

ESIEE PARIS  
RAPPORT DE PROJET  
INF-5201C

## **Projet**

Gestion d'un stock avec UML et Java  
**Analyse-Conception-Implémentation**

Groupe 3  
NGUYEN Hong Ngoc & PATOIS Thibault & ROLLOT Quentin

# Table des matières

<b>I</b>	<b>Analyse</b>	<b>3</b>
<b>1</b>	<b>Capture des exigences fonctionnels</b>	<b>4</b>
1.1	Diagramme des cas d'utilisation . . . . .	4
1.2	Description des cas importants . . . . .	5
1.2.1	Cas d'utilisation : Génération d'une commande . . . . .	5
1.2.2	Cas d'utilisation : Mettre à jour le stock . . . . .	6
1.2.3	Cas d'utilisation : Saisir un nouveau fournisseur . . . . .	6
1.2.4	Cas d'utilisation : Scan d'un code barre . . . . .	6
1.2.5	Cas d'utilisation : Remboursement d'un article . . . . .	6
1.2.6	Cas d'utilisation : Solde de la caisse . . . . .	7
1.3	Prototypes des interfaces . . . . .	8
1.3.1	Vue de l'accueil backend . . . . .	8
1.3.2	Vue de l'inventaire . . . . .	8
1.3.3	Vue d'un article . . . . .	9
1.3.4	Vue d'ajout et d'édition d'un article . . . . .	9
1.3.5	Vue d'ajout et d'édition d'un fournisseur . . . . .	10
1.3.6	Vue d'une commande . . . . .	10
1.3.7	Vue pour passer une commande . . . . .	11
1.3.8	Vue de la caisse . . . . .	11
<b>2</b>	<b>Etude du modèle statique</b>	<b>12</b>
2.1	Diagramme de classe . . . . .	12
2.2	Dictionnaire de données . . . . .	12
2.2.1	Articles . . . . .	12
2.2.2	Fournisseur . . . . .	13
2.2.3	Commande . . . . .	13
2.2.4	InterfaceUtilisateur . . . . .	13
2.2.5	Caisse . . . . .	13
2.2.6	Client . . . . .	14
2.2.7	Caisse . . . . .	14
<b>3</b>	<b>Etude du modèle dynamique</b>	<b>15</b>
3.1	Diagramme de séquence et diagramme de classe des principaux cas d'utilisation	15
3.1.1	Diagramme de séquence et de classe pour le cas d'utilisation : Modifier les informations d'un article . . . . .	15
3.1.2	Diagramme de séquence et de classe pour le cas d'utilisation : Lister les articles à commander . . . . .	16
3.2	Diagramme d'états de certaines classes pertinentes . . . . .	17
3.2.1	Diagramme d'état d'un article . . . . .	17

3.2.2	Diagramme d'état de la caisse . . . . .	17
<b>4</b>	<b>Synthèse</b>	<b>18</b>
<b>II</b>	<b>Conception</b>	<b>19</b>
<b>5</b>	<b>Conception générale</b>	<b>20</b>
5.1	Répartition en paquetage . . . . .	20
5.2	Choix des technologies . . . . .	21
5.3	Diagramme de composant du logiciel . . . . .	21
5.4	Diagramme de déploiement . . . . .	21
<b>6</b>	<b>Conception détaillée</b>	<b>22</b>
6.1	Diagrammes de classes techniques des paquetages . . . . .	22
6.2	Diagrammes de séquences techniques . . . . .	25
6.3	Architecture du projet : approche MVC . . . . .	25
<b>III</b>	<b>Implémentation</b>	<b>26</b>
<b>7</b>	<b>Implémentation</b>	<b>27</b>
7.1	Copies d'écran de l'implémentation . . . . .	27
7.2	Livrable du logiciel . . . . .	30
7.2.1	Installation . . . . .	30
7.2.2	Mode d'emploi . . . . .	30
7.3	Bilan de l'état du projet . . . . .	30

