



# **Rapport projet Java 2016 : LocAppli**



**Timothée Barbot, Stéphane Gâteau, Adrien Poupa**

**Efrei Promotion 2018**

**L'3 Groupe C**

## Sommaire

Introduction.....	page 3
Répartition des tâches.....	page 4
Mode d'emploi.....	page 5
Structure utilisée.....	page 11
Explication de code.....	page 24
Conclusion.....	page 27
Bibliographie.....	page 28

## Introduction

Peu après l'annonce du projet Java et la formation des équipes, nous voulions composer un projet qui soit le plus « innovant » possible. Pour ce faire, M. Lahlou nous a conseillé d'utiliser une couche d'abstraction plus haute que celle de JDBC traditionnellement utilisée pour manipuler des bases de données SQL ou Oracle en Java, **JPA**.

Chronologiquement, nous avons commencé la répartition des tâches : Timothée allait s'occuper de la persistance des données via JPA et un ORM qu'il fallait déterminer, Stéphane commençait l'apparence de l'application avec l'API Swing alors que Adrien s'occupait de l'implémentation du graphe donné dans le sujet (classes Vehicule, Auto, Moto, Exemple, containers, etc).

Pour le partage du code, nous avons mis en place un repository sur GitHub que vous pourrez retrouver ici : <https://github.com/AdrienPoupa/location> afin de voir l'évolution du code. Ceci nous a permis de travailler simultanément sur le projet voire même sur le même fichier et le même code sans annuler nos modifications respectives grâce à la gestion de conflits intégrée à Git.

Nous avons fait le choix de travailler sur IntelliJ plutôt que Eclipse ou NetBeans, en raison de la licence étudiante gratuite que nous accorde notre adresse mail Efrei et de la modernité de l'éditeur.

Une fois la base des classes mise en place et le partage du code fonctionnel, nous avons commencé le développement des fonctions spécifiques : mise en place du moteur de production Gradle et des dépendances sur Maven Central pour ajouter l'ORM eBean afin de stocker les données dans une base de données SQLite sans avoir à écrire une seule requête.

La mise en place de Gradle a ensuite permis de rajouter les dépendances requises (iTextPDF et JFreeChart) afin de travailler sur la génération des devis et factures en PDF ainsi que l'affichage des histogrammes. Enfin, une fois le code finalisé en backend, il a été possible de passer le fichier monolithique de tests en console, datant du premier jour de développement, en tests unitaires JUnit, dont la dépendance a été rajoutée à Gradle.

Dernière étape, la finalisation de l'interface graphique. Nous avons déjà l'interface « statique », sorte de coquille vide où aucune action n'était possible. La dernière phase du développement a consisté à animer cette dernière en ajoutant les actions aux boutons, ainsi qu'une grosse refactorisation du code.

## Répartition des tâches

Tâche	Timothée Barbot	Stéphane Gâteau	Adrien Poupa
Classes backend (Exemplaire, Vehicule...)	✓		✓
Containers	✓		✓
Persistence des données - JPA	✓		
Gradle, eBean	✓		
Génération PDF			✓
Structure du projet	✓		
Tests unitaires			✓
Onglets		✓	
Panneaux	✓	✓	
Tableaux			✓
Formulaires	✓	✓	
Fenêtres de dialogue	✓	✓	
Histogrammes			✓
Javadoc			✓
Rapport			✓

## Mode d'emploi

Au premier lancement de l'application la base de données est pré remplie, la fenêtre suivante s'affiche :

Numéro	Locataire	Date début	Date fin	Devis accepté	Prix TTC	Actions
1	Poupa Adrien	1 Janvier 20...	1 Février 2016	Oui	210.0	Détails
2	Poupa Adrien	7 Octobre 19...	6 Octobre 19...	Non	130.0	Détails
3	Rackam Jack	7 Avril 2016	19 Avril 2016	Non	200.0	Détails

On peut voir les détails ou rechercher une location. L'ajout d'une location se fait comme suit :

**Emprunteur :** Poupa Adrien

Type de location : ☒ Auto ☐ Moto

Choix du véhicule : Dacia - Sandero

Date de départ : 11-04-2016 Date de retour : 11-04-2016

☐ Assurance

**Rechercher disponibilité**

**Véhicules disponibles**

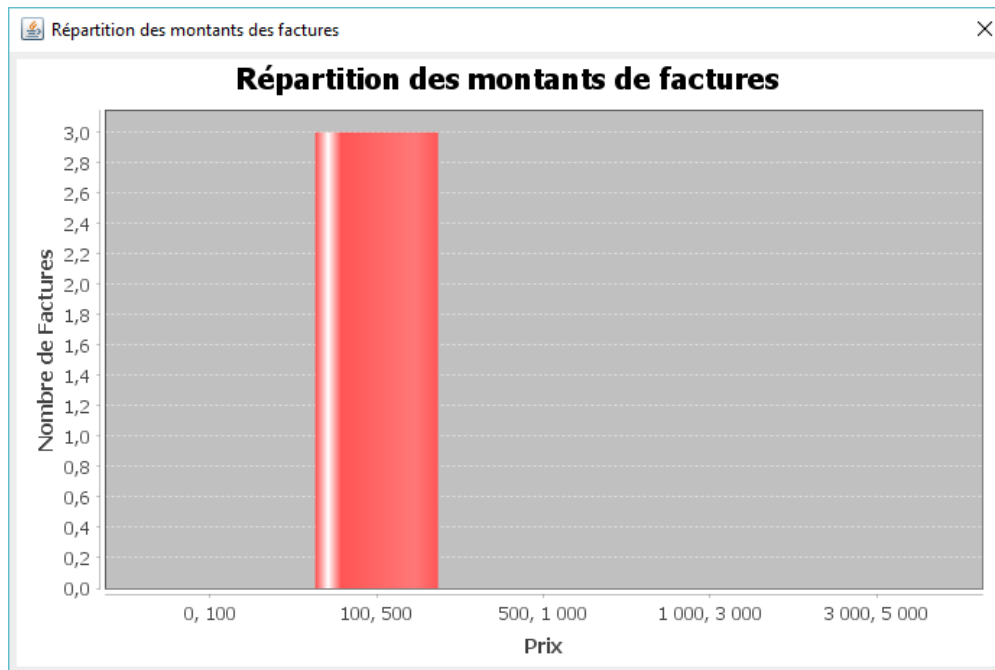
Numéro	Marque	Modèle/Cylin...	Kilométrage	Réservoir	Etat	Actions
1	Dacia	Sandero	49195	Plein	OK	Ajouter
2	Dacia	Sandero	23309	Plein	OK	Ajouter
3	Dacia	Sandero	10364	Plein	OK	Ajouter
136	Dacia	Sandero	60000	Plein	OK	Ajouter

**Véhicules loués**

Numéro	Marque	Modèle/Cylin...	Kilométrage	Réservoir	Etat	Actions
1	Dacia	Sandero	49195	Plein	OK	Suppri...
136	Dacia	Sandero	60000	Plein	OK	Suppri...

Annuler Valider

Le choix d'un emprunteur se fait dans la base de données, puis on peut choisir entre Auto et Moto, le menu déroulant du choix de véhicule se met à jour. On clique sur « Recherche disponibilité » pour voir les exemplaires disponibles, puis « Ajouter » pour mettre à jour le second tableau.



*Histogramme de la répartition des montants des factures*

L'onglet Emprunteurs s'affiche comme suit :

LocAppli

Locations Emprunteurs Exemplaïres Vehicules A propos

### Liste des emprunteurs

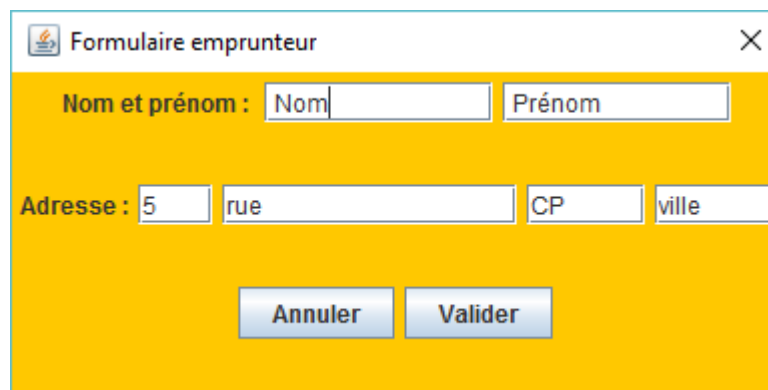
Ajouter un emprunteur

Recherche :

Numéro	Nom	Prénom	Adresse	Actions
1	Poupa	Adrien	3 avenue de la Ré...	Détails
2	Barbot	Timothée	5 avenue de la Ré...	Détails
3	Gâteau	Stéphane	15 avenue de la R...	Détails
4	Rackam	Jack	25 avenue de l'Esp...	Détails
5	Taer	Jean	2 rue des Trésorie...	Détails
6	Taer	Jeanne	2 rue des Trésorie...	Détails
7	Simon	Léna	45 rue des Filatier...	Détails
8	Dupont	David	4 rue des Chalets ...	Détails
9	Mandanda	Steve	4 Grand-Rue 1300...	Détails
10	Lee	Harvey	276 Promenade d...	Détails
11	Kevin	Harvey	276 Promenade d...	Détails
12	Jena	Harvey	276 Promenade d...	Détails
13	Snow	John	1 The Wall 56000 ...	Détails
14	Tarly	Samwell	1 The Wall 56000 ...	Détails
15	Targaryen	Daenerys	1 Palace 90000 M...	Détails
16	Mormont	Jorah	34 Friendzone ave...	Détails
17	Baratheon	Tommen	1 Palace 10000 Ki...	Détails
18	Lannister	Cersei	1 Palace 10000 Ki...	Détails
19	Lannister	Jaime	1 Palace 10000 Ki...	Détails
20	Stark	Ned	1 Palace 10000 Ki...	Détails
21	Poupa	Adrien	3 avenue de la Ré...	Détails

