

Software Engineering

Module 5: Requirements Analysis and Modeling

Outline

- Part I: Requirements Modeling Concept
 - Use Case Modeling and Specification
 - Domain Modeling
 - Activity Diagram
 - Sequence Diagram
 - State Chart/State Machine Diagram
- Part II: Requirements Document
 - Software Requirements Specification (SRS)

Note: The overall contents of the slide are based on the main reference that is Sommerville (2016) with other references specified directly in respective slides (if any)

Part I: Requirements modeling concept

Use Case Modeling and Specification, Sequence Diagram, Activity Diagram, Domain Modeling, State Chart/Machine Diagram

Note: Main reference for Part I: Sommerville (2016), Arlow and Neustadt (2002)

Objectives

- To define requirements using **use cases** and problem **domain classes**
- To identify and analyze events and resulting use cases for **use case diagram**
- To identify and analyze domain classes for **domain model class diagram**
- To produce **detail requirements** using sequence diagram, activity diagram and state chart/state machine diagram

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 represents a system using **graphical notation**, which is now almost always based on notations in the **Unified Modeling Language (UML)**
- System modeling helps the analyst to understand the **functionality of the system** and models are used to communicate with customers

Existing and Planned System Models

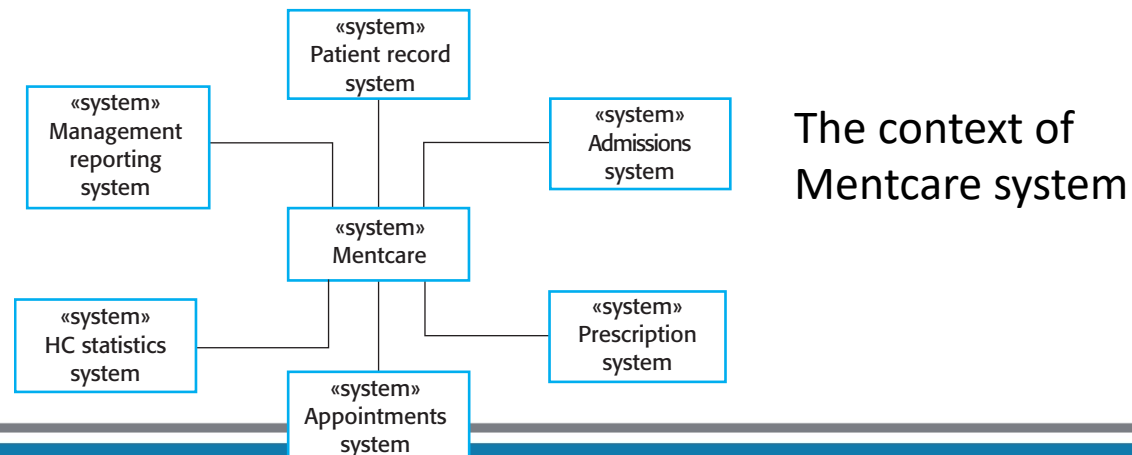
- Models of the **existing system** are used during requirements engineering
 - They help clarify what the existing system does and can be used as a **basis for discussing its strengths and weaknesses**
 - These then lead to requirements for the new system
- Models of the **new system** are used during requirements engineering to help **explain the proposed requirements to other system stakeholders**
 - Engineers use these models to discuss design proposals and to document the system for implementation

System Perspectives

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

Context Model

- Context models are used to illustrate the **operational context of a system** - they show what lies outside the system boundaries
- Social and organisational concerns may affect the decision on where to position system boundaries
- Architectural models show the system and its relationship with other systems



UML Diagram Types

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

Use of Graphical Models

- As a means of **facilitating discussion** about an existing or proposed system
 - Incomplete and incorrect models are fine at the early stage 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

Ways of Writing a System Requirements Specification

Recall

Notation	Description
Natural language	The requirements are written using numbered sentences in natural language. Each sentence should express one requirement.
Structured natural language	The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement.
Design description language	This approach uses a language like a programming language, but with more abstract features to specify the requirements by defining an operational model of the system. This approach is now rarely used although it can be useful for interface specifications.
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML use case and sequence diagrams are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don't understand a formal specification. They cannot check that it represents what they want and are reluctant to accept it as a system contract.

Interaction model

Use Case Diagram

Interaction Model

- 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
- For the scope of this course, **sequence diagrams** will be discussed later under **behavioural model**

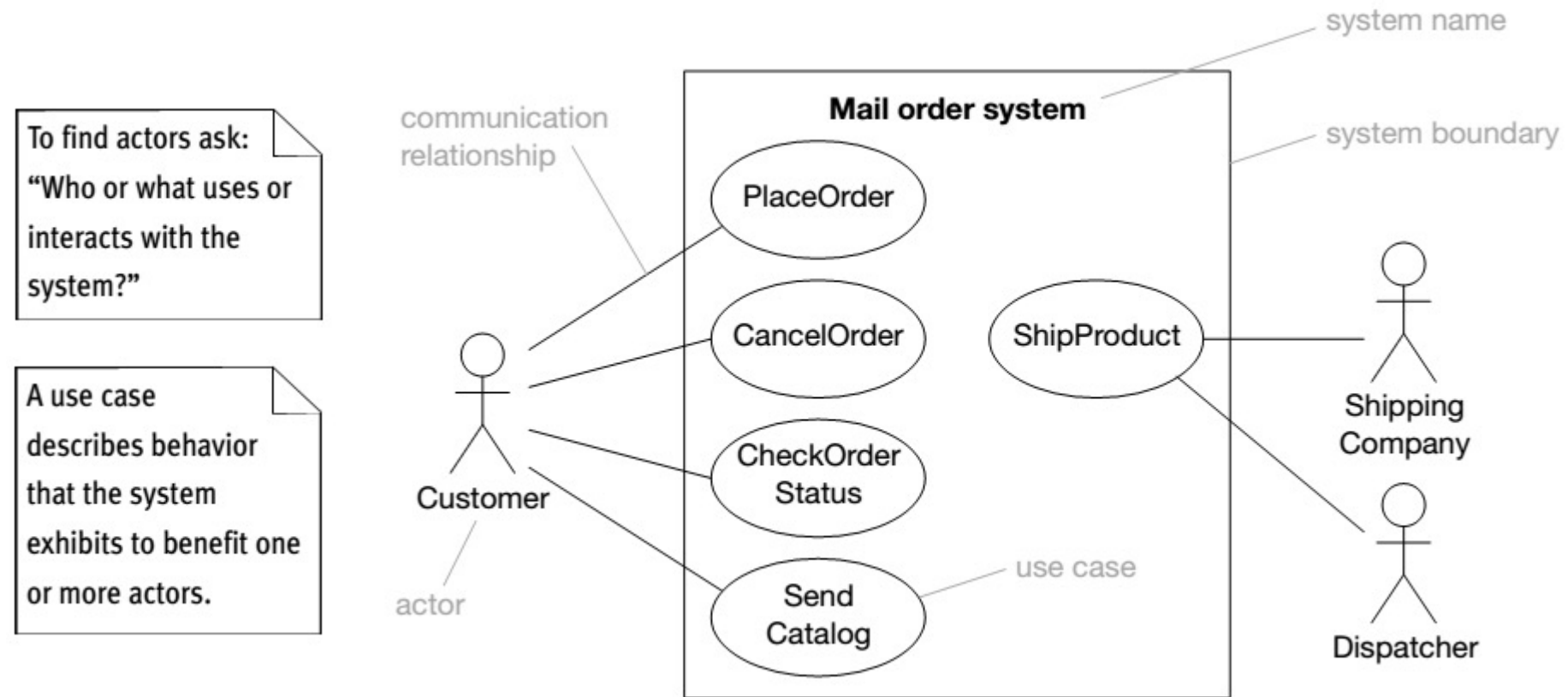
Use Case Modeling

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

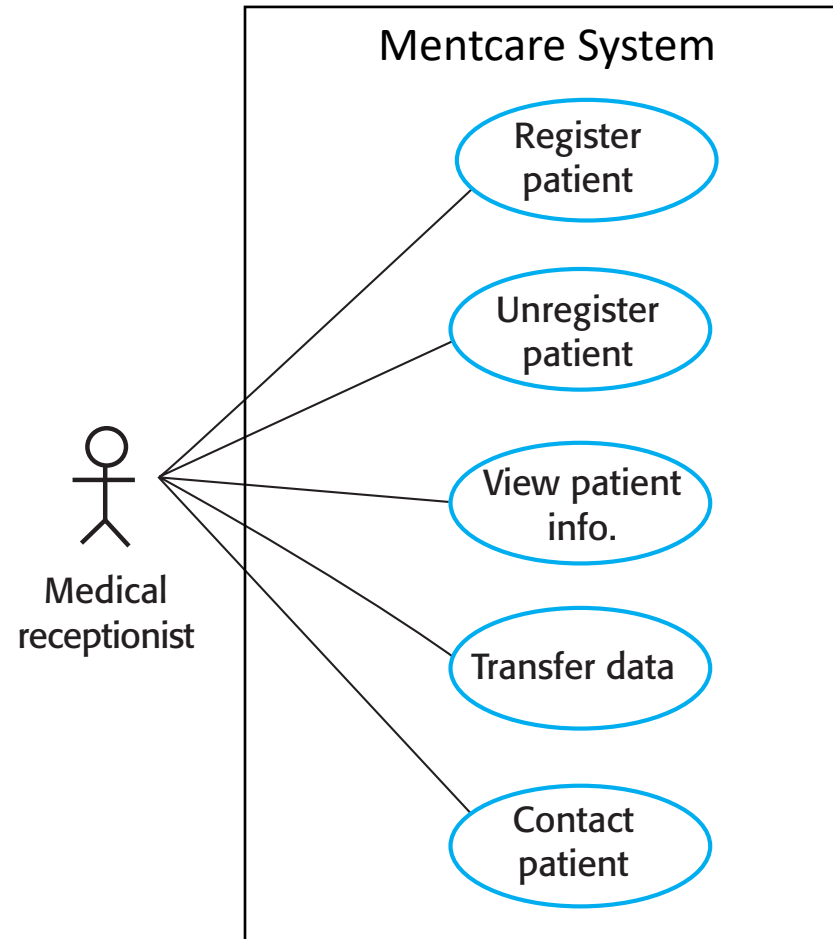
Stereotypes in UML

- Stereotypes allow us to **introduce new modeling elements** based on existing elements
- Can do this by appending the stereotype name in **guillemots («...»)** to the new element
- Each model element can have at most one stereotype
- Examples:
 1. <<include>>, <<extend>> in use case relationship
 2. <<entity>>, <<controller>>, <<boundary>> in class diagram as class stereotypes
 3. <<package>>, <<subsystem>> in package diagram

Use Case Diagram: Notation

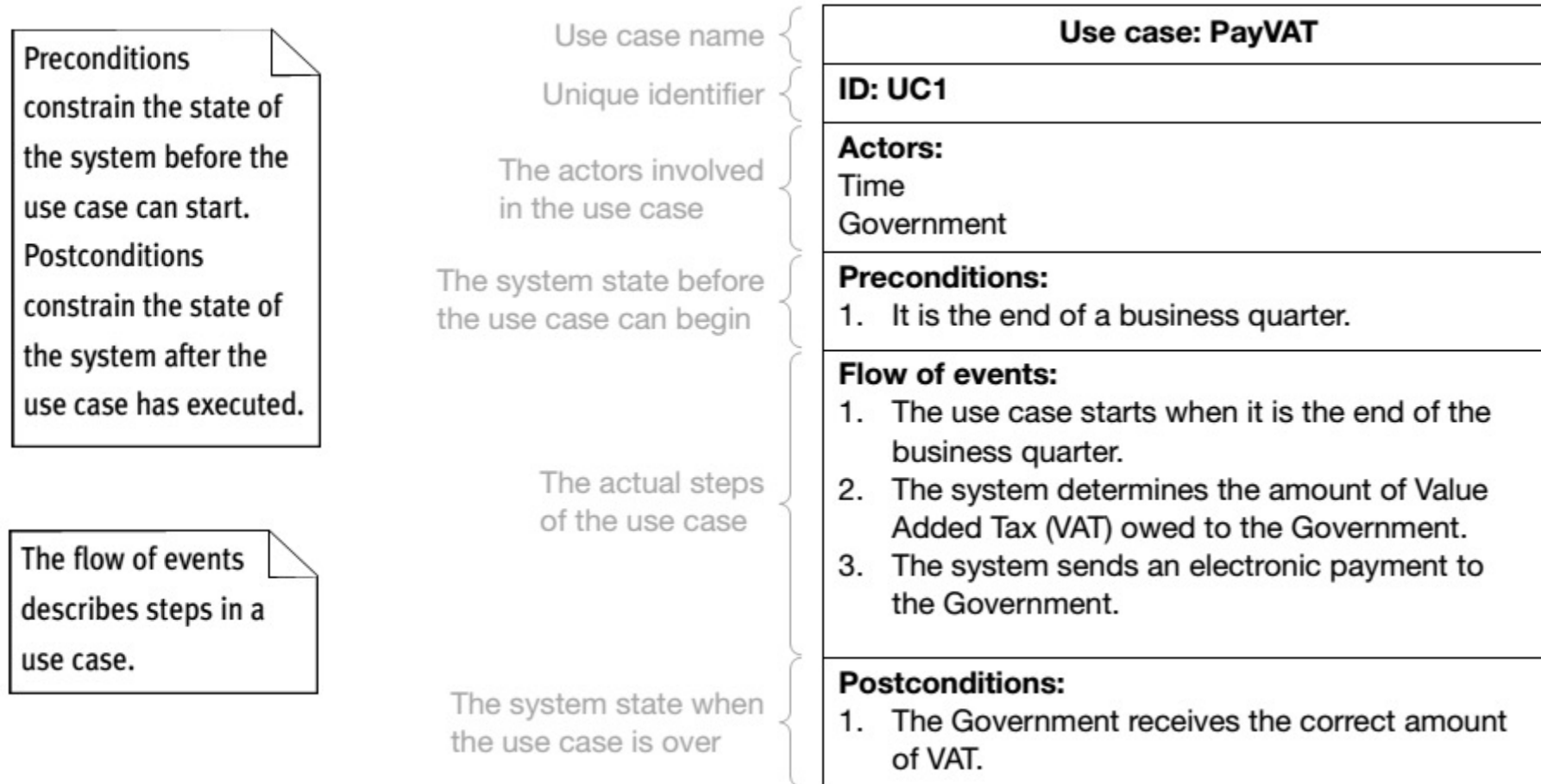


Example: Use Case Diagram for Mentcare System



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

Use Case Description



Branching within a Flow and Alternative Flow

Use case: FindProduct
ID: UC12
Actors: Customer
Preconditions:
Flow of events: <ol style="list-style-type: none">1. The Customer selects “find product”.2. The system asks the Customer for search criteria.3. The Customer enters the requested criteria.4. The system searches for products that match the Customer’s criteria.5. If the system finds some matching products then<ol style="list-style-type: none">5.1. For each product found<ol style="list-style-type: none">5.1.1. The system displays a thumbnail sketch of the product.5.1.2. The system displays a summary of the product details.5.1.3. The system displays the product price.6. Else<ol style="list-style-type: none">6.1. The system tells the Customer that no matching products could be found.
Postconditions:
Alternative flow: <ol style="list-style-type: none">1. At any point the Customer may move to different page.
Postconditions:

Use case: DisplayBasket
ID: UC11
Actors: Customer
Preconditions: <ol style="list-style-type: none">1. The Customer is logged on the system.
Flow of events: <ol style="list-style-type: none">1. The use case starts when the Customer selects “display basket”.2. If there are no items in the basket<ol style="list-style-type: none">2.1 The system informs the Customer that there are no items in the basket yet.2.2 The use case terminates.3. The system displays a list of all items in the Customer’s shopping basket including product ID, name, quantity and item price.
Postconditions:
Alternative flow 1: <ol style="list-style-type: none">1. At any time the Customer may leave the shopping basket screen.
Postconditions:
Alternative flow 2: <ol style="list-style-type: none">1. At any time the Customer may leave the system.
Postconditions:

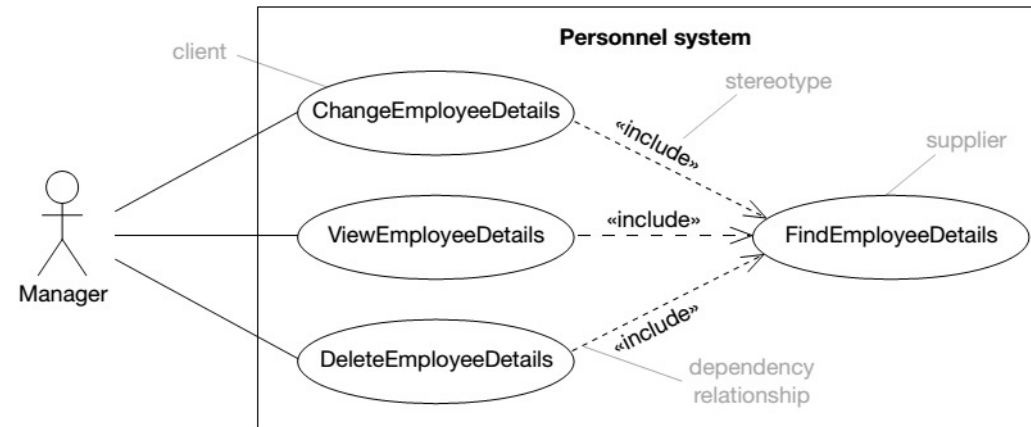
Repetition within a Flow: For/While

Use case: FindProduct
ID: UC12
Actors: Customer
Preconditions:
Flow of events: <ol style="list-style-type: none">1. The Customer selects “find product”.2. The system asks the Customer for search criteria.3. The Customer enters the requested criteria.4. The system searches for products that match the Customer’s criteria.5. If the system finds some matching products then<ol style="list-style-type: none">5.1. For each product found<ol style="list-style-type: none">5.1.1. The system displays a thumbnail sketch of the product.5.1.2. The system displays a summary of the product details.5.1.3. The system displays the product price.6. Else<ol style="list-style-type: none">6.1. The system tells the Customer that no matching products could be found.
Postconditions:
Alternative flow: <ol style="list-style-type: none">1. At any point the Customer may move to different page.
Postconditions:

Use case: ShowCompanyDetails
ID: UC13
Actors: Customer
Preconditions:
Flow of events: <ol style="list-style-type: none">1. The use case starts when the Customer selects “show company details”.2. The system displays a web page showing the company details.3. While the Customer is browsing the company details<ol style="list-style-type: none">3.1. The system plays some background music.3.2. The system displays special offers in a banner ad.
Postconditions:

<<include>> Relationship

«include» factors out steps common to several use cases into a separate use case which is then included.



