Rapport d’implémentation

Génie Logiciel

Projet de modélisation et d’implémentation

Simulateur pour un distributeur de billets de train





Table des matières

Introduction4

Diagramme de cas d’utilisation4

Description du diagramme4

Diagramme4

Description semi-formelle des cas d’utilisation5

Diagramme de classes10

Description du diagramme10

Diagramme11

Diagrammes de séquences12

1. Acheter billet12

Description du diagramme12

Diagramme12

2. Acheter abonnement12

Description du diagramme12

Diagramme12

3. Renouveler abonnement13

Description du diagramme13

Diagramme13

4. Acheter pass13

Description du diagramme13

Diagramme14

5. Paiement14

Description du diagramme14

Diagramme15

6. Impression15

Description du diagramme15

Diagramme16

7. Sortie de veille16

Description du diagramme16

Diagramme16

8. Vérifier horaire trains16

Description du diagramme16

Diagramme17

9. Créer/gérer une panne17

Description du diagramme17

Diagramme17

10. Recharger/vider nombre d’impressions18

Description du diagramme18

Diagramme18

11. Activer/désactiver composant optionnel19

Description du diagramme19

Diagramme19

12. Recharger/vider caisse19

Description du diagramme19

Diagramme20

Diagramme global d’interaction20

Description du diagramme20

Diagramme20

Diagramme d’état21

Description du diagramme21

Diagramme21

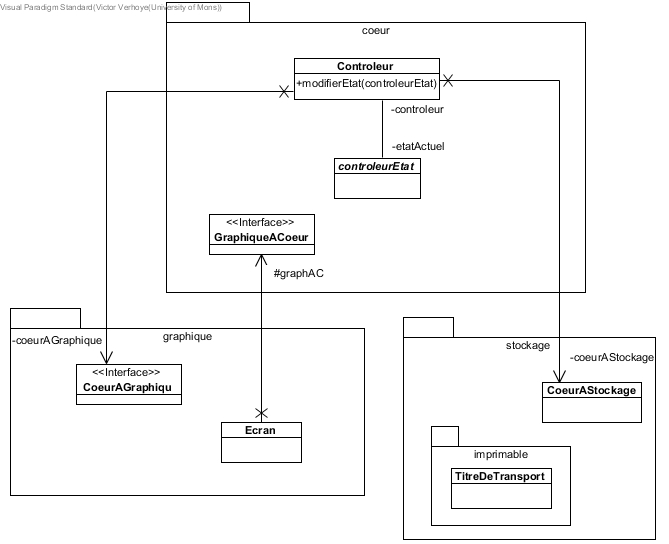
# 1. Introduction

Dans ce qui suit, nous allons parler des modifications apportées à notre application par rapport à ce qui avait été pensé lors de la modélisation, des design patterns utilisés et des problèmes connus. ATTENTION : si il y en a !!

# 2. Modifications apportées

Comme nous avons pu avoir l’occasion d’en parler avec Mr Hauweele, deux problèmes principaux ont mené à une quantité non-négligeable de modifications par rapport à notre modélisation : l’absence de design patterns et d’encapsulation entre la partie « cœur », la partie « stockage » et la partie « graphique » de notre application. Nous n’avions pas développé toute cette partie dans la modélisation de notre application car à l’époque où nous avons travaillé sur la modélisation, nous ne savions pas encore ce qu’étaient les design pattern. Concernant l’encapsulation, étant donné que c’était le premier projet d’informatique de notre parcours scolaire, nous manquions d’expérience et n’avions donc ni conscience de l’importance d’une bonne encapsulation, ni la bonne manière de s’y prendre.

Nous avons donc dû restructurer nos packages, et de ce fait la majorité des messages sont modifiés et ne correspondent plus à ceux indiqués dans les diagrammes de séquence. Nous avons tout de même essayé, tant bien que mal, d’être cohérent avec les méthodes qui existaient de base. Pour mieux comprendre la restructuration des packages, voici un condensé du nouveau diagramme de classe (celui-ci est bien sûr loin d’être complet, que ça soit au niveau des associations ou du nombre de classes) :



Voici une liste de modifications :

