ALGERIAN DEMOCRATIC AND PEOPLE'S REPUBLIC

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

UNIVERSITY OF RELIZANE

Faculty of Science and Technology

Department of Computer Science

SOFTWARE ENGINEERING

For Students in the 3rd Year – Bachelor Computer Science

BY DR. SEYYID AHMED MEDJAHED

2025-2026

Table of Contents

Table of Figures

Foreword

This course material is intended for 3rd Year Bachelor of Computer Science students. The primary objective of this module is to introduce students to the fundamental principles and practical applications of Software Engineering. The course aims to provide a deep understanding of the software development life cycle, software quality, modeling with UML, and best practices in designing, developing, testing, and maintaining software systems. Emphasis is placed on both theoretical concepts and practical skills to prepare students for real-world software engineering challenges.

Chapter 1: Introduction to Software Engineering

1. Definition and objectives of software engineering

The term software engineering is composed of two words, software and engineering.

Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be a collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called software product.

Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods.

IEEE defines software engineering as:

The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software.
Software Engineering is a discipline that aims to produce faultless software, delivered on time and on budget, and that meets the customer's needs. It represents the methodologies for successfully completing a complex IT project as a team by breaking down difficult problems into several smaller, easier-to-treat problems.

The life of a software is not limited to programming! Software Engineering help the team to respect the CQFD rule (Cost Quality Features Deadline).

Software engineering appeared in the 1970s to respond to the “software crisis”. This crisis emerged when we realized that the cost of software exceeded the cost of hardware. Another symptom of this crisis is the non-quality of the systems produced. The human and economic risks are significant.

Examples of some issues:

- In March 1993, the London Stock Exchange abandoned the **Taurus IT** project which was to completely monitor the execution of transactions. The system cost £60 million directly and market operators spent £400 million adapting their own software to it.
- VENUS mission: change to 500,000 km instead of 5,000 km due to the replacement of a comma (,) with a point (.).

- The **Therac-25**, a therapeutic irradiation device, caused the death of 2 peoples, and the irradiation of 4 others, due to a software error.
- An English warship was sunk by a French Exocet during the Falklands War. Results of the disaster: several hundred deaths. The English vessel had not, in fact, activated its defenses, the Exocet not being listed as an enemy missile.
- In 1983, the entire Colorado Valley was flooded. The cause? Poor modeling in the software of the dam opening time. (Inondée: mauvaise modélisation dans le logiciel du temps d'ouverture du barrage).
- A centenarian (106 years old) found herself summoned to school. The cause? the coding of the year of birth on two characters.
- Year 2000 bug. Cost of software upgrade in France: 500 billion francs.
- A victim in 2000 was fined $91,500 after returning a rented video tape (Cassette Video). The reason? The calculated delay being one hundred years because here again the coding of the year of the loans had been carried out on two characters, to save a little space.

2. Software Quality Issues

Software quality issues in software engineering refer to the problems or deficiencies that can arise during the development, deployment, or maintenance of software systems, which affect their performance, reliability, maintainability, usability, or compliance with requirements. These issues can significantly impact the success, cost, and user satisfaction of a software product.

Below is an overview of key software quality issues in software engineering:

- Poor team monitoring: no control, supervision and meetings.
- Technical problem: it is generally linked to poor programming and insufficient testing.
- Human problem: due to lack of communication, team spirit.

3. Quality of software

To avoid all these problems, the ISO 9126 standard has summarized the criteria for having quality software, we cite:

- **Reliability:** it is the confidence that a user can have in the software, for example, if the latter sends a message, is he sure of its arrival at destination.

- **Ergonomics:** this is the ease of use with or without training, whatever the skill, we also talk about ease of learning and the effort put into handling.
- **Extensibility:** is the ease of adapting to changes.
- **Integrity:** protect data against attacks.
- **Reusability :** writing code so that it can be reused in other software projects.
- **Portability:** the ability to be carried across multiple environments.
- **Maintainability:** the ability of the software to recover data in the event of an error.

4.Software life cycle

A software life cycle model (also called process model) is a descriptive and diagrammatic representation of the software life cycle. A life cycle model represents all the activities required to make a software product transit through its life cycle phases. It also captures the order in which these activities are to be undertaken. In other words, a life cycle model maps the different activities performed on a software product from its inception to retirement. Different life cycle models may map the basic development activities to phases in different ways. Thus, no matter which life cycle model is followed, the basic activities are included in all life cycle models though the activities may be carried out in different orders in different life cycle models. During any life cycle phase, more than one activity may also be carried out.

The different activities are:

- Feasibility.
- Specification: what is expected from this software as well as the client's needs.
- Project organization: what is needed to carry out this project, including cost analysis, planning, quality assurance, and task distribution.
- Design.
- Implementation: this involves programming and writing the source code.
- Testing.
- Delivery: installation, training, support.

- Maintenance.

Feasibility

This is a market study, i.e., whether it is worth creating this software or not. Before starting any software project, it is important to answer the following questions: Why and for whom are we developing this software? Is there a market for it? What is the budget and resources? Is the personnel qualified? All of this leads to answering the final question: Are we going to develop it or not?

Design

This is the stage where we ask the question, "How does the software work?" It includes both high-level design and detailed design.

- **High-Level Design:** This is the architectural description of the components, their interfaces, and their functionalities.
- **Detailed Design:** Here, additional details are added regarding the operation of the components.

Testing

Testing allows us to determine whether the software functions correctly. There are several types of tests:

- **Unit Test:** The software is tested by its developers.
- **System Test:** The software is tested in an environment similar to that of the client.
- **Alpha Test:** The client tests the software on the developer's site.
- **Beta Test:** Testing is conducted in the client's environment.
- **Acceptance Test:** Determines whether the client is satisfied.
- **Regression Test:** All tests and results are recorded to compare them with other versions.

Maintenance

It includes:

- **Perfective Maintenance:** This occurs following a change in the specifications.
- **Corrective Maintenance:** This is performed in case of errors.
- **Adaptive Maintenance:** This is necessary when there is a change in the environment.

3. Documentation Related to Software Development

- Requirements Specification: Initial description of desired functionalities.
- Specification (Needs Analysis).
- Project Schedule: Describes the order of tasks, deadlines, resources, etc.
- Software Test Plan: Describes the testing procedures.
- Software Design: Describes the structure of the software (How?).
- Quality Assurance Plan: Details the activities implemented to ensure quality.
- User Manual.
- Source Code.
- Test Report.
- Defect Report.

4. Lifecycle Types

Waterfall Lifecycle

This model was formalized in the 1970s. In this model, each phase ends after a fixed delay. At the end of each step, documents or software are produced. These results are thoroughly reviewed, and the next phase is only entered if these results are deemed satisfactory. After some modifications to the original waterfall model, the possibilities for feedback were added.

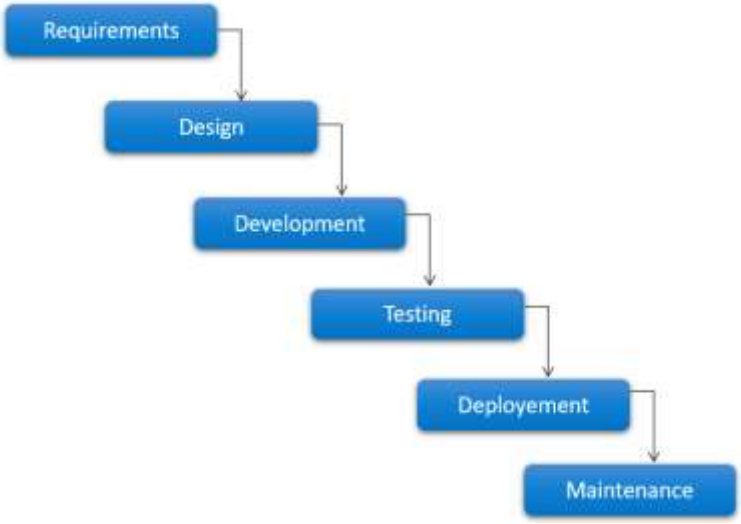


Figure 1. Waterfall lifecycle.

V Lifecycle

The V model is another way to present a process that remains linear. The V is traversed from left to right, following the shape of the letter: the construction activities precede the validation and verification activities. However, acceptance is prepared from the very beginning of the construction (arrows from left to right). This allows for a deeper focus on construction and better planning for the feedback process.

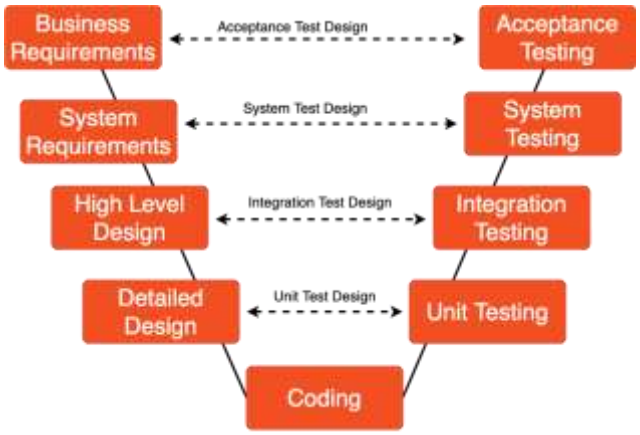


Figure 2. V lifecycle.

5. Requirements analysis

A very important step not to be overlooked → **Requirements analysis** (Analyse des besoins)

Study and critique of existing system are considered as the first step in software engineering. They involve a thorough examination and evaluation of current methodologies, tools, technologies, and practices in the field

Study of the Existing (State of the Art Analysis)

The analysis of the existing system allows for understanding the nature of the current system and describes the present solution within the domain of study in terms of organization.

The purpose of the analysis of the existing system is to identify the strengths and weaknesses of the current system. Thus, the analysis of the existing system provides a status report of the current system. We should do:

- Examination of current systems to understand methodologies, tools, and practices.
- Conduct a comprehensive review of academic papers, technical reports, and industry publications to gather information on current methodologies, technologies, and tools.
- Evaluate existing software development tools, frameworks, and platforms.
- Analyze current software development methodologies such as Agile, DevOps, Waterfall, and others.

Study real-world case studies to understand how current practices are applied in industry.

Critique of the Existing (Critical Analysis)

Evaluation of current systems to identify strengths, weaknesses, gaps, and improvement areas.

- Critiquing the existing system involves first identifying the positive and negative aspects of the current system and then proposing solutions from which the best option will be chosen for the proper functioning of the system.
- This step serves as the starting point of the study because it raises the group's awareness of the main issues and provides essential elements.

Example

Library management

Study of the Existing

The library is managed by a single person called the librarian. This individual is responsible for all manual operations, including:

- Inventory Register: Whenever a new document is acquired, the librarian registers this new document in the inventory register, noting the necessary information such as the call number, inventory number, title and subtitle, responsibility statement, publisher, date, ISBN, second ISBN, and keywords.
- Coding: The librarian assigns a code to each new document. This code consists of two parts. The first part consists of two letters, which represent the theme and the corresponding sub-theme. The second part is a sequential number. It is also important to note that the same documents with more than one copy share the same code. For example, the following coding will facilitate the identification and physical classification of the documents.
- Teachers are allowed to borrow a maximum of three documents at a time without a membership card; consequently, the librarian maintains a record for each teacher. In this case, the maximum loan duration varies from 7 to 20 days.
- To borrow documents, a member must present themselves at the counter with their membership card, indicating the relevant theme or sub-theme.
- The librarian searches for documents corresponding to these themes.
- If available, the member fills out a consultation request (including usage type (on-site or home), name, level, author, title, and signature) and submits it along with their membership card. The librarian then completes the loan procedure (noting the inventory number, call number, date of issue, and return date on the request) and finally hands over the document.
- If not available, the member can make a reservation. The librarian reserves the document for them.
- Similarly, to return the documents, the member must present themselves at the counter. The librarian receives the documents and returns their membership card.

Critique of the Existing

- Problem of Staff Duplication: There is not enough personnel; if the library manager is absent, the task becomes difficult and complicated.

- Lack of Research Tasks for Students: There is a complete absence of a manual or automated directory of available documents that should be accessible to students.
- Significant Volume of Information: This leads to complexity in performing tasks.
- Inadequate Coding: The coding lacks significance and consistency.
- Space Issues: There is not enough room for the library.

How to do requirements analysis

- Listen to the customer.
- Prepare questions and meetings.
- Read available documents.
- Prepare relevant questions.
- Know the role of each future user of the system.
- Think about all possible problems

Example of questions to ask

- Who is behind the request for this achievement?
- Who will use the proposed solution?
- What will be the environment of the solution?
- Who else should I contact?
- The questions must be complete and must include all the areas in question, this allows a good analysis in order to maximize the satisfaction of future users.

6. Type of requirements

We have two types of requirements:

- Functional Requirement: explicitly established.
- Non-Functional Requirement: expected, not expressed but necessary.

Functional Requirement

A functional requirement is a specification of behavior between inputs and outputs. It defines what a system should do, describing the specific behaviors or functions the system must perform. Functional requirements typically detail the necessary functionality that allows users to accomplish their tasks within the system, covering aspects such as calculations, data processing, user interactions, and system responses.

Non-Functional Requirement

A non-functional requirement specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. These requirements define the quality attributes, performance, and constraints of the system, focusing on how the system performs its functions rather than what functions it performs.

Example of requirements analysis

Scenario: A library wants to develop an application to manage its books, loans, and returns.

Requirements Collection

- **Interviews with library staff**:
    - Identify daily tasks, current issues, and desired improvements.
- **User questionnaires**
    - Understand readers' expectations regarding book searches, loan tracking, etc.
- **Observation**
    - Observe library operations to identify unexpressed needs.
- **Non-Functional Requirement**
    - The system must be accessible 24/7.
    - The system must be secure to protect user data.
- **Prioritization**
    - **Book search function**: High priority
    - **Borrowing and returning management**: High priority

- **Reminder notifications**: Medium priority
- **24/7 accessibility**: High priority
- **Data security**: High priority

- **Deliverable**: A requirements document containing a detailed list of prioritized functional and non-functional requirements.

7. Complete example

Library Management Application

Develop an application to manage books, loans, and returns in a library.

Requirement Analysis

Objective: Develop an application to manage books, loans, and returns in a library.

Stakeholders

- Librarians
- Library users (patrons)
- IT administrators

Information Gathering

- **Interviews with Library Staff**: Identify daily tasks, current issues, and desired improvements.
- **User Questionnaires**: Understand readers' expectations regarding book searches, loan tracking, etc.
- **Observation**: Observe library operations to identify unexpressed needs.

Functional Requirements

1. Book Management

- The system must allow the librarian to add, edit, and delete book records.
- The system must provide a search functionality for books by title, author, ISBN, and category.

- The system must track the availability status of each book (available, on loan, reserved).

2. Loan Management

- The system must allow the librarian to record the loan of a book to a library user.
- The system must update the status of a book when it is loaned out.
- The system must allow users to view their current loans and due dates.

3. Return Management

- The system must allow the librarian to record the return of a book.
- The system must update the status of a book when it is returned.
- The system must automatically calculate and record late fees, if applicable.

4. Notifications

- The system must send reminders to users for upcoming due dates.
- The system must send overdue notices to users who have not returned books on time.

Non-Functional Requirements

1. Accessibility

- The system must be accessible 24/7 through a web-based interface.

2. Security

- The system must secure user data and book records with authentication and authorization mechanisms.
- The system must ensure data integrity and prevent unauthorized access.

3. Performance

- The system must handle up to 1000 concurrent users without performance degradation.

4. Usability

- The system must have an intuitive and user-friendly interface for both librarians and users.

5. Reliability

- The system must have an uptime of 99.9%.

Functional Requirements Specification

1. Book Management

- **Add Book**:
  - **Description**: The system must provide a form for the librarian to add new book records.
  - **Fields**: Title, Author, ISBN, Category, Description, Availability status.
  - **Priority**: High
- **Edit Book**:
  - **Description**: The system must allow the librarian to edit existing book records.
  - **Fields**: Title, Author, ISBN, Category, Description, Availability status.
  - **Priority**: High
- **Delete Book**:
  - **Description**: The system must allow the librarian to delete book records.
  - **Priority**: Medium
- **Search Book**:
  - **Description**: The system must provide search functionality by title, author, ISBN, and category.
  - **Priority**: High

2. Loan Management

- **Record Loan**:
    - **Description**: The system must allow the librarian to record the loan of a book to a user.
    - **Fields**: User ID, Book ID, Loan Date, Due Date.
    - **Priority**: High
- **Update Status on Loan**:
    - **Description**: The system must update the status of a book when it is loaned out.
    - **Priority**: High
- **User View Loans**:
    - **Description**: The system must allow users to view their current loans and due dates.
    - **Priority**: High

3. Return Management

- **Record Return**:
    - **Description**: The system must allow the librarian to record the return of a book.
    - **Fields**: User ID, Book ID, Return Date.
    - **Priority**: High
- **Update Status on Return**:
    - **Description**: The system must update the status of a book when it is returned.
    - **Priority**: High
- **Calculate Late Fees**:
    - **Description**: The system must automatically calculate and record late fees, if applicable.

- **Priority**: Medium

4. Notifications

- **Reminder Notifications**:
  - **Description**: The system must send reminders to users for upcoming due dates.
  - **Frequency**: 3 days before the due date.
  - **Priority**: High
- **Overdue Notices**:
  - **Description**: The system must send overdue notices to users who have not returned books on time.
  - **Frequency**: 1 day after the due date.
  - **Priority**: High

Non-Functional Requirements Specification

- **Accessibility**
  - **Requirement**: The system must be accessible 24/7 through a web-based interface.
  - **Priority**: High
- **Security**
  - **Requirement**: The system must secure user data and book records with authentication and authorization mechanisms.
  - **Priority**: High
  - **Requirement**: The system must ensure data integrity and prevent unauthorized access.
  - **Priority**: High
- **Performance**

    - **Requirement**: The system must handle up to 1000 concurrent users without performance degradation.
    - **Priority**: High
- **Usability**
    - **Requirement**: The system must have an intuitive and user-friendly interface for both librarians and users.
    - **Priority**: High
- **Reliability**
    - **Requirement**: The system must have an uptime of 99.9%.
    - **Priority**: High

8. Documents

Specification document

A **specifications document** is a comprehensive written description of the requirements, features, and functionalities of a system, product, or project. It serves as a guideline for developers, designers, and stakeholders, detailing what needs to be built and how it should function. The document outlines both functional and non-functional requirements, providing clarity and a shared understanding of the project scope.

Requirements analysis focuses on the **what** and **why** of the system, while specification focuses on the **how**.

Requirements specification

The requirements specification (Cahier des charges) contains global descriptions of the functions of a new product or extensions of an existing product, and it must be validated by the client. It serves as the basis of the contract between the client and the developer.

The template for a requirements specification is as follows:

1. Cover Page

- Name of the project/software/product;

- Date;
- Version number;
- Stakeholders;
- Responsibilities of each stakeholder;
- Key changes since the previous version.

2. Introduction

- A summary containing the objectives;
- Deliverables: a list of what will be provided to the client (software, hardware, etc.);
- Definitions and acronyms.

3. Project Organization

- Decomposition of the project over time;
- The role of each stakeholder in the development.
- The limits;
- The interactions with existing hardware or software.

4. Management

- Define the objectives and their priorities;
- Define dependencies with other systems;
- Identify constraints;
- Risk management.

5. Techniques

- This section describes the methods and tools used.

6. Schedule and Budget

- This involves breaking down the project into phases (intermediate deliveries, interim payments, etc.), as well as the resources, budget, schedule, and any factors that may influence the timeline.

7. Functional Requirements

8. Non-Functional Requirements

9. Context of the Future System

Chapter 2: UML Modeling

1. Modeling

Modeling a system before its implementation allows for a better understanding of how the system works. It is also a good way to manage its complexity and ensure its coherence.

A model serves as a common, precise, and comprehensive language due to its graphical nature. In this regard, it is an effective means to enhance communication, which is essential for achieving a common understanding among the various stakeholders involved in the project to be realized.

2. Object-Oriented Modeling

Object-Oriented Modeling allows for a simplification of the real world. It addresses the challenge of finding the right level of abstraction for concepts from the real world.

An object is a structured entity characterized by attributes and methods. The object-oriented approach is a way to tackle a problem by breaking it down into smaller sub-problems: it begins by identifying the objects in the system and then their interactions. For example, the creation of a bank account requires interactions between the client and a customer advisor.

3. UML Definition

Unified Modeling Language (UML) is a standardized modeling language used in software engineering to visualize, specify, construct, and document the artifacts of a software system. UML provides a set of graphic notation techniques to create visual models of software systems, facilitating better understanding and communication among stakeholders.

