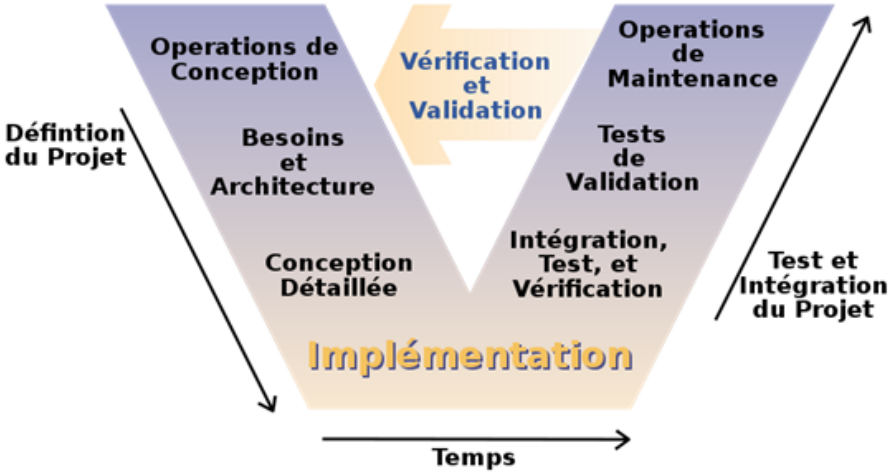
République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université de Relizane
Faculté des Sciences et Technologie
Département d'Informatique

Génie Logiciel

BENOTMANE ZINEB
3^{e} année Informatique
(LMD)



جامعة غليزان
RELIZANE UNIVERSITY

Table des matières

Table des figures

Liste des tableaux

Introduction

Programmer, coder c'est si simple, mais alors pourquoi s'intéresser au génie logiciel ? Pour répondre à cette question, prenons cet exemple : construire une balançoire ou construire un parc d'attraction, ce sont deux tâches complètement différentes ; programmer et développer un logiciel le sont aussi. Ce qu'on peut déduire de cet exemple est que sur des échelles différentes, l'effort fournit et la méthode utilisée sont différents.

Développer un logiciel ne s'arrête pas à l'écriture du code, mais cela consiste à réalise plusieurs étapes en plus de la programmation, comme la collecte des données, la conception et les test,... et donc, nous pouvons définir le Génie Logiciel comme étant une discipline qui vise à appliquer des méthodes scientifiques et des processus rigoureux pour la conception, la création, le test et la maintenance de logiciels.

L'objectif de ce module est de comprendre les principes, les méthodes et les techniques nécessaires pour développer des logiciels de manière efficace et fiable, en respectant la règle du CQFD (Coût, Qualité, Fonctionnalité, Délais).

A la fin de leurs apprentissage, les étudiants auront la capacité d'appliquer la méthodologie d'analyse, de conception et de développement des logiciels. En particulier, ils apprendront la modélisation objet avec le langage universel UML.

Ce polycopié est organisé comme suit :

— Partie cours

— Partie TD

— Partie TP

Fiche technique du module

Établissement : Université de Relizane

Faculté : Sciences et technologie

Département : Informatique

Public cible :troisième année Licence Informatique

Intitulé du cours : Génie Logiciel

Parcours : Systèmes Informatiques

niveau : 3ème année licence Informatique

Unité d'enseignement : Unité fondamentale , UEF521

Semestre : S5

Crédits :5

Coefficients : 2

Durée : 14 Semaines

Modalité d'évaluation : Examen : 40% , CC : 60%

Volume horaire global :63H

Horaire du cours : cours 1H30/semaine ; TD :1H30/Semaine, TP : 1H30/Semaine

Objectifs de l'enseignement : Apprendre à appliquer une méthodologie d'analyse et de conception pour le développement des logiciels. En particulier, apprendre la modélisation

objet avec le langage universel UML.

Connaissances préalables recommandées : Algorithmique, Système d'information, Programmation Orienté Objet.

Contenu de la matière :

Chapitre 1. Introduction

1. Définitions et objectifs

2. Principes du Génie Logiciel

3. Qualités attendues d'un logiciel.

4. Cycle de vie d'un logiciel.

5. Modèles de cycle de vie d'un logiciel.

Chapitre 2. Modélisation avec UML

1. Introduction Modélisation, Modèle, Modélisation Orientée Objet, UML en application.

2. Eléments et mécanismes généraux

3. Les diagrammes UML

4. Paquetages

Chapitre 3. Diagramme UML de cas d'utilisation :Vue fonctionnelle.

Intérêt et définition, Notation

Chapitre 4. Diagrammes UML de classes et d'objets : Vue statique

1. Diagramme de classes

2. Diagramme d'objets

Chapitre 5. Diagrammes UML : vue dynamique

1.Diagramme d'intéraction (séquence et collaboration)

2. Diagramme d'activité.

3.Diagramme d'état transition

Chapitre 6. Autres notions et diagrammes UML

1. Composants, déploiement, structures composite.

2. Mécanismes d'extension : langage OCL + les profils.

Chapitre 7. Introduction aux méthodes de développement : (RUP, XP)

Chapitre 8. Patrons de conception et leur place au sein du processus de développement

Première partie

Cours

Chapitre 1

Introduction au Génie Logiciel

Ce premier chapitre introduit le principe de base du génie logiciel, en commençant par une historique, sa définition et ses objectifs, son principe, les qualités attendues d'un logiciel en terminant par son cycle de vie et les modèles les plus connus.

1.1 Historique

Lorsqu'on parle d'un logiciel, on imagine bien la complexité de sa taille et/ou de sa structure. En général, lorsqu'on développe, on le fait pour d'autres domaines, car avec l'ère de la technologie, plusieurs secteurs ont franchis le pas du manuel vers l'automatisation dans un but économique,social ou concurrentiel, et plusieurs compagnies de développement ont commencé à produire des logiciels.
Cependant, la non qualité des systèmes informatique produits a provoqué beaucoup de risques humains et économiques, c'est ce qu'on appelle "la crise logiciel" , nous citons quelques exemples célèbres suivants :

— arrêt de Transpac pour 7.000 entreprises et 1.000.000 d'abonnés : surcharge du réseau,

— En mars 1993, la bourse de Londres a renoncé au projet informatique Taurus qui devait assurer complètement le suivi de l'exécution des transactions. Le système avait coûté directement 60 millions de livres et les opérateurs sur le marché avait dépensé 400 millions de livres pour y adapter leurs propres logiciels.

— mission VENUS : passage à 500.000 km au lieu de 5.000 km à cause du remplacement d'une virgule par un point,
— Le Therac-25, un appareil d'irradiation thérapeutique a provoqué la mort de 2 personnes, et l'irradiation de 4 autres, à cause d'une erreur logicielle.
— Un navire de guerre anglais a été coulé par un Exocet français, au cours de la guerre des Malouines. Bilan de la catastrophe : plusieurs centaines de morts. Le vaisseau anglais n'avait, en effet, pas activé ses défenses, l'Exocet n'étant pas répertorié comme missile ennemi.
— En 1983, toute la vallée du Colorado a été inondée. La cause ? Une mauvaise modélisation dans le logiciel du temps d'ouverture du barrage.
— Une centenaire (106 ans) s'est retrouvée convoquée à l'école. La cause ? le codage de l'année de naissance sur deux caractères.
— Bug de l'an 2000. Coût de la mise à niveau des logiciels à la France : 500 milliards de francs.
— Une victime de l'an 2000 s'est vue adresser une amende de 91 500 $ au retour d'une cassette vidéo louée. La raison ? Le retard calculé étant de cent ans car là encore le codage de l'année des emprunts avait été effectué sur deux caractères, pour gagner un peu de place.

Le Génie logiciel est apparu dans les années 70 pour répondre à cette crise lorsque l'on a pris conscience que le coût du logiciel dépassait le coût du matériel, souvent, il le dépasse très largement.

1.2 Définition

Le génie logiciel est une discipline qui a pour but la fabrication de logiciel sans fautes, livré dans les délais et le budget prévu à l'avance et satisfaisant les besoins du client.

1.3 le maître d'ouvrage

le maître d'ouvrage est une entité ou un individu qui définit les besoins et les exigences pour un projet de développement logiciel. C'est le client, le propriétaire ou l'utilisateur final du produit logiciel. Il est responsable de la définition des objectifs du projet, de l'identification des besoins des utilisateurs et des parties prenantes, et de la communication des exigences fonctionnelles et non fonctionnelles au chef de projet et à l'équipe de développement. Le maître d'ouvrage est également responsable de valider les résultats du projet et de s'assurer que le produit final répond aux attentes et aux besoins des utilisateurs. Une bonne communication et une collaboration étroite entre le maître d'ouvrage et l'équipe de développement sont essentielles pour assurer le succès du projet.

1.4 Qualité d'un logiciel

Souvent, le développeur se retrouve face à des problèmes liés à la qualité des logiciel, parmi les facteurs de non qualité, nous citons :

- Mauvaise spécification : cela peut être causé par des informations incomplètes ou vagues, un changement d'avis, un oubli de la part du client et/ou de l'informaticien.
- Mauvaise estimation du délai.
- Mauvaise répartition des tâches qui doit dépendre de la spécialité de chaque participant.
- Mauvais suivi de l'équipe : pas de contrôle, d'encadrement et de réunions.
- Problème technique : il est généralement lié à la mauvaise programmation et l'insuffisance de test.
- Problème humain : dû au manque de communication, de l'esprit d'équipe.

Pour éviter tous ces problème, la norme ISO 9126 a résumé les critère pour avoir un logiciel de qualité, nous citons :

- Fiabilité : c'est la confiance que peut avoir un utilisateur par rapport au logiciel, par exemple, si ce dernier envoi un message, est-il sûr de son arrivée à destinataire.
- Ergonomie : il s'agit de la facilité d'utilisation avec ou sans formation, quelque soit

la compétence, on parle aussi de facilité d'apprentissage et l'effort fourni pour la manipulation.

— Extensibilité : est la facilité d'adaptation aux changements.
— Intégrité : protéger les données contre les attaques.
— Réutilisabilité : écrire un code de sorte de pouvoir le réutiliser dans d'autre projets logiciel.
— Portabilité : la capacité d'être porté sur plusieurs environnements.
— Maintenabilité : la capacité du logiciel de récupérer les données en cas d'erreur.

1.5 cycle de vie d'un logiciel

Le mot "cycle de vie" résume les différentes phases du logiciel démarrant de l'idée jusqu'à son utilisation.
Découper la vie d'un logiciel en phases à pour objectif de maitriser les risques, les délais et les coûts et contrôler la qualité. Un logiciel doit être conforme aux attentes et exigences des futurs utilisateurs.
Le processus de développement d'un logiciel est l'ensemble d'activités ordonnées à suivre pour produire un logiciel qui répond à la règle de CQFD (Coût, Qualité, Fonctionnalité, Délai).
Une activité à un nom, une description et une manière d'être réalisée. Elle possède une entrée et une sortie ; l'entrée peut parvenir d'une autre activité, et la sortie peut servir pour entrée d'une autre activité.
les différentes activités sont :

— Faisabilité.
— Spécification : collecte des exigences, ie, ce qu'on attend de ce logiciel ainsi que les besoins du client.
— Organisation du projet : de quoi a-t-on besoin pour réaliser ce projet, cela comprend l'analyse des coûts, la planification, assurance qualité, la répartition des tâches.
— Conception.
— Implémentation : il s'agit de la programmation et l'écriture du code source.

— Test.

— Livraison : installation, formation, assistance.

— Maintenance.

1.5.1 Faisabilité

c'est une étude de marché, ie, la peine de faire ce logiciel ou non. Avant de démarrer tout projet logiciel, il est important de répondre aux questions suivante : pourquoi et pour qui développe-t-on ce logiciel ? y a t-il un marché pour cela, quel est le budget, les moyens ? le personnel, est-il qualifié ? tout cela mène à répondre à la question finale : on développe ou pas ?

1.5.2 Spécification : Analyse des besoins

Cette activité écrit les conditions que doit remplir le logiciel (Quoi ?)
L'étape de l'analyse des besoins est une étape très importante pour démarrer un projet, pour cela cette section est nécessaire pour permettre au développeur d'étudier le domaine d'application et l'état actuel de l'environnement du futur logiciel afin de déterminer les ressources disponibles et requises, les contraintes d'activation et de performance,etc..., dans le but d'éviter de développer un logiciel non adéquat.

Comment faire l'analyse des besoins ?

Pour faire une bonne analyse des besoins, il est recommandé de suivre les points suivants :

— Ecouter le client.

— Préparer des questions et des réunions.

— Lecture des documents disponibles.

— Préparer des questions pertinentes.

— Connaitre le rôle de chaque futur utilisateur du système.

— Penser à tous les problèmes possibles

Exemples de questions à poser :

— Qui est derrière la demande de cette réalisation ?
— Qui va utiliser la solution proposée ?
— Quel sera l'environnement de la solution ?
— A qui d'autres m'adresser ?
— ....

Les questions doivent être complètes et doivent inclure tous les domaines en questions, cela permet une bonne analyse afin de maximiser la satisfaction des futurs utilisateurs. Nous citons deux types de besoins :

— Les besoins fonctionnels : explicitement établis.
— Les besoins non fonctionnels : attendus, non exprimés mais nécessaires.

Cahier de charges

Le cahier de charges contient des descriptions globales des fonctions d'un nouveau produit ou des extensions d'un produit existant, il doit être validé par le client. Il est la base du contrat entre le client et le développeur. le modèle d'un cahier de charge est le suivant :

1. La couverture :
   — Nom du projet/ logiciel/ produit ;
   — Date ;
   — Numéro de version ;
   — Acteurs ;
   — Responsabilité de chaque acteur ;
   — Les changements clé depuis la version précédente ;
2. Introduction : Elle représente :
   — le résumé qui contient les objectifs ;
   — la fourniture : liste de ce qui est livré au client (logiciel, matériel,...) ;
   — les définitions et acronymes.
3. Organisation du projet :
   — Décomposition du projet dans le temps ;
   — Le rôle de chaque acteur du développement ;

— Les limites ;
— Les interactions avec un matériel ou logiciel déjà existant.

4. La gestion :
   — Définir les objectifs et leurs priorités ;
   — Définir les dépendances avec d'autres systèmes ;
   — Identifier les contraintes.
   — Gestion des risques.
5. Techniques : Il s'agit de la description des méthodes et outils employés.
6. Calendrier et budget : il s'agit de découper le projet en lots (livraisons intermédiaires, paiement intermédiaires,...), ainsi que les ressources, le budget et échéancier et tout ce qui peut influer sur le calendrier.
7. Besoins fonctionnels.
8. Besoins non fonctionnels.
9. Contexte du futur système.

1.5.3 Conception

Il s'agit de l'étape où on se pose la question suivante « Comment fonctionne le logiciel », elle comprend : la conception globale et la conception détaillée.
La conception globale : est la description architecturale en composants : les interfaces et leur fonctionnalités.
La conception détaillée : on rajoute des détails sur le fonctionnement des composants.

1.5.4 Test

: Les tests permettent de savoir si le logiciel fonctionne correctement, il existe plusieurs types de tests :

1. Test unitaire : le logiciel est testé par ses développeurs.
2. Test système : le logiciel est testé dans un environnement similaire à celui du client.
3. Test alpha : faire tester le client sur le site du développeur.

4. Test béta : Tester sur l'environnement du client.
5. Test d'acceptation : est-ce que le client est satisfait.
6. Test de régression : on enregistre tous les tests et tous les résultats obtenus afin de les comparer aux autres versions.

1.5.5 Maintenance

Elle comporte la maintenance perfective (suite à un changement dans la spécification), Corrective (en cas d'erreur), adaptative (en cas de changement d'environnement).

1.5.6 Documentation liées à la fabrication du logiciel

— Cahier des charges : Description initiale des fonctionnalités désirées.
— Spécification (Analyse des besoins).
— Calendrier du projet : décrit l'ordre des tâches, délais ressources,..
— Plan test du logiciel : décrit les procédures de test.
— Conception du logiciel : décrit la structure du logiciel (Comment ?)
— Plan d'assurance qualité : quelles sont les activités mises en œuvre pour assurer la qualité.
— Manuel d'utilisation.
— Code source.
— Rapport des tests.
— Rapport des défauts.

1.6 Modèle de cycle de vie d'un logiciel

Un modèle permet d'assurer une bonne compréhension du fonctionnement du système et de maîtriser la complexité de celui-ci.

1.6.1 Le modèle de cycle de vie en cascade

Ce modèle a été formalisé dans les années 70, (Figure 1.1). Dans ce modèle, chaque phase se termine après un délais fixé. A la fin de chaque étape, des documents ou alors des logiciels sont produits. Ces résultats sont revisités de façon approfondie et on ne passe à la phase suivante que si ces résultats sont jugés satisfaisant. Après quelques modifications du modèle en cascade original, les possibilités de retour arrière ont été ajoutés.

