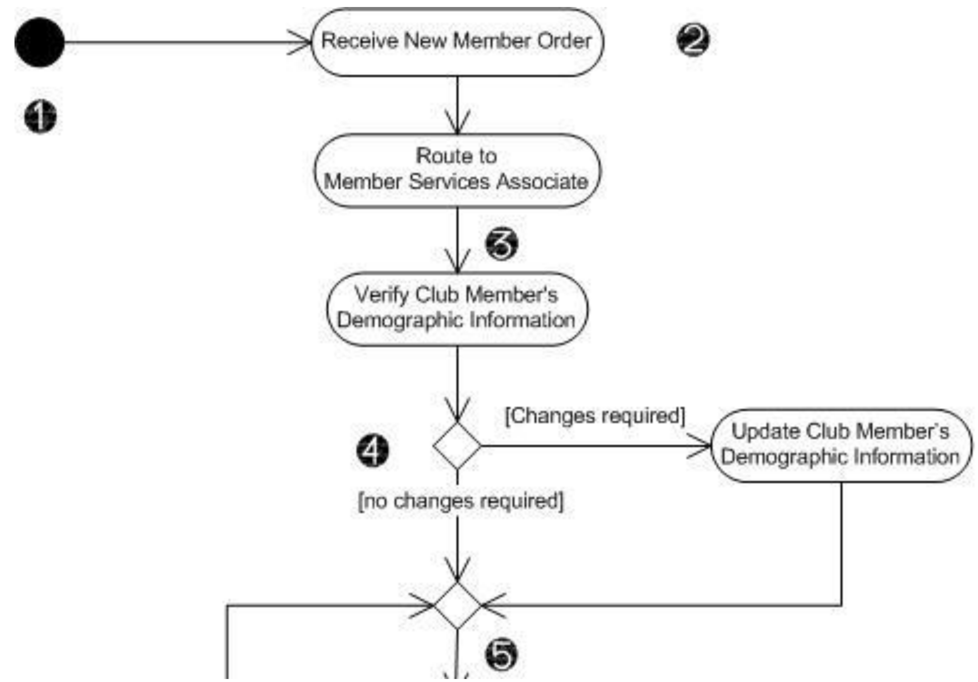
Stimulus	Description
Half power	The user has pressed the half-power button.
Full power	The user has pressed the full-power button.
Timer	The user has pressed one of the timer buttons.
Number	The user has pressed a numeric key.
Door open	The oven door switch is not closed.
Door closed	The oven door switch is closed.
Start	The user has pressed the Start button.
Cancel	The user has pressed the Cancel button.

Activity Diagram

- Built during analysis and design
- Purpose
  - Model business workflows
  - Model operations
  - steps of a use case
- Developed by analysts, designers and implementers



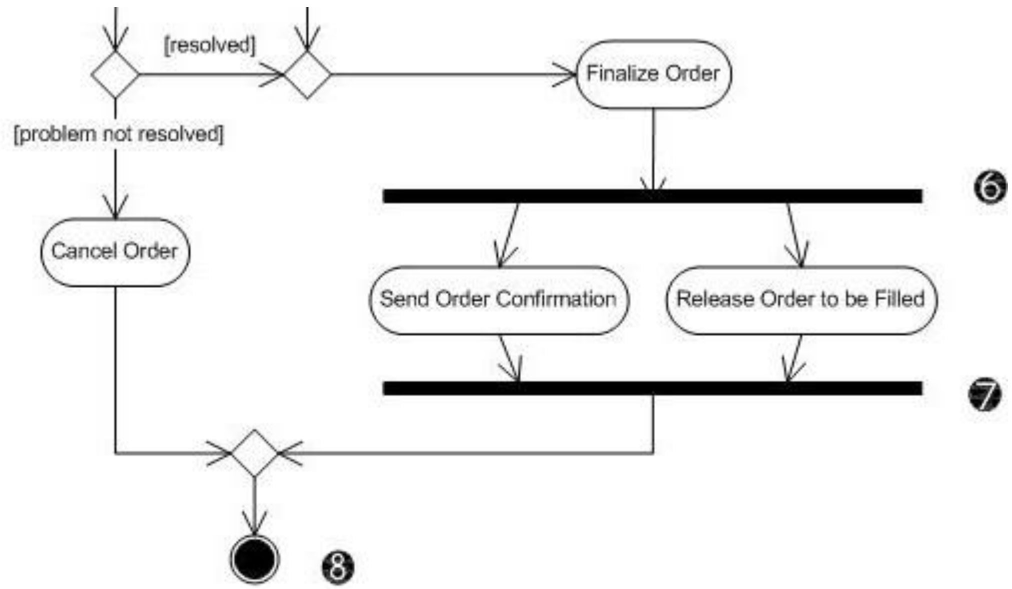
Activity Diagram Notations



1. **Initial node** - solid circle representing the start of the process.
2. **Actions** – rounded rectangles representing individual steps. The sequence of actions make up the total activity shown by the diagram.
3. **Flow** - arrows on the diagram indicating the progression through the actions. Most flows do not need words to identify them unless coming out of decisions.
4. **Decision** - diamond shapes with one flow coming in and two or more flows going out. The flows coming out are marked to indicate the conditions.
5. **Merge** - diamond shapes with multiple flows coming in and one flow going out. This combines flows previously separated by decisions. Processing continues with any one flow coming into the merge.

