

SOFTWARE ENGINEERING

Chapter 2

Modeling with UML

Mrs. S. HEDJAZ · 2025/2026



CHAPTER 2

Modeling with UML

Outline

01

Introduction

Modeling, why we model, and the object-oriented approach

02

Historic

Booch, OMT, OOSE and the birth of UML

03

General elements and mechanisms

Why UML, the 4+1 architecture, the modeling axes

04

UML diagrams

The 13 diagrams, what each one shows, and when to use it

01

Introduction

What modeling is, why we model, and the object-oriented approach

Modeling and models



Modeling: the act of designing a model in a dedicated modeling language. It consists of identifying the relevant characteristics of a system, in order to study it from the point of view of these characteristics.

WHAT IS A MODEL?



An abstraction

It helps us understand a complex object, by representing the system at different degrees of detail.



A simplification

A simplified representation of a reality: a software system, a building, a weather system...



Often graphical

“A small drawing is better than a long speech.” A diagram is read faster than a long text.

What makes a good model?



It must help us understand

A good model makes the studied system easier to understand. If the model is as complex as the system, it is useless.



It must allow simulation

A good model lets us simulate the system, so we can predict its behavior before building it.

MODELING IS NOT ONLY FOR SOFTWARE



A car

Plans, tests and engine trials come before production.



A house

An architect's plan, plus tests on the resistance of materials.



The weather

Meteorologists use models to predict tomorrow's weather.



Software

UML models the system before a single line of code is written.

Why model?



Understand the system

A model makes a complex system easier to understand for the whole team.



Talk with the customer

A diagram is understood by people who are not programmers.



Write the requirements

Requirements can be written accurately and completely, without knowing the details yet.



Focus on one aspect

Each diagram offers a different vision of the same system.



Test many solutions

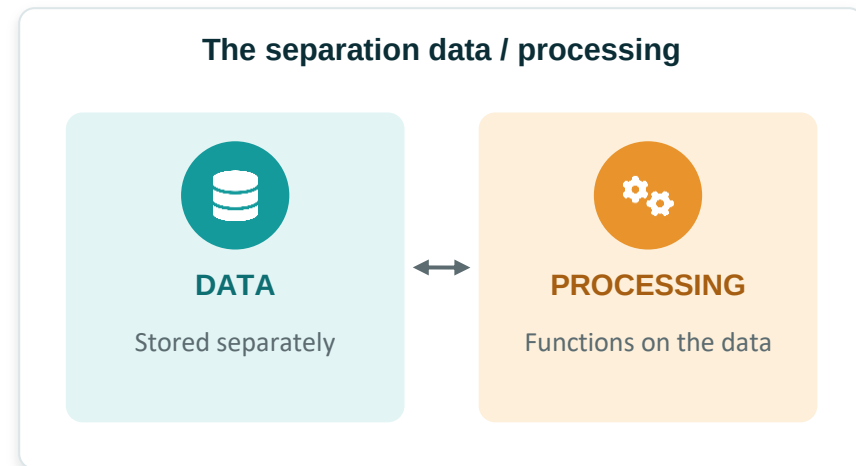
Trying solutions on a model costs less and takes less time than coding them.

Modeling methods: the functional approach



A **method** defines a reproducible approach to obtain reliable results. A software development method describes how to model and build software systems reliably.

Functional and systemic methods were imposed first (the 1970s and 1980s). They were directly inspired by the architecture of computers: as data and processing are physically separated in the machine, the same separation was transposed into the methods.



Limits of the functional approach



Cascade effect

A simple update at one point can cascade into many other functions, because they share the same data.



Many maintenance points

A major evolution multiplies the maintenance points: the software must be retouched in its entirety.



Solution: centralize the data of one type **and** its associated processing inside a single entity. This limits the maintenance points in the code and makes information easier to reach when the software evolves.

→ This solution is the object-oriented (OO) approach

Functional approach vs object approach

	Functional approach	Object approach
Data and processing	Separated	Grouped inside the object
Unit of decomposition	The function	The class
Organized around	What the system does	The concepts of the domain (client, order...)
Effect of a change	Cascades into many functions	Stays local, inside the object
Reuse	Difficult	Natural, through inheritance

In short: the object approach brings the processing to the data, instead of bringing the data to the processing.

Object approach: the object

An object is a self-contained entity, which brings together a set of consistent properties and the associated processing.



State

The values of its attributes at a given moment.



Behavior

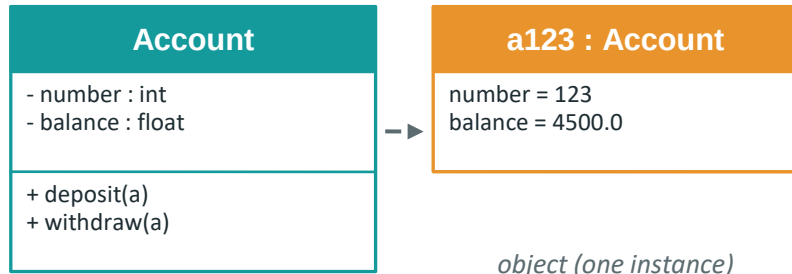
The operations (methods) it can perform.



