LIÈGE, JEUDI 21 JUIN 2018

DOSSIER TECHNIQUE

LABORATOIRE JAVA « LE GOURMET AUDACIEUX »

SCHÉMAS UML, EXPLICATION DU CODE POUR LES PARTIES DEMANDÉES, ÉTAPES DE DÉVELOPPEMENT ET DU TEST DE LA BIBLIOTHÈQUE, ET REPRÉSENTATION DES FICHIERS POUR LE LABORATOIRE DE JAVA DE L'UNITÉ D'ENSEIGNEMENT « DÉVELOPPEMENT ORIENTÉ OBJETS ET MULTITÂCHE » POUR L'ACTIVITÉ D'APPRENTISSAGE « PROGRAMMATION ORIENTÉE OBJET UNIX ET WINDOWS - JAVA ».

William Gathoye

1. DIAGRAMME UML DES CLASSES DE DONNÉES

Pour cette première partie d'évaluation, l'ensemble des classes de données, souvent appelées « data objects » dans le jargon du design logiciel, sont celles telles que demandées dans l'énoncé. Elles peuvent être considérées comme le modèle dans le cadre d'un patron MVC, même si toutefois ce patron de conception (design pattern) n'a pas été utilisé dans le logiciel.

Afin de les rendre utilisables dans l'application, quelques latitudes ont toutefois dû être apportées à ces classes.

1.1. RATTACHEMENT À UNE TABLE

Tout d'abord, afin de sauvegarder les données d'achats de boissons, nous avons dû considérer qu'un ensemble de boissons commandées au bar sont rattachées à une table, comme s'il s'agissait d'une simple commande de repas. Nous avons donc fait hériter notre classe « Drink » de « Plate » de façon à pourvoir facilement sérialiser par la suite les commandes (« PlateOrder ») qui contiennent une table de plats (« Plate »). Ce choix est justifié de plusieurs manières :

– Les boissons sont dans tous les cas, conformément à l'interface graphique, rattachées à une table et ajoutées à l'addition d'une table en particulier.

– Il n'est pas rare au sein d'une brasserie de demander à rajouter des boissons sur le compte. « Hé, Michel, la tournée est pour mon compte ! ». Ici le compte est simplement représenté par une table. Ce choix peut être audacieux, dans la mesure où il peut monopolise une table dans le cas où le client ne consomme pas de repas et se trouve simplement au bar. La notion de compte client pourrait donc être instaurée en guise d'amélioration au projet.

– Un plat dispose d'une catégorie « Drink » et même « Alcohol » ce qui conforte notre choix.

Dans les plats servis, le logiciel est en mesure de détecter si la boisson a été ajoutée dans le cadre d'un repas ou non. Cette détection pourrait permettre l'implémentation d'un droit de bouchon en guide d'éventuelle amélioration. En effet, les boissons, plus particulièrement les bouteilles, servies dans le cadre d'un repas sont souvent plus chères que dans un bar ou à fortiori dans une grande surface.

1.2. CLASSE « WAITER »

Toujours pour cette première partie d'évaluation, la classe « Waiter » représentant la personne qui sert le client, et qui utilise l'application salle n'est pas utilisée. Uniquement le nom du serveur est utilisé et rattaché à une table.

1.3. OPÉRATIONS SUR NOMBRES À VIRGULE FLOTTANTE

Les opérations sur des nombres à virgule flottante ont été remplacées par l'utilisation de la classe ad hoc à savoir « Big Decimal ». De part les approximations et notamment la définition de l'emplacement de la virgule à l'aide de la partie réservée à l'exposant (pour rappel : bit de signe, mantisse et exposant), il arrive que la machine n'ait pas anticipé le fait de réserver des chiffres après la virgule de façon suffisante (déterminé par la grandeur de l'exposant). Lorsque la machine doit alors additionner un autre nombre flottant, il arrive qu'il y ait des décalages de bits rendus nécessaires et que le résultat ne soit pas correct.

