République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la technologie Houari Boumediene

U.S.T.H.B

Faculté D'Electronique et Informatique

Département d'informatique

Projet GL3

Thème : Ascenseur virtuel

Réalisé par :

1) AIT AMARA Mohamed L2 ISILB G1 181831072170

Email : mohm2733@gmail.com

2) SOUIDI Mohamed Amine L2 ISILB G1 181831044438

Email: mouhfcb1970@gmail.com

Année universitaire : 2020/2021

Résumé

L'objectif principal de ce projet consiste à concevoir et développer une application java pour l'implémentation d'un ascenseur virtuel afin de maîtriser les testes avec Junit.

Mots clés : Java, Junit, ascenseur virtuel.

Abstract

The main objective of this project is to design and develop a java application for the implementation of a virtual elevator in order to master the tests with Junit.

Keywords: Java, Junit, virtual elevator.

Introduction général

L'informatique cette science de travail rationnel de l'information est considérée comme le support des connaissances dans les domaines scientifiques, économiques et sociaux notamment à l'aide des machines automatique. Le monde connaît une avance technologique considérable dans tous les secteurs qui étudient les techniques du traitement automatique de l'information de l'entreprise et d'autres établissements. L'informatisation est donc le phénomène le plus important de notre époque. Elle s'immisce maintenant dans la plupart des objets de la vie courantes et ce, que ce soit dans l'objet proprement dit, ou bien dans le processus de conception ou de fabrication de cet objet. Et dans ce cas nous avons utilisé cette technologie de l'informatisation pour contrôleur un ascenseur,

Dans ce cadre nous mettons un programme pour mettre en œuvre un contrôleur d'ascenseur.

Nous vous présentons dans notre première partie le contexte de notre travail.

La deuxième partie contiendra la spécification du système

Et La troisième partie sera consacrée pour l'étude conceptuelle de notre application suivant le processus UP d'UML (Unified Modeling Language).

À travers ce document, nous allons décrire en détail les différentes étapes de réalisation de ce projet.

Table of Contents

Chapitre 1 : Contexte et objectif

1 Introduction

Au cours de ce chapitre, nous allons introduire notre projet en étudiant son cadre général.
La présentation de la solution souhaitée nous mène à bien analyser les déférentes méthodologies de travail existantes afin de dégager celle la plus adéquate à notre projet.

2 Problématique

Notre problématique est de réaliser un ascenseur virtuel dont on va l'utiliser pour faire plusieurs tests avec.

3 Solution proposée

Après avoir détaillé notre problématique et bien analysé les différentes solutions existantes, la question suivante doit se poser naturellement :
Cependant, nous avons besoin d'une solution spécifique, qui répond parfaitement aux exigences spécifiées dans l'énoncé du projet, tout en tenant compte de la marge de créativité permise.

4 Méthodologie de travail et modélisation

Une méthodologie de développement est un cadre utilisé pour structurer, planifier et contrôler le développement d'une application. C'est le fait de modéliser un système avant sa réalisation pour bien comprendre son fonctionnement et assurer sa cohérence. Un modèle est ainsi un facteur de réduction des coûts et des délais. Il est donc indispensable pour assurer un bon niveau de qualité de produit dont la maintenance est efficace.
Nous présentons la principale méthodologie de développement dédiée aux démarches de conception orientées objet pour monter le processus suivi dans ce travail.