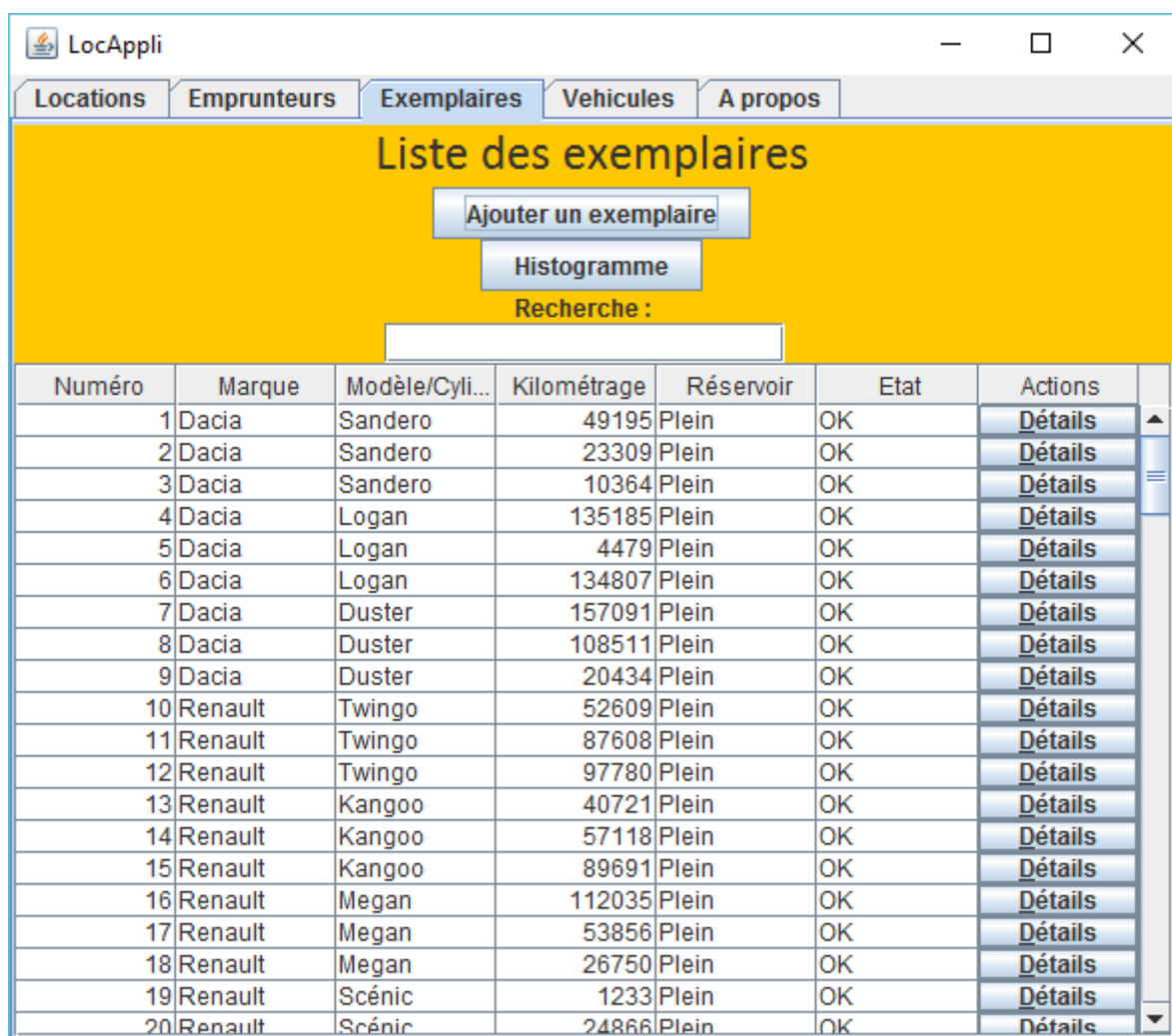
Comme dans tous les tableaux, il est possible de trier par numéro, nom, prénom ou de faire une recherche. Le tableau se met alors à jour dynamiquement.

L'ajout d'un emprunteur donne la fenêtre suivante :



The image shows a Java Swing window titled "Formulaire emprunteur". It has a yellow background. At the top, there's a label "Nom et prénom :" followed by two text input fields, "Nom" and "Prénom". Below this, there's a label "Adresse :" followed by four text input fields: "5", "rue", "CP", and "ville". At the bottom, there are two buttons: "Annuler" and "Valider".

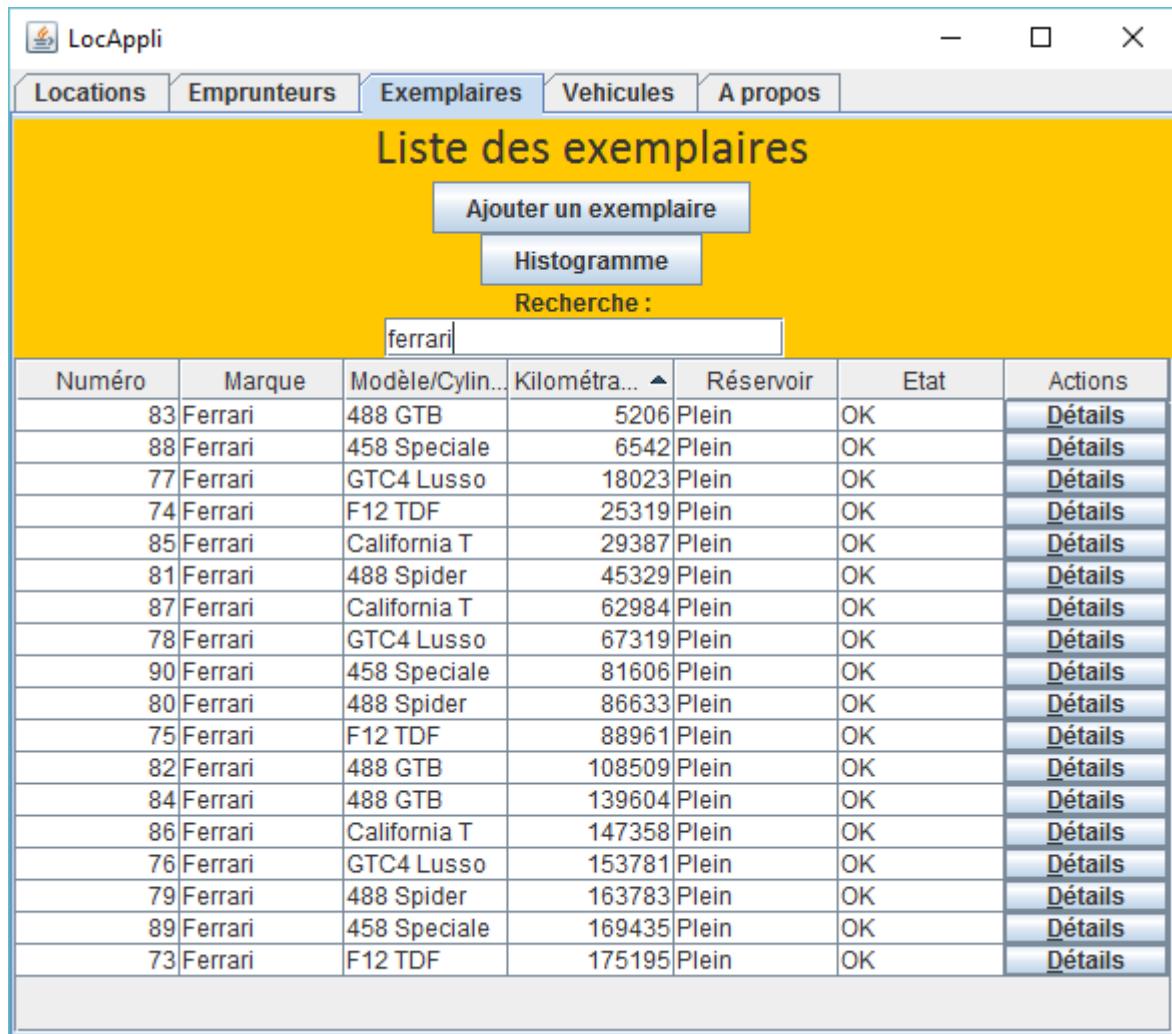
La liste des Exemples s'affiche toujours sous la forme d'un tableau :



The image shows a Java Swing application window titled "LocAppli". It has a yellow background. At the top, there's a menu bar with five items: "Locations", "Emprunteurs", "Exemplaires", "Vehicules", and "A propos". Below the menu bar, there's a title "Liste des exemplaires". Under the title, there are three buttons: "Ajouter un exemplaire", "Histogramme", and "Recherche :". Below the "Recherche :" button, there's a text input field. Below the input field, there's a table with 7 columns: "Numéro", "Marque", "Modèle/Cyli...", "Kilométrage", "Réservoir", "Etat", and "Actions". The table contains 20 rows of data. Each row has a "Détails" button in the "Actions" column.

