21 novembre 2016

Adrien Mousty

3ème informatique de gestion

HEPH Condorcet

Projet java   
station de ski

****

Table des matières

[2 Enoncé 4](#_Toc467438843)

[2.1 Optionnellement : 6](#_Toc467438844)

[2.2 Consignes 6](#_Toc467438845)

[2.3 Contrainte 7](#_Toc467438846)

[4 Schémas 8](#_Toc467438847)

[4.1 Diagramme de classe 8](#_Toc467438848)

[4.2 Diagramme use case 9](#_Toc467438849)

[4.3 Quelques diagrammes de séquence 10](#_Toc467438850)

[4.3.1 Création d’un client 10](#_Toc467438851)

[4.3.2 Savoir si un moniteur donné a des réservations pour une semaine donnée 10](#_Toc467438852)

[4.3.3 Effectuer une réservation 11](#_Toc467438853)

[5 Pour utiliser le programme 12](#_Toc467438854)

[5.1 En tant que moniteur 12](#_Toc467438855)

[5.2 En tant que client 12](#_Toc467438856)

[6 Découpe du projet - packages 13](#_Toc467438857)

[6.1 Access To Dao 13](#_Toc467438858)

[6.2 Dao 14](#_Toc467438859)

[6.3 Pojo 15](#_Toc467438860)

[6.4 Fenêtre 15](#_Toc467438861)

[6.5 Utilitaire 16](#_Toc467438862)

[6.5.1 ButtonColumn 16](#_Toc467438863)

[6.5.2 ComboItem 16](#_Toc467438864)

[6.5.3 DateLabelFormatter 16](#_Toc467438865)

[7 Bouts de code issu du projet 17](#_Toc467438866)

[7.1 Lambda expressions 17](#_Toc467438867)

[7.1.1 AnyMatch 17](#_Toc467438868)

[7.1.2 Stream, filter & collect 17](#_Toc467438869)

[7.2 Opérateur ternaire 18](#_Toc467438870)

[7.3 JTable 19](#_Toc467438871)

[7.3.1 Code couleur : 19](#_Toc467438872)

[7.3.2 Insertion de boutons dans un JTable 20](#_Toc467438873)

[8 Outils 21](#_Toc467438874)

[8.1 SQLite studio 21](#_Toc467438875)

[8.2 Git 22](#_Toc467438876)

[8.3 Git Hub 22](#_Toc467438877)

[8.4 WindowBuilder 23](#_Toc467438878)

[8.5 JDatePicker 24](#_Toc467438879)

[9 Difficultés rencontrées 25](#_Toc467438880)

[9.1 Les contraintes 25](#_Toc467438881)

[9.2 « Database locked » 25](#_Toc467438882)

[9.3 Git 26](#_Toc467438883)

[9.4 Manque de temps 26](#_Toc467438884)

[10 Note de mise à jour 27](#_Toc467438885)

[10.1 Alpha 27](#_Toc467438886)

[10.1.1 Version Alpha 0.0.1 27](#_Toc467438887)

[10.1.2 Version Alpha 0.0.2 27](#_Toc467438888)

[10.1.3 Version Alpha 0.0.3 27](#_Toc467438889)

[10.1.4 Version Alpha 0.0.4 27](#_Toc467438890)

[10.1.5 Version Alpha 0.0.5 27](#_Toc467438891)

[10.1.6 Version Alpha 0.0.6 27](#_Toc467438892)

[10.1.7 Version Alpha 0.0.7 27](#_Toc467438893)

[10.1.8 Version Alpha 0.0.8 27](#_Toc467438894)

[10.1.9 Version Alpha 0.0.9 27](#_Toc467438895)

[10.1.10 Version Alpha 0.0.10 28](#_Toc467438896)

[10.1.11 Version Alpha 0.0.11 28](#_Toc467438897)

[10.1.12 Version Alpha 0.0.12 28](#_Toc467438898)

[10.1.13 Version Alpha 0.0.13 28](#_Toc467438899)

[10.1.14 Version Alpha 0.0.14 28](#_Toc467438900)

[10.2 Beta 29](#_Toc467438901)

[10.2.1 Version Beta 0.1.0 29](#_Toc467438902)

[10.2.2 Version Beta 0.2.0 29](#_Toc467438903)

[10.2.3 Version Beta 0.3.0 29](#_Toc467438904)

[10.2.4 Version Beta 0.4.0 29](#_Toc467438905)

[10.2.5 Version Beta 0.5.0 30](#_Toc467438906)

[10.2.6 Version Beta 0.6.0 30](#_Toc467438907)

[10.2.7 Version Beta 0.7.0 30](#_Toc467438908)

[10.2.8 Version Beta 0.8.0 30](#_Toc467438909)

[10.2.9 Version Beta 0.9.0 30](#_Toc467438910)

[10.3 Version finale 31](#_Toc467438911)

[10.3.1 Version Finale 1.0.0 31](#_Toc467438912)

[10.3.2 Version Finale 1.1.0 31](#_Toc467438913)

[10.3.3 Version finale 1.2.0 31](#_Toc467438914)

[10.3.4 Version finale 1.3.0 31](#_Toc467438915)

[10.3.5 Version finale 1.4.0 31](#_Toc467438916)

[10.3.6 Version finale 1.5.0 32](#_Toc467438917)

[10.3.7 Version finale 1.6.0 32](#_Toc467438918)

[10.3.8 Version finale 1.7.0 32](#_Toc467438919)

[11 Conclusion 33](#_Toc467438920)

[12 Table des illustrations 34](#_Toc467438921)

[13 Annexes 35](#_Toc467438922)

# Énoncé

Voici les règles métiers de l’école de ski du Domaine Châtelet 260 m.

Gérez les moniteurs, les élèves et les réservations correspondantes.

Il y a des cours de différents niveaux, de différentes tranches d’âge et de différents types (snowboard ou ski). Attention que tous les moniteurs ne sont pas accrédités pour donner tous les types de cours par catégorie d’âge (adulte, enfant : entre 4 et 12 ans) ou par type de sport (snowboard, ski alpin, ski de fond et télémark).

Les cours particuliers sont possibles sur le temps de midi (pendant 1 ou 2 heures) lorsqu’il reste des disponibilités de moniteur (possibilité de réservation 1 mois à l’avance en dehors des périodes scolaires et 1 semaine lors des périodes scolaires). Ces cours sont donnés d’une à 4 personnes. Le prix s’élève à 50€ pour une heure et à 80€ pour 2 heures consécutives.

Pour les cours collectifs, il y a un nombre minimum et maximum d’élèves par cours :

* Entre 5 et 8 pour les cours enfants et snowboard
* Entre 6 et 10 pour les cours adultes
* Entre 5 et 8 pour les cours compétition et hors-piste (ski ou snowboard)

Une semaine de cours collectifs consiste en 6 ½ jours de cours :

* Matin : de 9 :00 à 12:00
* Après-midi : de 14 :00 à 17 :00

Niveaux enfants ski :

* Petit Spirou 100 € / semaine
* Bronze 120 € / semaine
* Argent 120 € / semaine
* Or 120 € / semaine
* Platine 120 € / semaine
* Diamant 120 € / semaine
* Compétition 130 € / semaine
* Hors-piste 130 € / semaine

Niveaux enfants snowboard (à partir de 6 ans) :

* 1 110€ / semaine
* 2 à 4 120 € / semaine
* Hors-piste 130€ / semaine

Niveaux en ski adulte :

* De 1 à 4 130 € / semaine
* Hors-piste 150 € / semaine
* Compétition 150 € / semaine

Niveaux en snowboard

* De 1 à 4 130 € / semaine
* Hors-piste 150 € / semaine

Niveaux télémark :

* De 1 à 4 120 € / semaine

Niveaux ski de fond :

* De 1 à 4 100 € / semaine

Une assurance optionnelle de 15 € / semaine est proposée aux élèves.