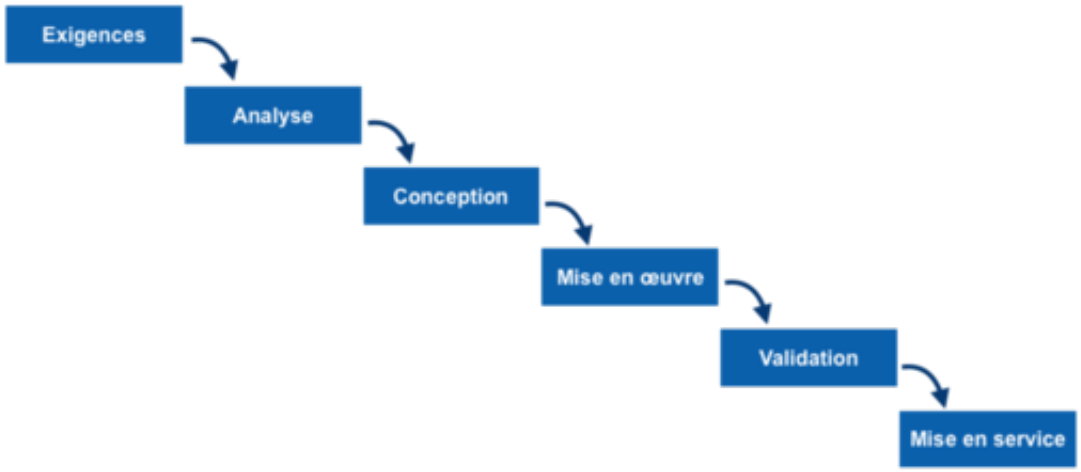


FIGURE 1.1 – Le modèle en cascade

1.6.2 Le modèle en V

Le modèle en V (Figure 1.2) est une autre façon de présenter une démarche qui reste linéaire. Le V est parcouru de gauche à droite en suivant la forme de la lettre : les activités de construction précèdent les activités de validation et vérification. Mais l'acceptation est préparée dès la construction (flèches de gauche à droite). Cela permet de mieux approfondir la construction et de mieux planifier la *remontée*.

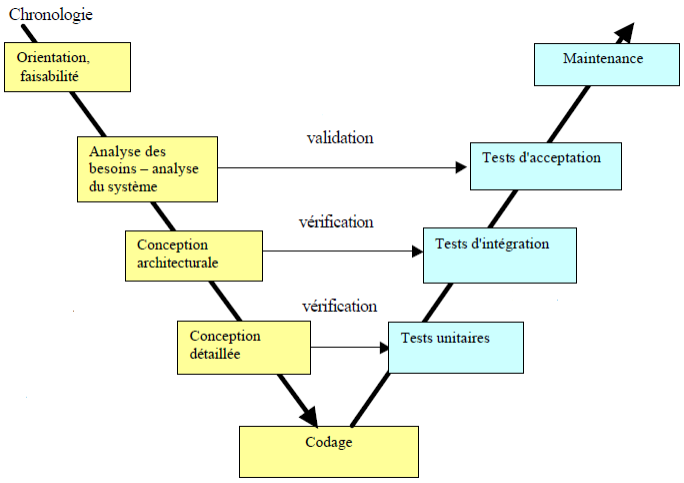


FIGURE 1.2 – Le modèle en V

1.7 Conclusion

Dans ce chapitre, nous avons présenté les notions du génie logiciel en passant en revue les différentes phases de son cycle de vie. Nous avons aussi expliqué l'importance du génie logiciel pour les développeurs à travers les avantages de découper le cycle de vie du logiciel en plusieurs étapes.

Chapitre 2

UML : Unified Moddeling Language

Pour construire un logiciel, on a besoin de comprendre de quoi il se compose et comment et à quel moment il interagit avec l'utilisateur, d'où le besoin de modéliser. Comme dans les plans d'architecture de maisons par exemple, plusieurs niveaux doivent être modélisés selon plusieurs vues ; et pour cela, nous avons besoin d'un langage unifié.

2.1 pourquoi modéliser ?

Modéliser un système avant sa réalisation permet de mieux comprendre le fonctionnement du système, c'est également un bon moyen de maîtriser sa complexité et d'assurer sa cohérence.
Un modèle est un langage commun, précis, compréhensif grâce à son côté graphique, à ce titre, c'est un moyen efficace pour privilégier la communication car elle est essentielle pour aboutir à une compréhension commune aux différentes parties prenantes du projet à réaliser.
Un modèle permet aussi de mieux répartir les tâches et d'automatiser certaines d'entre elles.
C'est également un facteur de réduction des coûts et des délais, il permet aussi d'assurer un bon niveau de qualité et une maintenance efficace.
En résumé, le résultat final dépend de la modélisation du projet.
Pour le génie logiciel, on modélise aux étapes de spécification et de conception.

2.2 Modélisation Orientée Objet

Le modèle du système dans les premières phases de ce processus est nécessairement une simplification du système réel. Le processus de modélisation vise à mieux cerner les limites du système à réaliser. Ensuite, les modèles sont raffinés de plus en plus pour aboutir au code.

La modélisation Orienté Objet permet une simplification du monde réel. Cette dernière répond à la problématique qui consiste à trouver le bon niveau d'abstraction des concepts du monde réel.

Un objet est une entité structurée caractérisée par des attributs et des méthodes.

L'approche orientée objet est une façon d'aborder un problème et de le découper en petits sous-problèmes : On commence par rechercher les objets du système puis leurs interactions, par exemple, la création d'un compte bancaire nécessite des interactions entre le client et un conseiller client.

2.3 les Languages

Un langage est un moyen de communication utilisant des lettres, des chiffres, du graphisme, ...

Il existe différents types de langages :

- Langage Informel : ces langages sont soit naturel structurés utilisant un modèle de documents précis avec des règles de rédaction, soit Pseudo-code utilisant une description algorithmique d'une tâche.
- semi-formel : il s'agit d'une notation graphique utilisant des diagrammes accompagnés de textes structurés offrant une vue statique et dynamique du système.
- Langage formel : ils sont précis s'appuyant sur des formalismes mathématique

A l'époque , il y avait à peu près une cinquantaine de langages de modélisation, il était impossible de comprendre tous les langages. En 1995 : les langages les plus utilisés ont été Unifiés pour donner naissance à UML.

2.4 UML : Unified Modeling Language

UML : Unified Modeling Language, est un langage graphique qui permet de représenter et de communiquer différents aspect d'un système d'information. C'est la notation de modélisation la plus répandue dans le monde.
UML est un langage de modélisation orienté objet, c'est-à-dire que toutes les entités modélisées sont des objets ou se rapportent à des objets.
UML est devenu le standard de l'industrie.
L'avantage est que étant graphique, UML permet de visualiser le système réalisé ; le modèle est divisé en vues sélectionnant les éléments pertinents puis en diagrammes de différents types. L'aspect graphique de UML retient le premier l'attention de ses utilisateurs.

2.5 Diagrammes UML

UML comporte treize diagrammes différents, chacun d'eux étant dédié à la représentation des concepts particuliers d'un système logiciel. Par ailleurs, UML modélise le système suivant deux modes de représentation :
1- Le premier mode représentant la structure statique du système.
2- Le deuxième mode représente la dynamique de fonctionnement.
ces deux représentations sont nécessaires et complémentaires pour schématiser la façon dont est composé le système et comment ses composantes fonctionnent entre elles. Les différents diagrammes UML sont présentés dans cette section et récapitulés dans la figure (Figure 2.1).

2.6 Les vues dans UML

En UML (Unified Modeling Language), il existe trois vues principales pour représenter un système informatique : la vue fonctionnelle, la vue statique et la vue dynamique.

La vue fonctionnelle : Cette vue décrit les fonctions et les services que le système doit fournir. Elle est représentée par des diagrammes de cas d'utilisation, qui montrent les

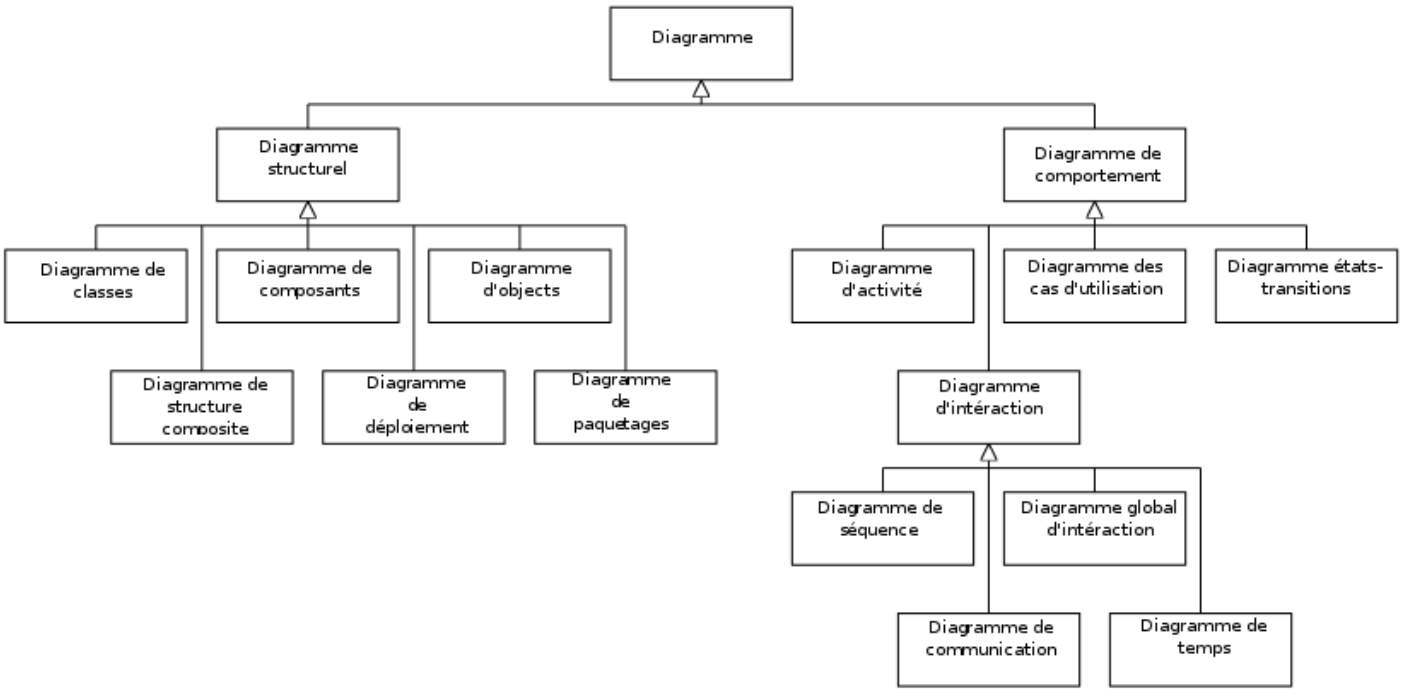


FIGURE 2.1 – Les Diagrammes UML

acteurs (utilisateurs) et les cas d'utilisation (actions) qu'ils peuvent effectuer. Cette vue est importante pour comprendre les besoins des utilisateurs et définir les fonctionnalités à implémenter.

La vue statique : Cette vue décrit la structure statique du système, c'est-à-dire les entités (objets, classes) qui composent le système et les relations entre elles. Cette vue est représentée par des diagrammes de classes, d'objets ou de composants. Cette vue est importante pour comprendre la structure du système et faciliter sa conception et sa maintenance.

La vue dynamique : Cette vue décrit le comportement dynamique du système, c'est-à-dire les événements qui se produisent et les interactions entre les entités. Cette vue est représentée par des diagrammes de séquence, d'états ou d'activités. Cette vue est importante pour comprendre comment le système fonctionne et comment il réagit à différentes situations.

Un système informatique peut aisément contenir des centaines de classes ou d'éléments de modélisation. Pour gérer cette complexité, UML fournit le concept de paquetage (en anglais, package) qui organise un espace de nommage.

2.7 Diagramme de paquetage

Comme nous l'avons mentionné précédemment, nous passons par plusieurs étapes pour réaliser un logiciel, notamment l'analyse des besoins, qui sera organisée en plusieurs sections : Étude de l'existant, Critique de l'existant, Besoins fonctionnels et non fonctionnels, ...), à partir de cette analyse, nous pouvons commencer à découper le système pour mieux le cerner er le modéliser.

Un diagramme de paquetage sert à découper un système en sous systèmes pour mieux le comprendre et mieux le modéliser comme l'exemple illustré dans la figure 2.7, où on souhaite réaliser un logiciel pour un hôtel : on divise notre système en plusieurs paquetages :

— Gestion des clients

— Gestion des employés

— Gestion des statistiques

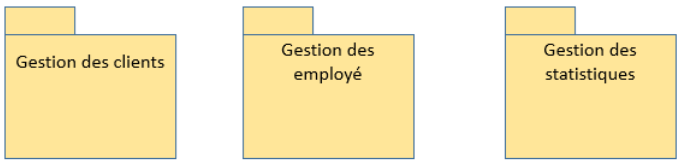


FIGURE 2.2 – Diagramme de Paquetage

Le paquetage contient des classes et pour établir les relations entre les paquetages, il suffit de reporter les relations entre les classes qui composent chacun.

Dans un diagramme de paquetage, il existe trois types de dépendances :

— B ¡¡import¿¿ A : B importe tous les éléments publiques de A.

— B ¡¡access¿¿ A : B peut accéder à tous les éléments publiques de A.

— B ¡¡merge¿¿ A : fusionner les deux paquetages.

2.8 Conclusion

La modélisation est l'étape intermédiaire entre la récolte des besoins du client et la réalisation du logiciel.

dans ce chapitre, nous avons présenté un des langages de modélisation les plus connus, il s'agit d'UML.

Chapitre 3

Diagramme de cas d'utilisation

Le maître d'ouvrage et les futurs utilisateurs du logiciel ne sont pas informaticiens dans la majorité des cas, et ont besoin d'un moyen pour exprimer leurs exigences d'une part ; d'une autre part l'informaticien a lui aussi besoin d'un outil pour comprendre leurs demande et attentes. Le diagramme de cas d'utilisation (use case en Anglais) permet de décrire de manière claire et concise les besoins et les fonctionnalités du système, en se concentrant sur les besoins de l'utilisateur.
Pour élaborer un diagramme de cas d'utilisation, on se base sur les entretiens établis avec les futurs utilisateurs.
L'avantage du diagramme de use case est qu'il se base (comme tout diagramme UML) sur la notation graphique qui permet de mieux comprendre le système.
Afin de dessiner un diagramme de use case, on pose la question suivante : "Que peut-on faire dans ce système ?"

3.1 Les éléments d'un diagramme de cas d'utilisation

la figure 1.2 représente les différents éléments d'un diagramme de cas d'utilisation.

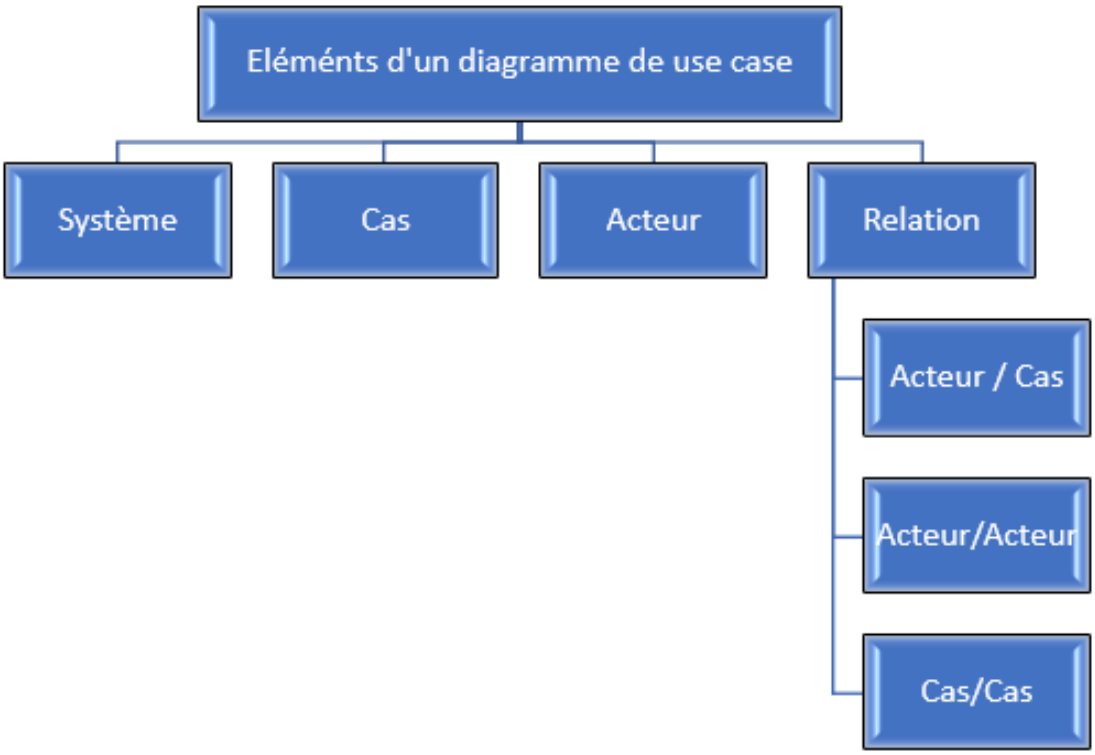


FIGURE 3.1 – Les éléments d'un diagramme de cas d'utilisation

3.1.1 Système :

Le système est l'élément central du diagramme de cas d'utilisation, car il représente l'ensemble des fonctionnalités et des processus du système que les acteurs peuvent utiliser pour atteindre leurs objectifs.
Le système est représenté par un rectangle avec un libellé qui entoure l'ensemble des cas d'utilisation et des acteurs associés. Ce rectangle délimite par rapport à son environnement et aux autres systèmes.