4.1 Le Processus Unifié (UP)

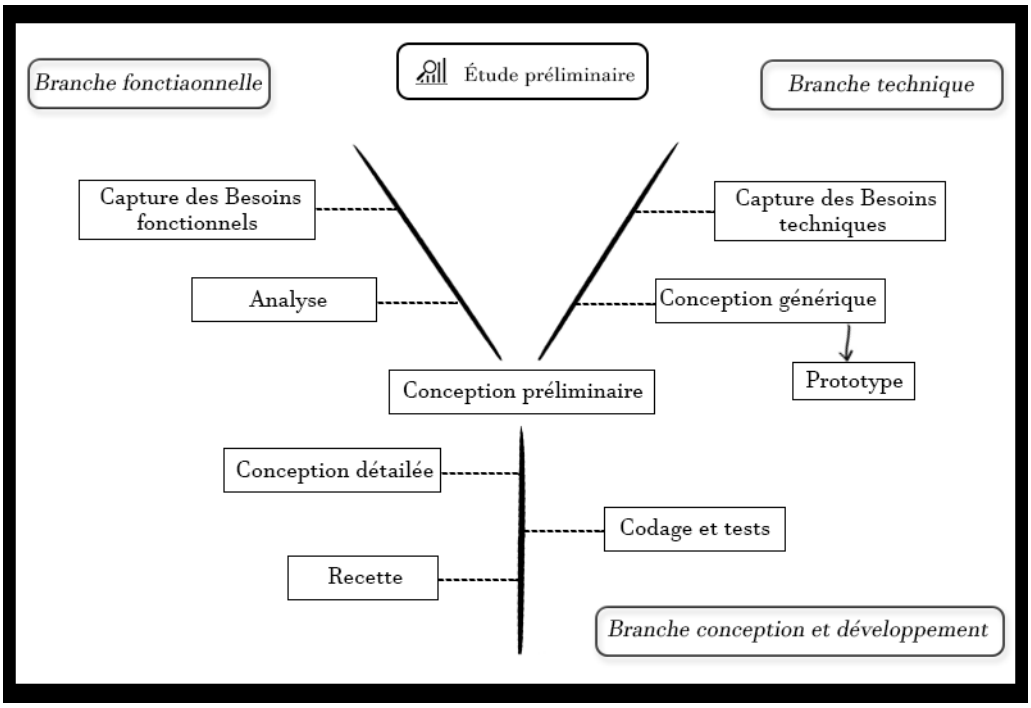
Le Processus Unifié est un processus de développement logiciel organisée suivant les quatre phases suivantes :
1. La phase d'initialisation permet de décider la poursuite du projet ou son arrêt.
2. La phase d'élaboration poursuit trois objectifs principaux en parallèle : (identifier et décrire, construire, lever les risques)
3. La phase de construction consiste à concevoir et implémenter l'ensemble des éléments opérationnels.
4. En fin, la phase de transition permet de faire passer le système informatique des mains des développeurs à celles des utilisateurs finaux.

4.1.1 Conclusion et choix de la Méthodologie à suivre :

Notre projet est basé sur un processus de développement bien défini qui va de la détermination des besoins fonctionnels attendus du système jusqu'à la conception et le codage final. C'est pour cela qu'on est besoin d'un un cycle de développement qui dissocie les aspects techniques des aspects fonctionnels tout en commençant par une étude préliminaire. Notre choix s'est alors porté vers la méthode 2TUP vu qu'elle est caractérisée par une approche nouvelle et originale et qu'elle respecte le cadre de notre projet.
Nous adaptons alors le processus 2TUP pour la suite de notre travail.

4.2 Mise en pratique du processus 2TUP

2 Tracks Unified Process est un processus unifié (c'est-à-dire construit sur UML, itératif, centré sur l'architecture et conduit par les cas d'utilisation). Ce processus de développement logiciel est caractérisé par le suivant :
→ Itératif et incrémental : Le projet est découpé en itérations de courte durée (environ
1 mois) qui aident à mieux suivre l'avancement global.
→ Centré sur l'architecture : Tout système complexe doit être décomposé en parties modulaires afin de garantir une maintenance et une évolution facilitées.
→Piloté par les risques : Les risques majeurs du projet doivent être identifiés au plus tôt, mais surtout levés le plus rapidement possible.
→ Conduit par les cas d'utilisation : Les cas d'utilisation du futur système sont identifiés et décrits avec précision.



Les étapes du processus 2TUP

Le principe fondateur du 2TUP est que toute évolution imposée à un logiciel peut se décomposer et se traiter parallèlement, suivant un axe fonctionnel et un axe technique. La réalisation du logiciel consiste à fusionner les résultats de ces deux branches du processus.
Selon le processus 2TUP que nous suivons, nous passons obligatoirement par les phases suivantes à la suite de notre travail :
1. L'étude préliminaire qui contient une description du service et les cas d'utilisation principaux, c'est une première version de la spécification générale.
2. La capture des besoins fonctionnels, qui définit le quoi faire à travers une spécification générale qui décrit le service à développer d'un point de vue fonctionnel et une spécification détaillée qui précise les traitements qui concernent chaque scénario des cas d'utilisation présent en spécification générale tout en respectant les contraintes fonctionnelles et non fonctionnelles.
3. L'Analyse où on effectue simultanément l'étude des données et l'étude des traitements à effectuer.
4. La capture des besoins techniques qui permettent de satisfaire les contraintes techniques présentes dans le cahier des charges et donc répondre aux attentes de client.
5. La conception générique qui définit le comment faire.
6. La conception préliminaire qui liste les exigences techniques à partir de l'étude préliminaire et définie l'architecture technique et les choix d'implémentation technique.
7. La conception détaillée qui précise l'implémentation technique de l'application. Elle consiste en la fusion de la spécification détaillée et de la conception générique, pour déterminer comment faire le quoi faire dans le détail. On y trouve le schéma de base de données, les diagrammes de classes et les

diagrammes de séquence supplémentaires qui détaillent les interactions entre les composants du logiciel.
8. Le Codage et les Tests qui décrivent comment est l'application est réalisée et installée sur un environnement d'exécution et comment la stratégie de validation est effectuée.