Activity Diagram Notations (cont.)

6. **Fork** – a black bar with one flow coming in and two or more flows going out. Actions on parallel flows beneath the fork can occur in any order or concurrently.



7. **Join** – a black bar with two or more flows coming in and one flow going out, noting the end of concurrent processing. All actions coming into the join must be completed before processing continues.
8. **Activity final** – the solid circle inside the hollow circle representing the end of the process.

Structural models

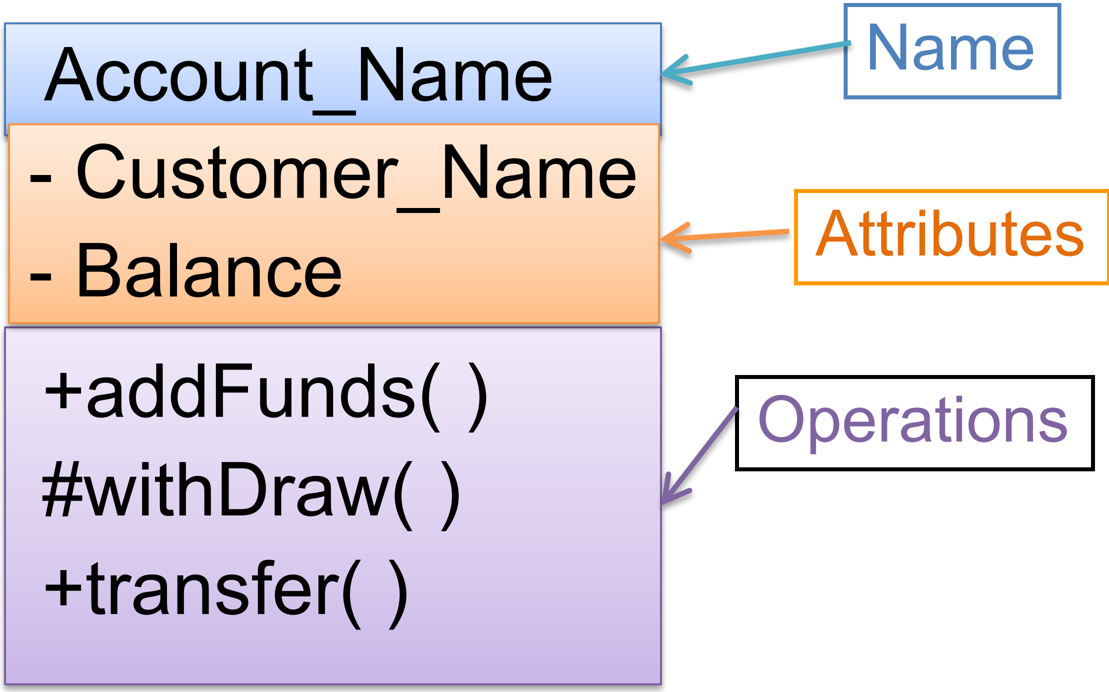
- Structural models of software display the **organization** of a system in terms of the **components** that make up that system and their relationships.
- Structural models may be **static** models, which show the structure of the system design, or **dynamic** models, which show the organization of the system when it is executing.
- You create structural models of a system when you are discussing and designing the system architecture.

Class diagrams

- Class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes.
- A class is a collection of objects with common structure, common behaviour, common relationships and common semantics.
- Classes are found by examining the objects in sequence and collaboration diagram
- A class is drawn as a rectangle with three compartments
- Classes should be named using the vocabulary of the domain
  - Naming standards should be created
  - e.g., all classes are singular nouns starting with a capital letter
- Detailed class diagrams are used for developers

Class representation

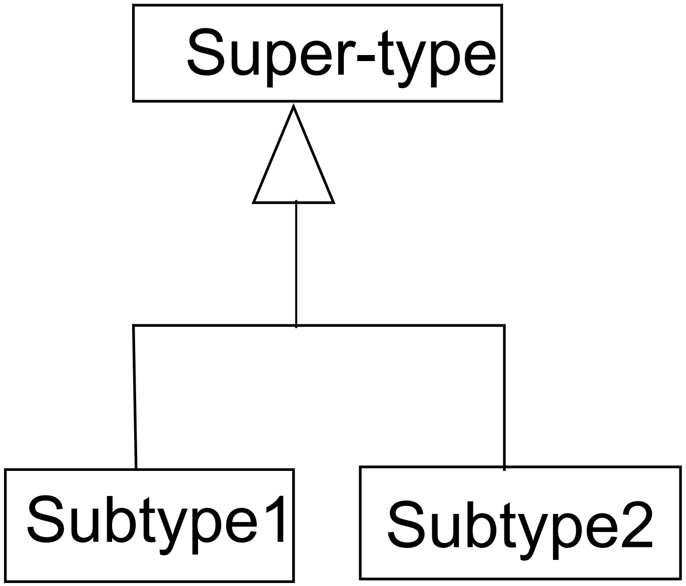
- Each class is represented by a rectangle subdivided into 3 compartments
  - Name
  - Attributes
  - Operations
- Modifiers are used to indicate visibility of attributes and operations.
  - '+' is used to denote *Public* visibility (everyone)
  - '#' is used to denote *Protected* visibility (friends and derived)
  - '-' is used to denote *Private* visibility (no one)
- By default, attributes are hidden and operations are visible.