3.1.2 Acteur :

Un acteur représente le rôle joué par une personne ou un système externe interagissant avec notre système.
L'acteur est représenté par un stickman s'il s'agit d'un humain ou un classeur s'il s'agit d'un système ou matériel, avec son nom en dessous.

le nom de l'acteur est son rôle dans le futur logiciel (3.2).
L'acteur peut être principale ou secondaire, dans le premier cas, ce type d'acteur a besoin du système, et agit directement sur ce dernier, dans le deuxième cas, pour la réalisation d'un certain use case, on a besoin de l'intervention de cet acteur. L'acteur secondaire et mis en général, à droite du système.

FIGURE 3.2 – Types d'acteurs

3.1.3 Cas d'utilisation

Il s'agit d'une ellipse avec une phrase verbale. Il modélise une fonctionnalité ou un service rendu par le système. Une fonction est un verbe à l'infinitif.

3.1.4 Relation

Dans un diagramme de cas d'utilisation, les relations sont utilisées pour décrire les interactions entre les acteurs et les cas d'utilisation. Elles mettent en évidence la façon dont les acteurs interagissent avec les différents cas d'utilisation et les liens entre ces derniers. Voici quelques relations couramment utilisées dans un diagramme de cas d'utilisation :

Association : Une association est utilisée pour représenter une relation de base entre un acteur et un cas d'utilisation. Elle indique que l'acteur participe au cas d'utilisation mais n'est pas directement impliqué dans son déclenchement ou son achèvement.

Inclusion : L'inclusion est utilisée lorsque plusieurs cas d'utilisation partagent une séquence d'actions commune. Elle permet de modéliser une relation de dépendance entre les cas d'utilisation. L'inclusion est représentée par une flèche pointant vers le cas d'utilisation inclus et indiquant la direction de l'inclusion.

Extension : L'extension est utilisée pour modéliser une relation dans laquelle un cas d'utilisation peut être étendu par un autre cas d'utilisation facultatif. L'extension permet d'ajouter des fonctionnalités supplémentaires à un cas d'utilisation de base en fonction de certaines conditions. Elle est représentée par une flèche pointant vers le cas d'utilisation d'extension et indiquant la direction de l'extension.

Généralisation : La généralisation est utilisée lorsque plusieurs acteurs ou cas d'utilisation partagent des caractéristiques communes. Elle permet de modéliser une relation de spécialisation-généralisation. Un acteur ou un cas d'utilisation général peut avoir des sous-acteurs ou des sous-cas d'utilisation spécialisés. La généralisation est représentée par une flèche avec un triangle creux pointant vers l'acteur ou le cas d'utilisation général.

3.2 Exemple d'un diagramme de cas d'utilisation

Soit un système de gestion de cours à distance qui fonctionne comme suit :

Un enseignant ou le chef de département peut ajouter/supprimer ou modifier un cours. Le cours ajouté prend un des deux formats : pdf ou MP4.

Le chef de département peut intégrer une liste d'étudiants à partir de la plateforme Etudiants (qui est un système externe).

L'étudiant peut consulter le cours et le télécharger s'il le souhaite.

Proposition du diagramme de use case pour le système de gestion de cours à distance :

Pour la réalisation du use case, on extrait, à partir du texte, les limites du système, les acteurs, les fonctionnalités et les relations, le Diagramme de use case correspondant est représenté dans la figure 3.2.

Système : Gestion des cours à distance.

a) Acteurs : Enseignant, Chef de département, Etudiant, Plateforme étudiants (secondaire).

Fonctionnalités :

— Ajouter un cours.
— Supprimer un cours.
— Modifier un cours.
— Intégrer la liste des étudiants.
— Consulter cours.
— Imprimer cours.

Relations :

— Association : Chaque relation entre un acteur et un cas veut dire que ce dernier la déclenche.
— Include : Pour ajouter, modifier ou supprimer un cours l'enseignant doit s'authentifier.
— Héritage Acteur-Acteur : Chef de département Hérite de enseignant car il peut réaliser tout ce que réalise un enseignant plus ses propres tâches.On évite de mettre une association entre le chef de département et les fonctionnalités héritées.
— Héritage UC-UC : le cours ajouté est soit sous format pdf soit MP4.
— Extende : le téléchargement est une option, l'étudiant consulte le cours et peut s'il le souhaite le télécharger.
— le cas "intégrer la liste" a besoin de l'acteur secondaire "Plateforme étudiants" pour se réaliser.

3.3 Description textuelle du cas d'utilisation ≪ réserver salle ≫

La description textuelle d'un cas d'utilisation est un document qui décrit de manière détaillée et compréhensible les différentes étapes et actions impliquées dans l'utilisation d'un système ou d'une application. Il fournit une vue d'ensemble claire du scénario d'utilisation, en mettant l'accent sur les objectifs, les acteurs impliqués, les préconditions et les résultats attendus.

La description textuelle d'un cas d'utilisation comprend généralement les éléments

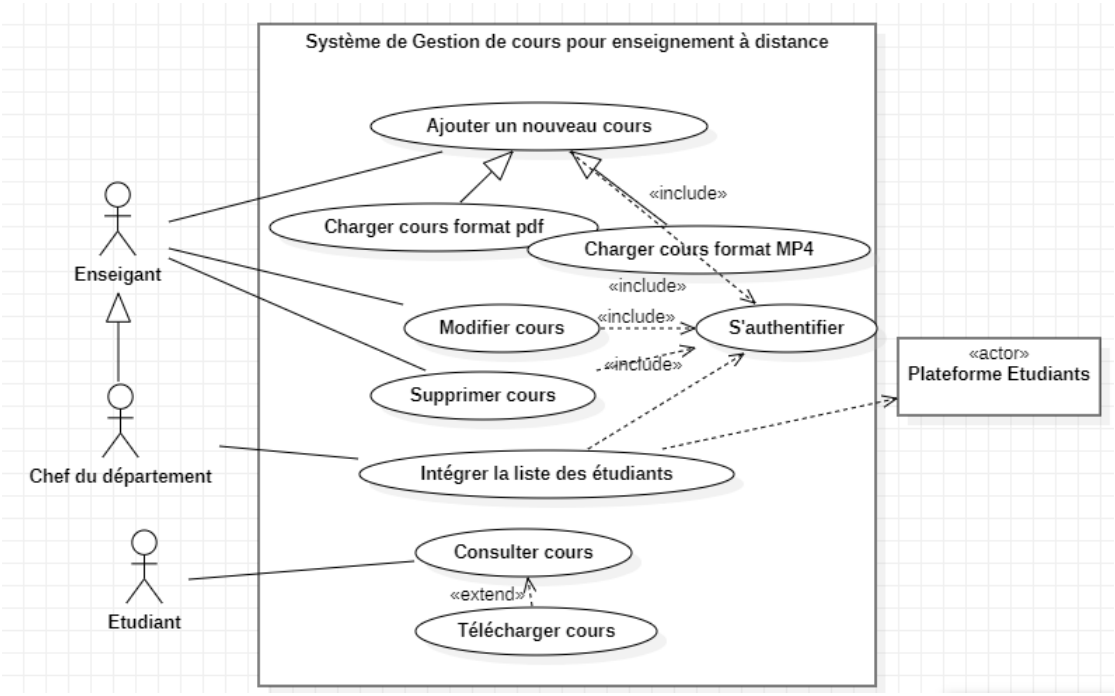


FIGURE 3.3 – Use case : Gestion de cours à distance

suivants :

- Titre : Le titre identifie de manière concise le cas d'utilisation spécifique.
- Acteurs : Les acteurs sont les personnes, les systèmes externes ou les entités qui interagissent avec le système ou l'application pour atteindre un objectif spécifique.
- Préconditions : Les préconditions décrivent l'état initial requis du système ou de l'application avant que le cas d'utilisation puisse être exécuté avec succès. Cela peut inclure des conditions telles que la connexion à un réseau, l'existence de données spécifiques ou l'accès à certaines fonctionnalités.
- Flux principal : Le flux principal décrit séquentiellement les étapes principales du scénario d'utilisation. Il détaille les actions spécifiques effectuées par les acteurs et les réponses du système à ces actions.
- Extensions : Les extensions décrivent les variations ou les chemins alternatifs du flux principal. Ils identifient les points de décision où des actions différentes peuvent être prises en fonction de certaines conditions.
- Postconditions : Les postconditions décrivent l'état final attendu du système ou de l'application après l'exécution réussie du cas d'utilisation. Cela peut inclure des

changements d'état, des résultats affichés ou des actions spécifiques déclenchées.

— Exceptions : Les exceptions sont des situations exceptionnelles qui peuvent se produire pendant l'exécution du cas d'utilisation. Ils décrivent les problèmes potentiels ou les erreurs qui peuvent survenir et comment ils doivent être gérés.

La description textuelle d'un cas d'utilisation est souvent utilisée comme référence pour les développeurs, les testeurs et les parties prenantes pour comprendre le fonctionnement et les exigences du système ou de l'application. Elle favorise également une communication claire et cohérente entre les différentes parties impliquées dans le développement et l'utilisation du système.

Soit le diagramme de use case représenté dans la figure 3.3.

La description textuelle correspondante est la suivante :

I.Identification Cas n∘1 Nom : réserver salle

Acteur : Enseignant ;

Description : la réservation d'une salle s'effectue par un enseignant après la vérification de la disponibilité).

Pré-condition : on doit vérifier la disponibilité de la salle.

Démarrage : l'enseignant clique sur le bouton réserver salle.

II. Description des scénarios

Scénario nominal

1) l'utilisateur demande la réservation d'une salle de cours.

2) le système affiche la page de réservation de salle.

3) L'utilisateur saisie une salle et un jour pour la réservation de la salle.

4) Le système vérifie la disponibilité de la salle (voir CU vérifier disponibilité.

5) Le système demande la confirmation de la réservation.

6) L'utilisateur confirme la réservation.

7) Le système affiche un message du bon déroulement de l'opération.

Scénarios alternatifs

3. a. L'utilisateur saisie un numéro de salle incorrect.

3. b. le système affiche un message d'erreur retour à 3.

5. a. L'utilisateur peut choisir de ne pas valider la réservation (fin du CU en échec.

Scénarios d'exception

1) Coupure de courant avant la confirmation de la réservation

– Les réservations restent sauvegardées pendant 24h.

(c) 3 : Fin et post-conditions

- Fin : scenario nominal (7), scenarios alternatifs (5.a).
- Post-conditions : la salle doit être réservée pour cette date.

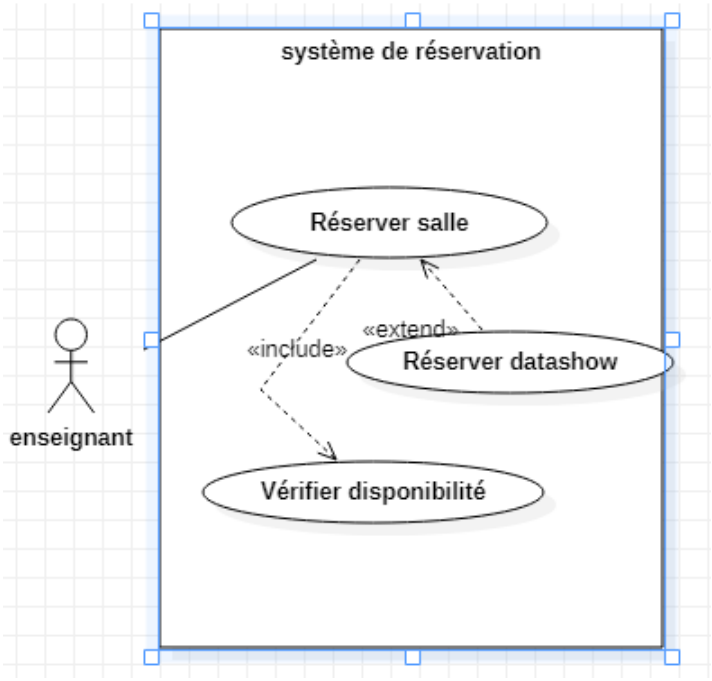


FIGURE 3.4 – Use case : Exemple

Chapitre 4

Diagramme de Classe

Tout système a besoin de stocker, gérer et manipuler des données de manière structurée. Ces données sont considérées comme étant des objets qui interagissent pour aboutir et réaliser un objectif commun.
Le diagramme de classe est un outil essentiel de la modélisation orientée objet. En effet, lorsqu'on souhaite concevoir un système, on s'aperçoit qu'un ensemble d'objets interagissent pour accomplir un cas d'utilisation.
Un diagramme de classe permet de représenter de manière visuelle les classes, les attributs, les relations et les méthodes d'un système ou d'une application.
Les diagrammes de classe fournissent une vue d'ensemble structurée des entités principales du système, ainsi que de leurs interactions et de leurs dépendances.
L'objectif principal de ce diagramme est de capturer la structure statique d'un système logiciel, en mettant l'accent sur l'aspect données du futur logiciel, en effet, en se focalisant sur les classes et leurs associations en leur offrant une représentation graphique, ce diagramme facilite la compréhension durant la réalisation du projet.

4.1 La Classe

Le monde est composé d'entités qui collaborent : ce sont des objets réels ayant une sémantique et caractéristiques communes, par exemple, les étudiants ont tous, des noms, prénoms, age, niveau.... ils ont les mêmes caractéristiques, alors pourquoi pas les mettre

dans une même classe. Donc une classe sert de modèle ou de moule pour créer des instances spécifiques appelées objets. Les objets sont des instances concrètes de la classe, dotées de valeurs spécifiques pour leurs attributs et capables d'exécuter les méthodes définies dans la classe.

La figure 4.1 représente les différents types de représentations graphiques d'une classe.

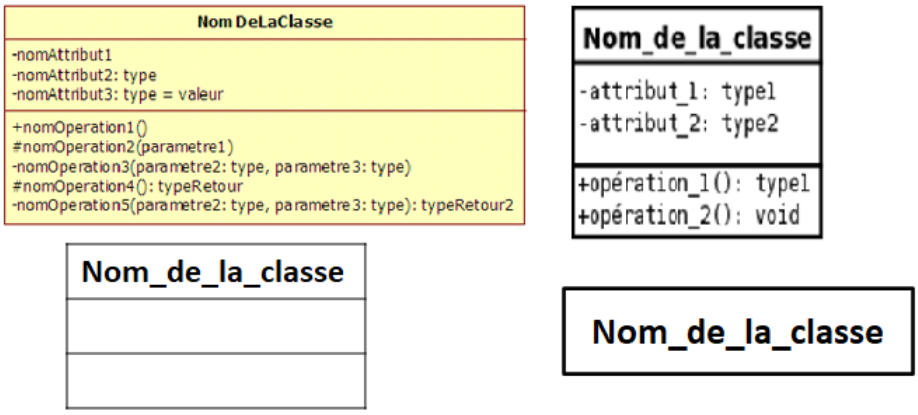


FIGURE 4.1 – Représentations graphiques d'une classe

La définition d'une classe comprend généralement les éléments suivants :

Nom de la classe : Un nom unique qui identifie la classe dans le programme.

Attributs : Les attributs sont des variables qui représentent les caractéristiques ou les données associées à la classe. Ils décrivent l'état de l'objet et sont déclarés au niveau de la classe. Par exemple, une classe "Personne" peut avoir des attributs tels que "nom", "âge" et "adresse".

Méthodes : Les méthodes sont des fonctions qui définissent le comportement de la classe. Elles spécifient les actions que les objets de la classe peuvent effectuer. Par exemple, une classe "Personne" peut avoir des méthodes telles que "se présenter" ou "changer d'adresse".

Constructeur : Le constructeur est une méthode spéciale appelée lors de la création d'un nouvel objet de la classe. Il permet d'initialiser les attributs de l'objet avec des valeurs spécifiques.

Visibilité : Les attributs et les méthodes peuvent avoir des niveaux de visibilité qui

déterminent leur accessibilité à d'autres classes. Les niveaux courants sont public, private et protected.

Relations : Une classe peut avoir des relations avec d'autres classes, telles que l'héritage, l'agrégation ou l'association. Ces relations définissent les liens et les interactions entre les différentes classes du système.

4.1.1 Encapsulation, visibilité, interface

Dans une classe, la visibilité est spécifiée pour chaque caractéristique (attributs, terminaisons d'association et opérations) afin d'indiquer si une autre classe peut y accéder.
L'encapsulation est un mécanisme qui regroupe les données et les méthodes au sein d'une structure tout en cachant l'implémentation interne de l'objet. Cela signifie que l'accès aux données se fait uniquement à travers les services fournis par l'objet, ce qui définit son interface (la manière dont il est utilisé de l'extérieur). L'encapsulation garantit ainsi l'intégrité des données contenues dans l'objet.

L'encapsulation permet également de définir différents niveaux de visibilité pour les éléments d'un conteneur. La visibilité détermine si un élément peut être référencé par un autre élément situé dans des classes différentes. Cette relation entre un élément et son conteneur (qui peut être un paquetage, une classe ou un autre espace de noms) peut avoir l'une des quatre visibilités prédéfinies :
Public : tout élément qui peut accéder au conteneur peut également accéder à l'élément indiqué.
Protected : seul un élément situé dans le conteneur ou l'un de ses descendants peut accéder à l'élément indiqué.
Private : seul un élément situé dans le conteneur peut accéder à l'élément.
Package : seul un élément déclaré dans le même paquetage peut accéder à l'élément.

4.1.2 L'Héritage

Un objet de la classe « Véhicule à Moteur » hérite des propriétés de la classe « Véhicule » Donc, un tel objet a un poids, une taille et un prix. De plus il peut s'arrêter et démarrer. Par

ailleurs, il a une puissance et une vitesse de pointe (propriétés que n'ont pas les véhicules sans moteur). Figure 4.1.2
On dit que la classe « VéhiculeAMoteur » est dérivée (ou héritée) de la classe « Véhicule ».

