

21 NOVEMBRE 2016

# PROJET JAVA STATION DE SKI



ADRIEN MOUSTY  
3EME INFORMATIQUE DE GESTION  
HEPH Condorcet

# 1 TABLE DES MATIERES

2	Enoncé .....	4
2.1	Optionnellement : .....	6
2.2	Consignes .....	6
2.3	Contrainte .....	7
4	Schémas .....	8
4.1	Diagramme de classe .....	8
4.2	Diagramme use case .....	9
4.3	Quelques diagrammes de séquence .....	10
4.3.1	Création d'un client .....	10
4.3.2	Savoir si un moniteur donné a des réservations pour une semaine donnée 10	
4.3.3	Effectuer une réservation .....	11
5	Pour utiliser le programme .....	12
5.1	En tant que moniteur .....	12
5.2	En tant que client .....	12
6	Découpe du projet - packages .....	13
6.1	Access To Dao .....	13
6.2	Dao .....	13
6.3	Pojo .....	14
6.4	Fenêtre .....	14
6.5	Utilitaire .....	15
6.5.1	ButtonColumn .....	15
6.5.2	Comboltem .....	15
6.5.3	DateLabelFormatter .....	15
7	Bouts de code issu du projet .....	16
7.1	Lambda expressions .....	16
7.1.1	AnyMatch .....	16
7.1.2	Stream, filter & collect .....	16
7.2	Opérateur ternaire .....	17
7.3	JTable .....	18
7.3.1	Code couleur : .....	18
7.3.2	Insertion de boutons dans un JTable .....	19
8	Outils .....	20

8.1	SQLite studio .....	20
8.2	Git .....	21
8.3	Git Hub .....	21
8.4	WindowBuilder .....	22
8.5	JDatePicker.....	23
9	Difficultés rencontrées .....	24
9.1	Les contraintes .....	24
9.2	« Database locked ».....	24
9.3	Git .....	25
9.4	Manque de temps.....	25
10	Note de mise à jour .....	26
10.1	Alpha .....	26
10.1.1	Version Alpha 0.0.1 .....	26
10.1.2	Version Alpha 0.0.2 .....	26
10.1.3	Version Alpha 0.0.3 .....	26
10.1.4	Version Alpha 0.0.4 .....	26
10.1.5	Version Alpha 0.0.5 .....	26
10.1.6	Version Alpha 0.0.6 .....	26
10.1.7	Version Alpha 0.0.7 .....	26
10.1.8	Version Alpha 0.0.8 .....	26
10.1.9	Version Alpha 0.0.9 .....	26
10.1.10	Version Alpha 0.0.10 .....	27
10.1.11	Version Alpha 0.0.11 .....	27
10.1.12	Version Alpha 0.0.12 .....	27
10.1.13	Version Alpha 0.0.13 .....	27
10.1.14	Version Alpha 0.0.14 .....	27
10.2	Beta.....	28
10.2.1	Version Beta 0.1.0 .....	28
10.2.2	Version Beta 0.2.0 .....	28
10.2.3	Version Beta 0.3.0 .....	28
10.2.4	Version Beta 0.4.0 .....	28
10.2.5	Version Beta 0.5.0 .....	29
10.2.6	Version Beta 0.6.0 .....	29
10.2.7	Version Beta 0.7.0 .....	29
10.2.8	Version Beta 0.8.0 .....	29

10.2.9	Version Beta 0.9.0 .....	29
10.3	Version finale .....	30
10.3.1	Version Finale 1.0.0.....	30
10.3.2	Version Finale 1.1.0.....	30
10.3.3	Version finale 1.2.0 .....	30
10.3.4	Version finale 1.3.0 .....	30
10.3.5	Version finale 1.4.0 .....	30
10.3.6	Version finale 1.5.0 .....	31
10.3.7	Version finale 1.6.0 .....	31
10.3.8	Version finale 1.7.0 .....	31
11	Conclusion .....	32
12	Table des illustrations .....	33
13	Annexes .....	34

## 2 ÉNONCE

---

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)