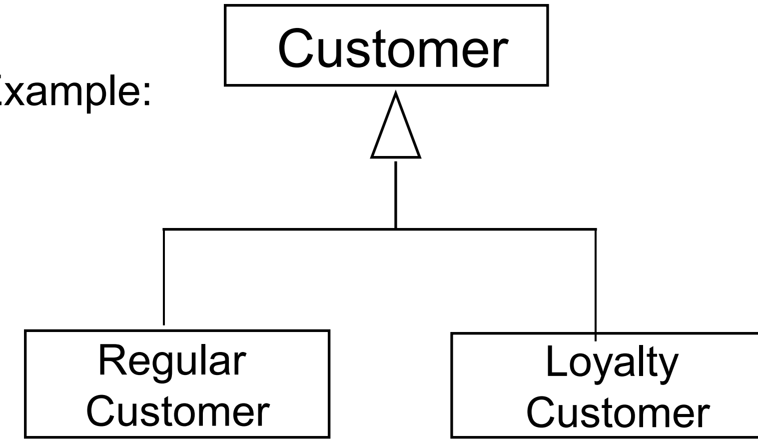
OO Relationships

- There are two kinds of Relationships
  - Generalization (parent-child relationship)
  - Association (student enrolls in course)
- Associations can be further classified as
  - Aggregation
  - Composition

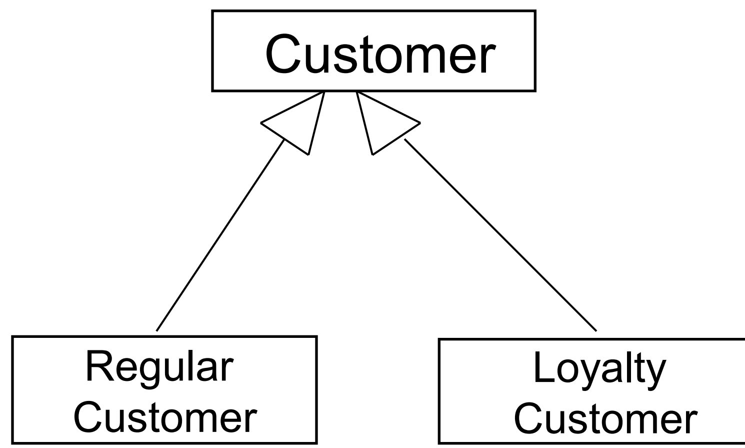
OO Relationships: Generalization



Example:



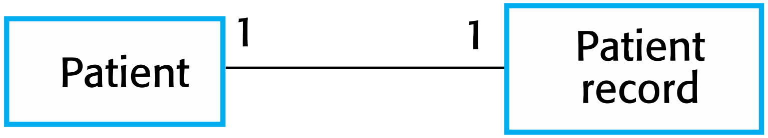
or:



- - Generalization expresses a parent/child relationship among related classes.
- - Used for abstracting details in several layers

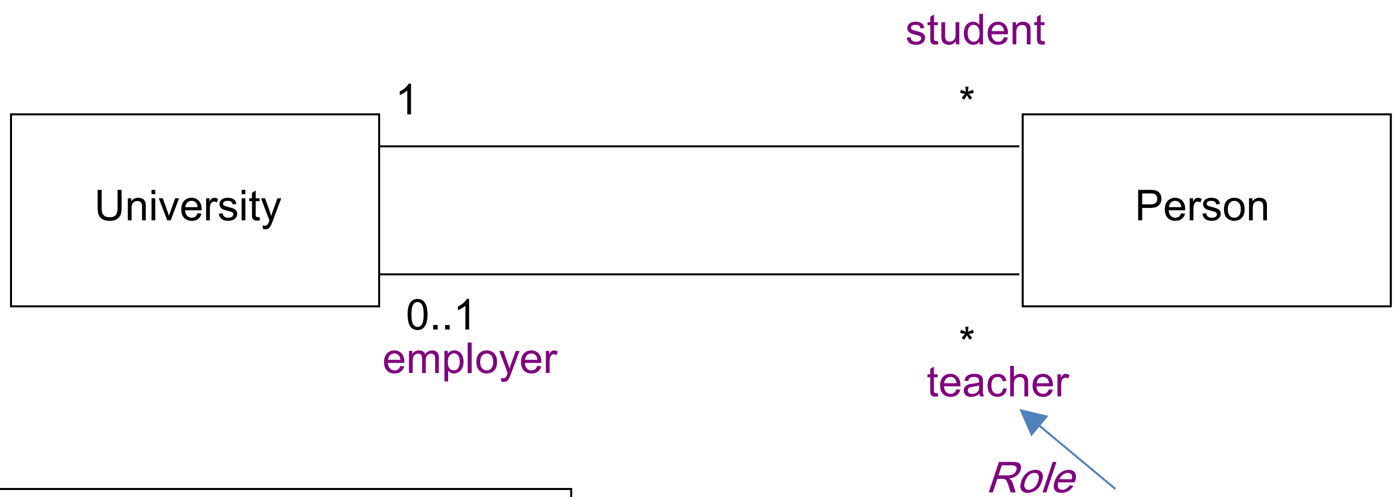
OO Relationships: Association

- Represent relationship between instances of classes
  - Student enrolls in a course
  - Courses have students
  - Courses have exams
  - Etc.