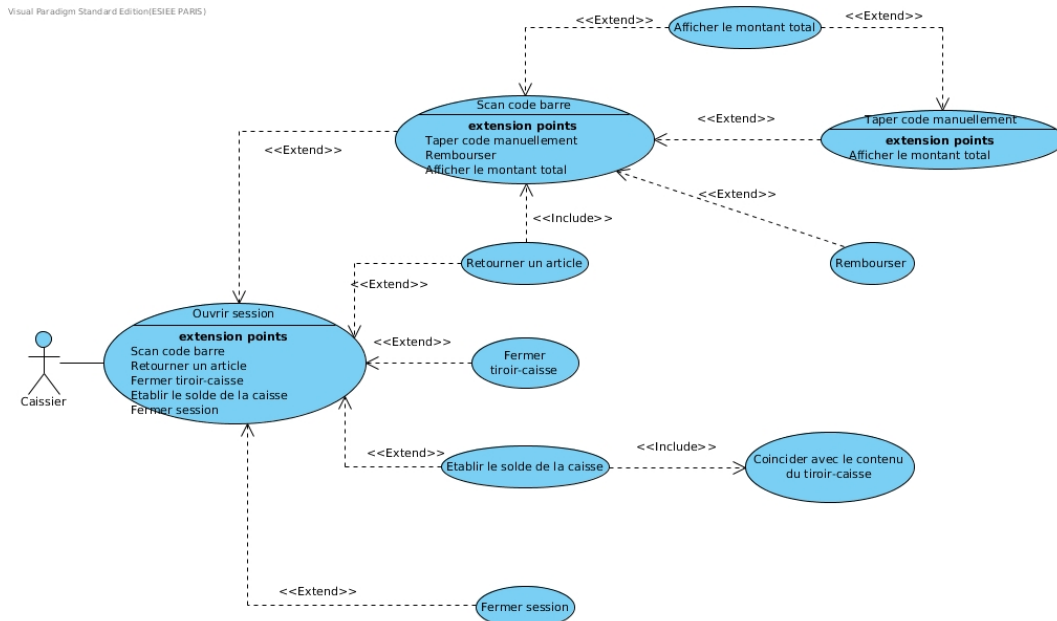
Première partie

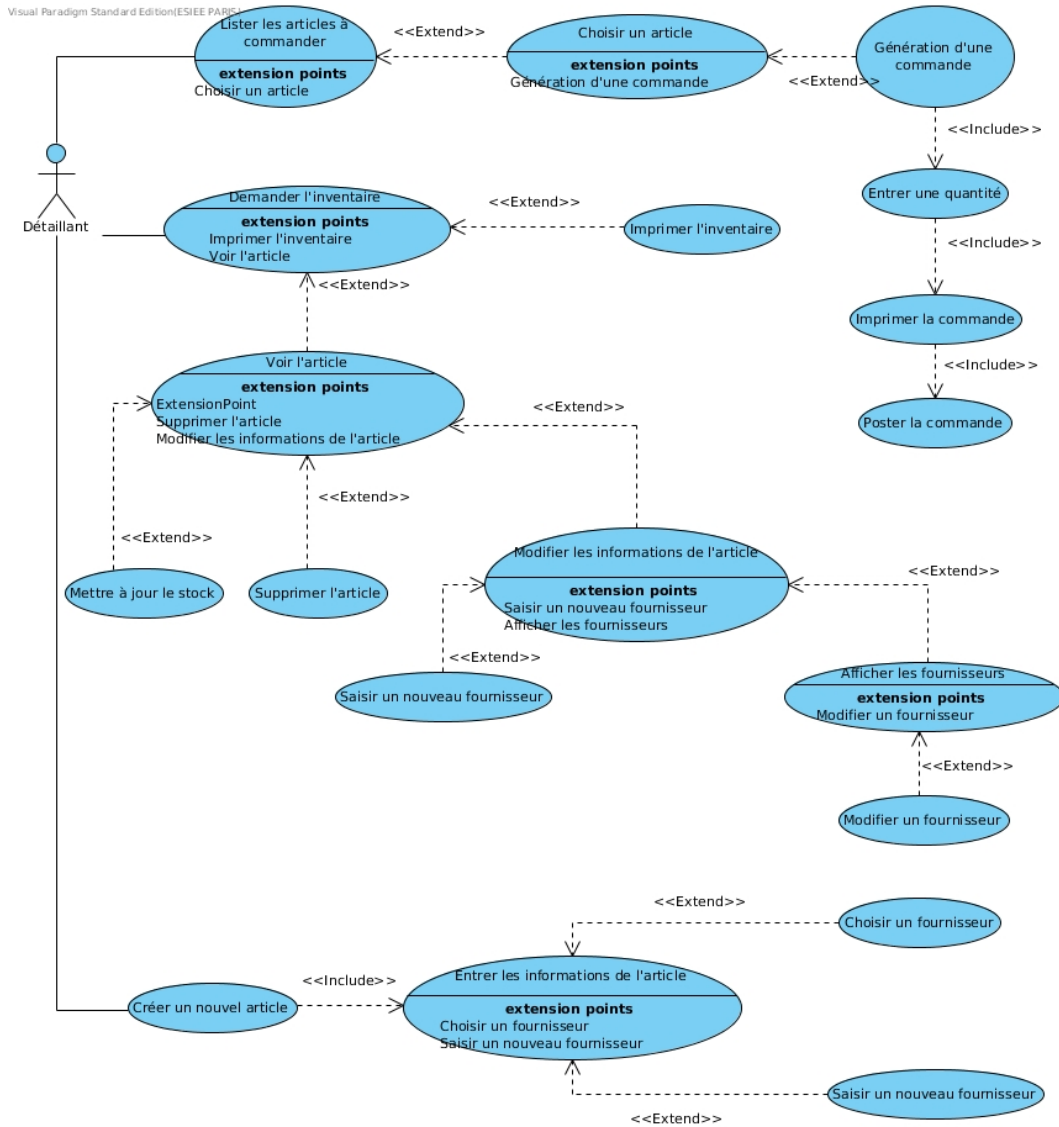
Analyse

# Chapitre 1

## Capture des exigences fonctionnels

### 1.1 Diagramme des cas d'utilisation





## 1.2 Description des cas importants

### 1.2.1 Cas d'utilisation : Génération d'une commande

- Objectif : Commander des articles dont la quantité est passée sous un seuil définit
- Acteurs concernés : Le détaillant
- Pré conditions : La quantité de l'article est passée sous un seuil et le détaillant à choisi l'article en question dans la liste des articles dont le niveau en stock est inférieur au seuil définit.
- Scénario nominal :
  1. Le détaillant affiche la liste des articles dont le niveau de stock est inférieur au seuil définit
  2. Le détaillant sélectionne l'article qui veut commander
  3. Le détaillant valide la commande

### **1.2.2 Cas d'utilisation : Mettre à jour le stock**

- Objectif : Mettre à jour le stock d'un article
- Acteurs concernés : Le détaillant
- Pré conditions : Le détaillant a sélectionné l'article concerné dans l'inventaire et a sélectionné l'option d'édition de l'article
- Scénario nominal :
  1. Le détaillant demande l'inventaire des stocks
  2. Le détaillant sélectionne l'article dont le niveau de stock doit être mis à jour
  3. Le détaillant sélectionne l'option d'édition
  4. Le détaillant modifie le niveau de stock

### **1.2.3 Cas d'utilisation : Saisir un nouveau fournisseur**

- Objectif : Ajouter un fournisseur à un article
- Acteurs concernés : Le détaillant
- Pré conditions : Le détaillant a sélectionné l'article concerné dans l'inventaire et a sélectionné l'option d'édition de l'article
- Scénario nominal :
  1. Le détaillant demande l'inventaire des stocks
  2. Le détaillant sélectionne l'article
  3. Le détaillant sélectionne l'option d'édition
  4. Le détaillant rajoute un fournisseur avec ses informations

### **1.2.4 Cas d'utilisation : Scan d'un code barre**

- Objectif : Scanner un article
- Acteurs concernés : Le caissier
- Pré conditions : Le caissier a démarré une session sur la caisse et un client s'est présenté devant la caisse avec un article.
- Scénario nominal :
  1. Le caissier ouvre une session sur la caisse
  2. Un client approche avec un article
  3. Le caissier scan le code barre de l'article
- Scénario alternatif :
  1. Le caissier ouvre une session sur la caisse
  2. Un client approche avec un article
  3. Le code barre est illisible et le caissier entre la référence de l'article puis appuie sur <Enter>

### **1.2.5 Cas d'utilisation : Remboursement d'un article**

- Objectif : Remboursement d'un article
- Acteurs concernés : Le caissier
- Pré conditions : Le caissier a démarré une session sur la caisse et un client s'est présenté devant la caisse avec un article à retourner. L'article doit être vendable.

- Scénario nominal :
  1. Le caissier ouvre une session sur la caisse
  2. Un client approche avec un article à retourner
  3. L'article en question est vendable
  4. Le caissier scan le code barre de l'article
  5. Le tiroir de la caisse s'ouvre
  6. Le caissier rembourse le client
- Scénario alternatif 1
  1. Le caissier ouvre une session sur la caisse
  2. Un client approche avec un article à retourner
  3. L'article n'est pas vendable et le client ne se fait pas rembourser
- Scénario alternatif 2
  1. Le caissier ouvre une session sur la caisse
  2. Un client approche avec un article à retourner
  3. L'article en question est vendable
  4. L'article est vendable et le code barre est illisible.
  5. Le caissier entre alors la référence de l'article puis appuie sur <Enter>.
  6. Le tiroir de la caisse s'ouvre
  7. Le caissier rembourse le client

### **1.2.6 Cas d'utilisation : Solde de la caisse**

- Objectif : Effectuer le solde de la caisse
- Acteurs concernés : Le caissier
- Pré conditions : Le caissier a démarré une session sur la caisse et appuie deux fois sur <Total>
- Scénario nominal :
  1. Le caissier ouvre une session sur la caisse
  2. Le caissier appuie deux fois sur la touche <Total>
  3. Le tiroir de la caisse s'ouvre et la caisse lance l'impression de la liste des tickets de caisse.
  4. Le caissier fait coïncider le contenu du tiroir de caisse avec le montant total de l'ensemble des opérations réalisées.