## 2.1 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

## 2.2 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.

## 2.3 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.



## 4 SCHEMAS

### 4.1 DIAGRAMME DE CLASSE

DIAGRAMME DE CLASSE - ECOLE DE SKI

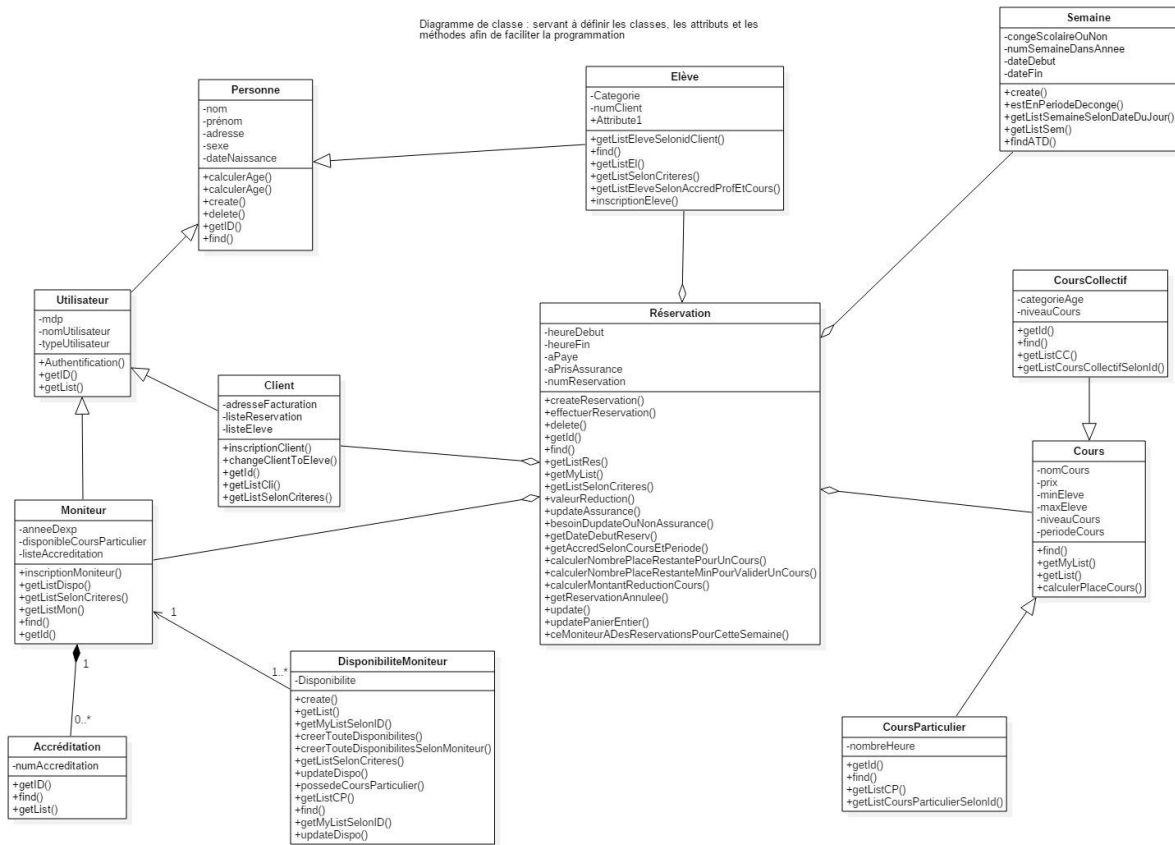


Figure 1 Diagramme de classe<sup>1</sup>

<sup>1</sup> Le diagramme avec une taille plus adéquate se trouve en annexe.

