

Engenharia de Software (14341, 16230, 15386)

Software Design Principles, UML, Patterns and Quality Attributes #2

(adapted from *Software Engineering: International Version (10th Edition)*, Ian Sommerville, Pearson, 2015)

Topics covered

1



Structural Models

Represent the organization of a system and the relationships between its parts.

- **Class diagrams**
  Classes, attributes, operations and relationships
- **Generalization**
  Inheritance and hierarchies
- **Aggregation**
  Whole-part relationships

2



Behavioral Models

Capture the dynamic behavior of a system over time.

- **State diagrams**
  Object life cycle and state transitions
- **Events and transitions**
  How the system reacts to internal and external events

3



Object Relationships

Model how objects interact and depend on each other.

- **Associations**
  Links and multiplicities
- **Composition**
  Strong ownership and lifetime dependency
- **Dependency & Realization**
  Usage and interface contracts

4



Model-Driven Engineering

Use models to guide, automate and validate the development of software systems.

- **Models as source of truth**
  Basis for analysis, design and implementation
- **From model to code**
  Model transformations and code generation
- **Continuous synchronization**
  Keeping models and code aligned



OUR GOALS

By the end of this lecture, you should be able to:



Recognize
the differences between structural and behavioral models.



Create
UML class and state diagrams for a software system.



Interpret
relationships such as associations, aggregation, composition and inheritance.



Select
the most appropriate UML diagram for a given engineering problem.


PROFESSIONAL SKILL
SPOTLIGHT
DEVELOP • APPLY • GROW

Empathy



Structural models

Structural models

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

Class diagrams

- Class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes.
- An 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 you are developing models during the early stages of the software engineering process, objects represent something in the real world, such as a patient, a prescription, doctor, etc.

Class diagrams

✧ Purpose:

- Represent the **system's static structure**.
- Show **classes**, **attributes**, **methods**, and **relationships** between them.
- Provide a **blueprint** for coding and database design.

✧ Best Practices:

- Keep attributes and methods **relevant to the current scope** — avoid clutter.
- Show **relationships clearly** (associations, aggregations, compositions, inheritance).
- Use **consistent naming conventions**.
- Group related classes logically to improve readability.
- Adjust the level of detail to fit your audience — provide more or less depth depending on their background and needs.

Class diagrams

CLASS BOX – COMPONENTS

Class Name
The name identifies the class and should be meaningful.

Attributes
Describe the data (properties) of the class.
Notation: visibility name: type [multiplicity] = default

Operations
Describe the behavior (services) of the class.
Notation: visibility name(param: type): returnType

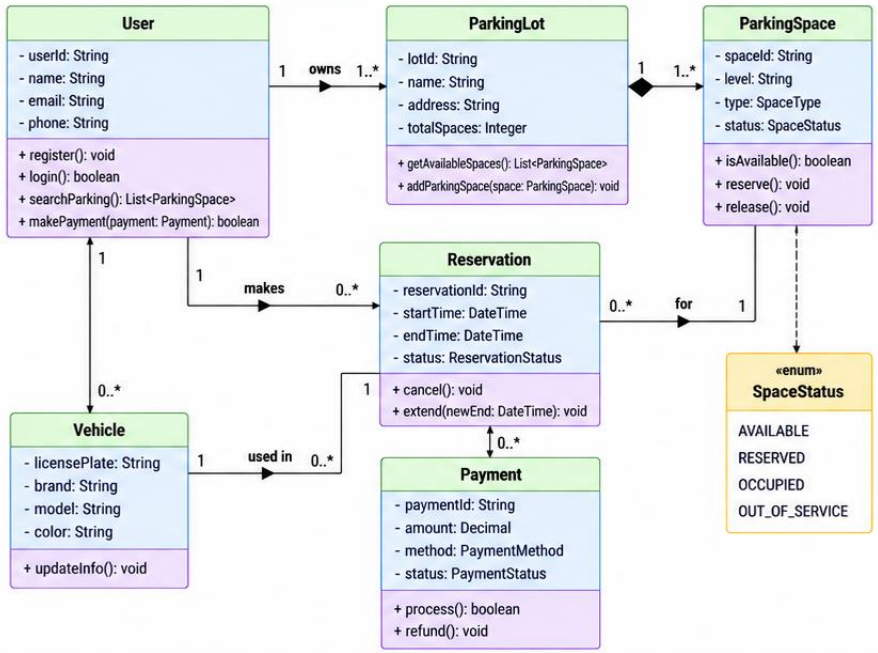
Visibility
Indicates the visibility level.

- `+` public
- `-` private
- `#` protected
- `~` package (default)

Note / Constraint
Used to add comments or constraints to elements.

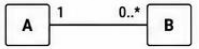
A class diagram represents the **static structure** of a system, showing its **classes**, their **attributes, operations**, and the **relationships** among them.

EXAMPLE – SMART PARKING SYSTEM

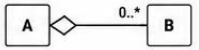


RELATIONSHIPS (ASSOCIATIONS)

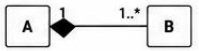
Association
A structural relationship between classes.



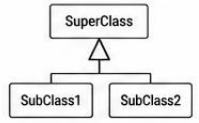
Aggregation
A weak whole–part relationship. Parts can exist independently.



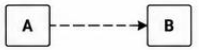
Composition
A strong whole–part relationship. Parts cannot exist without the whole.



Generalization (Inheritance)
An "is-a" relationship where a subclass inherits from a superclass.



Dependency
A "uses" relationship. A change in the supplier may affect the client.



Multiplicity
Indicates the number of instances participating in a relationship.