ChangeEmployeeDetails
ID: UC1
Actors: Manager
Preconditions: 1. A valid Manager is logged on to the system.
Flow of events: 1. The Manager enters the employee's ID number. 2. include (FindEmployeeDetails). 3. The Manager selects the part of the employee details to change. 4. ...
Postconditions:

ViewEmployeeDetails
ID: UC2
Actors: Manager
Preconditions: 1. A valid Manager is logged on to the system.
Flow of events: 1. The Manager enters the employee's ID number. 2. include (FindEmployeeDetails). 3. The system displays the employee details. 4. ...
Postconditions:

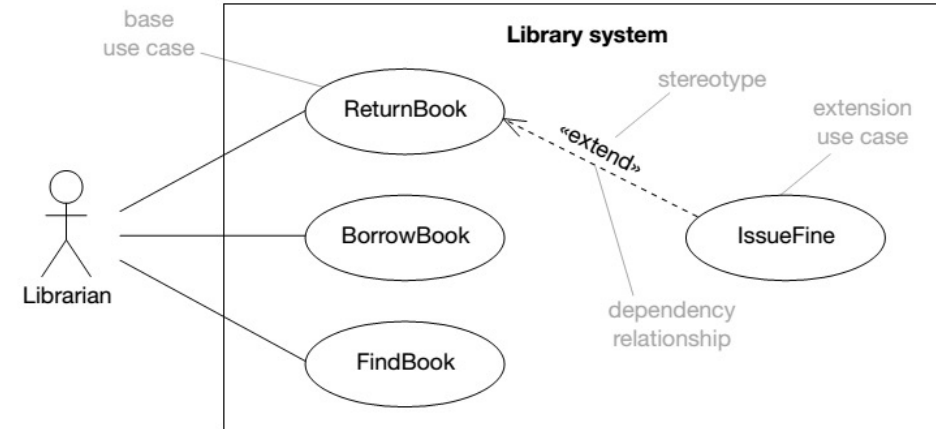
DeleteEmployeeDetails
ID: UC3
Actors: Manager
Preconditions: 1. A valid Manager is logged on to the system.
Flow of events: 1. The Manager enters the employee's ID number. 2. include (FindEmployeeDetails). 3. The system displays the employee details. 4. The Manager deletes the employee details. 5. ...
Postconditions:

<<extend>> Relationship

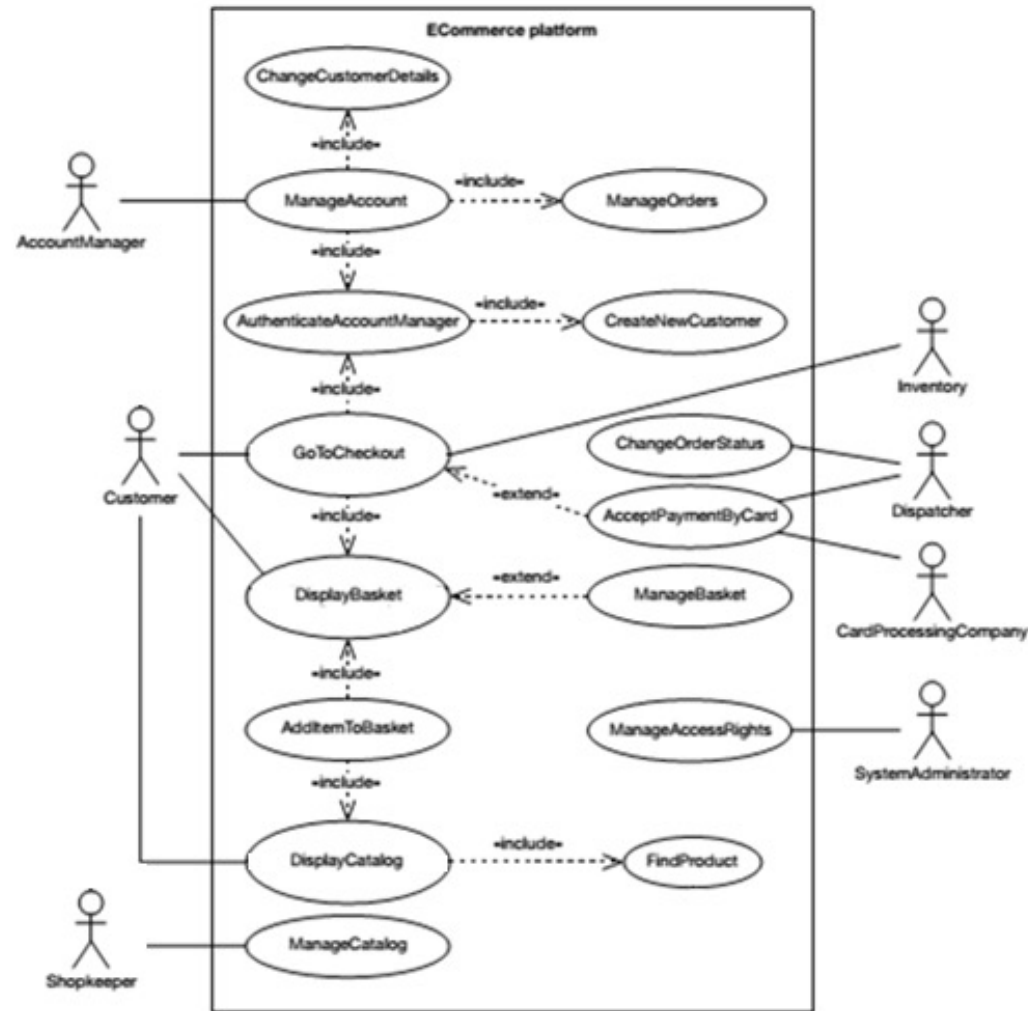
«extend» is a way of inserting new behavior into an existing use case.

	ReturnBook
	ID: UC9
	Actors: Librarian
	Preconditions: 1. A valid Librarian is logged on to the system.
	Flow of events: 1. The Librarian enters the borrower's ID number. 2. The system displays the borrower's details including the list of borrowed books. 3. The Librarian finds the book to be returned in the list of books. 4. The Librarian returns the book. 5. ...
	Postconditions: The book has been returned.

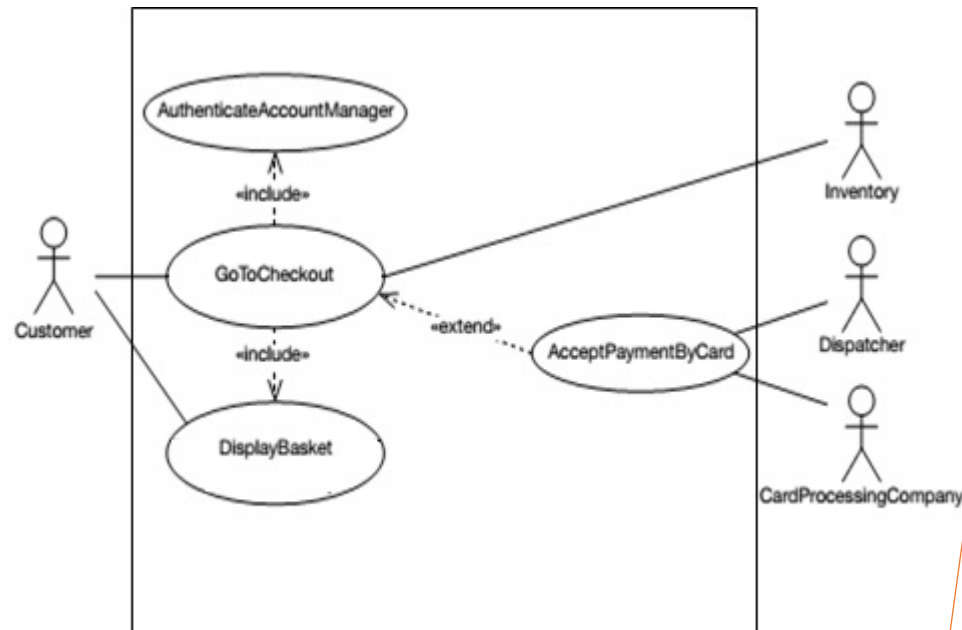
extension point — <overdueBook>



Example: Use Case Diagram for e-Commerce System



Example: Use Case Description for GoToCheckout



Use case: GoToCheckout
ID: UC12
Actors: A2 Customer
Includes: UC3 AuthenticateAccountManager UC7 DisplayBasket
Extension points: <acceptPayment>
Preconditions:
Flow of events: <ol style="list-style-type: none"> 1. The Customer selects "checkout". 2. include(AuthenticateAccountManager) 3. include(DisplayBasket) 4. If the Customer selects "proceed to checkout" <ol style="list-style-type: none"> 4.1 The system communicates with the Inventory actor to determine which parts of the order can be fulfilled immediately. 4.2 If there are any items in the order that can't be dispatched immediately <ol style="list-style-type: none"> 4.2.1 The system informs the Customer that some items are unavailable and that these items have been removed from the order. 4.3 The system presents the final order to the Customer and asks for confirmation to debit the card. 4.4 If the Customer accepts the final order <acceptPayment>
Postconditions:

Note: Extension points for <<extend>> relationship can be stated before preconditions in the use case description and in the flow of events where the extension occurs

Use Case Description for AcceptPaymentByCard and DisplayBasket

Extension use case: AcceptPaymentByCard
ID: UC1
Extends: UC12 GoToCheckout at <acceptPayment>
Insertion segment: <ol style="list-style-type: none">1. The system retrieves the Customer's credit card details.2. The system sends a message to the CardProcessingCompany that includes the merchant ID, merchant authentication, the Customer credit card details, and the amount of the order.3. The CardProcessingCompany validates the transaction.4. If the transaction succeeds<ol style="list-style-type: none">4.1 The Customer is notified that they have been billed.4.2 The Customer is given an order reference number for tracking the order.4.3 The Dispatcher is sent the order.5. If there is not enough credit<ol style="list-style-type: none">5.1 The system informs the Customer that there was not enough credit on the card to process the order.5.2 The Customer is given the option to change to another credit card.6. If there is some other error.<ol style="list-style-type: none">6.1 The Customer is notified that the order can't be processed at this time and to try again later.

Use case: DisplayBasket
ID: UC7
Actors: A2 Customer
Extension points:
Preconditions:
Flow of events: <ol style="list-style-type: none">1. If there are no items in the basket<ol style="list-style-type: none">1.1. The system displays the message "No items in basket yet".1.2. The use case terminates.2. The system displays a list of all of the items in the Customer's shopping basket. This list is a series of lines that include product ID, item name, quantity, item price, and total price.
Alternate flow: At any point the Customer may leave the shopping basket screen.
Postconditions:

Structural model

Class Diagram

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
- Create structural models of a system when discussing and designing the system architecture

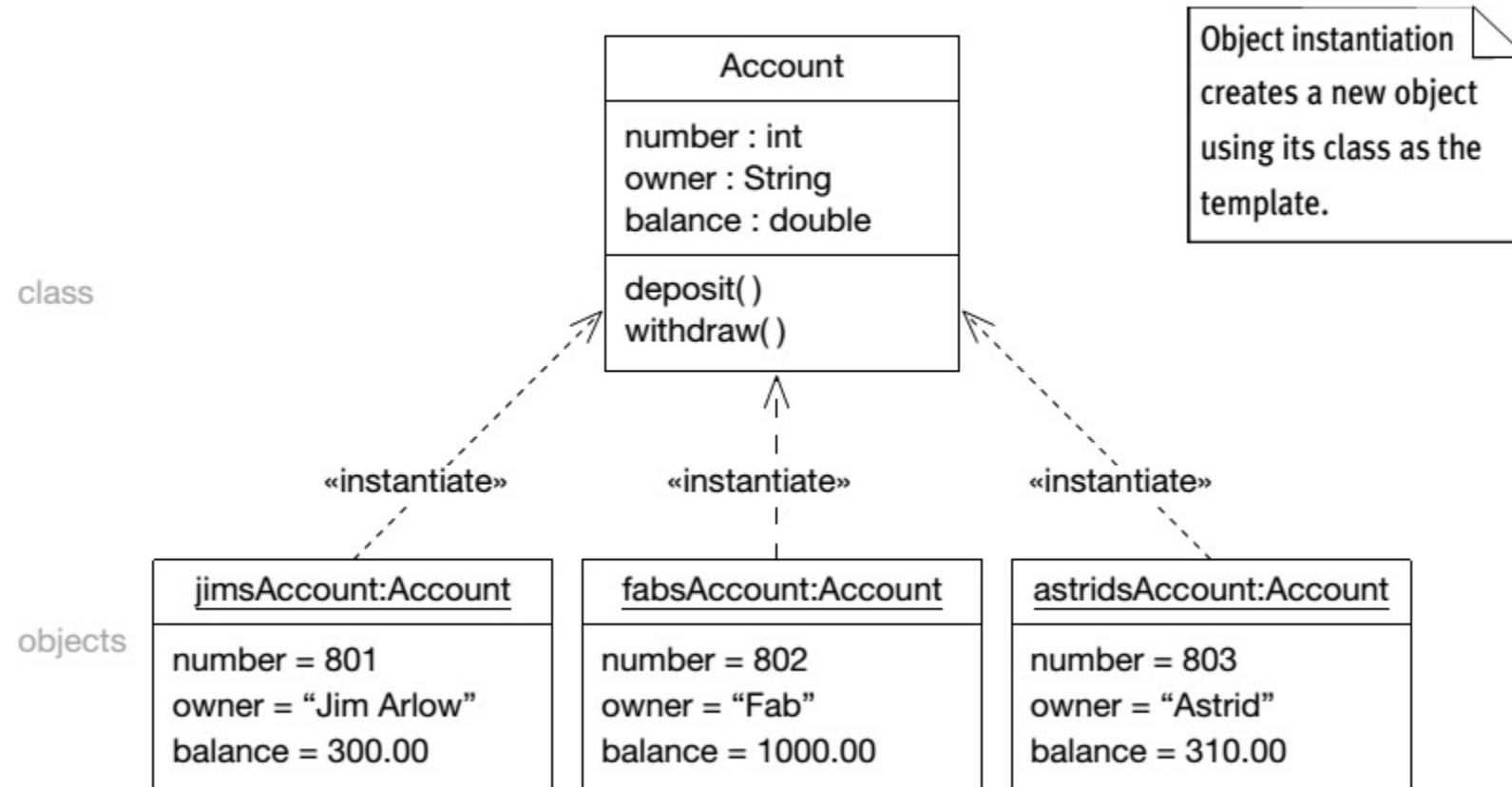
Class Diagrams

- 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 object class 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 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.

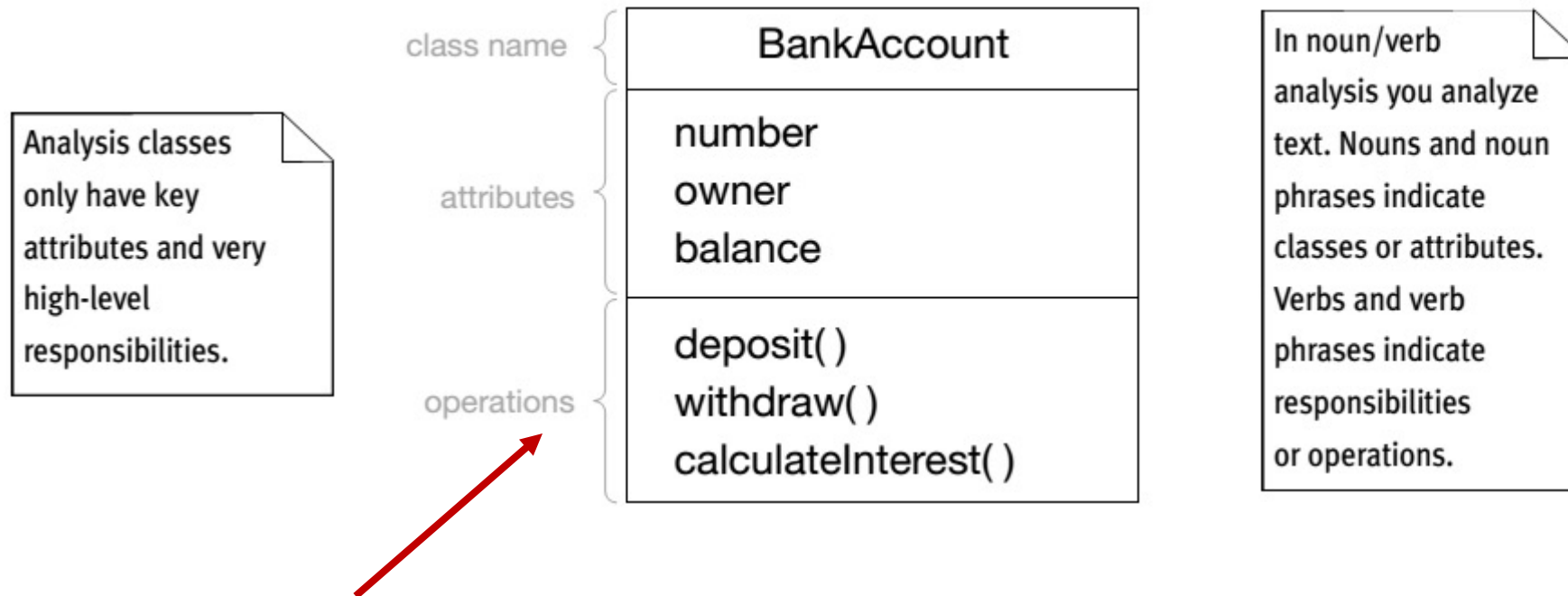
Domain Model Class Diagram Notation

- Class diagram key:
 - General class symbol: rectangle with three sections
 - Sections convey name, attributes, and behaviors
 - **Methods** (behaviors) **not shown** in domain model class diagram
 - Lines connecting rectangles show associations
 - Multiplicity reflected above connecting lines
- Domain class objects reflect business concern, policies, and constraints
- When moving to design, methods and details of methods are included (design class diagram)