- Association has two ends
  - Role names (e.g. enrolls)
  - Multiplicity (e.g. One course can have many students)
  - Navigability (unidirectional, bidirectional)

Association: Multiplicity and Roles



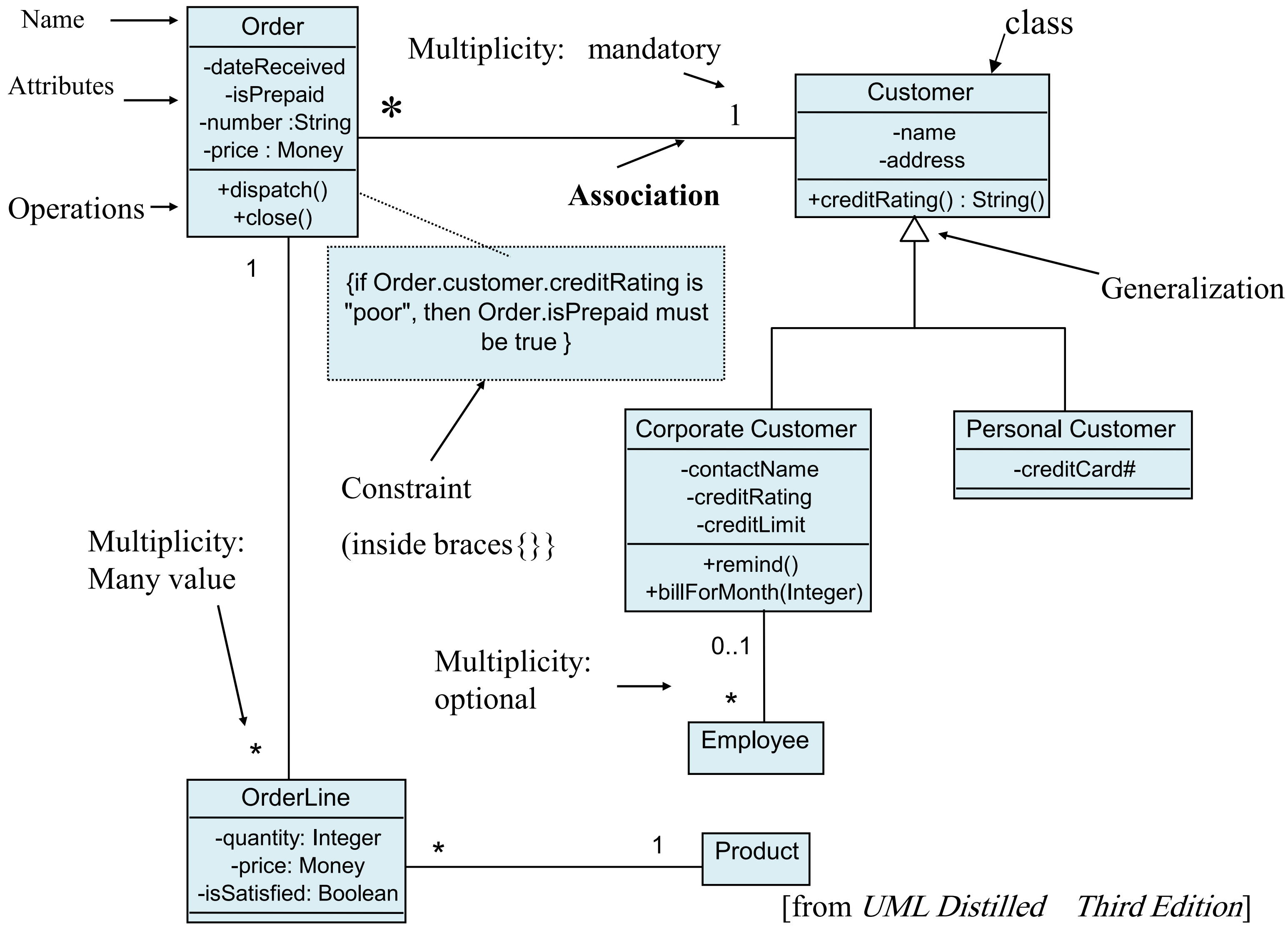
Multiplicity

Symbol	Meaning
1	One and only one
0..1	Zero or one
M..N	From M to N (natural language)
*	From zero to any positive integer
0..*	From zero to any positive integer
1..*	From one to any positive integer

