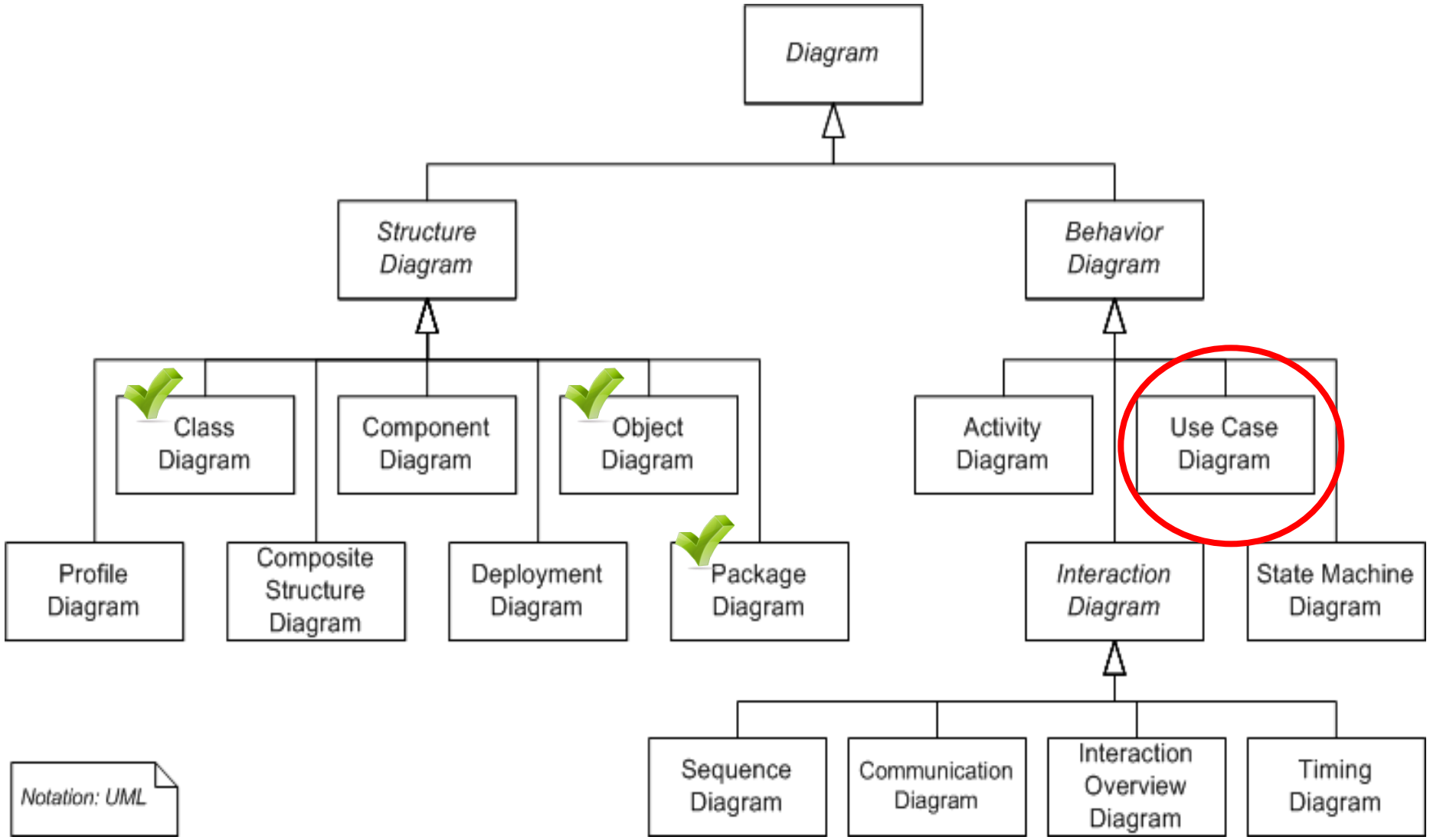
Use Case Diagrams

Week 5: System Modeling, Part 2

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

UML Diagrams



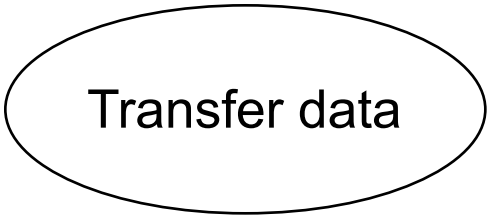
Requirements Discovery: A Reminder

- The process of gathering information about the required or existing systems and distilling the user and system requirements from this information.
- Sources of information during the requirements discovery phase include documentation, system stakeholders, and specifications of similar systems.
- You interact with stakeholders through interviews and observation.
- You may also use scenarios and prototypes to help stakeholders understand what the system will be like.
- Use cases are a requirements discovery technique that were first introduced in the **Objectory** method.
- They have now become a fundamental feature of the UML.

Use Cases

"A **use case** is a description of a set of sequences of actions, including variants, that a system performs to yield an observable result of value to an actor."

— *The UML User Guide, 2nd edition, 2005*



- A use case is rendered as an **ellipse** in a use case diagram.
- A use case is always labeled with its name.

Use Case Descriptions: An Example

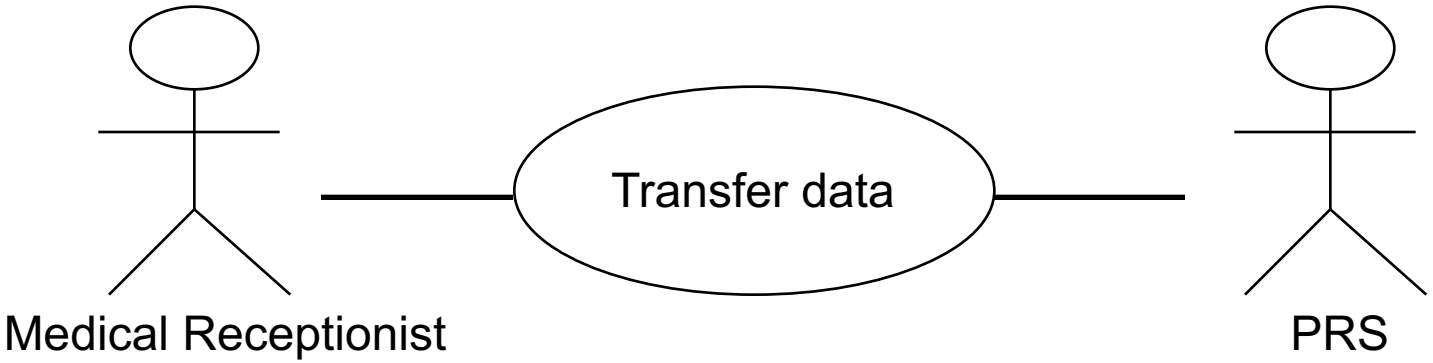
Mentcare system: Transfer data	
Actors	Medical receptionist, Patient records system (PRS)
Description	A receptionist may transfer data from the Mentcare system to a general patient record database that is maintained by a health authority. The information transferred may either be updated personal information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

Actors

"An actor is an idealization of a role played by an external person, process, or thing interacting with a system, subsystem, or class. An actor characterizes the interactions that outside users may have with the system."
- The UML Reference Manual, 2nd Ed, 2005

An actor is rendered as a **stick figure** in a use case diagram.

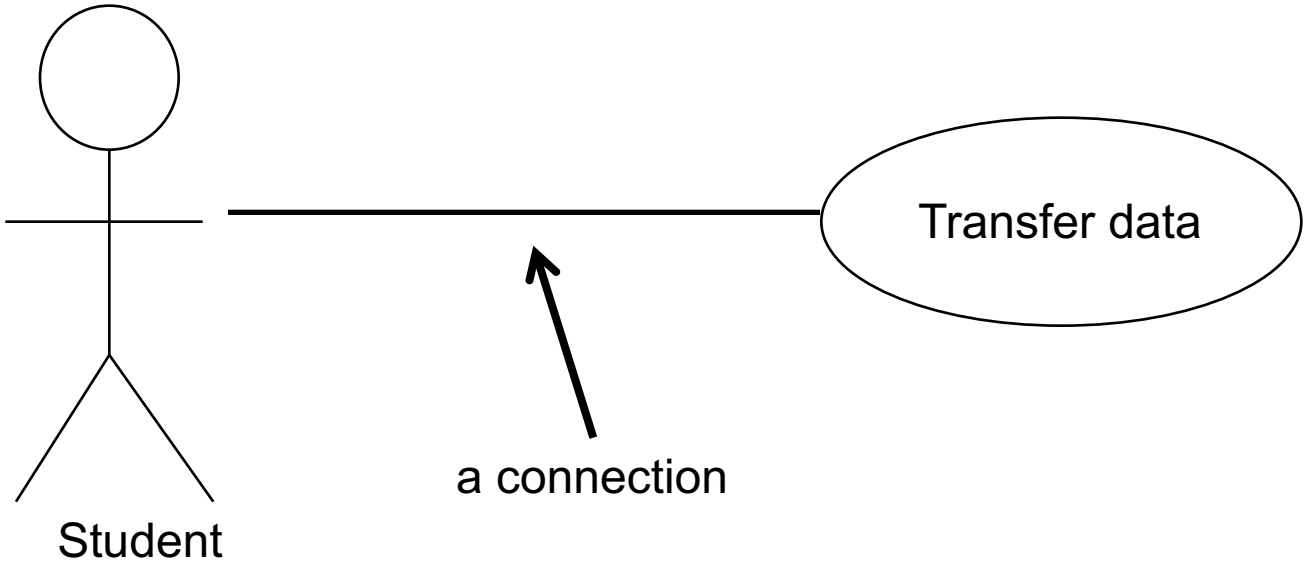
Each actor participates in one or more use cases.



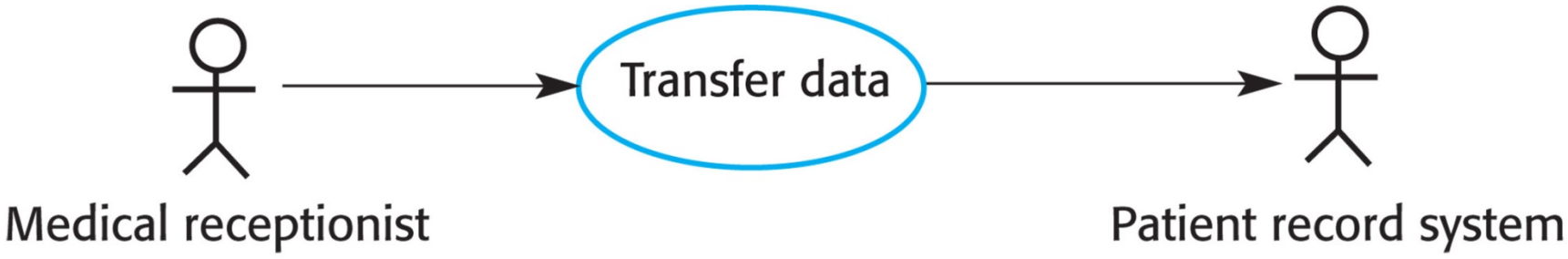
Conections

Connection

- A connection is an ‘association’ between an actor and a use case. (Only associations are allowed.)
- Indicates communication.

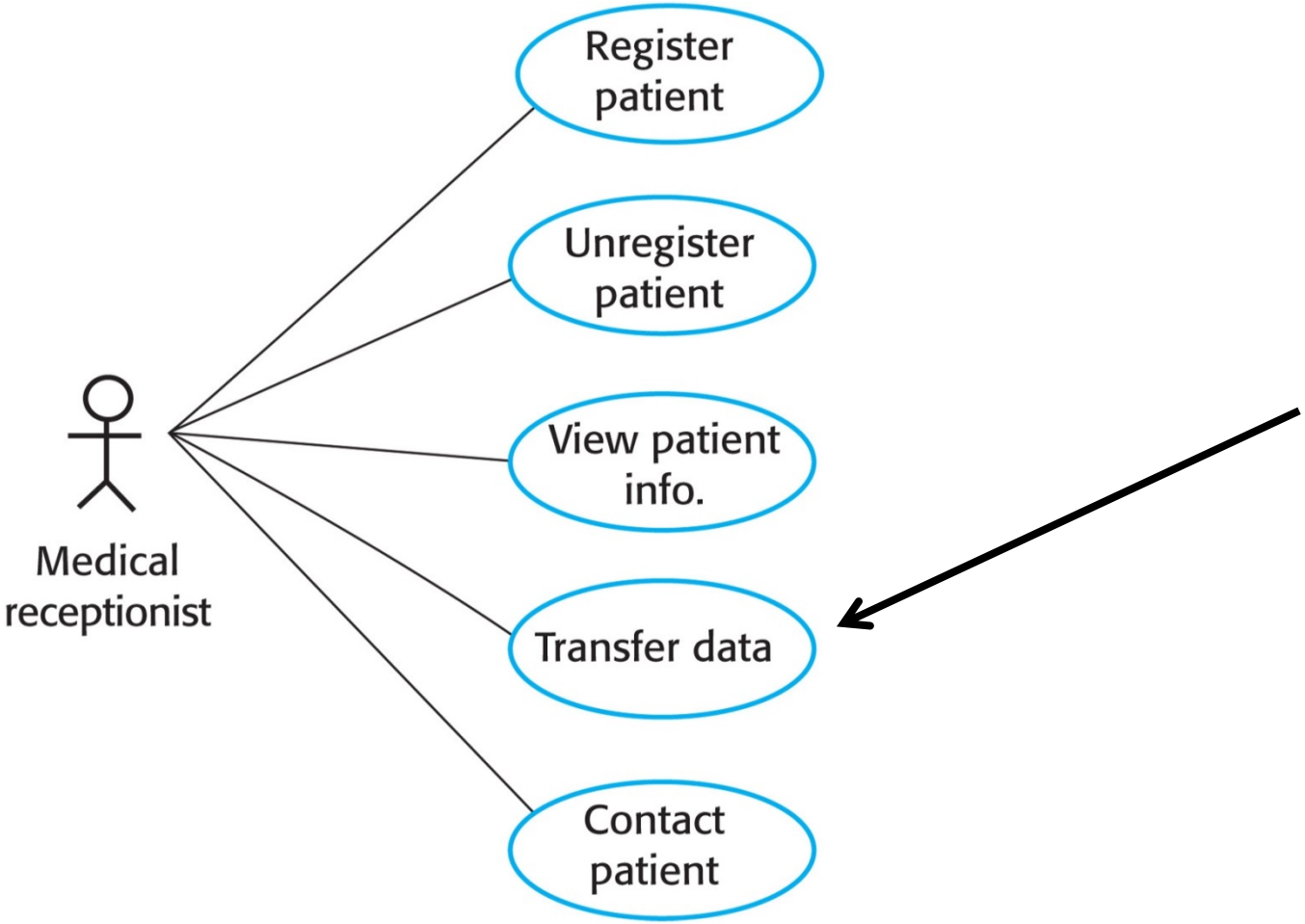


A Use Case for ‘Transfer Data’

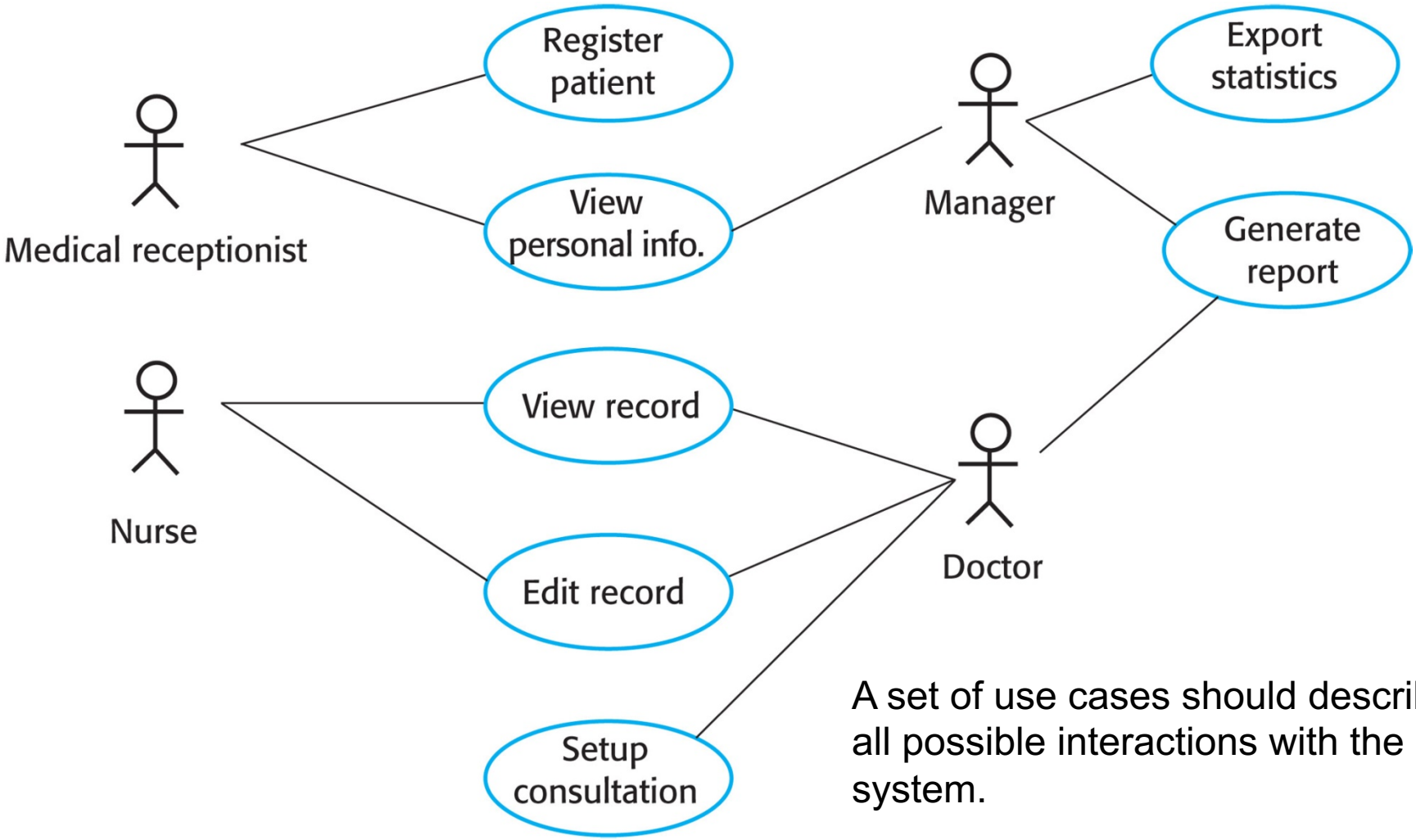


Mentcare system: Transfer data	
Actors	Medical receptionist, Patient records system (PRS)
Description	A receptionist may transfer data from the Mentcare system to a general patient record database that is maintained by a health authority. The information transferred may either be updated personal information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

Multiple Use Cases



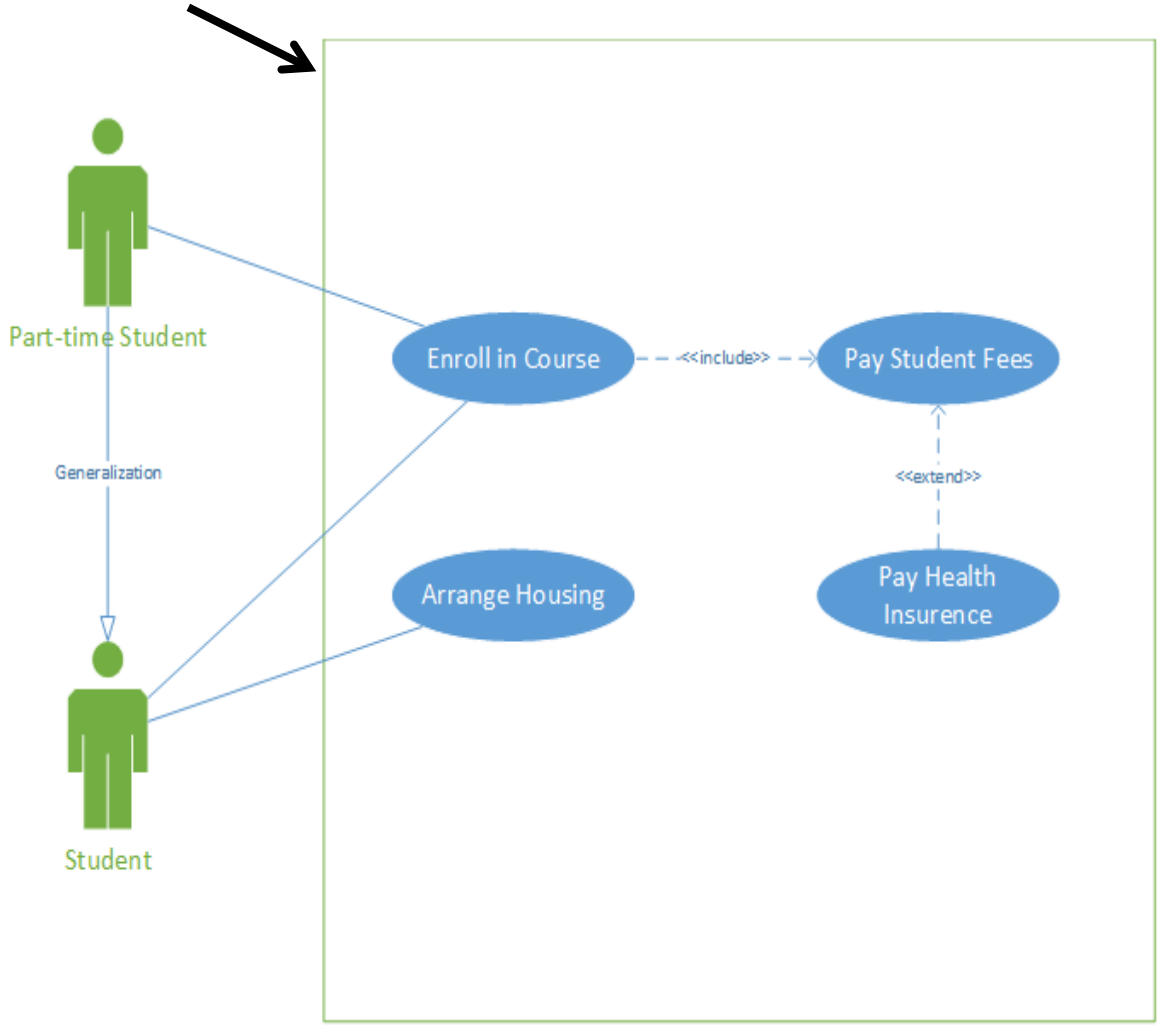
Multiple Use Cases for Mentcare



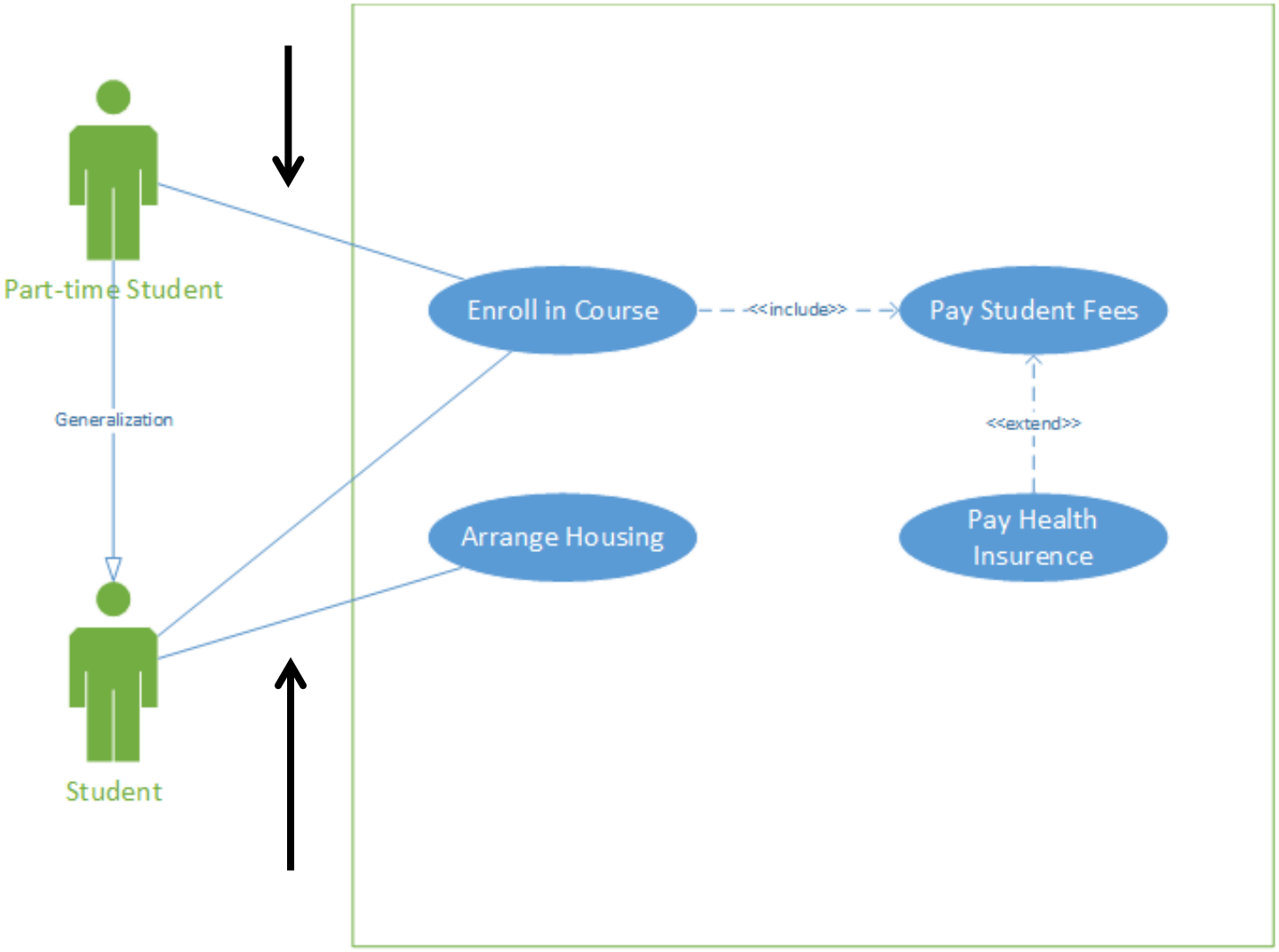
A set of use cases should describe all possible interactions with the system.

Boundaries

- A boundary is the dividing line between the system and its environment.
- Use cases are within the boundary.
- Actors are outside of the boundary.

