People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
Faculty of Science and Technology
Department of Mathematics and Computer Science
University of Ghardaia

3rd Year License Computer Science (LMD)

Course Handout

Software Engineering

Dr Ahmed Saidi

Academic year: 2025-2026

Contents

Introduction

Scope of the Subject

Software engineering is a discipline of computer science that applies scientific knowledge to build cost-effective solutions for computing and information processing challenges. It focuses on developing software systems that benefit humanity. This course teaches the principles of software engineering, including system requirements, engineering compromises, design, coding, and testing techniques, team software development, and tool application.

Course Pre-Requisites

- Familiar with the fundamental concepts of Algorithms, information systems, and object oriented programming (OOP).

Course Objectives

- Learn object modelling with the universal language UML
- Illustrate basic taxonomy and terminology of software engineering.
- Plan and monitor the control aspects of the project.

Course Outcomes

- Understand how software is built.
- Be able to understand a customer needs and follow a software development process.
- Develop software development skills.

Chapter 1

Introduction to Software Engineering

1.1 Terminology

- **Model:** representation of something or simplification of reality using a modeling language. Modeling languages can be classified as Formal modeling languages like math, Semi-formal modeling languages like graphs and diagrams, or informal modeling languages such as pseudo code or natural language (Figure 1.1).

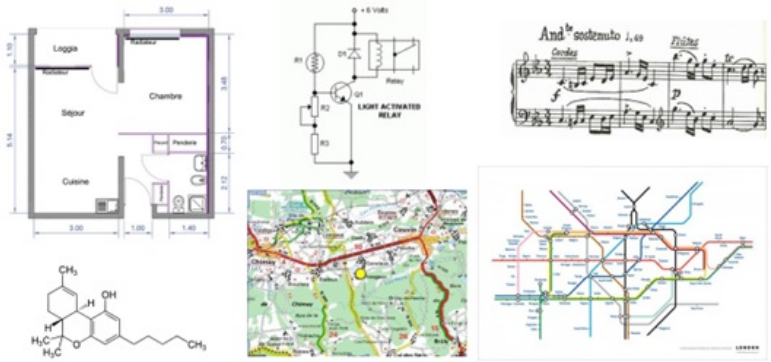


Figure 1.1: Example of models

- **Process:** A series of successive steps.
- **Engineering:** A set of methods, techniques, and tools dedicated to the design, development, and maintenance of industrial projects.
- **Prototype:** an early realization of a product, it's simple to illustrate how a future product looks (Figure 1.2).

Figure 1.2: Example of prototype

- **Software engineering:** apply engineering methods to the computer science field. Using the same approach, software engineering can be defined as a set of methods, techniques, and tools dedicated to the design, development, and maintenance of **software**.
- **Stakeholders:** The people or groups affected by a software development project inside the organization. Stakeholders exist either inside or outside the organization. For example, a customer or end user is a type of outside stakeholder, whereas a project manager is a type of inside stakeholder.
- **Software Requirement Specification (SRS) document:** A document that describes client requirements that need to be fulfilled for the successful development of the software.

1.2 The evolving role of software

- The role of computer software has undergone significant change over the last 50 years.
- Dramatic improvements in hardware performance, profound changes in computing architectures, vast increases in memory and storage capacity, and a wide variety of exotic input and output options have led to the development of sophisticated and complex computer-based systems.
- Popular books published during the 1970s and 1980s described the changing nature of computers and software and their impact on our culture. Some of them stated that computers and software caused a new industrial revolution which led to a transformation from an industrial society to an information society.

1.3 Positive impacts of software

- It speeds up processing.
- It increases work efficiency.
- It quickly solves complex problems.
- It has incredible computing, storage, and processing power.
- It has introduced new leisure activities.
- It introduces a new social dimension.

1.4 Impacts of low-quality software

Several disasters of varying degrees of severity have been caused by software "errors":

- In 1962, a space rocket was thrown off course by a mathematical formula that had been incorrectly transcribed into source code.
- In 1985, a machine was designed to treat the patients. Due to a bug in its software, at least five patients died.
- In 1983, at the height of the Cold War, Soviet surveillance software detected fake ballistic missiles being sent from the USA.

1.5 What makes software development difficult and complex?

Before responding to this question, let's know the difference between a normal product (ex: a car) and software(Figure 1.3).

Figure 1.3: Product vs Software

Table 1.1: Product vs Software

Car	**Software**
Concrete product	Product not concrete
Product Manufactured	Product developed
Components are easy to assemble and reuse	Difficult to assemble or reuse
Maintenance by component replacement	Maintenance by the developer itself
Development process based on machines	Development process based on humans

As we can see, software is an abstract product developed entirely by humans. As humans, we have always difficulty in matching between what we say and what we want(Figure 1.4). This language barrier makes it difficult for developers to understand customer needs. Software development difficulties can be summarized as follows:

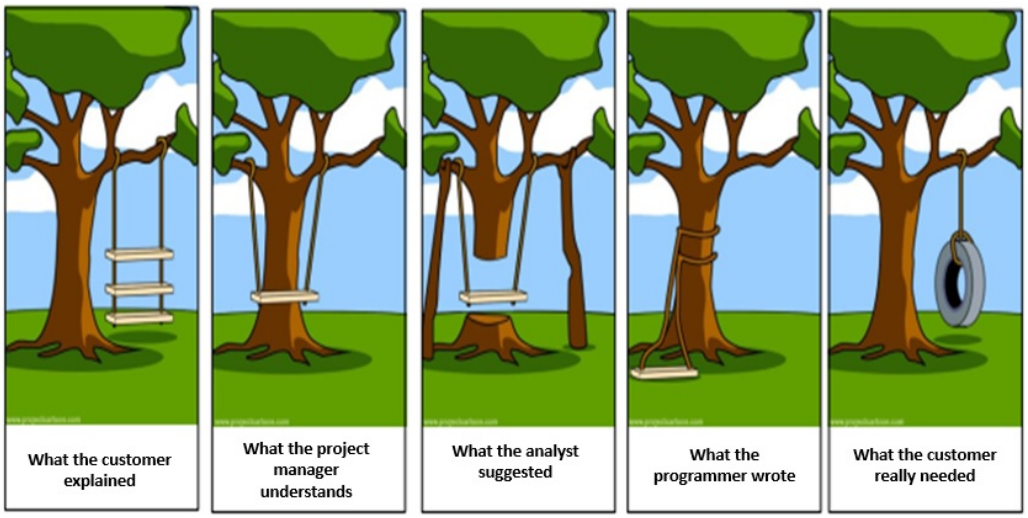


Figure 1.4: Reasons for low software quality

- Software is invisible and not concrete.
- Software failure mainly by human mistakes.
- Difficult to measure software quality.
- Critical consequences caused by minute modifications.
- Continues updates and maintenance because of the rapidly evolving technology.

- Deployment and programming errors.
- Mainly human software failures.

1.6 Software crisis

During the 1960s and 1970s, a crisis happened in the software world, which identified many of the problems of software development such as:

- Delivery times not respected.
- Budgets not respected.
- Does not meet user or customer needs.
- Difficult to use, maintain and upgrade.

1.7 Reasons for low software quality

- Software Complexity.
- Software is an invisible product.
- Difficult to measure software quality.
- Size of development teams.
- Lack of design methods.
- Lack of methods and tools for validation/verification phases.
- Neglect of the customer needs.
- Lack of customer involvement in the development process.

1.8 The challenges of software quality

- How to develop the right software that meets the customer's needs ?
- What do we expect from software ?.
- What are the quality criteria ?.
- How do you make quality software ?.

1.9 Software quality criteria

Software Quality shows how good and reliable a software product is. Like any product, the International Organization for Standardization (ISO) establishes a set of attributes to produce high-quality software such as(Figure 1.5):

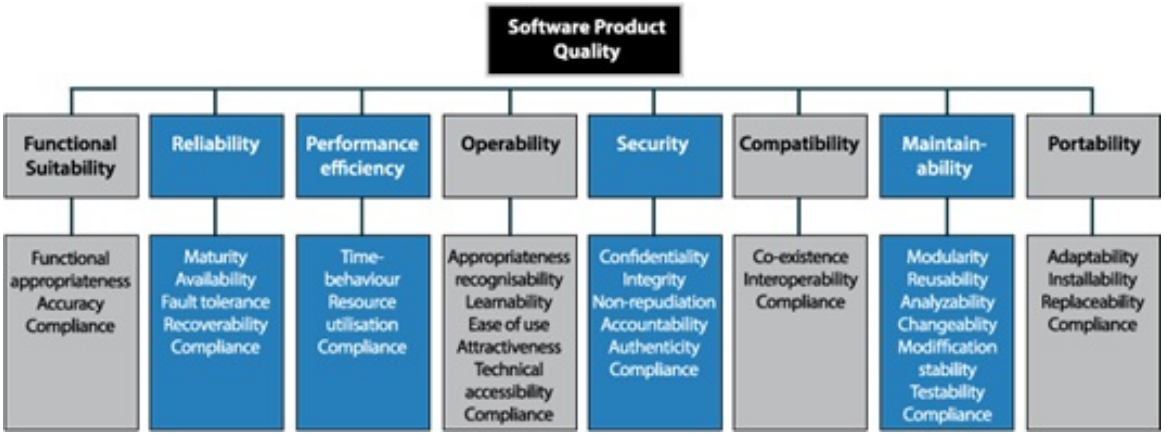


Figure 1.5: ISO 25010 software quality attributes

1. **Functional Suitability:** The software meets client needs and requirements.
   To achieve this attribute
   - Check SRS document
   - Check the software functions
   - Get the client's feedback
2. **Usability:** The software interface must be simple and easy to use.
   To achieve this attribute
   - Include UI/UX designer to your team.
   - Use professional design tools.
3. **Performance:** The software has good response time, throughput and fluidity.
   To achieve this attribute
   - Simplify the software.
   - Check algorithm complexity (loops, recursive functions...etc.).
   - Set computer requirements.

4. **Reliability:** The software must be fault tolerance.
   To achieve this attribute
   - Use software fault tolerance techniques such as: N-version programming, recovery blocks, rollback recovery...etc.
5. **Security:** The software protects its data and controls accessibility.
   To achieve this attribute
   - Eliminate vulnerability from your code.
   - Protect software data using cryptography and access control.
   - Set user roles and permissions.
   - Treat security risks at the design phase.
6. **Maintainability:** ease of correcting or transforming the software.
   To achieve this attribute
   - Check the usability, extensibility, and modularity of the software.
   - Anticipate future change.
7. **Portability:** ease of changing the environment (hardware or software ex: OS).
   To achieve this attribute
   - Make software independent of its execution environment.
   - Use virtual machines.
8. **Compatibility:** The Software must be able to interact synergistically with other software.
   To achieve this attribute
   - Use middleware and API (application programming interface).
   - Use standard file format (XML, TXT…etc.).
   - Use standard communication protocols (HTTP, HTTPS…etc.).

1.10 Exercices and solutions

1. For each software type (video game, mail server and anitvirus), rate their quality attributes (high, medium, or low).

Software	Functional Suitability	Usability	Performance	Reliability	Security	Maintainability	Portability	Compatibility
Video game								
Mail server								
Anitvirus								

2. What qualities does this software lack?

    - The software doesn't provide all the expected functionalities.
    - Software learning is so difficult.
    - Software results are sometimes erroneous.
    - The software consumes a lot of CPU for seemingly simple requests.

Solutions

Software	Functional Suitability	Usability	Performance	Reliability	Security	Maintainability	Portability	Compatibility
Video game	Low	High	High	Low	Low	High	Low	Low
Mail server	High	medium	low	High	High	High	Low	low
Anitvirus	High	High	High	High	High	High	Low	Low

1. What qualities does this software lack?

    - The software doesn't provide all the expected functionalities. **Functional Suitability**
    - Software learning is so difficult. **Usability**
    - Software results are sometimes erroneous. **Reliability**
    - The software consumes a lot of CPU for seemingly simple requests. **Performance**

Chapter 2

The Software development life cycle

2.1 What is SDLC ?

The Software development life cycle (SDLC) is a process composed of successive and organized phases to produce a high-quality software. It's composed of 7 phases such as:

1. Feasibility study
2. Requirements analysis and specification
3. Design
4. Programming
5. Validation and verification
6. Delivery
7. Maintenance

SDLC phases are often described in terms of

- **Phase input:** The results of the previous phase.
- **Phase output:** The results of the current phase.
- **Phase Description:** The problem addressed by the phase. As a result, we must ask the following question:

  What is the problem to be addressed by this phase?.

2.1.1 Feasibility study (Why?)

This phase concerns the overall definition of the problem. We ask ourselves why we should develop the software in the first place. In this phase, we must find a response to the following questions:

- Why develop the software?
- Are there better alternatives?
- How do you go about this development?
- Is there a software market?
- What resources are needed?
- Do we have the budget, personnel, and necessary equipment?

2.1.2 Requirement analysis and specification (What?)

This phase concerns the software's functionalities. In this phase, we try to answer the question: What functionality must the software contain?; It has the following objectives:

- Understanding client requirements.
- Establish a clear description of what the software must do (detailed functionalities, quality requirements, interface ...etc.).
- Clarify specifications (ambiguities, contradictions) by listing functional and non-functional requirements.

Functional and non-functional requirements:

1. **Functional requirements:** Describe the **important** functionalities of the software. They explain how the software must work. For example, in ATM machines, the functional requirements are to Withdraw Money, Check Balance, Deposit Money .... etc.
2. **Non-functional requirements:** Describe the **general** properties and constraints of the software. They are also known as software attributes. They explain how the software should perform for example: Reliability, usability, security…etc.

2.1.3 Design (How ?)

This phase develops a concrete solution to meet the software specification. In this phase, we try to answer the question: How to translate client requirements and software specifications into a real program?. It like a bridge between the specification and programming phases. This phase is divided in two parts:

- **Overall design:** Define the main software components. For instance, to design a house we start looking for its main parts like stairs and rooms without specifying the details of each room.
- **Detailed design:** Specify the Details of each component. Back to the house example, in the detailed design we go inside each room.

2.1.4 Programming

In this phase we translate our design into a code. It's a coding phase, we start creating and implementing our design into real software. Therefore, we must choose the development environment, the programming language(s), development standards...etc.

2.1.5 Validation and verification

As its name implies, this phase is composed from two parts (Figure 2.1):

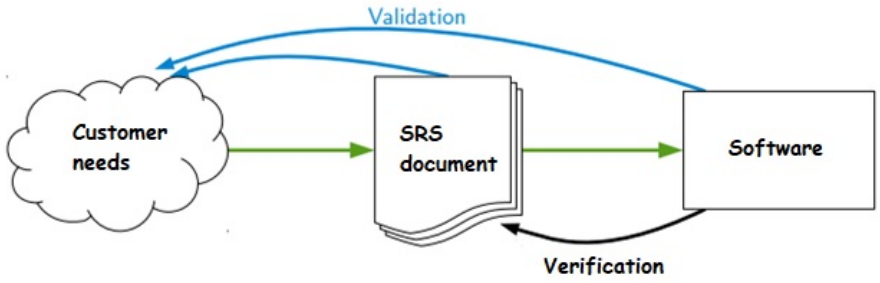


Figure 2.1: Validation and verification

- **Validation:** ensuring that the customer's needs are met (in terms of the client requirement).
- **Verification:** ensuring that the software meets its specification (using the SRS document).

2.1.6 Maintenance

The maintenance phase is necessary after the software delivery and the client feedback. Three types of maintenance can occur:

- **Corrective:** identify and correct errors found after delivery.
- **Adaptive:** adapt the software to changes in the environment (data format, execution environment, etc.).
- **Perfective:** enhance the software performance and functionalities; improve software maintainability.

2.2 SDLC model

SDLC models are divided into three types: linear, iterative, and incremental models.

2.2.1 Linear models

Organized sequentially with a clear distinction between steps (Figure 2.2). Among linear models, we found :

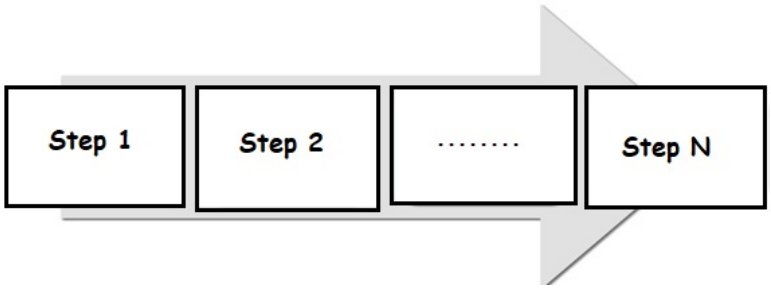


Figure 2.2: Linear models

Waterfall Model

This model has the following characteristics (Figure 2.3):

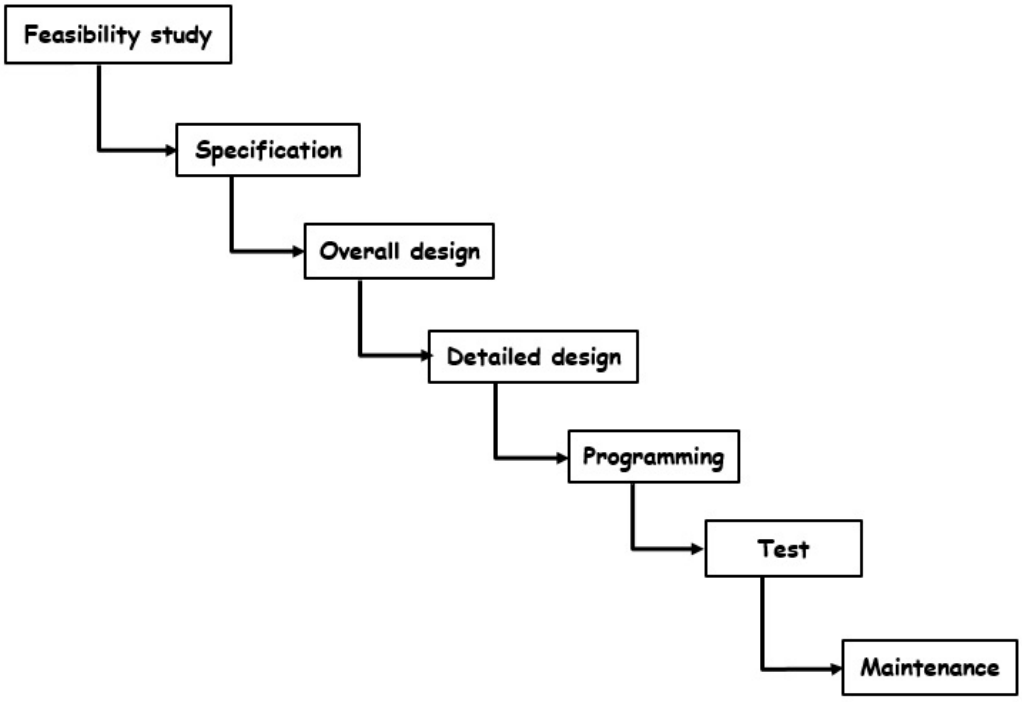


Figure 2.3: Waterfall Model

- The waterfall model is the most classic of all life cycles.
- Linear lifecycle with no evaluation between project start and validation.
- The project is divided into successive phases.
- Each phase corresponds to a specific main activity producing a certain number of deliverables.
- Each phase can only call into question the previous phase.

When we use it

- For big and complex projects, like government project.

Advantages

1. Sequential and strict model.
2. Clear phases.
3. Highly documents.

Disadvantages

- Real projects rarely follow a sequential development path.
- Establishing all requirements at the start of a project is difficult.
- Sensitivity to new requirements: redo all the steps.
- Well-suited when needs are clearly identified and stable.

"V" Model

This model has the following characteristics (Figure 2.5):

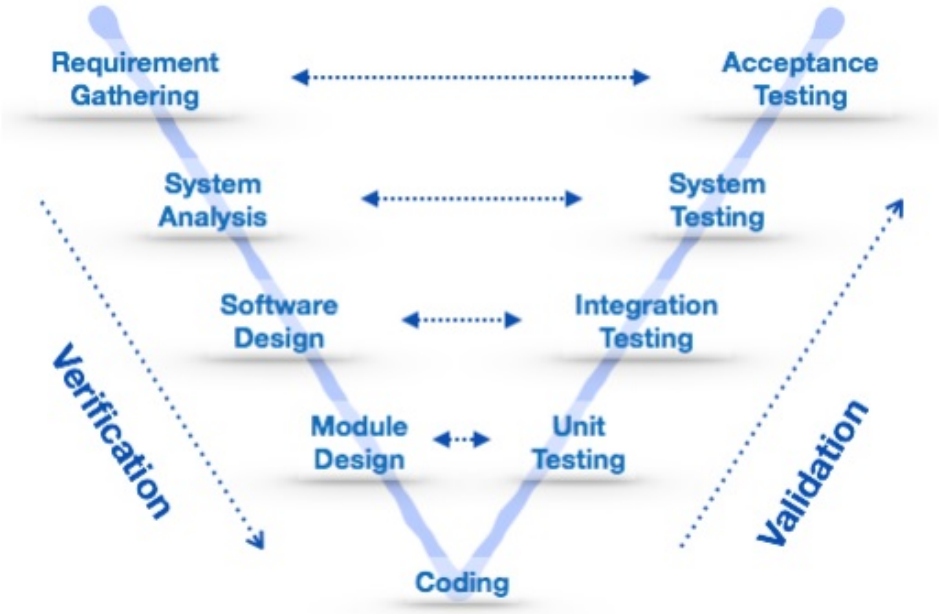


Figure 2.4: "V" Model

- Based on several tests. It's a test-oriented life cycle:
- Each phase (specification, design, and coding) has a corresponding verification activity (validation testing, integration testing, unit testing).
- Each upstream phase prepares the corresponding verification phase (verification is taken into account at the very moment of creation).

When we use it

- Ideal when needs are well known, and when analysis and design are clear.

Advantages

- Preparing the last phases (validation/verification) with the first (requirements analysis) avoids stating a property that cannot be objectively verified after implementation.

Disadvantages

- Are we building the right software? The software is used very (too) late.
- You have to wait a long time to find out if you've built the right software.
- It's hard to get users feedbacks when usable software isn't available only in the last phase.

Prototyping Model

This model has the following characteristics (Figure 2.5):

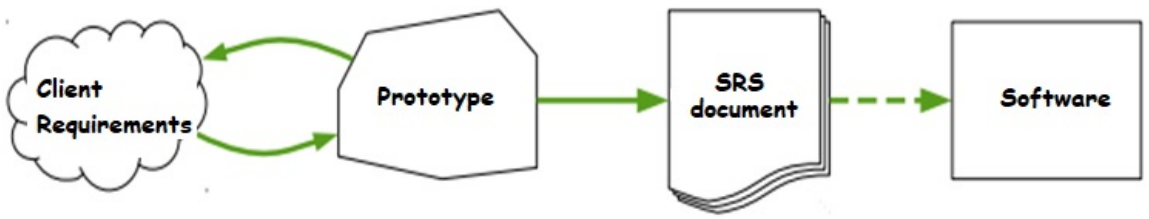


Figure 2.5: Prototyping Model

- Rapid prototype development with customers to validate their needs.
- Writing the specification from the prototype, followed by a linear development process.
- There are two types of prototype:
  1. **Disposable prototyping:** here, the prototype of the software is created for understanding client needs.
  2. **Evaluative prototyping:** here, prototype is kept throughout the development cycle. It is improved and completed to obtain the final software.

When we use it

- Concrete validation of requirements, less risk of specification errors.

Advantages

- The effort required to develop a prototype is more often than not offset by the effort saved by not developing unnecessary functions.

Disadvantages

- Quick decisions are rarely good decisions.
- Does the evolving prototype produce the required product?.

2.2.2 Incremental models

Figure 2.6: Incremental additions

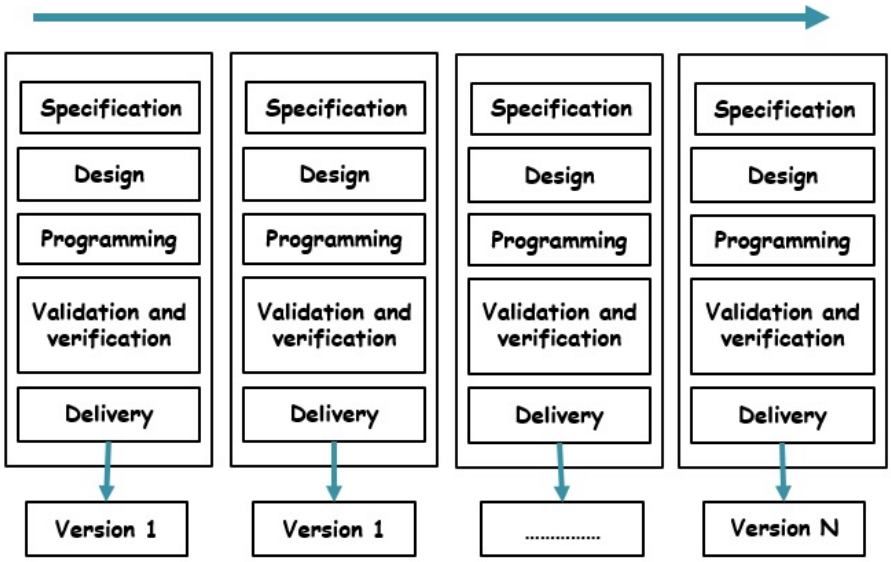


Figure 2.7: Incremental software subset

Incremental model has the following characteristics: It a step-by-step software development. For each step, we make:

1. Incremental additions until end of process (Figure 2.6).
2. A Minimal, functional software subset (Figure 2.7).

When we use it

- When the client demands a quick release of the software
- When the requirement are superior

Advantages

- Each development is less complex.
- Integration is progressive.
- Possible delivery after each increment.

Disadvantages

- Possible loss of the original software core;
- Possible loss of the previous increments;
- Possible errors when integrating a new increment.

2.2.3 Iterative models

Repetitive process until the software is completed and the customer is satisfied (Figure 2.8).

Figure 2.8: Iterative models

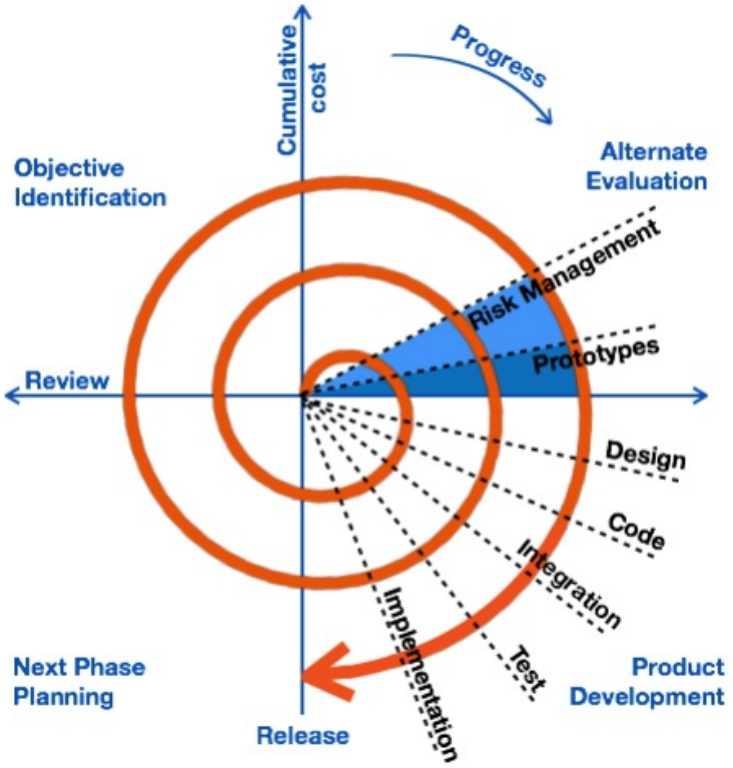


Figure 2.9: Spiral models

Spiral models

Among iterative models we found spiral model (Figure 2.9). Spiral Model has the following phases:

1. Customer consultation
2. Risk analysis
3. Design
4. Implementation
5. Tests
6. Planning for the next cycle

When we use it

- For critical software, like military or medical software.

Advantages

- Better risk management

Disadvantages

- It is not suitable for smaller projects.
- All the system architectue is dependent upon the risk analysis phase.

2.2.4 Agile model

The agile model combines incremental and iterative approaches. This model overcomes the rapid development of the technological environment and the instability of client requirements. Classic models expect the client to provide a detailed, validated expression of its requirements, which leads to a mismatch between the initial client requirements and the final software. The origin of agile models is linked to:

- The instability of the technological environment;
- The fact that customers are often unable to define their needs exhaustively at the early stage of a project. Thus, the term "agile" refers to the ability to adapt to contextual changes and specification modifications during the development process. Objectives of the agile model are:
- Accelerate the software development process by:
  1. Developing a minimal version (incremental and prototype approach);
  2. Integrating functionalities through an iterative process based on listening to the client and testing throughout the development cycle.
- Client satisfaction must be at the heart of every company's strategy.
- Include the client throughout the project realization process. (All agile methods apply this principle).

