Master informatique,
Université de
Marne La Vallée

Projet de Génie Logiciel

Symphonie D
(Dupratcated)

Manuel de l'utilisateur

GARCIA Laurent
PEÑA SALDARRIAGA Sebastián
PERSIN Nathalie
VALLEE Fabien

31/05/2005

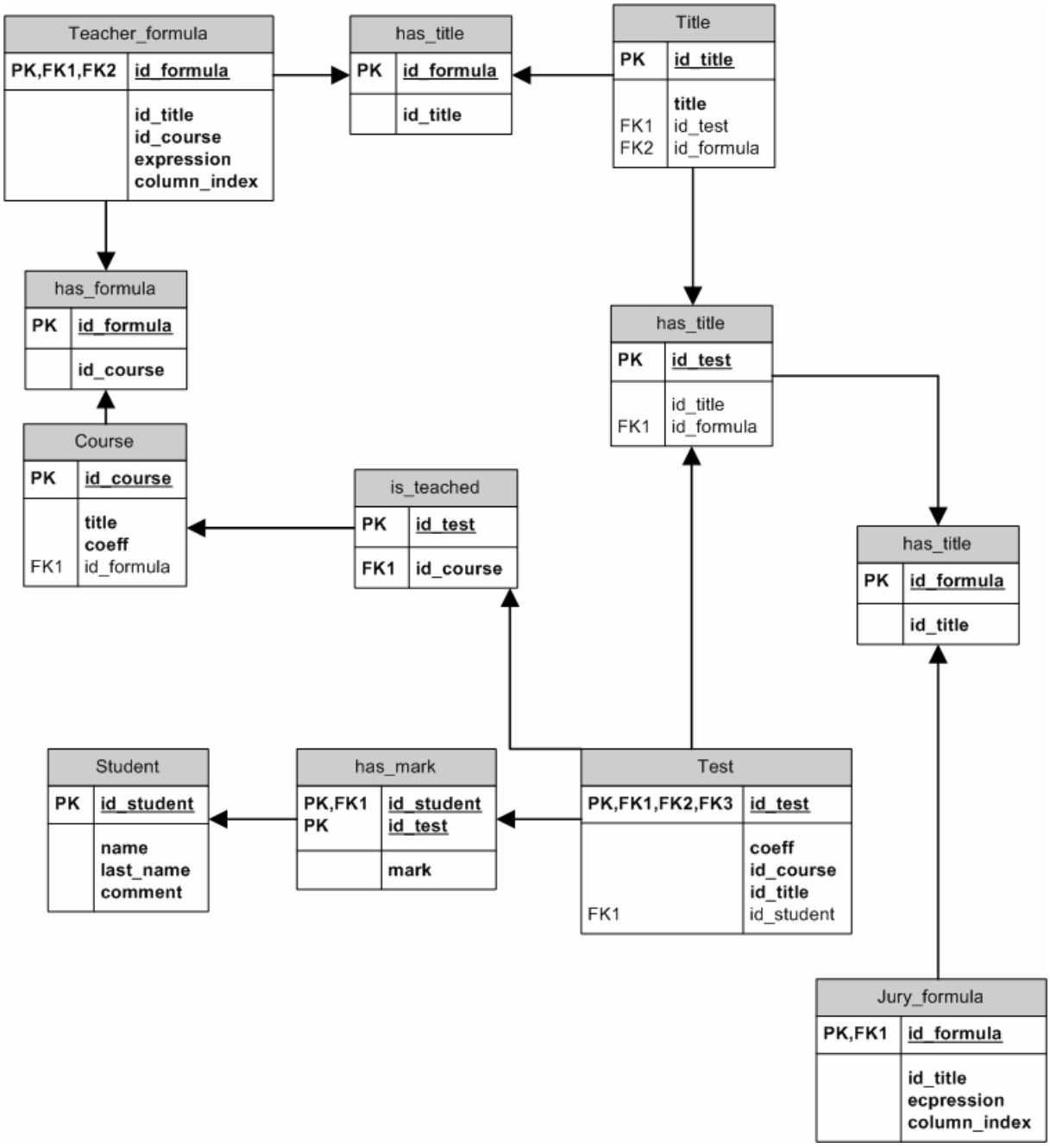
I. Introduction

Dans le cadre de notre formation en master informatique à l'université de Marne-la-Vallée, nous avons élaboré un projet en génie logiciel. Le but de ce projet est de réaliser un utilitaire de gestion des notes des étudiants. Il permet aux enseignants d'entrer les notes et commentaires, aux secrétaires d'éditer différentes feuilles de notes, et aux membres de jurys de noter les résultats des délibérés.

Le groupe Dupracated, composé de Laurent GARCIA, Nathalie PERSIN, Sebastian PENA SALDARRIAGA, et Fabien VALLEE, a élaboré ce projet en essayant de réaliser une application efficace et facile d'utilisation.

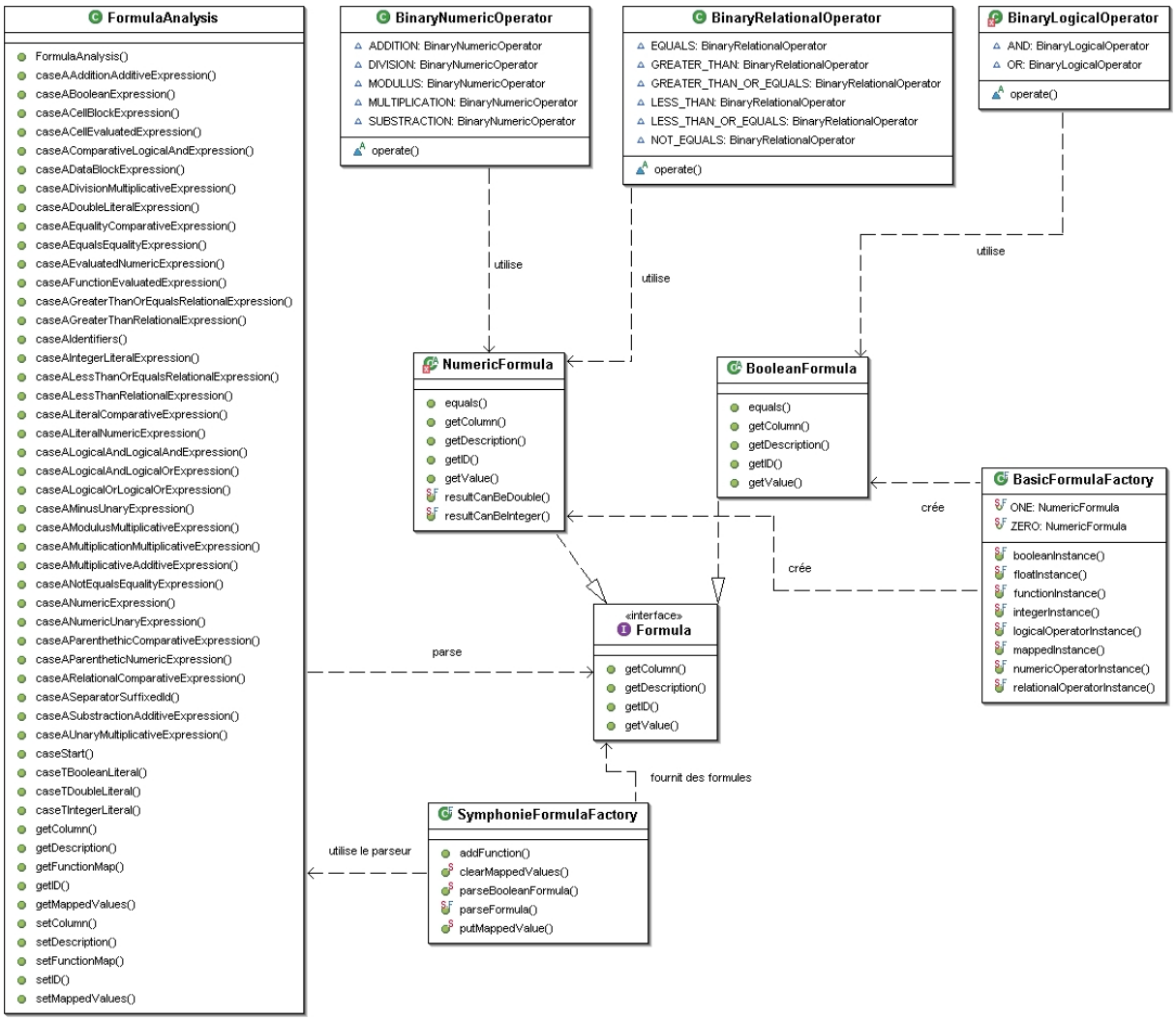
Vous trouverez dans ce rapport l'architecture complète du projet, accompagnée de diagrammes UML.