1	exactly one
0..1	zero or one
0..*	zero or more
1..*	one or more
m..n	between m and n

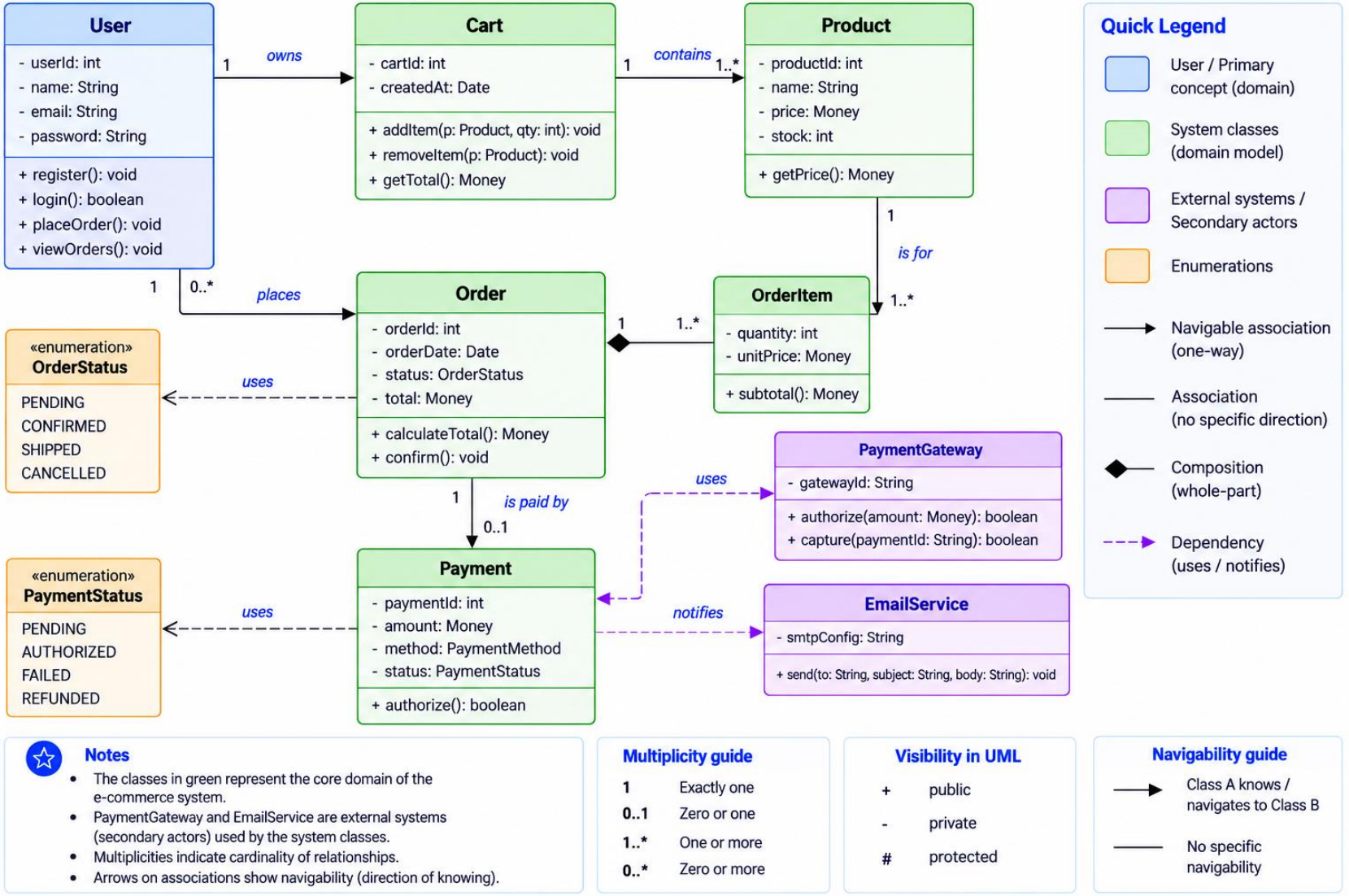
GOOD PRACTICES

- Use meaningful class names.
- Keep classes cohesive and focused.
- Use appropriate visibility.
- Model relationships with correct multiplicities.
- Avoid over-modeling: keep it simple.

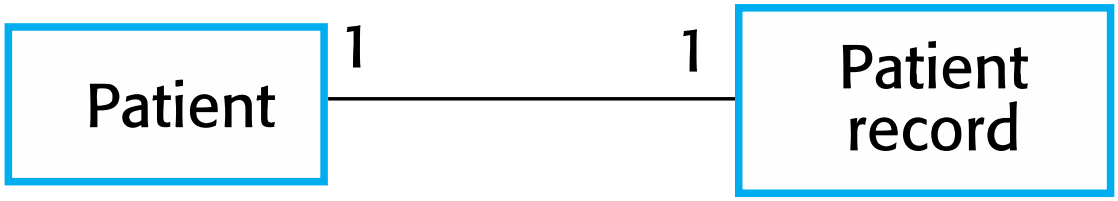
REMEMBER

- A class represents a set of objects with similar properties and behaviors.
- Attributes describe what the object has.
- Operations describe what the object can do.
- Relationships connect classes and define how they interact.