Agil models focus to:

- Get regular feedback so you can apply any necessary changes directly;
- Speed up software development.

2.2.5 Priorities for agile methods

- People and interactions are more important than processes and tools;
- Priority to code production over massive documentation;
- Collaboration with clients is preferable to contractual negotiation (the client must be able to provide continuous feedback on the software's adaptation to his expectations);
- Response to change is more important than following a flexibility and adaptation plan.

2.3 Classic vs Agile models

Classic	**Agile models**
Strict models	Incremental and iterative models
Very clearly defined steps	Small, frequent deliveries
Extensive documentation	Emphasis on code and less on documentation
Works well with large government projects	Suitable for small and medium-sized projects

2.4 Exercises and solutions

- Compare between waterfall, spiral, and V models.
- How to avoid low-quality software development?
- What is the purpose of specification in a software life cycle?
- Why do we follow a software lifecycle model?
- Which lifecycle model is best suited to the development of critical or medical software?

Solutions

SDLC	Advantages	Disadvantages
Waterfall	Simple and sequential	Difficult to update for new requirements
Spiral	Breakdown to simple tasks	Integration problems
V model	Based on tests	Too heavy

- How to avoid low-quality software development?
- **Through software engineering techniques and lifecycle models**
- What is the purpose of specification in a software life cycle?
- **Understanding the customer's needs and drawing up specifications. Establish a clear description of what the software needs to do**
- Why do we follow a software lifecycle model?
- **To develop quality software**
- Which lifecycle model is best suited to the development of critical or medical software?
- **Risk-oriented model (Spiral model)**

Chapter 3

Software Modeling

Diagrams	Objectif	View
Use case	Find user needs	Outside
System Sequence	Interaction scenarios between users and the software	Outside
Deployment	Physical log organization	Outside
State transition	Evolution of an object's state	Outside and inside
Activity	Sequence of actions representing software behavior	Outside and inside
Class	Internal software structure	Inside
Object	Software's internal state at a given moment	Inside
Object Sequence	Interaction scenarios with users or within the software	Inside
Component	Physical software components	Inside

3.1 Use Case diagram

3.1.1 Actor

What is it ?

- An actor is a **role** played by the user of the software system. In addition to natural persons, actors can be :
  - Peripherals handled by the system (printers...), robots, ...) ;
  - Software already available to be integrated into the project;
  - Computer systems external to the system but which interact with it, etc.

UML representation

- The actors are necessarily **outside** the system.
- Actors are often specified in the form of stylized characters (Figure 3.1).

Figure 3.1: Actor

- They can also be represented by a rectangle with the "actor" stereotype,or by a pictogram (e.g. a computer symbol)(Figure 3.2).

Figure 3.2: Actor

Warning

- An actor is a role, not a physical person.
- The same individual can be represented by several actors if he or she has several roles.
- If several people play the same role in the system, they will be represented by a single actor.
- An actor is not necessarily "human".

Primary or secondary actor

What is it ?

- A **primary actor** is the one for whom the use case produces an observable result.

Figure 3.3: Primary actor

- The **primary actor** initiates the exchanges required to carry out the use case (it is he who triggers the use case).
- **Secondary actors** are often asked for additional information; they can only consult or inform the system during the execution of the use case.

Figure 3.4: Secondary actor

- Wherever possible, place the primary actors on **the left** of the use cases and the secondary actors on **the right**.

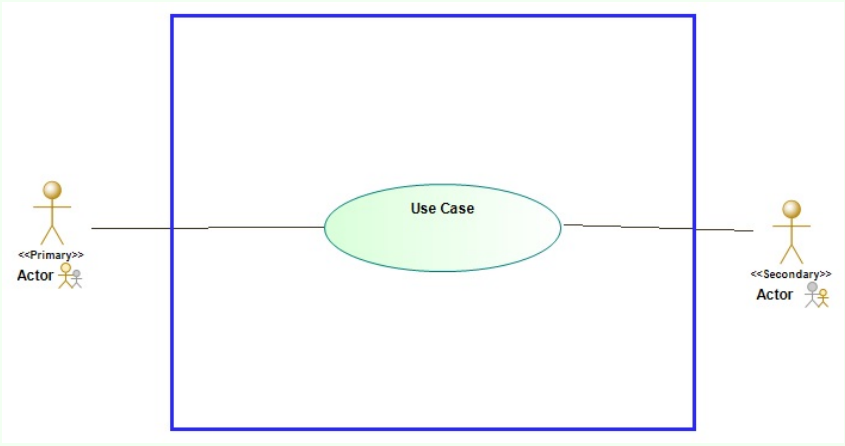


Figure 3.5: Primary and secondary actors

3.1.2 Use cases (UC)

What is it ?

- A **use case** is a specific way of using the system.
- It describes what the future system will have to do, without specifying how it will do it.
- Generally modeled as an **ellipse**
- The name can appear inside an ellipse or below it.
- May be represented by a rectangle with an ellipse pictogram

Figure 3.6: Use case

Identify use cases

- There's no mechanical, totally objective way of identifying use cases
- We need to look at the situation from each actor's point of view and determine :
- How he uses the system,
- In which cases it is used,
- Which functions it needs to access.

For each actor, it is necessary to :

- Look for the different intentions with which he uses the system.
- Determine the services to be provided in the specifications.
- Expected system functions.

Actor-use case relationship

- A line between an actor and a use case means that communication has been established.This is modeled as an association in UML.
- The observed system (subject) is modeled in the use case diagram as a large rectangle containing all the use cases.

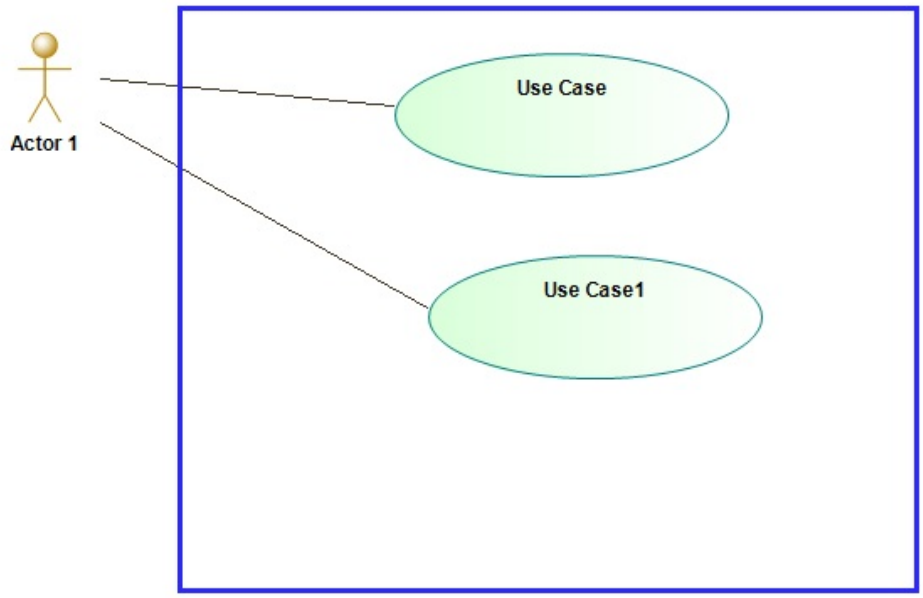


Figure 3.7: Actor-use case relationship

Actor-actor relationship

Only one possible relationship:

- Generalization/specialization

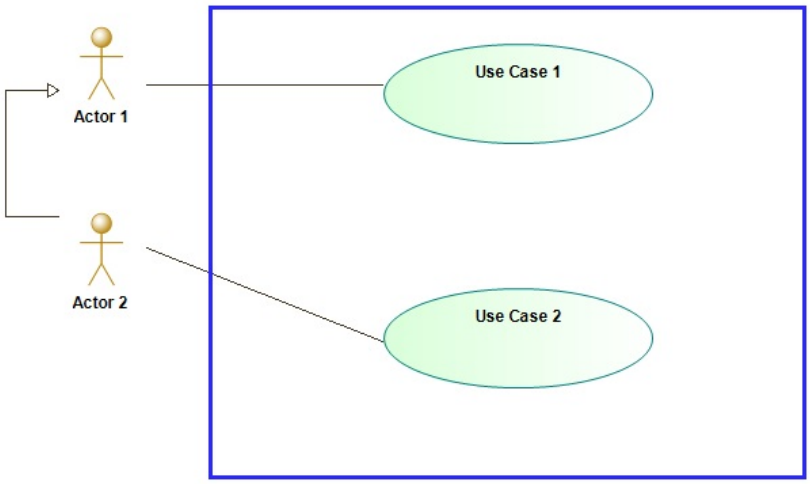


Figure 3.8: Generalization/specialization between actors

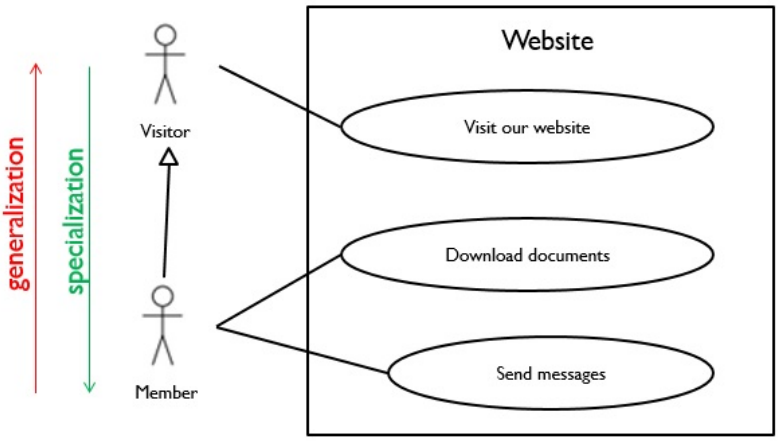


Figure 3.9: Example:Generalization/specialization between actors

Use case-use case relationship

1. **Generalization relationship**

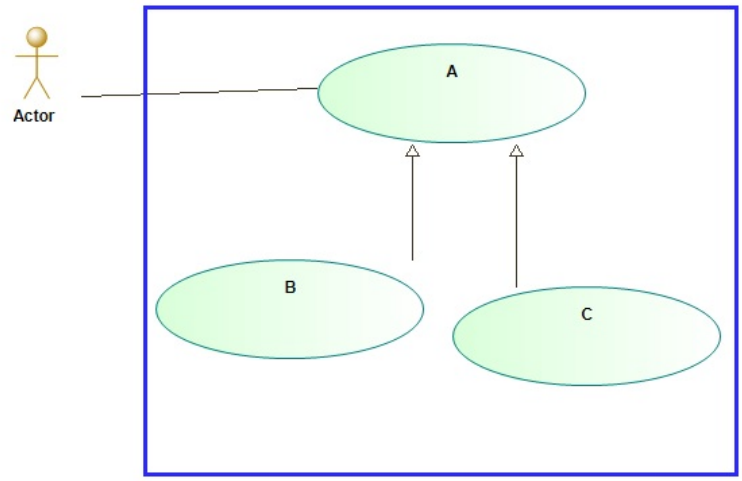


Figure 3.10: Generalization specialization between usec ases

- Use cases B and C are special types of use case A.

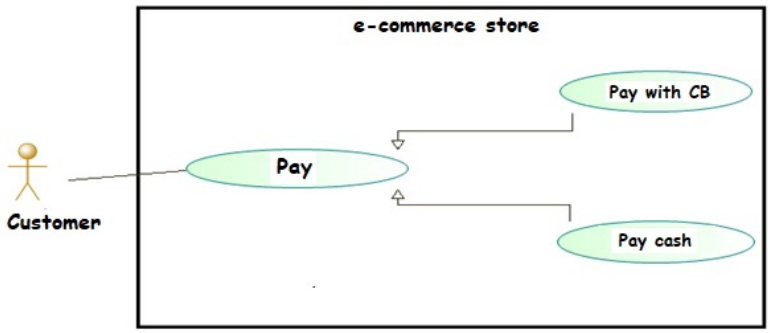


Figure 3.11: Example of Generalization specialization between usec ases

- A credit transfer is a special type of payment.
- A credit transfer is a type of payment.
- The arrow points to the general element.
- This generalization and specialization relationship is present in most UML diagrams, and translates into the concept of inheritance in object-oriented languages.

2. **Inclusion relationship**

- The include relationship allows functionality common to several use cases to be described by one use case (e.g. authenticate).
- The include relationship avoids multiple descriptions of the same behavior.
- When a case is too complex (involving too many actions), we can break it down into simpler cases.
- "include": the execution of one use case requires the execution of another use case.

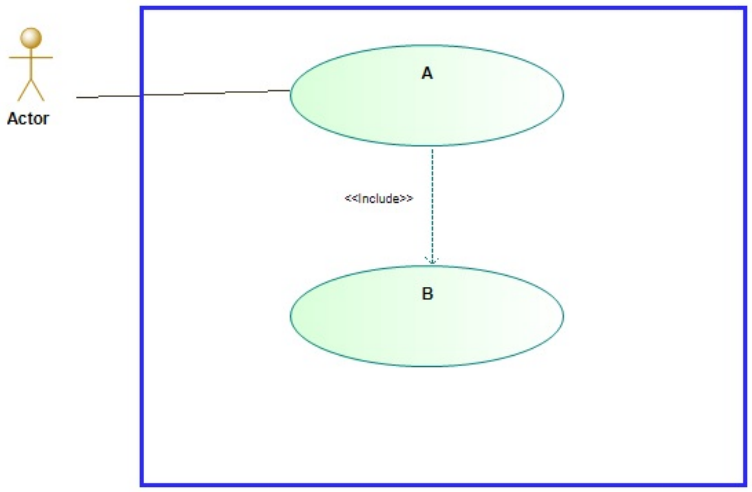


Figure 3.12: Inclusion relationship

- The execution of use case A requires the execution of use case B.

3. **Extension relationship**

- Use case requires the execution of another use case.

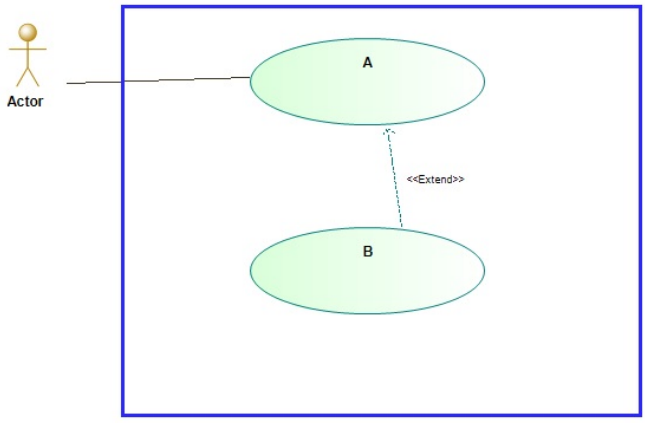


Figure 3.13: Extension relationship

- "extend": This relationship is mainly used to separate optional behavior (variants) from mandatory behavior.
- Use case A is completed by use case B.
- Use case A describes the basic functionality, use case B specifies the extensions.
- Use case A can be run alone or with extensions.

Warning

- **Staying legible**
- No more than 6 or 8 use cases per diagram.
- If necessary, make several diagrams (if disjoint cases between actors).
- For more details, use a text description.

3.1.3 Textual description of use cases

What is it ?

- Textual description of a use case
- Not standardized by UML, but strongly recommended
- Description fields (name, main actor, preconditions, etc.)
- Clear and informal

The form of a use case description sheet :

1. Section 1: Identification
2. Section 2: Description of scenarios
3. Section 3: End and post-conditioning
4. Section 4: complement

Example of a description of use cases (Withdraw money)

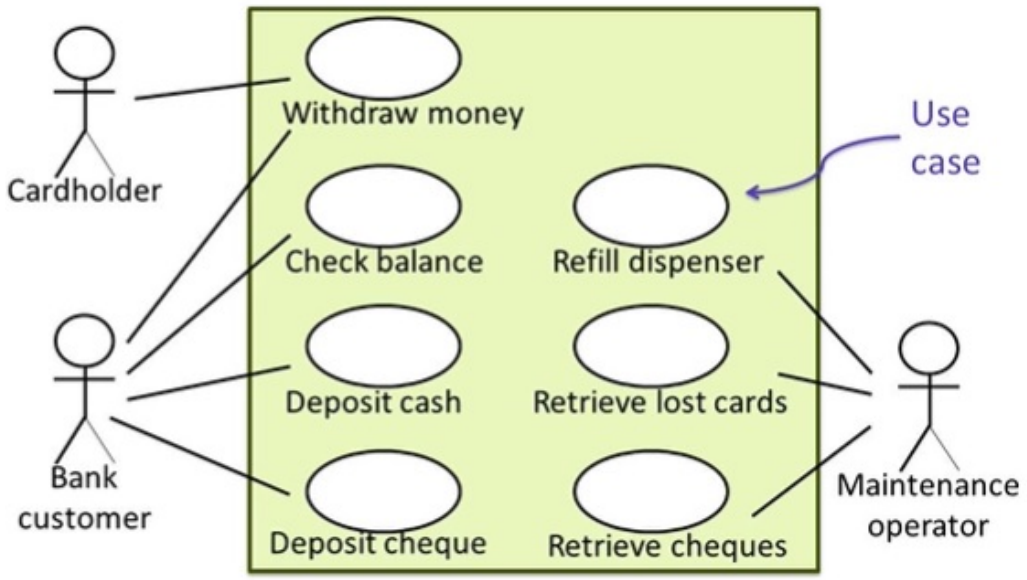


Figure 3.14: Use case Withdraw money

1. **Section 1: Identification Case n○1**
   - Title:Withdrawing money
   - Summary:This use case allows a cardholder who is not a bank customer to withdraw money, if his weekly credit allows it.
   - Actors: Cardholder (primary).
   - Creation date: 02/03/11 Update date: 05/05/11
2. **Preconditions**
   - The ATM cash register is full (at least one bill left!).
   - No card is already stuck in the reader.
   - The connection to the authorization system is operational.
3. **Section 2: Description of scenarios**
   (a) **Nominal scenario**
   - 8. The ATM asks the Cardholder to enter the desired withdrawal amount.
   - 9. The Cardholder enters the desired withdrawal amount.
   - 10. The ATM checks the amount requested against the weekly balance.
   - 11. The ATM asks the Cardholder if he wants a ticket.
   - 12. The Cardholder requests a ticket.
   - 13. The ATM returns the card to the Cardholder.
   - 14. The Cardholder takes back his card.
   - 15.The ATM issues tickets and a receipt.
   - 16.The Cardholder takes the tickets.

   (b) **Alternative sequences**
   - A2: amount requested greater than weekly balance
   - The A2 sequence starts at point 10 of the nominal scenario.
   - 11.The ATM informs the Cardholder that the amount requested exceeds the weekly balance.
   - The nominal scenario is repeated in point 8.

   (c) **A3: ticket refused**
   - The A3 sequence starts at point 11 of the nominal scenario.
   - 12.The Cardholder refuses the ticket.

- 13.The ATM returns the card to the Cardholder.
- 14.The Cardholder takes back his card.
- 15.The ATM delivers the banknotes.
- 16.The Cardholder takes the tickets.

(d) **Error sequences**

- E4 : card not included
- Sequence E4 starts at point 13 of the nominal scenario.
- 14. After 10 seconds, the ATM confiscates the card.
- 15. The Authorization System is informed; the use case ends in failure.

(e) **E5 : tickets not taken**

- Sequence E5 starts at point 15 of the nominal scenario.
- 16. After 10 seconds, the ATM takes back the banknotes.
- 17. The use case ends in failure.

4. **Section 3: End and post-conditions**

- The ATM cash register contains fewer banknotes than at the start of the use case (the number of missing banknotes depends on the withdrawal amount).
- A withdrawal transaction has been recorded by the ATM with all relevant information (amount, card number, date, etc.). Transaction details must be recorded for both successful and unsuccessful transactions.

3.2 Exercices and solutions

In a school, you want to manage the reservation of classrooms and teaching equipment (laptop and/or video projector). Only teachers are authorized to make reservations (subject to room and equipment availability). The room schedule can be consulted by anyone (teachers, students). On the other hand, the timetable summary by teacher (calculated from the room schedule) can only be consulted by teachers. Finally, for each course there is a teacher in charge, who is the only person who can edit the timetable for the entire course.

- Draw the use case diagram

Solutions

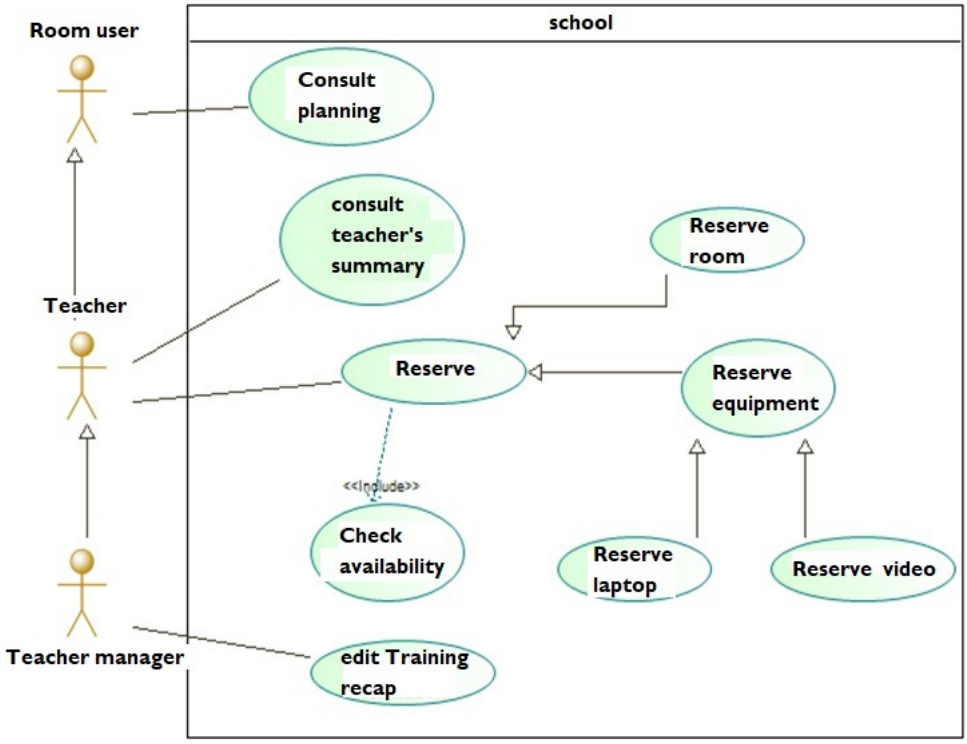


Figure 3.15: School Management System

3.3 Sequence diagram

What is it ?

- Sequence diagrams describe **HOW** system elements interact with each other and with the actors involved.
- Show interactions between objects from a temporal point of view
- Description of typical scenarios and exceptions

3.3.1 Types of sequence diagrams

- **System sequence diagram (external view)**
  - Use case details
  - Application: requirements specification
- **Object sequence diagram (inside view)**
  - Procedures and functions
  - Application: design

3.3.2 Elements of sequence diagrams

1. **Participants :**
   - Actor(s)
   - System(s)
2. **Lifelines**
   - Time (vertical dotted line)
   - Actors
   - Activation bar
3. **Messages**
   - Communication between lifelines

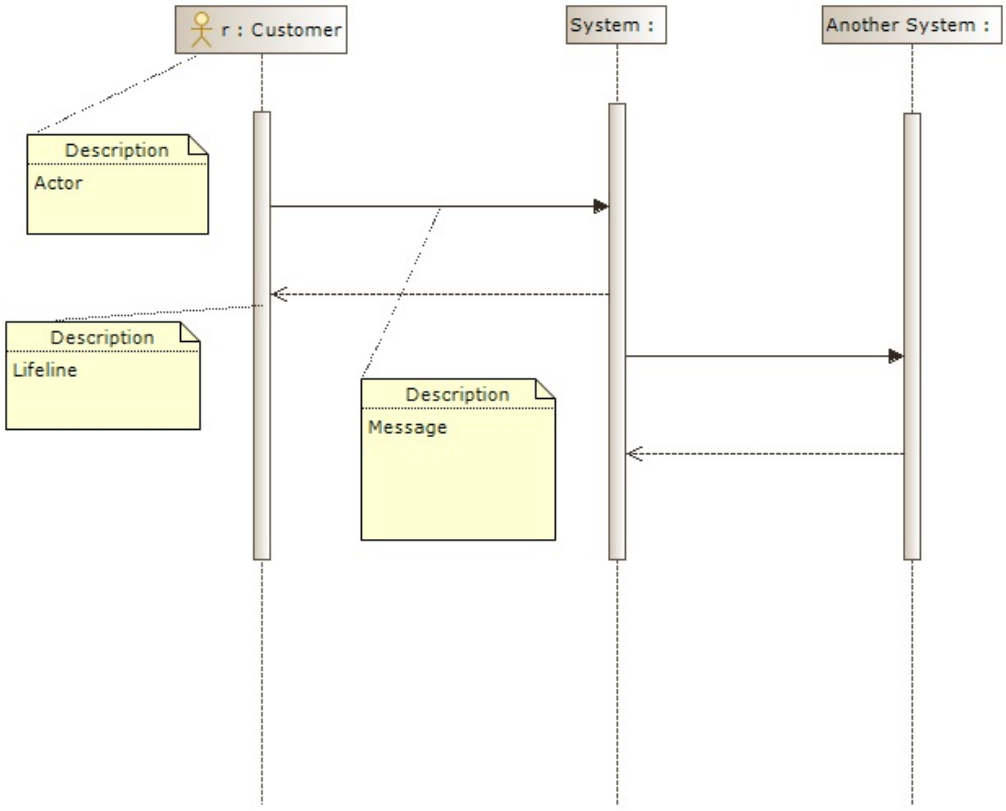


Figure 3.16: Elements of sequence diagrams

3.3.3 Message types

- **Synchronous message:** sender blocked while receiver processes message (call). Typically: method call (If object A invokes a method of object B,A remains blocked until B has finished.

Figure 3.17: Synchronous message

- **Asynchronous message:** non-blocking. The message sent can be taken into account by the receiver at any time or ignored.

Figure 3.18: Asynchronous message

- **Return message:** Method call messages can be associated with a return message (dotted line) indicating that control has been taken over by the object that sent the synchronous message.

Figure 3.19: Return message

- **Reflexive message:** an object can send messages to itself call to another object method recursive call
- **Message found**
  - message of unknown origin
  - outside the framework described by the Sequence Diagram
- **Lost message**
  - message sent, but never received

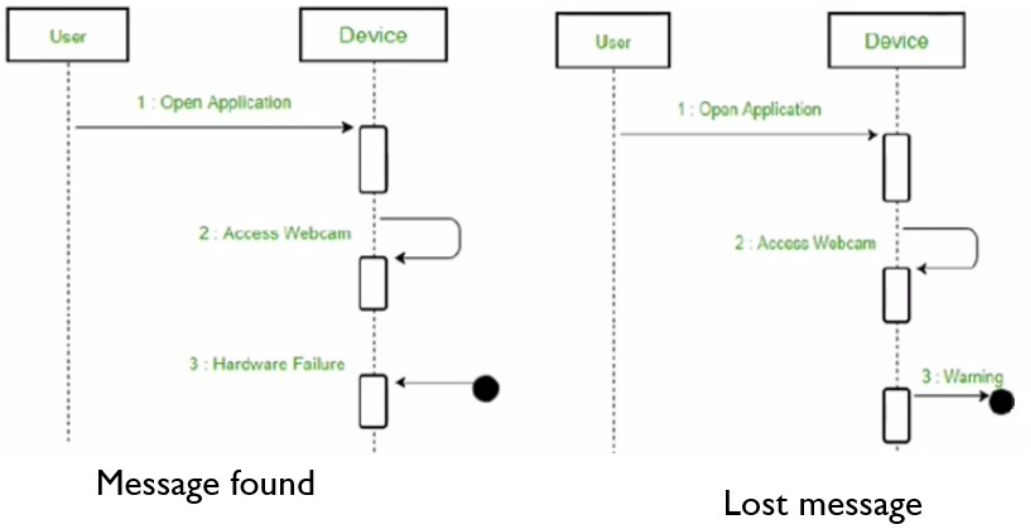


Figure 3.20: Lost message

These two types of messages are rarely used in the sequence diagram

3.3.4 Case study (ATM)

Sequence diagram «withdraw money » Scenario valid code

1. The customer enters his bank card
2. The machine checks the validity of the card and requests the code from the customer
3. If the code is correct, it sends a direct debit authorization request to the bank group.The latter returns the authorized balance to be debited.
4. The distributor proposes several amounts to be debited
5. The customer enters the amount to be withdrawn
6. After checking the amount against the authorized balance, the dispenser asks the customer if he or she would like a ticket.
7. Following the customer's response, the card is ejected and recovered by the customer.
8. Tickets are then issued, along with the ticket
9. The customer finally collects the tickets.