5 Conclusion

Dans ce premier chapitre, nous avons posé la problématique et la solution envisagée pour garantir la bonne entame de notre projet. Nous avons clôturé par une présentation de la méthodologie 2TUP que nous allons personnaliser un petit peu pour l'adopter à nos besoins et notre stratégie de développement de notre projet. Le chapitre suivant sera consacré l'étude préliminaire de notre projet.

Chapitre 2 : Etude préliminaire

1 introduction

Comme les bonnes questions représentent la moitié de la réponse dans la plupart des domaines, en informatique une bonne spécification des besoins est primordiale. En effet, elle représente le travail le plus délicat et le plus significatif, mais elle-même repose sur une bonne spécification des besoins. Dans cette section nous allons comprendre le contexte du système, il s'agit de déterminer les fonctionnalités et les acteurs et d'identifier les cas d'utilisation initiaux. Ainsi que la spécification des besoins. Nous allons utilisé une méthode 2TUP réduite qui contiendra 4 diagrammes car nous n'avons pas beaucoup de classes.

2 Description du contexte

2.1 Définition des acteurs :

Notre application contient un seul acteur qui est l'utilisateur. Ce dernier va déclencher l'ascenseur virtuel.

2.2 Diagramme de cas d'utilisation préliminaire (général) :

Notre programme contient un seul utilisateur qui va déclencher l'ascenseur.

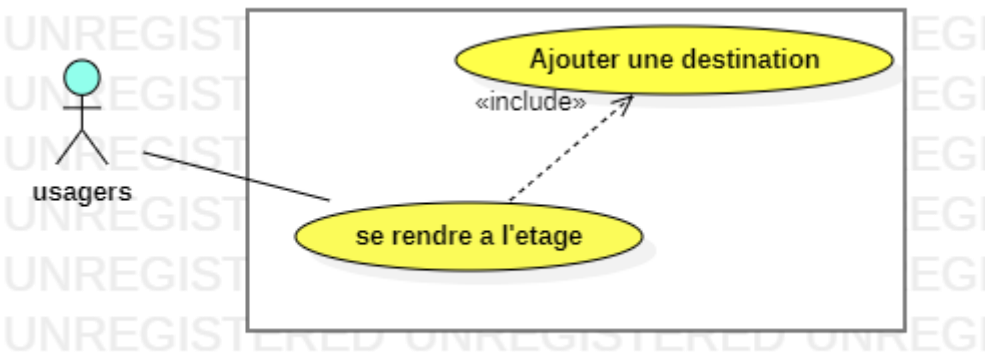


Figure : Diagramme de cas d'utilisation général

3 Besoins fonctionnels

Les besoins fonctionnels représentent les principales fonctionnalités du système. Ces besoins proviennent généralement des utilisateurs du système. Cette application va permettre à l'utilisateur de choisir la direction et destination de l'ascenseur.

4 Besoins non fonctionnels

4.1 Exigences de qualité

Ergonomie sobre et efficace : Utiliser une simple application java ne doit pas prendre beaucoup de temps ou demander une maîtrise de mathématiques ! L'ergonomie de notre application facilitera au maximum la démarche à l'aide d'une présentation claire et intuitive des messages affichés dans le terminal.

4.2 Exigences de performance

Besoin de sécurité : L'application doit garantir à l'utilisateur connecté l'intégrité et la confidentialité de ses données. La sécurité du système est assurée par l'authentification des clients par un login et un mot de passe crypté.
Performance : L'application doit fournir tous les statuts et informations en temps réel et d'une manière optimale. *La flexibilité :* Une ouverture sur d'autres domaines serait un atout majeur pour notre application. Ce point ne doit pas être mis à côté lors de la conception de l'architecture du projet.
L'intégrité : Le système doit savoir comment traiter les échecs des traitement, faire la capture des différentes erreurs d'entrées-sorties, effectuer le traitement des mauvaises données sans oublier l'intégrité des informations en entrées-sorties.
La portabilité : L'application doit s'exécuter sans problème sur tous les navigateurs. Il est ainsi primordial qu'elle soit compatible avec les différents systèmes d'exploitation.

5 Besoins techniques

Dans cette section on va juste dire pourquoi on a utilisé le langage java car on va détaillé les besoins techniques ultérieurement.

5.1 Pourquoi Java ?

Pour plusieurs raisons, nous allons adopter le langage Java pour développer notre application de gestion. Tout d'abord, Java est gratuit et ne nécessite aucune licence d'utilisation. Java est le langage de programmation le plus utilisé dans des applications similaires. Il existe une communauté de développeurs très active qui rend disponibles des dizaines de milliers de librairies Java de grande qualité ainsi qu'une vaste quantité de documentation et tutoriels pour le bénéfice de chacun notamment pour la libraire. Ces ressources facilitent notre travail et réduisent notre temps d'exécution.