Numéro	Marque	Modèle/Cyli...	Kilométrage	Réservoir	Etat	Actions
1	Dacia	Sandero	49195	Plein	OK	Détails
2	Dacia	Sandero	23309	Plein	OK	Détails
3	Dacia	Sandero	10364	Plein	OK	Détails
4	Dacia	Logan	135185	Plein	OK	Détails
5	Dacia	Logan	4479	Plein	OK	Détails
6	Dacia	Logan	134807	Plein	OK	Détails
7	Dacia	Duster	157091	Plein	OK	Détails
8	Dacia	Duster	108511	Plein	OK	Détails
9	Dacia	Duster	20434	Plein	OK	Détails
10	Renault	Twingo	52609	Plein	OK	Détails
11	Renault	Twingo	87608	Plein	OK	Détails
12	Renault	Twingo	97780	Plein	OK	Détails
13	Renault	Kangoo	40721	Plein	OK	Détails
14	Renault	Kangoo	57118	Plein	OK	Détails
15	Renault	Kangoo	89691	Plein	OK	Détails
16	Renault	Megan	112035	Plein	OK	Détails
17	Renault	Megan	53856	Plein	OK	Détails
18	Renault	Megan	26750	Plein	OK	Détails
19	Renault	Scénic	1233	Plein	OK	Détails
20	Renault	Scénic	24866	Plein	OK	Détails

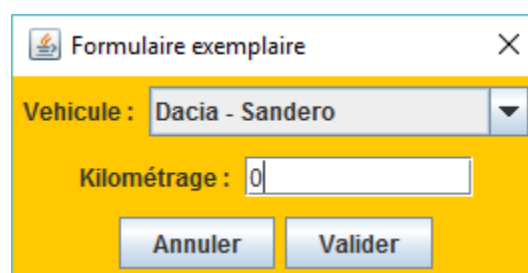
Là encore, tout est triable/recherchable/modifiable :



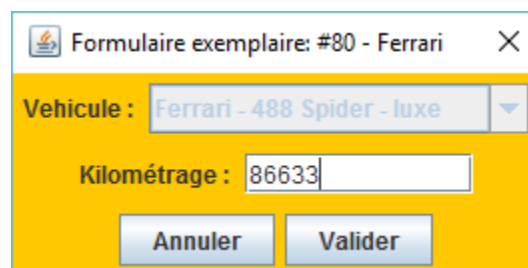
Numéro	Marque	Modèle/Cylin...	Kilométra...	Réservoir	Etat	Actions
83	Ferrari	488 GTB	5206	Plein	OK	Détails
88	Ferrari	458 Speciale	6542	Plein	OK	Détails
77	Ferrari	GTC4 Lusso	18023	Plein	OK	Détails
74	Ferrari	F12 TDF	25319	Plein	OK	Détails
85	Ferrari	California T	29387	Plein	OK	Détails
81	Ferrari	488 Spider	45329	Plein	OK	Détails
87	Ferrari	California T	62984	Plein	OK	Détails
78	Ferrari	GTC4 Lusso	67319	Plein	OK	Détails
90	Ferrari	458 Speciale	81606	Plein	OK	Détails
80	Ferrari	488 Spider	86633	Plein	OK	Détails
75	Ferrari	F12 TDF	88961	Plein	OK	Détails
82	Ferrari	488 GTB	108509	Plein	OK	Détails
84	Ferrari	488 GTB	139604	Plein	OK	Détails
86	Ferrari	California T	147358	Plein	OK	Détails
76	Ferrari	GTC4 Lusso	153781	Plein	OK	Détails
79	Ferrari	488 Spider	163783	Plein	OK	Détails
89	Ferrari	458 Speciale	169435	Plein	OK	Détails
73	Ferrari	F12 TDF	175195	Plein	OK	Détails

*Exemple de tri : Ferrari et kilométrage décroissant*

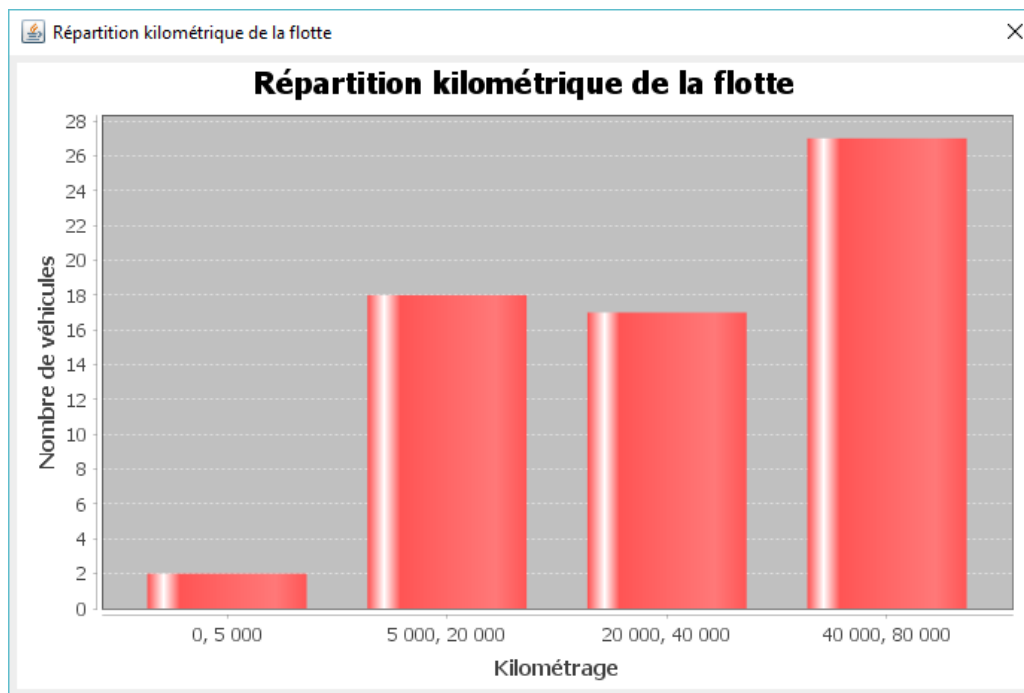
L'ajout d'un exemplaire propose son modèle ainsi que son kilométrage :



L'édition est similaire, avec des champs pré-remplis :







*Histogramme de la répartition kilométrique de la flotte*

La vue des véhicules est encore une fois similaire à la vue des exemplaires et des emprunteurs, avec les mêmes possibilités pour le tableau :

LocAppli

Locations Emprunteurs Exemplaires **Véhicules** A propos

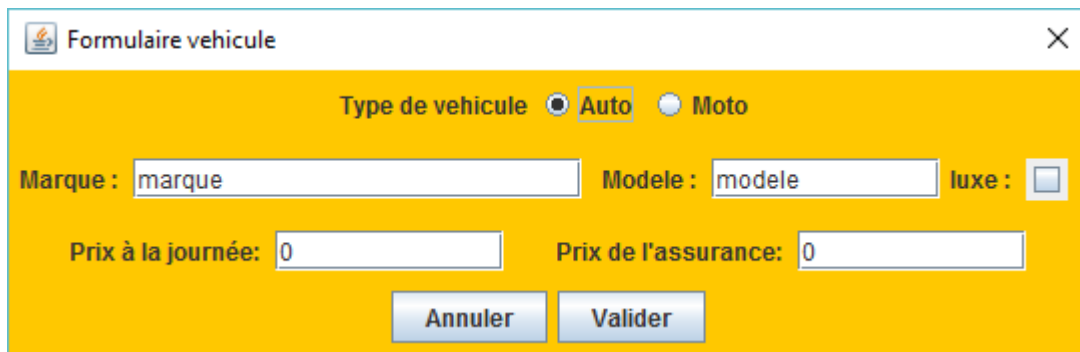
### Liste des véhicules

Ajouter un véhicule

Recherche :

Numéro	Marque	Modèle/Cylindr...	Prix journalier	Prix assurance	Actions
1	Dacia	Sandero	100	30	Détails
2	Dacia	Logan	70	30	Détails
3	Dacia	Duster	150	50	Détails
4	Renault	Twingo	120	40	Détails
5	Renault	Kangoo	120	40	Détails
6	Renault	Megan	120	40	Détails
7	Renault	Scénic	150	50	Détails
8	Renault	Espace	200	60	Détails
9	Renault	Zoe	110	40	Détails
10	Renault	Captur	140	50	Détails
11	Citroën	C1	110	20	Détails
12	Citroën	C2	130	30	Détails
13	Citroën	C3	150	40	Détails
14	Citroën	C4	160	50	Détails
15	Citroën	C4 Cactus	170	50	Détails
16	Mercedes	SLR (luxe)	300	100	Détails
17	Mercedes	SLK (luxe)	400	100	Détails
18	Mercedes	Classe E (luxe)	300	100	Détails
19	Mercedes	Classe B (luxe)	300	100	Détails
20	Mercedes	Classe C (luxe)	250	100	Détails
21	Mercedes	Classe A (luxe)	200	100	Détails