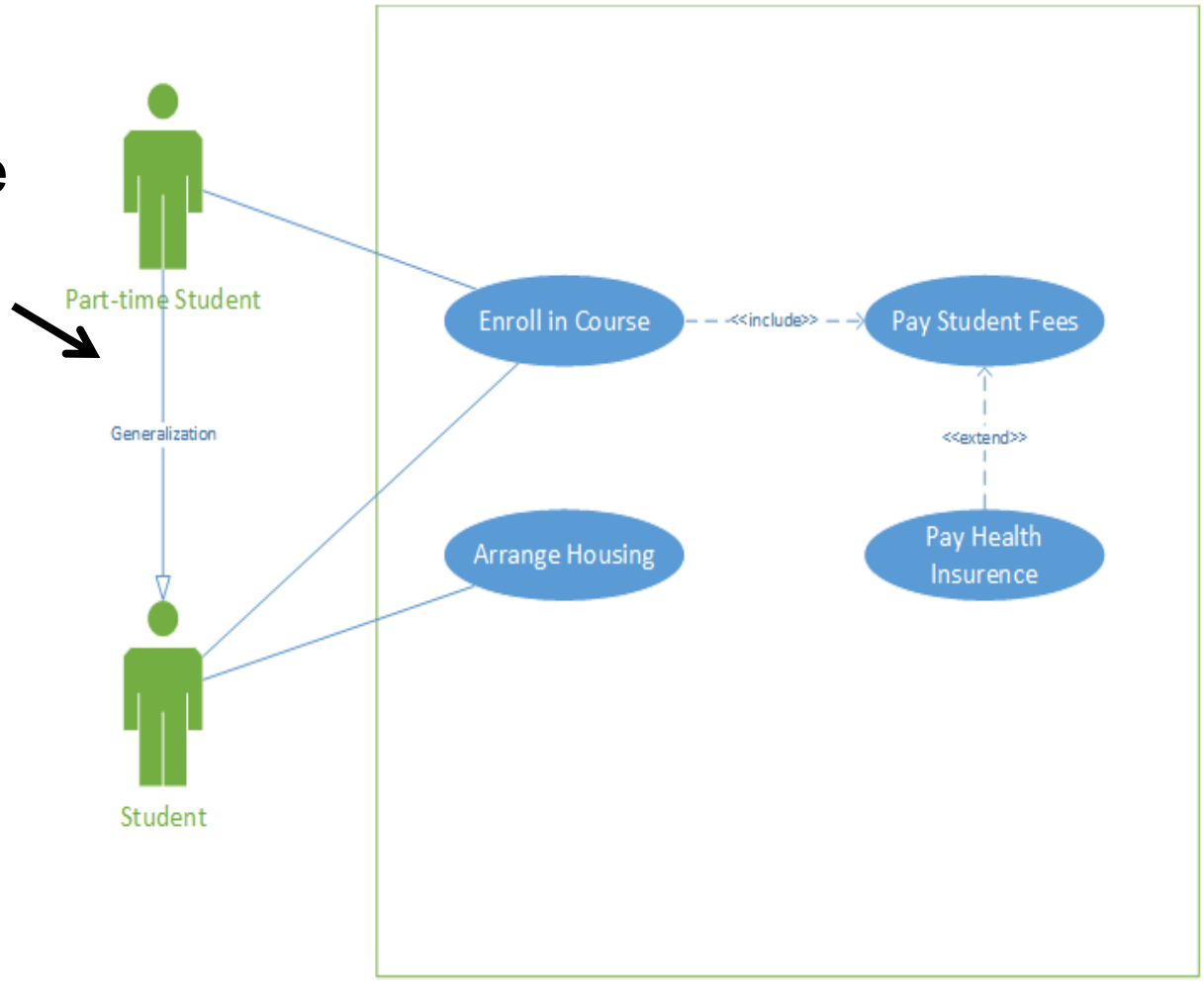


Relationships: Association



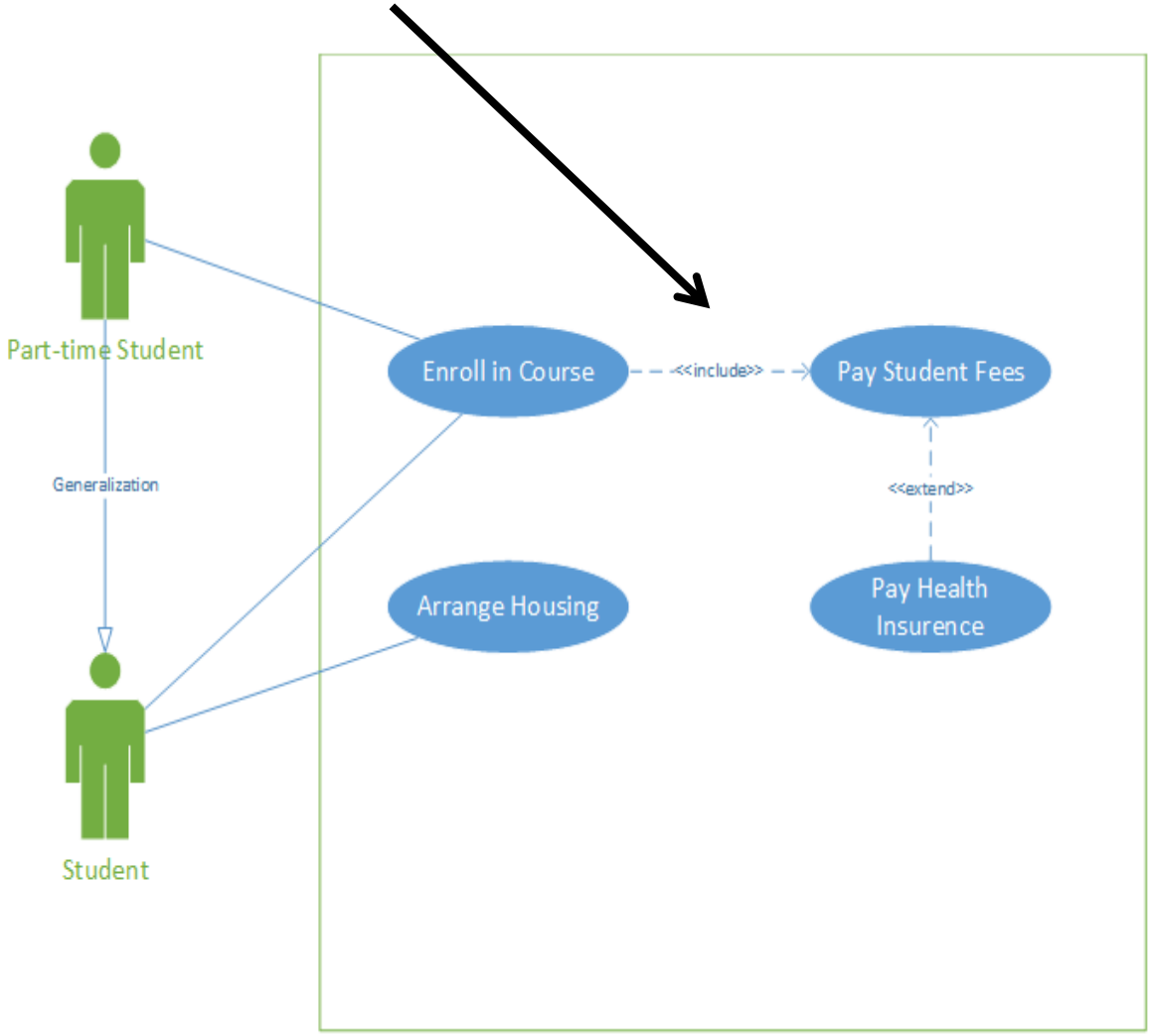
Relationships: Generalization

- One thing is more general than another thing.
- Arrow points to the general thing.



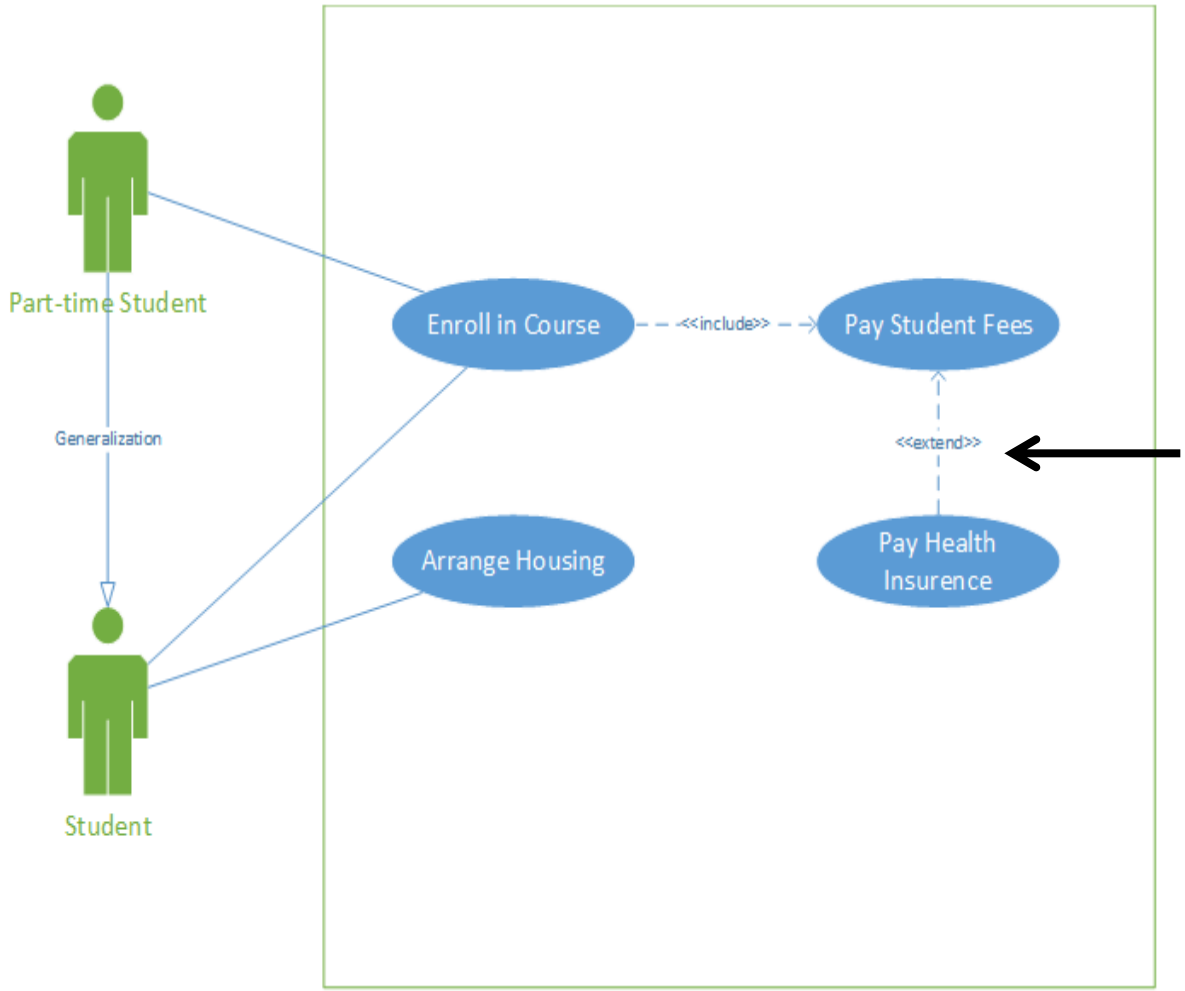
Relationships: <<Include>>

<<include>> indicates a use case that is used (invoked) by another use case.

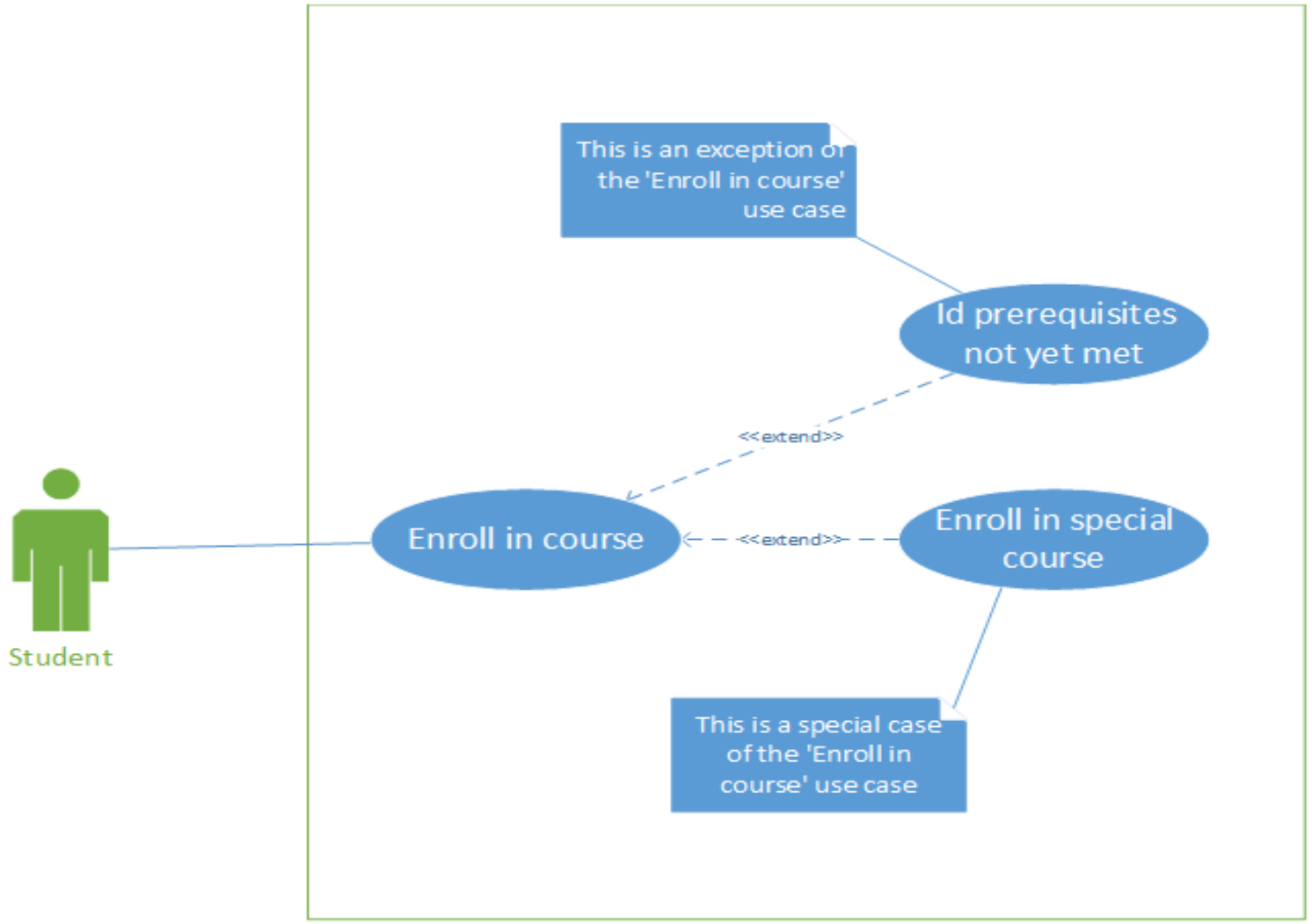


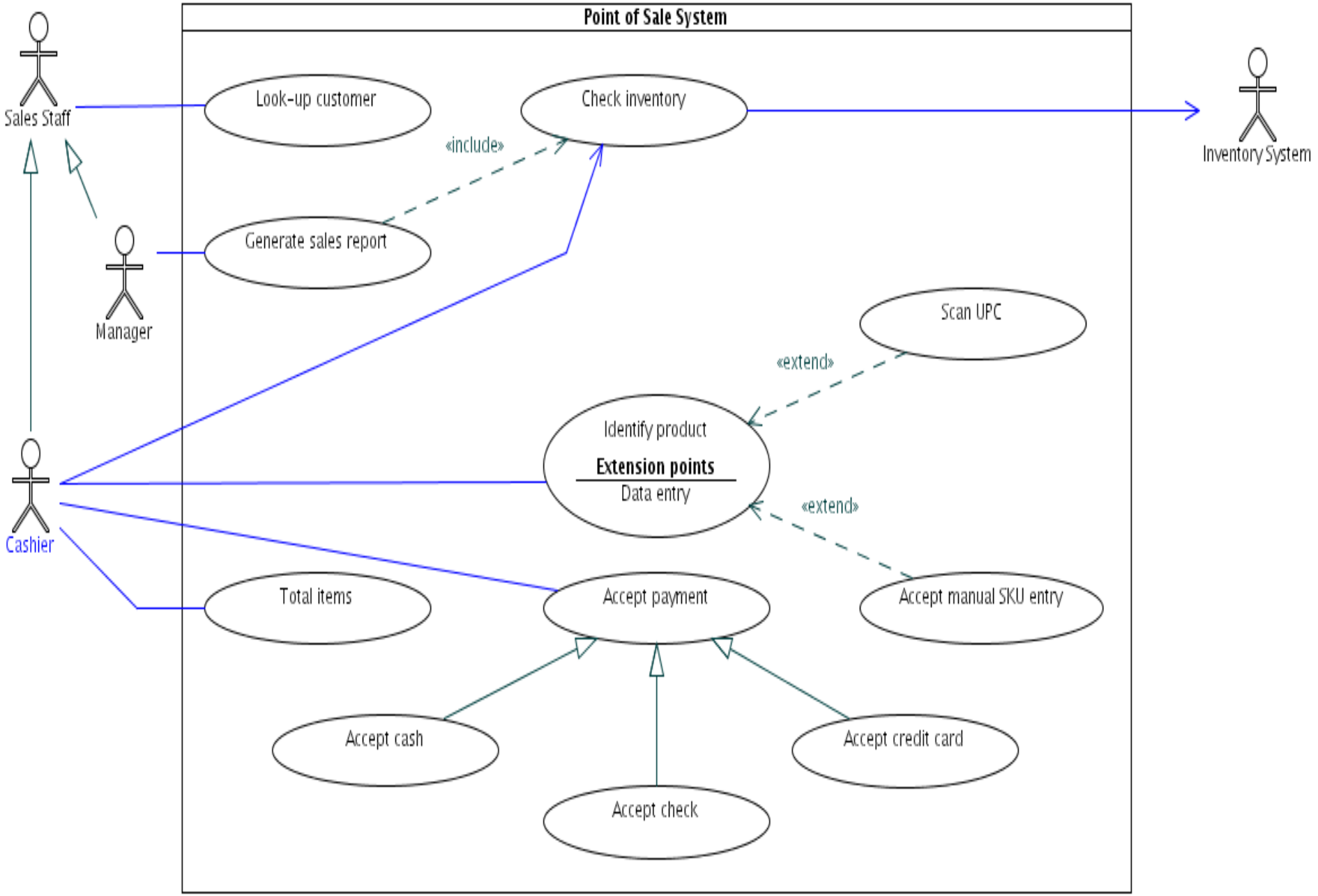
Relationships: <<Extend>>

- <<extend>> indicates **variations** or **exceptions**.
- Arrow goes from extended to basic use case.



Relationships: <<Extend>>




Point of Sale System
Sales Staff
Manager
Cashier
Look-up customer
Check inventory
Generate sales report
«include»
Inventory System
Scan UPC
«extend»
Identify product
Extension points
Data entry
«extend»
Total items
Accept payment
Accept manual SKU entry
Accept cash
Accept credit card
Accept check

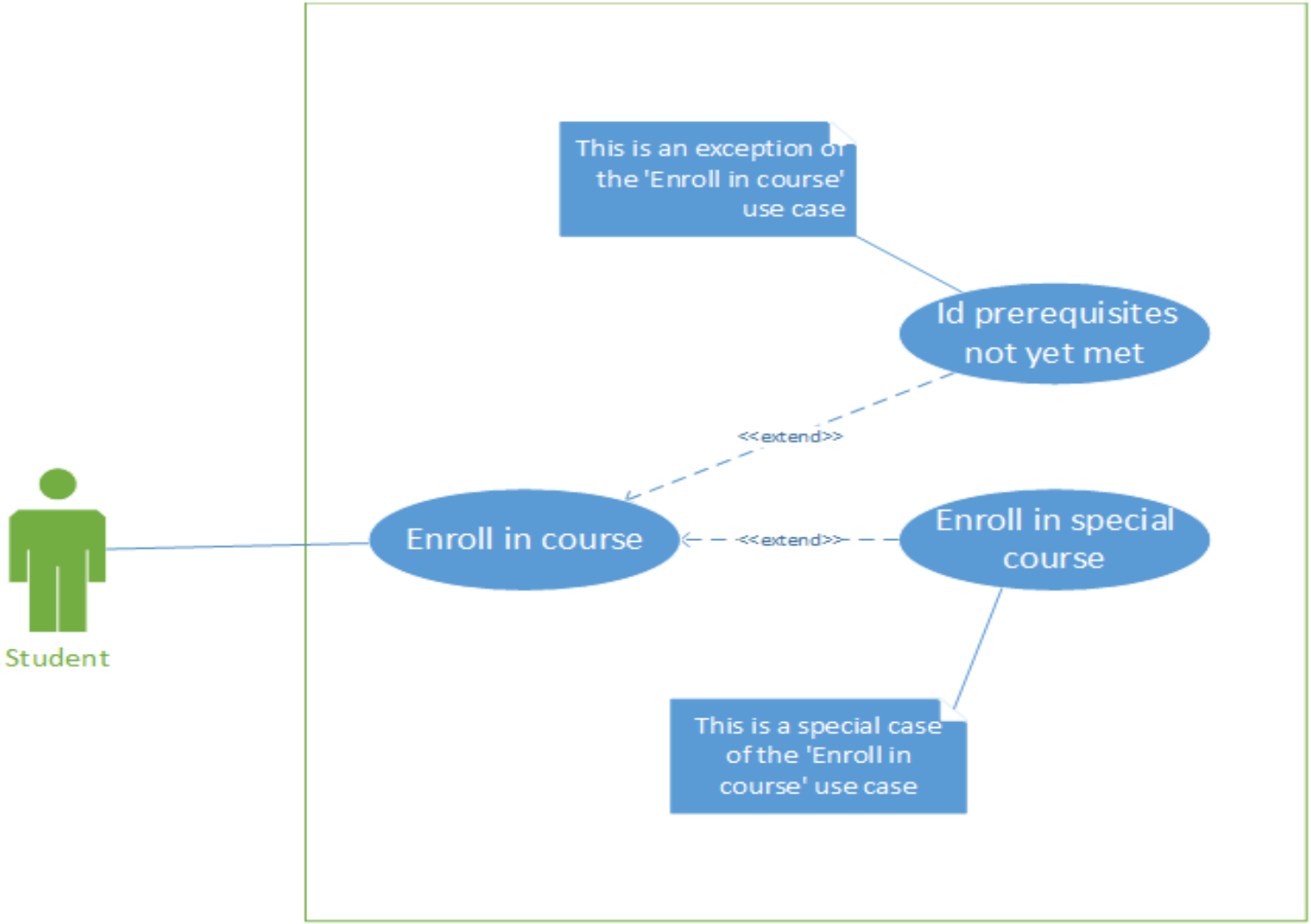
Use Case Modeling

- Most of a system's **functional requirements** can be expressed as use cases. The use case diagrams in UML are essential for managing these requirements.
- To model the requirements of a system with use cases:
  1. Establish the context of a system by identifying the actors that surround it.
  2. For each actor, consider the services that each actor expects/requires the system to provide.
  3. Name those services as use cases.
  4. Factor common services into a new use cases (<<include>>).
  5. Factor variant services into use cases that extend the original use cases (<<extend>>).
  6. Model these use cases, actors, and relationship in a use case diagram.
  7. Adorn these use cases with notes or constraints that assert **nonfunctional requirements**.

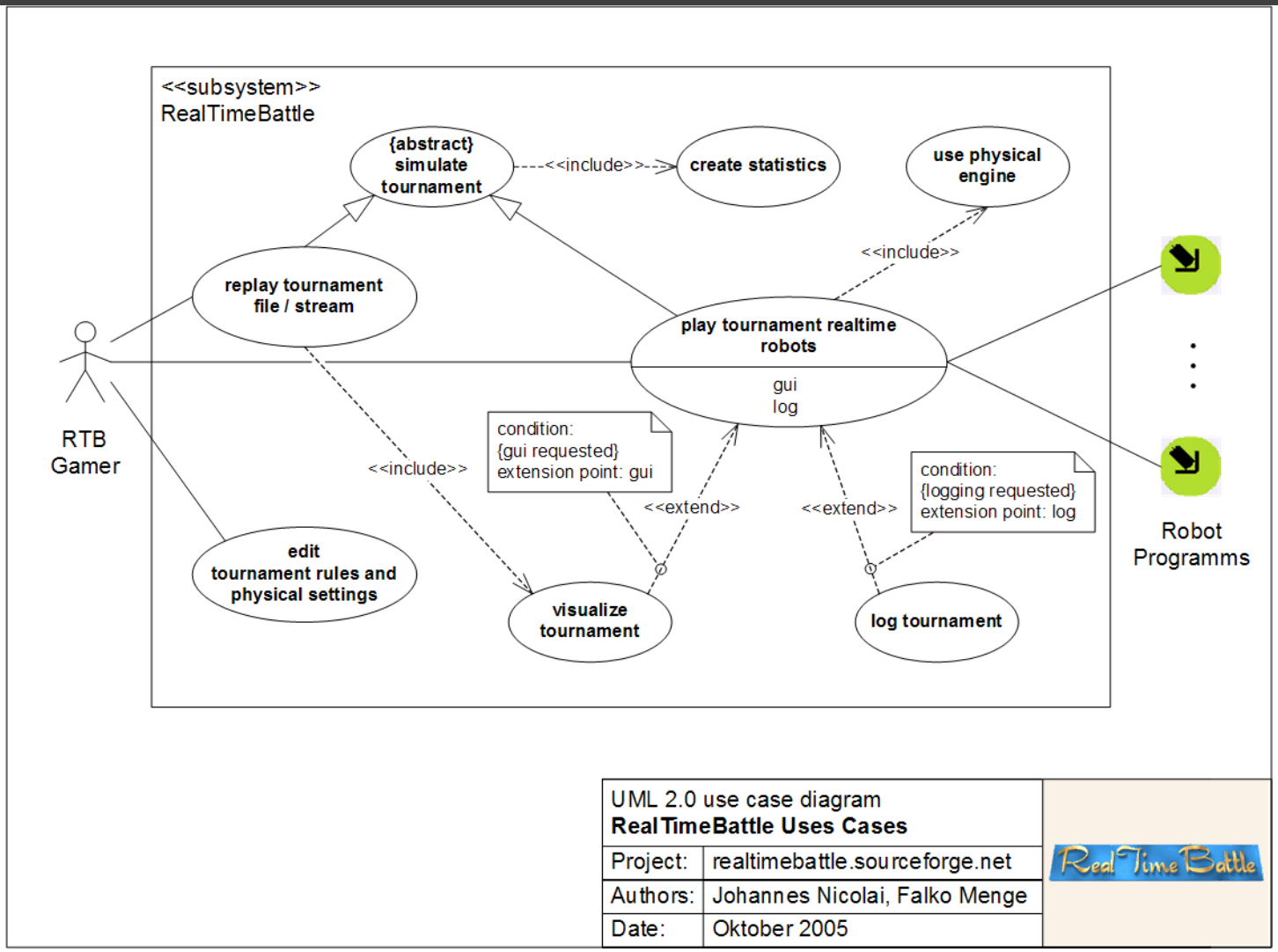
Adornments

7. Adorn these use cases with notes or constraints that assert nonfunctional requirements.
   - ❖ Adornments are textual or graphical items, which can be added to the basic notation of a UML building block in order to visualize some details from that element's specification.
     - ❖ One of the most important kinds of adornments is a **note**, which is a graphical symbol used for adding some comments or constraints to an element, to help clarify the models that are being created.
   - ❖ We may use notes to attach some additional information to our model, such as an explanation, a requirement, or just simply an observation.