Une réduction de 15% est octroyée aux élèves qui prennent des cours matin et après-midi.

L’application permettra de gérer au mieux les horaires des moniteurs et des cours qu’ils donnent selon leurs capacités et leurs disponibilités.

La station ouvre du samedi 03/12/2016 au dimanche 07/05/2017.

* Semaine commence à partir du 05

Les périodes de congé scolaires sont :

* Du 24/12/2016 => 07/01/2017 (Noël / Nouvel an)
* Du 25/02/2017 au 04/03/2017 (Carnaval)
* Du 01/04/2017 au 15/04/2017 (Pâques)

## Optionnellement :

* Rendre le modèle le plus modulaire possible afin de s’adapter facilement à d’autres contraintes pour une école ou un autre domaine :
* Horaires différents
* Lieux de cours différents (éventuellement différents lieux de rassemblements dans la même station)
* Paiement des moniteurs en fonction de leurs prestations effectives

Il vous est demandé de modéliser un tournoi à l’aide des notations UML suivantes :

* Cas d’utilisation (use cases).
* Diagramme de classes.
* Diagrammes de séquences

## Consignes

Le programme implémentant la modélisation UML sera écrit en Java. Les données seront sauvegardées dans une base de données de votre choix (Tout doit être inclus dans le programme pour son exécution). Vous utiliserez le pattern DAO.

Le dossier comprenant la modélisation UML (use cases, diagrammes de classes, diagrammes de séquences et explications), l’explication de l’implémentation du pgm Java et le code Java (rendre le workspace), sera à rendre **au plus tard le vendredi 18 novembre 2016 à 8h15’** c.-à-d. pendant le cours de programmation avancée.

Aucun délai supplémentaire ne sera accordé.

Si le programme n’est pas rendu dans les délais, l’étudiant se verra attribuer 0 pour ce travail.

Toute copie identique du programme sera sanctionnée par un 0 pour les étudiants concernés.

Ce programme interviendra pour 35% des points de la cote finale du cours de programmation avancée I.

## Contrainte

1. Le client peut faire plusieurs réservations.
2. Pas d’obligation vis-à-vis d’une interface graphique.
3. Attention au minium & maximum élèves pour les réservations.
4. On suppose que les cours sont suspendus le samedi sauf éventuellement les cours particuliers.
5. Le cours est annulé s’il n'y a pas assez d’inscrits.
6. Si un élève prend un cours au matin et un autre en après-midi, elle ne prend que 15€ d’assurance (15€/Semaine).
7. L’assurance pour les cours groupés et particuliers n'est pas la même.
8. Age minimum 4 ans sauf pour le snowboard, qui est de 6 ans.
9. Les cours de télémark et ski de fond ont eux aussi leurs cours particuliers.

# Schémas

## Diagramme de classe

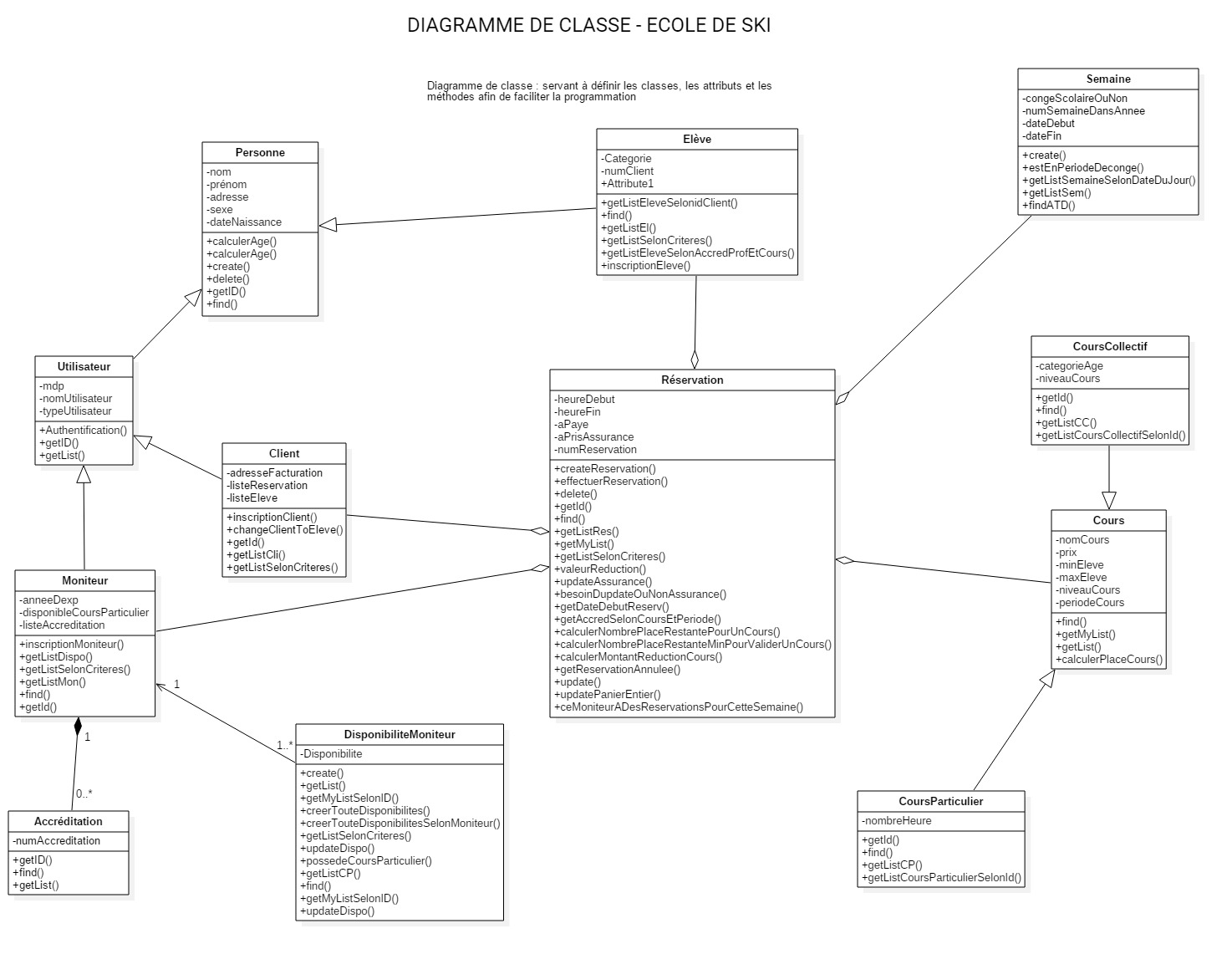
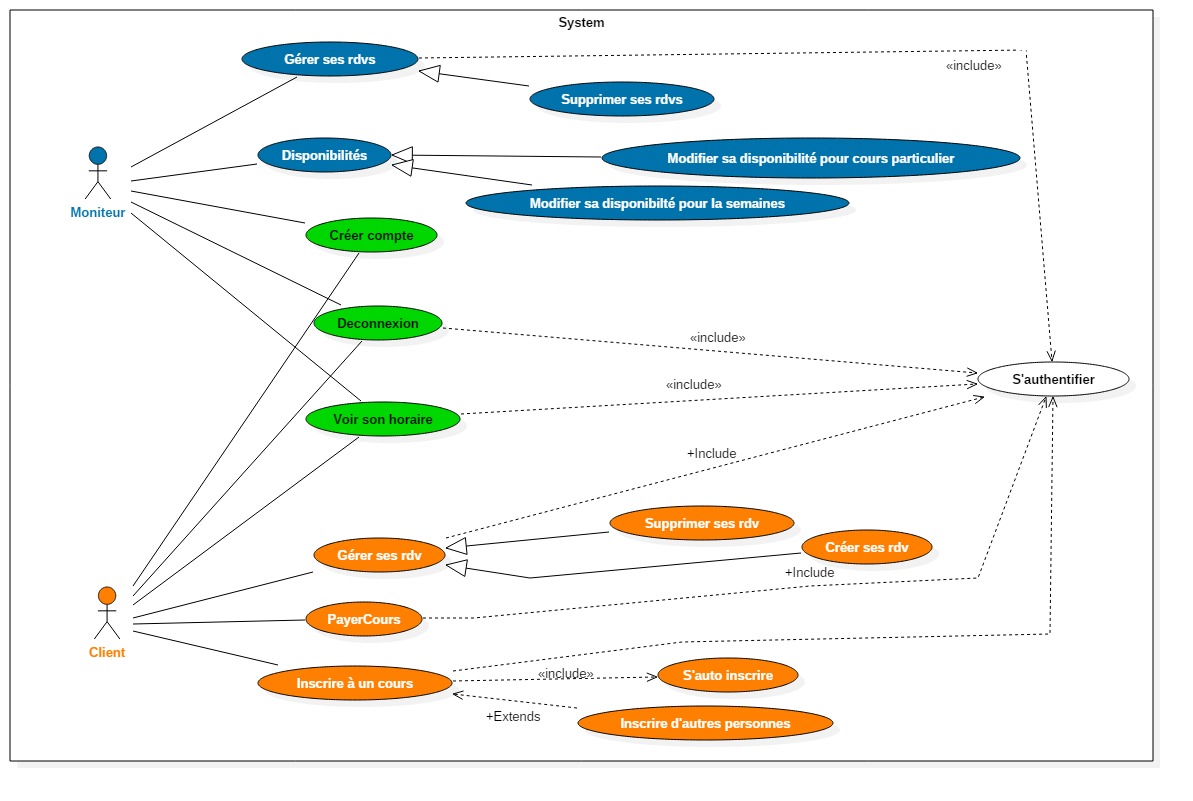