UML (Unified Modeling Language) is a graphical language that allows for the representation and communication of different aspects of an information system. It is the most widely used modeling notation in the world.

UML is an object-oriented modeling language, meaning that all modeled entities are objects or relate to objects.

UML has become the industry standard.

4. UML diagrams

UML comprises thirteen different diagrams, each dedicated to representing particular concepts of a software system. Furthermore, UML models the system using two modes of representation:

1. The first mode represents the static structure of the system.
2. The second mode represents the dynamic behavior of the system.

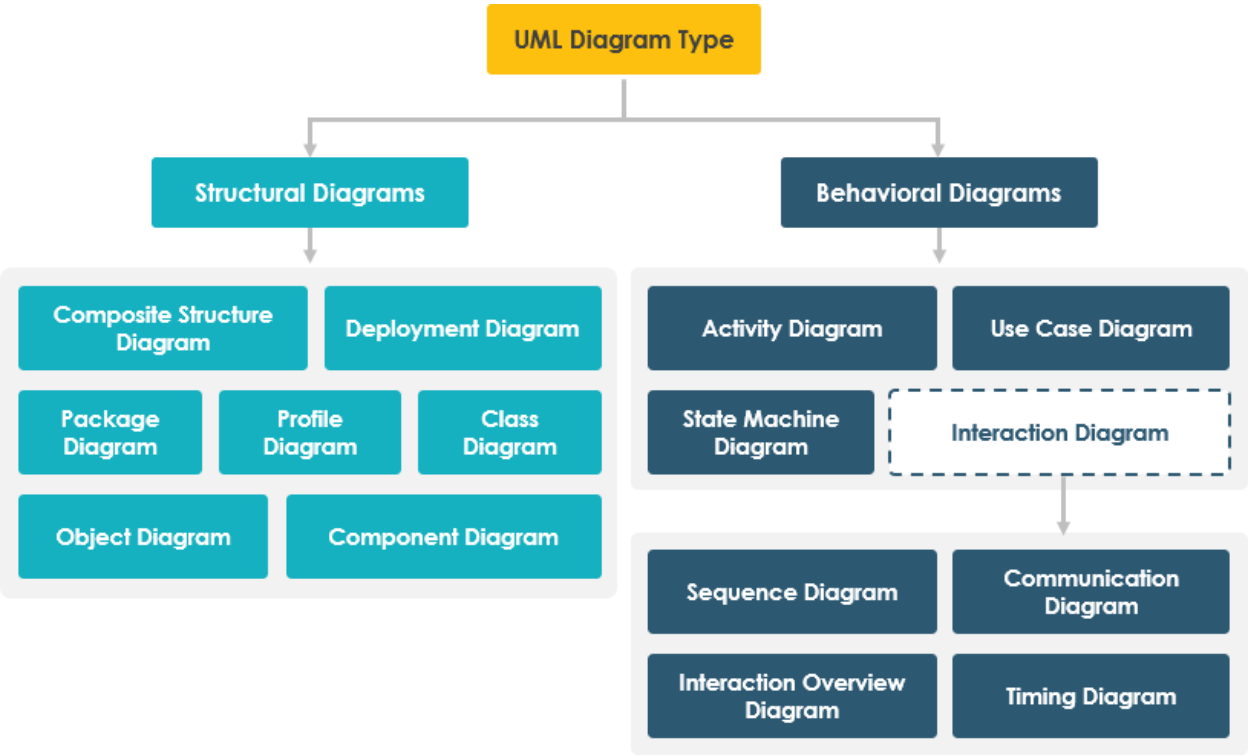


Figure 3. UML Diagram Type (https://guides.visual-paradigm.com).

5.UML views

In UML (Unified Modeling Language), there are three main views for representing a computer system: the functional view, the static view, and the dynamic view.

Functional view

This view describes the functions and services that the system must provide. It is represented by use case diagrams, which show the interactions between users (actors) and the system.

Focuses on the functionalities that the system provides, often capturing the interactions between users (actors) and the system.

Static view

This view describes the static structure of the system, meaning the entities (objects, classes) that make up the system and the relationships between them. This view is represented by **class diagrams**, **object diagrams**, or **component diagrams**. This view is important for understanding the structure of the system and facilitating its design and maintenance.

Represents the structure of the system, focusing on how the system is organized and the relationships between its components.

Dynamic view

This view describes the dynamic behavior of the system, meaning the events that occur and the interactions between entities. This view is represented by **sequence diagrams**, **state diagrams**, or **activity diagrams**. This view is important for understanding how the system operates and how it reacts to different situations.

Captures the behavior of the system over time, illustrating how the system responds to events and how its components interact during execution.

6. Package Diagram

A computer system can easily contain hundreds of classes or modeling elements. To manage this complexity, UML provides the concept of packaging, which organizes a namespace.

A **package diagram** is used to divide a system into subsystems to better understand and model it.

A **package diagram** is a type of UML diagram that shows how different packages (groups of related classes or components) are organized in a system. It illustrates the dependencies between these packages, providing a higher-level view of the system's structure and organization. Package diagrams are particularly useful for managing complexity in large systems by allowing developers to understand how different components interact.

Example: Hotel management

We divide our system into several packages:

- Customer Management

- Employee Management

- Statistics Management

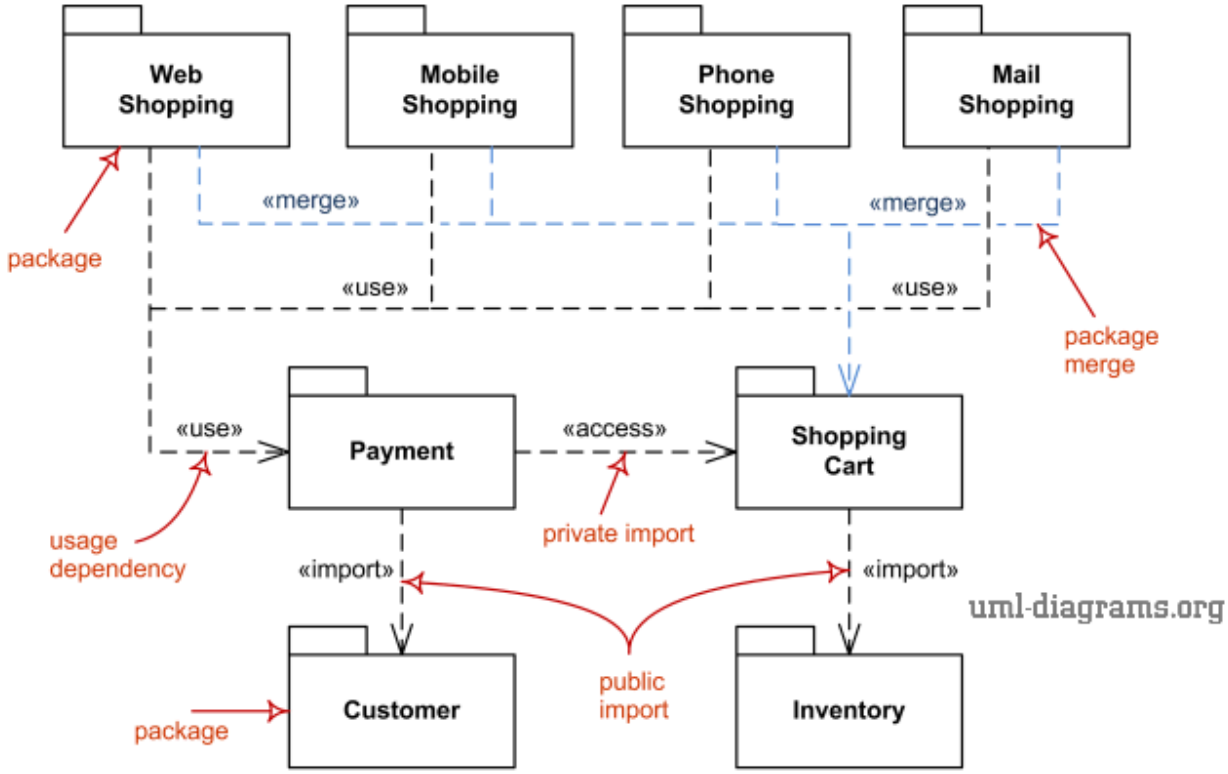


Figure 4. Package Diagram.

Chapter 3: Use Case Diagram

1. Definition

The Use Case Diagrams are created to answer the question "**WHAT?**".

Use cases delineate the system, its functions (its cases), and its relationships with its environment. They are a means of determining the system's needs. They allow users to be involved from the early stages of development to express their expectations and needs ("Requirements Analysis" Needs Analysis).

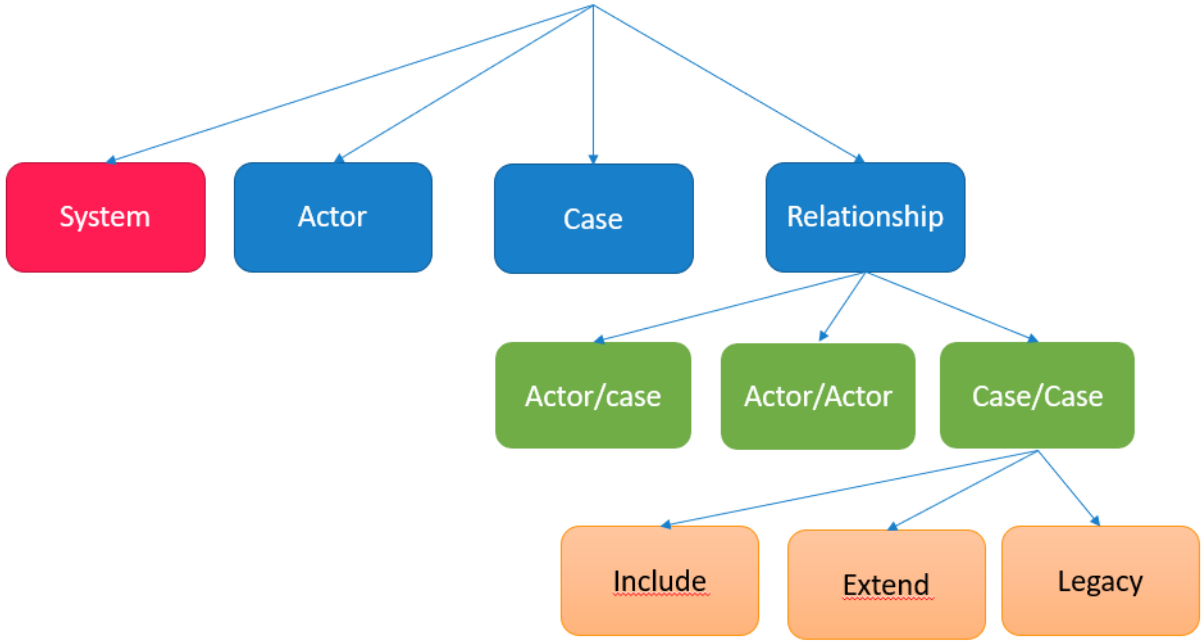


Figure 5. Components of Use Case Diagram.

2. The system

Functions are performed within a system.

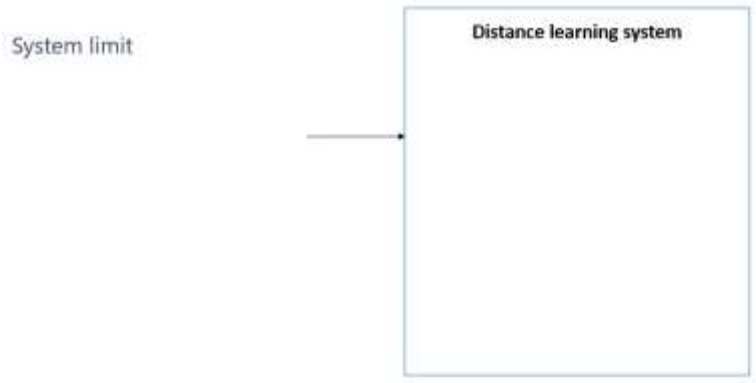


Figure 6. The system.

Example

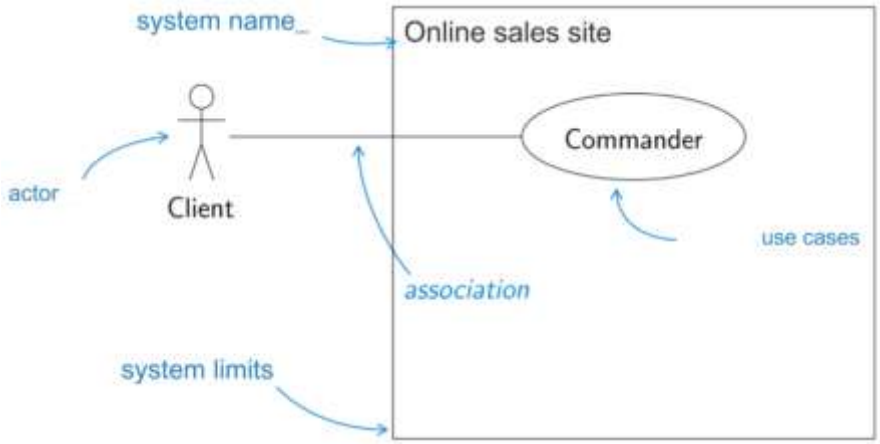


Figure 7. Example of Use Case Diagram.

3. The actor

The actor is the one who triggers (executes) the function

It is either human (represented by a stick man) or material or system represented by a binder (classeur).

The actor represents the role that a user or system plays.

Figure 8. The actor.

4. The use case

This is an ellipsis with a verb phrase. It models a feature or service provided by the system. A function is an infinitive verb.

Figure 9. The use case.

5. Association

Actor/Case

A line between the Actor and the Use Case represents the relationship between the two, i.e. the possibility that the Actor triggers the Use Case.

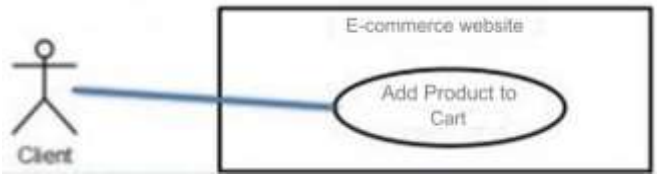


Figure 10. Actor/Case.

Generalization (Actor/Actor)

The inheriting actor can do everything the father actor can do + his own Cases.

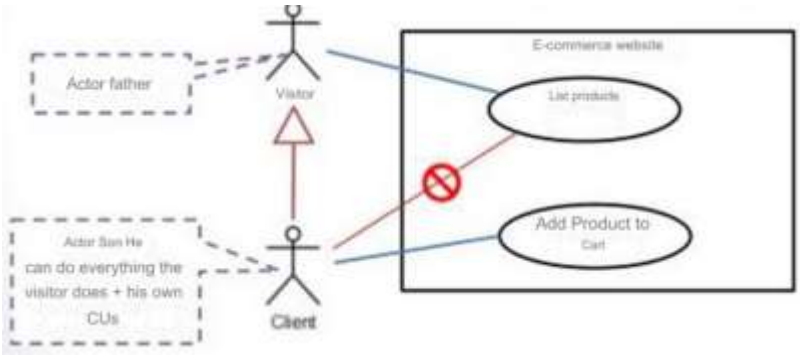


Figure 11. Generalization.

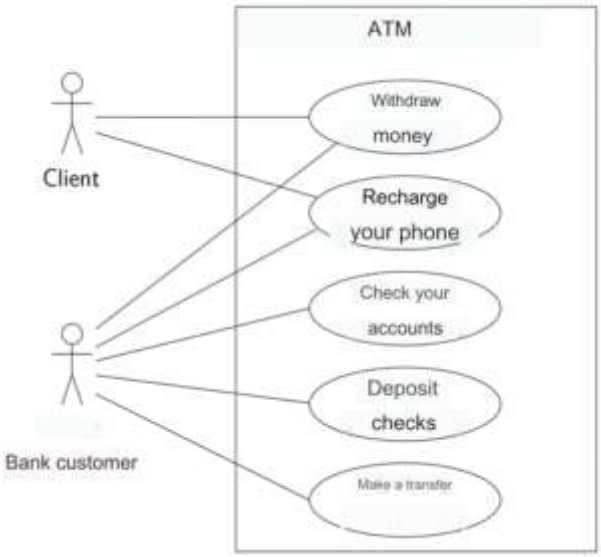
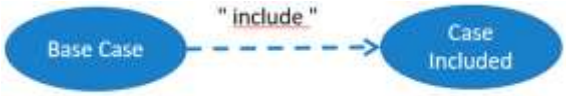


Figure 12. Example of generalization.

Case/Cases

1. Inclusion: “include” (Include case)



2. Extension: “Extend” (Optional case)

3. Generalization (Special case)

Example

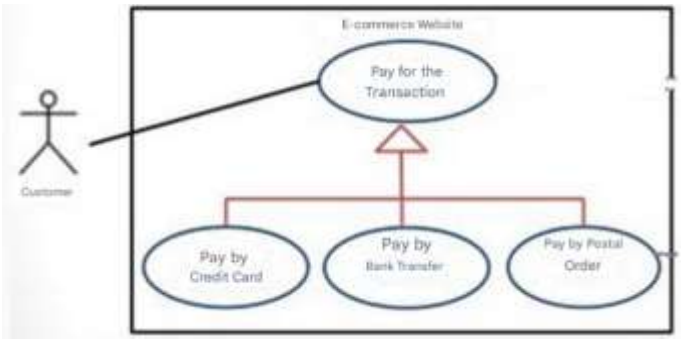


Figure 13. Example of include.

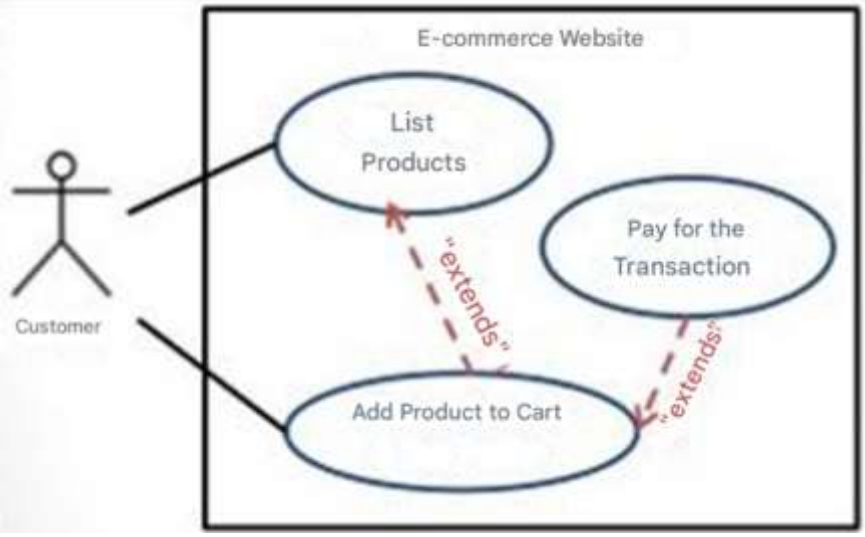


Figure 14. Example of extends.

6. Textual Description of the Use Case

The textual description of a use case is a document that describes in a detailed and comprehensible manner the different steps and actions involved in the use of a system or application. It provides a clear overview of the usage scenario, emphasizing the objectives, the involved actors, the preconditions, and the expected outcomes. The textual description of a use case generally includes the following elements:

- Title: The title concisely identifies the specific use case.
- Actors: The actors are the people, external systems, or entities that interact with the system or application to achieve a specific objective.
- Preconditions: Preconditions describe the initial state required of the system or application before the use case can be successfully executed. This may include conditions such as network connection, the existence of specific data, or access to certain functionalities.
- Main Flow: The main flow sequentially describes the primary steps of the usage scenario. It details the specific actions performed by the actors and the system's responses to these actions.
- Extensions: Extensions describe variations or alternative paths of the main flow. They identify decision points where different actions may be taken based on certain conditions.

- Postconditions: Postconditions describe the expected final state of the system or application after the successful execution of the use case. This may include state changes, displayed results, or specific triggered actions.
- Exceptions: Exceptions are exceptional situations that may occur during the execution of the use case. They describe potential issues or errors that may arise and how they should be managed.

Example

The corresponding textual description is as follows:

I. Identification

- Case No. 1
- Name: Reserve Room
- Actor: Teacher
- Description: The room reservation is made by a teacher after verifying availability.
- Pre-condition: Room availability must be checked.
- Start: The teacher clicks on the reserve room button.