Adornments: Notes



Adornments



http://rtb-team.sourceforge.net/analysis-diagrams/images/UML/RealTimeBattle%20-%20Use%20Cases%20(UML%202.0%20use%20case%20diagram).png

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

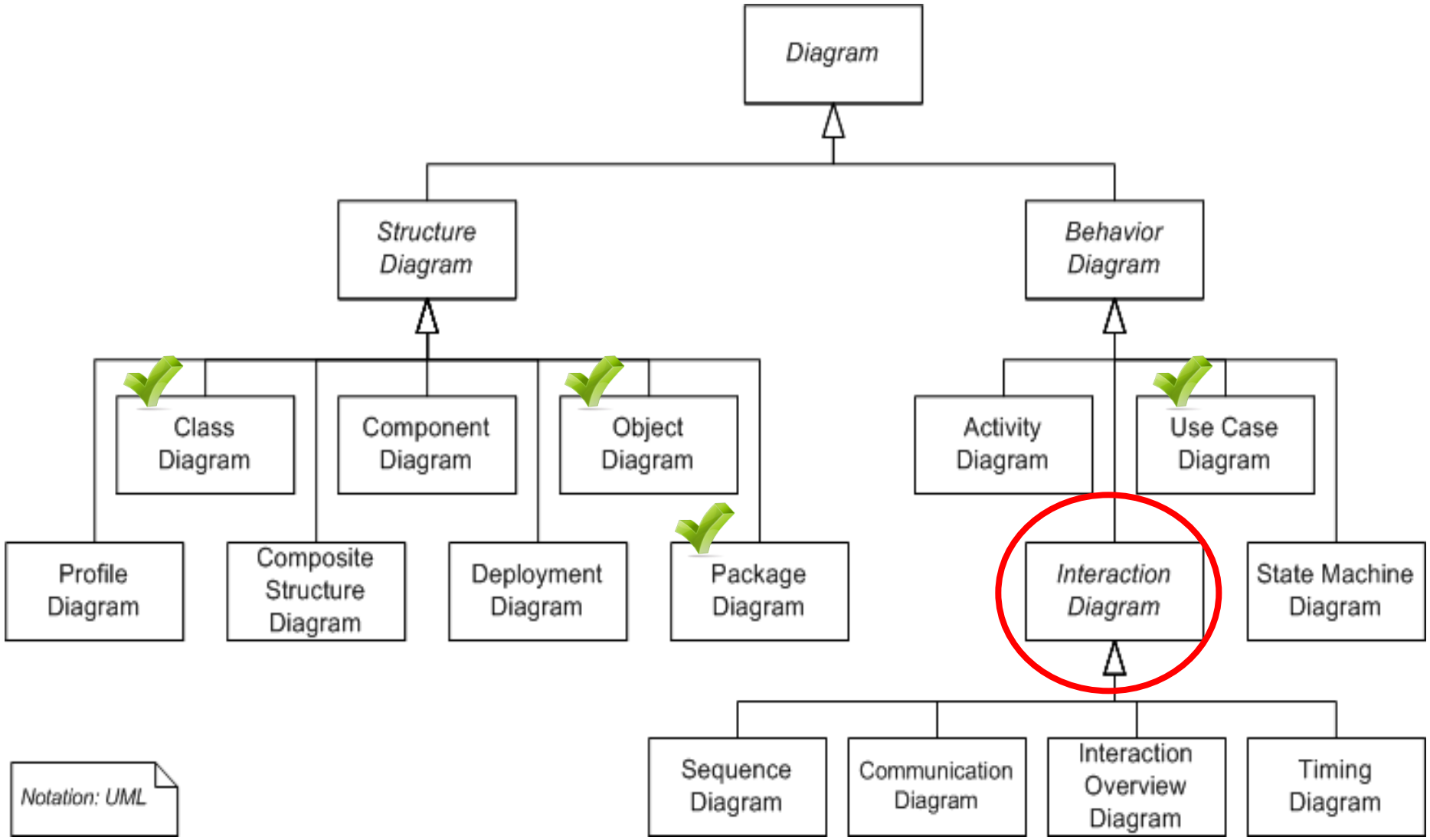
Sequence Diagrams

Week 5: System Modeling, Part 2

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

SYRACUSE UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

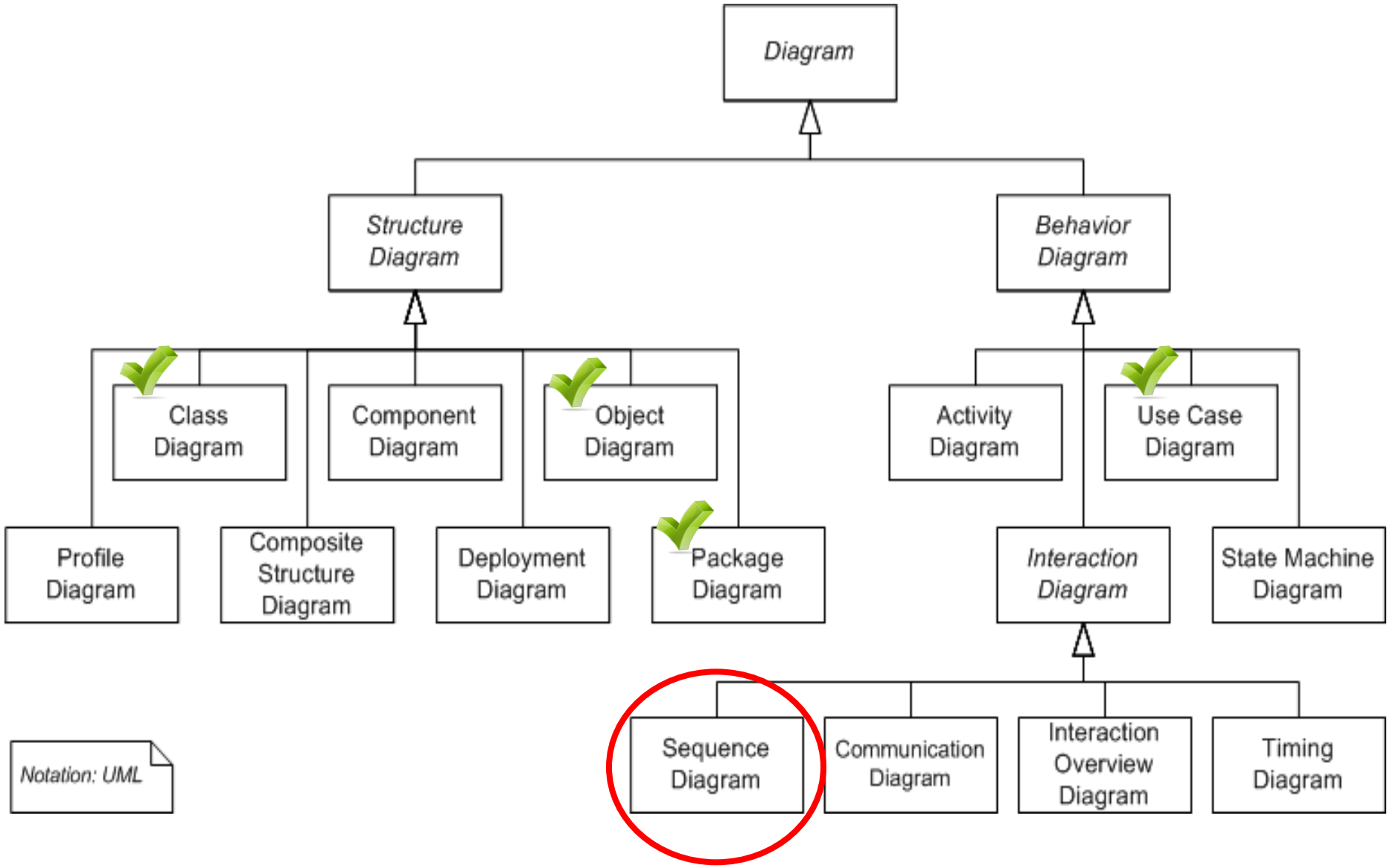
UML Diagrams



Interaction Diagrams

- A series of diagrams describing the **dynamic behavior** of an object-oriented system
  - **Dynamic behavior**: a set of messages exchanged among a set of objects within a context to accomplish a purpose
- Their purposes are to:
  - Model interactions between objects
  - Understand how a system (or a use case of a system) actually works
    - Often used to model the way a use case is realized through a sequence of messages between objects
  - Verify that a use case description can be supported by the existing classes
  - Identify responsibilities/operations, and assign them to classes

UML Diagrams



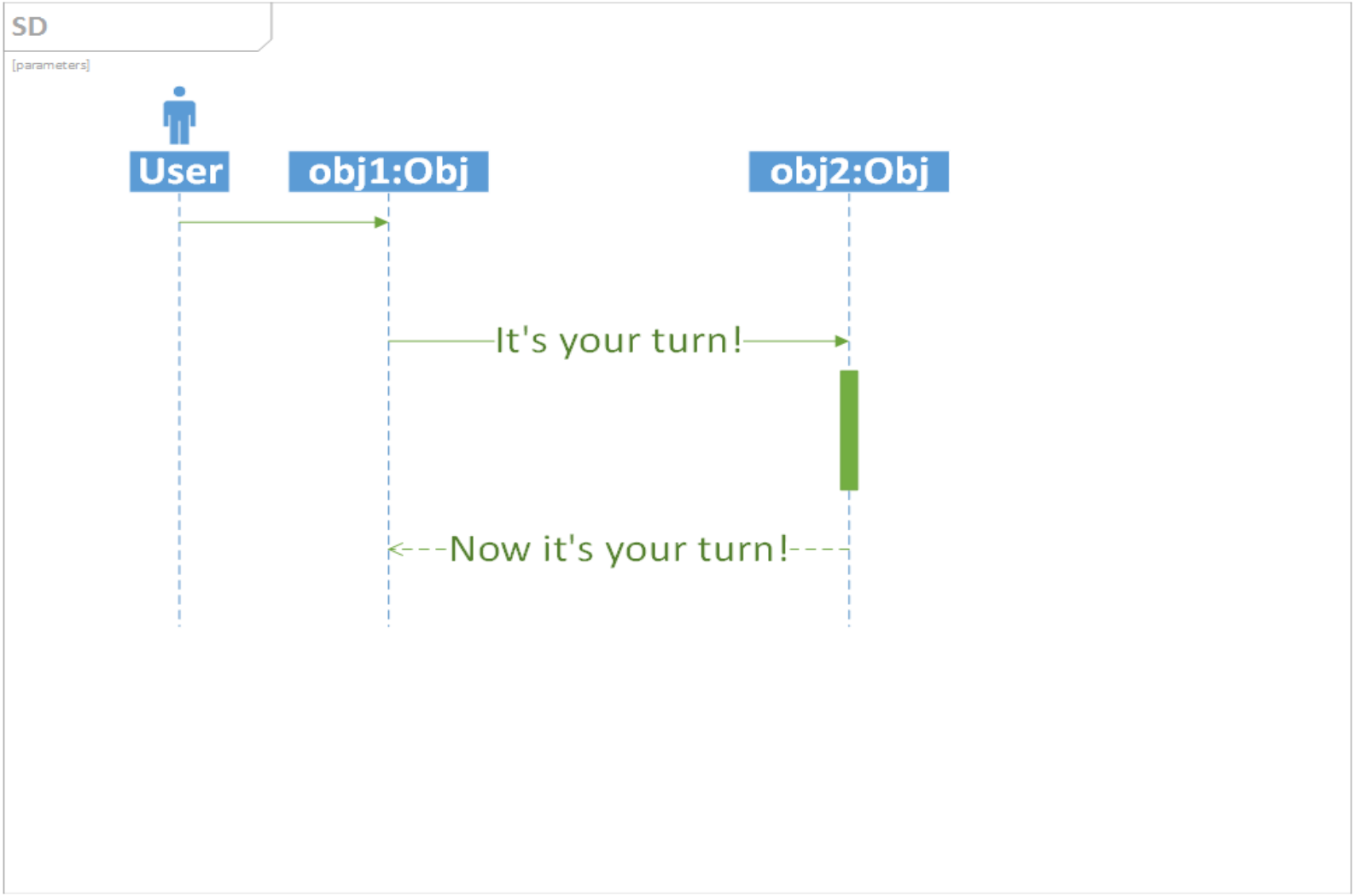
Sequence Diagrams

- A sequence diagram is an interaction diagram that emphasizes the time ordering of messages.

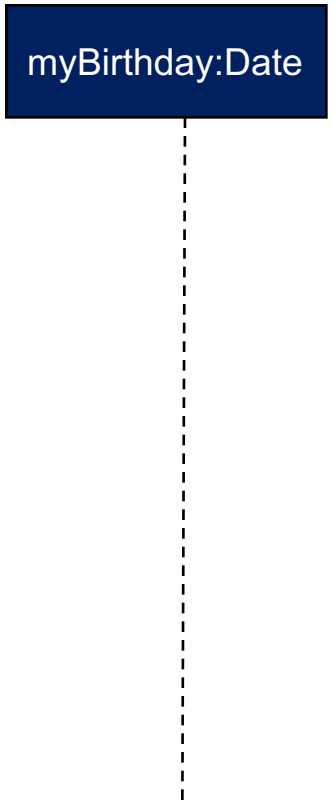
- It shows a set of roles, connectors among those roles, and messages sent and received by the instances (objects) playing the roles.

  —*The UML User Guide, 2nd edition*

A Very Simple Example



Sequence Diagrams

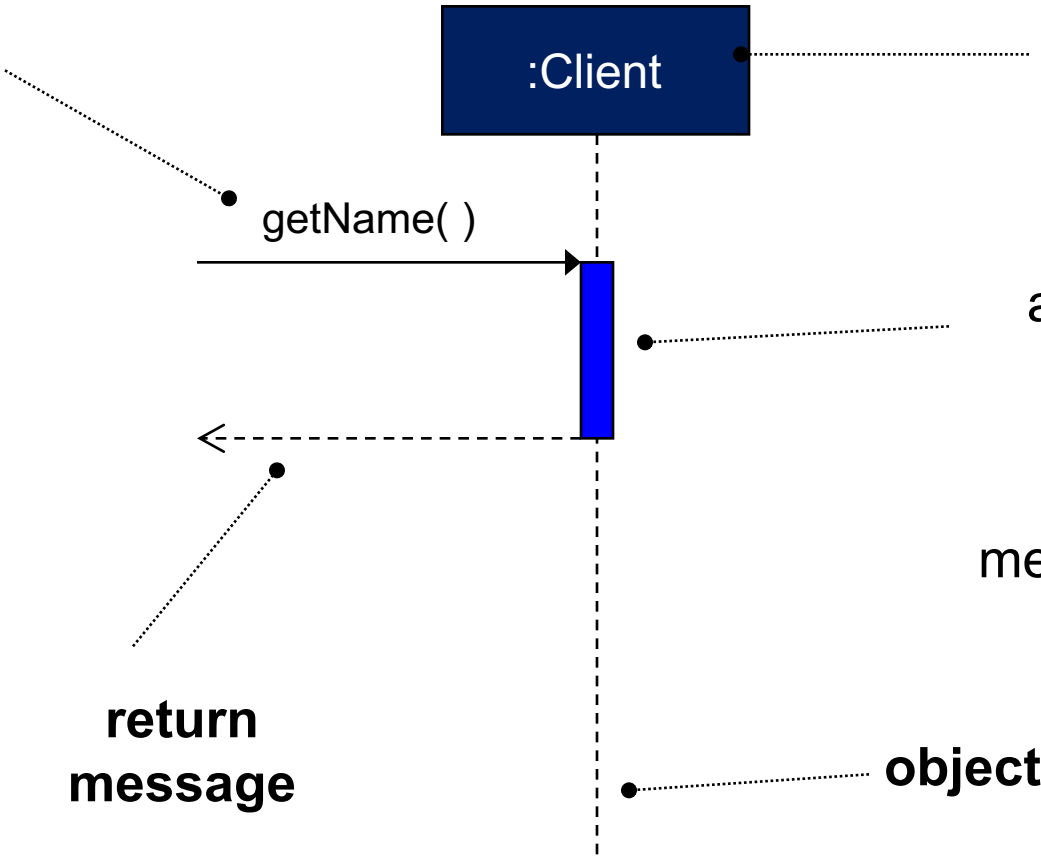


- An object in a sequence diagram is rendered as a box with a dashed line descending from it. The line is called the **object lifeline**, and it represents the existence of an object over a period of time.

Object naming:

- Syntax: [instanceName][:className]
- Name classes consistently with your class diagrams.
- Include instance names when objects are referred to in messages or when several objects of the same type exist in the diagram.

Sequence Diagrams



synchronous message synchronised with source by a return message

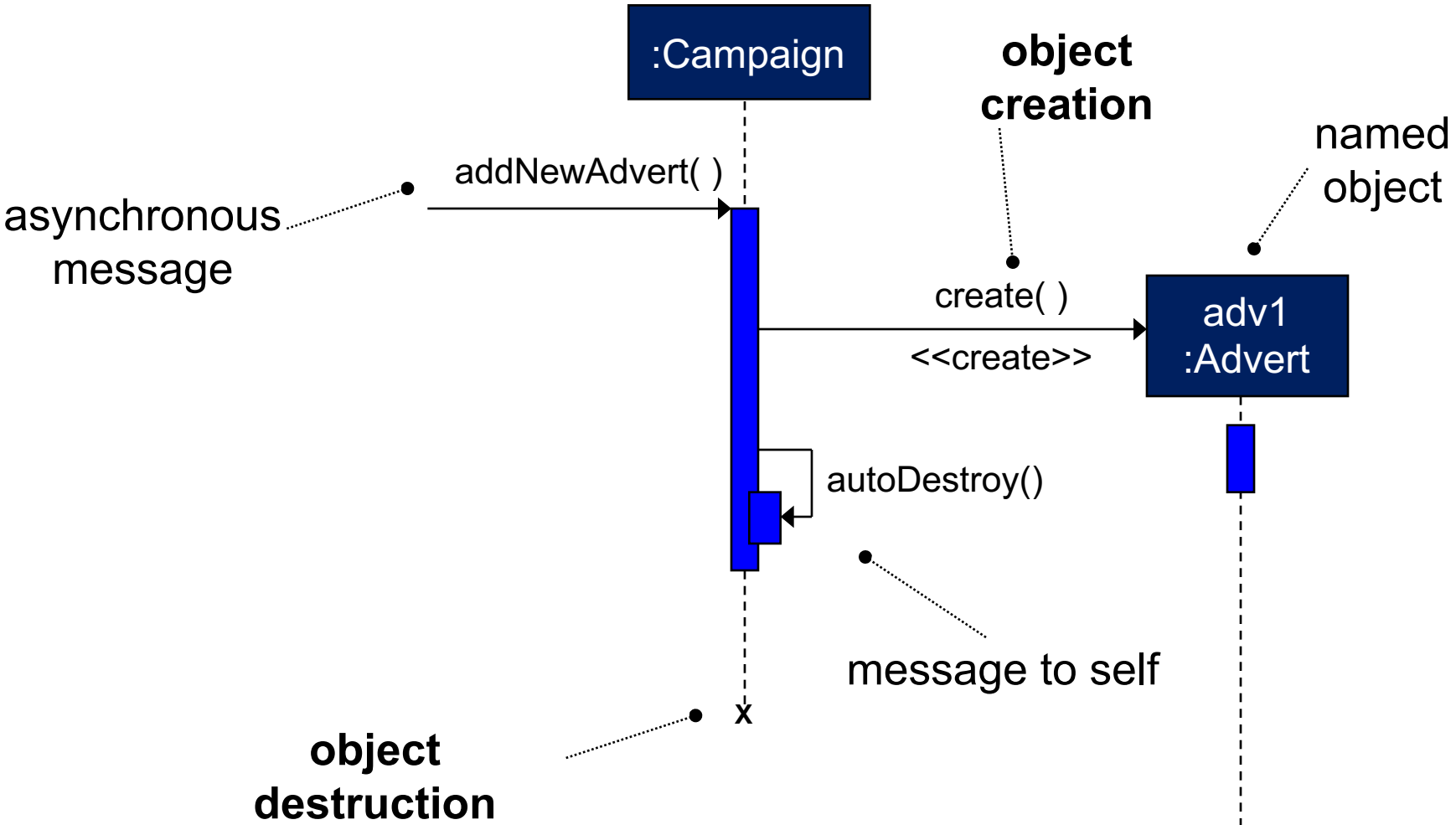
anonymous object not named

activation box (**focus of control**): when this message is active

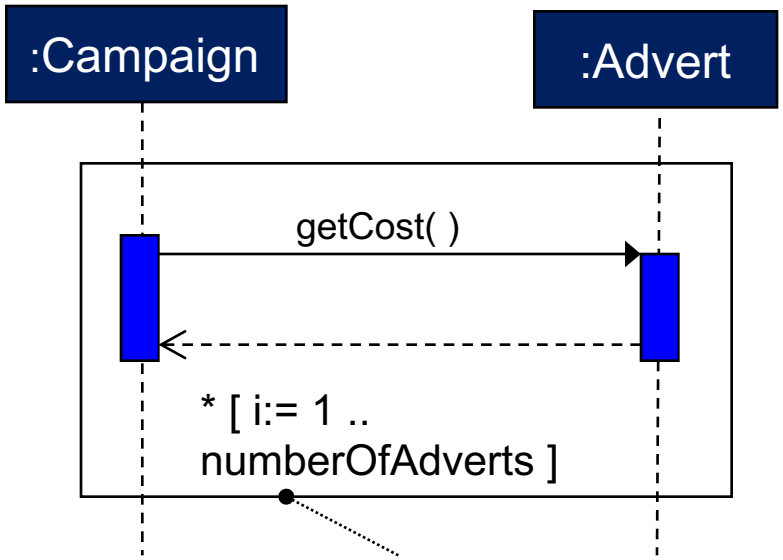
return message

object lifeline

Sequence Diagrams

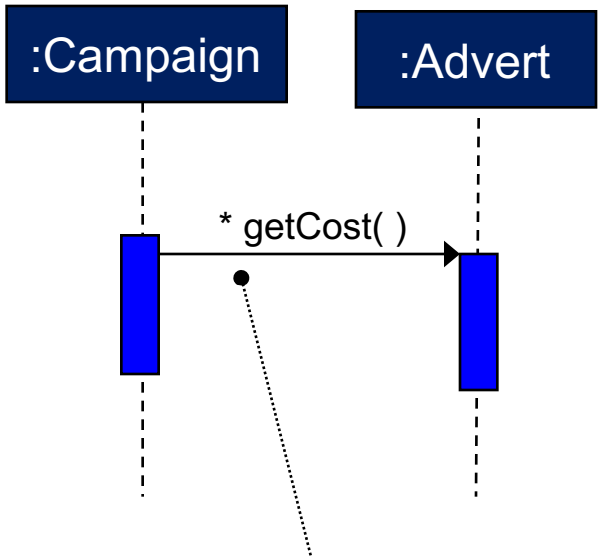


Sequence Diagrams



Repetition Notation 1

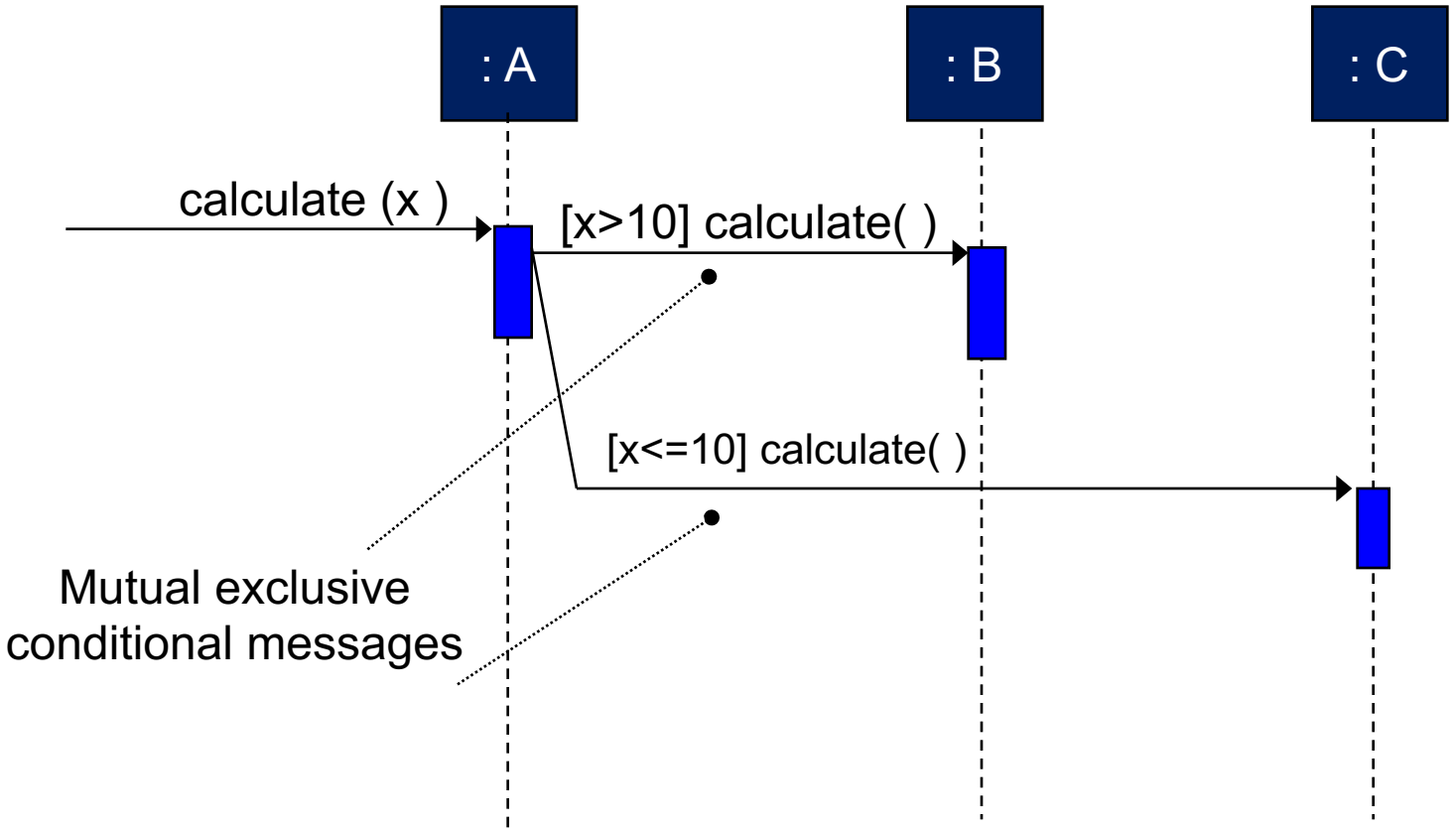
Usually used when the set of messages in the loop are more than one