Identity

What makes it different from every other object.

A class and one of its objects



class (the mould)

object (one instance)

Object approach: encapsulation

Encapsulation consists of hiding the implementation details of an object, by defining an **interface** that declares the services offered to the users of the object.

PUBLIC INTERFACE

+ deposit(a)

+ withdraw(a)

+ getBalance()



The user of the object only sees the left side.

HIDDEN INSIDE

- balance : float

- history : List

- checkLimit()



Protects the data

Internal attributes cannot be modified directly from outside.



Eases evolution

The implementation can change without changing the interface.



Stabilizes usage

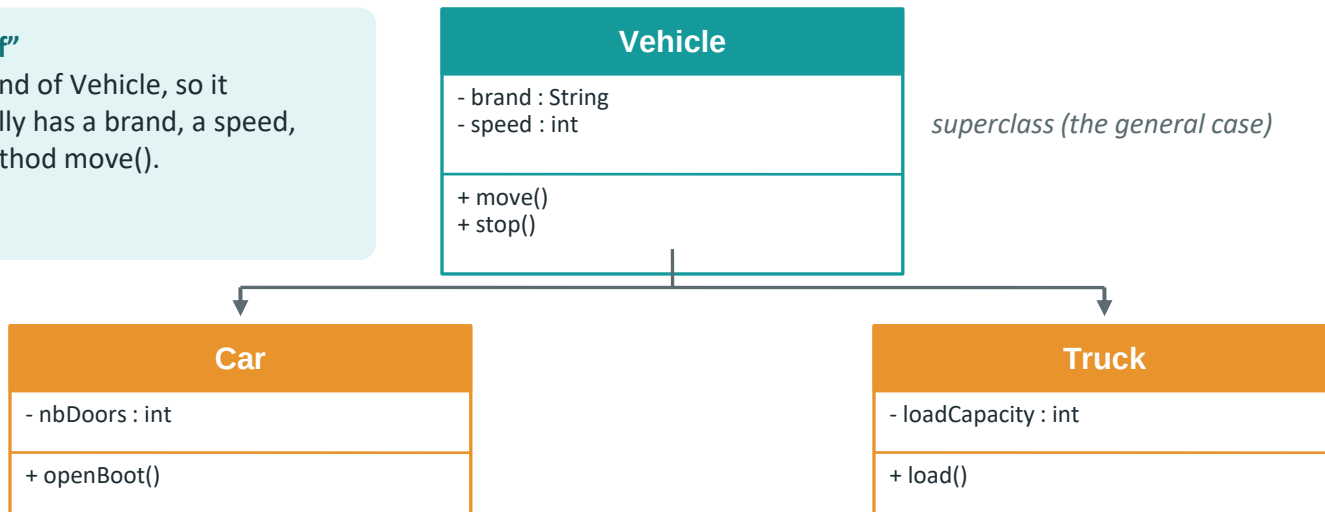
Code that uses the object keeps working after an internal change.

Object approach: inheritance

Inheritance is a mechanism for passing the properties of a class (its attributes and methods) to a subclass. It prevents duplication and encourages reuse.

"is a kind of"

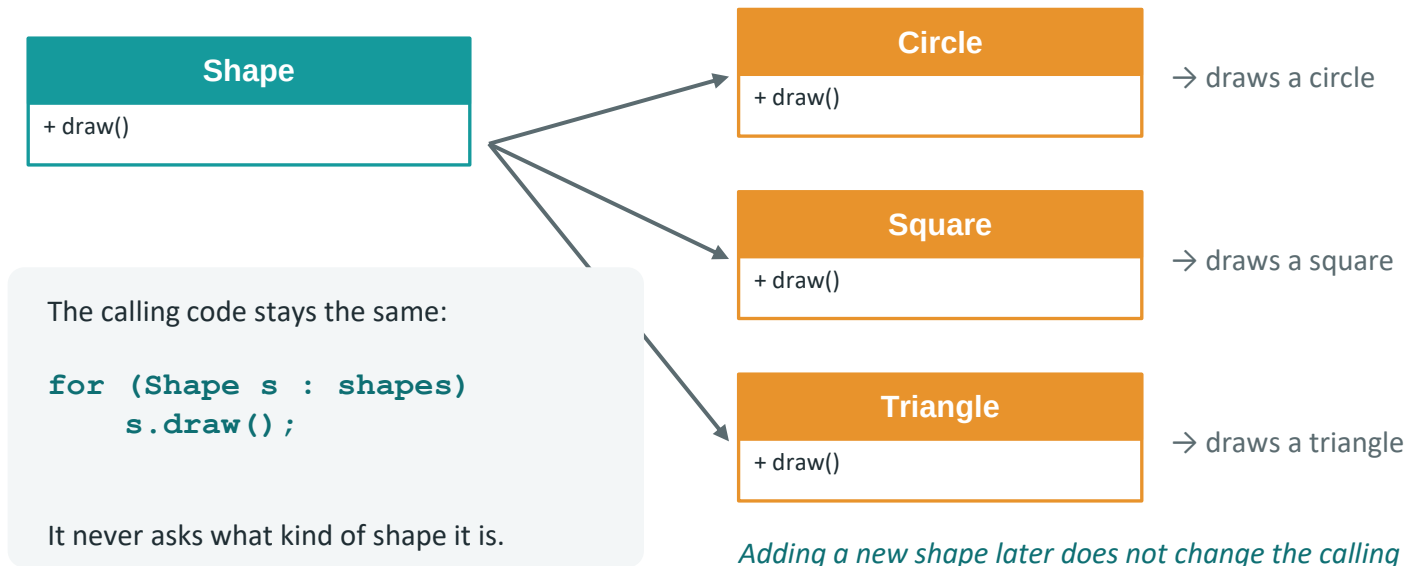
A Car is a kind of Vehicle, so it automatically has a brand, a speed, and the method move().