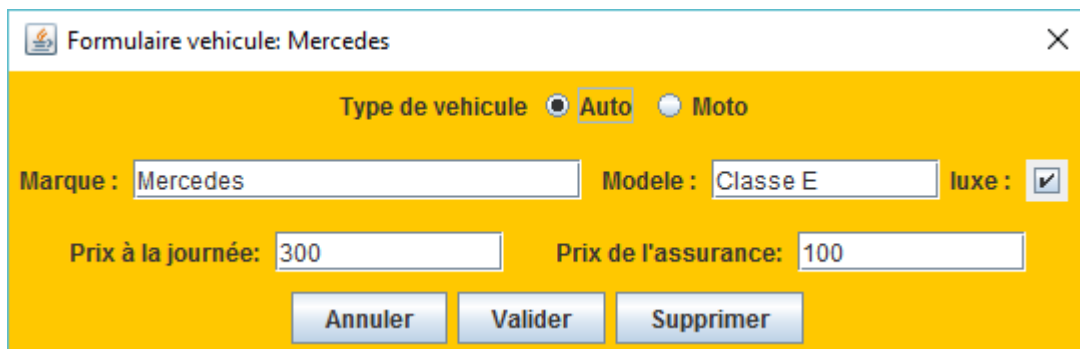
Ajout d'un véhicule :



The screenshot shows a Java Swing dialog box titled "Formulaire vehicule". It has a yellow background and a close button (X) in the top right corner. The dialog contains the following elements:

- Type de vehicule:** Two radio buttons, "Auto" (which is selected) and "Moto".
- Marque:** A text input field containing the placeholder text "marque".
- Modele:** A text input field containing the placeholder text "modele".
- lux:** A checkbox that is currently unchecked.
- Prix à la journée:** A text input field containing the value "0".
- Prix de l'assurance:** A text input field containing the value "0".
- Buttons:** Two buttons at the bottom, "Annuler" and "Valider".

Modification d'un véhicule :



The screenshot shows a Java Swing dialog box titled "Formulaire vehicule: Mercedes". It has a yellow background and a close button (X) in the top right corner. The dialog contains the following elements:

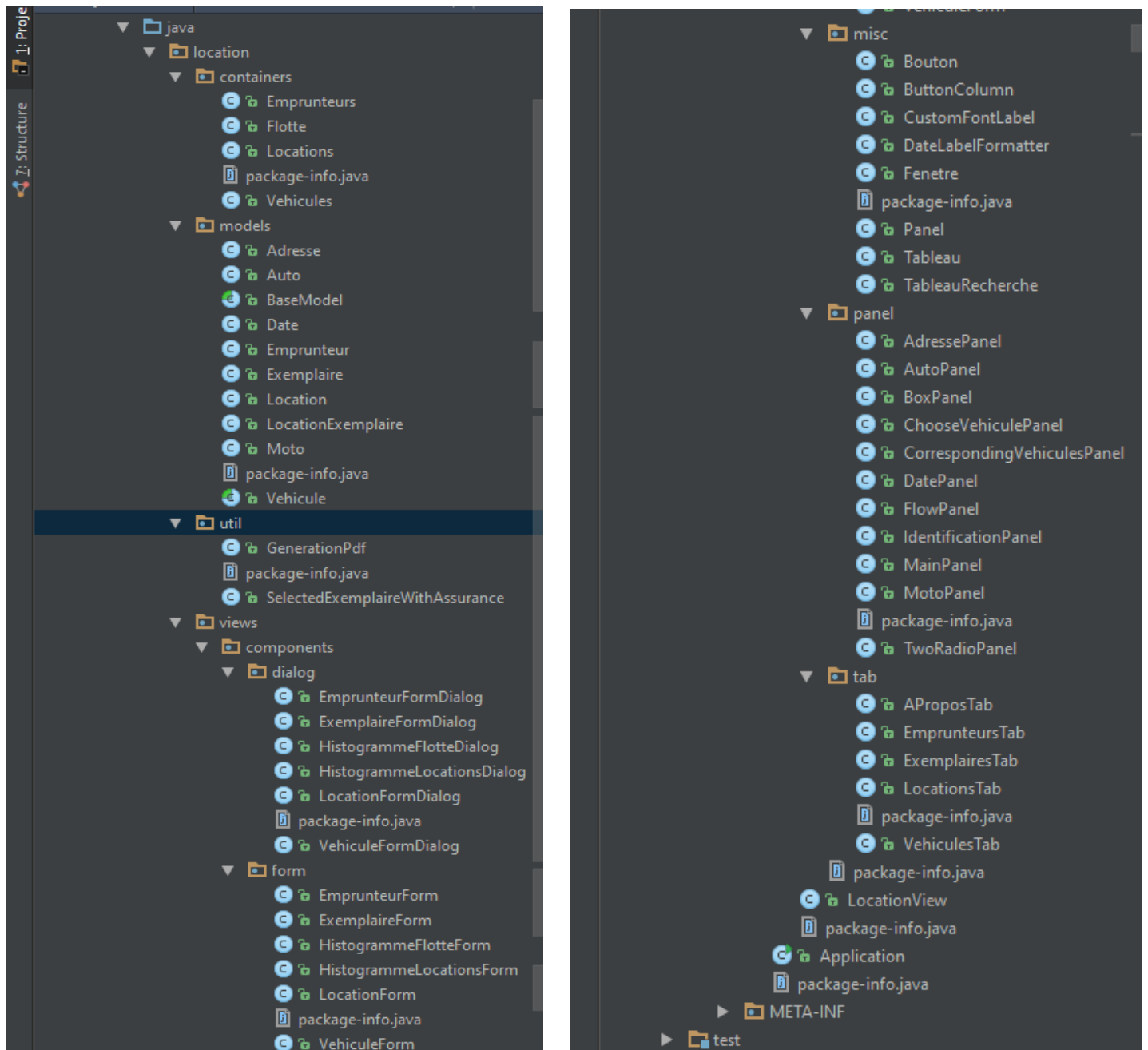
- Type de vehicule:** Two radio buttons, "Auto" (which is selected) and "Moto".
- Marque:** A text input field containing the value "Mercedes".
- Modele:** A text input field containing the value "Classe E".
- lux:** A checkbox that is currently checked.
- Prix à la journée:** A text input field containing the value "300".
- Prix de l'assurance:** A text input field containing the value "100".
- Buttons:** Three buttons at the bottom, "Annuler", "Valider", and "Supprimer".

## Structure utilisée

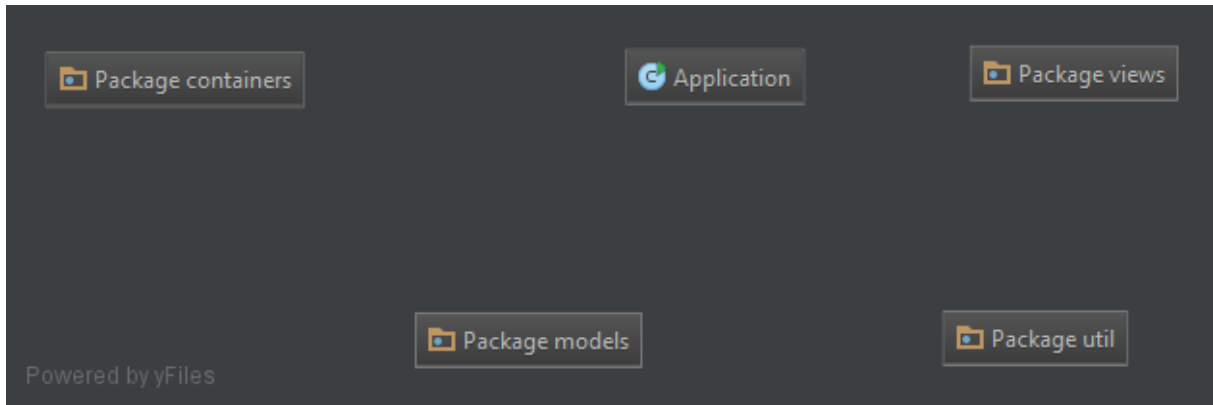
Pour des raisons de lisibilité, nous avons choisi de présenter des diagrammes au niveau de chaque package, en faisant figurer les attributs de chaque classe mais pas les méthodes, sans quoi les diagrammes devenaient illisibles du fait de leur taille.

Nous avons choisi une structure regroupant plusieurs packages dans le package principal (location).

L'arborescence est la suivante :



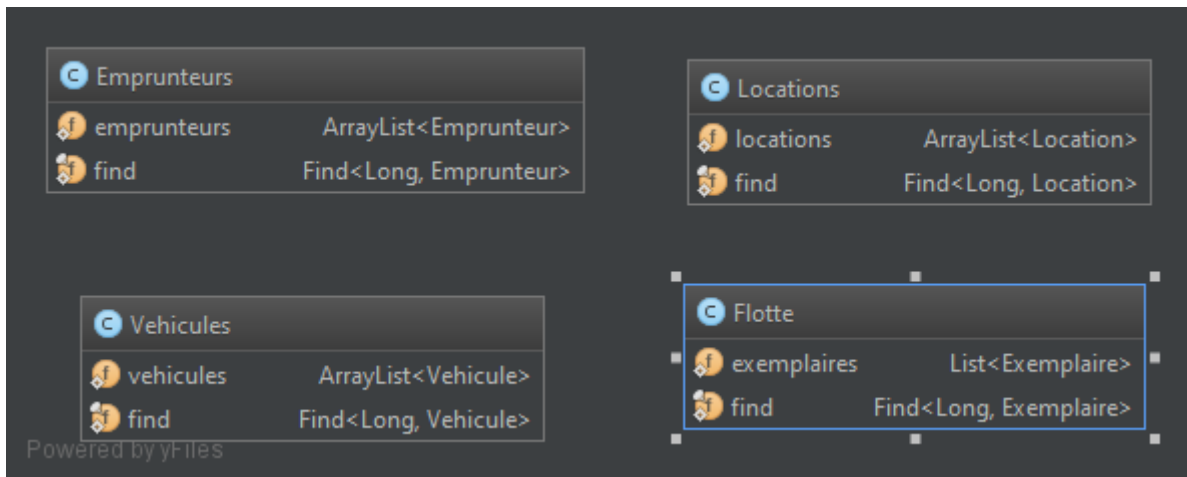
Arborescence du projet



### *Package principale Location*

Ce package comprend donc la classe principale appelée au démarrage du programme. Le reste des packages suit une structure en MVC (modèle vue contrôleur), ainsi qu'un package util supplémentaire pour la génération des fichiers PDF.