Figure 1 Diagramme de classe[[1]](#footnote-1)

## Diagramme use case

Figure Diagramme use case



## Quelques diagrammes de séquence

### Création d’un client

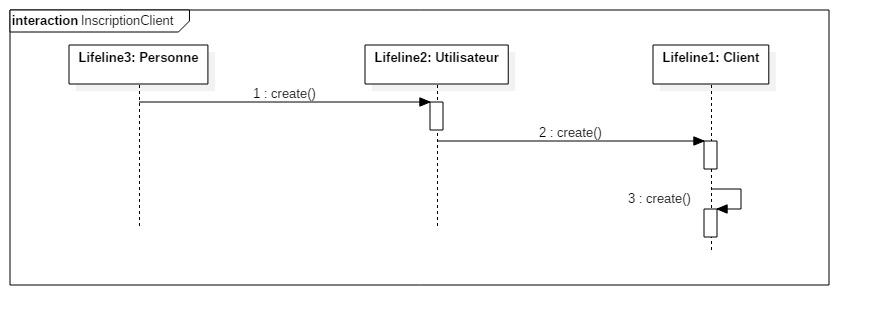


Figure 3 Création d'un client

Pour inscrire un client, un enregistrement Personne sera créé en premier.

### Savoir si un moniteur donné a des réservations pour une semaine donnée

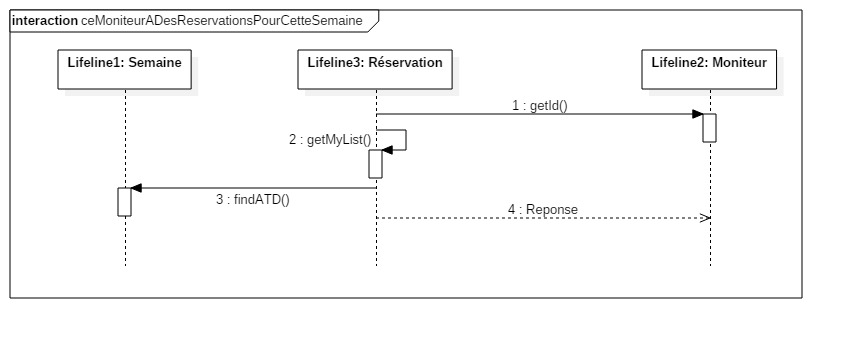
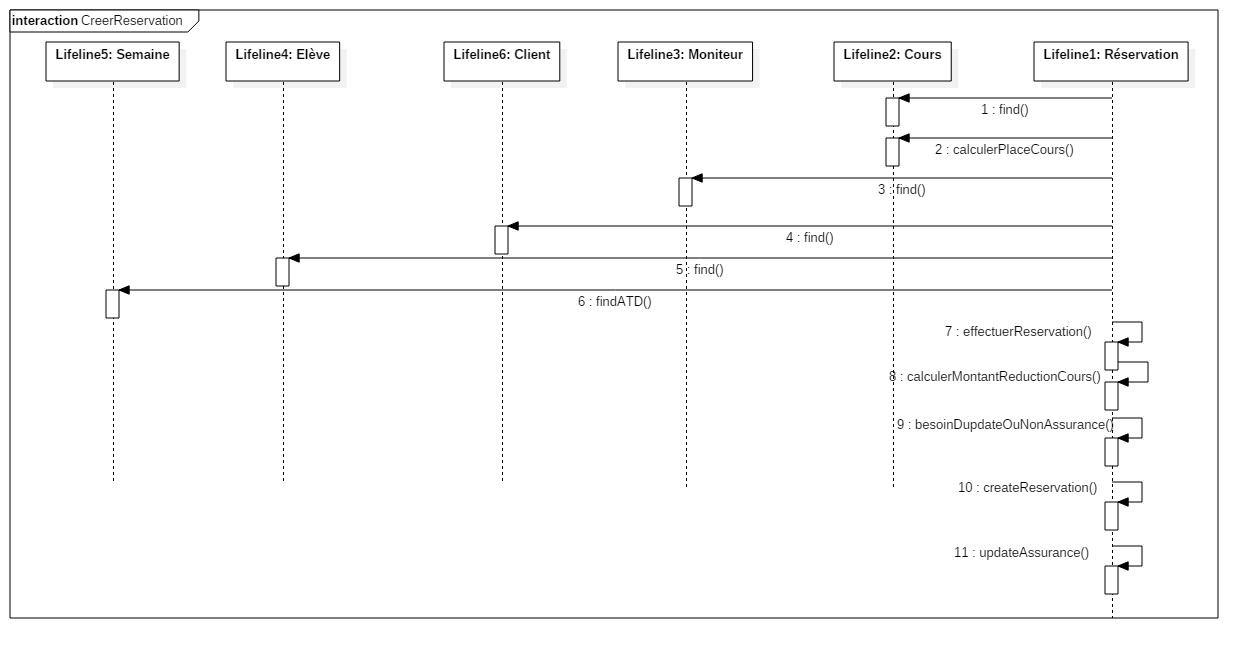


Figure 4 Savoir si le moniteur a des réservations pour une certaine semaine

1. Il faut récupérer les données pour n’avoir qu’une seule semaine ainsi qu’un seul moniteur.
2. Grâce à cela il va ressortir une liste pour un seul moniteur.
3. Ensuite nous avons besoin d’une semaine donnée, afin de comparer les données avec la liste.
4. S’il trouve des données, alors il retourne vrai. Le cas échéant il retourne faux.



### Effectuer une réservation

Figure Séquence : effectuer une réservation

1. Pour effectuer une réservation nous avons de beaucoup d’éléments.
2. Afin d’ajouter une réservation, il faut savoir s’il reste assez de places.
3. Après avoir reçu tous les éléments en paramètre, la fonction de vérification vérifie si l’élève possède l’âge nécessaire (si le cours est de type « Snowboard »).
4. Pas besoin de vérifier le reste, car ils sont déjà filtrés en amont dans les comboBox lors de la réservation.
5. Il calcule le montant et vérifie s’il y a une réduction
6. Si la réservation s’est correctement effectuée, on vérifie update l’assurance si besoin.
7. Création de la réservation
8. S’il faut update la précédente[[2]](#footnote-2) assurance, on le fait.