## 4.2 DIAGRAMME USE CASE

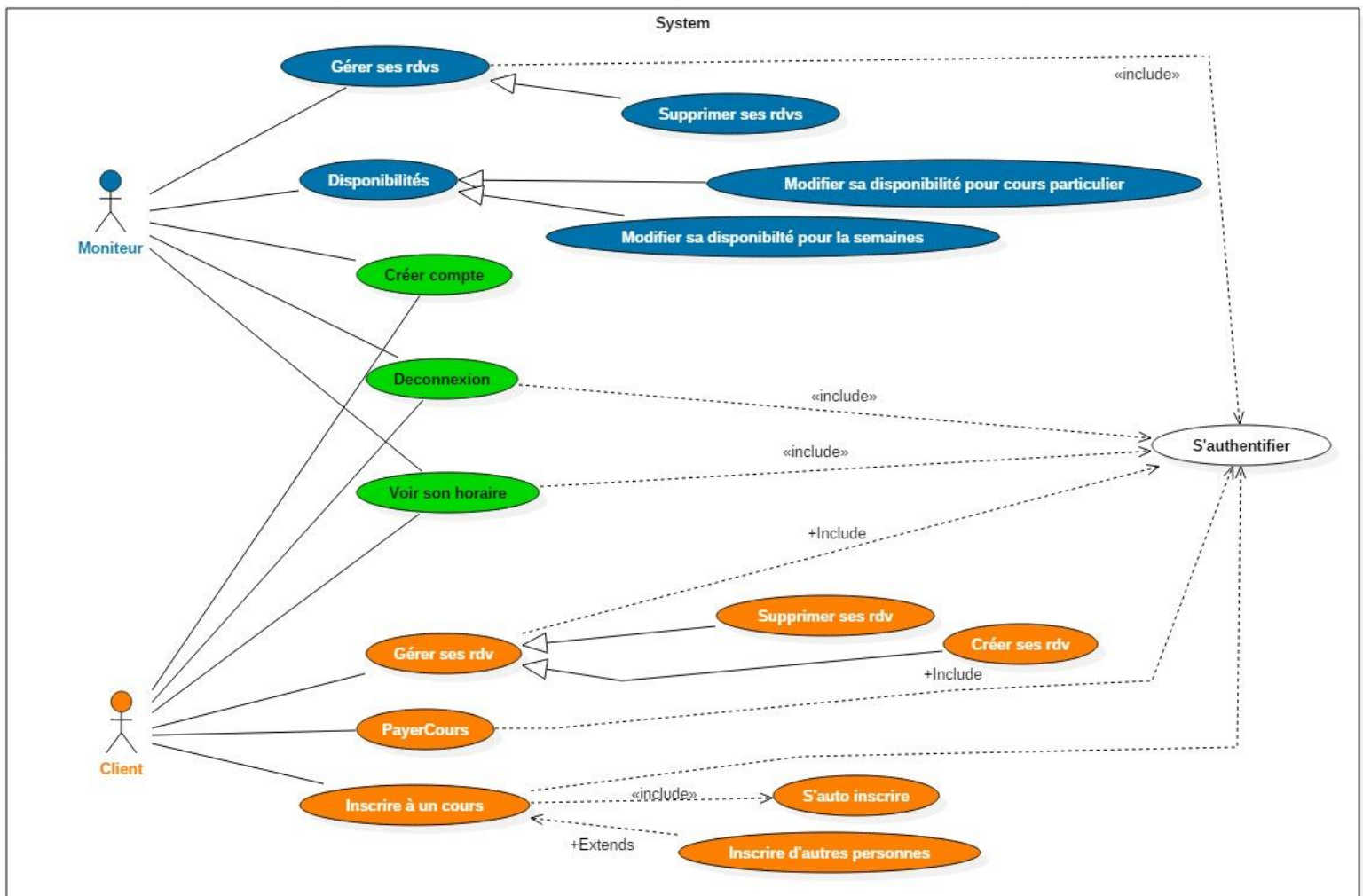


Figure 2 Diagramme use case

## 4.3 QUELQUES DIAGRAMMES DE SEQUENCE

### 4.3.1 Création d'un client

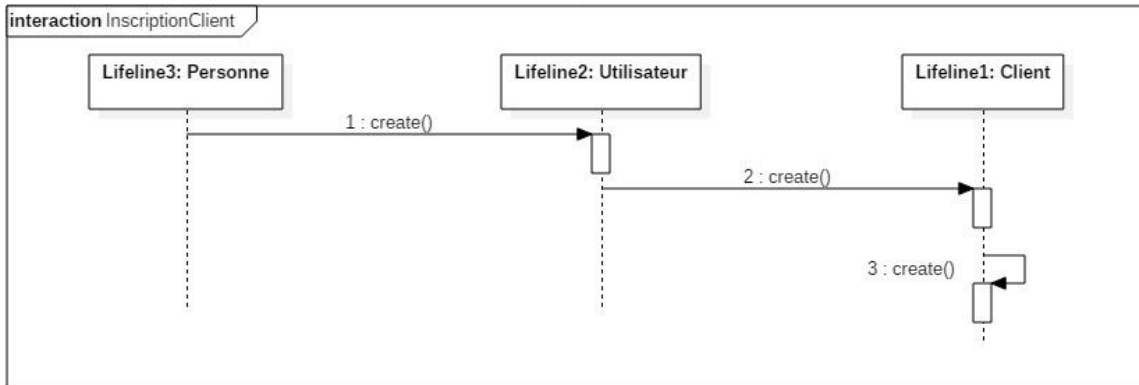


Figure 3 Création d'un client

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

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

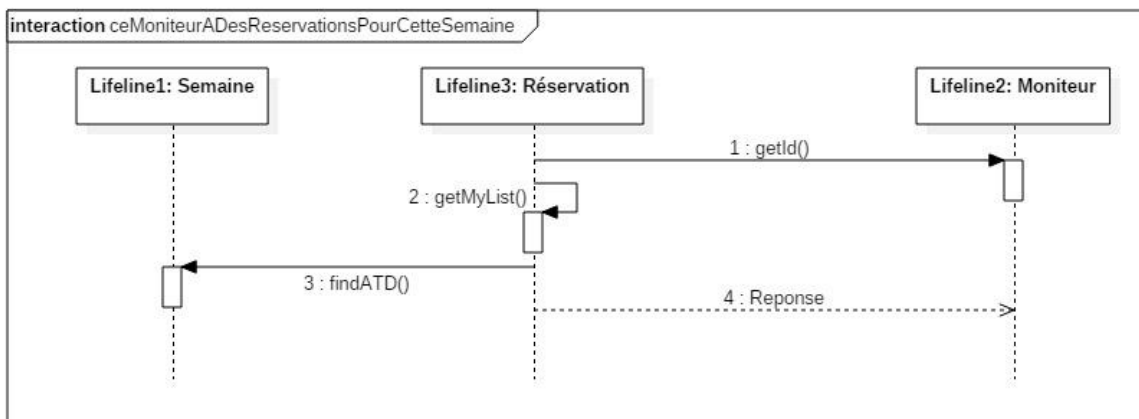


Figure 4 Savoir si le moniteur a des réservations pour une certaine semaine

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.

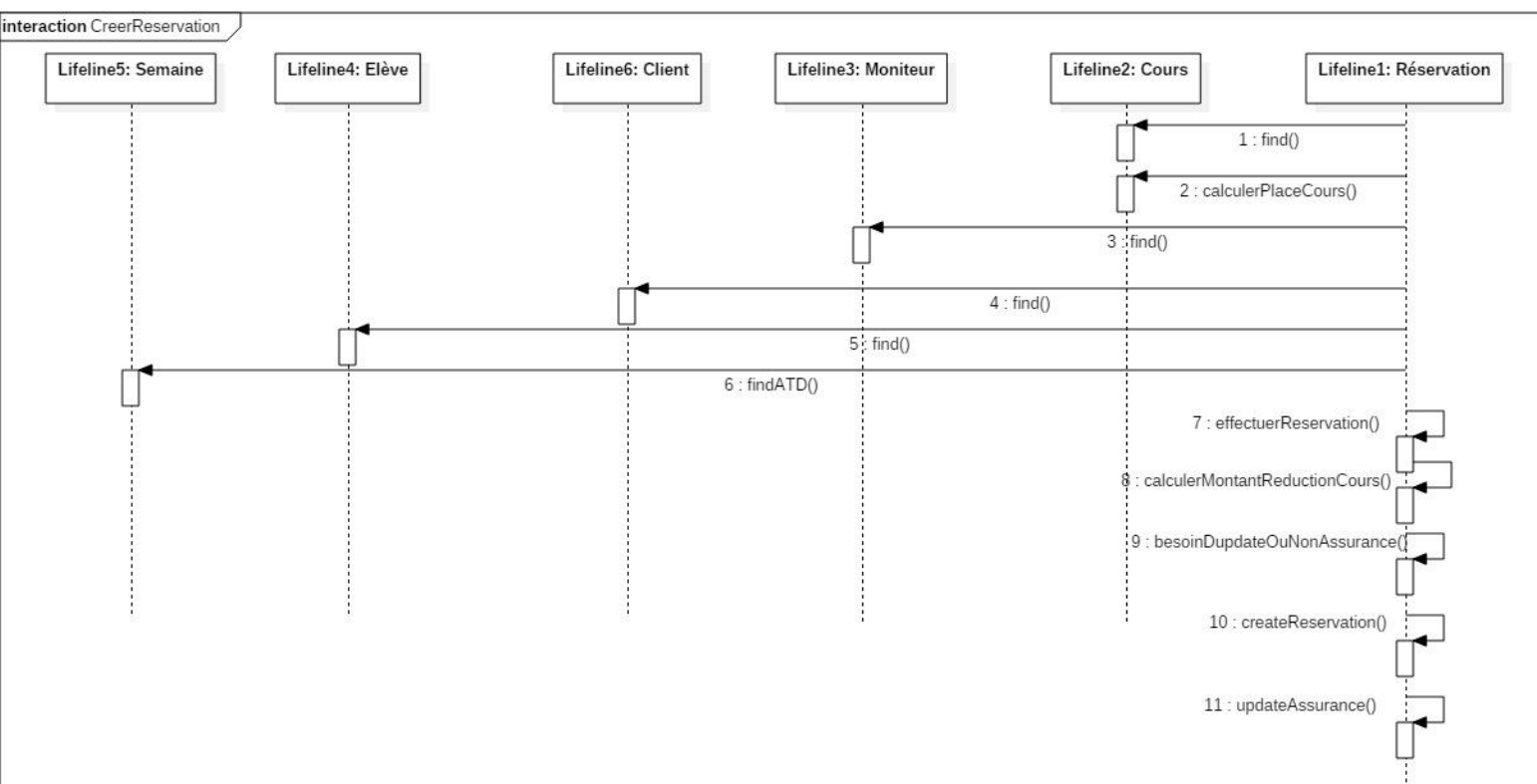


Figure 5 Séquence : effectuer une réservation

### 4.3.3 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<sup>2</sup> assurance, on le fait.

<sup>2</sup> 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.

