Chapter 5 Part 2 – System Modeling

Topics covered

- Context models
- Interaction models
- Structural models
- Behavioral models

System modeling

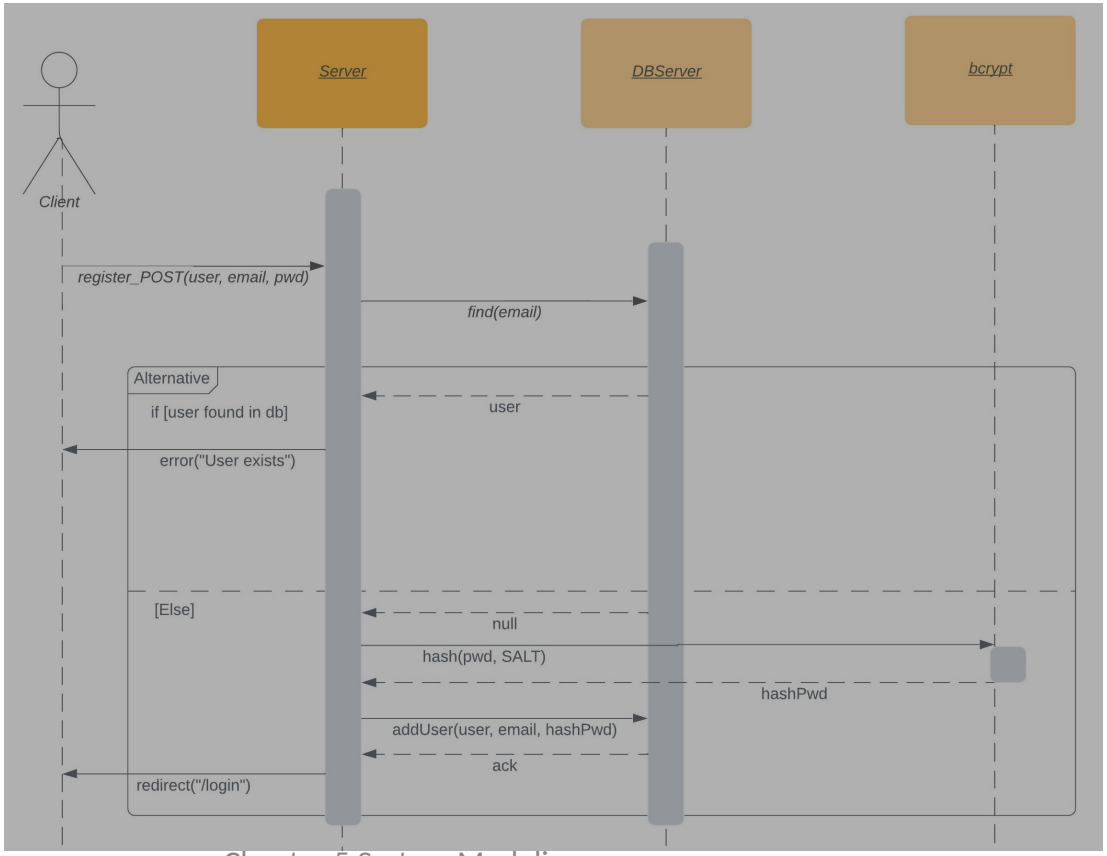
- System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system.
- System modeling involves most of the time a graphical notation, the most widely used one being the Unified Modeling Language (UML).
- System modelling helps the analyst understand the functionality of the system and models are used to better 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.

Example of a model

- Example of a model used to analyze the registration procedure and communicate the requirement to developers:



Different Uses 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.

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

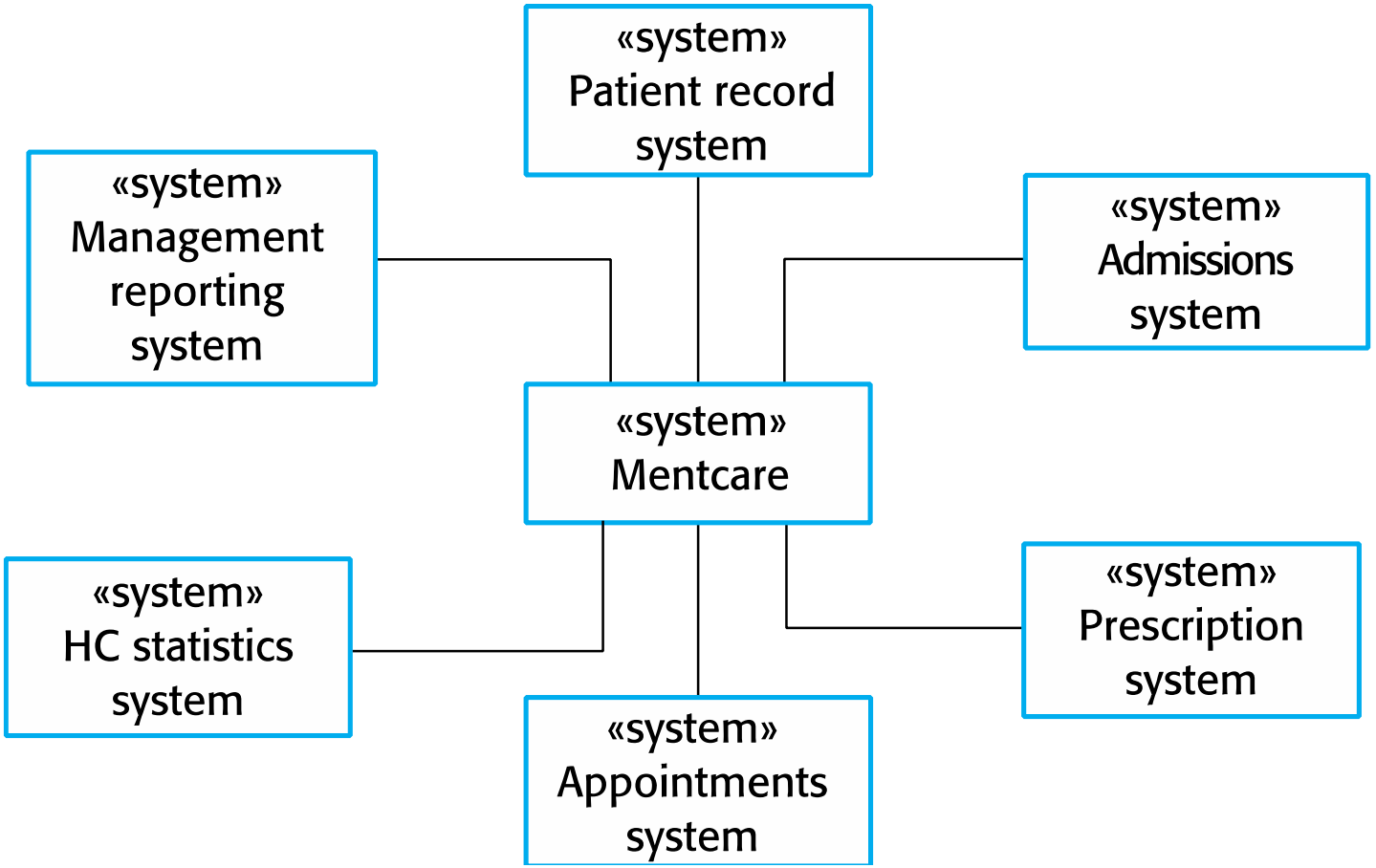
- Activity diagrams, which show the activities involved in a process or in data processing .
- Use case diagrams, which show the interactions between a system and its environment.
- Sequence diagrams, which show interactions between actors and the system and between system components.
- Class diagrams, 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.

Context models

Context models

- Context models are used to illustrate the operational context of a system - they show what lies outside the system boundaries.
- Architectural models show the system and its relationship with other systems.

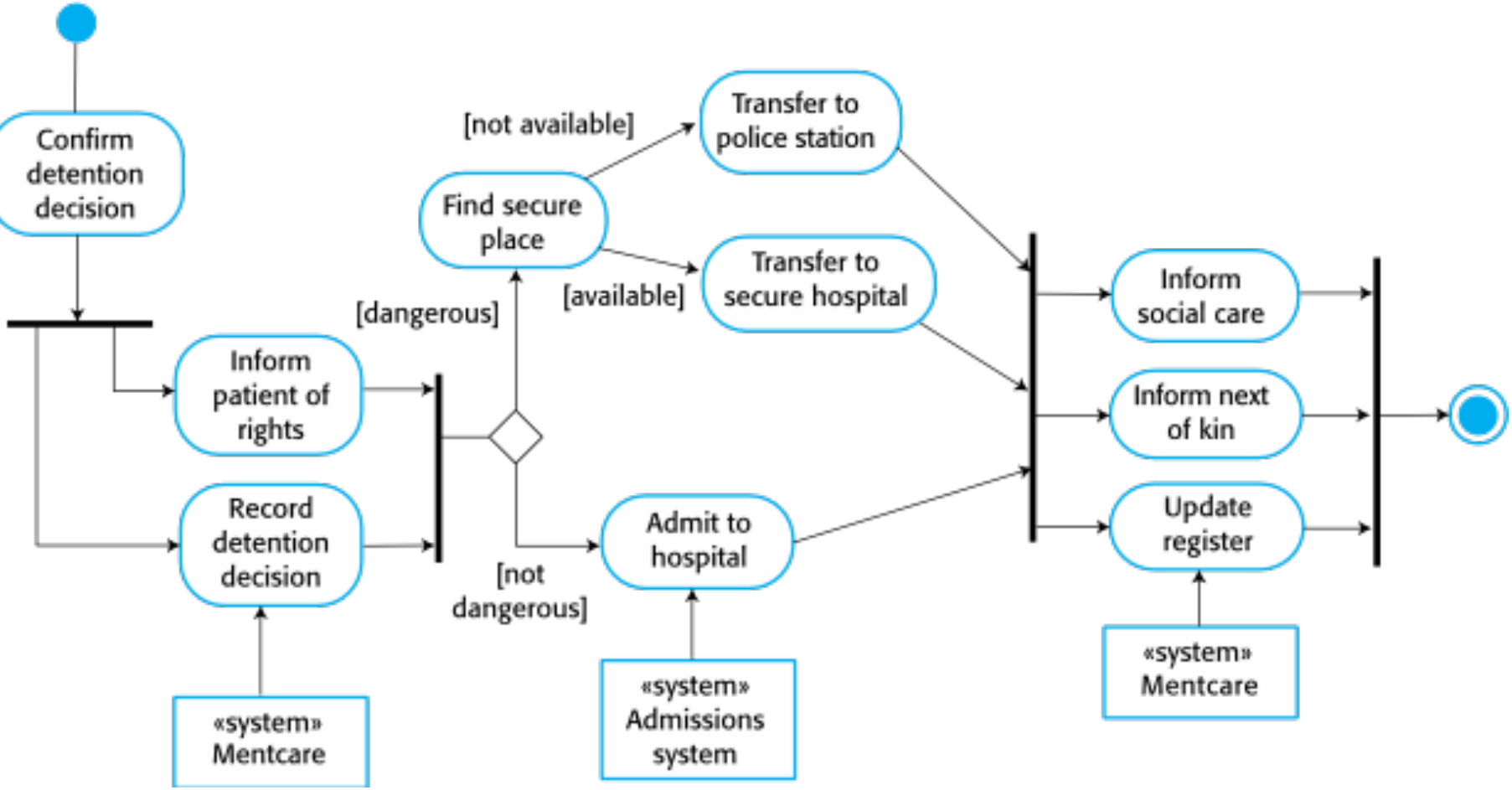
The context of the Mentcare system



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

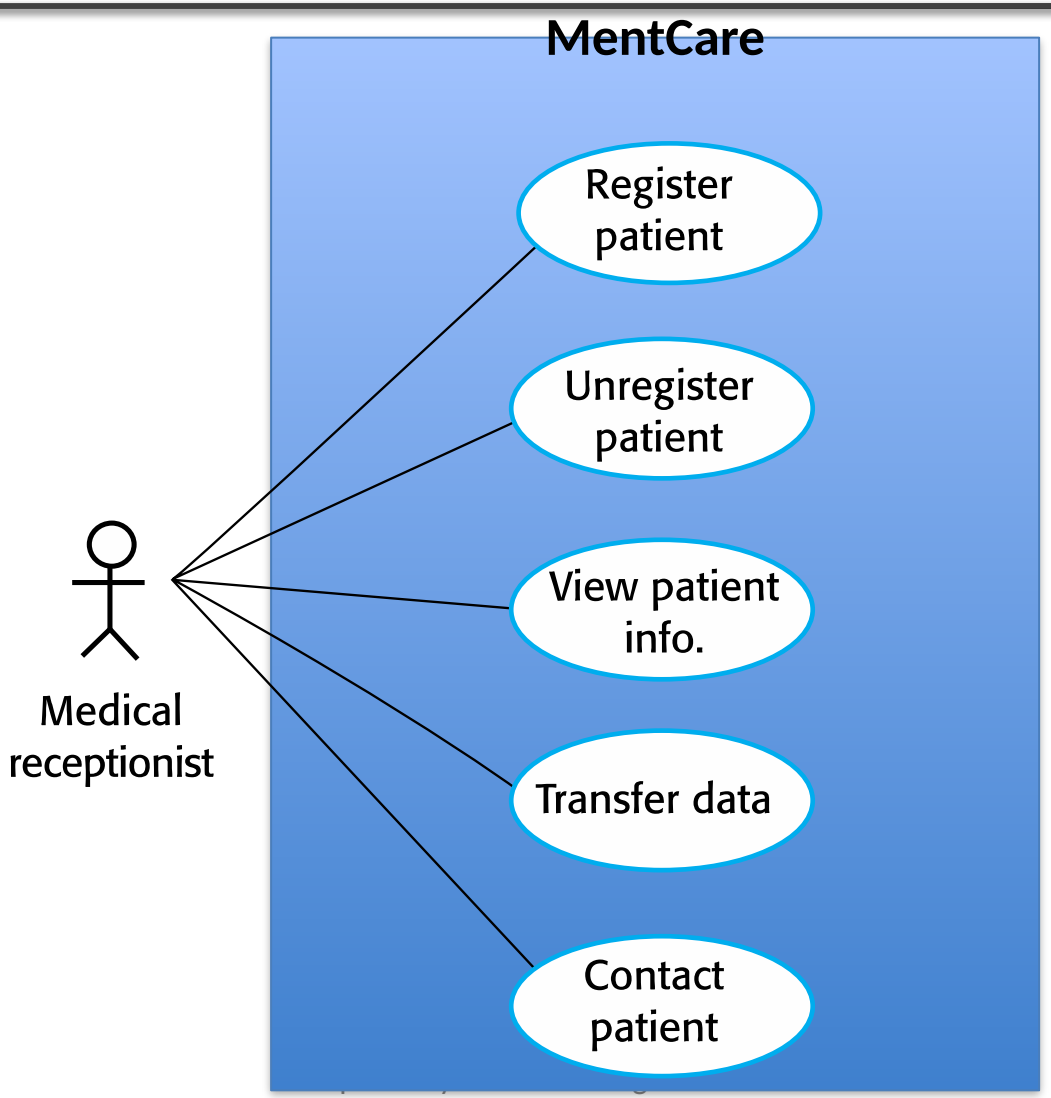
Interaction models

- 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 modeling.

Use case modeling

- Use cases were developed originally to support requirements elicitation.
- 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 diagramatically to provide an overview of the use case and in a more detailed textual form.
- Use case models and sequence diagrams present interactions at different levels of detail and so may be used together.