6 Conclusion

Durant ce chapitre, Nous avons non seulement exposé la mise en pratique du processus 2TUP, mais nous avons également capturé les différents besoins de notre projet. Nous avons classé ces besoins en besoins fonctionnelles, non fonctionnelles et techniques. Après l'analyse des exigences et le choix des techniques a utilisé, Il est temps de passer à la conception de notre application dans le chapitre suivant.

Chapitre 3 : Conception générique

1 Introduction

Dans cette partie nous traitons l'aspect conceptuel de notre application. Pour la conception et la réalisation de cette dernière, nous nous appuyons sur le formalisme UML basé sur les diagrammes et offrant une flexibilité marquante. La phase de conception permet de décrire de manière non ambiguë, le plus souvent en utilisant un langage de modélisation, le fonctionnement futur du système, afin d'en faciliter la réalisation.

2 Le langage UML

Pour faciliter notre tâche nous avons recours au langage de modélisation unifié (UML :
Unified Modelling Language) c'est une notation qui permet de modéliser un problème de façon standard. Ce langage qui est né de la fusion de plusieurs méthodes existantes auparavant est devenu une référence en termes de modélisation objet, UML est caractérisé par :
→C'est un langage formel et normalisé.
→ Il permet le gain de précision, encourage l'utilisation d'outils et constitue à cet effet un gage de stabilité.
→UML est un support de communication performant.
→Il cadre l'analyse et facilite la compréhension de représentations abstraites complexes.
Son caractère polyvalent et sa souplesse en font un langage universel.

3 Les diagrammes d'UML

UML 2 s'articule autour de treize types de diagrammes, chacun d'eux étant dédié à la représentation des concepts particuliers d'un système logiciel. Ces types de diagrammes sont répartis en deux grands groupes :

3.1 Six diagrammes structurels :

Diagramme de classes : Il montre les briques de base statiques : classes, associations, interfaces, attributs, opérations, généralisations, etc.
Diagramme d'objets : Il montre les instances des éléments structurels et leurs liens à l'exécution.
Diagramme de packages : Il montre l'organisation logique du modèle et les relations entre packages.
Diagramme de structure composite : Il montre l'organisation interne d'un élément statique complexe.
Diagramme de composants : Il montre des structures complexes, avec leurs interfaces fournies et requises.
Diagramme de déploiement : Il montre le déploiement physique des 'artefacts' sur les ressources matérielles.

3.2 Sept diagrammes comportementaux :

Diagramme de cas d'utilisation : Il montre les interactions fonctionnelles entre les acteurs et le système à l'étude.
Diagramme de vue d'ensemble des interactions : Il fusionne les diagrammes d'activité et de séquence pour combiner des fragments d'interaction avec des décisions et des flots.
Diagramme de séquence : Il montre la séquence verticale des messages passés entre objets au sein d'une interaction.
Diagramme de communication : Il montre la communication entre objets dans le plan au sein d'une interaction.
Diagramme de temps : fusionne les diagrammes d'états et de séquence pour montrer l'évolution de l'état d'un objet au cours du temps.
Diagramme d'activité : Il montre l'enchaînement des actions et décisions au sein d'une activité.
Diagramme d'états : Il montre les différents états et transitions possibles des objets d'une classe.

4 Modèles d'UML utilisés

Les diagrammes qu'on a montré dans la partie précédente sont des moyens de description des objets ainsi que des liens qui les relient. Nous n'utilisons pas les treize types de diagrammes proposés par UML 2, mais seulement trois d'entre eux en insistant particulièrement sur les diagrammes suivants :
Diagramme de cas d'utilisation (présenté dans le chapitre précédant).
Diagramme de séquence.
Diagramme de classes.
Cette limitation est largement suffisante pour la plupart des projets et particulièrement pour notre travail.