II. Architecture



III. Description des classes

1. Le package `fr.umlv.symphonie.data.formula`



Ce package, et ses sous packages définissent les classes et interfaces qui permettent d'ajouter le support des formules à notre logiciel.

L'interface paramétrée Formula, définit les méthodes de base que doit implémenter une formule.
Il existe deux sous-types principaux utilisés par Symphonie : `NumericFormula` et `BooleanFormula`.

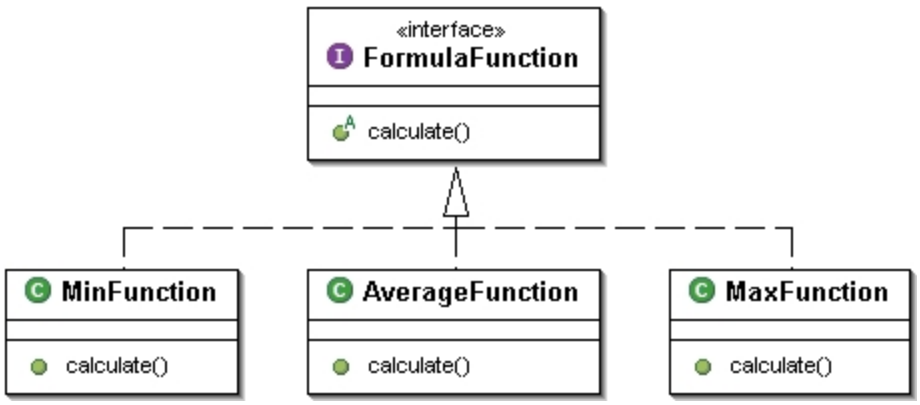
La classe `BasicFormulaFactory` est une `factory` qui permet de créer des objets Formula à partir des types primitifs : `int`, `float` et `boolean`. Elle permet aussi de créer des variables et des formules qui calculent le résultat entre deux opérandes et un opérateur binaire.

Le calcul des sommes, divisions, comparaisons et toute autre opération logique, est effectué par une instance unique d'une énumération définie selon le type d'opération. Il existe trois types d'opérateurs à savoir :
- numérique : + - / * %
- relationnel: < <= > >= == !=
- logique : && ||

La classe `FormulaAnalysis` est à la fois un parseur et un `builder` de formules.
Il utilise l'outil Sablecc qui construit un AST et permet de le parcourir grâce à un design-pattern `Visitor`.

La classe `SymphonieFormulaFactory` est la seule et unique responsable de la création de TOUTES les formules utilisées par Symphonie. Elle délègue la construction des formules à la classe `FormulaAnalysis` et ne s'occupe que de la logique factory : stockage des objets, réutilisation des formules, map des variables.

2. Le package `fr.umlv.symphonie.data.formula.function`

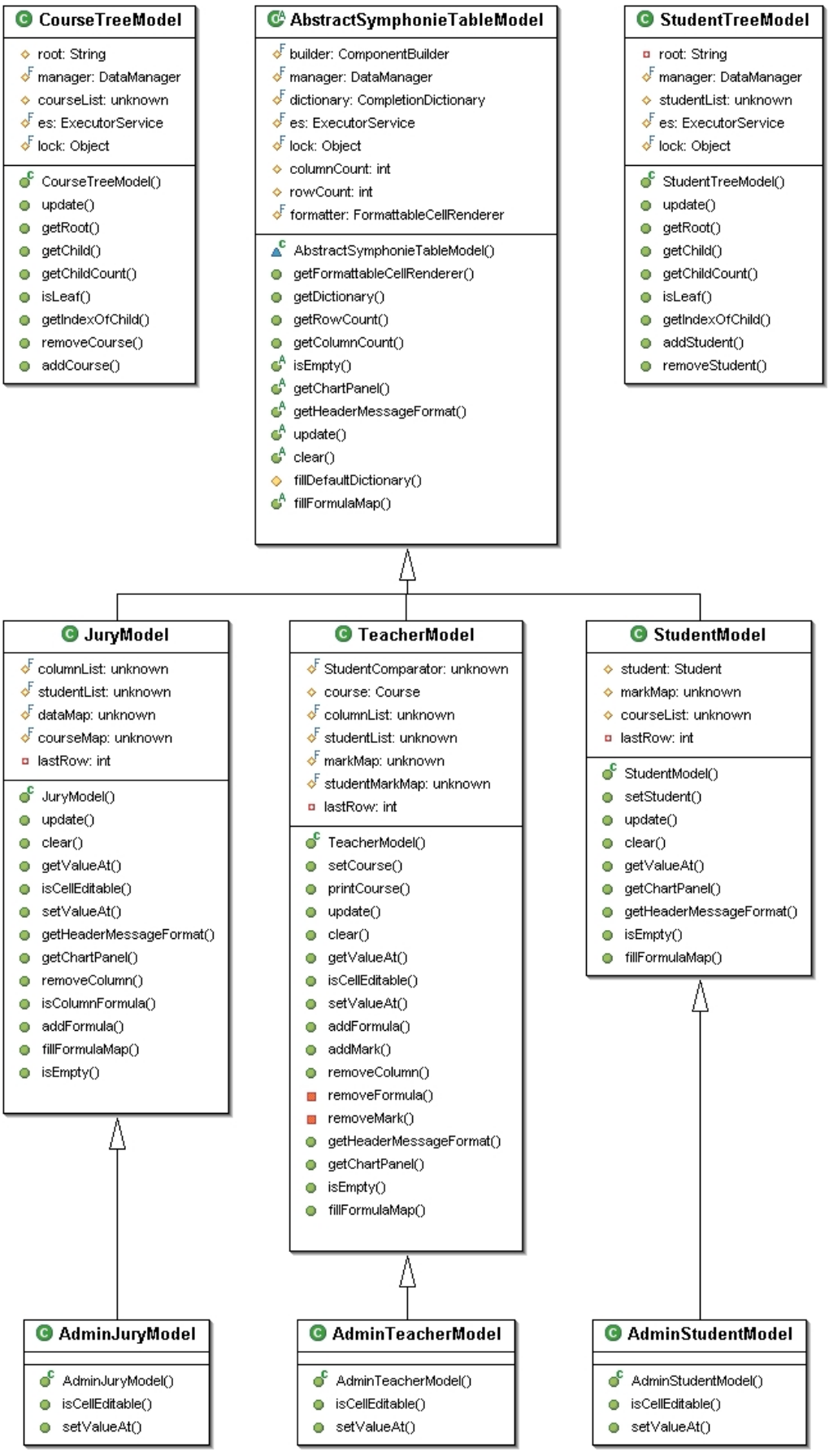


Ce package ajoute la notion de fonction aux formules Symphoniques.

Une fonction symphonie est une classe qui implémente l'interface `FormulaFunction`. Cette interface permet de manipuler indifféremment des fonctions sans se soucier de leur implémentation ni de leur résultat.

Par défaut, Symphonie fournit trois fonctions primitives : `MaxFunction`, `MinFunction` et `AverageFunction`.
Un utilisateur peut rajouter ses propres fonctions, la marche à suivre est décrite en IV.

3. Le package `fr.umlv.symphonie.model`



Ce package contient toutes les classes servant aux modèles graphiques des vues. Il y a deux types de modèles dans le programme symphonie : des modèles d'arbres (pour représenter les matières et les étudiants) et des modèles de table (pour afficher les données des différentes vues.

En ce qui concerne les modèles de table, il y en a deux par type de vue (étudiant, enseignant et jury) : une normale et une autre étendue pour le mode administrateur.