Use cases in the Mentcare system involving the role ‘Medical Receptionist’



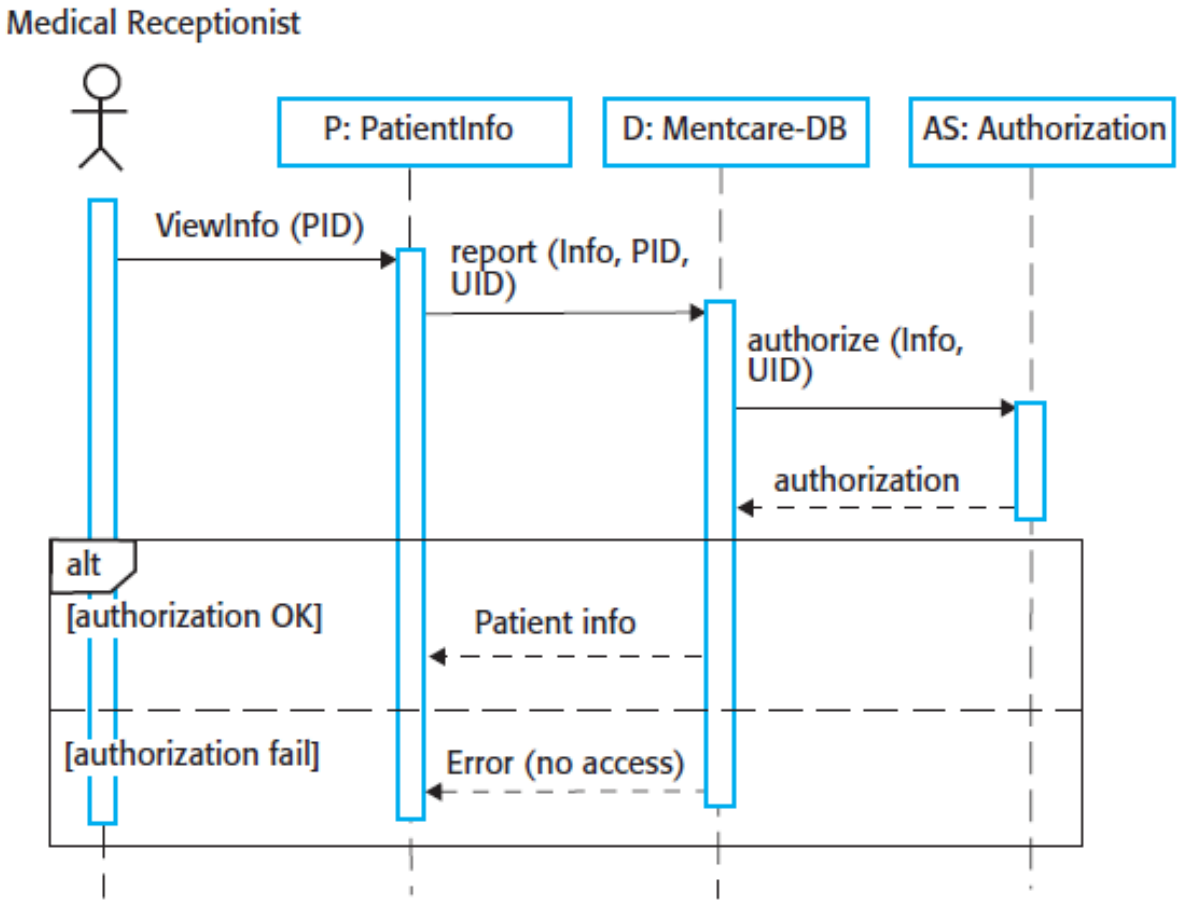
Tabular description of the ‘Transfer data’ use-case

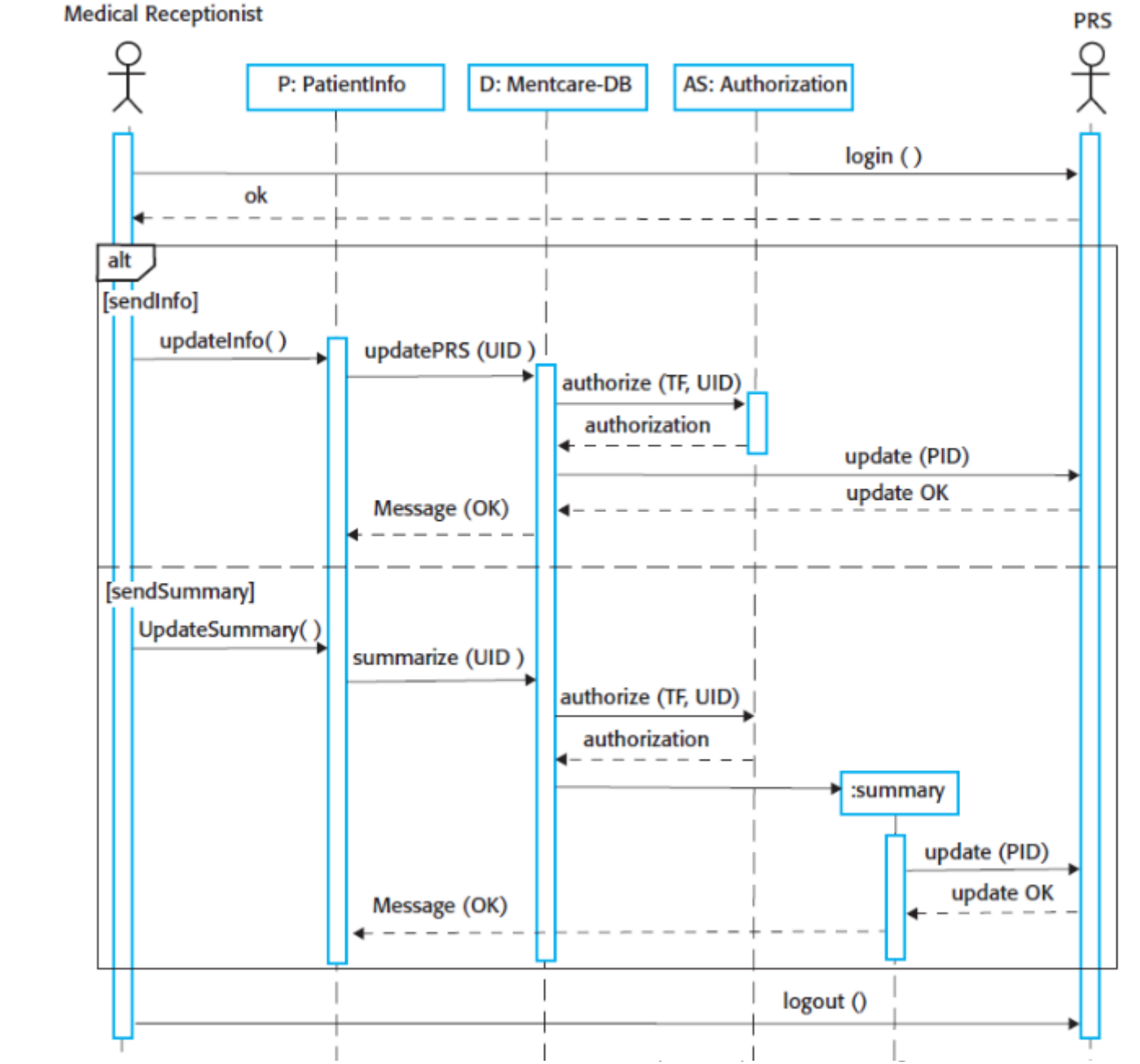
MHC-PMS: Transfer data	
Actors	Medical receptionist, patient records system (PRS)
Description	A receptionist may transfer data from the Mentcare system to a general patient record database that is maintained by a health authority. The information transferred may either be updated personal information (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.

Sequence diagrams

- Sequence diagrams are part of 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 for View patient information





Sequence diagram for Transfer Data

Structural models

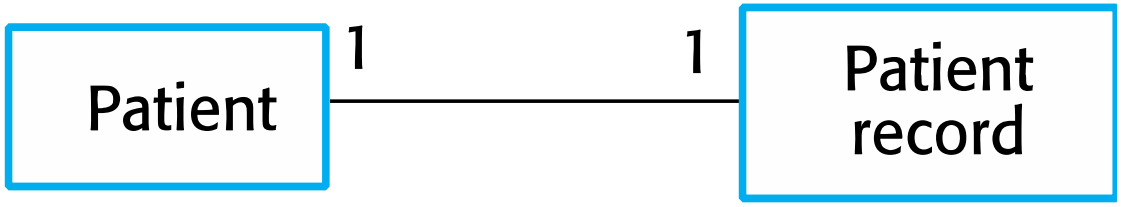
Structural models

- 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.

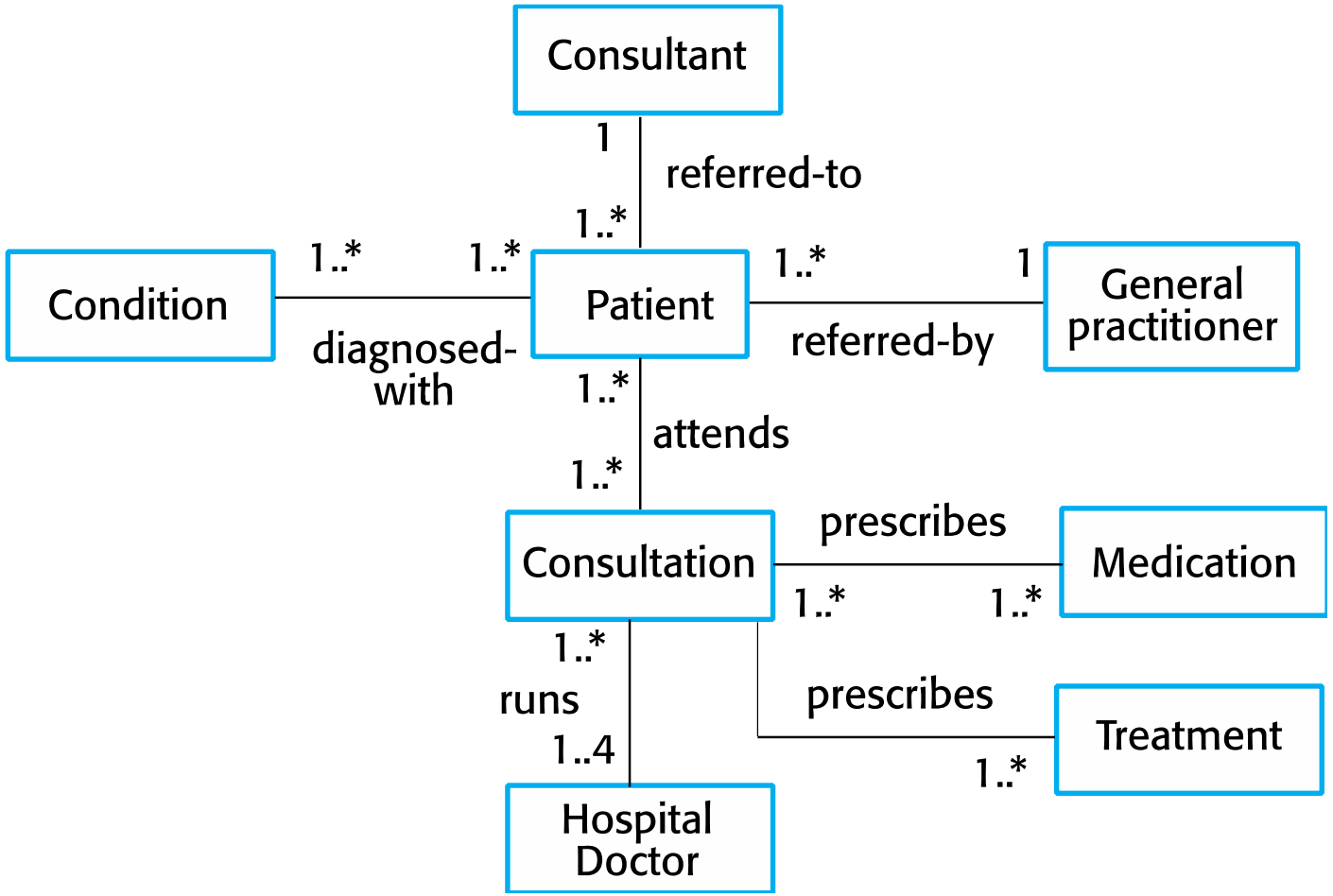
Class diagrams

- Class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes.
- An instance of a class (an object) can be thought of as a general definition of one kind of system object.
- An association is a link between classes that indicates that there is some relationship between these classes.
- When you are developing models during the early stages of the software engineering process, objects represent something in the real world, such as a patient, a prescription, doctor, etc.

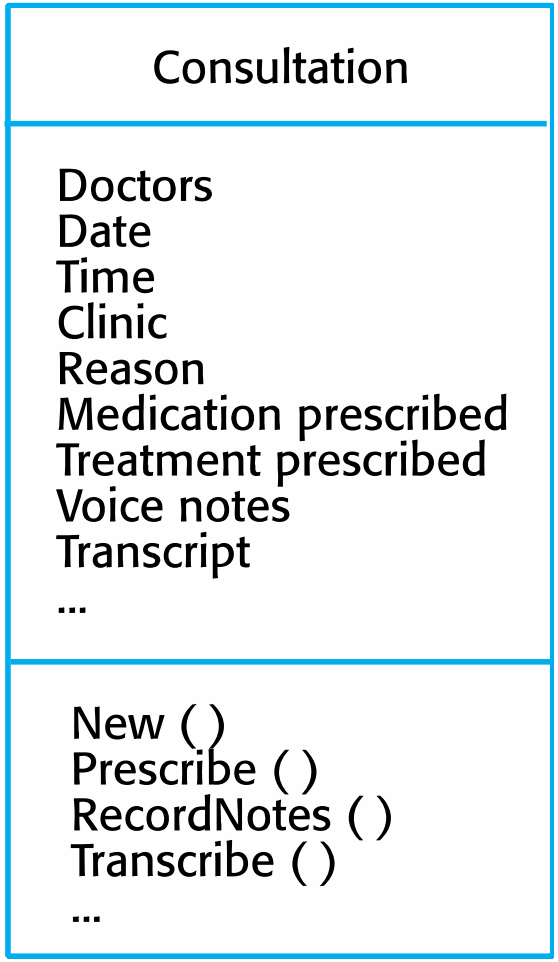
UML classes and association



Classes and associations in the MHC-PMS



The Consultation class



Mode detail on
Use Case Diagram

What is a Use case?

"Use cases are a means for specifying required usages of a system. Typically, they are used to capture the requirements of a system.

The key concepts associated with use cases are actors, use cases, and the system under consideration.

The users and any other systems that may interact with the subject are represented as actors. Actors always model entities that are outside the system.

The required behavior of the subject is specified by one or more use cases.

Use cases, actors, and systems are described using use case diagrams."

Source: UML Superstructure Specification v2.4.1 omg.org (summarized)

Elements of a Use Case Diagram

Actors:

- An actor is an external entity, it specifies a role played by a user or any other system that interacts with the system.
- An actor represents any entity (or entities) that exchanges information with the system. An actor in a use case diagram interacts with a use case to accomplish a specific goal.
- For example, for modeling a banking application, a customer entity is represented by an actor in the diagram. Similarly, the person who provides the service at the counter.

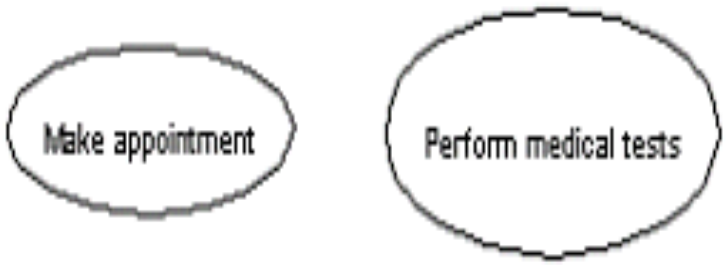
Elements of a Use Case Diagram

- To identify an actor, search in the problem statement for business terms that portray roles in the system. For example, in the statement "patients visit the doctor in the clinic for medical tests," "doctor" and "patients" are the roles played and can be easily identified as actors in the system.

Elements of a Use Case Diagram

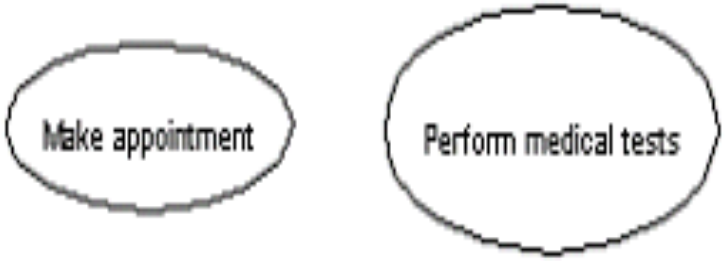
Use case:

- A use case in a use case diagram is a visual representation of a distinct functionality in a system. In other words, each use case must perform a distinct function.
- As the first step in identifying use cases, you should list the discrete functions in your problem statement. Each of these functions can be classified as a potential use case.