Chapitre 4 : Conception détaillée

1 Capture des besoins fonctionnels

1.1 Diagrammes de séquence système

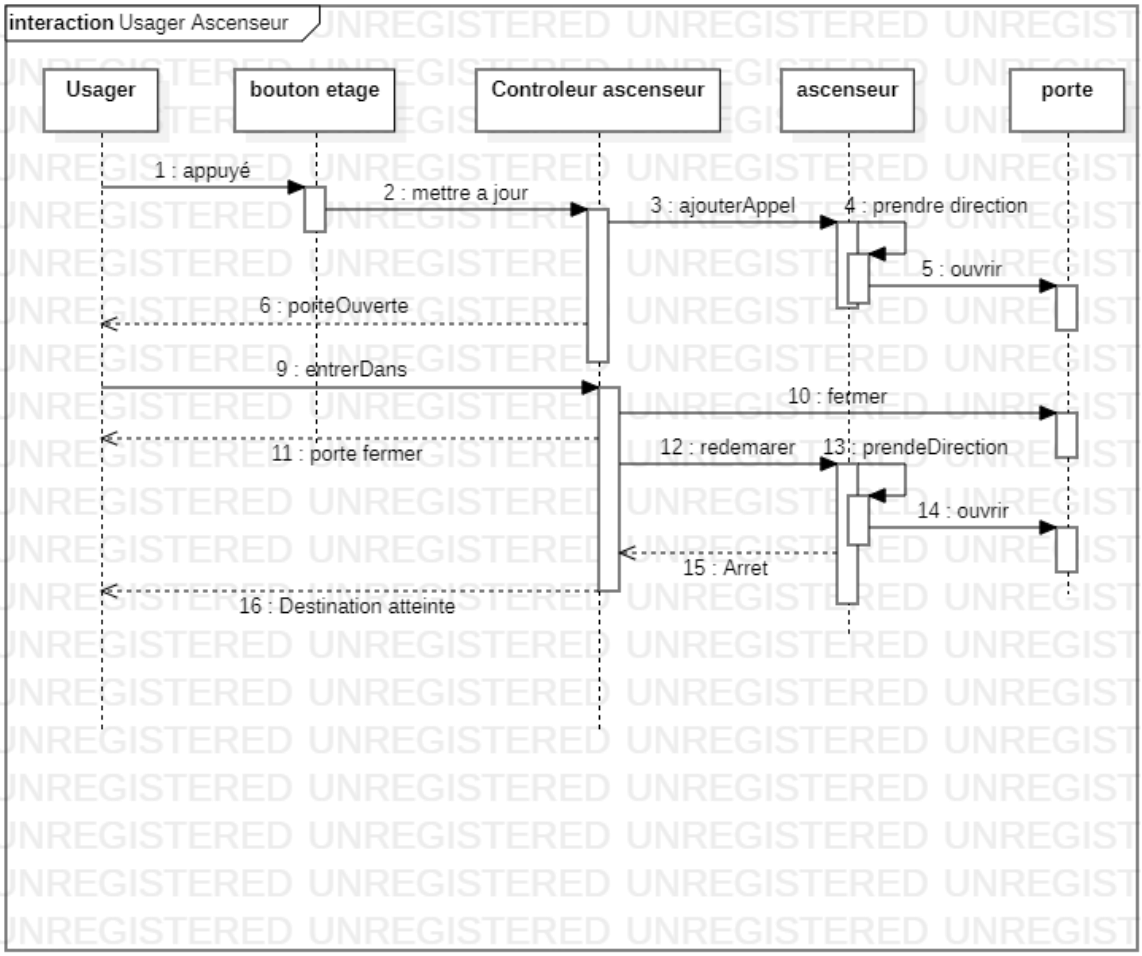


Figure : Diagramme de séquence système

1.2 Voici les classes d'equivalence :

Classe	Donnée en entrée	Domaine définition	Classe valide	représentant	Classe invalide	Représentant
Usager	Etage Destination	Int Int	[0 nbrMaxEtage]	RV = 1	CI_1 =]-∞ , -MaxInt [∪]MaxInt , +∞[CI_2 = [-MaxInt , 0 [	RI1 = 2 * 10^{10} RI2 = -2

	Direction	Direction	CV1 = { “Up”} CV2 = {“DOWN”} CV3 = {“NONE”}	RV1 = UP RV2 = DOWN RV3 = NONE	CI1 =author direction	RI1 = left
Porte	etage POuverte	Int Boolean	[0 nbrMaxEtage] CV1 = { “True”} CV2 = {“False”}	RV = 1 RV1 = True RV2 = False	CIE_1 =]-∞,-MaxInt [∪]MaxInt , +∞[CIE_2 = [-MaxInt , 0 [CIp =le reste des chaine	RI1 = $2 * 10^{10}$ RI2 = -2 RIp = “bb”
Controleur	ListePorte ListeApel Destination Evenement	Ensemble type port Ensemble type direction Int String	CV1 = { Portes } 0≤taille≤nbrMaxEtage CV = {direction} CV= [0 nbrMaxEtage] CV =tout chaine qui Représente l’état de l’ascenseur	RV =liste des poetes taille < NbrEtages RV =liste des directions RV = 1 RV = “ascenseur redémarrer ”	CIL_1 = taille >NbrEtage CIL_1= type des éléments différentes a porte CIL_1= type des éléments différentes a direction CI_1 =]-∞ , -MaxInt [∪]MaxInt , +∞[CI_2 = [-MaxInt , 0 [CI =les autre chaine	RIL_1 = taille=NbrEtage+1 RIL_2 ={ Usager } RIL_2 = {porte} RI = -2 RIp = “bxb..”
Ascenseur	Etage Direction	Int Direction	[0 nbrMaxEtage] CV1 = { Up, DOWN ,NONE}	RV = 1 RV1 = UP	CI_1 =]-∞ , -MaxInt [∪]MaxInt , +∞[CI_2 = [-MaxInt , 0 [CI =author direction	RI1 = $2 * 10^{10}$ RI2 = -2 RI1 = gauche