Role

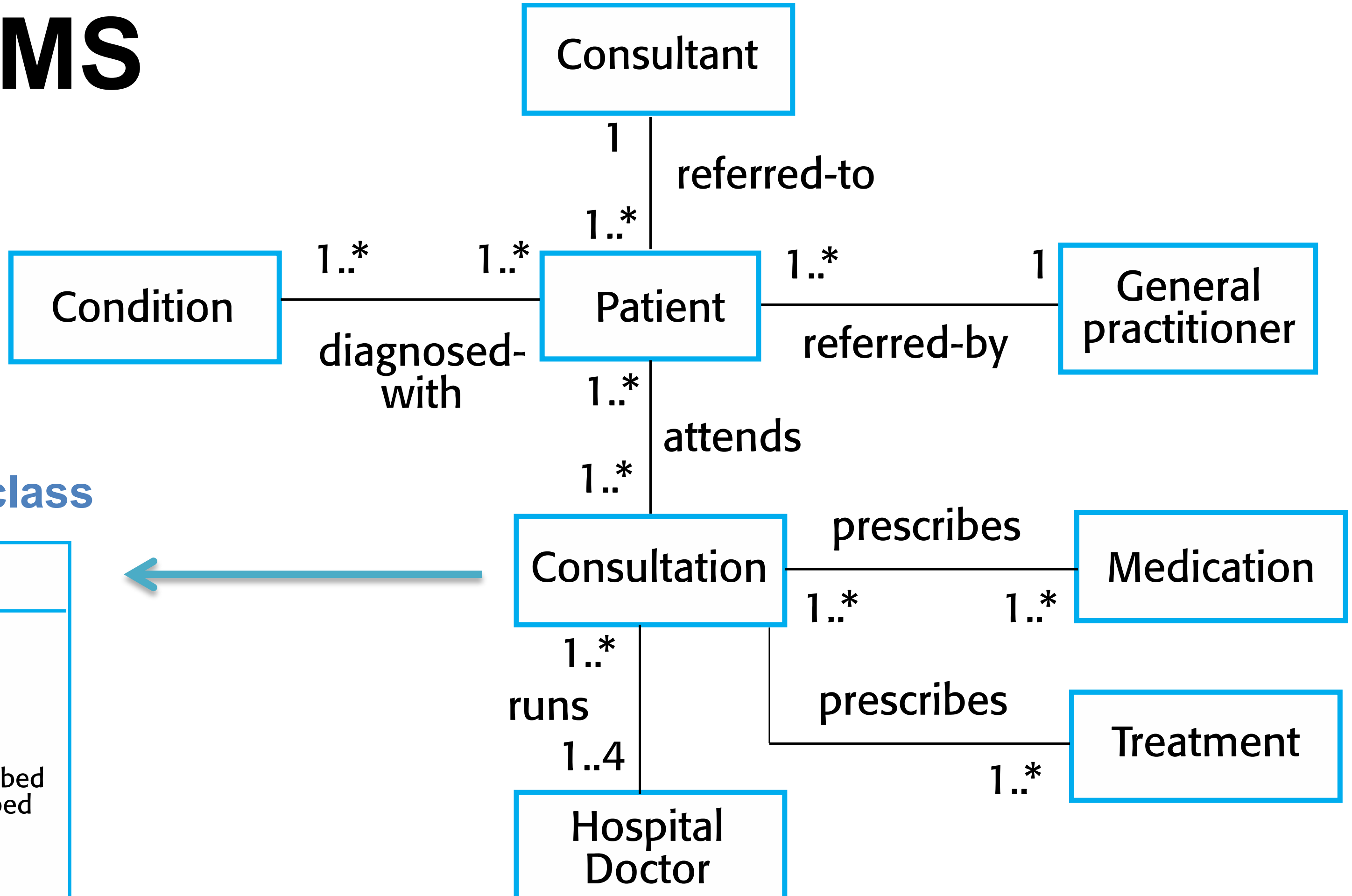
"A given university groups many people; some act as students, others as teachers. A given student belongs to a single university; a given teacher may or may not be working for the university at a particular time."

Class Diagram

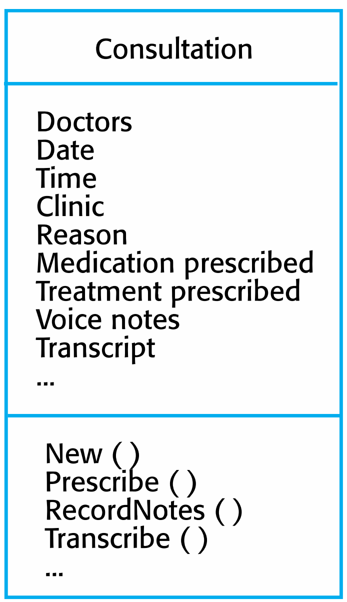


[from *UML Distilled Third Edition*]