Les classes de ce package servent d'intermédiaire entre les données brut et leur manipulation par l'utilisateur final. Elles permettent l'affichage et l'édition des valeurs, à différents degrés selon la vue dans laquelle on se trouve ainsi que le mode utilisé (normal ou administrateur).

- La classe `CourseTreeModel.java`

Cette classe sert à afficher toutes les matieres disponibles. Des listeners seront ajoutés à l'arbre utilisant ce modele, afin de le faire interagir avec la table affichée dans la vue enseignant.

- La classe `StudentTreeModel`

Cette classe sert à afficher tous les etudiants presents. Des listeners seront ajoutés à l'arbre utilisant ce modele, afin de le faire interagir avec la table affichée pour la vue etudiant.

- La classe abstraite `AbstractSymphonieTableModel`

Cette classe regroupe les points communs entre tous les modèles de tables dans les trois vues disponibles. Elle étend la classe abstraite AbstractTableModel. En plus des méthodes standard des `AbstractTableModel`, ces modèles doivent :

- être internationalisable
- interagir avec un serveur de données (présence d'un `DataManager`)
- créer un graphique de la vue courante qu'ils affichent
- être updatable et « clearable »

Cette classe abstraite procure donc des méthodes déjà codées (code commun) ou abstraites nécessaire au fonctionnement du programme.

- La classe StudentModel :

Etend `d'AbstractSymphonieTableModel`.
Cette classe permet l'affichage des notes d'un étudiant et sa moyenne, dans chaque matière. Rien n'est éditable en vue étudiant.

-

La classe `AdminStudentModel`

Cette classe étend de `StudentModel`. Elle rajoute des possibilités d'édition des notes, des intitules des matières et de leurs épreuves, ainsi que leurs coefficients.

- La classe `TeacherModel`

Cette classe étend `d'AbstractSymphonieTableModel`. Elle permet l'affichage des notes des étudiants dans une matière donnée, le tout par épreuves. Elle permet aussi l'ajout et la suppression d'épreuves, ainsi que de formules en colonne.

Ce modèle permet l'édition des intitules des épreuves, leurs coefficients, ainsi que les notes en elles-mêmes. Il est aussi possible de formater des cellules via ce modèle.

- La classe `AdminTeacherModel`

Dérivée de `TeacherModel`, elle ajoute la possibilité d'éditer le nom des étudiants.

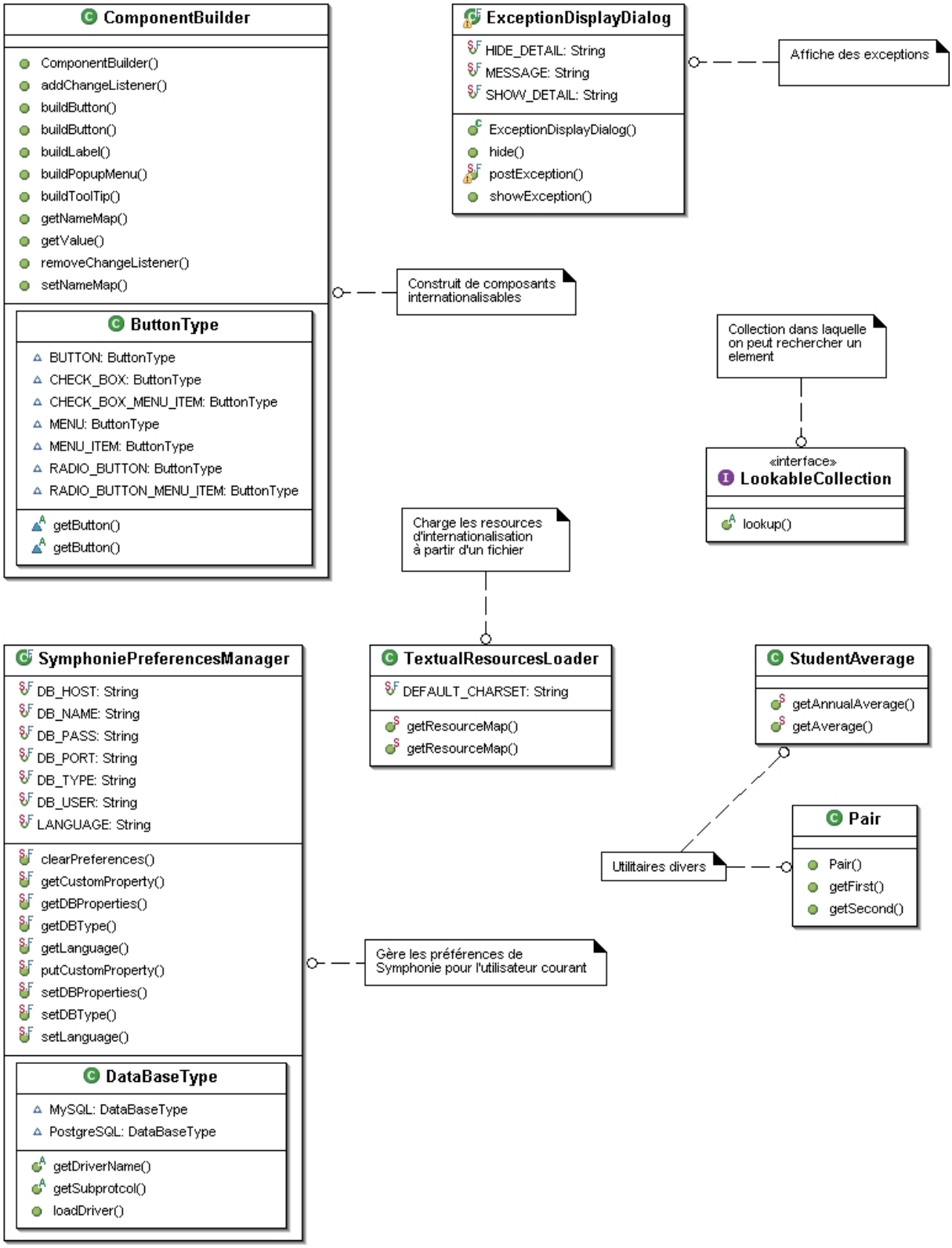
- La classe `JuryModel`

Cette classe dérive de `AbstractSymphonieTableModel` et permet d'afficher les moyennes de tous les étudiants dans toutes les matières, ainsi qu'un commentaire pour chacun d'eux. Les commentaires sont éditables. Ce modèle permet aussi d'ajouter/retirer des formules, ainsi que le formatage des cellules.

- La classe `AdminJuryModel`

Dérivée de `JuryModel`, cette classe ajoute des possibilités d'édition étendues :
Elle permet d'éditer les noms des étudiants, les intitules des matières et leurs coefficients.

4. Le package `fr.umlv.symphonie.util`



Ce package contient un ensemble de classes utilitaires hétéroclites.

La `SymphoniePreferencesManager` est responsable de la gestion des préférences pour un utilisateur donné, elle gère notamment la langue et la base de données préférées.