Les classes d’equivalece

2 Analyse

2.1 Diagramme de classe

Les diagrammes de classes de conception représentent bien la structure statique du code, par le biais des attributs et des relations entre classes, mais ils contiennent également les opérations (aussi appelées méthodes) qui décrivent les

responsabilités dynamiques des classes logicielles. L'attribution des bonnes responsabilités aux bonnes classes est l'un des problèmes les plus délicats de la conception orientée objet. Pour chaque service ou fonction, il faut décider quelle est la classe qui va le contenir.

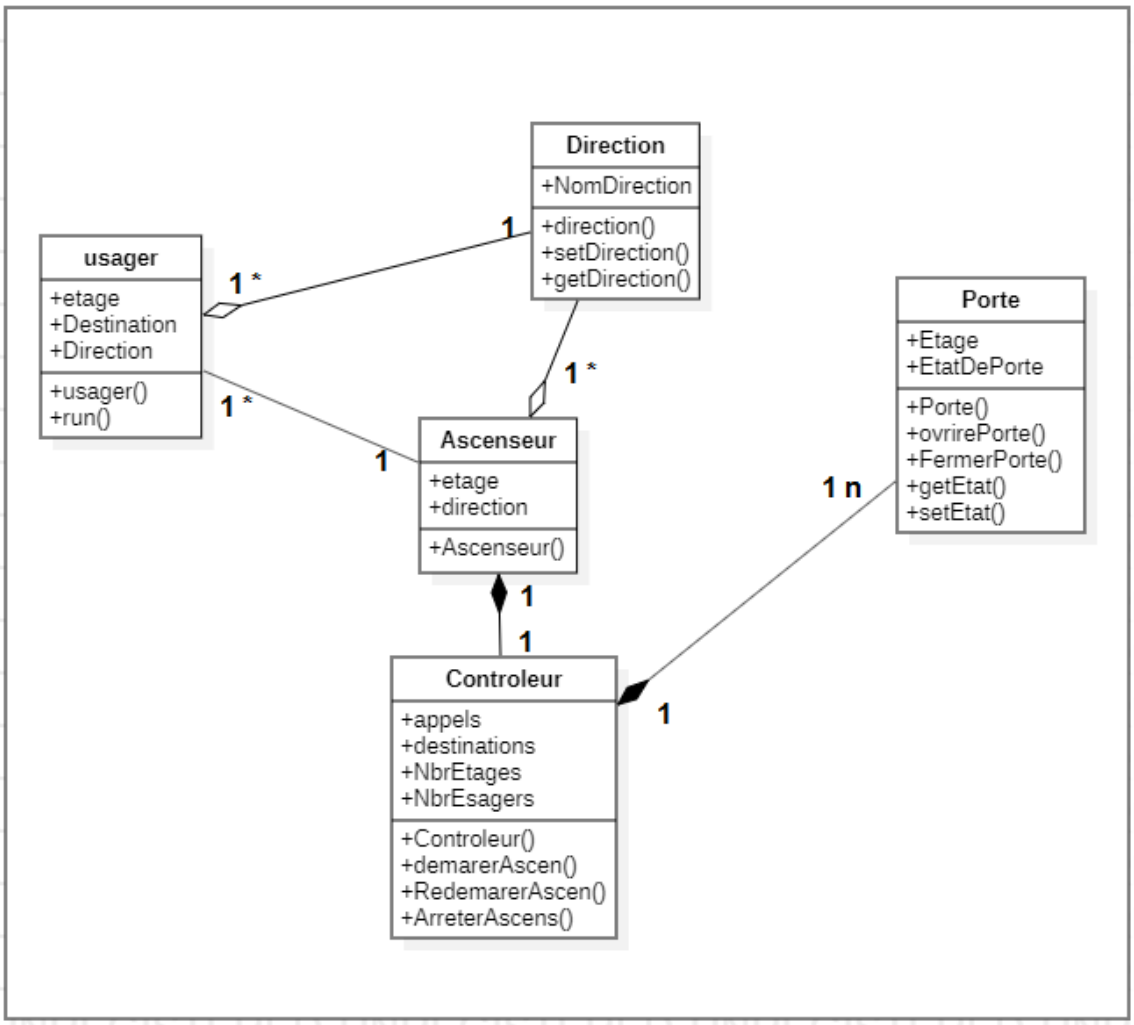


Figure : Diagramme de classe

Chapitre 5 : Implémentation et réalisation

1 Introduction

La phase qui suit une conception est éventuellement l'implémentation. Après les 4 chapitres précédents, nous arrivons à rassembler nos informations et réaliser notre système, nous commençons ce chapitre par décrire l'environnement de

développement matériel et logiciel que nous avons adoptés. Nous présentons ensuite quelques captures d'écrans de notre programme.