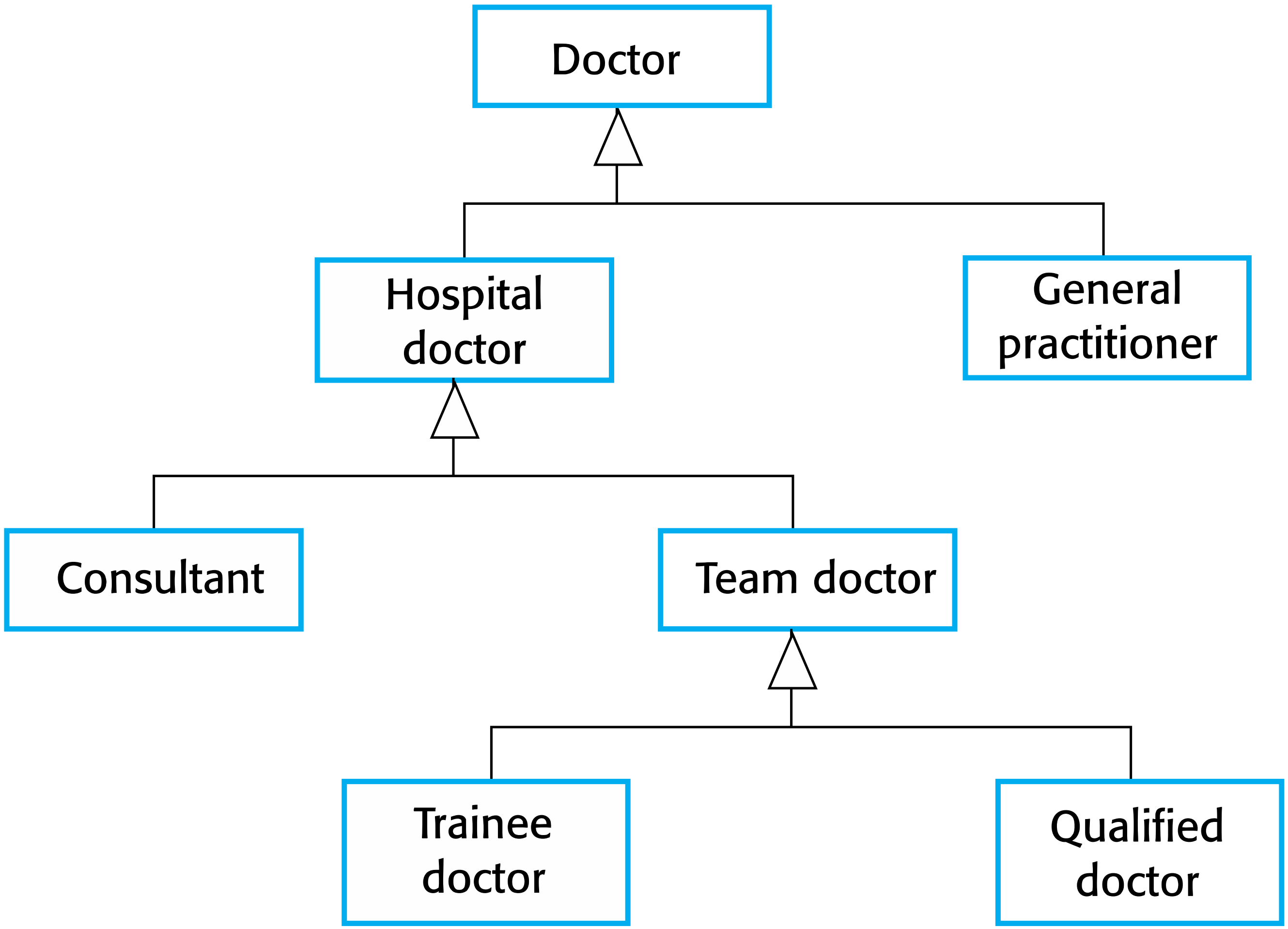
Classes and associations in the MHC-PMS



The Consultation class

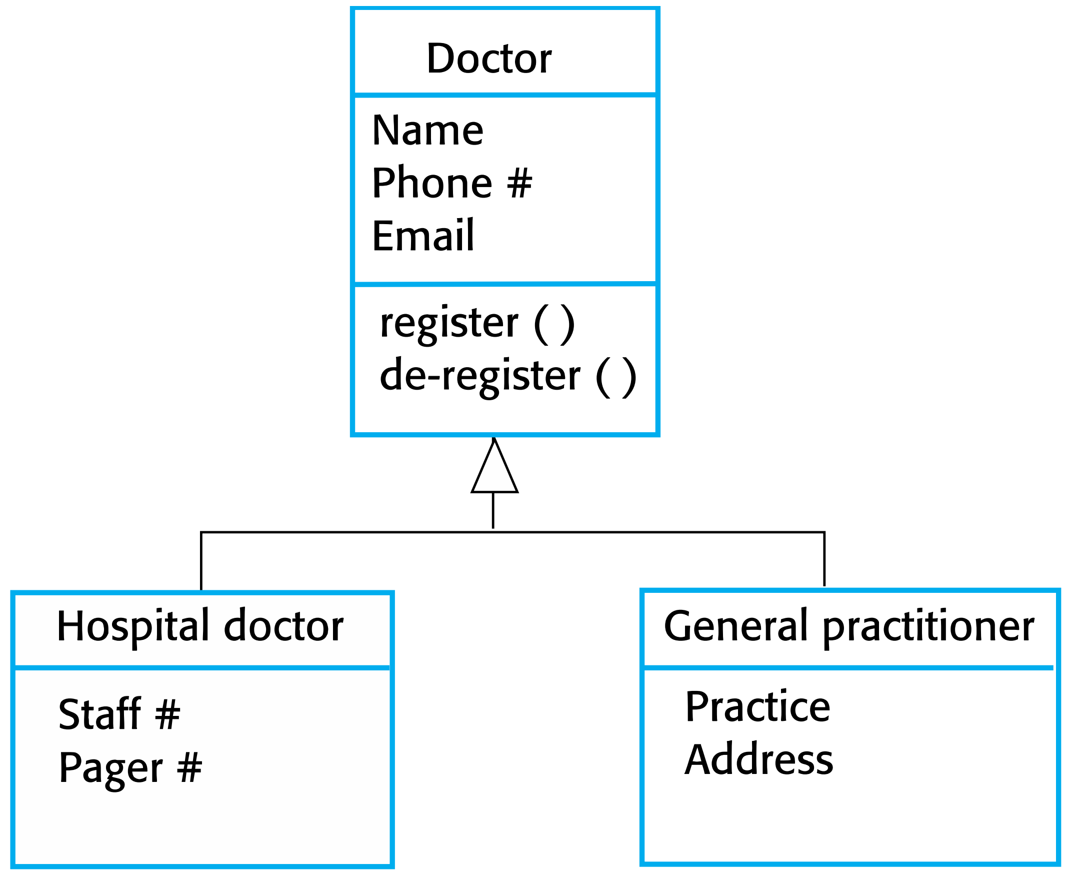


A generalization hierarchy

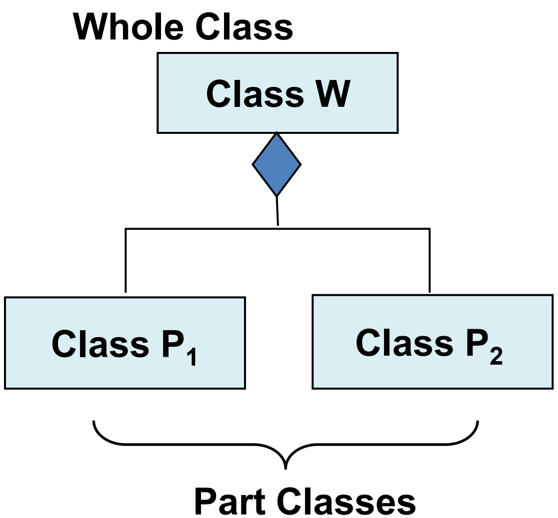


A generalization hierarchy with added detail

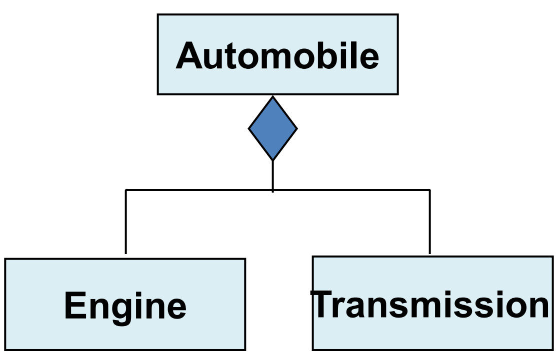
- An aggregation model shows how classes that are collections are composed of other classes.
- Aggregation models are similar to the part-of relationship in semantic data models.