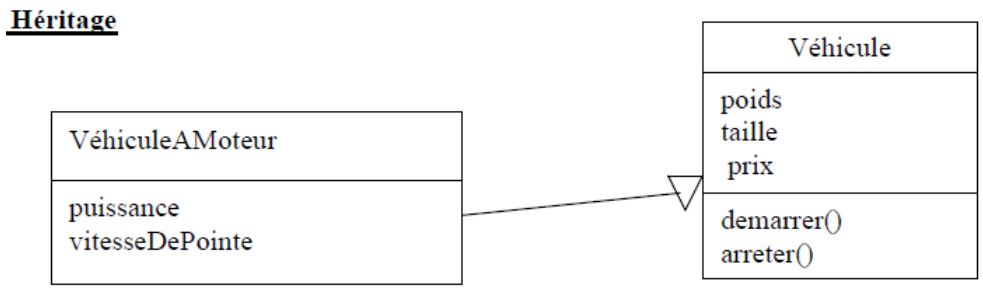


FIGURE 4.2 – Héritage

Les caractéristiques de l'héritage :

— la classe enfant possède toutes les caractéristiques de ses classes parents, mais elle ne peut accéder aux caractéristiques privées de cette dernière ;
— une classe enfant peut redéfinir (même signature) une ou plusieurs méthodes de la classe parent.
— toutes les associations de la classe parent s'appliquent aux classes dérivées ;
— une instance d'une classe peut être utilisée partout où une instance de sa classe parent est attendue. Exp.toute opération acceptant un objet d'une classe Animal doit accepter un objet de la classe Chat ;
— une classe peut avoir plusieurs parents, on parle alors d'héritage multiple .

4.1.3 Le polymorphisme

Il s'agit de plusieurs formes pour la même opération.
Exemple : Il est impossible de calculer la surface d'une forme graphique si on a aucune information sur cette forme [la classe forme graphique est donc une classe abstraite]. Exemple dans la figure 4.1.3

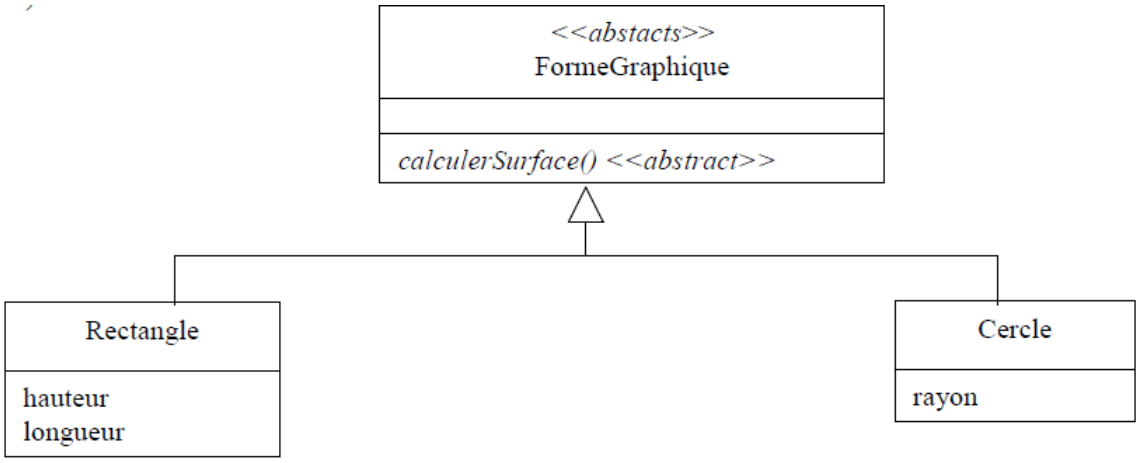


FIGURE 4.3 – Polymorphisme

4.2 Le diagramme de classe

Un diagramme de classes montre la structure statique d'un système. Il permet la visualisation des classes et des relations entre elles. Son but est d'expliquer ce qu'il faut réaliser plutôt que d'expliquer comment le réaliser.

4.3 Relations entre classes

4.3.1 Association

Dans un diagramme de classe, une association est utilisée pour représenter la relation entre deux classes. Elle indique qu'il existe une connexion ou une dépendance entre les objets de ces classes. L'association est généralement représentée par une ligne qui relie les deux classes et peut être annotée avec un nom et des multiplicités.
Voici quelques concepts clés liés à l'association dans un diagramme de classe :
Cardinalité : La cardinalité spécifie le nombre d'objets qui peuvent être associés entre deux classes. Elle est représentée par des chiffres ou des symboles près des extrémités de l'association. Par exemple, "0..1" signifie qu'il peut y avoir zéro ou un objet associé, "1" signifie exactement un objet, "0..*" signifie zéro ou plusieurs objets, etc.
Direction : Une association peut être unidirectionnelle ou bidirectionnelle. Dans une association unidirectionnelle, une classe est consciente de l'autre classe, mais l'inverse n'est pas

nécessairement vrai. Dans une association bidirectionnelle, les deux classes sont conscientes l'une de l'autre.
Rôle : Un rôle représente le rôle spécifique qu'une classe joue dans une association. Il peut être nommé pour indiquer la nature de la relation entre les deux classes. Par exemple, dans une association entre les classes "Étudiant" et "Cours", le rôle "étudie" peut être attribué à l'association.
Navigation : La navigation indique la capacité d'accéder à l'objet associé à partir d'une instance d'une classe. Dans une association bidirectionnelle, la navigation peut se faire dans les deux sens, tandis que dans une association unidirectionnelle, la navigation est limitée à une seule direction.
Multiplicité : La multiplicité spécifie le nombre d'objets d'une classe associée à un objet de l'autre classe dans une seule instance de l'association. Elle est généralement indiquée par des valeurs numériques ou des symboles comme "*" (zéro ou plusieurs), "1" (exactement un), etc.
Chaque terminaison d'association peut définir les multiplicités suivantes :

- 1 : C'est la valeur par défaut de toute terminaison qui précise qu'il y a un et un seul élément.
- 0..1 : Possibilité d'avoir 0 ou 1 élément.
- 1..* : Possibilité d'avoir 1 à n éléments.
- 0..* : Possibilité d'avoir 0 à n éléments.
- n : Définit un nombre précis d'éléments.
- n..m : Définit une fourchette précise d'éléments.
- * : équivalent de 0..*

Lorsque l'association contient un attribut, cette dernière est transformée en classe d'association contenant un attribut d'association, figure 4.3.1

Une association peut être binaire figure 4.3.1 ou n-aire figure 4.3.1

4.3.2 Agrégation et composition

Dans un diagramme de classe, la composition et l'agrégation sont des relations qui permettent de représenter la structure et les associations entre les classes. La composition

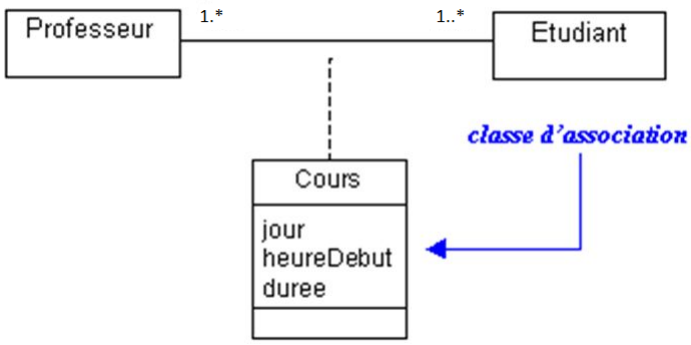


FIGURE 4.4 – Attribut d'association

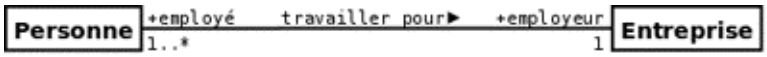


FIGURE 4.5 – Association binaire

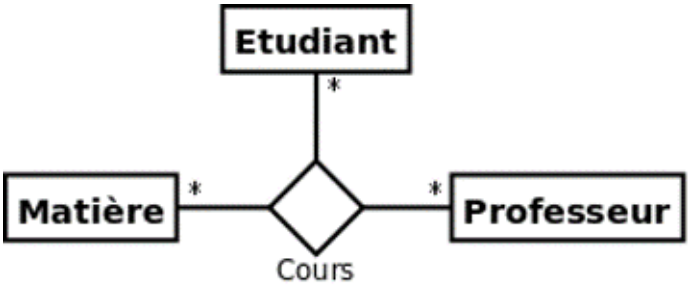


FIGURE 4.6 – Association n-aire

est une relation de tout ou partie, où une classe (le tout) est composée d'instances d'une autre classe (les parties). Cela signifie que lorsque le tout est détruit, les parties le sont également. La composition est représentée par un diamant noir plein sur le côté du tout et une ligne pleine reliant le tout aux parties.

L'agrégation, quant à elle, est une relation de tout ou partie moins forte que la composition. Elle indique qu'une classe (le tout) peut être composée d'instances d'une autre classe (les parties), mais que les parties peuvent également exister indépendamment du tout. Cela signifie que lorsque le tout est détruit, les parties peuvent continuer à exister. L'agrégation est représentée par un losange noir vide sur le côté du tout et une ligne pleine reliant le tout aux parties.

La composition et l'agrégation sont utiles pour représenter des structures hiérarchiques ou des relations tout-partie dans un système. Elles permettent de modéliser les associations

entre les classes de manière précise et de spécifier la nature des liens entre les objets. L'utilisation appropriée de la composition et de l'agrégation dans un diagramme de classe contribue à une meilleure compréhension du système et facilite la conception et la mise en œuvre des relations entre les classes.

FIGURE 4.7 – Agrégation et compostion

Chapitre 5

Diagrammes UML : vue dynamique

5.1 Diagramme de Séquences

Une interaction est un comportement qui comprend un ensemble de messages échangés entre un ensemble d'objets pour accomplir un objectif. Elle est utilisée pour modéliser la dynamicité entre les objets ainsi que leur collaboration pour accomplir un but.
Un diagramme de séquence est un type de diagramme de comportement et d'interaction. Il représente l'interaction entre les différents objets d'un système au fil du temps. Les diagrammes de séquence sont souvent utilisés pour visualiser et comprendre le flux d'exécution d'un scénario ou d'une fonctionnalité spécifique dans un système.

Dans les sections précédentes, nous avons mentionné que chaque objet possède des attributs qui décrivent les informations sur ce dernier. Un objet possède un comportement décrit par des opérations ; un diagramme de séquence montre la séquence chronologique des messages échangés entre les objets d'un système. Il met en évidence les interactions entre les objets, les ordres d'exécution des actions et les changements d'état au fil du temps. Les messages sont envoyés entre objets pour exécuter des méthodes, et donc, un message est une fonction ou procédure appelée par un objet pour un autre objet. voir la figure 5.1

Il permet de modéliser et de communiquer efficacement les scénarios d'utilisation, les flux de contrôle et les interactions entre les composants d'un système. Ils est largement utilisé dans le processus de conception logicielle pour clarifier et documenter les comportements attendus d'un système.

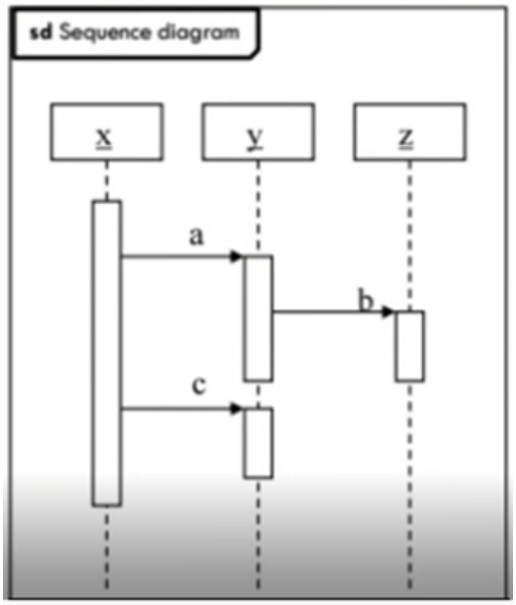


FIGURE 5.1 – Diagramme de séquence général

Sa représentation graphique montre comment les objets d'un système interagissent entre eux et échangent des messages tout au long du processus, permettant ainsi de visualiser et de comprendre le flux d'exécution d'un scénario spécifique.

La figure 5.1 représente un exemple qui définit une interaction d'objets, dans cette figure, l'instance de la classe Vendeur appelle la méthode Modifier(prénom Client) de l'objet de la classe FenêtreClient, ce dernier à son tour, appelle une méthode MAJ(prenom Client) définie dans la classe Client.

5.1.1 Éléments d'un diagramme de Séquence

Système

Le système est représenté par un rectangle qui le limite. A son coin gauche, un pentagone contiendra son nom. Les objets (instances) sont représenté en horizontal et lignes de vie en vertical. La communication est représentée par des messages horizontal entre les objets. Un diagramme de séquence se lit de haut en bas.

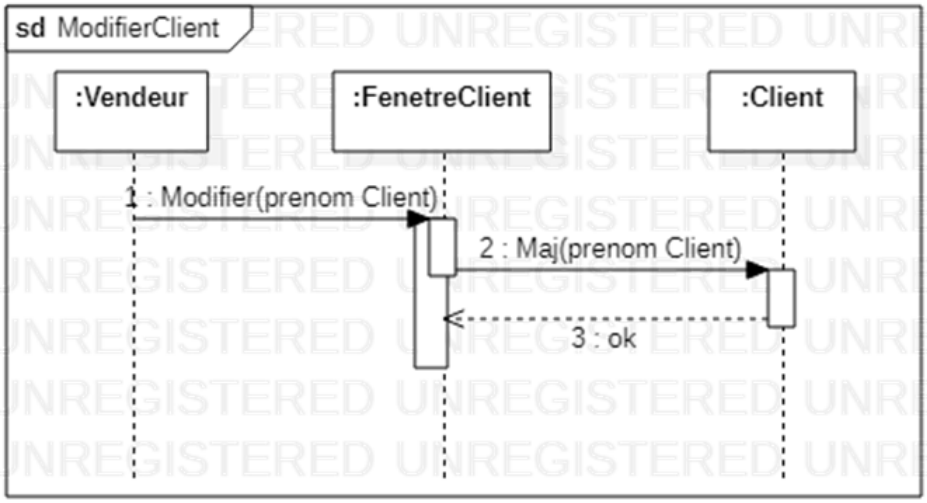


FIGURE 5.2 – Exemple d'un appel de méthode

La ligne de vie

La ligne de vie (figure 5.1.1)est un élément clé d'un diagramme de séquence, utilisé pour représenter la durée de vie d'un objet, elle est représentée par une ligne verticale qui démarre à partir de la boîte d'objet.

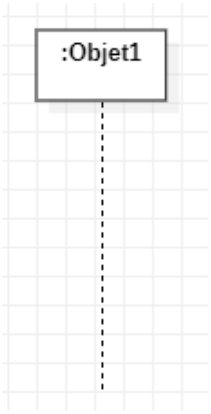


FIGURE 5.3 – Ligne de vie

La ligne de vie représente la durée pendant laquelle l'objet existe et participe à l'interaction. Elle montre le début et la fin de la vie de l'objet dans le contexte du scénario ou de la fonctionnalité modélisée.

Les lignes de vie sont placées verticalement sur le diagramme pour indiquer l'ordre séquentiel des objets impliqués dans l'interaction. Les objets situés plus haut sur le dia-

gramme sont activés avant ceux situés plus bas.

Lorsque la ligne de vie atteint sa fin, cela indique que l'objet n'est plus actif et cesse d'exister dans l'interaction. La disparition de l'objet est généralement représentée par un point ou une marque à la fin de la ligne de vie.

La syntaxe du libellé de la ligne de vie est : nomLigneDeVie [selecteur] : NomClasseOuActeur, par exemple :

omar : Etudiant ou **: Etudiant** ou **mohamed :**

Plusieurs objets peuvent être activés en parallèle et exécuter des actions simultanées. Dans ce cas, plusieurs lignes de vie peuvent être dessinées à des hauteurs équivalentes pour indiquer le parallélisme des activités.

Lors de l'exécution d'une méthode, l'objet est activé et la ligne pointillée verticale de la ligne de vie est remplacée par un rectangle vertical durant l'activité de l'objet.

Messages

Les messages échangés sont de deux types : messages synchrones et asynchrones, ils jouent un rôle essentiel dans la création et la destruction d'instances. Ils permettent d'établir une communication spécifique entre différentes lignes de vie. Les types de messages les plus couramment utilisés sont les suivants :

— L'envoi d'un signal

— L'invocation d'une opération

a) Message Asynchrone

Une interruption ou un événement sont de bons exemples de ce type de messages.