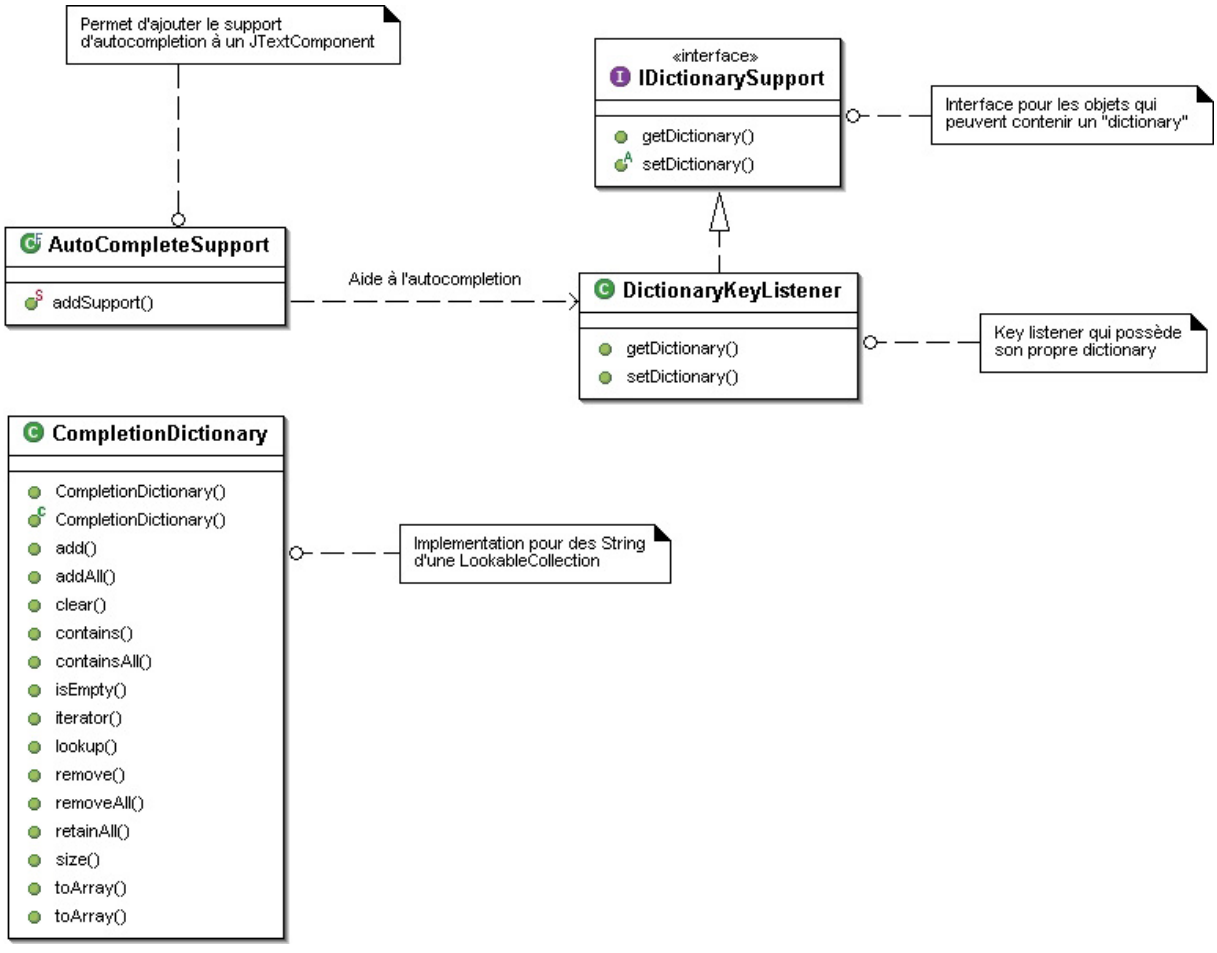
L'interface `LookableCollection` est la superinterface de tous les dictionnaires d'autocompletion.

La classe `ComponentBuilder` est utilisée pour la construction de composants graphiques internationalisables, il utilise le design-pattern Observer afin de prévenir tous ses composants d'un éventuel changement de langue.
Les ressources textuelles du `ComponentBuilder` sont généralement fournies par la classe `TextualResourceLoader`, cette classe une sorte de `ResourceBundle`, mais adapté à notre besoin.

La classe `ExceptionDisplayDialog` permet d'afficher dans une boite de dialogue n'importe quelle Exception. Une instance unique de cette classe est responsable de l'affichage de toutes les exceptions de symphonie.

La classe Pair est une classe paramétrée qui permet de stocker deux objets de type non-défini au préalable, quant à `StudentAverage` elle contient des méthodes statiques auxquelles est délégué le calcul de la moyenne pour tous les élèves.

5. Le package `fr.umlv.symphonie.util.completion`



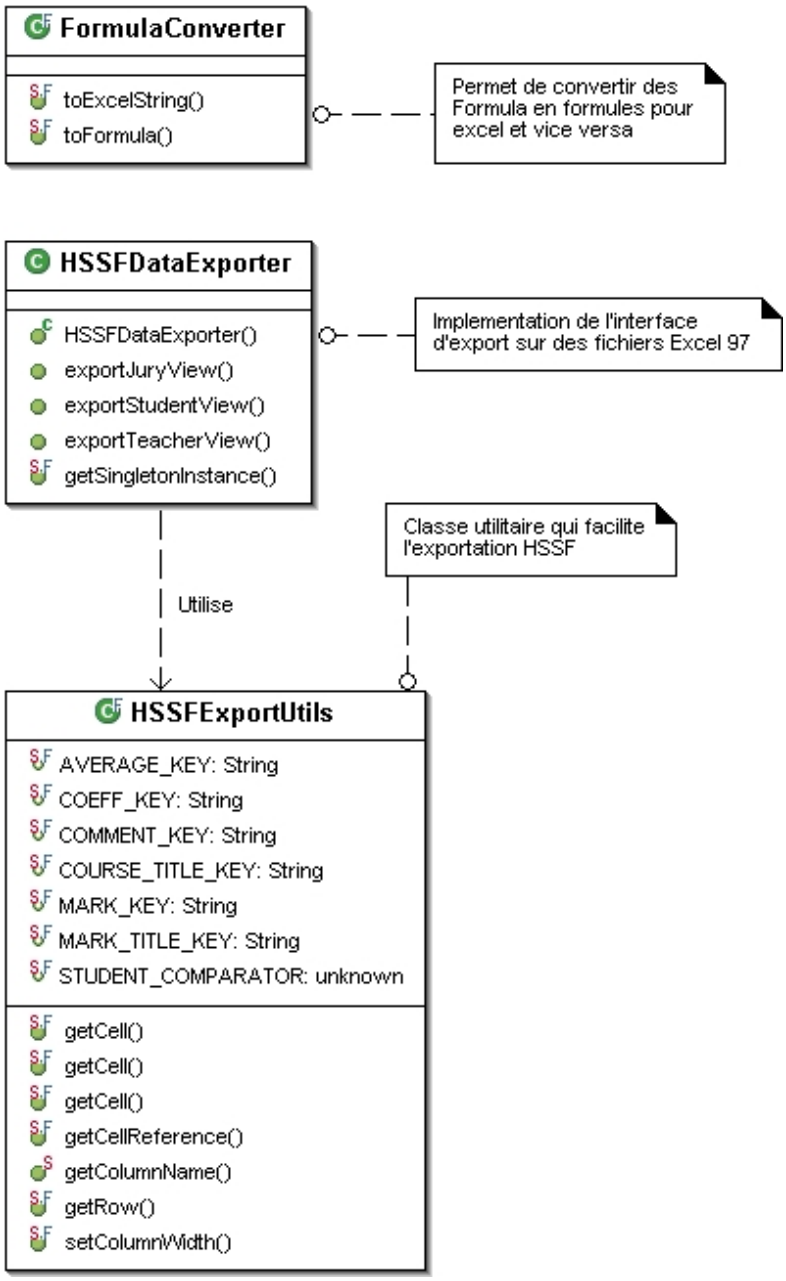
Package qui permet d'ajouter le support d'autocompletion aux composants texte de Symphonie.

Le travail d'autocompletion est réalisé par la classe `DictionaryKeyListener`. Cette classe rajoute le support d'un dictionnaire, `IDictionarySupport`, à un `KeyListener` classique.

La classe `CompletionDictionary` est une classe qui a été créée dans le but de faciliter l'utilisation des dictionnaires d'autocompletion.

La classe `AutoCompleteSupport` agit comme un `Decorator` et permet d'attacher à la volée, la fonctionnalité de l'autocomplétion à un composant texte pré-éxistant.