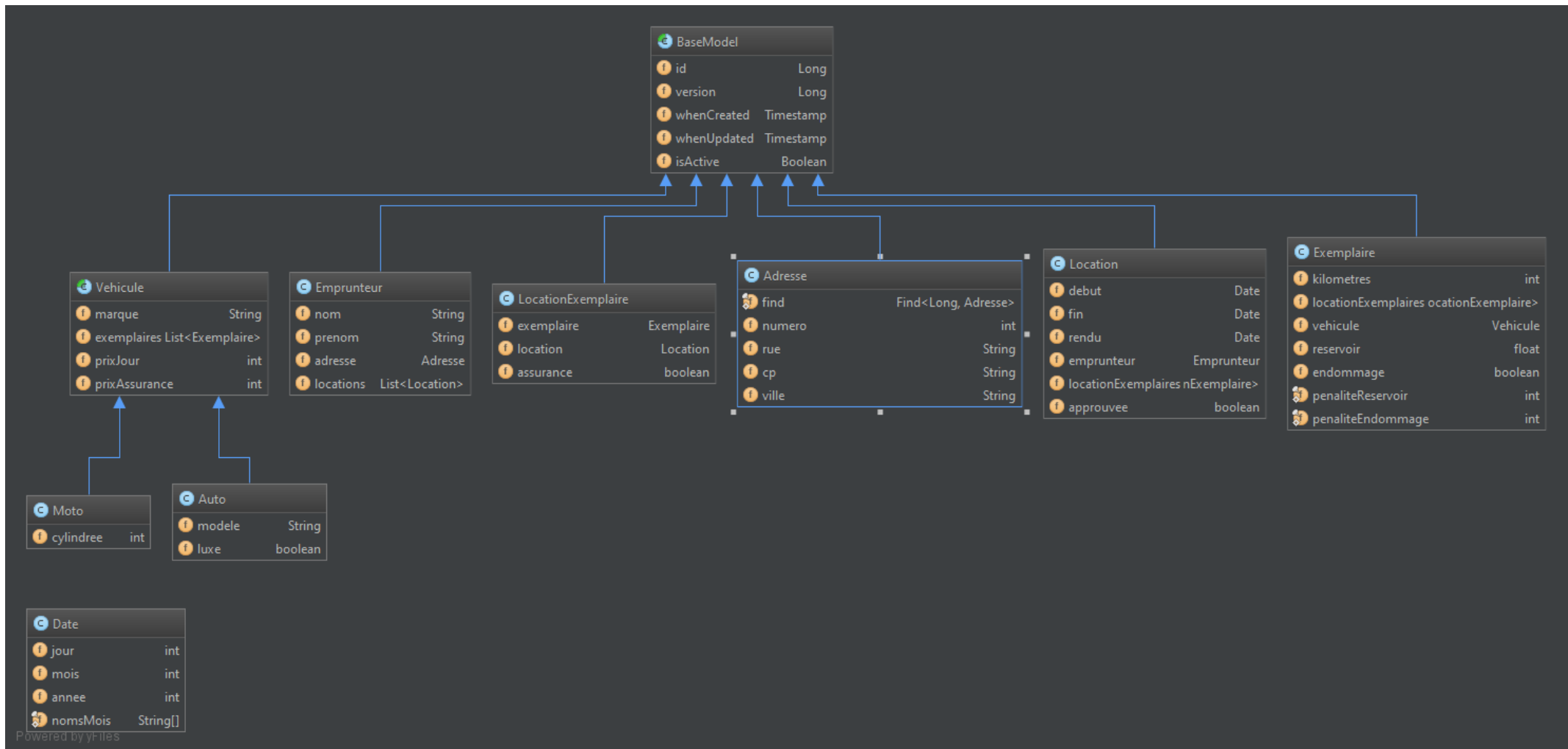
Regardons désormais package par package.



### *Package Containers*

Chaque classe du package containers contient des éléments de son nom au singulier, à l'exception de la Flotte.

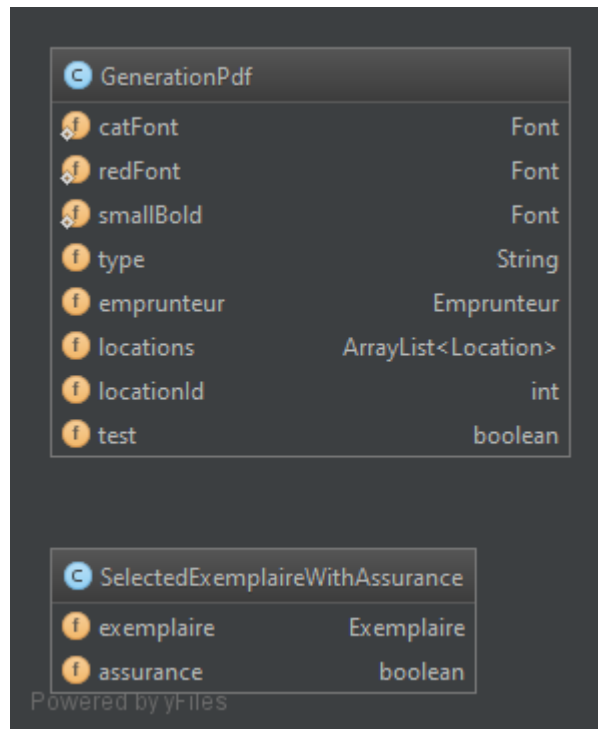
- Emprunteurs contient une liste d'instances d'Emprunteur
- Vehicules contient une liste d'instances de Vehicule
- Locations contient une liste d'instances de Location
- Flotte contient une liste d'instances d'Exemplaire



### Package Models

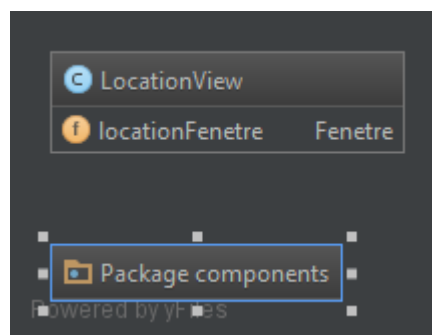
Le package models constitue le cœur du back-end de l'application. C'est lui qui gère toutes les locations, tous les véhicules, tous les exemples, tous les emprunteurs. Toutes les classes héritent de BaseModel pour le fonctionnement de JPA (retour d'un ID, sa version, dates de modification, objet actif) à l'exception de Date qui est intégrée dans les autres classes.

LocationExempleire sert de classe de liaison entre une location et potentiellement plusieurs exemples.



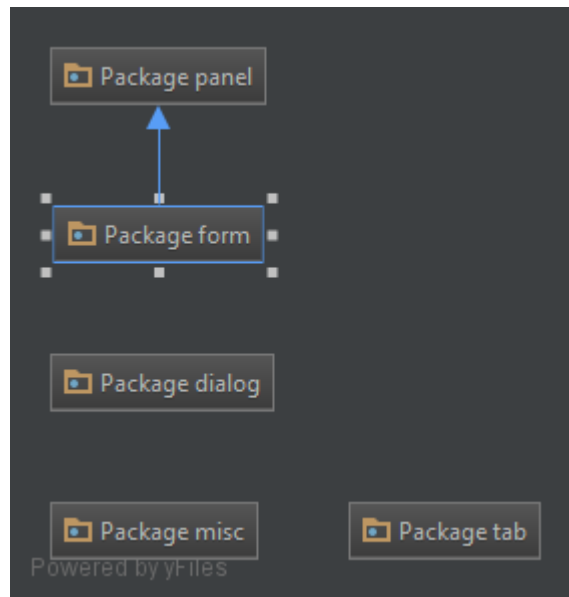
### Package Utils

Le package utils sert principalement à la génération de PDF via la classe `GenerationPdf` s'appuyant sur `itextpdf`.