Lorsqu'une opération financière est requise, il est de l'obligation du développeur de l'application ou de son commanditaire d'appliquer certaines règles notamment en matière d'arrondi. Il s'agit d'une obligation légale. Cette classe tient compte de ces spécificités.[1]

Dans le cadre d'application d'ingénierie ou à gestion financière, utiliser des entiers non signés et enregistrer le signe et la virgule dans une autre colonne de la base de données est également une opération courante. Ceci revient donc à gérer ses propres nombres en virgule flottante, sans utiliser le type natif imposé par la plateforme, ici le JRE Java.

1.4. CHAMP SUPPLÉMENTAIRE DANS « PLATEORDER »

Nous avons décidé d'ajouter un champ supplémentaire dans les commandes (classe « PlateOrder ») de façon à ne pas devoir recalculer la valeur d'un plat lorsqu'on affiche à nouveau une table sélectionnée.

En plus, ce choix permettrait de faire varier le prix des plats en fonction du temps, sans que le prix des précédents achats en soit affecté. Le prix est de cette manière figé pour une commande. Dans le cadre d'opérations comptables, il s'agit également d'une obligation que de stocker le prix du produit pour une commande.

1.5. LÉGENDE DU SCHÉMA UML

– La classe « Plate » implémente l'interface « Service », ce lien est marqué par un trait discontinu se terminant par une flèche à tête fermée blanche.

– La classe « Plate » est une classe abstraite (elle ne peut donc pas être instanciée directement). Elle est représentée par une mise en forme en italique.

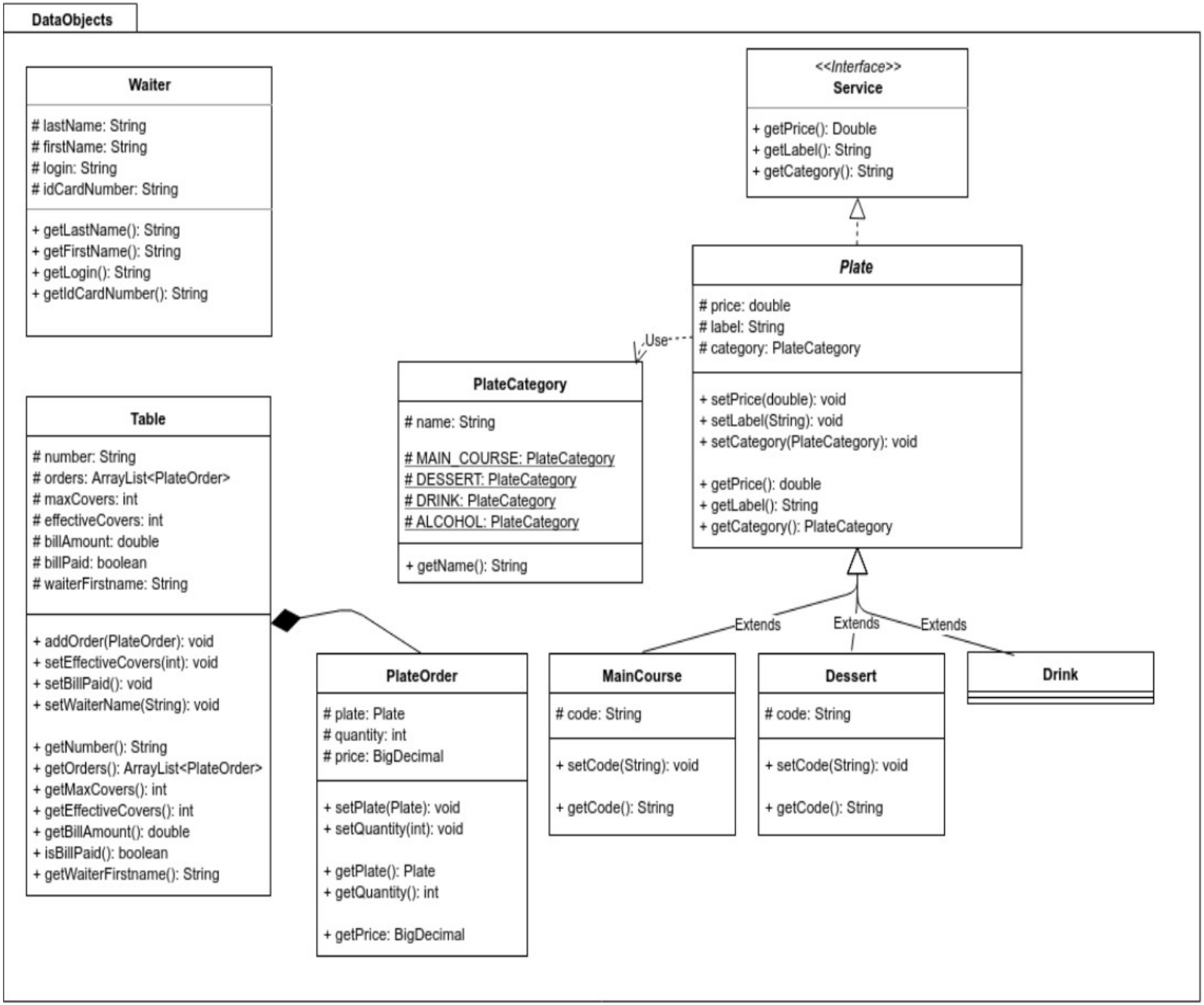
– La classe « Plate » utilise une catégorie de plat « PlateCategory ». Normalement, on aurait dû utiliser une agrégation forte (également appelée composition) marqué par un trait continu terminé par un losange blanc. Ici, l'objet utilisé est cependant contenu dans la classe « PlateCategory ». Il n'est donc pas supprimé lorsqu'un « Plat » est détruit de la mémoire (RAM, fichier…). C'est pour cette raison, de par l'utilisation de ces membres statiques que nous avons représenté ce lien, de façon subjective, à l'aide d'une simple relation d'utilisation.

