

# Design Pattern

---

## Les unités de mesures

Présenté par :

**Fadwa ABBES**

**Mohamed LMOUATASSIME**

Encadré par :

**M. Dominique MICHEL**

## Remerciement

*Nous tenons à présenter nos vifs remerciements à tous ceux qui nous ont contribués par leur collaboration et leur soutien à la réalisation de ce travail.*

*Nous tenons tout particulièrement à exprimer notre profonde gratitude et respect à notre encadrant M. Dominique Michel pour ses conseils précieux et pour l'intérêt qu'il a manifesté pour le projet.*

*Nos remerciements vont aussi à tous nos enseignants.*

*Que tous ceux qui nous ont aidés, trouvent ici le témoignage de notre respect et l'expression de nos remerciements.*

*Nous remercions les membres de jury pour l'honneur qu'ils nous font en acceptant de juger ce travail.*

## Table des matières

Remerciement .....	1
I. Présentation du sujet .....	4
II. Les langages et les outils utilisés .....	5
1. Power AMC.....	5
2. Eclipse.....	5
3. Java .....	5
4. Windows Builder .....	5
III. Etude de l'existant.....	6
1. Problématique.....	6
2. Point de vue programmation .....	8
a) Les solutions déjà existantes .....	8
b) Contraintes des solutions existantes.....	8
IV. Les solutions suivies .....	9
1. Design Pattern Money.....	9
2. Design Pattern Chain Of Responsibility .....	9
3. Design Pattern Composite.....	10
V. Solution implémentée .....	13
1. Design Pattern Composite.....	13
2. Design Pattern Singleton .....	14
3. Généricité .....	14
4. Méthode de conversion .....	16
A. Conversion des unités simples .....	16
B. Conversion des unités composées .....	18
5. Stockage des unités et de leurs valeurs .....	19
6. Des exemples de l'application .....	20
VI. Conclusion et Perspectives.....	22
Bibliographie.....	23
Annexe.....	24

## Table des figures

Figure 1 Exemple 1 de conversion.....	6
Figure 2 Exemple 2 de conversion.....	7
Figure 3 Diagramme de classes avec COR.....	9
Figure 4 Diagramme de classes 1 Composite.....	10
Figure 5 Référence entre les classes .....	13
Figure 6 Généricité entre les classes .....	15
Figure 7 Exemple de conversion de distances .....	17
Figure 8 Exemple de conversion de devises.....	18
Figure 9 Exemple de conversion de vitesses.....	19
Figure 10 Exemple graphique de conversion de distances .....	20
Figure 11 Exemple graphique de conversion de devises .....	20
Figure 12 Exemple graphique de conversion de vitesses.....	21
Figure 13 Classe Unité .....	24
Figure 14 Classe UnitComp.....	25
Figure 15 Diagramme de classe de la solution implémentée .....	26

## I. Présentation du sujet

Ce rapport est le compte rendu du projet d'initiation à la recherche réalisé par Fadwa ABBES & Mohamed Lmouatassime durant le deuxième semestre en première année Master de l'année universitaire 2015-2016.

Après quelques études et recherches sur un certain nombre de projets qui nous ont intéressés, nous avons choisi, avec un grand intérêt, de travailler sur ce projet sous l'encadrement de M. Dominique Michel.

Ce qui nous a motivés pour le choix de ce sujet c'est la découverte de nouveaux Designs Patterns et la personnalisation d'un Design Pattern qui nous permettra de répondre à nos besoins fonctionnels tout en évitant les mauvaises habitudes du codage.

Nous commençons par introduire les Designs Patterns ou patrons de conception. Ce sont des moyens appliqués dans la phase de conception du cycle de vie d'un logiciel, qui permettent d'atteindre les objectifs d'un logiciel ou d'une application sans perte de temps ni de mémoire.

L'arrangement des classes dépend des besoins à réaliser et permet de respecter les bonnes pratiques telles que : la réutilisation du code, éviter d'écrire du code inutile par rapport aux fonctionnalités demandées, éviter la redondance, avoir une hiérarchie claire, ...

Notre travail est composé de trois grandes parties:  
La première partie consiste à étudier les Designs Patterns qui peuvent être utiles pour la conversion des unités.

La deuxième partie est l'application des notions déduite précédemment afin de pourvoir personnalisé un Design Pattern qui traite ces conversions tout en restant le plus général possible. Et la dernière partie est l'application du Design Pattern personnalisé dans la phase du développement du cycle de vie de l'application.

Dans la suite du rapport nous présentons les Designs Patterns utiles pour les conversions des unités et la solution proposée.

## II. Les langages et les outils utilisés

Dans cette partie, nous allons élaborer les outils utilisés tout au long de notre travail sur ce projet.

### 1. Power AMC

Est un logiciel de conception.  
Il inclut les modélisations de bases de données (MPD, MCD),  
UML, modélisation de processus métier, ... .  
Il permet de modéliser les traitements informatiques et leurs bases associées.



Initialement créé sous le nom AMC\*Designor par l'éditeur Powersoft, puis racheté par Sybase il a été renommé PowerAMC.

### 2. Eclipse

C'est un environnement de développement intégré libre extensible, universel et polyvalent, permettant de créer des projets de développement mettant en œuvre plusieurs langages de programmation.



Eclipse IDE est principalement écrit en Java et est également utilisé pour écrire des extensions grâce à des bibliothèques spécifiques.

### 3. Java

Java est un langage de programmation orienté objet, utilisé dans le développement des applications basées sur le modèle orienté objet. Il permet d'avoir des applications bien structurées.



La particularité et l'objectif central de Java est que les applications écrites dans ce langage sont très facilement portables sur plusieurs systèmes d'exploitation tels que UNIX, Windows, Mac OS.

### 4. Windows Builder

C'est un plugin Eclipse de conception graphique.

Il permet de créer des applications Java GUI sans dépenser beaucoup de temps à écrire du code. Grâce à ses outils visuels, Windows Builder facilite la création des formes simples et des fenêtres complexes. Le code généré est modifiable, ce qui permet d'ajouter des événements et de contrôler les propriétés.

### III. Etude de l'existant

#### 1. Problématique

Internet a permis l'échange facile de données notamment entre différents pays, chaque pays voulant garder ses propres manières, notamment ses propres méthodes de calcul et mesure, cela engendre un problème de communication entre différents membres. Pour clarifier notre problème on cite ci-dessous deux situations concrètes :

##### Situation 1

Comme exemple un étudiant qui veut déménager à Londres, à la recherche d'un loyer convenable il trouve sur des sites des annonces de location qui proposent certaines propriétés pour un certain prix.

The screenshot shows a web interface for finding properties. At the top, it says 'MORE THAN ONE PLACE MATCHED 'CHELSEA' IN THE UK'. Below this, there are several search filters:

- Choose your location**: A dropdown menu with two options: 'Chelsea, Central London' and 'Chelsea Harbour, Chelsea, London'. Below the menu is a link that says 'or change your location'.
- Search radius**: A dropdown menu with the option 'Within 10 miles'.
- Price range (£)**: Two dropdown menus. The first has '400 PCM' and the second has '1,250 PCM'.
- No. of bedrooms**: Two dropdown menus. The first has 'No min' and the second has 'No max'.
- Property type**: A dropdown menu with the option 'Any'.
- Added to site**: A dropdown menu with the option 'Anytime'.
- Include Let Agreed properties**: A checkbox that is currently unchecked.
- Find properties**: A blue button with white text.