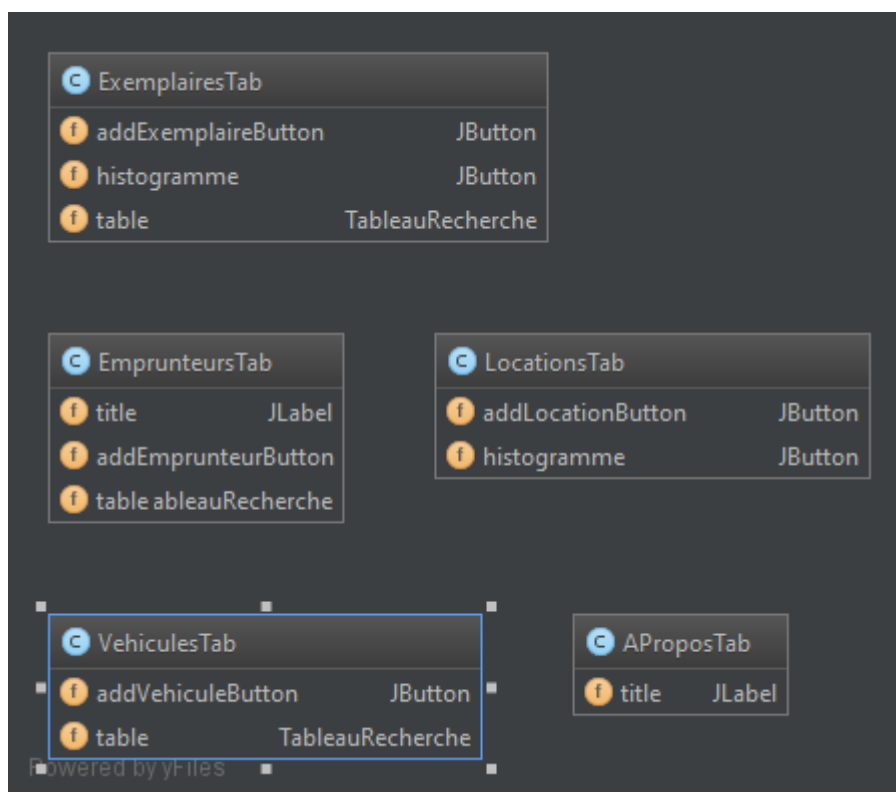


### Package Views

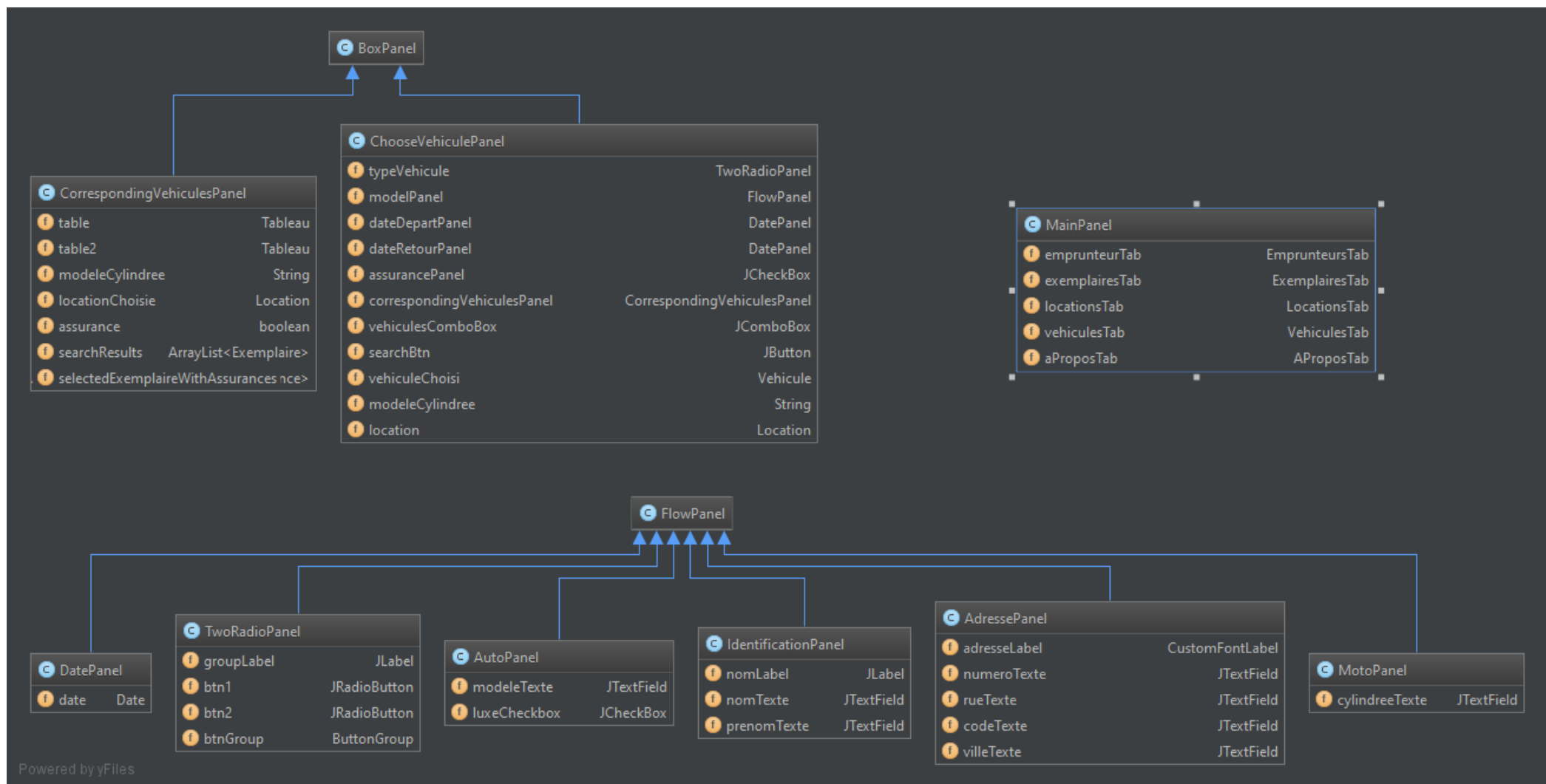
Le package views contient principalement un package components contenant tous les sous-packages appelés par l'application, et une classe `LocationView` appelée par la classe principale, qui appelle ensuite les fonctions nécessaires dans le package components.

*Package Components*

Le package components contient, comme son nom l'indique, tous les composants nécessaires à l'affichage de l'interface utilisateur basée sur Swing.

*Package Tab*

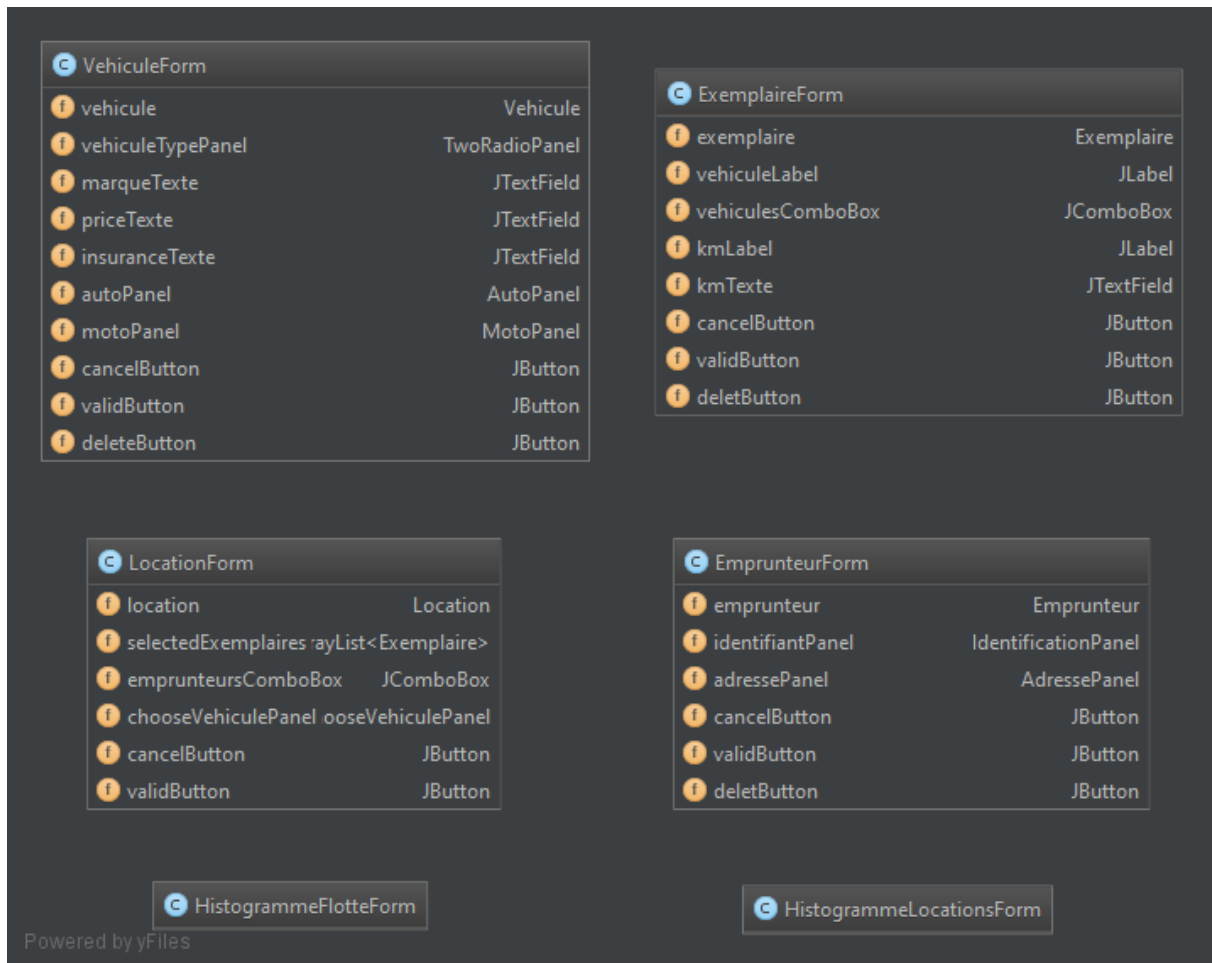
Le package tab gère les onglets du programme, c'est-à-dire le contenu de chaque vue du programme accédée par un onglet. Chaque onglet a sa propre classe.