– La classe « PlateOrder » est utilisée par la classe « Table ». Il s'agit d'une agrégation par valeur (une table peut être constituée d'une commande ou non).

1 https://www.ibm.com/developerworks/library/j-math2/index.html#floatingpoint

– Les classes « MainCourse », « Dessert » et « Drink » héritent toutes de la classe abstraite « Plate ». Elles héritent donc de l'ensemble de ses propriétés et implémentent les méthodes abstraites qui n'ont donc pas d'implémentation dans la classe « Plate ». Ici, la classe « Plate » possède une implémentation pour chacune de ses méthodes, elle ne dispose donc pas de méthodes abstraites. Aucune des 3 classes enfants (« MainCourse », « Dessert » et « Drink ») n'implémente donc une méthode qui ne possède pas déjà d'une implémentation dans la classe parente « Plate ».

  Les classes « MainCourse » et « Dessert » implémentent cependant chacune une méthode additionnelle permettant de définir et récupérer un code de plat (« public String getCode() » et « public void setCode(String code) »).


DataObjects
Waiter
lastName: String
firstName: String
login: String
idCardNumber: String
+ getLastName(): String
+ getFirstName(): String
+ getLogin(): String
+ getIdCardNumber(): String
<<Interface>>
Service
+ getPrice(): Double
+ getLabel(): String
+ getCategory(): String
Plate
price: double
label: String
category: PlateCategory
+ setPrice(double): void
+ setLabel(String): void
+ setCategory(PlateCategory): void
+ getPrice(): double
+ getLabel(): String
+ getCategory(): PlateCategory
Use
PlateCategory
name: String
MAIN_COURSE: PlateCategory
DESSERT: PlateCategory
DRINK: PlateCategory
ALCOHOL: PlateCategory
+ getName(): String
Table
number: String
orders: ArrayList<PlateOrder>
maxCovers: int
effectiveCovers: int
billAmount: double
billPaid: boolean
waiterFirstname: String
+ addOrder(PlateOrder): void
+ setEffectiveCovers(int): void
+ setBillPaid(): void
+ setWaiterName(String): void
+ getNumber(): String
+ getOrders(): ArrayList<PlateOrder>
+ getMaxCovers(): int
+ getEffectiveCovers(): int
+ getBillAmount(): double
+ isBillPaid(): boolean
+ getWaiterFirstname(): String
PlateOrder
plate: Plate
quantity: int
price: BigDecimal
+ setPlate(Plate): void
+ setQuantity(int): void
+ getPlate(): Plate
+ getQuantity(): int
+ getPrice: BigDecimal
Extends
Extends
Extends
MainCourse
code: String
+ setCode(String): void
+ getCode(): String
Dessert
code: String
+ setCode(String): void
+ getCode(): String
Drink