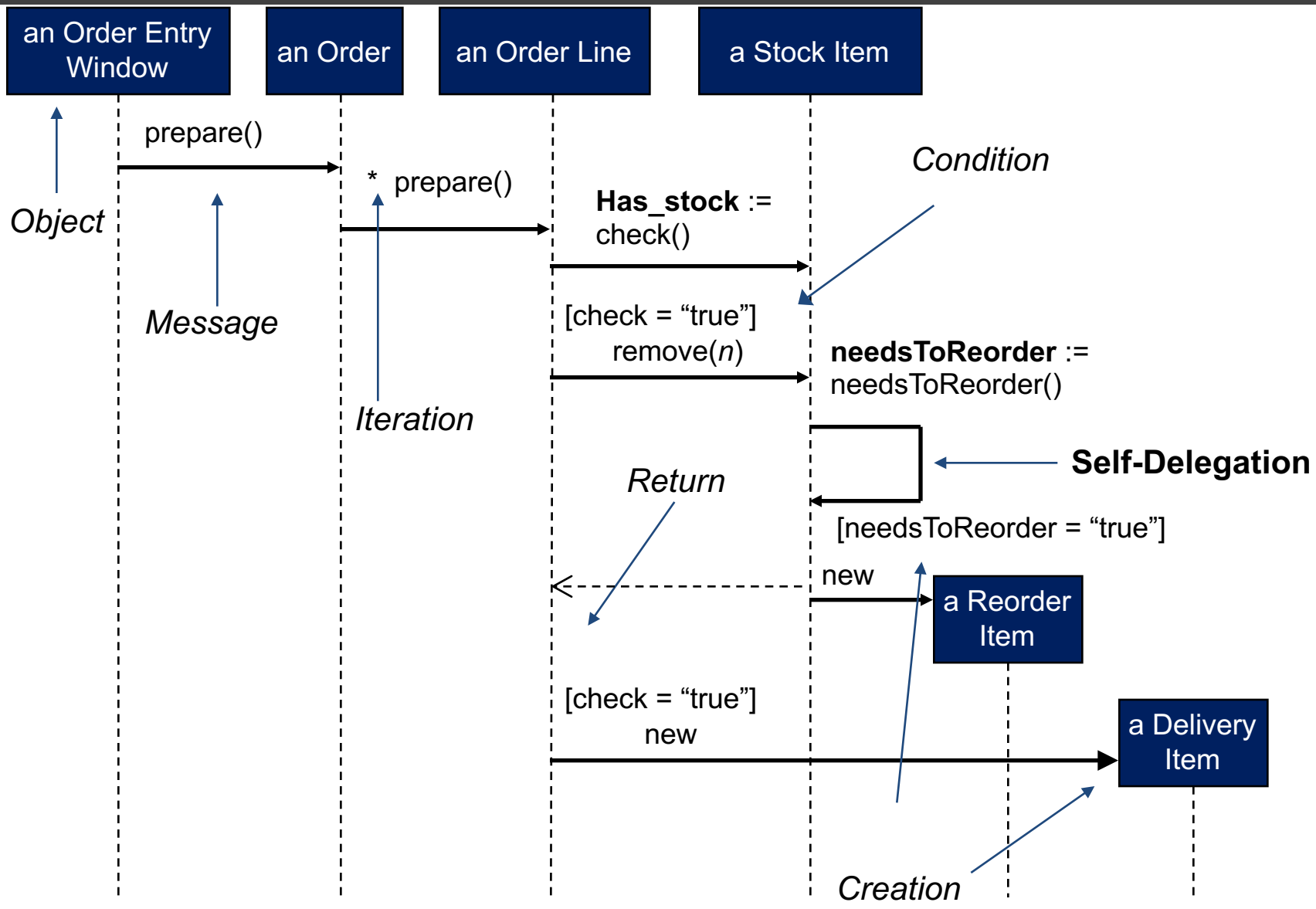


Repetition Notation 2

Used to indicate that the message is sent repetitively

Sequence Diagrams

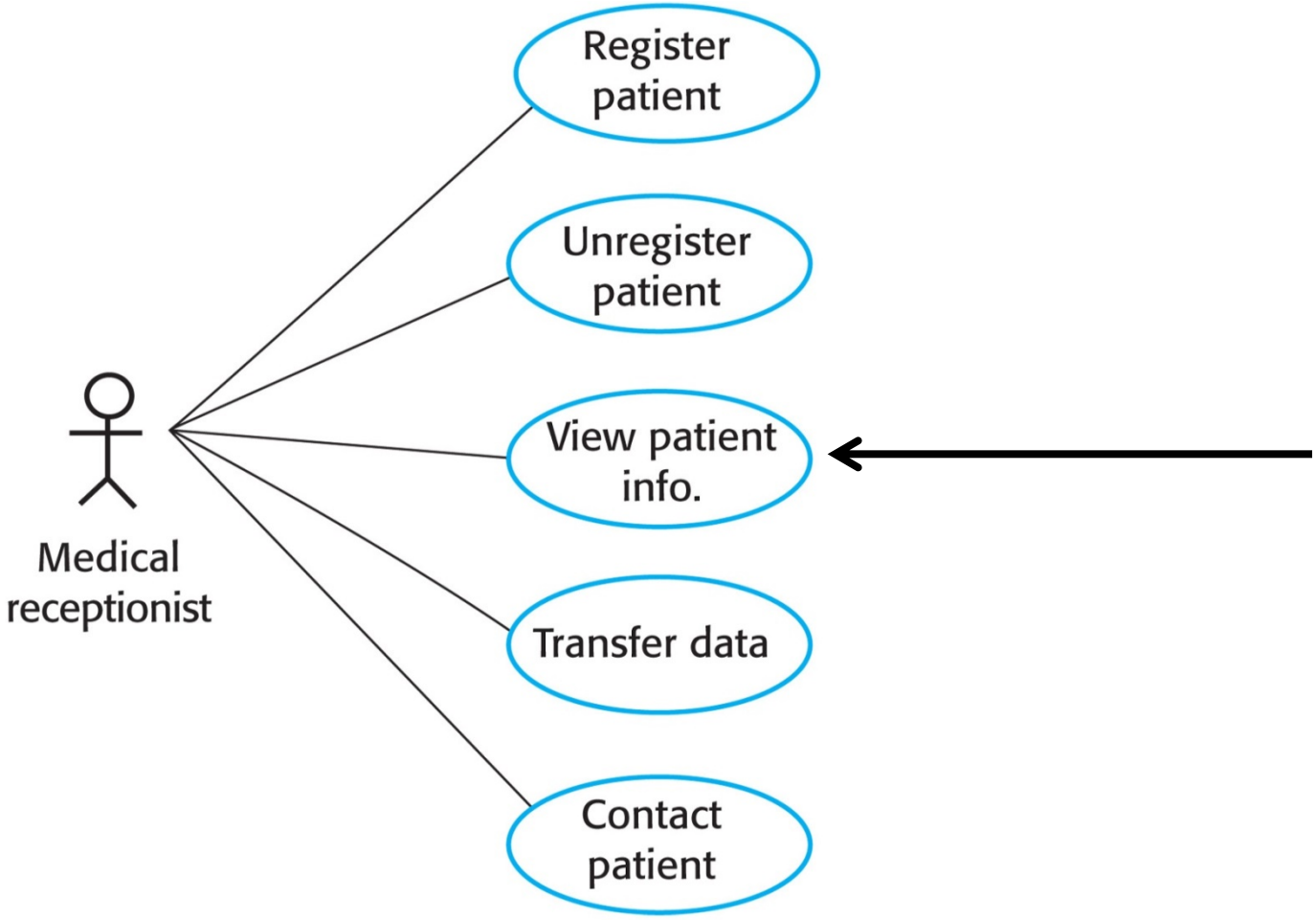



an Order Entry Window
an Order
an Order Line
a Stock Item
Object
prepare()
Message
* prepare()
Iteration
Has_stock :=
check()
Condition
[check = "true"]
remove(n)
needsToReorder :=
needsToReorder()
Self-Delegation
Return
[needsToReorder = "true"]
new
a Reorder Item
[check = "true"]
new
a Delivery Item
Creation

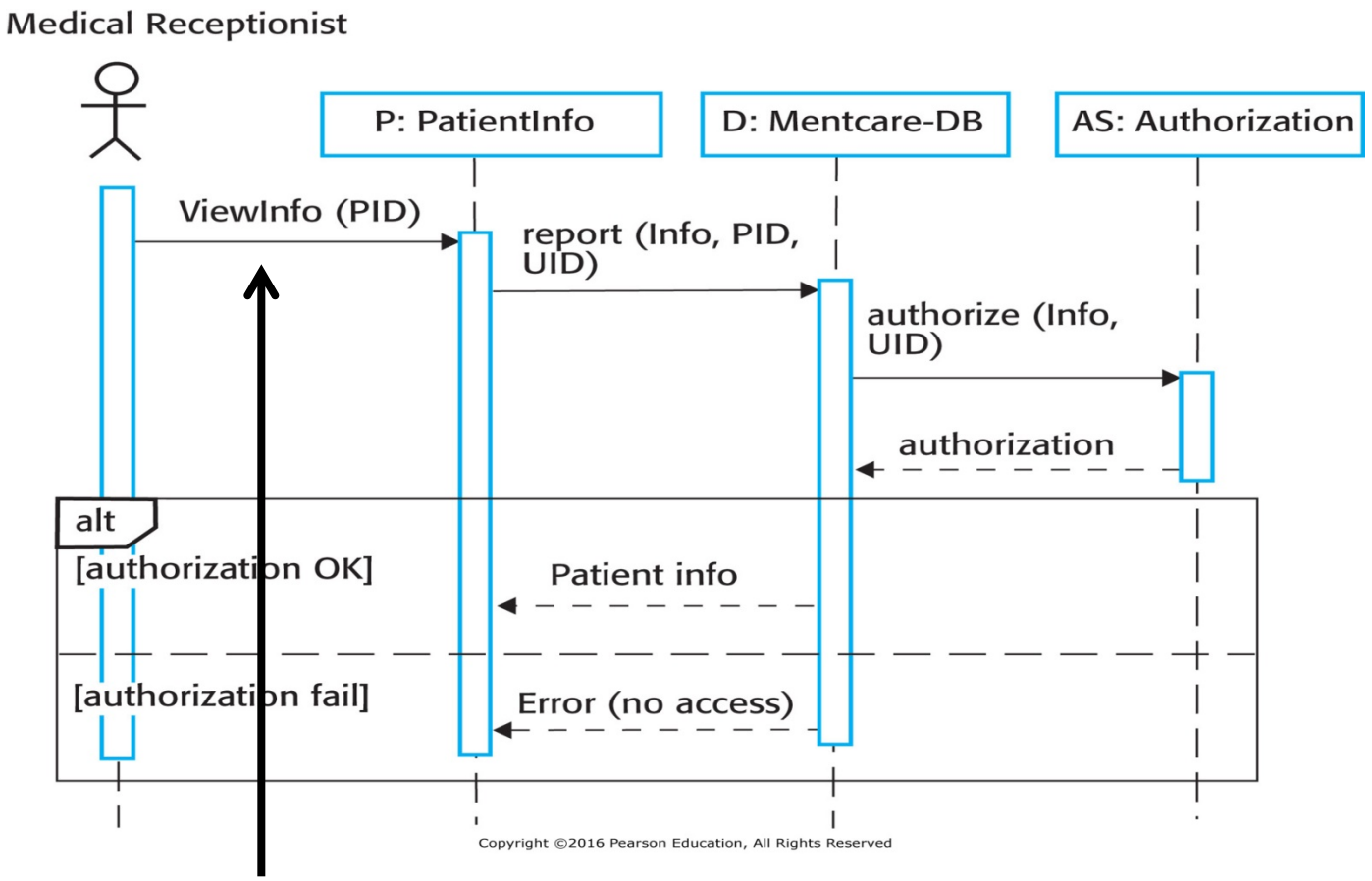
Sequence Diagrams for Use Cases

- A series of diagrams describing the dynamic behavior of an object-oriented system
  - Dynamic behavior: a set of messages exchanged among a set of objects within a context to accomplish a purpose
- Their purposes are to:
  - Model interactions between objects
  - **Understand how a system (or a use case of a system) actually works**
    - **Often used to model the way a use case is realized through a sequence of messages between objects**
  - Verify that a use case description can be supported by the existing classes
  - Identify responsibilities/operations, and assign them to classes

Use Case: View Patient Information

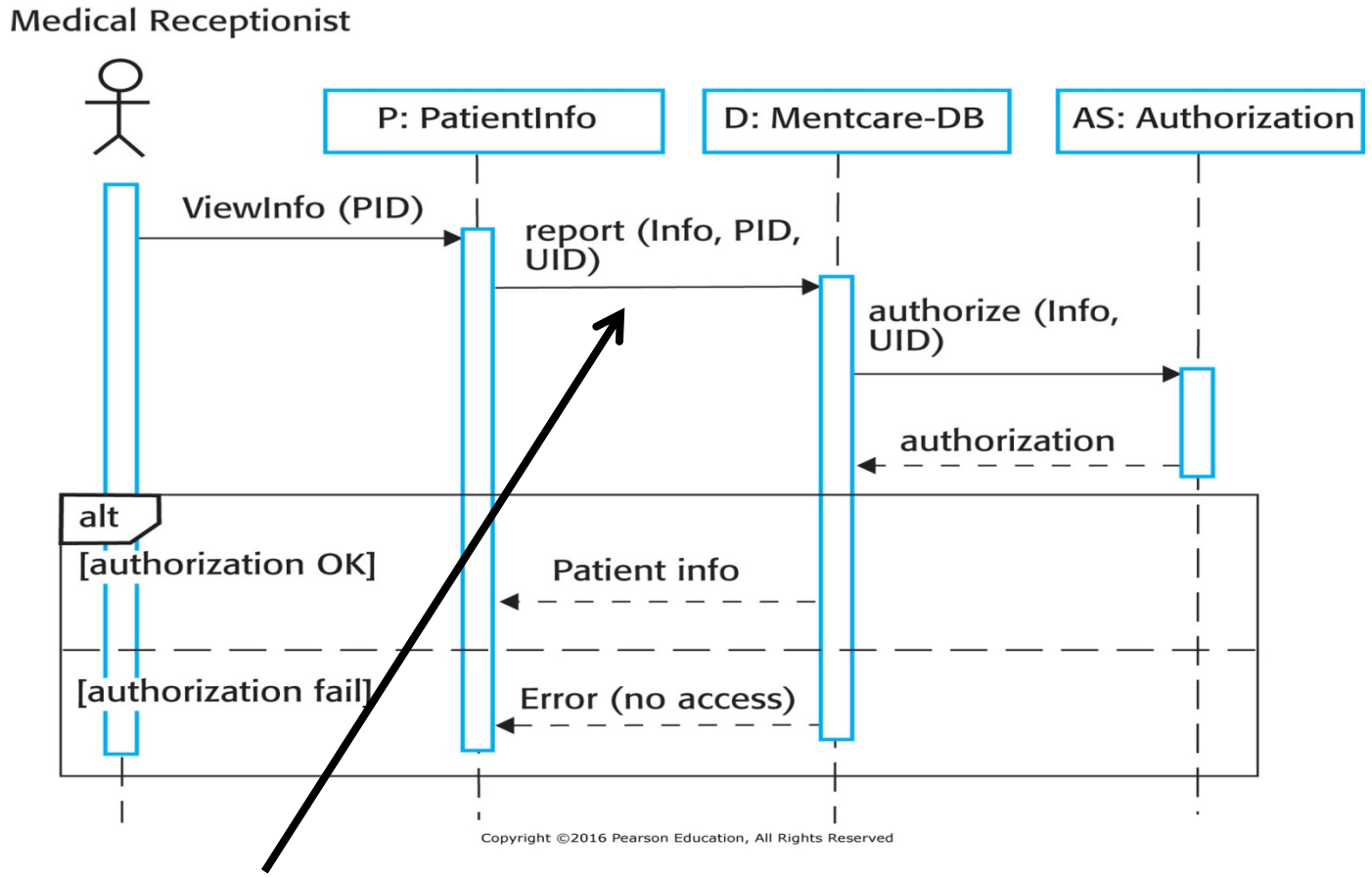


SD: View Patient Information



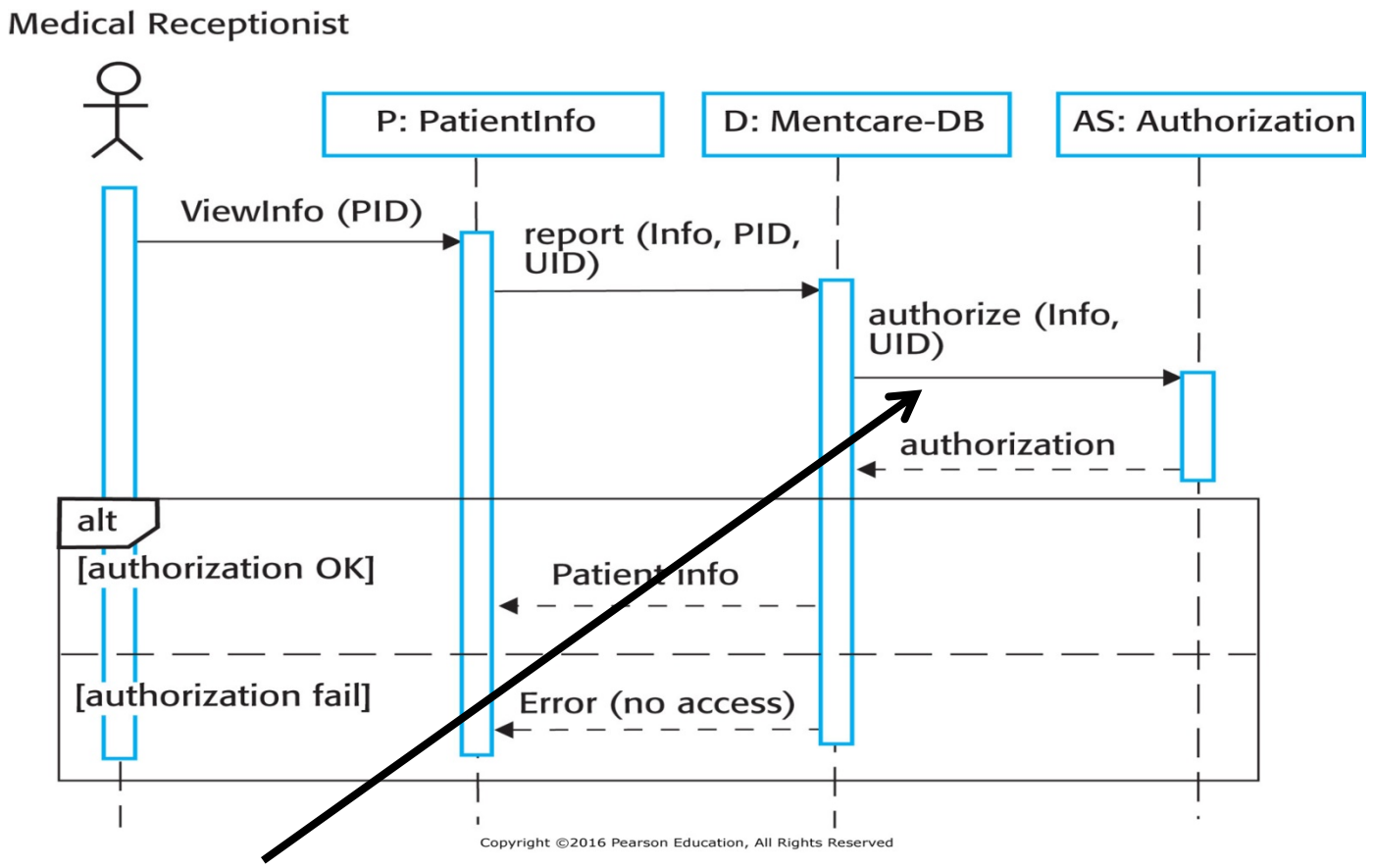
The medical receptionist triggers the ViewInfo method in an instance P of the PatientInfo class, supplying the patient's ID, PID. P is a user interface object, which is displayed as a form showing patient information.

SD: View Patient Information



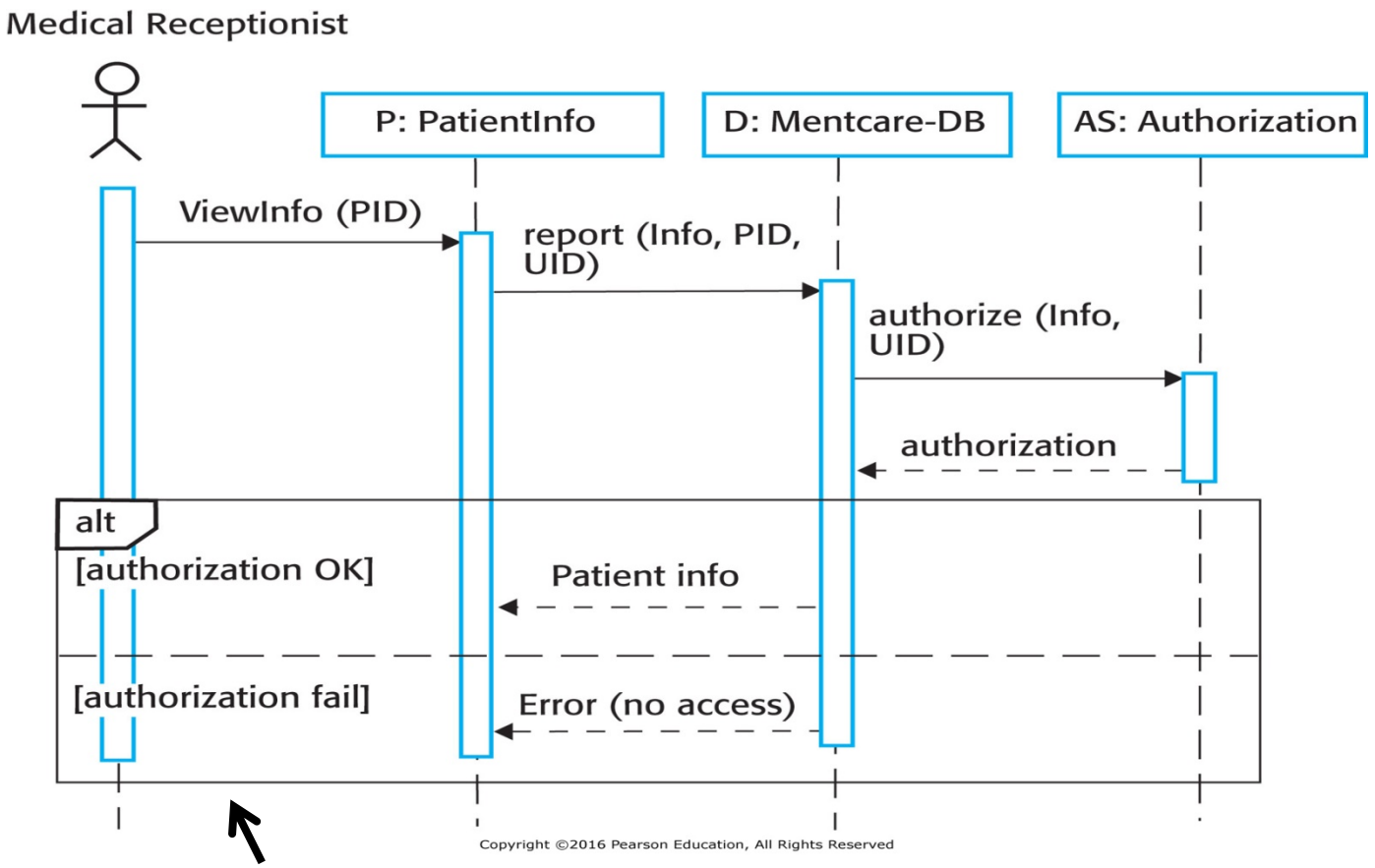
The instance P calls the database to return the information required, supplying the receptionist's identifier to allow security checking (at this stage, we do not care where this UID comes from).

SD: View Patient Information



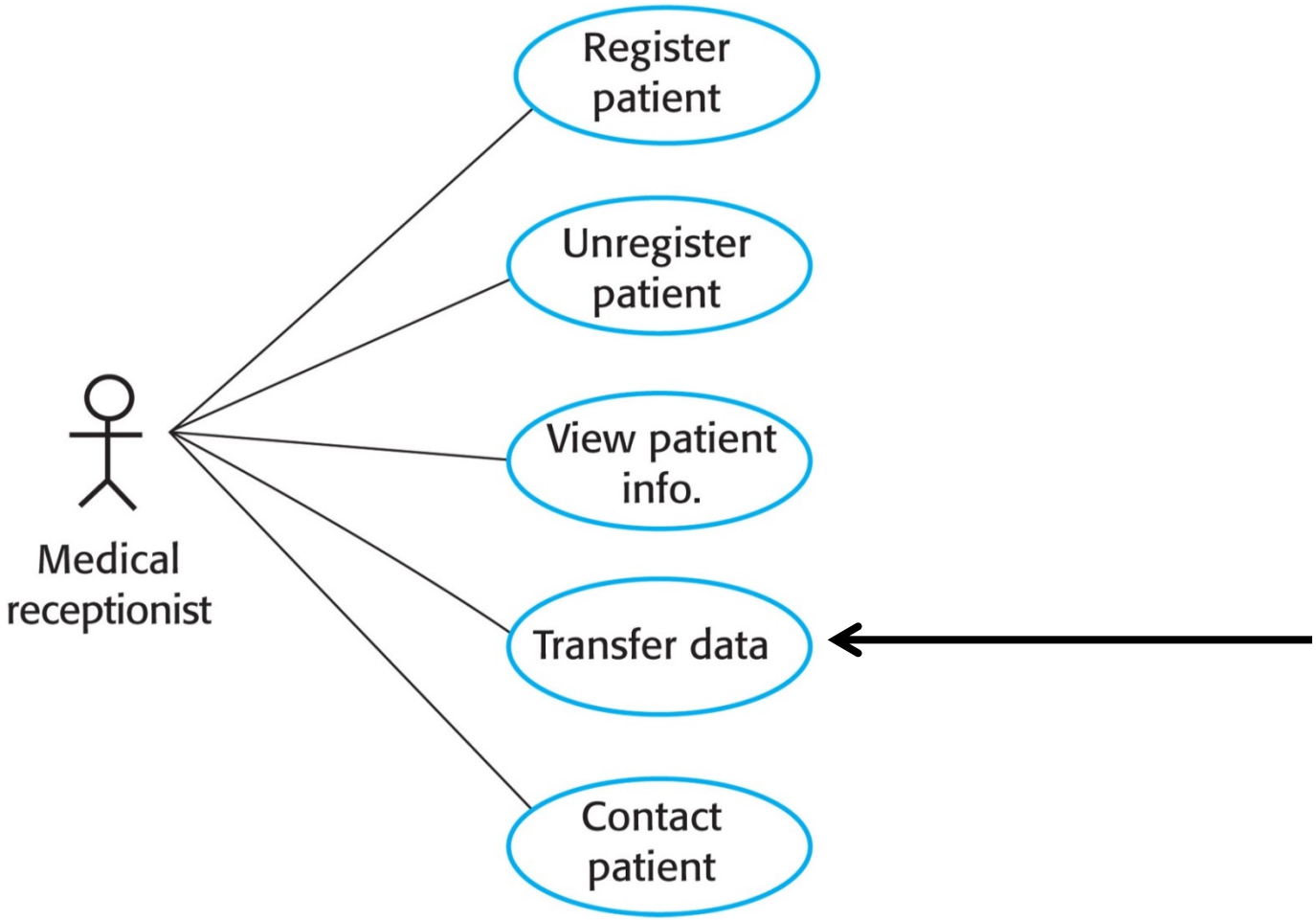
The database checks with an authorization system that the user is authorized for this action.