There are two black arrows pointing to the right, one from the 'Search radius' dropdown and one from the 'Price range (£)' dropdowns.

Figure 1 Exemple 1 de conversion

En étant inexpérimenté à ces unités il ne pourra pas bien estimer la valeur de ces propriétés.

Le mieux pour lui est de pouvoir l'exprimer par ces unités nationales « mètre, euro »

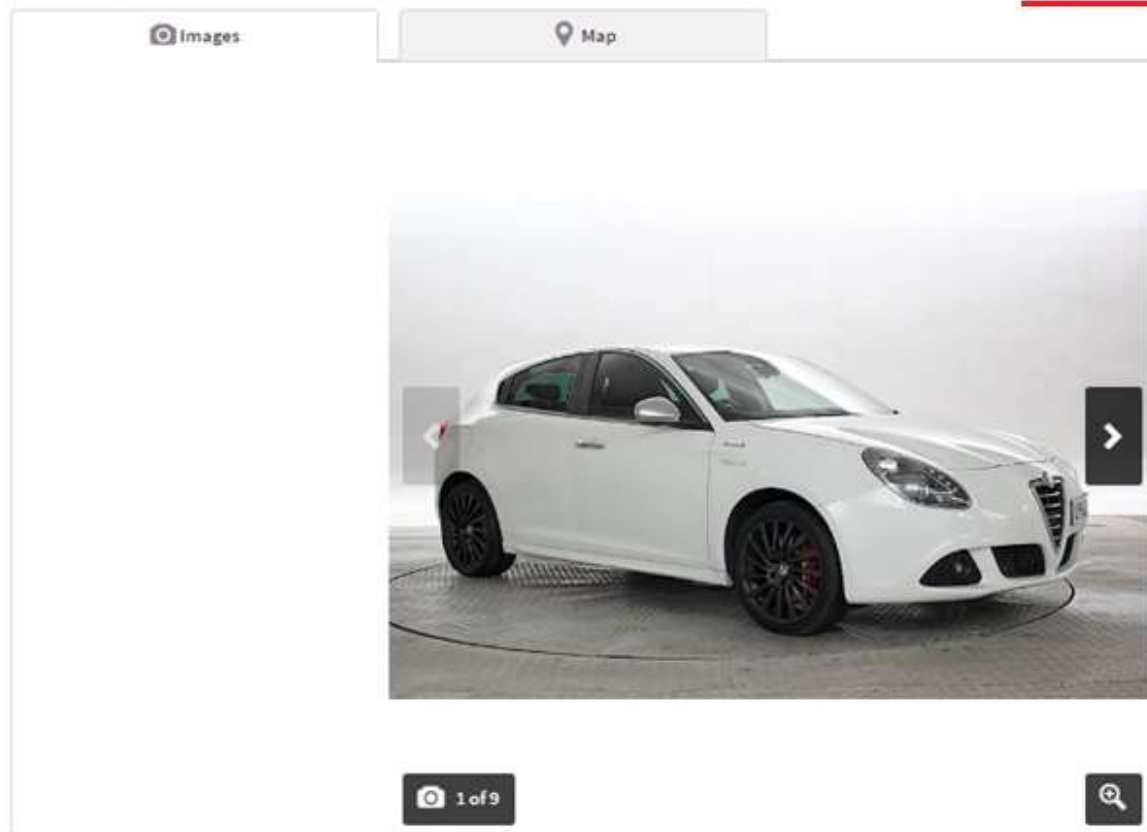
##### Situation 2

***Un amateur de voiture veut acheter une voiture anglaise son prix est estimé en pounds et la distance qu'elle a déjà parcouru en mille pour lui c'est dur de préciser si cette voiture est une bonne offre ou non.***

2013 (62 Reg) Alfa Romeo Giulietta 2.0 JTDm-2 170 Sportiva Ghiaccio White 5 STAN

West London, London

£7,799.00



Posted	8 mins ago	Body type	Other
Make	Alfa Romeo	Fuel type	Diesel
Model	Giulietta	Transmission	Manual
Year	2013	Colour	White
Mileage	91858	Engine size	1956
Seller type	Trade		

Figure 2 Exemple 2 de conversion



## 2. Point de vue programmation

Dans notre projet nous essayerons de résoudre cette problématique au point de vue programmation, notre but sera de donner une solution générale pour tout type de conversion et que cette solution peut être extensible et exploite la relation entre les unités existantes.

### a) Les solutions déjà existantes

Parmi les solutions existantes, on a trouvé que le design pattern Money qui dérive de Quantity est le plus adéquat.

### Quantity

Il existe de nombreux cas où nous voulons représenter des quantités dimensionnées telles que les distances en Pieds ou en mètre, les masses en kilogramme, ... . D'habitude la manipulation de ces quantités n'est pas évidente à cause des systèmes limités.

Mais l'utilisation de certaines dimensions nécessite l'ajout de nouveaux types par exemple le type Argent. Ce dernier représente un cas particulier de Quantity.

Quantity
amount : Number units: Unit
+ , - , * , / < , > , = , to(unit) : quantity toString() parse(string): quantity

### Money

Les opérations de conversion d'argent sont affectées par les taux de change qui sont toujours en évolution.

Les résultats sont plus exacts avec des conversions automatiques grâce au Design Pattern Money.

Une caractéristique particulièrement utile avec Money en plus de ceux pour Quantity est la manipulation de l'arrondissement. L'argent est souvent exprimé avec une partie décimale, mais il ne faut pas utiliser des nombres réels pour la manutention de l'argent d'où le but de l'arrondissement.

Un objet Money, cependant, peut coder ses propres règles pour arrondir, d'où l'inutilité d'avoir les connaissances par rapport aux règles d'arrondissement.

### b) Contraintes des solutions existantes

Suite à nos recherches nous nous sommes aperçus De l'inexistence de solutions qui gèrent à bien la représentation des unités et qui exploitent la relation existante entre différentes unités.

## IV. Les solutions suivies

Suite à notre étude du projet et les propositions de notre encadrant M. Dominique Michel, nous avons décidé de baser notre solution sur les arbres syntaxiques qui nous permettront de schématiser les relations entre les unités disponibles lors des conversions.

Nous nous sommes basées sur cette idée comme idée de base durant la phase de conception. Plusieurs solutions ont alors vu le jour à la quête de la plus adéquate

Dans cette partie nous présentons les différentes pistes que nous avons étudiées.

### 1. Design Pattern Money

Ce Design Pattern est une bonne solution pour les conversions d'argent grâce à sa manipulation des quantités des unités. Cependant, il ne permet pas de traiter la relation entre les unités et donc pour chaque unité il est impératif de définir une méthode de conversion vers toutes les autres unités chose qui rend le code redondant et difficile à maintenir.