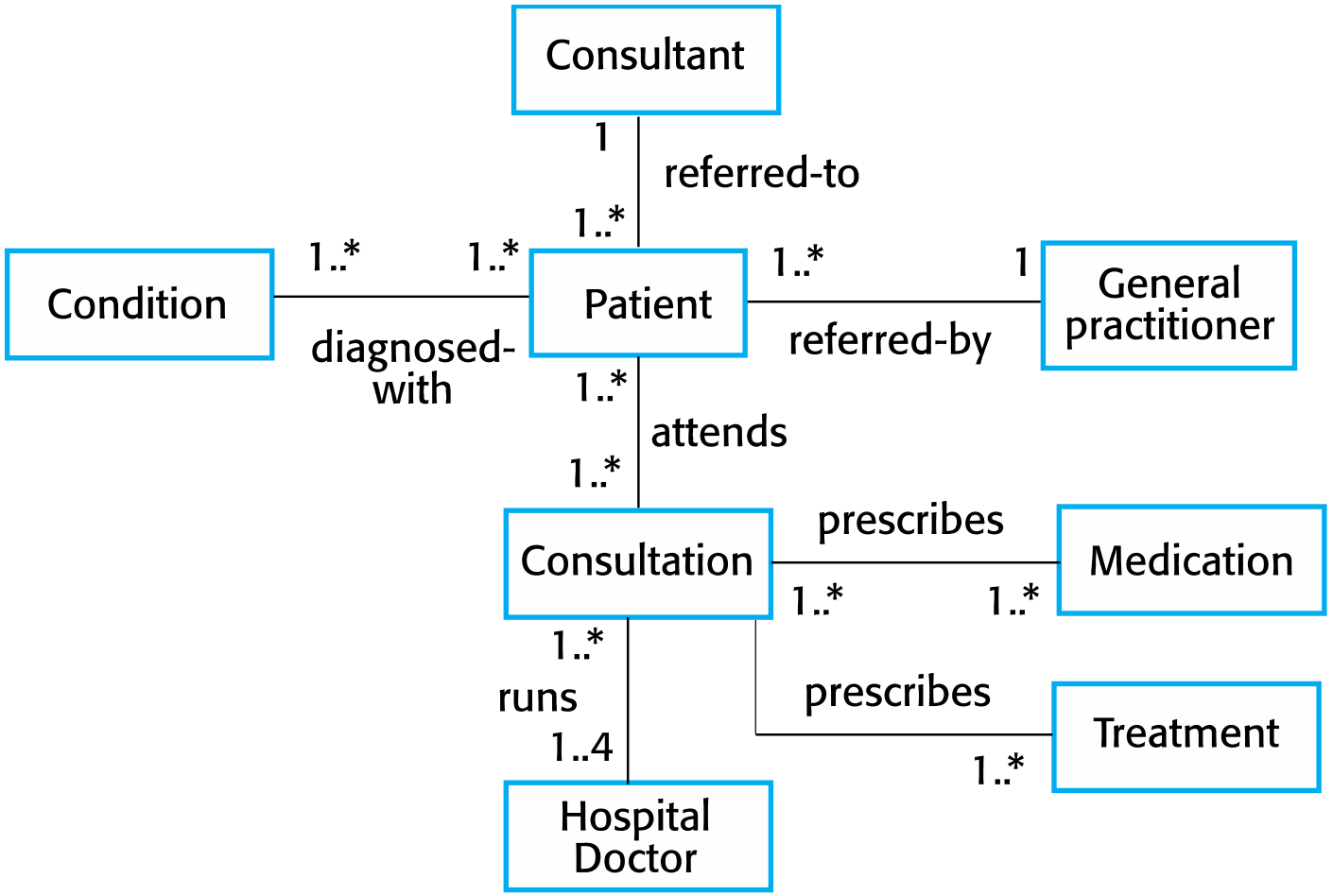
Class diagram for e-commerce system



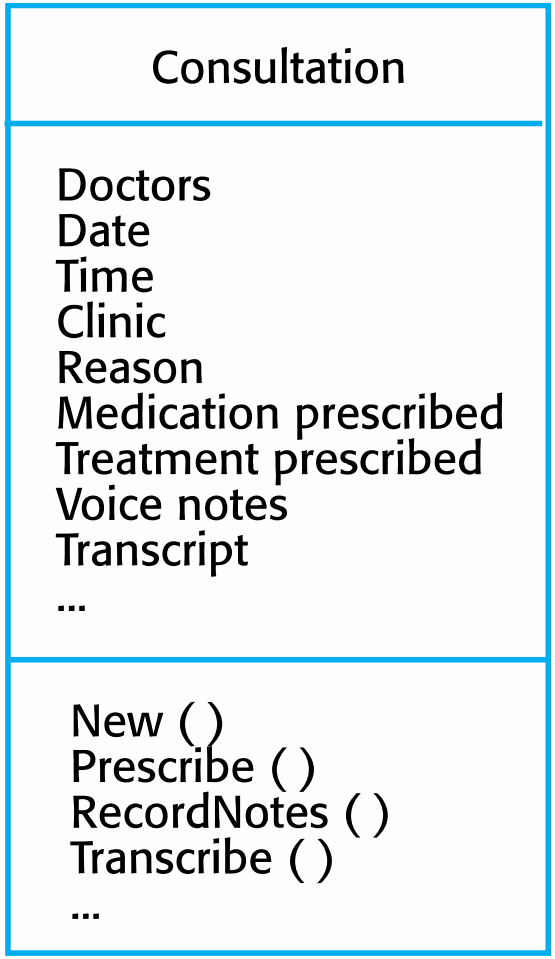
UML classes and association



Classes and associations in the MHC-PMS



The Consultation class



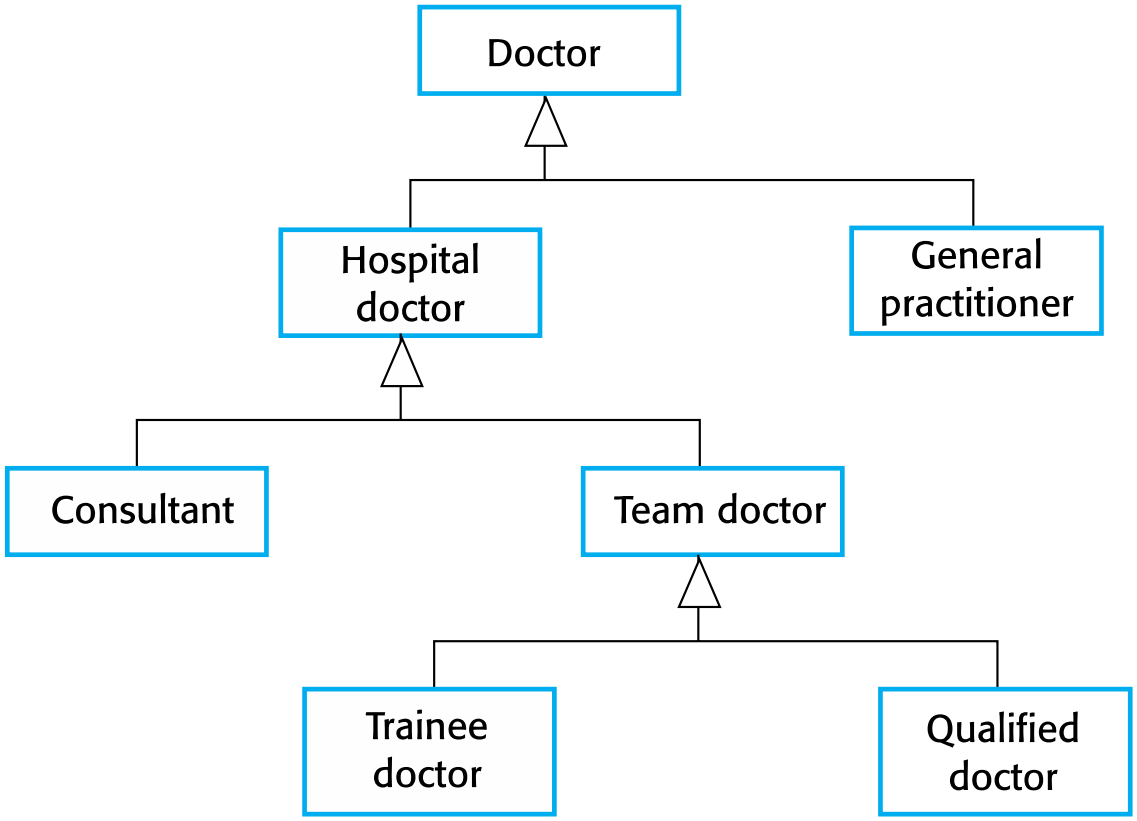
Generalization

- Generalization is an everyday technique that we use to manage complexity.
- Rather than learn the detailed characteristics of every entity that we experience, we 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. squirrels and rats are rodents.