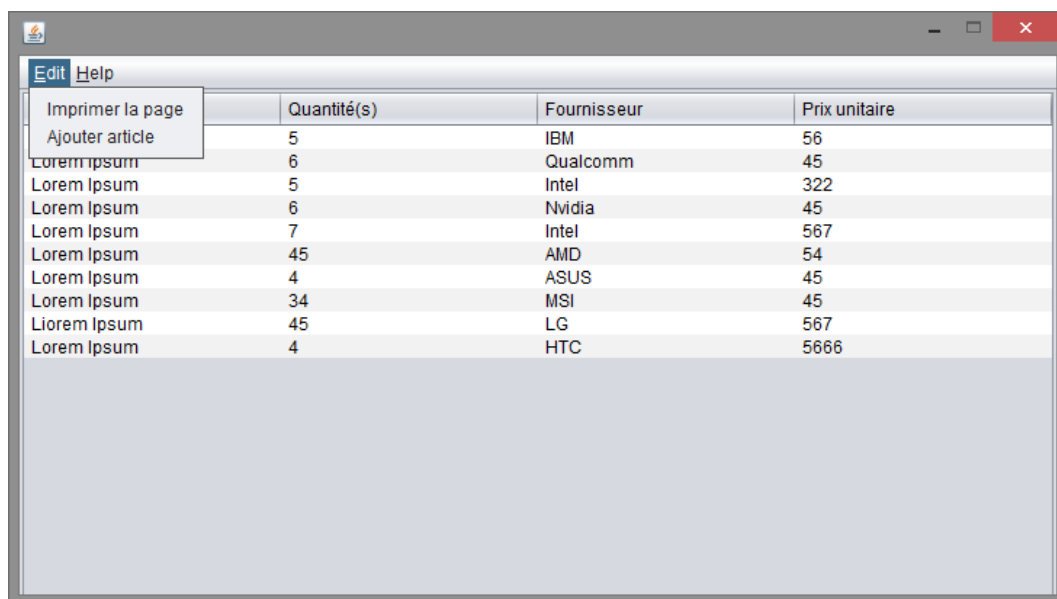


## 1.3 Prototypes des interfaces

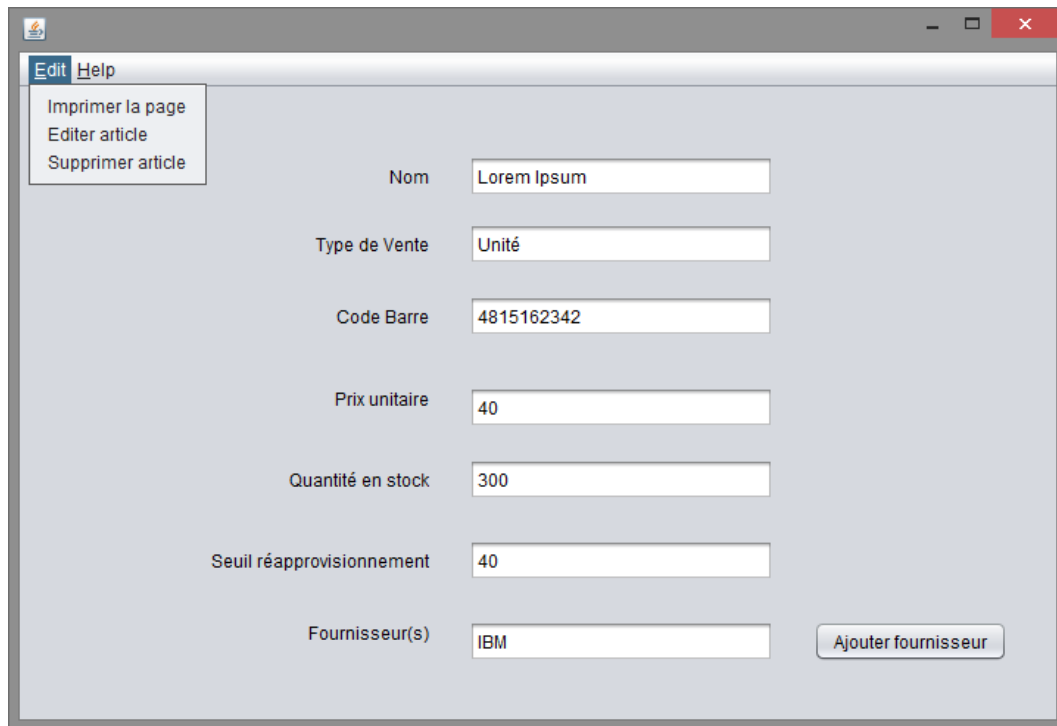
### 1.3.1 Vue de l'accueil backend



### 1.3.2 Vue de l'inventaire



### 1.3.3 Vue d'un article



Menu: Edit Help

- Imprimer la page
- Editer article
- Supprimer article

Nom: Lorem Ipsum

Type de Vente: Unité

Code Barre: 4815162342

Prix unitaire: 40

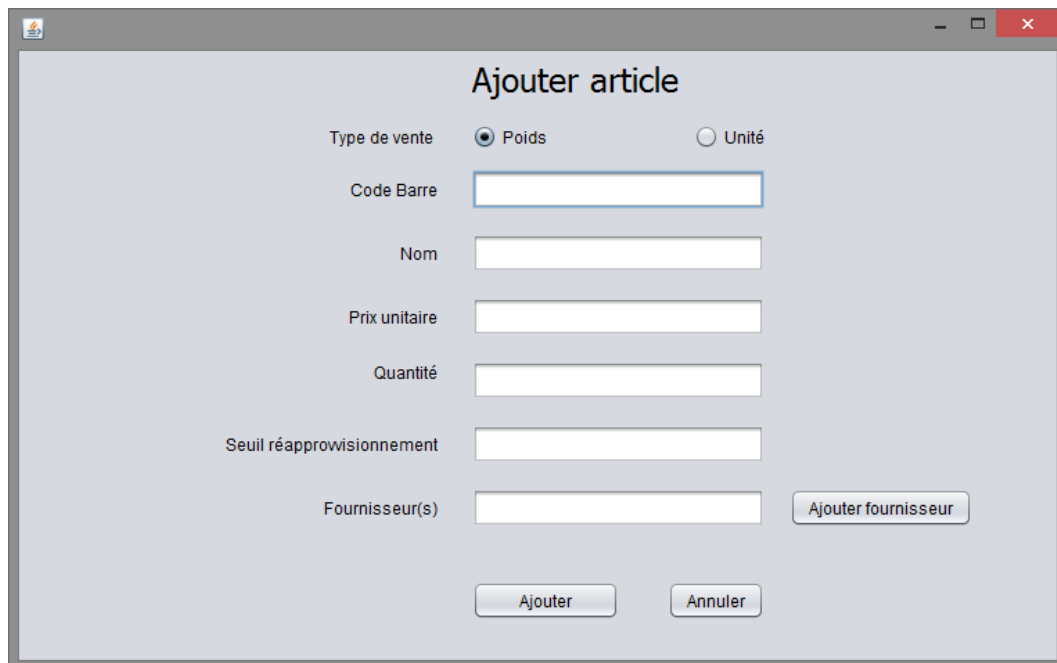
Quantité en stock: 300

Seuil réapprovisionnement: 40

Fournisseur(s): IBM

Ajouter fournisseur

### 1.3.4 Vue d'ajout et d'édition d'un article



Ajouter article

Type de vente: ☒ Poids ☐ Unité

Code Barre:

Nom:

Prix unitaire:

Quantité:

Seuil réapprovisionnement:

Fournisseur(s):

Ajouter fournisseur

Ajouter Annuler

### 1.3.5 Vue d'ajout et d'édition d'un fournisseur



Ajouter fournisseur

Nom

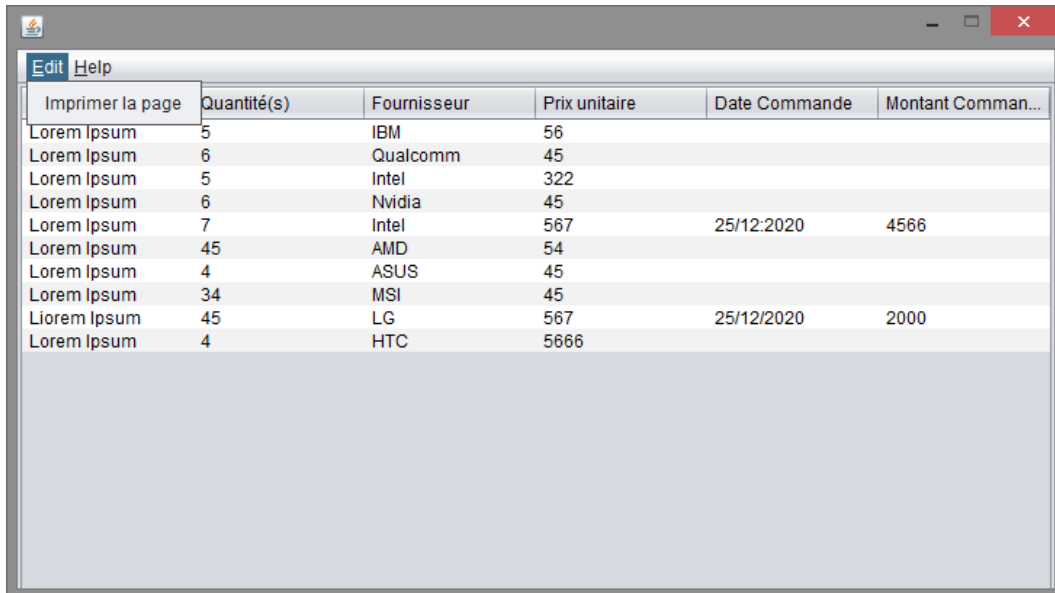
Adresse

Code Postal

Numéro de téléphone

Ajouter Annuler

### 1.3.6 Vue d'une commande



	Quantité(s)	Fournisseur	Prix unitaire	Date Commande	Montant Comman...
Imprimer la page	5	IBM	56		
Lorem Ipsum	6	Qualcomm	45		
Lorem Ipsum	5	Intel	322		
Lorem Ipsum	6	Nvidia	45		
Lorem Ipsum	7	Intel	567	25/12/2020	4566
Lorem Ipsum	45	AMD	54		
Lorem Ipsum	4	ASUS	45		
Lorem Ipsum	34	MSI	45		
Liorem Ipsum	45	LG	567	25/12/2020	2000
Lorem Ipsum	4	HTC	5666		

### 1.3.7 Vue pour passer une commande

	quantité(s)	Fournisseur(s)	Prix Unitaire	Montant Commande
Lorem Ipsum	5	IBME	5€	250€