Elements of a Use Case Diagram

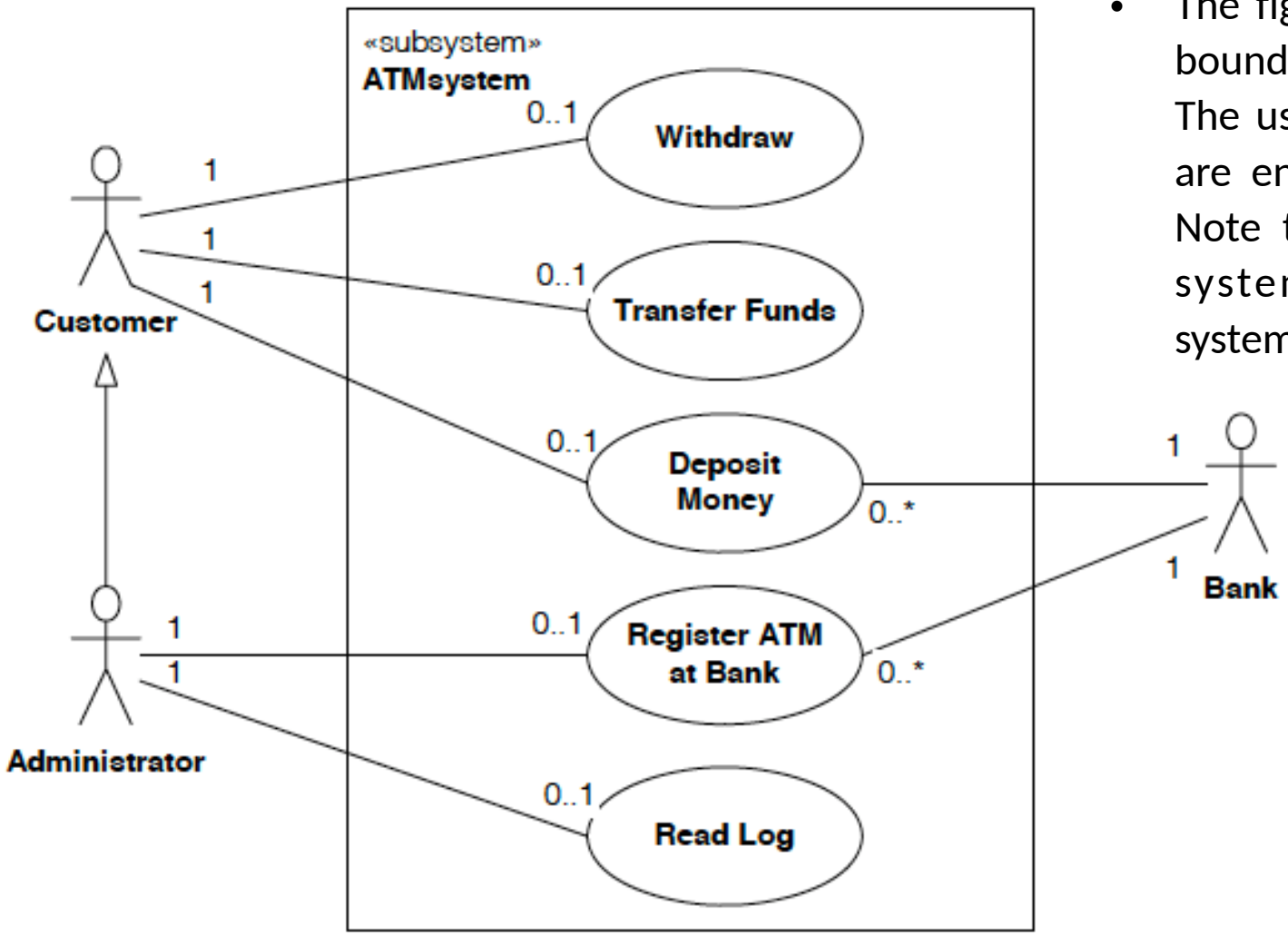
- The Figure below shows two uses cases: "Make appointment" and "Perform medical tests" in the use case diagram of a clinic system. As another example, consider that a use case such as "manage patient records" can in turn have sub-use-cases like "manage patient's personal information" and "manage patient's medical information.“
- A use case can be stated as a present –tense verb phrase, containing the verb and the object of the verb e.g. **compute gpa of student**



Elements of a Use Case Diagram

- System boundary: A system boundary defines the scope of what a system will be. A system cannot have infinite functionality. So, it follows that use cases also need to have definitive limits defined. A system boundary of a use case diagram defines the limits of the system. The system boundary is shown as a rectangle spanning all the use cases in the system.

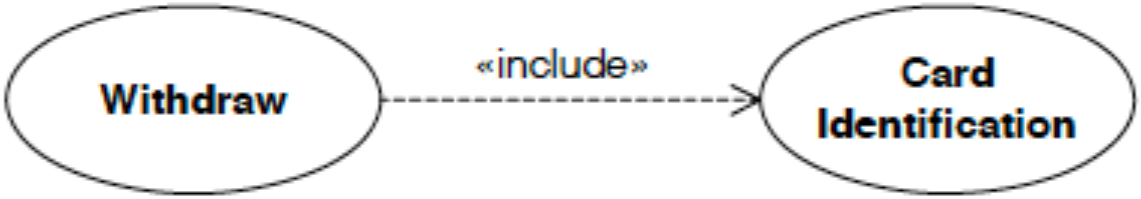
Example of a Use Case diagram of an ATM system



- The figure shows the system boundary of the ATM system. The use cases of this system are enclosed in a rectangle. Note that the actors in the system are outside the system boundary.

Relationships in Use Cases

- Include: An include relationship defines that a use case contains the behavior defined in another use case.
- Literally speaking, in an *include* relationship, the source use case includes the functionality described in the target use case. An include relationship is depicted with a directed arrow having a dotted line. The stereotype "<<include>>" identifies the relationship as an include relationship.



Relationships in Use Cases

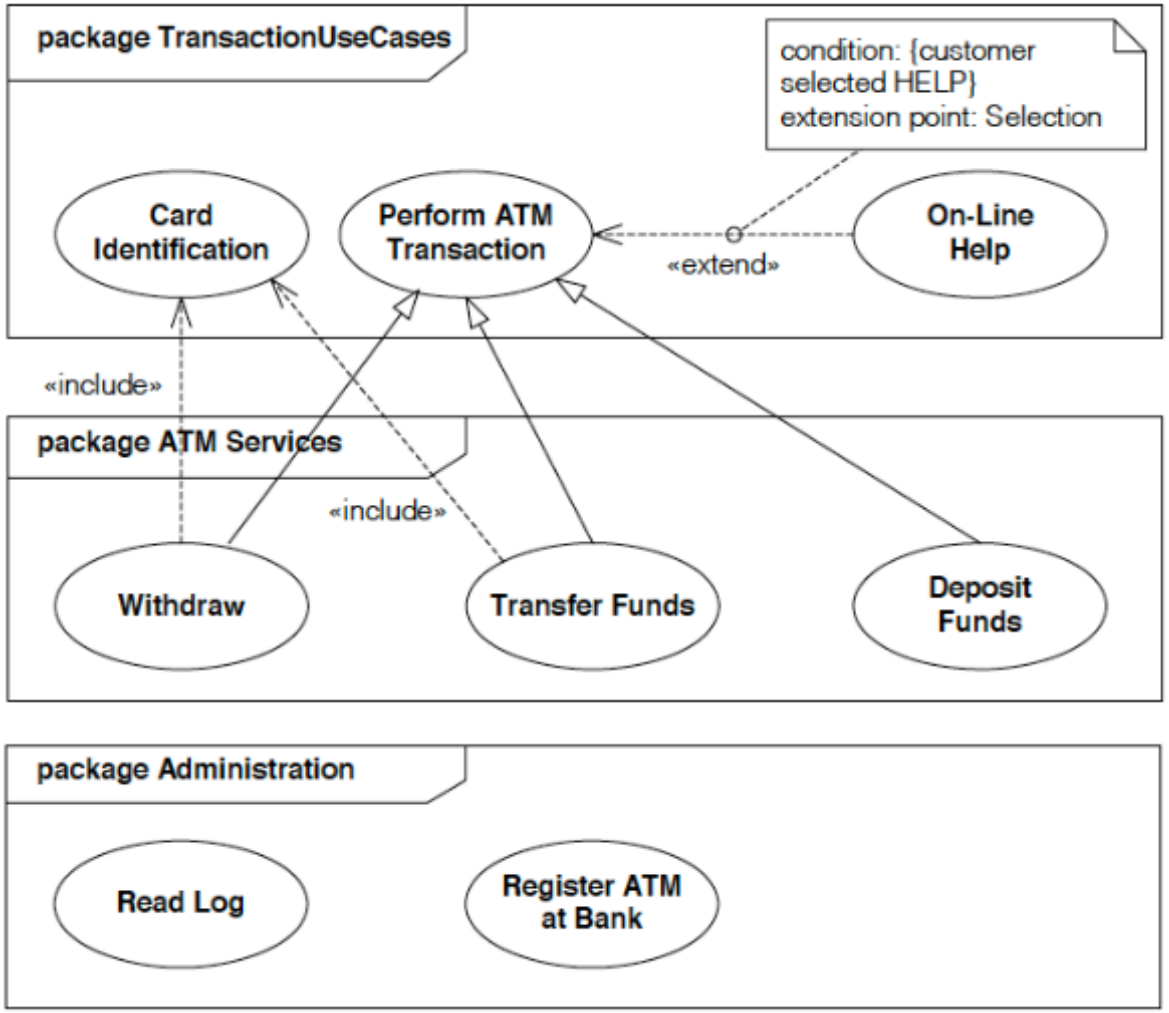
- **Extend:** A relationship from an extending use case to an extended use case that specifies how and when the behavior defined in the extending use case can be inserted into the behavior defined in the extended use case.
- Extend relationship is also used to design handling exceptions.

Example of an extend relationship

- Example of an extend relationship between the "Perform medical tests" (target) and "Perform Pathological Tests" (source) use cases. The "Perform Pathological Tests" use case enhances the functionality of the "Perform medical tests" use case. Essentially, the "Perform Pathological Tests" use case is an extension of the generic "Perform medical tests" use case: It is executed **when medical tests show abnormal results**.

Relationships in Use Cases

- Generalization: it is a directed relationship between use cases. The source use case in the generalization relationship is a specialization of the target use case.
- The source use case performs **a kind** of target use case's function.



Relationships in Use Cases

Difference between Generalization and Extension:

- When you establish a **generalization** relationship between use cases, this implies that the target use case can be replaced by the source use case. On the other hand, an **extend** relationship between use cases implies that the source use case enhances the functionality of the target use. The target use case in an extend relationship cannot be replaced by the source use case.

Exercise

Draw a use case diagram for a university registration system:

The system is used by students to register for courses. Assume that the student accesses an electronic course catalog to find out about available courses. Courses may have limited enrollment, so the registration process must include checks that places are available. Students have to complete all pre-requisites of a course before being allowed to register to that course. A student may send a message to the department's chair to request that the pre-requisite condition be overlooked if he is taking the required course in parallel.

Use-case diagram for a university registration system

ATM Example

- An ATM system has a switch that allows an operator to start and shutdown the system. The operator can configure the system to connect it to the bank system. He can run a maintenance check process that checks that the system is working properly. When a problem is detected, the operator has to solve the problem or request a more specialized maintenance team to solve the problem. The operator has to shut down the system until the other team arrives.
- A customer is required to insert his ATM card and enter his PIN. The PIN and card ID will be sent to the bank for validation before any transaction. If the PIN is not valid, then a message “invalid PIN” is displayed. If the PIN is valid, the customer will be able to perform one of the following transactions: make a cash withdrawal, make a deposit, make a money transfer to previously saved accounts, make a balance inquiry.

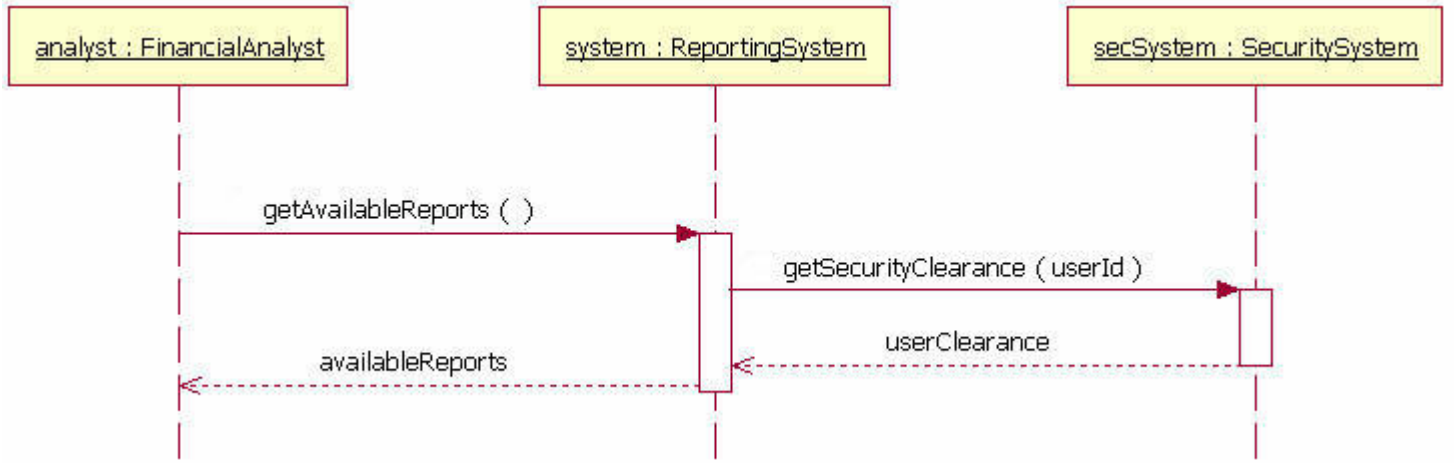
Sequence Diagram

Interaction Diagrams

- An Interaction is a behaviour that comprises a set of messages exchanged between a set of objects within a context to accomplish a purpose.
- A message is a specification of a communication between objects that conveys information.
- In UML, interaction diagrams are used for the following purposes:
  - to model the dynamic behavior of a system.
  - To visualize the communication and sequence of message passing in the system

Interactions- showing the dynamic aspect

Sequence diagrams, commonly used by developers, model the interactions between objects in a single use case. They illustrate how the different parts of a system interact with each other to carry out a function, and the order in which the interactions occur when a particular use case is executed.



Sequence Diagram

- A sequence diagram is two dimensional. Graphically a sequence diagram is a table that shows objects arranged along the X-axis and messages ordered in increasing time along the Y-axis.
- A Sequence Diagram should be modeled for the main scenario of the use case, and frequent or complex alternative scenarios.
- All systems/components are treated as black boxes

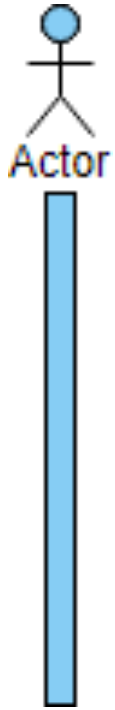
Lifeline

- A lifeline represents an individual participant in the Interaction.

Actor:

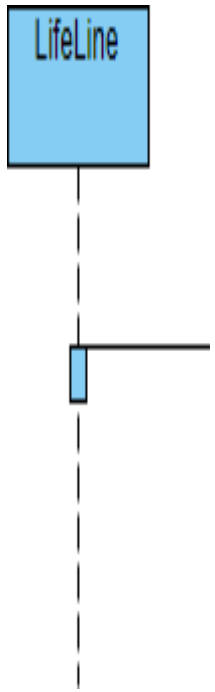
- An Actor represents a role played by an entity that interacts with the system (e.g., by exchanging signals and data). An actor is external to the subject (i.e., in the sense that an instance of an actor is not a part of the instance of its corresponding subject). They typically represent roles played by human users, external hardware, or other systems.
- Note That:

1. An actor does not necessarily represent a specific physical entity but merely a particular role of some entity
2. A person/component may play the role of several different actors and, conversely, a given actor may be played by different persons.



Activation

- An activation is represented by a thin rectangle on a lifeline) representing the period during which an element is performing an operation. The top and the bottom of the rectangle are aligned with the initiation and the completion time respectively.
- When a lifeline is reacting to a message, it has a focus of control. As the interaction progresses over time, the focus of control moves between various lifelines. This movement is called a flow of control.



Sequence Diagram-Messages

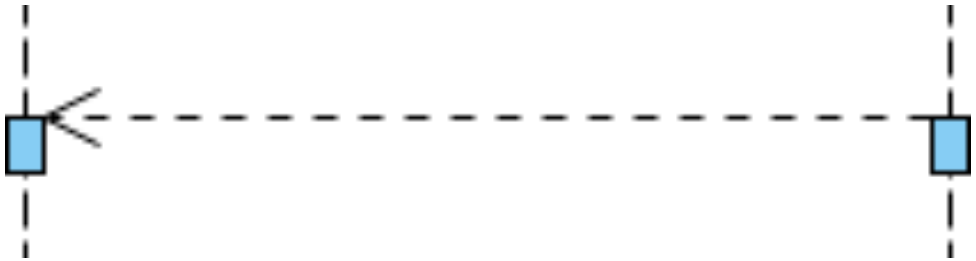
1. Call Message

A call message defines a particular communication between lifelines of an interaction, which represents an invocation of operation of the target lifeline.

A communication can be, for example, raising a signal, invoking an Operation, creating or destroying an Instance.

2. Return Message

- A return message defines a particular communication between lifelines of an interaction, which represents the passing of information back to the caller.



3. Self (reflexive) Message

- When an object sends a message to itself, it is called a reflexive message. It is indicated with a message arrow that starts and ends at the same lifeline as shown in the example below.