## 5 POUR UTILISER LE PROGRAMME

---

### 5.1 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

### 5.2 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.

## 6 DECOUPE 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.

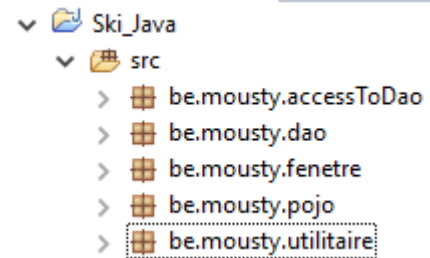
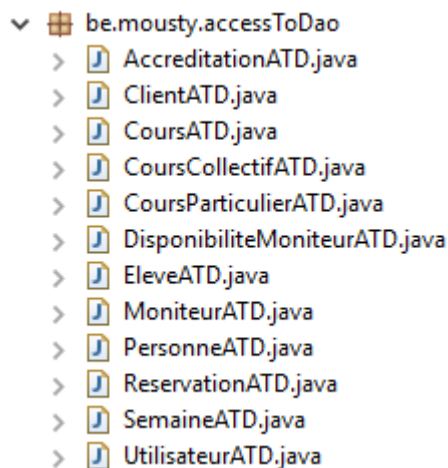


Figure 6 Découpe du projet

### 6.1 ACCESS TO DAO

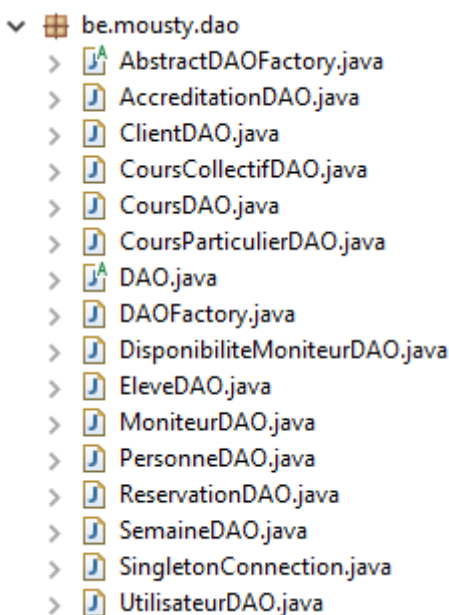


Les classes ATD sont en fait les classes métier.

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 7 Classe métier

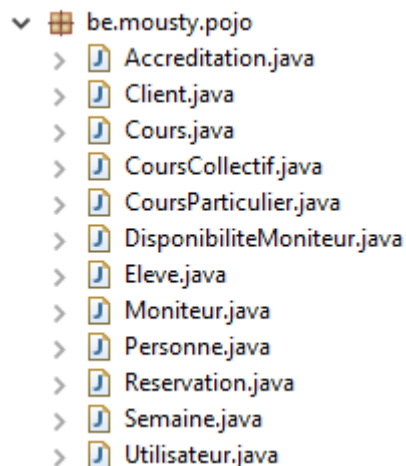
### 6.2 DAO



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 8 DAO

### 6.3 POJO<sup>3</sup>



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.

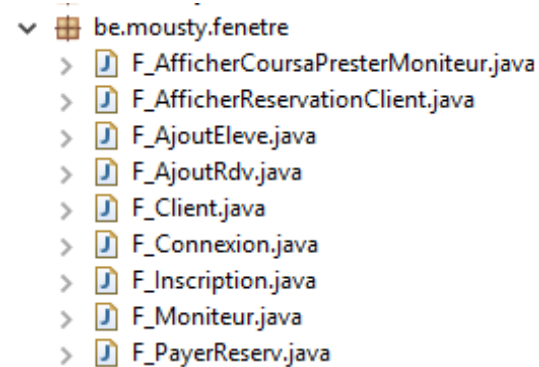
Ils ne possèdent **aucune méthode** ni d'autres constructeurs qu'un constructeur par défaut.

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 9 POJO

### 6.4 FENETRE



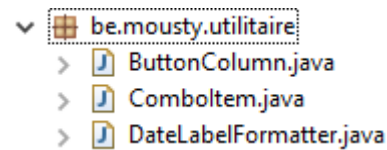
Les fenêtres sont la partie visible du programme. Elles sont générées grâce à WindowsBuilder puis modelée, un peu comme nous avons fait avec Blend lors de projets précédents en C#.

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.

Figure 10 JFrame

<sup>3</sup> Plain Old Java Object

## 6.5 UTILITAIRE



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

Figure 11 Utilitaire

### 6.5.1 ButtonColumn<sup>4</sup>

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

### 6.5.2 ComboItem<sup>5</sup>

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

```

5 //CLASSE UTILISEE POUR RECUPERER LID DE LA PERSONNE
6 public class ComboItem
7 {
8     private String key;
9     private int value;
10    private Date valueDate;
11    private String valueString;
12
13    public ComboItem(String key, int value)
14    {
15        this.key = key;
16        this.value = value;
17    }
18
19    public ComboItem(String key, Date valueDate)
20    {
21        this.key = key;
22        this.valueDate = valueDate;
23    }
24
25    public ComboItem(String key, String valueString)
26    {
27        this.key = key;
28        this.valueString = valueString;
29    }
30
31    @Override
32    public String toString() { return key; }
33    public String getKey() { return key; }
34    public int getValue() { return value; }
35    public Date getValueDate() { return valueDate; }
36    public String getValueString() { return valueString; }
37 }

```

### 6.5.3 DateLabelFormatter<sup>6</sup>

Lié au fonctionnement de JDatePicker.

<sup>4</sup> <https://github.com/germaan/DSS/blob/master/Practica02/src/interfaz/ButtonColumn.java>

<sup>5</sup> <http://stackoverflow.com/questions/17887927/adding-items-to-a-jcombobox>

<sup>6</sup> <http://www.codejava.net/java-se/swing/how-to-use-jdatepicker-to-display-calendar-component>



## 7 BOUTS DE CODE ISSU DU PROJET

### 7.1 LAMBDA EXPRESSIONS

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

#### 7.1.1 AnyMatch

```
/**
 * Objectif : Savoir si un moniteur a encore des cours particuliers à prester.
 * @param numMoniteur
 * @return
 */
public boolean ceMoniteurDoitPresterCoursParticulier(int numMoniteur){
    return this.getMyList(numMoniteur).stream().anyMatch(t -> t.getCours().getPrix() < 90);
}
```

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.