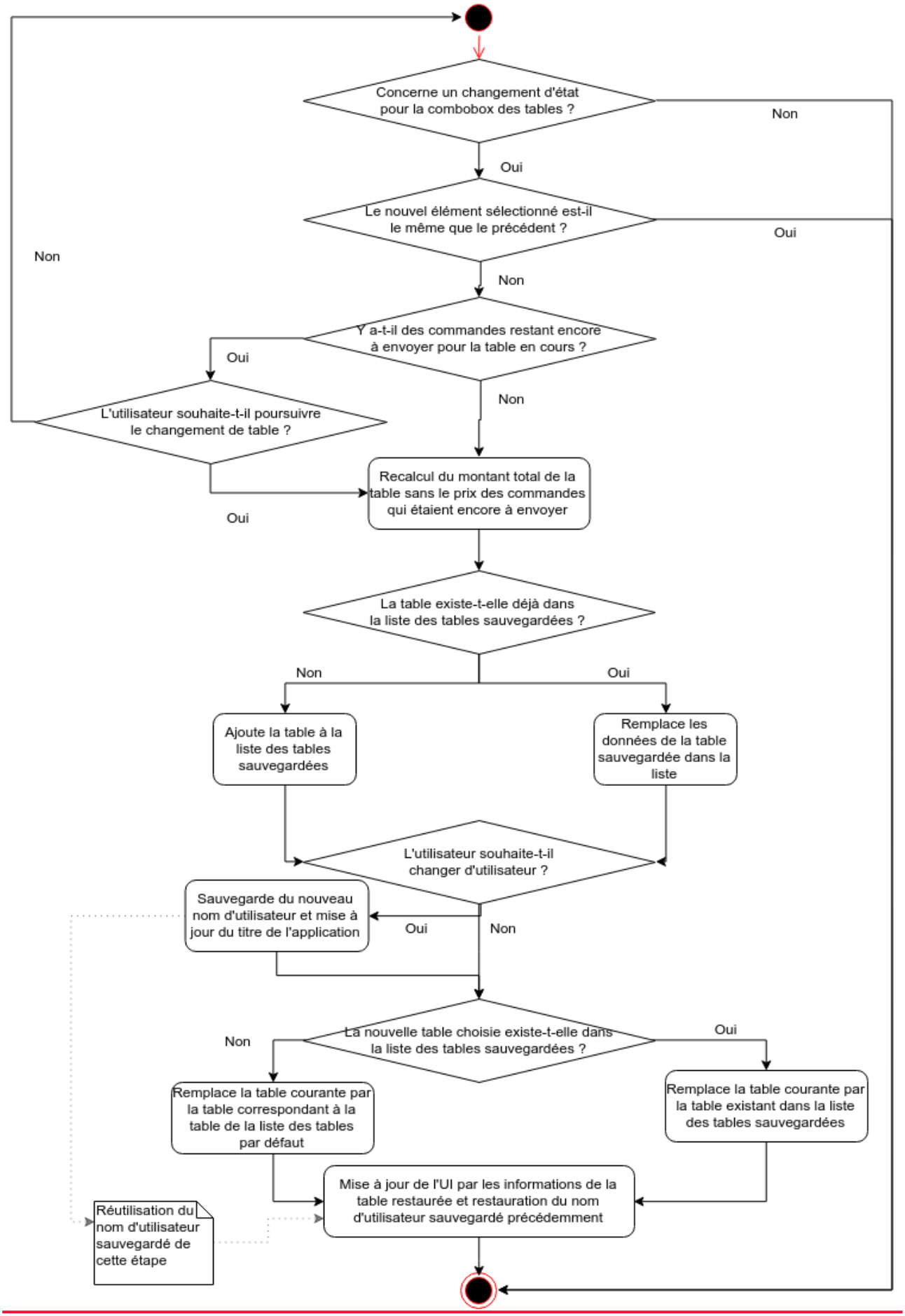
2. DIAGRAMME D'ACTIVITÉS (POUR LE CHANGEMENT D'UTILISATEUR)