Ce type de message n'attend pas de réponse et ne bloque pas l'émetteur qui ne se soucie pas de l'arrivée de ce message à destination.

un exemple de ces deux types de message est présenté dans la figure 5.1.1

Graphiquement, un message asynchrone est représenté par une flèche en traits pleins et à l'extrémité ouverte partant de la ligne de vie d'un objet expéditeur et allant vers celle de l'objet cible (figure 5.1.1 -(a)).

b) Message Synchrone

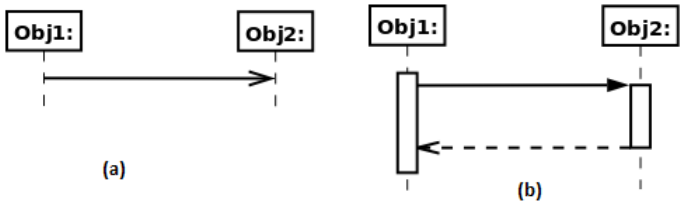


FIGURE 5.4 – (a) Message asynchrone- (b) Message synchrone

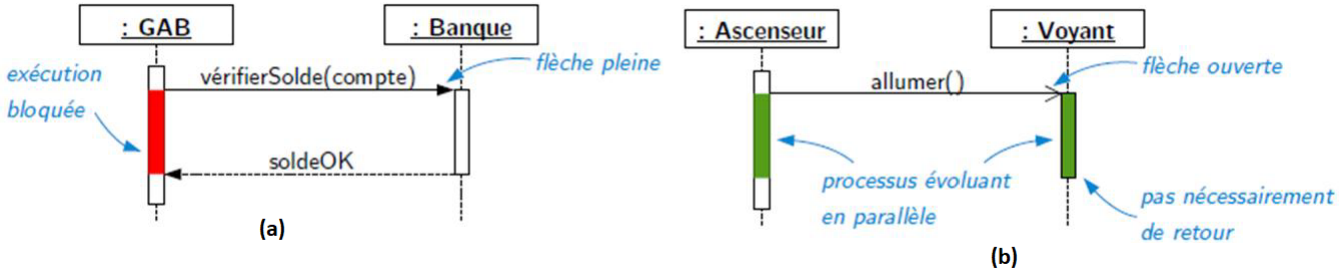


FIGURE 5.5 – Exemple : Message synchrone (a)- asynchrone (b)

Contrairement au premier, ce type de message bloque l'émetteur lors de l'invocation de l'opération.

Graphiquement, un message synchrone se représente par une flèche en traits pleins et à l'extrémité pleine partant de la ligne de vie d'un objet expéditeur et allant vers celle de l'objet . Ce message peut être suivi d'une réponse qui se représente par une flèche en pointillé (figure 5.1.1 -(b)).

c)Message de Construction/Destruction d'objets

On crée un nouvel objet par une flèche dirigée vers le sommet d'une ligne de vie, et c'est à partir de cette flèche que démarre la ligne de vie, cet opération s'agit de l'instanciation d'une classe (voir figure 5.1.1). La suppression d'un objet est indiquée par une croix qui marque la fin de la ligne de vie de l'objet.

d) Message perdu, message trouvé

Un message complet est caractérisé par la connaissance des événements d'envoi et de réception. Comme mentionné précédemment, un message complet est représenté par une simple flèche allant de l'émetteur au récepteur.

Un message perdu se produit lorsque l'événement d'envoi est connu, mais pas l'événement

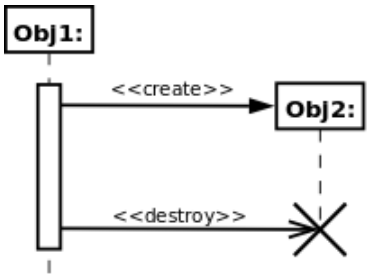


FIGURE 5.6 – Message de Construction/Destruction d'objets

de réception. Il est représenté par une flèche pointant vers une petite boule noire (voir figure 5.1.1).
Un message trouvé se produit lorsque l'événement de réception est connu, mais pas l'événement d'émission. Une flèche partant d'une petite boule noire représente un message trouvé (voir figure 5.1.1).

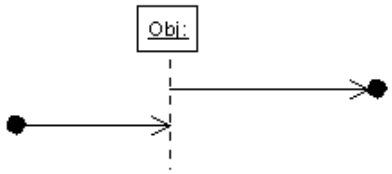


FIGURE 5.7 – Message perdu/message trouvé

5.1.2 Fragments d'interaction combinés

Les fragments d'interaction combinés dans un diagramme de séquence sont des éléments utilisés pour modéliser des scénarios complexes et des conditions de contrôle dans un système. Ils permettent de représenter des comportements conditionnels, répétitifs et parallèles. Ces fragments sont symbolisés graphiquement à l'aide de rectangles avec des coins arrondis et des lignes de contrôle. En les combinant, on peut décrire des séquences d'actions et de messages plus complexes, prenant en compte des conditions, des itérations et des branches. Cela permet de représenter le comportement dynamique d'un système de manière précise dans un diagramme de séquence.
les opérateurs d'interaction sont :

— les opérateurs de choix et de boucle : alternative, option, break et loop ;

— les opérateurs contrôlant l'envoi en parallèle de messages : parallel et critical region ;
— les opérateurs contrôlant l'envoi de messages : ignore, consider, assertion et negative ;
— les opérateurs fixant l'ordre d'envoi des messages : weak sequencing , strict sequencing.

Nous citons quelque uns dans les prochains paragraphes.

5.1.3 Alternatif if..then..else

Dans un diagramme de séquence, l'alternative (if-then-else en anglais) est utilisée pour représenter des branches conditionnelles dans le flux d'exécution. Elle permet de spécifier différentes séquences d'actions ou de messages en fonction d'une condition donnée. L'alternative est composée de trois parties principales : la condition, la branche ″then″ (si la condition est vraie) et la branche ″else″ (si la condition est fausse).

La condition de l'alternative est généralement une expression booléenne qui évalue un état ou une propriété. Elle est représentée par un losange avec la condition écrite à l'intérieur. Les flèches de sortie du losange indiquent les branches ″then″ et ″else″.

La branche ″then″ représente la séquence d'actions ou de messages à exécuter lorsque la condition est évaluée comme vraie. Elle est généralement représentée par une ligne solide avec des messages ou des actions le long de cette ligne.

La branche ″else″ représente la séquence d'actions ou de messages à exécuter lorsque la condition est évaluée comme fausse. Elle est généralement représentée par une ligne pointillée avec des messages ou des actions le long de cette ligne.

L'utilisation de l'alternative dans un diagramme de séquence permet de représenter différentes séquences d'actions en fonction d'une condition donnée. Cela est utile lorsque le flux d'exécution d'un système dépend de certaines conditions ou décisions. L'alternative aide à modéliser les scénarios où différentes actions doivent être prises en fonction des résultats des évaluations conditionnelles.

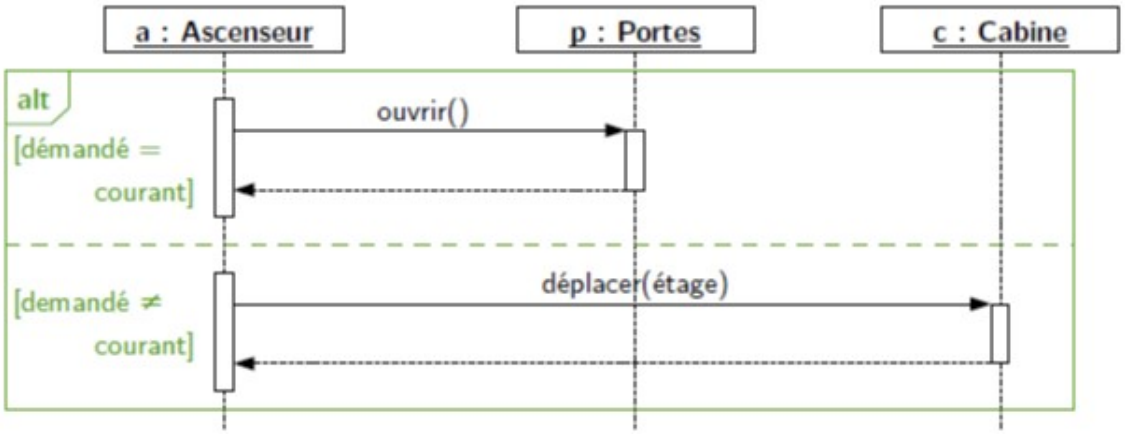


FIGURE 5.8 – Alternatif if then else

Loop

Dans un diagramme de séquence, une boucle (ou loop en anglais) est utilisée pour représenter une itération ou une répétition d'un ensemble d'actions ou de messages. Elle indique qu'un certain groupe d'actions ou de messages sera exécuté plusieurs fois selon une condition spécifiée. Il existe deux types de boucles couramment utilisées dans un diagramme de séquence : la boucle conditionnelle et la boucle itérative.

La boucle conditionnelle, également appelée boucle "while" ou boucle "do-while", est utilisée lorsque la répétition dépend d'une condition. Elle spécifie que les actions ou les messages à l'intérieur de la boucle seront exécutés tant que la condition donnée est vraie. La condition est généralement représentée par une flèche de retour pointant vers le début de la boucle ou vers une autre partie du diagramme, indiquant ainsi le retour en arrière pour répéter les actions.

La boucle itérative, également appelée boucle "for" ou boucle "foreach", est utilisée lorsque la répétition est basée sur un ensemble d'éléments ou de valeurs. Elle spécifie qu'un ensemble d'actions ou de messages sera exécuté pour chaque élément de cet ensemble. La boucle itérative est généralement représentée par une barre horizontale avec une flèche de retour pointant vers le début de la boucle. Une variable de contrôle peut être utilisée pour parcourir les éléments de l'ensemble.

L'utilisation de boucles dans un diagramme de séquence permet de modéliser les comportements itératifs ou répétitifs dans un système. Elles aident à illustrer les séquences

d'actions ou de messages qui se répètent jusqu'à ce qu'une condition soit satisfaite ou que tous les éléments d'un ensemble soient traités. Les boucles sont particulièrement utiles pour représenter des scénarios tels que la vérification d'une liste d'éléments, l'itération sur une collection de données ou la réalisation d'une action jusqu'à ce qu'une certaine condition soit remplie.

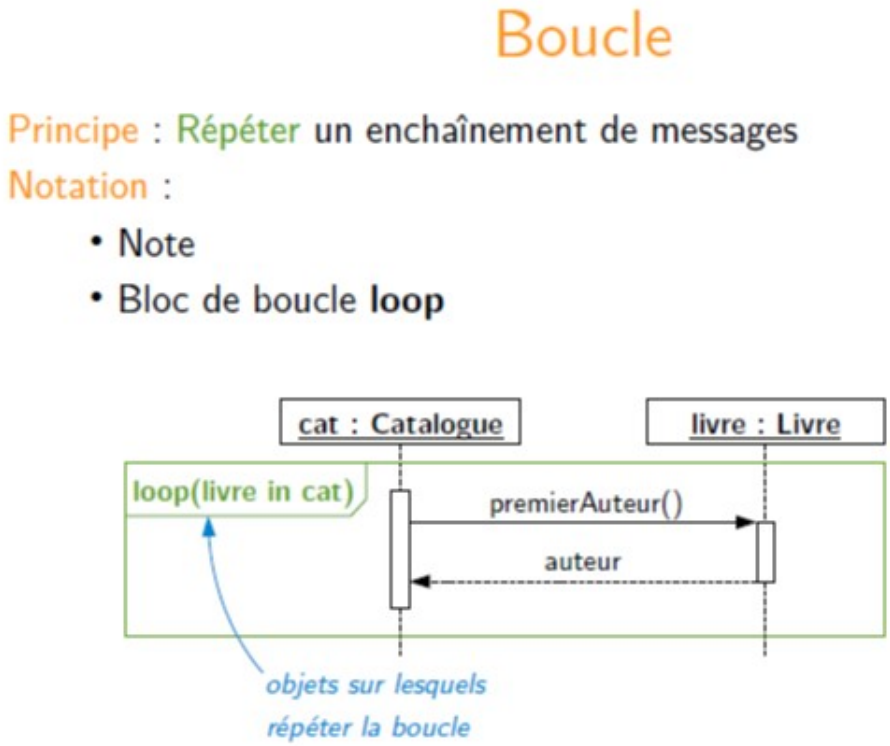


FIGURE 5.9 – Loop

Message réflexif

5.2 Diagramme d'activité

Le diagramme d'activité est un modèle permettant de représenter une partie dynamique du projet logiciel. Il représente graphiquement le flux des activités, actions et décisions au sein d'un système, d'un processus ou d'un scénario. Un diagramme d'activité se compose de noeuds, de transitions et de connexions, qui décrivent les différentes étapes et les conditions de passage d'une activité à une autre.

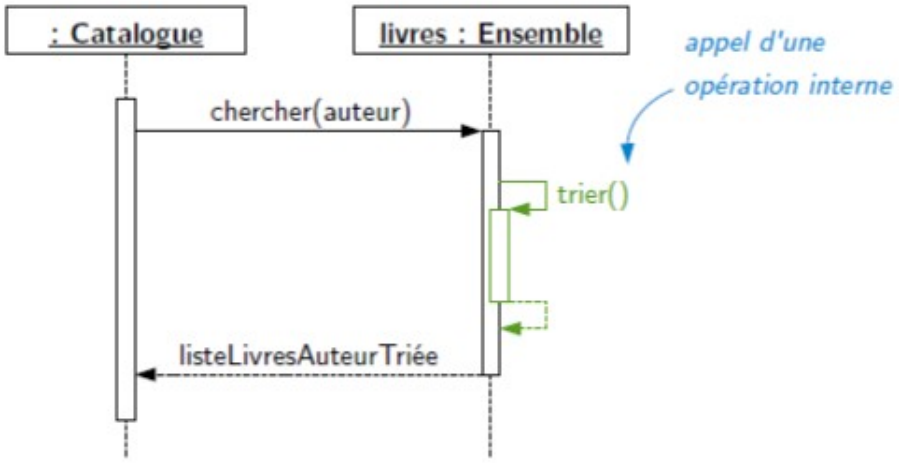


FIGURE 5.10 – Message reflexif

Il permet de visualiser les séquences temporelles, les branchements conditionnels, les boucles et les parallélismes, offrant ainsi une vue d'ensemble claire et compréhensible du déroulement des activités. Les diagrammes d'activité facilitent la communication entre les concepteurs, les développeurs et les parties prenantes, favorisant ainsi la collaboration et l'identification des points forts et des faiblesses d'un système ou d'un processus.
Les diagrammes d'activités permettent de mettre l'accent sur les traitements. Ils sont donc particulièrement adaptés à la modélisation du cheminement de flots de contrôle et de flots de données. Ils permettent ainsi de représenter graphiquement le comportement d'une méthode ou le déroulement d'un cas d'utilisation.
Une activité définit un comportement décrit par un séquencement organisé d'unités dont les éléments simples sont les actions. Le flot d'exécution est modélisé par des nœuds reliés par des arcs (transitions). Le flot de contrôle reste dans l'activité jusqu'à ce que les traitements soient terminés.
Une activité est un comportement (behavior en anglais).

5.2.1 Eléments d'un diagramme d'activité

La figure 5.2.1 représente les éléments d'un diagramme d'activité.

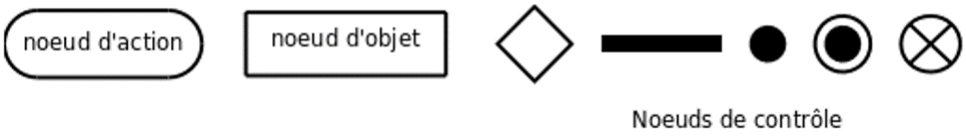


FIGURE 5.11 – Eléments du diagramme d'activité

5.3 Diagramme d'état transition

Un diagramme d'état-transition est un diagramme qui exprime le comportement.
Il décrit le fonctionnement d'un objet lorsqu'il reçoit un évènement (ou avant sa réception)
Il représente les différents états dans lesquels peut se trouver un objet et la façon avec laquelle cet objet passe à un autre état en réponse d'un évènement.
un exemple est montré dans la figure 5.3.1.
Un diagramme d'état transition se composent de :
Etat : Décrit un moment de la vie d'une instance. Cette instance ne peut se trouver que dans un seul état à la fois.
Les instance d'une même classe réagissent de la même façon aux évènements. **Transition :** La réponse d'une instance dans un état donné à un évènement.
Évènement : déclenche la transition.
Action : la réalisation d'une opération lorsque la transition est réalisée.

5.3.1 L'Etat

A sa création, l'objet se trouve dans un état initial. Lorsque le système n'a plus besoin de l'objet, il se retrouve dans un état final.
Un objet peut ne pas avoir d'état final.

5.3.2 La transition

Une Transition définit la Réponse d'un Objet à l'arrivée d'un Événement.
Elle indique qu'un Objet qui se trouve dans un État peut «Transiter» vers un autre État en exécutant éventuellement certaines activités.

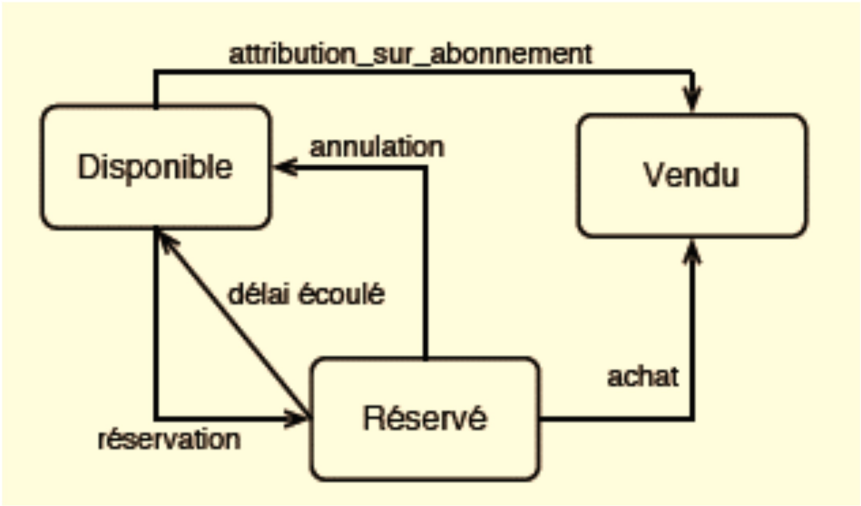


FIGURE 5.12 – Exemple d'un diagramme d'état transition

5.3.3 L'évènement

il provoque le franchissement de la transition.