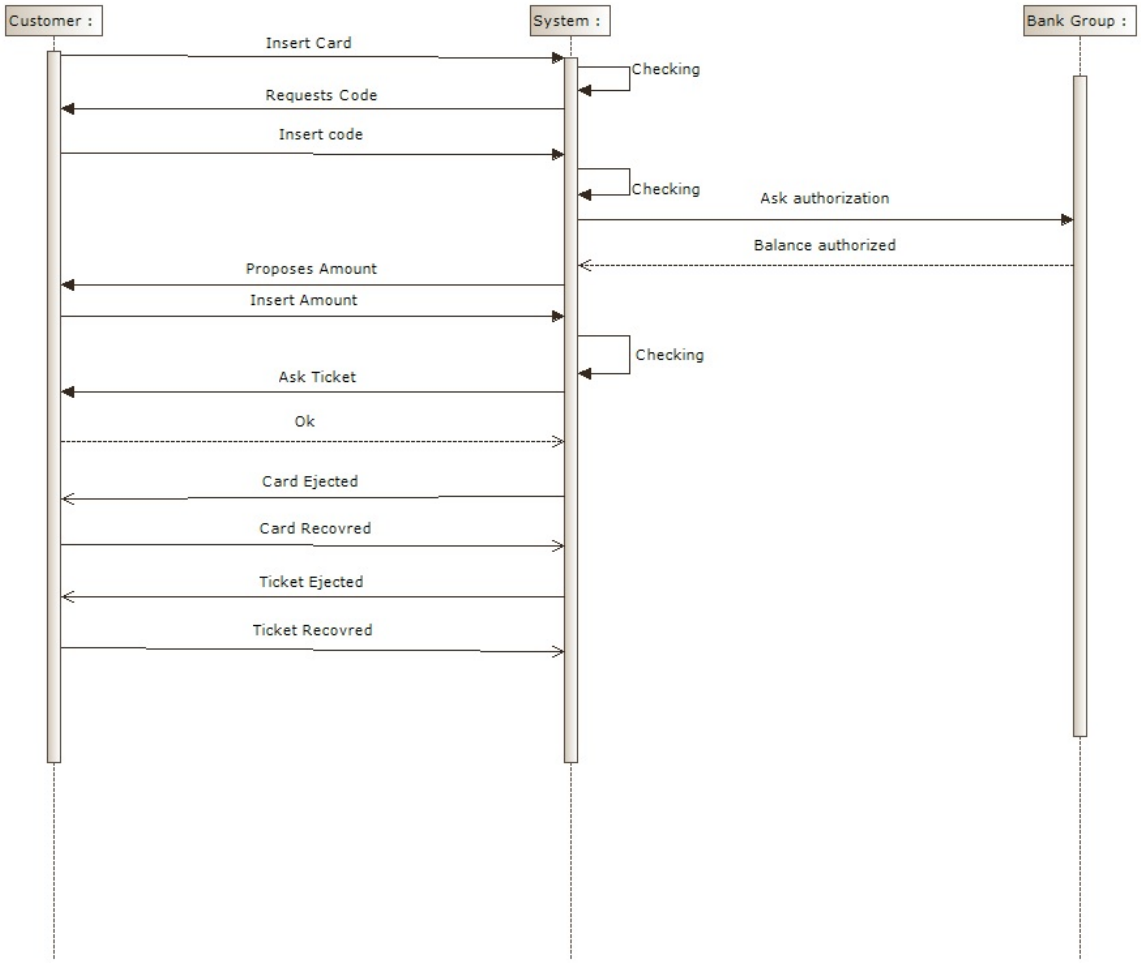


Figure 3.21: -ATM- Case Study

3.3.5 Combined fragments

- A combined fragment breaks down a complex interaction into sufficiently simple fragments to be understood
- Sequence fragment
- Interaction framework
- UML notation
  - Rectangle that groups sub-sections of the sequence diagram
  - Fragment operator in top left corner

Figure 3.22: Fragment Operator

Fragment operators

- **Operator "Opt":** Fragment covered if a condition is verified

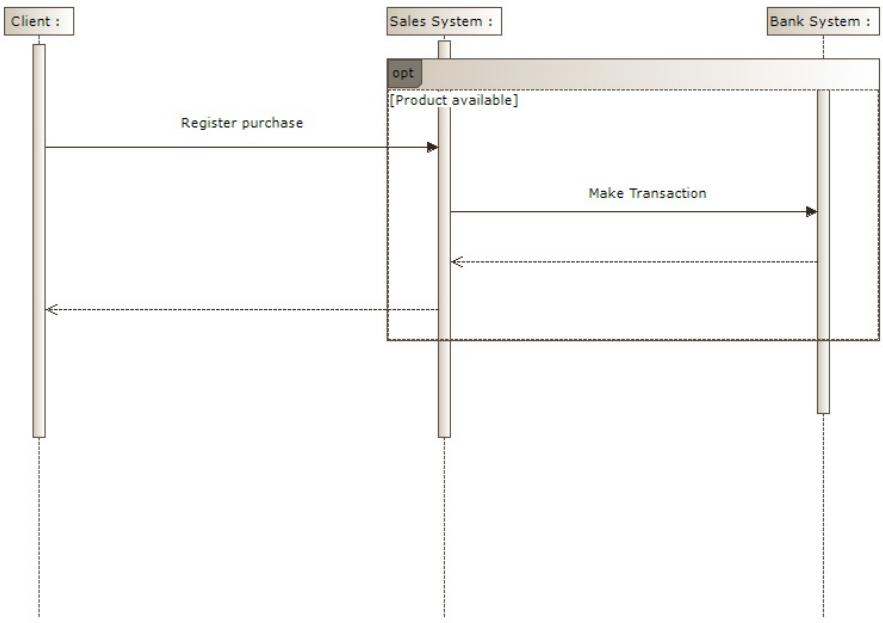


Figure 3.23: Opt Operator

- **Operator "Alt":** Equivalent to the control structure "if .. then ..else

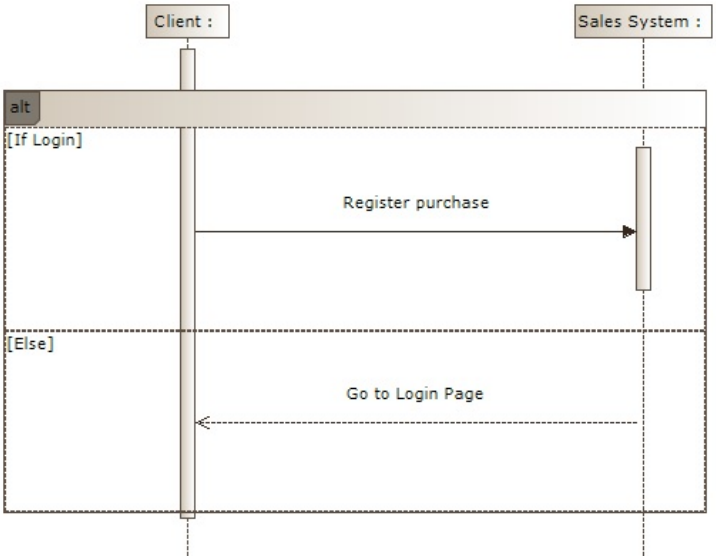


Figure 3.24: Alt Operator

- **Operator "Loop":** Fragment repeats as long as condition is met Loop (initial value, maximum, condition).

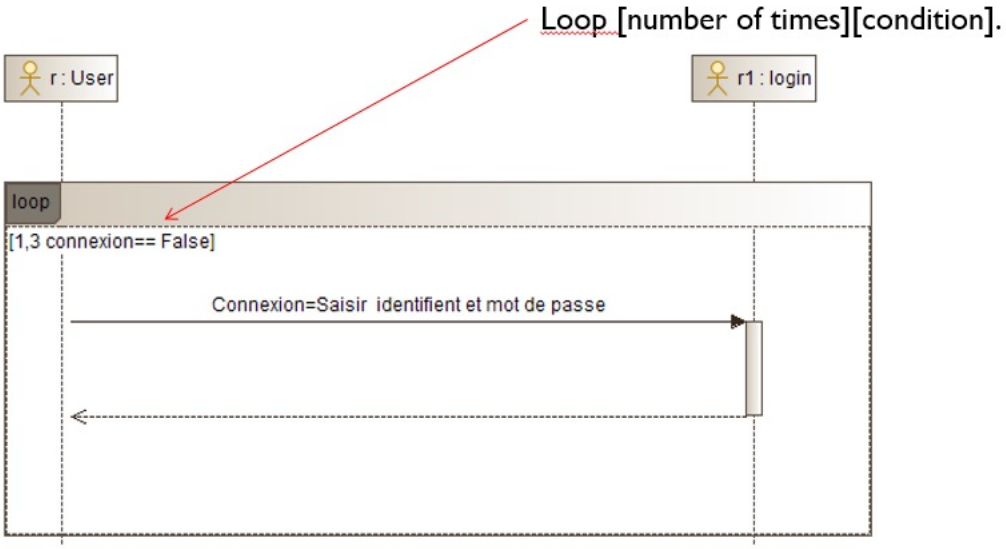


Figure 3.25: Loop Operator

- **Operator "par":** operations within the fragment run in parallel

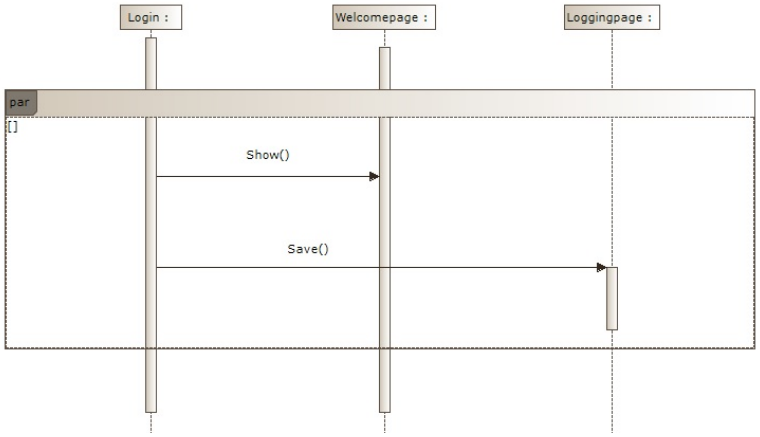


Figure 3.26: Loop Operator

- **Operator "Ref":** Reference to another sequence diagram. It's useful for including several scenarios in the same sequence diagram. Ex:Valid scenario code / Invalid scenario code.

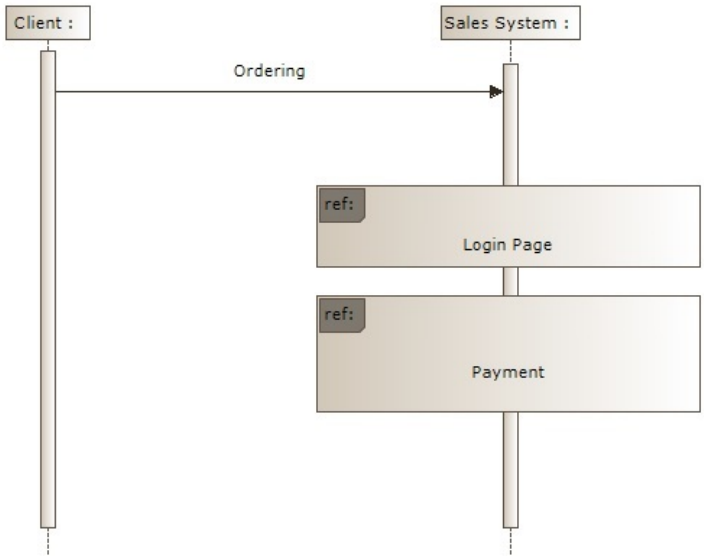


Figure 3.27: Ref Operator

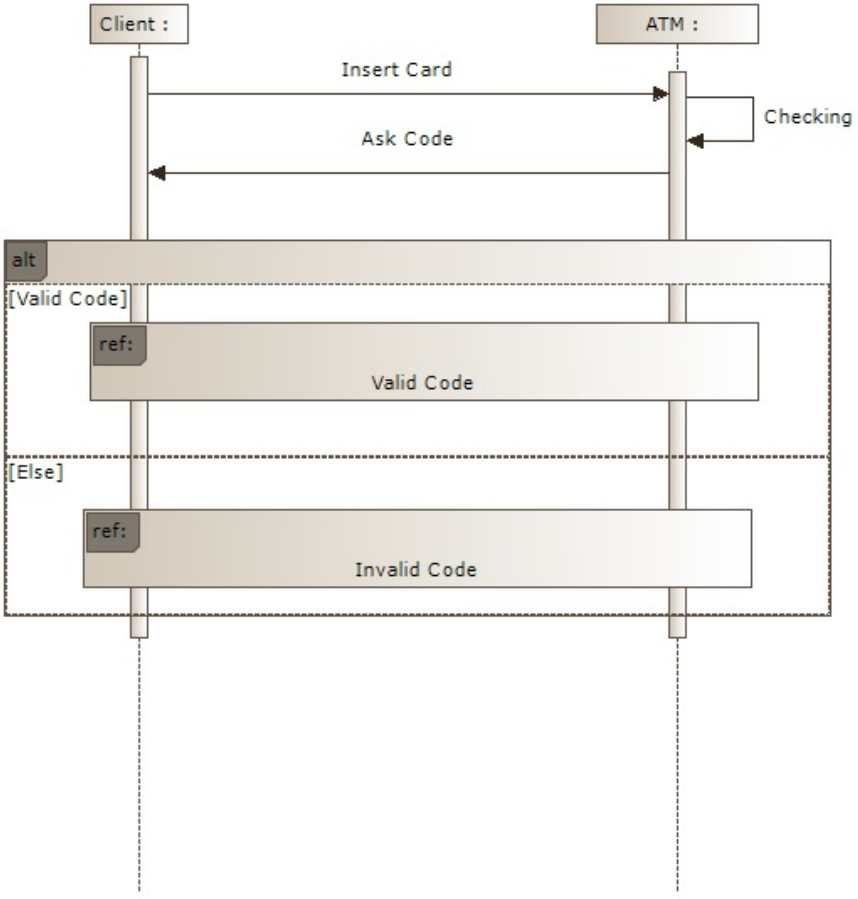


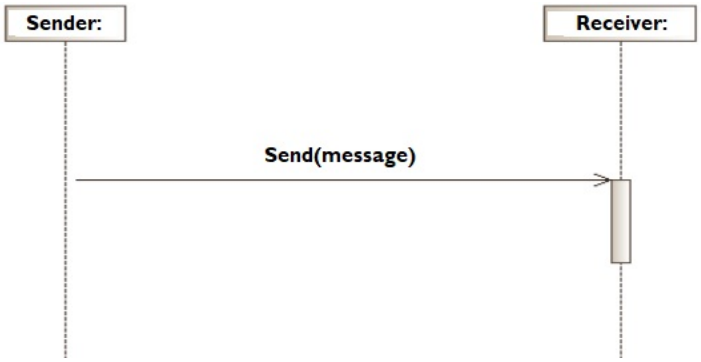
Figure 3.28: Example Ref Operator

3.4 Exercices and solutions

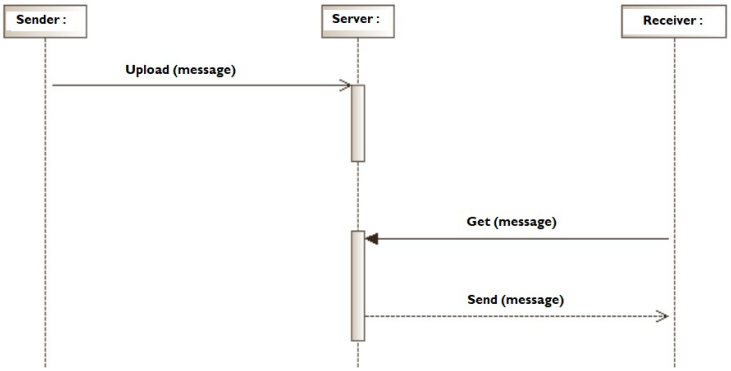
- When an e-mail is sent by the sender, the sender doesn't want to wait for the recipient to receive it, and there's no intermediary.
- A mail server acts as an intermediary between the sender and receiver of an e-mail. The server is always on. Can synchronous messages be used to send and retrieve e-mails?

Solutions

- When an e-mail is sent by the sender, the sender doesn't want to wait for the recipient to receive it, and there's no intermediary.



- A mail server acts as an intermediary between the sender and receiver of an e-mail. The server is always on. Can synchronous messages be used to send and retrieve e-mails?



3.5 Class diagram

What is it ?

- Representation of the system as a set of interacting objects
- Representation of the system's internal structure and logic
- Inspired by real-world objects
- Abstraction and decomposition of the system into objects

3.5.1 Class

What is it ?

- Objects with the same type (Figure 3.29)
- Characteristics: Attributes (information, properties, etc.)
- Behavior: Operations (methods, messages, etc.)
- Each object is an instance of a class.

3.5.2 Object

Compared to a class, an object has:

- **An identity**
  - Two different objects have different identities
  - The object can be designated (referred to)
- **A state (attributes)**
  - Set of properties/characteristics defined by values
  - To personalize/differentiate from other objects
  - May change over time
- **Behavior (methods)**
  - Set of processes that an object can perform (or be made to perform)

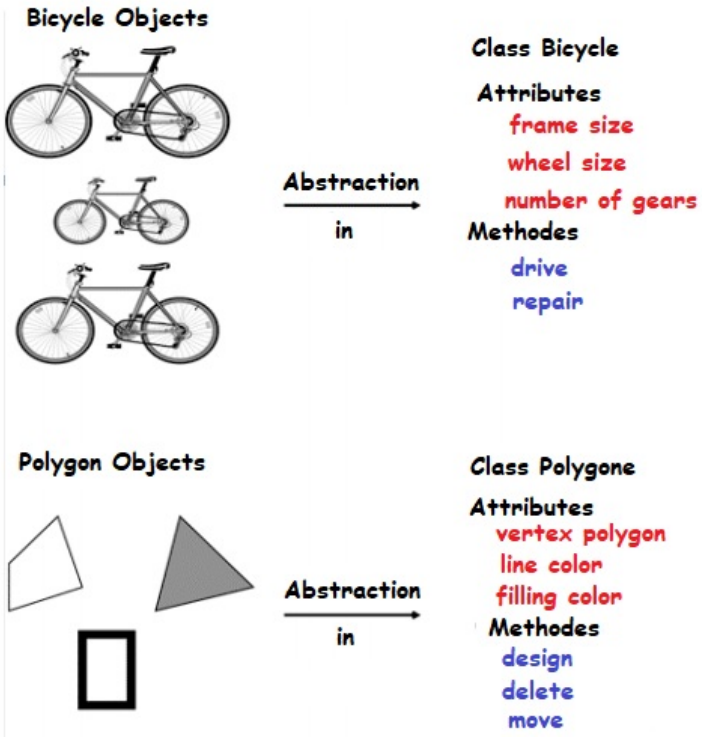


Figure 3.29: Object and class

Object= Identity + State + Behavior

Class representation

- 3-compartment rectangle
- Noun (singular, capital)
- Attributes
- Operations

More or less detail as required (Figure 3.30)

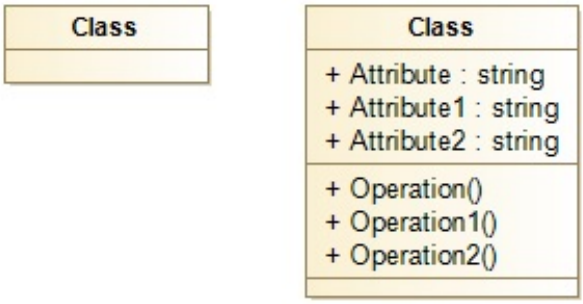


Figure 3.30: Class representation

Object representation (Figure 3.31)

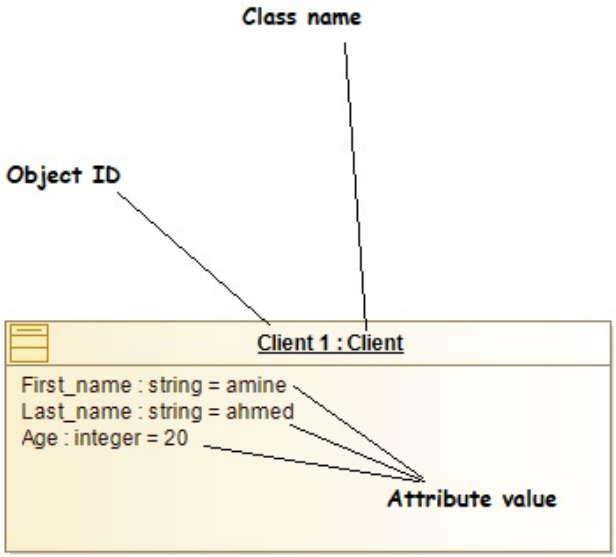


Figure 3.31: Object representation

3.6 Relationship between classes

3.6.1 Inheritance(Generalization/Specialization)

- Inheritance is a specialization generalization relationship (Figure 3.32).

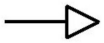


Figure 3.32: Inheritance Symbol

- Specialized elements inherit the structure and behavior of more general elements (attributes and operations).
- Principle of substitution: all properties of the parent class must be valid for the child classes.

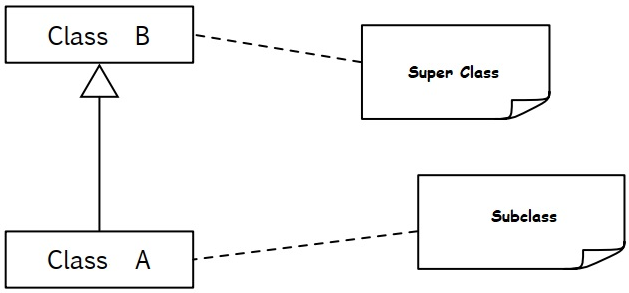


Figure 3.33: Example of Inheritance

- A is a B" or "A is a kind of B" principle: all instances of the subclass are also instances of the superclass. For example, any operation that accepts an object of class Animal must accept any object of class Cat (the reverse is not always true).

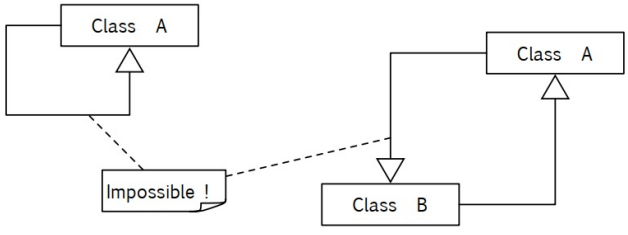
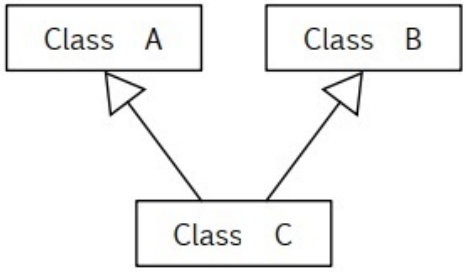


Figure 3.34: Reflexive ezlatioship

- No-reflexive, no-symmetrical relationship! (Figure 3.34)

Multiple inheritance

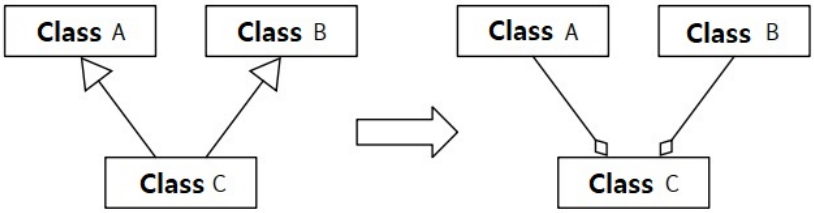
- A class can have several parent classes.This is known as multiple inheritance.



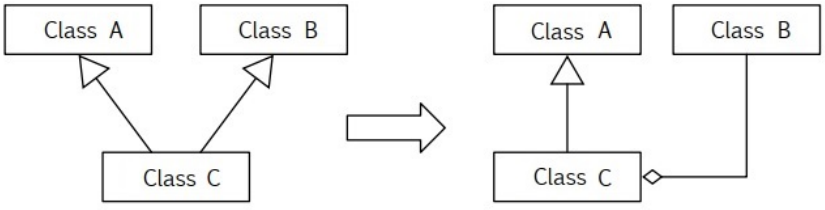
- Incompatible with java
- The C++ language is one of the object languages that enable its effective implementation.

How to avoid Multiple Inheritance?

- First solution: delegate



- Second solution: inherit the most important class and delegate the others



3.6.2 Association

What is it ?

- Bidirectional semantic connection between classes (Figure 3.35)
- Representation of associations :

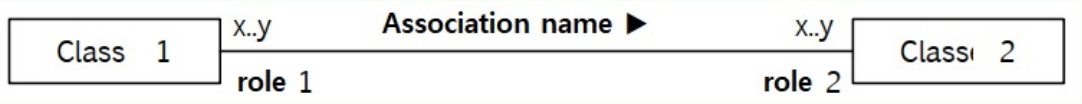


Figure 3.35: Association

- Name: verbal form, with a reading direction Roles: nominal form, describes one end of the association
- Multiplicity: 1, 0..1, 0..*, 1..*, n..m

- Multiplicity

1	One
0..1	zero or one
N	N (integer number)
M .. N	From M to N (integer numbers)
*	From zero to many
0 .. *	From zero to many
1 .. *	From one to many

- Association navigability (Figure 3.36)

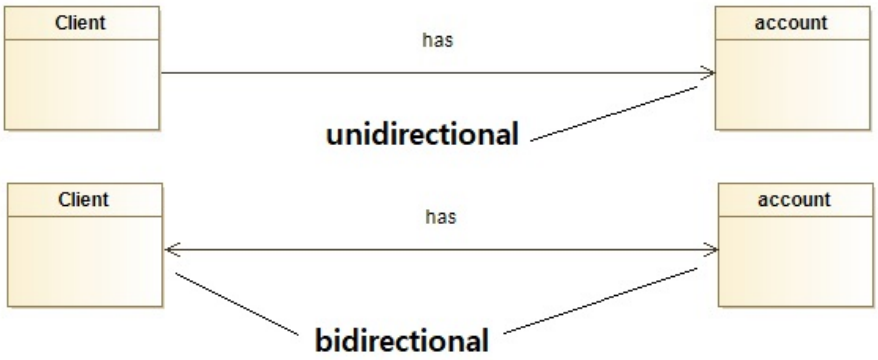


Figure 3.36: Association navigability

- Association Naming associations (Figure 3.37)

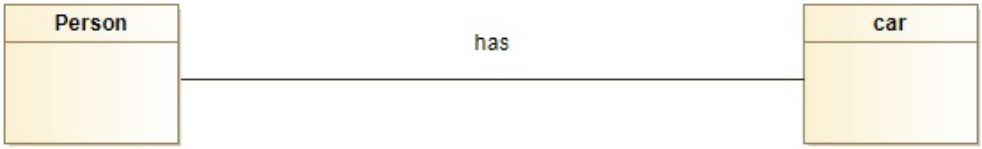


Figure 3.37: Association Naming associations

- We can add a reading direction (Figure 3.38)

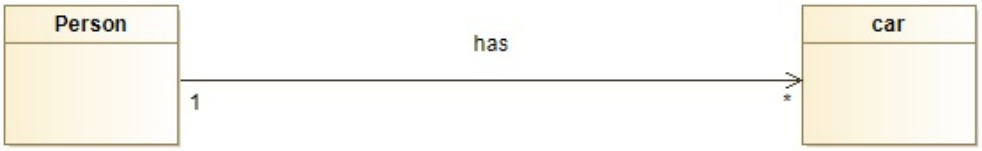
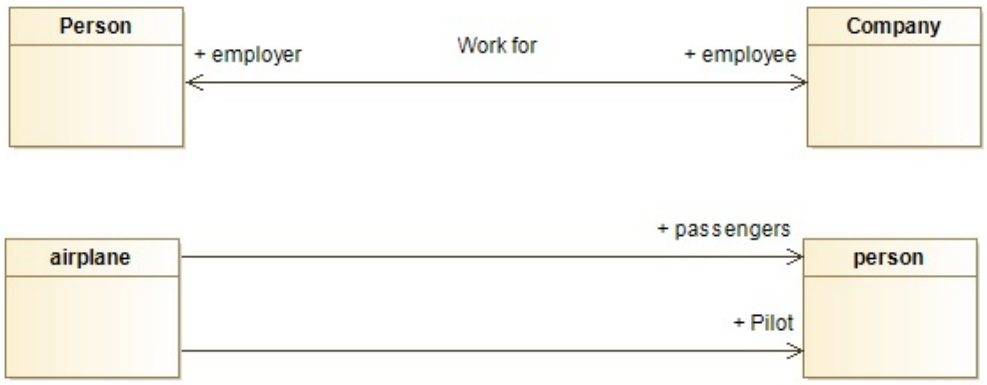
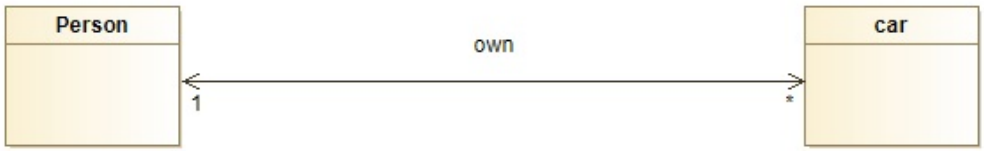


Figure 3.38: Association a reading direction

- Association end name: Role
  - Each end of the association can be named

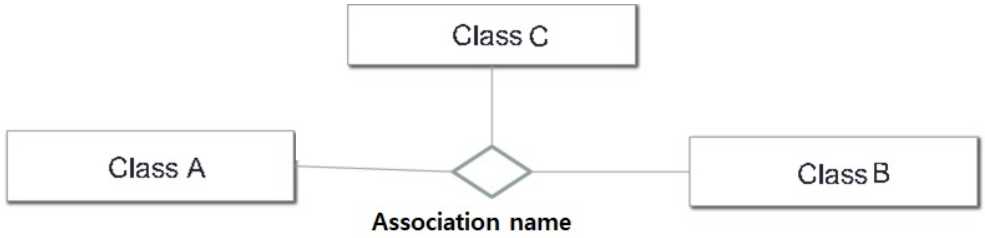


- Multiple associations
  - Multiplicity specifies the number of instances of a class that can be linked to a single instance of an associated class. It constrains the number of linked objects.