OO Relationships: Composition

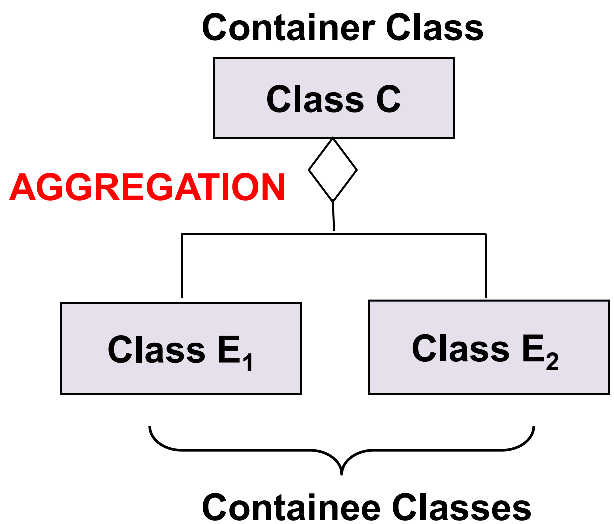


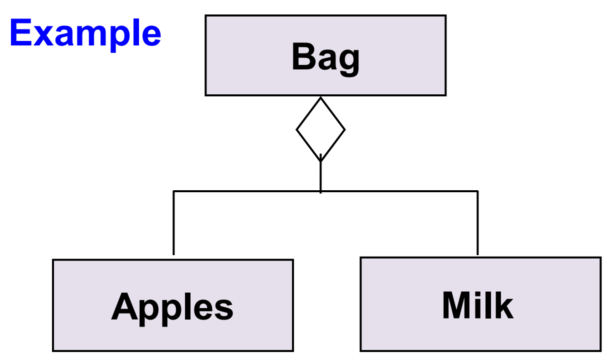
Example



- **Composition:** expresses a relationship among instances
- of related classes. It is a specific kind of Whole-Part relationship.

OO Relationships: Aggregation





- Aggregation: expresses a relationship among instances of related classes. It is a specific kind of Container-Containee relationship.