SD: View Patient Information

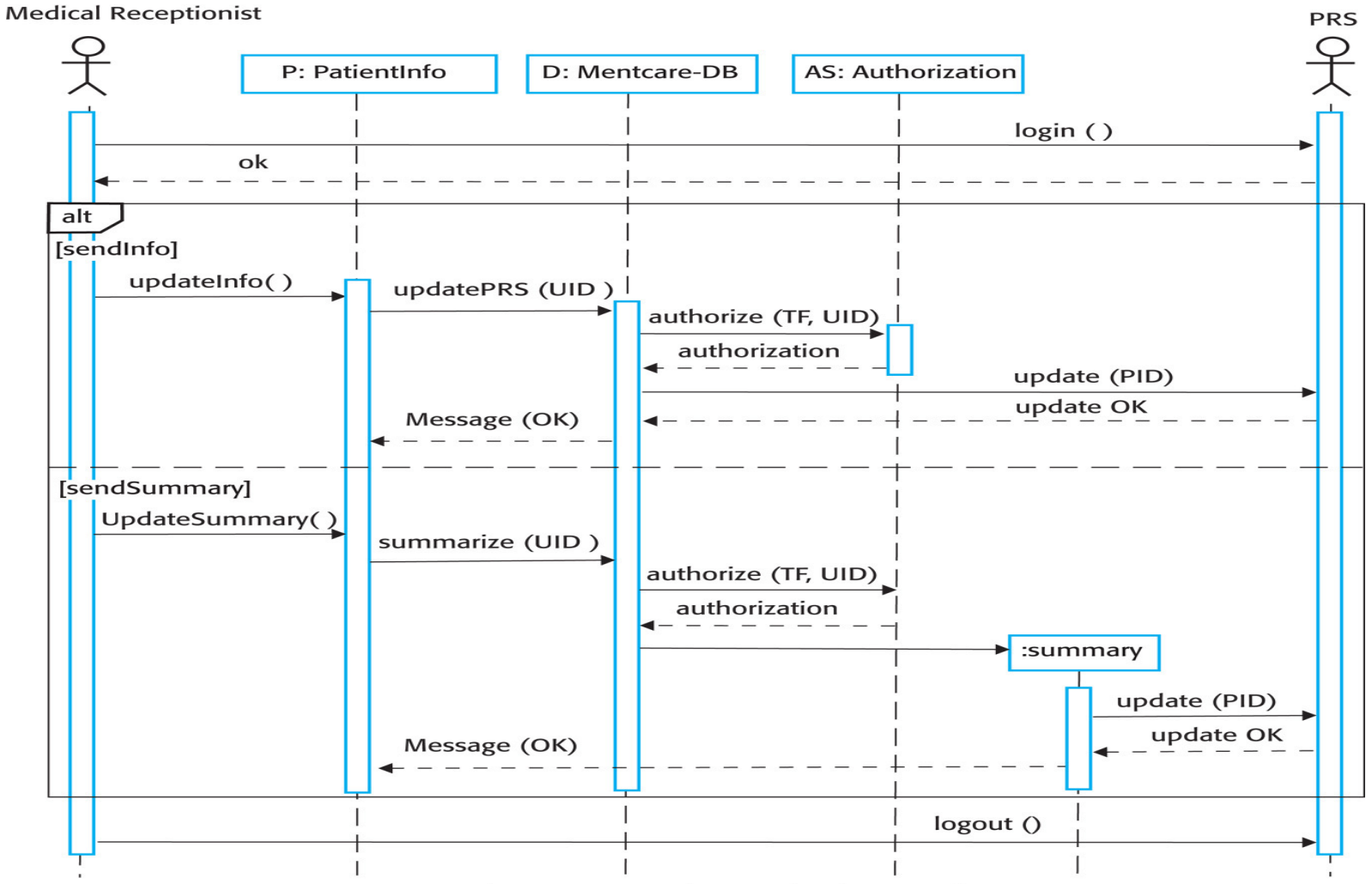


If authorized, the patient information is returned, and a form on the user's screen is filled in. If authorization fails, then an error message is returned.

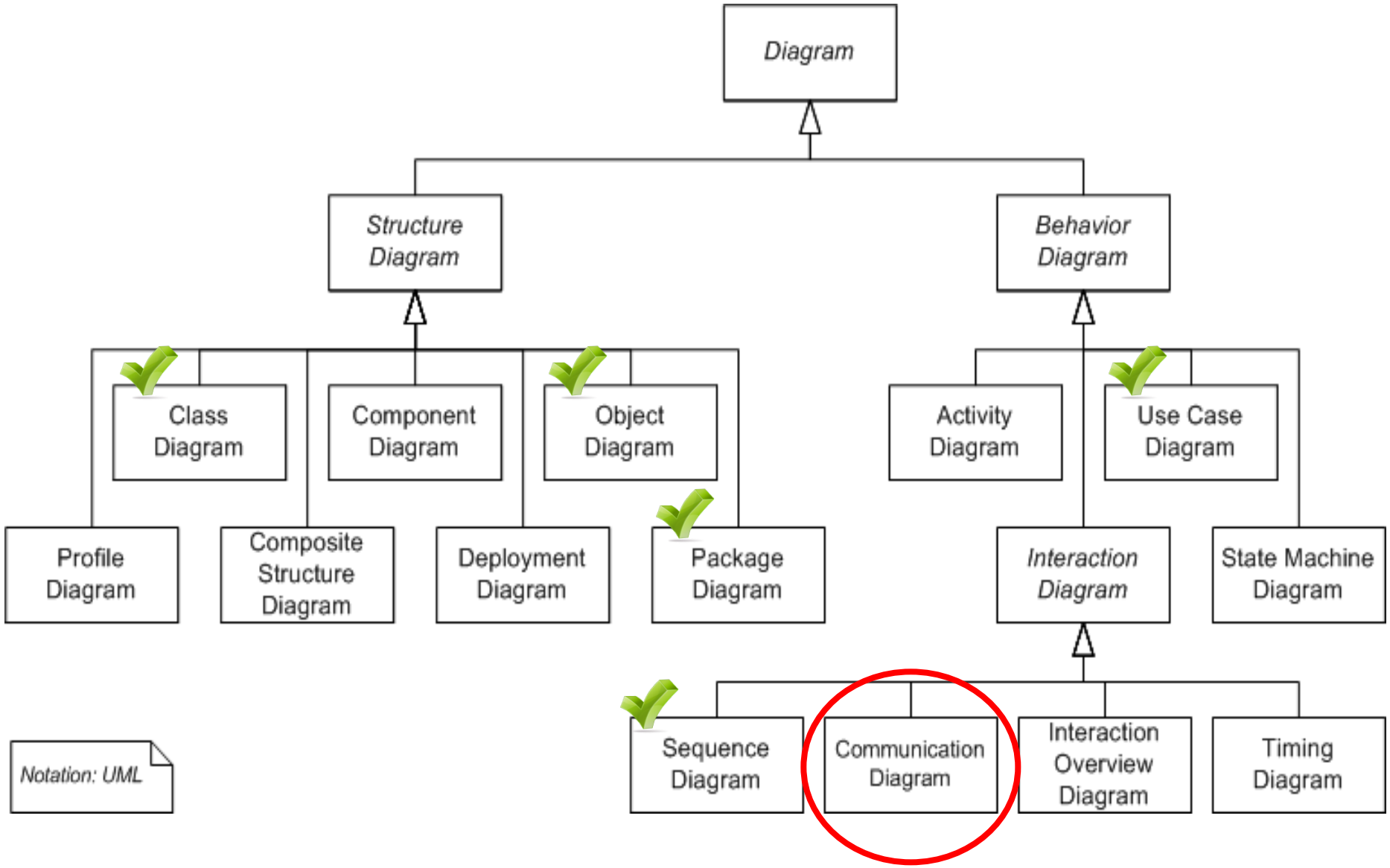
Use Case: Transfer Data



SD: Transfer Data

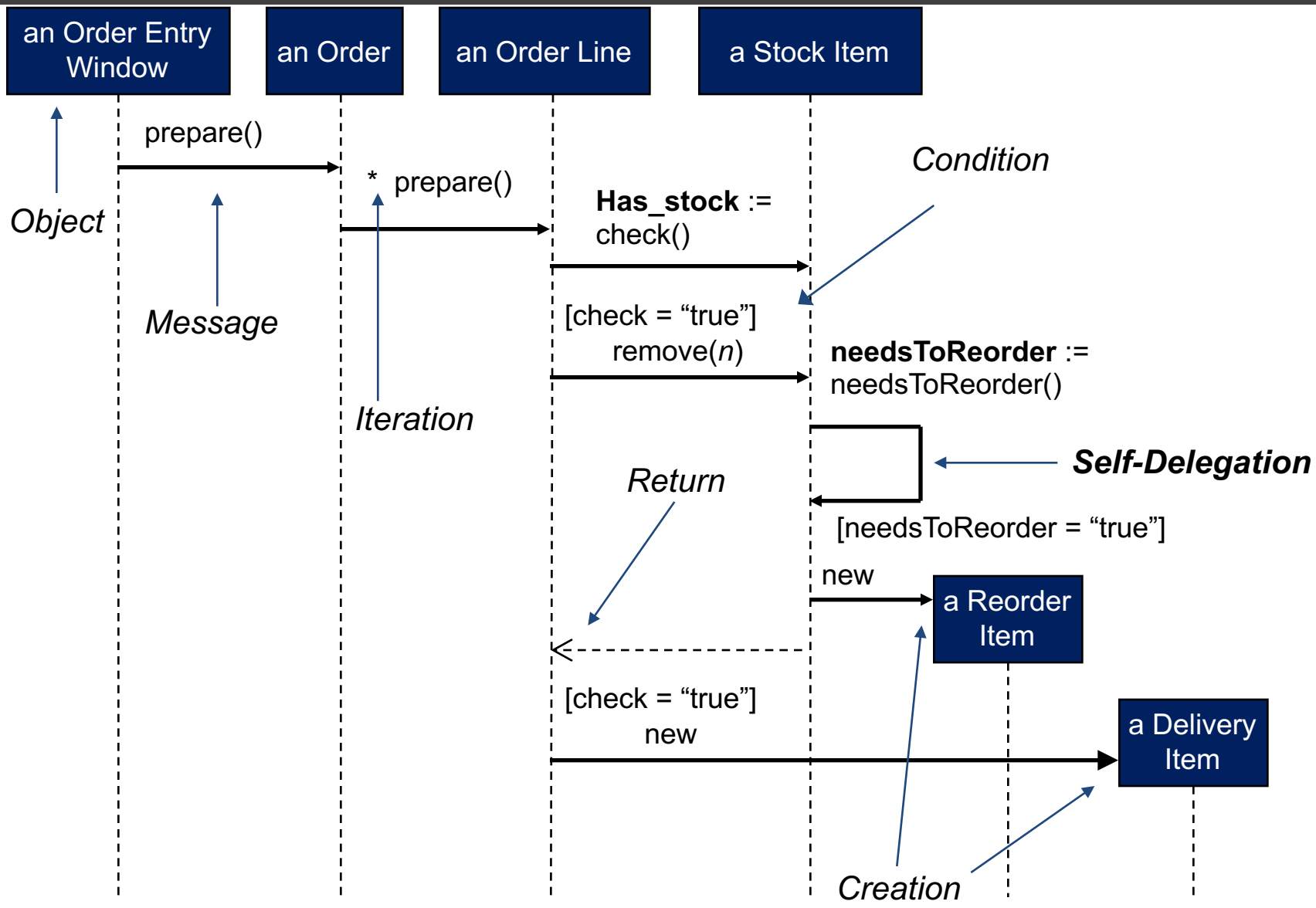


UML Diagrams

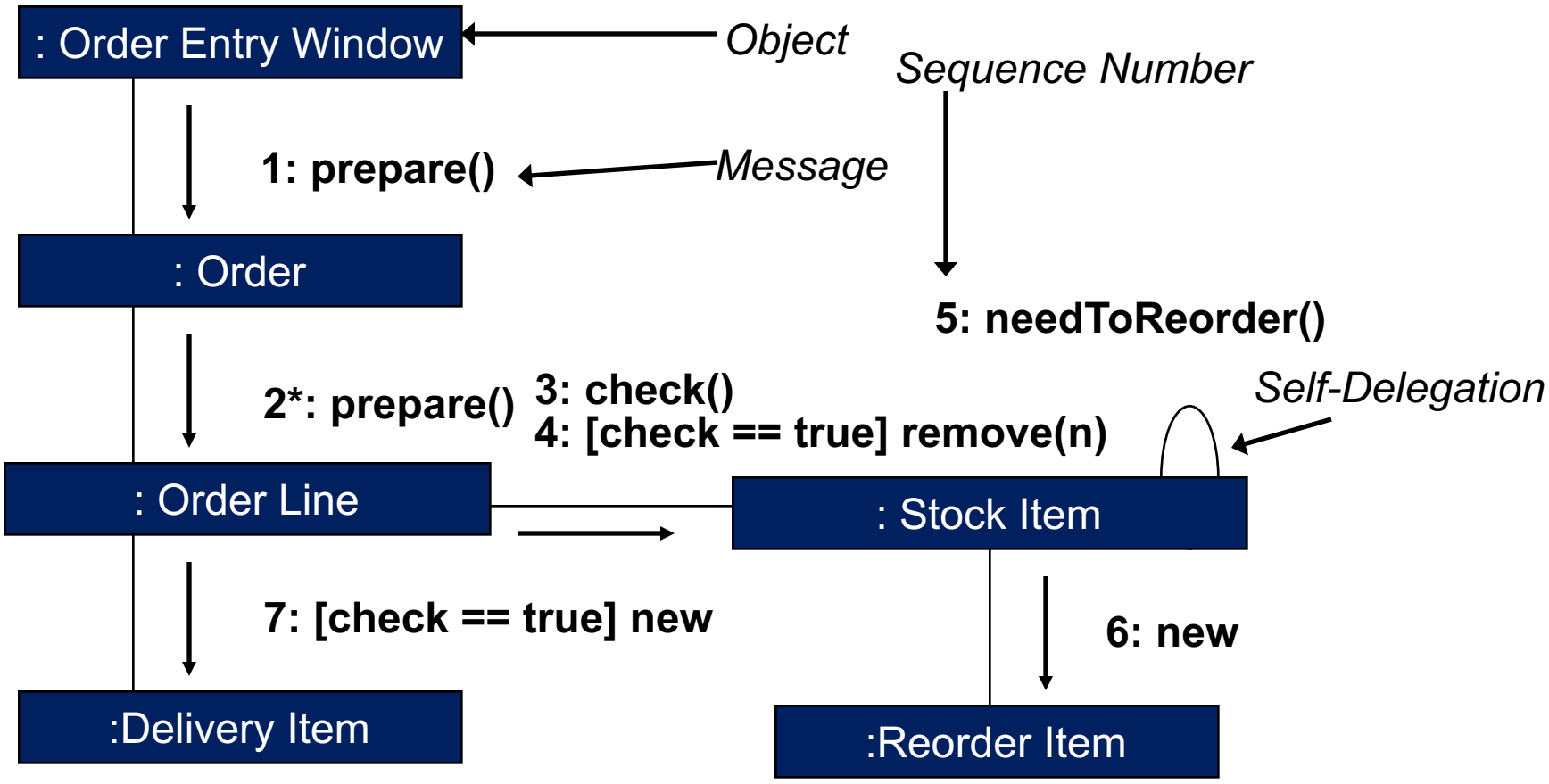


Communication Diagrams

- Also known as collaboration diagrams.
- Unlike a sequence diagram, you don't have to show the lifeline of an object explicitly in a collaboration diagram.
  - The sequence of events are indicated by sequence numbers preceding messages.
- Like a sequence diagram, object identifiers are of the form *objectName : className,* and either the objectName or the className can be omitted, and the placement of the colon indicates either an objectName:, or a :className.
- Both a communication diagram and a sequence diagram derive from the same information, so you can take a diagram in one form and convert it into the other. They are semantically equivalent.


an Order Entry Window
an Order
an Order Line
a Stock Item
Object
prepare()
Message
* prepare()
Iteration
Has_stock :=
check()
Condition
[check = "true"]
remove(n)
needsToReorder :=
needsToReorder()
Self-Delegation
Return
[needsToReorder = "true"]
new
a Reorder Item
[check = "true"]
new
a Delivery Item
Creation

Communication Diagrams



SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

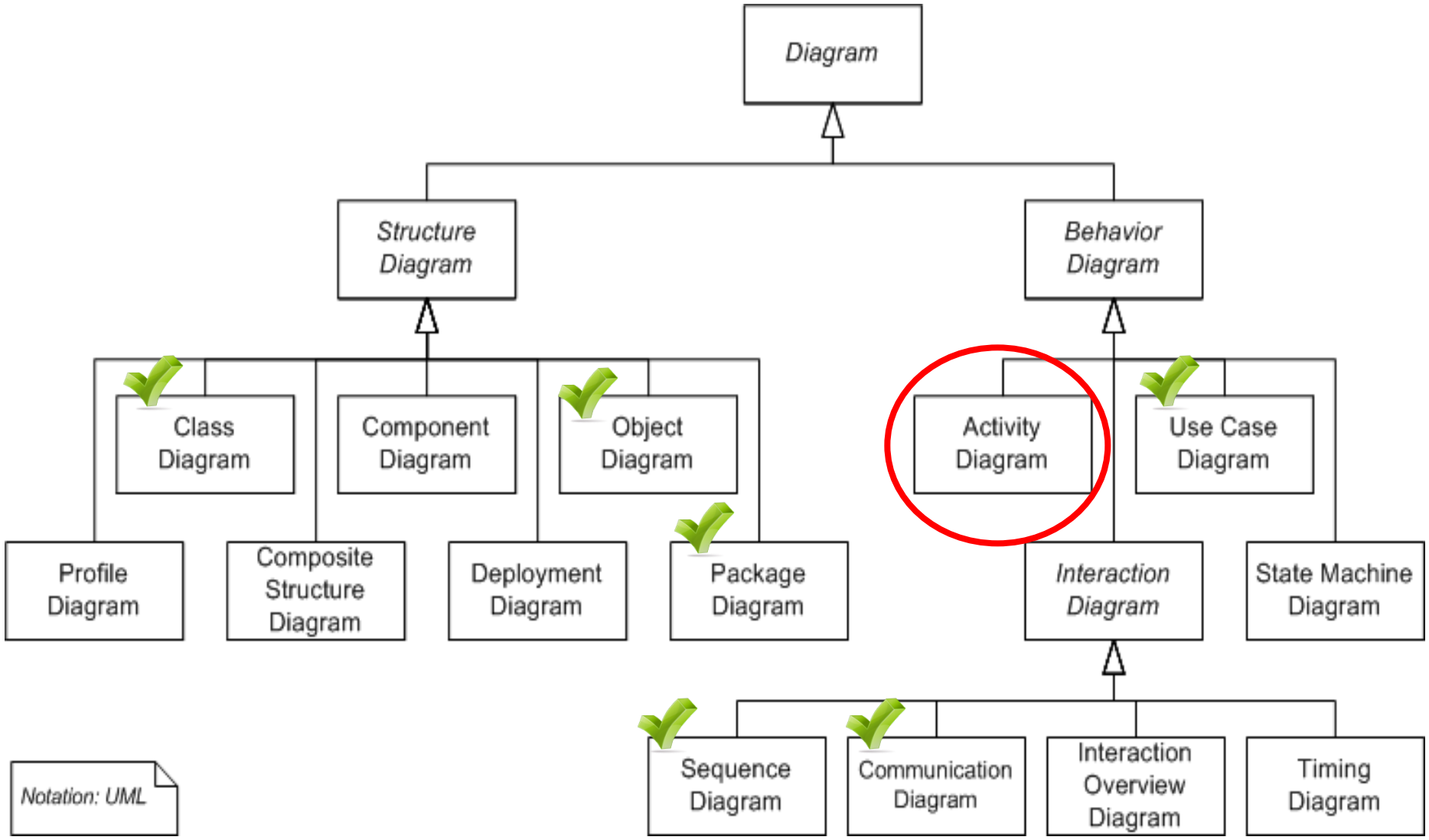
Activity Diagrams

Week 5: System Modeling, Part 2

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

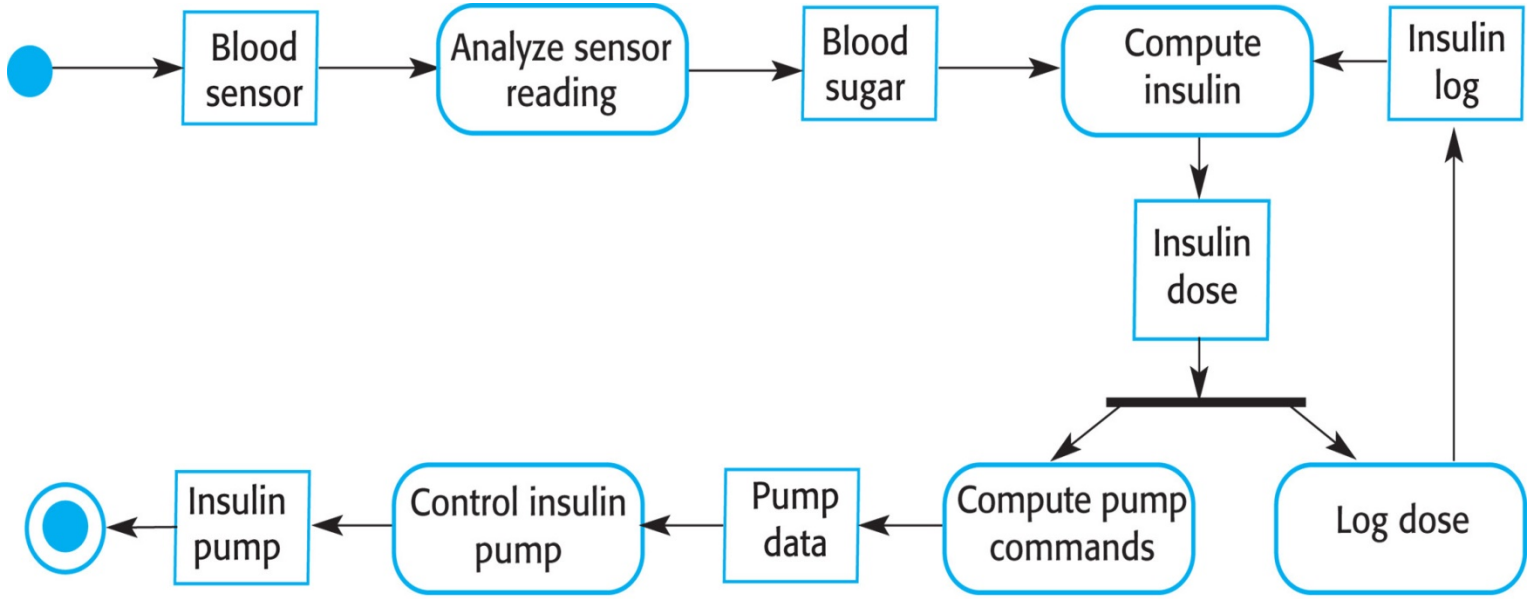
SYRACUSE UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

UML Diagrams



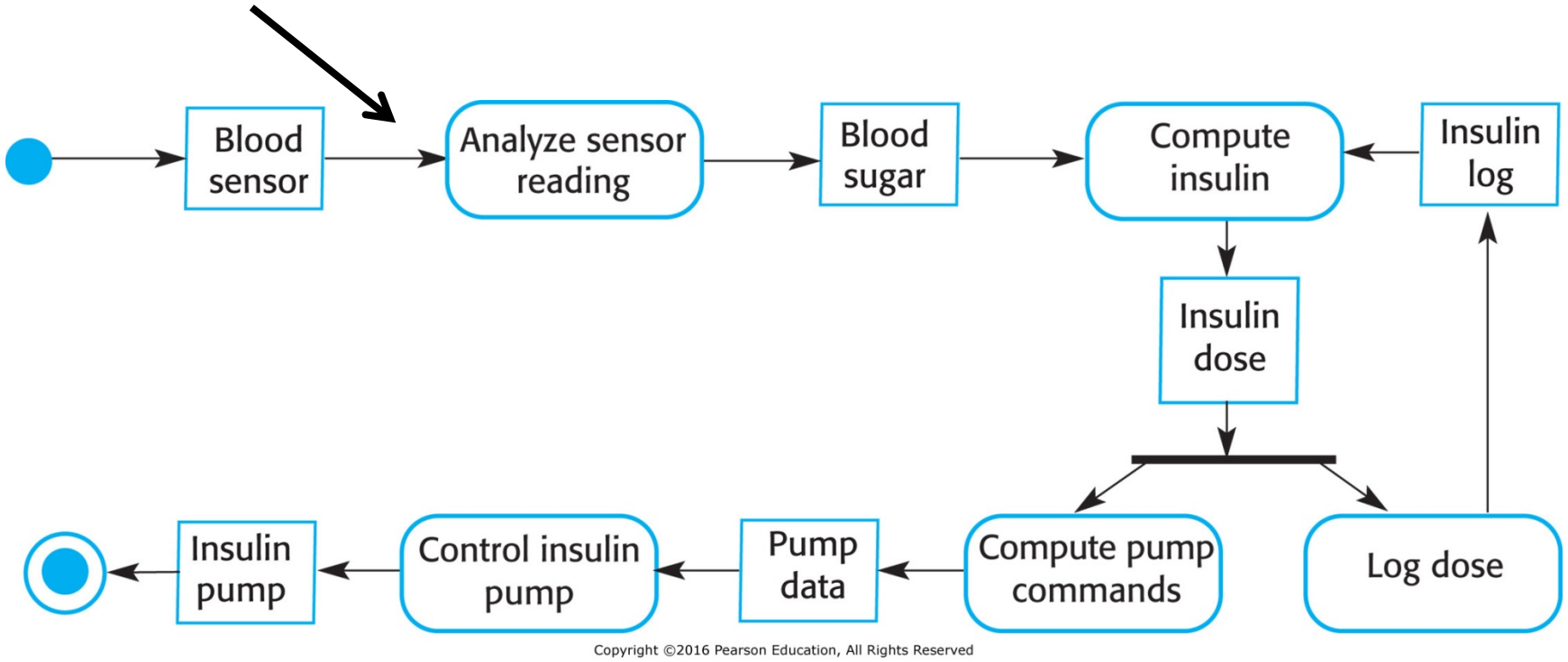
Activity Diagrams

- Useful to specify or model system behavior (or actions)
- Based on data flow models—a graphical representation (with a directed graph) of how data move around an information system