### 1.3.8 Vue de la caisse

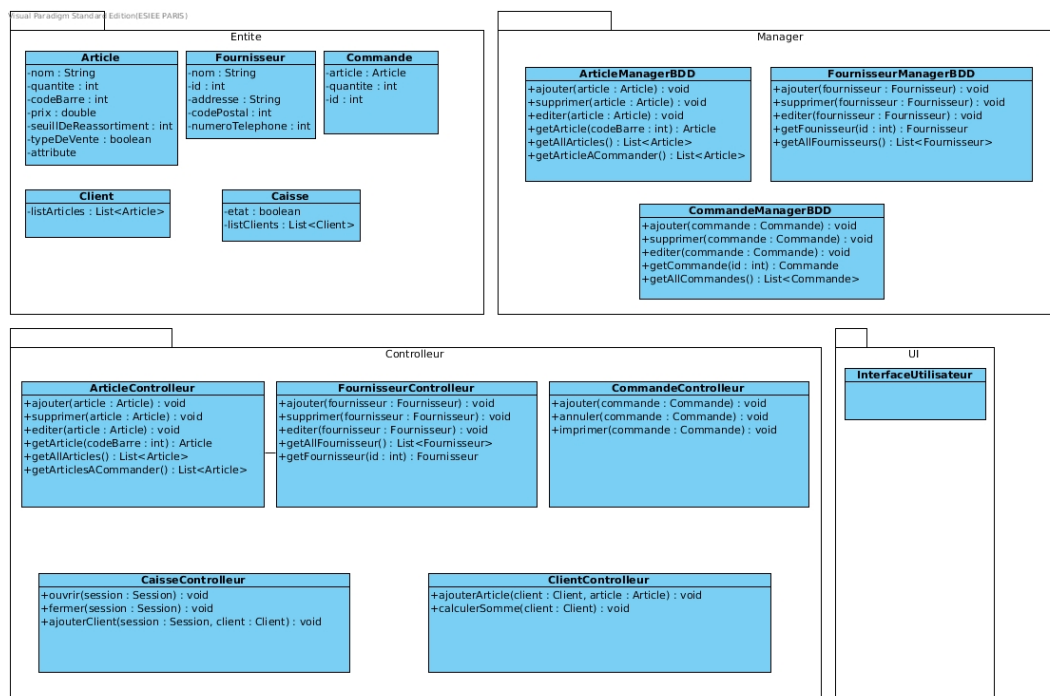
Article	Quantité	Prix
Lorem	1	34
Lorem	1	45
Lorem	2	56
Lorem	1	67
Lorem	1	78
Lorem	1	12
Lorem	3	34

TOTAL

# Chapitre 2

## Etude du modèle statique

### 2.1 Diagramme de classe



### 2.2 Dictionnaire de données

#### 2.2.1 Articles

##### Article

La classe article décrit les caractéristiques d'un article : son identifiant, nom, prix et quantité en stock. Cette classe sera instanciée autant de fois qu'il y aura d'article différent.

##### ArticleContrôleur

La classe ArticleController effectue toutes les actions sur les objets articles : retourner une liste d'objets, traiter des objets etc ... Elle ne sera instanciée qu'une seule fois.

## **ArticleManager**

ArticleManager fait le lien entre la base de donnée et les entités Articles. Elle sera également instanciée qu'une seule fois. Ses fonctions essentiels sont la sauvegarde, l'ajout et l'édition d'un article dans la base de donnée.

### **2.2.2 Fournisseur**

#### **Fournisseur**

La classe Fournisseur décrit les caractéristiques d'un article : son nom et son identifiant. La classe fournisseur sera instanciée autant de fois qu'il y aura de fournisseur différent.

#### **FournisseurController**

La classe FournisseurController effectue toutes les actions sur les objects : retourner une liste, suppression, édition etc ... Elle ne sera instanciée qu'une seule fois.

#### **FournisseurManager**

FournisseurManager fait le lien entre la base de donnée et les entités Fournisseur. Elle sera également instanciée qu'une seule fois. Ses fonctions essentiels sont la sauvegarde, l'ajout et l'édition d'un fournisseur dans la base de donnée.

### **2.2.3 Commande**

#### **Commande**

La classe Commande décrit les caractéristiques d'une Commande : sa référence, les articles commandés et leur quantité. La classe Commande sera instanciée autant de fois qu'il y aura de commande différentes.

#### **CommandeController**

La classe CommandeController effectue toutes les actions sur les commandes : retourner la liste des commandes, annulation ... Elle ne sera instanciée qu'une seule fois.

#### **CommandeManager**

CommandeManager fait le lien entre la base de donnée et les entités Commande. Elle sera également instanciée qu'une seule fois. Ses fonctions essentiels sont la sauvegarde, l'ajout et l'édition d'une Commande dans la base de donnée.

### **2.2.4 InterfaceUtilisateur**

L'interfaceUtilisateur sera le controller principal des vues. Il instanciera et affichera toutes les vues décrivent dans la partie précédente.

### **2.2.5 Caisse**

L'objet caisse représente la caisse. Il contiendra en attribut les différents états de la caisse. La caisse sera instancié qu'une seule fois.

### **2.2.6 Client**

#### **Client**

L'object Client représente un client passant en caisse. Cette classe se chargera de retenir tous les articles achetés par un client. Elle sera donc instanciée autant de fois que de client passeront à la caisse.

#### **ClientController**

Comme toujours, le ClientController effectuera les actions sur les objects Clients. Cette classe sera instanciée qu'une seule fois.

### **2.2.7 Caisse**

#### **Caisse**

La classe objet Caisse représente une session de la caisse. Elle sera instanciée autant de fois qu'un caissier démarrera une nouvelle session. Elle contiendra tous les états de la caisse et la liste des clients qui se seront présenté devant la caisse.

#### **CaisseController**

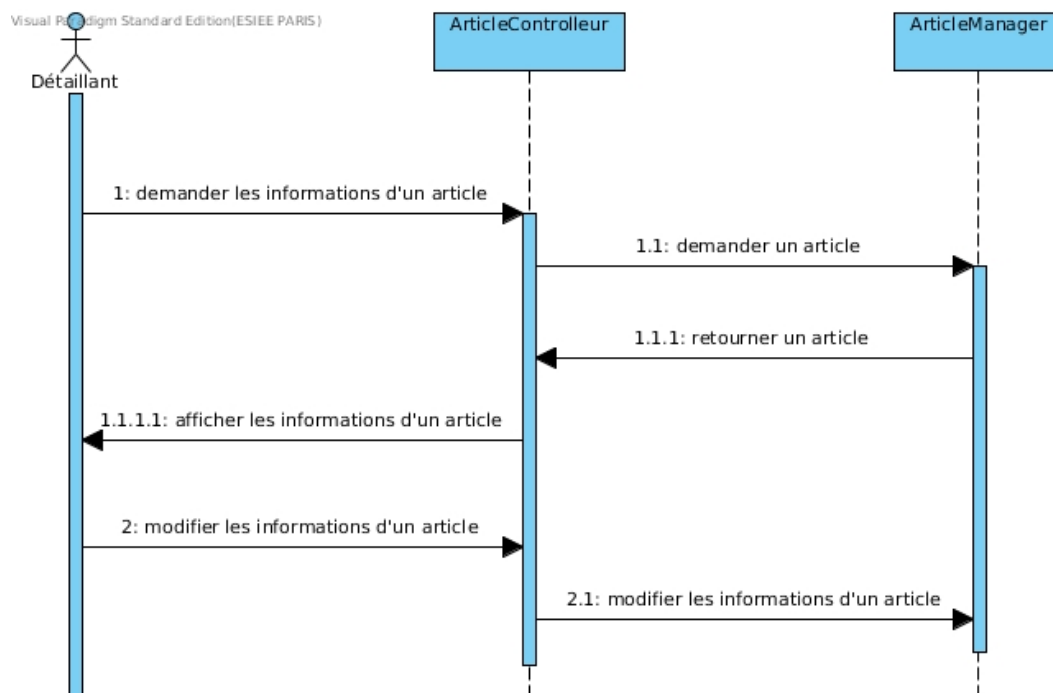
La classe CaisseController effectue toutes les actions sur les caisses : ouverture du tiroir, impression des tickets de caisse etc ... Elle ne sera instanciée qu'une seule fois.