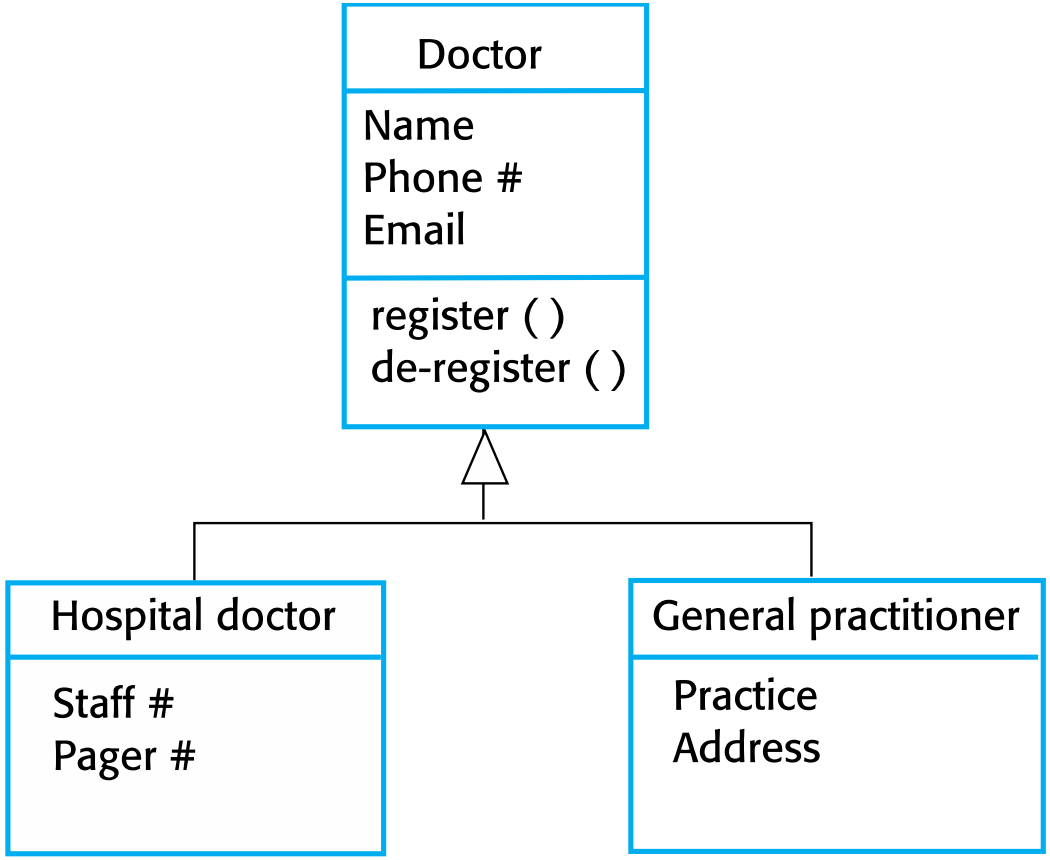
Generalization

- In modeling systems, it is often useful to examine the classes in a system to see if there is scope for generalization. If changes are proposed, then you do not have to look at all classes in the system to see if they are affected by the change.
- In object-oriented languages, such as Java, generalization is implemented using the class inheritance mechanisms built into the language.
- 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.