2 Environnement de développement

2.1 Environnement matériel

Pour la réalisation de ce projet on a disposé d' :

. Un ordinateur :

_ Marque : HP
_ Processor : Intel(R) Core(TM) i3-5500U CPU 2.50GHz x 2
_ RAM : 4Go
_ Disque dur : 500Go
_ Système d'exploitation : Windows 8.1

. Un ordinateur :

_ Marque : Asus
_ Processor : Intel(R) Core(TM) i5-8500U CPU 1.80GHz x 4
_ RAM : 8Go
_ Disque dur : 1To HDD + 128Go SSD
_ Système d'exploitation : Windows 10

2.2 Environnement logiciel

Cette section décrit l'environnement logiciel avec lequel nous avons réalisé ce projet avec le rapport.

2.2.1 NetBeans IDE :

est un environnement de développement intégré (IDE) développé par « The appache software fondation ». NetBeans est un IDE complet misant sur la productivité avec des systèmes d'auto-complétion intelligente, d'analyse de code en temps réel, de refactoring avancé l'intégration d'outils de tests et de debugging et une pléthore de raccourcis clavier permettant de réaliser presque n'importe quelle tâche rapidement sans jamais lever les mains du clavier pour utiliser la souris.

2.2.2 Microsoft word :

est un logiciel de traitement de texte publié par Microsoft.

2.2.3 Astah :

Anciennement appelé **Jude**, **Astah** est un outil de modélisation UML créé par la compagnie japonaise ChangeVision[1]. Il fonctionne avec l'environnement d'exécution Java. Le nom vient de l'acronyme *Java and UML developers' environment*.

3 Présentation de la solution

Nous allons présenter dans cette partie la description textuelles des tests.

Le test de la classe Porte :

Nous avons effectué ce test ci-dessous pour vérifier l'état de la porte « la porte est fermée ou non »

```
@Test
public void testPorte()
{
    System.out.println("\n-> ouverture/fermeture d'une porte au 2eme l'etage\n");

    // Valider le comportement : la porte est bien fermee?
    assertFalse(controleur.portes[1].check_porteOuverte());
    assertEquals(controleur.etageArret,-1);
    System.out.println("1-- La "+controleur.portes[1]+" est fermee, et aucun signal d'arret.");


    // Simuler l'arr◆t de l'ascenseur au 1er etage
    controleur.etageArret=2;

    // Attendre ouverture de la porte (1/1 delai)...
    try { Thread.sleep(500); }
    catch(InterruptedException e) { System.out.print("Erreur dans Thread.sleep\n"); }

    // Valider le comportement : la porte est ouverte?
    assertTrue(controleur.portes[1].check_porteOuverte());
    System.out.println("2-- La "+controleur.portes[1]+" est bien ouverte.");

    // Attendre fermeture de la porte (delai complet)...
    try { Thread.sleep(3000); }
    catch(InterruptedException e) { System.out.print("Erreur dans Thread.sleep\n"); }

    // Valider le comportement : la porte s'est refermee?
    assertFalse(controleur.portes[1].check_porteOuverte());
    System.out.println("3-- La "+controleur.portes[1]+" est refermee.");
}
```

Le test sue la classe usager :

Nous avons effectué ce test pour s'assurer de l'étage de l'appel de l'usager et aussi l'arrivée de ce dernier.