### Package Panel

Le package panel constitue l'essentiel des briques réutilisables de l'interface Swing, chaque création de nouveau panneau (JPanel) passe par un appel à une classe de ce package. Deux classes sont essentielles : BoxPanel et FlowPanel, utilisées par des classes filles à chaque fois qu'une fenêtre a besoin d'une disposition de type BorderLayout ou FlowLayout. MainPanel, elle, sert à donner vie aux onglets (Tabs) décrits ci-dessus.



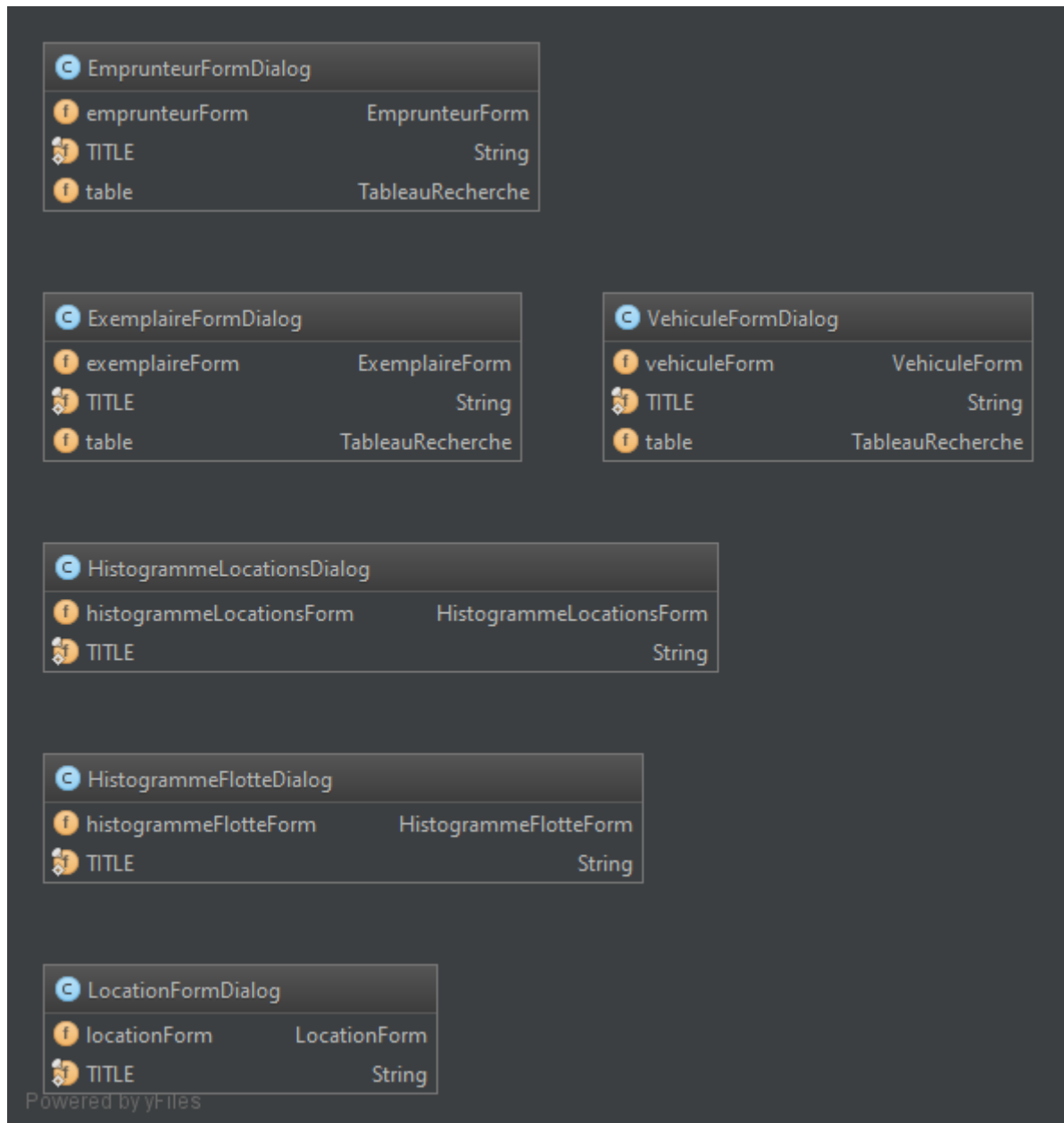


### Package Form

Le package form hérite du package précédent, panel. Il sert à la mise en forme de tous les formulaire type insertion/modification/suppression d'une location, d'un emprunteur, d'un exemplaire ou d'un véhicule.

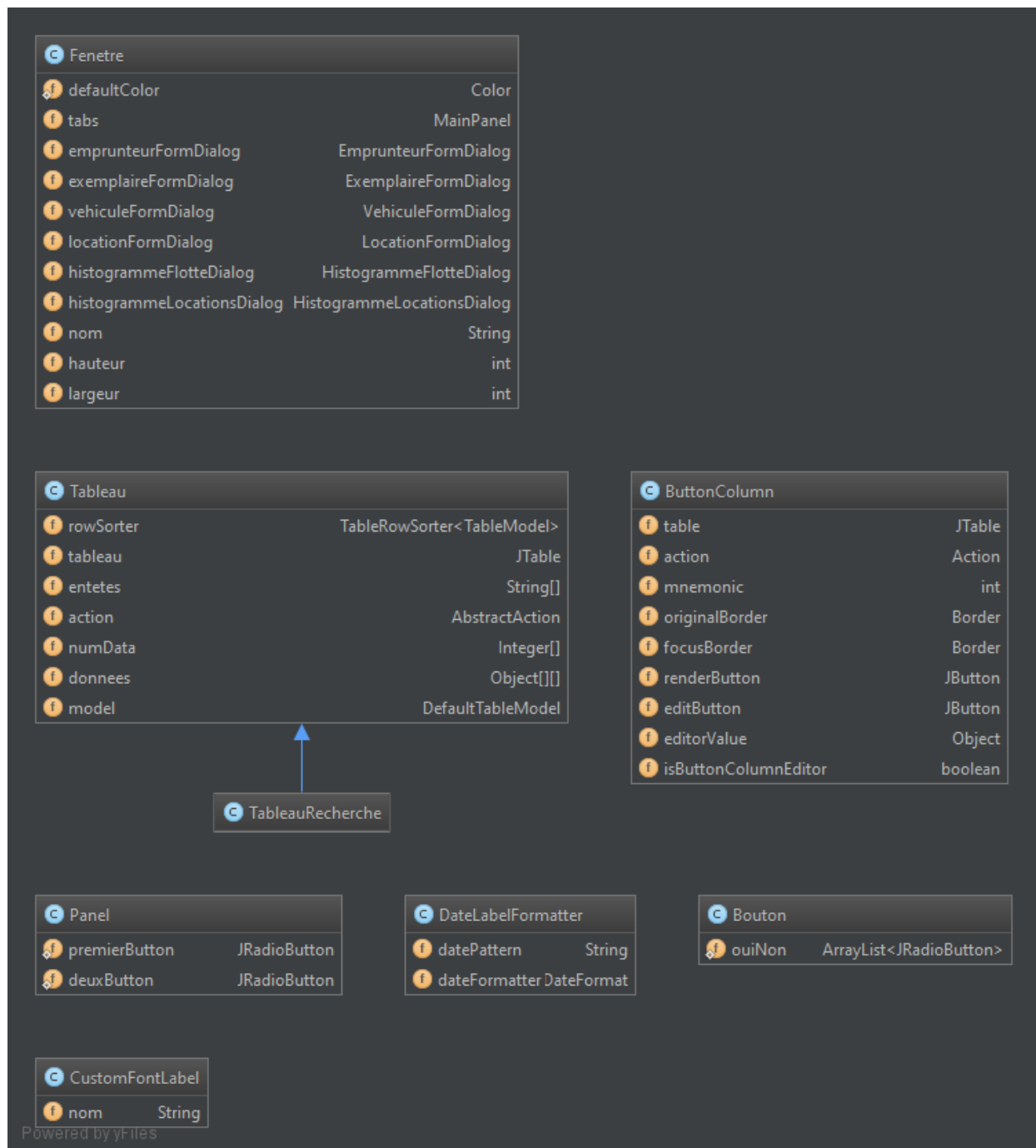
Il contient également l'affichage des deux histogrammes de flotte et de répartition numéraire des locations.

Note : une fois le code rendu, nous nous sommes rendu compte que nous aurions du faire une classe BaseForm pour les attributs communs à tous les formulaires (boutons de validations).



### *Package Dialog*

Le package dialog sert à la génération des fenêtre modales, ce sont ses classes qui gèrent les actions entreprises lors du clic d'un bouton.



### Package Misc

Dans ce package se trouvent toutes les classes qui n'ont pas pu être rangées ailleurs. On retrouve :

- Panel et Bouton pour la génération de deux boutons radio côte à côte type Oui/Non
- CustomFontLabel pour la génération de texte dans une police précise
- DateLabelFormatter pour la conversion au format SimpleDateFormat d'une Date du package Models et vice versa
- Fenetre pour la génération de la fenêtre de base de l'application
- Tableau pour la génération d'un tableau JTable prenant en compte les colonnes, les données à remplir ainsi que le type de chaque colonne afin que le tri lors d'un clic sur le nom de la colonne se fasse correctement
- TableauRecherche similaire à Tableau, avec ajout d'un formulaire de recherche dynamique
- ButtonColumn pour la gestion d'un bouton dans une colonne d'un tableau



### Package Test

Le package test est extérieur au package location principal. Il sert à tester l'ensemble des fonctions, principalement du package models à l'aide de JUnit. Le but est de vérifier des assertions telles qu'une nouvelle instance d'adresse doit retourner un texte particulier lors de son affichage, ou vérifier le bon fonctionnement des getter et setter de données.