II. Description of Scenarios

Nominal Scenario

1. The user requests the reservation of a classroom.
2. The system displays the room reservation page.
3. The user enters a room and a date for the reservation.
4. The system checks the availability of the room (see UC check availability).
5. The system requests confirmation of the reservation.
6. The user confirms the reservation.
7. The system displays a message indicating the successful completion of the operation.

Alternative Scenarios

1. The user enters an incorrect room number.

2. The system displays an error message and returns to step 3.

3. The user may choose not to confirm the reservation (end of UC in failure).

Exception Scenarios

1. Power outage before confirmation of the reservation:

   - Reservations remain saved for 24 hours.

End and Post-conditions

- End: Nominal scenario, alternative scenarios.
- Post-conditions: The room must be reserved for that date.

Chapter 4: Class and Object Diagram

1. Introduction

As we mentioned in the previous chapter, the use case diagram shows the major functionalities of the system and the actors triggering them. The class diagram shows us the internal structure of the system. It provides an abstract representation of the system objects that will interact to realize the use cases.

This is a static view, because the temporal factor in the behavior of the system is not considered. The class diagram models the concepts of the application domain by presenting the classes of the system and their relationships.

The main elements of this static view are classes and their relationships: association, generalization, and several types of dependencies.

A class diagram is a type of static structure diagram in the Unified Modeling Language (UML) that describes the structure of a system by showing its classes, attributes, operations (or methods), and the relationships among the objects. Class diagrams are the main building block of object-oriented modeling and are used for both conceptual (high-level) and detailed (low-level) design of a system.

An object diagram is a type of static structure diagram in UML that shows a complete or partial view of the structure of a modeled system at a specific time. Unlike class diagrams, which depict abstract classes and their relationships, object diagrams show instances of classes (objects) and the

relationships between them at a particular moment. Object diagrams are useful for modeling the static design view of a system.

2. Class and object diagram

A class diagram represents the internal structure of the software.

An object diagram represents the state of the software (objects + relationships). It is a diagram that evolves with the execution of the software (creation and deletion of objects, modification of the state of objects and their relationships).

- Object: A concrete entity of the application domain.
- Described by:
    - Identity (memory address)
    - State (attributes)
    - Behavior (operations)

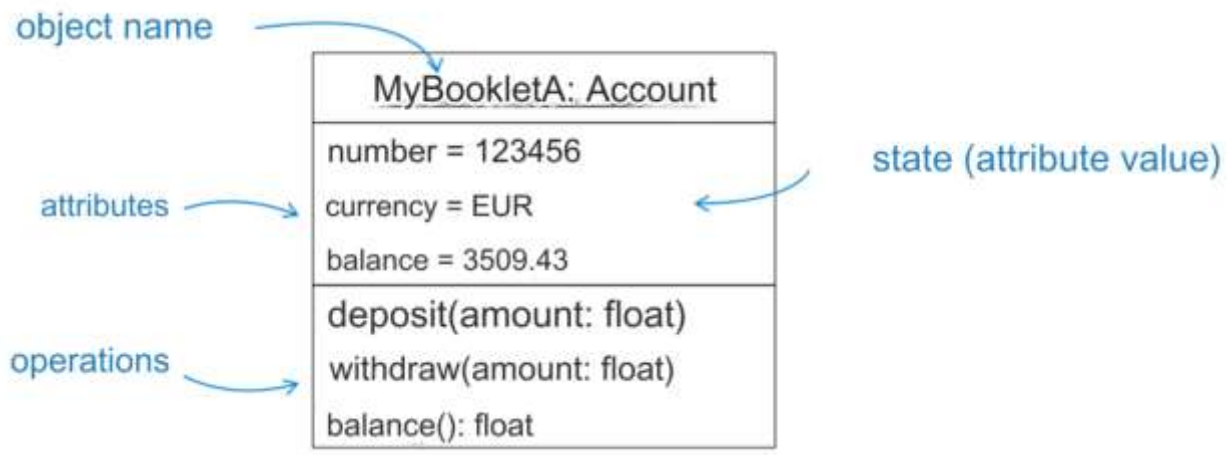


Figure 15. Exemple of Object.

3. Class

Grouping of objects of the same nature (same attributes + same operations).

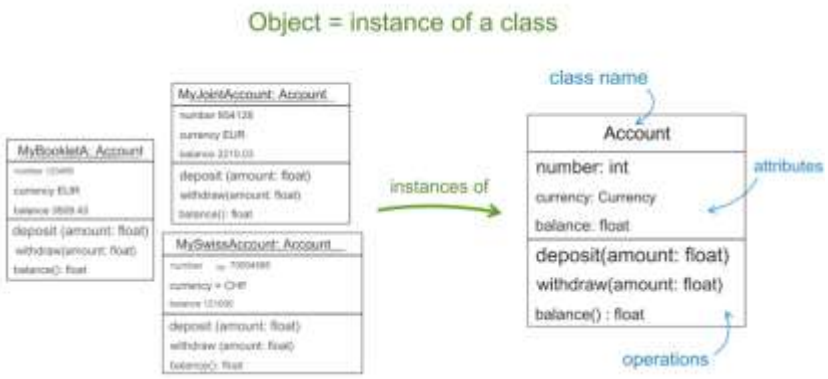


Figure 16. Class.

A class is the abstract description of a set of objects that are part of the same domain (possessing the same characteristics).

A class is represented by a workbook divided into three parts:

- The name of the class that begins
- always with a capital letter.
- The attributes that represent the data of the object.
- The operations that represent the behavior of an object.

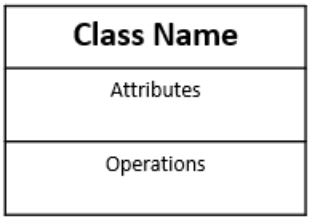


Figure 17. Example class.

Example

Instantiating the CAR class is creating an object of the CAR class. An instance of the CAR class is a given car (an object "of type" CAR)

Figure 18. Example of class.

Encapsulation

The user can use a class (create/manipulate objects) without knowing the body of the class. If the class is properly defined, the user cannot violate the integrity of the object [e.g., transform an ordered list into an unordered list].

The implementation is hidden and not accessible to the user.

The legacy

An object of the class "Motor Vehicle" inherits the properties of the class "Vehicle". So, such an object has a weight, a size and a price. In addition, it can stop and start. In addition, it has a power and a top speed (properties that vehicles without an engine do not have).

“VehicleMotor" is said to be derived (or inherited) from the class "Vehicle".

These are multiple forms for the same operation.

Example

It is impossible to calculate the surface area of a graphic shape if we have no information about this shape [the graphic shape class is therefore an *abstract class]*.

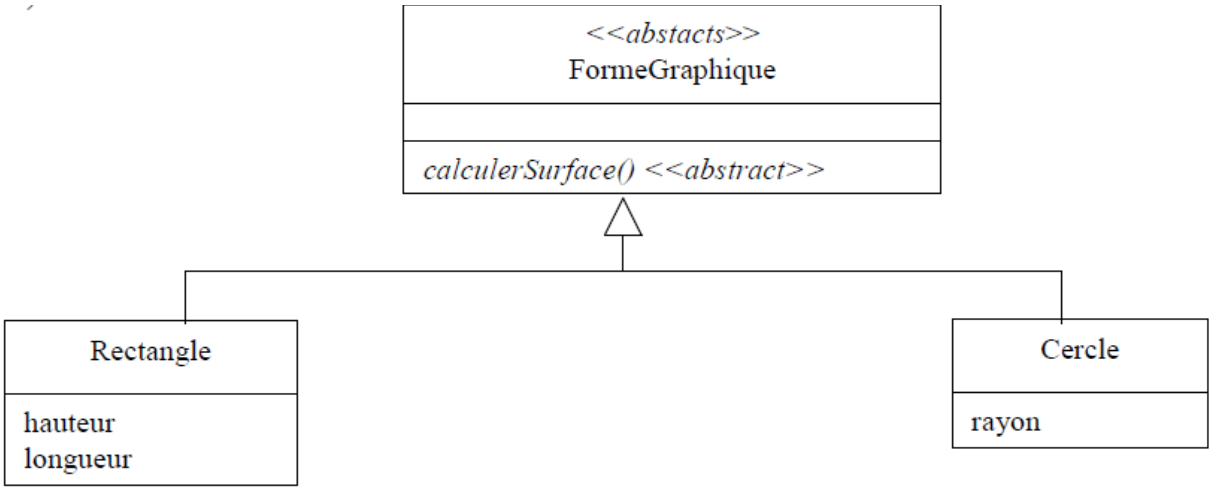


Figure 19. Exxample of classes.

4. Class diagram

A class diagram shows the static structure of a system. It allows the visualization of classes and the relationships between them. Its purpose is to explain what to achieve rather than to explain how to achieve it.

Visibility

- private: accessibility only by the object. Attributes are generally of "private" visibility.

+ public: accessible by any other object. Operations are generally "public".

protected: accessible by objects of subclasses (derived classes) of the initial class.

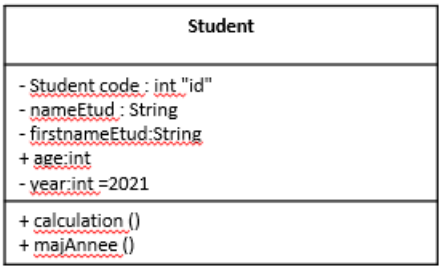


Figure 20. Visibility.

The attribute

The syntax of an attribute is as follows:

< visibility > AttributeName : AttributeType = DefaultValue

visibility : + public / # protected / - private

attributeName : name of the attribute

type: type of the attribute

DefaultValue : initialization value

The operation (method)

OperationName (ArgumentName : ArgumentType = DefaultValue , ...): Return type

- visibility : + public / # protected / - private
- nameOperation : name of the operation
- [<param1>, ..., <paramN>] : list of parameters.
- typeReturn : return type of the operation

Signature of a method

The Signature of a method includes the list of arguments with their type and the type returned by the operation

- example: ***calculateAge (dateNaiss : Date): int***
- Polymorphism is the definition of multiple signatures for the same operation. Example:
- ***calculationAge (dateBirth : Date): int***
- ***calculateAge () : void***

Association

Associations are binary. An association can only be named using roles.

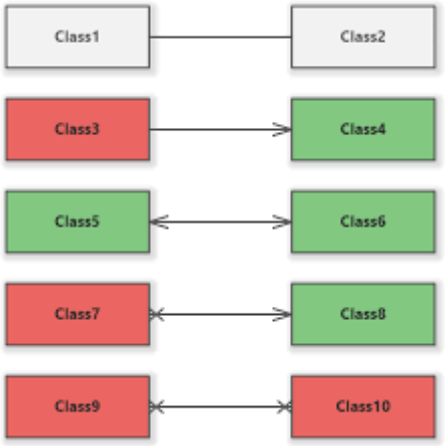


Figure 21. Association.

Multiplicity

Each association termination can define the following multiplicities:

- 1: This is the default value for any ending that specifies that there is one and only one element.
- 0..1: Possibility of having 0 or 1 element.
- 1..*: Possibility of having 1 to n elements.
- 0..*: Possibility of having 0 to n elements.
- n: Sets a specific number of elements.
- n..m: Defines a specific range of elements.
- * : equivalent of 0..*

Aggregation and composition

Aggregation

Aggregation is a type of association that represents a "whole-part" relationship between objects, where the whole and the part can exist independently. It is often described as a weak ownership relationship. In aggregation, the part may belong to multiple wholes or may continue to exist even if the whole is destroyed.

For example, in a university system, a Department and a Professor can be related through aggregation. A department may include several professors, but professors can still exist without

being assigned to any department. In this case, the department aggregates the professors, but it does not strictly own or control their entire lifecycle.

In UML diagrams, aggregation is depicted with a hollow diamond at the whole (owner) end of the association line.

Composition

Composition is a stronger form of association, where the part is strictly owned by the whole, and its lifecycle is fully dependent on that of the whole. This means if the whole is destroyed, the parts that belong to it are also destroyed. It reflects a tight coupling between the container and the contained objects.

For instance, consider a House and its Rooms. A room cannot meaningfully exist outside of the house it belongs to. If the house is demolished, the rooms are as well. This kind of relationship is modeled as a composition.

In UML, composition is represented using a filled (black) diamond at the owner's end of the association line.

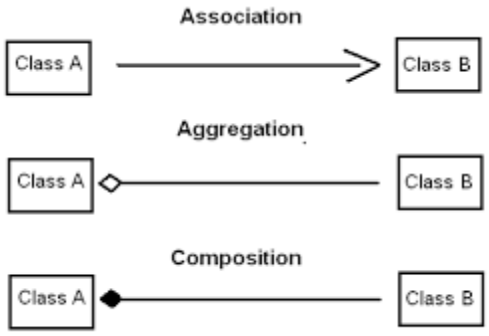


Figure 22. Aggregation and Composition.

Chapter 5: UML Diagram. Dynamic View

1. What is an interaction?

It is a behavior that involves a set of messages exchanged between a set of objects to accomplish a goal. It is used to model dynamicity and shows the interaction of objects along lifelines representing a general order (sequence).

We saw previously that each object has attributes that describe the information about the latter.

An object has a described behavior by operations.

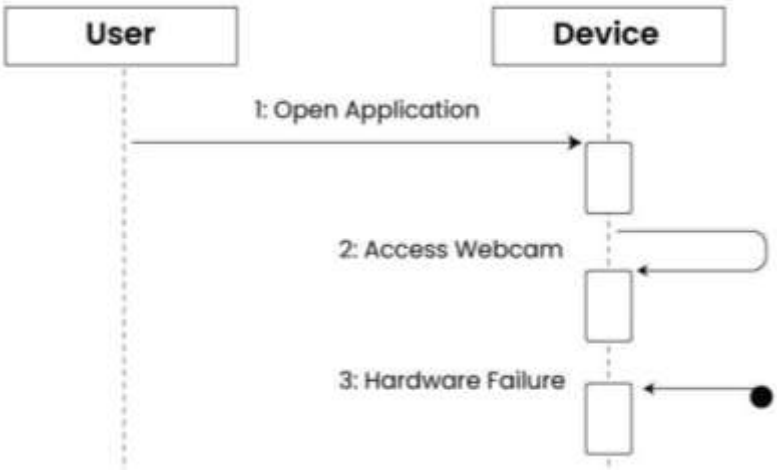


Figure 23. Diagram Sequence.

An object can send and receive messages.

These messages are sent between objects to execute methods.

A message is a function or procedure called by one object for another object.

2. How to draw a sequence diagram

Life line

Represented by a rectangle to which is attached a dotted vertical line containing a label:

< object_name (or actor)>:< its_type >

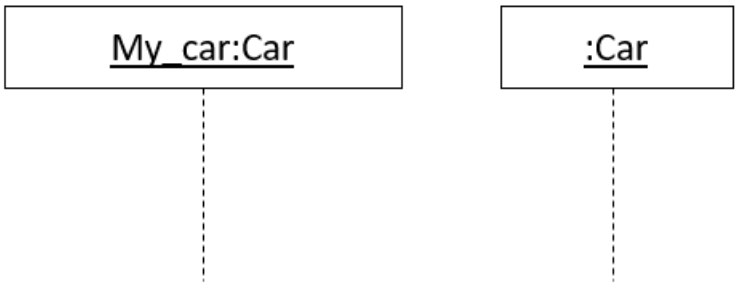


Figure 24. Lifeline.

Messages

It is a special communication between the life lines, it can be:

- Sending a signal
- Invoking an operation
- Creating or destroying an instance

3. Types of messages

Asynchronous message

Event, interrupt are examples of signals, they do not wait for a response and do not block the transmitter.

Synchronous message

The transmitter remains blocked for the duration of the operation invocation.

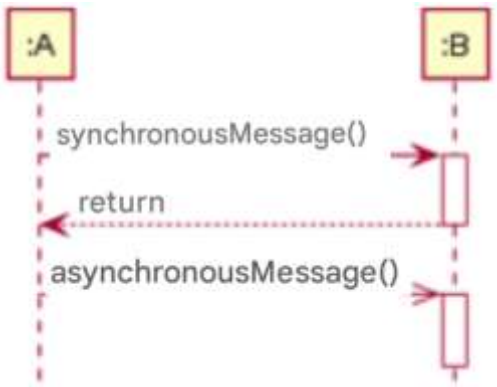


Figure 25. Asynchronous/Synchronous message.

Instance creation and destruction message

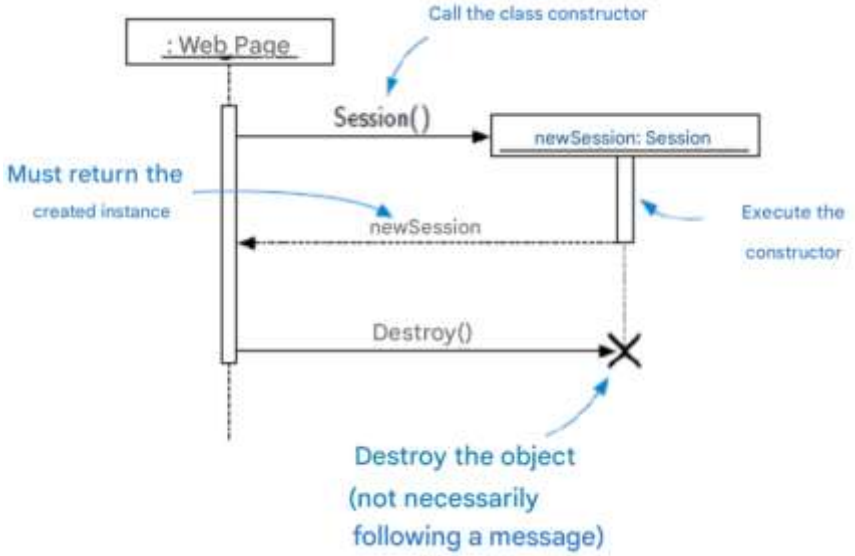


Figure 26. Instance creation and desctruction message.

Answer

In most cases, the reception of a message is followed by the execution of a class method.

Syntax of a response:

Attribute= message:return_value

Lost message and found message:

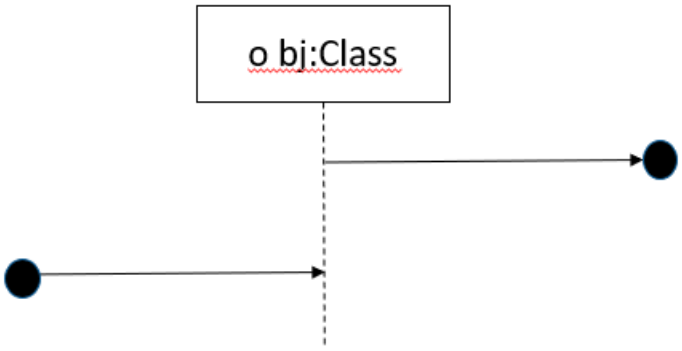


Figure 27.Lost/found message.

4. Types of fragments

alt

Express it (if… then … else)

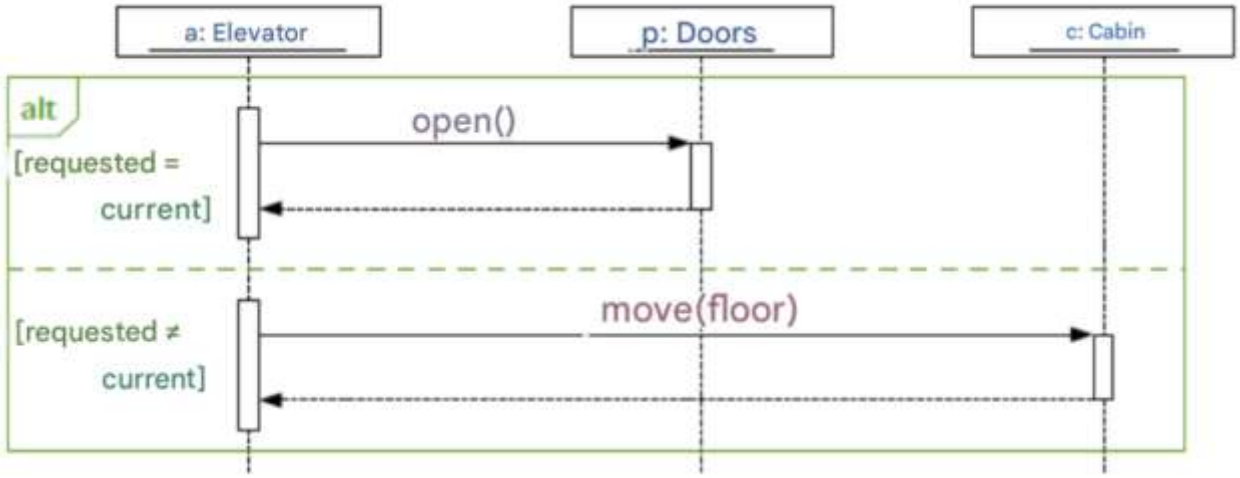


Figure 28. alt.