6. Le package `fr.umlv.symphonie.util.dataexport`



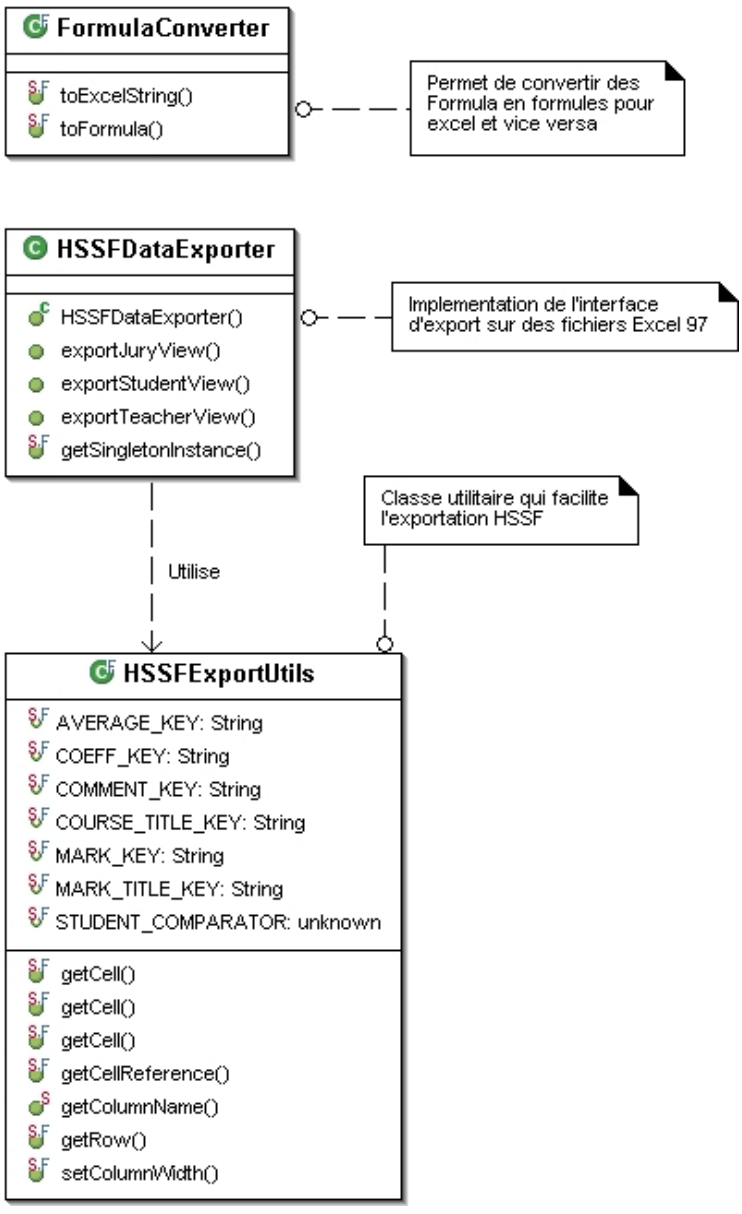
- L'interface `DataExporter`

Il s'agit d'une interface définissant les méthodes nécessaires à l'exportation d'une vue en un format (xml, excell). Il existe une méthode pour chaque vue, avec en paramètre le nom du fichier qui sera créé, un objet DataManager pour récupérer les données de la base de donnée, et pour certaines méthodes un objet en rapport avec celle-ci (Student, Course).

- La classe `DataExporterException :` `extends Exception`

Il s'agit d'une classe nous permettant de créer une exception qui se lèverait au cours d'une exportation (non respect de la dtd, problème avec la base de donnée, les formules...).

7. Le package `fr.umlv.symphonie.util.dataexport.hssf`

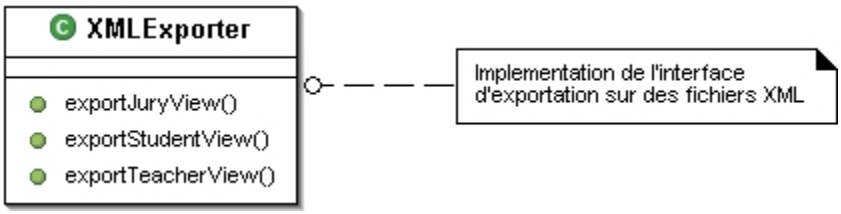


Le package fourni une implémentation de l'interface `DataExporter` qui exporte les vues du logiciel sous forme de fichiers XLS (Excel 97).

La classe `HSSDataExporter` est responsable de la création et l'export en fichiers xls. La mise en forme et toutes les opérations liées aux actions sur les feuilles de calcul sont déléguées à la classe utilitaire `HSSFExportUtils`.
La logique et le traitement des classes métier reposent entièrement sur la classe `HSSFDataExporter`.

Enfin la classe `FormulaConverter` est utilisée pour convertir les formules du format natif Symphonie vers le format Excel et vice versa.

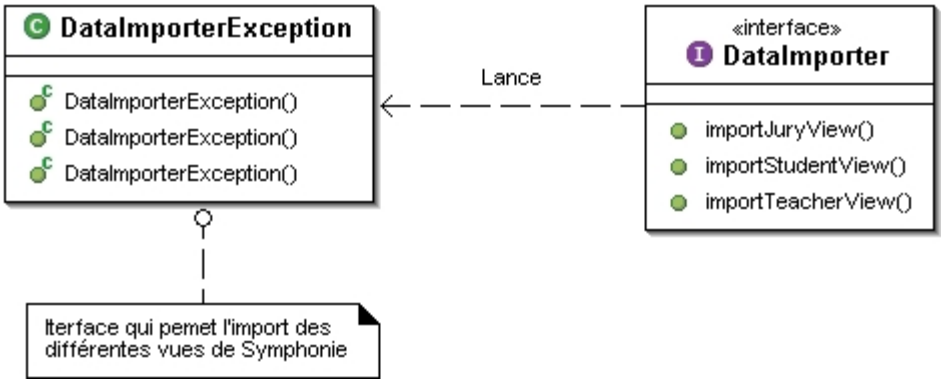
8. Le package `fr.umlv.symphonie.util.dataexport.xml`



- La classe `XMLExporter : implements DataExporter`

Il s'agit d'une classe qui exporte une vue au format xml. Elle possède en plus des méthodes de l'interface, des méthodes propres au format de l'exportation (`addCourseNode`, `addStudentNode`...). Elle utilise `org.w3c.dom` pour générer du xml valide en fonction d'une DTD, ainsi que de `javax.xml`. Les données sont récupérées à partir de la base de donnée (`DataManager`).

9. Le package `fr.umlv.symphonie.util.dataimport`



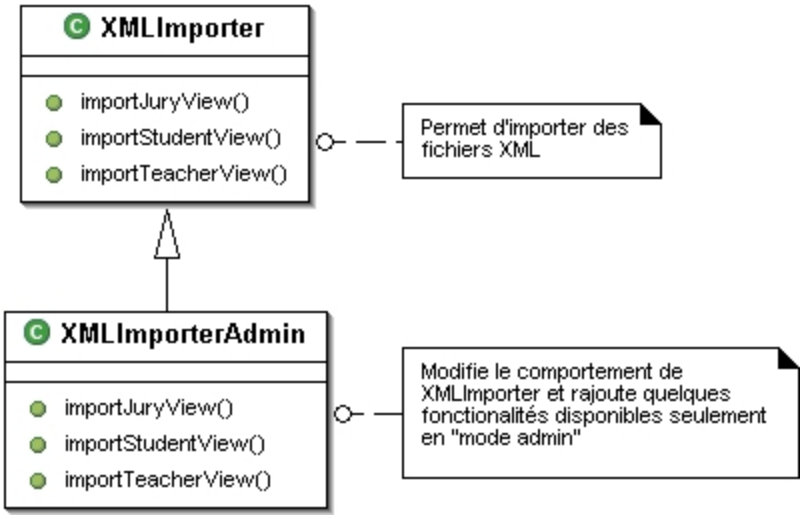
- L'interface `DataImporter`

Il s'agit d'une interface définissant les méthodes nécessaires à l'importation d'une vue en un format (xml, excell). Il existe une méthode pour chaque vue, avec en paramètre le nom du fichier qui sera créé, et un objet `DataManager` pour récupérer les données de la base de donnée.

- La classe `DataImporterException : extends Exception`

Il s'agit d'une classe nous permettant de créer une exception qui se lèverait au cours d'une exportation (non respect de la DTD, problème avec la base de donnée, les formules...).