# Pour utiliser le programme

## En tant que moniteur

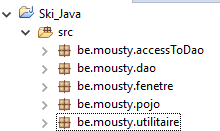
|  |  |
| --- | --- |
| Nom d’utilisateur | Mot de passe |
| mon1 | test |
| mon2 | test |
| mon3 | test |
| mon4 | test |
| mon5 | test |
| mon6 | test |
| mon7 | test |
| mon9 | test |
| mon10 | test |

## En tant que client

|  |  |
| --- | --- |
| Nom d’utilisateur | Mot de passe |
| adri | test |
| nao | test |
| antho | test |
| nekfeu | test |
| bernard | test |

* Attention, il faut mettre les noms d’utilisateur en minuscule.

# Découpe du projet - packages

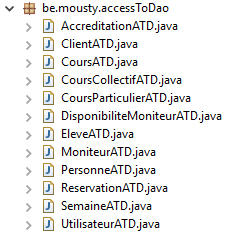


Afin de respecter le DAO ainsi que les principes de la programmation objet, il a fallu séparer le projet en 3 parties. De plus, j’ai ajouté un 4ème package, « utilitaire » ainsi qu’un 5ème package contenant toute la partie visible par l’utilisateur du programme.

Figure Découpe du projet

## Access To Dao

Les classes ATD sont en fait les classes métier.



Les classes ATD déclenchent un traitement du DAO via le polymorphisme. Cela permet d’ajouter une couche d’abstraction et de le rendre indépendant au type de base de données.

Figure Classe métier

## Dao

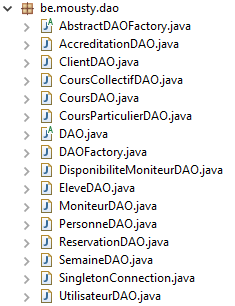
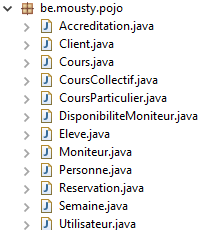
Le modèle DAO propose de regrouper les accès aux données persistantes dans des classes à part. Il n’y aura donc que dans les classes « DAO » que nous allons accéder aux données de la base de données.

Figure DAO

## Pojo[[3]](#footnote-3)

Les classes POJOs sont manipulées dans les classes DAO. Elles représentent l’état de la base de données. On peut donc y insérer les IDs, ce qui ne peut pas être fait dans les classes métiers.



Ils ne possèdent **aucune méthode** ni d’autres constructeurs qu’un constructeur par défaut.

Ils possèdent des propriétés.

Néanmoins j’ai ajouté pour chaque classe un constructeur prenant en paramètre une classe métier afin de faciliter la conversion de type.

Figure POJO

## Fenêtre

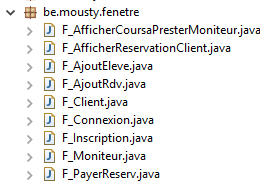
Les fenêtres sont la partie visible du programme. Elles sont générées grâce à WindowsBuilder puis modelée, un peu comme nous avons fait avec Blend lors de projets précédents en C#.

Figure JFrame

Toutes les fenêtres utilisent JFrame. C’est l'équivalent de la classe Frame de l'AWT. Elles utilisent un panneau de contenu (contentPane) pour insérer des composants.

## Utilitaire

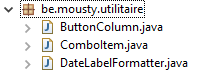
 La classe utilitaire contient des classes essentiellement trouvées sur internet puis modifiée à ma guise.

Figure Utilitaire

### ButtonColumn[[4]](#footnote-4)

Cet utilitaire offre un rendu et un éditeur qui ressemble à un JButton.

On l’utilise ensuite alors pour une colonne spécifiée dans la table.

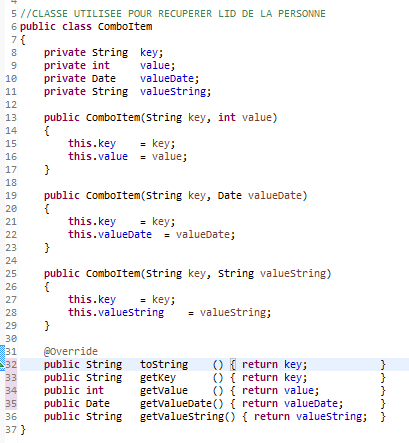
Le bouton est déclenché par un clic de souris.

1. Lorsque le bouton est invoqué, l'Action fournie est appelée.
2. La source de l'action sera le tableau.

* Un exemple se trouve au point 7.3

### ComboItem[[5]](#footnote-5)

Permet simple d’ajouter comme valeur n’importe quel type (date, integer, String, etc.)



### DateLabelFormatter[[6]](#footnote-6)

Lié au fonctionnement de JDatePicker.

# Bouts de code issu du projet

## Lambda expressions

Depuis Java 8, l’utilisation de lambda a fortement évolué, il était donc normal d’en implémenter dans mon projet.

### AnyMatch

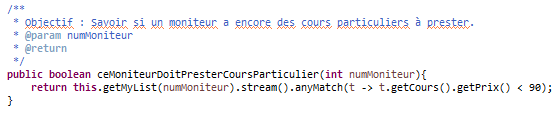


Figure 12 Lambda, anyMatch

* Si un cours ayant un prix inférieur à 90€ se trouve dans la liste, alors la fonction retournera true. Le cas échéant, il retournera false.

### Stream, filter & collect

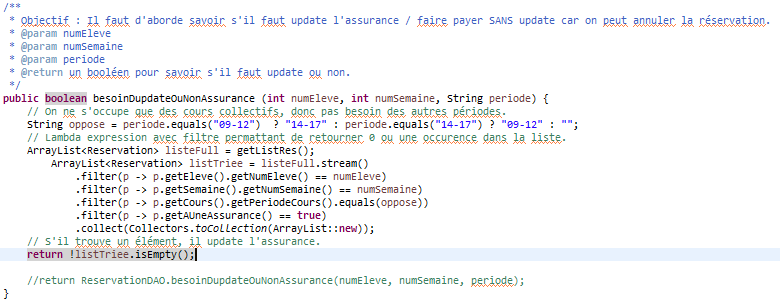


Figure 13 Stream, filter & collect

La variable « oppose » aura la valeur opposée à la période entrée en paramètre. Uniquement si la période correspond aux horaires des cours collectifs. On ne gère par celle des cours particuliers car pour ce type de cours l’assurance sera payée quoi qu’il arrive.

Ensuite, nous insérons dans la liste « listeTriee » les éléments correspondants aux différents filtrés entrée.

Nous retournons « listeFull.stream() » en type ArrayList via .collect().

S’il y a des éléments dans cette liste, nous devons update l’assurance. Car elle a déjà été payée la première fois.

## Opérateur ternaire

L’opérateur ternaire est une alternative plus compacte à la structure conditionnelle « if … else ».

Dans l’exemple ci-dessous elle est imbriquée.



Figure 14 Opérateur ternaire

## JTable

L’utilisation de JTable facilite beaucoup l’affichage d’une grande quantité d’information.