Class and Object Instantiation

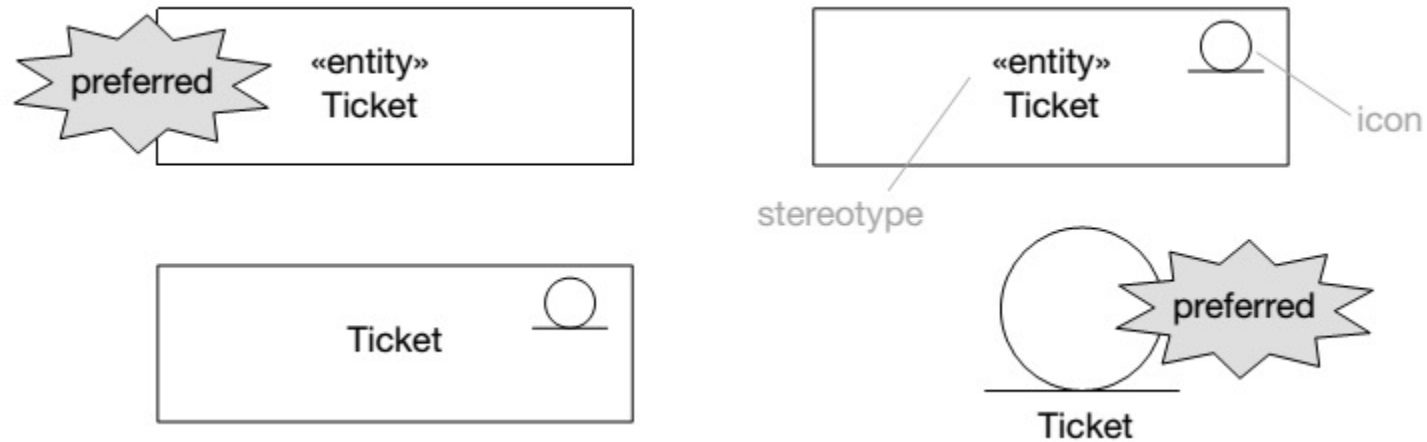


Class as a Template of Objects



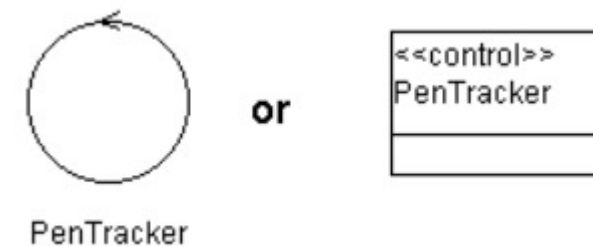
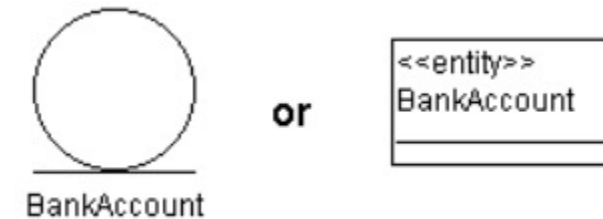
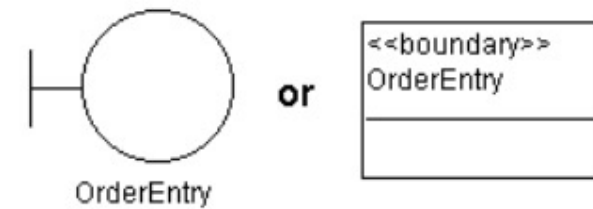
Note: Do not include operation part in analysis stage – only in design stage

Class Stereotype Syntax



Types of Class Stereotype

- **Boundary class**: interacts with actors outside the system and other objects in the system
- **Entity class**: a passive class i.e. does not initiate interactions on its own, store and manage information, normally persistent data
- **Control class**: coordinate boundary and entity object(s) – collect inform from boundary and dispatch to entity, usually has behavior specific for one use case

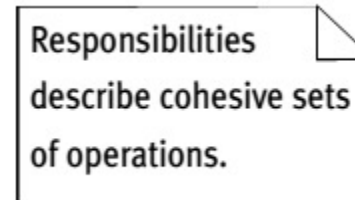


Source: https://www.ibm.com/support/knowledgecenter/en/SS6RBX_11.4.2/com.ibm.sa.oomethod.doc/topics

Good Analysis Class

What makes a good analysis class:

- its name reflects its intent
- it is a crisp abstraction that models one specific element of the problem domain
- it maps on to a clearly identifiable feature of the problem domain
- it has a small, well-defined set of responsibilities
- it has high cohesion
- it has low coupling



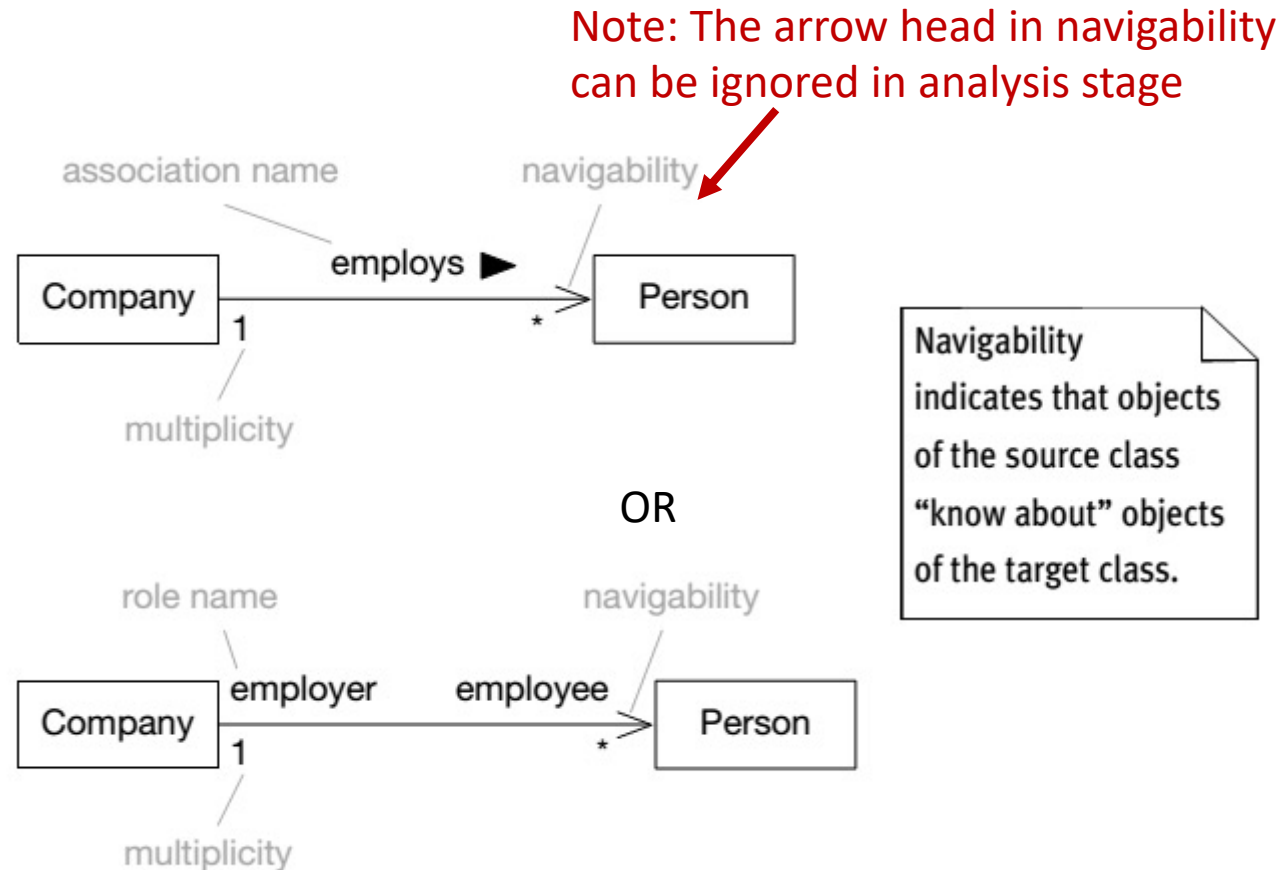
Responsibilities
describe cohesive sets
of operations.

Note: Our SE course focuses on domain models. Thus, the concepts of high cohesion and low coupling will be elaborated in design stage where we include operations in class diagram.

Association

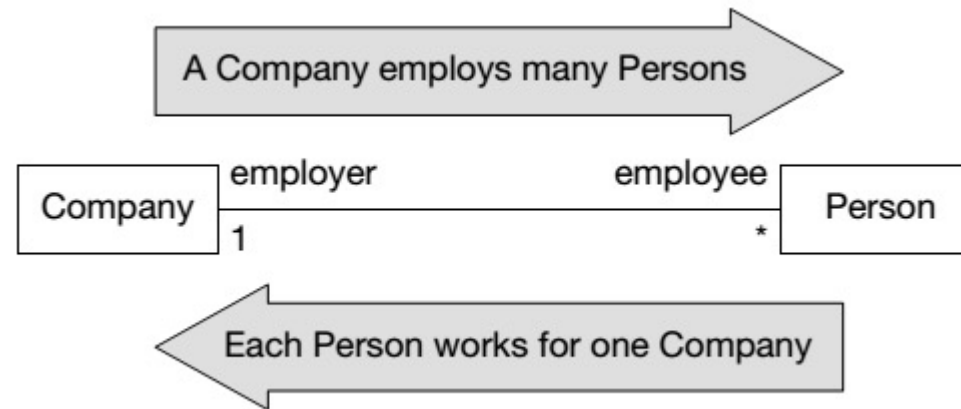
Association names are verb phrases that indicate the semantics of the association.

Role names are noun phrases that indicate the roles played by objects linked by instances of the association.



Multiplicity

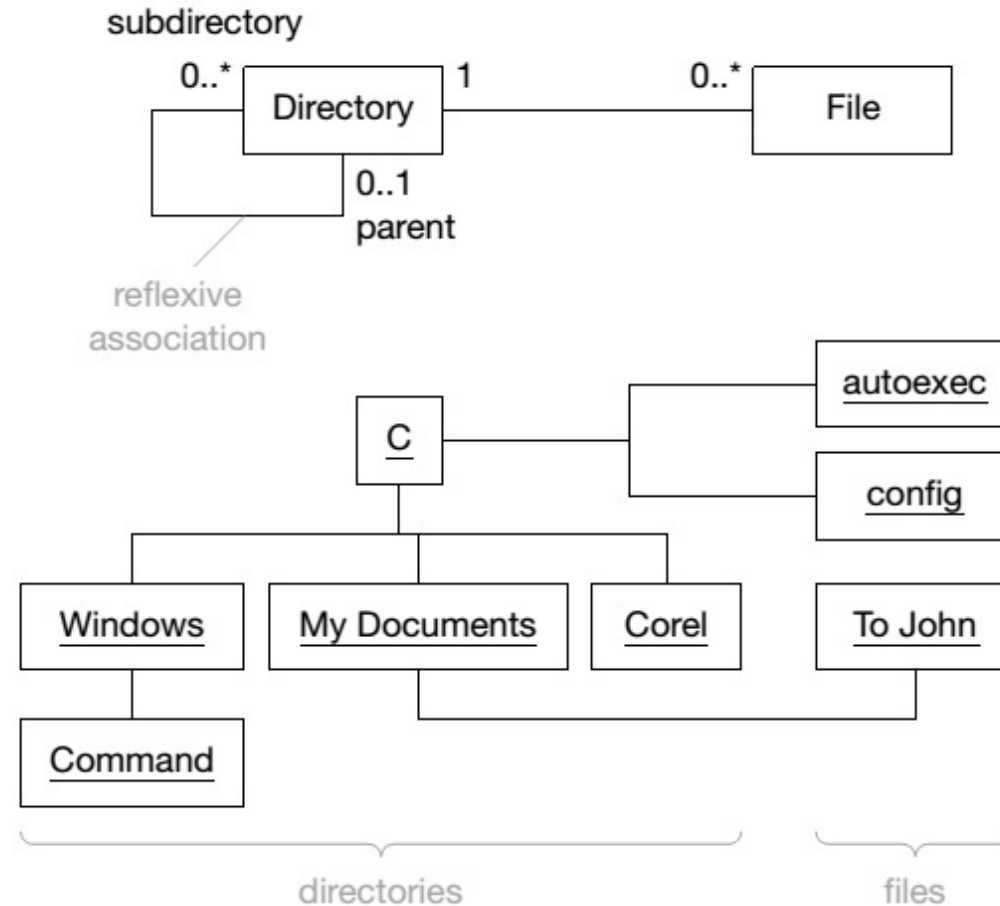
Multiplicity specifies the number of objects that can participate in a relationship at any point in time.



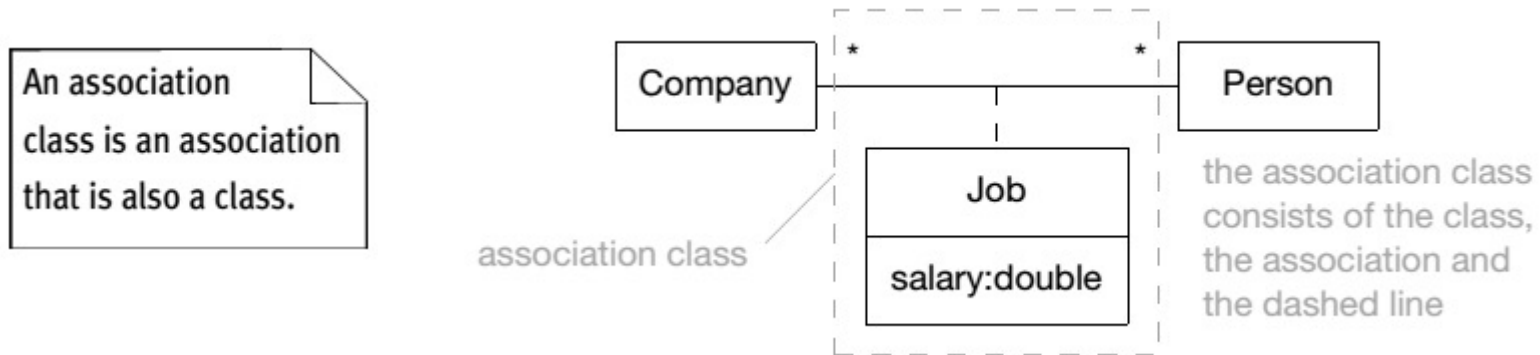
Adornment	Semantics
0..1	Zero or 1
1	Exactly 1
0..*	Zero or more
*	Zero or more
1..*	1 or more
1..6	1 to 6
1..3,7..10,15, 19..*	1 to 3 <i>or</i> 7 to 10 <i>or</i> 15 exactly <i>or</i> 19 to many

Reflexive Association

When class has an association to itself, it is a reflexive association.



Association Class



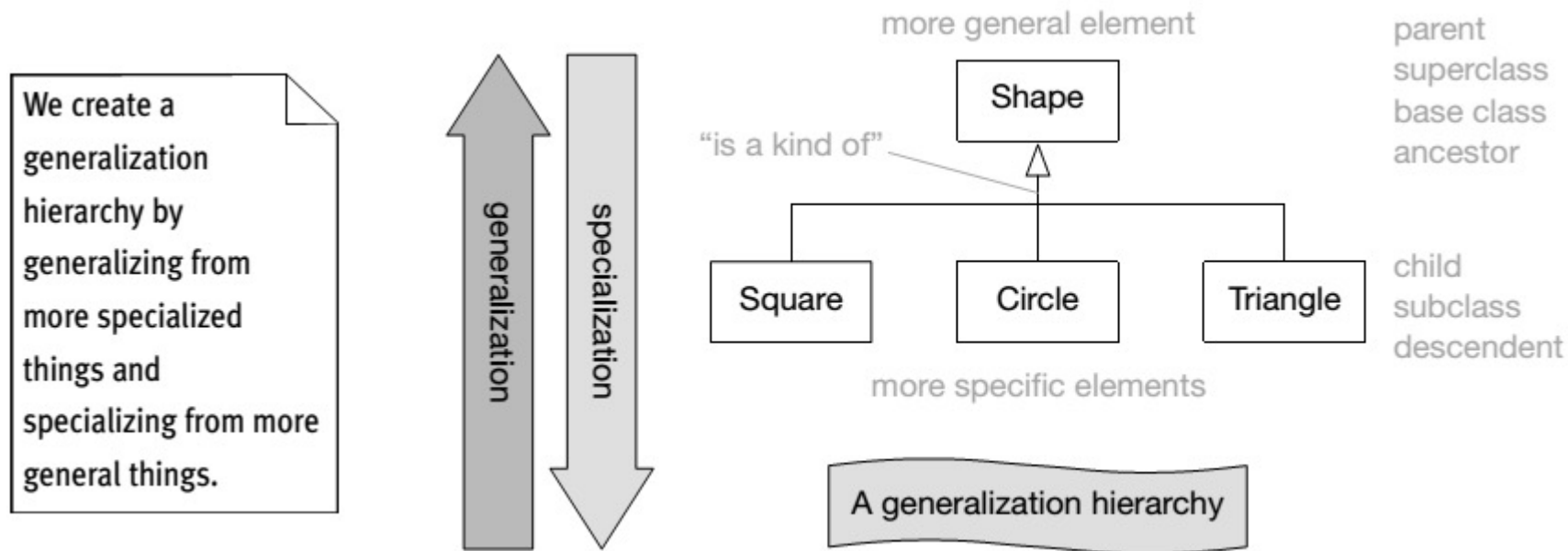
Generalization...