  - Example: a person can own several cars (between zero and any number); a car is owned by a single person.

- Association n-aire
  - In general, associations are binary
  - N-ary: at least three classes involved
  - Use only when no other solution is possible!



- Reflective association
  - Linking objects of the same class

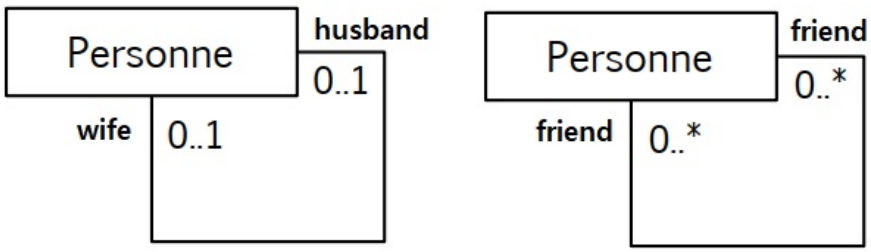
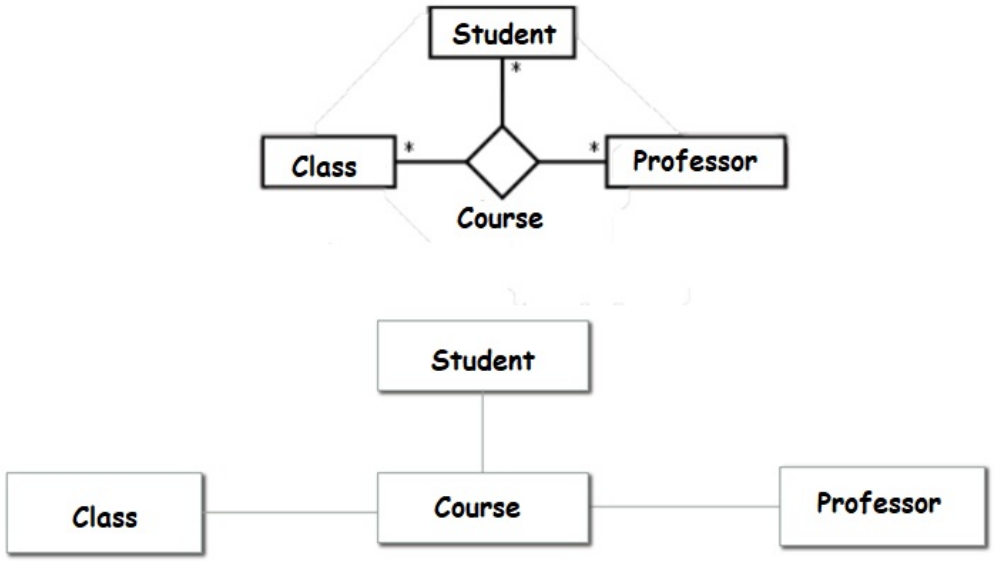


Figure 3.39: Reflective association

- Allocated association
  - Includes an association class
  - Information (attributes, methods) specific to the association.



3.6.3 Aggregation

What is it ?

- An aggregation is a special case of a non- symmetrical association expressing a content relationship of an element in a set.
- Aggregations don't need to be named: implicitly they mean "contains", "is composed of".
- Aggregation is represented by the addition of an empty diamond on the side of the aggregate (the set) (Figure 3.40).

UML representation

Figure 3.40: Aggregation

Example:

3.6.4 Composition

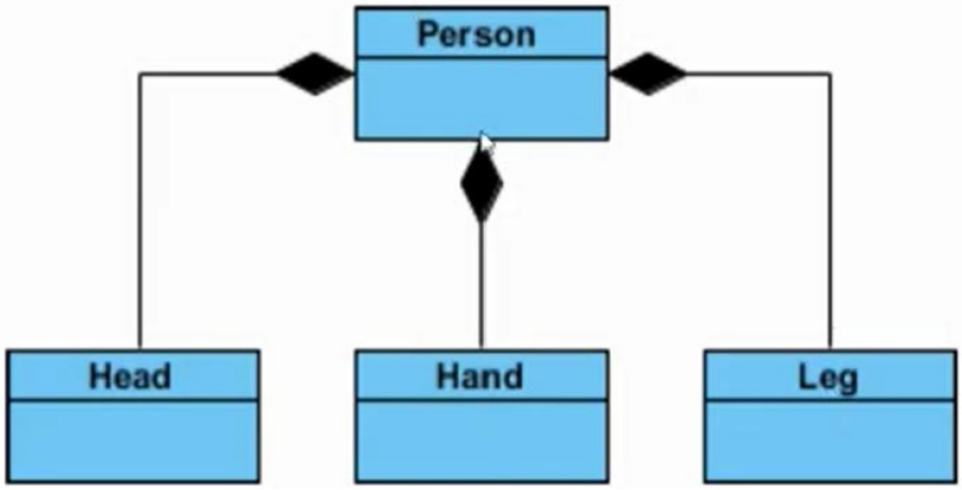
What is it ?

- A composition is a stronger aggregation implying that:
- An element can only belong to a single composite aggregate (non-shared aggregation);
- The destruction of the composite aggregate (the whole) leads to the destruction of all its elements (the parts) (Figure 3.41)

UML representation

Figure 3.41: Composition

Example:



3.6.5 When to use a composition rather than an aggregation ?.

Aggregation vs. composition

- To decide whether to use a composition rather than an aggregation, you need to ask yourself the following questions:
- Does the destruction of the composite object (the whole) necessarily imply the destruction of the component objects (the parts)? This is the case if the components have no autonomy vis-à-vis the composites.
- When we copy the composite, do we also have to copy the components, or can we "reuse" them, in which case a component can be part of several composites?
- If the answer to these two questions is "**yes**", a composition must be used.

3.6.6 Abstract class

What is it ?

- A method is said to be abstract when we know its header (signature), but not how it can be implemented. It's up to child classes to define abstract methods.
- A class is said to be abstract when it defines at least one abstract method, or when a parent class contains an abstract method not yet implemented.

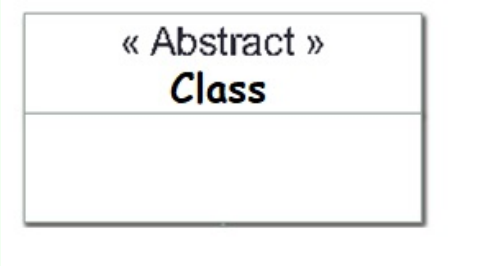


Figure 3.42: Abstract class

3.6.7 Interfaces

What is it ?

- Not a class
- Abstract method list
- No attribute except "final static" constant
- Defines a service and cannot be used to create objects
- Implemented by at least one concrete class.
- An interface specifies a set of operations (behavior). It's a contract
- Linked classes agree to respect the contract
- They must implement interface operations.

Interface relationship

- Interface implementation
- Definitions of all abstract methods
- A class can implement several interfaces
- An interface can be implemented by several classes

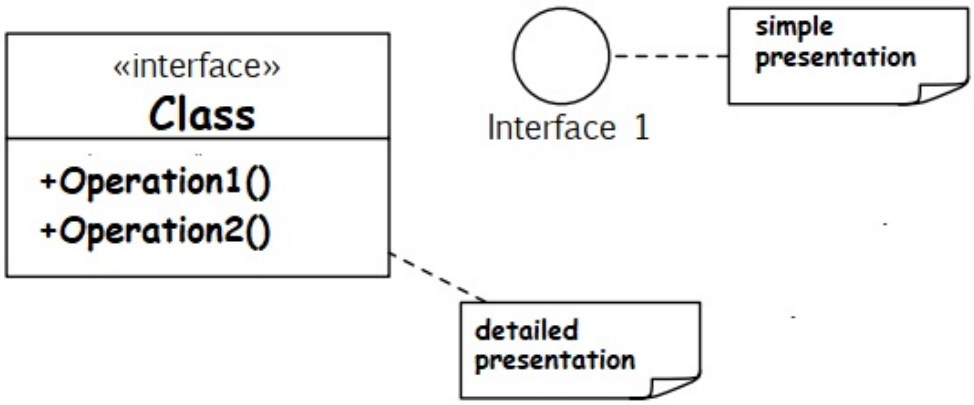


Figure 3.43: Interface relationship

UML representation

- Two types of interface relationships:
- "Realize" stereotype
- "Use" stereotype

Figure 3.44: Two types of interface relationship

Class diagram Methodology

- Find the class of the domain studied (Find **Nouns**)
- Find the association between classes (the **verbs** that link the nouns)
- Refine the diagram by eliminating redundant, irrelevant classes and associations.
- Once the classes are well established, look for the inheritance and aggregation relationships
- Add attributes for each class
- Check that you can create use cases by traversing the class diagram

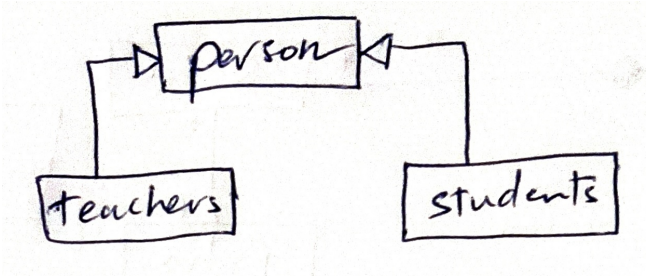
3.7 Exercices

Propose a corresponding class model.

3.7.1 Exercise 1

- Students and teachers are two different kinds of people.

Solution

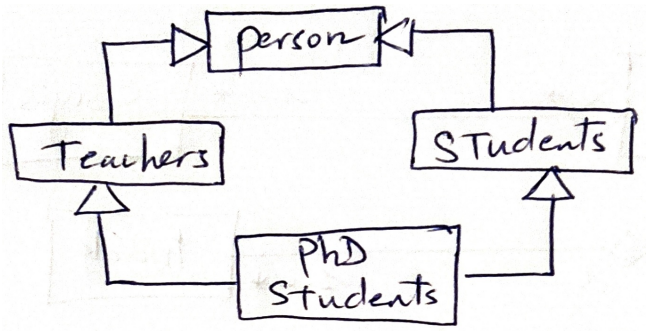


3.7.2 Exercise 2

Propose a corresponding class model.

- A PhD student is a student who teaches. Complete the previous class model.

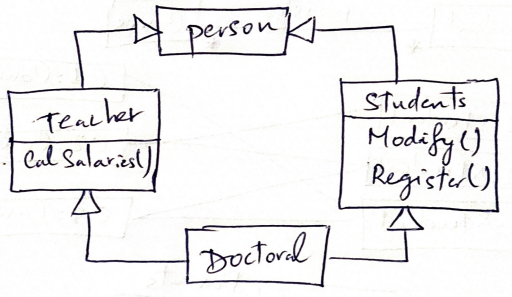
Solution



3.7.3 Exercise 3

Doctoral candidates and students must register at the beginning of the year and modify their registration if necessary. We know everyone's first and last names. We need to be able to calculate the salaries of doctoral students as well as teachers. Add these elements to the previous model

Solution

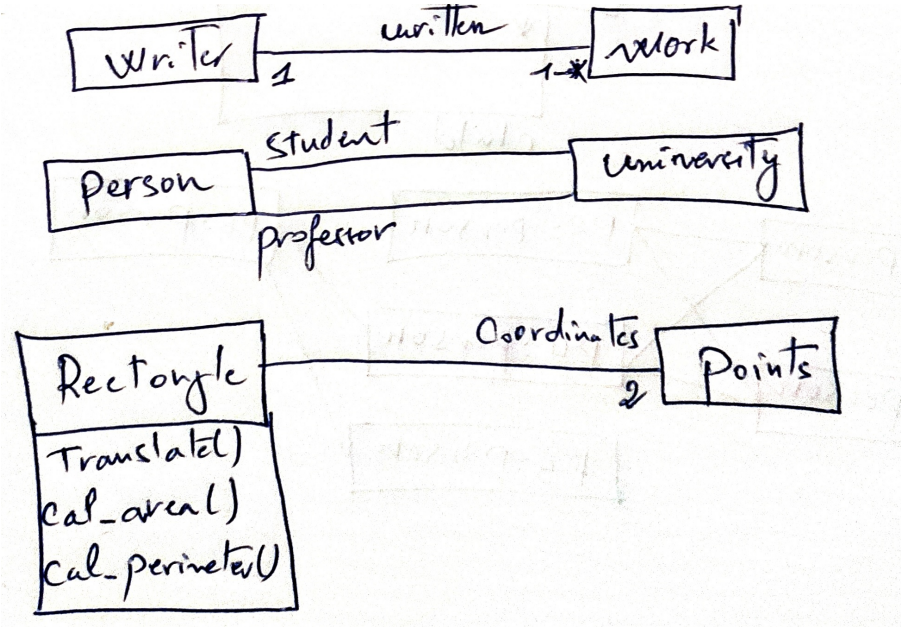


3.7.4 Exercise 4

Propose a corresponding class model.

1. Every writer has written at least one work
2. People can be associated with universities as students as well as professors.
3. A rectangle has two vertices which are points.A rectangle is constructed from the coordinates of two points.You can also calculate its area and perimeter, and translate it.

Solution



3.7.5 Exercise 5

Propose a corresponding class model.

1. Cinemas are made up of several screens.
2. Films are shown in theaters.
3. The corresponding screenings take place at a specific time each.

3.7.6 Exercise 6

Propose a corresponding class model.

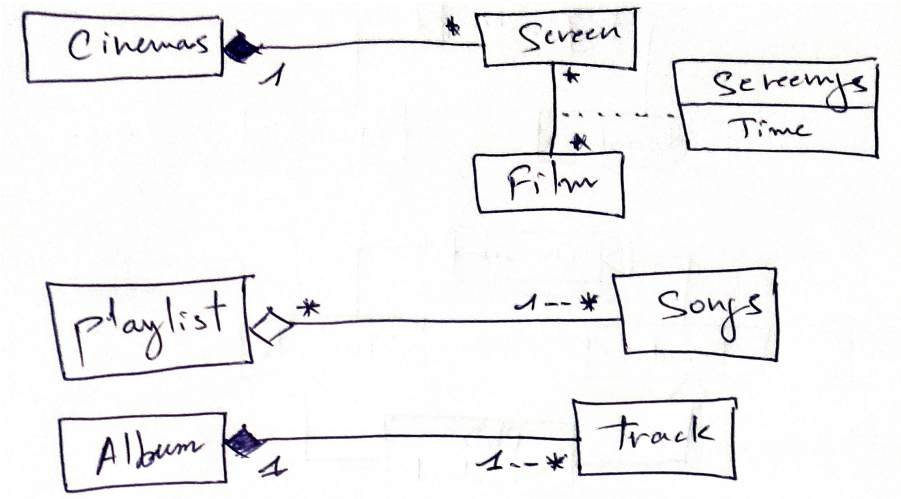
- A playlist is made up of a set of songs.
- A song can belong to several playlists
- Delete list does not delete songs

3.7.7 Exercise 7

Propose a corresponding class model.

- A track only belongs to one album
- Deleting an album deletes all its tracks

Solution



3.8 Object diagram

What is it ?

- It represents objects (i.e. class instances) and their links (i.e. relationship instances)
- It Used to illustrate the class model
- It Used to take a snapshot of the system at a given moment.
- The class diagram models **the rules**, but the object diagram models **facts**.

UML representation

- ID to differentiate the object of the same class
- No operating compartment
- Can be partially defined (attribute not filled in)

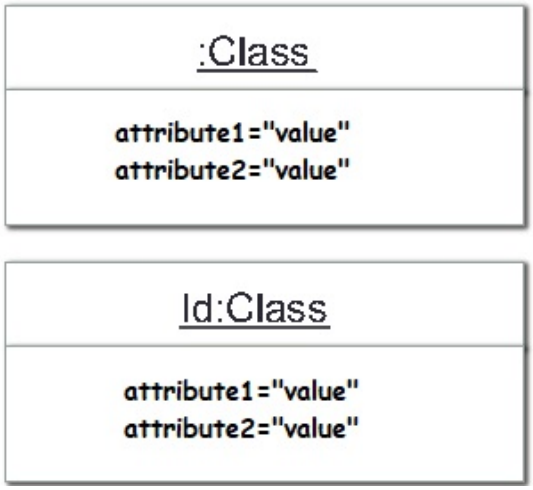
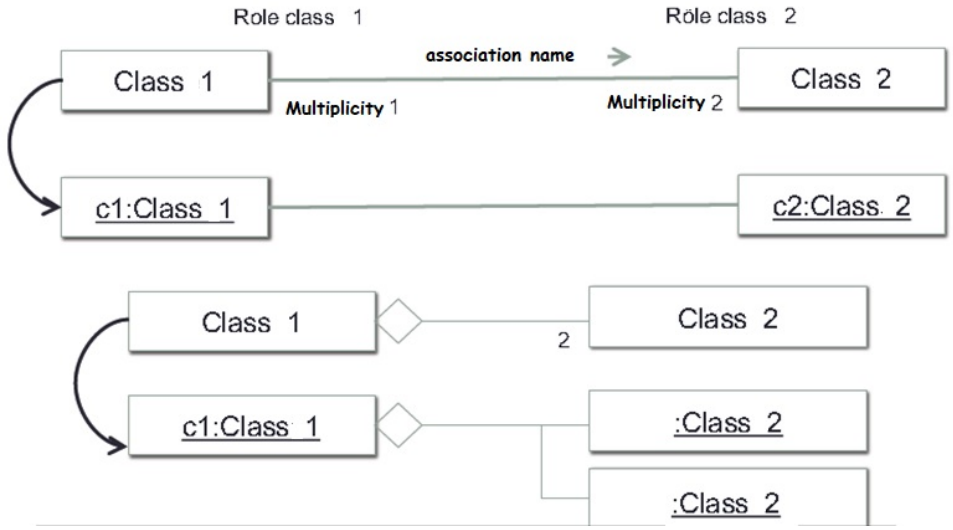


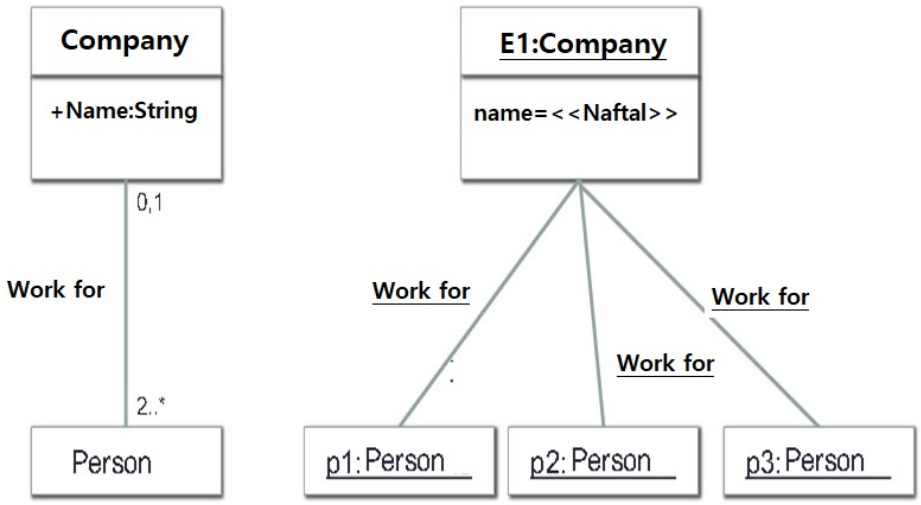
Figure 3.45: Object representation

Link between objects (instance of relationship)

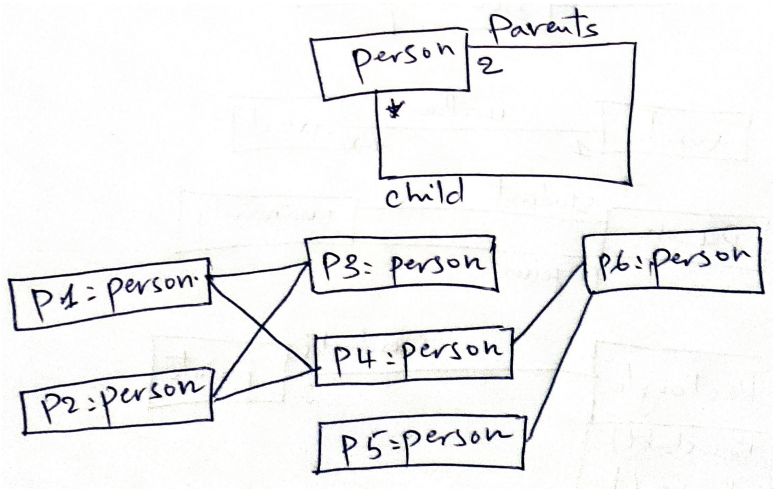
Link between objects

- Compliance with association rules
- Number of possible links between objects depends on the multiplicity of the corresponding class associations
- Link between objects(relationship instance)





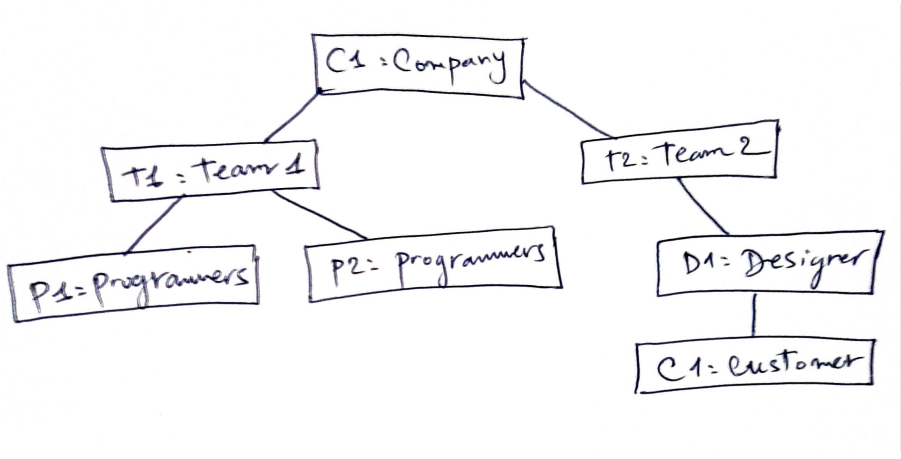
Example of reflexive association:



3.9 Exercices and solutions

- **Give the object diagram corresponding to the following Situation:**
- "The Delta company is made up of two IT teams. Mohamed and Ryad are two programmers working in Team 1. Walid is a designer assigned to team 2, and is responsible for communicating with the customer, Mr AMMAR.

Solutions

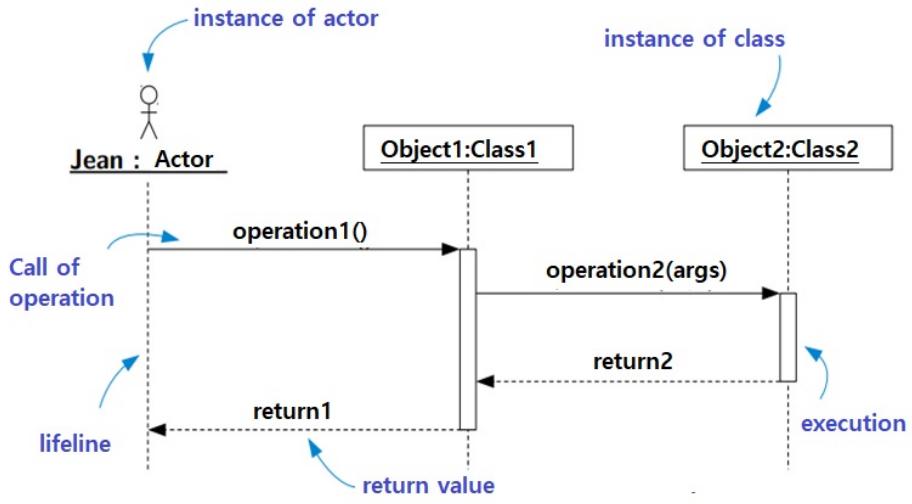


3.10 Object sequence diagram

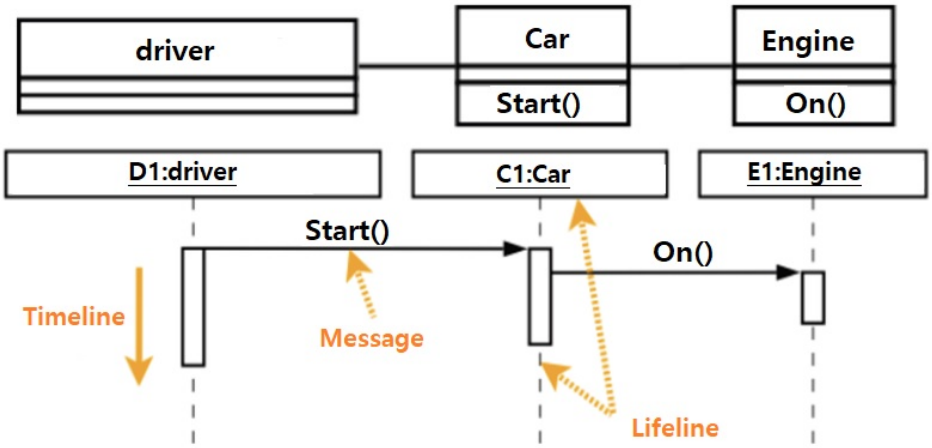
What is it ?

- Represent communications with and within the software
- Temporal representation of interactions between objects .
- Chronology of messages exchanged between objects and with actors.
- In design phase: Describe the realization of use cases on the system described by the class diagram.
- Internal view of system operation
- Instance-level description (state of the system at a given time)
- Description of specific scenarios
- Representation of message exchanges between system objects chronologically

3.10.1 Elements of an object sequence diagram



Example:



Message types

- Object creation

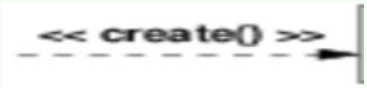


Figure 3.46: Object creation message

- Object deletion

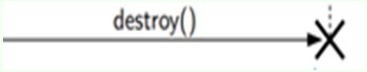


Figure 3.47: Object deletion message

- Synchronous message: Sender blocked waiting for return

Figure 3.48: Synchronous message

- Asynchronous message: Sender not blocked, continues execution

Figure 3.49: Asynchronous message

3.11 State-transition diagram

What is it ?

- Model the internal behavior of an object using a deterministic finite-state automaton
- Corresponds to a single class or object
- Used to model object lifecycles
- Representation of changes in the state of an object, in response to events (interactions with other objects or actors)
- Objective: Describe the dynamic behavior of an entity (software, component, object, etc.). The Behavior described by states and transitions between states
- Describe the changes in state of an object or component, in response to interactions with other objects and components or actors
- Grouping a set of scenarios

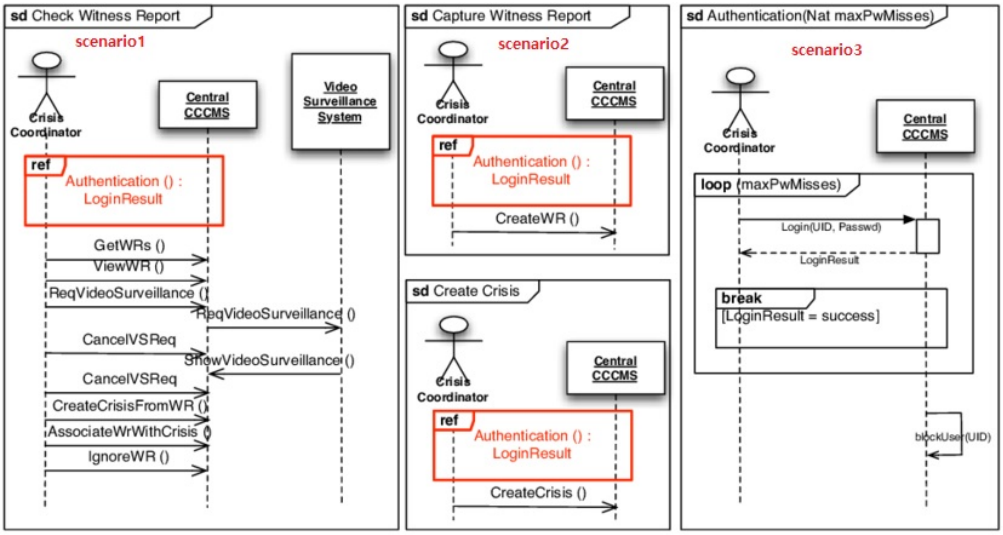


Figure 3.50: Multiple scenarios

Why we use it

Several scenarios in different diagrams.The sequence diagram does not allow you to see all these scenarios in a single sequence diagram even with the use of "Ref" fragments it becomes very complicated. The solution, then, is to use the state diagram, which shows all the different scenarios in a single diagram without it becomes very complicated.