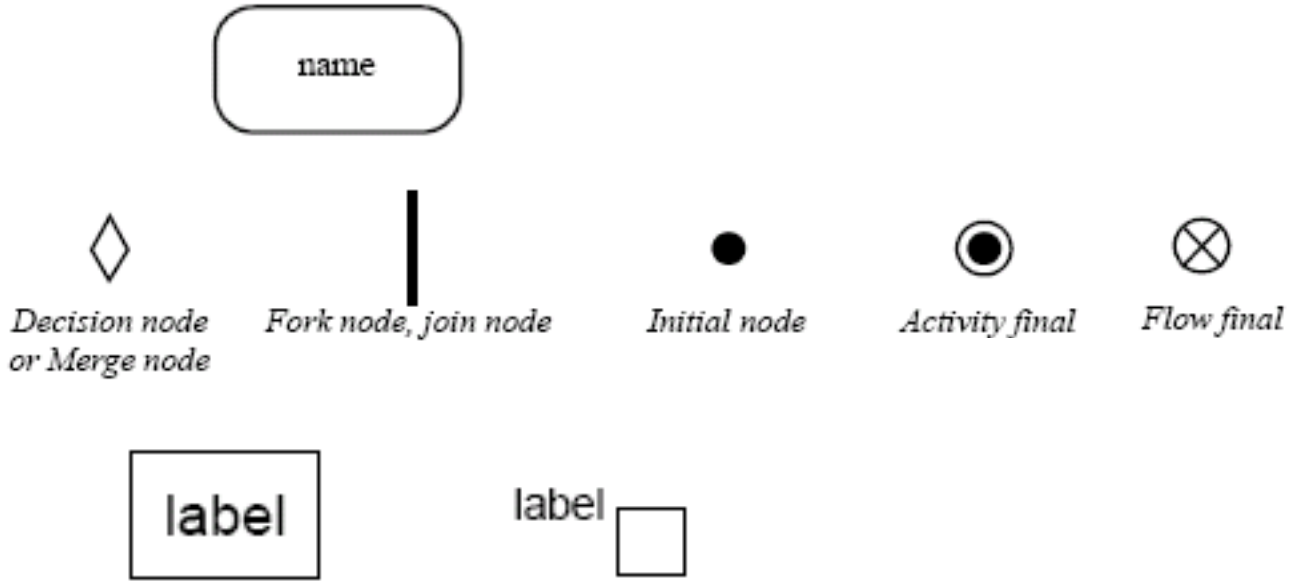
Flows

- **Flow**: permits the interaction between two nodes of the activity diagram (represented by edges in activity diagram)

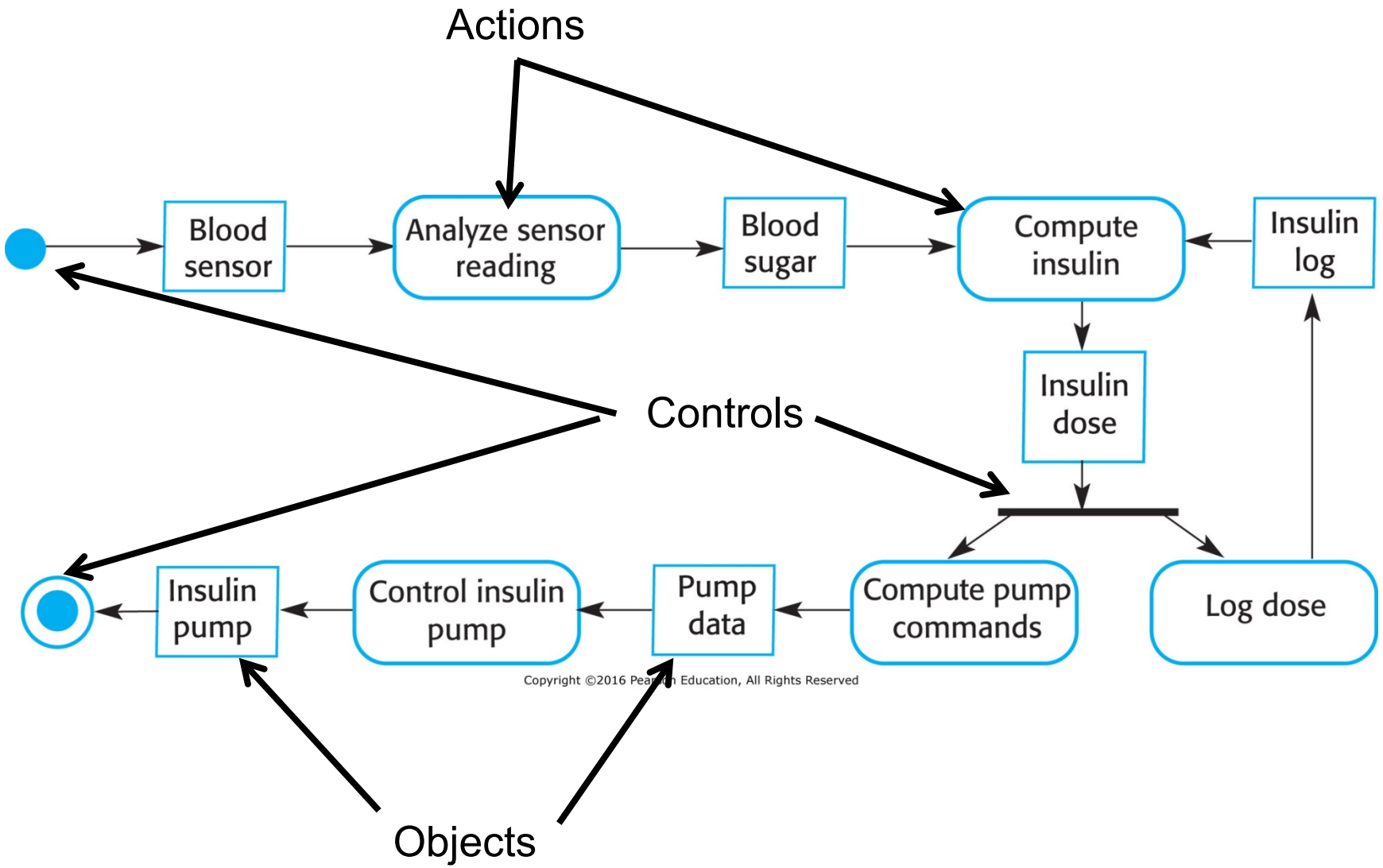


Activity Nodes

- Activity nodes are the fundamental units of executable functionality.
- Three type of activity nodes:
  - **Action nodes**: executable activity nodes; the execution of an action represents some transformations or processes.
  - **Control nodes**: coordinate flows in an activity diagram between other nodes.
  - **Object nodes**: indicate an instance of a particular object; may be available at a particular point in the activity.

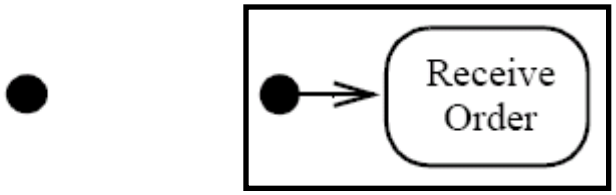


Activity Diagrams

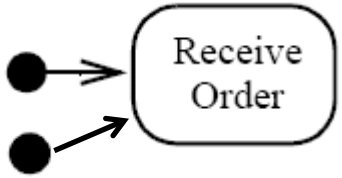


Control Nodes: Initial Nodes

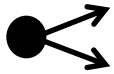
- In an activity the flow starts in initial nodes, which return the control immediately along their outgoing edges.



- If there are more than one initial node, control is placed in each initial node when the activity is started, initiating multiple flows.



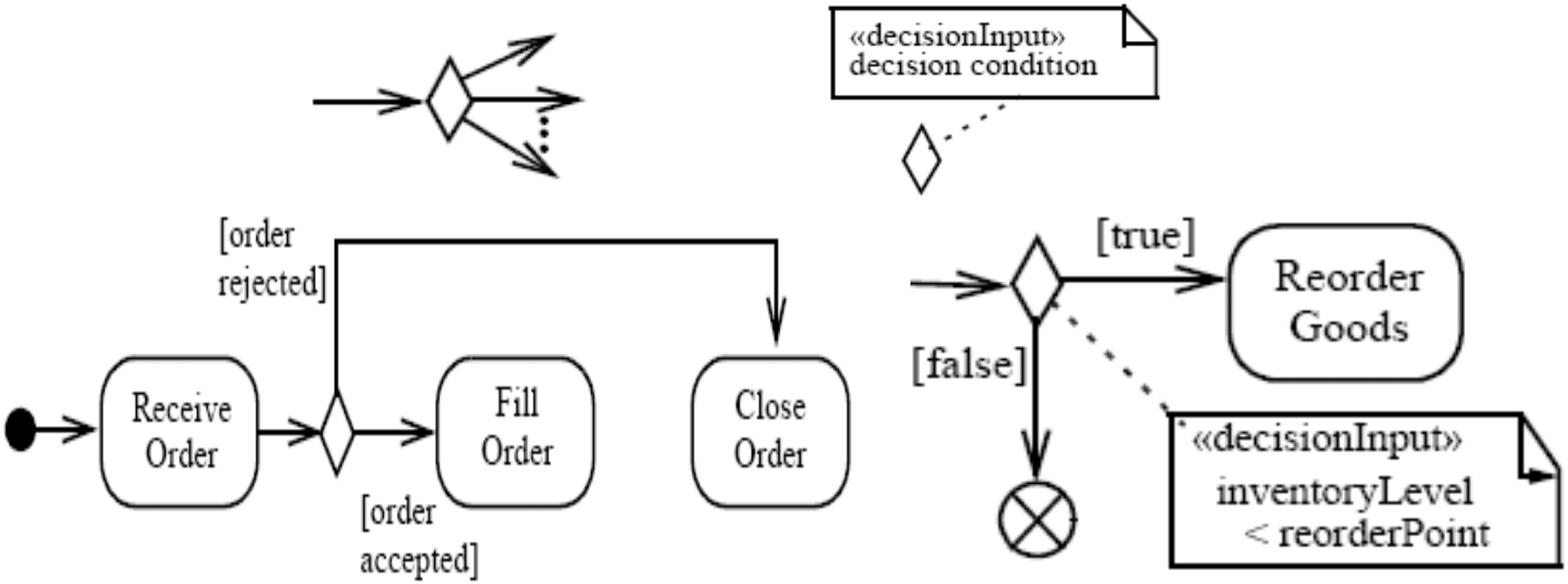
- If an initial node has more than one outgoing edge, only one of these edges will receive control.



A or B ?

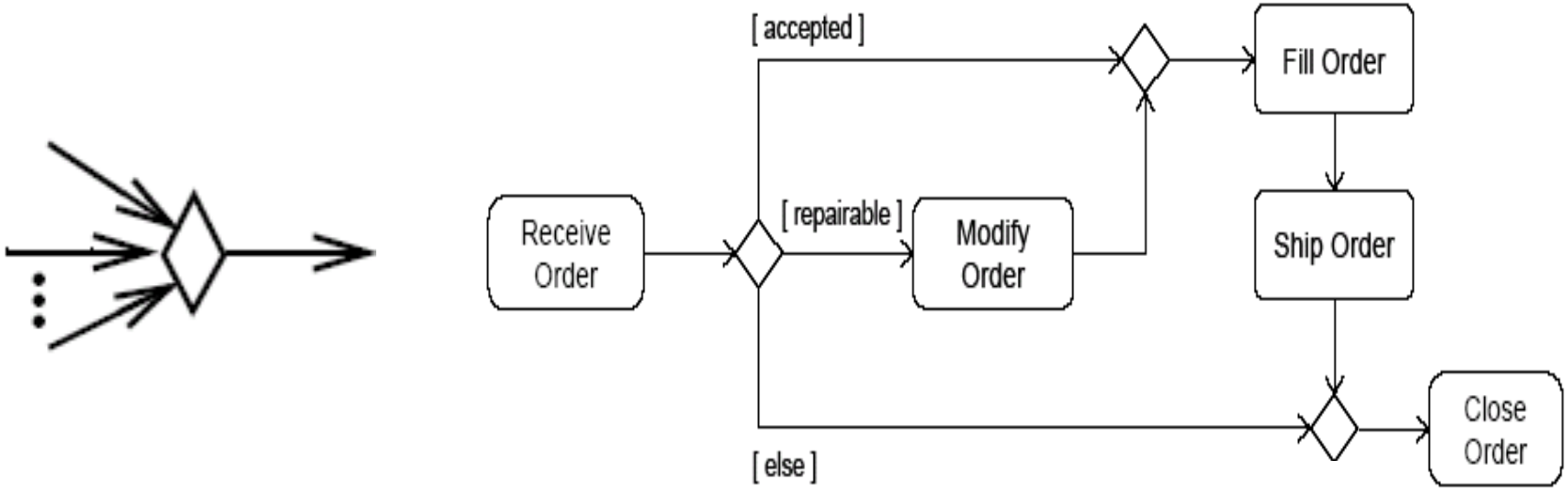
Control Nodes: Decision Nodes

- Route the flow to one of the outgoing edges.
- **Guards** are specified on the outgoing edges or with the stereotype «decisionInput».



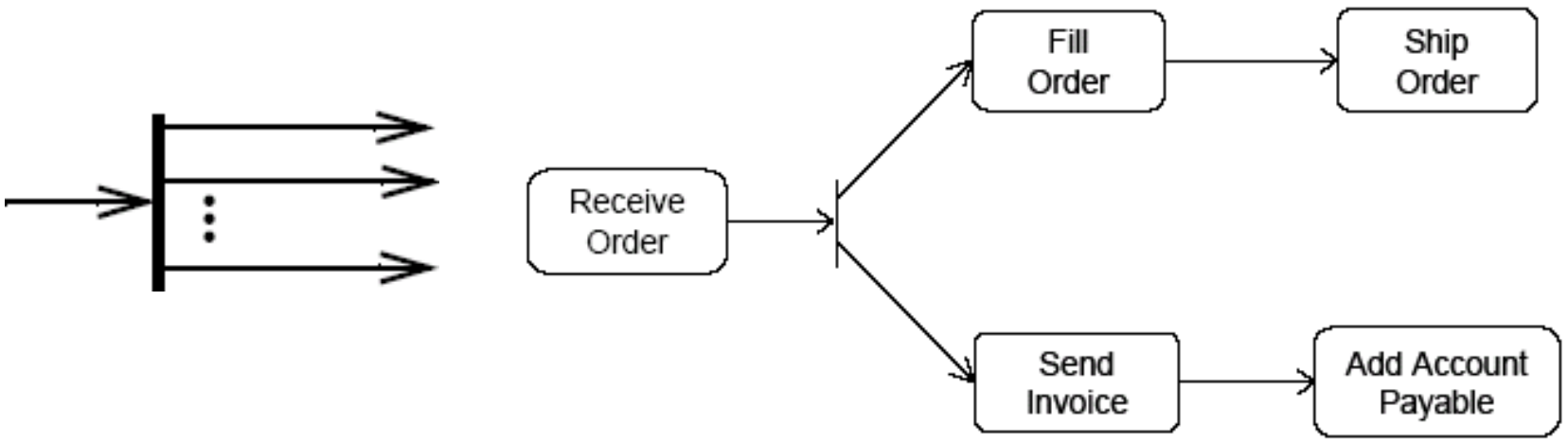
Control Nodes: Merge Nodes

- Bring together multiple alternate flows.
- All controls and data arriving at a merge node are immediately passed to the outgoing edge.
- There is no synchronization of flows.



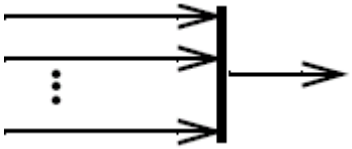
Control Nodes: Fork Nodes

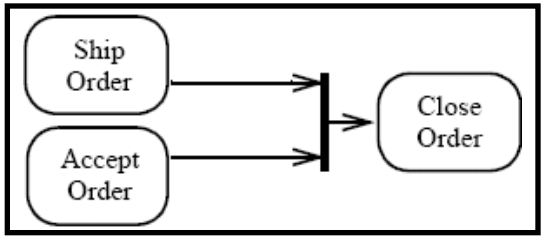
- Fork nodes split flows into multiple concurrent flows.



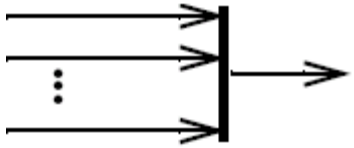
Control Nodes: Join Nodes

- Join nodes synchronize multiple flows.

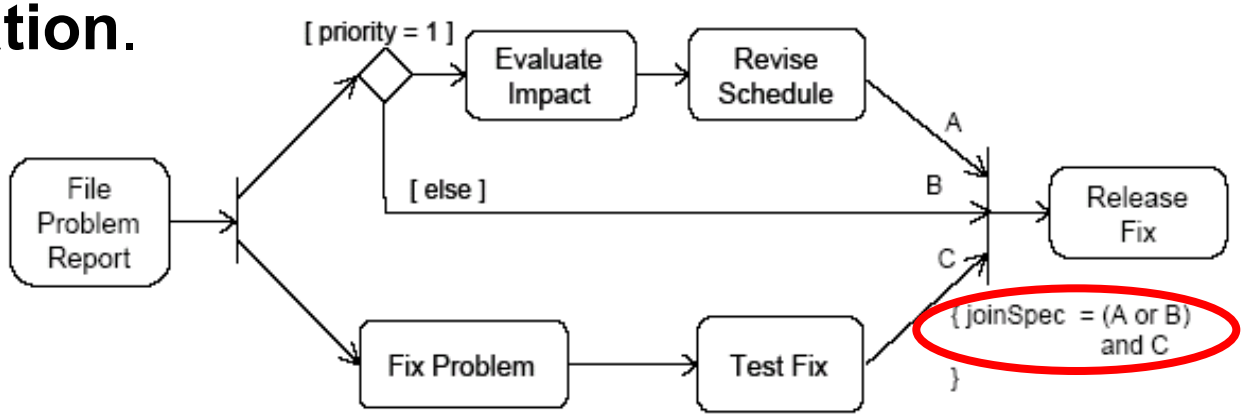




- Generally, controls or data must be available on every incoming edge in order to be passed to the outgoing edge, but the user can specify different conditions under which a join accepts incoming controls and data using a **join specification**.

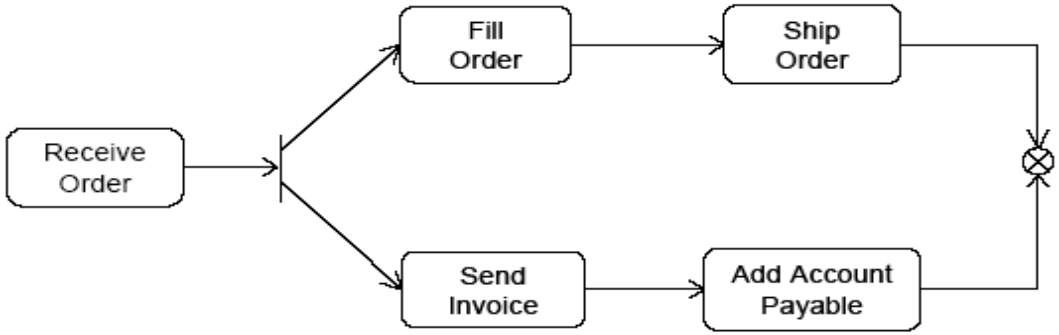


{joinSpec = ...}

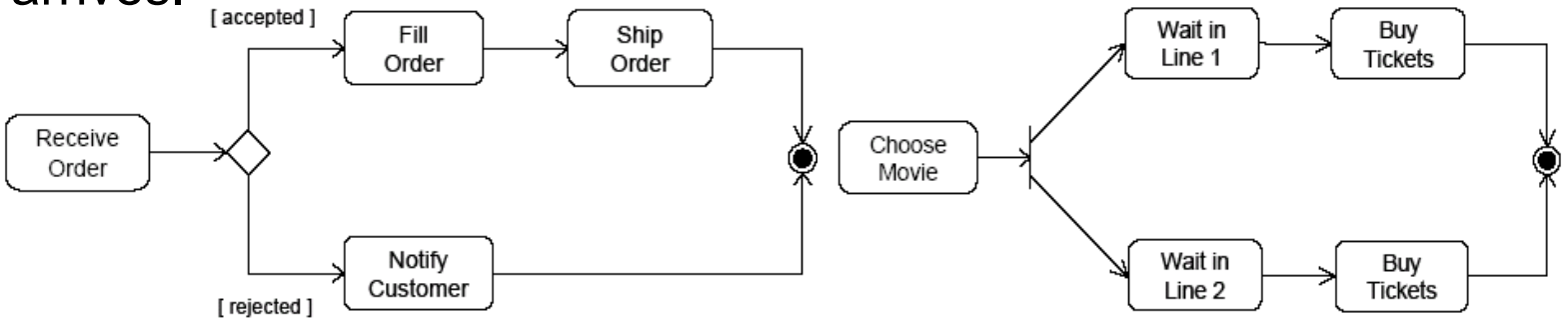


Control Nodes: Final Nodes

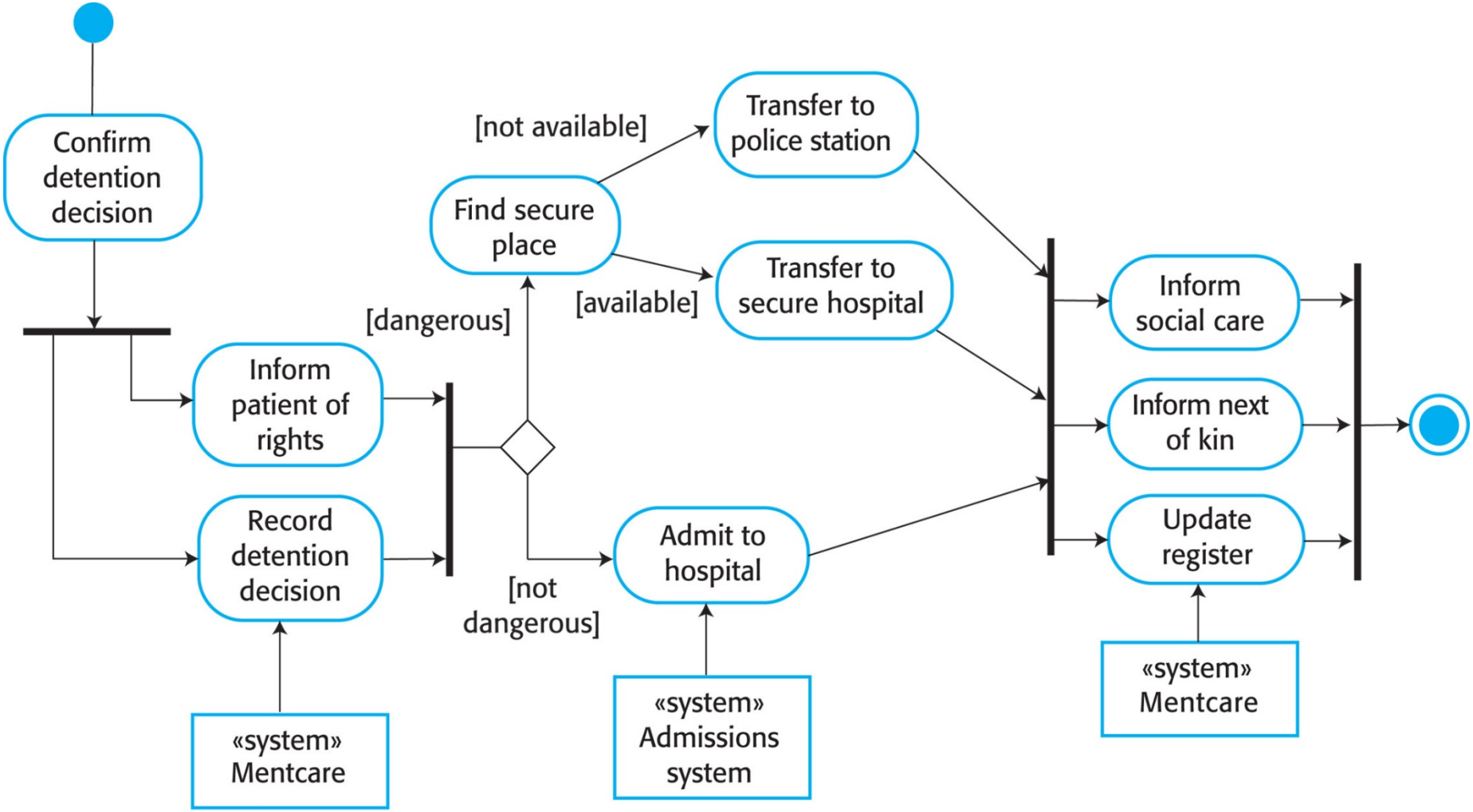
- **Flow final**:
  - ❖ Terminates the data flow that arrives into it.
  - ❖ The activity is terminated when all flows in the graph are terminated.



- **Final node (activity final)**: The activity is terminated when the first flow arrives.



AD: A More Sophisticated Example



SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

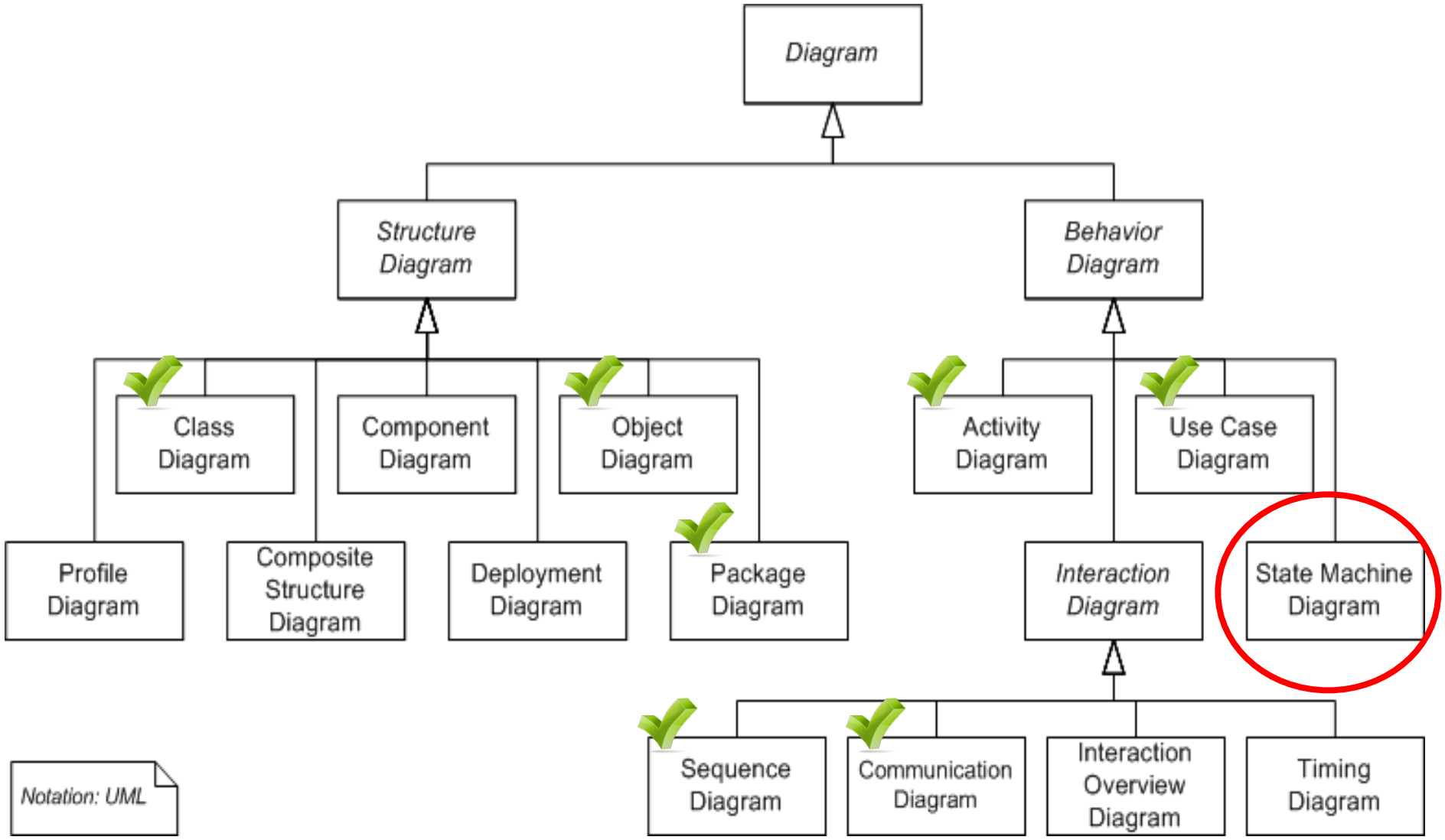
State Diagrams

Week 5: System Modeling, Part 2

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

UML Diagrams



State Machine

"A state diagram shows a state machine."