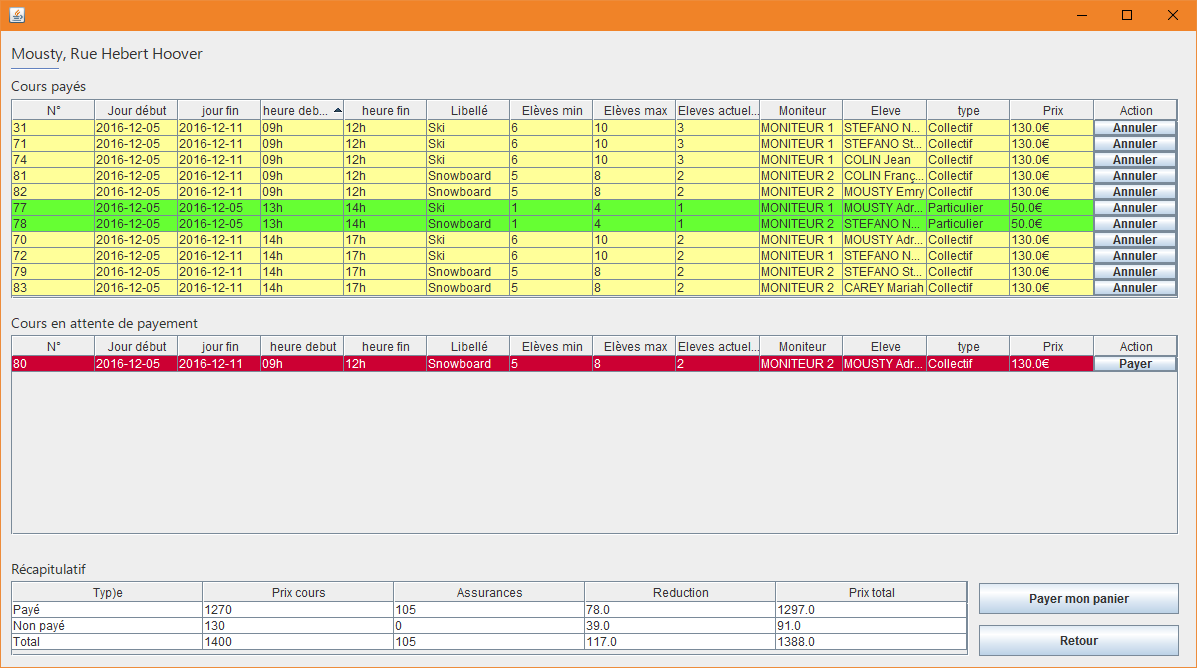


Figure 15 JTable

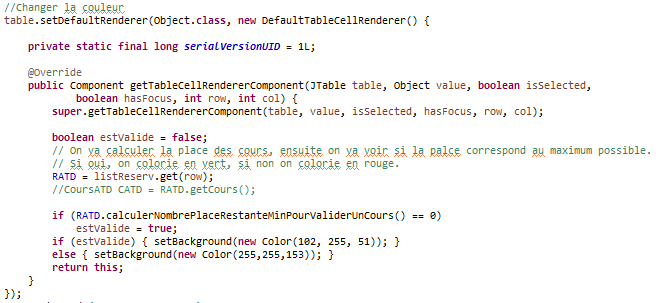
### Code couleur :

Jaune : les réservations en suspens car le quota n’est pas encore rempli,

Vert : les réservations sont validées, le quota est rempli,

Rouge : les réservations sont non validées, car en attente de payement.

Pour ce faire, voici le petit bout de code qui met en place tout ceci :



Pour générer des couleurs en hexadécimal, j’ai utilisé ce site : http://www.toutimages.com/generateur\_nr\_2.htm

### Insertion de boutons dans un JTable

Cela permet de gérer chaque réservation de manière individuelle, qu’on veuille supprimer ou payer une réservation.

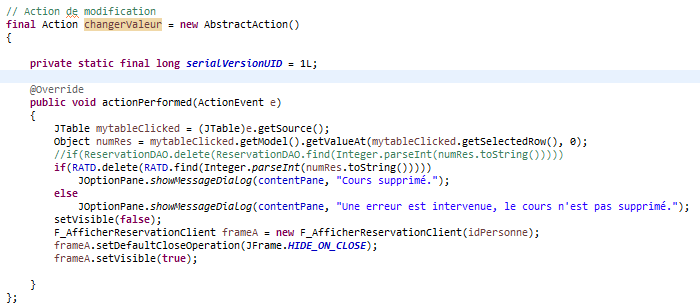
Pour éviter de dupliquer du code, j’ai créé la classe ButtonColumn. Une partie du code provient d’internet.

Ensuite, dans le code JFrame, je gère le bouton comme suit :



Les paramètres sont :

1. La table sur lequel appliquer les boutons.
2. La fonction qui sera exécutée lors du clic.



* *Via le numéro de la réservation relative à la ligne cliquée, il va tenter de supprimer la réservation.*
* *myTableClicked représente la table sur laquelle agir.*
* *numRes récupère le n° de réservation qui est le 1er élément de la lignée cliquée*
* *Il faut « parser » le n° de la réservation avant de l’insérer en paramètre.*
* *Après la tentative de suppression je ferme puis réafficher la fenêtre afin d’actualiser les valeurs.*

1. La colonne sur laquelle appliquer le bouton.

# Outils

## SQLite studio

Au détriment de Access, j’ai décidé d’utiliser une base de données de type SQLite, car je le trouve plus simple à utiliser et moins lourd.

Afin d’utiliser ce type de base de données, j’ai utilisé le logiciel **SQLite Studio**.

C’est un gestionnaire de base de données SQLite avec des fonctionnalités très intéressantes tel que :

* Portable - pas besoin d'installer ou de désinstaller.
* Interface intuitive
* Puissant, mais léger et rapide,
* Open source et gratuit (licence GPLv3).
* Possibilité de tester des requêtes SQL.

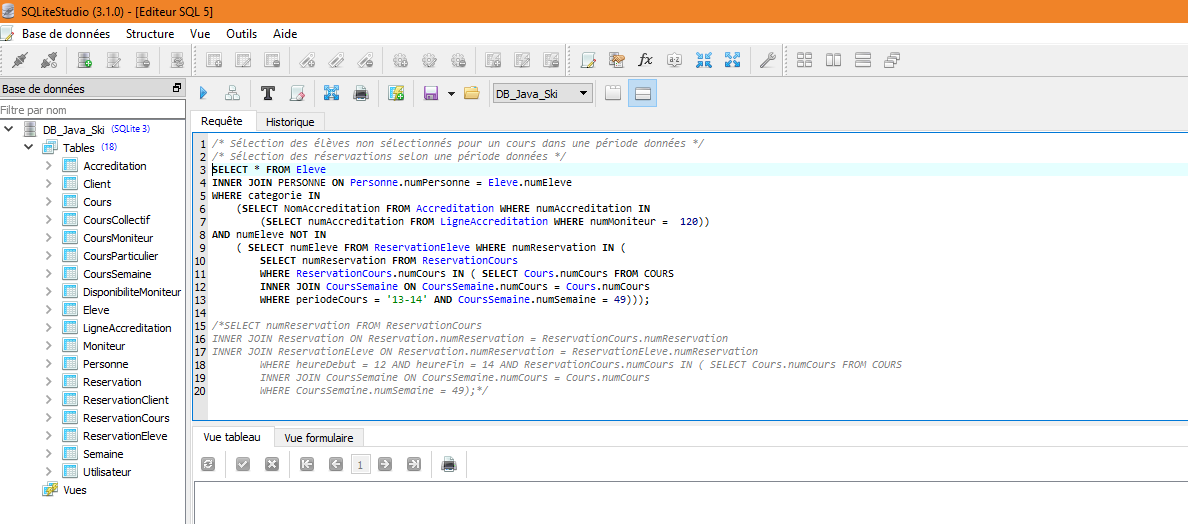


Figure 16 SQLite Studio

Afin d’utiliser ma base de données en JAVA, j’ai dû installer une librairie externe : C:\Users\Admin\AppData\Local\Microsoft\Windows\INetCacheContent.Word\2016-11-20_18h37_13.png

## Git

En cours nous avons appris à utiliser Git. Bien que celui-ci ne soit pas vraiment utile dans un projet personne, son importance est indéniable lorsque nous sommes plus d’une personne à développer le même projet.

Git est un outil permettant de gérer l'évolution du contenu d'une arborescence.

J’ai préféré utiliser le GUI de Git (Gitk) afin de mieux visualiser mes précédents « commits ».