Ce diagramme correspond à la fonction du changement d'utilisateur tel que demandé pour cette première évaluation.

Cette fonction est en réalité composée de plusieurs parties qui ne sont pas visibles en UML. Le diagramme montre en effet le fonctionnement global de l'algorithme.

Ce dernier est en réalité scindé en deux parties. La première dans le code captant l'événement de changement d'élément au sein du menu déroulant (combobox) de sélection de table, avec une seconde partie qui ne contient que la sauvegarde de la table courante qui est placée dans une fonction à part. En effet, cette dernière fonction a été créée dans l'optique d'être appelée lorsque l'utilisateur souhaite mettre fin au programme et que la sauvegarde de la table en cours est nécessaire.

Dans pareil cas de figure, le code relatif à la détection de l'événement de changement de table ainsi que la partie relative au changement d'utilisateur n'a donc pas raison d'être dans cette fonction pure de sauvegarde.


Concerne un changement d'état pour la combobox des tables ?
Non
Oui
Le nouvel élément sélectionné est-il le même que le précédent ?
Oui
Non
Non
Y a-t-il des commandes restant encore à envoyer pour la table en cours ?
Oui
L'utilisateur souhaite-t-il poursuivre le changement de table ?
Non
Oui
Recalcul du montant total de la table sans le prix des commandes qui étaient encore à envoyer
La table existe-t-elle déjà dans la liste des tables sauvegardées ?
Non
Oui
Ajoute la table à la liste des tables sauvegardées
Remplace les données de la table sauvegardée dans la liste
L'utilisateur souhaite-t-il changer d'utilisateur ?
Sauvegarde du nouveau nom d'utilisateur et mise à jour du titre de l'application
Oui
Non
La nouvelle table choisie existe-t-elle dans la liste des tables sauvegardées ?
Non
Oui
Remplace la table courante par la table correspondant à la table de la liste des tables par défaut
Remplace la table courante par la table existant dans la liste des tables sauvegardées
Mise à jour de l'UI par les informations de la table restaurée et restauration du nom d'utilisateur sauvegardé précédemment
Réutilisation du nom d'utilisateur sauvegardé de cette étape