#### 7.1.2 Stream, filter & collect

```
/**
 * Objectif : Il faut d'abord savoir s'il faut update l'assurance / faire payer SANS update car on peut annuler la réservation.
 * @param numEleve
 * @param numSemaine
 * @param periode
 * @return un booléen pour savoir s'il faut update ou non.
 */
public boolean besoinDupdateOuNonAssurance (int numEleve, int numSemaine, String periode) {
    // On ne s'occupe que des cours collectifs, donc pas besoin des autres périodes.
    String oppose = periode.equals("09-12") ? "14-17" : periode.equals("14-17") ? "09-12" : "";
    // Lambda expression avec filtre permettant de retourner 0 ou une occurrence dans la liste.
    ArrayList<Reservation> listeFull = getListRes();
    ArrayList<Reservation> listTriee = listeFull.stream()
        .filter(p -> p.getEleve().getNumEleve() == numEleve)
        .filter(p -> p.getSemaine().getNumSemaine() == numSemaine)
        .filter(p -> p.getCours().getPeriodeCours().equals(oppose))
        .filter(p -> p.getAuneAssurance() == true)
        .collect(Collectors.toCollection(ArrayList::new));
    // s'il trouve un élément, il update l'assurance.
    return !listTriee.isEmpty();
}
//return ReservationDAO.besoinDupdateOuNonAssurance(numEleve, numSemaine, periode);
}
```

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 pas 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.

## 7.2 OPERATEUR TERNAIRE

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

Dans l'exemple ci-dessous elle est imbriquée.

```
int prixAssurance = assurance ? (RATD.besoinUpdateOuNonAssurance(numEleve, numSemaine, periode) ? 0 : 15) : 0;
```

Figure 14 Opérateur ternaire

## 7.3 JTABLE

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

Mousty, Rue Hebert Hoover

Cours payés

N°	Jour début	jour fin	heure deb...	heure fin	Libellé	Elèves min	Elèves max	Eleves actuel...	Moniteur	Eleve	type	Prix	Action
31	2016-12-05	2016-12-11	09h	12h	Ski	6	10	3	MONITEUR 1	STEFANO N...	Collectif	130.0€	Annuler
71	2016-12-05	2016-12-11	09h	12h	Ski	6	10	3	MONITEUR 1	STEFANO St...	Collectif	130.0€	Annuler
74	2016-12-05	2016-12-11	09h	12h	Ski	6	10	3	MONITEUR 1	COLIN Jean	Collectif	130.0€	Annuler
81	2016-12-05	2016-12-11	09h	12h	Snowboard	5	8	2	MONITEUR 2	COLIN Franç...	Collectif	130.0€	Annuler
82	2016-12-05	2016-12-11	09h	12h	Snowboard	5	8	2	MONITEUR 2	MOUSTY Emry	Collectif	130.0€	Annuler
77	2016-12-05	2016-12-05	13h	14h	Ski	1	4	1	MONITEUR 1	MOUSTY Adr...	Particulier	50.0€	Annuler
78	2016-12-05	2016-12-05	13h	14h	Snowboard	1	4	1	MONITEUR 2	STEFANO N...	Particulier	50.0€	Annuler
70	2016-12-05	2016-12-11	14h	17h	Ski	6	10	2	MONITEUR 1	MOUSTY Adr...	Collectif	130.0€	Annuler
72	2016-12-05	2016-12-11	14h	17h	Ski	6	10	2	MONITEUR 1	STEFANO N...	Collectif	130.0€	Annuler
79	2016-12-05	2016-12-11	14h	17h	Snowboard	5	8	2	MONITEUR 2	STEFANO St...	Collectif	130.0€	Annuler
83	2016-12-05	2016-12-11	14h	17h	Snowboard	5	8	2	MONITEUR 2	CAREY Mariah	Collectif	130.0€	Annuler

Cours en attente de paiement

N°	Jour début	jour fin	heure debut	heure fin	Libellé	Elèves min	Elèves max	Eleves actuel...	Moniteur	Eleve	type	Prix	Action
80	2016-12-05	2016-12-11	09h	12h	Snowboard	5	8	2	MONITEUR 2	MOUSTY Adr...	Collectif	130.0€	Payer

Récapitulatif

Type	Prix cours	Assurances	Reduction	Prix total
Payé	1270	105	78.0	1297.0
Non payé	130	0	39.0	91.0
Total	1400	105	117.0	1388.0

Payer mon panier

Retour

Figure 15 JTable

### 7.3.1 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 paiement.

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

```
//changer la couleur
table.setDefaultRenderer(Object.class, new DefaultTableCellRenderer() {

    private static final long serialVersionUID = 1L;

    @Override
    public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected,
        boolean hasFocus, int row, int col) {
        super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, col);

        boolean estValide = false;
        // On va calculer la place des cours, ensuite on va voir si la palce correspond au maximum possible.
        // Si oui, on colorie en vert, si non on colorie en rouge.
        RATD = listReserv.get(row);
        //CoursATD CATD = RATD.getCours();

        if (RATD.calculerNombrePlaceRestanteMinPourValiderUnCours() == 0)
            estValide = true;
        if (estValide) { setBackground(new Color(102, 255, 51)); }
        else { setBackground(new Color(255,255,153)); }
        return this;
    }
});
```

Pour générer des couleurs en hexadécimal, j'ai utilisé ce site :

[http://www.toutimages.com/generateur\\_nr\\_2.htm](http://www.toutimages.com/generateur_nr_2.htm)

### 7.3.2 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 :

```
new ButtonColumn(table, changerValeur, headerInfoCours.length-1);
```

Les paramètres sont :

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

```
// Action de modification
final Action changerValeur = new AbstractAction()
{
    private static final long serialVersionUID = 1L;

    @Override
    public void actionPerformed(ActionEvent e)
    {
        JTable mytableClicked = (JTable)e.getSource();
        Object numRes = mytableClicked.getModel().getValueAt(mytableClicked.getSelectedRow(), 0);
        //if(ReservationDAO.delete(ReservationDAO.find(Integer.parseInt(numRes.toString()))))
        if(RATD.delete(RATD.find(Integer.parseInt(numRes.toString()))))
            JOptionPane.showMessageDialog(contentPane, "Cours supprimé.");
        else
            JOptionPane.showMessageDialog(contentPane, "Une erreur est intervenue, le cours n'est pas supprimé.");
        setVisible(false);
        F_AfficherReservationClient frameA = new F_AfficherReservationClient(idPersonne);
        frameA.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
        frameA.setVisible(true);
    }
};
```

- ➔ 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 1<sup>er</sup> 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 réaffiche la fenêtre afin d'actualiser les valeurs.
3. La colonne sur laquelle appliquer le bouton.