A generalization hierarchy



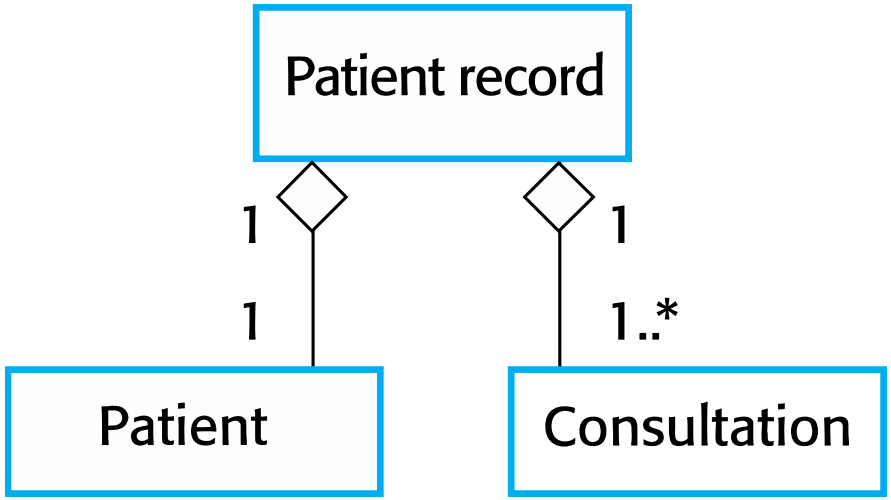
A generalization hierarchy with added detail