3.11.1 Status

What is it ?

- State: abstraction of a moment in the life of an entity during which it satisfies a set of conditions.
- **Initial state:** System initialization, execution of object constructor

Figure 3.51: Initial state

- **End state:** End of system life, destruction of object

Figure 3.52: End state

- **Intermediate states:** stages in the life of a system or object

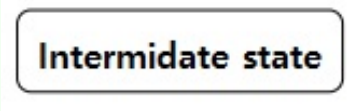


Figure 3.53: Intermediate states

3.11.2 Transition

What is it ?

- A transition represents the instantaneous passage from one state to another
- A transition is triggered by an event:
- The arrival of an event conditions the transition
- A Transition represente the change of state, Ex: case of a lamp
- When the event occurs, if the condition is verified, then action is performed.

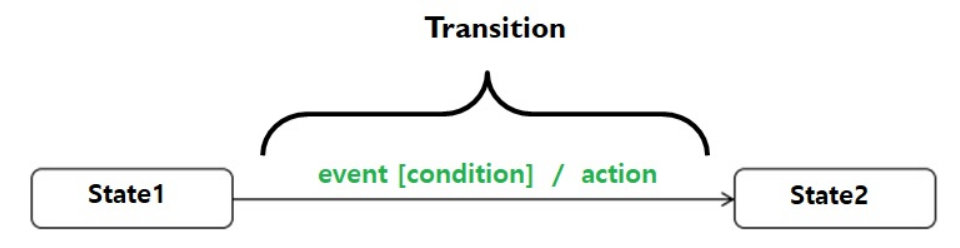


Figure 3.54: Transition

3.11.3 Event

What is it ?

- Event: instantaneous fact coming from outside the system and occurring at a given time.
- Event types :
- **Signal**: reception of an asynchronous message
- **Operation call** (**synchronous**): linked to use cases, class diagram operation, etc.
- Satisfaction of a Boolean condition: **when**(**cond**), continuously evaluated until true

3.11.4 Time event

What is it ?

- Relative date: when(date = date)
- Absolute date: after(duration)

3.11.5 Action

What is it ?

- Action: System reaction to an event
- Characteristics: atomic, instantaneous, non-interruptible
- Examples of actions (syntax left free) :
  - Assignment
  - Sending a signal
  - Operation call
  - Object creation or destruction

3.11.6 Using state-transition diagrams

What is it ?

- **In the analysis phase :**
  - Description of system dynamics as seen from outside
  - Summary of use case scenarios
  - Events = actor action
- **In the design phase :**
  - Description of the dynamics of a particular object
  - Events = operation calls

3.11.7 Composite states

What is it ?

- Composite state: state grouping together a set of states
- Objectives :
  - Prioritize states
  - Structuring complex behaviors
  - Factoring in actions

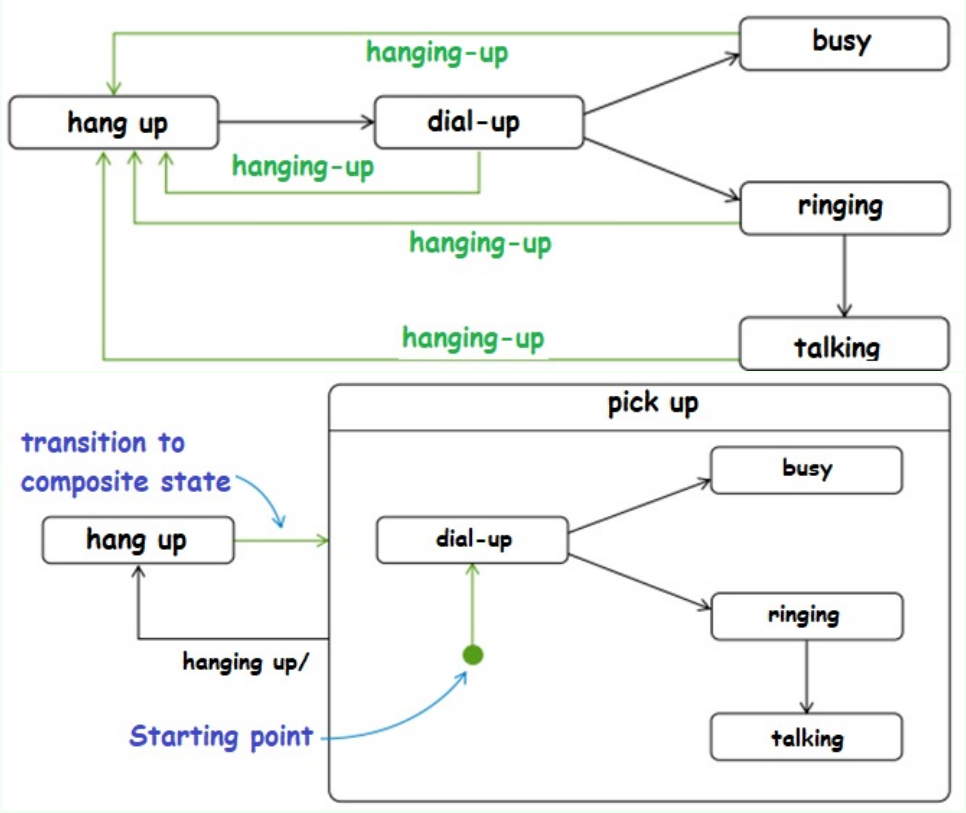


Figure 3.55: Composite states

3.11.8 Orthogonal state

What is it ?

- Composite state in which several state are active simultaneously (competition/parallelism).
- Global active status = one active status per region

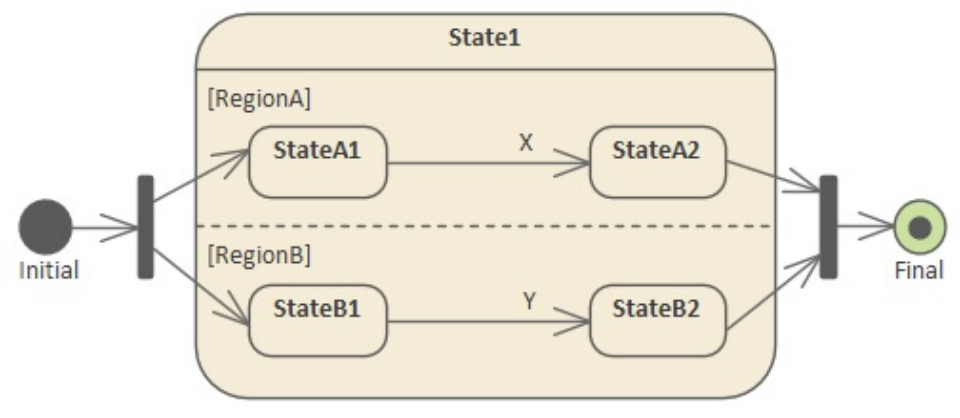


Figure 3.56: Orthogonal state

3.11.9 Internal actions, activities and events

What is it ?

- Additional characteristics of a state
- Internal events: on entry, on exit, during state
- Activity
- Status reset by external events

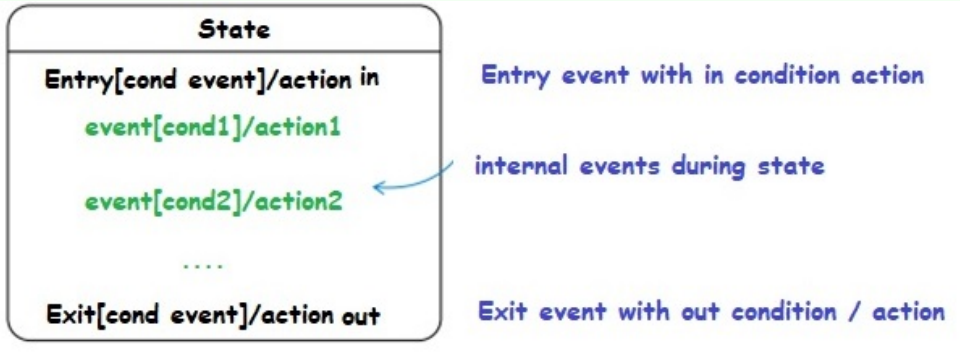


Figure 3.57: Internal events

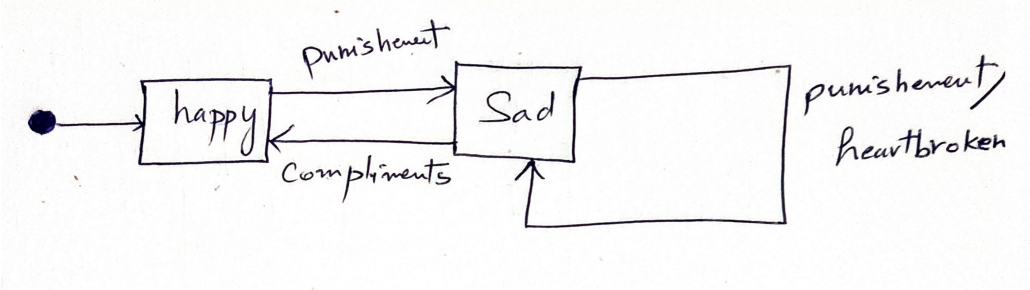
3.12 Exercices and solutions

3.12.1 Exercice 1: Digital Pets

You need to create a program to manage digital pets.What happens to the animal when it receives different stimuli is determined by its current state. You decide to model the digital pet with a state diagram. The animal's behavior in the digital program is as follows:

- When the animal is turn on, it starts with the happy state.
- If the animal is happy and receives a punishment, it becomes sad.
- If the animal is sad and receives compliments, it becomes happy.
- If the animal is sad and still receives punishment, it will be heartbroken. Identify the states and transitions of the digital animal and draw a state-transition diagram.

Solution

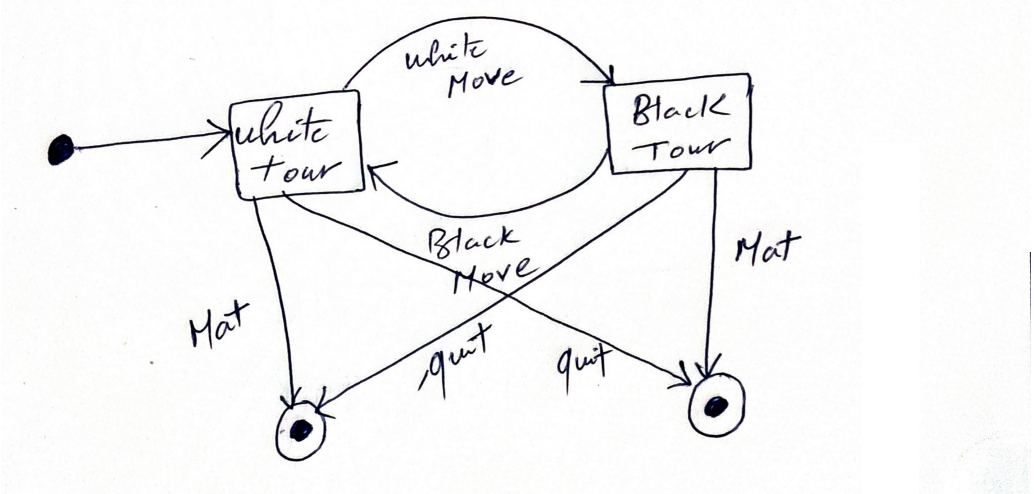


3.12.2 Exercice 2:Chess game

A chess game can be in three states:

- The white tour
- The blacks' turn.
- Game over.
- The events to be taken into consideration are :
- A move of pieces by the black player
- A move of pieces by the white player
- Checkmate ensures victory for one of these players (black or white, as the case may be), which means the end of the game.
- When one of these players quits, the game is over. Draw the state diagram for a chess game.

Solution



3.13 Activity diagram

What is it ?

- Variant of state-transition diagrams
- Represent the internal behavior of a method or use case, execution of an operation
- Specify the processing of an operation (describe the logic for executing an operatio operation) (modelling the flow of control streams and data)
- Used to model the dynamics of a task or process.

3.13.1 Start state and end state

- states: start state and end state

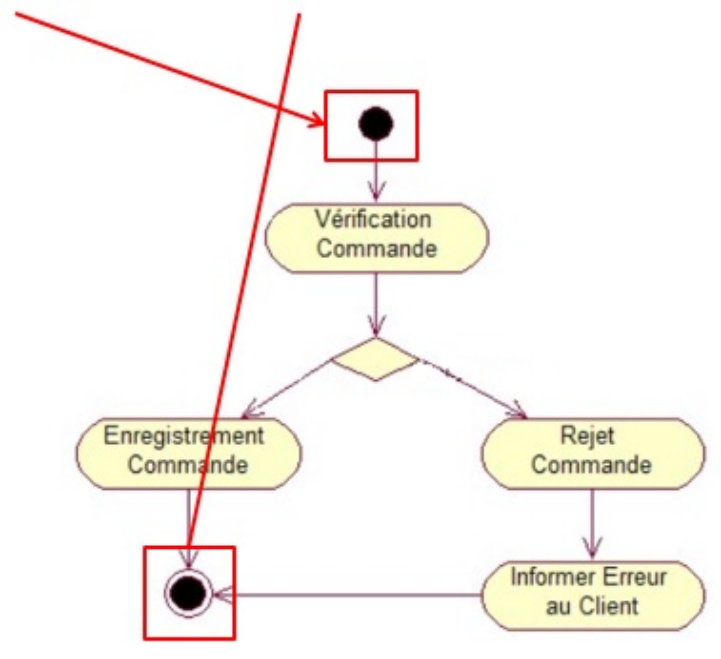


Figure 3.58: Start / End states

3.13.2 Activities

- Activitiy : An activity is something that happens in the process (in the workflow).
  - An action, an event, ...
  - By a person, a computer, ...

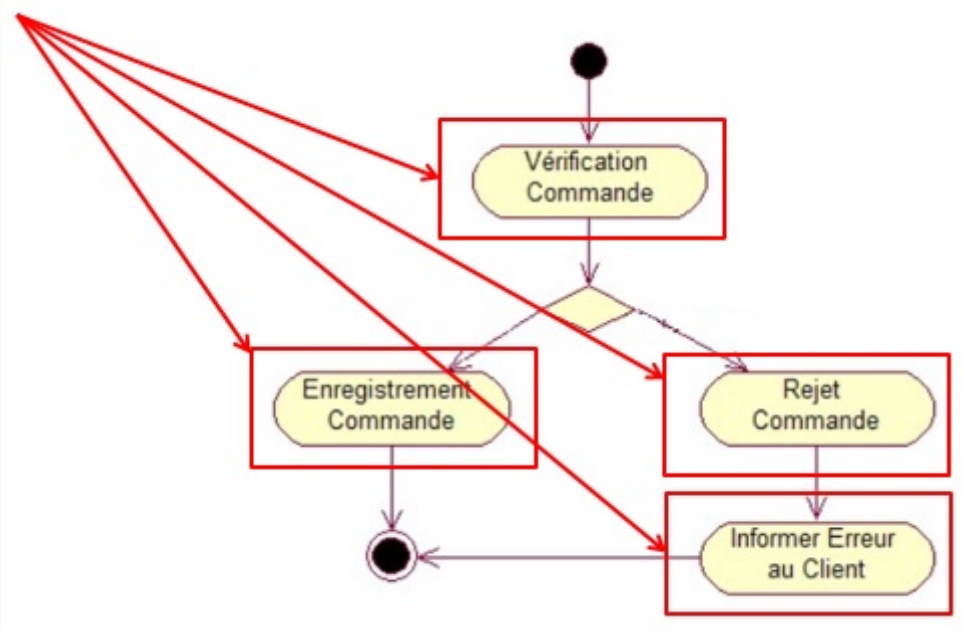


Figure 3.59: Activities

3.13.3 Transition

- Transition represents the passage from one activity to another.

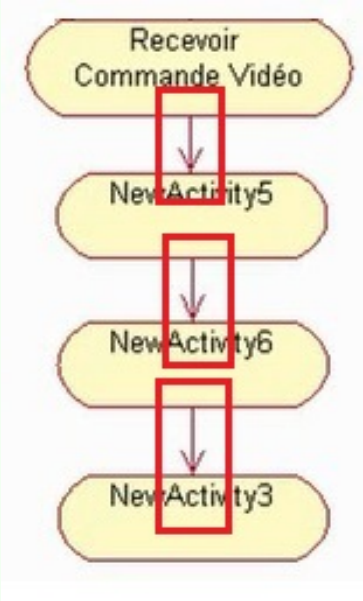


Figure 3.60: Transition

Sequential transitions

- Transition (sequentially organized)

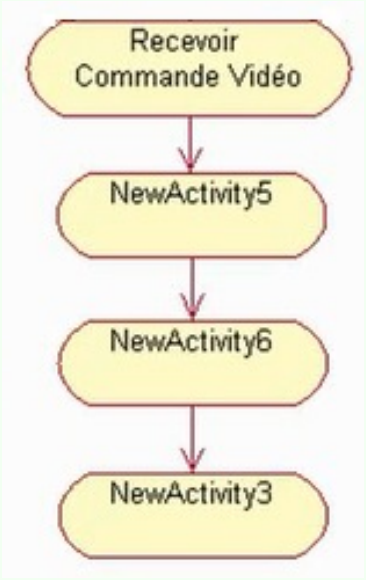


Figure 3.61: Sequential transitions

Alternative transitions

- Alternative transitions (packaged by a guard) represented by: IF / Else

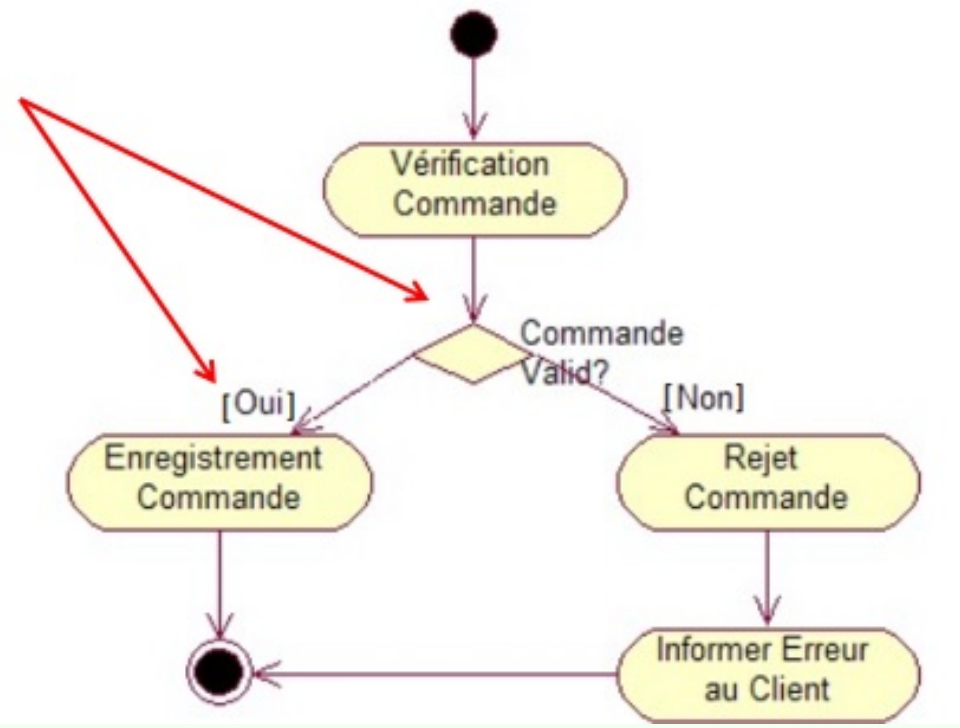


Figure 3.62: Alternative transitions

Transitions Synchronization

- Transitions Synchronization (case 1): An incoming transition and several outgoing transitions in this case both activities are performed at the same time.

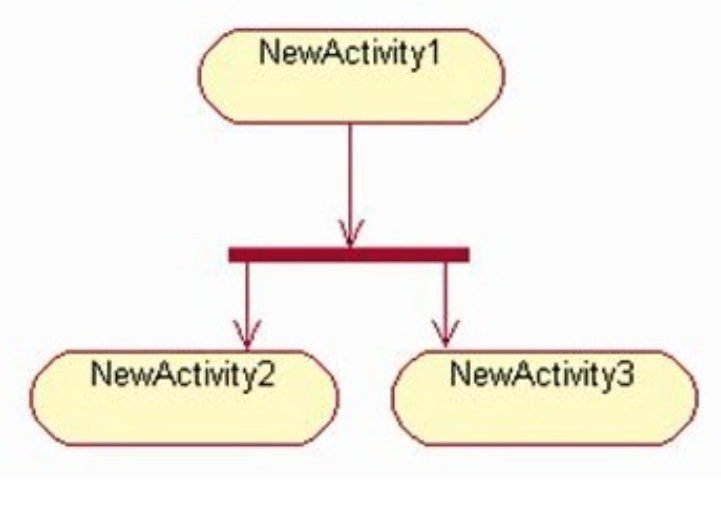


Figure 3.63: Parallel transition

- Transitions Synchronization (case 2):several incoming Transitions and an outgoing transition named **joint**.

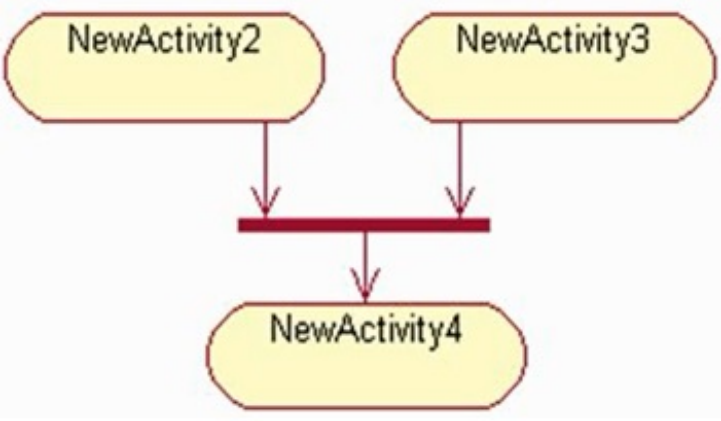


Figure 3.64: Joint transition

Swimlane

- **Swimlane**: identifies the actor responsible for the activity.

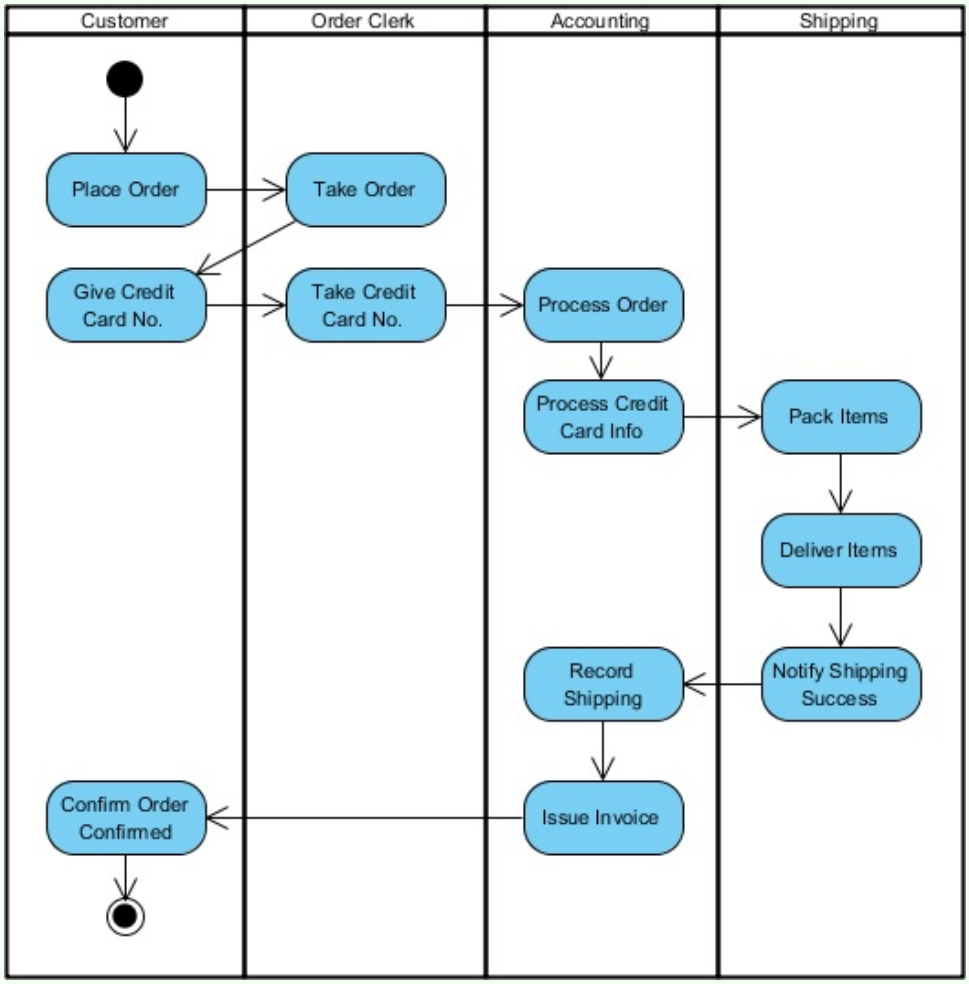
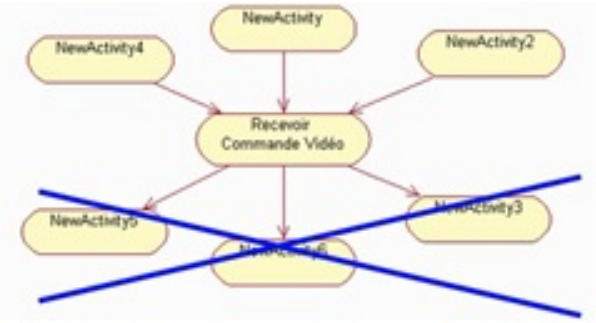


Figure 3.65: Swimlane

Warning

- Transitions Synchronization makes no sense in the following two cases:
- One transition in and one transition Out
- Several transitions **in** and Several outgoing transitions **Out**



3.14 Exercices and solutions

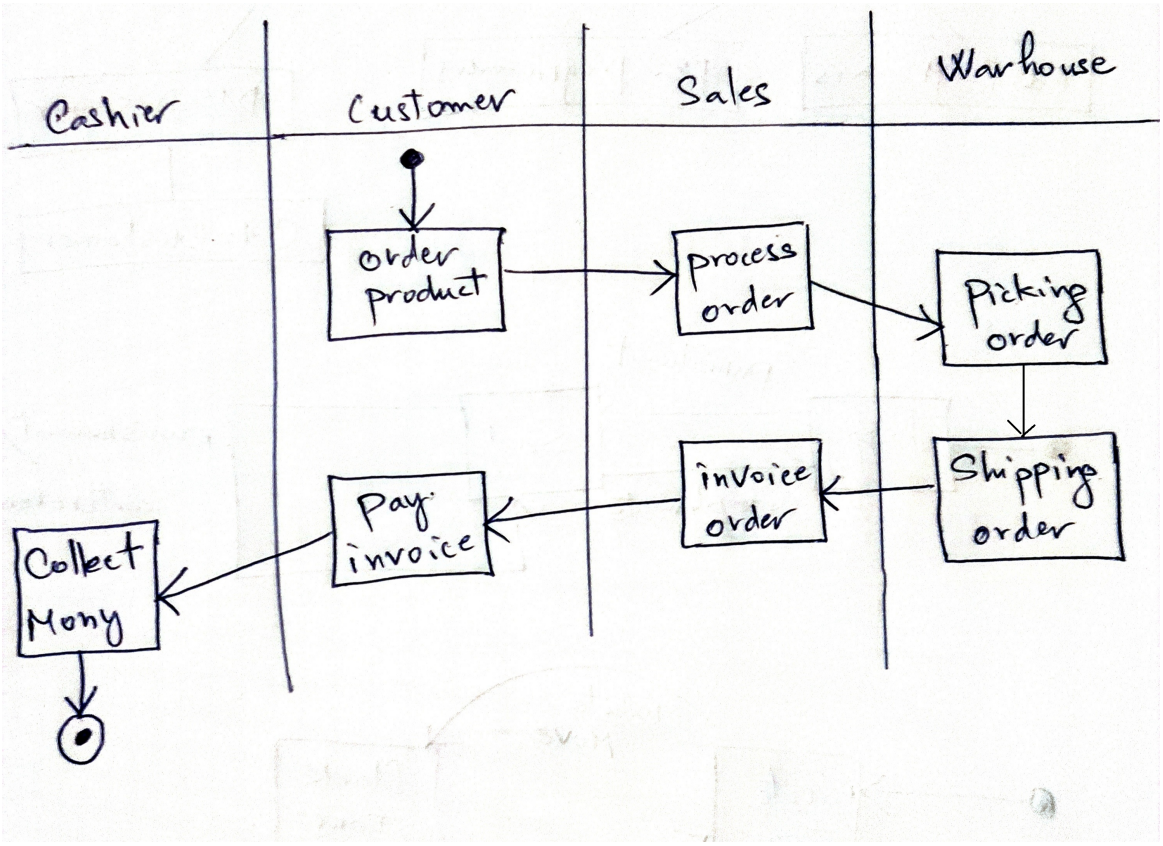
3.14.1 Exercice 1

The process involves the following actors:

- **Customer:** who orders a product and pays the invoice
- **Cashier:** who collects the customer's money
- **Sales:** who processes and invoices the customer's order
- **Warehouse:** responsible for picking and shipping the order.