## Chapitre 3

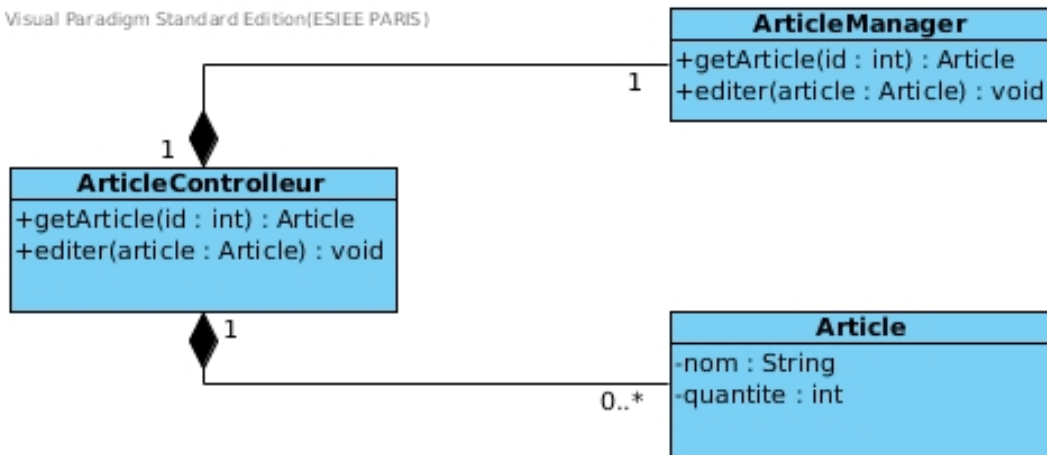
# Etude du modèle dynamique

### 3.1 Diagramme de séquence et diagramme de classe des principaux cas d'utilisation

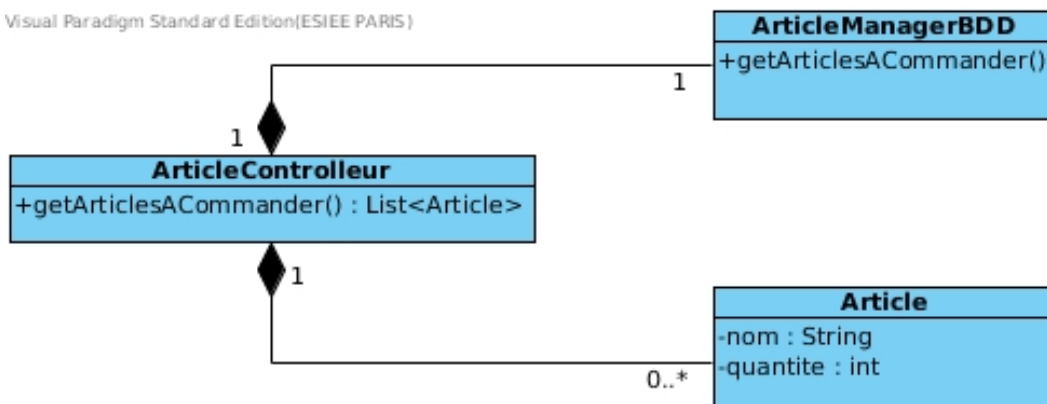
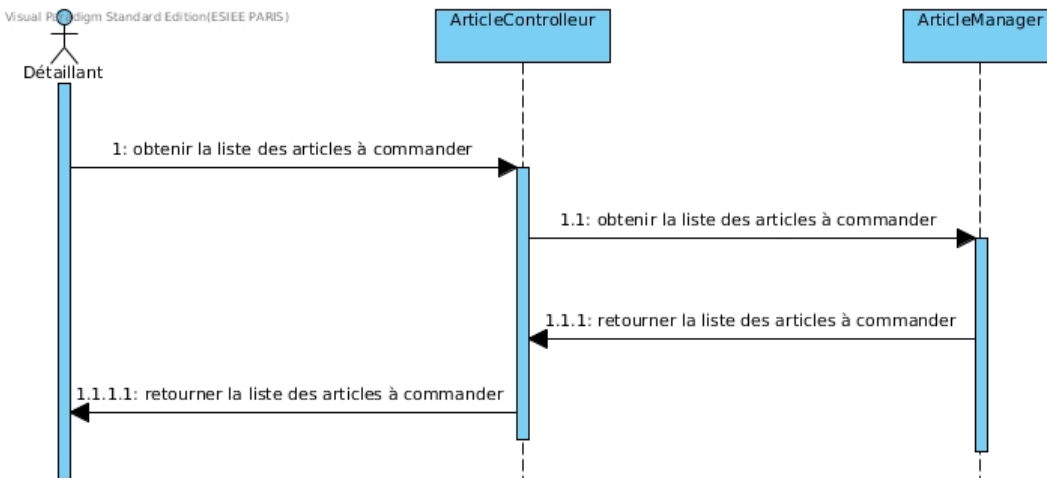
#### 3.1.1 Diagramme de séquence et de classe pour le cas d'utilisation : Modifier les informations d'un article