subclasses: they inherit everything above and add what is specific

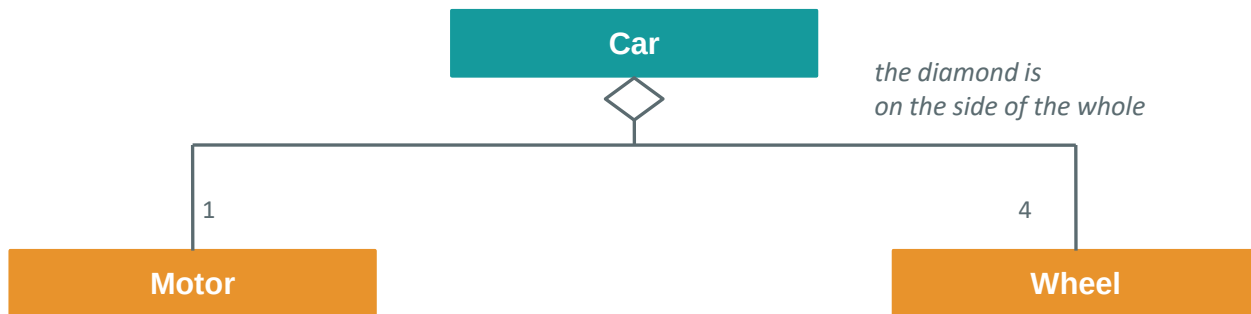
Object approach: polymorphism

Polymorphism is the ability of the same operation to execute differently depending on the class in which it is located. It increases the genericity of the code.



Object approach: aggregation

Aggregation is a relationship between two classes, specifying that the objects of one class are components of the other. It allows us to assemble basic objects to build more complex objects.



Aggregation (empty diamond)

The part can exist without the whole. A wheel still exists if the car is destroyed.



Composition (filled diamond)

The part cannot exist without the whole. The lines of an invoice disappear with the invoice.

02

Historic

From the object methods of the 1990s to the birth of UML

The method explosion of the early 1990s

≈ 50

object-oriented methods between
1990 and 1995

Each method had its own notation for the same ideas. The same concept — a class, an object, an association — was drawn differently in each one.



No exchange

Teams using different methods could not exchange their models.



No tool compatibility

Each CASE tool supported its own notation only.



Hard to learn

A new notation had to be learned for every method.

THREE DOMINANT METHODS

Booch

Grady Booch

OMT

James Rumbaugh

OOSE

Ivar Jacobson

They will merge into UML.

The Booch method



Grady Booch

Rational Software, 1991–1993

Logical level

- Class diagrams
- Object (instance) diagrams
- State / transition diagrams

Physical level

- Module diagrams (package principle)
- Process diagrams



Strong point: a very rich notation for design and for the physical architecture of the system.

OMT (Object Modeling Technique)



James Rumbaugh

General Electric, 1991

Static axis

- Identifies the properties of objects
- And their links with other objects

Dynamic axis

- Life cycle of the objects
- Their behavior and their states
- The events that trigger state changes

Functional axis

- Specifies the functions performed by the objects
- Through their methods



Strong point: the three axes of modeling — static, dynamic and functional — that UML still uses today.

OOSE (Object Oriented Software Engineering)



Ivar Jacobson

Ericsson, 1992

Five models

- Description of requirements
- Analysis
- Conception (design)
- Implantation
- Test

Three types of objects

- Entities: the information of the domain
- Controls: the processing logic
- Interfaces: the contact with the outside



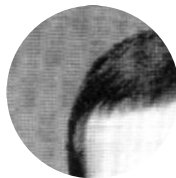
Main contribution: the concept of use cases, which describes the system from the point of view of its users.