Construct an activity diagram (using activity swimlane) to model the process of ordering a product.

Solution



3.15 Component and deployment diagram

3.15.1 Component diagram

What is it ?

- The **component diagram** is used to represent a system's **software components** and the links between them.
- A **component** is a self-contained, replaceable and reusable piece of software that provides or receives a specific service: source files (.java, .cpp, .h, .es ...) libraries (dll, jar ...), executables ...
- **Components** provide services via **interfaces**. A **component** can be replaced by any other compatible component, i.e. one with the same **interfaces**.
- A **component** can evolve independently of the applications or other components that use it, as long as the interfaces are respected.

Example: parallel with computer components.

A computer is a set of modular components that **provide** and **receive** services (motherboard, graphics card, hard disk, keyboard, screen, etc.). Each of these components can be replaced by another (not necessarily identical) component, provided it has **compatible interfaces**.

Warning

In UML, components are not hardware elements, but **software elements** which will be installed on hardware elements (as we'll see when we come to the deployment diagram).

Component

- Component name : Used to distinguish a component from other components. It can be a simple name or a compound name indicating the package to which the component belongs.
- Stereotypes: Specifies a component that designates :
  - "**executable**": a program that can run on a node;
  - "**Library**: a static or dynamic library;
  - "**table**": a database table;
  - "**file**": a file containing source code or data;
  - "**document**": a document.

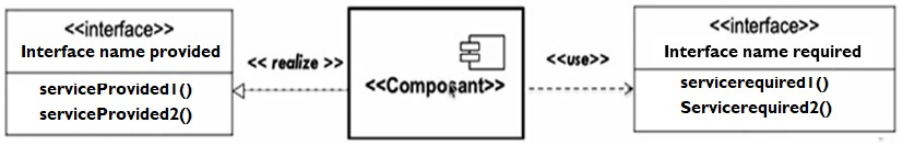
Figure 3.66: Component

Interfaces

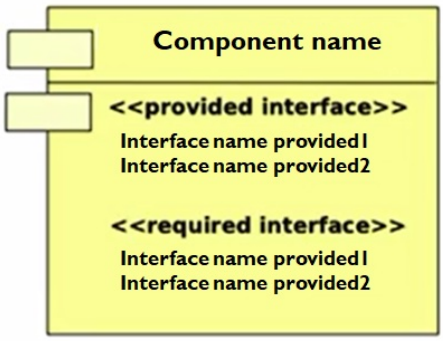
- There are two types of interface:
- **Required interfaces:** These are interfaces that provide a service to the component, and which it needs to function. They are linked to the component by a dotted arrow bearing the «**use**» stereotype.
- **Provided interfaces:** These are interfaces through which the component itself provides a service. are connected to the component by a dotted arrow on which appears the stereotype «**realize**» .

UML representation

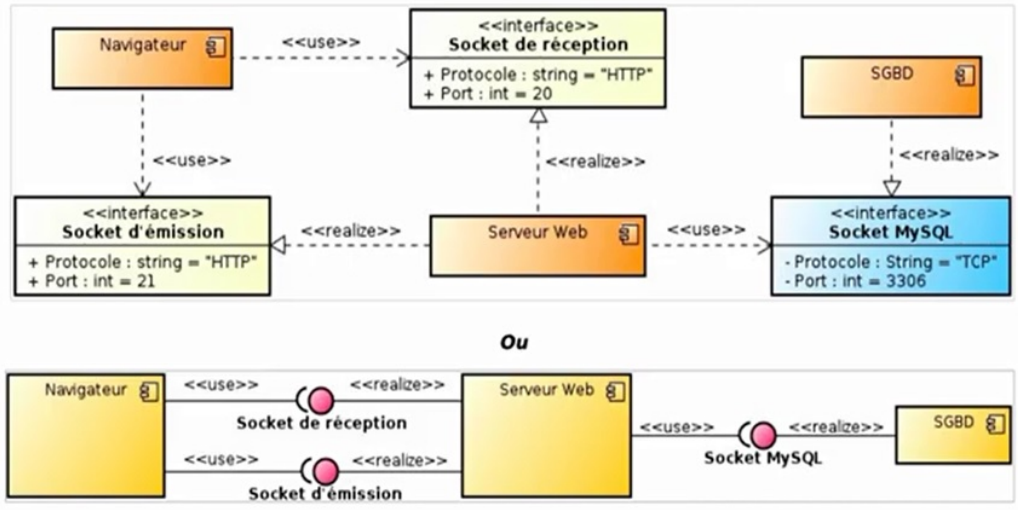
- In a separate component folder, where the various services are listed (**more details**).
- Another way to represent the component diagram (**simplify**)



- Integrated into the component representation:



Example:



3.15.2 Deployment diagram

What is it ?

- The physical layout of the hardware resources that make up the system and shows the distribution of the components (software elements) that run on this hardware.
- Communication paths between different hardware resources.

1. Material resources are represented by nodes

2. Communication d by a straight line linking the nodes

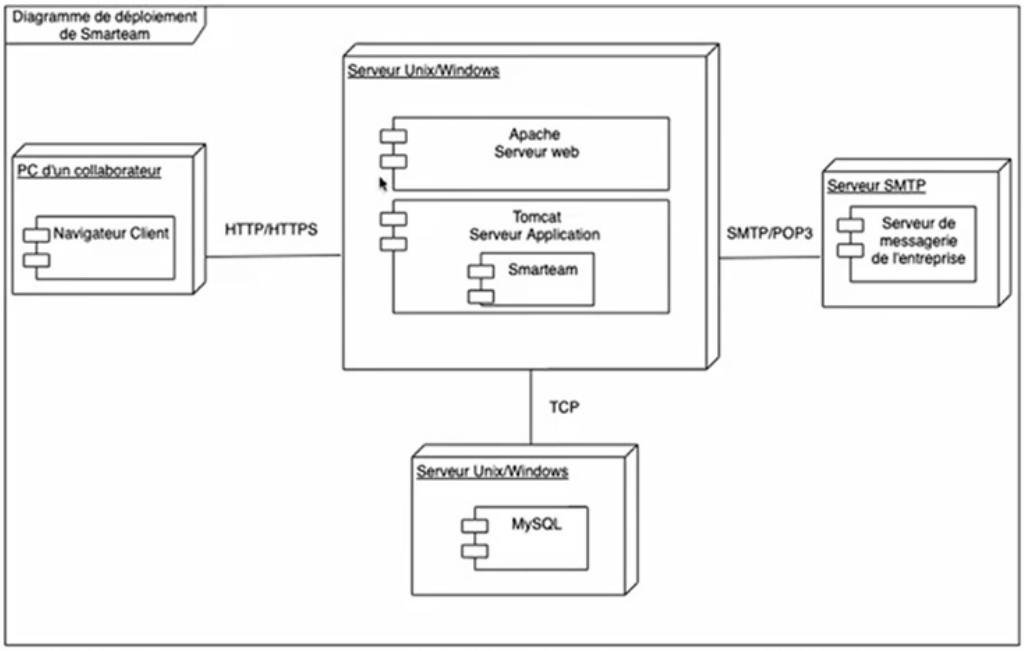
3.16 Exercices and solutions

3.16.1 Exercice 1:-The Smarteam application-

The Smarteam application The application architecture consists of the following resources:

- **Web client:**This represents the web browsers used to connect the various employees to the application.
- **UNIX I WINDOWS server:** This is the server on which the application is deployed. This server hosts the two servers:
- **Apache web** server
- **Tomcat application** server
- The two servers are merged into Apache-Tomcat version x. x.
- The UNIX or WINDOWS must be accessible to users on a LAN or WAN network.
- As Tomcat-compatible JVM JDK x.x must be installed first.
- **DBMS:** MySQL x.x database management system for data storage data
- **SMTP server:** forwards e-mails generated by the application to the appropriate when creating their profiles.

Solution



3.16.2 Exercice 2

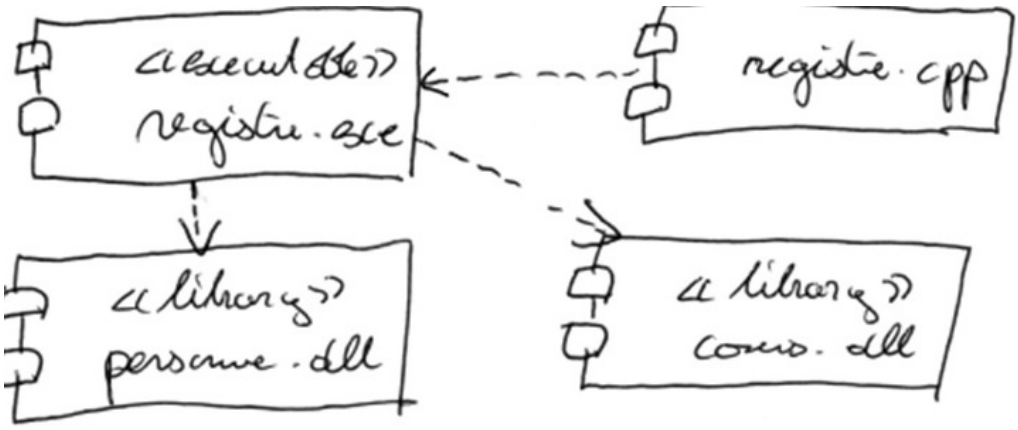
- The **image.java** component depends on the **lmageObserver** interface of the **component.java** component.

Draw the component diagram. Consider an application made up of the following files:

- Source code **register.cpp**
- An executable program **register.exe**
- Dynamic libraries **personne.dll** and **cours.dll**. Dynamic link libraries are used when running an application

Give the corresponding component diagram.

Solutions


«executable»
registre.exe
registre.cpp
«library»
personne.dll
«library»
compo.dll

Chapter 4

Exams with Answers

Université de Ghardaïa
Faculté des Sciences et Technologies
Département des Mathématiques et de l'Informatique
3ème Année Licence
Module: Génie logiciel
Année Universitaire 2021-2022

Examen Final

Durée : 01 h 30

Question du cours : Répondez brièvement à chaque question **(09 points)**

1. Quelle est l'abréviation du mot UML ? **(01 point)**
2. Comment éviter le développement d'un logiciel a faible qualité ? **(01 point)**
3. Quelle est l'objectif de la spécification dans un cycle de vie du logiciel ? **(01 point)**
4. Pourquoi nous suivons un modèle de cycle de vie du logiciel ? **(01 point)**
5. Quelle est le modèle de cycle de vie le plus adapté au développement du logiciel critique ou médical ? **(01 point)**
6. Compléter le tableaux suivant : Donnez un seule avantage et un seule inconvénient pour chaque cycle de vie **(02 point)**

Cycle de vie	Avantages	Inconvénient
Cascade		
Modèle en V		
Prototypage		
Modèle incrémentaux		

7. **Compléter les phrases suivantes : (02 points)**

   1. Spécialisation et Généralisation sont deux principes liés à ….**(0,5 point)**
   2. Un diagramme état de transition c'est un type des diagrammes …**(0,5 point)**
   3. Quand la classe A partage les mêmes attributs avec la classe B on doit utiliser …**(0,5 point)**
   4. Quand la classe A est une composant de la classe B on doit utiliser …**(0,5 point)**

Exercice n° 01 : (06 points)

Une banque a plusieurs agences réparties sur le territoire nationale. Une banque est caractérisée par le nom de son directeur général, son capital global, son propre nom et de l'adresse de son siège social. Le directeur général est identifié par son nom, son prénom et son revenu. Une agence a un numéro d'agence et une adresse. Chaque agence emploie plusieurs employés, qui se caractérisent par leurs nom, prénom et la date d'embauche. Les employés peuvent demander leur mutation d'une agence à une autre, mais un employé ne peut travailler que dans une seule agence. Les employés d'une agence ne font que gérer des clients. Un client ne peut avoir des comptes que dans une seule agence de la banque. L'agence crée des comptes pour différents clients. Les clients ont un nom, un prénom et une adresse. Les comptes ont un numéro et un taux d'intérêt.

Questions : -Dessiner le diagramme de classe.

Exercice n° 02 : (06 points)

Dans un école de formation, les inscriptions se déroulent de la façon suivante : Au début de chaque semestre, un catalogue des cours proposés est fourni par la scolarité aux étudiants.

Ce catalogue ne peut être créé avant que tous les cours ne soient affectés à des enseignants. Pour cela, chaque enseignant doit indiquer les cours qu'il prévoit d'enseigner.

Les étudiants doivent remplir des fiches d'enregistrement qui indiquent leurs choix de cours. Chaque étudiant doit suivre cinq cours choisis dans le catalogue. Il devra indiquer aussi trois cours supplémentaires. En effet, il se peut que, parmi les cours choisis, l'un des cours doit être dispensé à au moins 5 étudiants et au plus 30 étudiants. Par exemple, si un cours est choisi par moins de 5 étudiants, il est supprimé. Si un cours est choisi par plus de 30 étudiants, il est marqué comme trop plein. Ces activités et les fiches d'inscription sont gérées par la scolarité.

Une fois la période d'inscription terminée, un programme est exécuté pour affecter les étudiants aux cours. Après que tous les étudiants aient été correctement affectés aux différents cours. L'information est transmise au système de facturation qui facturera l'étudiant pour son semestre.

Questions : - Dessiner le diagramme de cas d'utilisation.

Bonne Chance

Université de Ghardaïa
Faculté des Sciences et Technologies
Département des Mathématiques et de l'Informatique
3ème Année Licence
Module: Génie logiciel
Année Universitaire 2021-2022

Examen Final (Corrigé type)

Durée : 01 h 30

Question de cours : Répondez brièvement à chaque question **(09 points)**

1. Quelle est l'abréviation du mot UML ? **(01 point)**

Unified modeling Language / language de modélisation unifié

2. Comment éviter le développement d'un logiciel a faible qualité ? **(01 point)**

A travers les techniques d'ingénierie du logiciel et les modèles du cycle de vie.

3. Quelle est l'objectif de la spécification dans un cycle de vie du logiciel ? **(01 point)**

Comprendre les besoins du client et la réalisation de cahier de charge.
Établir une description claire de ce que doit faire le logiciel

4. Pourquoi nous suivons un modèle de cycle de vie du logiciel ? **(01 point)**

Pour développer un logiciel de qualité.

5. Quelle est le modèle de cycle de vie le plus adapté au développement du logiciel critique ou médical ? **(01 point)**

Modèle orienté risque (modèle en Spiral)

6. Compléter le tableaux suivant : Donnez une seule avantage et une seule inconvénient pour chaque cycle de vie **(02 point)**

Cycle de vie	Avantages	Inconvénient
Cascade	**Simple et séquentielle (0,25)**	**Difficile de mettre à jour pour des nouvelle exigences (0,25)**
Modèle en V	**Basé sur les tests (0,25)**	**Très lourd (0,25)**
Prototypage	**Simplifier les besoins du client à travers un prototype (0,25)**	**Difficile de validé ce prototype (0,25)**
Modèle incrémentaux	**Découpage vers des taches simples (0,25)**	**Problèmes d'intégration (0,25)**

7. **Compléter les phrases suivantes : (02 points)**

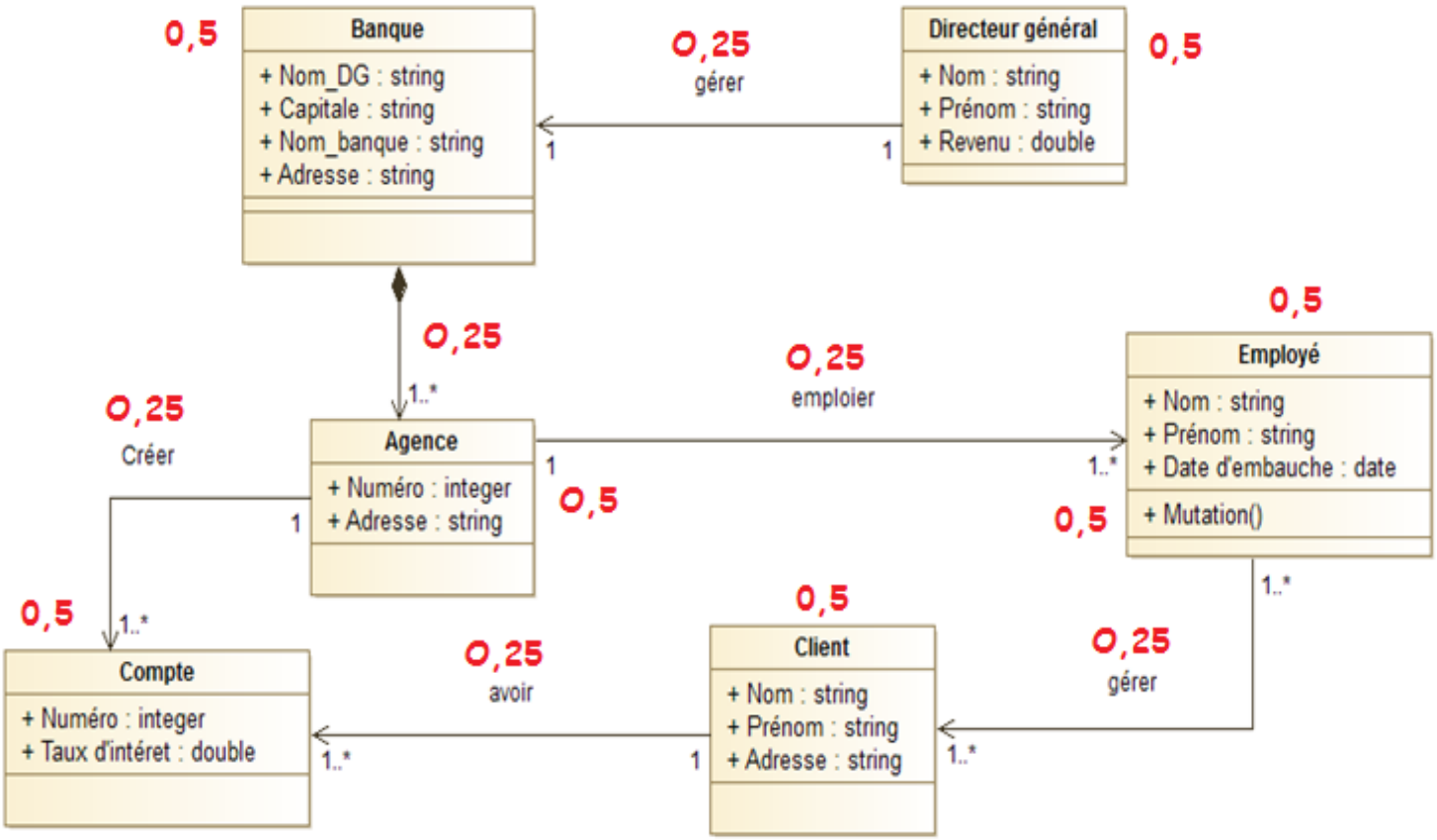
- Spécialisation et Généralisation sont deux principes liés à l'**héritage** **(0,5 point)**
- Un diagramme état de transition c'est un type des diagrammes **Comportementaux** **(0,5 point)**
- Quand la classe A partage les mêmes attributs avec la classe B on doit utiliser l'**héritage** **(0,5 point)**
- Quand la classe A est une composent de la classe B on doit utiliser **composition ou agrégation** **(0,5 point)**

Exercice n° 01 : Diagramme de classe (05 points)

Une banque a plusieurs agences réparties sur le territoire nationale. Une banque est caractérisée par le nom de son directeur général, son capital global, son propre nom et de l'adresse de son siège social. Le directeur général est identifié par son nom, son prénom et son revenu. Une agence a un numéro d'agence et une adresse. Chaque agence emploie plusieurs employés, qui se caractérisent par leurs nom, prénom et date d'embauche. Les employés peuvent demander leur mutation d'une agence à une autre, mais un employé ne peut travailler que dans une seule agence. Les employés d'une agence ne font que gérer des clients. Un client ne peut avoir des comptes que dans une seule agence de la banque. Les clients ont un nom, un prénom et une adresse. Les comptes ont un numéro et un taux d'intérêt.

Questions : -Dessiner le diagramme de classe.

Solution :



Exercice n° 02 : Diagramme de cas d'utilisation (06 points)

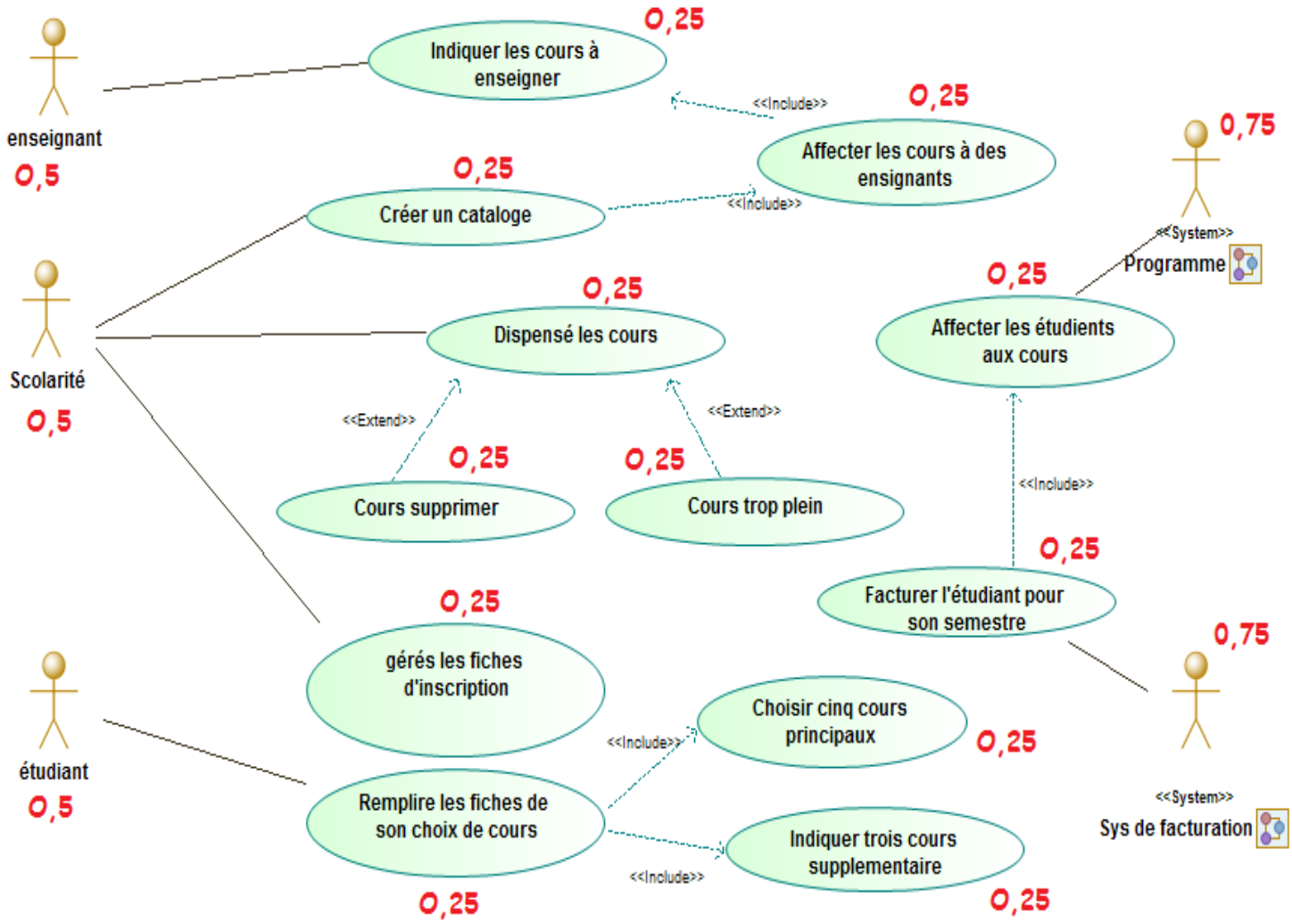
Dans un école de formation, les inscriptions se déroulent de la façon suivante : Au début de chaque semestre, un catalogue des cours proposés est fourni par la scolarité aux étudiants.

Ce catalogue ne peut être créé avant que tous les cours ne soient affectés à des enseignants. Pour cela, chaque enseignant doit indiquer les cours qu'il prévoit d'enseigner.

Les étudiants doivent remplir des fiches d'enregistrement qui indiquent leurs choix de cours. Chaque étudiant doit suivre cinq cours choisis dans le catalogue. Il devra indiquer aussi trois cours supplémentaires. En effet, il se peut que, parmi les cours choisis, l'un des cours doit être dispensé à au moins 5 étudiants et au plus 30 étudiants. Par exemple, si un cours est choisi par moins de 5 étudiants, il est supprimé. Si un cours est choisi par plus de 30 étudiants, il est marqué comme trop plein. Ces activités et les fiches d'inscription sont gérées par la scolarité.

Une fois la période d'inscription terminée, un programme est exécuté pour affecter les étudiants aux cours. Après que tous les étudiants aient été correctement affectés aux différents cours. L'information est transmise au système de facturation qui facturera l'étudiant pour son semestre.

Questions : - Dessiner le diagramme de cas d'utilisation.



Université de Ghardaïa
Faculté des Sciences et Technologies
Département des Mathématiques et de l'Informatique
3ème Année Licence
Module: Génie logiciel
Année Universitaire 2022-2023

Examen Final (les téléphones portables sont interdits) Durée : 01 h 30

Exercice (Etude de cas)

Nous souhaitons modéliser le système de gestion des revues nationales. Chaque revue est décrite par un code unique, un titre, une adresse, un site web et un type, pouvant être scientifique, politique ou sportive.

Chaque revue est dirigée par un rédacteur en chef, qui est un journaliste, et emploie un certain nombre de journalistes ainsi que d'autres fonctionnaires (administrateur du système, comptable, infographe….).

Chaque fonctionnaire est identifié par un numéro et décrit par son nom, prénom, date de naissance, adresse, téléphone et email. Nous distinguons deux types de journalistes : permanent et correspondant.

Un rédacteur est un journaliste permanent. Un journaliste permanent est attaché à une seule revue.

Un journaliste écrit des articles. Un article est publié dans un seul numéro d'une revue donnée. Un numéro d'une revue est composé d'un certain nombre d'articles. Il est identifié par un numéro unique et est décrit par le mois et l'année de publication.

Un article est caractérisé par un code unique, un titre, un domaine, un thème, et un contenu.

Seuls les journalistes permanents peuvent faire des interviews avec différentes catégories de personnalités. Chaque interview faite par un journaliste donné avec une personnalité donnée, ne pourra apparaitre que dans un seul numéro de revue. Une personnalité sera décrite par un numéro unique, son nom, prénom, fonction, domaine d'intérêt, adresse et email.

En plus à cette description, l'analyse des besoins a fait ressortir un ensemble de fonctionnalités, dont les suivantes :

Le rédacteur en chef a la responsabilité de gérer le recrutement de son personnel, de choisir les articles qui seront publiés et d'en sélectionner les numéros de revue correspondants. Il peut refuser la publication d'un article donné pour un certain motif. Ceci dit, dans le cas où la revue reçoit de nombreuses appréciations de la part de ses lecteurs concernant un article donné, le rédacteur en chef peut décider de publier une extension à cet article. Ce dernier sera identifié par un nouveau numéro mais tout en gardant le lien vers l'article d'origine.

Chaque journaliste peut consulter tous les articles publiés, proposer un article au rédacteur en chef, porter des mises à jours sur l'article avant sa publication.