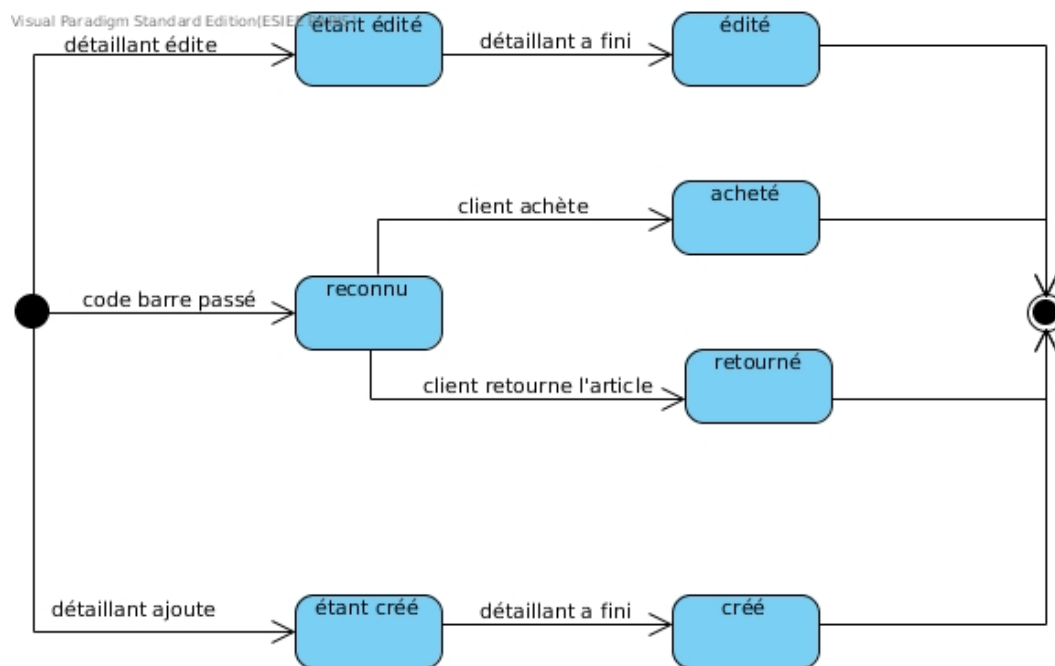


### 3.1.2 Diagramme de séquence et de classe pour le cas d'utilisation : Lister les articles à commander

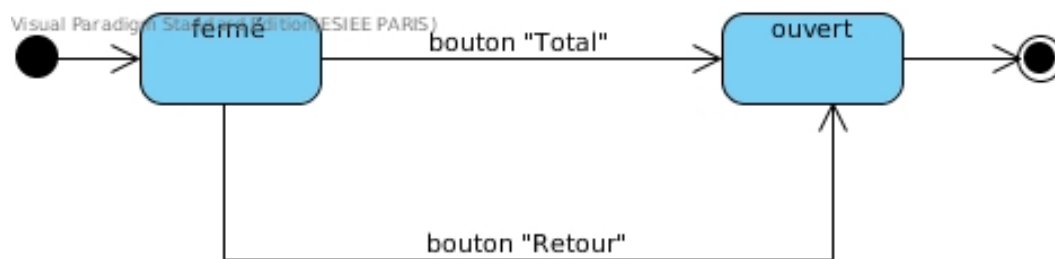


## 3.2 Diagramme d'états de certaines classes pertinentes

### 3.2.1 Diagramme d'état d'un article

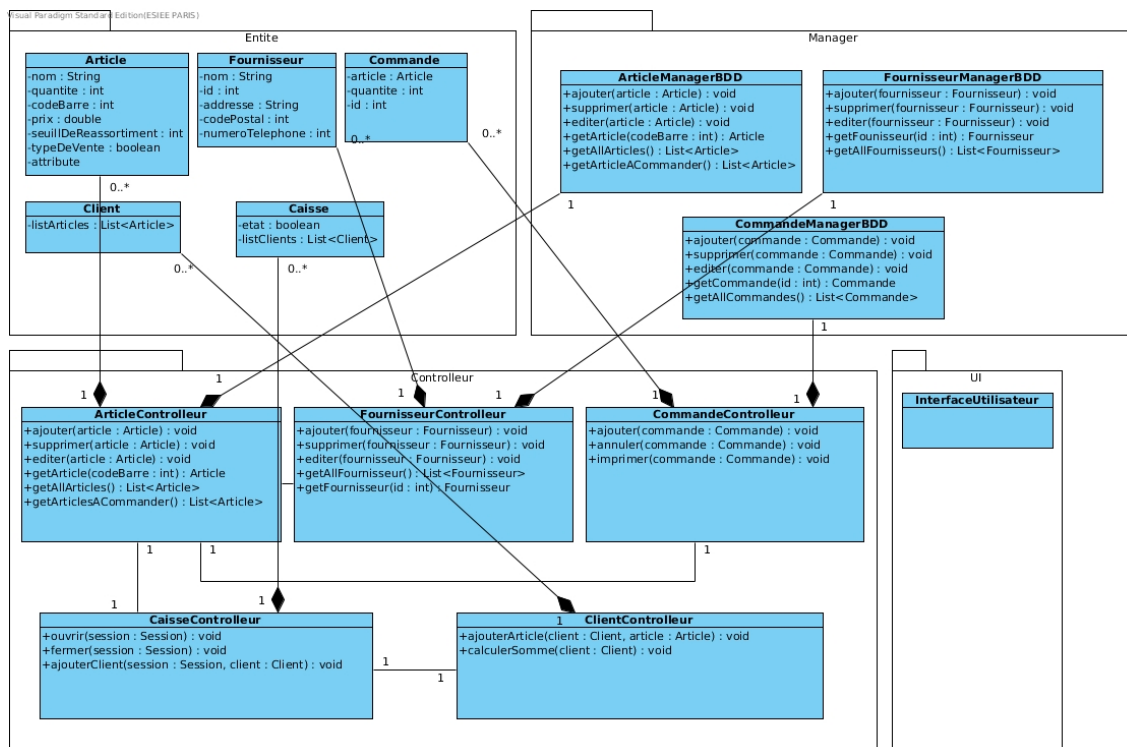


### 3.2.2 Diagramme d'état de la caisse



# Chapitre 4

## Synthèse



Deuxième partie

Conception

## Chapitre 5

# Conception générale

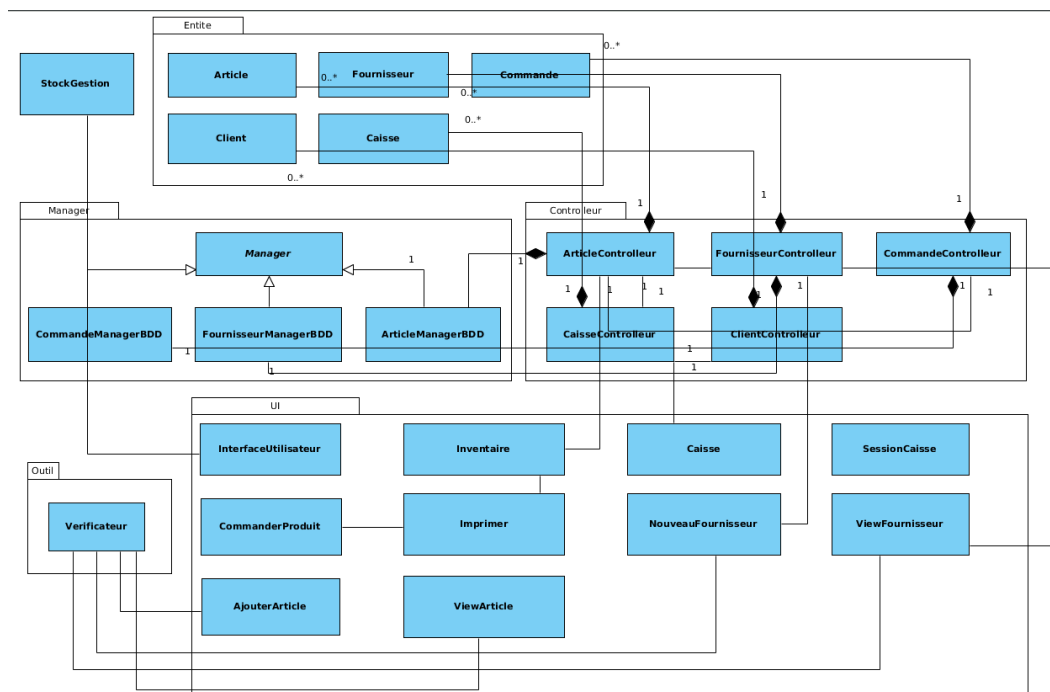
### 5.1 Répartition en paquetage

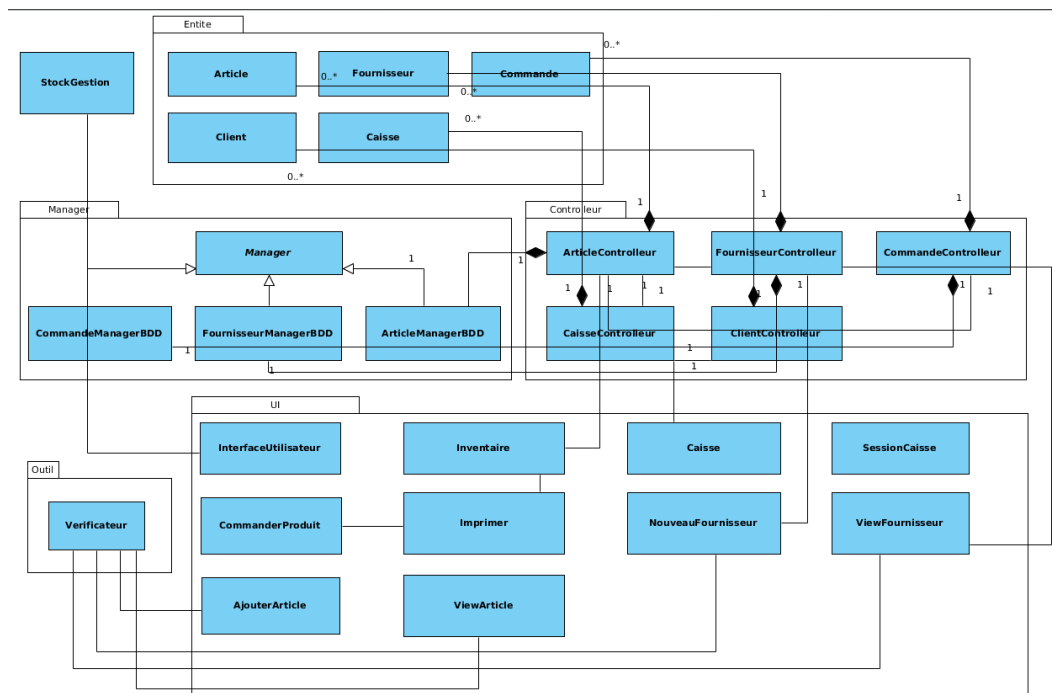
Dans notre projet, nous avons un paquetage par défaut *stockgestion*. Dans ce paquetage, nous retrouvons 5 sous-paquetages *Entite*, *Manager*, *Controlleur*, *Outil* et *UI*.

Chaque package a des fonctionnalités bien délimité.

- UI regroupe toutes les classes de l'interface utilisateur.
- Manager s'occupe de toutes les interactions avec la base de donnée.
- Controlleur gèrent les interactions entre toutes les classes et les actions de l'utilisateur.
- Outil contient toutes les classes Outils
- Entite contient toutes les représentations objets de notre application.

Ci-dessous, le diagramme de Package qui représente les interactions entre nos packages :





## 5.2 Choix des technologies

Le projet consiste à implémenter une solution de suivi de stock et de gestion d'une caisse. Elle ne requiert aucun traitement lourd, nous n'avons donc pas implémenter de traitement multithread.

Notre projet est bâti autour d'un MVC comprenant un package de controleurs, un package de vue et un package de manager. Les managers traitent avec une base de donnée SQL nommé Derby. Nous avons fait ce choix car ils nous semblait important de stocker les données dans une base de donnée type SQL. De plus, nous avons déjà travailler avec Derby dans des projets précédents.

L'interface utilisateur est animé par swing. Ce fût un choix pratique avant tous, notamment car Neteans propose un générateur de template puissant nommé Matisse.

Notre solution compte également de nombreux Singleton. En effet, les controleurs et les managers sont des singletons. La raison de cette implémentation a déjà été expliqué dans la aprtie précédente.

## 5.3 Diagramme de composant du logiciel

Quentin ?