Loop

Principle: Repeat a chain of messages

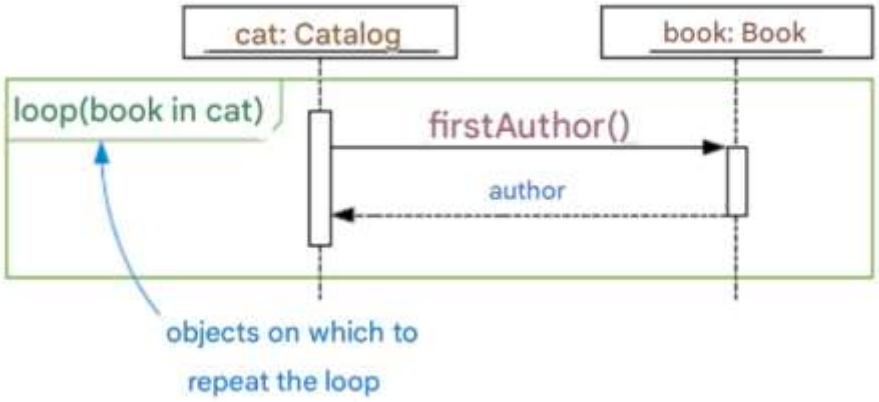


Figure 29. Loop.

Reflective message

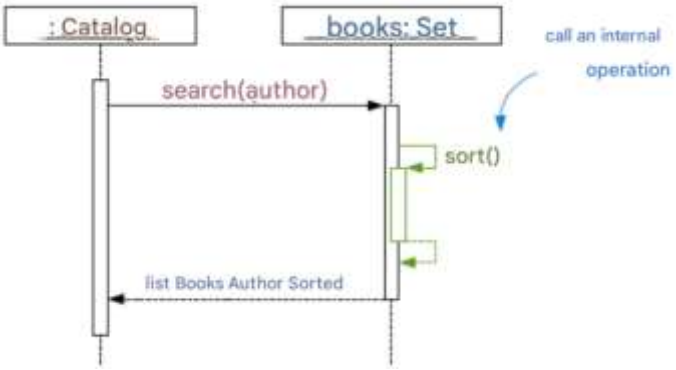


Figure 30. Reflective message.

Reference

Reference to another diagram.

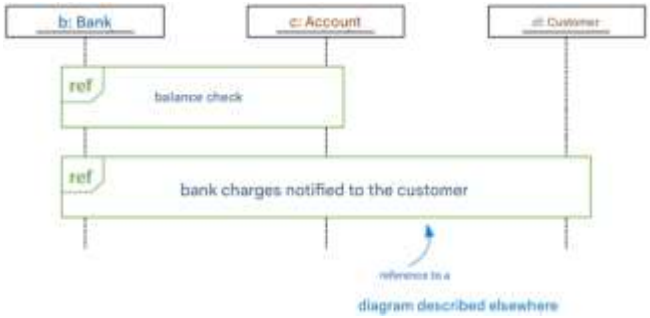


Figure 31. Reference.

Example

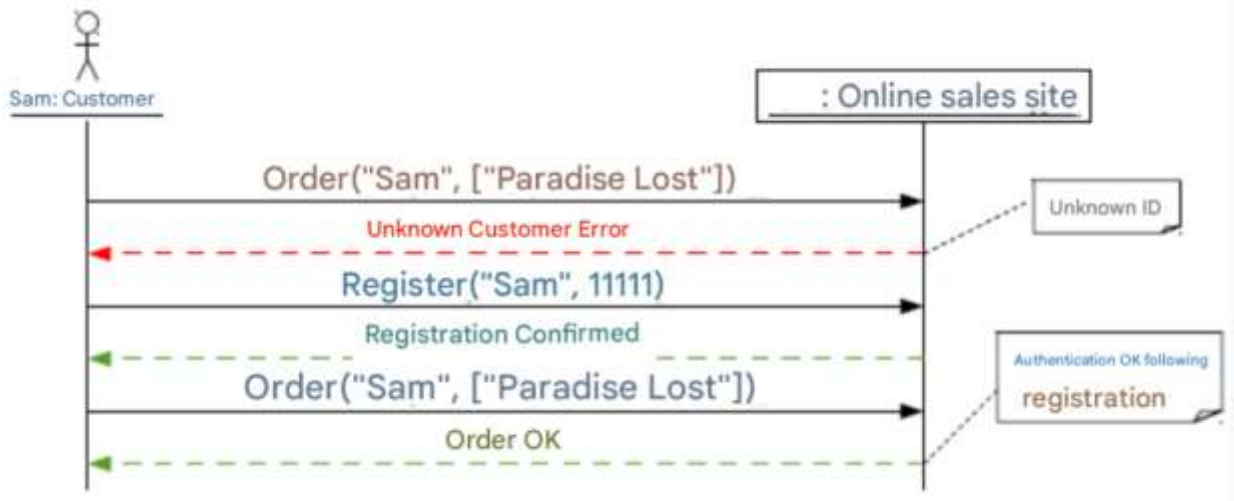


Figure 32. Sequence diagram example.

5. Activity diagram

Definition

Activity diagrams allow you to focus on processing. They are therefore particularly suitable for modeling the path of control flows and data flows. They thus allow you to graphically represent the behavior of a method or the progress of a use case.

An activity defines a behavior described by an organized sequencing of units whose simple elements are actions. The execution flow is modeled by nodes connected by arcs (transitions). The control flow remains in the activity until the processing is completed.

An activity is a behavior.

Components

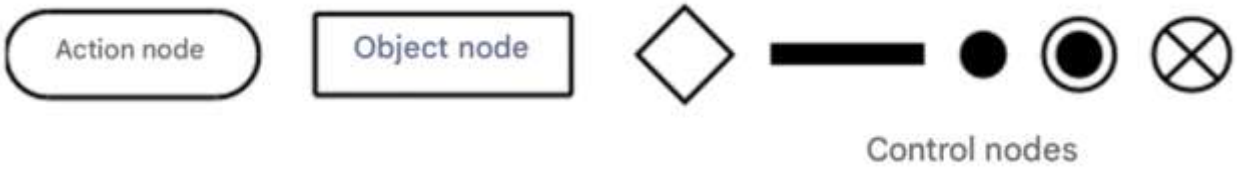


Figure 33. Components of Activity diagram.

Example

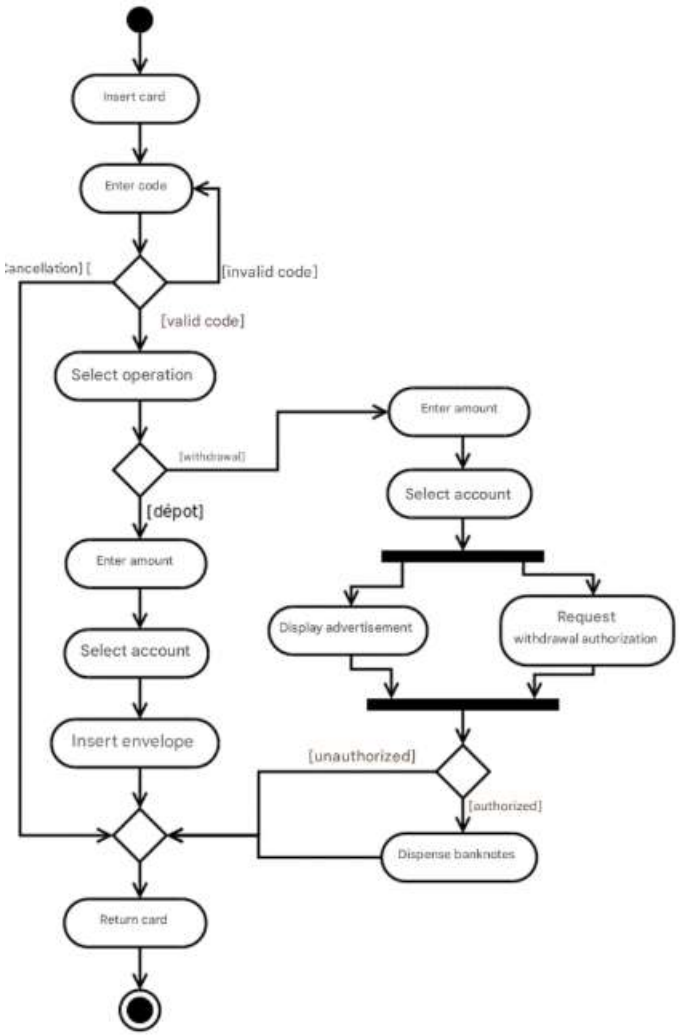


Figure 34. Activity diagram example.

6. State Transition Diagram

Definition

A state diagram is a diagram that expresses the behavior. It describes how an object works when it receives an event (or before it is received). It represents the different states in which an object can find itself and the way in which this object passes to another state in response to an event.

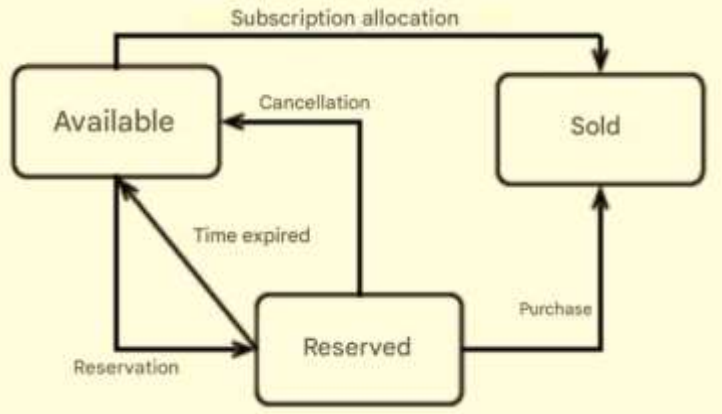


Figure 35. State transition diagram.

State

Describes a moment in the life of an instance. This instance can only be in one state at a time.

Instances of the same class react to events in the same way.

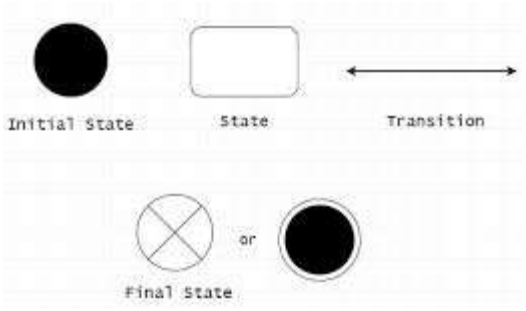


Figure 36. Components of state transition diagram.

- When it is created, the object is in an initial state.
- When the system no longer needs the object, it ends up in a final state.
- An object may not have a final state.

Transition

The response of an instance in a given state to an event.

- Event: Triggers the transition.
- Action: performing an operation when the transition is performed.

A Transition defines an Object's Response to the occurrence of an Event. It indicates that an Object that is in a State can "Transit" to another State, possibly performing certain activities.

EVENT

It causes the crossing of the transition.

- GUARD-GUARD CONDITION: [] associate a condition with a transition. For the transition to be crossed, the condition must be verified in addition to the reception of the associated event, if it exists. It is a boolean expression (on the attributes of the object or the parameters of the triggering event) which must be true for the transition to be triggered ;
- EFFECT: ACTIVITY, a set of actions to be performed when triggered,
- a primitive operation such as an instruction to assign a value to an attribute;
- sending a signal;
- calling an operation; etc.
- when the action execution is completed, the target state of the transition becomes active,

Types of events

1. Signal Event
2. Operation Call Event (Call Event) (example openFile ())
3. Change Event
4. Time Event

Temporal event

- Events are generated by the flow of time.
- They are specified either absolutely (precise date),
- Absolutely (triggered on a specific date)
- Syntax: when (date= “precise expression of a date”) ex: when (date=17/12/2010)
- Relatively (triggered after a certain duration spent in the current state).
- Syntax: after (“expression of a duration”) ex: after (10 seconds)

Chapter 6: Exercises

Exercise 1

Sarah is a sports coach at a local club. She wants to carry out a mini project to develop an application that will facilitate the calculation of **players' performance scores**. For this, she meets with another coach to discuss the idea.

Sarah: How do you track absences from training sessions?
Coach: Each player is allowed to miss a maximum of 2 sessions without justification and 4 sessions with justification; otherwise, they are suspended from the next match. It's hard for me to remember if a player provided a justification, so I keep all the medical or parental notes to assign an attendance score later.

Sarah: How do you plan to use the application?
Coach: Ideally on my phone during practice, but I could also use it on my tablet or laptop.

Sarah: How does a training session usually take place?
Coach: We start with warm-ups, then move to technical drills and teamwork exercises. I evaluate players based on their participation, discipline, and performance. Sometimes players volunteer to demonstrate drills in front of others.

Sarah: How important is the training score for a player?
Coach: The training score contributes to the overall performance score. It's calculated as follows:
Final performance = Match performance * 0.7 + Training performance * 0.3.
So if a player trains well, it improves their selection chances for upcoming matches.

Sarah: Do you really need this app?
Coach: Definitely! It will help me and other coaches because we currently calculate training scores manually or in Excel, combining attendance (out of 5), participation (out of 5), and performance (out of 10). The sheets are messy, and sometimes we can't recall which player did what. A digital app would make this process much faster, clearer, and more organized.

Sarah: How do you calculate the training score?

Coach: Training score = attendance + participation + performance

Sarah: Can players see their detailed training scores?

Coach: Yes, they usually ask me directly, but it would be better if they could check their scores through the app.

Sarah: When you write "+" beside a player's name, what does it mean?

Coach: It means a positive remark — for example, the player helped a teammate, showed leadership, or performed well in a drill.

Sarah: And the "–"?

Coach: That's when the player shows poor behavior or refuses to participate; I subtract points from their score.

#	First Name	Last Name	01/04/25	08/04/25	15/04/25	22/04/25
1	Amir	Benali	P+	P+	A	P
2	Lina	Haddad	P++	P	P	A
3	Karim	Youssef	P-	P	P+	P-
4	Naila	Mansour	A	A	A	P-

Questions

1. **Define the state of the art (Existing analysis, Critical analysis).**
   What tools or methods are currently used by coaches to track player performance?
   What are their limitations?
2. **Propose a needs analysis (requirements).**
   What does the coach need the application to do?
   What problems should it solve?
3. **What are the different functionalities of the future system?**
   List and describe the main functions (e.g., attendance tracking, score calculation, report generation, etc.).

4. **Define the users.**

   Who will use the application? (e.g., coaches, players, administrators).

Solutions

1. State of the Art (Existing Analysis and Critical Analysis)

Existing Analysis

Currently, most sports coaches in small clubs manage training attendance and performance evaluations **manually** or through **basic spreadsheets** such as Excel or Google Sheets.
Some clubs also use **general fitness tracking apps** (like Strava or TeamSnap), but these tools are not specifically designed for **training evaluation and player performance scoring** in a team context.

Typical existing solutions include:

- **Excel or paper sheets** for attendance and performance.
- **WhatsApp groups** for communication and reporting absences.
- **Manual reports** for match readiness or progress evaluation.
- **Generic sports management apps**, which are often expensive or complex for small clubs.

Critical Analysis

Aspect	Existing Solutions	Limitations
Attendance Tracking	Done manually or on paper	Risk of loss, errors, and inconsistency
Score Calculation	Manual or Excel formulas	Time-consuming, prone to miscalculation
Justification Management	Kept as paper certificates	Difficult to verify quickly
Performance Visualization	None or very limited	No summary dashboards or player comparisons
Accessibility	Not available on all devices	Limited portability and data sharing

Coaches need a **simple, mobile-friendly, and customized application** to manage attendance, justifications, and performance scores efficiently, while allowing players to view their progress in real time.

2. Needs Analysis (Requirements)

Functional Needs

1. Manage player profiles (name, position, photo, contact info).
2. Record attendance for each training session (present, absent, justified).
3. Enter justification documents (e.g., upload photo of a note).
4. Evaluate players based on:
    - Attendance (out of 5)
    - Participation (out of 5)
    - Performance (out of 10)
5. Automatically calculate the **training score**:
   Training Score = Attendance + Participation + Performance
6. Automatically calculate the **final performance score**:
   Final = (Match Performance × 0.7) + (Training Performance × 0.3)
7. Display individual and group performance summaries.
8. Allow players to access their scores via mobile or web.
9. Allow coaches to annotate positive (+) or negative (–) behavior.
10. Generate PDF or Excel reports for the club's administration.

Non-Functional Needs

- **Usability:** Simple, intuitive interface.
- **Portability:** Works on both mobile and PC.
- **Security:** Secure login for each user.
- **Scalability:** Should support multiple teams and coaches.
- **Reliability:** Data should be saved automatically.

3. Functionalities of the Future System

Functionality	Description	Users Concerned
Authentication	Login for coaches and players	Coach, Player
Player Management	Add, modify, or delete player info	Coach
Session Management	Create new training sessions, record attendance	Coach
Attendance Tracking	Mark players as present (P), absent (A), or justified (J)	Coach
Performance Evaluation	Input attendance, participation, and performance scores	Coach
Behavior Notes	Add "+" for positive actions, "–" for misconduct	Coach
Score Calculation	Automatic computation of training and final performance scores	System
Justification Upload	Attach or view justification documents	Coach
Statistics & Reports	Display averages, rankings, and performance charts	Coach, Player
Player Access Portal	Allow players to check their detailed performance	Player
Export Data	Export data to PDF or Excel	Coach

4. Definition of Users

User Type	Description	Main Actions
Coach	Main user who records attendance, enters performance data, and evaluates players.	- Manage sessions- Enter scores- View reports- Upload justifications
Player	Can view their performance details and check their attendance.	- View personal scores- See feedback (+/–)- View overall progress
Administrator (optional)	Oversees the club database and manages multiple coaches or teams.	- Manage accounts- Backup data- Generate club-wide reports

Exercise: Library Book Loan Management Application

Mohamed is a university librarian who wants to carry out a mini project to develop an application that will **simplify the management of book loans and returns**. To define his needs, he meets with another librarian, **Mr. Karim**.

Mohamed: How do you currently manage book loans?
Karim: For now, everything is done manually using a notebook or Excel sheet. I record the student's name, the book's title, and the loan date. Sometimes I forget to record when the book is returned.

Mohamed: How do you handle late returns?
Karim: We allow students to keep a book for 15 days. If the return is delayed, they get a penalty. Unfortunately, it's difficult to calculate how many days they are late without going through all the dates one by one.

Mohamed: How would you like to use the application?
Karim: Ideally on my computer, but if it's available on mobile, that would be perfect.

Mohamed: How do you organize the list of books?
Karim: I have an Excel file with book titles, authors, and the number of available copies. But when a book is borrowed, I have to update it manually. It's tiring, and sometimes I forget to do it.

Mohamed: How important is a book tracking system for the library?
Karim: It's essential. Every semester, we lose several books because we can't remember who borrowed them or when. An app could automatically update availability and send reminders for due dates.

Mohamed: Do you think students would use it?
Karim: Absolutely! If students could check online which books are available, they wouldn't need to come to the library just to ask.

Mohamed: What about fines?

Karim: Each overdue day adds a penalty of 5 DZD per day. Currently, we calculate that manually — which leads to many errors.

Mohamed: So, you would like the app to calculate fines automatically?

Karim: Yes, and it should also generate reports for each student and each book.

#	Student Name	Book Title	Loan Date	Return Date	Status
1	Ahmed Bensaid	Database Systems	01/04/25	20/04/25	Returned
2	Amir Lounes	Artificial Intelligence	03/04/25	-	Not Returned
3	Youssef Karim	Modern Programming	05/04/25	30/04/25	Returned Late
4	Nabil Benali	Data Structures	10/04/25	-	Not Returned

Questions

1. **Define the state of the art (Existing analysis, Critical analysis).**
2. **Propose a needs analysis (requirements).**
3. **What are the different functionalities of the future system?**
4. **Define the users.**

Solution

1. State of the Art (Existing Analysis and Critical Analysis)

Existing Analysis

At present, most small libraries and university reading rooms still manage their book loan systems **manually** or with **basic spreadsheet tools** such as Microsoft Excel or Google Sheets. Some larger institutions use **library management software** (e.g., Koha, PMB, or Alexandria), but these are often too complex and require constant maintenance.

Commonly used tools and practices:

- **Paper registers** to record borrowings and returns.
- **Excel files** with lists of books and students.