## 8 OUTILS

### 8.1 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.

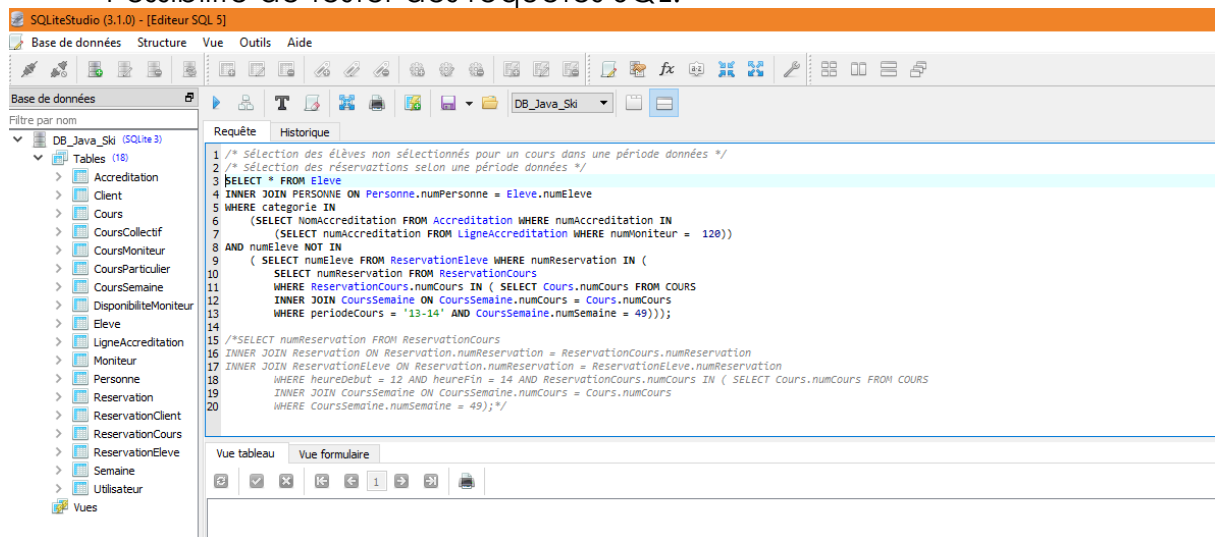



Figure 16 SQLite Studio

Afin d'utiliser ma base de données en JAVA, j'ai dû installer une librairie externe :

 [sqlite-jdbc-3.14.2.1.jar](#)