LocationDevisFactureTest sert à vérifier que la génération de documents PDF pour les factures et les devis est bien conforme à un document généré précédemment qui est correct.

Enfin, LegacyTest contient les anciens tests qui n'étaient pas sous la forme JUnit que nous n'avons pas eu le temps de convertir au nouveau format, notamment les tests de recherche dans un containers, devenus de toute façon obsolète depuis l'utilisation de JPA.

Du fait de l'utilisation de JPA, nous n'avons pas eu à créer de base de données nous-même, cette dernière étant générée dynamiquement par eBean, qui joue le rôle de l'ORM.

La structure ainsi réalisée est la suivante :

location	
id	integer
is_active	integer
debut_jour	integer
debut_mois	integer
debut_annee	integer
fin_jour	integer
fin_mois	integer
fin_annee	integer
rendu_jour	integer
rendu_mois	integer
rendu_annee	integer
emprunteur_id	integer
approuvee	integer
version	integer
when_created	text
when_updated	text

vehicule	
id	integer
dtype	text
is_active	integer
marque	text
prix_jour	integer
prix_assurance	integer
version	integer
when_created	text
when_updated	text
modele	text
luxe	integer
cylindree	integer

emprunteur	
id	integer
is_active	integer
nom	text
prenom	text
numero	integer
rue	text
cp	text
ville	text
version	integer
when_created	text
when_updated	text

exemplaire	
id	integer
is_active	integer
kilometres	integer
vehicule_id	integer
reservoir	real
endommagement	integer
version	integer
when_created	text
when_updated	text

location_exemplaire	
id	integer
is_active	integer
exemplaireid	integer
locationid	integer
assurance	integer
version	integer
when_created	text
when_updated	text

exemplaire_location	
location_id	integer
exemplaire_id	integer

Powered by yf-ile

Exemple de fichier PDF généré pour le devis :

**LocAppli**  
**Devis #99999999**

**Devis pour Adrien Poupa**

Prestation	Période	Montant TTC
Véhicule loué : Dacia Sandero	1 Janvier 2016 - 1 Février 2016	180.0€
Assurance	1 Janvier 2016 - 1 Février 2016	30€

**Total devis : 210.0€**

**Attention : le devis ne contient d'éventuels frais de réparation et de plein qui peuvent s'appliquer**

**(C) 2016 LocAppli**

**Stéphane Gâteau, Adrien Poupa, Timothée Barbot**

Exemple de fichier PDF généré pour la facture :

**LocAppli**  
**Facture #99999999**

**Facture pour Adrien Poupa**

Prestation	Période	Montant TTC
Véhicule loué : Dacia Sandero	1 Janvier 2016 - 1 Février 2016	180.0€
Assurance	1 Janvier 2016 - 1 Février 2016	30€

**Total à régler : 210.0€**

**(C) 2016 LocAppli**

**Stéphane Gâteau, Adrien Poupa, Timothée Barbot**

## Explication de code

Le lancement de l'application se passe dans le fichier Application.java, dont la fonction main appelle une fonction de récupération du singleton :

```
/**
 * Lancement de l'application
 * @param args arguments par défaut
 */
public static void main(String[] args){
    Application app = getApp();
}
```

```
/**
 * Obtention du singleton
 * @return singleton App
 */
public static Application getApp()
{
    if(app == null){
        app = new Application();
    }

    return app;
}
```

Puis appel du constructeur privé qui lance la fenêtre Swing dans un contexte favorable aux threads :

```
/**
 * Constructeur privé
 */
private Application(){
    System.out.println("Lancement de l'application en cours...");

    initEbeanServer();

    // initialize data of the application
    if (isDeveloppement) {
        initData();
    }

    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            // initialize frame
            view = new LocationView();
        }
    });
}
```

Le fichier LocationView.java est ensuite appelé :

```
/**
 * Construction de la fenêtre de l'application
 */
public LocationView() {
    locationFenetre = new Fenetre("LocAppli", 600, 600);
}
```



Création d'une Fenetre.java :

```
/**
 * Constructeur par défaut
 * @param nom nom fenêtre
 * @param hauteur hauteur
 * @param largeur largeur
 */
public Fenetre(String nom, int hauteur, int largeur){
    super(nom);
    this.largeur = largeur;
    this.hauteur = hauteur;
    initFenetre();
}

/**
 * Initialisation de la fenêtre
 */
public void initFenetre(){
    setSize(largeur, hauteur);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    tabs = new MainPanel();
    add(tabs);
    setVisible(true);
}
```

Création du MainPanel.java pour la création des onglets :

```
/**
 * Constructeur par défaut
 */
public MainPanel() {
    super();
    initMainPanel();
}

/**
 * Initialisation du panneau
 */
private void initMainPanel(){
    emprunteurTab = new EmprunteursTab();
    exemplairesTab = new ExemplairesTab();
    locationsTab = new LocationsTab();
    vehiculesTab = new VehiculesTab();
    aProposTab = new AProposTab();

    addTab("Locations", locationsTab);
    addTab("Emprunteurs", emprunteurTab);
    addTab("Exemplaires", exemplairesTab);
    addTab("Vehicules", vehiculesTab);
    addTab("A propos", aProposTab);

    setVisible(true);
}
```

Enfin, c'est dans le fichier d'onglet qu'est ajouté le contenu de la fenêtre :

```
/**
 * Initialisation de la fenêtre
 */
public EmprunteursTab() {
    super();
    setBackground(Color.orange);
    setLayout(new BorderLayout(this, BorderLayout.Y_AXIS));
    initContent();
}

/**
 * Contenu de la fenêtre
 */
private void initContent() {
    // rajout du titre
    title = new CustomFontLabel("Liste des emprunteurs", "Calibri",
    Font.PLAIN, 25);
    title.setAlignmentX(Component.CENTER_ALIGNMENT);
    add(title);

    // Bouton rajout Ajouter un emprunteur
    addEmprunteurButton = new JButton("Ajouter un emprunteur");
    addEmprunteurButton.addActionListener(new ButtonListener());
    addEmprunteurButton.setAlignmentX(Component.CENTER_ALIGNMENT);
    add(addEmprunteurButton);

    // Données tableau
    // Initialisation du tableau, variable table... voir code complet
    add(table);
}
```

Les données du tableau sont récupérées via un container :

```
Object[][] donnees = new
Object[location.containers.Emprunteurs.get().size()][5];
```

## Conclusion

Il a été très formateur de travailler sur des notions que nous n'avions pas vues en cours, telles JPA, eBean, Gradle, JUnit ou des composants Swing que nous n'avions pas abordé.

La répartition des tâches et l'entente entre nous a été bonne et sans problème, le seul désagrément que nous avons rencontré a été le décalage horaire entre la Malaisie et l'Angleterre quand il a fallu finaliser le projet. Cependant, cela a été formateur dans le sens où il s'agit de situations que nous pourrions être amenés à rencontrer en milieu professionnel.

Nous aurions pu améliorer de nombreuses choses, comme par exemple l'apparence : dans l'idéal, il aurait fallu essayer de faire une interface plus épurée et moins « brute ». Nous aurions pu rajouter des images et des sons lors de la confirmation d'une réservation, par exemple.

Les tests unitaires et le rapport ont été écrits à la fin du projet. La transcription d'un gros fichier de test avec affichage dans la console en petites fonctions d'assertion JUnit a pris du temps, alors que nous aurions pu directement écrire ces tests et ainsi adopter une démarche de TDD (test-driven development).

Enfin, nous avons sous-estimé la durée de la rédaction d'une Javadoc. Nous avons en effet commencé par développer sans nous préoccuper des commentaires, ce qui a rendu la masse de travail assez grande quand il a fallu tout commenter en une seule fois. Il faudrait mettre ces commentaires en même temps que les fonctions sont développées.

## Bibliographie

Liste des ressources utilisées pour mener à bien le projet :

- Interface Swing
  - Tableau
    - Bouton dans un tableau : <https://tips4java.wordpress.com/2009/07/12/table-button-column/>
    - Recherche dans un tableau : <http://stackoverflow.com/questions/22066387/how-to-search-an-element-in-a-jtable-java>
    - Tri dans un tableau : <http://stackoverflow.com/questions/6592192/why-does-my-jtable-sort-an-integer-column-incorrectly>
  - Onglets
    - <https://docs.oracle.com/javase/tutorial/uiswing/components/tabbedpane.html>
    - <http://www.wideskills.com/java-tutorial/java-jtabbedpane-class-example>
- Tests unitaires JUnit : <https://github.com/junit-team/junit4/wiki>
- Génération PDF : <http://www.vogella.com/tutorials/JavaPDF/article.html>
- Persistance des données via JPA
  - eBean : <http://ebean-orm.github.io/docs/>
  - Intégration eBean dans IntelliJ : <https://plugins.jetbrains.com/plugin/7801>
  - Gradle : <http://stackoverflow.com/questions/22823061/gradle-ebean-enhancement-entity-not-enhanced>
  - <https://github.com/ebean-orm/avaaje-ebeanorm>