10. Le package `fr.umlv.symphonie.util.dataimport.xml`



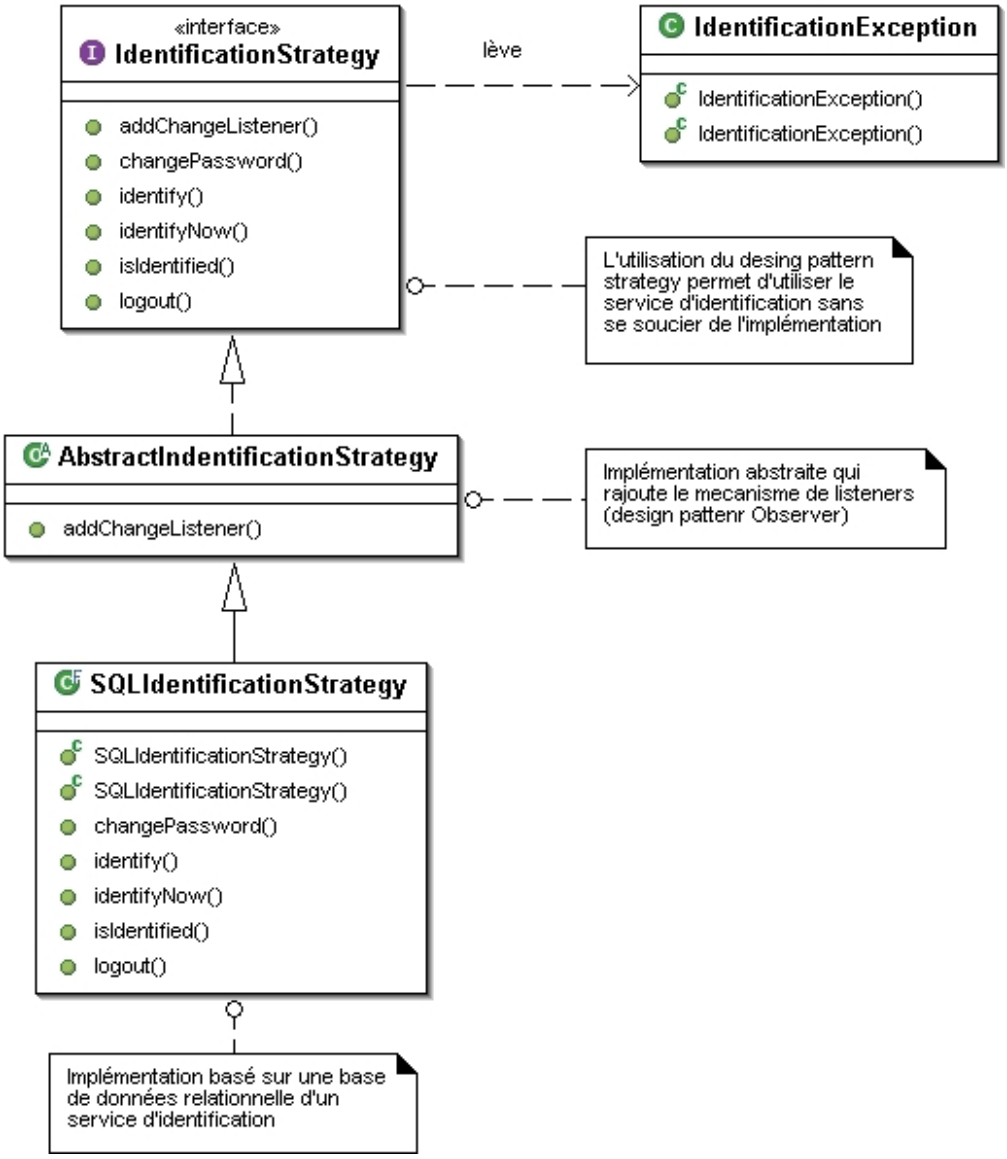
- La classe `XMLImporter : implements DataImporter`

Il s'agit d'une classe qui importe une vue au format xml. Elle possède en plus des méthodes de l'interface, des méthodes propres au format de l'importation (`getCourseNodes`, `getStudentNodes`...). Elle utilise org.w3c.dom pour vérifier la validité du fichier en fonction d'une dtd et pour manipuler les noeuds, ainsi que de `javax.xml` pour parser. Les données sont mises à partir de la base de donnée (`DataManager`) en fonction des droits d'un simple utilisateur (non administrateur).

- La classe `XMLImporterAdmin : extends XMLImporter`

Il s'agit d'une classe qui redéfinit les méthodes de l'interface, pour qu'elles puissent correspondre au mode administrateur. En effet, l'administrateur possède les droits que l'utilisateur n'a pas, comme pour ajouter, et aussi mettre à jour certaines valeurs. A partir de la base de donnée (DataManager), les données sont ajoutées si l'id du noeud a pour valeur –1, sinon elles sont mises à jour.

11. Le package `fr.umlv.symphonie.util.identification`



Ce package fournit le service d'identification du logiciel.

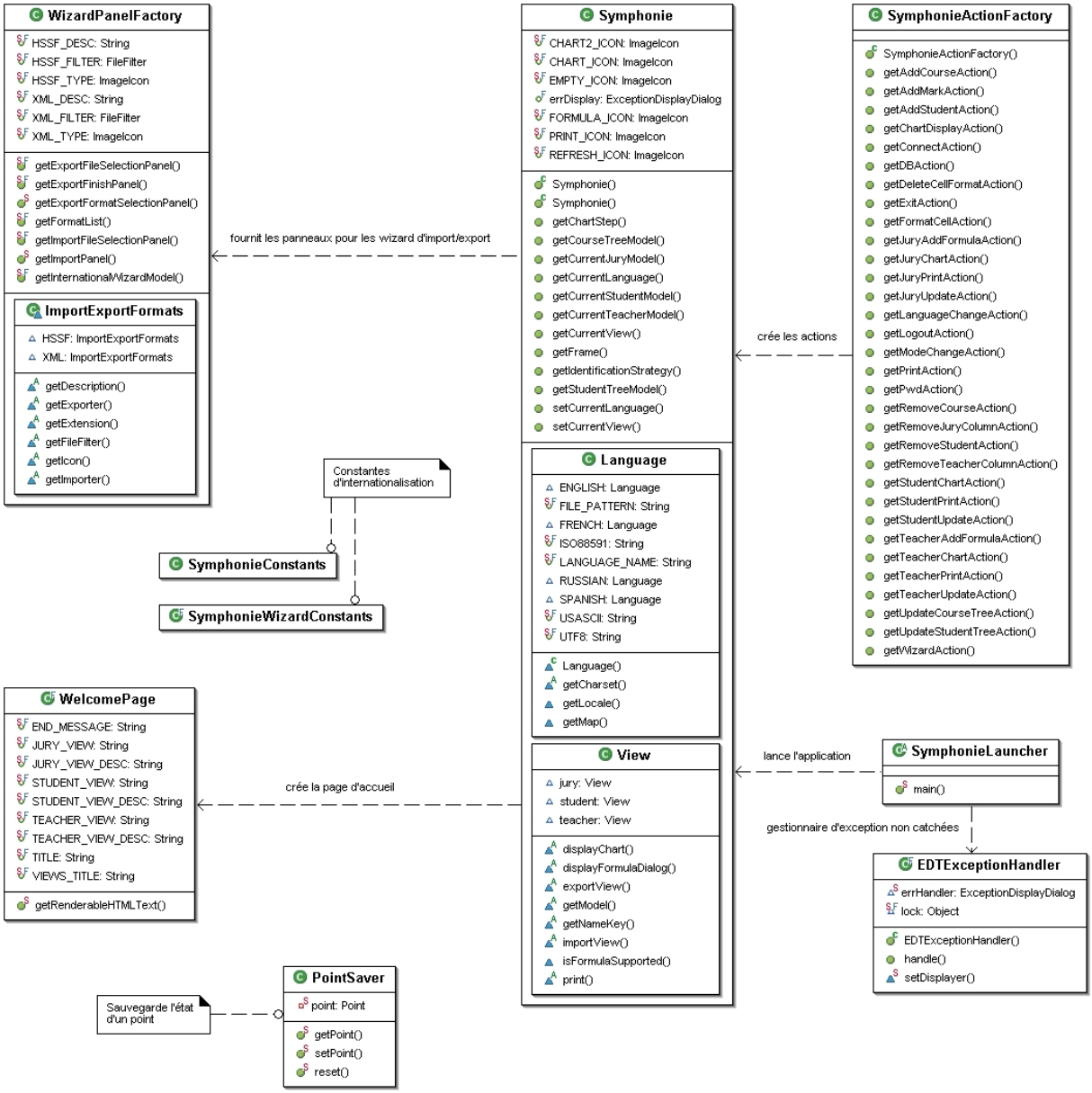
Le service d'identification utilise le design pattern `Strategy` couplé à un mécanisme d'`Observers` afin de permettre une abstraction totale de la logique d'identification et la notification des changements d'état du service.