- Generalization is an everyday technique that can be used to **manage complexity**
- Rather than learning the detailed characteristics of every entity, place these entities in more **general classes** (animals, cars, houses, etc.) and learn the characteristics of these classes
- This allows us to infer that different members of these classes have some **common characteristics** e.g. square, circle and triangle are shape

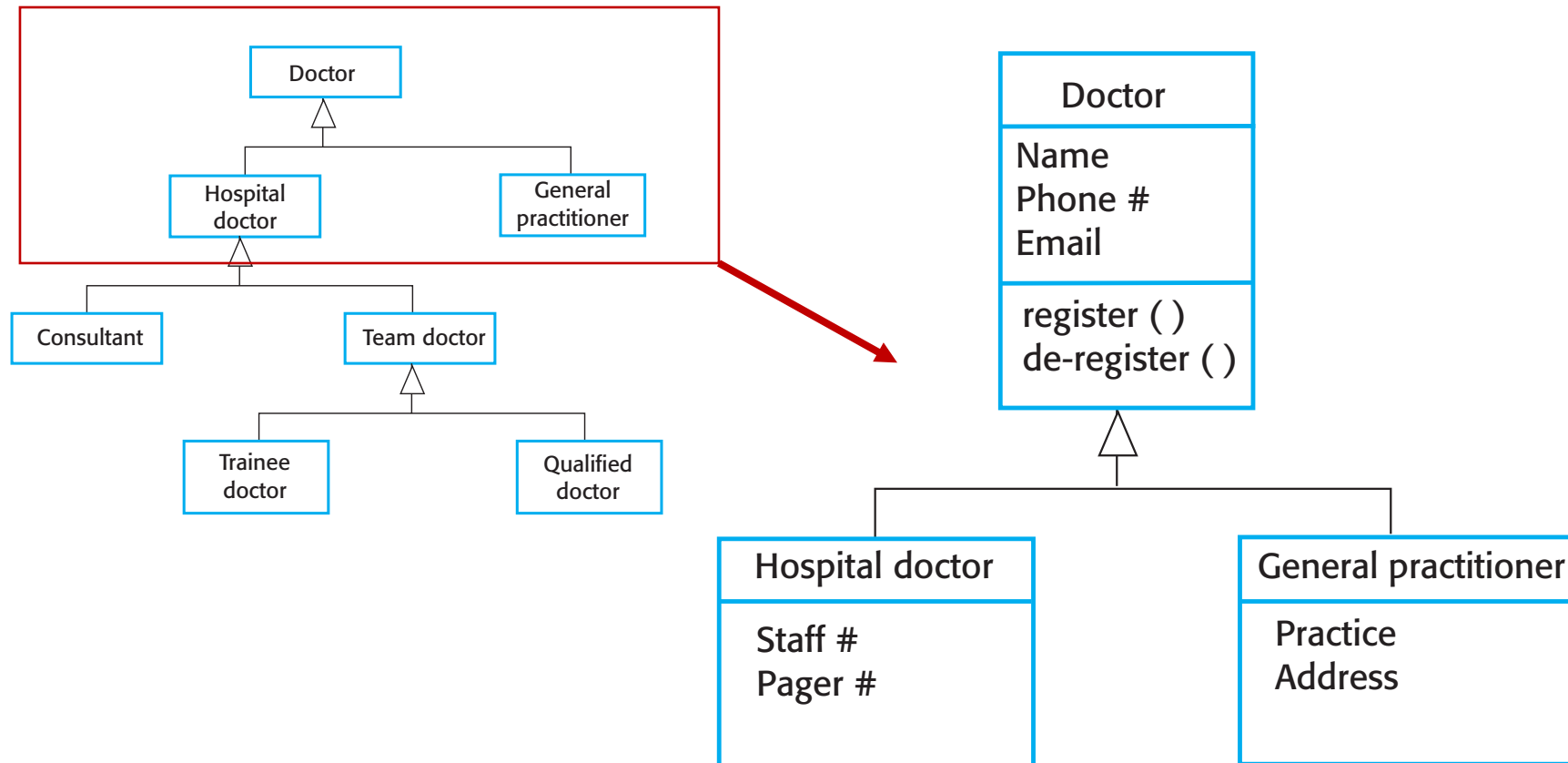
Generalization

- In modeling systems, it is often useful to **examine the classes** in a system to see if there is **scope for generalization**
- Do not have to look at all classes in the system to see if they are affected by the change in future
- In a generalization, the **attributes and operations** associated with higher-level classes are also associated with the lower-level classes
- The lower-level classes are **subclasses** inherit the attributes and operations from their **superclasses**
- These lower-level classes then **add more specific attributes and operations**

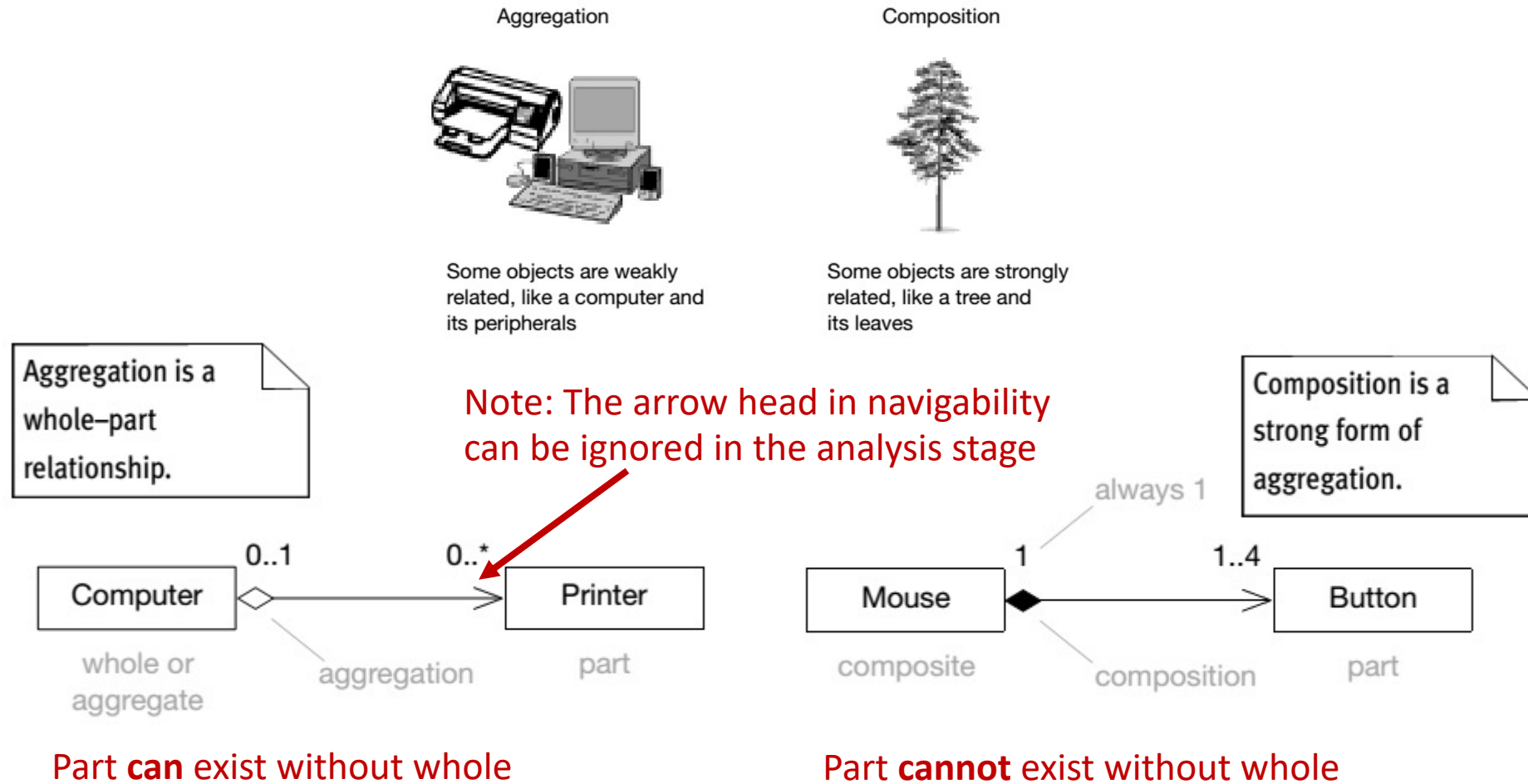
Class Generalization



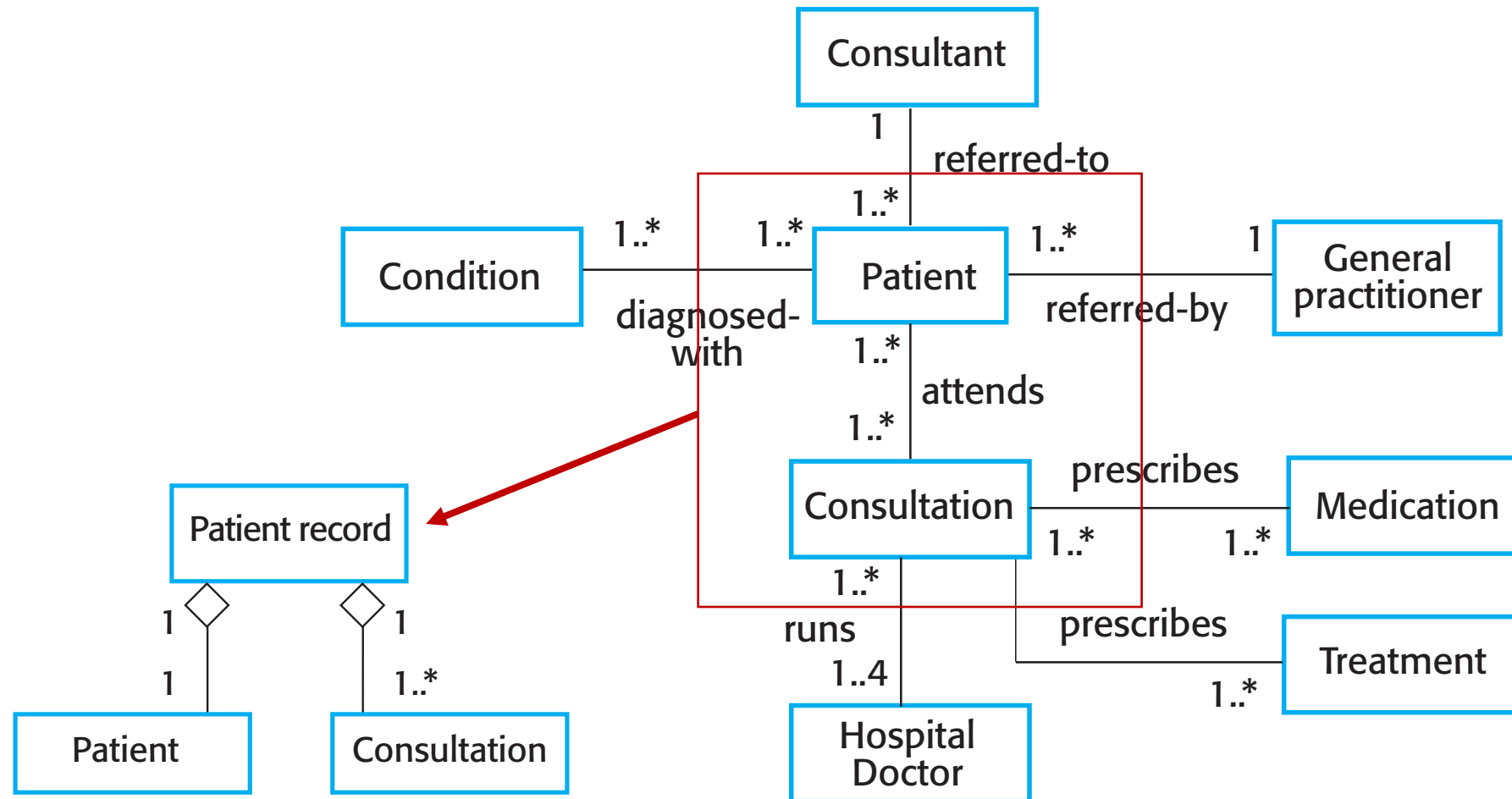
Example: Generalization Hierarchy



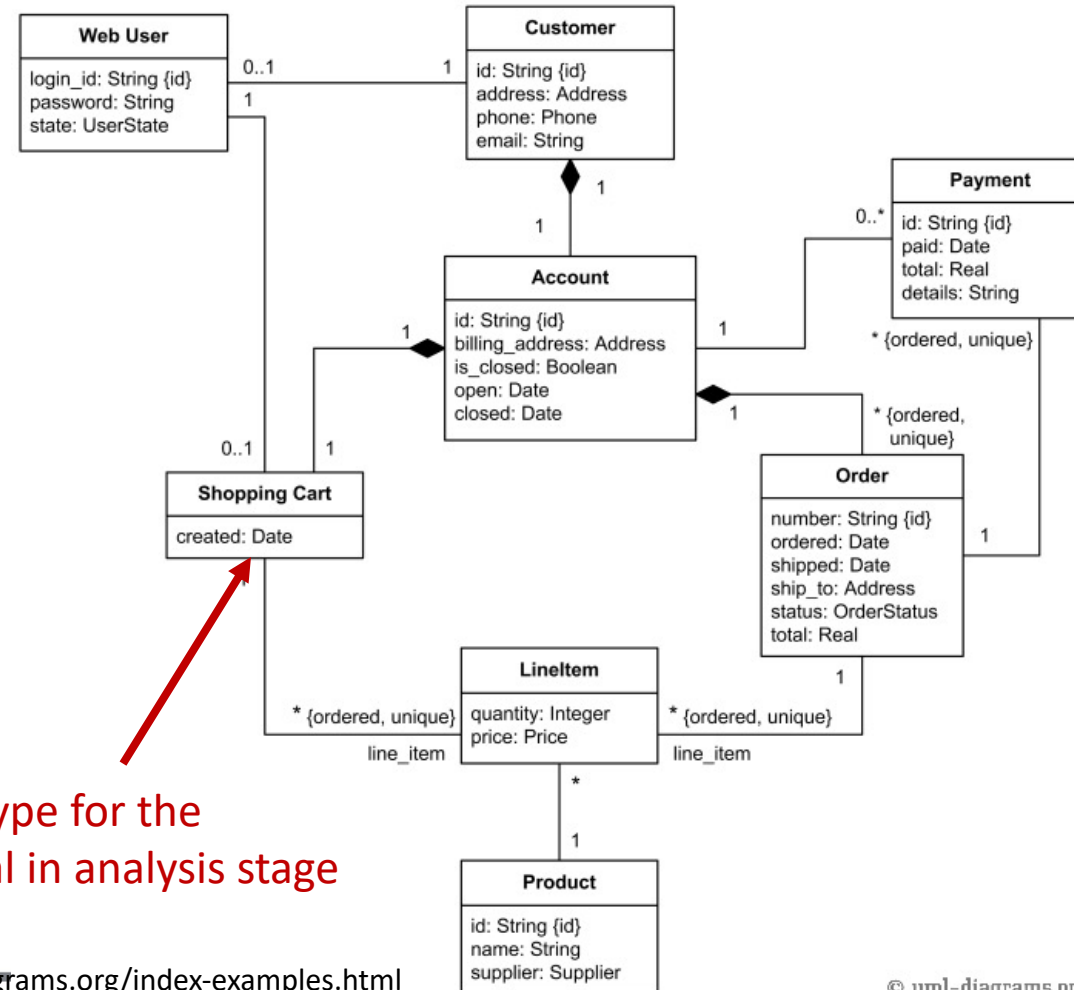
Aggregation and Composition



Example: Classes and Associations in MHC-PMS



Example: Domain Model for Online Shopping



Note: Stating the type for the attribute is optional in analysis stage

source: <https://www.uml-diagrams.org/index-examples.html>

© uml-diagrams.org

Activity Diagram, Sequence Diagram, State Diagram (State Chart/Machine)

Behavioral model

Behavioral Model

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

Data-Driven Modeling

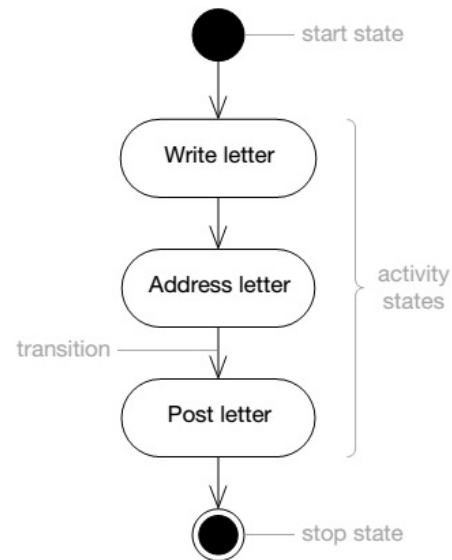
- Many business systems are **data-processing systems** that are primarily driven by data, controlled by the data input to the system, with relatively little external event processing
- Data-driven models show the sequence of actions involved in processing **input data** and generating an **associated output**
- They are particularly useful during the **analysis of requirements** as they can be used to show end-to-end processing in a system

Activity Diagram

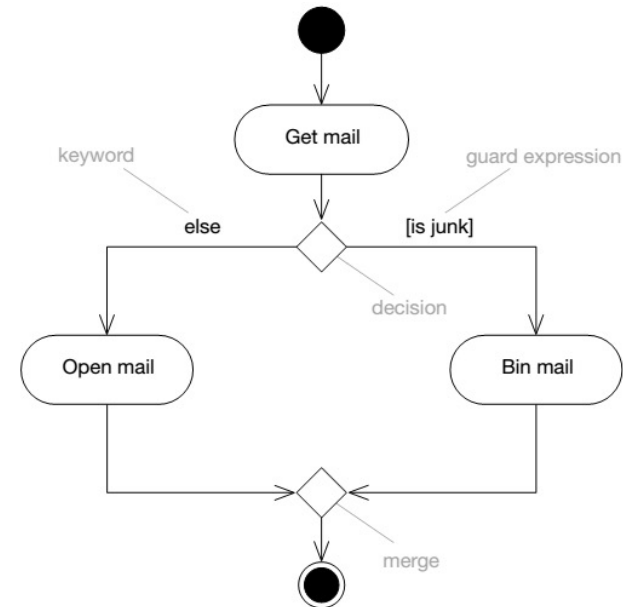
- Activity diagrams are “**OO flowcharts**” that allow us to model a process as a collection of **activities and transitions** between those activities
- Activity diagrams are really just **special cases of statecharts** where every state has an entry action that specifies some process or function that occurs when the state is entered
- An activity diagram **can be attached to any modeling element** for the purpose of modeling the behavior of that element **at different abstraction levels**
- Activity diagrams are typically attached to: use cases, classes, interfaces, components, nodes, collaborations, operations and methods

Activity Diagram: Transition and Decision

Transitions indicate the movement from one state to another.