5.3.4 GARDE-CONDITION DE GARDE

Entre [], on l'utilise pour associer une condition à une transition. Pour que la transition soit franchie, il faut que la condition soit vérifiée en plus de la réception de l'événement associé, si celui-ci existe.
C'est une expression booléenne(sur les attributs de l'objet ou les paramètres de l'événement déclencheur) qui doit être vraie pour que la transition soit déclenchée.

5.3.5 Type d'évènement

il existe 4 types d'évènement :

— l'envoi d'un signal ;
— l'appel d'une opération ; etc
— lorsque l'exécution de l'action est terminée, l'état
— cible de la transition devient actif,

Un exemple d'évènement temporel :
Les événements temporels sont générés par l'écoulement du temps, Ils sont spécifiés soit

de manière absolue (date précise) :

— De manière absolue (déclenchement à une date précise), Syntaxe : when(date= « expression précise d'une date ») ex : when(date=17/12/2010)

— De manière relative (déclenchement après une certaine durée passée dans l'état actuel), Syntaxe : after(« expression d'une durée ») ex : after(10secondes)

la transition interne et externe est montré dans l'exemple de la figure 5.3.5.

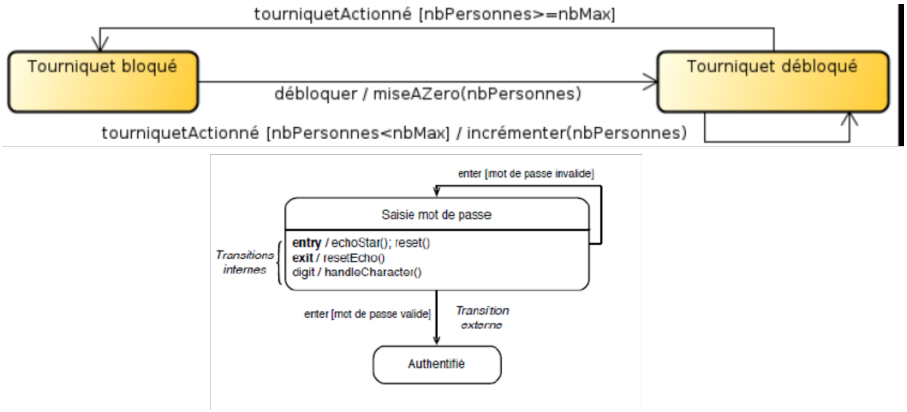


FIGURE 5.13 – Transition Interne et externe

Soit l'exemple suivant :

Un distributeur automatique fonctionne de la manière suivante :

Le matin, l'agent branche le distributeur, il s'allume et reste inactif en attendant la manipulation par un client. Un client arrive et insère sa carte, 2 cas sont possibles :

Carte non valide : le distributeur rend la carte et reste inactif.

Carte est valide : le distributeur devient actif en attente de la saisie du code, et initialise une variable compteur (nbrEssais) à 0, chaque fois que le client saisie le code nbrEssai s'incrémente.

Le client saisie le code, et 3 cas se présentent :

Code est erroné, nbrEssai est incrémenté, le distributeur attend.

Si le nbrEssai =3, le distributeur avale la carte est redevient inactif.

Le code est valide, le distributeur devient en préparation d'une transaction.

Le client choisit un montant, le distributeur devient en transaction et rend la carte.

Si le client valide en appuyant sur un bouton « ok », les billet sont délivrés et le distributeur devient inactif.

Le client appuis sur annulé, le distributeur devient inactif.

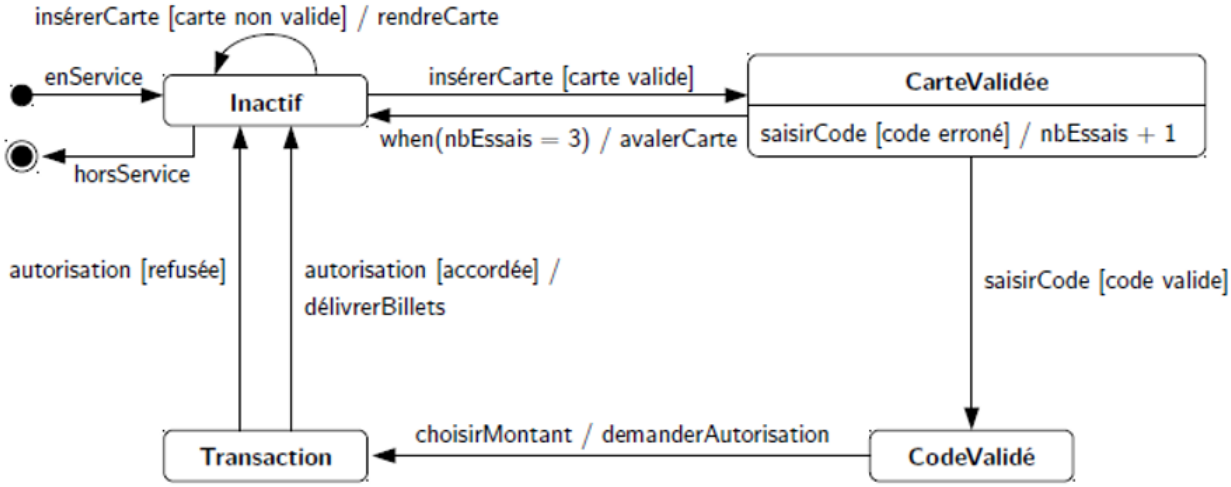


FIGURE 5.14 – Diagramme d'état transition de distributeur

Chapitre 6

Autres notions et diagrammes UML

Au cours de développement d'un logiciel, des conditions et des règles sont imposées sur les objets et leurs relations, pour les spécifier et pour définir les contraintes et les expressions sur les modèles de données, le langage OCL (Object Constraint Language) s'avère très utile. OCL est un langage formel qui utilise la logique des prédicats. Une contrainte est représentée par une chaîne de texte placée entre accolades :.

6.1 Représentation d'une contrainte

UML permet d'associer une contrainte à un élément de modèle de plusieurs façons : par exemple sur la figure **??**, on souhaite imposer une contrainte sur l'attribut "âge" de la classe Employé, tel que âge¡60 ans.

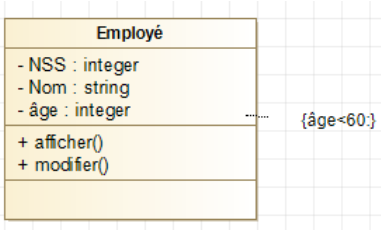


FIGURE 6.1 – Exemple de contrainte OCL

Types de données : OCL prend en charge différents types de données tels que les types primitifs (entier, chaîne de caractères, booléen), les types de collection (ensemble, séquence, sac) et les types définis par l'utilisateur (classes UML).

6.2 Les contraintes en langage OCL

Contraintes statiques : OCL permet de définir des contraintes statiques qui doivent être respectées par les objets du modèle. Ces contraintes spécifient des conditions qui doivent être vraies pour que le modèle soit considéré comme valide. Par exemple, on peut définir une contrainte sur une classe pour s'assurer qu'un attribut spécifique ne dépasse pas une valeur maximale.

Contraintes dynamiques : OCL permet également de spécifier des contraintes dynamiques qui décrivent le comportement et les invariants d'un système lors de l'exécution. Ces contraintes peuvent être utilisées pour vérifier les préconditions et les postconditions des opérations, les invariants de classe, etc.

Navigation et références : OCL fournit des opérateurs de navigation pour accéder aux propriétés, aux relations et aux éléments d'un modèle. Il est possible de naviguer à travers les associations, d'accéder aux attributs et aux opérations d'une classe, et de filtrer les collections en utilisant des conditions.

Expressions conditionnelles : OCL prend en charge des expressions conditionnelles, telles que les conditions "if-then-else", qui permettent de définir des comportements conditionnels basés sur des évaluations booléennes.

Opérations et fonctions : OCL permet de définir des opérations et des fonctions pour encapsuler des comportements réutilisables. Ces opérations peuvent être définies au niveau de la classe ou du modèle, et elles peuvent être utilisées pour effectuer des calculs, des filtrages ou des transformations de données.

OCL est souvent utilisé en conjonction avec UML pour spécifier des règles de validation, des invariants, des préconditions et des postconditions dans le développement de logiciels. Il offre un moyen formel et précis d'exprimer des contraintes et des conditions sur les modèles de données, ce qui facilite la vérification et la validation du système.

Deuxième partie

Partie Travaux Dirigés

Fiche TD1 : Analyse des besoins

Mohamed est un étudiant en L3, il souhaite réaliser un mini projet, pour proposer une application qui facilitera le calcul de la note TD. Pour cela, il s'est donné un Rdv avec un enseignant.

Mohamed : Comment comptez-vous les absences ? Enseignant : l'étudiant à le droit de s'absenter au maximum 3 fois sans justificatif et 5 fois avec justificatif, sinon il sera exclu. Et moi je trouve une difficulté pour mémoriser si l'étudiant à ramener un justificatif pour cela je tris tous les certificat reçus pour donner une note sur 5 pour les présences.

Mohamed : vous compter utiliser l'application comment ?

Enseignant : sur mon téléphone si c'est possible, ou sur mon pc.

Mohamed : Comment se déroule une séance de TD ?

Enseignant : je donne aux étudiants une fiche TD avec une série d'exercices qu'ils essayent de résoudre pendant le TD en se référant au cours, si on ne termine pas la fiche, les étudiants essayent de travailler à la maison. Pendant le TD, des étudiants volontaires, montent au tableau afin que leurs camarades comprennent mieux..

Mohamed : quelle est l'importance de la note TD pour un étudiant ? Enseignant : la note TD est introduite dans la moyenne du module, elle est calculée de la manière suivante : la note de l'examen *0.6 +la note de TD*0.4. Dons si l'étudiant travaille bien pendant les TD, ça sera un plus dans sa moyenne du module.

Mohamed : Avez-vous vraiment besoin de cette application ?

Enseignant : elle me sera très utile, pour moi et pour mes collègues, car lorsqu'on doit saisir les notes de TD dans la plateforme, il faut d'abord les calculer, pour chaque étudiant. On le fait ou manuellement ou dans Excel avec la saisi de la note présence sur 5, participation sur 5, et test sur 10, et comme vous le voyer dans le document (liste) que je vous ai donné, c'est incompréhensible et parfois, on n'arrive pas à se souvenir du visage de l'étudiant. Une application nous fera gagner beaucoup de temps, et sera plus claire et plus visible qu'un papier avec du stylo partout.

Mohamed : Comment se calcule la note de CC ?

Enseignant : la note CC= la présence + la participation+ le Test Mohammed : est-ce que

l'étudiant a le droit de voir sa note de TD en détail ?
Enseignant : oui, en général, l'étudiant demande à l'enseignant directement de lui donner les détails, mais il est préférable qu'il accède à sa note directement.

Université de Relizane
Faculté des sciences et Technologie
Département Informatique
L3- Groupe 2

	Nom	Prénom	11/10/21	18/10/21	19/10/21	20/10/21
1	Abdelhak	Mohamed	P +	P+	A	P
2	Cherif	Imane	P ++	P	P	A
3	Karim	Nour	P -	P	P+	P-
4	Mohamed	amine	A	A	A	P-

Mohammed : Lorsque vous mettez + qu'est-ce-que ça veut dire ? Enseignant : le + est ambigu, car des fois, au moment de saisir les notes finales, je ne me souviens pas si c'est un travail fait à la maison où pendant la séance TD, ou que l'étudiant est monté au tableau. Mohammed : et le -? Enseignant : lorsque l'étudiant a un comportement indiscipliné pendant une séance, je met un – pour le soustraire de la note finale.

Fiche TD 2

Exercice 1 :

On souhaite réaliser une application pour une entreprise qui fabrique des meubles (chambres, salon, cuisine,...). Après s'être entretenu avec le directeur, ce dernier nous informe qu'il désire, plus précisément, avoir un logiciel de gestion commerciale permettant à l'agent commercial d'enregistrer la commande client. Si ce client est nouveau, il sera enregistré dans la base pour pouvoir le retrouver en cas de futures commandes.

Le chef service commercial étudie la commande enregistrée et établis un contrat avec le client selon le montant commandé ; si le montant est inférieur à 500mille da, le contrat sera de type convention sinon (supérieur à 1 million da) ça sera un marché. Le chef service établit un ordre de fabrication qui sera exporté par le logiciel de suivi de production (du service production). A la fin de la fabrication, le chef service importe à partir du logiciel de production l'état de production réalisée, et établit alors un ordre de paiement qui sera envoyé à la comptabilité.

Chaque mois l'agent commercial édite des états statistiques à destination du directeur qui le consulte et peut, s'il le souhaite l'imprimer.

Grâce à ce logiciel le chef service peut effectuer en plus de ses tâches toutes les opérations de l'agent. Par soucis de sécurité, toutes les opérations nécessitent une authentification.

Question :proposer un diagramme de Use case pour cette application.

Exercice 2 : Soit un système de retrait d'un guichet automatique qui fonctionne généralement de la manière suivante :

Un technicien est envoyé par la banque afin d'alimenter le coffre en déposant l'argent, et de mettre à jour le stock du coffre, le client peut par la suite retirer la somme qu'il souhaite, après s'être authentifier en insérant sa carte et en introduisant son mot de passe.

Le coffre contient les deux devises : dinars et euro.

Question :proposer un diagramme de Use case pour ce système.

Fiche TD 3

On souhaite réaliser un site pour les fans de la Formule 1 :

Une course se déroule sur un circuit dans une ville particulière.

Pour remporter le championnat, le pilote doit cumuler les victoires sur plusieurs courses (obtenir beaucoup de points pour bien se classer sur plusieurs courses). Chaque pilote conduit une voiture lors d'une course dont les propriétés sont : la longueur, largeur, poids. Cette voiture appartient à une écurie ou constructeur (Mercedes, Ferrari,…) et donc le pilote signe un contrat avec ce constructeur. Lors de la course le pilote fait plusieurs tours avec sa voiture (de 50 à 75) en un temps déterminé, chaque tour possède un record. Pendant la course, le pilote s'arrête au stand technique de l'écurie pour un changement de pneus ou d'ajout de carburant. L'écurie met en place des stratégies concernant les pneus, la technologie ou le carburant pour améliorer la performance de la voiture et passer un temps minimal en stand.

Dans la formule 1, le pilote joue pour son propre titre et pour celui de l'écurie (Mercedes, Ferrari,…), pour gagner le championnat, l'écurie mise sur le nombre de voitures et la stratégie adoptée par le stand technique.

Exemple sur les informations auxquelles les fans peuvent accéder :

1) Hamilton conduit pour Mercedes a remporté le championnat 2018, car il a obtenu 340 points et 5 victoires. McLaren a remporté ce championnat comme constructeur et a obtenue 750 points.

2) Hamilton a obtenu 40 point sur la course d'Arabie Saoudite (Djeddah). Il a réalisé le 28ème tour en 1m02s, le record de ce tour est de 59 sec.

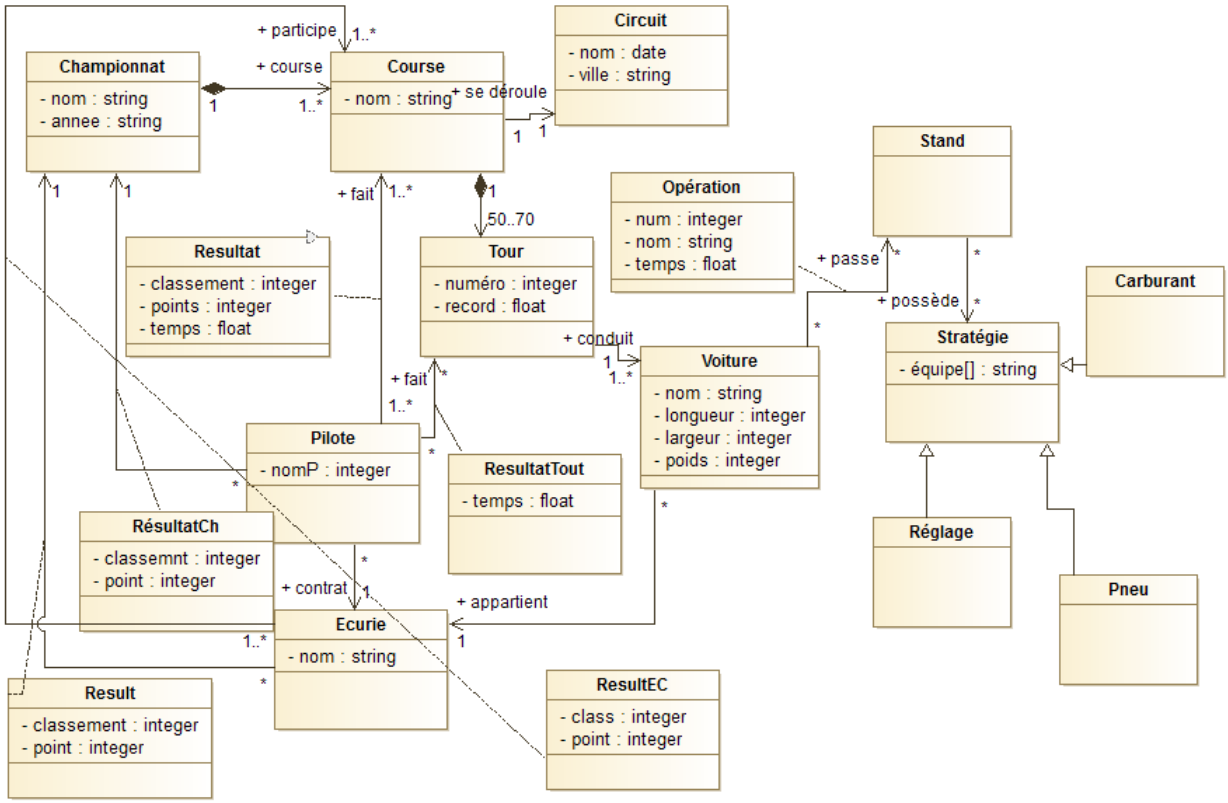
3) La voiture Ferrari N°502 est passée 8 fois en stand technique lors de la course d'Espagne (Barcelone). Lors de son deuxième passage, les pneus ont été changés en 2 sec 10ms.