The “three amigos” and the unification



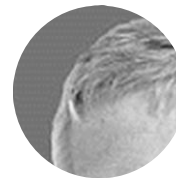
Grady Booch

Design notation and physical architecture



James Rumbaugh

The static, dynamic and functional axes



Ivar Jacobson

Use cases and the user's point of view

1994

Booch and Rumbaugh unify their methods at Rational

1995

Jacobson joins them · Unified Method 0.8 (OOPSLA'95)

1996

They choose a language, not a method
→ UML 0.9

The history of UML, step by step

SOURCE METHODS

Booch'91 / Booch'93

OMT-1 / OMT-2

OOSE

Other methods and partners



Unified Method 0.8

Oct 1995 · OOPSLA'95



UML 0.9

June 1996 · beta version



UML 1.0

Jan 1997 · submitted to the OMG



UML 1.1

Sept 1997 · standardized by the OMG (Nov 1997)



UML 1.2 → 1.x

June 1998 → 2002 · revisions



UML 2.0

2005 · a major revision by the OMG



Today: UML 2.5.1

The versions of UML

Version	Date	What it brings
Unified Method 0.8	Oct 1995	First joint proposal of Booch and Rumbaugh (OOPSLA'95)
UML 0.9	June 1996	First version carrying the name UML; Jacobson has joined
UML 1.0	Jan 1997	Submitted to the OMG by Rational and its partners
UML 1.1	Nov 1997	Adopted as a standard by the OMG
UML 1.2 → 1.5	1998 – 2002	Corrections and clarifications of the standard
UML 2.0	2005	Major revision: 13 diagrams, new diagram types
UML 2.5.1	2017	The current version of the standard

OMG = Object Management Group, the international consortium that maintains the UML standard.

03

General elements and mechanisms

What UML is, the 4+1 architecture, and the three modeling axes

Why UML?



Graphical and simple

A diagram is read faster than a long document.



A standard

Maintained by the OMG: the same notation for everybody.



Represents entire systems

Not only the software part: hardware, deployment, processes.



A common language

Usable by all methods, and adapted to all phases of development.



Iterative and incremental

The model grows step by step, through successive versions.



Guided by user needs

Use cases put the needs of the users at the start of the work.

UML is also architecture-centric: the architecture of the system is at the centre of the development work.

UML is a language, not a method



UML = a LANGUAGE

It gives the notation: the symbols, the diagrams, and their meaning. It tells you how to write a model.



A METHOD

It gives the approach: which activity comes first, who does it, and what must be produced. It tells you how to run the project.

UML is a toolbox used by several methods and development processes:

Unified Process (UP)