3. BIBLIOTHÈQUE MYUTILS

Cette bibliothèque ayant dû être développée à l'extérieur de NetBeans, nous avons dû effectuer l'ensemble de la compilation manuellement. De façon à automatiser le processus, nous avons développé un script Bash (appelé par le biais d'un Makefile).

```
#!/usr/bin/env bash

. "${0%/*}/utils.sh"

function finish {
    warning "SIGINT received. Exiting..."
    cleanup
    exit
}
trap finish SIGINT

function cleanup() {
    rm -v MANIFEST.MF
}

while [[ $# -gt 0 ]]; do
    key="$1"
    case $key in
        -c|--clean)
            cleanRequested=true
            shift
            ;;
        -t|--test)
            testsRequested=true
            shift
            ;;
        *)
            die "'$key' is an invalid parameter. Exiting..."
    esac
done

requireDeps "javac jar java javadoc"
dest="${0%/*}/../dist"
src="../sources/"
libraryName="StringSlicer.jar"
appName="stringslicer_test.jar"
mainClass="be.wget.inpres.java.restaurant.myutils.tests.StringSlicerTest1"
oldPath="${PWD}"

mkdir -p "$dest"
cd "$dest"

if [ "$cleanRequested" = "true" ]; then
    info "Cleaning '$dest' content..."
    rm -r ./*
fi

# Don't use for loop, they are vulnerable. Use a while loop with "here
# strings".
# src.: http://mywiki.wooledge.org/BashFAQ/001
while IFS= read -r file; do
    info "Compiling file \"$file\" to Java bytecode..."
    javac -classpath "$src" -d . "$file" -Xlint:unchecked
```

```
done <<< "$(find "$src" -type f -name '*.java')"

info "Compilation terminated."

info "Gathering library class files..."
classFiles=()
while IFS= read -r file; do
    classFiles+=("$file")
done <<< "$(find . -type f -name '*.class' ! -name '*test*')"

info "Creating library jar..."
# By default, even if a standard manifest file is created automatically, we
# need a custom one to specify a Main-Class and a Class-Path
cat <<EOF > MANIFEST.MF
Manifest-Version: 1.0
Created-By: Manual compilation by William Gathoye

EOF
# Do not specify the path as argument to the jar executable, otherwise, it will
# create an archive with the destination subfolder in it.
jar cvfm "$libraryName" MANIFEST.MF "${classFiles[@]}"

info "Gathering test class files..."
classFiles=()
while IFS= read -r file; do
    classFiles+=("$file")
done <<< "$(find . -type f -iname '*test*.class')"

cat <<EOF > MANIFEST.MF
Manifest-Version: 1.0
Created-By: Manual compilation by William Gathoye
Class-Path: StringSlicer.jar
Main-Class: $mainClass

EOF
jar cvfm "$appName" MANIFEST.MF "${classFiles[@]}"

info "Creating javadoc..."
javadoc -d "docs/"
"$src/be/wget/inpres/java/restaurant/myutils/StringSlicer.java"

cleanup

if [ "$testsRequested" = "true" ]; then
    cd "$oldPath"
    info "Executing tests..."
    info "Testing \"java -classpath $dest $mainClass\""
    java -classpath "$dest" "$mainClass"
    info "Testing \"java -jar $dest/$appName\""
    java -jar "$dest/$appName"
    cd "$dest"
fi

info "Usage:"
info "Execute the main executable with \"java -classpath $dest $mainClass\""
info "OR"
info "Execute the jar file with \"java -jar $dest/$appName\""
```

Comme le requiert Java, pour respecter notre propre namespace « be.wget.inpres.java.restaurant.myutils », nous avons dû créer une arborescence où chaque sous dossier est créé par la sous-chaîne délimitée par les points dans le namespace.

L'arborescence :

```
./sources
./sources/be
./sources/be/wget
./sources/be/wget/inpres
./sources/be/wget/inpres/java
./sources/be/wget/inpres/java/restaurant
./sources/be/wget/inpres/java/restaurant/myutils
./sources/be/wget/inpres/java/restaurant/myutils/tests
./sources/be/wget/inpres/java/restaurant/myutils/tests/test1.java
./sources/be/wget/inpres/java/restaurant/myutils/StringSlicer.java
```

Le fichier StringSlicer.java contient notre classe, test1 contient le teste de notre bibliothèque. Ce dernier appelle donc StringSlicer.

Étapes de la compilation de la bibliothèque :

– Se placer dans le dossier où se trouve le fichier Makefile ou appeler le fichier Bash directement. Se placer dans un autre dossier fonctionne également, car le script Bash tient compte du chemin dans lequel on se trouve et change de dossier vers celui contenant nos sources de façon transparente.

– Créer un dossier de destination ou placer les ficher .class

– Prendre le chemin relatif menant aux fichiers sources

– Compiler les fichiers sources en gardant la même arborescence que celle des fichiers sources.