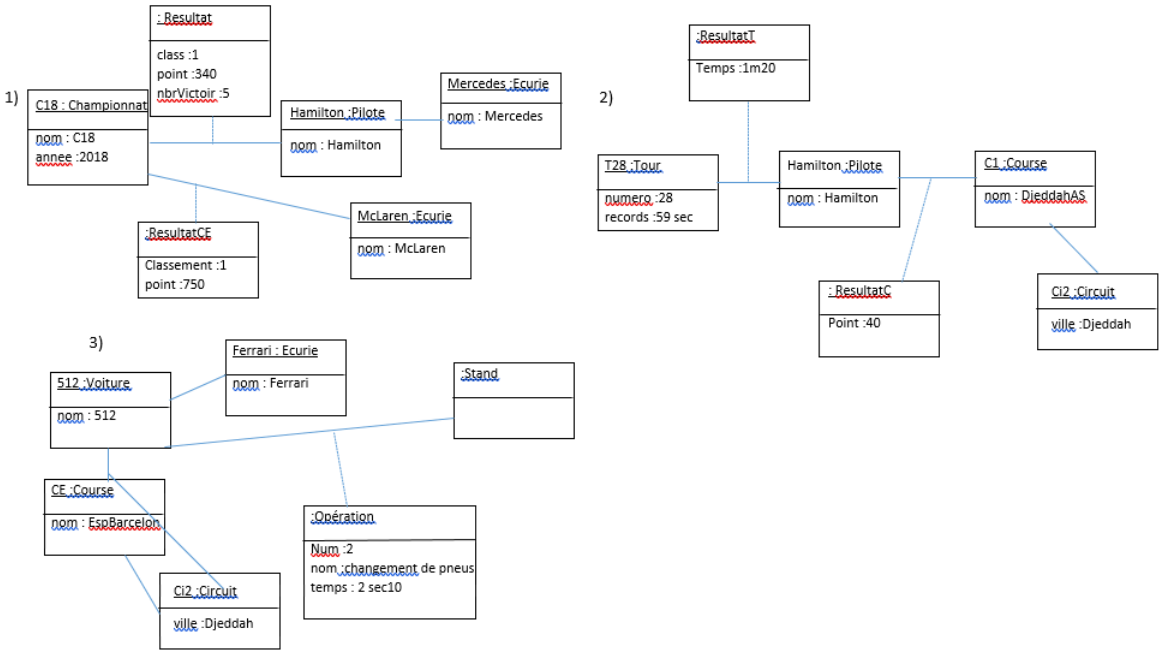
Q1) Proposez un diagramme de classe.

Q2) Proposez 3 diagramme d'objets : un pour chaque exemple (1,2,3).

Correction TD3

Question1 Question2





Fiche TD 4

Exercice 1 : On s'intéresse à la modélisation dynamique de la gestion d'une agence de location de voiture. Pour louer une voiture, on a le scénario suivant :

1) Le client se présente à l'agence, le gérant saisit la fonctionnalité pour louer une voiture au client à partir de l'application.

2) D'abord, il faut vérifier si le client a le droit de louer (toutes les voitures louées ont été rendues, pièce d'identité valide, nombre de voitures louées n'a pas atteint le max…).

3) En suite, il faut vérifier si la voiture demandée (marque et modèle) est disponible.

4) Si tout est ok, on crée une nouvelle location avec la date de location et la date de retour, associé avec le client la voiture choisie.

5) On rend la voiture non disponible.

6) On incrémente le nombre de voitures louées par le client.

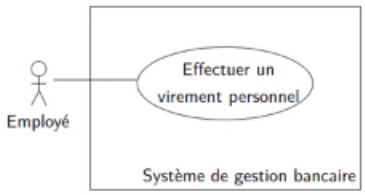
Etablir le diagramme de séquence de ce scénario de cas d'utilisation Louer une voiture

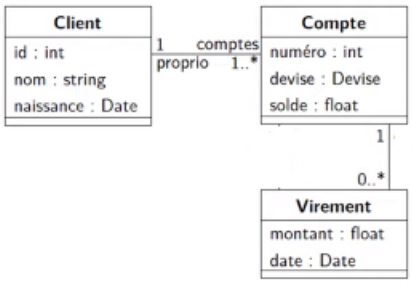
Exercice 2 :

Proposer un diagramme de séquence pour le cas « effectuer un virement personnel.

Question2

L'employé demande d'effectuer un virement à partir d'un compte client à un autre compte

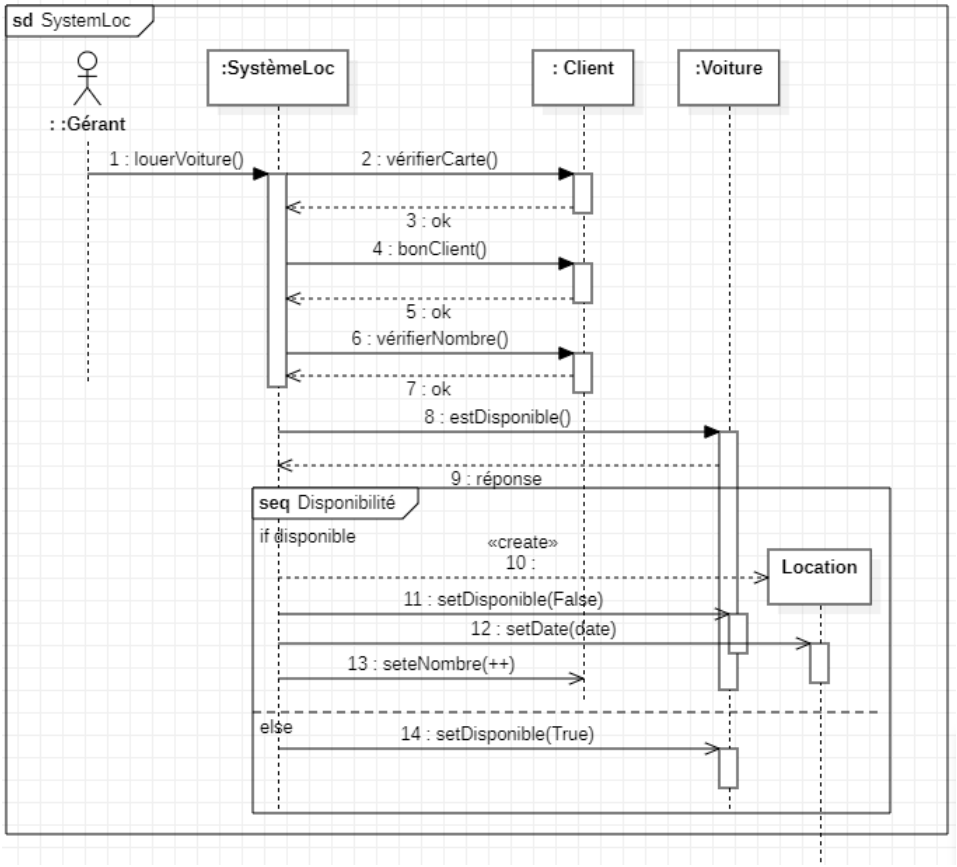




de la même banque. Pour cela :

- Il remplit un formulaire et l'envoi dans le système de gestion de banque (SGB), les informations nécessaire s'agit de l'identifiant du client source, et celui du client destinataire ainsi le montant à transférer.
- Le système cherche le client dans la liste des clients par son identifiant pour voir si c'est un client de la banque.
- Il envoi les numéros de clients pour obtenir leur numéro de comptes.
- Une fois trouvé, il vérifie le solde du compte client source (est-ce qu'il est ¿ au montant).
- Un nouveau virement est créé avec les deux compte et le montant .
- Deux opérations sont faite, une pour retirer le montant du premier compte, une deuxième pour déposer le montant dans le compte du destinataire.

Correction TD4



Fiche TD5

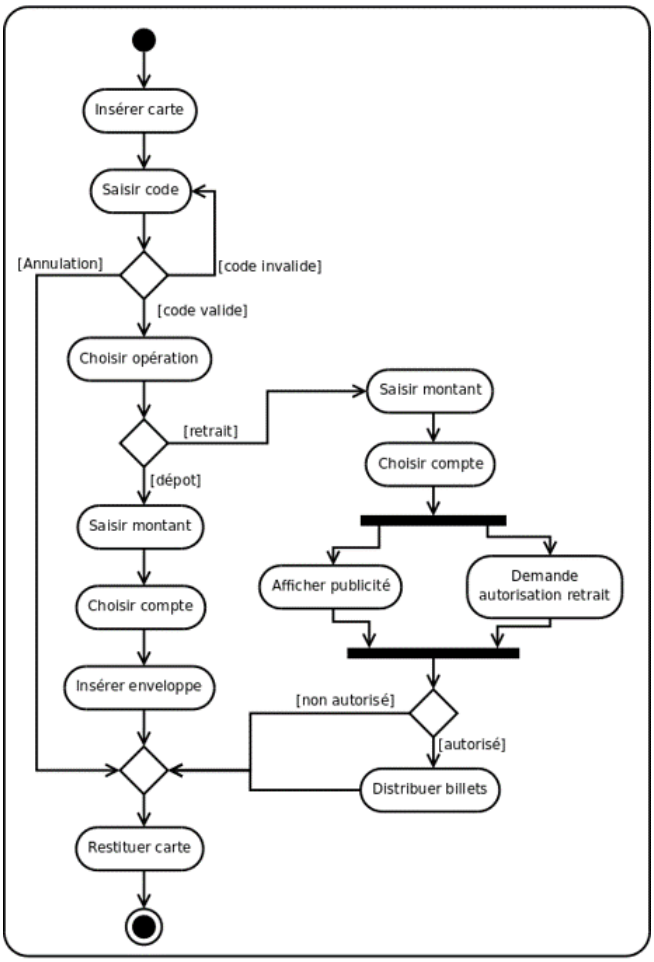
Exercice 1 :

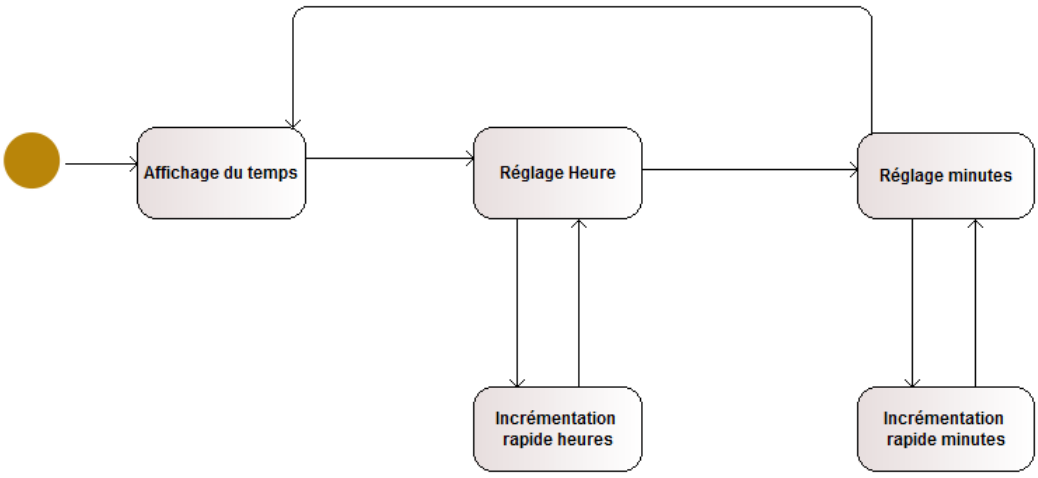
Soit un distributeur de billets fonctionnant de la manière suivante :
Un client introduit sa carte, la validité de cette carte est vérifiée. il est ensuite invité à saisir le code. Après 3 tentatives, la carte est avalée. sinon, le client indique le montant qu'il désire retirer, le solde de son compte bancaire est alors consulté pour s'assurer que le retrait est possible. En cas de solde insuffisant, le client en est informé et peut alors saisir un montant inférieur. Si le solde du compte est suffisant, le distributeur restitue la carte et délivre alors les billets accompagnés d'un reçu.
Décrire le fonctionnement de ce distributeur avec un diagramme d'activité.

Exercice 2 :

Une montre électronique contient deux boutons (bouton «mode», bouton « avance ») et fonctionne de la manière suivante :
L'heure est affichée, lorsqu'on appuis sur « mode » la montre passe en mode réglage de l'heure et la partie heure commence à clignoter : à chaque appui sur « avance », l'heure s'incrémente d'une unité. Si on appuis plus que deux secondes sur « avance », l'heure est modifiée rapidement jusqu'à ce qu'on relâche pour revenir au mode réglage normal. Lorsqu'on appui une nouvelle fois sur « mode », la montre passe en mode réglage des minutes et la partie minute commence à clignoter : à chaque appui sur « avance », la minute s'incrémente d'une unité. Si on appuis plus que deux secondes sur « avance », les minutes sont modifiées rapidement jusqu'à ce qu'on relâche pour revenir au mode réglage normal. On appuis une nouvelle fois sur « mode » pour revenir à l'affichage de l'heure.
Proposez un diagramme d'état transition.

Correction Fiche TD5





Troisième partie

Partie Travaux Pratiques

Fiche TP : Réalisation d'un mini Projet "Location de Voitures"

On souhaite réaliser un mini projet pour ne agence de location de voiture : Le gérant de l'agence peut afficher toute les voitures disponibles, puis , s'il le souhaite, ajouter une nouvelle voiture en introduisant son code, son matricule, sa marque et son année ou en supprimer une. Un client est Identifié par son nom, prénom et son adresse. Le client peut louer un ou plusieurs voitures. Une voiture appartient à une catégorie selon le type du permis de conduire. Chaque location d'un véhicule par un client est enregistrée avec la date début de la location et la date retour qui peut être prolongée.

- Le système proposé doit permettre les deux paramètres suivants :
- Le client peut prolonger la durée de location.
- Afficher la fiche de chaque voiture.

Réaliser un mini projet pour ce système de location de voitures.

Activité 1 : donner les différents diagrammes UML en utilisant Modélio.

Activité 2 : Création de la BDD

1- Proposer le modèle relationnel correspondant.

2- Démarrer Xamp, lancer PhpMyAdmin et créer la BDD « Location », ainsi que les différentes tables.

Activité 3 : Implémentation des classes

1- Créer un nouveau projet TpGLlocation

2- Créer les classes : Voiture, Client, Location, Catégorie avec les constructeurs, setter et getter

Activité 4 : Création de Frame

On commence par créer les deux frames, selon la figure ci-dessous,qui permettront de :

- Afficher toute les voitures (appelée AfficherVoitures).
- Ajouter une nouvelle voiture (Appeler Ajouter)

  Remarque : supprimer les méthodes Main() des deux JFrame.

Correction fiche TP

Activité 1 :

Diagramme de use case- Figure III

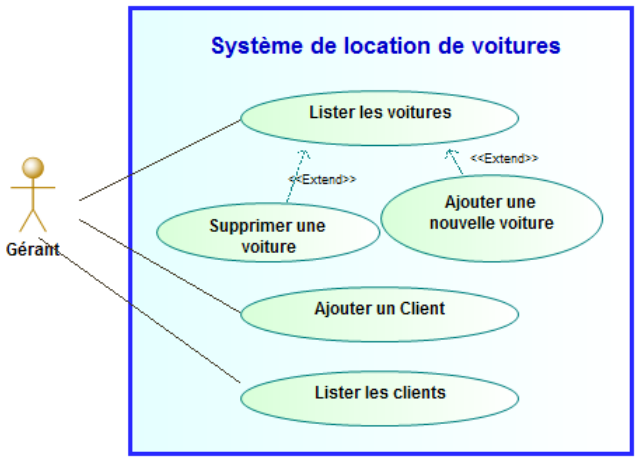


FIGURE 6.2 – use case

Diagramme de classe- Figure III

Activité 2 : Création de la BDD

1) Proposer le modèle relationnel correspondant.

Client(codeC, nomC,prenomC,adresseC)

Voiture(codeV, marque, matricule,annee, catégorie)

Louer(codeC,codeV,dateDebut, dateRetour, duree)

Categorie(numero, type, permis).

2) Démarrer Xamp, lancer PhpMyAdmin et créer la BDD « Location », ainsi que les différentes tables.