Le système donne la possibilité aux lecteurs de commander en ligne un numéro de revue donné sous le format papier. Deux types de lecteurs sont distingués : abonné ou non (lecteur occasionnel) :

- Une réduction de 10% est faite au profit des abonnés. L'abonné s'identifie par son numéro de carte d'abonnement et sélectionne le(s) numéro(s) commandé(s). Puis, il procède au payement en ligne en faisant entrer son numéro de carte bancaire. Une fois avoir vérifié le solde de l'abonné, ce dernier pourra récupérer sa commande qui lui sera livrée par un agent de livraison.
- Par contre, si le lecteur n'est pas abonné à la revue, alors il devra entrer aussi dans le système son nom et prénom ainsi que son adresse et son email et procéder aux mêmes étapes décrites ci-dessus (payement en ligne…).

Une fois la commande est arrivée à destination, le lecteur procède à la vérification. Dans le cas où ce dernier (abonné ou non) trouve des problèmes dans sa livraison (exemple : erreur de numéro de revue) alors il pourra signaler et demander une autre livraison.

De plus, les lecteurs non abonnés ont la possibilité de télécharger les versions numériques correspondantes aux numéros de revues qu'ils ont payées, alors que les abonnés peuvent télécharger toutes les versions publiées sur le site.

La sécurité des données est un aspect important, pour cela afin de garder une traçabilité et des accès personnalisés au système, tous les employés doivent s'authentifier avant d'effectuer toute opération.

Questions:

1) Tracer le diagramme de cas d'utilisation. (**10 points**)

2) Tracer le diagramme de classe. (**10 points**)

Université de Ghardaïa
Faculté des Sciences et Technologies
Département des Mathématiques et de l'Informatique
3ème Année Licence
Module: Génie logiciel
Année Universitaire 2022-2023

Examen Final (Corrigé type)

Durée : 01 h 30

Diagramme de cas d'utilisation:

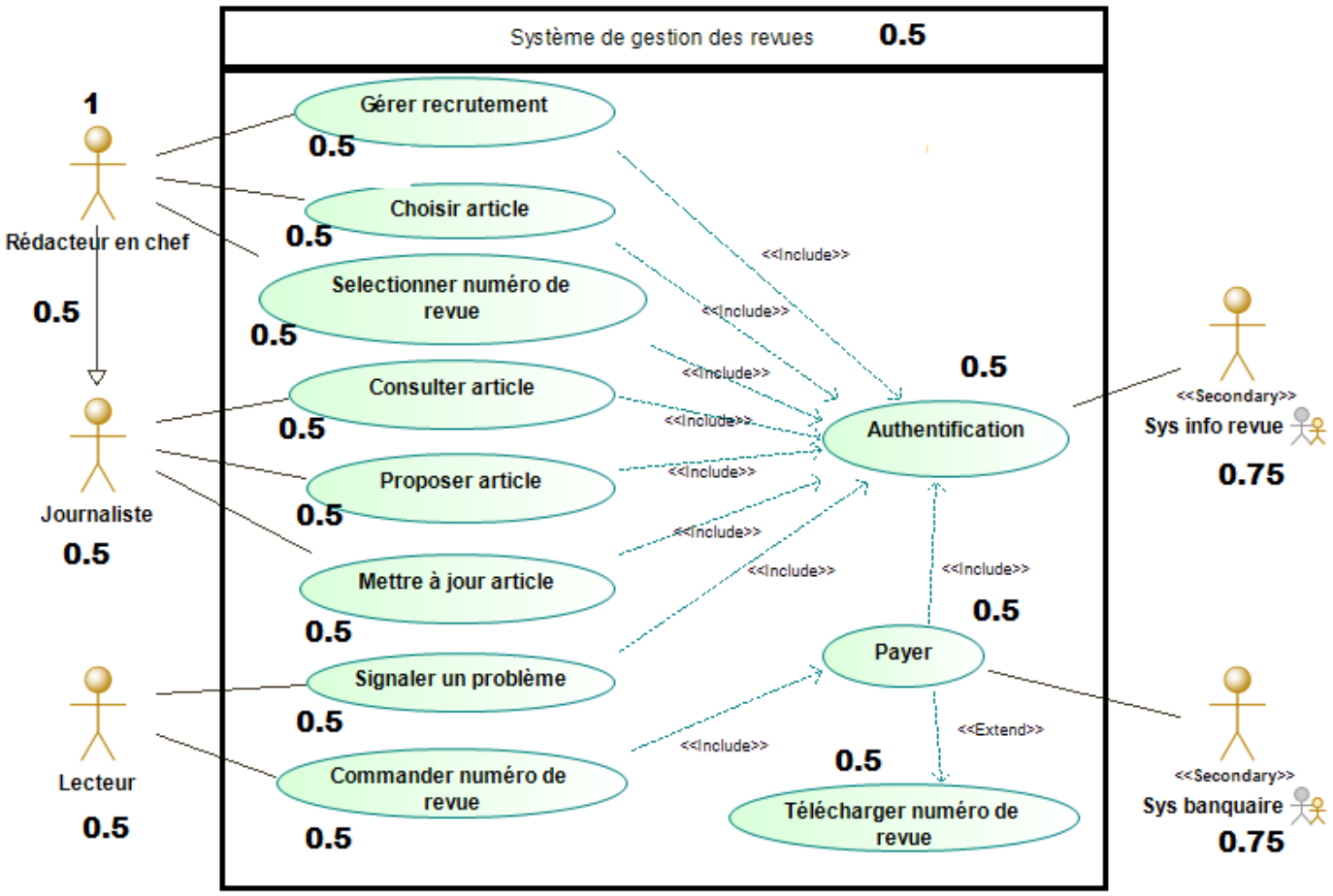
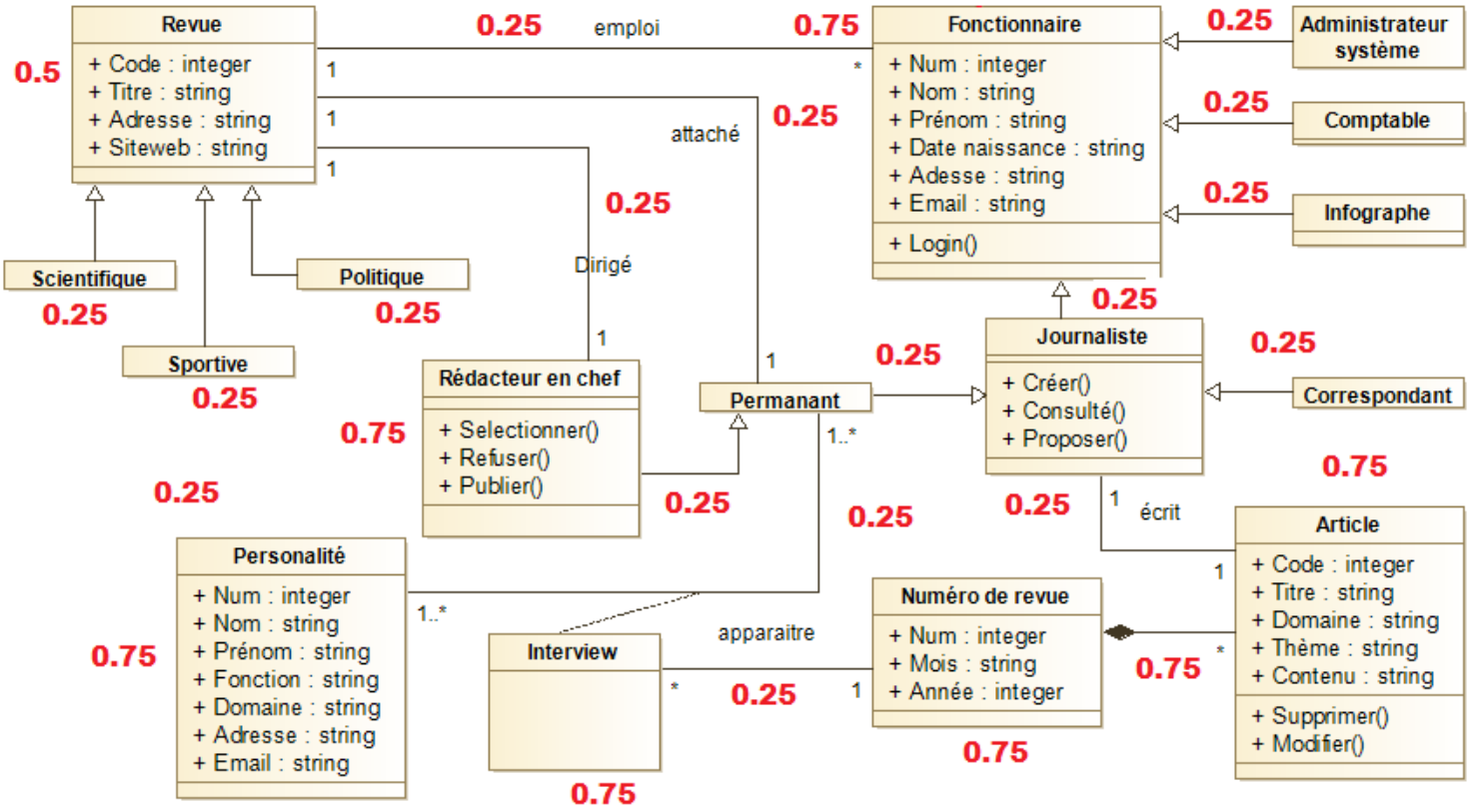


Diagramme de classe:



Université de Ghardaïa
Faculté des Sciences et Technologies
Département des Mathématiques et de l'Informatique
3ème Année Licence
Module: Génie logiciel
Année Universitaire 2022-2023

Examen TD

Durée : 01 h 30

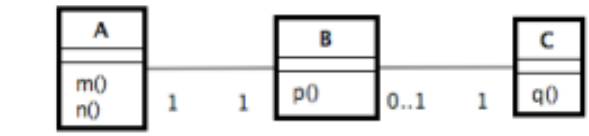
Exercice n° 01 : (06 points)

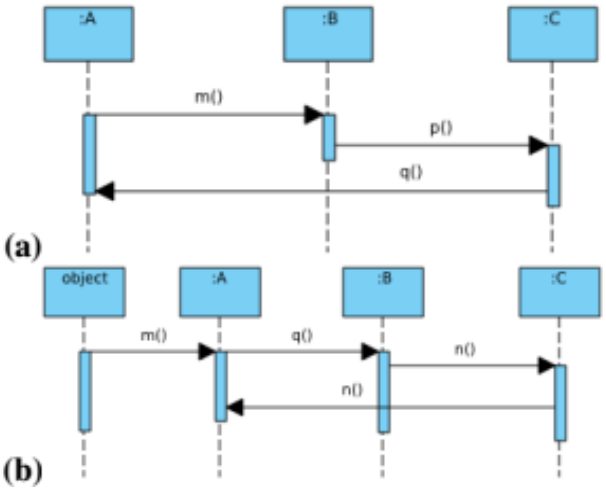
L'analyse d'un système, qui permet d'écouter de la musique en streaming sur internet, a permis d'obtenir le tableau des exigences suivant:

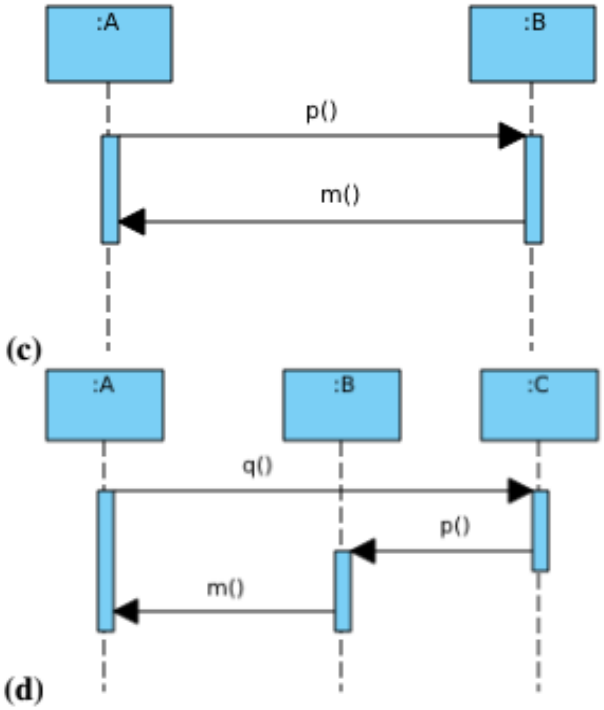
N°	Exigence	Type (F, NF)
1	Le système doit permettre de jouer de la musique.	
2	Le système doit permettre de chercher une musique.	
3	Le système doit permettre de démarrer la musique dans les 2 secondes suivant un clic.	
4	Le système doit permettre de créer une playlist.	
5	Le système doit permettre de baisser et augmenter le volume	
6	Le système doit avoir une belle IHM, ergonomique et pratique à utiliser.	
7	Le système ne doit pas permettre d'utiliser plus de 50% de la bande passante	
8	Le système doit permettre une consommation raisonnable du temps de CPU.	
9	Le système doit permettre la lecture de la music depuis URL.	
10	Le système doit gérer la plupart des formats connus et utilisés sur l'internet.	
11	Le système doit permettre d'activer l'égaliseur de son.	
12	Le système doit permettre de répéter une musique.	

On vous demande d'aider les développeurs de ce système à trier les exigences selon leur nature (**F** : fonctionnelles ou **NF** : non-fonctionnelles).

Exercice n° 02 :(04 points) Soit les diagrammes de séquence objet (a,b,c et d) liés au diagramme de classe suivant :







- Est-ce que ces diagrammes sont corrects ?

Exercice n° 03 : (10 points)

Un système va être développé pour contrôler N ascenseurs dans un bâtiment de M étages. Notre problème concerne la logique nécessaire au déplacement des ascenseurs entre les étages en accord avec les contraintes suivantes:

a. chaque ascenseur possède un ensemble de M boutons, un pour chaque étage. Un bouton s'allume lorsqu'il est appuyé et provoque le déplacement de l'ascenseur vers l'étage correspondant.
b. chaque étage, à l'exception du premier et du dernier, possède deux boutons, un pour demander la montée et un pour demander la descente. Ces boutons s'allument lorsqu'ils sont appuyés. Ils s'éteignent quand l'ascenseur arrive à l'étage, et celui-ci se déplace ensuite dans la direction demandée.
c. quand un ascenseur n'est pas requis, il reste à l'étage où il se trouve et ferme ses portes.

Questions :

Décrire à l'aide d'un diagramme de séquence le scénario suivant:

- requête d'ascenseur depuis l'étage

Université de Ghardaïa
Faculté des Sciences et Technologies
Département des Mathématiques et de l'Informatique
3ème Année Licence
Module: Génie logiciel
Année Universitaire 2022-2023

Examen TD Correction

Durée : 01 h 30

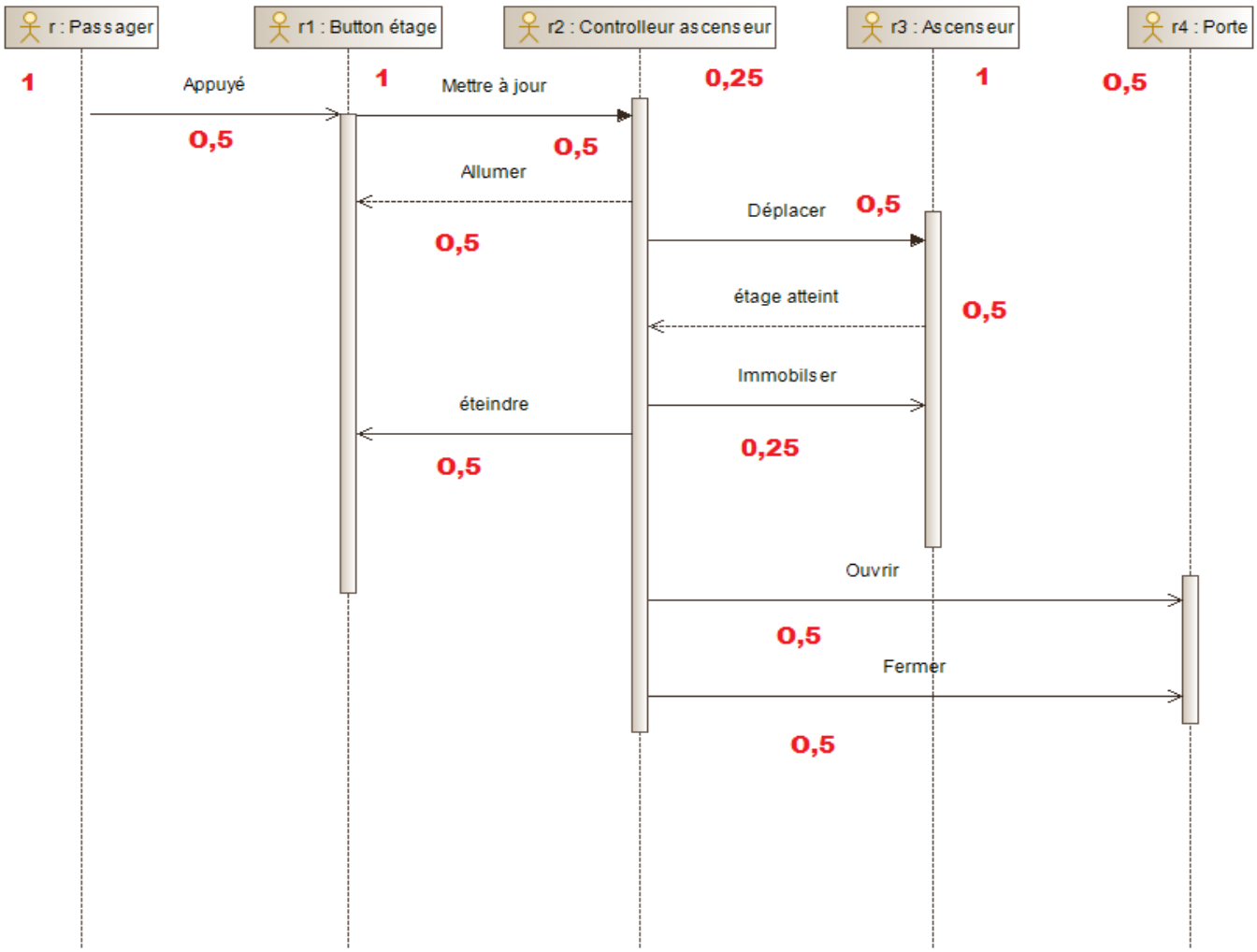
Exercice n° 01 : (06 points) 0,5 x 12

#	Exigence	Type (F, NF)
1	Le système doit permettre de jouer de la musique.	**F**
2	Le système doit permettre de chercher une musique.	**NF**
3	Le système doit permettre de démarrer la musique dans les 2 secondes suivant un clic.	**NF**
4	Le système doit permettre de créer une playlist.	**F**
5	Le système doit permettre de baisser et augmenter le volume	**F**
6	Le système doit avoir une belle IHM, ergonomique et pratique à utiliser.	**NF**
7	Le système ne doit pas permettre d'utiliser plus de 50% de la bande passante	**NF**
8	Le système doit permettre une consommation raisonnable du temps de CPU.	**NF**
9	Le système doit permettre la lecture de la music depuis URL.	**F**
10	Le système doit gérer la plupart des formats connus et utilisés sur l'internet.	**F**
11	Le système doit permettre d'activer l'égaliseur de son.	**NF**
12	Le système doit permettre de répéter une musique.	**F**

Exercice n° 02 :(04 points) (01x 4)

(a), (b) et (d) sont incorrects, (c) est correct.

Exercice n° 03 : (10 points)



+ 02 points : (concernant les règles du diagramme de séquence, type des messages, la barre d'activation,…etc)

Université de Ghardaïa
Faculté des Sciences et Technologies
Département des Mathématiques et de l'Informatique
3ème Année Licence
Module: Génie logiciel
Année Universitaire 2022-2023

Examen Final (les téléphones portables sont interdits) Durée : 01 h 30

Exercice (Etude de cas)

La société de transport collectif 'DZBus' a fait appel à une entreprise de développement pour informatiser la gestion de son activité. Le système à développer est une application web appelée "LogiLigneBus". La société a exigé que le futur site soit compatible avec le plus grand nombre de navigateurs web sur le marché.

La société 'DZbus' possède un parc de bus et emploie un responsable d'exploitation. Ce dernier devra à travers le système affecter des chauffeurs à différents bus et différentes lignes. Chaque ligne est une suite d'arrêts, pour chaque arrêt, le responsable d'exploitation fixe les heures de passages du bus. Les chauffeurs peuvent consulter et éventuellement imprimer leur activité de la journée. L'activité d'un chauffeur est constituée d'un ensemble de créneaux et de lignes correspondantes affectées.

Les chauffeurs peuvent demander un changement de ligne, pour cela, le chauffeur se connecte au système via son compte et choisit l'opération « émettre une demande de changement ». Le système lui retourne une fiche à remplir. Le chauffeur renseigne la ligne actuelle et choisit entre deux types de lignes : ligne interwilaya ou inttawilaya . Si le type choisi est interwilaya le chauffeur doit choisir la wilaya désirée et doit accompagner sa demande d'une autorisation délivrée par la wilaya. Si la ligne est intrawilaya une liste de lignes est ensuite retournée. Le chauffeur sélectionne la ligne désirée. Le système enregistre sa demande et affiche un message de confirmation.

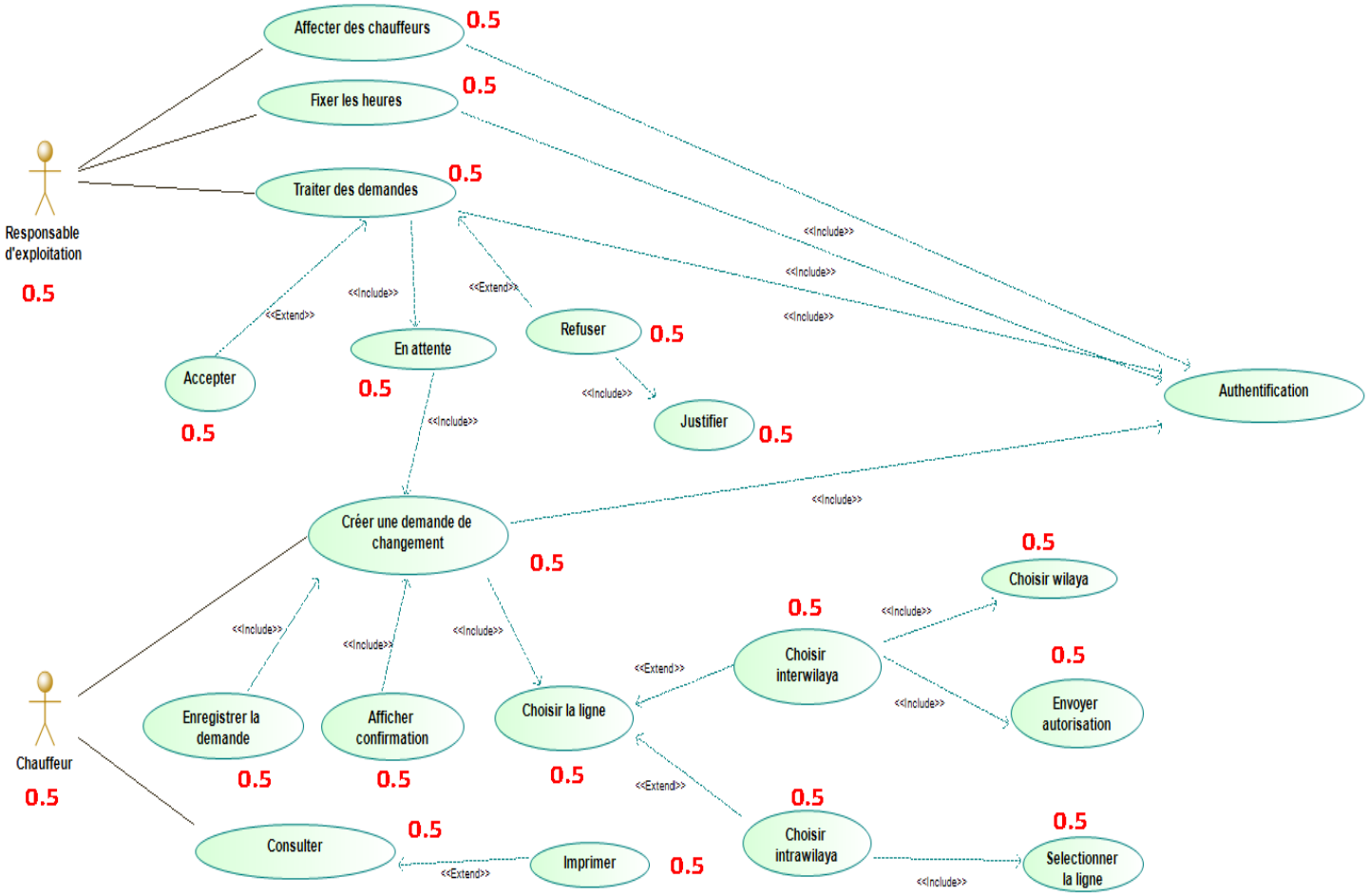
Dans tous les cas, la demande de changement reste en attente jusqu'à ce quelle soit traitée par le responsable d'exploitation qui peut soit accepter ou refuser. Dans le cas de refus, il doit impérativement justifier le refus.

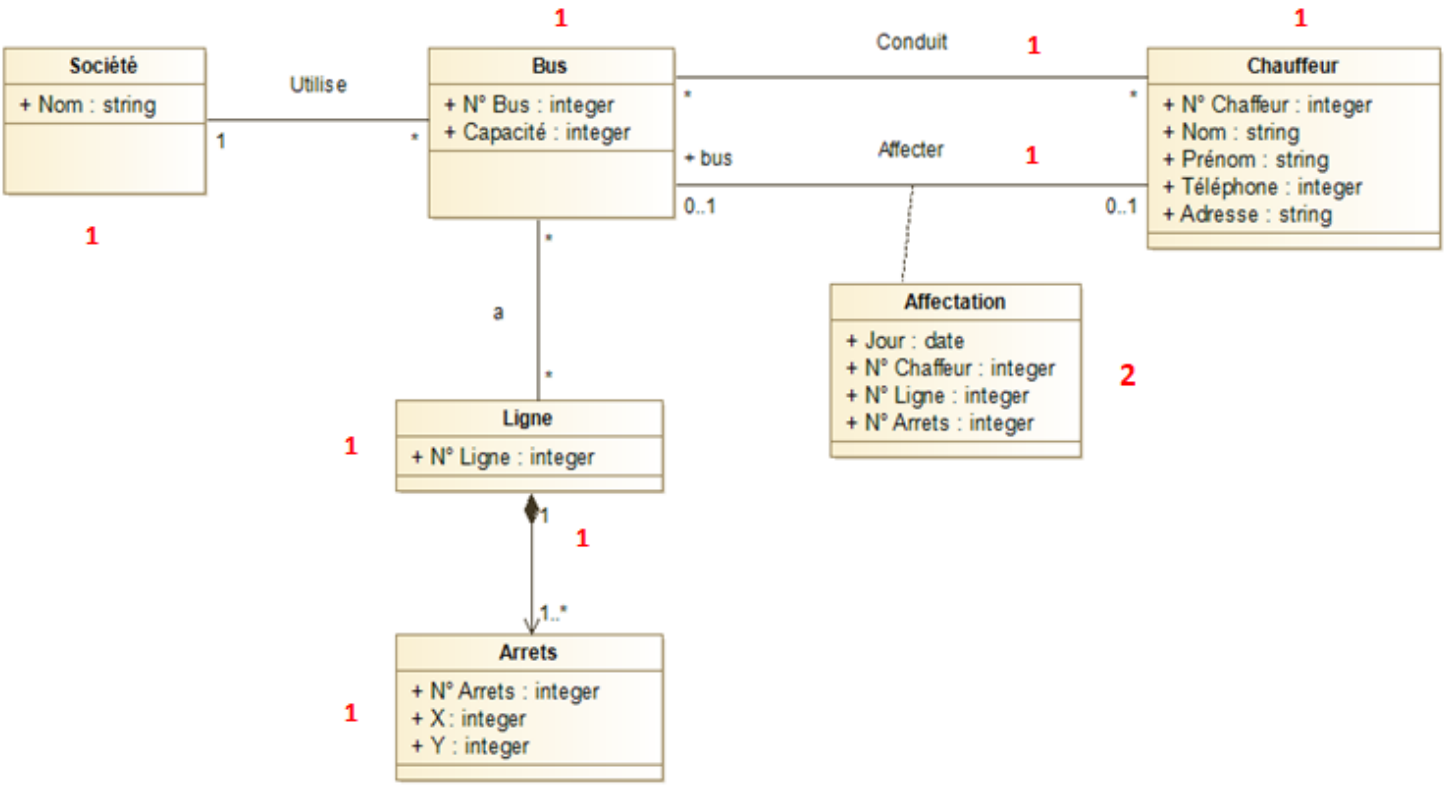
1. Dessiner le diagramme de cas d'utilisation. . (**10 points**)

2. Etablir le diagramme de classes du domaine en considérant la description du système et les informations additionnelles suivantes : . (**10 points**)

- La société utilise des bus. Ils sont identifiés par un numéro unique et on conserve leur capacité en nombre de passagers.
- Les chauffeurs sont identifiés par un numéro unique. Les informations les concernant sont leur nom, prénom, téléphone et adresse
- Les chauffeurs peuvent potentiellement conduire tous les bus, mais un Chauffeur n'est affecté qu'à un seul bus par jour.
- Les différentes lignes de bus portent un numéro et sont constituées par une séquence d'arrêts. Chaque arrêt est défini par un numéro unique et ses coordonnées géographiques.