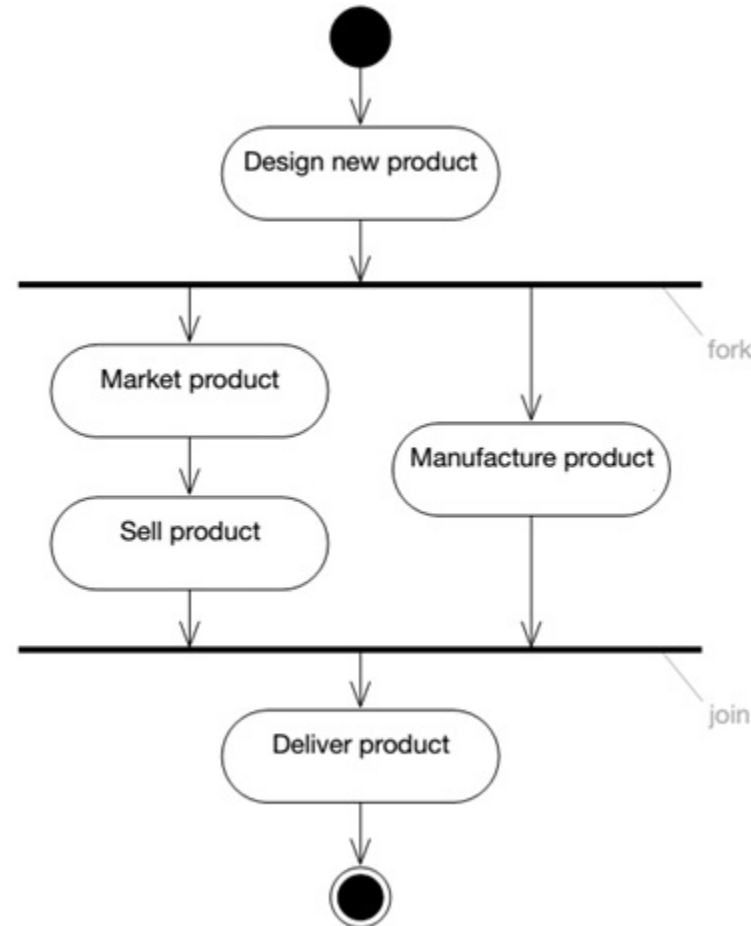
Decisions allow you to model decision points.



Activity Diagram: Fork and Join

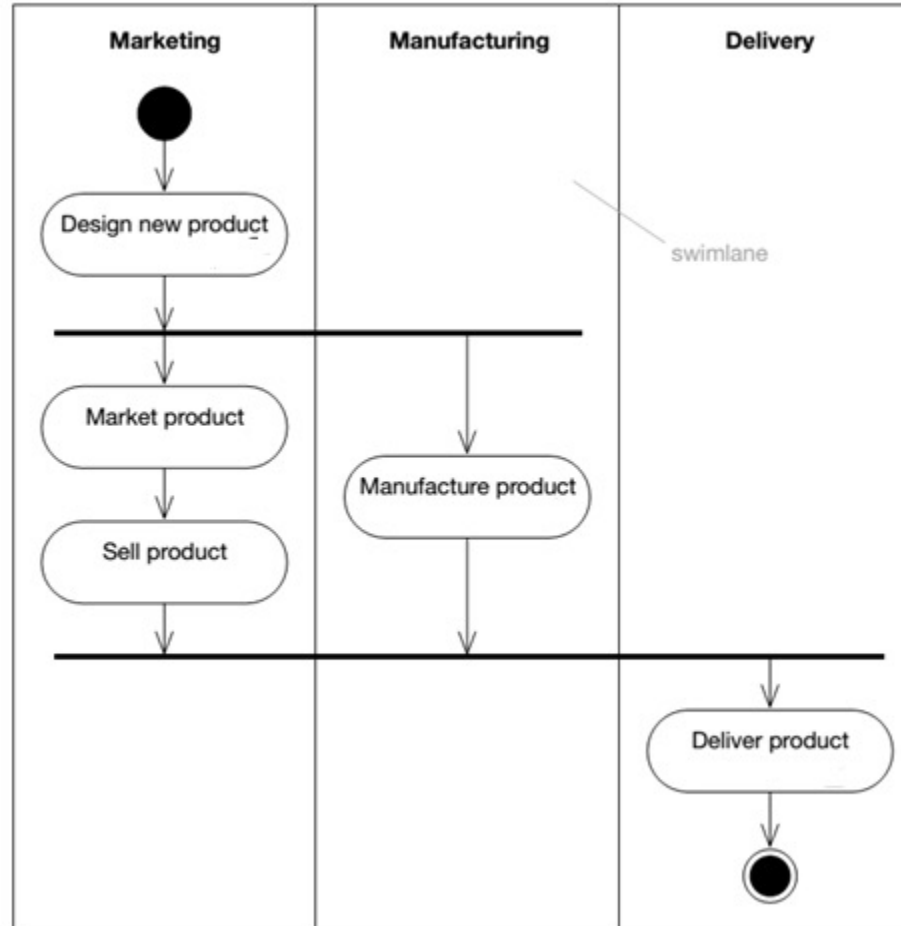
Forks split a path into two or more concurrent flows.

Joins synchronize two or more concurrent flows.



Activity Diagram: Swimlanes

Swimlanes allow us to partition activity diagrams.

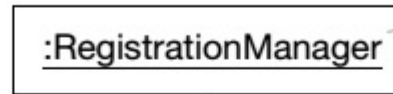


Sequence Diagram

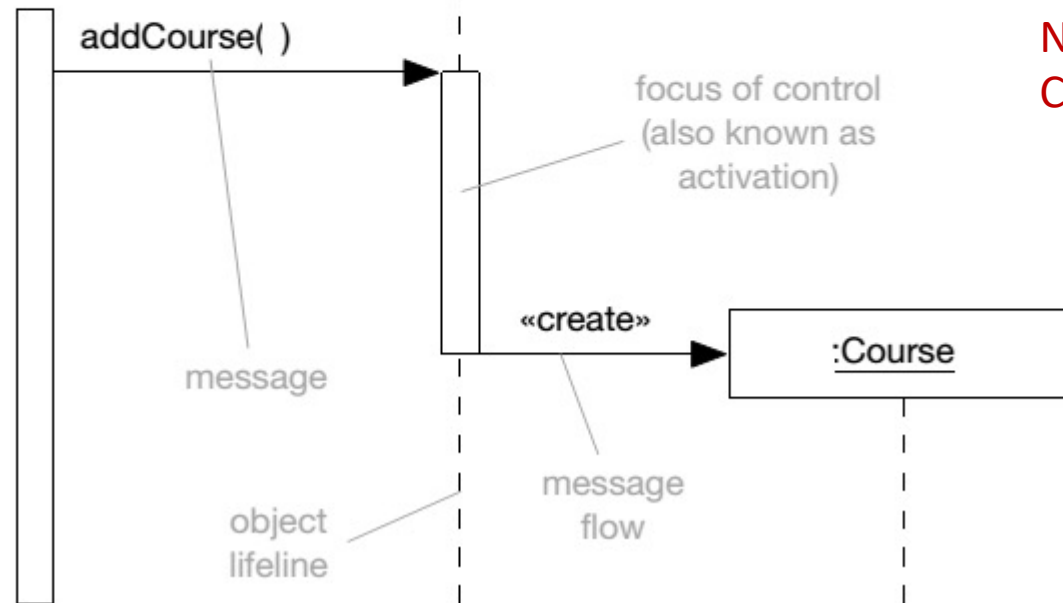
- Sequence diagrams are part of the UML and are used to model the **interactions** between the actors and the objects showing their **behavior** within a system
- A sequence diagram shows the **sequence of interactions** that take place during a particular use case or use case instance (**scenario**)
- 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 Syntax

Sequence diagrams show object interactions arranged in a time sequence.



instance or classifier role



Use case: AddCourse
ID: UC8
Actors: Registrar
Preconditions: The Registrar has logged on to the system.
Flow of events: 1. The Registrar selects "add course". 2. The system accepts the name of the new course. 3. The system creates the new course.
Postconditions: A new course has been added to the system.

Note: Related Use Case Description

Syntax/Notation Details

- **Lifeline** represents an individual participant (or object) in the interaction
- **Focus of control (activation)** specifies a behavior or interaction within the lifeline also known as execution specification
- **Message** defines a particular **communication** between lifelines of an interaction
- Examples of communication: raising a signal, invoking an operation, creating or destroying an instance

Types of Message

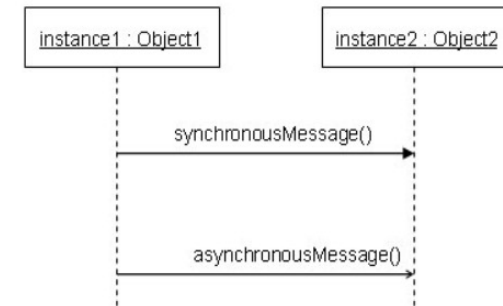
- **Synchronous:**
 - If a caller sends a **synchronous message** (in sequence), it must wait until the message is done before it proceeds with its business
- **Asynchronous:**
 - If a caller sends an **asynchronous message**, it can continue processing and does not have to wait for a response
 - Usually used in **multithreaded applications** and in message-oriented middleware
- **Return message:**
 - Return messages are an **optional** part of a sequence diagram
 - The use of return messages depends on the level of detail/abstraction that is being modeled
 - Return messages are useful if finer detail is required; otherwise, the invocation message is sufficient



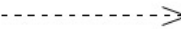
(Source: <https://www.ibm.com/developerworks/rational/library/3101.html>)

Message in Sequence Diagram

- To show an object (lifeline) sending a message to another object, draw a line to the receiving object with a **solid arrowhead** (if a **synchronous** call operation) or with a **stick arrowhead** (if an **asynchronous** signal).
- Besides message calls, there are **return messages**. These return messages are **optional**; a return message is drawn as a dotted line with an open arrowhead back to the originating lifeline, and above this dotted line you place the return value from the operation.

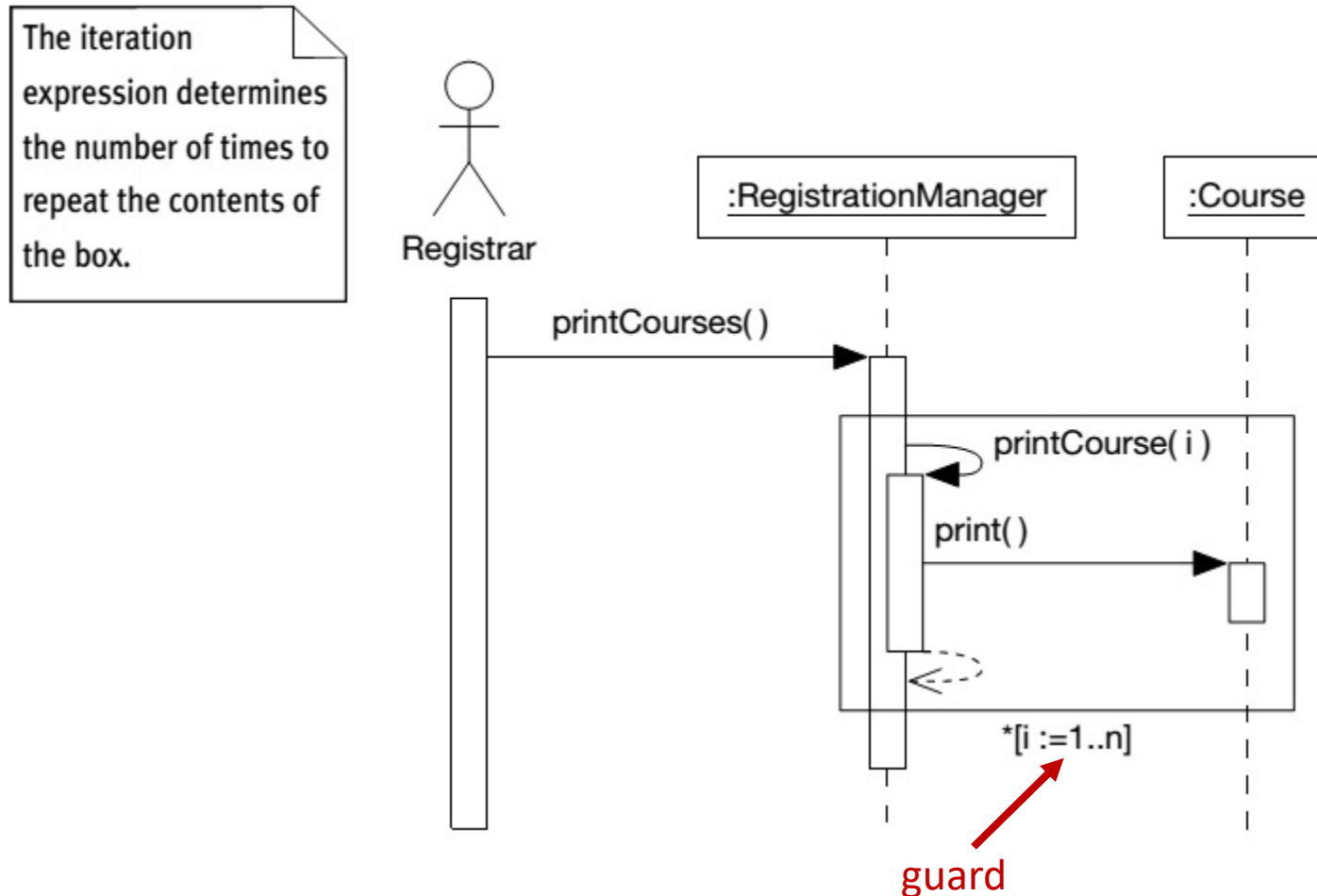
Source: <https://www.ibm.com/developerworks/rational/library/3101.html>



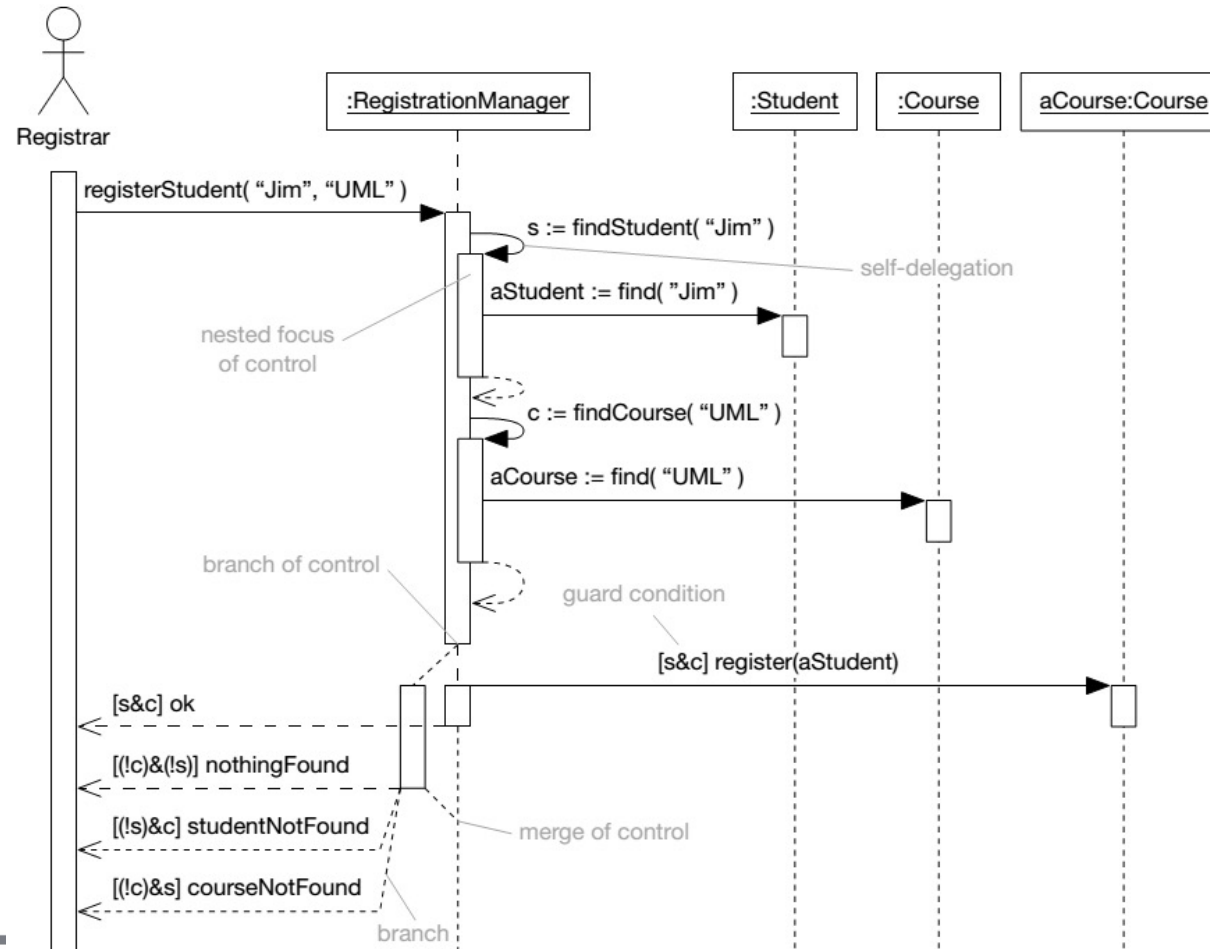
Message flow	Semantics
	Procedure call – the sender waits until the receiver has finished This is the most common option
	Asynchronous communication – the sender carries on as soon as the message has been sent; it does not wait for the receiver This is often used when there is concurrency
	Return from a procedure call – the return is always implicit in a procedure call, but it may be explicitly shown using this arrow

Source: Arlow and Neustadt (2002)