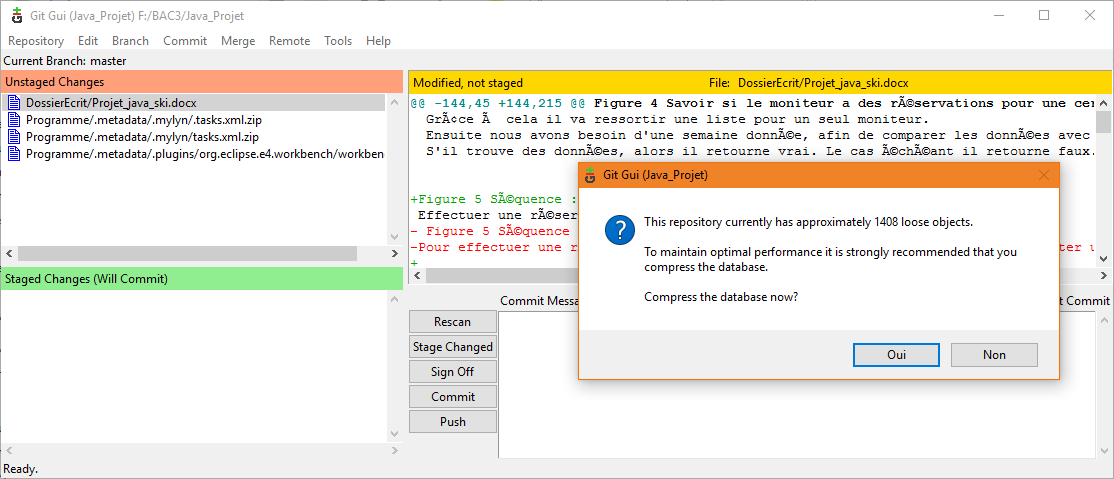


Figure 17 GitK

## Git Hub

Git Hub est le prolongement de Git. Il permet de mettre nos « commits » sur un serveur.

Dans un travail à plusieurs, cela pourrait être quelque chose d’intéressant.

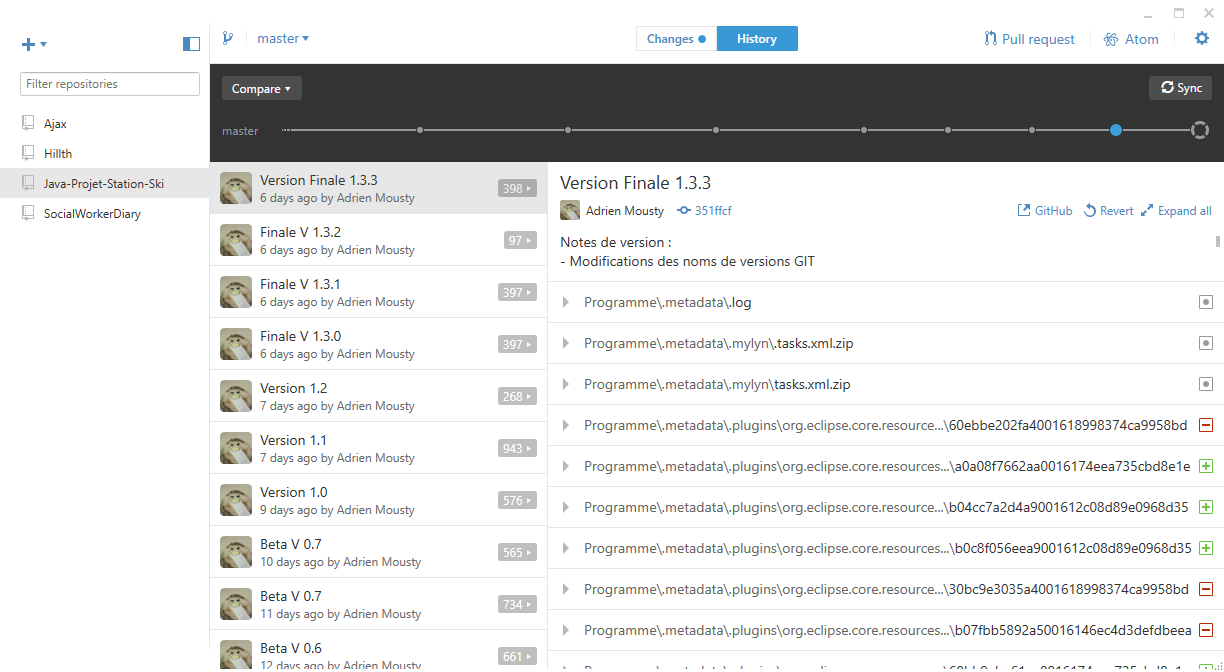


Figure 18 Git Hub

## WindowBuilder

WindowBuilder est composé de SWT Designer et Swing Designer. Il facilite la création d’applications Java GUI en diminuant le temps passé à écrire du code.

Il permet aussi bien de créer des fenêtres complexes que des fenêtres plus simples. Le code Java est généré automatiquement.

Vis-à-vis de son implémentation dans Eclipse, il se fait de la même façon que « Glassfish ».

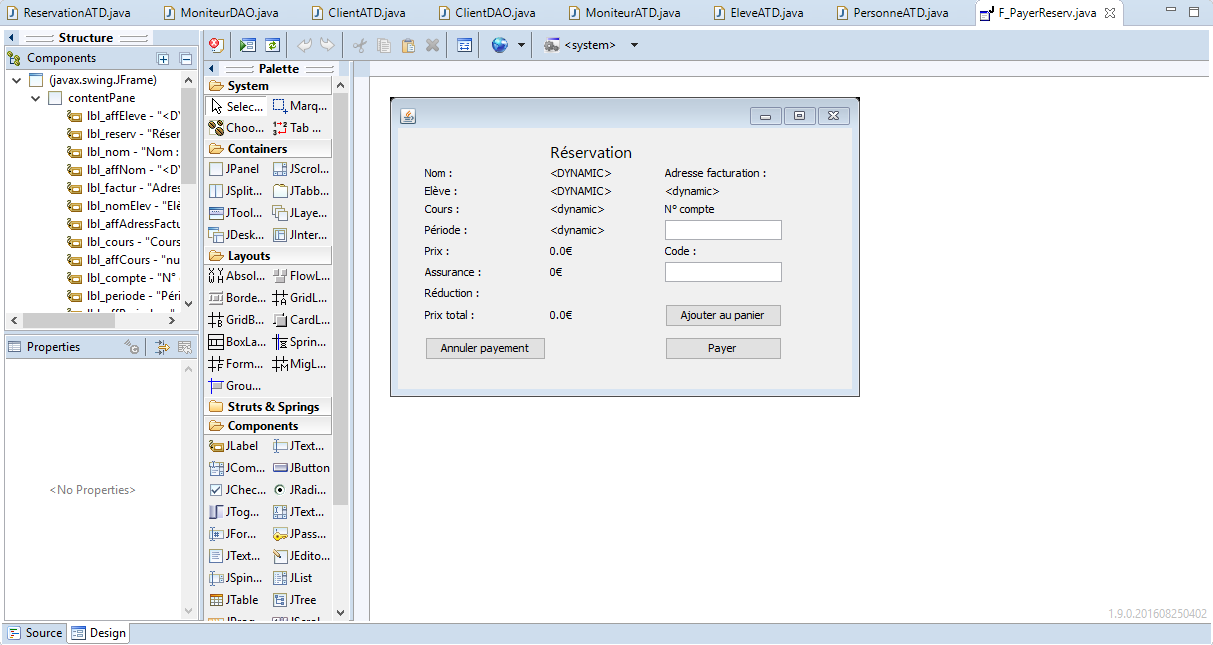


Figure 19 WindowBuilder

## JDatePicker

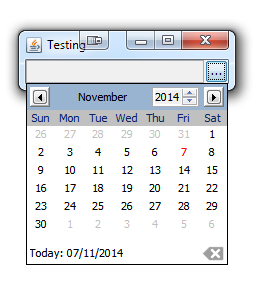
C:\Users\Admin\AppData\Local\Microsoft\Windows\INetCacheContent.Word\2016-11-20_18h54_34.pngUn bon datePicker faisant défaut au Java, j’ai cherché sur internet quelque chose « d’user friendly ». Après un peu de rechercher, JDatePicker sort du lot. C’est un outil vraiment très intéressant bénéficiant d’un certain support.