- **Manual calculation** of loan duration and penalties.
- **Manual updates** to the number of available copies.
- **Physical reminders** (phone calls or sticky notes) for late returns.

Critical Analysis

Aspect	Existing Methods	Limitations
Book tracking	Paper or Excel sheets	Easily lost or outdated
Loan follow-up	Manual verification	Time-consuming, error-prone
Late returns	Manual penalty calculation	Lack of consistency
Student reminders	Manual or verbal	Students often forget deadlines
Book availability	Updated manually	Information often inaccurate
Data storage	Local (paper/PC)	No synchronization or backup

The current approach is **inefficient**, **time-consuming**, and **error-prone**. A **digital application** accessible from both mobile and computer would make the process faster, more reliable, and more transparent for both librarians and students.

2. Needs Analysis (Requirements)

Functional Requirements

1. Manage a **list of books** (title, author, category, available copies, etc.).
2. Manage a **list of students** (name, ID, contact info).
3. Record each **loan operation** (student name, book title, loan date).
4. Record each **return operation** and automatically determine if it's late.
5. Calculate **penalty fees automatically** based on the number of overdue days (e.g., 5 DZD/day).
6. Automatically **update the number of available copies** after each loan or return.
7. Generate **daily or monthly reports** (borrowed books, returned books, overdue books).
8. Allow **search and filtering** by student name, book title, or status (Returned, Not Returned, Late).

9. Allow **email or SMS reminders** for students with late returns.
10. Display **statistics** (most borrowed books, most active students, total penalties).

Non-Functional Requirements

- **Usability:** Simple and intuitive interface for librarians.
- **Portability:** Accessible from both PC and smartphone.
- **Security:** Authentication required (login/password).
- **Performance:** Fast loading and quick search results.
- **Reliability:** Automatic data backup and recovery.
- **Scalability:** Can handle hundreds of books and students.

3. Functionalities of the Future System

Functionality	Description	Users Concerned
Authentication	Secure login for librarians	Librarian
Student Management	Add, modify, or delete student records	Librarian
Book Management	Add new books, update quantities, or remove entries	Librarian
Loan Management	Record when a book is borrowed	Librarian
Return Management	Record book returns and check for lateness	Librarian
Penalty Calculation	Automatically compute late fees (5 DZD/day)	System
Availability Update	Decrease/increase copies after loan/return	System
Search & Filtering	Quickly find books or students	Librarian
Reports & Statistics	Generate summaries by day, week, or month	Librarian
Reminders	Notify students of overdue books via message or email	System
Data Export	Export reports to PDF or Excel	Librarian

4. Definition of Users

User Type	Description	Main Actions
Librarian (Primary User)	Main user of the application, manages loans, returns, and penalties.	- Add students and books- Register loans and returns- Check penalties- Generate reports
Student (Secondary User)	Consults their loan history and sees due dates.	- View borrowed books- Check due dates- Receive notifications
Administrator (Optional)	Supervises multiple librarians and manages system data.	- Manage user accounts- Backup and restore data- View overall library statistics

Summary

The **Library Book Loan Management Application** proposed by Mohamed will:

- Automate **loan tracking** and **return reminders**.
- Calculate **penalties** automatically.
- Keep an up-to-date list of **book availability**.
- Allow **students to view their status** and **librarians to save time**.

This system will therefore make library management more **efficient, reliable, and transparent**.

Exercise: School Canteen Management Application

Mohamed works as an IT student developing small software projects. He wants to create an application that helps **manage meal reservations and payments in a school canteen**.
To understand the real needs, he meets with **Mr. Yassine**, the canteen manager.

Mohamed: How do you currently manage meal reservations?

Yassine: We use a paper list where students sign up daily. I then count the total meals and send it to the kitchen. It's not easy when someone cancels or changes their reservation.

Mohamed: Do you record payments?

Yassine: Yes, but manually. I use a notebook where I note who paid for the week or month. Sometimes I forget to update it, and that causes confusion.

Mohamed: How would you like to use the application?

Yassine: On my PC at work, but if there's a mobile version, that would be great for checking daily reservations.

Mohamed: What information do you keep about students?

Yassine: Their full name, class, and whether they pay daily, weekly, or monthly.

Mohamed: What problems do you face?

Yassine: Some students forget to pay, others reserve meals they don't eat, and I often have to check receipts manually. Also, it's hard to calculate how much money the canteen earns each month.

Mohamed: Would it be useful if students could reserve meals online?

Yassine: Yes, definitely. It would save time and reduce errors.

Mohamed: How do you handle absences?

Yassine: If a student doesn't come, we try to remove their reservation manually, but sometimes it's too late and the meal is wasted.

#	Student Name	Class	Payment Type	Reservation	Meal Taken	Amount Paid
1	Ahmed Bensaid	3A	Monthly	Yes	Yes	2500 DZD
2	Karim Lounes	3B	Weekly	Yes	No	600 DZD
3	Nabil Khelifi	2C	Daily	No	No	0 DZD
4	Youssef Mansour	1A	Monthly	Yes	Yes	2500 DZD

Questions

1. Define the **state of the art** (existing analysis and critical analysis).
2. Propose a **needs analysis** (requirements).
3. What are the **different functionalities** of the future system?
4. Define the **users** of the system.

Solution

1. State of the Art

Existing Analysis

Currently, most school canteens — especially in public institutions — use **manual methods** or **Excel files** to record:

- Student names and classes,
- Meal reservations,
- Payments (daily, weekly, monthly),
- Meal consumption.

Some schools use **basic digital spreadsheets**, but they are not connected to payment tracking or real-time reservations.

Critical Analysis

Aspect	Current Method	Limitations
Reservation	Paper or Excel	No real-time updates; duplication errors
Payment Tracking	Manual receipts	Time-consuming and prone to mistakes
Absence Handling	Manual cross-checking	Hard to manage and track refunds
Statistics	Manual calculation	Difficult to know total income or number of meals
Accessibility	Paper-based	No student self-service or mobile access

The existing process is **manual**, **slow**, and **error-prone**.
A dedicated digital system would make meal reservations, payments, and reporting **faster, clearer, and more accurate**.

2. Needs Analysis (Requirements)

Functional Requirements

1. Manage a **database of students** (name, class, payment type).
2. Record **meal reservations** for each day or week.
3. Track **meal consumption** (if a meal was taken or not).
4. Manage **payments** (daily, weekly, monthly) and calculate totals.
5. Generate **reports** (daily reservations, monthly income, unpaid students).
6. Allow **students to reserve or cancel** their meals online.
7. Automatically **update available seats or meal counts** for the kitchen.
8. Manage **refunds** for absences or cancellations.
9. Display **statistics** (number of meals served, total payments).
10. Export data to **PDF or Excel** for administrative reporting.

Non-Functional Requirements

- **Usability:** Intuitive interface for both staff and students.
- **Portability:** Web or mobile access.
- **Security:** Login required for both students and managers.
- **Performance:** Real-time update of reservations.
- **Reliability:** Automatic backup of reservation data.

3. Functionalities of the Future System

Functionality	Description	Users Concerned
Authentication	Secure login for manager and students	Manager, Student
Student Management	Add, edit, or delete student profiles	Manager
Meal Reservation	Students can reserve meals in advance	Student
Meal Validation	Manager marks which students took meals	Manager
Payment Management	Record or verify payments	Manager
Automatic Calculation	System calculates daily and monthly income	System
Absence Tracking	Manage missed reservations and possible refunds	Manager
Statistics and Reports	View totals, unpaid lists, and charts	Manager
Notifications	Send reminders for payment or reservation	System
Export Data	Generate reports in Excel or PDF	Manager

4. Definition of Users

User Type	Description	Main Actions
Canteen Manager (Primary User)	Manages reservations, payments, and reports	- Register students- Validate meals- View statistics- Export data
Student (Secondary User)	Makes or cancels meal reservations	- Reserve or cancel meals- Check payment status- Receive notifications
Administrator (Optional)	Manages overall system configuration	- Manage accounts- Backup data- Monitor system usage

Summary

The proposed **Canteen Management Application** will:

- Simplify **meal reservations** and **payments**.

- Prevent **waste** by accurately tracking attendance.
- Provide **real-time statistics** for management.
- Allow **students** to reserve meals independently and transparently.

This solution increases **efficiency**, reduces **errors**, and modernizes the school's canteen service.

Exercise 1: Hospital Appointment Management System

Context

Mohamed, a software engineering student, wants to create an application to **help hospitals manage appointments between patients and doctors** more efficiently.
He interviews **Dr. Youssef**, the head of the internal medicine department, to better understand the hospital's current problems.

Mohamed: How do patients currently make appointments?
Dr. Youssef: Mostly by phone or directly at the reception. The receptionist writes their names and appointment times in a notebook or Excel file.

Mohamed: What problems do you face with this method?
Dr. Youssef: Many! Sometimes two patients are given the same time slot by mistake, or appointments are forgotten. Patients also call frequently to change or cancel appointments, and the receptionist can't keep up.

Mohamed: Would you prefer an app that works on a computer or mobile phone?
Dr. Youssef: Both. Receptionists would use it on a PC, and patients should be able to book through their phones.

Mohamed: How are medical records handled?
Dr. Youssef: They are stored separately in files. It would be helpful if the app could at least show the last visit date and doctor seen.

Mohamed: What happens when a patient misses an appointment?

Dr. Youssef: We just mark them as "Absent," but that's not useful since there's no automatic tracking.

#	Patient Name	Doctor	Department	Appointment Date	Status
1	Ahmed Bensaid	Dr. Youssef	Cardiology	04/04/25	Attended
2	Karim Lounes	Dr. Farid	Pediatrics	05/04/25	Cancelled
3	Nabil Amine	Dr. Youssef	Cardiology	06/04/25	Absent
4	Youssef Mansour	Dr. Rami	Dentistry	06/04/25	Attended

Questions

1. Define the state of the art (existing analysis, critical analysis).
2. Propose a needs analysis (requirements).
3. What are the functionalities of the future system?
4. Define the users.

Solution

1. State of the Art

Most hospitals still use **manual methods or Excel sheets** to manage appointments. Some use basic systems that don't synchronize between departments.

- **Problems:** double bookings, missing data, delays, and no notifications.
- **Need:** a centralized, digital system accessible by both staff and patients.

2. Needs Analysis

Functional:

- Manage patient and doctor records.
- Book, modify, or cancel appointments.
- Notify patients by SMS/email.

- Show daily schedule per doctor.
- Track patient visit history.
- Generate reports (missed appointments, doctor activity).

Non-functional:

- Secure access.
- Multi-user support.
- Mobile and web compatibility.
- Data backup and reliability.

3. Functionalities

Functionality	Description	Users
Appointment Management	Book, edit, cancel appointments	Receptionist, Patient
Notifications	Send SMS/email reminders	System
Doctor Schedule	View daily or weekly calendar	Doctor
Patient Records	Store visit and doctor info	System
Reports	Statistics of visits, absences	Admin
Search & Filter	Find appointments quickly	All

4. Users

- **Patient:** Book, view, or cancel appointments.
- **Doctor:** View their own appointments.
- **Receptionist:** Manage all appointments.
- **Administrator:** Supervise users and data.

Exercise 2: Gym Membership and Attendance Tracking System

Context

Mohamed is designing an application for a **local gym** that needs to track memberships, payments, and attendance.
He meets **Mr. Rachid**, the gym owner, to understand daily challenges.

Mohamed: How do you manage member registrations now?
Rachid: Everything is written in a notebook. We record names, subscription type, and payment date.

Mohamed: What are the main issues?
Rachid: We often lose track of who has paid or whose subscription expired. Also, we can't easily see who attends regularly.

Mohamed: Would you like members to check their status themselves?
Rachid: Yes, that would save us a lot of time.

Mohamed: What types of subscriptions do you offer?
Rachid: Daily, weekly, monthly, and yearly. Prices vary depending on the plan.

Mohamed: Do you track attendance daily?
Rachid: Yes, we mark present members on paper, but sometimes we forget.

#	Member Name	Subscription	Start Date	Expiry Date	Attendance (Days)	Status
1	Ahmed Bensaid	Monthly	01/04/25	30/04/25	15	Active
2	Karim Lounes	Weekly	08/04/25	15/04/25	3	Expired
3	Nabil Khelifi	Yearly	01/01/25	31/12/25	45	Active
4	Youssef Mansour	Monthly	10/04/25	09/05/25	10	Active

Solution

1. State of the Art

Gyms often use paper or Excel sheets for membership tracking. Some use complex software not suited for small gyms.

- **Problems:** missed renewals, lost data, poor attendance tracking.
- **Need:** a simple system for managing subscriptions and attendance efficiently.

2. Needs Analysis

Functional:

- Manage member profiles.
- Record subscriptions, renewals, and payments.
- Track attendance automatically (barcode or manual).
- Notify members of expiry.
- Generate reports (income, attendance).

Non-functional:

- User-friendly interface.
- Secure access.
- Works offline or online.
- Fast reporting.

3. Functionalities

Functionality	Description	Users
Membership Management	Add/edit/delete member info	Manager
Subscription Tracking	Record start and end dates	Manager
Attendance Tracking	Mark attendance manually or with card	Manager
Payment Management	Track fees and income	Manager
Notification	Send renewal reminders	System
Reports	Generate summary by month or member	Manager
Search	Quick lookup by name or ID	All

4. Users

- **Manager:** Main user who manages all operations.
- **Member:** Can view their status and renewal date.
- **Administrator (optional):** Backs up data, manages accounts.

Exercise 3: Taxi Booking and Dispatch Application

Context

Mohamed wants to design an application for a **local taxi company** that currently handles all bookings manually by phone.
He interviews **Mr. Samir**, the dispatcher in charge of coordinating drivers and clients.

Mohamed: How do customers currently book taxis?
Samir: By phone. I write down the name, location, and time in a notebook and then call a driver.

Mohamed: What are the difficulties you face?
Samir: Sometimes drivers forget pickups or arrive late. It's also hard to track which driver is free or how many trips were done per day.

Mohamed: Would you like to track drivers in real time?
Samir: Yes, that would be great. Even just marking who's available and where would help.

Mohamed: How do customers pay?
Samir: Mostly in cash. But we're thinking of allowing online payments or fare estimation.

Mohamed: How do you calculate fares?
Samir: It depends on distance and time, but we often estimate manually.

#	Client Name	Pickup Location	Destination	Driver	Status	Fare (DZD)
1	Ahmed Bensaid	Downtown	Airport	Driver A	Completed	1800
2	Karim Lounes	Station	Hospital	Driver B	Cancelled	0
3	Nabil Khelifi	Mall	University	Driver C	In Progress	-
4	Youssef Mansour	Airport	Hotel	Driver A	Completed	900

Solution

1. State of the Art

Taxi companies often rely on **manual phone-based bookings** or use third-party apps like **Uber or Bolt** that charge high commissions.
Small companies lack real-time tracking or integrated fare calculation.

Problems: double bookings, lost records, driver confusion, no analytics.
Need: a lightweight system for managing drivers, bookings, and payments.

2. Needs Analysis

Functional:

- Register clients and drivers.
- Create, modify, or cancel bookings.
- Assign available drivers automatically.
- Calculate fares by distance/time.
- Display real-time trip status.
- Generate reports (completed, canceled trips).
- Allow customers to rate service.

Non-functional:

- Real-time data sync.
- Secure access.
- Works on mobile.
- Fast, reliable, and simple interface.

3. Functionalities

Functionality	Description	Users
Booking Management	Create, modify, or cancel rides	Dispatcher, Client
Driver Assignment	Automatically assign available driver	System

Fare Calculation	Compute fare based on route	System
Trip Tracking	Update ride status (In Progress, Completed)	Driver
Notifications	Inform clients when taxi is near	System
Reports	Total rides, revenue, cancellations	Manager
Payment Tracking	Manage cash and online payments	Manager
Ratings	Allow client feedback	Client

4. Users

- **Client:** Books and tracks taxi.
- **Driver:** Accepts or declines rides, updates status.
- **Dispatcher:** Oversees all rides and driver activity.
- **Administrator:** Manages users and system configuration.

Exercise

Imagine you are tasked with developing a Library Management System (LMS). This system should help librarians manage the catalog of books, track borrowed books, and handle user registrations. Users should be able to search for books, borrow and return them, and receive notifications about due dates.

1. For this exercise, list and classify the following requirements into two categories: Functional (F), and Non-Functional (NF).

Requirements:

1. The system must support at least 100 concurrent users.
2. Users should be able to search for books by title, author, or ISBN.
3. The system should provide a user-friendly interface.
4. Users must be able to register with their email addresses.
5. The system must ensure that all personal data is encrypted.
6. The system should allow librarians to add, update, or remove books from the catalog.

7. The system should send an email notification to users when a borrowed book is due in 3 days.

8. The system must be available 99.9% of the time.

9. Users should be able to view their borrowing history.

10. The system should process search queries within 2 seconds.

2. Draft three more requirements for the LMS and classify them as functional or non-functional.

Exercise

You are working on a project to develop an online food delivery application. The application will allow users to browse restaurants, place orders, and track delivery status. Restaurants can manage their menu and view order history. The application should provide a seamless and secure user experience.

1. Identify and classify each of the below requirements into functional (F) and non-functional (NF) categories.

Project Description:

1. Users should be able to create an account using their email or social media accounts.

2. The application must support at least 1,000 concurrent users without performance degradation.

3. Users should be able to browse restaurants by cuisine, rating, and location.

4. The application should load the homepage within 3 seconds.

5. Restaurants should be able to add, update, or remove items from their menu.

6. Users should receive real-time notifications about their order status.

7. The application should use SSL encryption for all data transmissions.

8. The system should provide 24/7 customer support through live chat.

9. Users should be able to pay using various payment methods, including credit cards, debit cards, and digital wallets.

10. The application should be available on both iOS and Android platforms.

11. The system should log all transactions for auditing purposes.

12. The application should provide personalized recommendations based on user preferences and past orders.

2. Read the following requirements and classify them as functional or non-functional.

1. The system should automatically apply discounts and promotions to eligible orders.

2. The application must recover from system failures within 5 minutes.

3. Users should be able to leave reviews and ratings for restaurants.

4. The application should be able to handle up to 500,000 orders per day.

5. The user interface should be intuitive and easy to navigate.

Exercise

You are assigned to develop a mobile banking application. This application will enable users to perform various banking operations such as checking their account balance, transferring funds, paying bills, and locating nearby ATMs. The application needs to be secure, user-friendly, and perform efficiently under different conditions.

Project Description:

1. The application must allow users to log in using biometric authentication (fingerprint or facial recognition).
2. Users should be able to view their account balance in real-time.
3. The application should complete fund transfers within 5 seconds.
4. Users must be able to add and manage multiple bank accounts within the application.
5. The application should provide transaction history for the past 6 months.
6. All sensitive data must be encrypted both in transit and at rest.
7. Users should receive push notifications for all transactions.

8. The application should support multiple languages to cater to a diverse user base.
9. The system should handle at least 10,000 concurrent users without performance degradation.
10. Users should be able to report lost or stolen cards through the application.
11. The user interface should be intuitive and easy to navigate, adhering to modern design principles.
12. The application should have a built-in chat support feature for customer assistance.
13. The system must comply with all relevant financial regulations and standards.
14. The application should provide customizable spending alerts for users.
15. The application must have an uptime of 99.9%.

Question 01: Identify and classify each of the above requirements into functional (F) and non-functional (NF) categories.

Question 02: Review the following requirements and classify them as functional or non-functional.

1. The application should allow users to schedule future payments for bills.
2. The application must load the dashboard within 3 seconds.
3. Users should be able to set and track their savings goals.
4. The application should provide support for visually impaired users.
5. The system should back up all user data daily.

Exercise

You are tasked with developing an e-commerce platform that will allow users to buy and sell products online. The platform should cater to both individual buyers and sellers, providing a seamless and secure shopping experience. Sellers can list their products, manage inventory, and track sales. Buyers can search for products, add them to their cart, and make purchases. The platform should also handle payments and provide order tracking.

Instructions:

1. Read the project description carefully.

2. Define at least 10 functional requirements for the e-commerce platform based on the scenario provided.
3. Ensure each requirement is clear, specific, and describes an action that the system should perform.

Project Description:

- The platform must allow users to create and manage their accounts.
- Sellers should be able to list their products with details such as name, description, price, and images.
- Buyers should be able to search for products by category, name, or price range.
- The platform should support a shopping cart feature where buyers can add, remove, or modify products before purchasing.
- Payments should be processed securely through various payment methods like credit cards, debit cards, and digital wallets.
- The platform should provide order confirmation and tracking for buyers.
- Sellers should have access to sales analytics and inventory management tools.
- Users should be able to leave reviews and ratings for products they have purchased.
- The platform should support promotional codes and discounts.
- Customer support should be available through live chat.