Object class aggregation models

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

The aggregation association



Behavioral models

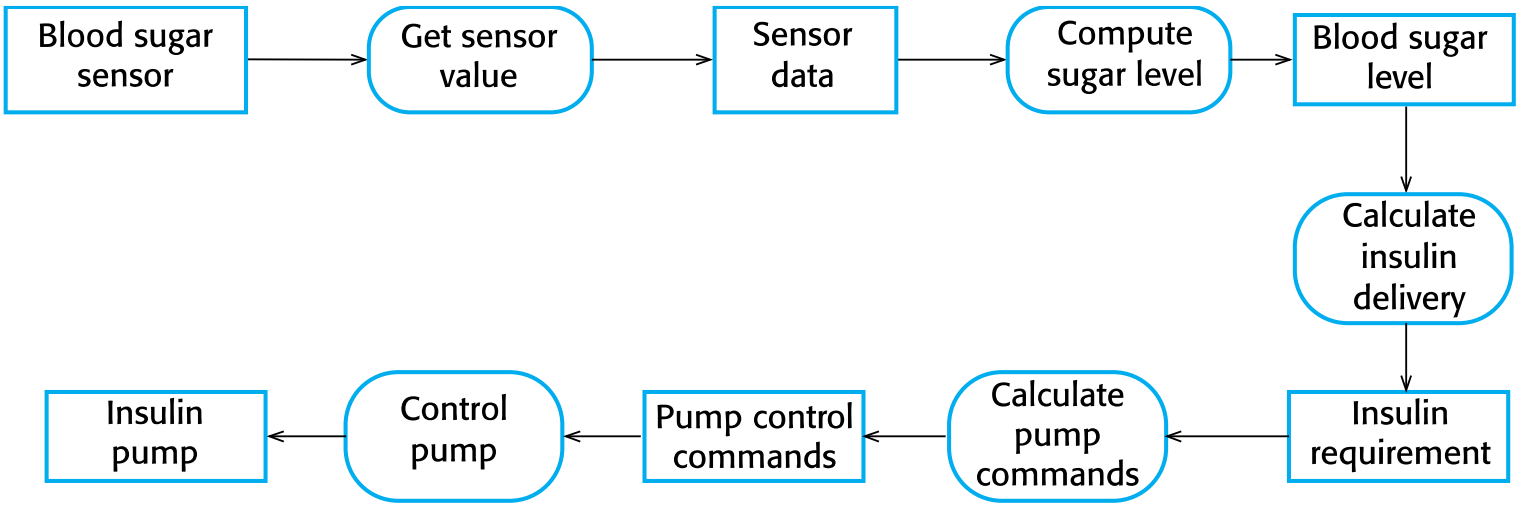
Behavioral models

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

Data-driven modeling

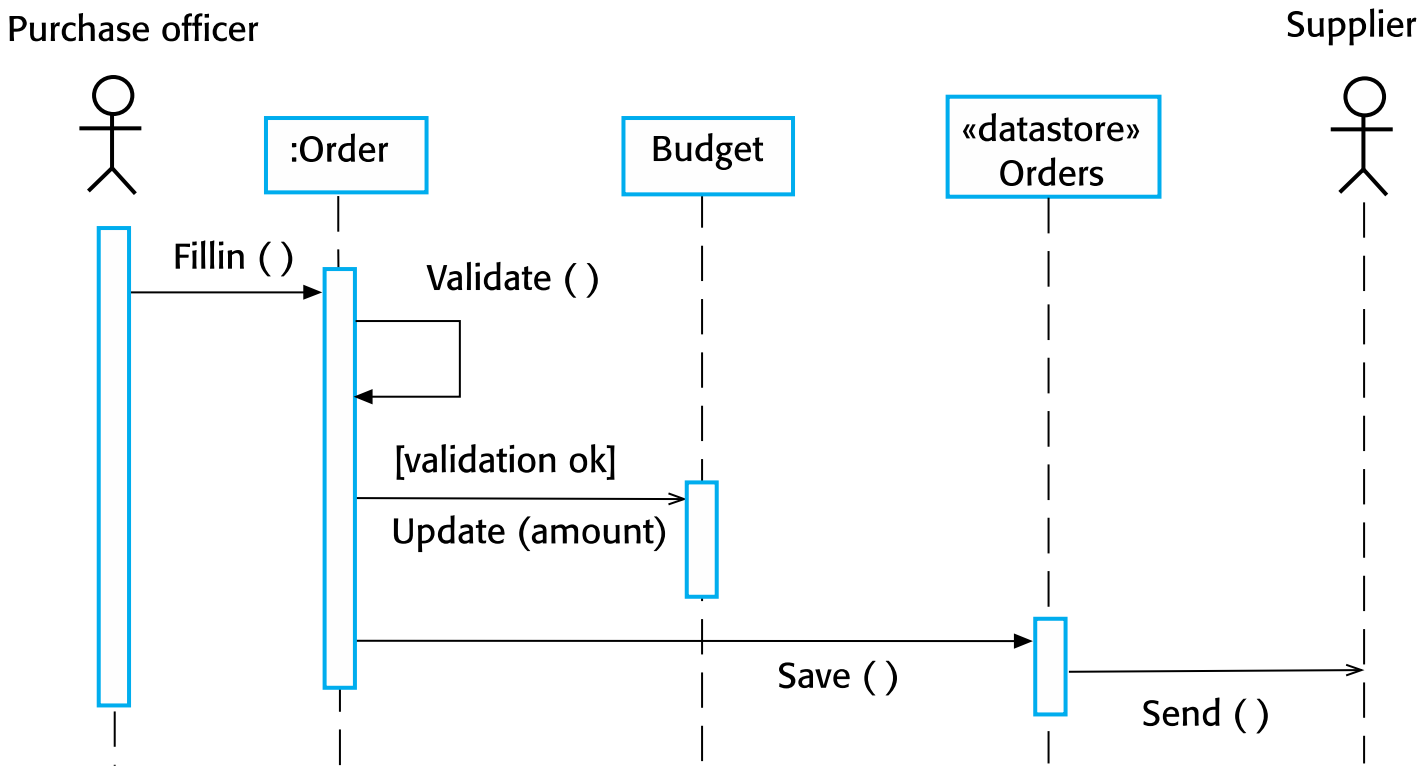
- Many business systems are data-processing systems that are primarily driven by data. They are controlled by the data input to the system, with relatively little external event processing.
- Data-driven models show the sequence of actions involved in processing input data and generating an associated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

An activity model of the insulin pump's operation



- Processing steps (rounded rectangles) representing activities
- Data flowing between these steps (rectangles), represented as objects

Order processing



- A sequence diagram can be used to model interaction, in which messages are only sent from left to right

Event-driven modeling

- Real-time systems are often event-driven, with minimal data processing.
- Event-driven modeling shows how a system responds to external and internal events.
- It is based on the assumption that a system has a finite number of states and that events (stimuli) may cause a transition from one state to another.

State machine models

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

State diagrams

✧ Purpose:

- Model the **lifecycle of an object** in response to events.
- Show **states**, **transitions**, **events**, and **actions** that define the behaviour of the system.
- Useful for systems with **complex, event-driven logic**.

✧ Best Practices:

- Clearly define **start** and **end** states.
- Keep the number of states **manageable** — avoid overcomplication.
- Use **meaningful names** for states and events.
- Show **only relevant transitions** to maintain clarity.
- Group related transitions logically to highlight patterns.
- Include **guard conditions** and **actions** only when they add clarity.

State diagrams

ELEMENTS OF A STATE DIAGRAM

Initial State
The starting point of the state machine.

State
Represents a condition or situation during the life of an object.

Transition
A change from one state to another triggered by an event.

[event] [action]

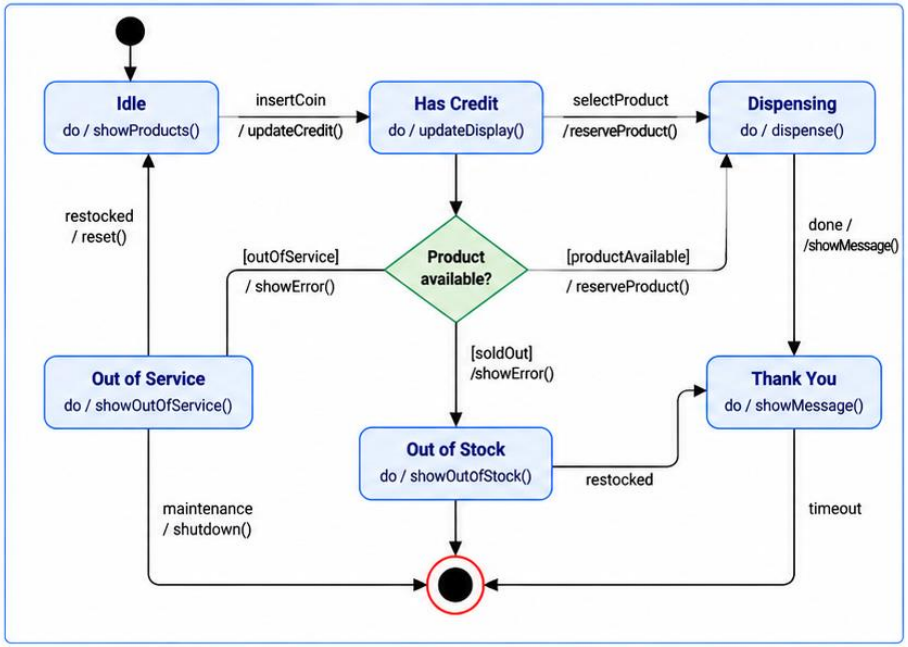
Trigger and Action
Format: event [guard] / action
event: occurrence that triggers the transition
guard: condition (optional)
action: executed when the transition occurs (optional)

Decision (Choice)
Evaluates guard conditions and selects one transition.

Final State
Indicates the end of the state machine.

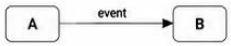
A state diagram models the **dynamic behavior** of an object, showing the **sequence** of **states** it can be in and the **events** that **cause transitions between** them.