## 5.4 Diagramme de déploiement

Ngocky ?

## Chapitre 6

# Conception détaillée

### 6.1 Diagrammes de classes techniques des paquetages

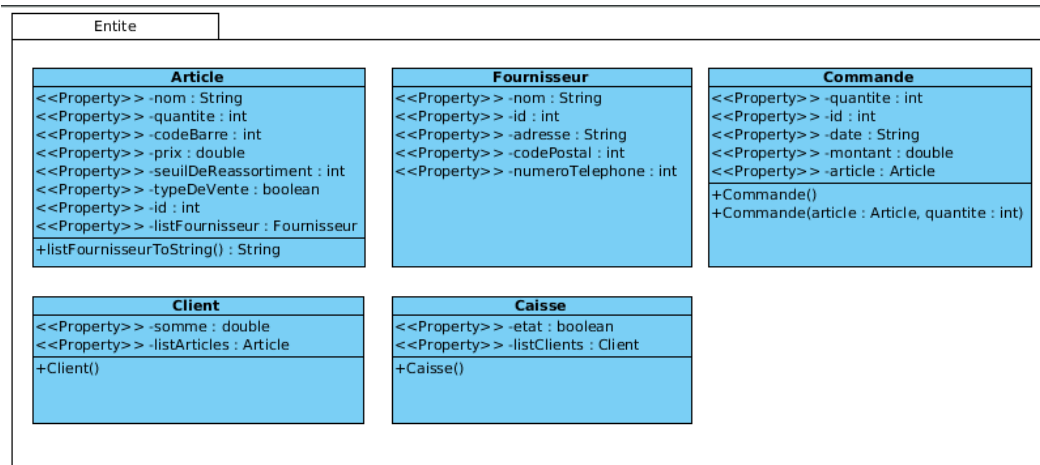
A la racine de notre projet, nous retrouvons le paquetage **stockgestion** créé par défaut. Ce paquetage ne contient qu'une seule classe **StockGestion** qui a le rôle d'initialiser et de rafraîchir toutes les interfaces d'utilisateur lors des changements dans les données.

stockgestion	🔍
--------------	---

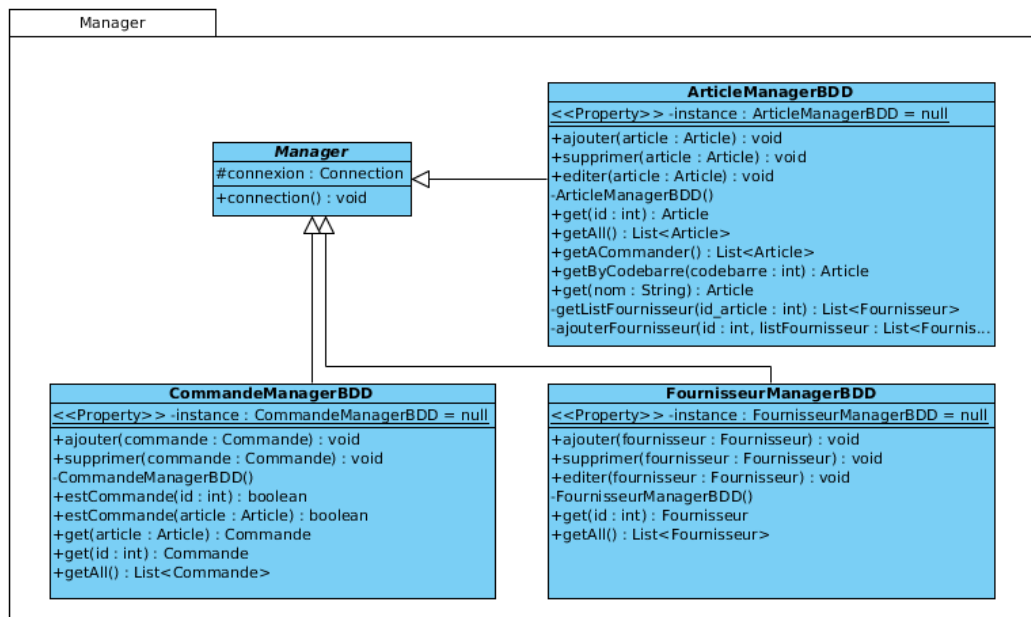
StockGestion
-articleControlleur : ArticleControlleur -caisseControlleur : CaisseControlleur -clientControlleur : ClientControlleur -commandeControlleur : CommandeControlleur -fournisseurControlleur : FournisseurControlleur -<<Property>> -instance : StockGestion = null
-ajouterArticle : AjouterArticle -caisse : Caisse -commanderProduit : CommanderProduit -inventaire : Inventaire -viewArticle : ViewArticle
-StockGestion() -refreshInventaire() : void -refreshCommanderProduits() : void -refreshListeFournisseur() : void +refreshUI() : void

Dans le paquetage **stockgestion**, nous retrouvons 5 sous-paquetages. Chacun contient des classes qui jouent un certain rôle précis dans l'application.

Premièrement, nous avons le paquetage **Entite**. Dans ce paquetage, chaque classe représente une entité utilisé dans l'application : un article, un fournisseur, une commande, un client ou encore une caisse.

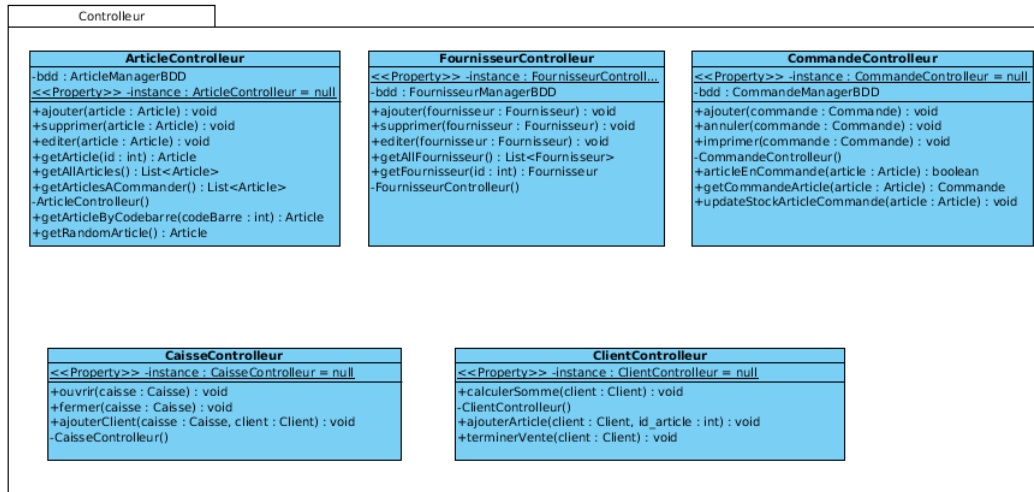


Ensuite, nous avons le paquetage **Manager**. Chaque classe dans ce paquetage s'occupe de la connexions avec la base de données ainsi que de toutes les opérations liées à cette base. Nous retrouvons ici la classe mère **Manager** qui contient la méthode de connexion à Derby. Nous avons 3 classes filles, chacune s'occupe des opération des articles, des commandes et des fournisseurs respectivement.

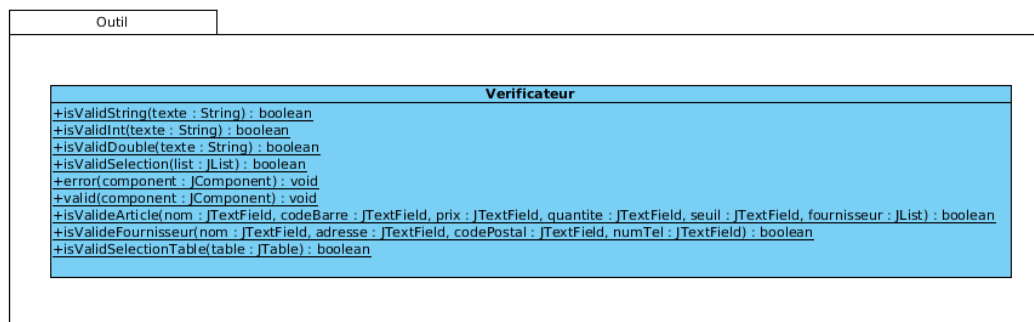


Nous avons le paquetage **Contrroleur** qui contient des classes de contrôleurs. Ces classes réalisent les opérations sur les entités en appelant les méthodes des managers.