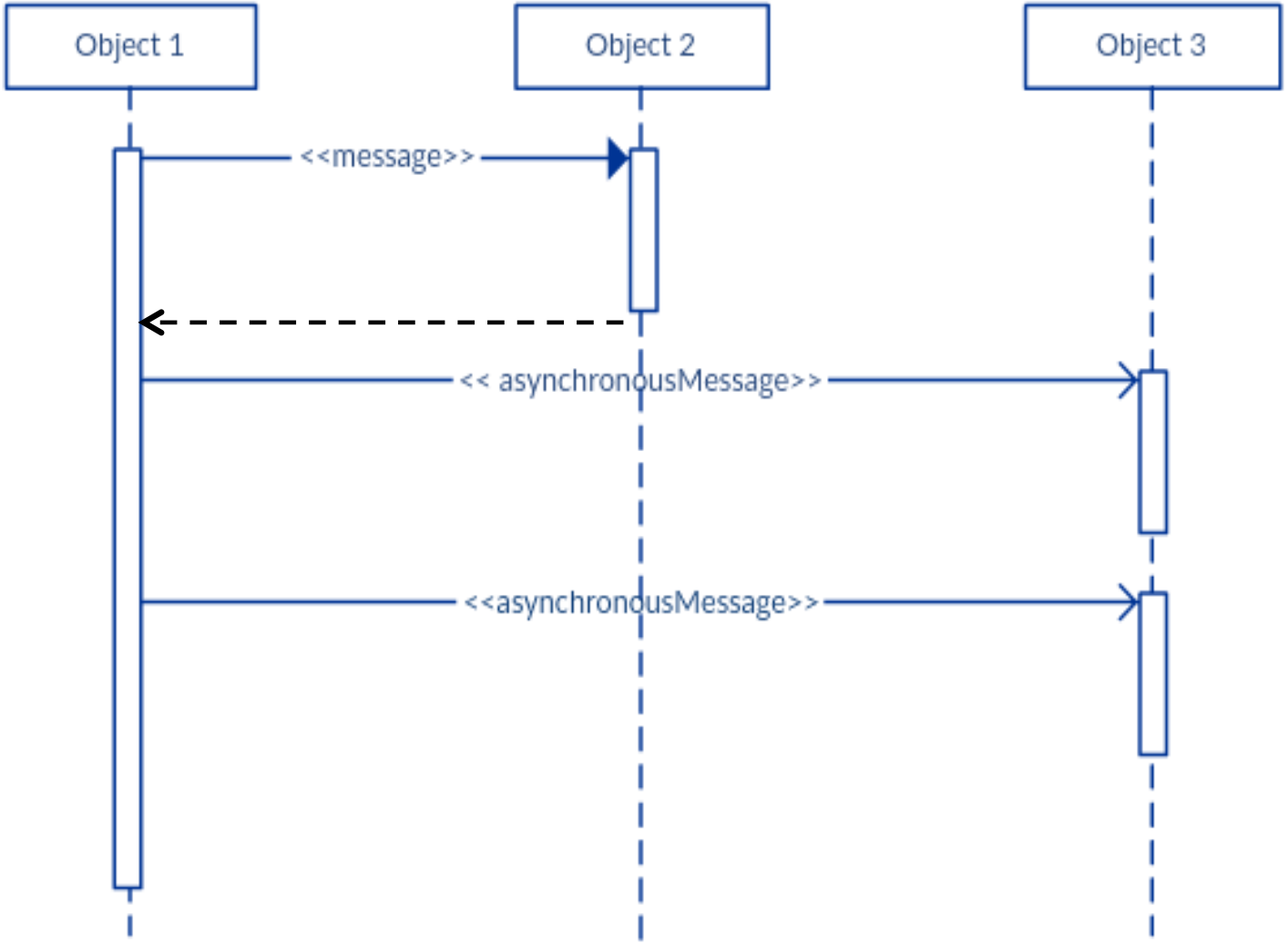
Synchronous message

A synchronous message is used when the sender waits for the receiver to process the message and returns before carrying on with another message. The arrowhead used to indicate this type of message is a solid one, like the one below.

Elements of Sequence Message Arrows

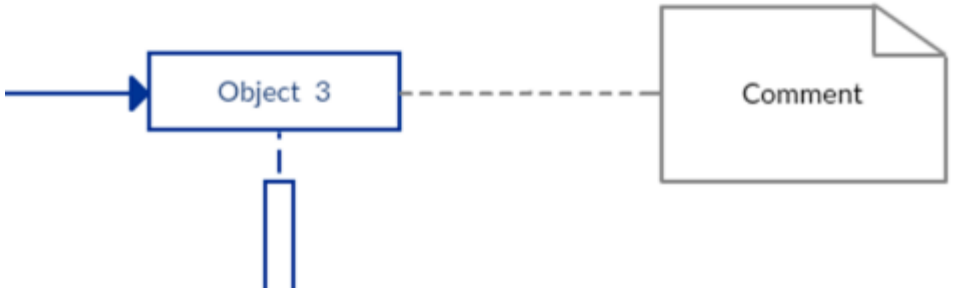
- **Asynchronous message**
- An asynchronous message is used when the message caller does not wait for the receiver to process the message and return before sending other messages to other objects within the system. The arrowhead used to show this type of message is a line arrow like shown in the example below.

Elements of Sequence Message Arrows



Note

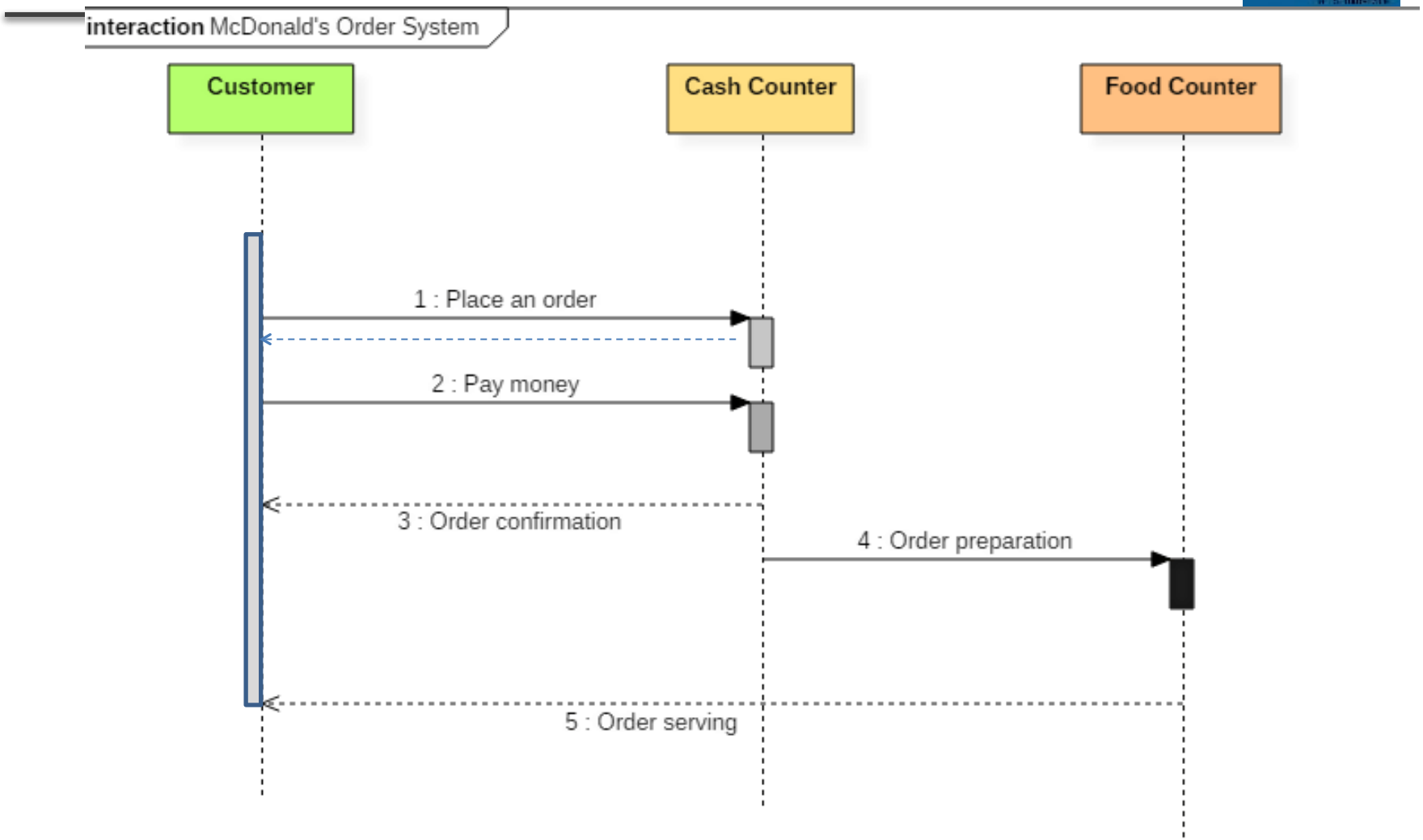
- A note (comment) gives the ability to attach various remarks to elements. A comment carries no semantics, but may contain information that is useful to a modeler.



Sequence diagram of Mcdonald's Ordering System

- The ordered sequence of events in a given sequence diagram is as follows:
- Place an order.
- Pay money to the cash counter.
- Order Confirmation.
- Order preparation.
- Order serving.

Mcdonald's Ordering System



Alternatives

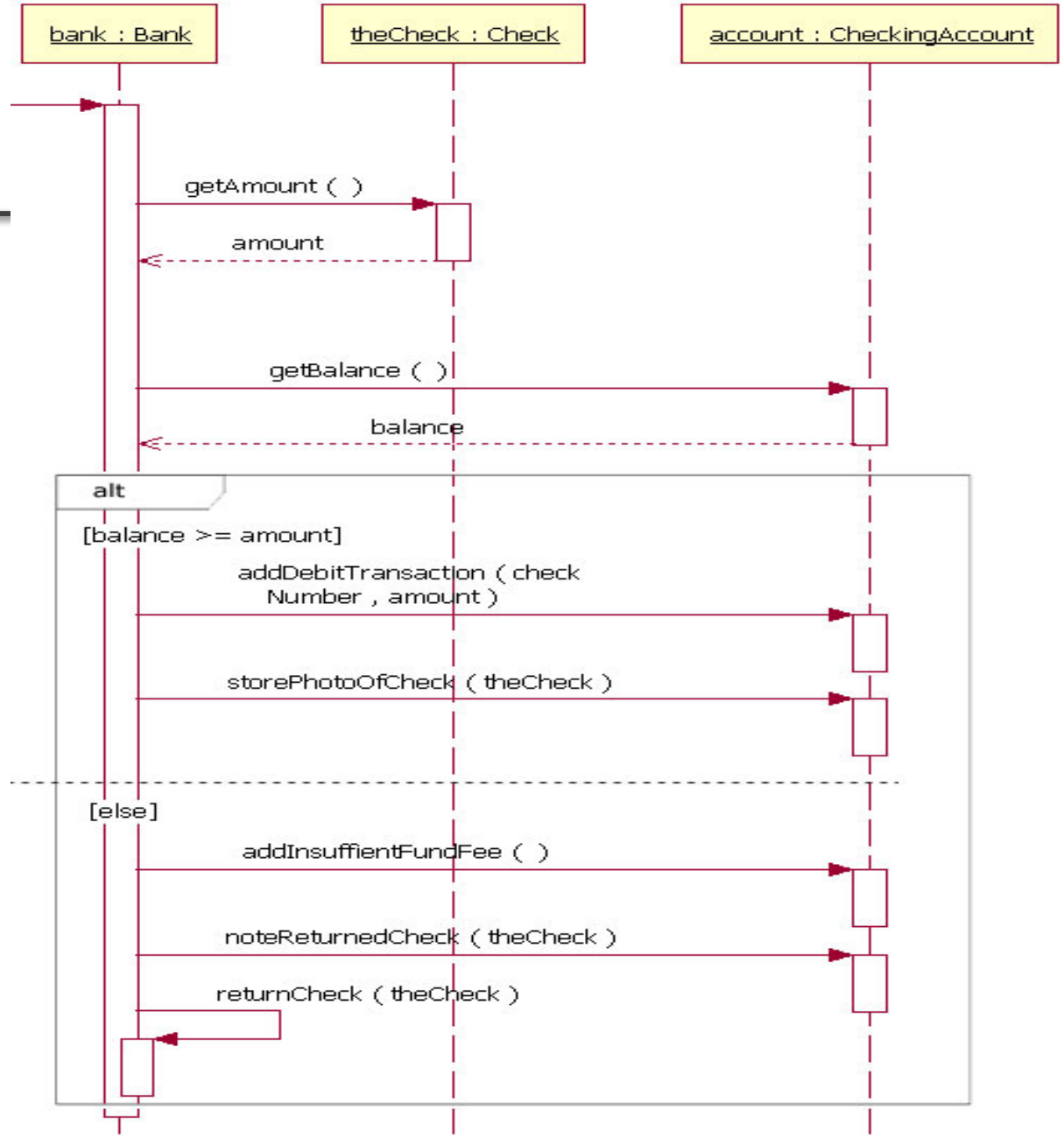
- Alternatives are used to designate a mutually exclusive choice between two or more message sequences.
- Alternatives allow the modeling of the classic “if then else” logic (e.g. if I buy three items, then I get 20% off my purchases; else I get 10% off my purchase.)

Alternatives

- An alternative combination fragment is drawn using a frame.
- The word “alt” is placed inside the frame’s name box.
- The larger rectangle is then divided into what UML calls **operand**.
- Operands are separated by a dashed line (one are the messages following “then” path and another are the messages following “else” path).
- Each operand is given a guard to test against, and this guard is placed on the top of the operand. If an operand’s guard equates to “true”, then that operand is the operand to follow.

Alternatives

- Fig on the next slide shows the sequence starting at the top, with the bank object getting the check's amount and the account's balance.
- At this point in the sequence the alternative combination fragment takes over. Because of the guard "[balance >= amount]", if the account's balance is greater than or equal to the amount, then the sequence continues with the bank object sending the addDebitTransactrion and storePhotoOfCheck messages to the account object.
- However, if the balance is not greater than or equal to the amount, then the sequence proceeds with the bank object sending the addInsufficientFundFee and noteReturned Check message to the account object and the returnCheckmessage to itself. The second sequence is called when the balance is not greater than or equal to the amount because of the "[else]" guard.


bank : Bank
theCheck : Check
account : CheckingAccount
getAmount ( )
amount
getBalance ( )
balance
alt
[balance >= amount]
addDebitTransaction (check Number , amount)
storePhotoOfCheck (theCheck)
[else]
addInsuffientFundFee ( )
noteReturnedCheck (theCheck)
returnCheck (theCheck)


Software Engineering

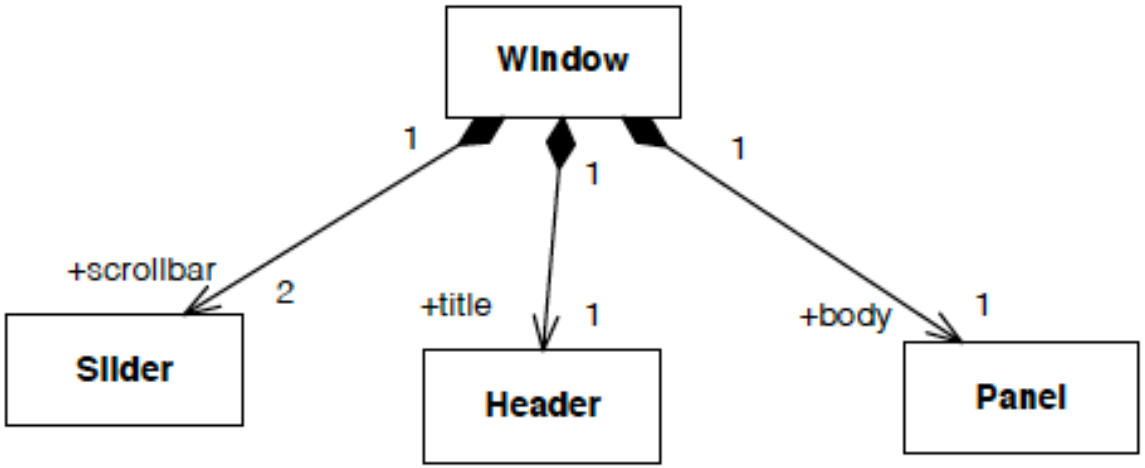
Class Diagram

Class Diagram

- Class diagram is a static diagram. It gives a static view of a system.
- Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software.
- UML Class diagram is a type of structure diagram.
- UML Class diagram gives an overview of a software system by displaying classes, attributes, operations, and their relationships.

Advantages of Class Diagram

- Class Diagram is used to model how the system will be structured (or is structured if it is an existing system).
- It helps for better understanding of the general organization of the system.
- Useful for developers and other stakeholders to implement and also document the system.



h

Class Diagram in Software Development Lifecycle

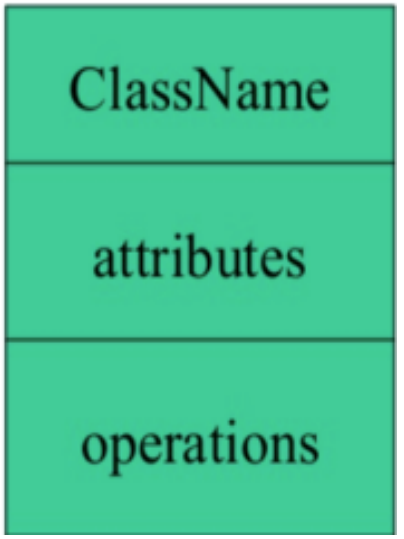
- Class diagrams can be used in various software development phases. We can model class diagrams from three different perspectives.
- **1. Conceptual perspective:** Conceptual diagrams are describing things in the real world. You should draw a diagram that represents the concepts in the domain under study. These concepts are represented by classes.
- **2. Specification perspective:** Specification perspective describes software abstractions or components with specifications and interfaces.
- **3. Implementation perspective:** This type of class diagrams is used for implementation in a specific language. Implementation perspective.

Essential elements of A UML class diagram

Essential elements of UML class diagram are:

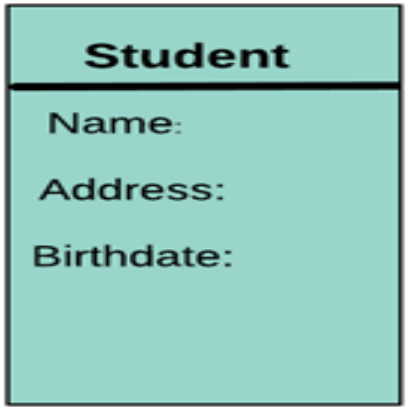
- Class Name (Always starts with a capital letter)
- Attributes
- Operations

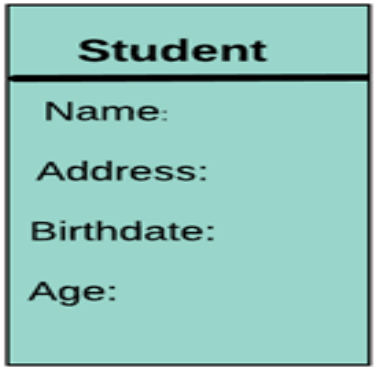
A class is the blueprint of an object which can share the same relationships, attributes, operations, & semantics. The class is rendered as a rectangle, including its name, attributes, and operations in separate compartments.