### 2. Design Pattern Chain Of Responsibility

Ce design Pattern a été utilisé précédemment pour ce sujet. Le diagramme suivant présente son application sur le projet.

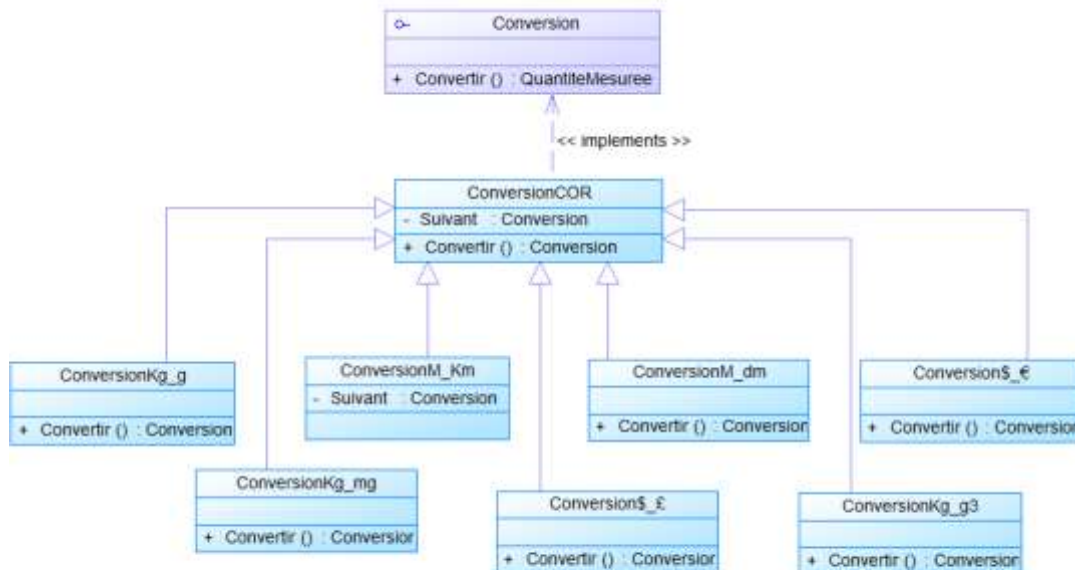


Figure 3 Diagramme de classes avec COR

En étudiant le diagramme, nous avons conclu que ce Design Pattern ne répond pas à nos exigences. Premièrement, ce modèle nécessite autant de classes que de conversions. En plus, il présente une solution statique qui ne permet pas la mise à jour des unités.

### 3. Design Pattern Composite

Après plusieurs modélisations, nous avons remarqué que le Design Pattern Composite est le plus adapté à notre projet et est celui qui répond le plus à nos exigences, telles que le parcours des arbres syntaxiques et l'optimisation du code en évitant la redondance.

Le diagramme suivant présente notre première manipulation de ce patron de conception.