Exercise

You are tasked with developing a Hospital Management System (HMS) that will streamline the operations of a hospital. The system should facilitate patient registration, appointment scheduling, medical record management, billing, and reporting. Healthcare providers should be able to access patient records, update treatment information, and manage their schedules. Administrators should have tools for generating reports and managing hospital resources.

Instructions:

1. Read the project description carefully.
2. Define at least 10 functional requirements for the Hospital Management System based on the scenario provided.

3. Ensure each requirement is clear, specific, and describes an action that the system should perform.

Project Description:

- The system must allow patients to register online and in-person.
- Patients should be able to schedule, reschedule, or cancel appointments.
- Healthcare providers should be able to access and update patient medical records.
- The system should generate bills for patients based on treatments and services received.
- Administrators should be able to manage hospital staff schedules and resources.
- The system must provide reminders and notifications for upcoming appointments to patients and providers.
- Healthcare providers should be able to prescribe medications and order lab tests electronically.
- The system should allow patients to view their medical history and lab results online.
- Administrators should be able to generate various reports such as patient admission rates, financial reports, and resource utilization.
- The system should support multiple user roles with different access permissions (e.g., patient, doctor, nurse, administrator).

Exercise

In a school, there is a need to manage the reservation of classrooms as well as educational equipment (laptop and/or video projector). Only teachers are authorized to make reservations (subject to the availability of the room or equipment). The schedule of classrooms can be viewed by everyone. However, the hourly summary by teacher (calculated from the classroom schedule) can only be viewed by teachers. Finally, for each training program, there is a responsible teacher who is the only one who can edit the hourly summary for the entire training program.

Exercise 01.a

The university library requires a management system to facilitate both students and librarians. Students should be able to register, search for books, borrow them, return them, and check their

borrowing history. Librarians, on the other hand, are responsible for managing the library's collection, including adding new books, removing old ones, and maintaining borrowing records. The system must clearly distinguish between student and librarian roles.

Exercise 01.b

An online shopping platform is needed to provide customers with the ability to browse products, add items to their cart, place orders, make payments, and track their orders. The system should also include an administrator role that manages the products by adding, removing, or updating product details, as well as generating sales reports. Payments are handled through a third-party payment gateway, which verifies each transaction before the order is confirmed.

Exercise 01.c

An ATM system is designed to allow users to interact with their bank accounts through automated machines. The user inserts their card and enters a PIN, which must be verified by the bank system. Once authenticated, users can withdraw cash, deposit money, and check their account balance. The bank system updates account balances accordingly. If a user fails to provide the correct PIN three consecutive times, the bank system blocks the card to prevent fraud.

Question: provide the corresponding use case diagram.

Exercise 02

The customer inserts their card, which is immediately verified for validity. They are then prompted to enter the card's PIN. After three unsuccessful attempts, the card is retained. Otherwise, the customer can indicate the amount they wish to withdraw, and the balance of their bank account is checked to ensure the withdrawal is possible. If the balance is insufficient, the customer is informed and can then enter a lower amount. If the account balance is sufficient, the ATM returns the card and dispenses the cash along with a receipt.

Question: Describe the operation of this ATM using an activity diagram.

Exercise

The nominal sequence section of the cash withdrawal use case contains the following elements:

1. The teller enters the customer's account number;
2. The application validates the account with the central system;
3. The teller requests a withdrawal of 1000 DA;
4. The teller system queries the central system to ensure that the account is sufficiently funded;
5. The central system debits the account;
6. In return, the system notifies the teller that they can issue the requested amount.

Question: Provide the sequence diagram associated with this textual description.

Exercise

An academy wishes to manage the courses offered in several colleges. For this, the following information is available:

- Each college has a website and is structured into departments, each of which groups specific teachers. Among these teachers, one is the head of the department.

- A teacher is defined by their name, first name, phone number, email, date of appointment, and their index. Each teacher teaches only one subject.

- Students, on the other hand, follow several subjects and receive a grade for each of them. For each student, we want to manage their name, first name, phone number, email, and their year of entry into the college.

- A subject can be taught by several teachers but always takes place in the same classroom (each having a determined number of seats).

- We want to be able to calculate the average grade per subject as well as per department.

- We also want to calculate the overall average grade of a student and display the subjects in which they have not been graded.

- Finally, we must be able to print the profile (first name, phone number, email) of a teacher or a student.

Question: Provide the corresponding class diagram.

Laboratory Sessions

In laboratory sessions, the work will be done using java language and IDE Eclipse or Netbeans. For the Database, we will use MySQL.

Exercise : University Library Management System

Context

Mohamed, a computer science student, wants to develop a **Library Management Application** for his university as a mini-project.
He schedules a meeting with **Mr. Rachid**, the librarian, to understand the **current process** and **pain points**.

Mohamed: How do you currently manage books and students in the library?
Rachid: Everything is recorded in registers. We have separate notebooks for student details, issued books, and returns. Sometimes we use Excel, but it's not very reliable.

Mohamed: How do students borrow a book?
Rachid: They come to the counter, we check manually if the book is available, and then note down their name, the book title, and the due date in a register.

Mohamed: Do you face any difficulties with this method?
Rachid: Yes, many! For example, it's hard to track who has overdue books. Sometimes books are lost and we can't find out who borrowed them. Also, finding a specific book or knowing how many copies are available takes time.

Mohamed: How would you like the system to work?
Rachid: Ideally, I want to enter all book details into a database — title, author, category, copies, and availability. When a student borrows a book, I should just select their name and the book, and the system updates automatically.

Mohamed: Should students be able to use the application directly?
Rachid: Yes. It would be great if they could **search books online**, see their **borrowing history**, and **reserve** books.

Mohamed: How do you handle late returns or fines?
Rachid: We charge 50 DZD per day after the due date. But now, we calculate manually — it's very time-consuming.

Mohamed: Do you have digital versions (PDFs, e-books)?
Rachid: A few, yes. I'd like them to be available for online reading if possible.

Mohamed: Who else uses the system besides you?
Rachid: I have two assistants who help manage the books, and the university administration sometimes requests statistics about book usage.

Example of Current Manual Record

#	Student Name	Book Title	Author	Date Borrowed	Due Date	Date Returned	Fine (DZD)
1	Ahmed Bensaid	Data Structures	Mark Weiss	01/04/25	10/04/25	11/04/25	50
2	Karim Lounes	Clean Code	Robert Martin	03/04/25	13/04/25	13/04/25	0
3	Nabil Amine	AI Fundamentals	Stuart Russell	05/04/25	15/04/25	—	—
4	Youssef Mansour	Java Programming	Herbert Schildt	06/04/25	16/04/25	—	—

Questions

1. Define the **State of the Art** (existing analysis and critical analysis).
2. Propose a **Needs Analysis** (functional and non-functional requirements).
3. List the **Functionalities** of the future system.
4. Define the **Users** of the system.

Solution

1. State of the Art

Most university libraries still use **manual registers or Excel files** for book management.
This method is prone to **errors**, **loss of data**, and **inefficiency**.

Existing systems include basic software like Koha or simple desktop programs, but they are often **too complex** or **not localized** for small institutions.

Problems Identified:

- Difficult to find or track books.
- No automatic fine calculation.

- Overdue books not easily detected.
- No central database for students and books.
- Time wasted on manual data entry and searching.

Need:
A simple, digital, and user-friendly library management system for both librarians and students.

2. Needs Analysis

Functional Requirements

- Manage book records: add, edit, delete, and search by title, author, or category.
- Manage student records: registration, borrowing history, fines.
- Record book borrowings and returns.
- Automatically calculate fines for late returns.
- Allow students to search and reserve books online.
- Generate reports (borrowed books, overdue books, active users).
- Provide notifications (email/SMS) for due dates or reserved book availability.
- Handle both physical and digital (PDF/e-book) materials.

Non-Functional Requirements

- Multi-user access (librarian, assistant, student).
- Secure login and authentication.
- Reliable data backup.
- Fast response time for search queries.
- Compatibility with both web and mobile platforms.
- Intuitive and user-friendly interface.

3. Functionalities of the Future System

Functionality	Description	Users
Book Management	Add, edit, delete, and search books	Librarian
Student Management	Register students, view records	Librarian
Borrow/Return Books	Manage loan transactions	Librarian
Fine Calculation	Automatic fine for late returns	System
Reservation	Reserve book copies online	Student
Search Books	Search by title, author, or category	Student
Notifications	Remind students about due dates	System
Report Generation	View statistics and summaries	Admin/Librarian
Digital Access	Read or download e-books	Student

4. Users of the System

User	Responsibilities
Librarian	Main user; manages all books, borrowings, and returns.
Library Assistant	Supports librarian in data entry and book management.
Student	Searches, reserves, and borrows books.
Administrator	Manages users, system settings, and data backup.

Java Classes

1. Book.java

```
package library;

import java.util.Objects;

public class Book {

    private int bookId;
    private String title;
    private String author;
    private String category;
    private int copiesAvailable;
    private String digitalLink; // Optional: for e-book

    public Book(int bookId, String title, String author, String category, int
copiesAvailable, String digitalLink) {
        this.bookId = bookId;
        this.title = title;
        this.author = author;
        this.category = category;
        this.copiesAvailable = copiesAvailable;
        this.digitalLink = digitalLink;
    }

    // Getters and Setters
    public int getBookId() { return bookId; }
    public String getTitle() { return title; }
    public String getAuthor() { return author; }
    public String getCategory() { return category; }
    public int getCopiesAvailable() { return copiesAvailable; }
    public String getDigitalLink() { return digitalLink; }

    public void setCopiesAvailable(int copiesAvailable) {
this.copiesAvailable = copiesAvailable; }

    // Business methods
    public void updateAvailability(int delta) {
        this.copiesAvailable += delta;
    }

    public boolean isAvailable() {
```

```
        return copiesAvailable > 0;
    }

    @Override
    public String toString() {
        return "Book: " + title + " by " + author + " (" + category + "),
copies: " + copiesAvailable;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Book)) return false;
        Book book = (Book) o;
        return bookId == book.bookId;
    }

    @Override
    public int hashCode() {
        return Objects.hash(bookId);
    }
}
```

2. Student.java

```
package library;

import java.util.ArrayList;
import java.util.List;

public class Student {

    private int studentId;
    private String name;
    private String department;
    private String email;
    private String phone;
    private List<Borrowing> borrowingHistory;

    public Student(int studentId, String name, String department, String
email, String phone) {
        this.studentId = studentId;
        this.name = name;
        this.department = department;
        this.email = email;
        this.phone = phone;
        this.borrowingHistory = new ArrayList<>();
    }

    // Getters and Setters
    public int getStudentId() { return studentId; }
    public String getName() { return name; }
    public String getDepartment() { return department; }
    public String getEmail() { return email; }
    public String getPhone() { return phone; }
    public List<Borrowing> getBorrowingHistory() { return borrowingHistory; }
```

```
    // Business methods
    public void reserveBook(Book book) {
        if (book.isAvailable()) {
            System.out.println(name + " reserved the book: " +
book.getTitle());
        } else {
            System.out.println("Book " + book.getTitle() + " is not available
for reservation.");
        }
    }

    public void addBorrowingRecord(Borrowing borrowing) {
        borrowingHistory.add(borrowing);
    }

    public void viewHistory() {
        System.out.println("Borrowing history for " + name + ":");
        for (Borrowing b : borrowingHistory) {
            System.out.println(b);
        }
    }

    @Override
    public String toString() {
        return name + " (" + department + ")";
    }
}
```

3. Borrowing.java

```
package library;

import java.time.LocalDate;
import java.time.temporal.ChronoUnit;

public class Borrowing {

    private int borrowId;
    private Student student;
    private Book book;
    private LocalDate borrowDate;
    private LocalDate dueDate;
    private LocalDate returnDate;
    private double fineAmount;

    private static final double DAILY_FINE = 50.0; // DZD per day

    public Borrowing(int borrowId, Student student, Book book, LocalDate
borrowDate, LocalDate dueDate) {
        this.borrowId = borrowId;
        this.student = student;
        this.book = book;
        this.borrowDate = borrowDate;
        this.dueDate = dueDate;
        this.returnDate = null;
        this.fineAmount = 0.0;
    }
```

```
    public void markReturned(LocalDate returnDate) {
        this.returnDate = returnDate;
        calculateFine();
        book.updateAvailability(+1);
    }

    public void calculateFine() {
        if (returnDate != null && returnDate.isAfter(dueDate)) {
            long daysLate = ChronoUnit.DAYS.between(dueDate, returnDate);
            fineAmount = daysLate * DAILY_FINE;
        } else {
            fineAmount = 0.0;
        }
    }

    public double getFineAmount() { return fineAmount; }

    @Override
    public String toString() {
        return "Borrowing [Book=" + book.getTitle() +
                ", Borrow Date=" + borrowDate +
                ", Due Date=" + dueDate +
                ", Returned=" + (returnDate != null ? returnDate : "Not yet")
+
                ", Fine=" + fineAmount + " DZD]";
    }
}
```

4. Librarian.java

```
package library;

import java.util.ArrayList;
import java.util.List;

public class Librarian {

    private int librarianId;
    private String name;
    private String username;
    private String password;
    private List<Book> books;
    private List<Student> students;
    private List<Borrowing> borrowings;

    public Librarian(int librarianId, String name, String username, String
password) {
        this.librarianId = librarianId;
        this.name = name;
        this.username = username;
        this.password = password;
        this.books = new ArrayList<>();
        this.students = new ArrayList<>();
        this.borrowings = new ArrayList<>();
    }
```

```
    // Book management
    public void addBook(Book book) {
        books.add(book);
        System.out.println("Book added: " + book.getTitle());
    }

    public void removeBook(Book book) {
        books.remove(book);
        System.out.println("Book removed: " + book.getTitle());
    }

    public void listBooks() {
        System.out.println("Books available in library:");
        for (Book b : books) {
            System.out.println(b);
        }
    }

    // Student management
    public void registerStudent(Student student) {
        students.add(student);
        System.out.println("Student registered: " + student.getName());
    }

    public void recordBorrowing(Borrowing borrowing) {
        borrowings.add(borrowing);
        borrowing.getBook().updateAvailability(-1);
        borrowing.getStudent().addBorrowingRecord(borrowing);
        System.out.println("Borrowing recorded for: " +
borrowing.getStudent().getName());
    }

    public void generateReport() {
        System.out.println("Library Report:");
        System.out.println("Total Books: " + books.size());
        System.out.println("Total Students: " + students.size());
        System.out.println("Total Borrowings: " + borrowings.size());
    }
}
```

5. Administrator.java

```
package library;

public class Administrator {

    private int adminId;
    private String name;
    private String username;
    private String password;

    public Administrator(int adminId, String name, String username, String
password) {
        this.adminId = adminId;
        this.name = name;
        this.username = username;
        this.password = password;
```

```
    }

    public void manageUsers() {
        System.out.println("Administrator " + name + " is managing user
accounts...");
    }

    public void backupData() {
        System.out.println("System data backed up successfully by " + name);
    }

    @Override
    public String toString() {
        return "Administrator: " + name;
    }
}
```

6. TestLibraryApp.java (Demo Runner)

```
package library;

import java.time.LocalDate;

public class TestLibraryApp {

    public static void main(String[] args) {

        Librarian librarian = new Librarian(1, "Mr. Rachid", "rachid",
"1234");
        Administrator admin = new Administrator(1, "Admin Mohamed", "admin",
"admin123");

        Book b1 = new Book(101, "Data Structures", "Mark Weiss", "Computer
Science", 3, null);
        Book b2 = new Book(102, "Clean Code", "Robert Martin", "Software
Engineering", 2, null);

        librarian.addBook(b1);
        librarian.addBook(b2);

        Student s1 = new Student(1, "Ahmed Bensaid", "CS", "ahmed@mail.com",
"0555123456");
        librarian.registerStudent(s1);

        Borrowing bor1 = new Borrowing(1, s1, b1, LocalDate.of(2025, 4, 1),
LocalDate.of(2025, 4, 10));
        librarian.recordBorrowing(bor1);

        bor1.markReturned(LocalDate.of(2025, 4, 12)); // Returned late
        System.out.println(bor1);

        librarian.generateReport();
        admin.backupData();
    }
}
```

Database

1. Create the Database

```
CREATE DATABASE library_management CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci;
USE library_management;
```

2. Table: books

Each book has a unique ID, title, author, category, and number of available copies.

```
CREATE TABLE books (
    book_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(150) NOT NULL,
    author VARCHAR(100) NOT NULL,
    category VARCHAR(50),
    copies_available INT DEFAULT 0,
    digital_link VARCHAR(255)
);
```

3. Table: students

Each student has identifying and contact information.

```
CREATE TABLE students (
    student_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    department VARCHAR(100),
    email VARCHAR(100) UNIQUE,
    phone VARCHAR(20)
);
```

4. Table: librarians

Each librarian manages books and borrowing transactions.

```
CREATE TABLE librarians (
    librarian_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(100) NOT NULL
);
```

5. Table: administrators

System administrators can back up data or manage accounts.

```
CREATE TABLE administrators (
    admin_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
```

```
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(100) NOT NULL
);
```

6. Table: borrowings

Represents the relationship between a student and a borrowed book.
Also stores borrow and return dates, and fine amount.

```
CREATE TABLE borrowings (
    borrow_id INT AUTO_INCREMENT PRIMARY KEY,
    student_id INT NOT NULL,
    book_id INT NOT NULL,
    librarian_id INT,
    borrow_date DATE NOT NULL,
    due_date DATE NOT NULL,
    return_date DATE DEFAULT NULL,
    fine_amount DECIMAL(10,2) DEFAULT 0.00,

    CONSTRAINT fk_borrow_student FOREIGN KEY (student_id)
        REFERENCES students(student_id) ON DELETE CASCADE,

    CONSTRAINT fk_borrow_book FOREIGN KEY (book_id)
        REFERENCES books(book_id) ON DELETE CASCADE,

    CONSTRAINT fk_borrow_librarian FOREIGN KEY (librarian_id)
        REFERENCES librarians(librarian_id) ON DELETE SET NULL
);
```

7. Optional: Table for Login Logs or System Backups

To track administrator activities:

```
CREATE TABLE system_logs (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    admin_id INT,
    action VARCHAR(255),
    log_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (admin_id) REFERENCES administrators(admin_id)
);
```

DAO Classes

1. Database Connection (DBConnection.java)

```
package library.dao;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBConnection {
```

```
    private static final String URL =
"jdbc:mysql://localhost:3306/library_management";
    private static final String USER = "root"; // change to your MySQL
username
    private static final String PASSWORD = ""; // change to your MySQL
password
    private static Connection connection = null;

    // Get a single connection instance (Singleton pattern)
    public static Connection getConnection() {
        if (connection == null) {
            try {
                Class.forName("com.mysql.cj.jdbc.Driver");
                connection = DriverManager.getConnection(URL, USER,
PASSWORD);
                System.out.println("✅ Database connected successfully!");
            } catch (ClassNotFoundException | SQLException e) {
                System.err.println("❌ Database connection failed: " +
e.getMessage());
            }
        }
        return connection;
    }

    // Close connection
    public static void closeConnection() {
        if (connection != null) {
            try {
                connection.close();
                System.out.println("🔒 Database connection closed.");
            } catch (SQLException e) {
                System.err.println("❌ Error closing connection: " +
e.getMessage());
            }
        }
    }
}
```

2. Book Data Access Object (BookDAO.java)

```
package library.dao;

import library.model.Book;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class BookDAO {

    public void addBook(Book book) {
        String sql = "INSERT INTO books (title, author, category,
copies_available, digital_link) VALUES (?, ?, ?, ?, ?)";
        try (Connection conn = DBConnection.getConnection();
             PreparedStatement ps = conn.prepareStatement(sql)) {

            ps.setString(1, book.getTitle());
            ps.setString(2, book.getAuthor());
```

```
            ps.setString(3, book.getCategory());
            ps.setInt(4, book.getCopiesAvailable());
            ps.setString(5, book.getDigitalLink());
            ps.executeUpdate();

            System.out.println("Book added successfully.");
        } catch (SQLException e) {
            System.err.println(" Error adding book: " + e.getMessage());
        }
    }

    public List<Book> getAllBooks() {
        List<Book> list = new ArrayList<>();
        String sql = "SELECT * FROM books";

        try (Connection conn = DBConnection.getConnection();
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {

            while (rs.next()) {
                Book b = new Book(
                    rs.getInt("book_id"),
                    rs.getString("title"),
                    rs.getString("author"),
                    rs.getString("category"),
                    rs.getInt("copies_available"),
                    rs.getString("digital_link")
                );
                list.add(b);
            }
        } catch (SQLException e) {
            System.err.println(" Error fetching books: " + e.getMessage());
        }

        return list;
    }

    public void updateBookCopies(int bookId, int copies) {
        String sql = "UPDATE books SET copies_available = ? WHERE book_id =
?";
        try (Connection conn = DBConnection.getConnection();
             PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setInt(1, copies);
            ps.setInt(2, bookId);
            ps.executeUpdate();
            System.out.println(" Book copies updated successfully.");
        } catch (SQLException e) {
            System.err.println(" Error updating book: " + e.getMessage());
        }
    }
}
```