Class Attributes

- An attribute is a named property of a class which describes the object being modeled. In the class diagram, this element is placed just below the name compartment.
- A derived attribute is computed from other attributes. For example, an age of the student can be computed from his/her birth date.





Attributes characteristics

- The attributes are generally written along with the visibility factor.
- Public, private, protected and package are the four visibilities which are denoted by +, -, #, or ~ signs respectively.
- Visibility describes the accessibility of an attribute of a class.
- Attributes must have a meaningful name that describes the use of it in a class

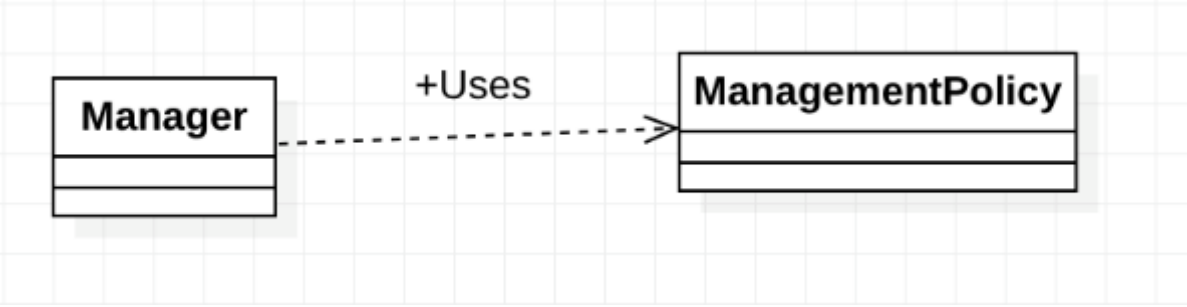
Relationships

There are mainly three kinds of relationships in UML:

- Dependencies
- Generalizations
- Associations
  - Aggregation
  - composition

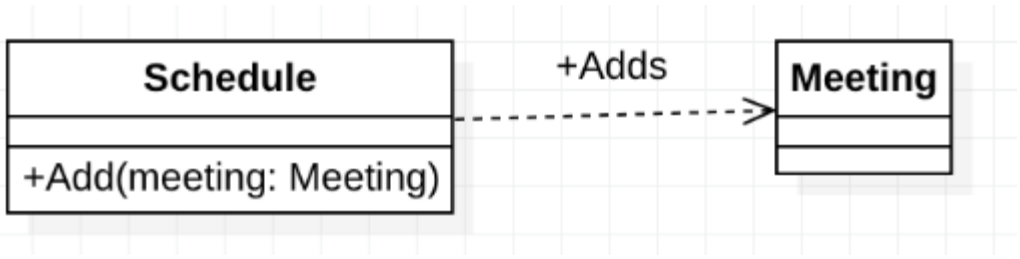
Relationship-Dependency

- It is a weak relationship.
- A dependency signifies a supplier/client relationship between model elements where the modification of the supplier may impact the client model elements.
- A dependency implies the semantics of the client is not complete without the supplier.
- Can have a name that explains how there is dependency.



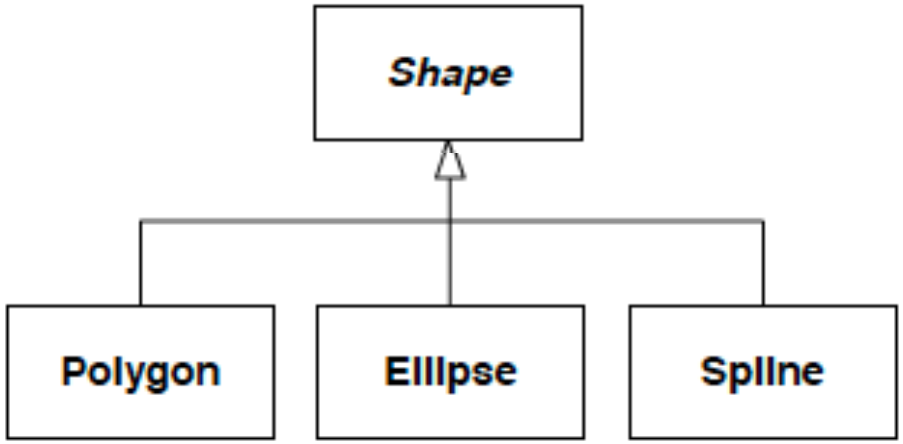
Examples of Dependency

- A class uses another as a parameter when calling a method.
- A class uses another inside the code of one of its methods.
- If the supplier class changes, the client class is affected weakly.



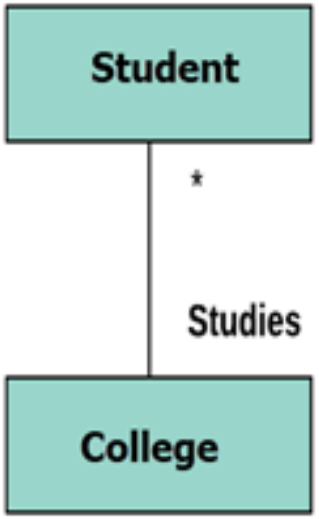
3. Relationships- Generalization

- A generalization connects a subclass to its superclass. A sub-class inherits from its superclass.
- Generalization relationship is not used to model interface implementation.
- In UML, Classes can inherit from multiple super-classes.



3. Relationships-Association

- This kind of relationship represents static relationship between classes. For example; an employee works for an organization.
- Here are some rules for Association:
- Association is named using a verb or a verb phrase or noun or noun phrase.
- It should be named to indicate the role played by the class attached at the end of the association path.



3. Relationships-Association(Multiplicity)

- A multiplicity is a factor associated with an attribute. It specifies how many instances of attributes are created when a class is initialized. If a multiplicity is not specified, by default one is considered as a default multiplicity.
- Let's say that there are 100 students in one college. The college can have multiple students (0..*).

Possible associations

- Zero to 1 (0…1)
- 1 to 1 (1..1)
- Zero to many (0… 6)
- 1 to many (1… *)

: sometimes you want to stress out the fact that one is the minimum, like a plane has at least one engine, a course has at least one topic…

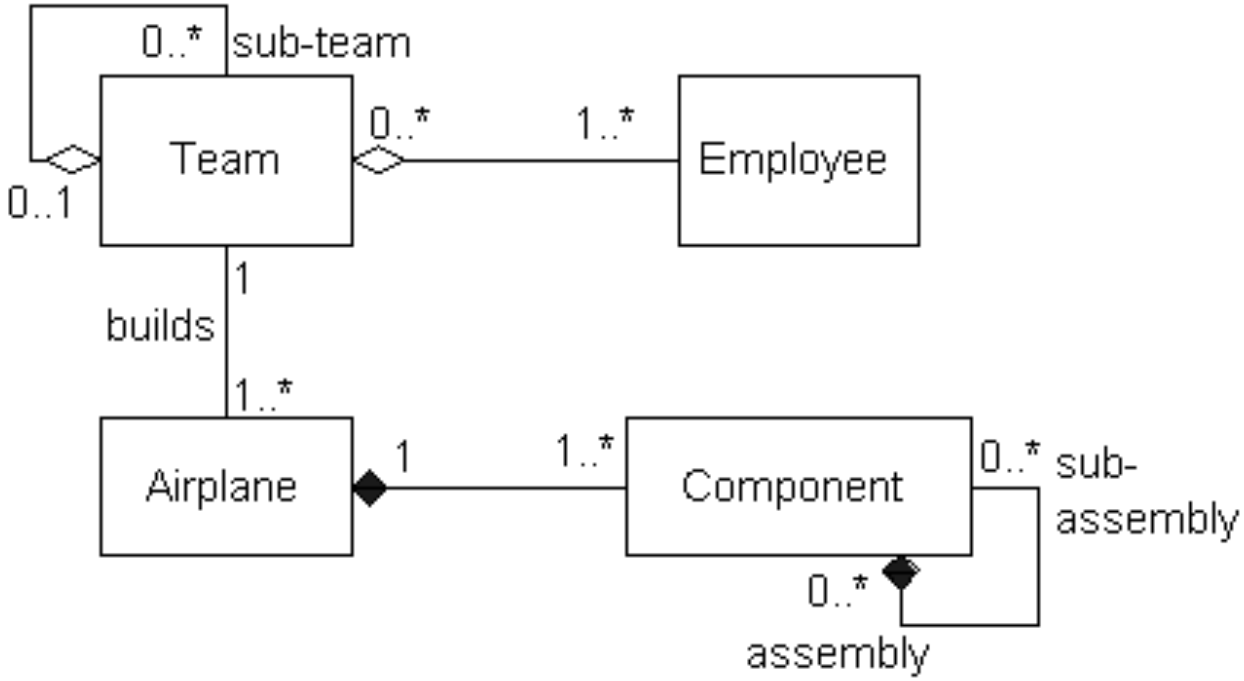
3. Relationships- Aggregation

Aggregation is a special type of association that models a ***whole-part*** relationship between aggregate and its parts.

In an aggregation relationship, the part object remains even when the whole object is destroyed.

Aggregation relationship is denoted using a straight line with an empty arrowhead at one end.

3. Relationships- Aggregation

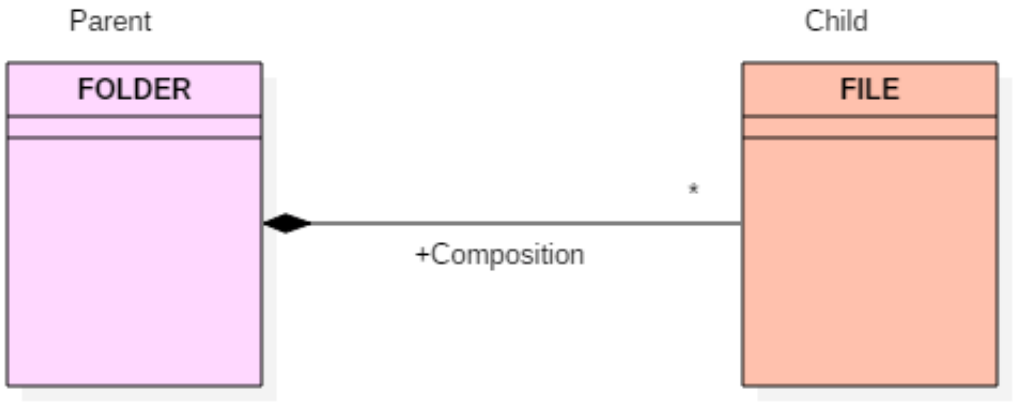


For exa oyee
class will remain even if the Team class is deleted/removed.

Relationship- Composition

- The composition is a special type of association which denotes strong ownership between two classes when one class is a part of another class.
- It is a two-way association between the objects.
- It is a whole/part relationship.
- If a composite is deleted, all other parts associated with it are deleted.
- Composition is described as an association decorated with a filled black diamond at the aggregate (whole) end.

Example 2-Composition



The folder could contain many files, while each File has exactly one Folder parent. If a folder is deleted, all contained files are removed as well.
In a composite aggregation, an object may be a part of only one composite at a time.

Aggregation vs. Composition

Aggregation

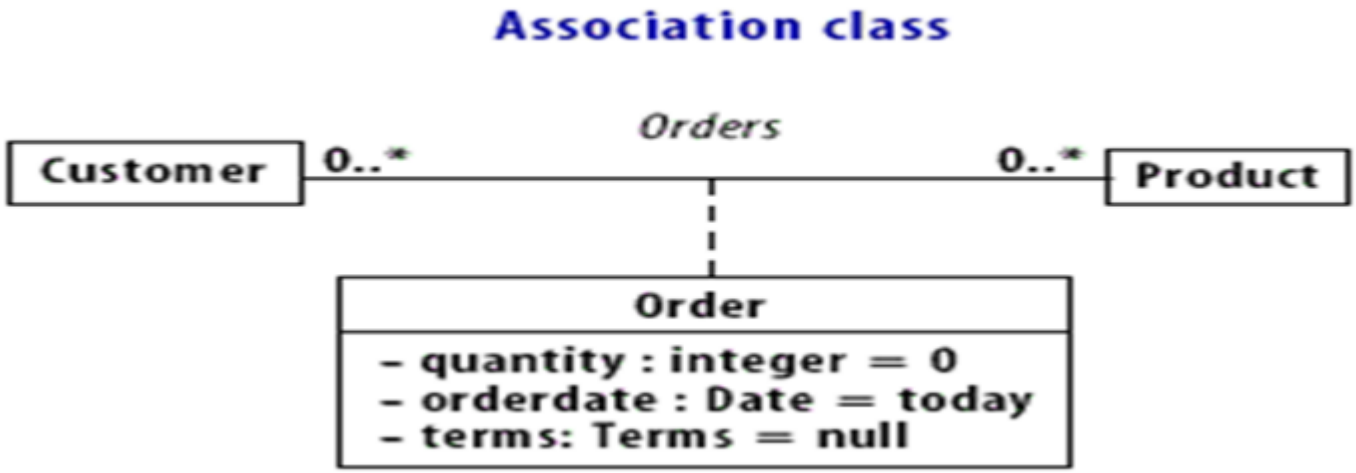
- Aggregation indicates a relationship where the child can exist separately from their parent class. Example: A team contains employees, but when a team is deleted the employees still exist.

Composition

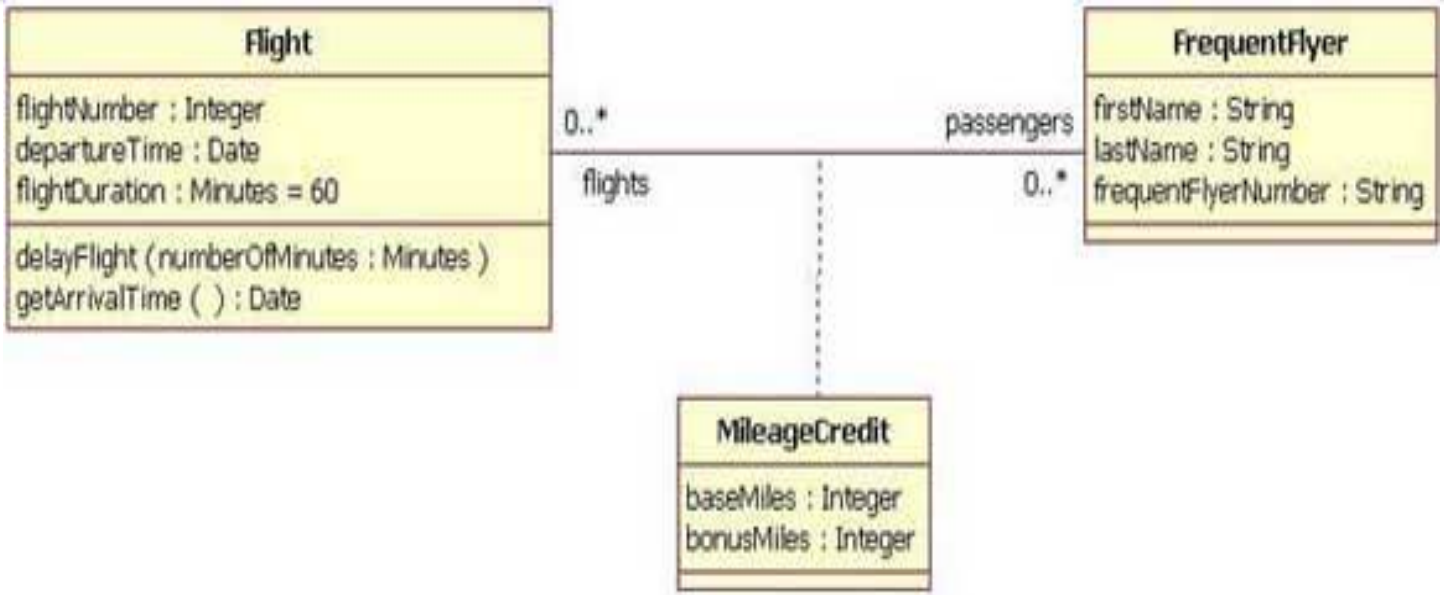
- Composition displays a relationship where the child will never exist independently of the parent. Example: House (parent) and Room (child). Rooms do not exist without a House.

Association Class

- When modeling an association, sometimes we need to include another class because it includes valuable information about the relationship. For this you would use an *association class* that you tie to the primary association. An association class is represented like a normal class. The difference is that the association line between the primary classes intersects a dotted line connected to the association class.