* La méthode calculerPrix prenait un titre de transport comme paramètre auparavant. Elle prend maintenant tous les paramètres d’un futur titre car, étant donné que le titre de transport est créé après que le distributeur a connaissance de tous ses paramètres (afin de respecter l’encapsulation), on ne pourrait le passer en paramètre vu qu’il n’existe pas encore. Vu que le prix des différents titres est stocké dans des tables différentes en fonction du titre et que ces prix n’ont aucun lien, nous avons aussi du séparer calculerPrix en calculerPrixAbo, calculerPrixBillet et calculerPrixPass.
* Nous avons ajouté une classe monnayeur qui va se charger de tout ce qui est rendu d’argent. Nous avons fait cela car, dû à notre problème d’encapsulation, une classe propre au graphique (FenetreSimulation) stockait des informations (nombre de pièces restantes,…), ce qui est le boulot du package stockage.
* Concernant les bases de données, nous avons décidé de séparer le travail en la gestion concernant les horaires de train (HoraireTrains), les titres de transport (BDDTitre) et la banque (BDDBanque). Vu qu’une partie de leur comportement est commun, la classe GestionBaseDeDonnées est devenue abstraite et contient tout ce qu’il leur est commun. De ce fait, toutes les méthodes qu’on pouvait trouver auparavant dans celle-ci sont réparties dans ses sous-classes.
* Pour les cartes bancaires, nous ne faisons plus une recherche de toutes les cartes que nous affichons, mais nous laissons plutôt à l’utilisateur la possibilité de taper le numéro de carte qu’il désire, et nous vérifions dans la base de données si ce numéro existe bien, si le code PIN donné est correct, et si il a assez d’argent pour payer son titre de transport (méthode verifPaiement).
* Afin de respecter l’idée d’un titre de transport unique, le nombre de titres demandé n’est plus un paramètre propre au titre lui-même.
* A COMPLETER AVEC VOS IDEES DANS CE QUE VOUS AVEZ FAIT ! POUR CELA , IL FAUT QUE VOUS REGARDIEZ PAR RAPPORT A LA MODELISATION (SUFFIT DE REGARDER LE DIAGRAMME DE CLASSES DANS LE RAPPORT PDF) PERSO G CHANGE DEUX TROIS NOMS DE CLASSES DANS MON IMPLEMENTATION CAR ELLES AVAIENT D AUTRES NOMS DANS LA MODELISATION

# 3. Justifications concernant les design patterns utilisés

## 3.1. State Design Pattern

Afin de représenter le comportement du distributeur, nous avons choisi d’utiliser des états. Le contrôleur (classe Controleur) du distributeur aura ainsi son état (attribut etatActuel) qui changera en fonction des actions de l’utilisateur. En fonction de l’état dans lequel le contrôleur se trouve, son comportement sera différent. Les changements d’état nous permettent aussi de demander à l’interface graphique (package interfaceGraphique) d’afficher ce qu’il faut quand il faut.

## 3.2 Singleton Design Pattern

Nous avons adopté ce Design Pattern pour plusieurs classes différentes :

* Controleur
* Tous les états
* A COMPLETER

Et la justification est plus ou moins la même pour chacune d’entre-elles : ?????????????????????????????????????????????????????? je sais pas trop quoi dire

NOTES A SUPPRIMER :

Choses restantes :

* Ecran pas encore non tactile
* Tests unitaires pour vérifier application (correspondances avec énoncé) et pas de bogue
* Si pas de code scanneur, pas d’achat d’abo ??? bizarre (renouvellement ok par contre) 🡪 ATTENTION, a discuter
* Panne doit comprendre (au moins) : lecteur carte, carte coincée, scanneur code, coin slot ou bill slot 🡪 pas de paiement en monnaie, Imprimante 🡪 tout le distrib hors services
* Paiement valide en fonction des composants actifs ! si pas assez d’argent, il faut bien rendre l’argent et proposer un mode de paiement !
* Junit 4.6 ou sup obligé
* Pouvoir executer application en utilisant maven par la ligne de commande !!
* Fonctionnel sur les trois OS
* ML-<noms de famille>-implementation
* ATTENTION : nos noms doivent figurer dans la javadoc de chaque fichier de l’implementation ?

Implémentation contient :

* Code source, tests unitaires, interface graphique 🡪 compilable avec maven sur tout OS
* Fichier .jar auto-executable
* Un doc pdf avec rapport, modifs (justifiées), design patterns, problemes connus
* Autre pdf : manuel
* Attention : ne pas mettre d’accents, ne passe pas sur certains OS