17/10/2016 14:55

Executable Jar File

4 221 Ko

## 8.2 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 ».

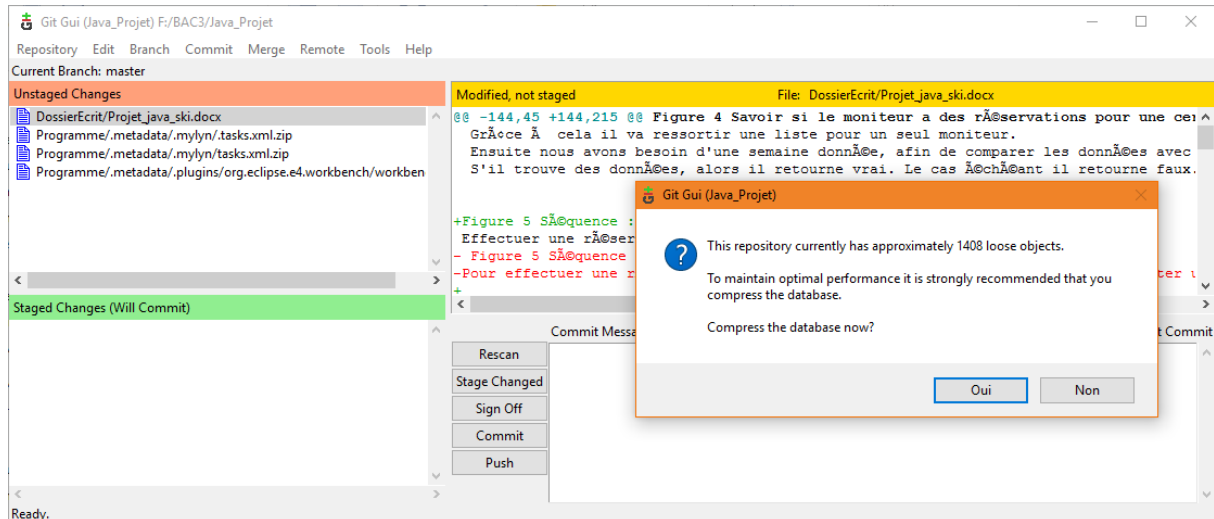


Figure 17 GitK

## 8.3 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.

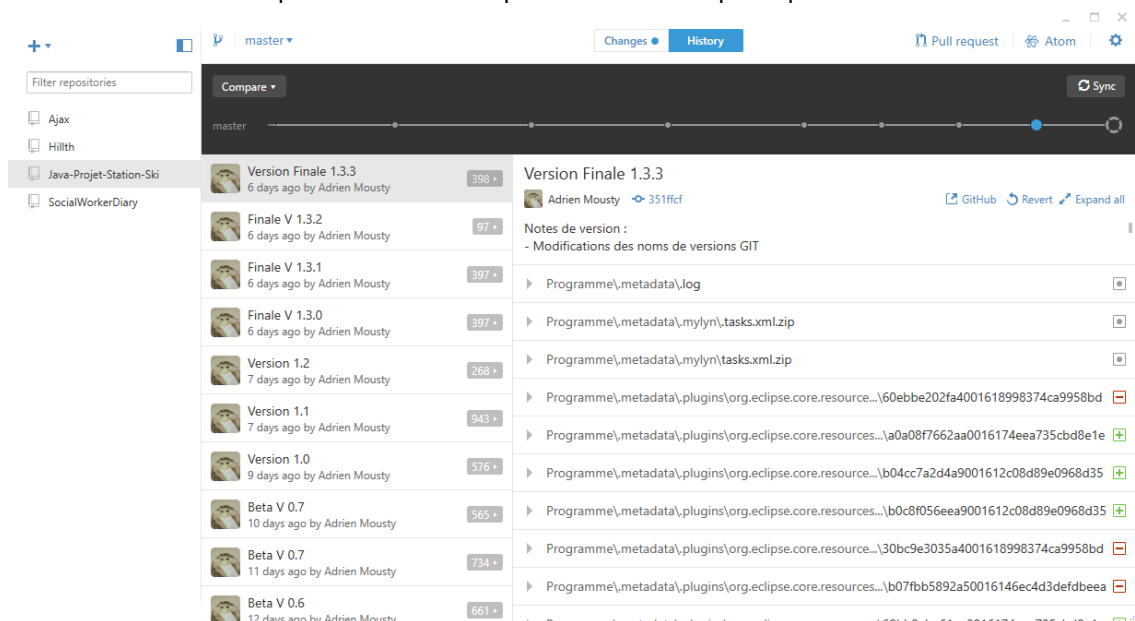


Figure 18 Git Hub

## 8.4 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 ».

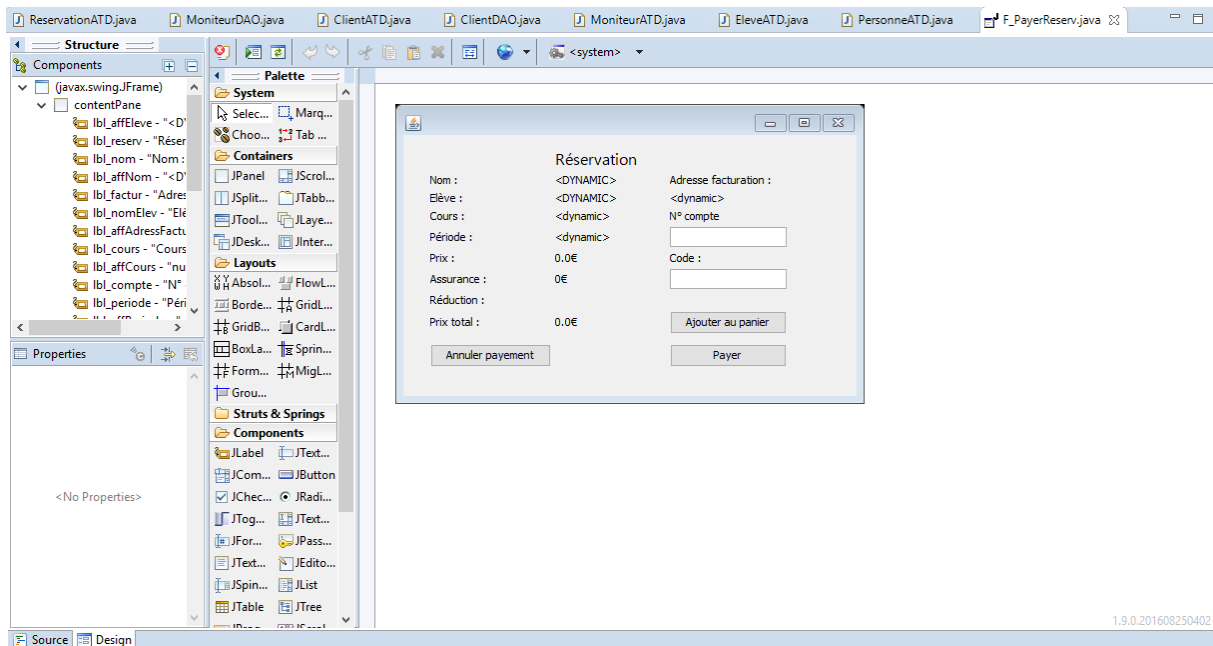


Figure 19 WindowBuilder

## 8.5 JDATEPICKER

jdatepicker-1.3.4.jar 14/11/2016 17:45 Executable Jar File 42 Ko

Un bon datePicker faisant défaut au Java, j'ai cherché sur internet quelque chose « d'user friendly ». Après un peu de recherche, JDatePicker sort du lot. C'est un outil vraiment très intéressant bénéficiant d'un certain support.

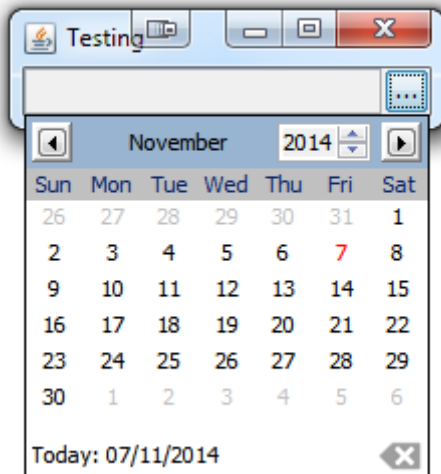


Figure 20 JDatePicker

Il s'implémente comme suit :

```
// DatePicker
SqlDateModel model = new SqlDateModel();
Properties p = new Properties();
p.put("text.today", "Today");
p.put("text.month", "Month");
p.put("text.year", "Year");
JDatePanelImpl datePanel = new JDatePanelImpl(model, p);
JDatePickerImpl datePicker = new JDatePickerImpl(datePanel, new DateLabelFormatter());
datePicker.getJFormattedTextField().setToolTipText("Date de naissance");

datePicker.setBounds(10, 146, 130, 23);
contentPane.add(datePicker);
```

Figure 21 JDatePicker en Java



## 9 DIFFICULTES RENCONTREES

---

### 9.1 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.

### 9.2 « 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.

### 9.3 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.

```

MINGW64:/c:/Users/Admin/OneDrive/BAC3-DRIVE/Java_Projet
Admin@DESKTOP-RE7M4EE MINGW64 ~/OneDrive/BAC3-DRIVE/Java_Projet (master|REBASE-i)
$ git config --global core.editor "'C:\Program Files\Notepad++\notepad++.exe' -m
ultiInst -notabbar -nosession -noPlugin"
  