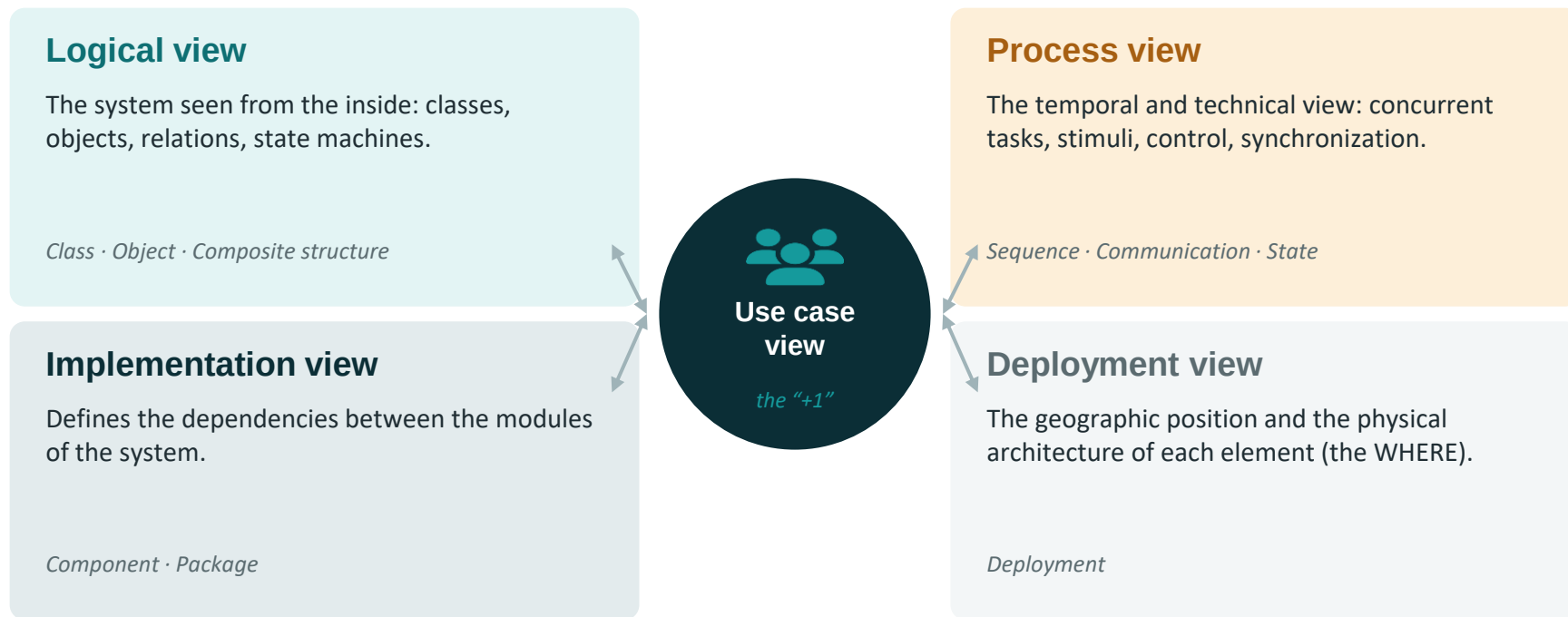
RUP

2TUP

Agile methods with modeling

The authors of UML recommend using the language together with an approach or a method.

The 4+1 architecture



The 4+1 views in detail



The view of the use cases

The description of the model “seen” by the actors of the system. It corresponds to the needs expected by each actor.



The logical view

Expresses the abstract perspective of the solution: classes, objects, relations, transition state machines.



The implementation (component) view

Defines the dependencies between the modules of the system.



The behavior (process) view

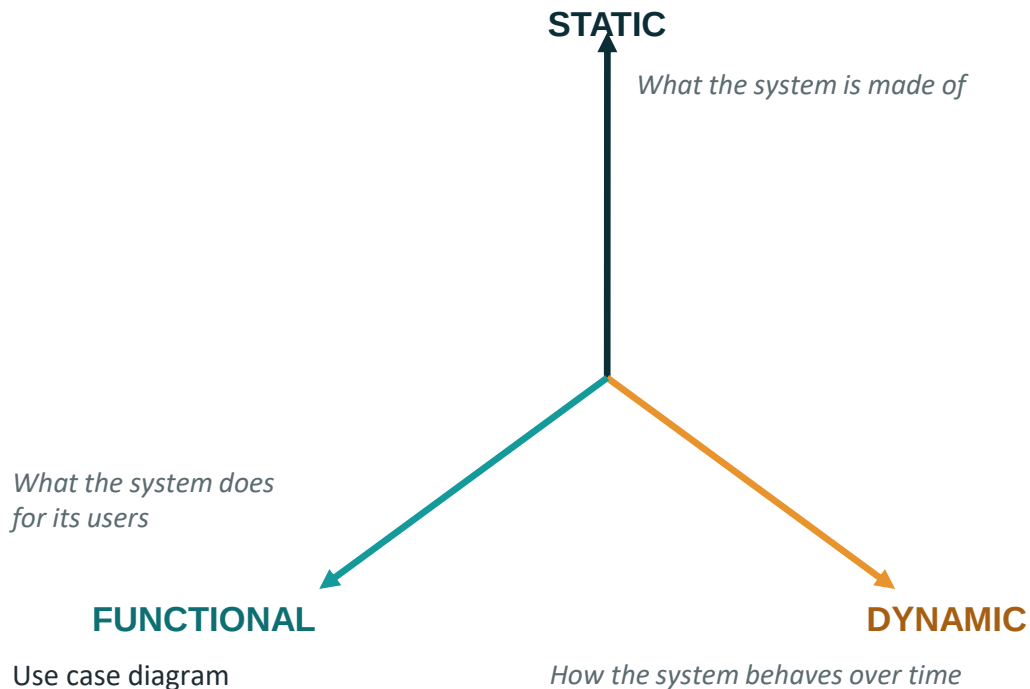
The temporal and technical view: concurrent tasks, stimuli, control, synchronization.



The deployment view

Describes the geographic position and the physical architecture of each element of the system.

The three modeling axes of UML



- Class diagram
- Object diagram
- Component diagram
- Deployment diagram
- Package diagram
- Composite structure diagram

- Sequence diagram
- Communication diagram
- Interaction overview diagram
- Timing diagram
- State transition diagram
- Activity diagram

04

UML diagrams

The 13 diagrams: what each one shows and when to use it

The 13 UML diagrams at a glance

A UML diagram is a graphical representation that focuses on one precise aspect of the model. Its level of detail characterizes the level of abstraction of the model. The notation is standardized.



Structure diagrams (6)

What the system is made of

- Class diagram
- Object diagram
- Component diagram
- Deployment diagram
- Package diagram
- Composite structure diagram



Behavior diagrams (7)

How the system behaves

- Use case diagram
- Sequence diagram
- Communication diagram
- Interaction overview diagram
- Timing diagram
- State transition diagram
- Activity diagram

Since UML 2.2 a 14th diagram exists — the profile diagram — used to extend UML itself.