– Créer le fichier manifeste pour la bibliothèque

– Créer le jar pour la logique métier (sans les tests)

– Créer un nouveau manifeste pour notre jar de test qui inclu le Class-Path menant à notre bibliothèque et spécifie la classe Main contenue dans nos tests

– Créer le jar de test

Si les tests sont demandés comme arguments à l'application, exécuter la commande java en spécifiant le bon chemin vers les jars pour prouver que tout fonctionne pour le mieux.

4. FICHIERS PROPERTIES

Notre application tient compte de 2 fichiers de properties :

– Un pour nos paramètres globaux d'application (settings.conf). Ce fichier peut être partagé entre le gestionnaire de salle et ou le gestionnaire de cuisine. Nous n'avons pas tenu compte de l'écriture simultanée vers ce fichier.

  Par défaut, sa localisation est placée dans le dossier courant, pour NetBeans, il s'agit du dossier du projet.

– Un fichier pour nos utilisateurs (users.conf), uniquement pour l'application de salle. Sa structure comprend le nom d'utilisateur et le digest de son mot de passe (résultat de la fonction de hashage SHA-512).

  Par défaut, le fichier est préfixé par une ligne de commentaire indiquant quand il a été écrit.

```
wagner=1ae039b8233b849f24173e0484274eaf093a8e62422f32379591156451a620c05ffe47a0
bda3ee25f108db2c64b8aca9cdd15d744fece285ebd06c93b3a59588
wget=3627909a29c31381a071ec27f7c9ca97726182aed29a7ddd2e54353322cfb30abb9e3a6df2
ac2c20fe23436311d678564d0c8d305930575f60e2d3d048184d79
```