EXAMPLE – VENDING MACHINE

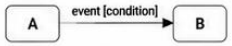


TRANSITIONS AND NOTATION

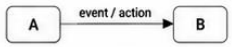
Simple Transition
Triggered by an event.



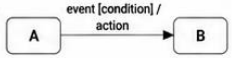
Transition with Guard
Occurs only if the guard condition is true.



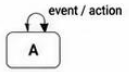
Transition with Action
Executes an action when the transition occurs.



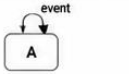
Transition with Guard and Action
Uses both a guard condition and an action.



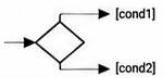
Internal Transition
Does not change the state; only performs an action.



Self Transition
Leaves and re-enters the same state.



Choice (Decision)
Selects one path among alternatives.



Final State
Indicates the end of the state machine.



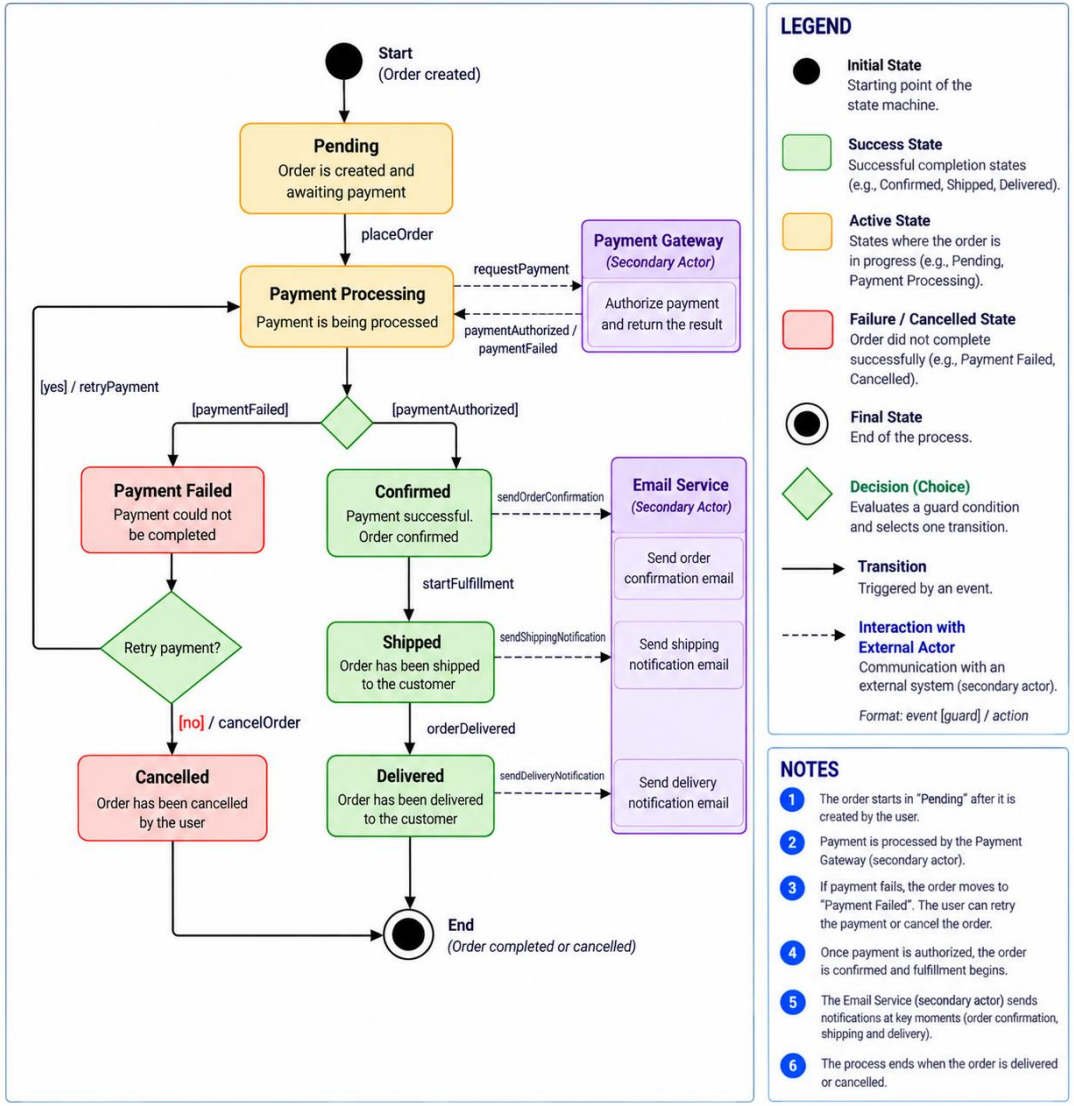
GOOD PRACTICES

- ✓ Keep states meaningful and at the right level of abstraction.
- ✓ Use clear event names.
- ✓ Add guards only when necessary.
- ✓ Keep actions short and non-blocking.
- ✓ Avoid too many transitions from a state.
- ✓ Use consistent naming and notation.