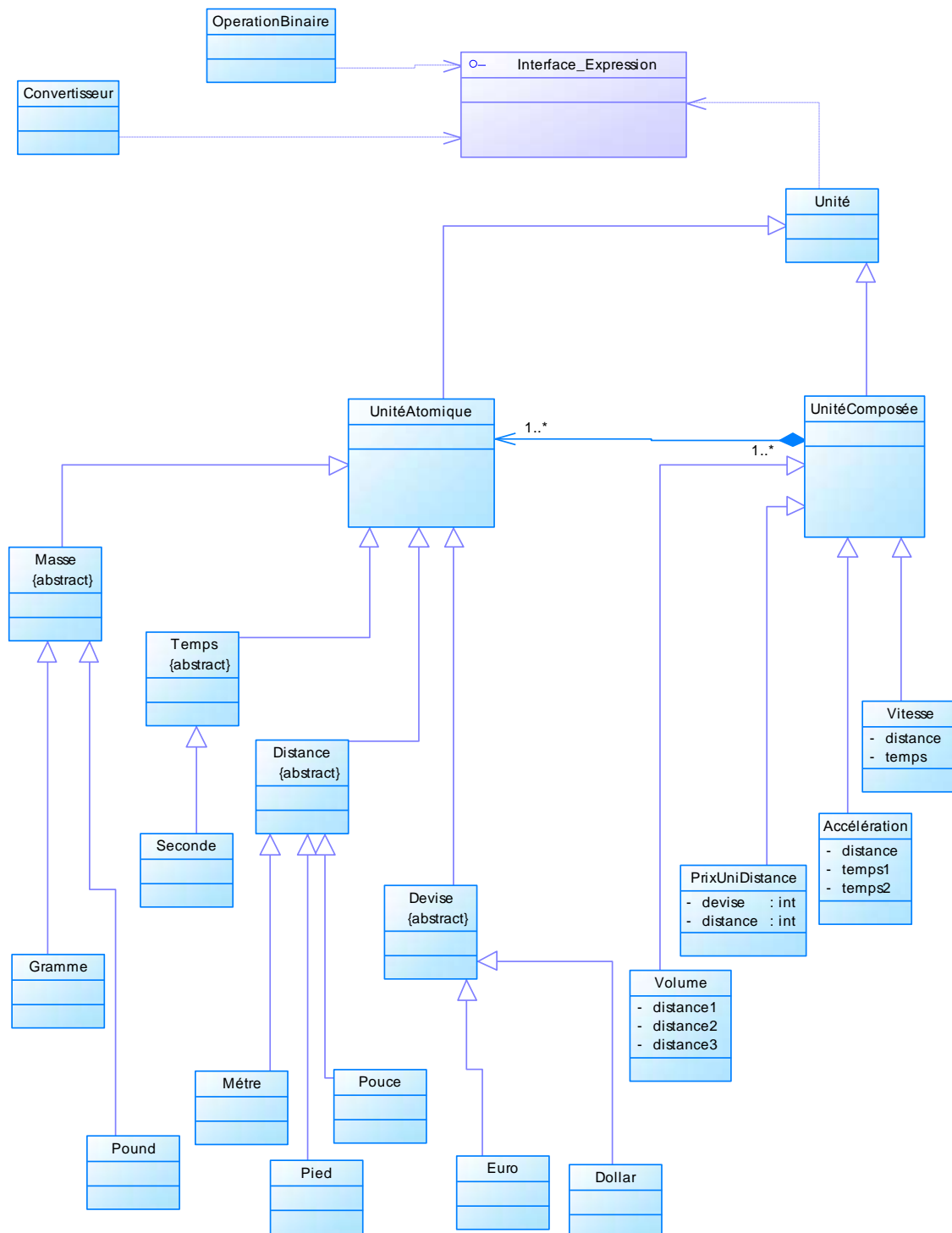


Figure 4 Diagramme de classes 1 Composite

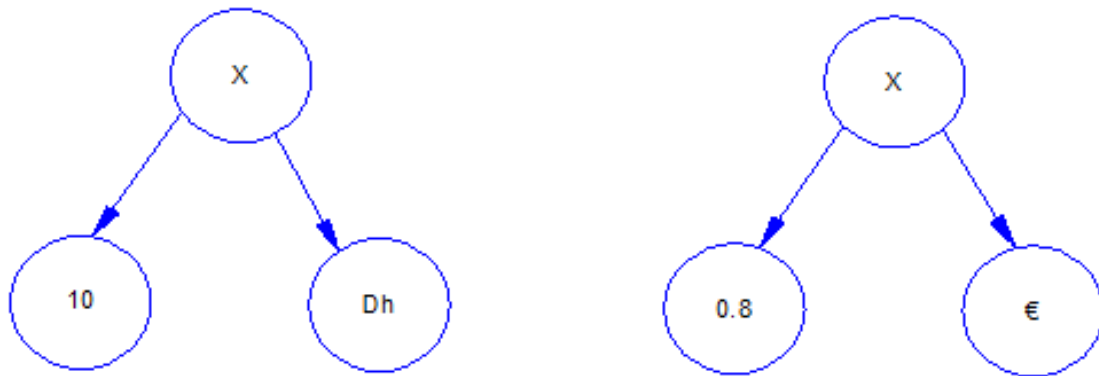
En réalisant ce modèle nous avons essayé de sauvegarder, dans chaque unité, des arbres qui schématisent les relations entre les unités. Donc pour trouver une valeur recherchée, il suffit de parcourir l'arbre en profondeur.

Les schémas suivants illustrent un exemple d'utilisation d'arbre pour représenter les liens entre les unités.

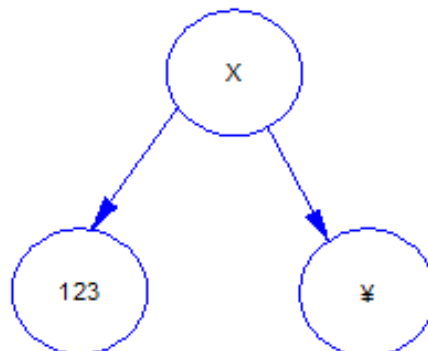
✓ *Schémas de fonctionnement*

Dans la classe Dollar nous créons deux arbres, chacun contient la devise vers laquelle nous effectuons la conversion avec son coefficient.

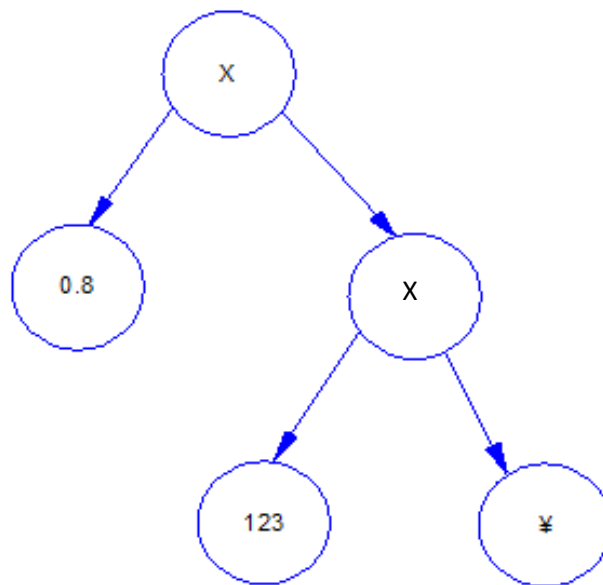
Le premier arbre permet de convertir du Dollar vers le Dirham, et le deuxième vers l'Euro.



Nous ajoutons un arbre dans la classe Euro permettant de convertir vers le Yen, représenté par le schéma suivant.



En exploitant ces deux classes, nous pouvons construire un arbre qui permet de convertir le Dollar en Yen de la manière schématisée ci-dessous.



✓ *Les inconvénients de cette solution*

En analysant ce modèle, nous avons extrait les inconvénients suivants :

- Définition d'une classe pour chaque unité, et de plusieurs arbres pour chaque classe.
- Parcours de l'arbre à sens unique.
- Multitude de tests afin de vérifier l'unité recherchée.

Donc ce modèle n'est pas assez optimisé.

## V. Solution implémentée

« VOUS TROUVEREZ EN ANNEXE LE DIAGRAMME DE CLASSE AINSI QUE LES DEUX CLASSES PRINCIPALES UNITE ET UNITECOMP »

Sur ce qui suit, nous vous présentons la solution choisie composée du design pattern Composite et plusieurs autres designs patterns que nous citerons plus tard.

### 1. Design Pattern Composite

Dans cette solution, nous avons remédié aux lacunes de la solution précédente. Chaque unité a une référence vers une autre unité et un coefficient, cette référence est nulle et ne s'instancie que dans les classes des unités relatives.

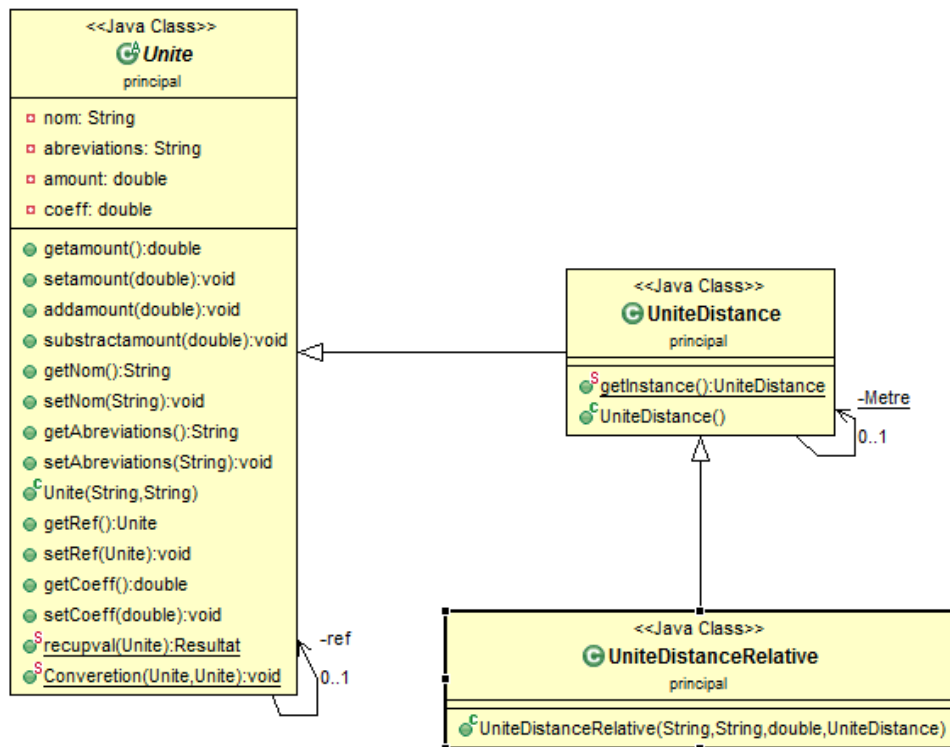


Figure 5 Référence entre les classes

Afin d'optimiser le code nous avons inclus aussi, le Design Pattern Singleton.