Iteration in Sequence Diagram

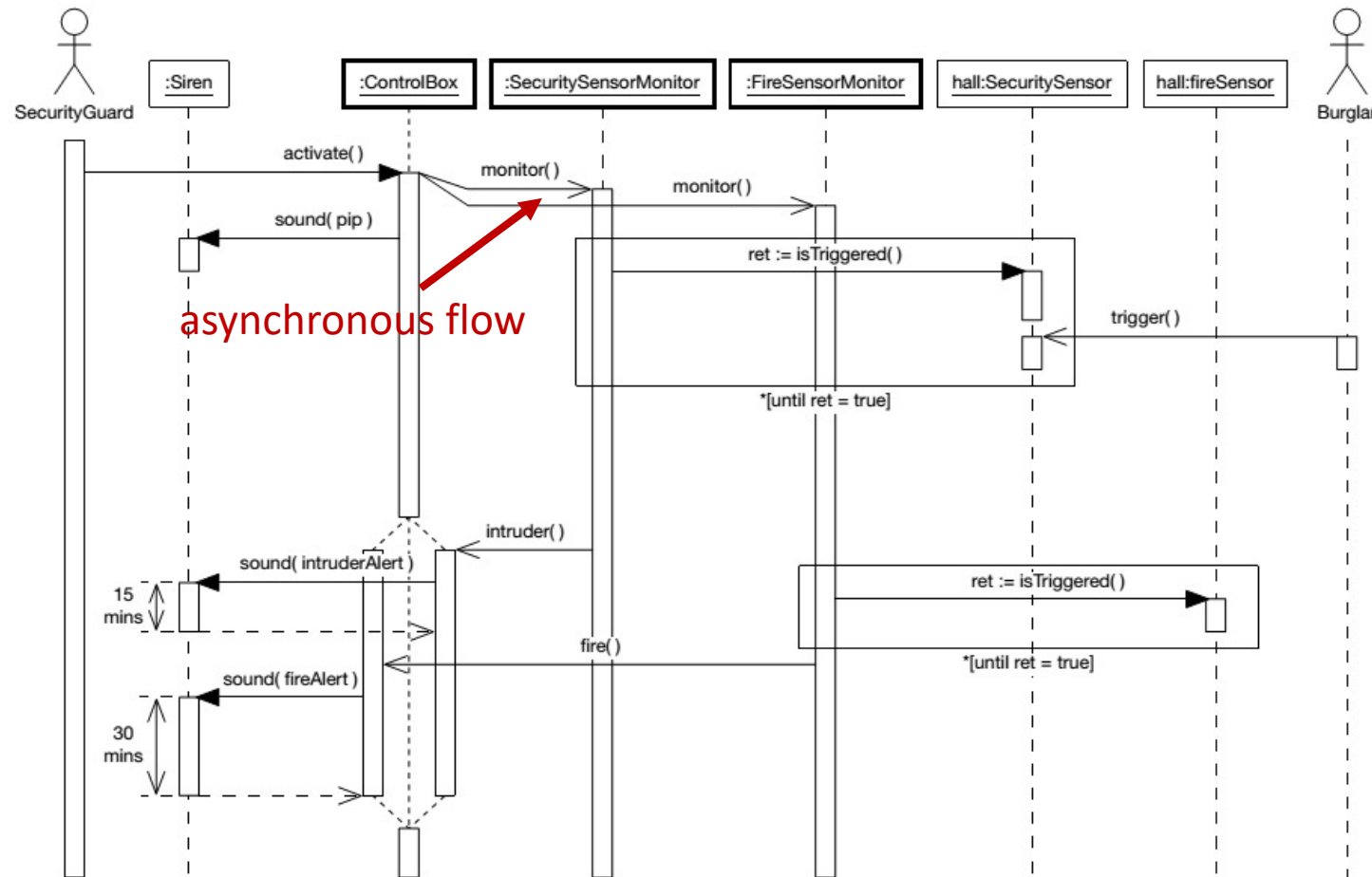


Branching and Self-Delegation

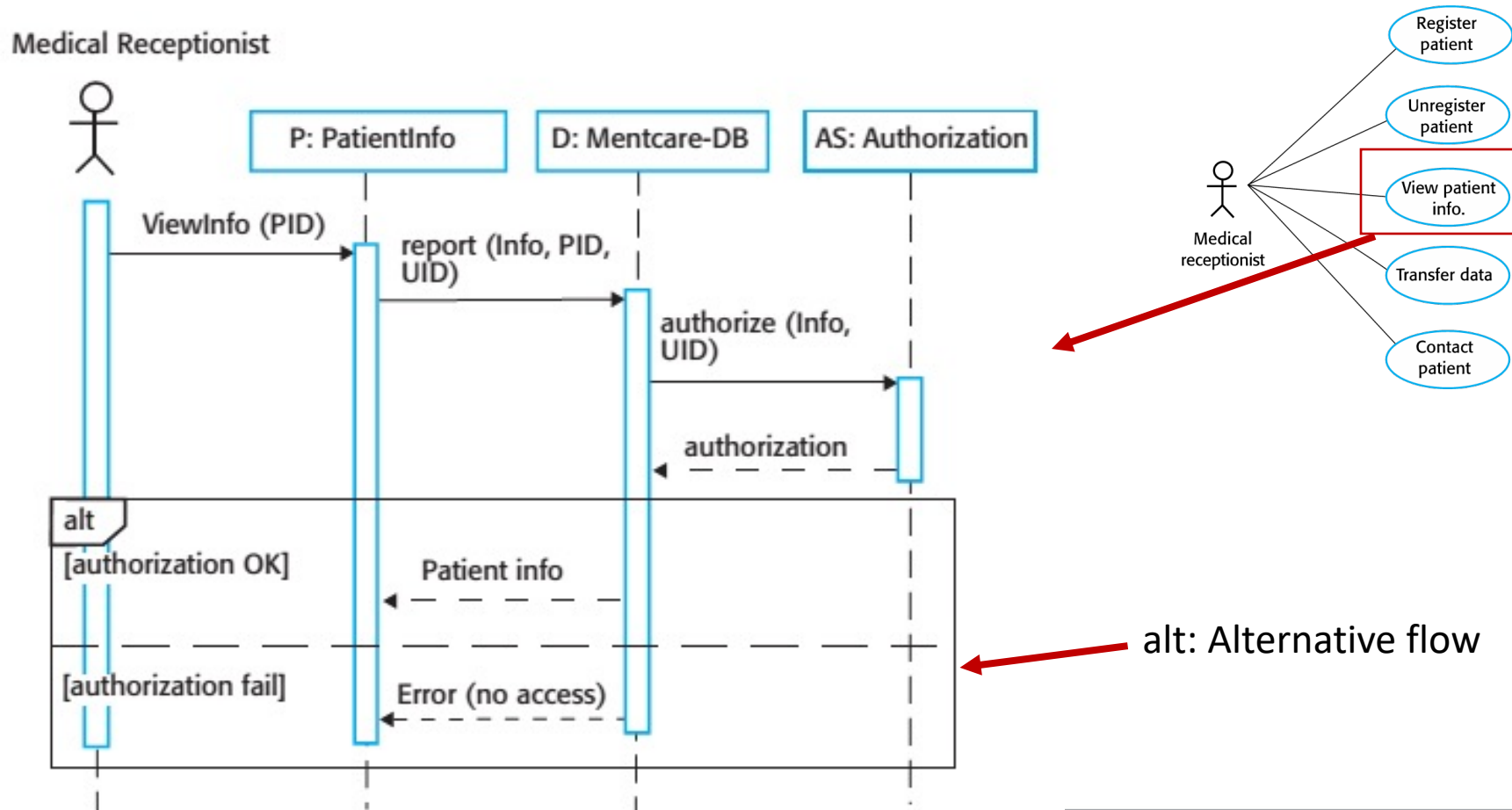


Sequence diagrams
are very good at
showing branching.

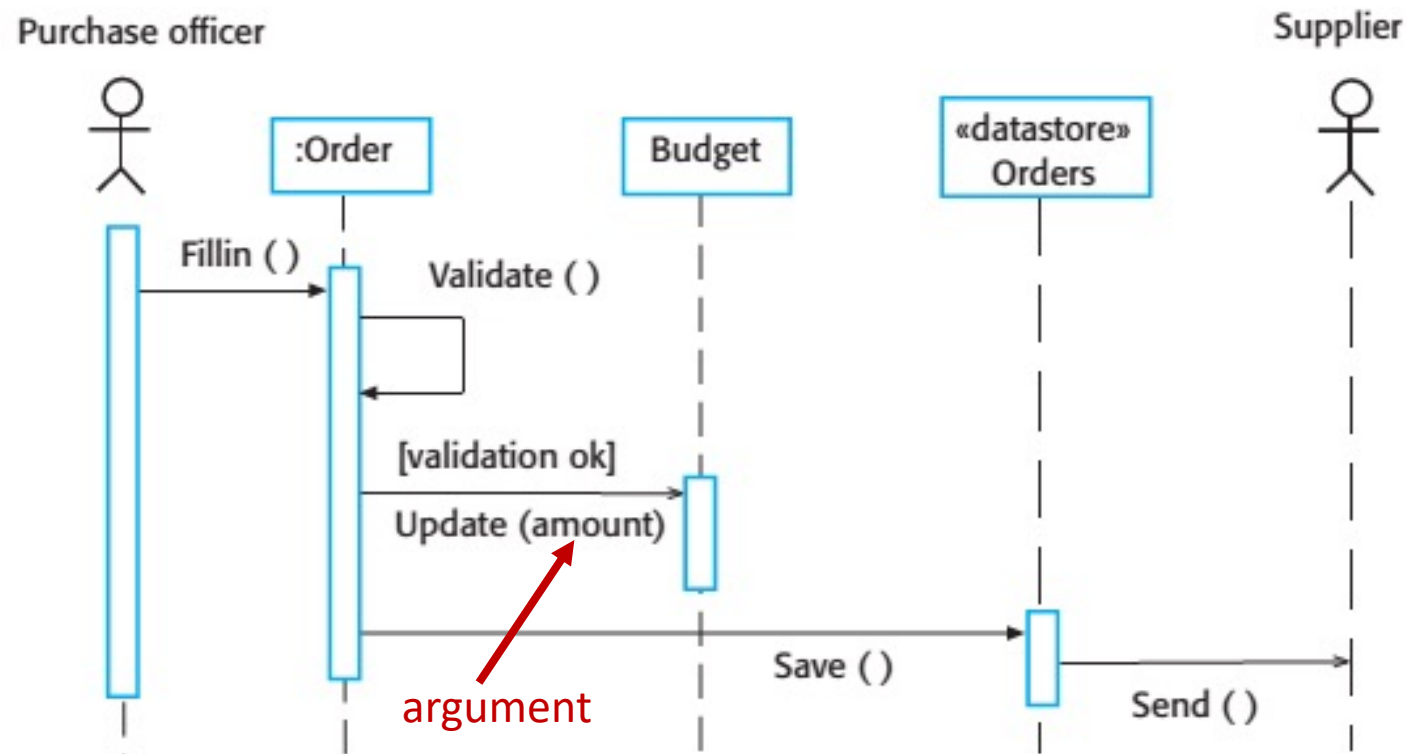
Concurrency: Active Objects



Example: Sequence Diagram for View Patient Info in MHC-PMS

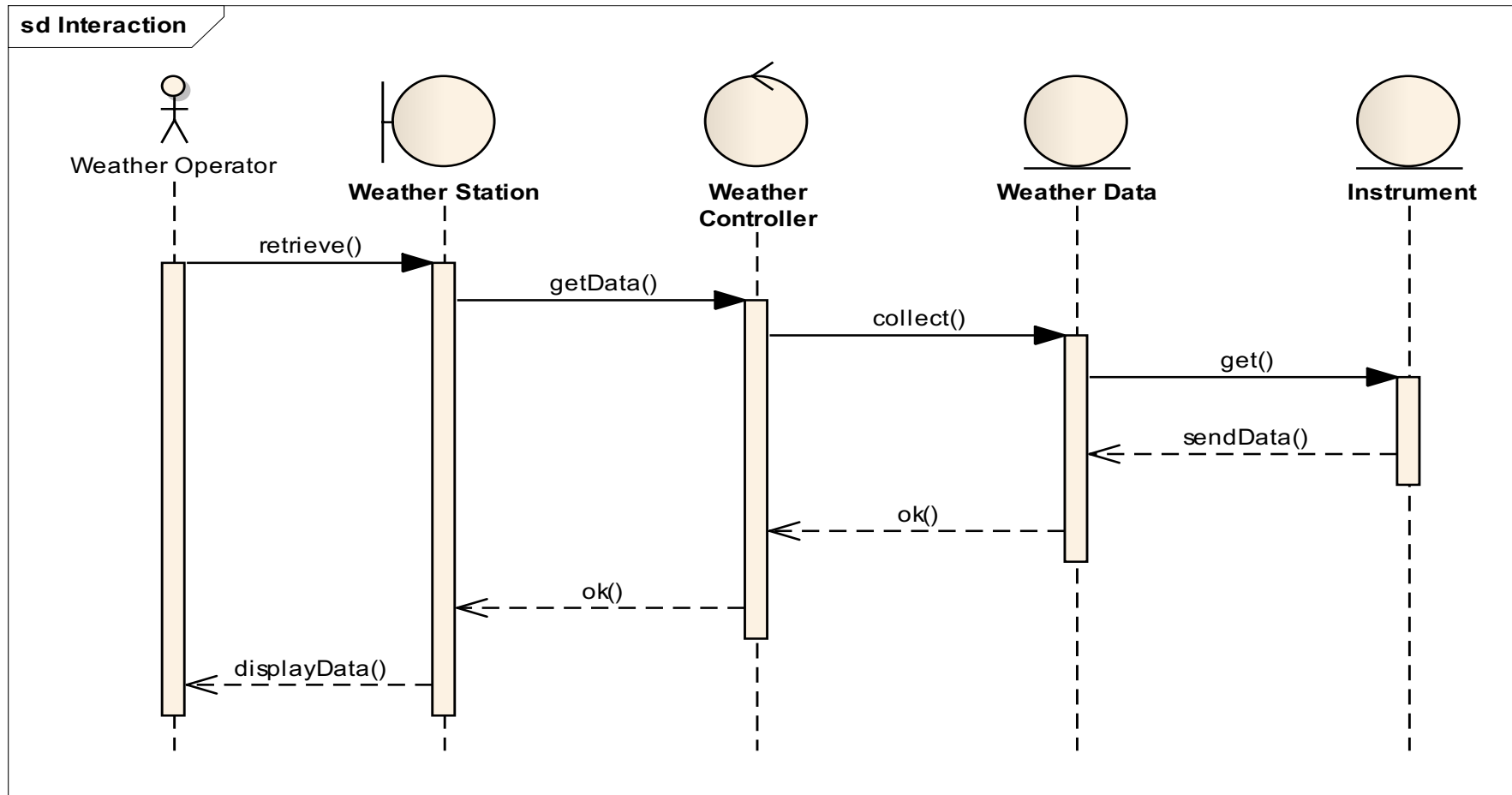


Example: Sequence Diagram for Process Order



Note: Sequence diagram also shows the interaction (interaction model) besides behaviour – in the analysis stage, the argument can be included (optional)

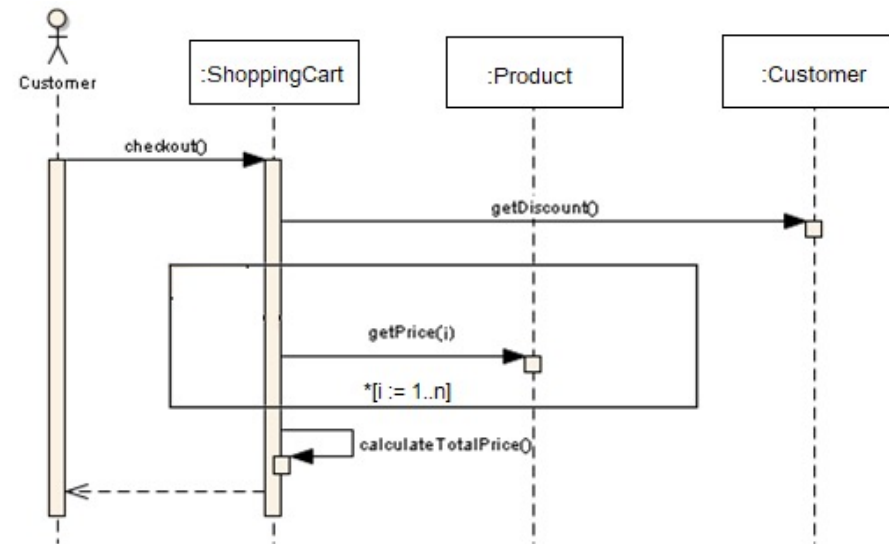
Example: Sequence Diagram Using Class Stereotypes



Note: Sequence diagram with return messages (optional) using class stereotypes (optional)

Sequence Diagram: Scope for Analysis

- Each use case must have at least one sequence diagram that comprises normal flow and alternative flows – split only if it is cluttered to combine all
- Refer to use case description when creating a sequence diagram
- Focus on object interaction in domain model (class diagram for analysis) for a particular use case
- Not necessary to include boundary, controller and data layer that could be added in design stage



Event-Driven Modeling

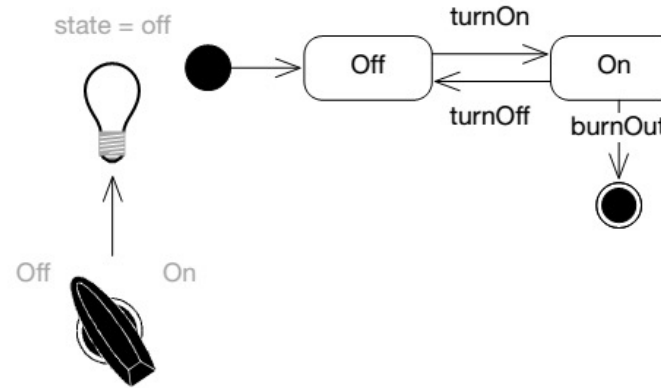
- Some systems mainly real-time systems are often event-driven, with minimal data processing
- Examples: (Source: <https://www.bbconsult.co.uk/blog/event-driven-applications>)
 - A robot has reached its destination (real-time warehouse management)
 - An illegal trade pattern has been detected (fraud detection)
 - A loan application has been accepted/rejected (commercial business)
- Event-driven modeling shows how a system responds to **external and internal events**; is based on the assumption that a system has a finite number of states and the **events (stimuli) may cause a transition from one state to another**

State Chart or State Machine Diagram

- In OO approaches, **state chart is drawn for a single class** to show the lifetime behavior of a single object
- State chart is also known as state machine diagram
- State: a **condition** during the life of an object when it satisfies some conditions, performs some actions, or waits for an event
 - It is found by examining the attributes and links defined for the object
 - represented as a rectangle with rounded corners
- Transition: represents a **change** of the internal condition/state of an object

Event and State

Events cause transitions between states.