Aggregation vs. Composition

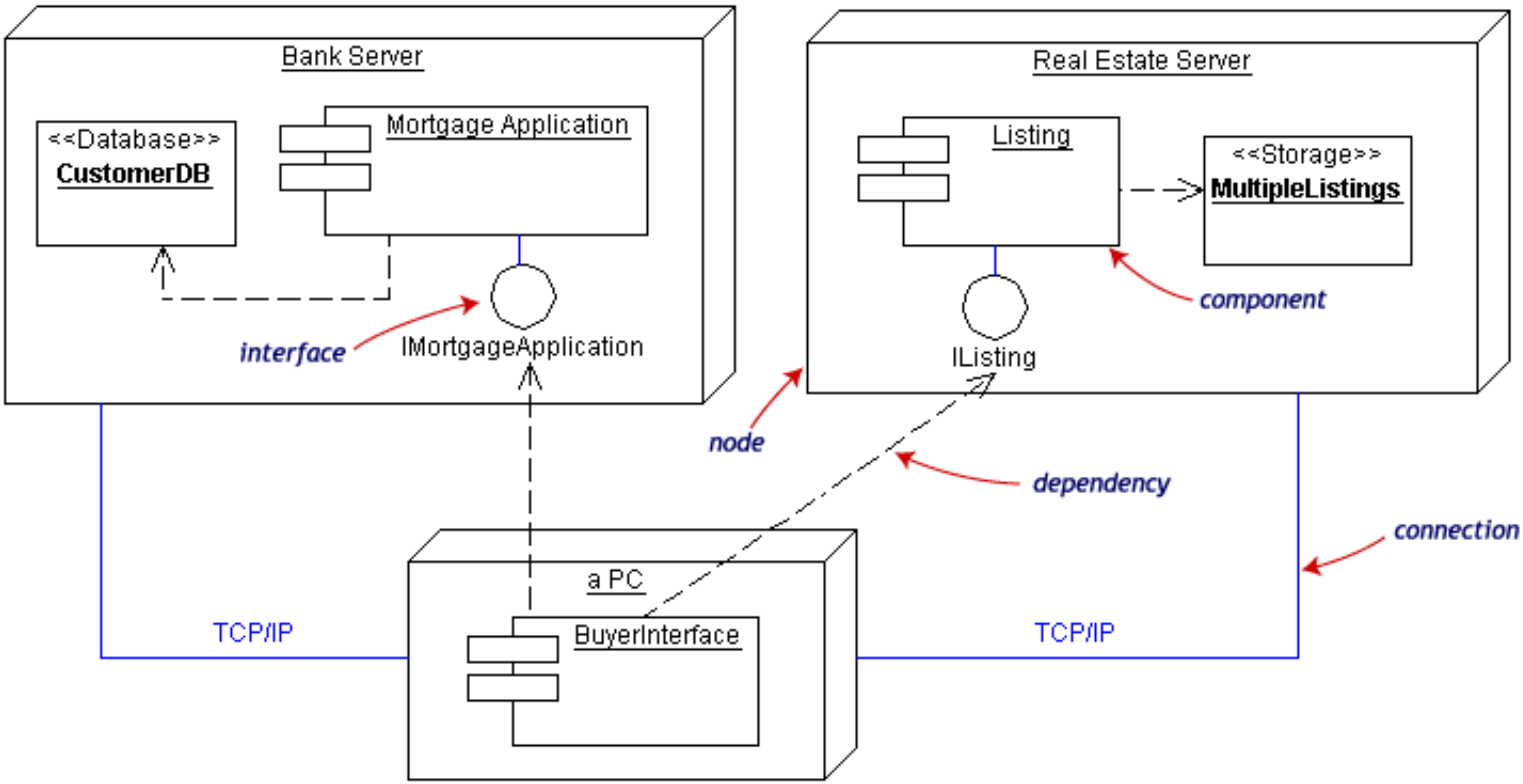
- **Composition** is really a strong form of **aggregation**
  - components have only one owner
  - components cannot exist independent of their owner
  - components live or die with their owner
  - e.g. Each car has an engine that can not be shared with other cars.
- **Aggregations** may form "part of" the aggregate, but may not be essential to it. They may also exist independent of the aggregate.
  - e.g. Apples may exist independent of the bag.

The Physical Model

- The component diagram shows the relationship between software components, their dependencies, communication, location and other conditions.
- A deployment diagram illustrates the physical deployment of the system into a production (or test) environment. It shows where components will be located, on what servers, machines or hardware. It may illustrate network links, LAN bandwidth & etc.

The Physical Model

- This diagram shows the relationships among software and hardware components involved in real estate transactions.



Component Diagram

- Captures the physical structure of the implementation
- Built as part of architectural specification
- Purpose
  - Organize source code
  - Construct an executable release
  - Specify a physical database
- Developed by architects and programmers

Deployment Diagram

- Captures the topology of a system’s hardware
- Built as part of architectural specification
- Purpose
  - Specify the distribution of components
  - Identify performance bottlenecks
- Developed by architects, networking engineers, and system engineers

Summary

- A model is an abstract view of a system that ignores system details. Complementary system models can be developed to show the system’s context, interactions, structure and behaviour.
- Context models show how a system that is being modeled is positioned in an environment with other systems and processes.
- Use case diagrams and sequence diagrams are used to describe the interactions between users and systems in the system being designed. Use cases describe interactions between a system and external actors; sequence diagrams add more information to these by showing interactions between system objects.
- Structural models show the organization and architecture of a system. Class diagrams are used to define the static structure of classes in a system and their associations.
- Behavioral models are used to describe the dynamic behavior of an executing system. This behavior can be modeled from the perspective of the data processed by the system, or by the events that stimulate responses from a system.
- Activity diagrams may be used to model the processing of data, where each activity represents one process step.
- State diagrams are used to model a system’s behavior in response to internal or external events.

References

- **Chapter 5.** I. Sommerville. **Software Engineering**. 10th Edition, Pearson, 2016.