## 2. Design Pattern Singleton

Le Design Pattern Singleton permet de créer une seule instance de la classe en question, grâce à une méthode statique qui vérifie si une instance a déjà été créée.

Par la suite, nous appelons la classe en question, sans devoir l'instancier, à partir de la méthode statique déclarée « getInstance ».

Nous avons utilisé ce Design Pattern pour affecter les unités de références aux classes simples telles que : Seconde ->UnitéTemps/ Dollar -> UnitéDevise/ mètre -> UnitéDistance.

## 3. Généricité

La généricité résout une grande partie de l'optimisation de code, et exige des contraintes sur les classes.

Le principe de la généricité est de développer des objets manipulant plusieurs types de données.

Nous avons appliqué la généricité sur la classe UnitéComposée de laquelle héritent toutes les unités composées telles que : UnitéVitesse et PrixUnitDistance pour imposer des contraintes sur ces dernières. La classe UniteComp est, comme son nom l'indique, une classe composée de deux unités. Le diagramme suivant montre la relation entre la classe Unite et la classe UnitComp.

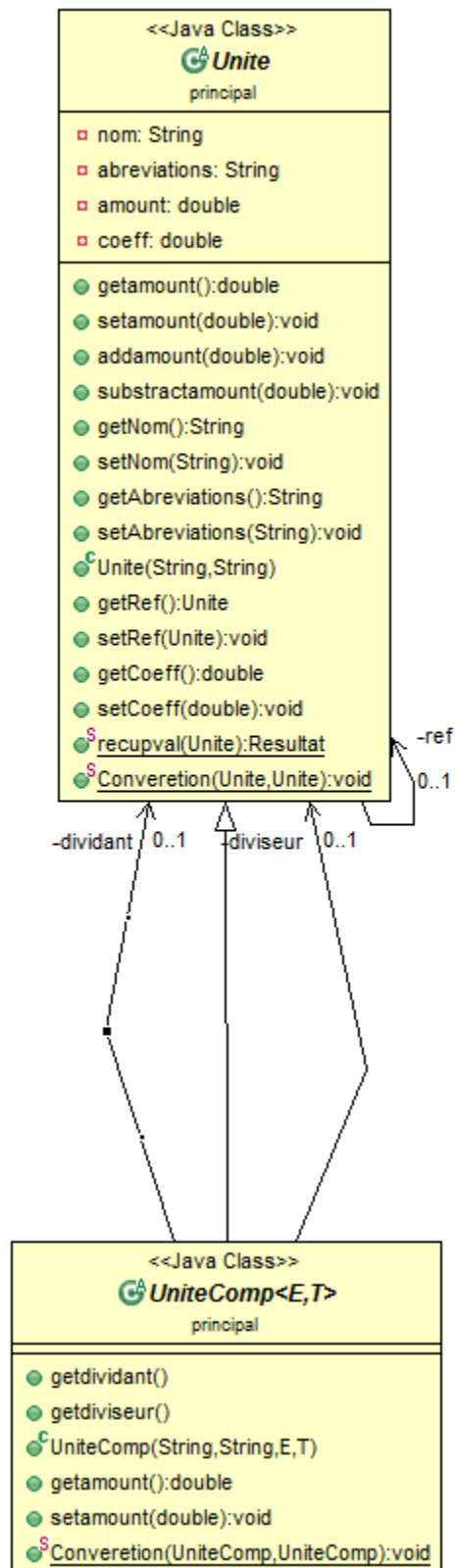


Figure 6 Généricité entre les classes



La classe `UniteVitesse` qui héritera de la classe `UniteComp` doit imposer aux unités qui la composent d'être de type `UniteDistance` et `UniteTemps` donc elle aura comme signature :

```
public class UniteVitesse extends UniteComp <UniteDistance,UniteTemps>
```

## 4. Méthode de conversion

### A. Conversion des unités simples

Les exemples suivants permettent de mieux expliquer le principe de conversion des unités simples.

Nous exploitons les relations entre les Unité des références et les unités relatives, afin de pouvoir effectuer la conversion entre unité de même grandeur.

#### Exemple 1

Le tableau suivant détermine les références des unités de distance, utilisées dans notre exemple, sachant que le mètre est notre unité de référence pour les distances.

Unités	Références
Pied	ft
Pouce	in
centimètre	cm
Mètre	m
Mille	mi
kilomètre	km

Sachant que :

- 1 ft = 12 in
- 1 in = 2.54 cm
- 1 cm = 0.01 m
- 1 mi = 1.609 km

Nos unités de distance sont représentées de la manière suivante :

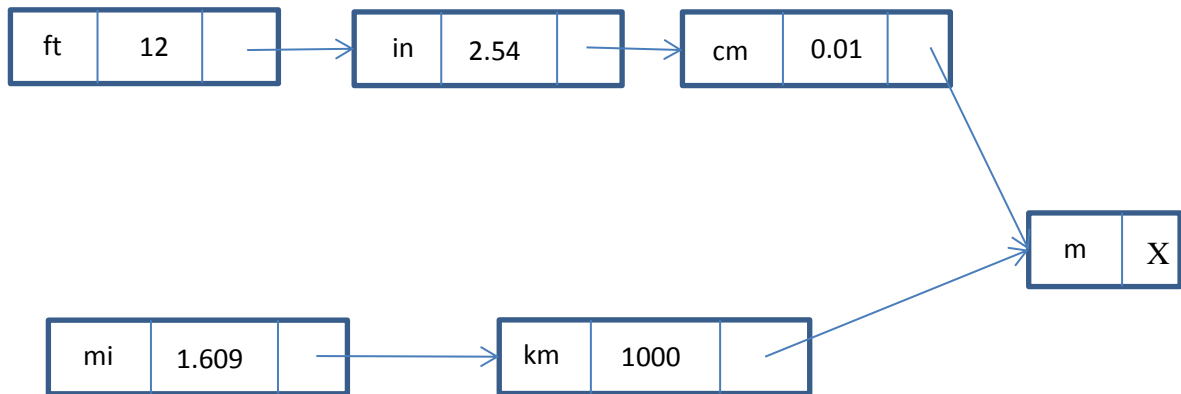


Figure 7 Exemple de conversion de distances

- ✓ Pour convertir une unité vers une autre nous convertissons chacune deux vers l'unité de référence et nous effectuons la division des coefficients.  
**Comme exemple la convention du pied vers le mètre.**
- ✓ Nous récupérons le coefficient du pied vers le pouce, nous le multiplions par le coefficient du pouce vers le cm et ainsi de suite jusqu'à atteindre l'unité de référence de base qui est le mètre.

$$1 \text{ ft} = (12 * 2.54 * 0.01) \text{ m} = 0.3048 \text{ m}$$

- ✓ Pour convertir 1 m en mi, nous récupérons le coefficient du mille vers le km, nous le multiplions par le coefficient du km vers le m, par la suite nous calculons l'inverse.  
Le résultat est l'inverse de la multiplication :

$$1 \text{ m} = 1 / (1.609 * 1000) \text{ mi} = 0,000621371 \text{ mi}$$

En exploitant ces deux résultats nous pouvons en conclure

$$1 \text{ ft} = 0.3048 * 0,000621371 \text{ mi} = 0,000189394 \text{ mile}$$

**Remarque :** nous avons utilisé la méthode `recupval` qui retourne une instance de la classe résultat. Cette méthode contient le coefficient de l'unité convertie vers l'unité référence de base et le nom de l'unité de base. Cela nous permet de tester si les unités à convertir sont de même type et si elles ont un lien entre elles.