`IdentificationStrategy` est l'interface principale du package. La classe `AbstractIdentificationStrategy` a été créée afin de faciliter l'implémentation de l'interface principale en ajoutant la logique des listeners et les notifications de changement d'état.

Symphonie utilise par défaut une instance de la `SQLIdentificationStrategy` pour gérer l'identification du super utilisateur.

La classe `IdentificationException` doit encapsuler toute exception susceptible de faire échouer l'identification quelque soit la logique sous-jacente.

12. Le package `fr.umlv.symphonie.view`



Ce package contient la vue principale du logiciel ainsi que son point d'entrée.

Symphonie a été conçu afin de permettre facilement son internationalisation. Ceci est réalisé grâce à la classe `ComponentBuilder`, vu plus loin, et les classes `SymphonieConstants` et `SymphonieWizardConstants`.

La classe `WelcomePage` permet de créer le code HTML, internationalisable, de la page d'accueil du logiciel.

La classe `WizardPanelFactory` est chargée de la création et la gestion des assistants d'import/export.
Elle contient une énumération interne qui représente les format supportés d'import/export, un format est responsable de fournir outre un nom et une description les objets qui permettent de faire l'import/export, cf. `fr.umlv.symphonie.util.dataexport` et `dataimport`.

La classe Symphonie constitue notre application. Elle n'est pas une `JFrame`, c'est une entité logique qui contient un ensemble d'états (States) représentés par la classe interne `View`. Le comportement de l'application est dépendant de chaque `View` et des données sous-jacentes à la vue.
La création et gestion des actions possibles sur le logiciel sont gérées par la classe `SymphonieActionFactory`.

L'énumération `Symphonie.Language` définit les langues supportées par le logiciel. Chaque langue est responsable de la gestion de ses propres ressources textuelles.

La classe `SymphonieLauncher` est le point d'entrée de l'application.
La classe `EDTExceptionHandler` s'occupe de la gestion des exceptions non catchées ou qui surviennent inopinément.

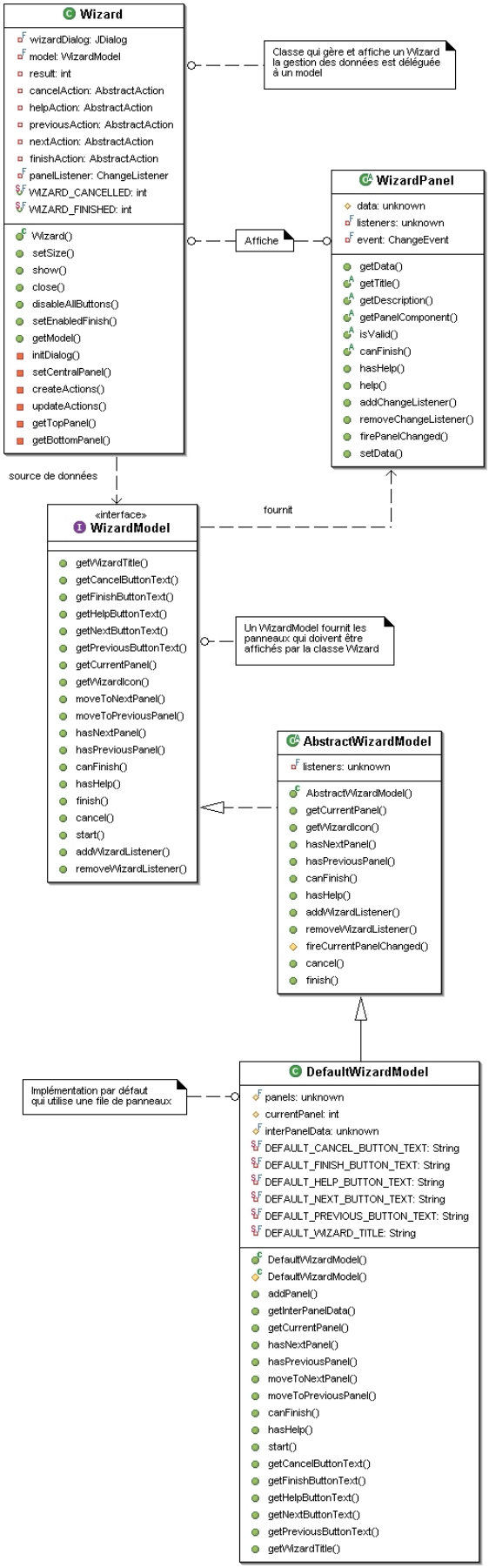
13. Le package `fr.umlv.symphonie.view.dialog`

Ce package contient toutes les boîtes de dialogue utilisées pour l'interaction avec l'utilisateur.

A un niveau génie logiciel ce package ne présente pas d'intérêt majeur, c'est pourquoi la description énumérative des classes nous a paru superflue.

La seule chose à savoir c'est que tous les dialogues ont une caractéristique commune : ils peuvent être affichés ou cachés.

14. Le package `fr.umlv.symphonie.wizard`



Le package fournit une implémentation MVC d'un assistant.

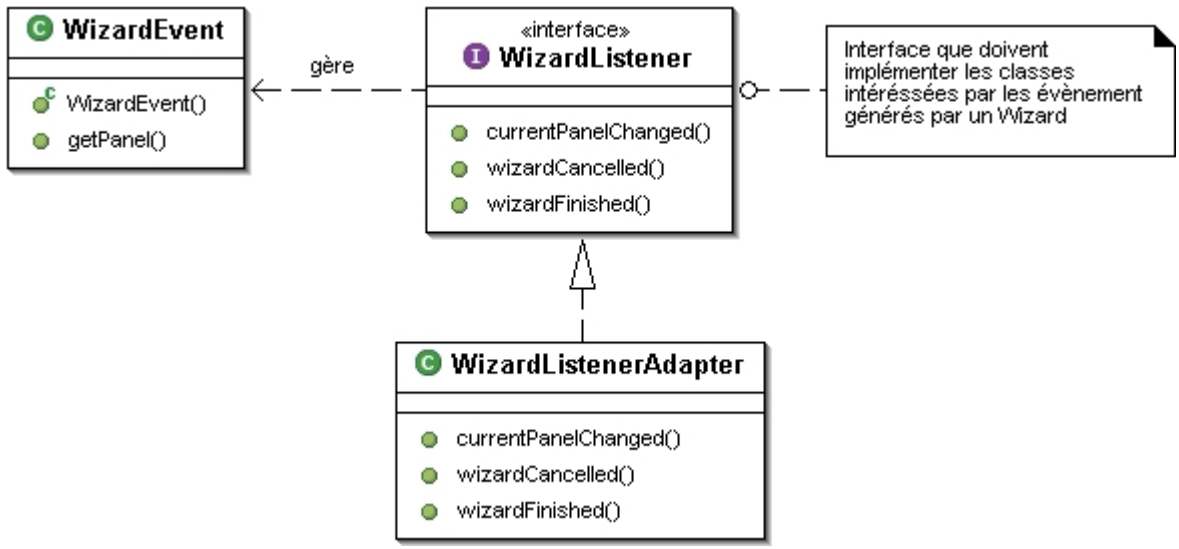
La classe `WizardModel` constitue la source des données d'un `Wizard`, en l'occurrence les données sont des `WizardPanel`.
Le comportement d'un Wizard est strictement dépendant du son modèle de données sous-jacent.

Il existe deux implémentations d'un `WizardModel` : `AbstractWizardModel` (avec listeners) et `DefaultWizardModel` (first panel in, first panel shown).

Un panel affichable dans un `Wizard` doit hériter de la classe abstraite `WizardPanel`, le comportement d'un panel est strictement dépendant de l'implémentation des méthodes abstraites de `WizardPanel`.