"A state machine is a behavior that specifies the sequence of states an object goes through during its lifetime in response to events, together with its responses to those events."

"A state is a condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for an event."

"An event is a specification of a significant occurrence that has a location in space and time."

—*The UML User Guide, 2nd Edition*

❖There are in general four types of events: call events, change events, signal events, and time events.

Four Things You Should Remember

1. An object must be in some specific state at any given time during its life cycle.

2. An object transitions from one state to another as the result of some **event** that affects it.

3. There can be only one initial state in a state diagram, but there may be many intermediate and final states.

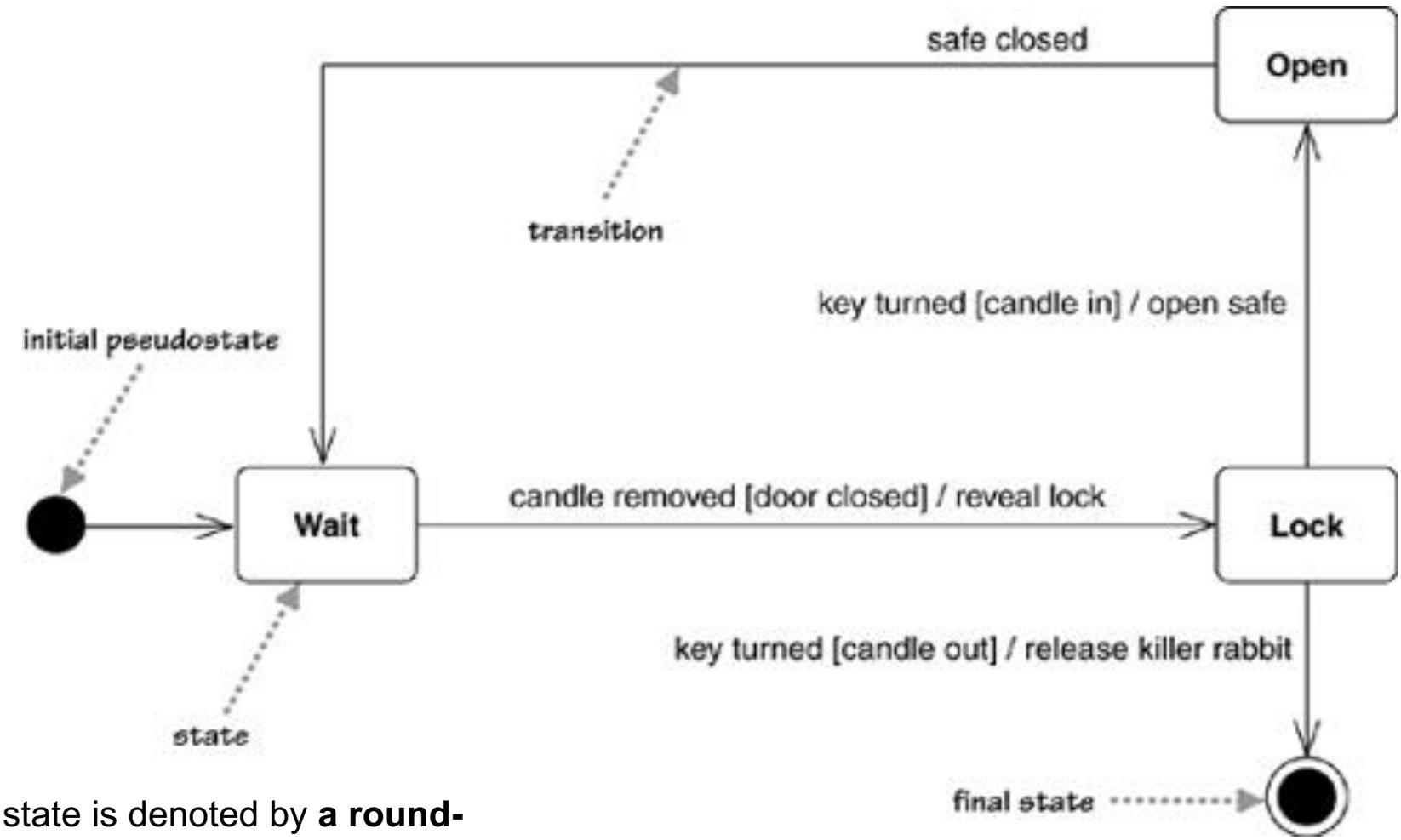
4. You may create a state diagram for any class, operation in a class, collaboration, or use case in a UML model.

A State Diagram for a Controller

- This example is from the book *UML Distilled*, by Martin Fowler. It's about a controller of a secret panel in a Gothic castle. Its "specifications" follow:

> "In this castle, I want to keep my valuables in a safe that's hard to find.
>
> So to reveal the lock to the safe, I have to remove a strategic **candle** from its holder, but this will reveal the lock only while the door is closed.
>
> Once I can see the lock, I can insert my **key** to open the safe. For extra safety, I make sure that I can open the safe only if I replace the candle first.
>
> If a thief neglects this precaution, I'll unleash a nasty monster to devour him."

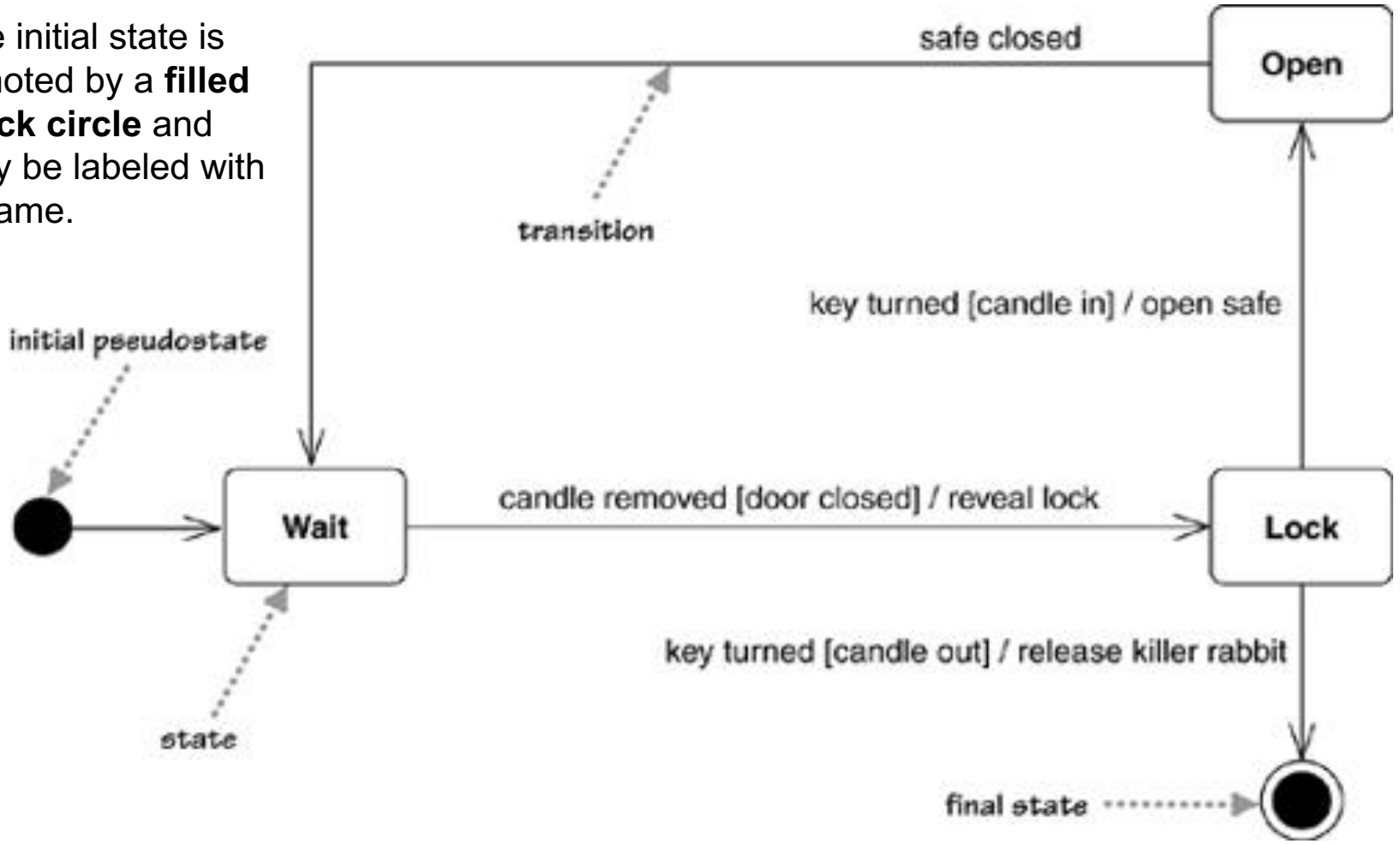
A State Diagram for a Controller



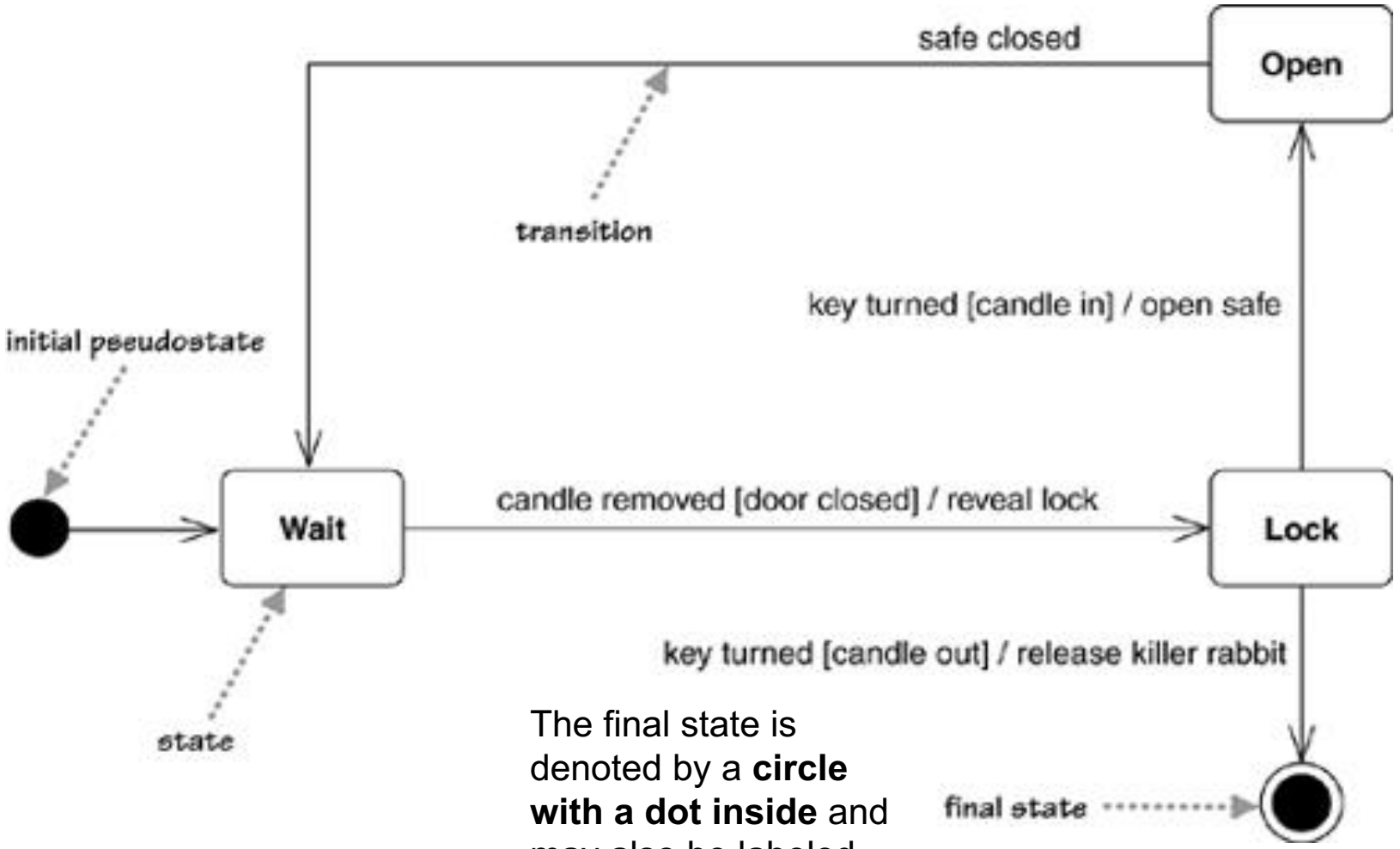
A state is denoted by **a round-cornered rectangle** with the name of the state written inside it.

A State Diagram for a Controller

The initial state is denoted by a **filled black circle** and may be labeled with a name.



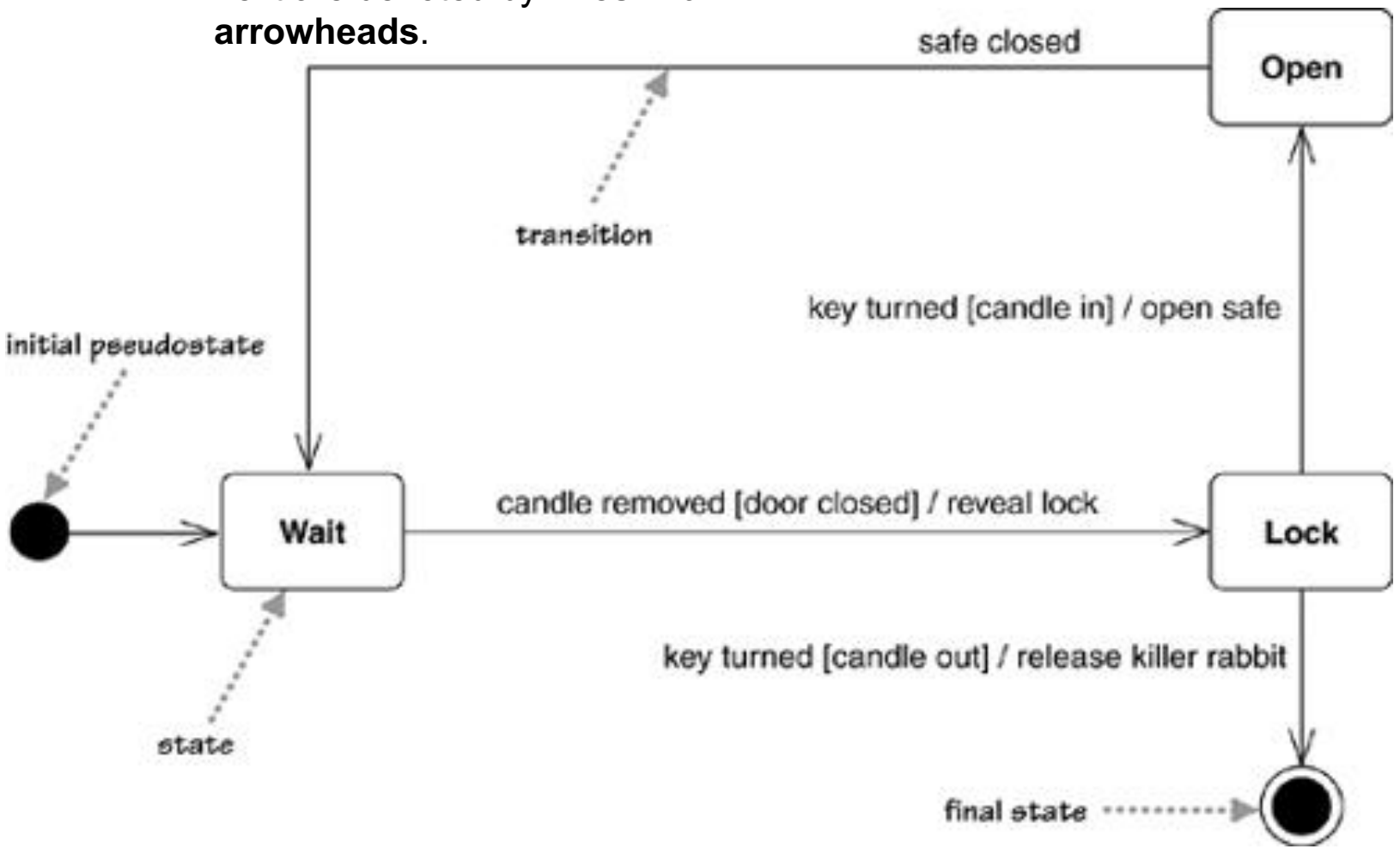
A State Diagram for a Controller



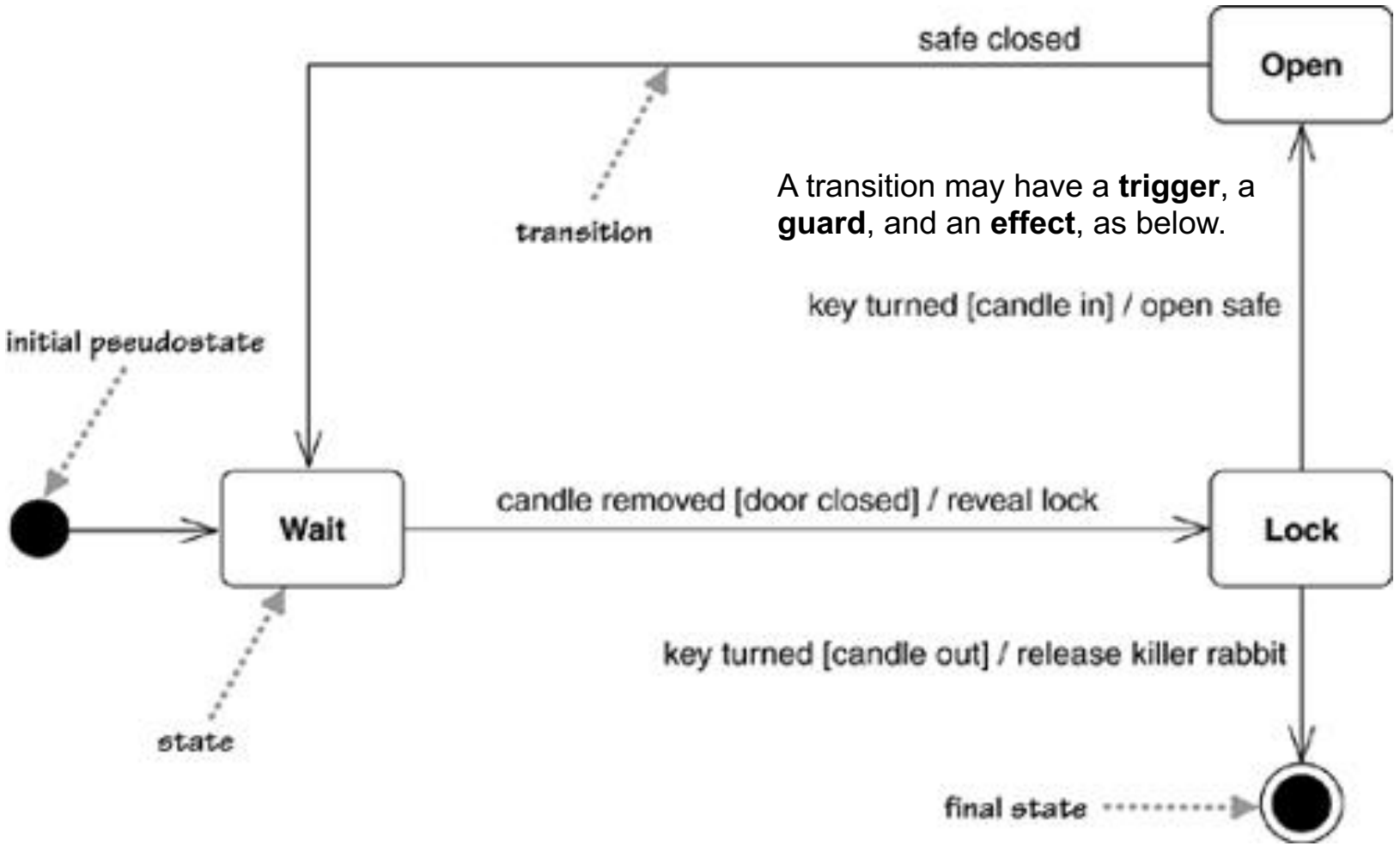
The final state is denoted by a **circle with a dot inside** and may also be labeled with a name.

A State Diagram for a Controller

Transitions from one state to the next are denoted by **lines with arrowheads**.



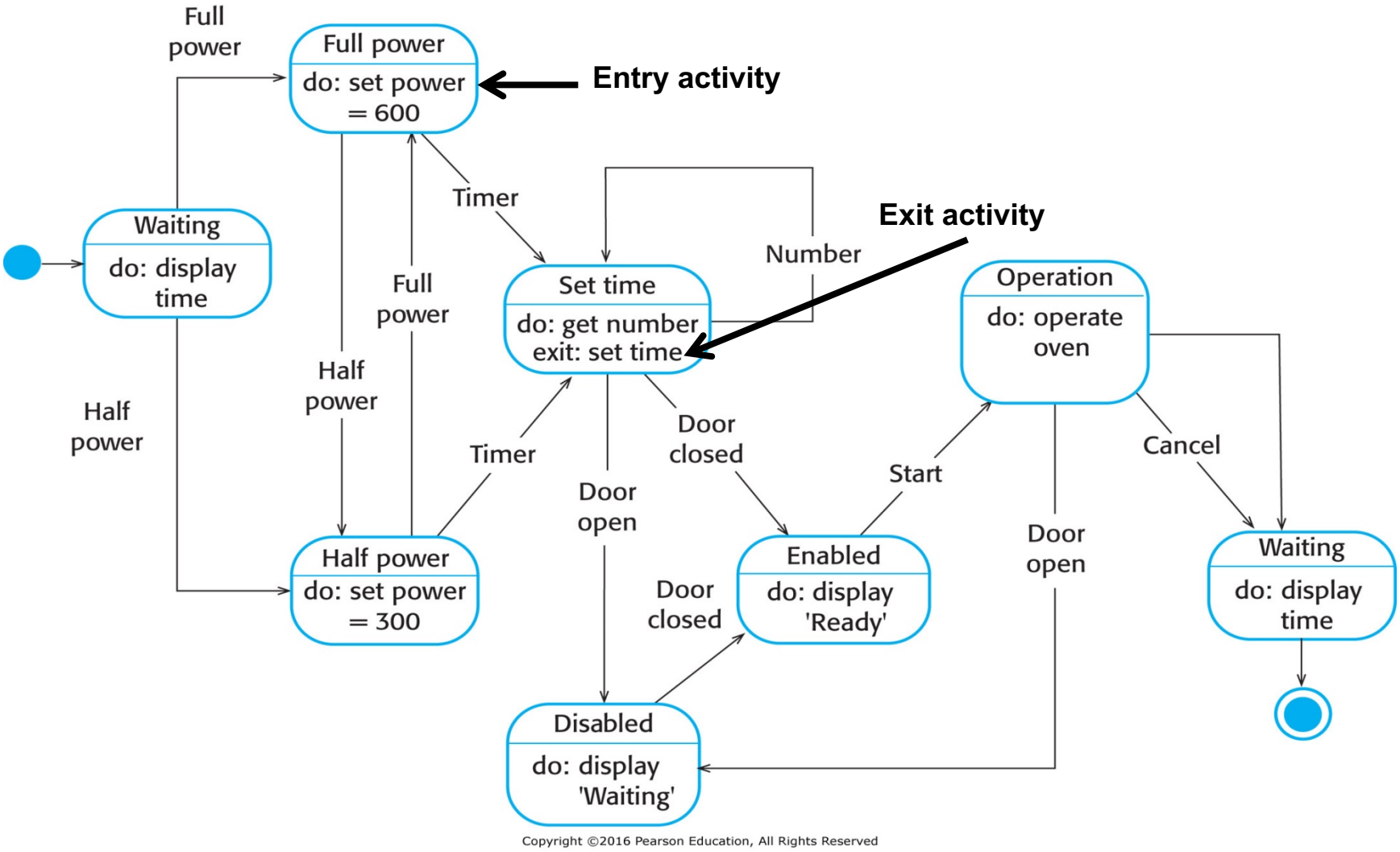
A State Diagram for a Controller



State Diagrams: Transitions

- **Trigger** is the cause of the transition, which could be a signal, an event, a change in some condition, or the passage of time.
- **Guard** is a condition that must be true in order for the trigger to cause the transition.
- **Effect** is an action that will be invoked directly on the object that owns the state machine as a result of the transition.

An Example from the Textbook



SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Identifying Classes, Attributes, and Operations

Week 5: System Modeling, Part 2

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Identifying Classes: An Example

System Requirements (An example)

1. A local **bank** intends to install a new automated teller machine (**ATM**) to allow users (i.e., bank **customers**) to perform basic financial **transactions**.
2. Each user can have only one **account** at the bank.
3. ATM users should be able to view their account **balance**, withdraw **cash**, and deposit **funds**. The user interface of the automated teller machine contains:
   a) A **screen** that displays messages to the user
   b) A **keypad** that receives numeric input from the user
   c) A **cash dispenser** that dispenses cash to the user and
   d) A **deposit slot** that receives **deposit envelopes** from the user.
4. The cash dispenser begins each day loaded with 500 **$20 bills**.
5. An ATM session consists of authenticating a user (i.e., proving the user's identity) based on an **account number** and personal identification number (**PIN**), followed by creating and executing financial transactions. To authenticate a user and perform transactions, the ATM must interact with the bank's account information **database**. For each bank account, the database stores an account number, a PIN, and a balance indicating the amount of money in the account….

This example is from Java for Programmers, 2nd edition, Deital & Deitel, 2011

Identifying Classes: An Example

- Identify nouns and noun phrases in the requirement document:

bank	money/funds
account number	ATM
screen	PIN
keypad	bank database
customer/user	cash dispenser
balance inquiry	transaction
$20 bill/cash	withdrawal
account	deposit slot
deposit	deposit envelope
balance	

Identifying Classes: An Example

- Create classes only for the nouns and noun phrases that have significance in the system.
  - ❖ We don’t model “bank” as a class, because the bank is not a part of the ATM system—the bank simply wants us to build the ATM.

Identifying Classes: An Example

- Identify nouns and noun phrases in the requirement document:

bank ✗	money/funds
account number	ATM
screen	PIN
keypad	bank database
customer/user	cash dispenser
balance inquiry	transaction
$20 bill/cash	withdrawal
account	deposit slot
deposit	deposit envelope
balance	

Identifying Classes: An Example

- Create classes only for the nouns and noun phrases that have significance in the system.

  - ❖ Customer” and “user” also represent outside entities; we do not need to model them as classes in the system.