## Exemple 2

Le tableau suivant détermine les références des unités utilisées dans l'exemple.  
Le dollar est l'unité de référence pour les devises.

Unités	Références
Dollar	USD
Euro	EUR
Dirham marocain	MAD

- 1 EUR = 1.131 USD
- 1 MAD = 0.091 EUR



Figure 8 Exemple de conversion de devises

Nous appliquons le même principe vu avec l'unité distance.

- ✓ Pour convertir 1 MAD en USD, nous multiplions les coefficients de l'unité donnée jusqu'à l'unité de référence demandée.

$$1 \text{ MAD} = (0.091 \times 1.131) \text{ USD} = 0.102921 \text{ USD}$$

- ✓ Pour convertir 1 USD en EUR, nous calculons l'inverse de la multiplication des coefficients :

$$1 \text{ USD} = 1/1.131 \text{ EUR} = 0.8843 \text{ EUR}$$

## B. Conversion des unités composées

La conversion des classes composées se basent sur la conversion des classes simples. Comme nous l'avons mentionné précédemment, une classe composée dépend de deux classes simples. Donc pour une convention globale, il suffit de convertir chaque unité simple de son côté.

### Exemple :

Nous prenons l'unité composée vitesse.

- ✓ Pour convertir 1km/h en m/s, nous parcourons l'unité simple distance et l'unité simple temps.

$$1 \text{ km/h} = 1000/3600 \text{ (m/s)} = 0.2778 \text{ m/s}$$

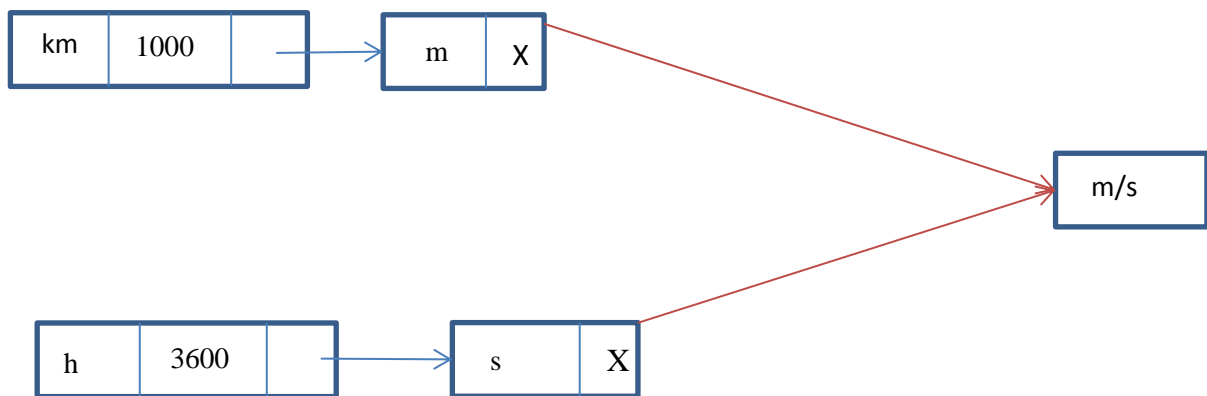


Figure 9 Exemple de conversion de vitesses

## 5. Stockage des unités et de leurs valeurs

Pour faciliter l'extensibilité et la maintenance de notre application ainsi que pour éviter d'instancier à chaque fois toutes les unités, nous les avons stockées dans des fichiers Texte.

Pour exploiter ces fichiers, nous avons créé une classe « LoadF » où nous avons définie des méthodes qui lisent ces documents et retournent une HashMap qui permet de retrouver une unité à partir de son abréviation.

Ce bout de code nous permet de charger les unités de Distance et d'affecter au dollar une valeur de 5:

```
Map<String,UniteDistance> Distance= LoadF.LoadDistance();
Distance.get("M").setamount(5);
```

Ci-dessous nous présentons un exemple de fichier texte que nous avons utilisé.

```
1 Kilometre km 1000 M
2 centimetre cm 0.01 M
3 millimetre mm 0.001 M
4 inche in 2.54 cm
5 feet ft 12 in
6 Yard yd 0.9144 M
7 Mile mil 1.6 km
```

Nous pouvons ajouter une nouvelle unité à condition que celle-ci soit en relation avec une autre déjà existante, sinon le programme génère une exception lors de la conversion.

## 6. Des exemples de l'application

Nous avons créé une interface graphique simple pour éviter de faire les tests en mode console, et améliorer l'affichage du résultat.

La figure ci-dessous présente un exemple de conversion de 1 Km en Mille.

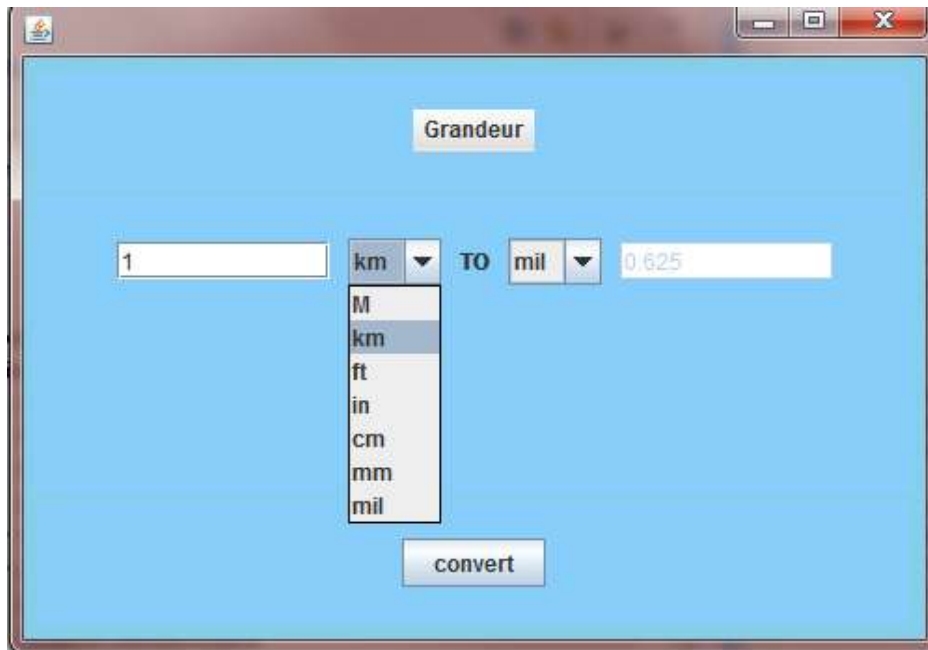


Figure 10 Exemple graphique de conversion de distances

Un deuxième exemple de conversion de 12 Dollar en Euro.