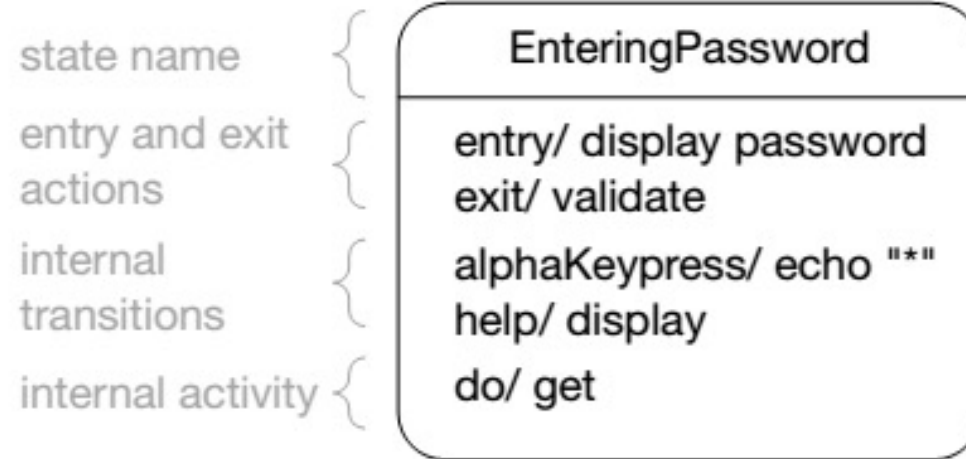
A state is a semantically significant condition of an object.

```
class Color
{
    int red;
    int green;
    int blue;
}
```

State Syntax

Actions are
instantaneous and
uninterruptible.

Activities take
finite time and are
interruptible.



Action syntax: event trigger/ action

Activity syntax: do/ activity

State

- State: a semantically **significant condition of an object**
- Object state is determined by object **attribute** values, **relationships** to other objects, **activities** the object is performing
- State syntax:
 - entry action: performed immediately **on entry** to the state
 - exit action: performed immediately **on exit** from the state
 - internal transitions: these are caused by **events that are not significant enough to warrant a transition to a new state**, the event is processed by an internal transition within the state
 - internal activity: a piece of **work that takes a finite amount of time** and which may be interrupted

Identifying the Object Behavior

- In brief, a state in a state chart similar to **status condition**:
 - Spans many business events
 - Developed for complex problem domain classes
- State chart/state machine diagram
 - Composed of ovals representing status of object
 - Arrows represent transitions

Identify States

- Guidelines to help identify states:
 - Check naming convention for status conditions
 - **Simple states** reflect simple conditions such as “On”
 - **Complex states** labeled with gerunds or verb phrases
 - Example: “Being shipped”
 - **Active states** usually not labeled with nouns
 - Describe only states of being of the object itself
 - Status conditions reported to management/customers
 - Example: “Shipped”

Transition

Transitions show movement between states.

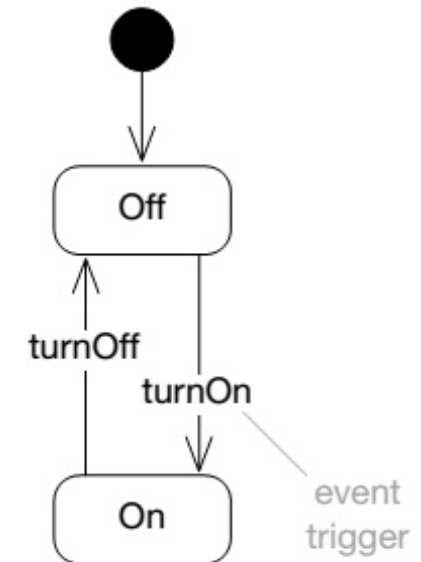


- Transitions have a simple syntax that may be used for external transitions (shown by an arrow) or internal transitions (nested within a state).
- Every transition has **three optional elements**:
 - an **event** – this is an external or internal occurrence that triggers the transition
 - a **guard condition** – this is a Boolean expression that must evaluate to true before the transition can occur
 - an **action** – this is a piece of work associated with the transition, and occurs when the transition fires

Event

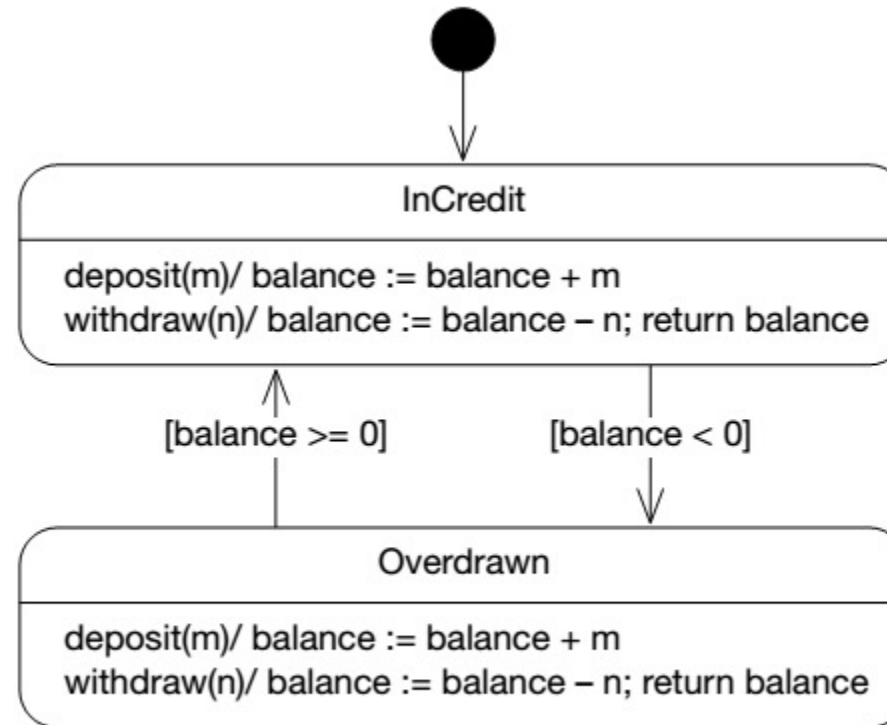
- UML defines an event as, “the specification of a noteworthy occurrence that has location in time and space.”
- Events trigger transitions in state machines
- Events may be shown externally on transitions
- There are **four types of event**, each of which has different semantics:
 1. call event
 2. signal event
 3. change event
 4. time event

Events trigger transitions.



Example: Call Event State Chart

A call event causes a method to be executed.

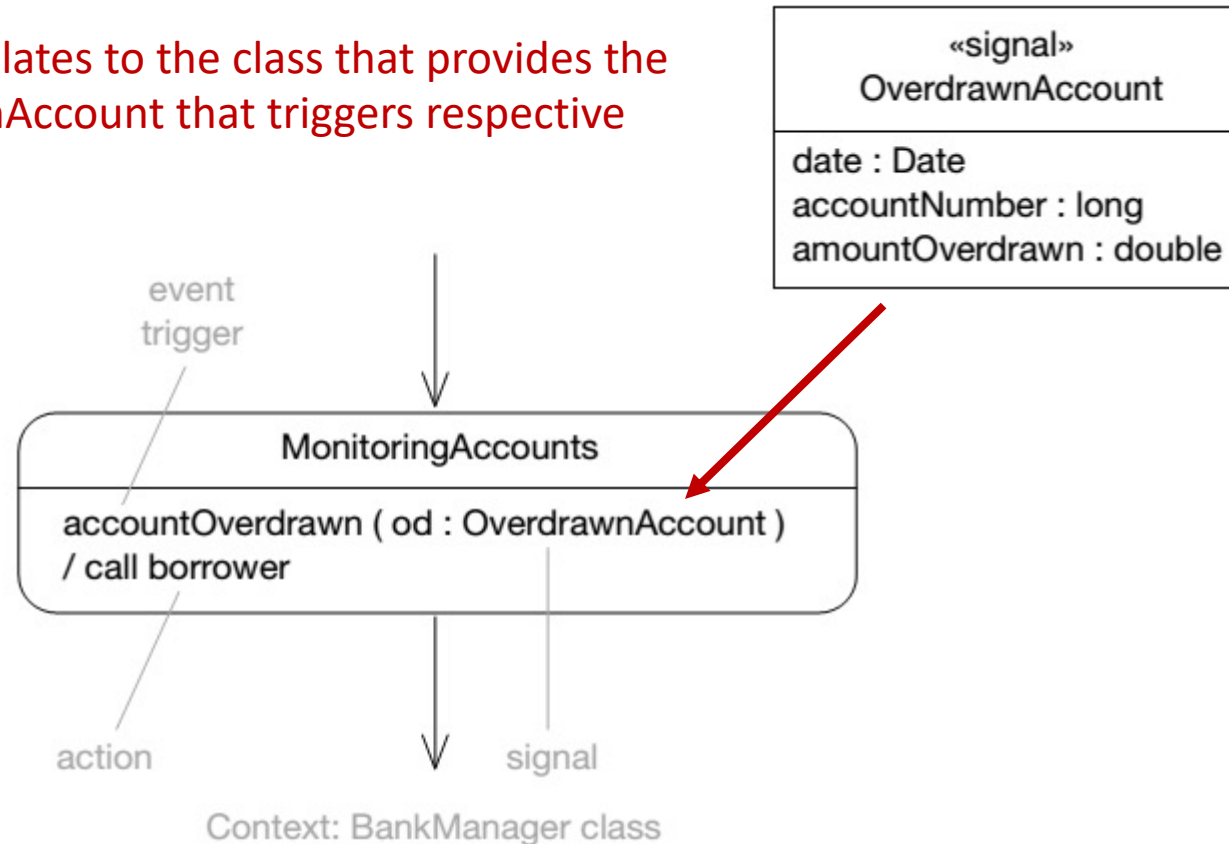


Context: BankAccount class

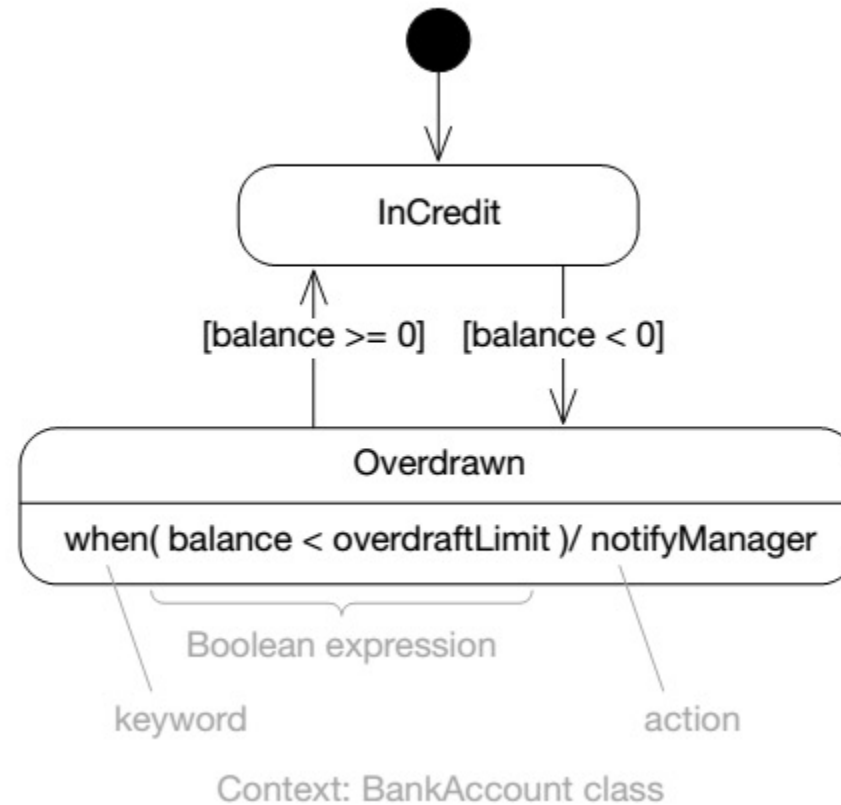
Example: Signal Event State Chart

Note: A state chart relates to the class that provides the signal e.g. OverdrawnAccount that triggers respective event and action

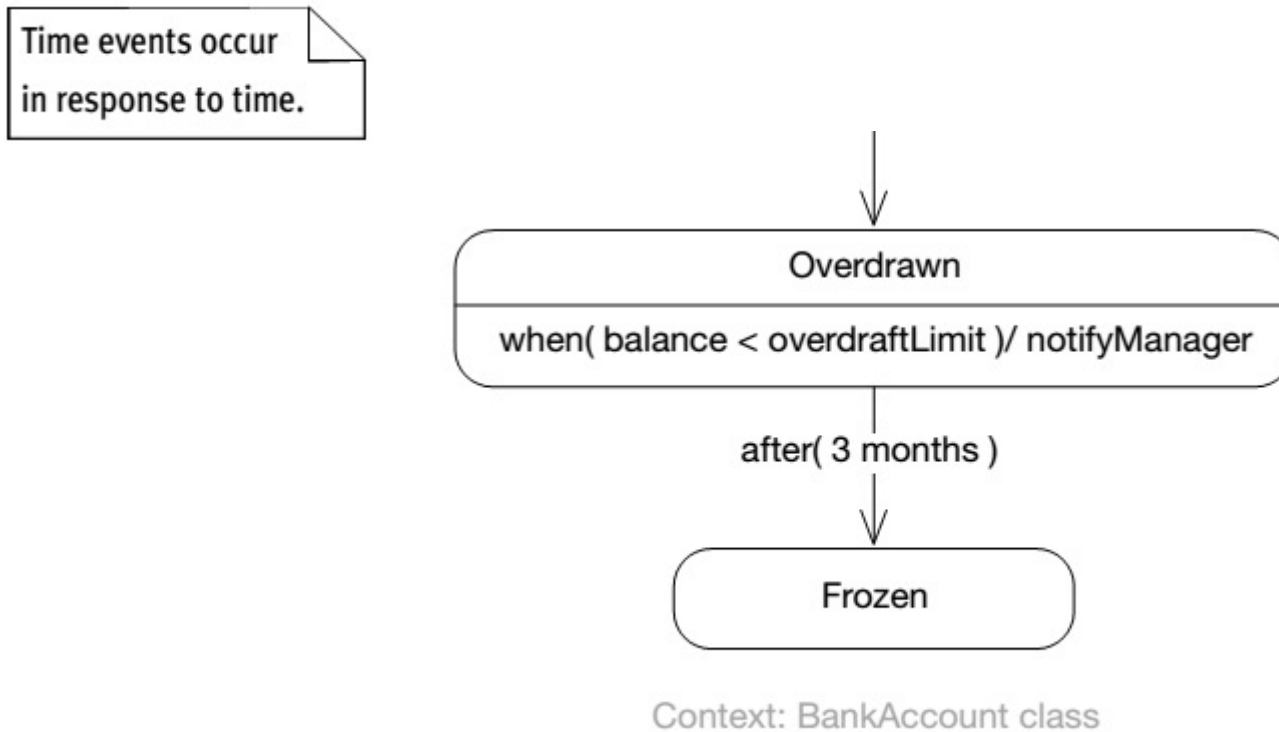
A signal is a one-way asynchronous communication between objects.