```

Figure 22 Git - notepad++

```

1 pick d55b1c5 Ajout de la BD et du Window Builder
2 pick c539967 Modification DB - schema - POJO
3 pick 962d28f Ajout base du DAO
4 pick d979860 Maj Java - modif DAO - Ajout JFrame
5 pick e1b337d Ajout de fenêtres, modification DAO, travail sur l'inscription
6 pick a9e9a5c Grosse modification (creer) DAO - modification mineure BD
7 pick 37139ac Inscription client effectuée - moniteur en cours
8 pick 8195575 - Fin d'inscription - Fin de connexion - Ajout de quelques vérifications à l'inscription - Déplacement des méthodes dans les classes correspondantes
9 pick 2d37b8c - Modification DB (clé étrangère) - Modification globale inscription - Un client peut devenir un élève
10 pick d1ec09b Backup avant gros changement (Find (T obj))
11 pick c3c1665 backup 2
12 pick 30026c4 - Ajout delements dans les classes POJO - Amélioration du code - Backup propre
13 pick ad6dd18 - Affiche les dates dans la prise de RDV selon la date actuelle
14 pick 2921e24 - Changement sqlLite - Ajout des classes CoursParticulier / Collectif - Modification checkbox fenetre RDV - Grosse MAJ dans la BD niveau cours
15 pick bb0d1b1 BETA V0.1 - Ajout de la sélection des cours selon les critères moniteur, période, élève, type de cours - Modifications dans cours et cours collectif - Passage en
16 pick 9c2d9c6 - BETA 0.2 - Fin affichage reservation - Debut Reservation DAO - Modifications mineurs DB - Ajout au dossier du diagramme de classe d'Anthony
17 pick 8789101 - Beta V 0.21 - Maj reservation (pas encore effectuée)
18 pick d0e1b0c Beta V 0.3 - MAJ DAO - Avancement dans les reservation - Bloqué pour les propositions logiques des cb_xxx
19 pick db83bd3 V Beta 0.3 - Fix (partiel) de l'ajout d'un RDV (combobox) - Ajout d'une requête complexe pour tenter de patcher le problème cité - Ajout de la facture - Ajout des
20 pick a89c5e0 V Beta 0.35 - Ajout des classes POJO et DAO lié aux disponibilités - Implémentations de leurs fonctions de base - Requête SQL pour pré-remplir les données pour ch
21 pick 8b52863 Beta V 0.4 - Enorme perte de temps pour tenter un refresh - Prise en compte des disponibilités moniteur - Ajout interface pour modifier disponibilité - Prémisse d
22 pick 7dd5f84 Beta V 0.5 - Backup propre - Nettoyage du code - Vérification des try/catch dans les DAO - Suppression d'un RDV - Modification assurance (pris en compte quand la
23 pick f741a14 Beta V 0.51 - Ajout d'éléments propre dans la DB
24 pick f228b27 Beta V 0.6 - Grosse maj des requêtes SQL pour ajouter RDV - Optimisation dans Semaine DAO - Update des assurances selon type de cours - Suppression puis réécriture
25 pick 96f37bc Beta V 0.6 - Correction bug reservation - Prise en compte de l'age pour snowboard - Assurance par type de cours - Vérification pour réduction de 15%
26 pick f87e19b Beta V 0.7 - Grosse MAJ du code -> Ajout classe ATD -> Modification DAO -> Modification package -> Ajout package utilitaire -> Modification clas
27 pick f34a065 Beta V 0.7 - Transformation petit à petit des fenêtres avec les classes métier
28 pick ba8f726 Version 1.0
29 pick 259e1ed Version 1.1 - Affichage en cas d'éventuel remboursement pour un cours à venir - Ajout d'une fenêtre de paiement - Modification problème d'affichage pour les cour
30 pick c55e38c Version 1.2 - Ajout du dossier word à l'archive - Reglage problème affichage divers (catégorie, paiement, cours) - Réglage problème affichage assurance - Vérifica
31 pick 5cc8428 Version 1.4
32 pick 2ecc8d4 Revert "Version 1.4"
33 pick 14cac97 Version 1.4.1
  
```

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.

### 9.4 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.

## 10 NOTE DE MISE A JOUR

---

### 10.1 ALPHA

#### 10.1.1 Version Alpha 0.0.1

- Nouveau dossier.
- Suppression du projet en console.

#### 10.1.2 Version Alpha 0.0.2

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

#### 10.1.3 Version Alpha 0.0.3

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

#### 10.1.4 Version Alpha 0.0.4

- Implémentation (partielle) du DAO.

#### 10.1.5 Version Alpha 0.0.5

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

#### 10.1.6 Version Alpha 0.0.6

- Ajout de JFrames.
- Modification du DAO.
- Implémentation partielle de l'inscription.

#### 10.1.7 Version Alpha 0.0.7

- Modification de l'inscription.
- Modifications mineures de la base de données.

#### 10.1.8 Version Alpha 0.0.8

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

#### 10.1.9 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.

#### **10.1.10 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.

#### **10.1.11 Version Alpha 0.0.11**

- Divers backups avant une grosse modification du code.

#### **10.1.12 Version Alpha 0.0.12**

- Ajout d'éléments dans les classes POJO.
- Amélioration du code.
- Version propre.

#### **10.1.13 Version Alpha 0.0.13**

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

#### **10.1.14 Version Alpha 0.0.14**

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

## 10.2 BETA

### 10.2.1 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.

### 10.2.2 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.

#### 10.2.2.1 Version Beta 0.2.1

- Mise à jour de la réservation

### 10.2.3 Version Beta 0.3.0

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

#### 10.2.3.1 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.

#### 10.2.3.2 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.

### 10.2.4 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.

### 10.2.5 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.

#### 10.2.5.1 Version Beta 0.5.1

- Ajout de nouveaux enregistrements dans la base de données.

### 10.2.6 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).

### 10.2.7 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%.

### 10.2.8 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.

### 10.2.9 Version Beta 0.9.0

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

## 10.3 VERSION FINALE

### 10.3.1 Version Finale 1.0.0

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

### 10.3.2 Version Finale 1.1.0

- Affichage en cas d'éventuel remboursement pour un cours à venir.
- Ajout d'une fenêtre de paiement.
- 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 ».

### 10.3.3 Version finale 1.2.0

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

### 10.3.4 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.

### 10.3.5 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.

#### **10.3.6 Version finale 1.5.0**

- Implémentation du panier.
- Update des fonctions de paiement.
- 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.

#### **10.3.7 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.

#### **10.3.8 Version finale 1.7.0**

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



## 11 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.

## 12TABLE DES illustrations

Figure 1 Diagramme de classe .....	8
Figure 2 Diagramme use case .....	9
Figure 3 Création d'un client .....	10
Figure 4 Savoir si le moniteur a des réservations pour une certaine semaine ..	10
Figure 5 Séquence : effectuer une réservation .....	11
Figure 6 Découpe du projet .....	13
Figure 7 Classe métier .....	13
Figure 8 DAO .....	13
Figure 9 POJO .....	14
Figure 10 JFrame.....	14
Figure 11 Utilitaire .....	15
Figure 12 Lambda, anyMatch.....	16
Figure 13 Stream, filter & collect .....	16
Figure 14 Operateur ternaire .....	17
Figure 15 JTable .....	18
Figure 16 SQLite Studio .....	20
Figure 17 GitK .....	21
Figure 18 Git Hub .....	21
Figure 19 WindowBuilder .....	22
Figure 20 JDatePicker .....	23
Figure 21 JDatePicker en Java.....	23
Figure 22 Git - notePad++ .....	25
Figure 23 Git - Rebuild.....	25

# 13 ANNEXES

## DIAGRAMME DE CLASSE - ECOLE DE SKI

Diagramme de classe : servant à définir les classes, les attributs et les méthodes afin de faciliter la programmation