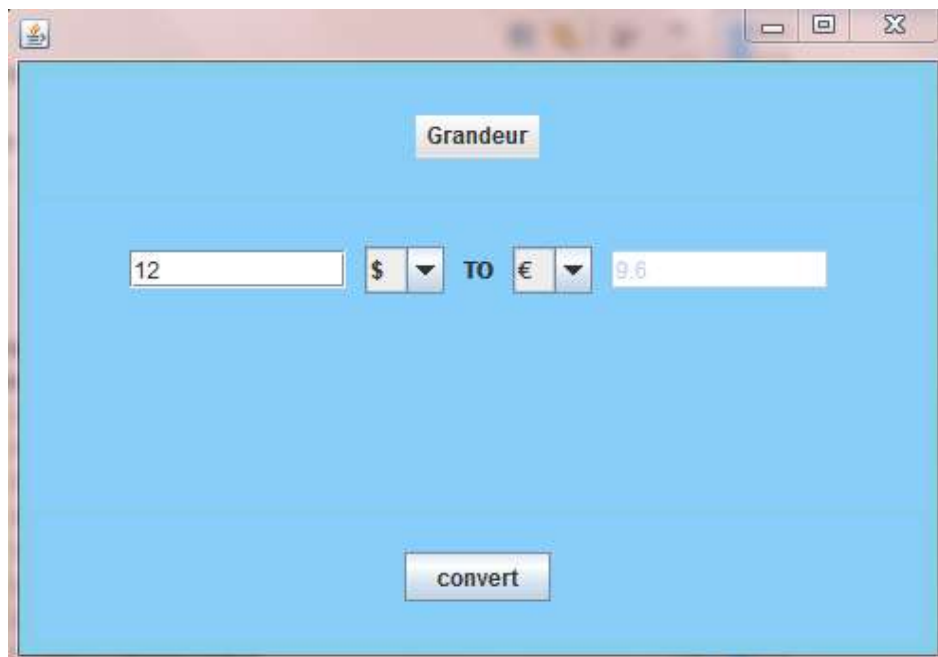
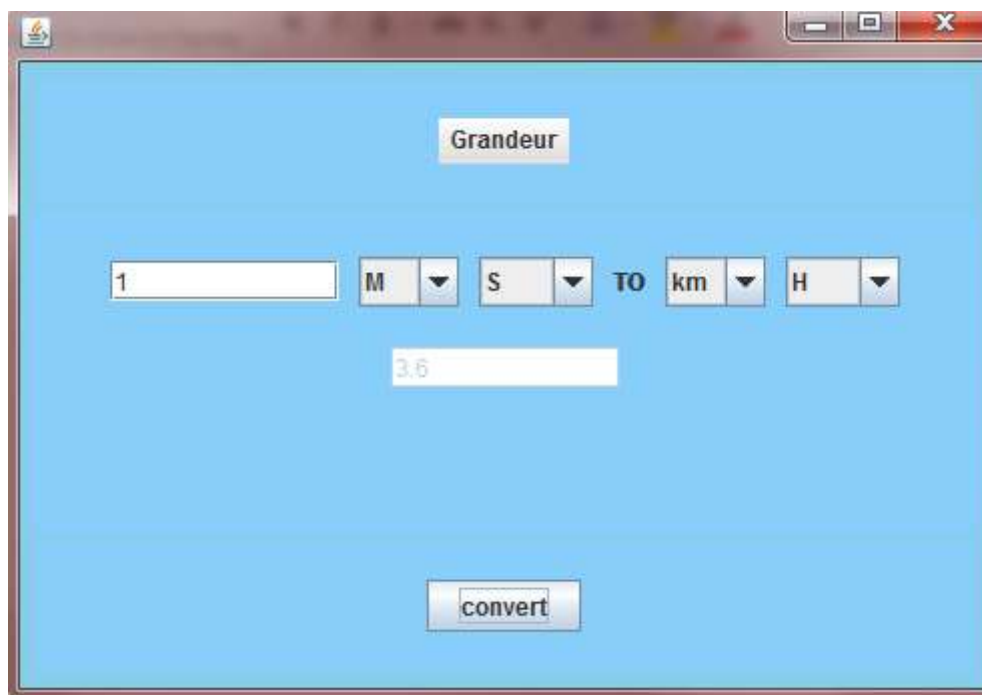


Figure 11 Exemple graphique de conversion de devises

Ci-dessous nous convertissons un 1 M/S en Km/H.



The screenshot shows a software window titled "Grandeur" with a light blue background. At the top, there is a label "Grandeur". Below it, there is a row of input fields and dropdown menus. The first field contains the number "1". This is followed by a dropdown menu showing "M", then another dropdown menu showing "S", then the text "TO", then a dropdown menu showing "km", and finally a dropdown menu showing "H". Below this row, there is a text field containing the result "3.6". At the bottom center of the window, there is a button labeled "convert".

Figure 12 Exemple graphique de conversion de vitesses

## VI. Conclusion et Perspectives

Pendant toute la durée de notre projet, nous avons dû surmonter plusieurs difficultés au niveau du temps, de la manipulation de nouvelles approches et de la programmation.

Cependant, ce travail était une bonne opportunité pour l'application de nos connaissances avec des nouveaux acquis que nous avons vus en cours durant cette année universitaire tels que les Designs Patterns.

Nous nous sommes concentrés, au début, sur l'étude de plusieurs solutions afin d'extraire les plus adéquates et conformes aux besoins et aux exigences du projet.

Ce projet nous a permis d'apprendre de nouvelles méthodes de travail en programmation et de découvrir de nouveaux Designs Patterns. Nous avons intégré ces méthodes afin d'aboutir à la solution finale.

En guise de perspective, le projet peut être amélioré par :

- le traitement des unités plus complexes telles que la force, l'énergie, ... .
- L'interaction avec une base de données pour importer les unités, au lieu d'utiliser un fichier texte.
- L'amélioration des interfaces graphiques.

## Bibliographie

Debrauwer, L. (s.d.). *Design patterns en Java : les 23 modèles de conception : descriptions et solutions illustrées en UML 2 et Java*.

*eclipse*. (s.d.). Récupéré sur <https://eclipse.org/windowbuilder/>

*MartinFowler*. (s.d.). Récupéré sur <http://martinfowler.com/eaDev/quantity.html>

MICHEL, M. D. (s.d.). Cours Designs Patterns.