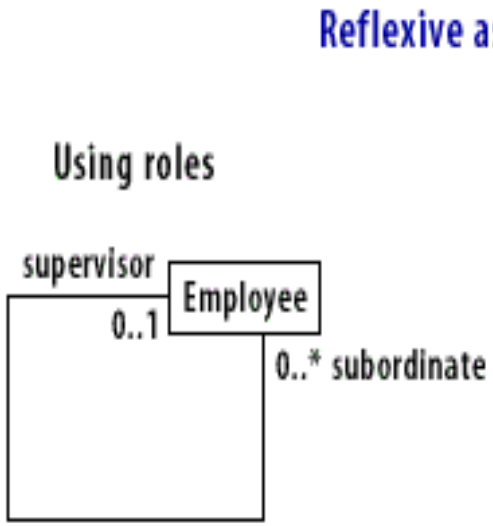
Association Class

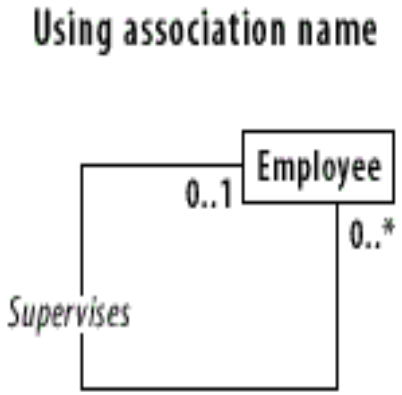


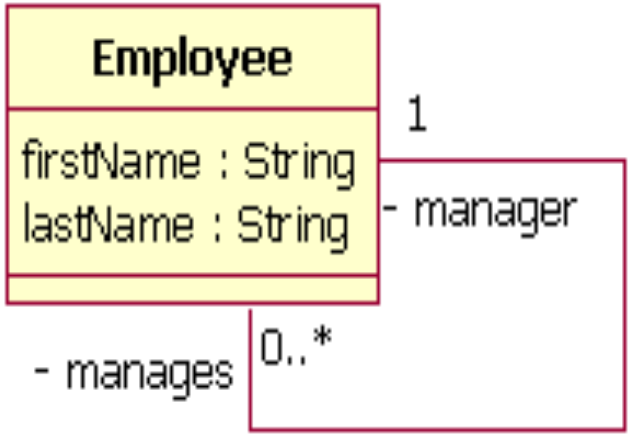
Here, the association between the Flight class and the FrequentFlyer class results in an association class called MileageCredit. This means that when an instance of a Flight class is associated with an instance of a FrequentFlyer class, there will also be an instance of a MileageCredit class.

Reflexive Association

Reflexive association is between objects of the same class.

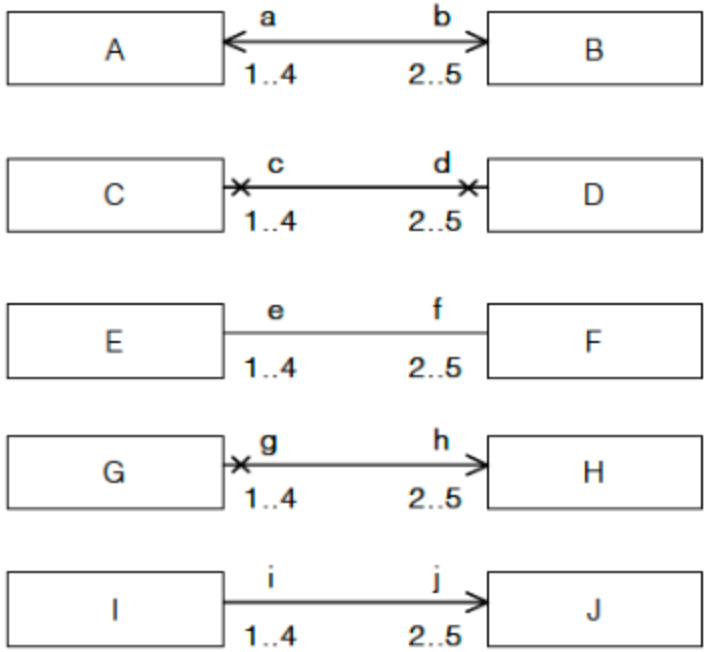






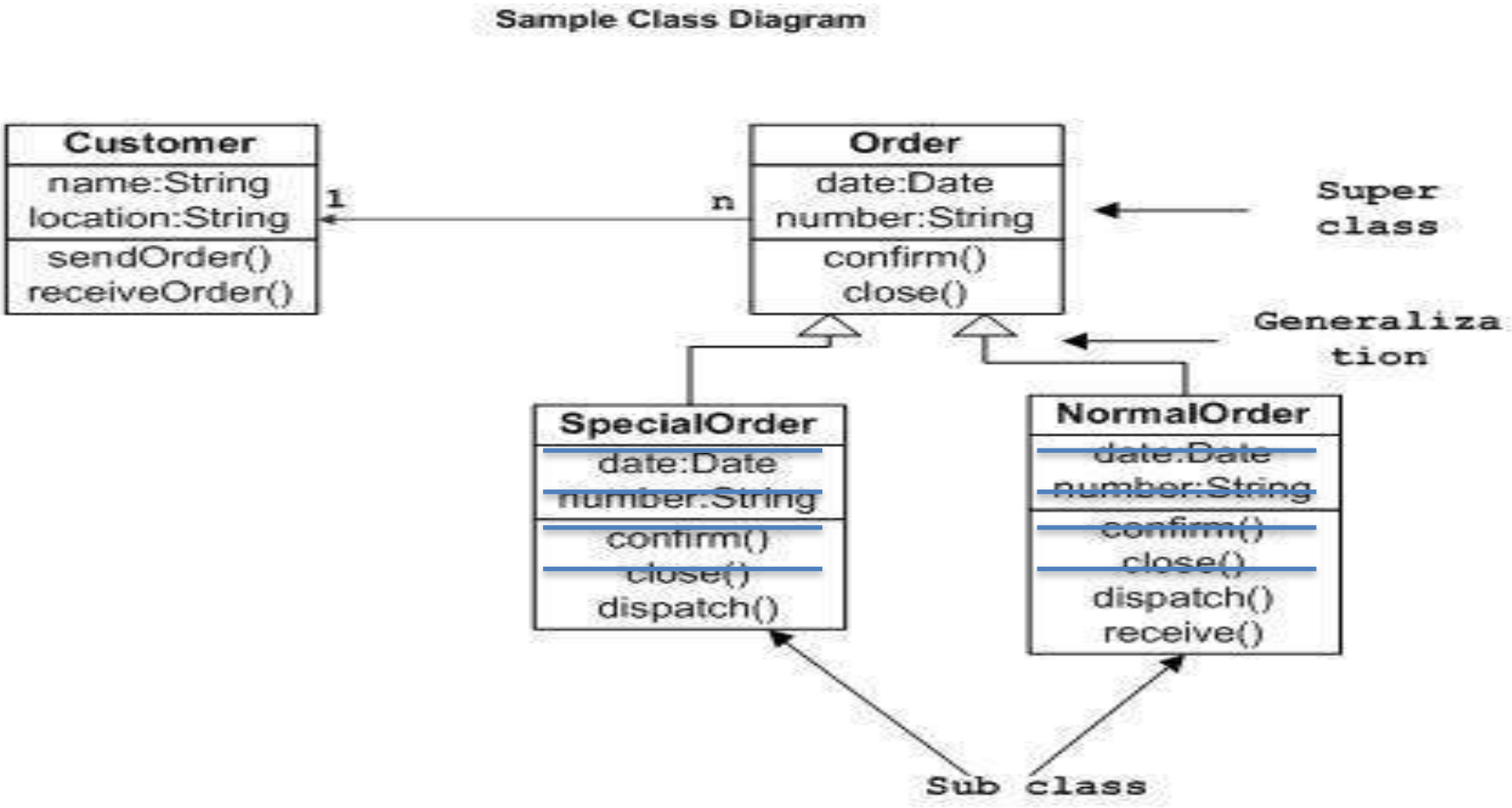
h

Navigability of associations



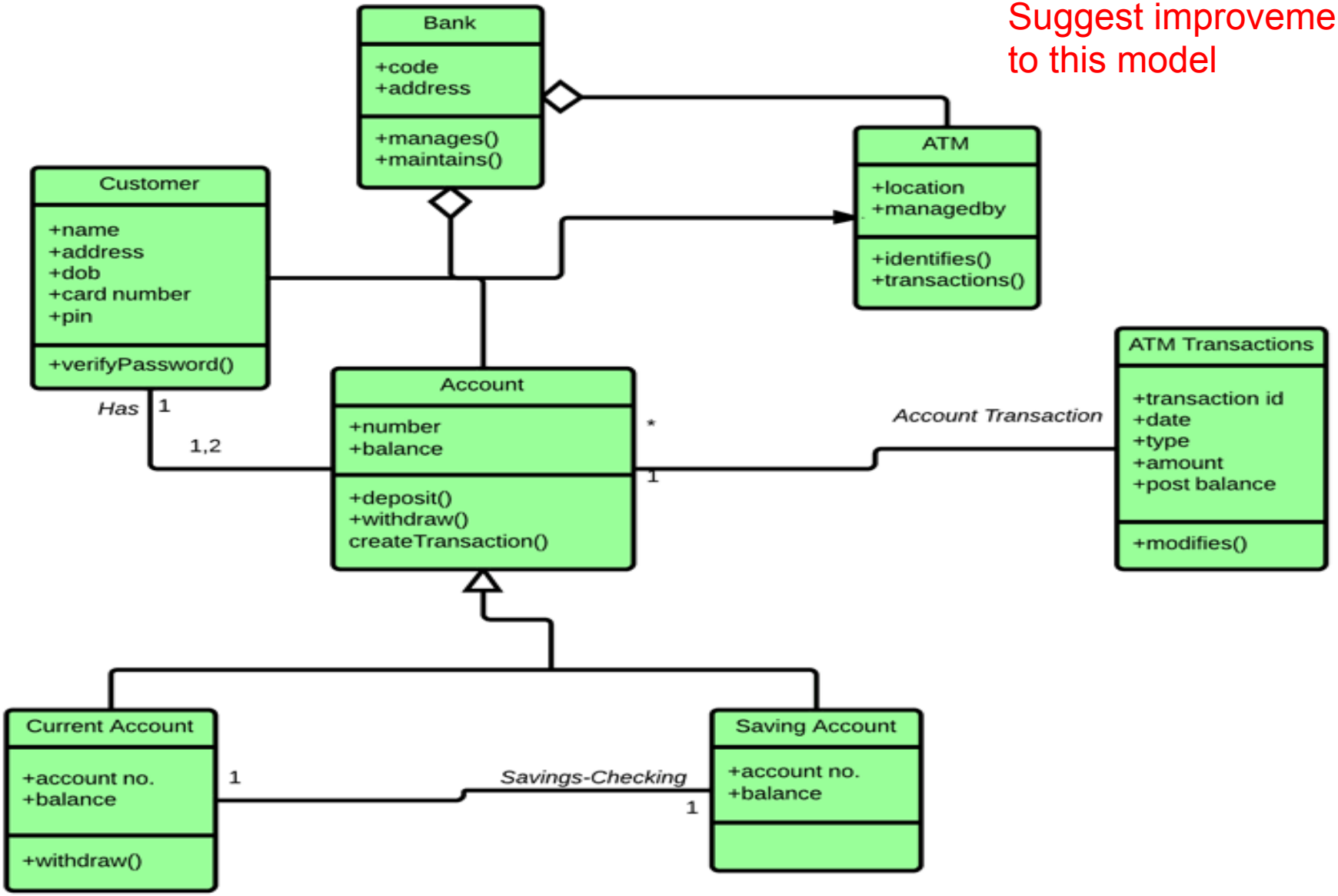
- The top pair AB shows a binary association with two navigable ends.
- The second pair CD shows a binary association with two non-navigable ends.
- The third pair EF shows a binary association with unspecified navigability.
- The fourth pair GH shows a binary association with one end navigable and the other non-navigable.
- The fifth pair IJ shows a binary association with one end navigable and the other having unspecified navigability.

Example of UML Class Diagram



Exercise on UML Class Diagrams

Suggest improvements to this model



Best practices of Designing of the Class Diagram

- Here are some points which should be kept in mind while drawing a class diagram:
  - The name given to classes in a diagram must be meaningful.
  - The relationships between elements need to be identified and specified precisely (naming association ends, multiplicity, navigability).
  - The responsibility for every class needs to be identified.
  - For every class, a minimum number of properties should be specified.
  - User notes should be included whenever you need to clarify some gray areas of the diagram.

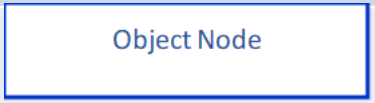
Behavioral Models

What is an Activity diagram?

- A UML activity diagram helps to visualize a certain use case at a more detailed level. It is a behavioral diagram that illustrates the flow of activities through a system.
- UML activity diagrams can also be used to depict a flow of events in a business process. They can be used to examine business processes in order to identify its flow and requirements.
- Activity diagram is another important behavioral diagram in UML diagram to describe dynamic aspects of the system.

Activity Diagram Symbols

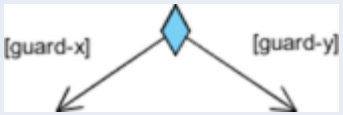
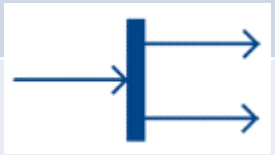
- UML has specified a set of symbols and rules for drawing activity diagrams. Following are the commonly used activity diagram symbols with explanations.

Symbol	Name	Use
	Start/ Initial Node	Used to represent the starting point or the initial state of an activity
	Object Node	Represent an object that is connected to a set of Object Flows
	Activity	Used to represent the activities of the process

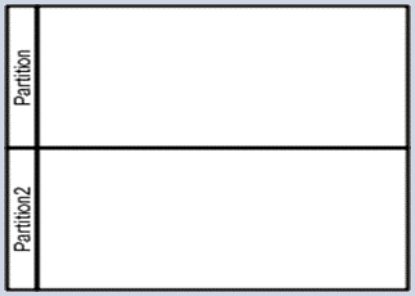
Activity Diagram Symbols

	Action	Used to represent a single step of an activity
	Control Flow	Used to represent the flow of control from one action to the other
	Object Flow	Used to represent the path of objects moving through the activity
	Activity Final Node	Used to mark the end of all control flows within the activity

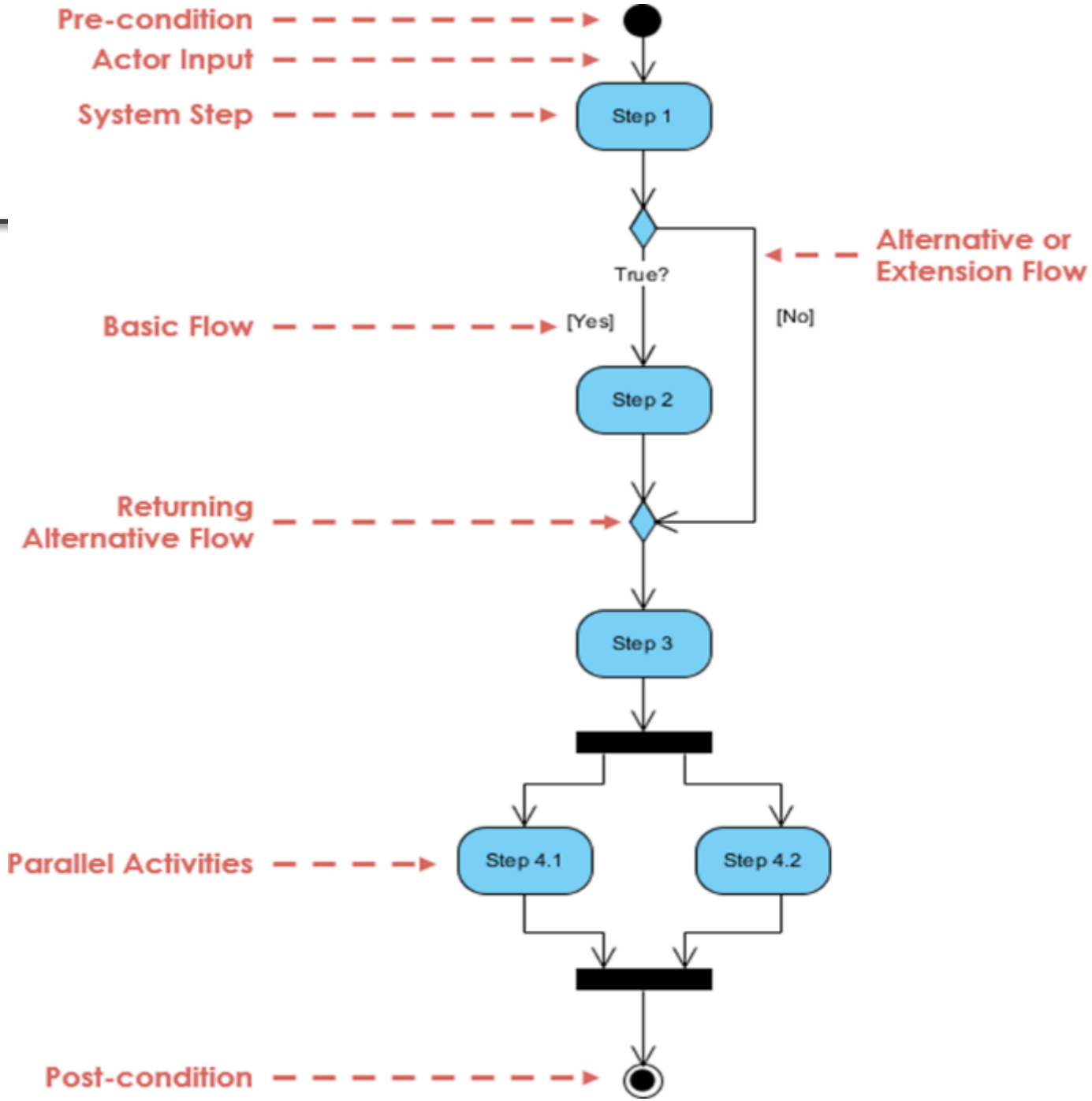
Activity Diagram Symbols

	Flow Final Node	Used to mark the end of a single control flow
	Decision Node	Used to represent a conditional branch point with one input and multiple outputs
	Merge Node	Used to represent the merging of input flows. It has several inputs, but one output.
	Fork	Used to represent a flow that may branch into two or more parallel flow of activities or