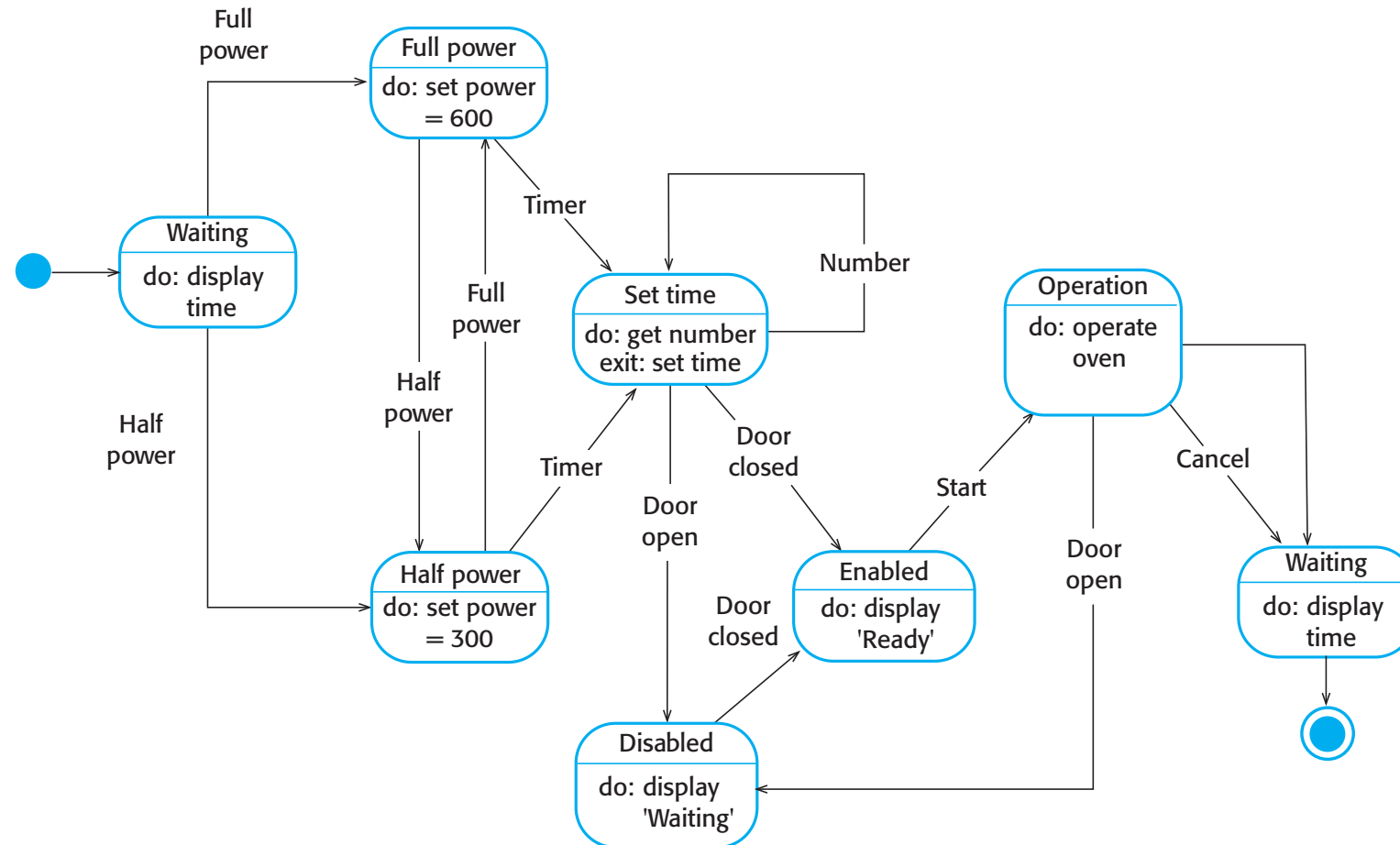
Example: Change Event State Chart



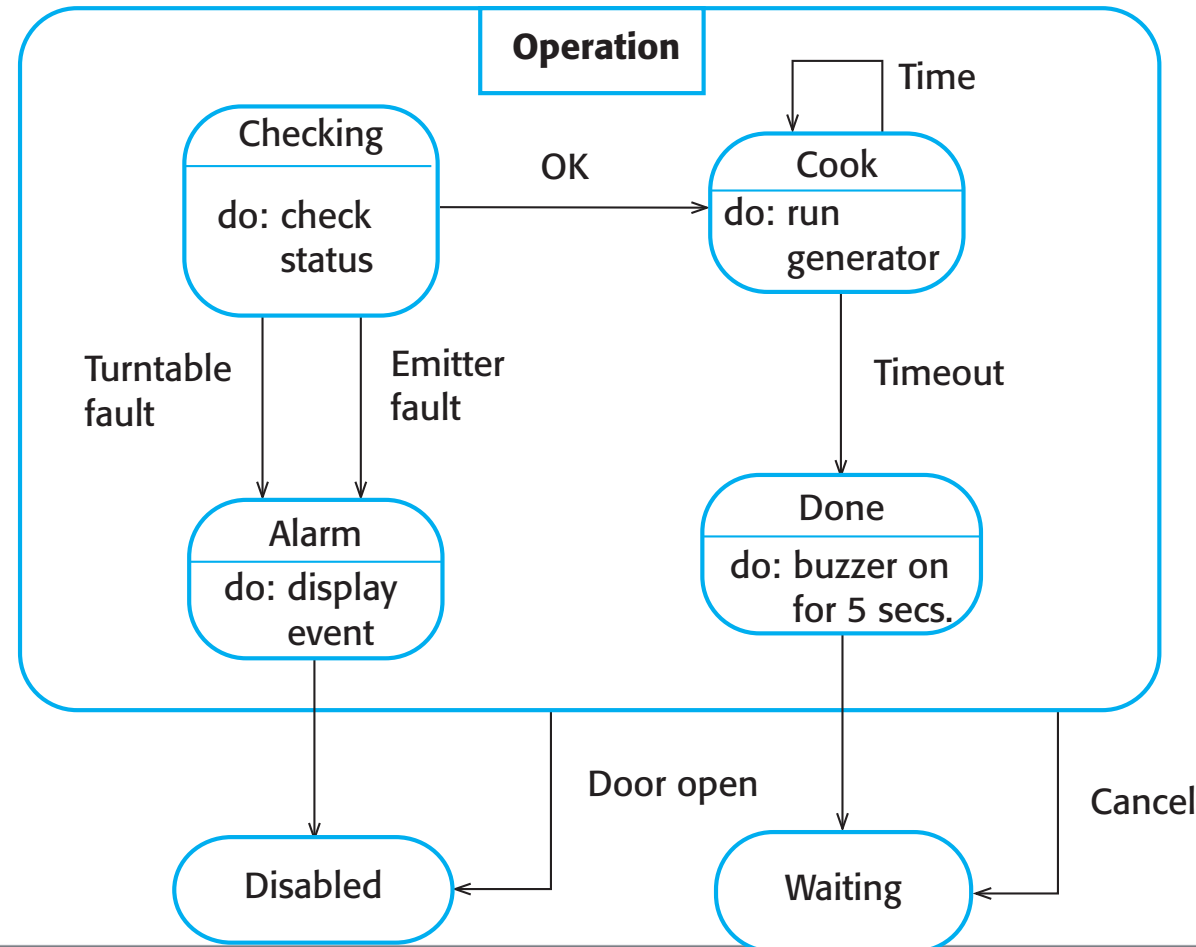
Example: Time Event State Chart



Example: State Diagram of a Microwave Oven



Example: Microwave Oven Operation



Rules for Developing State Charts

1. Select the **classes** that will require state charts
2. List all the **status conditions** for each group
3. Specify **transitions** that cause object to leave the identified state
4. **Sequence** state-transition combinations in correct order
5. Identify **concurrent** paths (if any)
6. Look for **additional transitions**
7. **Expand** each transition as appropriate
8. **Review** and test each state chart

Key Points...

- A model is an abstract view of a system that ignores system details. Complementary system models can be developed to show the system's context, interactions, structure and behavior.
- Context models show how a system that is being modeled is positioned in an environment with other systems and processes.
- Use case diagrams and sequence diagrams are used to describe the interactions between users and systems in the system being designed. Use cases describe interactions between a system and external actors; sequence diagrams add more information to these by showing interactions between system objects.

Key Points

- Structural models show the organization and architecture of a system. Class diagrams are used to define the static structure of classes in a system and their associations.
- Behavioral models are used to describe the dynamic behavior of an executing system. This behavior can be modeled from the perspective of the data processed by the system, or by the events that stimulate responses from a system.
- Activity diagrams may be used to model the processing of data, where each activity represents one process step.
- State diagrams are used to model a system's behavior in response to internal or external events.

PART II: requirements document

Software Requirements Specification (SRS)

Objective

- To understand how requirements may be organized in a software requirements document
- To adopt a standard in producing a requirements document
- To focus on specific requirements that include the analysis models

Software Requirements Document

- Software requirements document is an **official statement** of what is required for the reference of software developers
- Should include both definitions of **user requirements** and specifications of the **system requirements**
- It is NOT a design document
- As far as possible, it should set **WHAT** the system should do rather than HOW it should do it

User and System Requirements

Recall

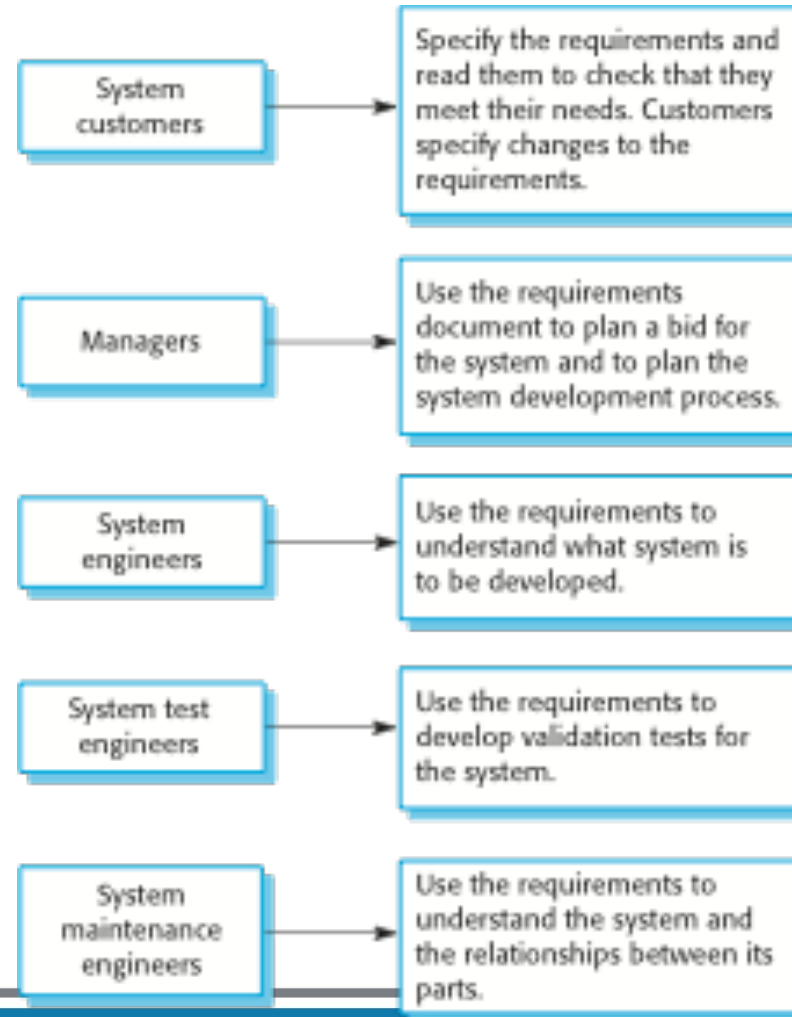
User requirement definition

1. The MHC-PMS shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System requirements specification

- 1.1 On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
- 1.2 The system shall automatically generate the report for printing after 17.30 on the last working day of the month.
- 1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
- 1.4 If drugs are available in different dose units (e.g. 10mg, 20 mg, etc.) separate reports shall be created for each dose unit.
- 1.5 Access to all cost reports shall be restricted to authorized users listed on a management access control list.

Users of a Requirements Document



Requirements Document Variability

- Information in requirements document depends on **type of system** and the **approach used** in its development
- Systems developed incrementally will, typically, have less detail in the requirements document
- Requirements **documents standards** have been designed e.g. IEEE standard
- These are mostly applicable to the requirements for large software projects

IEEE Std 830-1998



SRS Outline of IEEE Std 830-1998

Table of Contents

1. Introduction

1.1 Purpose

1.2 Scope

1.3 Definitions, acronyms, and abbreviations

1.4 References

1.5 Overview

2. Overall description

2.1 Product perspective

2.2 Product functions

2.3 User characteristics

2.4 Constraints

2.5 Assumptions and dependencies

3. Specific requirements (See 5.3.1 through 5.3.8 for explanations of possible specific requirements. See also Annex A for several different ways of organizing this section of the SRS.)

Appendixes

Index

See page 11 of IEEE Std 830-1998

Example of Template for the Front Part of SRS Document

i. Cover Page

<Document Name>
Project Title
Document Version Number
Printing Date
<Department and University>

ii. Revision Page

- a. Overview
- b. Target audience
- c. Project team members *[state the module assigned to each member]*
- d. Version control history

Version	Primary Author(s)	Description of Version	Date Completed

iii. Table of Contents

iv. Additional Materials

- a. Definition
- b. Acronyms/Abbreviations
- c. References
- d. Appendices

Section 1: Introduction

1 Introduction

1.1 Purpose

This subsection should:

- a) *Delineate the purpose of the SRS;*
- b) *Specify the intended audience for the SRS.*

1.2 Scope

This subsection should:

- a) *Identify the software product(s) to be produced by name (e.g., Host DBMS, Report Generator, etc.);*
- b) *Explain what the software product(s) will, and, if necessary, will not do;*
- c) *Describe the application of the software being specified, including relevant benefits, objectives, and goals;*
- d) *Be consistent with similar statements in higher-level specifications (e.g., the system requirements specification), if they exist.*

1.3 Definitions, acronyms, and abbreviations

This subsection should provide the definitions of all terms, acronyms, and abbreviations required to properly interpret the SRS. This information may be provided by reference to one or more appendixes in the SRS or by reference to other documents.

1.4 References

This subsection should:

- a) *Provide a complete list of all documents referenced elsewhere in the SRS;*
- b) *Identify each document by title, report number (if applicable), date, and publishing organization;*
- c) *Specify the sources from which the references can be obtained.*

1.5 Overview

This subsection should:

- a) *Describe what the rest of the SRS contains;*
- b) *Explain how the SRS is organized.*

Section 2: Overall Description, Sub-Section 2.1

2 Overall Description

*This section of the SRS should describe the general factors that affect the product and its requirements. **This section does not state specific requirements.** Instead, it provides a background for those requirements, which are defined in detail in Section 3 of the SRS, and makes them easier to understand.*

[Include use case diagram here]

2.1 Product perspective

*This subsection of the SRS should put the **product into perspective with other related products**. If the product is independent and totally self-contained, it should be so stated here. If the SRS defines a product that is a component of a larger system, as frequently occurs, then this subsection should relate the requirements of that larger system to functionality of the software and should identify interfaces between that system and the software.*

***A block diagram** showing the major components of the larger system, interconnections, and external interfaces can be helpful.*

*This subsection should also describe **how the software operates inside various constraints**. For example, these constraints could include:*

- a) System interfaces;*
- b) User interfaces;*
- c) Hardware interfaces;*
- d) Software interfaces;*
- e) Communications interfaces;*
- f) Memory;*
- g) Operations;*
- h) Site adaptation requirements.*

[Organise as below 2.1.1 to 2.1.8]

Sub-sections 2.1.1 to 2.1.3

2.1.1 System interfaces

*This should **list each system interface** and identify the **functionality of the software** to accomplish the system requirement and the interface description to match the system.*

[Users or other systems interfacing with the system and each function based on the use case diagram]

2.1.2 User interfaces

This should specify the following:

*a) The **logical characteristics** of each interface between the software product and its users.*

*This includes those **configuration characteristics** (e.g., required screen formats, page or window layouts, content of any reports or menus, or availability of programmable function keys) necessary to accomplish the software requirements.*

*b) All the aspects of **optimizing the interface** with the person who must use the system.*

This may simply comprise a list of do's and don'ts on how the system will appear to the user. One example may be a requirement for the option of long or short error messages. Like all others, these requirements should be verifiable, e.g. "A clerk typist grade 4 can do function X in Z min after 1 h of training" rather than "a typist can do function X." (This may also be specified in the Software System Attributes under a section titled Ease of Use.)

2.1.3 Hardware interfaces

*This should specify the logical characteristics of each **interface between the software product and the hardware** components of the system. This includes configuration characteristics (number of ports, instruction sets, etc.). It also covers such matters as what devices are to be supported, how they are to be supported, and protocols. For example, terminal support may specify full-screen support as opposed to line-by-line support.*

Sub-sections 2.2 & 2.3

2.2 Product functions

*This subsection of the SRS should provide a **summary of the major functions** that the software will perform. For example, an SRS for an accounting program may use this part to address customer account maintenance, customer statement, and invoice preparation without mentioning the vast amount of detail that each of those functions requires.*

Sometimes the function summary that is necessary for this part can be taken directly from the section of the higher-level specification (if one exists) that allocates particular functions to the software product. Note that for the sake of clarity:

- a) The functions should be organized in a way that makes the **list of functions understandable to the customer** or to anyone else reading the document for the first time.*
- b) Textual or graphical methods can be used to **show the different functions and their relationships**. Such a diagram is not intended to show a design of a product, but simply shows the logical relationships among variables.*

2.3 User characteristics

*This subsection of the SRS should describe those **general characteristics of the intended users** of the product including educational level, experience, and technical expertise. It should not be used to state specific requirements, but rather should provide the reasons why certain specific requirements are later specified in Section 3 of the SRS.*

Sub-sections 2.4 to 2.6

2.4 Constraints

This subsection of the SRS should provide a general description of any other items that will limit the developer's options. These include:

- a) Regulatory policies;*
- b) Hardware limitations (e.g., signal timing requirements);*
- c) Interfaces to other applications;*
- d) Parallel operation;*
- e) Audit functions;*
- f) Control functions;*
- g) Higher-order language requirements;*
- h) Signal handshake protocols (e.g., XON-XOFF, ACK-NACK);*
- i) Reliability requirements;*
- j) Criticality of the application;*
- k) Safety and security considerations.*

2.5 Assumptions and dependencies

This subsection of the SRS should list each of the factors that affect the requirements stated in the SRS. These factors are not design constraints on the software but are, rather, any changes to them that can affect the requirements in the SRS. For example, an assumption may be that a specific operating system will be available on the hardware designated for the software product. If, in fact, the operating system is not available, the SRS would then have to change accordingly.

2.6 Apportioning of requirements

This subsection of the SRS should identify requirements that may be delayed until future versions of the system.

Section 3: Specific Requirements

3 Specific Requirements

*This section of the SRS should contain **all of the software requirements to a level of detail sufficient to enable designers to design a system** to satisfy those requirements, and **testers to test that the system** satisfies those requirements. Throughout this section, every stated requirement should be externally perceivable by users, operators, or other external systems. These requirements should **include at a minimum a description of every input (stimulus) into the system, every output (response) from the system, and all functions performed by the system in response to an input or in support of an output.***

*[Include domain model i.e. **class diagram** without the operation part – only attributes without details on visibility and type]*

As this is often the largest and most important part of the SRS, the following principles apply:

- a) Specific requirements should be stated in conformance with all the characteristics described in 4.3. (IEEE Std 830-1998)*
- b) Specific requirements should be cross-referenced to earlier documents that relate.*
- c) All requirements should be **uniquely identifiable**. [Provide ID to each functional requirement]*
- d) Careful attention should be given to organizing the requirements to maximize readability.*

Note: Focus on “Specific Requirements” for the scope of this course – see the given System Documentation template that combines the required details for analysis (Section 2), design (Section 3 to 6) and testing (Section 7 and 8) to meet the minimum requirements of this course

Appendix A.5 Template of SRS Section 3: Organized by Feature (IEEE Std 830-1998)

- Sections 3.1.1 to 3.1.4 provide the details of what described in Sections 2.1.2 to 2.1.5
- For each functional requirement, include the **use case diagram** and its details using **use case description**
- Include **sequence diagram** and **activity diagram** for each use case i.e. functional requirement
- Include **state diagram** for classes with states under respective use cases

3.	Specific requirements
3.1	External interface requirements
3.1.1	User interfaces
3.1.2	Hardware interfaces
3.1.3	Software interfaces
3.1.4	Communications interfaces
3.2	System features
3.2.1	System Feature 1
3.2.1.1	Introduction/Purpose of feature
3.2.1.2	Stimulus/Response sequence
3.2.1.3	Associated functional requirements
3.2.1.3.1	Functional requirement 1
.	.
.	.
3.2.1.3.n	Functional requirement <i>n</i>
3.2.2	System feature 2
.	.
.	.
3.2.m	System feature <i>m</i>
.	.
.	.
3.3	Performance requirements
3.4	Design constraints
3.5	Software system attributes
3.6	Other requirements

Our
Focus

Key Points

- The software requirements document is an **agreed statement** of the system requirements
- It should be organized so that both system customers or **users** and software **developers** can use it
- Documentation standard can be referred and customized based on stakeholders' needs