## Annexe

```

1 package principal;
2 import Exception.ExceptionType;
3 public abstract class Unite {
4     private String nom;
5     private String abbreviations;
6     private double amount=0;
7     private Unite ref=null;
8     private double coeff=1;
9     public double getamount() {
10         return amount;
11     }
12     public void setamount(double amount) {
13         this.amount = amount;}
14     public String getNom() {
15         return nom;
16     }
17     public void setNom(String nom) {
18         this.nom = nom;
19     }
20     public String getAbbreviations() {
21         return abbreviations;
22     }
23     public void setAbbreviations(String abbreviations) {
24         this.abbreviations = abbreviations;
25     }
26     public Unite(String nom, String abbreviations) {
27         this.nom = nom;
28         this.abbreviations = abbreviations;
29     }
30     public Unite getRef() {
31         return ref;
32     }
33     public void setRef(Unite ref) {
34         this.ref = ref;
35     }
36     public double getCoeff() {
37         return coeff;
38     }
39     public void setCoeff(double coeff) {
40         this.coeff = coeff;
41     }
42     public static Resultat recupval(Unite p){
43         if(p.ref==null){
44             return new Resultat (1,p.getNom());
45         }
46         Resultat r=recupval(p.ref);
47         r.setResultat(r.getResultat()*p.coeff);
48         return r;
49     }
50     public static void Converetion(Unite from, Unite to) throws ExceptionType {
51         Resultat ra=recupval( from) ;
52         Resultat rb=recupval(to);
53         if (!ra.getNomClasse().equals(rb.getNomClasse()))
54             throw new ExceptionType();
55         Double coef=ra.getResultat()/rb.getResultat();
56         to.setamount(from.getamount()*coef);
57     }
58 }
--

```

Figure 13 Classe Unité

```

1 package principal;
2
3 import Exception.ExceptionType;
4
5 public abstract class UniteComp <E extends Unite, T extends Unite > extends Unite {
6
7
8     private E dividant;
9     private T diviseur;
10
11     public E getdividant(){
12         return dividant;
13     }
14     public T getdiviseur(){
15         return diviseur;
16     }
17     public UniteComp(String nom, String abbreviations, E dividant,T diviseur) {
18         super(nom, abbreviations);
19         this.dividant=dividant;
20         this.diviseur=diviseur;
21         this.diviseur.setamount(1);
22         // TODO Auto-generated constructor stub
23     }
24
25     @Override
26     public double getamount(){
27
28         return dividant.getamount()/diviseur.getamount();
29     }
30     public void setamount(double D){
31         dividant.setamount(D);
32         diviseur.setamount(1);
33     }
34
35     public static void Converetion(UniteComp from, UniteComp to) throws ExceptionType {
36         Unite.Converetion(from.dividant, to.dividant);
37         Unite.Converetion(from.diviseur, to.diviseur);
38     }
39 }
40
41 }
42

```

Figure 14 Classe UnitComp

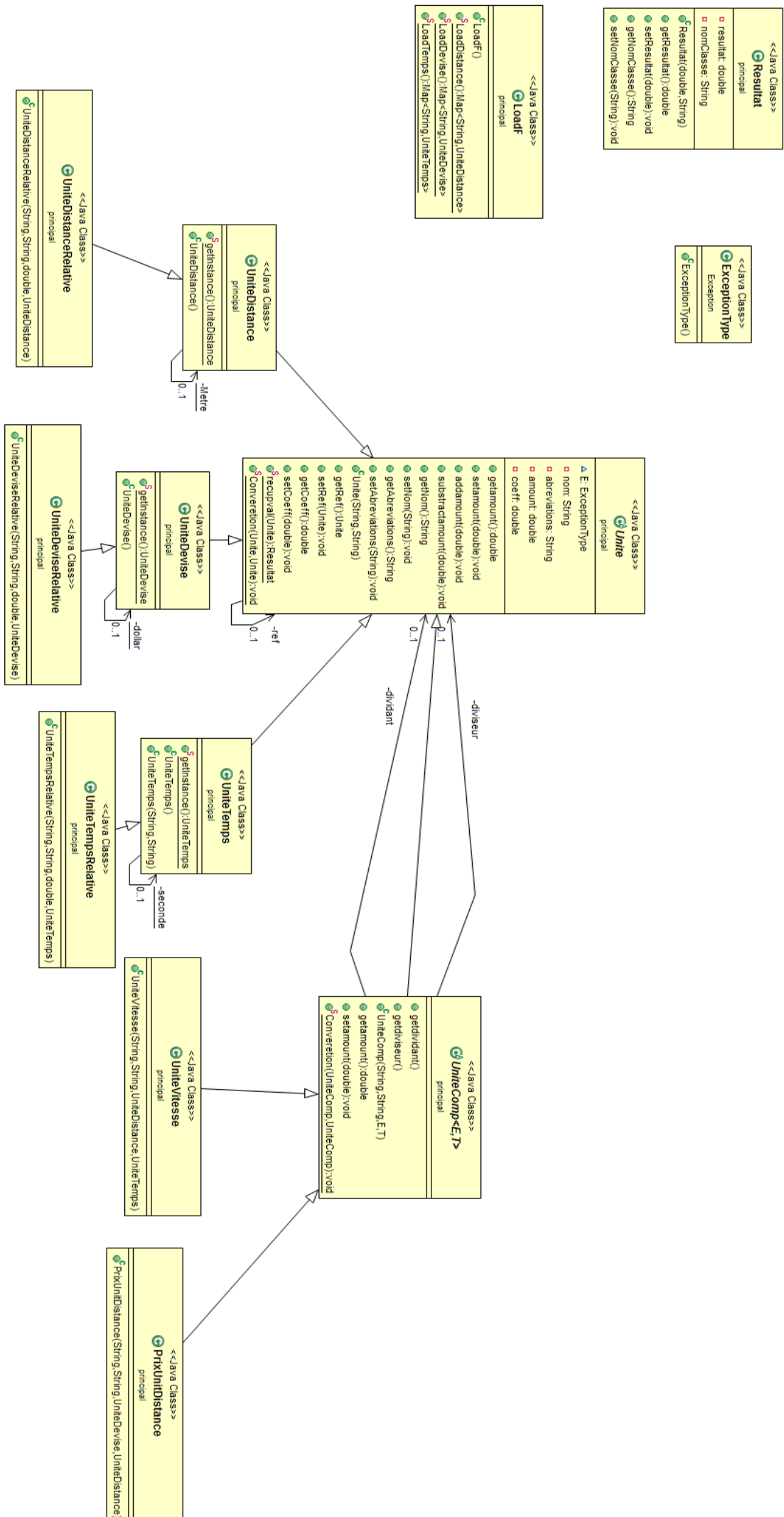


Figure 15 Diagramme de classe de la solution implémentée

## Historique des Réunions

Durant la période de réalisation du projet, Nous avons eu des réunions avec notre encadrant M Dominique MICHEL. Ci-dessous vous trouverez le résumé des plus importants rendez-vous :

**Décembre 2015** : Au cours de cette réunion, M Dominique Michel nous a présenté le projet avec quelques détails pour pouvoir entamer les recherches.

**Décembre 2015** : Après une première étude, nous avons fixé cette réunion pour mieux comprendre quelques notions.

**Janvier 2016** : Nous avons détaillé l'exploitation des unités lors des conversions.

**Avril 2016** : Durant cette réunion, nous avons proposé à notre encadrant le premier diagramme de classes et nous avons discuté par rapport aux choix de certains Designs Patterns.

**Mai 2016** : Nous avons consacré cette réunion pour présenter l'application, à notre encadrant, et pour résoudre quelques problèmes de code.