Diagrams 1 to 3

The functional view and the static structure of the system.



1. Use Case Diagram

Identifies the possibilities of interaction between the system and the actors (external stakeholders): all the functionalities the system must provide.

Behavior



2. Class Diagram

Represents the classes involved in the system, with their attributes, their operations and the relations between them.

Structure



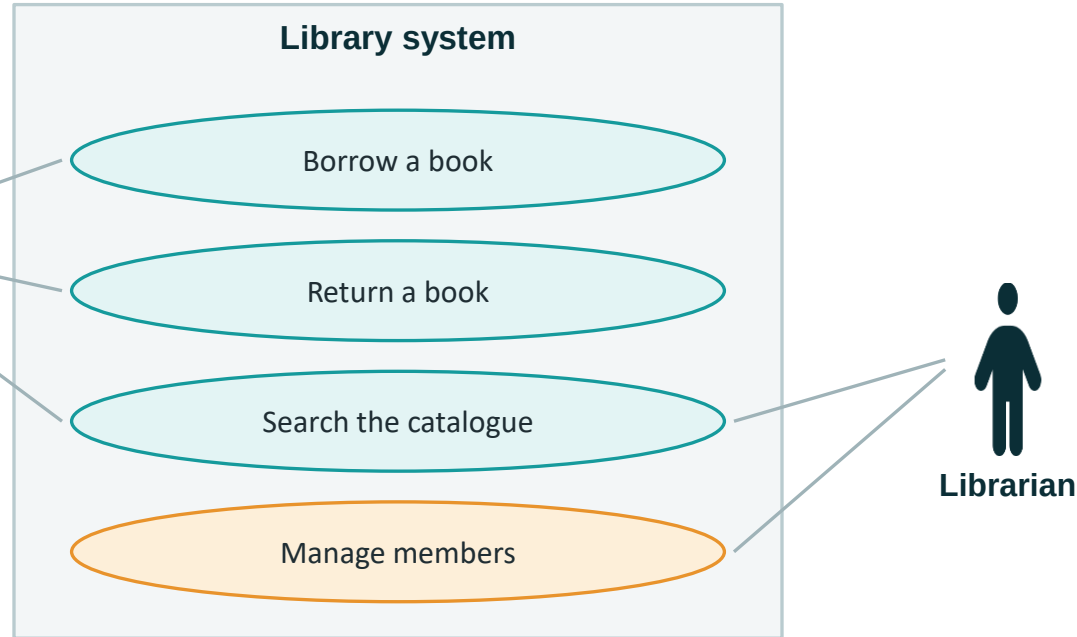
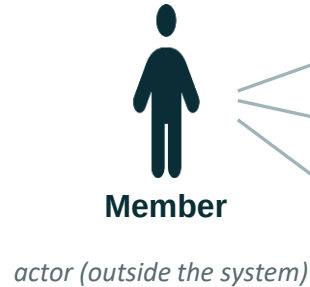
3. Object Diagram

Represents the instances of classes (objects) used in the system, with the values of their attributes at a given moment.

Structure

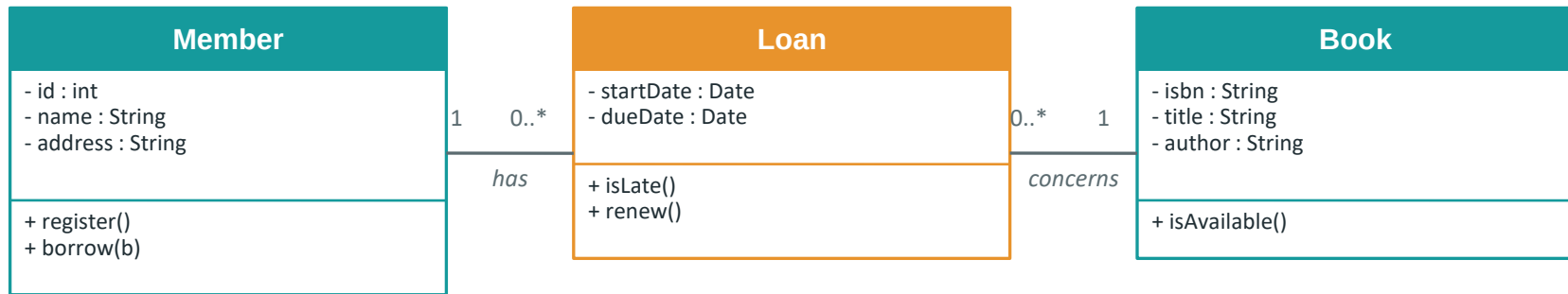
Example: use case diagram

Answers: who uses the system,
and what for?



the rectangle is the boundary of the system; each ellipse is one functionality

Example: class diagram



Class name

The name of the concept, in the coloured header.