3. Student Data Access Object (StudentDAO.java)

```
package library.dao;

import library.model.Student;
```

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class StudentDAO {

    public void addStudent(Student student) {
        String sql = "INSERT INTO students (name, department, email, phone)
VALUES (?, ?, ?, ?)";
        try (Connection conn = DBConnection.getConnection();
             PreparedStatement ps = conn.prepareStatement(sql)) {

            ps.setString(1, student.getName());
            ps.setString(2, student.getDepartment());
            ps.setString(3, student.getEmail());
            ps.setString(4, student.getPhone());
            ps.executeUpdate();

            System.out.println(" Student added successfully.");
        } catch (SQLException e) {
            System.err.println(" Error adding student: " + e.getMessage());
        }
    }

    public List<Student> getAllStudents() {
        List<Student> list = new ArrayList<>();
        String sql = "SELECT * FROM students";

        try (Connection conn = DBConnection.getConnection();
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {

            while (rs.next()) {
                Student s = new Student(
                    rs.getInt("student_id"),
                    rs.getString("name"),
                    rs.getString("department"),
                    rs.getString("email"),
                    rs.getString("phone")
                );
                list.add(s);
            }
        } catch (SQLException e) {
            System.err.println(" Error fetching students: " +
e.getMessage());
        }

        return list;
    }
}
```

4. Borrowing Data Access Object (BorrowingDAO.java)

```
package library.dao;

import library.model.Borrowing;
```

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class BorrowingDAO {

    public void recordBorrowing(Borrowing b) {
        String sql = "INSERT INTO borrowings (student_id, book_id,
librarian_id, borrow_date, due_date, fine_amount) VALUES (?, ?, ?, ?, ?, ?)";
        try (Connection conn = DBConnection.getConnection();
             PreparedStatement ps = conn.prepareStatement(sql)) {

            ps.setInt(1, b.getStudentId());
            ps.setInt(2, b.getBookId());
            ps.setInt(3, b.getLibrarianId());
            ps.setDate(4, Date.valueOf(b.getBorrowDate()));
            ps.setDate(5, Date.valueOf(b.getDueDate()));
            ps.setDouble(6, b.getFineAmount());
            ps.executeUpdate();

            System.out.println("Borrowing recorded successfully.");
        } catch (SQLException e) {
            System.err.println("Error recording borrowing: " +
e.getMessage());
        }
    }

    public void markReturned(int borrowId, java.time.LocalDate returnDate) {
        String sql = "UPDATE borrowings SET return_date = ? WHERE borrow_id =
?";
        try (Connection conn = DBConnection.getConnection();
             PreparedStatement ps = conn.prepareStatement(sql)) {

            ps.setDate(1, Date.valueOf(returnDate));
            ps.setInt(2, borrowId);
            ps.executeUpdate();

            System.out.println("Book return recorded.");
        } catch (SQLException e) {
            System.err.println("Error updating return: " + e.getMessage());
        }
    }

    public List<Borrowing> getAllBorrowings() {
        List<Borrowing> list = new ArrayList<>();
        String sql = "SELECT * FROM borrowings";

        try (Connection conn = DBConnection.getConnection();
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {

            while (rs.next()) {
                Borrowing b = new Borrowing(
                    rs.getInt("borrow_id"),
                    rs.getInt("student_id"),
                    rs.getInt("book_id"),
                    rs.getInt("librarian_id"),
```

```
                    rs.getDate("borrow_date").toLocalDate(),
                    rs.getDate("due_date").toLocalDate(),
                    rs.getDate("return_date") != null ?
rs.getDate("return_date").toLocalDate() : null,
                    rs.getDouble("fine_amount")
                );
                list.add(b);
            }
        } catch (SQLException e) {
            System.err.println(" Error fetching borrowings: " +
e.getMessage());
        }

        return list;
    }
}
```

5. Librarian Data Access Object (LibrarianDAO.java)

```
package library.dao;

import library.model.Librarian;
import java.sql.*;

public class LibrarianDAO {

    public boolean authenticate(String username, String password) {
        String sql = "SELECT * FROM librarians WHERE username = ? AND
password = ?";
        try (Connection conn = DBConnection.getConnection();
             PreparedStatement ps = conn.prepareStatement(sql)) {

            ps.setString(1, username);
            ps.setString(2, password);
            ResultSet rs = ps.executeQuery();
            return rs.next();
        } catch (SQLException e) {
            System.err.println(" Authentication failed: " + e.getMessage());
        }
        return false;
    }

    public void addLibrarian(Librarian l) {
        String sql = "INSERT INTO librarians (name, username, password)
VALUES (?, ?, ?)";
        try (Connection conn = DBConnection.getConnection();
             PreparedStatement ps = conn.prepareStatement(sql)) {

            ps.setString(1, l.getName());
            ps.setString(2, l.getUsername());
            ps.setString(3, l.getPassword());
            ps.executeUpdate();
            System.out.println(" Librarian added successfully.");
        } catch (SQLException e) {
            System.err.println(" Error adding librarian: " + e.getMessage());
        }
    }
```

```
}
```

6. Model Classes (Example)

Here's one as a reminder (you already have them, but for clarity):

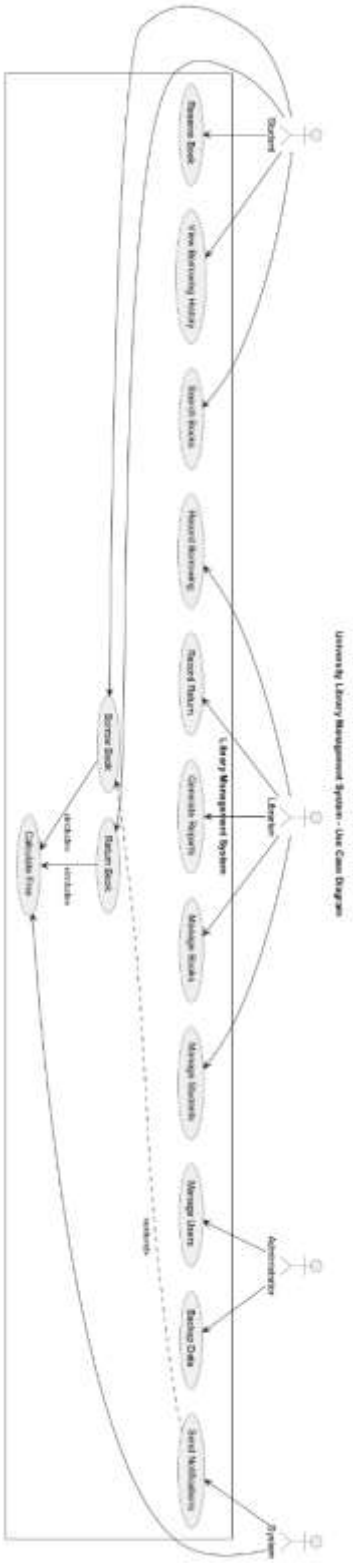
```
package library.model;

public class Book {
    private int bookId;
    private String title;
    private String author;
    private String category;
    private int copiesAvailable;
    private String digitalLink;

    public Book(int bookId, String title, String author, String category, int
copiesAvailable, String digitalLink) {
        this.bookId = bookId;
        this.title = title;
        this.author = author;
        this.category = category;
        this.copiesAvailable = copiesAvailable;
        this.digitalLink = digitalLink;
    }

    public Book(String title, String author, String category, int
copiesAvailable, String digitalLink) {
        this(0, title, author, category, copiesAvailable, digitalLink);
    }

    // Getters & Setters
    public int getBookId() { return bookId; }
    public String getTitle() { return title; }
    public String getAuthor() { return author; }
    public String getCategory() { return category; }
    public int getCopiesAvailable() { return copiesAvailable; }
    public String getDigitalLink() { return digitalLink; }

    public void setCopiesAvailable(int copiesAvailable) {
this.copiesAvailable = copiesAvailable; }
}
```

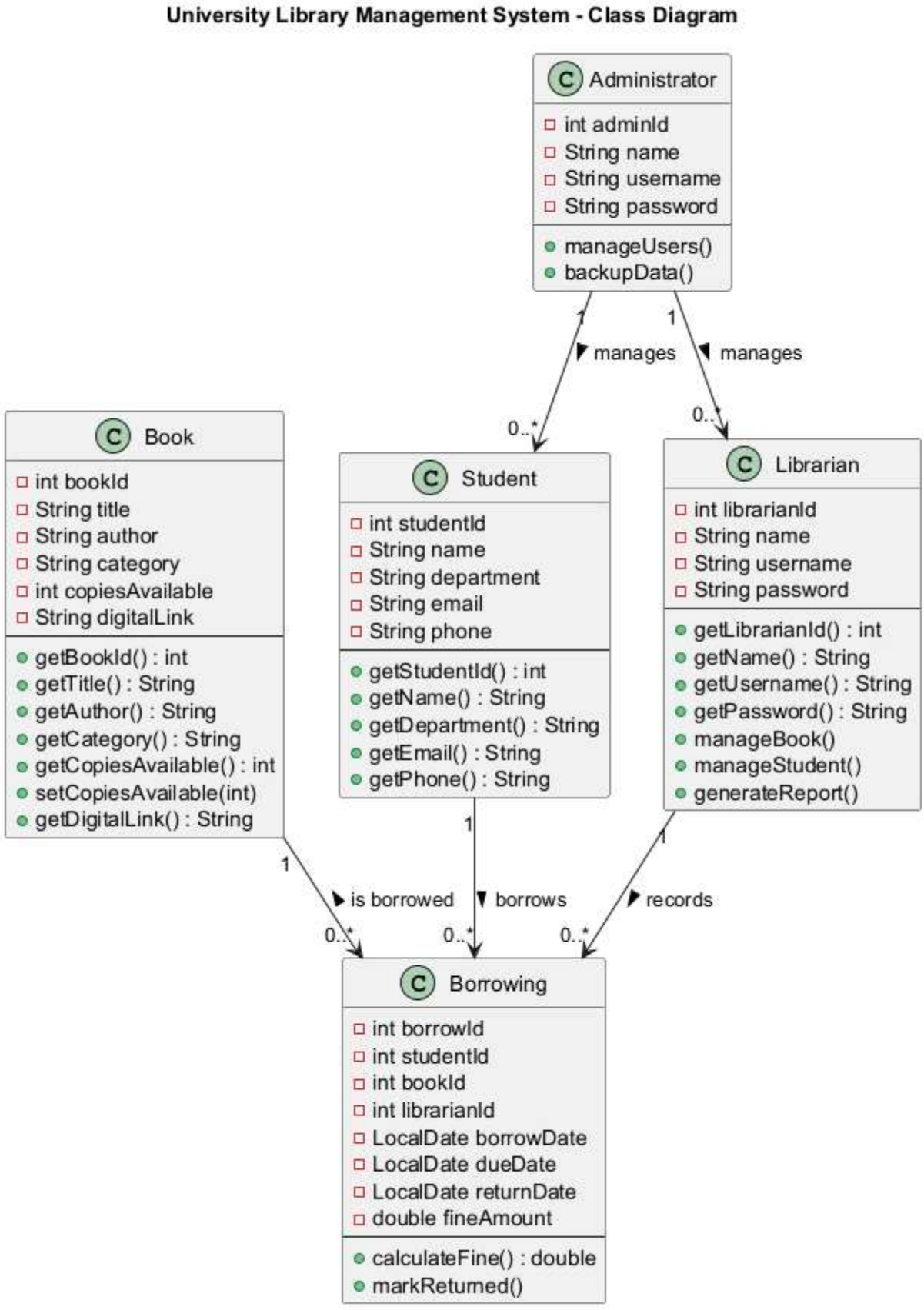
7. Test Example (Main.java)

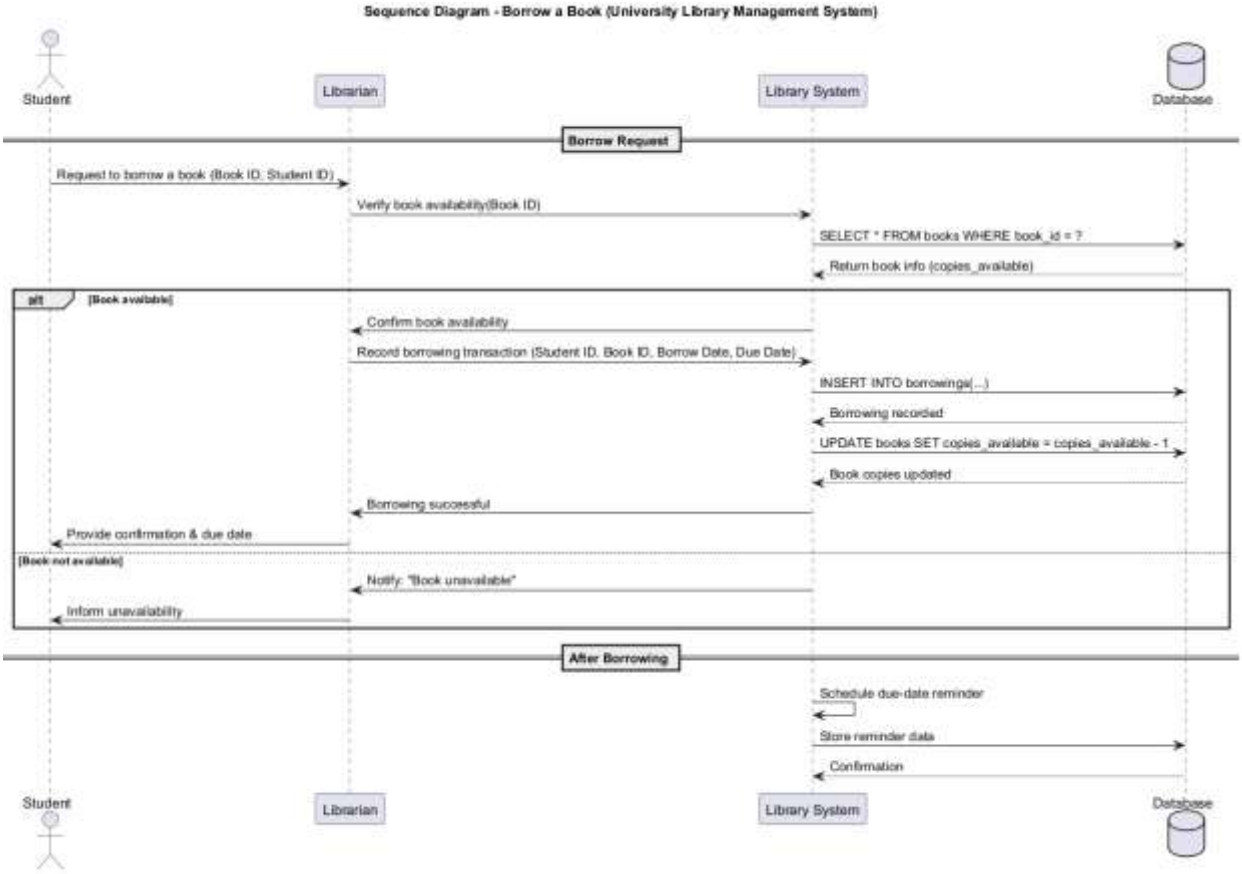
```
package library;

import library.dao.*;
import library.model.*;
import java.util.List;

public class Main {
    public static void main(String[] args) {
```

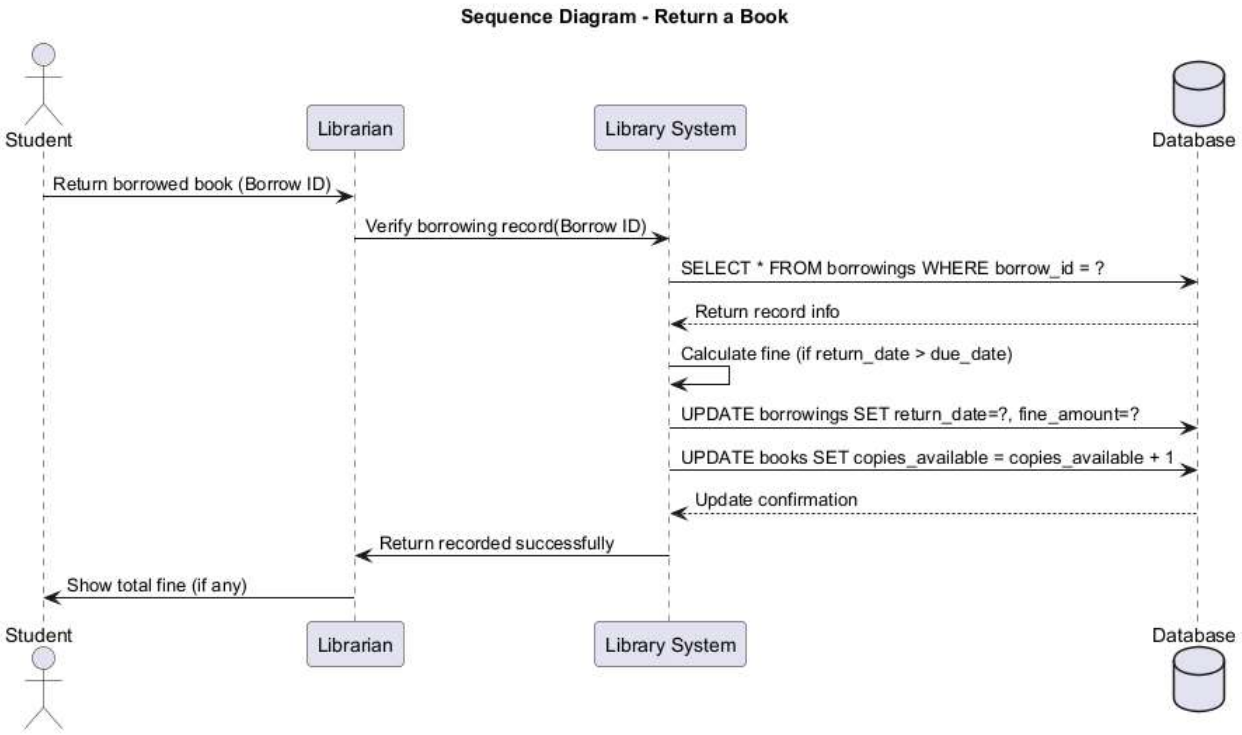
```
        BookDAO bookDAO = new BookDAO();
        StudentDAO studentDAO = new StudentDAO();
        LibrarianDAO librarianDAO = new LibrarianDAO();

        // Add a book
        Book b = new Book("Java Programming", "Herbert Schildt",
"Programming", 5, null);
        bookDAO.addBook(b);

        // Add a student
        Student s = new Student(0, "Mohamed Karim", "Computer Science",
"mkarim@mail.com", "0555123456");
        studentDAO.addStudent(s);

        // Test authentication
        boolean loggedIn = librarianDAO.authenticate("rachid", "1234");
        System.out.println(loggedIn ? "Login success" : " Login failed");

        // Display books
        List<Book> books = bookDAO.getAllBooks();
        books.forEach(book -> System.out.println(book.getTitle() + " by " +
book.getAuthor()));
    }
}
```