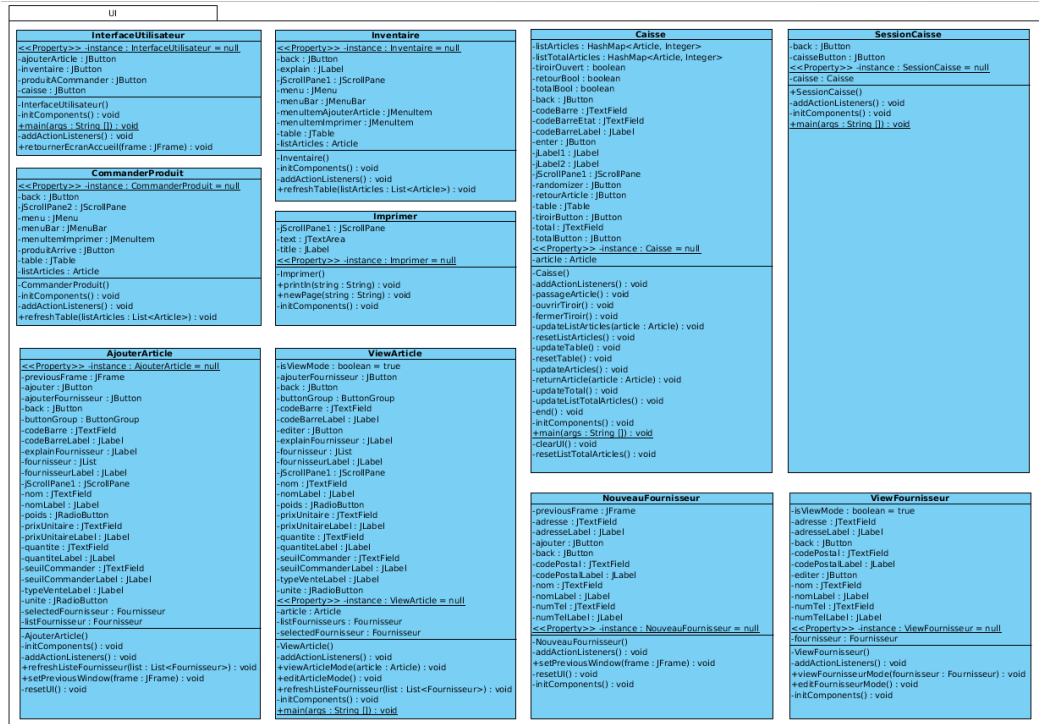




Pour les besoins très spécifiques de notre application, nous avons créé le paquetage **Outil**. On y trouve la classe **Verificateur** qui s'occupe de toutes les vérifications des entrées utilisateurs de notre application, ainsi permet d'éviter des erreurs avec la base de données (un numéro mal écrit, un texte avec des caractères spéciaux...).



Enfin, nous avons le paquetage **UI** qui contient toutes les classes de notre interface utilisateur. Chaque classe représente une fenêtre différente, chaque fenêtre correspond à une fonctionnalité de notre application.



## 6.2 Diagrammes de séquences techniques

Quentin ?

## 6.3 Architecture du projet : approche MVC

Notre application repose sur une architecture MVC. Les vues, les controllers et les managers sont ainsi séparés et placés dans des packages différents. Comme illustré plusieurs fois, le package UI s'occupe de l'aspect vue du modèle MVC. Le package Manager correspondant au manager et s'occupe de toutes les transactions avec la base de donnée.

Enfin le package contrôleur contient tous les contrôleurs et gèrent toutes les actions de l'utilisateur.

# Troisième partie

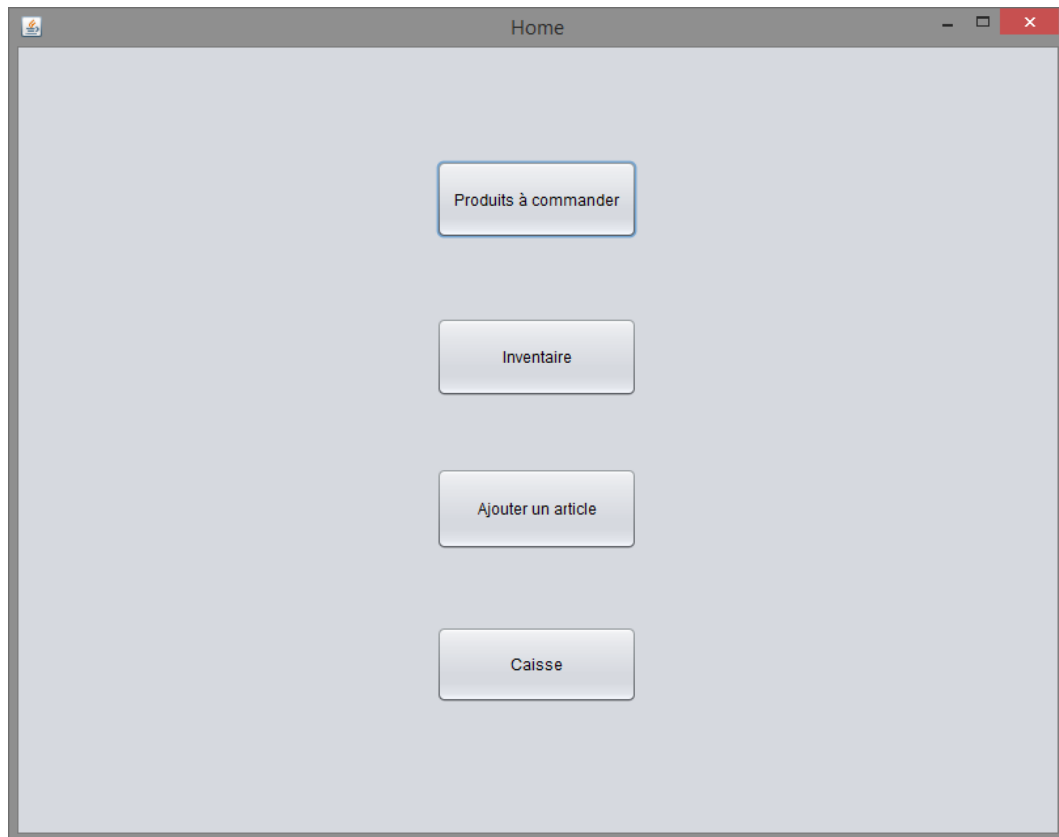
## Implémentation

## Chapitre 7

# Implémentation

### 7.1 Copies d'écran de l'implémentation

Ci-dessous quelque impression de l'implémentation :



Commander des produits

Options

Nom	Quantité	Fournisseur	Prix unitaire	Date commandé	Montant commandé
Celeron J1850	3	Intel	82	2014/11/30 15:15:15	164
Celeron N2808	19	Intel	107	2015/01/02 19:46:43	3 638
Celeron N2815	15	Intel	107		
Sempron 2650	13	AMD	34		

Quantité :

Commander

Annuler

Retourner à l'écran d'accueil

Ajouter un article

Retour

Type de vente
☐ poids
☐ unité

Code barre

Nom

Prix unitaire

Quantité

Seuil de réapprovisionnement

Fournisseur

Nouveau

Multi-sélection: Ctrl/Maj + clic gauche

Ajouter



## 7.2 Livrable du logiciel

### 7.2.1 Installation

1. Ouvrez le projet avec Netbeans.
2. Dans l'onglet *Service*, créez une nouvelle base de donnée nommé **uml** avec en login : **uml** et en password : **groupe3**.
3. Connectez vous à cette nouvelle base de donnée.
4. Faites un click droit sur la base de donnée et sélectionner *exécuter*.
5. Copiez-collez le script scriptBDD.sql et exécuter
6. Copiez-collez le script ajoutBDD.sql et exécuter
7. Vous pouvez maintenant exécuter notre application

### 7.2.2 Mode d'emploi

Une fois le programme lancez, suivi simplement les indications sur l'écran et naviguez à l'aide des boutons.

## 7.3 Bilan de l'état du projet

Notre solution remplit aujourd'hui toutes les exigences du cahier des charges. Cependant il est envisageable d'imaginer d'autres possibilités.

Il serait ainsi intéressant de garder en mémoire les articles passés en manuel à la caisse pour, dans le cas échéants, corriger le code barre.

Le cahier des charges ne le précise pas, mais la possibilité de passer commande de plusieurs articles dans une même commande pourrait être ajoutée.

Une grosse évolution serait la mise en place d'une API côté serveur. Ainsi des clients, type mobiles, pourraient vérifier l'état des stocks n'importe où et n'importe quand.