Activity Diagram Symbols

	Join node	Used to represent a flow that may branch from two or more parallel flow of activities (or actions) into a single flow
	Note/ Comment	Used to add relevant comments to elements
	A way to group activities performed by the same actor on an activity diagram or to group activities in a single thread. The partition can be in a horizontal direction or vertical direction	A way to group activities performed by the same actor on an activity diagram or to group activities in a single thread

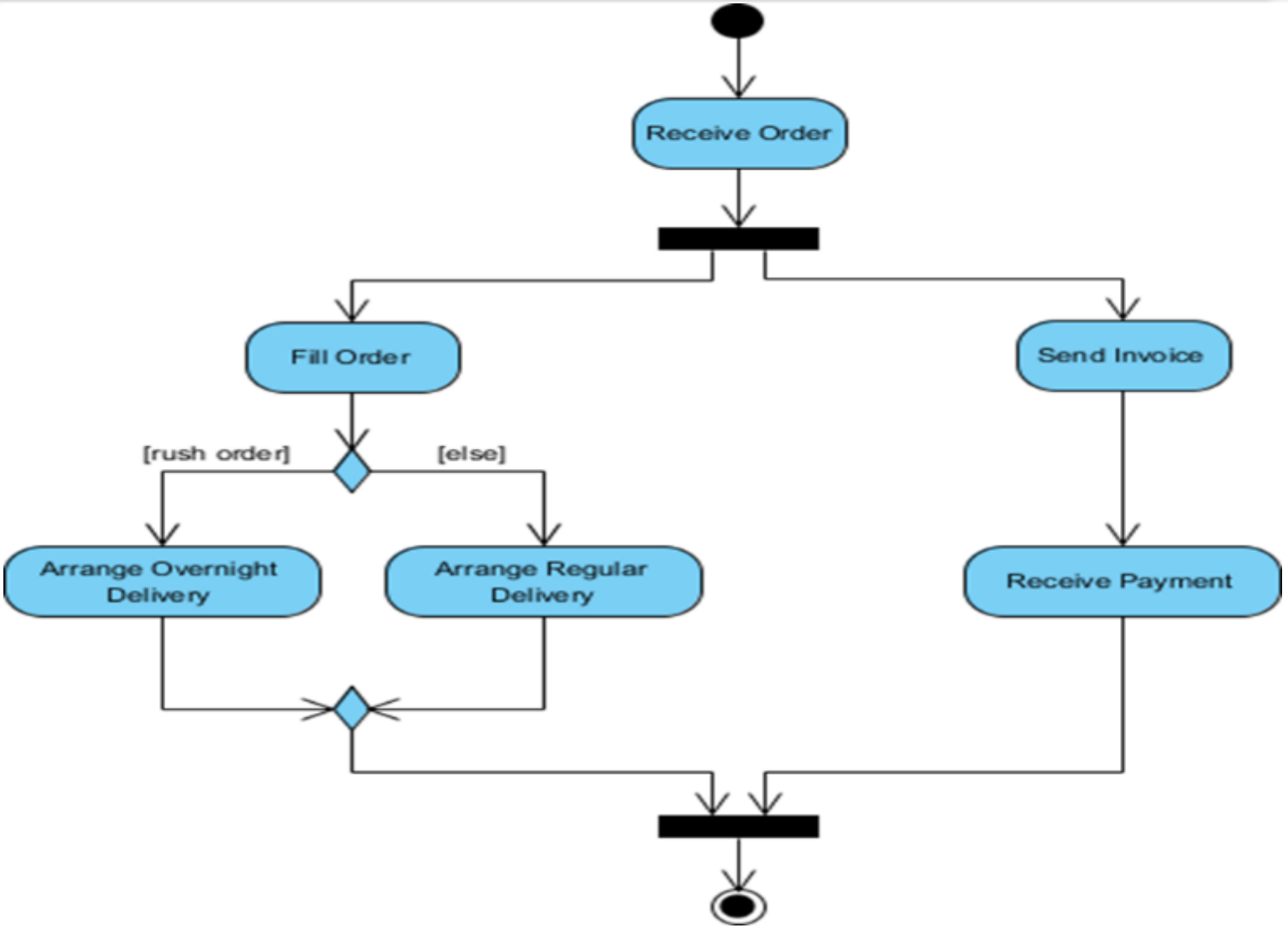
Activity Diagram - Example



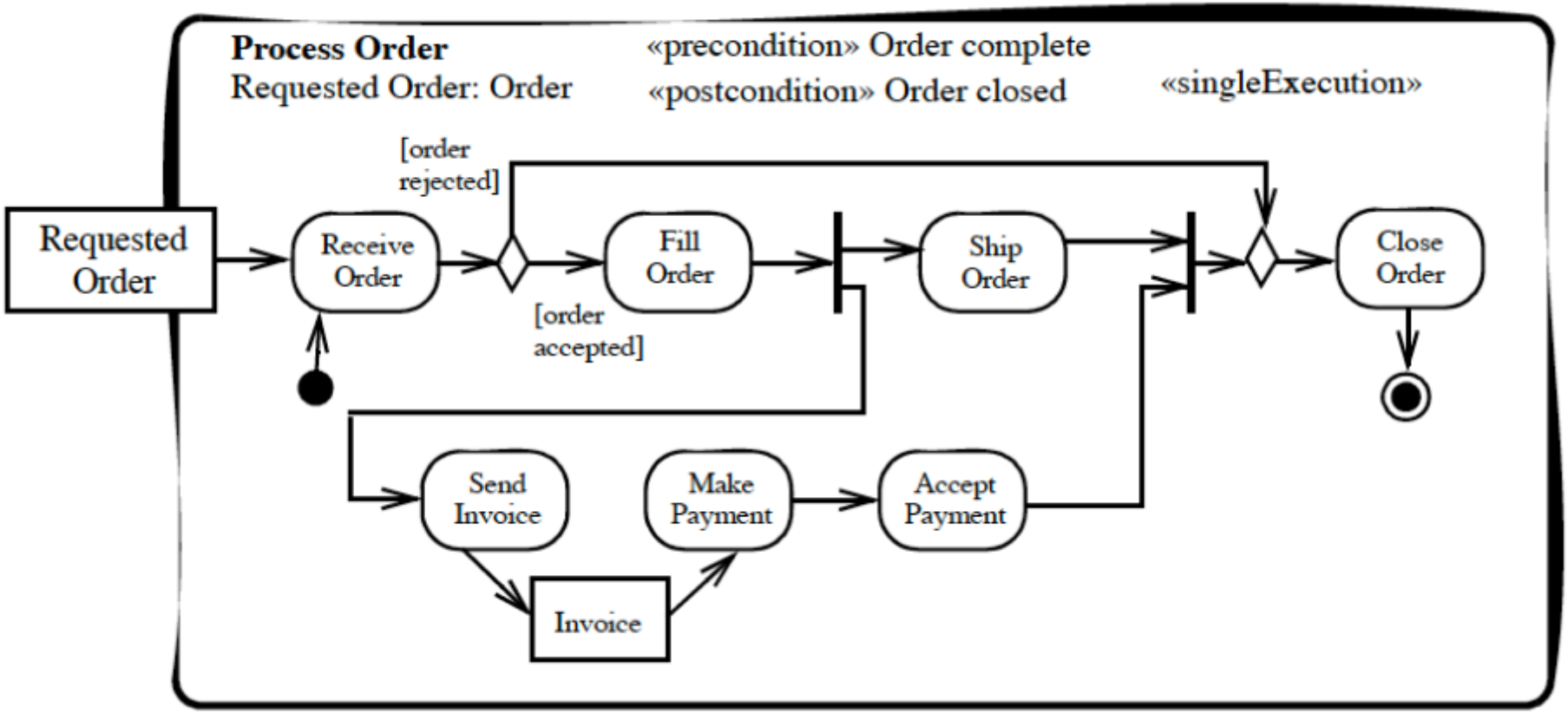
Activity Diagram - Process Order

- In the following slide we have an activity diagram that describes the workflow for processing an order:
- Once the order is received, the activities split into two parallel sets of activities. One side fills and sends the order while the other handles the billing.
- On the Fill Order side, the method of delivery is decided conditionally. Depending on the condition either the Overnight Delivery activity or the Regular Delivery activity is performed.
- Finally, the parallel activities combine to close the order.