Attributes

The data of the class. “-” means private (hidden).

Operations

The services of the class. “+” means public (offered).

Multiplicity

1, 0..*, 1..* : how many objects take part in the relation.

Diagrams 4 to 6

The physical organization of the system and the organization of the model itself.



4. Component Diagram

Shows the components of the system from a physical point of view, as they are implemented: files, libraries, databases, executables.

Structure



5. Deployment Diagram

Represents the hardware elements (computers, peripherals, networks, storage systems) and how the components are distributed on them.

Structure



6. Package Diagram

A package is a logical container that groups and organizes the elements of the model. This diagram shows the dependencies between packages.

Structure

Diagrams 7 to 9

The inside of a class, and the interactions between objects.



7. Composite Structure Diagram

Describes the relations between the components of a class in a “white box” format: what the class contains and how its parts are connected. (New in UML 2.x)

Structure



8. Sequence Diagram

Sequential representation of the processing and of the interactions between the elements of the system and its actors, over time.

Behavior

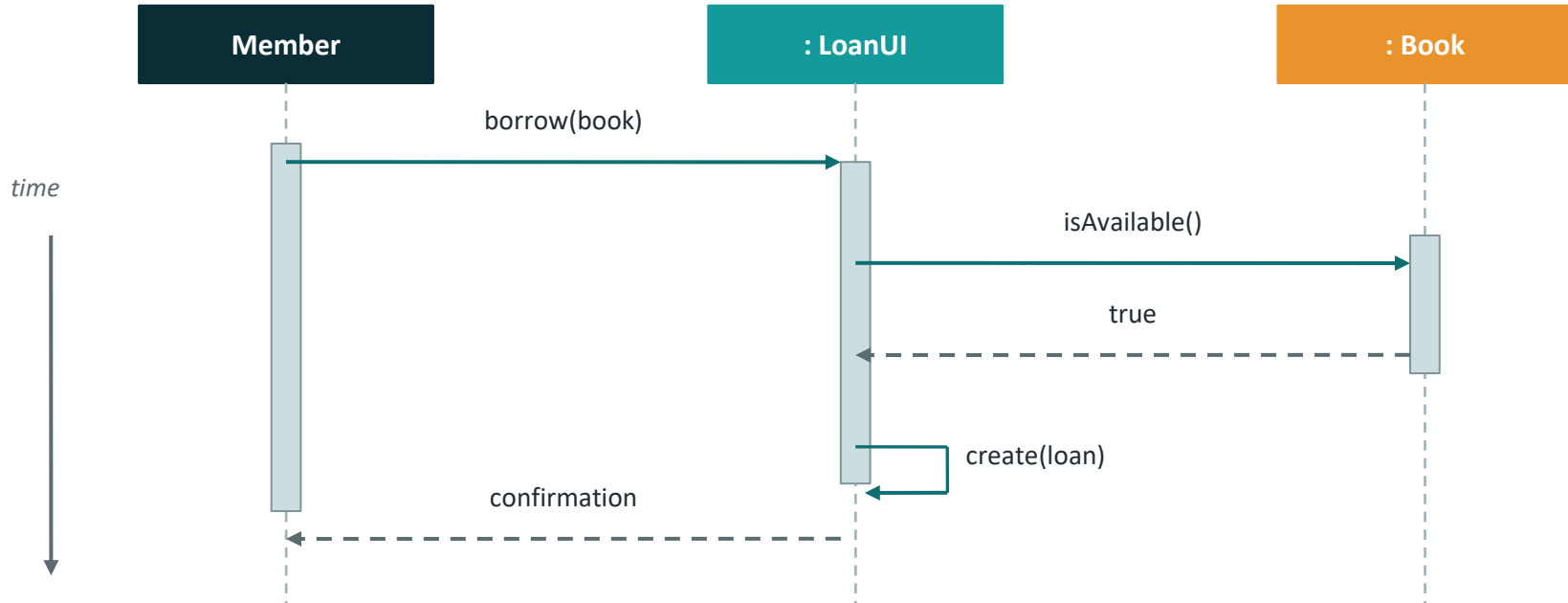


9. Communication Diagram

A simplified representation of a sequence diagram, which focuses on the exchange of messages between objects rather than on time.

Behavior

Example: sequence diagram



The vertical dashed line is the lifeline; the narrow rectangles are the activation bars.

Diagrams 10 to 13

Behavior over time: scenarios, states and activities.



10. Interaction Overview Diagram

Describes the possible sequences between scenarios that were already identified as sequence diagrams. It is a variant of the activity diagram. (UML 2.x)

Behavior



11. Timing Diagram

Describes the variations of a piece of data over time. Used for real-time and embedded systems. (UML 2.x)

Behavior



12. State Transition Diagram

Describes the behavior of the system or of one of its components in the form of a finite state machine: states and transitions.