Les classes intéressées par les évènements générés par un `Wizard` doivent implémenter l'interface `WizardListener` du package `fr.umlv.symphonie.util.wizard.event` décrit ci-dessous.

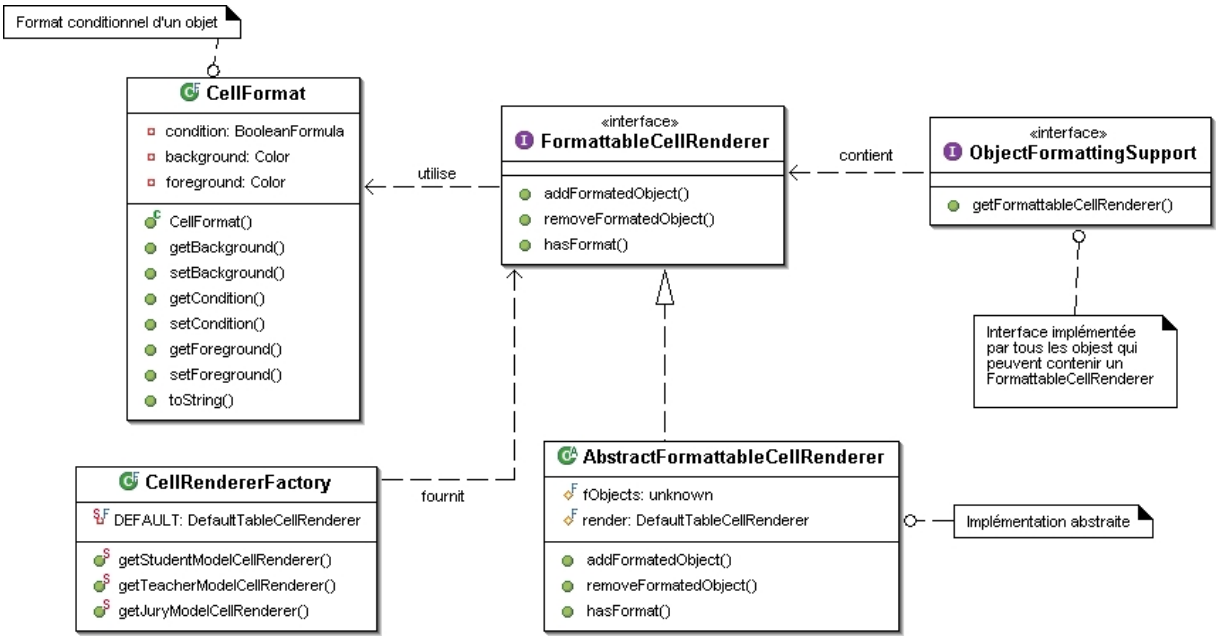
15. Le package `fr.umlv.symphonie.wizard.event`



Un `WizardListener` est un listener capable de gérer les évènements générés par le `Wizard` et représentés par la classe `WizardEvent`.

La classe `WizardListenerAdapter`, qui est une implémentation "vide" d'un listener existe pour le confort du programmeur qui désire créer un listener qui ne gère que certains évènements parmi les trois existants :
- le panel a changé
- l'assistant s'est fini
- l'assistant a été annulé.

16. Le package `fr.umlv.symphonie.view.cells`



Package responsable du formatage des cellules à l'intérieur des vues de Symphonie.

Tout d'abord il existe l'interface `ObjectFormattingSupport` qui doit être implantée par toutes les classes et/ou interfaces qui peuvent posséder un renderer qui leur est propre.

L'interface `FormattableCellRenderer` définit l'interface d'un `ListCellRenderer` qui peut gérer un ensemble d'objets qui seraient traités de façon spécifique selon un formatage conditionnel. La classe `AbstractFormattableCellRenderer` est une implémentation abstraite de cette interface.

Le format à proprement parler est donné par la classe `CellFormat`. Elle contient toutes les données nécessaires au rendu de l'objet.

La classe `CellRendererFactory` est responsable de la création des gestionnaires de rendu spécifiques à chaque vue de Symphonie.

IV. HOW-TO

Etant donné que notre logiciel ne permet pas de charger de fonctions au runtime au démarrage via un répertoire plug-in ou autres, le seul moyen de rajouter une fonction c'est de le rajouter dans le code existant.

Pour ce faire il faut créer une classe qui implémente l'interface `FormulaFunction`.
Ce sont typiquement des classes sans état avec une méthode qui calcule une valeur en fonction des paramètres fournis, et associés à un mot clé.
Il faut aussi ajouter le chemin de la classe dans le system classpath afin de pouvoir la charger.

Ensuite on peut rajouter la fonction grâce à la méthode statique `addFunction` de la classe `SymphonieFormulaFactory`.

Il est très important d'ajouter la fonction AVANT, de préférence dans le constructeur de Symphonie, le parsing des formules sous peine de se retrouver avec des `NullPointerException` à l'évaluation.

Il faut noter aussi que la portabilité Excel est directement liée à l'existence du mot-clé auxquels la fonction est associée.

Notre logiciel ne gère pas les mots-clés. La grammaire des formules a été prévue pour mais l'implémentation n'a pas été faite. L'usage des mots-clés, y compris `$data`, génère une `UnsupportedOperationException` à l'exécution.

V. Bugs connus

1. Import/Export lorsque le logiciel est en russe.

Plusieurs exceptions peuvent survenir lors de cette opération. Ceci est dû à l'incompatibilité des jeux de caractères aussi bien des fichiers XLS que celui défini par notre DTD : UTF8, ISO-8859-1. Il n'est pas exclue que d'autres exceptions surviennent lors des opération comme le parsing des formules. Nous avons constaté la non-portabilité des Lexers sablecc sur certains OS, notamment ceux générés sous linux ne marchent pas toujours sous windows.

2. Enable/Disable des actions qui dépendent des vues Student et Teacher du mode administration.

Les boutons de la barre d'outils ajouter/supprimer un étudiant dans la vue `Student` et ajouter/supprimer matière dans la vue `Teacher` ne sont pas activés que lorsqu'on sélectionne pour la deuxième fois un élément dans l'arbre de sélection.
Ceci arrive également lors de la désélection (ou la sélection de la racine de l'arbre), il faut cliquer deux fois pour que le bouton soit désactivé. Nous ne savons pas pourquoi cela arrive, nous avons pu seulement constaté que le listener de sélection pour un évènement n'est appelé que lorsqu'un nouvel évènement arrive.

3. Formatage conditionnel multiple involontaire dans la vue Jury

Supposons que deux étudiants ont une moyenne de 7.5 dans deux matières différentes. Ensuite je formate la cellule de la moyenne du premier étudiant. La moyenne du deuxième étudiant se trouve formatée aussi.

Pour expliquer ce bug il faut savoir deux choses :

- La moyenne d'un étudiant dans une matière n'est pas stockée dans la base de données mais calculée à l'exécution, ni représentée par une classe métier.
- Les objets formatés sont stockés dans une HashMap.

Dans une `HashMap` deux objets `Float` qui valent 7.5 sont égaux même si leur référence n'est pas la même, c'est pourquoi au rendu tous ces objets-là sont formatés. L'utilisation d'une `IdentityHashMap` ne résout pas le problème car il suffit de changer une note pour que la moyenne soit recalculé et que la référence vers l'objet change, d'autant plus que les `Float` sont des objets non mutables.

4. Clique droit dans le "no mans land" des arbres de sélection.

Un clic droit dans le petit espace existant entre deux feuilles de l'arbre génère un `IndexOutOfBoundsException`.

Ceci est dû au fait qu'aucune feuille de l'arbre ne correspond à l'espace dans lequel on a cliqué. Lorsqu'on tente de retrouver la ligne qui correspond au point le composant renvoie -1, cet indice est utilisé pour forcer la sélection dans l'arbre d'où l'exception.

VI. Conclusion

Symphonie nous a permis de travailler en équipe, de mettre en œuvre à la fois la théorie des designs pattern, mais aussi les notions de gestion de projet et de temps.
Ce projet nous aura aussi permis de chercher et connaître des API qui seront utiles pour d’autres réalisations.