Figure 20 JDatePicker

Il s’implémente comme suit :

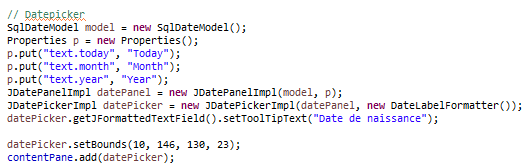


Figure 21 JDatePicker en Java

# Difficultés rencontrées

## Les contraintes

Les très nombreuses contraintes lors de la réservation nous obligent à mettre en place énormément de filtres. Parfois ceux-ci empiètent sur les autres ce qui rend le programme très « error prone ».

En effet, parfois lorsque l’on règle un filtre, un ou plusieurs autres deviennent incorrects. Ce qui veut dire aussi que lorsqu’on modifie un élément, il faut dès lors re-tester tout le reste. Ce qui est quelque chose de relativement chronophage.

## « Database locked »

Parfois , sur la fin, lors de phase de debug, la base de données se verrouille. Le problème est que le « getStackTrace() » n’indique pas où se trouve l’erreur. Ce qui la rend extrêmement difficile à trouver.

## Git

En voulant modifier les notes de versions ainsi que le nom de mes « commits » avec une manipulation relativement simple, j’ai perdu toutes mes données.

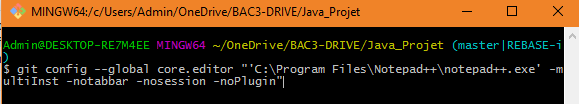


Figure 22 Git - notePad++

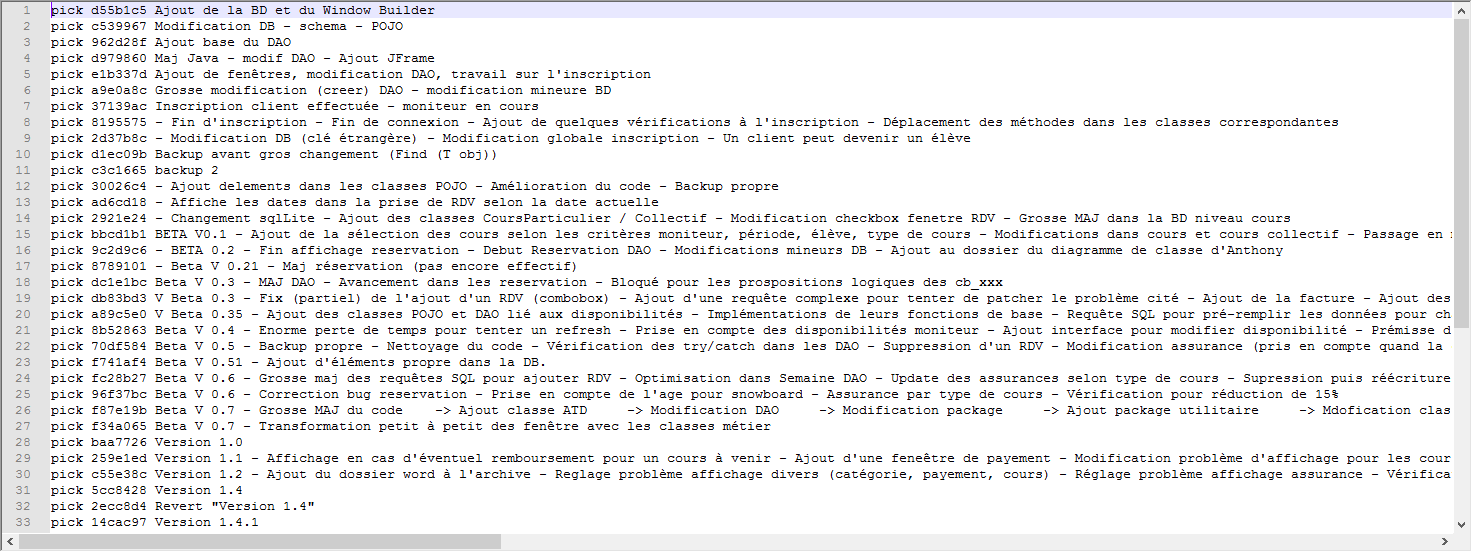


Figure 23 Git - Rebuild

J’ai voulu les modifier car je n’étais pas du tout satisfait de ma façon de noter mes commits, les trouvant trop différents les uns des autres.

J’aurais dû dès le début fixer un canevas et m’y fixer.

## Manque de temps

C’est bien le principal problème de ce projet. Après avoir passé un temps considérable dessus, je ne le trouve toujours pas aboutit.

# Note de mise à jour

## Alpha

### Version Alpha 0.0.1

* Nouveau dossier.
* Suppression du projet en console.

### Version Alpha 0.0.2

* Ajout de la base de données SQLITE.
* Ajout de WindowBuilder.

### Version Alpha 0.0.3

* Modification de la base de données.
* Modification du diagramme de classe.
* Ajout des classes POJO.

### Version Alpha 0.0.4

* Implémentation (partielle) du DAO.

### Version Alpha 0.0.5

* Ajout de JFrames.
* Modification du DAO.
* Modification des classes POJO.

### Version Alpha 0.0.6

* Ajout de JFrames.
* Modification du DAO.
* Implémentation partielle de l’inscription.

### Version Alpha 0.0.7

* Modification de l’inscription.
* Modifications mineures de la base de données.

### Version Alpha 0.0.8

* Fin de l’inscription pour le client.
* Implémentation partielle des classes liées au moniteur et son inscription.

### Version Alpha 0.0.9

* Fin d'inscription.
* Fin de connexion.
* Ajout de quelques vérifications à l'inscription.
* Déplacement des méthodes dans les classes correspondantes.

### Version Alpha 0.0.10

* Modification de la base de données (clé étrangère).
* Modification globale inscription.
* Un client peut maintenant devenir un élève via un bouton.

### Version Alpha 0.0.11

* Divers backups avant une grosse modification du code.

### Version Alpha 0.0.12

* Ajout d’éléments dans les classes POJO.
* Amélioration du code.
* Version propre.

### Version Alpha 0.0.13

* Afficher les dates dans la prise de RDV selon la date actuelle.

### Version Alpha 0.0.14

* Gros changement dans la base de données.
* Ajout classes CoursParticulier & CoursCollectif.
* Modification checkbox dans la fenêtre pour la réservation.

## Beta

### Version Beta 0.1.0

* Ajout de la sélection des cours selon les critères : moniteur, période, élève, type de cours.
* Modification dans coursCollectif et cours.
* Passage en version BETA.

### Version Beta 0.2.0

* Fin de l’implémentation de l’affichage pour les réservations.
* Implémentation de la réservation dans le DAO.
* Modifications mineures dans la base de données.

#### Version Beta 0.2.1

* Mise à jour de la réservation

### Version Beta 0.3.0

* Mise à jour du DAO
* Implémentation (en cours) des réservations

#### Version Beta 0.3.1

* Fix partiel de l’ajout d’une réservation (pour les comboBox).
* Ajout d'une requête complexe pour tenter de « patcher » le problème majeur de la 0.3.0.
* Ajout de la facture.
* Ajout des assurances.
* Ajout d'une requête pour mieux gérer les assurances.
* Modification un peu partout dans DAO.
* Modification de certaines tables dans la DB.
* Ajout de la table disponibilités pour moniteur.

#### Version Beta 0.3.2

* Ajout des classes POJO et DAO lié aux disponibilités.
* Implémentations de leurs fonctions de base.
* Requête SQL pour pré-remplir les données pour chaque nouveau moniteur.
* Prémisse d'un tableau pour gérer les dispos.
* Suppression de la classe utilitaire.
* Suppression des warnings dans une majorité des classes.