Behavior

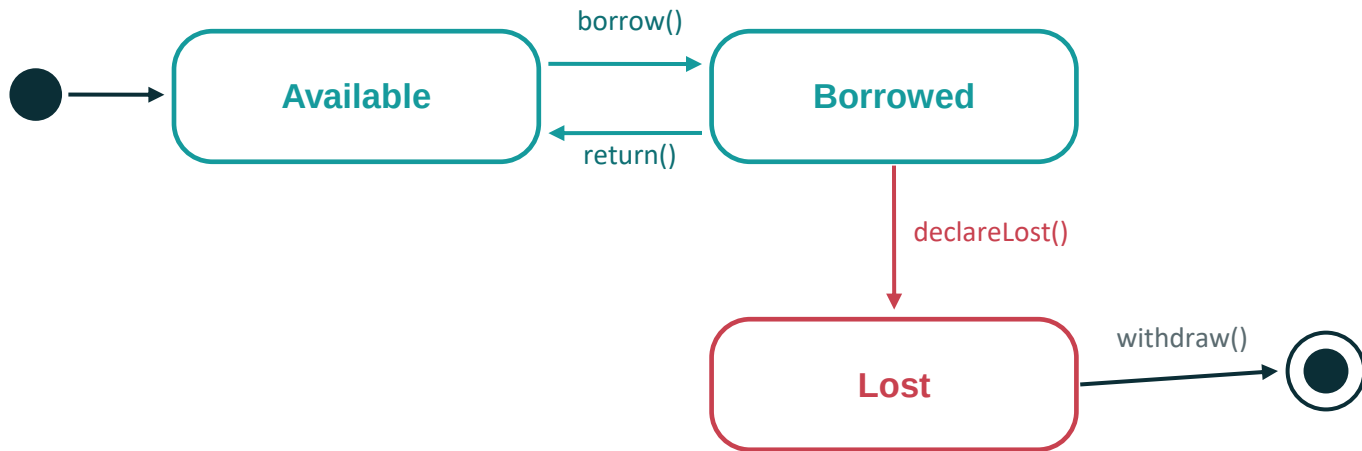


13. Activity Diagram

Describes the behavior of the system or of its components in the form of a flow, or chain, of activities.

Behavior

Example: state transition diagram



Initial state

the black dot: where the life of the object starts

State

a rounded rectangle: a situation in the life of the object

Transition

an arrow carrying the event that causes the change

Final state

the circled dot: the life of the object is finished

Use it for objects with a clear life cycle: an order, a reservation, a document, a copy of a book.

Summary: which diagram answers which question

Diagram	Axis	4+1 view	It answers the question...
Use case	Functional	Use case	Who uses the system, and what for?
Class	Static	Logical	Which concepts exist, and how are they linked?
Object	Static	Logical	What does the system look like at one precise moment?
Component	Static	Implementation	Which software parts make up the system?
Deployment	Static	Deployment	On which machines does each part run?
Package	Static	Implementation	How is the model organized, and what depends on what?
Composite structure	Static	Logical	What is inside a class, and how are its parts connected?
Sequence	Dynamic	Process	In which order are the messages exchanged?
Communication	Dynamic	Process	Who talks to whom?
Interaction overview	Dynamic	Process	How do the scenarios follow each other?
Timing	Dynamic	Process	How does a value change over time?
State transition	Dynamic	Process	Which states does an object go through?
Activity	Dynamic	Process	In which order are the activities carried out?

Do not try to draw all 13 diagrams: choose the ones that answer the questions you really have.

In practice: which diagrams do we really use?



REQUIREMENTS

Use case diagram

Capture what the users need, before anything else.



ANALYSIS

Class diagram

The heart of the model: the concepts of the domain and their links.



ANALYSIS / DESIGN

Sequence diagram

Describe the important scenarios, step by step.



DESIGN

State transition diagram

For objects with a real life cycle: order, reservation, document.

The others, when the project needs them: deployment diagram for a distributed system · component and package diagrams for a large architecture · timing diagram for real-time systems.

KEY TAKEAWAYS

What to remember from Chapter 2



A model is a simplification: it must help us understand and simulate.



The object approach groups data and processing in one entity.



Five OO concepts: object, encapsulation, inheritance, polymorphism, aggregation.



UML (1996) unifies Booch, OMT and OOSE — the work of the “three amigos”.



UML is a LANGUAGE, not a method: use it with a development process.



13 diagrams, 3 axes (static, dynamic, functional), 5 views (4+1).

Bibliography



UML 2 — pratique de la modélisation

Benoît Charroux, Yann Thierry-Mieg, Aomar Osmani

[Open link](#)



UML 2 par la pratique

Pascal Roques · Éditions Eyrolles



Les cahiers du programmeur — UML 2

Pascal Roques · Éditions Eyrolles



OMG Unified Modeling Language — the official specification

Object Management Group

[Open link](#)