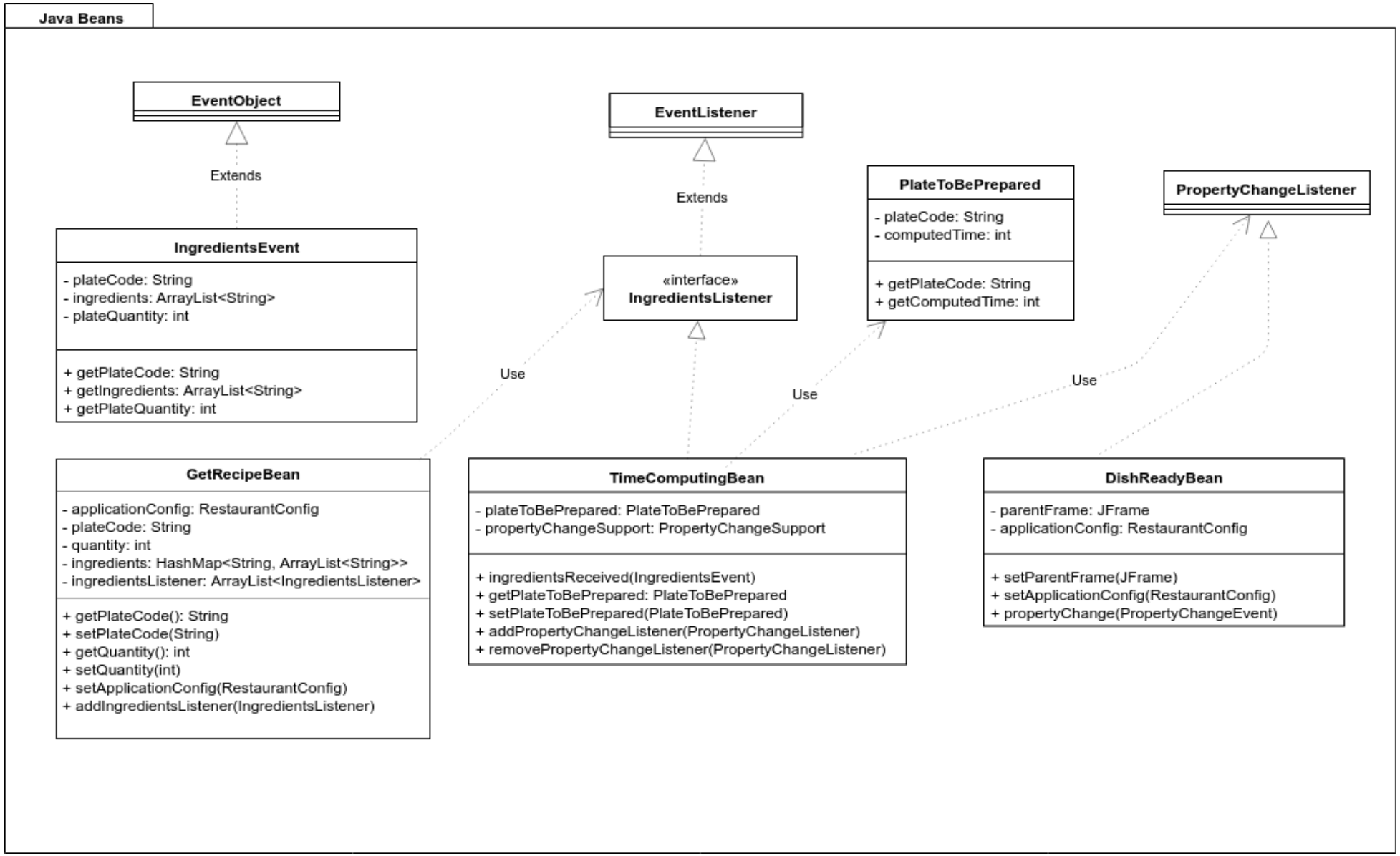
5. FICHIER DE PLATS

Un fichier sérialisé de plats (plats principaux) nommé plates.txt, comporte l'ensemble des plats par défaut. Libre à l'utilisateur d'en rajouter manuellement selon cette même structure, pourvu qu'il respecte le même format et ne tente pas de réutiliser le même code de plat ou tente d'utiliser des prix sous un format non numéraire.

La structure du fichier contenant les plats par défaut :

```
VRH&Veau au rollmops sauce Herve&15.75
CC&Cabillaud chantilly de Terre Neuve&16.9
FE&Fillet de boeuf Enfer des papilles&16.8
GF&Gruyère farci aux rognons-téquila&13.4
PA&Potée auvergnate au miel&12.5
```

6. DIAGRAMME DE CLASSES DE BEANS


Java Beans
EventObject
Extends
IngredientsEvent
- plateCode: String
- ingredients: ArrayList<String>
- plateQuantity: int
+ getPlateCode: String
+ getIngredients: ArrayList<String>
+ getPlateQuantity: int
EventListener
Extends
«interface»
IngredientsListener
PlateToBePrepared
- plateCode: String
- computedTime: int
+ getPlateCode: String
+ getComputedTime: int
PropertyChangeListener
Use
Use
Use
GetRecipeBean
- applicationConfig: RestaurantConfig
- plateCode: String
- quantity: int
- ingredients: HashMap<String, ArrayList<String>>
- ingredientsListener: ArrayList<IngredientsListener>
+ getPlateCode(): String
+ setPlateCode(String)
+ getQuantity(): int
+ setQuantity(int)
+ setApplicationConfig(RestaurantConfig)
+ addIngredientsListener(IngredientsListener)
TimeComputingBean
- plateToBePrepared: PlateToBePrepared
- propertyChangeSupport: PropertyChangeSupport
+ ingredientsReceived(IngredientsEvent)
+ getPlateToBePrepared: PlateToBePrepared
+ setPlateToBePrepared(PlateToBePrepared)
+ addPropertyChangeListener(PropertyChangeListener)
+ removePropertyChangeListener(PropertyChangeListener)
DishReadyBean
- parentFrame: JFrame
- applicationConfig: RestaurantConfig
+ setParentFrame(JFrame)
+ setApplicationConfig(RestaurantConfig)
+ propertyChange(PropertyChangeEvent)