Examen rattrapage -correction




1
1
Société
+ Nom : string
Utilise
1
*
Bus
+ N° Bus : integer
+ Capacité : integer
Conduit
1
*
*
Chauffeur
+ N° Chaffeur : integer
+ Nom : string
+ Prénom : string
+ Téléphone : integer
+ Adresse : string
+ bus
Affecter
1
0..1
0..1
1
*
a
*
Affectation
+ Jour : date
+ N° Chaffeur : integer
+ N° Ligne : integer
+ N° Arrets : integer
2
1
Ligne
+ N° Ligne : integer
1
1
1..*
1
Arrets
+ N° Arrets : integer
+ X : integer
+ Y : integer

Examen Final (les téléphones portables sont interdits) Durée : 01 h 30

Questions de cours : (02pts)

1. Quelle est la différence principale entre le diagramme de séquence système et le diagramme de séquence objets ?
2. Donnez deux diagrammes UML utilisés en phase de spécification ?

Exercice 1: (10 pts)

Considérons le système d'un distributeur automatique de boissons (café, jus, thé,..etc). La machine délivre à l'utilisateur la boisson qu'il a sélectionné si ce dernier a introduit une somme d'argent suffisante (en espèce ou carte bancaire). Dans le cas du liquide, La machine rend éventuellement la monnaie suivant le stock de pièce dont elle dispose. Dans le cas du paiement avec une carte bancaire, la machine connecte le système d'information de la banque. Lorsque le stock de boisson n'est pas suffisant, la machine informe l'entreprise chargée de sa maintenance par le biais d'une connexion électronique. Un opérateur de maintenance est alors envoyé par l'entreprise pour le renouveler et récupérer l'argent liquide. Le client achète une boisson de la façon suivante : il entre son choix en sélectionnant une boisson dans un menu, introduit de l'argent dans le lecteur de billets ou le monnayeur électronique multi-pièces., le distributeur délivre à l'utilisateur la boisson choisie. Le distributeur est très sophistiqué et est relié à un système bancaire et accepte ainsi un paiement par carte bancaire.

Question 1 : Dessinez le diagramme de cas d'utilisation.

Question 2 : Mohammed qui est un opérateur de maintenance du distributeur de boissons se sert aussi de la machine pour acheter un café. Pour modéliser cette activité de Mohammed, doit-on modifier le diagramme précédent (corresponde la question **1**) ?. Si la réponse est oui, donc rajoutez cette modification au diagramme.

Question 3 : La machine est dotée d'un bouton d'annulation permettant d'annuler l'achat de la boisson. Dans ce cas la somme d'argent est retournée au client. Rajouter ce cas d'utilisation à votre diagramme.

Exercice 2: (08 pts)

Une académie souhaite gérer les cours présentés dans plusieurs écoles. Pour cela, on dispose des renseignements suivants :

• Chaque école possède d'un site Internet.

• Chaque école est structurée en départements, qui regroupent chacun des enseignants spécifiques.

• Un enseignant se définit par son nom, prénom, tél, mail, date de prise de fonction et son indice.

• Chaque enseignant n'enseigne qu'une seule matière.

• Les étudiants suivent quant à eux plusieurs matières et reçoivent une note pour chacune d'elle.

• Pour chaque étudiant, on veut gérer son nom, prénom, tél, mail, ainsi que son année d'entrée au collège.

• Une matière peut être enseignée par plusieurs enseignants mais a toujours lieu dans la même salle de cours (chacune ayant un nombre de places déterminé).

• On désire pouvoir calculer la moyenne par matière ainsi que par département.

• On veut également calculer la moyenne générale d'un élève et pouvoir afficher les matières dans lesquelles il n'a pas été noté.

• Enfin, on doit pouvoir imprimer la fiche signalétique (nom, prénom, tél, mail) d'un enseignant ou d'un élève.

Question : Elaborez le diagramme de classe (visibilités des attributs et les types des données n'sont pas obligatoires).

Université de Ghardaïa
Faculté des Sciences et Technologies
Département des Mathématiques et de l'Informatique

3ème Année Licence
Module: Génie logiciel
Année Universitaire 2023-2024

Examen Final (Correction)

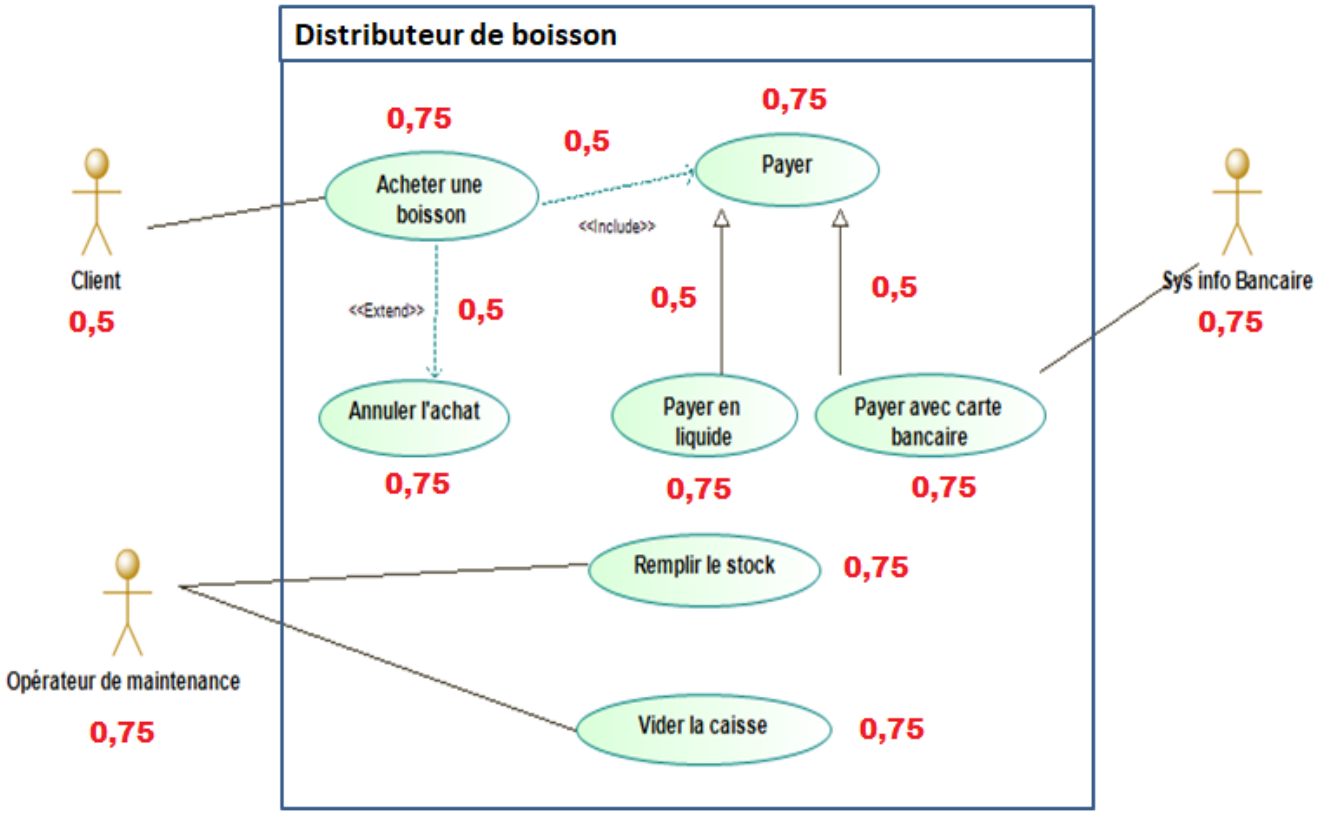
Durée : 01 h 30 30

Questions de cours : (02pts)

1. Diagramme de séquence système (vue **externe**) / séquence objets (vue **interne**). **(01pt)**
2. Diagramme de cas d'utilisation, de séquence ou d'activité. **(01 pt)**

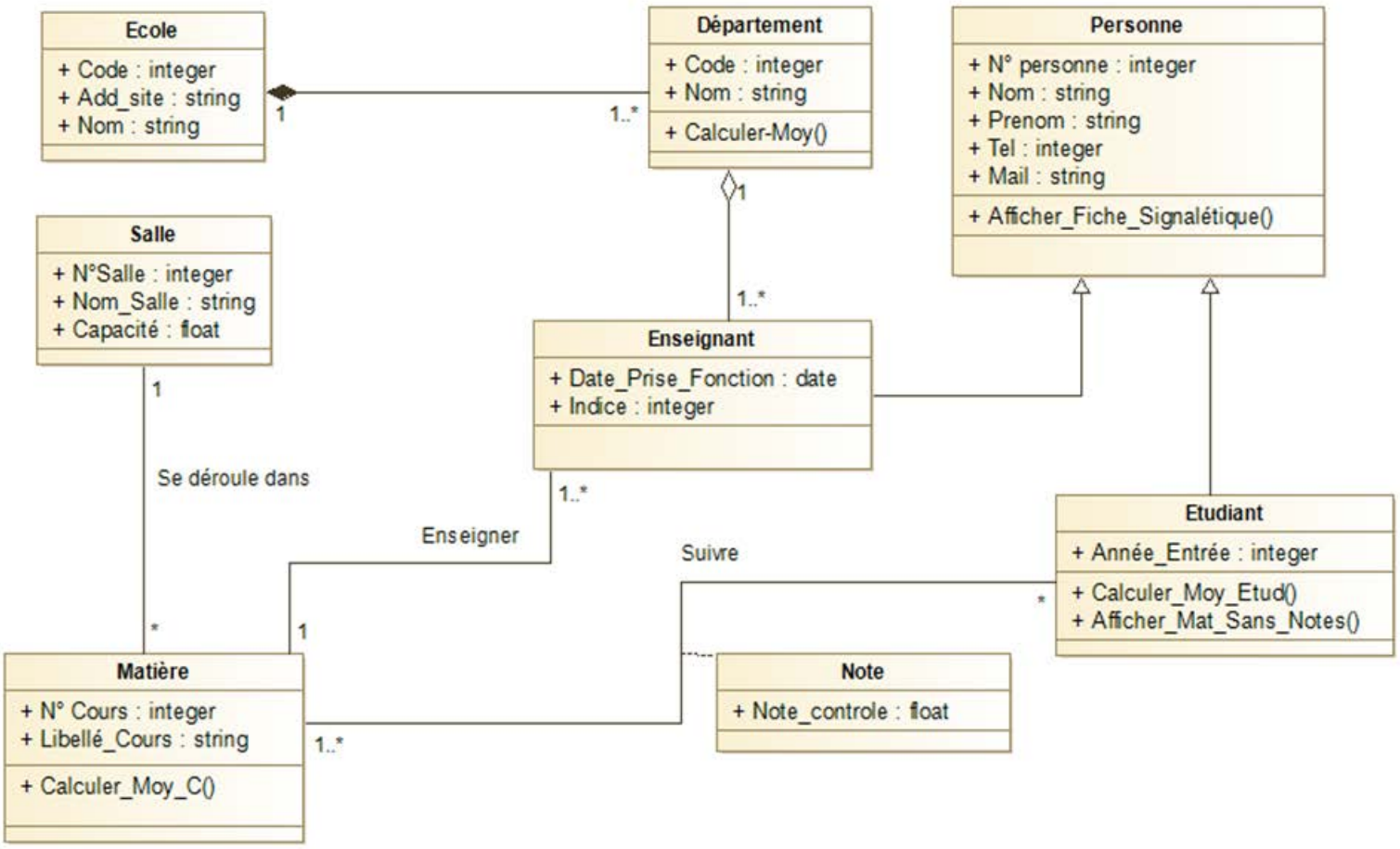
Exercice 1: (10 pts)

Question 1 :



Question 2 : Non, aucune modification n'est nécessaire. **0,75**

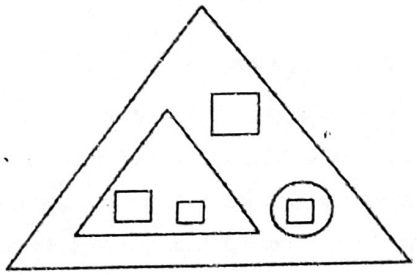
Exercice 2: (08 pts)


Ecole
+ Code : integer
+ Add_site : string
+ Nom : string
1
1..*
Département
+ Code : integer
+ Nom : string
+ Calculer-Moy()
1
1..*
Personne
+ N° personne : integer
+ Nom : string
+ Prenom : string
+ Tel : integer
+ Mail : string
+ Afficher_Fiche_Signalétique()
Salle
+ N°Salle : integer
+ Nom_Salle : string
+ Capacité : float
1
Se déroule dans
*
Enseignant
+ Date_Prise_Fonction : date
+ Indice : integer
1..*
Enseigner
1
Etudiant
+ Année_Entrée : integer
+ Calculer_Moy_Etud()
+ Afficher_Mat_Sans_Notes()
Suivre
*
Matière
+ N° Cours : integer
+ Libellé_Cours : string
+ Calculer_Moy_C()
1..*
Note
+ Note_controle : float

Université de Ghardaïa *Faculté des Sciences et Technologies* *Département des Mathématiques et de l'Informatique* *3ème Année Licence* *Année Universitaire 2023-2024*	*Module: Génie logiciel* **Examen TD** **Durée : 01 h 30** **ممنوع إستعمال الهاتف النقال أثناء الإمتحان**

Exercice n° 01 :(08 points)

Le dessin ci-dessous représente des figures (triangles, carrés ou cercles) emboîtés. Les triangles contiennent une ou plusieurs figures. Les carrés ne contiennent rien. Les cercles contiennent exactement une figure. Les figures possèdent des « côtés ». On dira que les cercles ont un seul côté, les triangles trois côtés et les carrés quatre côtés.



Travail à faire :

1- Modéliser un diagramme de classes correspondant à la figure. Le diagramme comprendra les classes "Figure", "Cercle", "Carré", "Triangle" et "Coté" et des associations à déterminer.

2- Placer les ordres de multiplicité (de cardinalités) sur ce diagramme.

Exercice n° 02 : (12 points)

La société de transport collectif 'DZBus' a fait appel à une entreprise de développement pour informatiser la gestion de son activité. Le système à développer est une application web appelée "LogiLigneBus". La société a exigé que le futur site soit compatible avec le plus grand nombre de navigateurs web sur le marché.

La société 'DZbus' possède un parc de bus et emploie un responsable d'exploitation. Ce dernier devra à travers le système affecter des chauffeurs à différents bus et différentes lignes. Chaque ligne est une suite d'arrêts, pour chaque arrêt, le responsable d'exploitation fixe les heures de passages du bus. Les chauffeurs peuvent consulter et éventuellement imprimer leur activité de la journée. L'activité d'un chauffeur est constituée d'un ensemble de créneaux et de lignes correspondantes affectées.

Les chauffeurs peuvent demander un changement de ligne, pour cela, le chauffeur se connecte au système via son compte et choisit l'opération « émettre une demande de changement ». Le système lui retourne une fiche à remplir. Le chauffeur renseigne la ligne actuelle et choisit entre deux types de lignes : ligne interwilaya ou intrawilaya. Si le type choisi est interwilaya le chauffeur doit choisir la wilaya désirée et doit accompagner sa demande d'une autorisation délivrée par la wilaya. Si la ligne est intrawilaya une liste de lignes est ensuite retournée. Le chauffeur sélectionne la ligne désirée. Le système enregistre sa demande et affiche un message de confirmation.

Dans tous les cas, la demande de changement reste en attente jusqu'à ce qu'elle soit traitée par le responsable d'exploitation qui peut soit accepter ou refuser. Dans le cas de refus, il doit impérativement justifier le refus.

- Tracer le diagramme de cas d'utilisation.

Université de Ghardaïa *Faculté des Sciences et Technologies* *Département des Mathématiques et de l'Informatique* *3ème Année Licence* *Année Universitaire 2023-2024*	*Module: Génie logiciel* **Examen TP** **Durée : 01 h 30** **ممنوع إستعمال الهاتف النقال أثناء الإمتحان**

Exercice n° 01 : (10 points)

Construire **un diagramme d'activité** en utilisant les couloirs d'activités pour décrire la connexion d'un client à un serveur SSH. On considère trois acteurs: le client, le démon SSH (i.e. le serveur logiciel) et la machine serveur. Une fois la connexion établie entre le client et le serveur, le démon demande un mot de passe au client, ce dernier dispose de trois tentatives avant que la connexion ne soit rompue (déconnecté). Les tentatives infructueuses sont enregistrées dans un fichier sur le serveur. Une fois l'identification faite, un terminal est ouvert et l'utilisateur peut alors saisir des commandes qui sont interprétées par le démon et exécutées sur le serveur. La commande exit déconnecte le client du serveur.

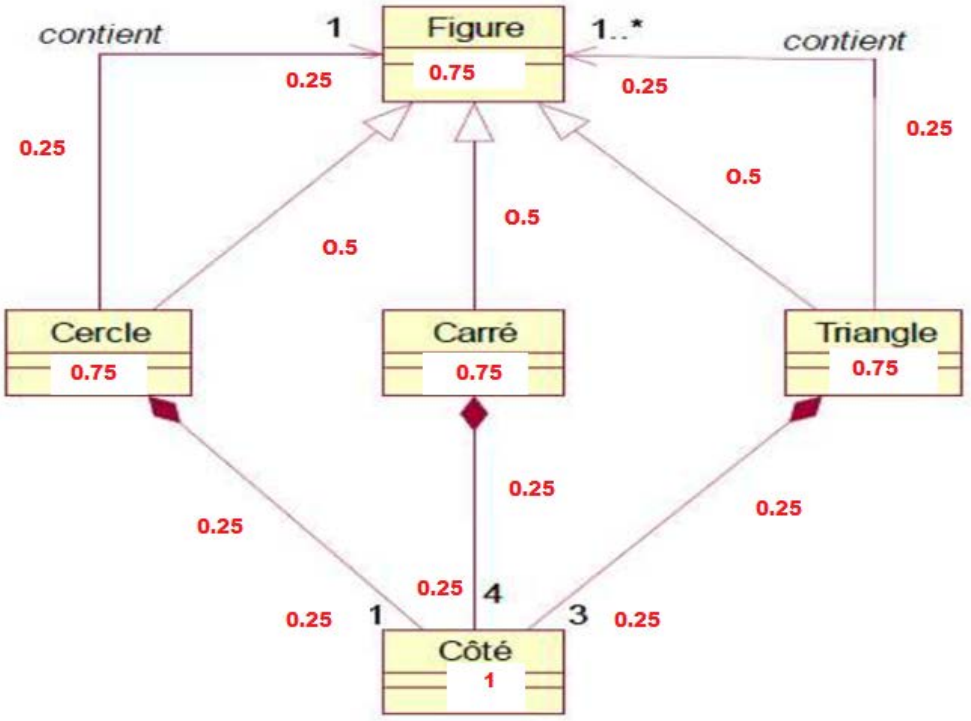
Exercice n° 02 : (10 points)

Soit à représenter **le diagramme d'état-transition** d'un objet personnel en suivant les événements de gestion depuis le recrutement jusqu'à la mise en retraite. Après le recrutement, une personne est considérée en activité dès sa prise de fonctions dans l'entreprise. Au cours de sa carrière, nous retiendrons seulement les événements : congé de maladie et prise de congé annuel. En fin de carrière, nous retiendrons deux situations : la démission et la retraite.

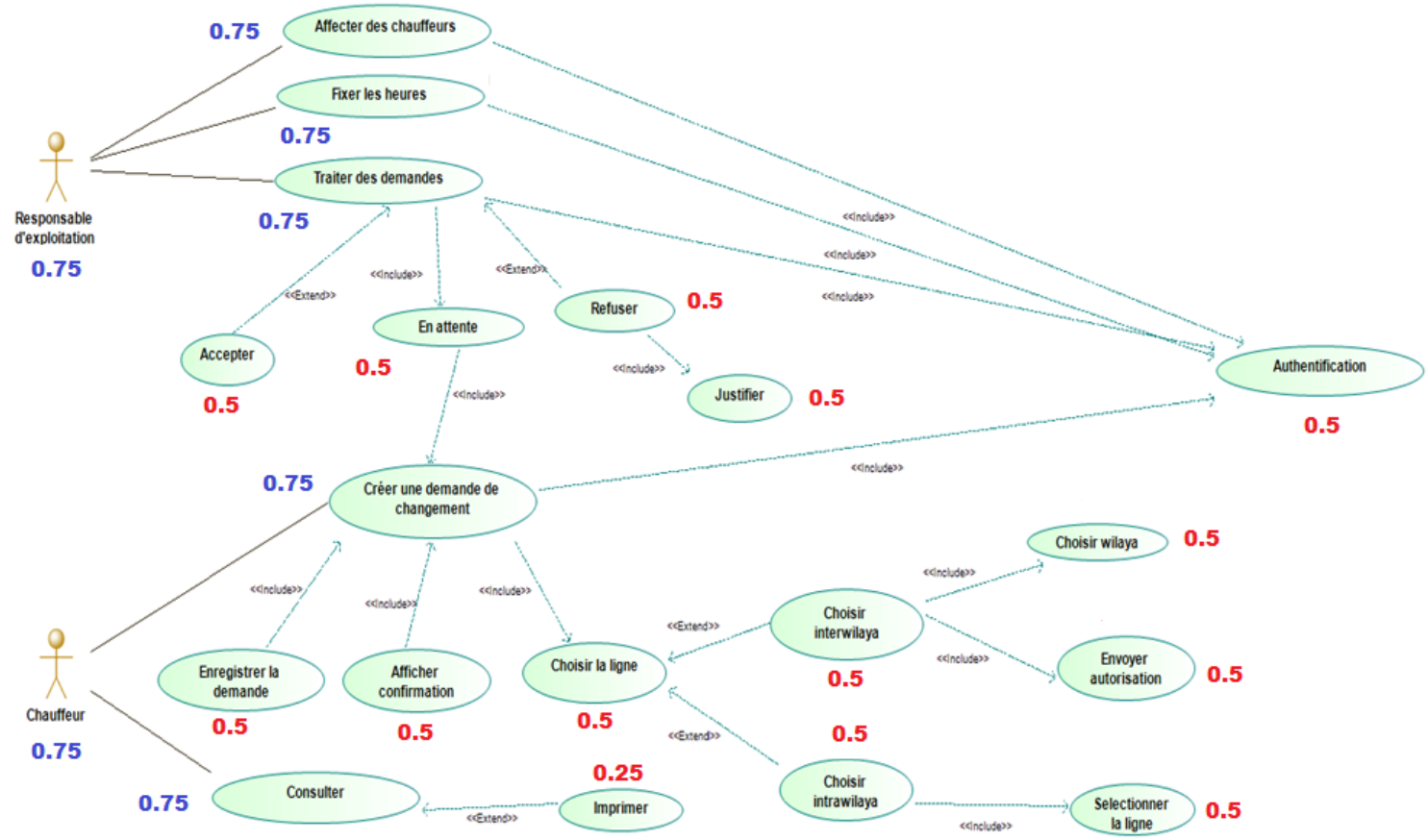
بالتوفيق

Université de Ghardaïa *Faculté des Sciences et Technologies* *Département des Mathématiques et de l'Informatique* *3ème Année Licence* *Année Universitaire 2023-2024*	*Module: Génie logiciel* **Examen TD** **Durée : 01 h 30** **ممنوع إستعمال الهاتف النقال أثناء الإمتحان**

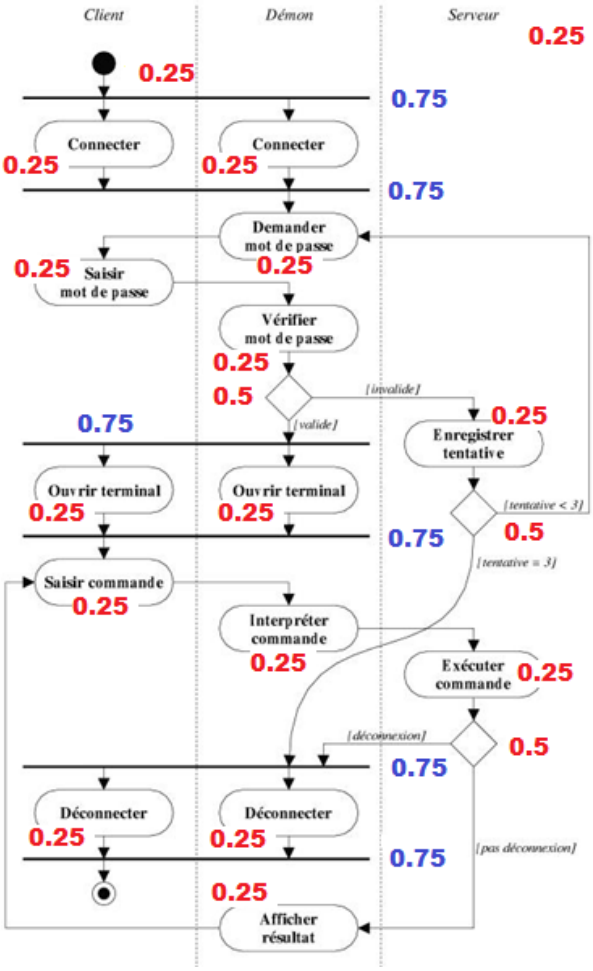
Exercice n° 01 :(08 points)



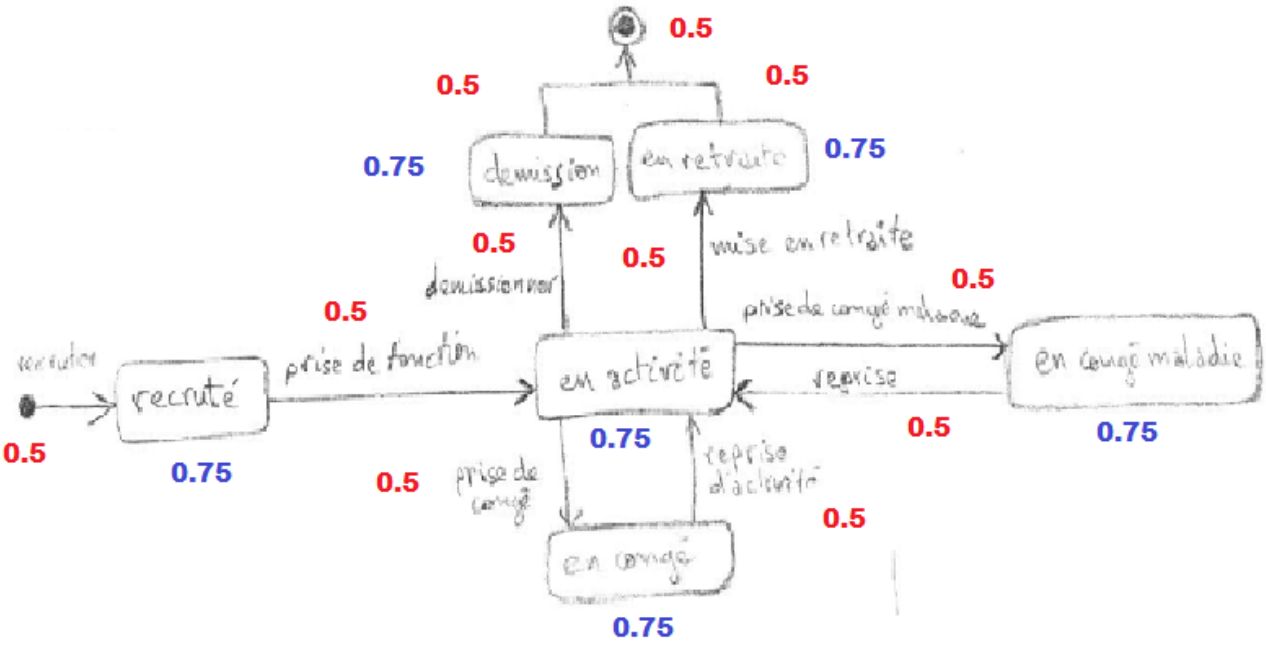
Exercice n° 02 : (12 points)



Exercice n° 01 : (10 points)



Exercice n° 02 : (10 points)



بالتوفيق

Bibliography

[1] Scott Ambler. *The Elements of UML(TM) 2.0 Style*. Cambridge University Press, 2002.

[2] RAJIB MALL. *Fundamentals of software engineering*. PHI, 2014.

[3] Robert C. Martin. *UML for Java Programmers*. Eyrolles, 2003.

[4] Pascal Roques. *UML 2 en action: De l'analyse des besoins à la conception*. Eyrolles, 2011.

[5] Doug Rosenberg. *Use case driven object modeling with UML*. Springer, 1999.

[6] Ian sommerville. *Software Engineering*. Pearson, 2016.