Reamarque : pour les clés étrangères (Foreign key) , exemple catégorie dans Voiture, sélectionner INDEX puis aller à structure-¿ vue relationnel

voir figure III et III

Activité 3 : Implémentation des classes

Créer un nouveau projet TpGLlocation

Créer les classes : Voiture, Client, Location, Catégorie avec les constructeurs, setter et getter

Activité 4 : Création de Frame

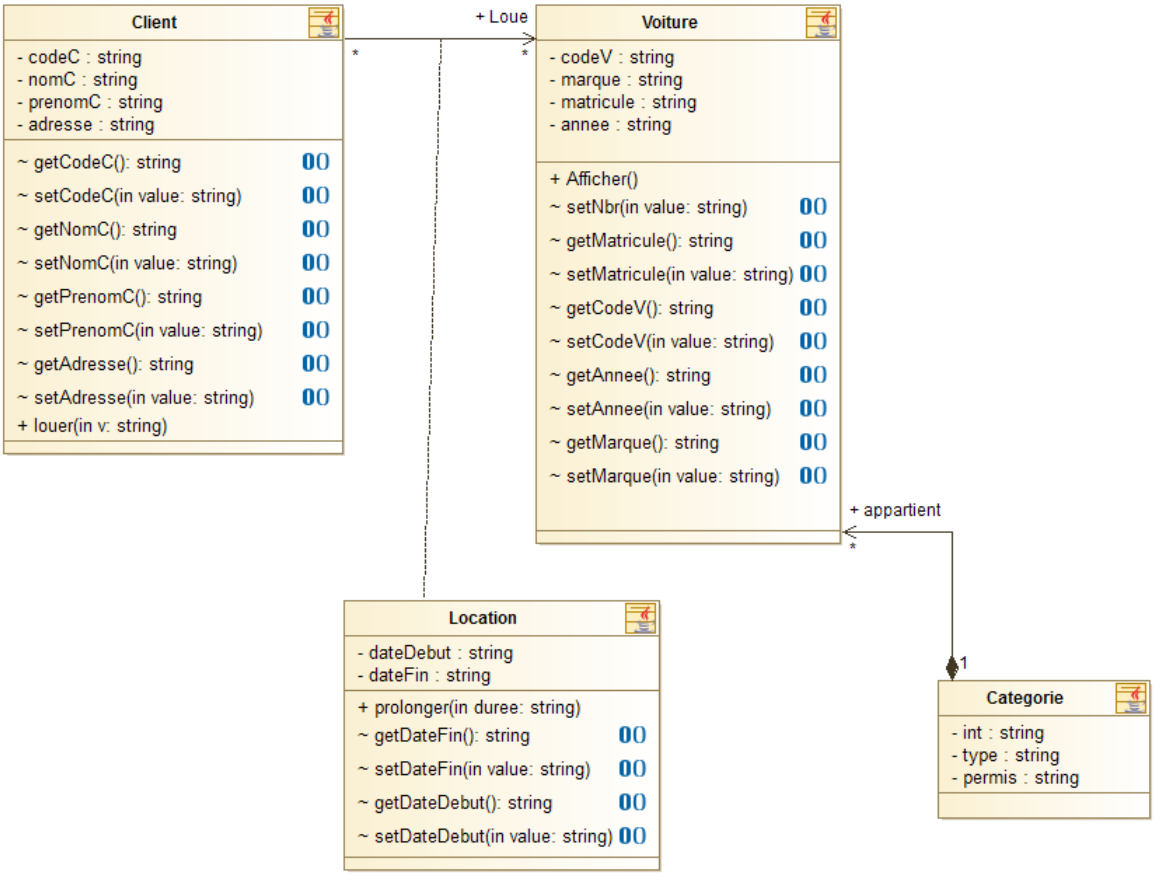


FIGURE 6.3 – use case

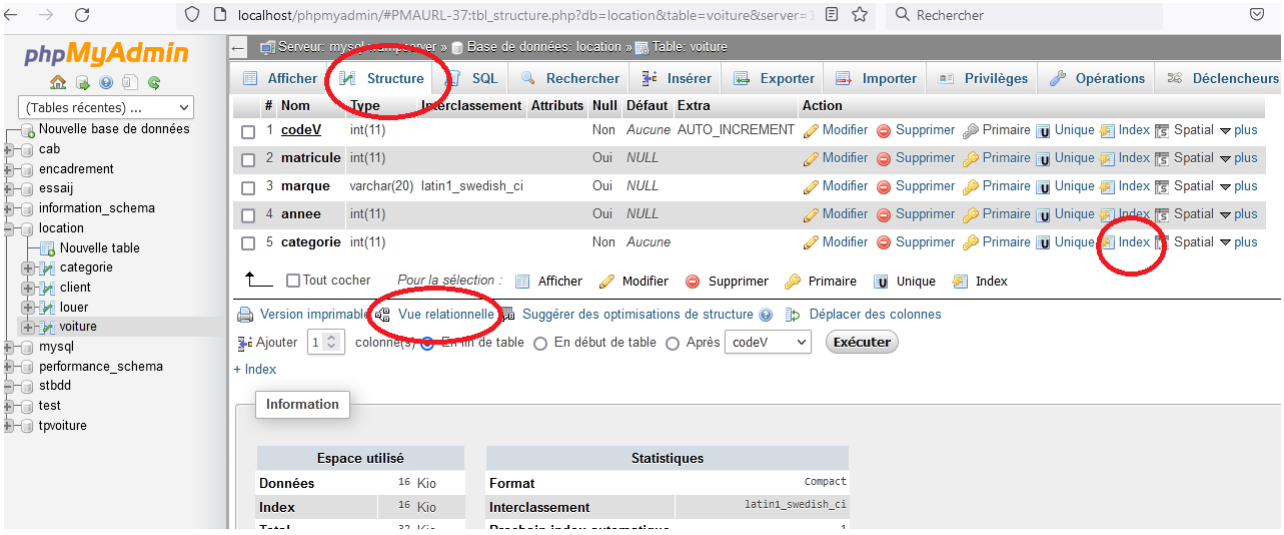


FIGURE 6.4 – phpMyAdmin-1

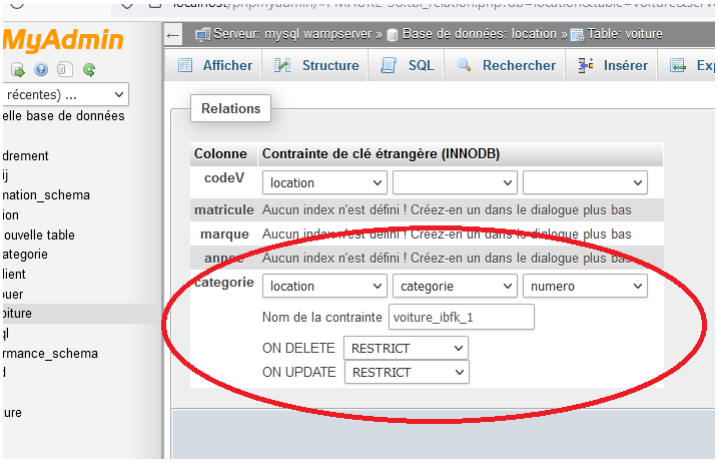


FIGURE 6.5 – phpMyAdmin-2

On commence par créer les deux frames, selon la figure ci-dessous,qui permettront de :

— Afficher toute les voitures (appelée AfficherVoitures).

— Ajouter une nouvelle voiture (Appeler Ajouter)

Remarque : supprimer les méthodes Main() des deux JFrame.

Ecrire sur main() du projet

```
public static void main(String[] args)
AfficherVoitures av=new AfficherVoitures();
av.setVisible(true);
```

Ecrire sur le bouton Ajouter ActionPerformed :

```
private void nouvelleActionPerformed(java.awt.event.ActionEvent evt)
Ajouter a=new Ajouter();
a.setVisible(true);
```

voir les figures III et III

Activité 4 : Connecter Java à la BDD

1- Ajouter le package mysql-connector-java-8.0.30.jar aux librairies du projet

Bouton droit Add Jar/Folder

2-Créer une classe de connexion appelée Connexion, et écrire le code de la figure III.

Ce lien explique bien comment se fait la connexion

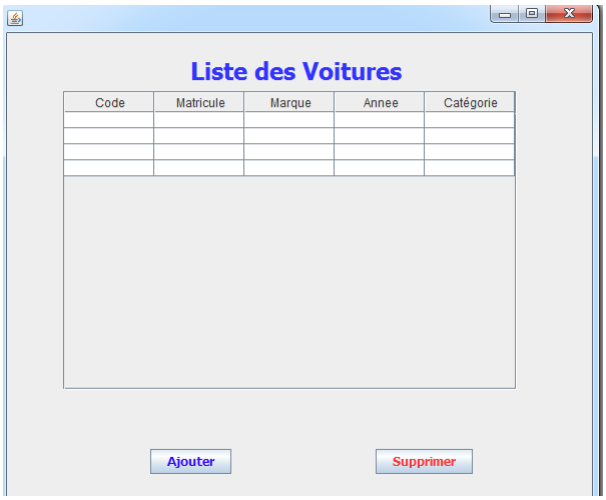


FIGURE 6.6 – interface1

FIGURE 6.7 – interface2

https ://www.jmdoudoux.fr/java/dej/chap-jdbc.htm

- On pouvait mettre * dans les import, mais pour ne pas gaqpier l'espace mémoire, on écrit que les librairies dont on a besoin.
- On crée le constructeur Connexion()
- JDBC est l'acronyme de Java DataBase Connectivity et désigne une API pour permettre un accès aux bases de données avec Java.
- Il y a 4 classes importantes : DriverManager, Connection, Statement (et PreparedStatement), et ResultSet, chacune correspondant à une étape de l'accès aux données , figure III

```java
package tpglnov;

import java.sql.Statement;
import java.sql.Connection;
import java.sql.DriverManager;

public class Connexion {
    Statement start;
     private void connect(){
        try{
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con=DriverManager.getConnection("jdbc:mysql://localhost/location","root","");
            start=con.createStatement();
            System.out.println("connexion établie");
        }catch(Exception e){
            e.printStackTrace();
        }
    }
      public  Statement getStart(){
       return start;
   }
}
```

FIGURE 6.8 – code java1

— La connexion à une base de données requiert au préalable le chargement du pilote JDBC qui sera utilisé pour communiquer avec la base de données. Une fabrique permet alors de créer une instance de type Connection qui va encapsuler la connection à la base de données.

— Pour se connecter à une base de données via ODBC, il faut tout d'abord charger le pilote JDBC-ODBC qui fait le lien entre les deux. Class.forName(”com.mysql.cj.jdbc.Driver”);

— Une clause catch contient les instructions à exécuter si une exception est levée par une instruction du bloc try . On souhaite généralement que le bloc try se déroule sans problème. Si toutefois une erreur se produit, on veut pouvoir contrôler ce qui se passe et on transmet donc le contrôle au bloc catch

— Le Statement contient des bibliothèques qui nous permettent d'écrire les requêtes SQL.

— Pour se connecter à une base de données, il faut instancier un objet de la classe Connection en lui précisant sous forme d'URL la base à accéder.

— La méthode getConnection() peut lever une exception de la classe java.sql.SQLException.

— Le code suivant décrit la création d'une connexion avec un user et un mot de passe : Connection con=DriverManager.getConnection(”jdbc :mysql ://localhost/location”,”root”,””);

Activité5 : Afficher le contenu d'une table sur une jFrame (java) à partir de MySql

Pour cela, il faut d'abord importer les données dans une ArrayListe puis la jTable.

Créer une méthode qui prend ce qu'il ya dans SQL

- Classe AfficherVoiture –¿On créer une méthode voituresList() de type ArrayList¡Voiture¿ On crée un objet de type connexion, un statement(qui nous permettra d'écrire nos requêtes)
- On déclare une arrayListe de type Voiture
- Lorsque nous avons une requête SQL, l'écrire à l'interieur du « Try-catch »
- Déclarer rs de type ResultSet pour contenir le résultat de la requête et écrire la requête
- Déclarer auto de type Voiture qui va utiliser le constructeur de Voiture pour importer les données des enregistrement.
- Ajouter ce nouvel objet créé à la liste.

On retourne cette liste dans notre méthode, figure III.

Classe	Rôle
DriverManager	Charger et configurer le driver de la base de données.
Connection	Réaliser la connexion et l'authentification à la base de donnée
Statement (et PreparedStatement)	Contenir la requête SQL et la transmettre à la base de donnée
ResultSet	Parcourir les informations retournées par la base de données c sélection de données

FIGURE 6.9 – les classes

Appeler la méthode showVoiture() dans le constructeur

Executer.

Si des ligne vides-¿ aller a Design-¿table Content–¿Delete

voir figure III

Activité 6 : Supprimer une voiture de la liste

On se connecte car on va supprimer aussi de la BDD, figure III

Activité 7 : Ajouter une nouvelle voiture

```
    public ArrayList<Voiture> VoitureList(){
    Connexion con=new Connexion();
    con.connect();
    Statement stat=con.getStart();
    // On déclare une arrayListe de type Voiture
    ArrayList<Voiture> listeDesVoitures= new ArrayList<>();
    try{

    ResultSet rs= stat.executeQuery("select * from voiture");
    Voiture auto;
    while(rs.next()){
    auto=new Voiture(rs.getInt("codeV"), rs.getInt("matricule"),rs.getString("marque"),rs.getInt("annee"),rs.getInt("
        listeDesVoitures.add(auto);
    }
    }catch(Exception e){
        JOptionPane.showMessageDialog(this, e.getMessage(),"err", JOptionPane.ERROR_MESSAGE);
    }
    return listeDesVoitures;
}
```

FIGURE 6.10 – code java 2

```
public void showVoiture(){
    // list reçoi la liste des voiture retournés par la méthode Voitur
    ArrayList<Voiture> list= VoitureList();
    // lorsque je veut appeler JTable j'appelle une copie avec l'instr
    // maTable est le nom de la variable de Jtable
    DefaultTableModel model= (DefaultTableModel)maTable.getModel();
    Object[] row= new Object[5];
    for (int i = 0; i < list.size(); i++) {
        row[0]=list.get(i).getCodeV();
        row[1]=list.get(i).getMatricule();
        row[2]=list.get(i).getMarque();
        row[3]=list.get(i).getAnnee();
        row[4]=list.get(i).getNumero();
        // ajouter
        model.addRow(row);

    }
}
```

FIGURE 6.11 – code java 3

Modifier le champs catégorie en Combo Box

Ecrire la méthode cat() et l'appeler du constructeur.

voire figure III

Pour ajouter un nouvel enregistrement à la BDD : écrire le code de la figure III

```
    private void suppActionPerformed(java.awt.event.ActionEvent evt) {
        Connexion con= new Connexion();
        con.connect();
        Statement stat=con.getStart();
        DefaultTableModel model= (DefaultTableModel)maTable.getModel();

        // reccuoérer la ligne sélectionée
        int ligne=maTable.getSelectedRow();
        //ligne=-1 veut dire que la ligne n'est pas sélectionnée
        if(ligne !=-1){
        try{
        int id=(int)maTable.getModel().getValueAt(ligne,0);//0 veut dire le premier champs (codeV)
        int deleted=stat.executeUpdate("delete from voiture where codeV="+id);
        model.removeRow(ligne);
        if (deleted>0){
            JOptionPane.showMessageDialog(null, "suppression OK");
        }else{JOptionPane.showMessageDialog(null, "Pas de suppression");}
}catch(Exception e){
    JOptionPane.showMessageDialog(this, e.getMessage(),"erreur de suppression", JOptionPane.ERROR_
}}
        else{JOptionPane.showMessageDialog(null,"Veuillez selectionner une ligne svp","errrrrr", J
}
    }
```

FIGURE 6.12 – code java4

```
    public Ajouter() {
        initComponents();
        cat();
    }
    public void cat(){
        Connexion co=new Connexion();
        co.connect();
        Statement stat=co.getStart();
        try{
           ResultSet res=stat.executeQuery("select numero from categorie");
           while(res.next()){
               comboCat.addItem(res.getString("numero"));
           }
        }catch(Exception e){
        JOptionPane.showMessageDialog(this, e.getMessage(),"err", JOptionPane.ERR
    }
    }
```

FIGURE 6.13 – code java 5

```
private void AjouterActionPerformed(java.awt.event.ActionEvent evt) {
    //si le champs est vide
    if(textCodeV.getText().equals("")){
        JOptionPane.showMessageDialog(this, "le champs est vide","veuillez remplir svp", JOptionPane.ERROR_MESS.
    }else if(comboCat.getSelectedItem()==""){
      JOptionPane.showMessageDialog(this, "catégorie vide","err", JOptionPane.ERROR_MESSAGE);
    }else{
        Connexion con=new Connexion();
            con.connect();
            Statement stat=con.getStart();
        try{

            stat.executeUpdate("INSERT INTO voiture VALUES('"+Integer.parseInt(textCodeV.getText())+"','"+Integ
        JOptionPane.showMessageDialog(this, "ok","voiture ajoutée", JOptionPane.ERROR_MESSAGE);

        }catch(Exception e){
    JOptionPane.showMessageDialog(this, e.getMessage(),"err", JOptionPane.ERROR_MESSAGE);
}


    }
}
```

FIGURE 6.14 – code java 6

Bibliographie

[1] UML Distilled : A Brief Guide to the Standard Object Modeling Language" de Martin Fowler.

[2] UML Introduction au génie logiciel et à la modélisation" de Delphine Longuet

[3] Applying UML and Patterns : An Introduction to Object-Oriented Analysis and Design and Iterative Development" de Craig Larman.

[4] UML 2.0 in a Nutshell" de Dan Pilone et Neil Pitman.

[5] UML 2 -De l'apprentissage à la pratique" de Laurent AUDIBERT

[6] The Unified Modeling Language User Guide" de Grady Booch, James Rumbaugh et Ivar Jacobson.