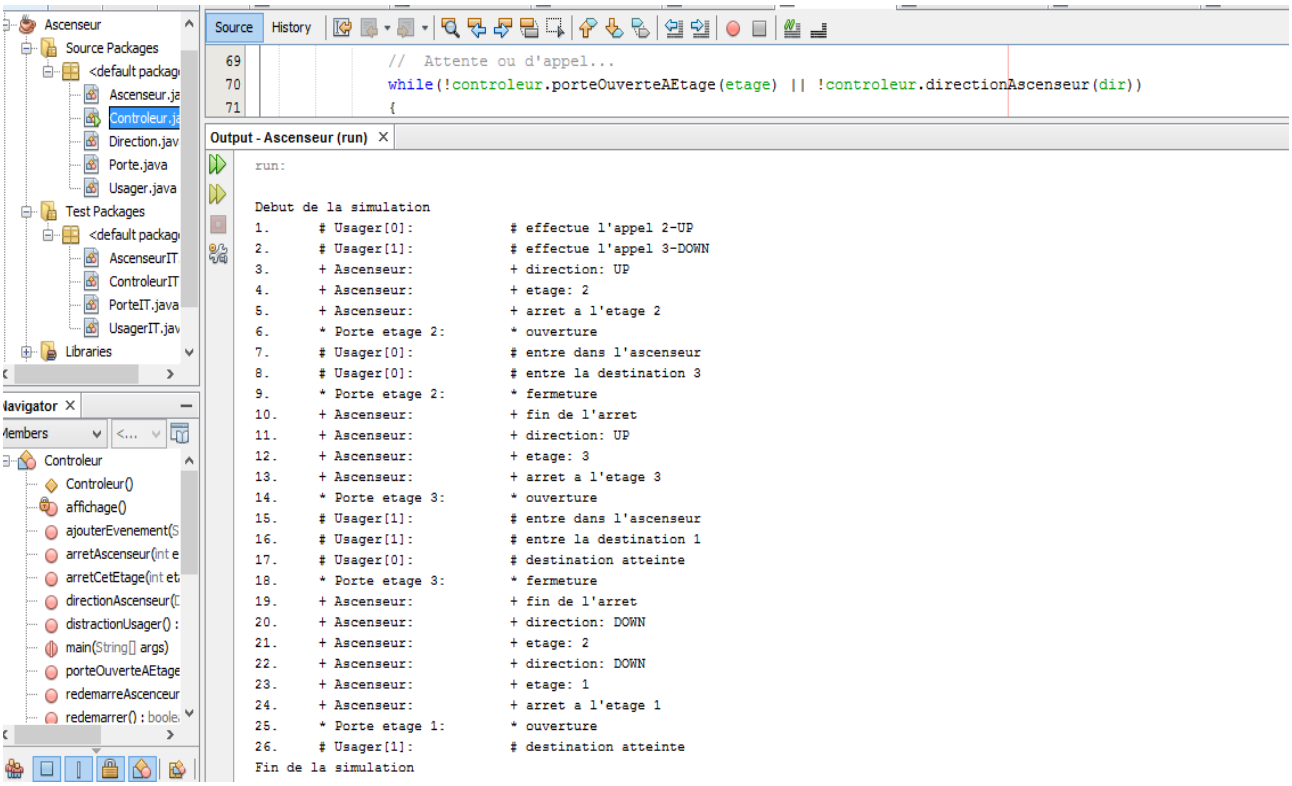
```java
@Test
public void testUsager() {
    System.out.println("\n-> usager effectue appel au 2eme l'etage\n");
    assertEquals(controleur.usager0.getEtage(), 2);
    System.out.println("1-- L' " + controleur.usager0.toString() + " est au 2eme etage sa direction et vers le haut");


    // Attendre ouverture de la porte (1/1 delai)...
    try { Thread.sleep(5000); }
    catch(InterruptedException e) { System.out.print("Erreur dans Thread.sleep\n"); }

    assertEquals(controleur.ascenseur.etage, 3);
    System.out.println("Usager bien arrivee");



}
```

Le test de la classe Ascenseur :

Dans ce test on vérifie le choix de la direction de l'ascenseur.

```java
@Test
public void testChoisirDirection()
{
    System.out.println("\n->  choisir Direction\n");
    controleur.appels[1] =Direction.UP;
    System.out.println("(a) L'"+controleur.ascenseur.toString()+" est au 1er etage");

    // Valider le comportement :
    assertEquals(controleur.ascenseur.choisirDirection(), Direction.UP);

    System.out.println("(b) La direction choisie est bien UP");

    try { Thread.sleep(1000); }
    catch(InterruptedException e) { System.out.print("Erreur dans Thread.sleep\n"); }

    System.out.println("\n->  choisir Direction  2\n");

    controleur.appels[1] =Direction.DOWN;
    System.out.println("(a) L'"+controleur.ascenseur.toString()+" est au 3eme etage");

    // Valider le comportement :
    assertEquals(controleur.ascenseur.choisirDirection(), Direction.DOWN);

    System.out.println("(b) La direction choisie est bien DOWN");

}
```

Voici le résultat affiché de notre programme :



4 Conclusion

Dans ce chapitre, nous avons présenté l'environnement matériel et logiciel utilisé lors du développement de l'application. Ensuite, on a présenté quelques captures d'écran montrant le bon fonctionnement de notre application.

Conclusion générale

Le présent rapport est réalisé dans le cadre de notre projet du module GL3.
Pour pouvoir compléter notre mission, nous avons détaillé les différentes étapes d'analyse, de conception et de réalisation de ce système. Nous avons utilisé UML comme un langage de modélisation et 2TUP comme un processus de développement.
Ce projet était bénéfique pour nous dans plusieurs sens. Il nous a permis de maîtriser l'aspect teste et validation du génie logiciel et il était une occasion pour améliorer nos connaissances dans les environnements de travail et de maitriser les langages de programmation pour qui seront certes utiles au niveau professionnel.
Enfin, nous espérons que nous avons réussi à réaliser le travail demandé.