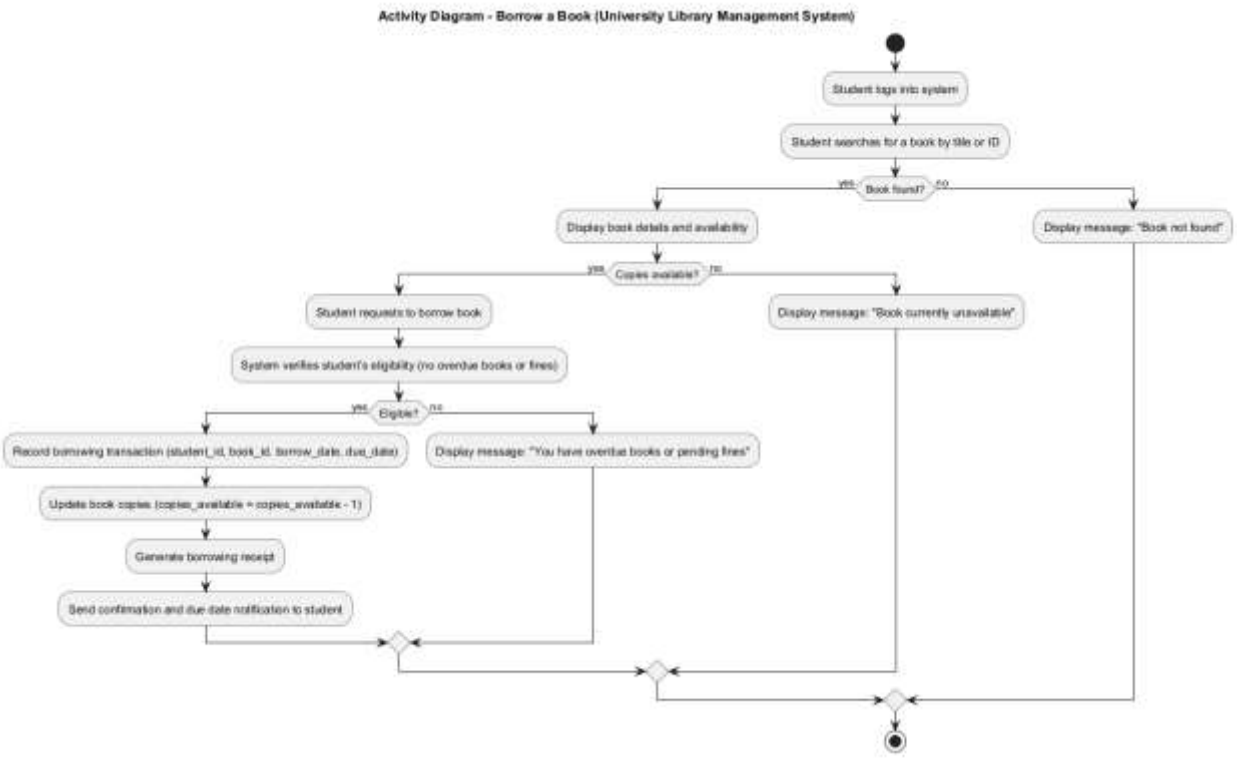
UML Diagram of this project

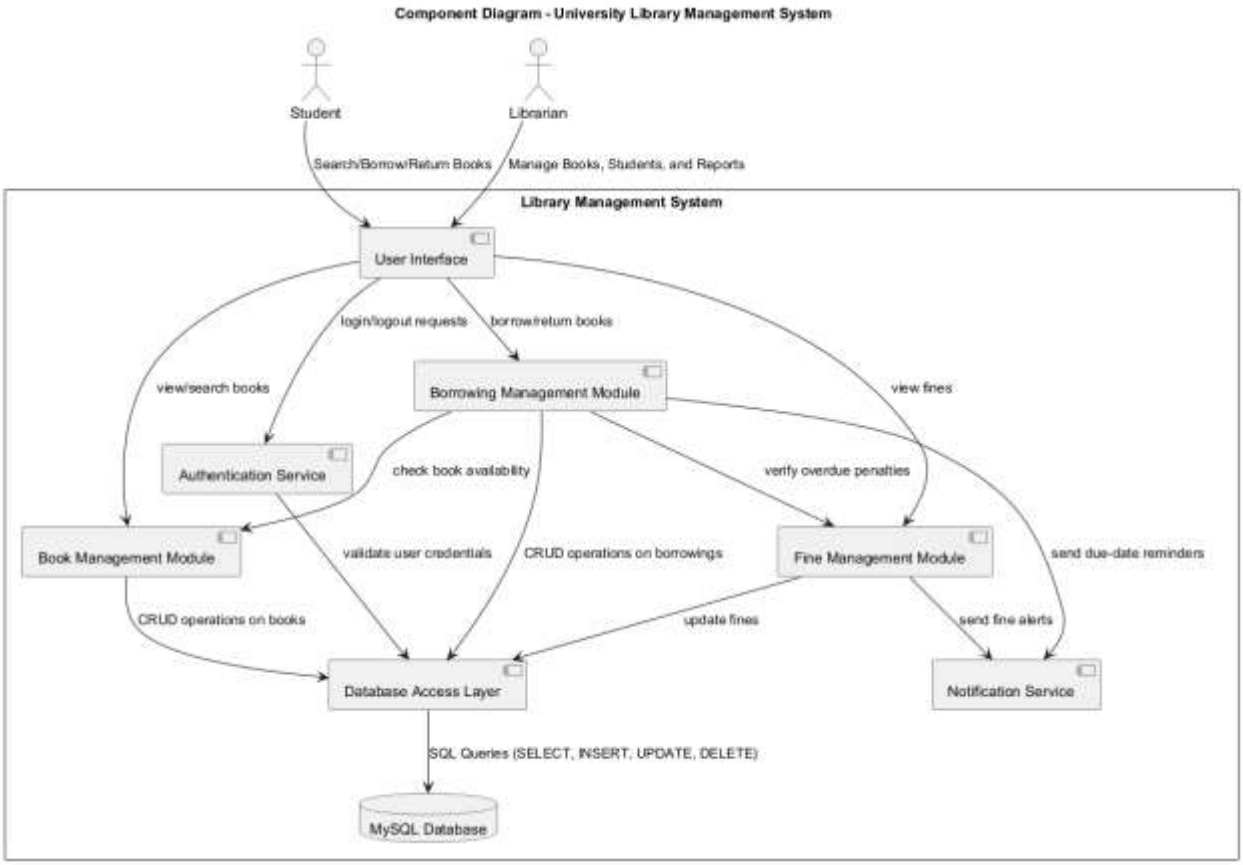


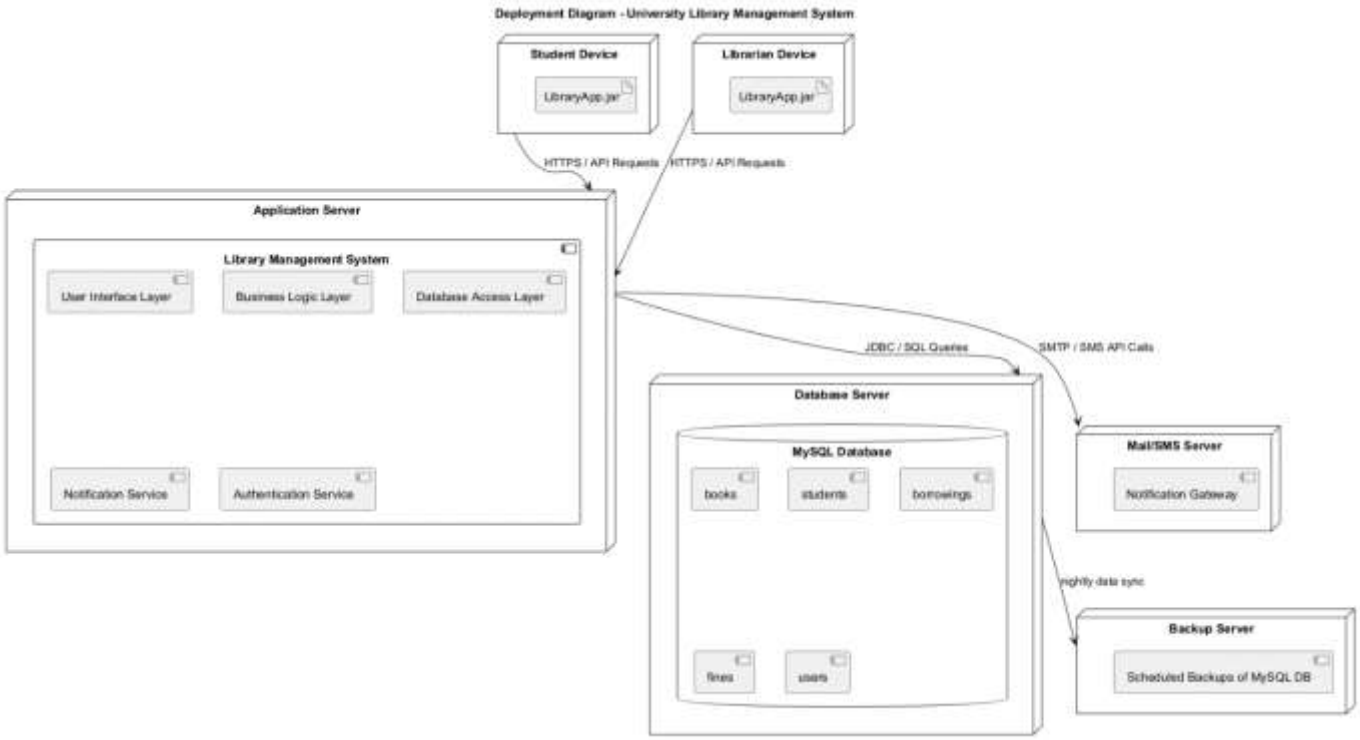

University Library Management System - Class Diagram
Administrator
int adminId
String name
String username
String password
manageUsers()
backupData()
1
manages
0..*
1
manages
0..*
Book
int bookId
String title
String author
String category
int copiesAvailable
String digitalLink
getBookId() : int
getTitle() : String
getAuthor() : String
getCategory() : String
getCopiesAvailable() : int
setCopiesAvailable(int)
getDigitalLink() : String
Student
int studentId
String name
String department
String email
String phone
getStudentId() : int
getName() : String
getDepartment() : String
getEmail() : String
getPhone() : String
Librarian
int librarianId
String name
String username
String password
getLibrarianId() : int
getName() : String
getUsername() : String
getPassword() : String
manageBook()
manageStudent()
generateReport()
1
is borrowed
0..*
1
borrows
0..*
1
records
0..*
Borrowing
int borrowId
int studentId
int bookId
int librarianId
LocalDate borrowDate
LocalDate dueDate
LocalDate returnDate
double fineAmount
calculateFine() : double
markReturned()

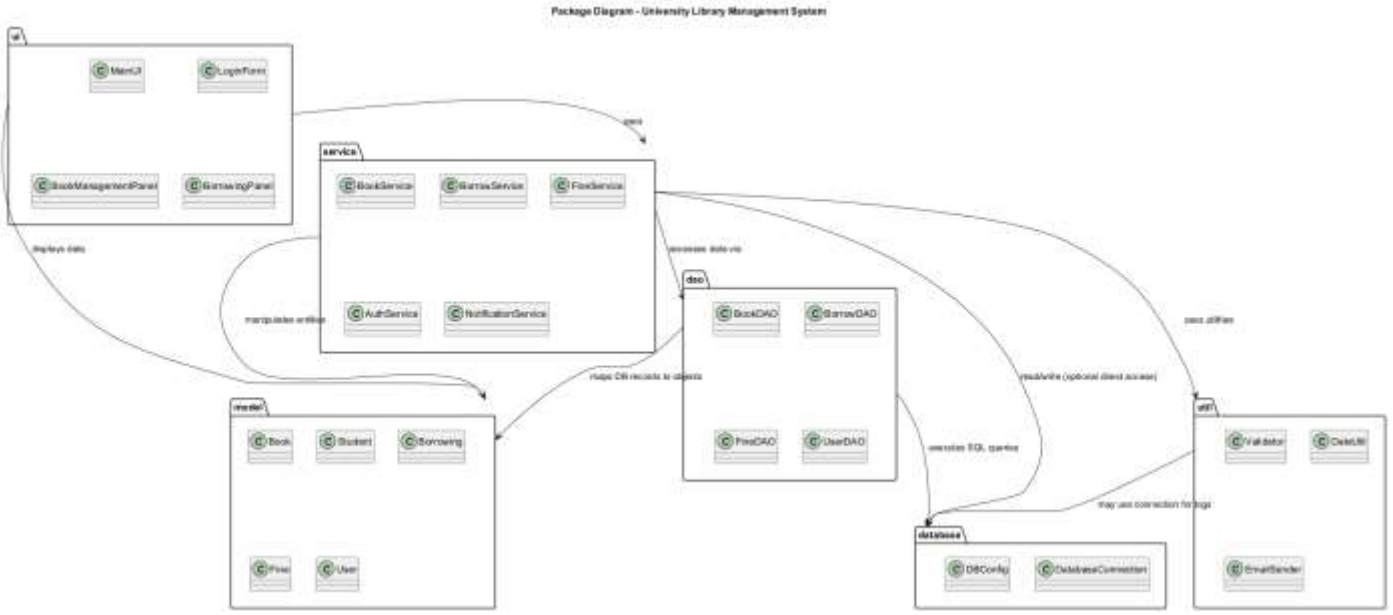

Sequence Diagram - Borrow a Book (University Library Management System)
Student
Librarian
Library System
Database
Borrow Request
Request to borrow a book (Book ID, Student ID)
Verify book availability(Book ID)
SELECT * FROM books WHERE book_id = ?
Return book info (copies_available)
alt
[Book available]
Confirm book availability
Record borrowing transaction (Student ID, Book ID, Borrow Date, Due Date)
INSERT INTO borrowings(...)
Borrowing recorded
UPDATE books SET copies_available = copies_available - 1
Book copies updated
Borrowing successful
Provide confirmation & due date
[Book not available]
Notify: "Book unavailable"
Inform unavailability
After Borrowing
Schedule due-date reminder
Store reminder data
Confirmation
Student
Librarian
Library System
Database

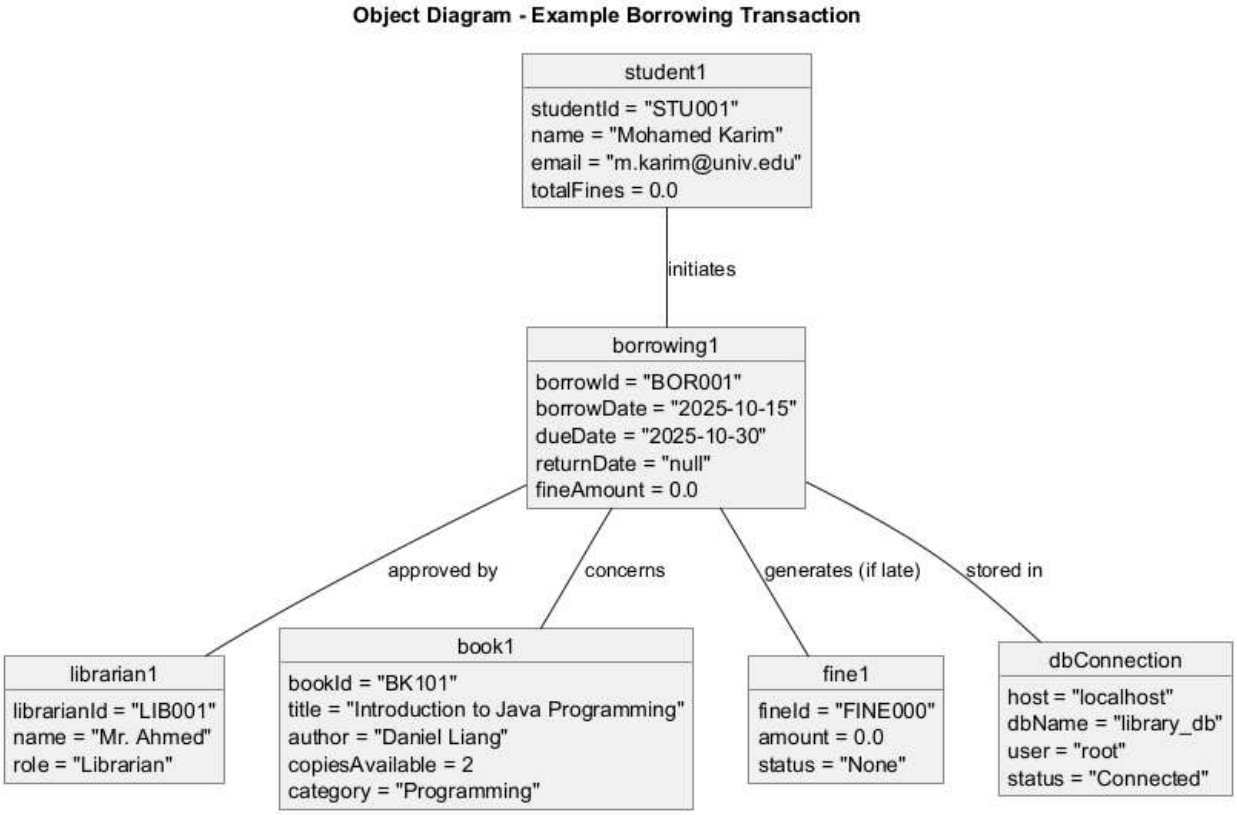

Sequence Diagram - Return a Book
Student
Librarian
Library System
Database
Return borrowed book (Borrow ID)
Verify borrowing record(Borrow ID)
SELECT * FROM borrowings WHERE borrow_id = ?
Return record info
Calculate fine (if return_date > due_date)
UPDATE borrowings SET return_date=?, fine_amount=?
UPDATE books SET copies_available = copies_available + 1
Update confirmation
Return recorded successfully
Show total fine (if any)
Student
Librarian
Library System
Database


Activity Diagram - Borrow a Book (University Library Management System)
Student logs into system
Student searches for a book by title or ID
Book found?
yes
no
Display book details and availability
Display message: "Book not found"
Copies available?
yes
no
Student requests to borrow book
Display message: "Book currently unavailable"
System verifies student's eligibility (no overdue books or fines)
Eligible?
yes
no
Record borrowing transaction (student_id, book_id, borrow_date, due_date)
Display message: "You have overdue books or pending fines"
Update book copies (copies_available = copies_available - 1)
Generate borrowing receipt
Send confirmation and due date notification to student


Component Diagram - University Library Management System
Student
Librarian
Search/Borrow/Return Books
Manage Books, Students, and Reports
Library Management System
User Interface
login/logout requests
borrow/return books
view/search books
view fines
Borrowing Management Module
Authentication Service
check book availability
verify overdue penalties
Book Management Module
validate user credentials
CRUD operations on borrowings
Fine Management Module
send due-date reminders
CRUD operations on books
update fines
send fine alerts
Database Access Layer
Notification Service
SQL Queries (SELECT, INSERT, UPDATE, DELETE)
MySQL Database


Deployment Diagram - University Library Management System
Student Device
LibraryApp.jar
Librarian Device
LibraryApp.jar
HTTPS / API Requests
HTTPS / API Requests
Application Server
Library Management System
User Interface Layer
Business Logic Layer
Database Access Layer
Notification Service
Authentication Service
JDBC / SQL Queries
SMTP / SMS API Calls
Database Server
MySQL Database
books
students
borrowings
fines
users
Mail/SMS Server
Notification Gateway
nightly data sync
Backup Server
Scheduled Backups of MySQL DB


Package Diagram - University Library Management System
ui
MainUI
LoginForm
BookManagementPanel
BorrowingPanel
uses
service
BookService
BorrowService
FineService
AuthService
NotificationService
dao
BookDAO
BorrowDAO
FineDAO
UserDAO
model
Book
Student
Borrowing
Fine
User
database
DBConfig
DatabaseConnection
util
Validator
DateUtil
EmailSender


Object Diagram - Example Borrowing Transaction
student1
studentId = "STU001"
name = "Mohamed Karim"
email = "m.karim@univ.edu"
totalFines = 0.0
initiates
borrowing1
borrowId = "BOR001"
borrowDate = "2025-10-15"
dueDate = "2025-10-30"
returnDate = "null"
fineAmount = 0.0
approved by
concerns
generates (if late)
stored in
librarian1
librarianId = "LIB001"
name = "Mr. Ahmed"
role = "Librarian"
book1
bookId = "BK101"
title = "Introduction to Java Programming"
author = "Daniel Liang"
copiesAvailable = 2
category = "Programming"
fine1
fineId = "FINE000"
amount = 0.0
status = "None"
dbConnection
host = "localhost"
dbName = "library_db"
user = "root"
status = "Connected"

References

LECTURE NOTES ON SOFTWARE ENGINEERING Course Code: BCS-306 By Dr. H.S.Behera Asst. Prof K.K.Sahu Asst. Prof Gargi Bhattacharjee

I. Sommerville, Software Engineering, 10th ed. Boston, MA: Pearson, 2015.

F. P. Brooks Jr., The Mythical Man-Month: Essays on Software Engineering, 2nd ed. Boston, MA: Addison-Wesley, 1995.

S. McConnell, Code Complete: A Practical Handbook of Software Construction, 2nd ed. Redmond, WA: Microsoft Press, 2004.

E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Boston, MA: Addison-Wesley, 1994.