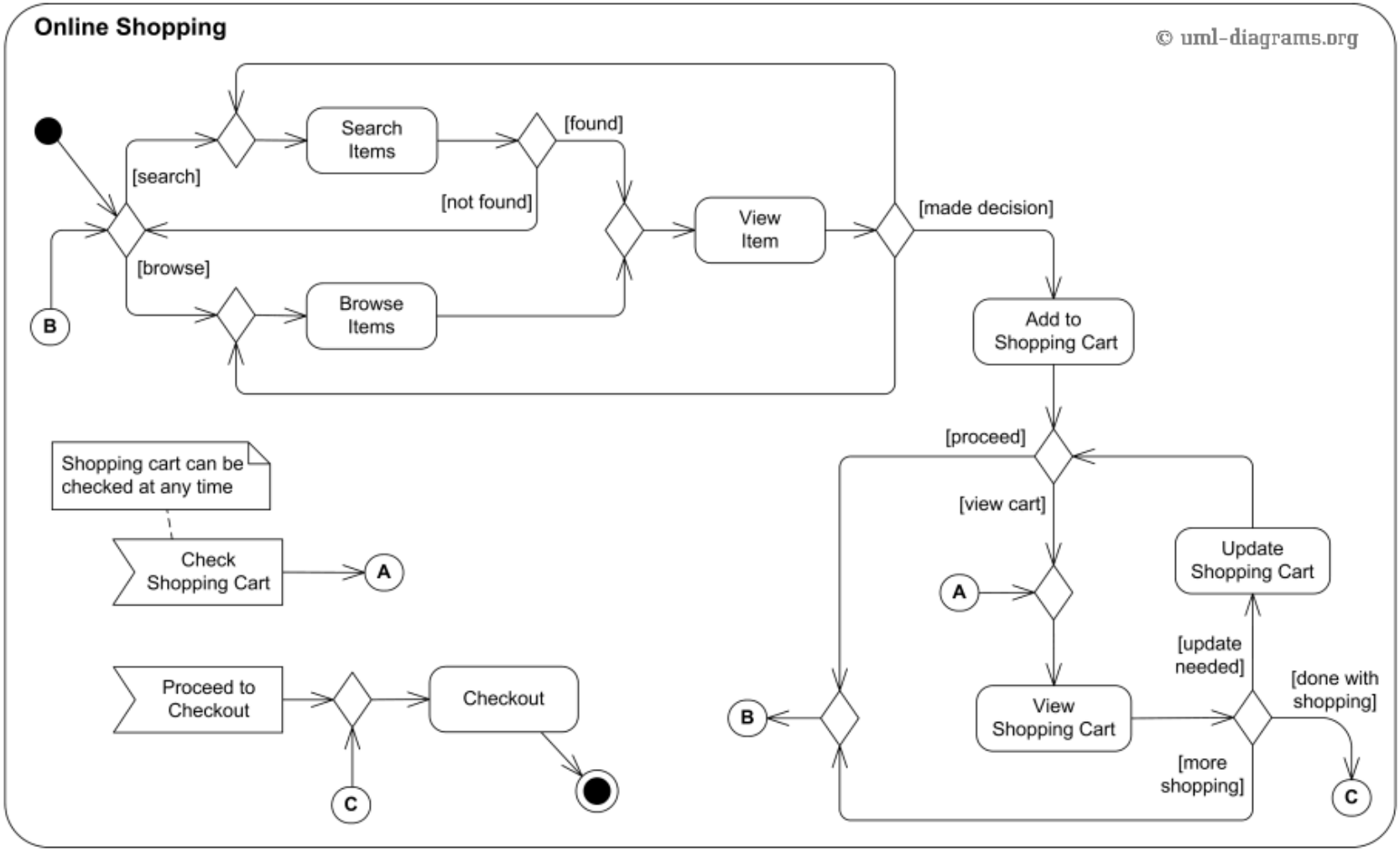
Activity Diagram -Process Order



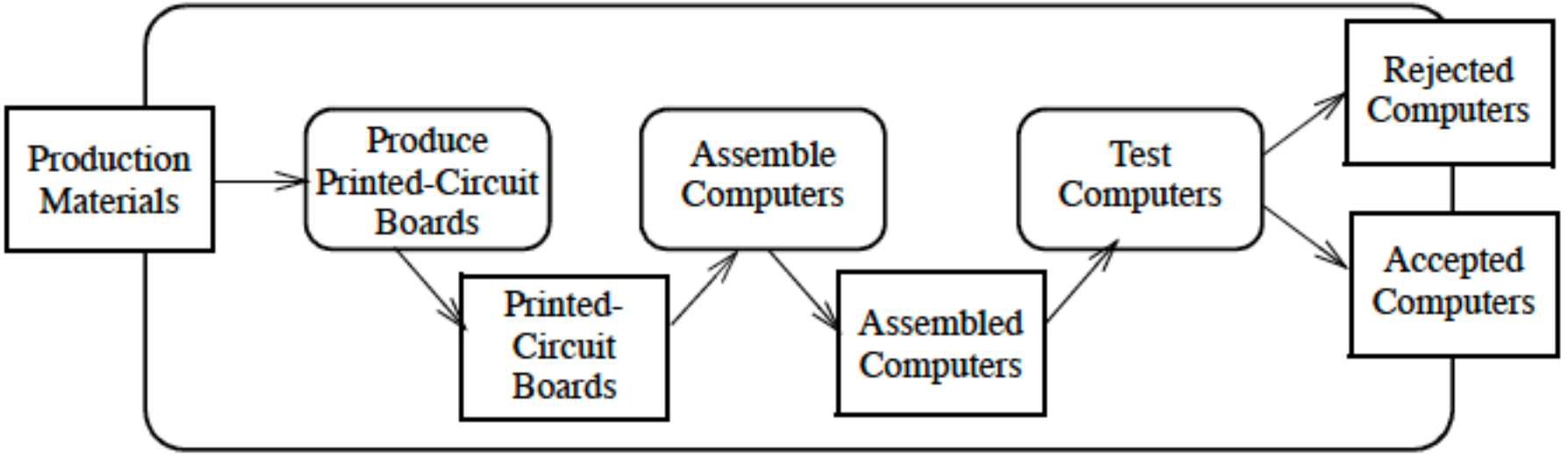
Activity Diagram for Order Handling



Activity diagram of : online shopping

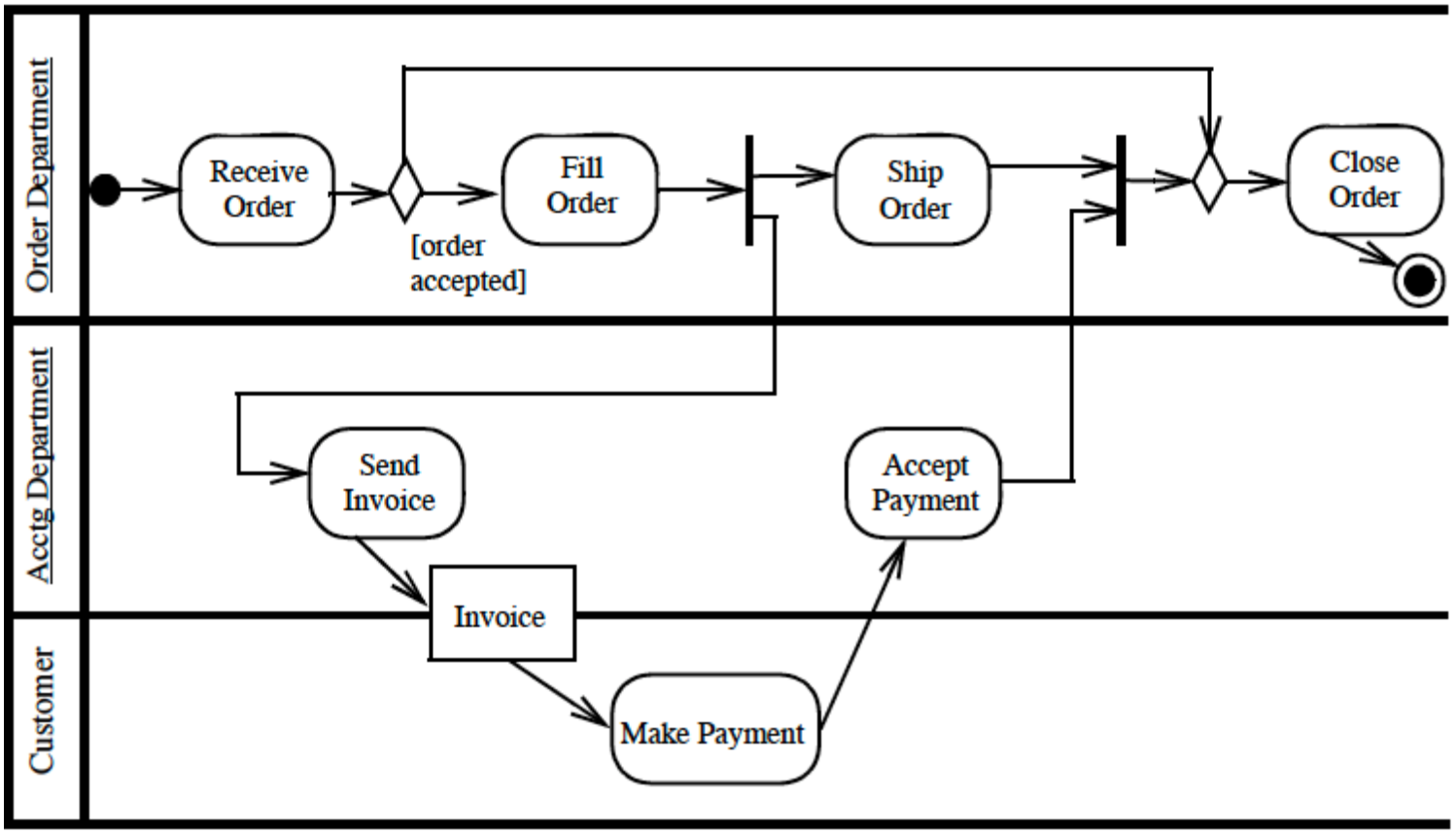


Activity diagram with emphasis on Object nodes

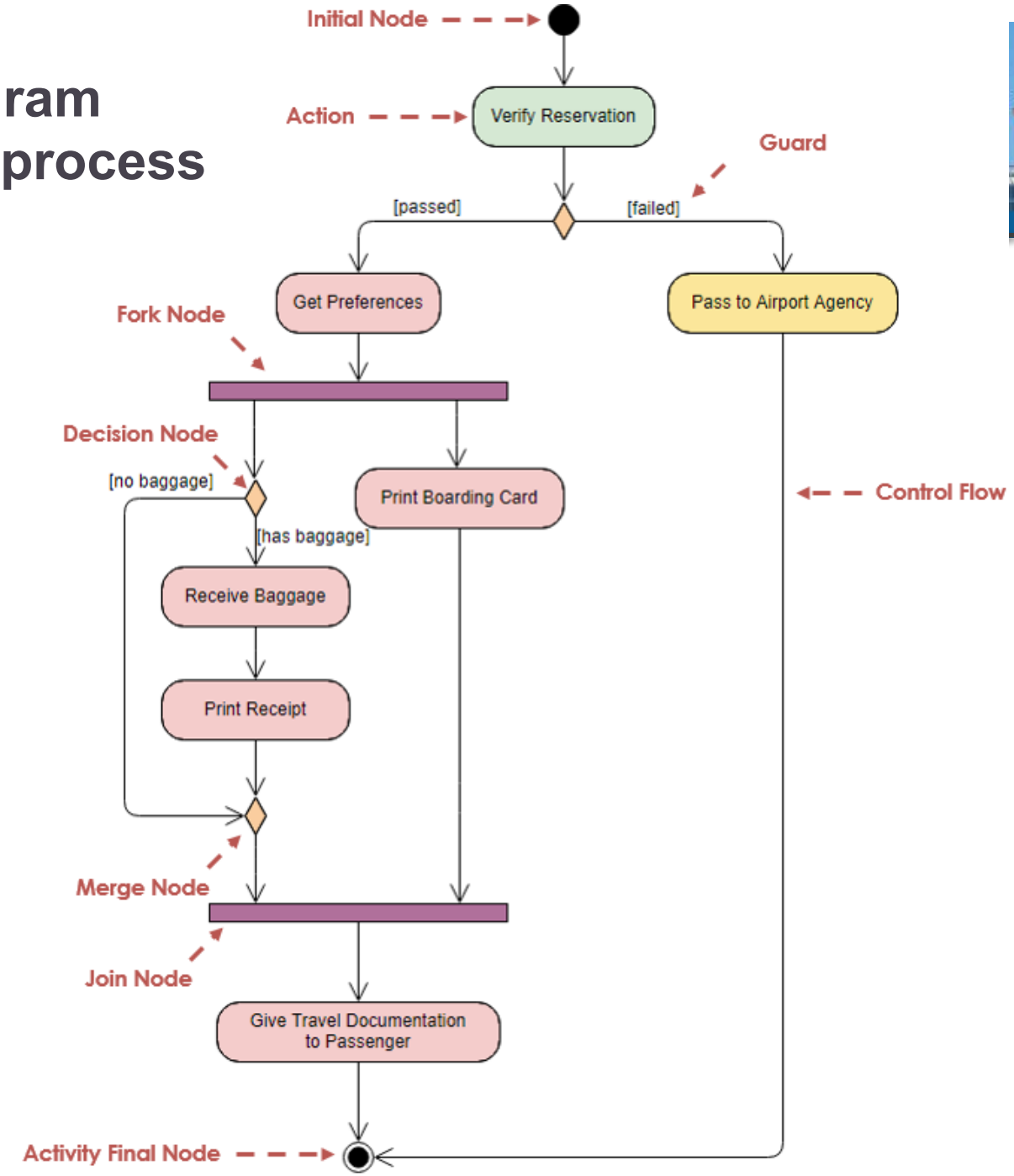


Activity Diagrams with Swimlanes

A swimlane is a way to group activities performed by the same actor on an activity diagram or to group activities in a single thread



Activity Diagram
Reservation process

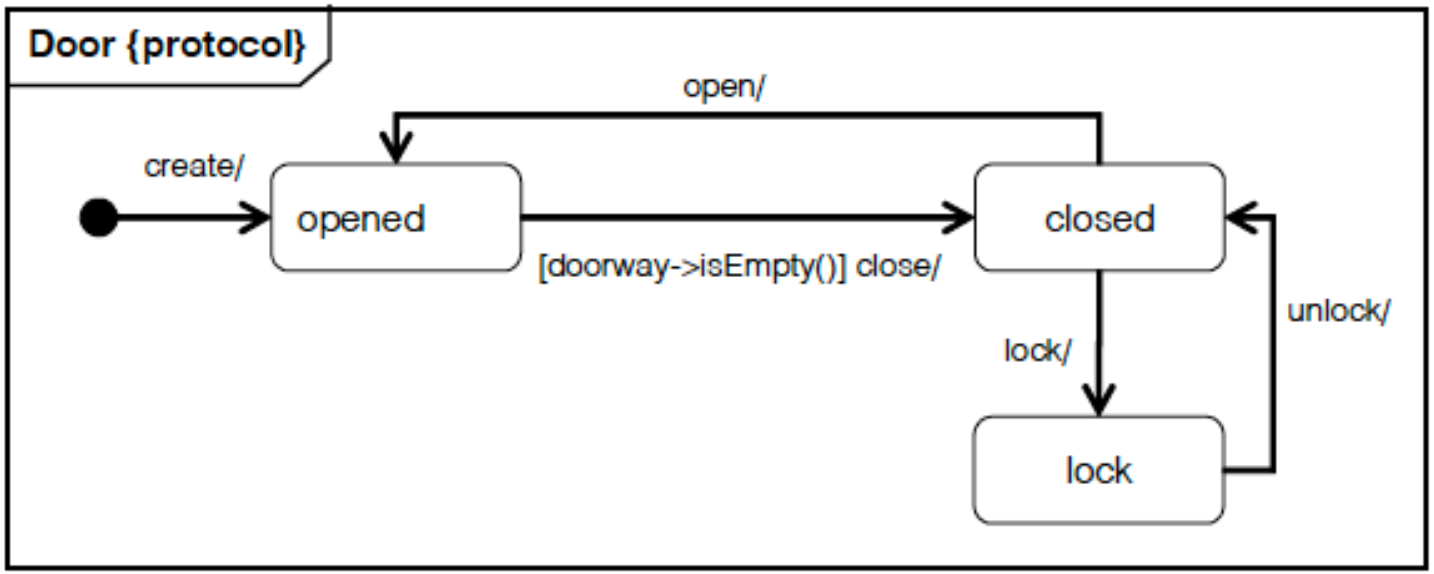


Statechart (or state Machine) Diagram

- It's a behavioral diagram. State diagrams are also referred to as State machines and Statechart Diagrams.
- The State machine diagram can be used for modeling discrete behaviour through finite state transition systems.
- A state diagram is used to model the dynamic behavior of a class in response to time and external stimuli.

Uses of Statechart Diagram

- Statecharts are used to model the events responsible for change from states (we do not show what processes cause those events).



Basic Components of a State Chart Diagram

- **Initial state –** We use a black filled circle represent the initial state of a System or a class.

Figure – initial state notation

- **Transition –** We use a solid arrow to represent the transition from one state to another. The arrow is labelled with the event which causes the change of state.

Figure – transition

Basic Components of a State Chart Diagram

- **State –** We use a rounded rectangle to represent a state. A state represents the conditions or circumstances of an object of a class at an instant of time.

Figure – state notation

Basic Components of a State Chart Diagram

- **Self transition –** We use a solid arrow pointing back to the state itself to represent a self transition. There might be scenarios when the state of the object does not change upon the occurrence of an event. We use self transitions to represent such cases.

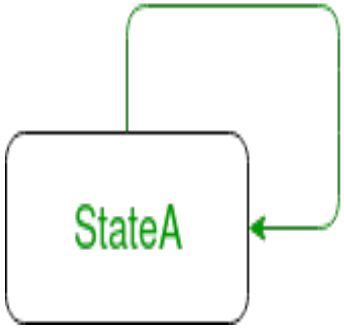


Figure – self transition notation

Basic Components of a State Chart Diagram

- **Composite state** – We use a rounded rectangle to represent a composite state also. We represent a state with internal activities using a composite state.

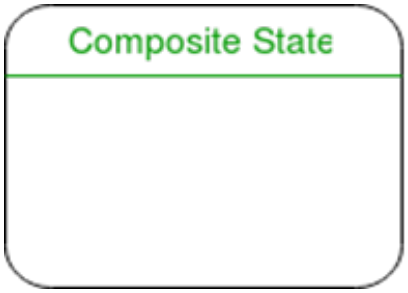
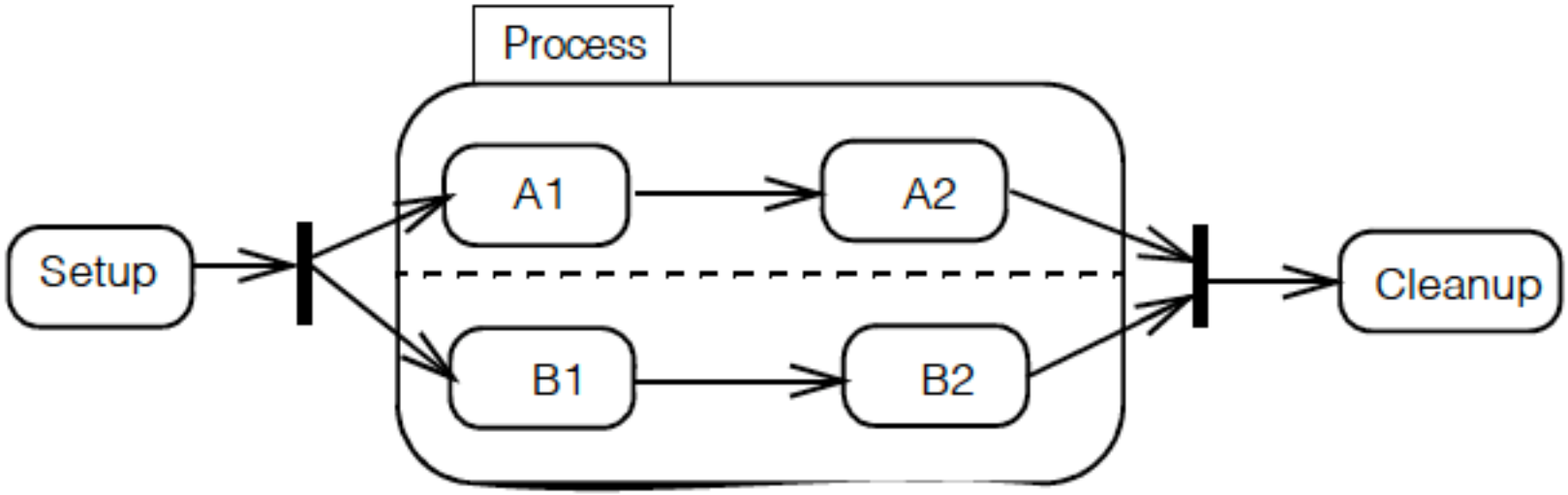


Figure – a state with internal activities

- **Final state –** We use a filled circle within a circle notation to represent the final state in a state machine diagram.

Figure – final state notation

Fork and join in state diagrams



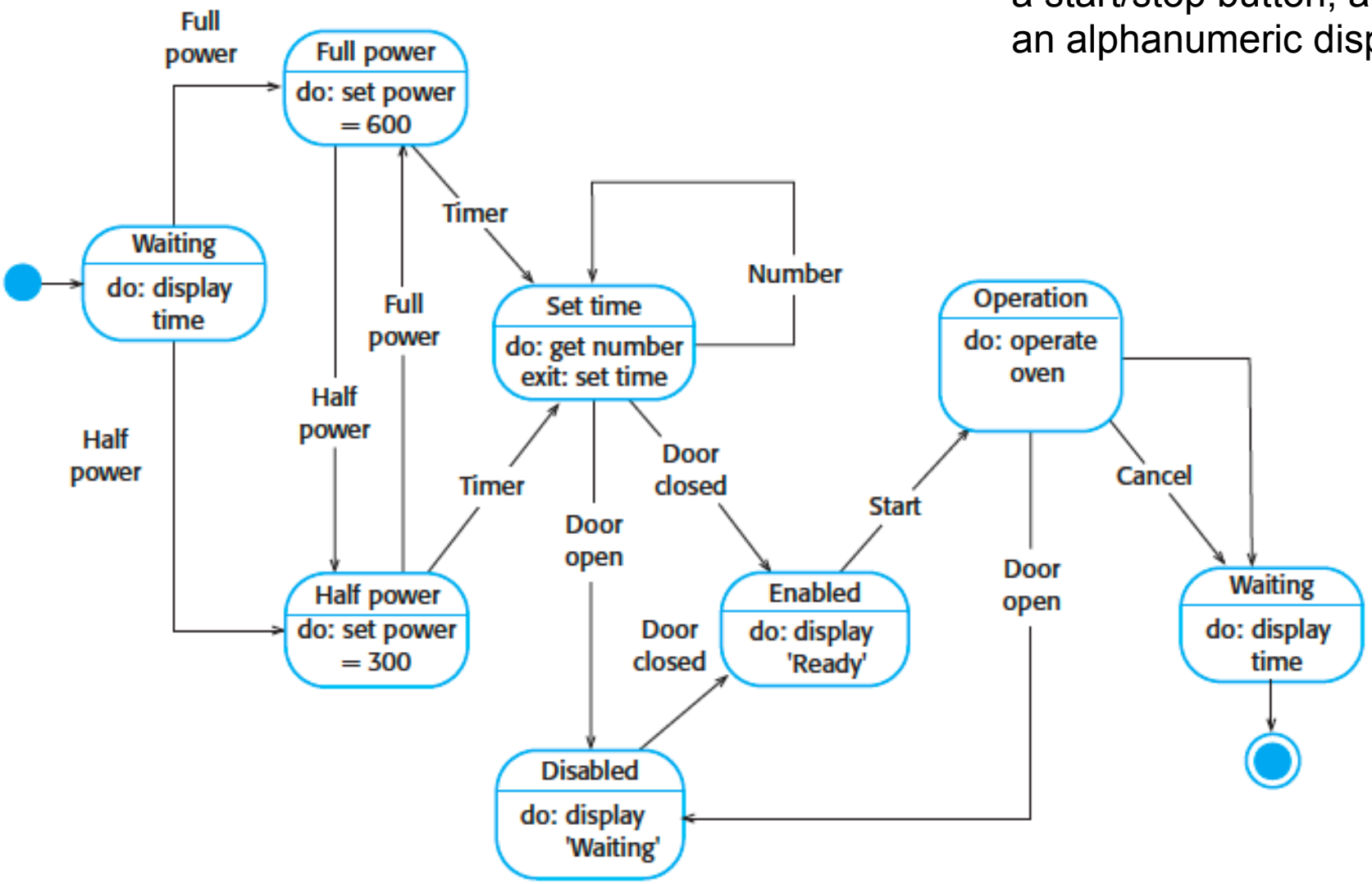
Only transitions that occur in mutually orthogonal regions may be fired simultaneously.

Steps to Draw a State Diagram

- Identify the initial state and the final terminating states.
- Identify the possible states in which the object can exist (boundary values corresponding to different attributes guide us in identifying different states).
- Label the events which trigger these transitions.

State diagram of a microwave oven

This simple oven has a switch to select full or half power, a numeric keypad to input the cooking time, a start/stop button, and an alphanumeric display.



State diagram example: Petrol pump