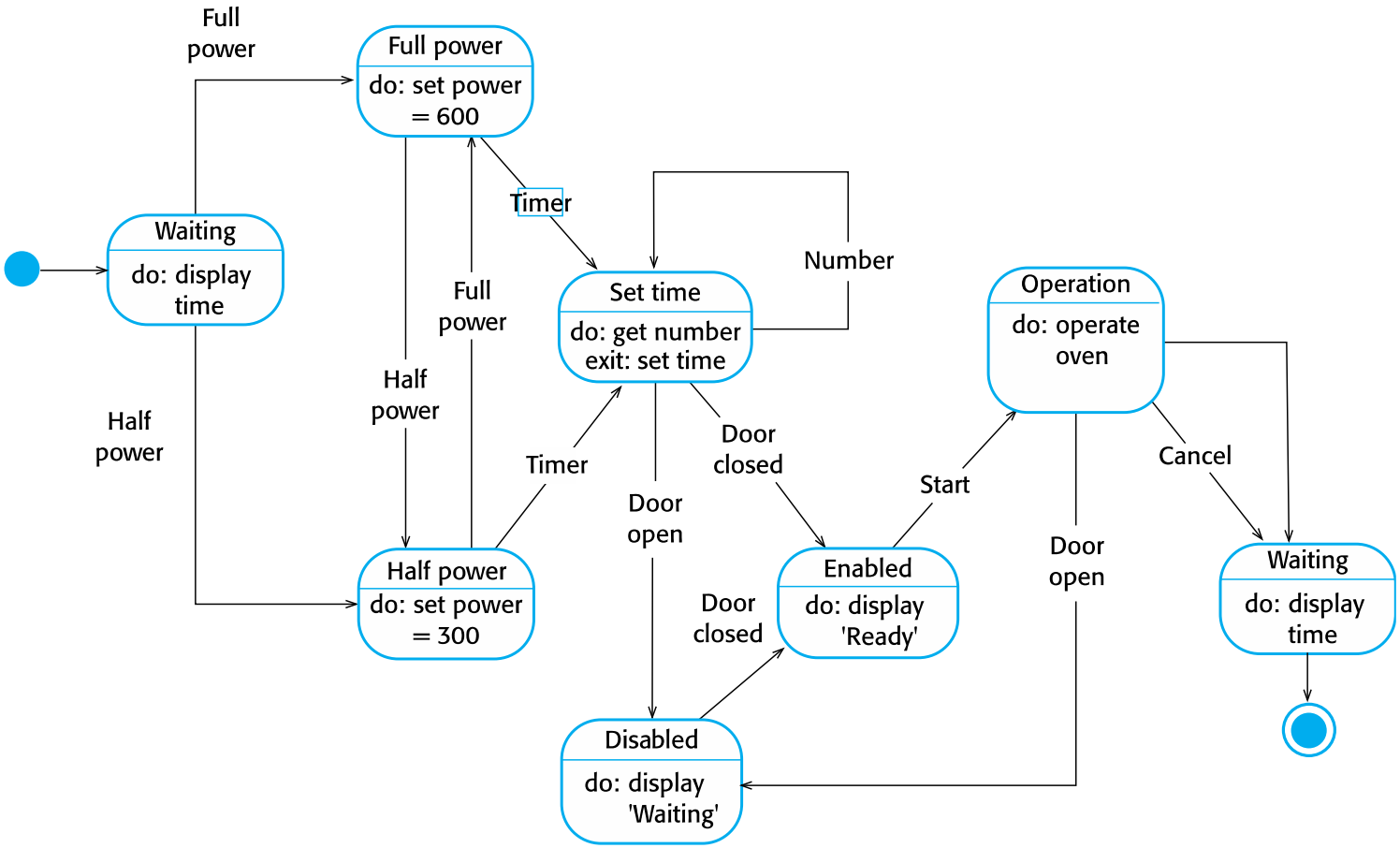
REMEMBER

- A state represents a condition of an object.
- A transition occurs due to an event.
- Guards add conditions to transitions.
- Actions represent the effect of transitions.
- The model describes the life cycle behavior of the object.

State diagram for e-commerce system



State diagram of a microwave oven



States and stimuli for the microwave oven (a)

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

States and stimuli for the microwave oven (b)

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


IN-CLASS EXERCISE
PRACTICE • APPLY • UNDERSTAND


Work individually.
Hand in your work on a sheet of paper with your student number and name.
The exercise will be given by the instructor during the class.

Engineering insight

Principles to remember

Model the what and the how.
Structure explains what exists.
Behavior explains how it changes.

Model relationships, not just objects.
Collaboration, ownership and dependency drive software behavior.

Abstract at the right level.
Too much detail obscures.
Too little detail misleads.

Models evolve with understanding.
Start simple, validate, refine and iterate.

Good models accelerate everything.
They improve communication, reduce rework and enable model-driven implementation.

Engineering Insight

Good models simplify complexity without losing what matters.

Software engineering is about choosing the **right level of abstraction**—not modeling every detail.



Course takeaway
Use the simplest model that accurately captures the system's intent.

Clarity → Design → Software

Remember:

We don't just create diagrams.
We build **shared understanding** that leads to **better systems.**

Understand | Design | Deliver Value

Key points

1



Structural models organize the system.

- They describe the system in terms of its components and relationships.
- Class diagrams capture the static structure of the system.

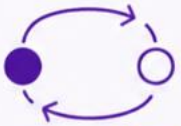
2



Relationships express how objects collaborate.

- Associations connect classes.
- Aggregation / Composition show part-whole relationships.
- Generalization captures inheritance and shared characteristics.

3



Behavioral models describe system dynamics.

- State diagrams model object life cycles.
- They show how objects respond to events through states and transitions.
- Useful for capturing rules, constraints and valid sequences.

4



Abstraction with generalization manages complexity.

- Group specific classes under more general superclasses.
- Changes at higher levels propagate to all subclasses.

5



Models are the bridge to implementation.

- Good models improve communication and reduce ambiguity.
- In model-driven engineering, models can be transformed into code.

6



Quality models improve quality software.

- They support analysis, validation and decision making.
- Clear models lead to better design, maintainability and evolution.



Engineering Insight

Good models simplify complexity without losing what matters.
They are the foundation of good engineering.



QUESTIONS?

Let's clarify, explore and learn together!

Concepts

Anything you didn't understand?

Applications

How do we apply these ideas?



Decisions

Want to discuss any trade-offs?

Next Steps

What would you like to learn next?

No question is too small!

Your questions help everyone learn and grow.

Ask freely, challenge ideas, and let's build understanding **together!**