### Version Beta 0.4.0

* Énorme perte de temps pour tenter un « refresh ».
* Prise en compte des disponibilités moniteur.
* Ajout interface pour modifier disponibilité.
* Prémisse d'une suppression d'un cours.

### Version Beta 0.5.0

* Backup propre.
* Nettoyage du code.
* Vérification des try/catch dans les DAO.
* Suppression d'un RDV.
* Modification assurance (pris en compte quand la case est cochée).
* Suppression de beaucoup de warnings.

#### Version Beta 0.5.1

* Ajout de nouveaux enregistrements dans la base de données.

### Version Beta 0.6.0

* Grosse maj des requêtes SQL pour ajouter RDV.
* Optimisation dans Semaine DAO.
* Update des assurances selon types de cours.
* Suppression puis réécriture des cours dans la DB.
* Les élèves sont proposés selon leurs cours déjà pris.
* Les moniteurs sont proposés selon leurs disponibilités et leurs cours (implémentation partielle).

### Version Beta 0.7.0

* Correction bug pour effectuer une réservation.
* Prise en compte de l'âge pour snowboard.
* Implémentation de l’assurance selon le type de cours.
* Vérification pour la réduction du tarif de 15%.

### Version Beta 0.8.0

* Grosse MAJ du code suite à une discussion avec Kamal, Anthony, Vincent et Xavier.
* Ajout classe ATD.
* Modification DAO.
* Modification package.
* Ajout package utilitaire.
* Modification classe POJO.
* Cette version n’est pas encore compilable.

### Version Beta 0.9.0

* Transformation petit à petit des fenêtres avec les classes métier.
* Cette version n’est pas encore compilable.

## Version finale

### Version Finale 1.0.0

* Fin de l’implémentation des classes ATD (Access To Dao, ou **classe Métier**).
* Cette version est compilable et stable.

### Version Finale 1.1.0

* Affichage en cas d'éventuel remboursement pour un cours à venir.
* Ajout d'une fenêtre de payement.
* Modification problème d'affichage pour les cours.
* Ajout d'éléments dans le tableau cours.
* Les cours collectifs n'ont plus de limitation de temps.
* Ajout des jours pour les cours particuliers.
* Suppression de classes inutilisées dans le programme.
* Affichage uniquement des réservations à venir.
* Mise à jour du moniteur selon la semaine.
* Tentative d'ajout d'une progress bar, annulée car trop « time consuming ».

### Version finale 1.2.0

* Ajout du dossier Word à l'archive.
* Réglage problème affichage divers (catégorie, payement, cours).
* Réglage problème affichage assurance.
* Vérification compte – mot de passe bien rempli.

### Version finale 1.3.0

* Correction du bug « s'ajouter comme élève ».
* Retrait de certains champs pré-remplis pour faciliter le debug.
* Ajout de diverses vérifications.
* Ajout de JOptionPane un peu partout, à la place de simples messages.
* Ajout d'une checkbox, le moniteur peut dire s'il est disponible ou non pour prester les cours de type « particulier ».
* Ajout de messages d'indication pour faciliter l'utilisation de l'application.

### Version finale 1.4.0

* Ajout des listes (cours, réservation, élève).
* Correction du bug de suppression.
* Modification de la base de données.
* Ajout de commentaires dans le code.

### Version finale 1.5.0

* Implémentation du panier.
* Update des fonctions de payement.
* Vérification lors de l'update (payer tout le panier) que le nombre max d'élèves est respecté.
* Correction de bug d'affichage.
* Modification de la fenêtre « cours client ».
* Séparation de la fenêtre pour afficher les réservations entre moniteur et client.
* Suppression RDV selon n°semaine.

### Version finale 1.6.0

* Amélioration de la gestion des réservations.
* Ajout d’une comboBox pour le type de cours pour faciliter l’ajout d’une réservation.

### Version finale 1.7.0

* Amélioration globale du code.
* Ajout de lambda et autres filtres.

# Conclusion

Pour ma part, ce fut un des projets les plus difficiles que j’ai eu à faire lors de mes 3 années en informatique de gestion. Après avoir travaillé dessus d’arrache-pied, je ne suis toujours pas satisfait du résultat.

De plus, faire un projet seul après avoir fait ce genre de projet en groupe est moins intéressant. Il est plus difficile de palier à ses lacunes et il est aussi moins aisé de discuter du projet avec quelqu’un d’autre car il ne voit pas vraiment comme nous travaillons.

Néanmoins, je sens que grâce à celui-ci j’ai appris de mes erreurs. Que grâce à lui je ne perdrais plus autant de temps sur certaines parties, comme la mise en place d’un DAO.

# Table des **illustrations**

[Figure 1 Diagramme de classe 8](#_Toc467438934)

[Figure 2 Diagramme use case 9](file:///F:\BAC3\Java_Projet\DossierEcrit\Projet_java_ski.docx#_Toc467438935)

[Figure 3 Création d'un client 10](#_Toc467438936)

[Figure 4 Savoir si le moniteur a des réservations pour une certaine semaine 10](#_Toc467438937)

[Figure 5 Séquence : effectuer une réservation 11](file:///F:\BAC3\Java_Projet\DossierEcrit\Projet_java_ski.docx#_Toc467438938)

[Figure 6 Découpe du projet 13](file:///F:\BAC3\Java_Projet\DossierEcrit\Projet_java_ski.docx#_Toc467438939)

[Figure 7 Classe métier 13](file:///F:\BAC3\Java_Projet\DossierEcrit\Projet_java_ski.docx#_Toc467438940)

[Figure 8 DAO 13](file:///F:\BAC3\Java_Projet\DossierEcrit\Projet_java_ski.docx#_Toc467438941)

[Figure 9 POJO 14](file:///F:\BAC3\Java_Projet\DossierEcrit\Projet_java_ski.docx#_Toc467438942)

[Figure 10 JFrame 14](file:///F:\BAC3\Java_Projet\DossierEcrit\Projet_java_ski.docx#_Toc467438943)

[Figure 11 Utilitaire 15](file:///F:\BAC3\Java_Projet\DossierEcrit\Projet_java_ski.docx#_Toc467438944)

[Figure 12 Lambda, anyMatch 16](#_Toc467438945)

[Figure 13 Stream, filter & collect 16](#_Toc467438946)

[Figure 14 Operateur ternaire 17](#_Toc467438947)

[Figure 15 JTable 18](#_Toc467438948)

[Figure 16 SQLite Studio 20](#_Toc467438949)

[Figure 17 GitK 21](#_Toc467438950)

[Figure 18 Git Hub 21](#_Toc467438951)

[Figure 19 WindowBuilder 22](#_Toc467438952)

[Figure 20 JDatePicker 23](#_Toc467438953)

[Figure 21 JDatePicker en Java 23](#_Toc467438954)

[Figure 22 Git - notePad++ 25](#_Toc467438955)

[Figure 23 Git - Rebuild 25](#_Toc467438956)

# Annexes

1. Le diagramme avec une taille plus adéquate se trouve en annexe. [↑](#footnote-ref-1)
2. Car lorsque une personne a déjà une assurance pour un cours au matin/après-midi et qu’il réserve pour l’autre, il ne paye pas l’assurance même s’il la coche.

   Ensuite, il va update l’ancienne assurance. [↑](#footnote-ref-2)
3. **Plain Old Java Object** [↑](#footnote-ref-3)
4. https://github.com/germaaan/DSS/blob/master/Practica02/src/interfaz/ButtonColumn.java [↑](#footnote-ref-4)
5. http://stackoverflow.com/questions/17887927/adding-items-to-a-jcombobox [↑](#footnote-ref-5)
6. http://www.codejava.net/java-se/swing/how-to-use-jdatepicker-to-display-calendar-component [↑](#footnote-ref-6)