Identifying Classes: An Example

- Identify nouns and noun phrases in the requirement document:

bank ✗	money/funds
account number	ATM
screen	PIN
keypad	bank database
customer/user ✗	cash dispenser
balance inquiry	transaction
$20 bill/cash	withdrawal
account	deposit slot
deposit	deposit envelope
balance	

Identifying Classes: An Example

- Create classes only for the nouns and noun phrases that have significance in the system.

  - ❖ We do not need to model “$20 bill” as a class. These are physical objects in the real world, but they’re not part of what is being automated.
  - ❖ We can adequately represent the presence of **bills** in the system using an **attribute** of the class that models the **cash dispenser**.

Identifying Classes: An Example

- Identify nouns and noun phrases in the requirement document:

bank ✗	money/funds
account number	ATM
screen	PIN
keypad	bank database
customer/user ✗	cash dispenser
balance inquiry	transaction
$20 bill/cash ✗	withdrawal
account	deposit slot
deposit	deposit envelope
balance	

Identifying Classes: An Example

- Create classes only for the nouns and noun phrases that have significance in the system.

  - ❖ We do not need to model “deposit envelope” as a class either.
  - ❖ We can assume there is going to be an **operation,** performed by the class that models the **deposit slot,** that acknowledges the receipt of an envelope.
  - ❖ That should be sufficient to represent the presence of an envelope in the system.

Identifying Classes: An Example

- **Identify nouns and noun phrases in the requirement document:**

bank ✗	money/funds
account number	ATM
screen	PIN
keypad	bank database
customer/user ✗	cash dispenser
balance inquiry	transaction
$20 bill/cash ✗	withdrawal
account	deposit slot
deposit	deposit envelope ✗
balance	

Identifying Classes: An Example

- Create classes only for the nouns and noun phrases that have significance in the system.

  - Representing various amounts of “money,” including an account’s “balance,” as **attributes** of classes, seems most appropriate, since they do not seem have to exhibit behaviors.

Identifying Classes: An Example

- Identify nouns and noun phrases in the requirement document:

bank ✗	money/funds ✗
account number	ATM
screen	PIN
keypad	bank database
customer/user ✗	cash dispenser
balance inquiry	transaction
$20 bill/cash ✗	withdrawal
account	deposit slot
deposit	deposit envelope ✗
balance ✗	

Identifying Classes: An Example

- Create classes only for the nouns and noun phrases that have significance in the system.

  - Likewise, although the nouns “account number” and “PIN” represent significant pieces of information in the system, they could simply be **attributes** of a bank account, since they do not seem have to exhibit behaviors.
  - We can most appropriately model them as attributes of an **account** class.

Identifying Classes: An Example

- Identify nouns and noun phrases in the requirement document:

bank ✗	money/funds ✗
account number ✗	ATM
screen	PIN ✗
keypad	bank database
customer/user ✗	cash dispenser
balance inquiry	transaction
$20 bill/cash ✗	withdrawal
account	deposit slot
deposit	deposit envelope ✗
balance ✗	

Identifying Classes: An Example

- Create classes only for the nouns and noun phrases that have significance in the system.

  - Although the requirements document frequently describes a "transaction" in a general sense, we should refine the broad notion of a financial transaction: "**ATM users should be able to view their account balance, withdraw cash and deposit funds**."
    - Instead of creating a generic **Transaction** class, we could model the three types of transactions mentioned in the requirements (i.e., "**balance inquiry**," "**withdrawal**," and "**deposit**") as individual classes.

Identifying Classes: An Example

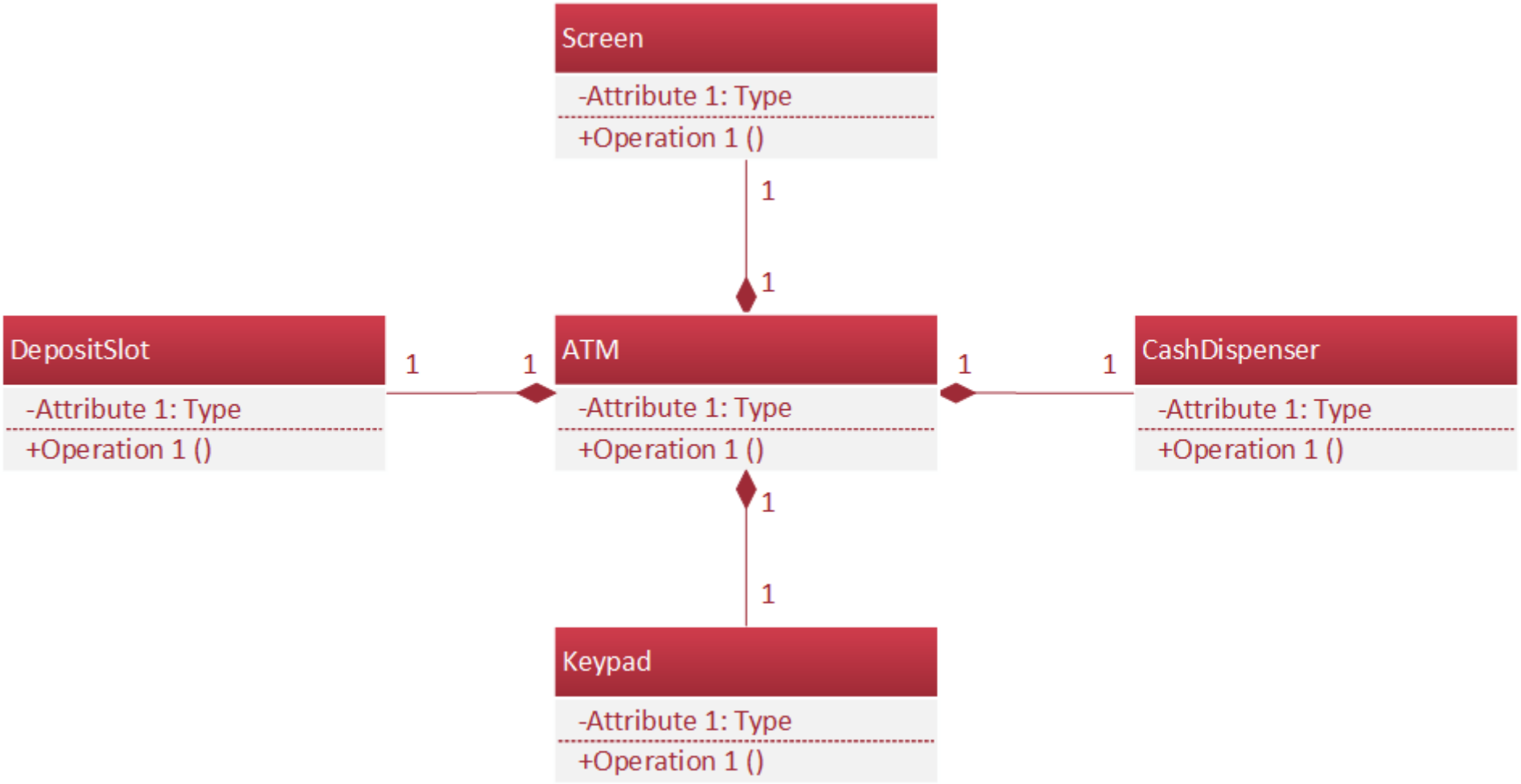
- Identify nouns and noun phrases in the requirement document:

bank ✗	money/funds ✗
account number ✗	ATM
screen	PIN ✗
keypad	bank database
customer/user ✗	cash dispenser
balance inquiry	transaction ✗
$20 bill/cash ✗	**withdrawal**
account	deposit slot
deposit	deposit envelope ✗
balance ✗	

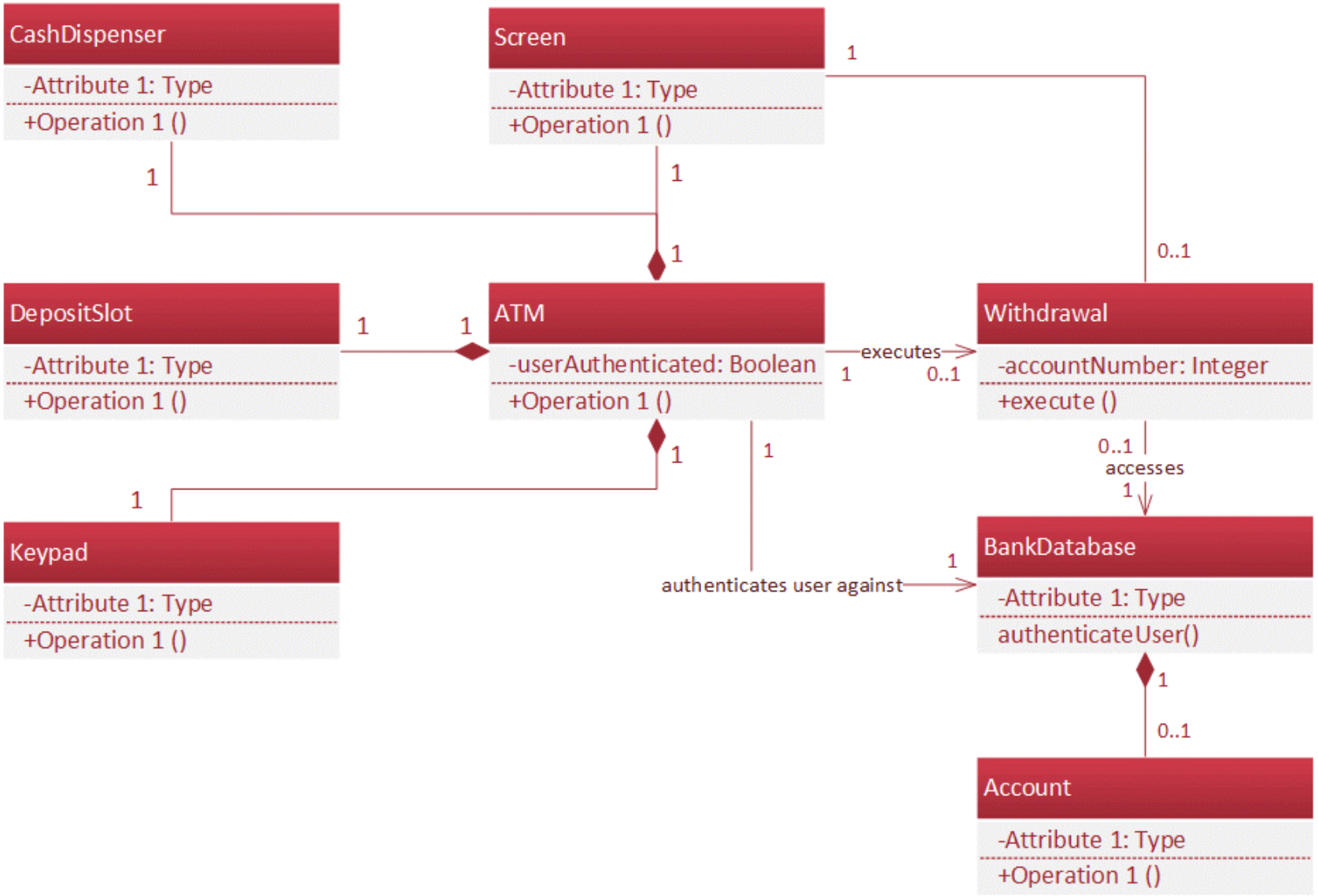
Identifying Classes: An Example

- The final list
  - ATM
  - screen
  - keypad
  - cash dispenser
  - deposit slot
  - account
  - bank database
  - **balance inquiry**
  - **withdrawal**
  - **deposit**

Identifying Classes: An Example



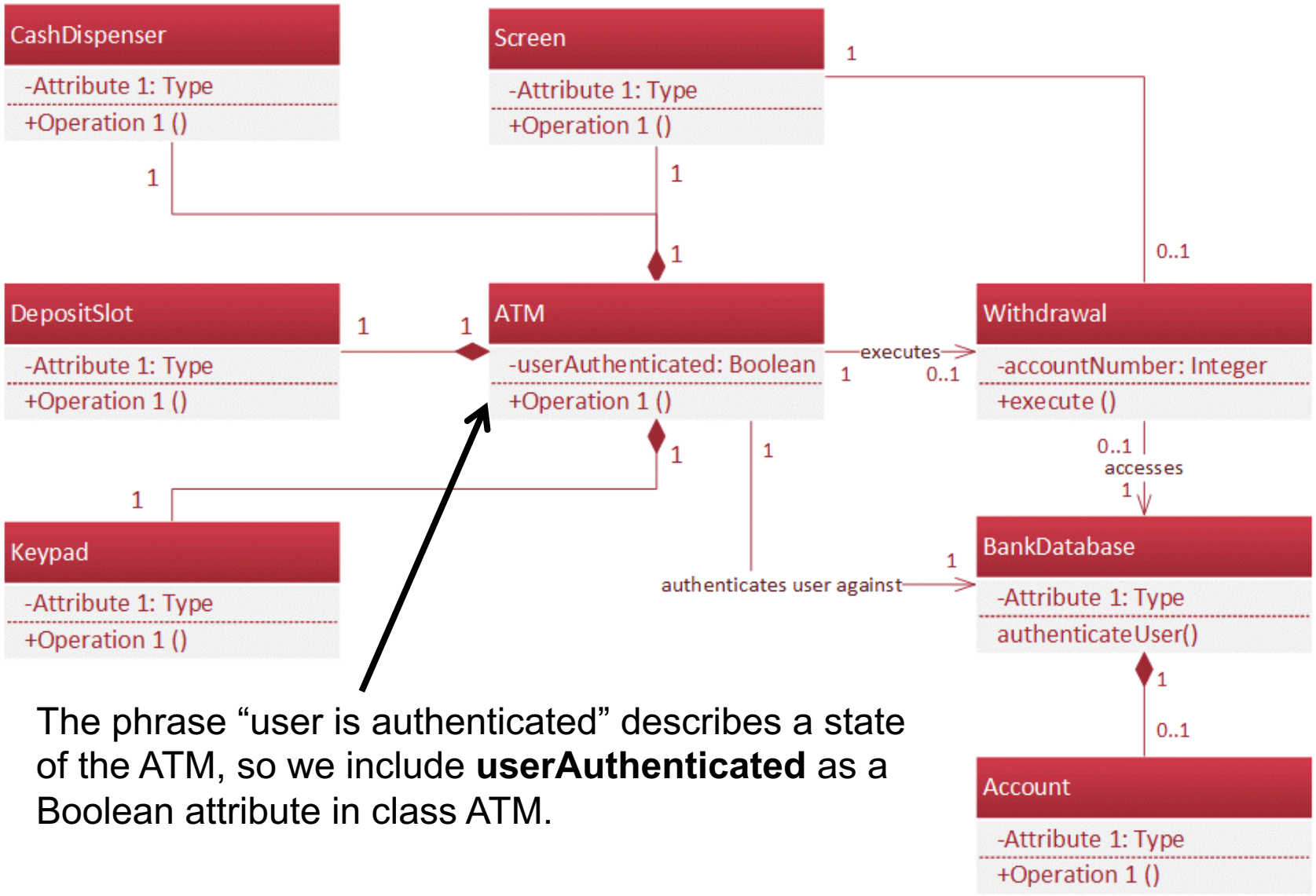
Identifying Classes: An Example



Identifying Attributes

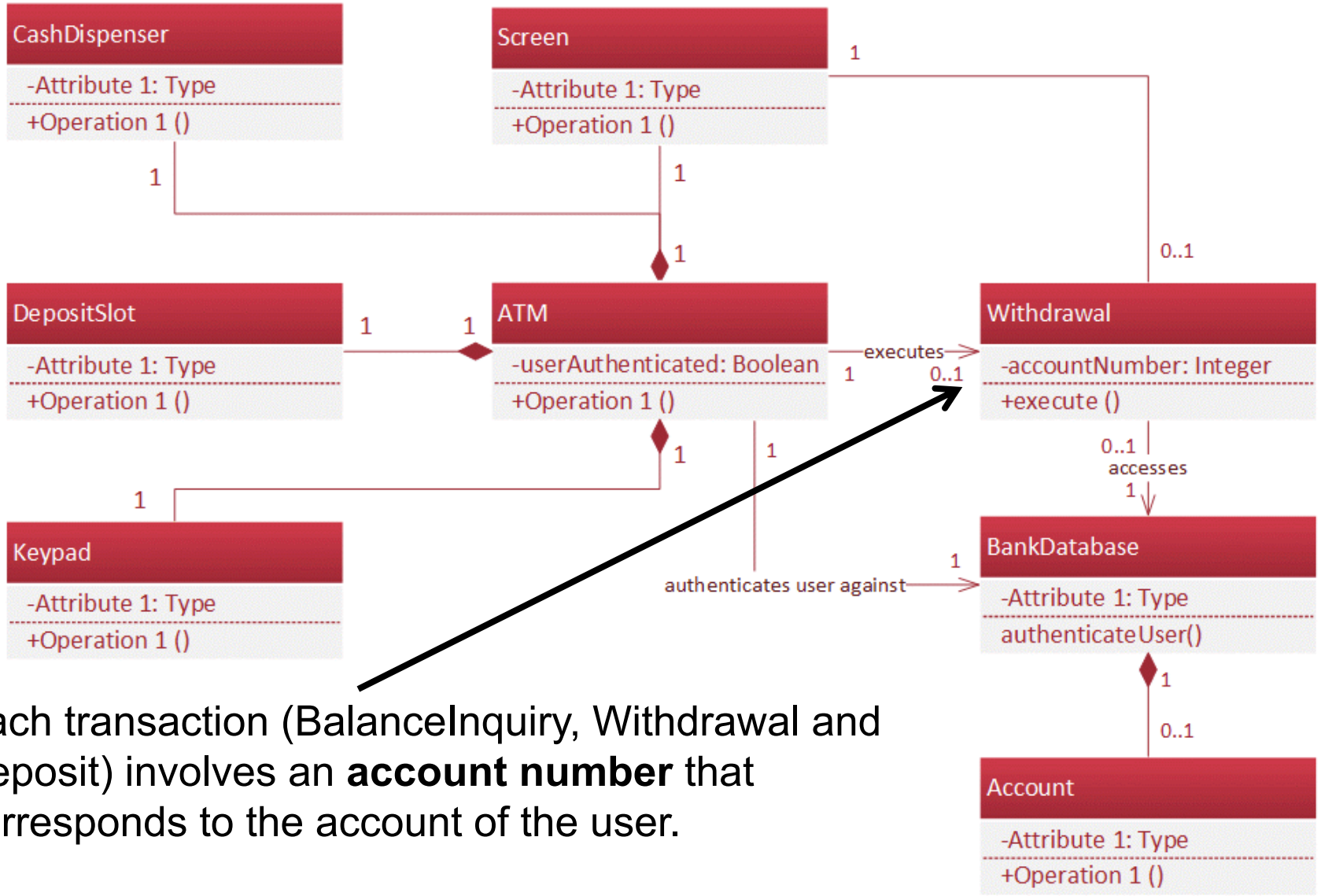
- We can identify many attributes of the classes in our system by looking for **descriptive words and phrases** in the requirements document.
  - For each such word and phrase that plays a significant role in the ATM system, we create an attribute and assign it to one or more of the identified classes.
  - We also create attributes to represent any additional data that a class may need, as such needs become clear throughout the design process.

Identifying Classes: An Example



The phrase “user is authenticated” describes a state of the ATM, so we include **userAuthenticated** as a Boolean attribute in class ATM.

Identifying Classes: An Example

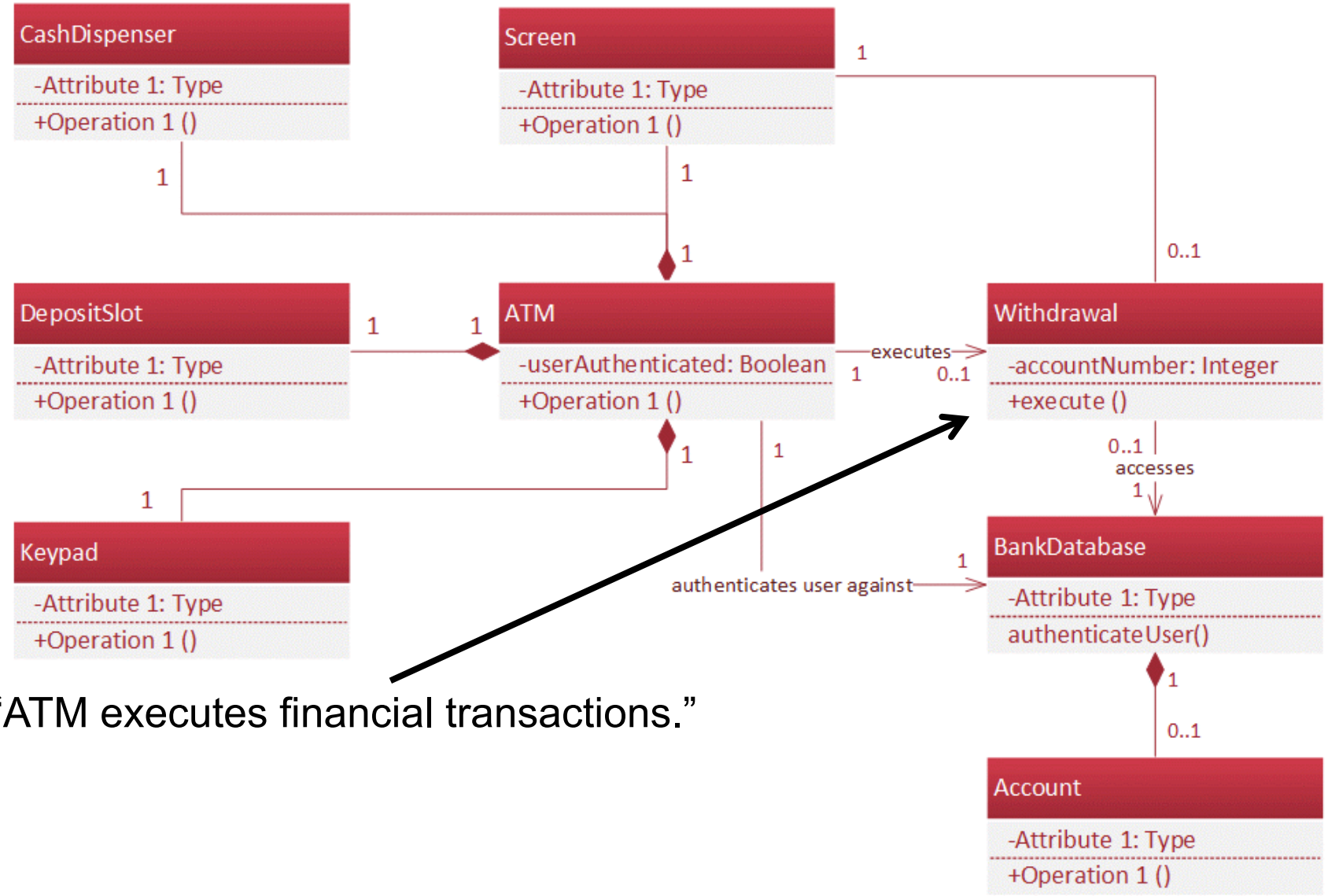


Each transaction (BalanceInquiry, Withdrawal and Deposit) involves an **account number** that corresponds to the account of the user.

Identify Class Operations

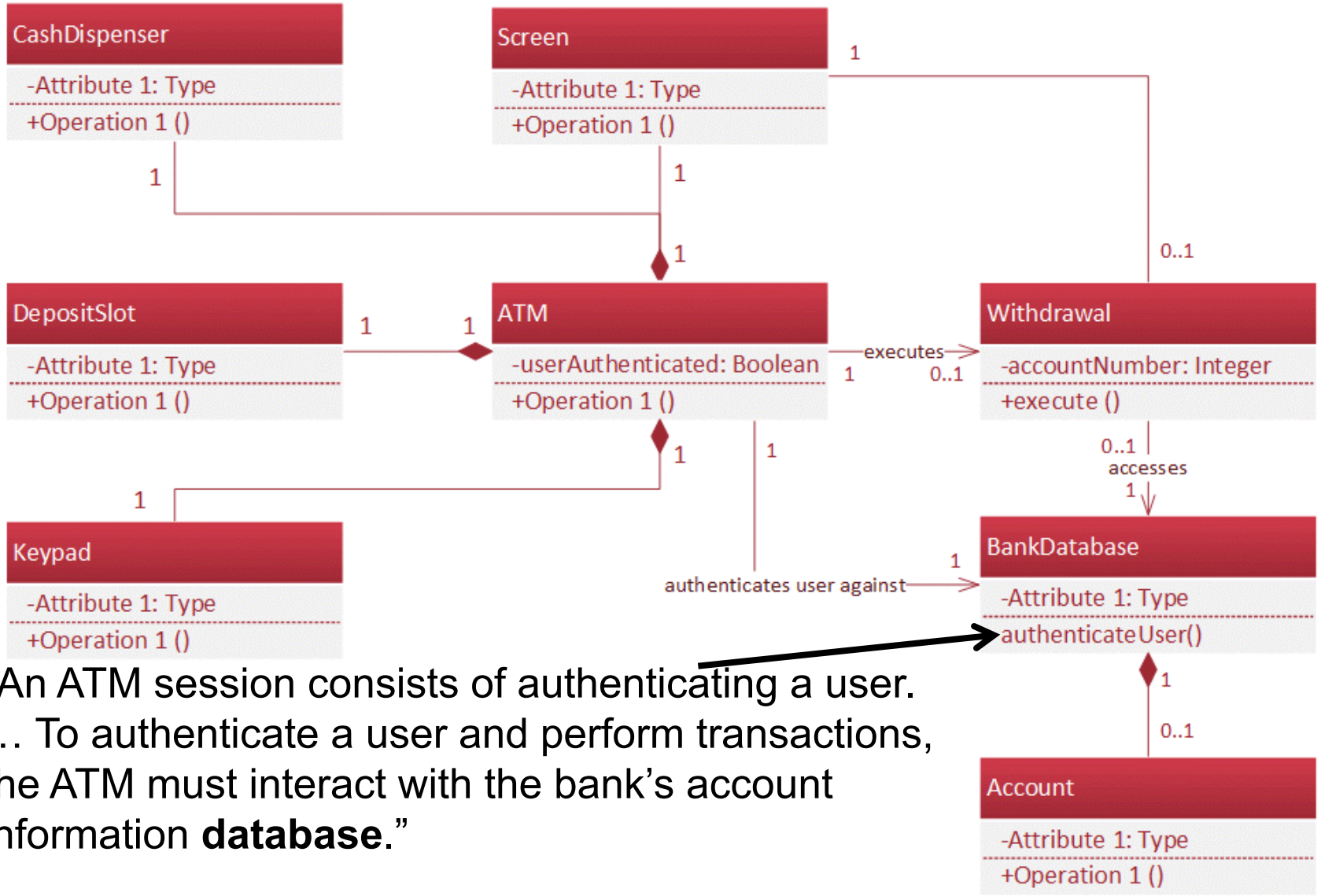
- We can derive many of the class operations by examining the key verbs and verb phrases in the requirements document.
  - ❖ We then relate these verbs and verb phrases to classes in our system.
  - ❖ The verb phrases extracted from the SRS in the following table help us determine the operations of each class.
  - ❖ Example: “ATM executes financial transactions.”

Identifying Classes: An Example



"ATM executes financial transactions."

Identifying Classes: An Example



"An ATM session consists of authenticating a user. … To authenticate a user and perform transactions, the ATM must interact with the bank's account information **database**."

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE