

## Laboratoire de Java

### 2<sup>ème</sup> informatique de gestion et 2<sup>ème</sup> informatique et systèmes (industrielle et réseaux) 2017-2018

Claude Vilvens – Jean-Marc Wagner – Christophe Charlet



## Projet "Le gourmet audacieux"

### Contexte de développement

#### 1. Préambule

L'Unité d'Enseignement " **Développement orienté objets et multitâche** " (gestion: 9 ECTS (105 h) / indus & réseaux: 8 ECTS (94 h)) se structure en trois Activités d'apprentissage de la manière suivante :

- ♦ AA: Programmation du multi tâche léger- Threads (gestion: 30h (29%) / indus & réseaux: 19h (20%))
- ♦ AA: Programmation orientée objet Unix et Windows - Java (gestion: 30h (29%) / indus & réseaux: 30h (32%))
- ♦ AA: Programmation orientée objet Windows- C# (gestion: 45h (42%) / indus & réseaux: 45h (48%))

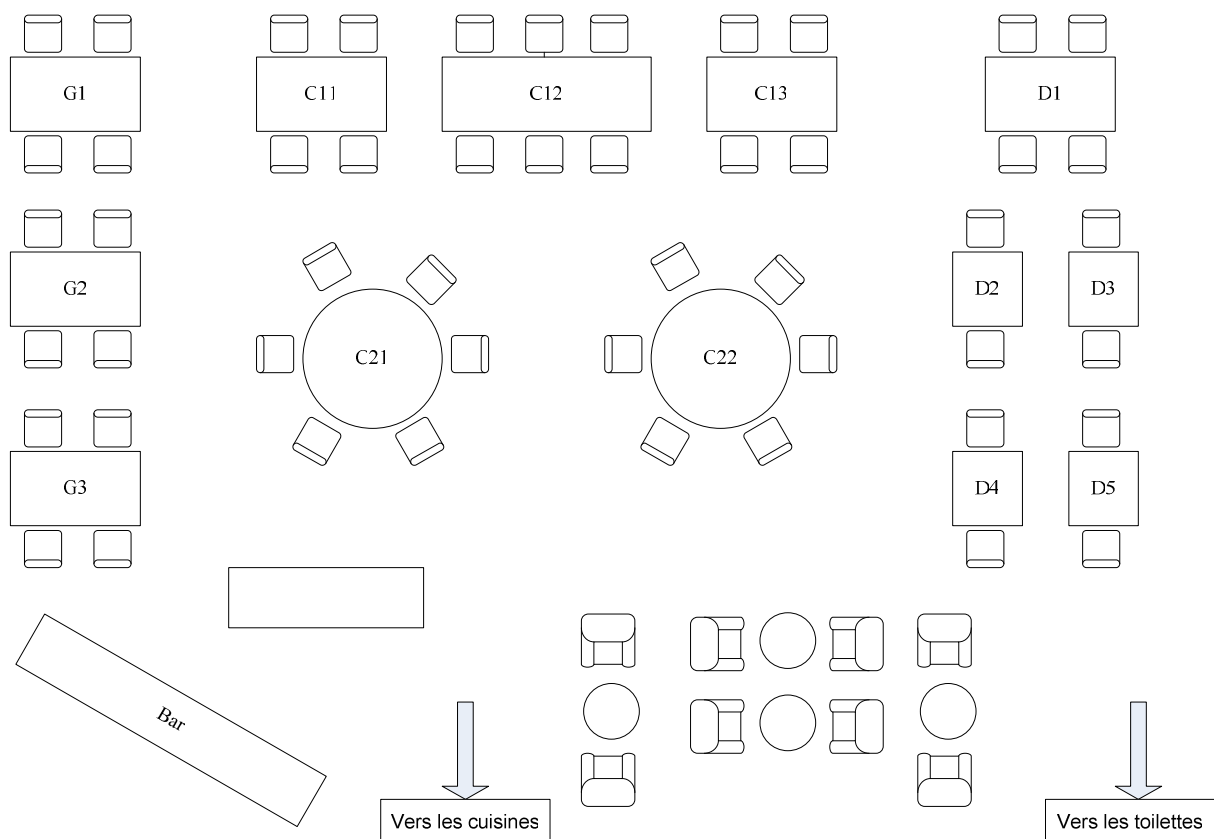
Les travaux de programmation réseaux présentés ici constituent la description technique des travaux de laboratoire de l'AA " **Programmation orientée objet Unix et Windows - Java** ".

## 2. Le projet "Le gourmet audacieux"

Le thème de ce laboratoire de programmation Java est la gestion élémentaire d'un restaurant dénommé "**Le gourmet audacieux**". Ce restaurant propose à une clientèle d'affaires ou familiale des plats qui sortent des sentiers battus, mariant des saveurs de façon parfois ... audacieuse (voir la carte dans les copies écran ci-dessous). Vu la consistance des plats proposés, le restaurant ne propose pas d'entrées. Par contre, il propose des desserts ... tout aussi "audacieux". La salle du restaurant se veut un endroit convivial :



et comporte des tables réparties selon le schéma suivant :



Le projet "**Le gourmet audacieux**" à développer est destiné principalement aux serveurs qui travaillent en salle (application **ApplicationSalle**), mais une interaction est prévue avec le personnel qui travaille en cuisine (application **ApplicationCuisine**). Les boissons servies sont gérées par le bar (application **ApplicationBar**), qui travaille en fait de manière autonome et se contente d'envoyer des additions de boissons à l'application principale - cette dernière application ne sera pas implémentée par manque de temps de laboratoire.

L'idée générale est que l'application **ApplicationSalle** propose un interface permettant aux serveurs (serveuses) d'encoder les plats et desserts commandés et d'être prévenu(e)s de leur disponibilité quand ils sont prêts à être servis.

Il va de soi que les possibilités d'une telle application réellement utilisée dans un restaurant sont plus nombreuses. Dans le contexte de ce laboratoire, seules certaines fonctionnalités auront le temps d'être d'implémentées.

Le développement de ces applications fera intervenir :

- ◆ la conception de classes et interfaces répartis dans des packages structurés;
- ◆ les bases de la programmation Java (notamment les packages, interfaces et classes abstraites, exceptions);
- ◆ les interfaces graphiques principalement Swing (notamment les JComboBox, JList et JTable);
- ◆ les classes utilitaires (containers comme Vector, LinkedList et Hashtable, StringTokenizer, Date-Calendar-DateFormat-Timezone);
- ◆ l'utilisation de la ligne de commande élémentaire (javac, java, jar);
- ◆ les flux et plus particulièrement les techniques de sérialisation sur fichier, l'utilisation de fichiers Properties et la lecture/écriture de fichiers textes;
- ◆ l'utilisation d'une librairie de communication réseau élémentaire (simple échange de chaînes de caractères) disponible sous forme d'un jar;
- ◆ les Java Beans avec les chaînes d'événements propriétaires et de type "propriété liée";
- ◆ la portabilité Windows-Unix (exécution d'une application Java à distance avec un serveur X-Window);
- ◆ une première approche des threads.

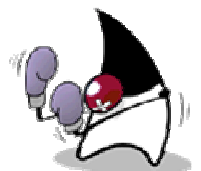
### 3. Les objectifs et les outils



L'ensemble des travaux proposés ici est à réaliser en utilisant l'environnement de développement **NetBeans version 8.\***, basé sur le **JDK 1.8**. Comme éditeur de texte plat, on pourra utiliser **JEdit 4.2** tandis **Xming 6.9** sera le serveur X-Window privilégié.

Le dossier est, *à priori*, à réaliser **par équipe de deux étudiants** mais **peut aussi être présenté par un étudiant seul** qui le désirerait, avec des avantages et des inconvénients qui s'échangent selon le choix posé.

Un petit conseil : ***lisez bien l'ensemble de l'énoncé*** avant de concevoir (d'abord) ou de programmer (après) une seule ligne ;-). Ceci vous permettra non seulement d'avoir une vision globale du projet mais aussi de déjà remarquer des traitements communs à des points différents des applications. Prévoyez une schématisation des diverses classes (peut-être des diagrammes de classes UML ?) et élaborer d'abord "sur papier" (donc sans programmer directement) les divers scénarios correspondant aux fonctionnalités demandées.



### 4. Les règles d'évaluation

1) L'évaluation établissant la note de l'AA de " Programmation orientée objet Unix et Windows - Java" [ex "Réseaux et technologie Internet"] est réalisée de la manière suivante :

- ◆ **théorie** : un examen écrit en juin 2018, sur base des notes de cours ("Java I: programmation de base") et de listes de points de théorie à préparer, listes fournies au fur et à mesure; il sera coté sur 20;

♦ laboratoire : 2 évaluations (une fin avril, non remédiable, et une durant la 1<sup>ère</sup> session, remédiable), chacune cotée sur 20; la **moyenne pondérée de ces 2 cotes** (poids respectifs de 4/10 et 6/10) fournit une note de laboratoire sur 20;

♦ note finale : **moyenne de la note de théorie (poids de 5/10) et de la note de laboratoire (poids de 5/10)**.

Cette procédure est d'application tant en 1<sup>ère</sup> qu'en 2<sup>ème</sup> session.

2) *Dans le cas où les travaux sont présentés par une équipe de deux étudiants*, chacun d'entre eux doit être capable d'expliquer et de justifier l'intégralité du travail (pas seulement les parties du travail sur lesquelles il aurait plus particulièrement travaillé).

3) En 2<sup>ème</sup> session, un **report de note** est possible séparément pour **la note de laboratoire** ainsi que pour **la note de théorie** **pour des notes supérieures ou égales à 10/20**.

Toutes les évaluations (théorie ou laboratoire) ayant des **notes inférieures à 10/20** sont **à représenter dans leur intégralité**.

La première partie des travaux de programmation réseaux sera **évaluée** par l'un des professeurs du laboratoire **à partir de la semaine du 30 avril 2018** (avec rentrée d'un dossier papier - le délai est à respecter impérativement).

La deuxième partie sera **évaluée** lors de l'examen de laboratoire en juin 2018 (le dossier papier n'est plus nécessaire).

## Evaluation 1

Les applications demandées doivent fonctionner sur une machine **Windows (PC ou portable)** **et** sur une machine **UNIX (Sunray avec terminal ou émulation de terminal)** de l'InPrES.

*Dossier :*

- ◆ schéma UML des classes utilisées dans l'application
- ◆ explication et code correspondant à un changement de table dans l'application.

## I. Fonctionnalités de base du projet "Le gourmet audacieux"

### 1. L'application ApplicationSalle

Lorsqu'un serveur a pris la commande d'une table, il entre dans l'application et se fait reconnaître comme utilisateur agréé (procédure classique de login-password) :



Pour l'instant, la validation d'un serveur se base sur une hashtable statique en mémoire (clé=login=prénom, valeur=mot de passe).

En cas de succès, il parvient à l'interface graphique qui se présente, dans un premier temps, sous l'aspect suivant (modifiable selon les souhaits des commanditaires):

Restaurant "Le gourmet audacieux" : Dimitri

Table : ? Plats servis: RIEN

Boissons (bar) : ? EUR

ajouter

Addition : **NON PAYEE**

Encaisser

Encodage des commandes :

Plats : VRH: Veau au rollmops sauce herve (15.75 EUR) Quantité: ?

Commander plats

Remarques: ??

Desserts: D\_MC: Mousse au chocolat salé (5.35 EUR) Quantité: ?

Commander desserts

Commandes à envoyer :

RIEN

Envoyer

LES APPHORISIES DE BRILLAT-SAVARIN

"l'ordre des comestibles est, des plus substantiels aux plus légers"

☐ Commande envoyée ☐ Plats prêts Lire plats disponibles

Le rôle de cet interface graphique est clairement d'enregistrer une commande de plats et desserts pour une table donnée et d'envoyer cette commande à la cuisine (qui utilise l'application ApplicationCuisine).

## 2. Au préalable : les classes de base

Une modélisation minimale de l'application considérée implique l'utilisation de classes "Serveur", "Plat" et "Table". Plus précisément :

- 1) la classe **Serveur** se contente d'encapsuler les nom, prénom et login (en principe son prénom, éventuellement complété des premières lettres du nom si risque d'ambigüité) du serveur ainsi que son numéro de carte d'identité;
- 2) la classe **Plat** est une classe abstraite qui comporte l'information du prix; elle implémente (partiellement donc) l'interface **Service** qui déclare les méthodes :

```
double getPrix();
String getLibelle();
String getCategorie();
```



En ce qui concerne la catégorie, la classe **CategoriePlat** encapsule un nom de catégorie (un String) et, pour faciliter son emploi, contient des variables membres static instances de cette classe :

static final CategoriePlat PLAT\_PRINCIPAL, DESSERT, BOISSON, ALCOOLS, etc.

Les classes **PlatPrincipal** (catégorie CategoriePlat.PLAT\_PRINCIPAL) et **Dessert** (catégorie CategoriePlat.DESSERT) implémentent bien sûr la classe Plat. L'une et l'autre contiennent aussi un code (ex: "FE" pour "Filet de boeuf Enfer des papilles", "D\_DG" pour le dessert "Dame grise").

Dans une première approche (mais cela changera dans la suite), les plats sont référencés dans une hashtable statique (clé=code, valeur=objet PlatPrincipal qui contient le nom et le prix). On pratiquera de même pour les desserts.

3) la classe **Table** est évidemment la pierre d'angle du restaurant. Elle comporte

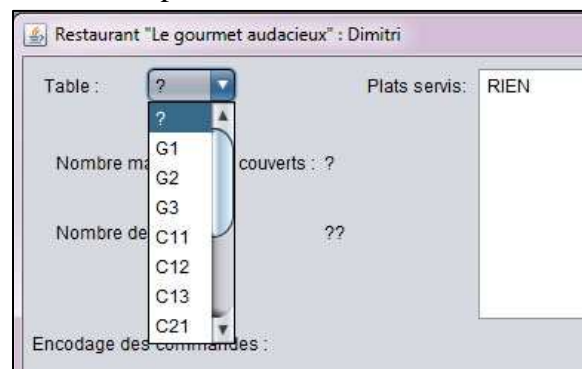
- ◆ un numéro de table;
- ◆ un container (Vector, ArrayList, etc) des commandes de plats qui y sont servis;
- ◆ le nombre maximum de couverts et le nombre effectif de couverts (il peut n'y avoir que trois convives sur une table de 4 places);
- ◆ l'addition;
- ◆ le fait que l'addition a été payée (synonyme de départ des clients) ou pas;
- ◆ le prénom du serveur qui a pris la table en charge (nous supposons ici que ce serveur ne change pas durant une séance).

La classe **CommandePlat** permet de référencer un Plat (plat principal ou dessert) avec la quantité commandée. Il est ainsi plus facile de mémoriser les commandes d'une table (sinon, il faudrait mentionner chaque plat le nombre de fois qu'il a été commandé :- ( ...).

Si les clients changent, les tables ne changent pas en ce qui concerne leur numéro et nombre maximum de couverts. Dans une première approche (mais cela changera dans la suite), les tables sont référencées dans une hashtable statique (clé=numéro de table, valeur=objet Table qui contient donc le nombre maximum de couverts).

### 3. La prise de commande

Le serveur commence bien sûr par choisir la table :



Une fois ce choix effectué, le nombre de couverts possible pour cette table est affiché:

Le serveur peut alors encoder les plats et desserts choisis par cette table ainsi que le nombre de chacun d'entre eux :

puis

L'appui sur le bouton "Commander plats" a pour effet d'envoyer la commande dans la liste "Commandes à envoyer" et aussi d'incrémenter le nombre effectif de couverts :



Bien sur, ceci a aussi pour effet d'ajouter les plats dans le container de plats de la table. Si le nombre effectif de couverts dépasse le nombre maximum de couverts de la table, l'exception **TooManyCoversException** est lancée. Elle ne signifie pas forcément une erreur (on a pu ajouter un couvert en plus si les clients se serrent un peu ;-)) : ce sera au serveur de choisir dans la boîte de dialogue qui apparaît dans ce cas si on continue ou si on annule tout (donc erreur véritable).

La procédure est similaire pour les desserts, mais sans l'incréméntation et la vérification du nombre effectifs de couverts (certains convives sont des amateurs de desserts ;-)).

Plats : VRH: Veau au rollmops sauce herve (15.75 EUR) Quantité: 2

Commander plats

Remarques: ??

Desserts: D\_MC: Mousse au chocolat salé (5.35 EUR) Quantité: ?

Commander desserts

D\_MC: Mousse au chocolat salé (5.35 EUR)

D\_SC: Sorbet citron courgette Colonel (6.85 EUR)

D\_CJ: Duo de crêpes Juliettes (6.00 EUR)

D\_DG: Dame grise (5.55 EUR)

D\_CB: Crème très brûlée Carbone (7.00 EUR)

Une fois toutes les commandes encodées, on peut envoyer la commande globale des plats par appui sur le bouton "Envoyer" qui bien sûr envoie cette commande à l'application ApplicationCuisine (voir plus loin) et recopie les plats choisis dans la liste "Plats servis" (un plat commandé est considéré comme servi et à payer !) :

Restaurant "Le gourmet audacieux" : Dimitri

Table : G3 Plats servis: RIEN

Boissons (bar) : ? EUR

ajouter

Addition : **NON PAYEE**

Encaisser

Encodage des commandes :

Plats : GF: Gruyère farci aux rognons-téquila (13.4 EUR) Quantité: 1

Commander plats

Remarques: ??

Desserts: D\_MC: Mousse au chocolat salé (5.35 EUR) Quantité: ?

Commander desserts

Commandes à envoyer :

2 VRH: Veau au rollmops sauce herve (15.75 EUR)  
1 GF: Gruyère farci aux rognons-téquila (13.4 EUR)

Envoyer

LES AFFORISSES DE BRILLAT-SAVARIN

"l'ordre des comestibles est  
des plus substantiels aux plus légers"

☐ Commande envoyée ☐ Plats prêts Lire plats disponibles

Restaurant "Le gourmet audacieux" : Dimitri

Table : G3 Plats servis: 2 VRH: Veau au rollmops sauce herve (15.75 EUR) (c)  
1 GF: Gruyère farci aux rognons-téquila (13.4 EUR) (c)

Boissons (bar) : ? EUR

ajouter

Addition : **NON PAYEE**

Encaisser

L'addition de la table est évidemment incrémentée des prix des différents plats et desserts.

## II. Fonctionnalités de changement de table et d'addition

### 4. Changement de table et de serveur

Jusqu'à présent, tout s'est passé sur une seule table avec intervention du serveur qui s'en occupe. A remarquer : on supposera qu'un serveur qui encode ses commandes le fera en terminant par l'envoi de la commande aux cuisines (cela semble raisonnable - sinon, quel intérêt ?).

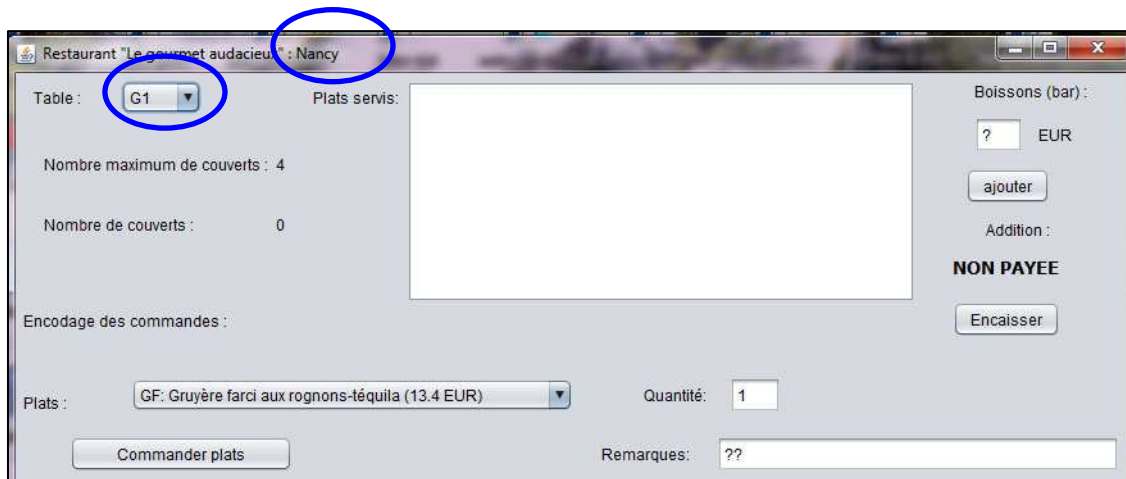
Mais bien sûr, un autre serveur peut vouloir intervenir sur une autre table dont il est (ou va devenir) le responsable de service. Pour ce faire, une sollicitation sur la boîte combo des numéros de table fera apparaître une boîte de dialogue demandant si le serveur change :



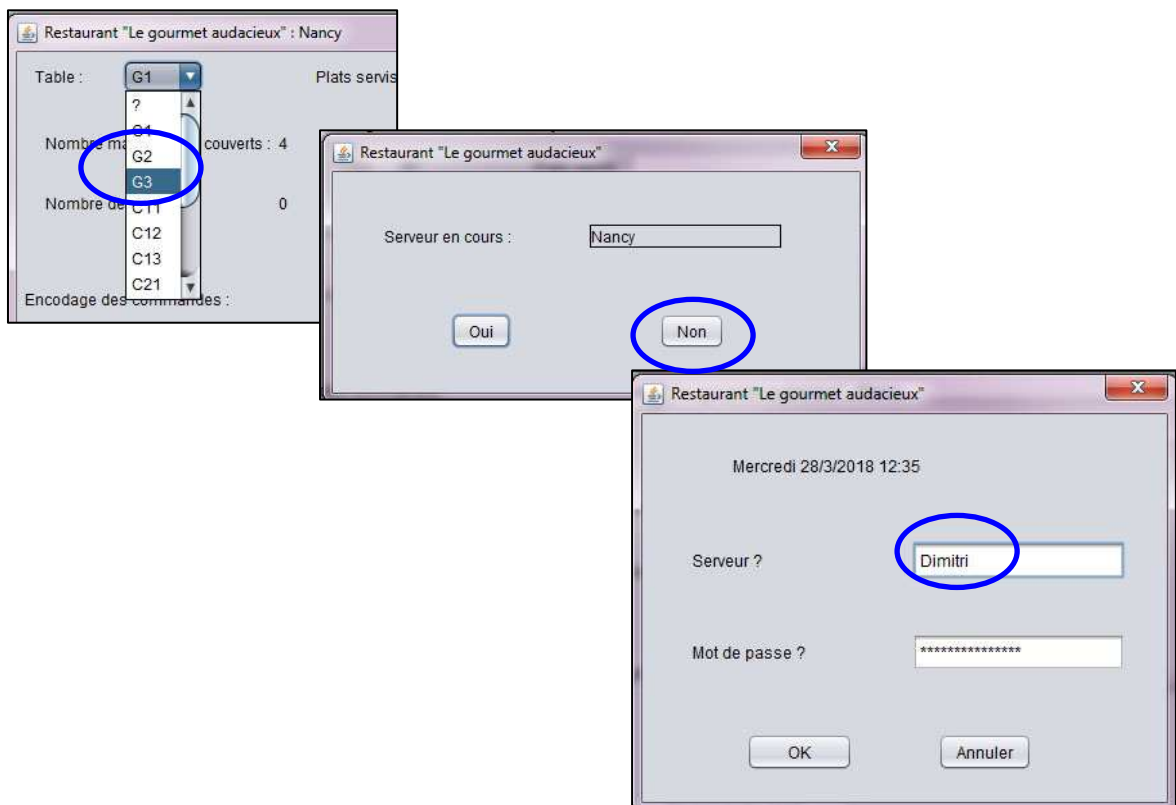
Si non, le nouveau serveur (ou la nouvelle serveuse) se fait reconnaître :



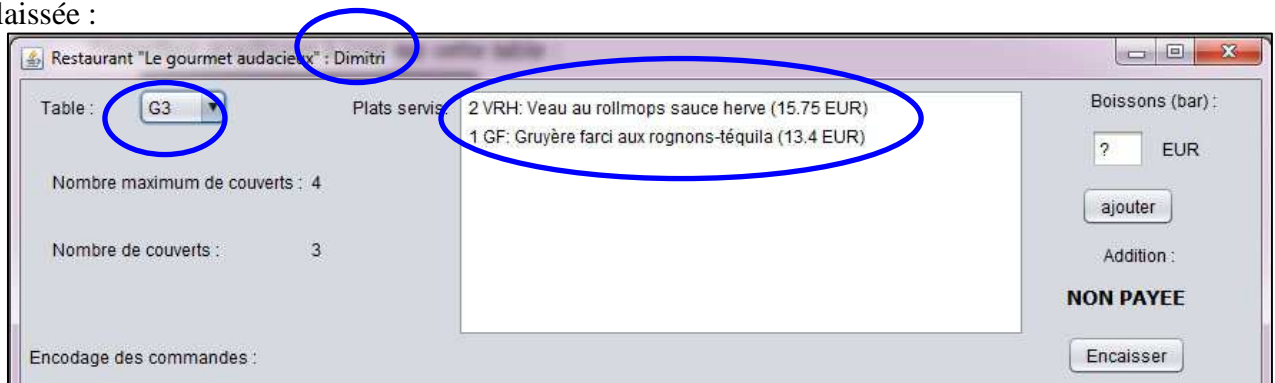
Après quoi ce serveur se voit (ré)attribuer la table et l'interface graphique est simplement mis à jour avec les informations de la table choisie (si il en existe) :



Bien sûr, un retour à une table pour laquelle on a déjà enregistré les commandes mettra l'interface graphique à jour sur cette table :



Ici, Dimitri a repris la main pour revenir sur la table G3 qu'il retrouve dans l'état où il l'a laissée :



## 5. Les boissons et l'addition

Le bar fonctionne de manière autonome (il est possible de simplement passer boire un verre - voir plan du restaurant avec les fauteuils près du bar). Le serveur (la serveuse) responsable d'une table se contente donc d'enregistrer le total des consommations de boissons qui sont ajoutées à l'addition :

Restaurant "Le gourmet audacieux" : Lydia

Table : G1 Plats servis: 2 CC: Cabillaud chantilly de Terre Neuve (16.9 EUR) (c)  
1 PA: Potée auvergnate au miel (12.5 EUR) (c)

Nombre maximum de couverts : 4  
Nombre de couverts : 3

Encodage des commandes :

Plats : PA: Potée auvergnate au miel (12.5 EUR) Quantité: 1

Boissons (bar) : 12.30 EUR

ajouter

Addition : **NON PAYEE**

Encaisser

Restaurant "Le gourmet audacieux" : Lydia

Table : G1 Plats servis: 2 CC: Cabillaud chantilly de Terre Neuve (16.9 EUR) (c)  
1 PA: Potée auvergnate au miel (12.5 EUR) (c)  
Boissons avec repas (12.3 EUR)

Nombre maximum de couverts : 4  
Nombre de couverts : 3

Encodage des commandes :

Plats : PA: Potée auvergnate au miel (12.5 EUR) Quantité: 1

Boissons (bar) : 12.30 EUR

ajouter

Addition : **NON PAYEE**

Encaisser

Quant à l'addition (appui sur le bouton "Encaisser"), on peut vouloir simplement la visualiser, ou l'imprimer ou encore l'encaisser (remarquer les tooltips) :

Restaurant "Le gourmet audacieux"

Table: G1

Nombre de couverts : 3

A PAYER: 58.599999999999994

☒ Consulter ☐ Imprimer ☐ Encaisser

Juste pour voir

Ok Annuler

Restaurant "Le gourmet audacieux"

G1

3

58.599999999999994

☐ Imprimer ☐ Encaisser

Fabriquer souche

Annuler

Enregistrer le paiement

Pour l'instant, le seul effet d'un choix sera d'afficher dans la console l'opération choisie (voir plus loin pour plus évolué).

## **6. La portabilité**

Pour rappel, afin d'illustrer la portabilité de Java,

l'application demandée doit fonctionner sur une machine **Windows**  
**et** sur une machine **UNIX** de l'INPRES.

Concrètement, l'application peut être lancée depuis Netbeans sur la machine de développement, mais **par la ligne de commande "java -jar ..."** (donc, pas avec Netbeans) sur la machine Unix Sunray. Il faut être capable d'expliquer le contenu du fichier manifeste.



## Evaluation 2

*Dossier :*

- ◆ étapes manuelles du développement et du test de la librairie MyUtils;
- ◆ fichiers properties;
- ◆ fichier texte des plats;
- ◆ schéma UML (diagramme de classe statique) de la chaîne de beans du dernier point de l'énoncé.

### III. Développement manuel

#### 7. Un peu de ligne de commande

Le découpage de chaînes de caractères en des composantes déterminées par un (ou plusieurs) délimiteurs est une technique qui revient couramment dans cet énoncé. Aussi, il est demandé construire **manuellement** une **librairie MyUtils** qui, pour l'instant, ne contient que la classe **StringSlicer** dont le constructeur réclame la chaîne à analyser et la chaîne de délimiteurs à utiliser pour cette analyse et qui possède les méthodes :

public int **getComponents**(boolean afficher)

- fournit le nombre de composantes et les affiche sur la console si le paramètre est à true

public Vector **listComponents**()

- fournit un Vector qui contient les composantes détectées

public LinkedHashSet **listUniqueComponents**()

- idem mais fournit une liste qui contient les composantes détectées sans répétitions et dans leur ordre d'apparition.

Un Javadoc sera joint à la distribution de cette (microscopique) librairie.

On demande de développer cette classe de librairie puis de la tester par un programme de tests élémentaires **manuellement** c'est-à-dire en utilisant les utilitaires de la ligne de commande javac, java, javadoc et jar (donc, par exemple, les deux jars de librairie et de test devront être créés manuellement).

### IV. Communication et persistance des données du projet "Le gourmet audacieux"

#### 8. L'application ApplicationCuisine

Cette application est celle qui fonctionne dans la cuisine du restaurant. Elle permet de faire parvenir à la cuisine la liste des plats commandés et, à l'inverse, permet de signaler au personnel de salle que certains plats sont prêts à être enlevés pour être servis aux clients. Cette application a l'aspect suivant :



On y remarquera donc l'utilisation de deux JTable :

- ◆ la première pour visualiser les détails d'une commande reçue;
- ◆ la deuxième pour gérer les plats en préparation qui devront ensuite être enlevés.

## 9. La communication ApplicationSalle-ApplicationCuisine

### 9.1 Les classes basiques de communication réseau

En fait, les deux applications peuvent communiquer entre elles en envoyant des chaînes de caractères. Il s'agit en fait d'une communication réseau TCP/IP, transparente pour le développeur, à priori utilisée sur une seule machine (en "localhost") mais en fait utilisable entre deux machines distinctes.

Pour cela, il est fourni deux classes (appartenant au package **network**) au sein du fichier **BasicStringNetworkLib.jar** (disponible sur l'EV [centre de ressources Vilvens]) :

◆ **NetworkBasicServer** : classe utilisant un thread permettant la réception d'une chaîne de caractères sur un port donné de la machine utilisée; en fait, les chaînes reçues sont mémorisées dans une liste et la méthode `getMessage()` fournit la première chaîne reçue ou "RIEN" si la liste est vide. Les méthodes principales sont :

```
public NetworkBasicServer (int p, JCheckBox cb);
```

Constructeur. On fournit le port d'écoute et la checkbox qui sera cochée dans le GUI de l'application si un message entrant est disponible.

```
public String getMessage();
```

Lecture du message suivant ("RIEN" si il n'y en a pas).

```
public void sendMessage(String m)
```

Envoi d'un message.

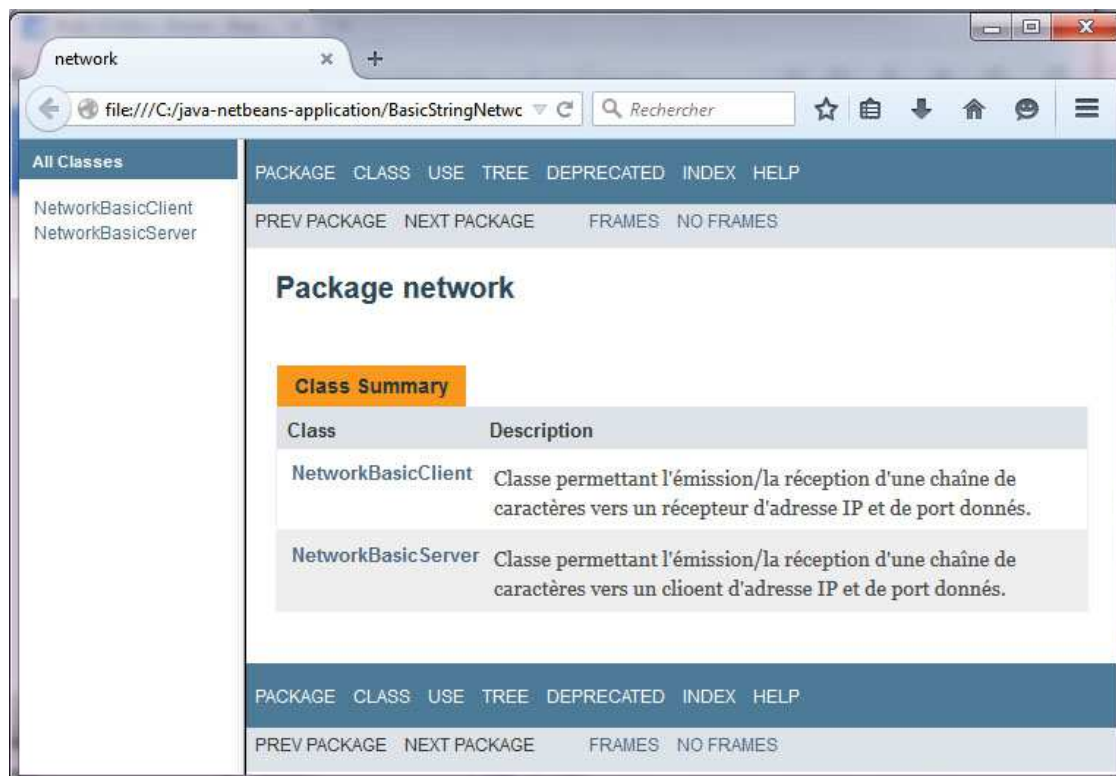
♦ **NetworkBasicClient** : classe utilisant un thread (transparent pour l'utilisateur) permettant l'émission d'une chaîne de caractères vers un récepteur d'adresse IP et de port donnés. Les méthodes principales sont :

**public NetworkBasicClient** (String a, int p);  
Constructeur avec connexion.

**public String sendString**(String s)  
Envoi d'un message avec attente bloquante de la réponse.

**public void sendStringWithoutWaiting**(String s)  
Envoi d'un message sans attente de réponse (simple notification).

Ces deux classes sont documentées par les javadocs qui se trouvent dans **BasicNetworkDoc.zip** (disponible au même endroit) :

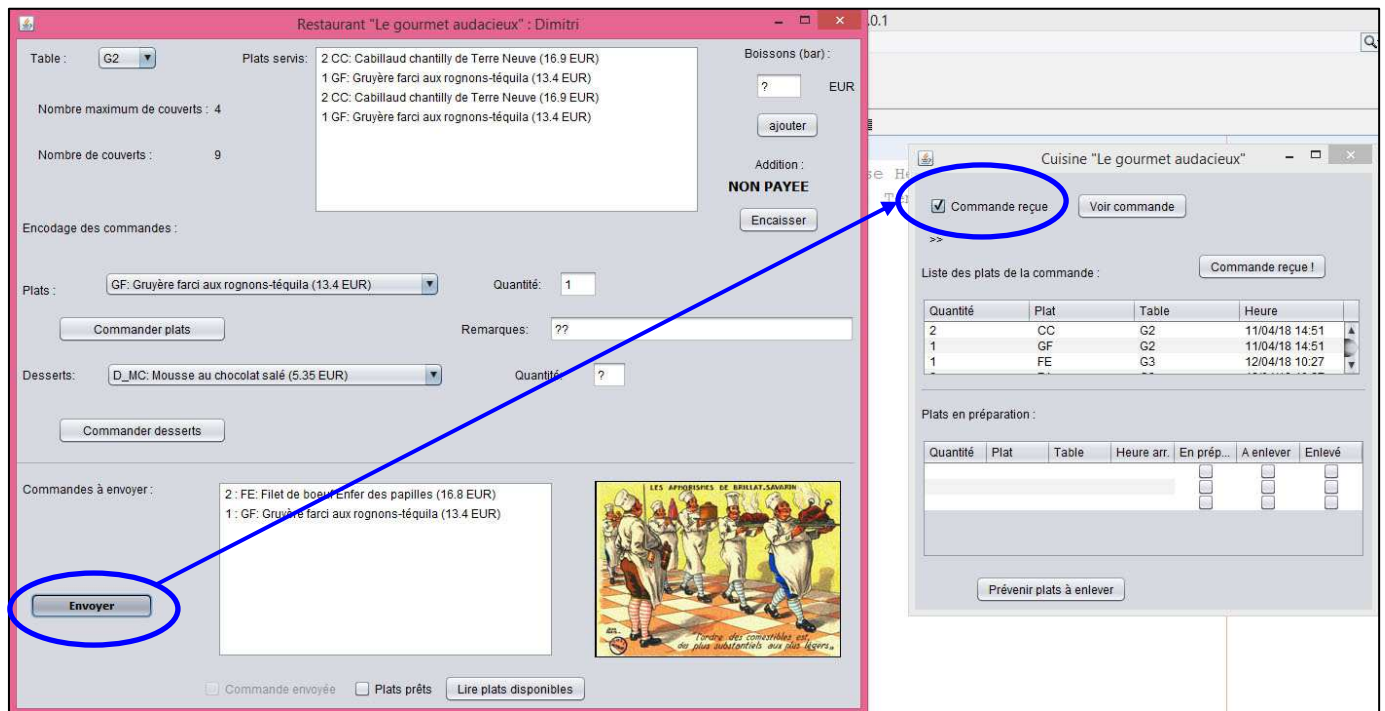


#### Remarque

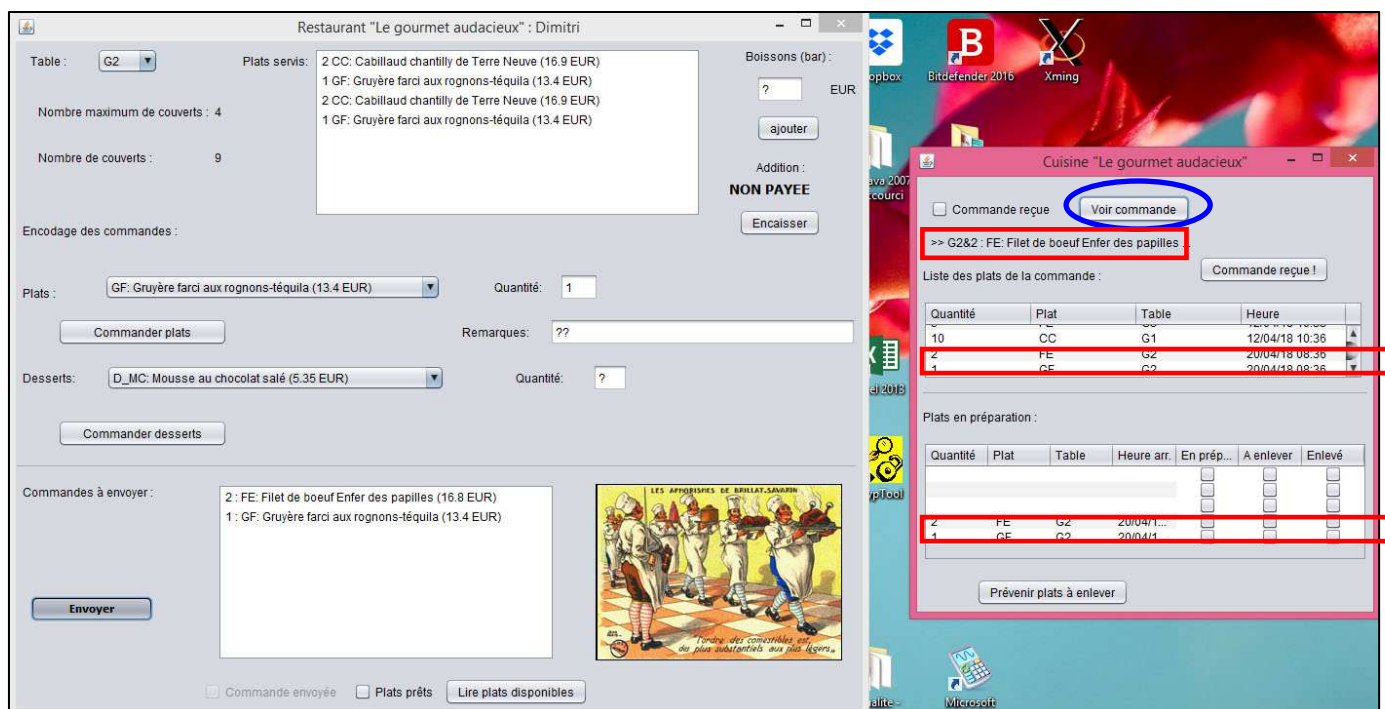
Dans cette librairie plus qu'élémentaire, les fins de connexions ne sont pas gérées explicitement, si bien qu'un arrêt brutal d'une application lance une exception, que l'on pourra ignorer ici.

## **9.2 La communication réseau d'ApplicationSalle vers ApplicationCuisine**

Quand un serveur appuie sur le bouton "Envoyer", l'application de la salle envoie à l'application Cuisine (qui attend donc sur un port donné) une chaîne de caractères contenant les plats se trouvant dans la boîte de liste "Commandes à envoyer" :

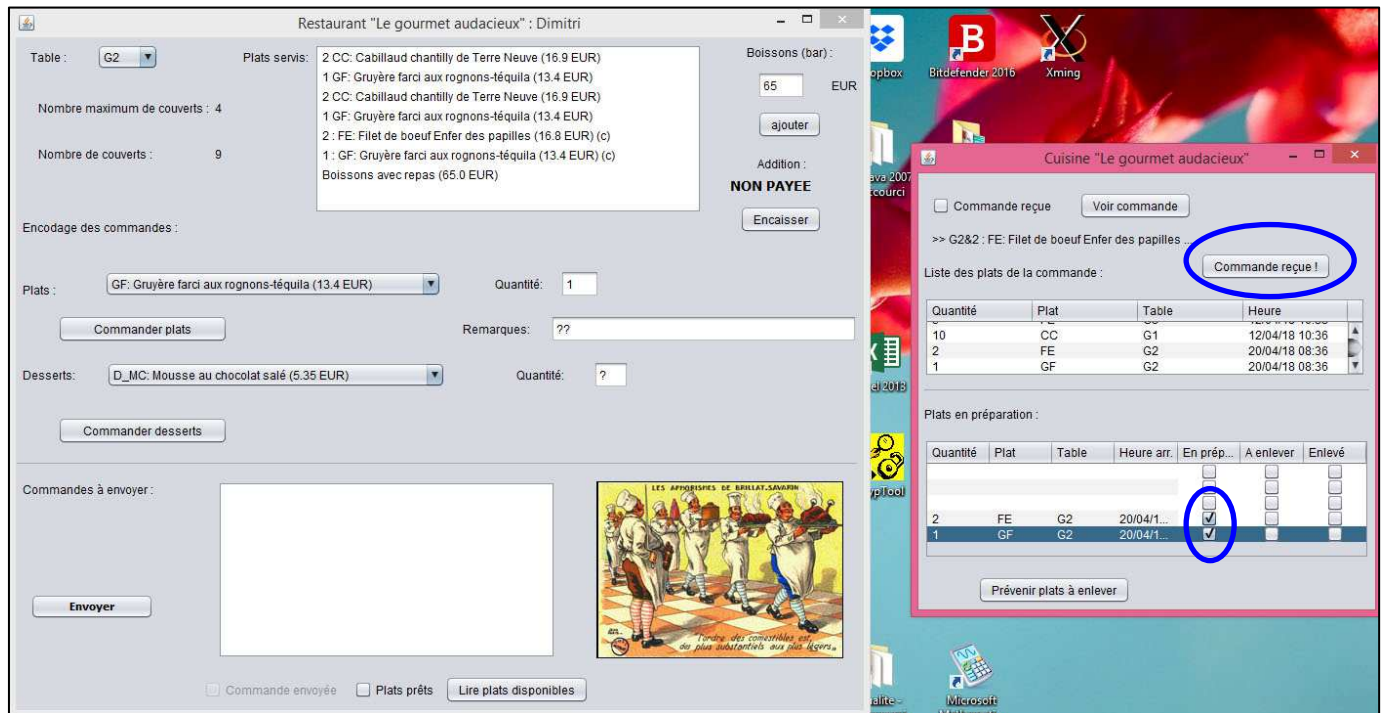


ApplicationSalle attend donc une réponse et reste donc bloquée tant que cette réponse ne sera pas arrivée. De son côté, ApplicationCuisine a vu sa checkbox "Commande reçue" cochée. Le chef de cuisine peut prendre connaissance de la commande reçue par un appui sur le bouton "Voir commande" :

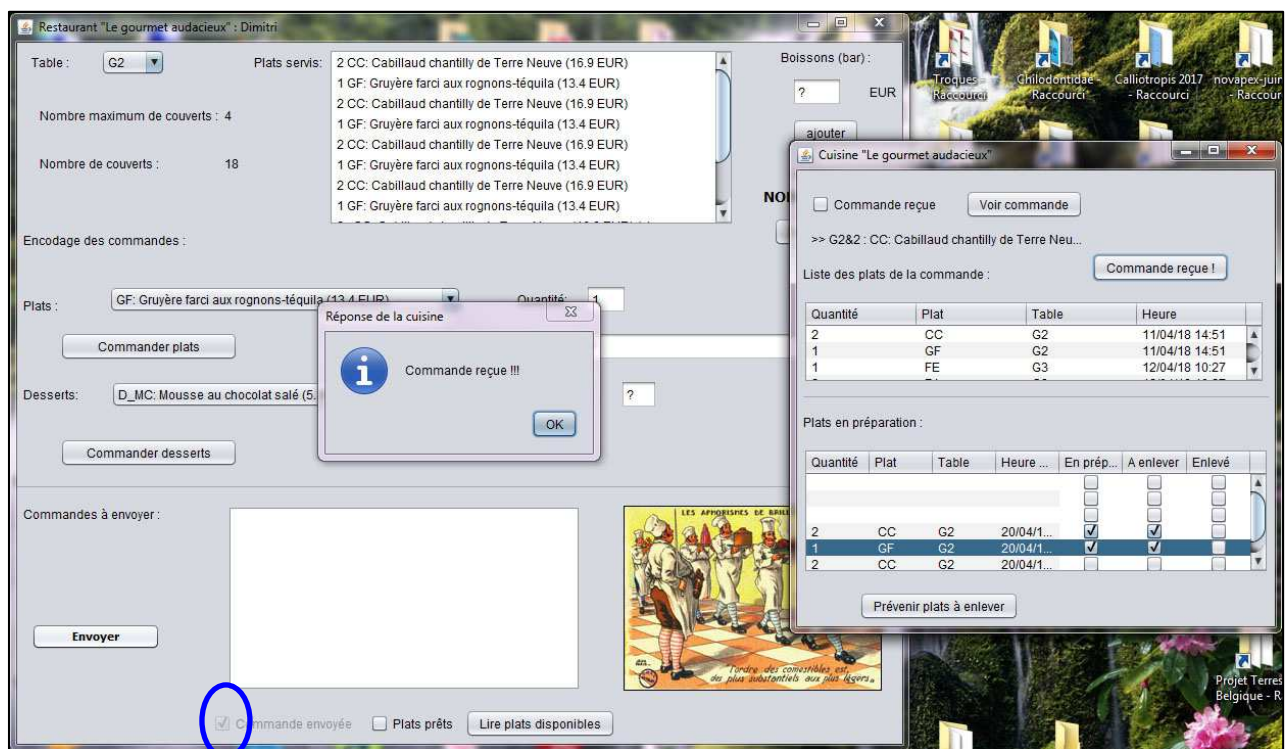


On peut voir que la commande apparaît derrière le symbole >> et qu'elle a été analysée pour produire des lignes supplémentaires dans les deux JTables (on utilisera la librairie créée plus haut, accessible sous forme de jar monté dans le projet). Le cuisinier peut donc à présent accepter cette commande en appuyant sur le bouton commande reçue (scénario de base).





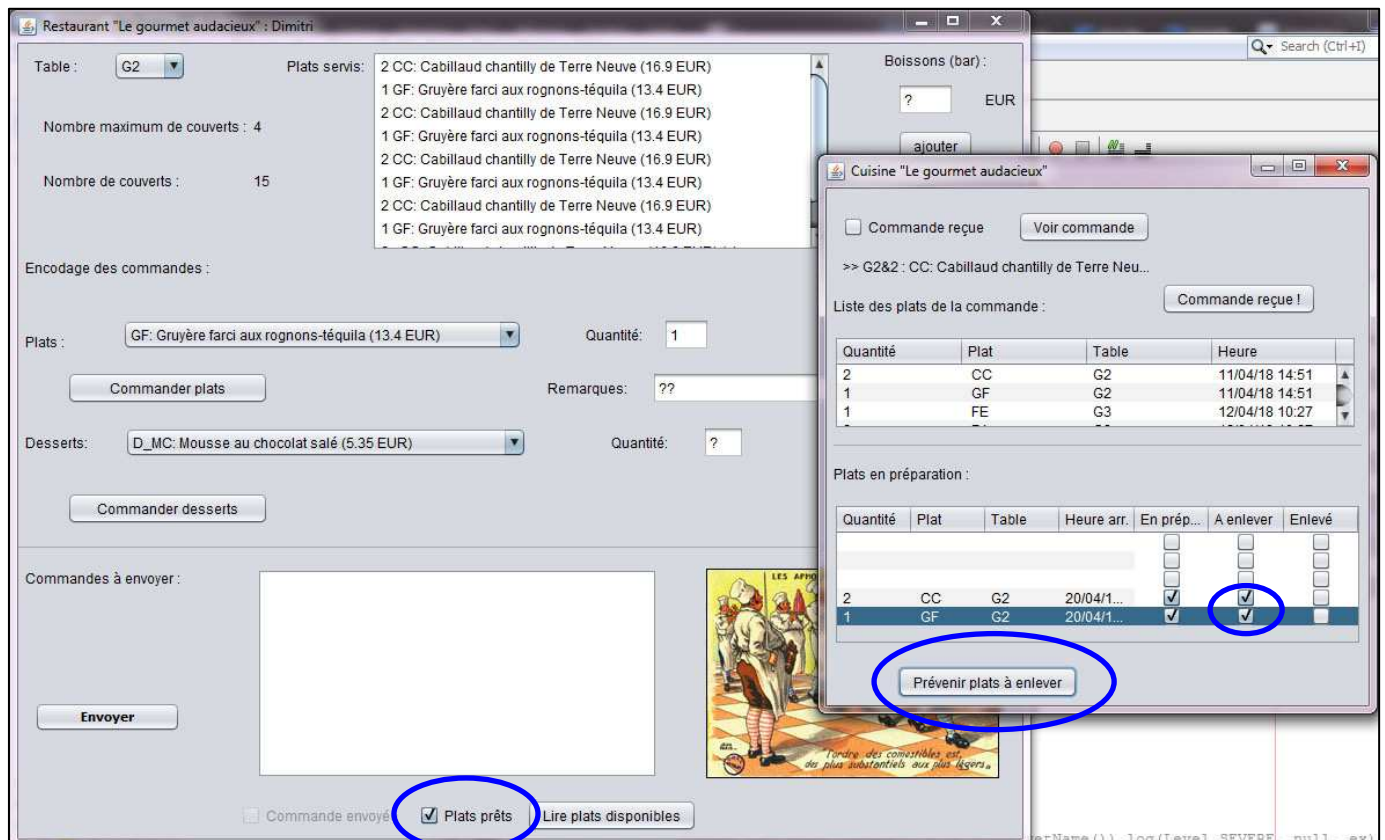
Ceci provoque l'envoi de la réponse à ApplicationSalle, du type :



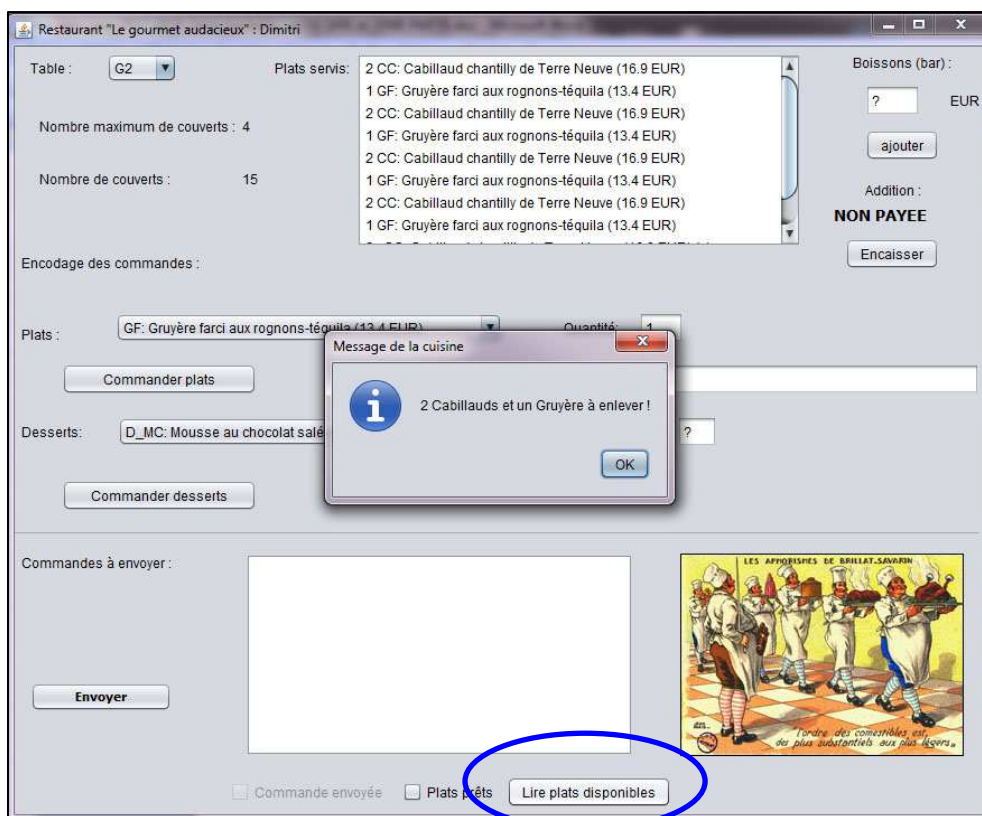
**BONUS:** Eventuellement (seulement si le temps de développement le permet), on peut imaginer que le cuisinier puisse refuser la commande (épuisement de certains ingrédients) - il faut dans ce cas ajouter une bouton "Commande refusée" qui fait apparaître une boîte de dialogue permettant d'introduire la raison : ceci constituera alors le message reçu par ApplicationSalle.

### 9.3 La communication réseau d'ApplicationCuisine vers ApplicationSalle

Lorsqu'un plat (ou plusieurs) est(sont) prêt(s), le chef cuisinier coche les checkbox "A enlever" correspondantes. L'appui sur le bouton "Prévenir plats à enlever" va envoyer sous forme de chaîne de caractères les plats ainsi cochés :



Le serveur peut alors lire quels sont les plats disponibles :





Il restera à cocher les checkbox "Enlevé" quand le serveur sera effectivement passé prendre les plats.

## **10. Propriétés, flux et sérialisation**

### **10.1 Fichier propriétés**

On demande d'utiliser des fichiers propriétés pour

- ◆ la configuration des applications (nom du restaurant, ports utilisés pour les communications, etc)
- ◆ la gestion des mots de passe des serveurs;

### **10.2 Sauvegarder l'état d'ApplicationSalle**

Il peut arriver qu'un serveur distrait arrête accidentellement l'application ApplicationSalle - il en est de même pour un chef de cuisine qui arrêterait l'application ApplicationCuisine. Il est évidemment souhaitable que le relancement des applications restaure les données encodées durant la session courante de midi ou du soir.

La hashtable contenant les tables sera donc sérialisée à chaque modification de table dans un fichier tables.data. Au démarrage de l'application, si ce fichier existe, cette hashtable des tables sera restaurée; bien sûr, si le fichier n'existe pas, les tables seront initialisées comme lors de l'évaluation 1.

### **10.3 Sauvegarder l'état d'ApplicationCuisine**

Pour sérialiser les données des JTables de cette application, on utilisera les méthodes deJTable

- ◆ à l'écriture :

```
public Vector getDataVector()
```

- c'est ce Vector qui sera sérialisé;

- ◆ à la relecture :

```
public void setDataVector(Vector dataVector, Vector columnIdentifiers)
```

Le "dataVector" est celui qui sera désérialisé. Pour obtenir les noms des colonnes, le plus simple est d'utiliser les méthodes

```
public int getColumnCount()
```

```
public String getColumnName(int column)
```

pour construire le Vector des columnIdentifiers.

### **10.4 Fichiers textes des plats**

Les plats sont à présent écrits dans un **fichier de texte** plats.txt et sont lus au démarrage de l'ApplicationSalle (ceci ne concerne donc que les la CategoriePlat.PLAT\_PRINCIPAL). Le fichier en question ressemble dont à ceci :

#### **plats.txt**

```
PA & Potée auvergnate au miel & 12.50
GF & Gruyère farci aux rognons-téquila & 13.40
CC & Cabillaud chantilly de Terre Neuve & 16.90
FE & Filet de boeuf Enfer des papilles & 16.80
VRH & Veau au rollmops sauce herve & 15.75
PHB & Pied d'hippopotame à la banane écrasée & 18.90
```

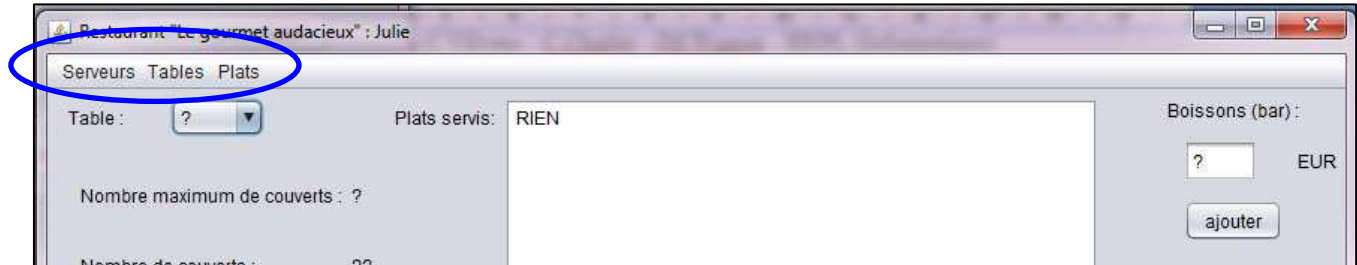
et peut être modifié directement par le chef de cuisine. On remarquera l'utilisation de séparateurs dans chaque ligne.

## V. Quelques fonctionnalités supplémentaires




### 11. Un menu pour ApplicationSalle

#### 11.1 Des items de menu classique pour ApplicationSalle

Afin de gérer au mieux le restaurant, l'application va se voir munir d'un menu :



Chaque item fait apparaître une boîte de dialogue. Plus précisément :

	<p><u>Serveurs</u></p> <ul style="list-style-type: none"> <li>♦ modifier le mot de passe du serveur en cours</li> <li>♦ ajouter un serveur (avec son mot de passe)</li> </ul>
	<p><u>Tables</u></p> <ul style="list-style-type: none"> <li>♦ liste des tables avec leurs plats et leur addition</li> <li>♦ nombre total de clients</li> <li>♦ somme totale des additions</li> </ul>
	<p><u>Plats</u></p> <p>liste des plats principaux liste des desserts ---- créer un plat supprimer un plat</p>



Remarque: La réalisation de ces opérations devrait normalement être soumise à des contrôles d'identité et d'accès, mais nous n'en tiendrons pas compte ici par manque de temps.

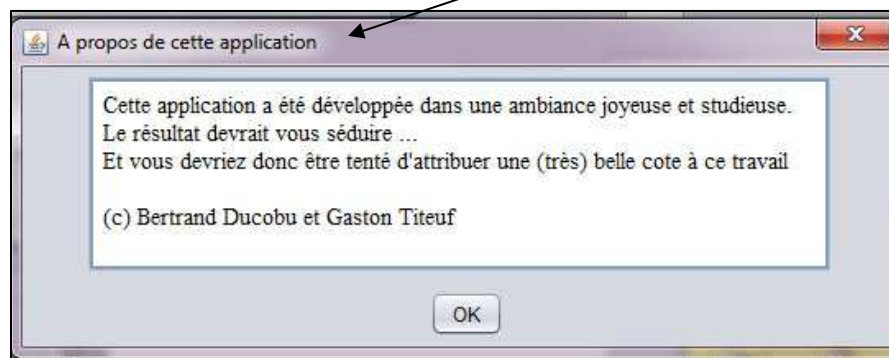
#### 11.2 Les items de menu Paramètres et Aide

Dans le menu principal, on remarque aussi l'existence d'items qui sont alignés à droite ("Paramètres" et "Aide"):



avec comme items respectifs :

	<p><u>Paramètres</u></p> <ul style="list-style-type: none"> <li>♦ informations sur le système hôte (système d'exploitation, répertoire courant, etc)</li> <li>♦ choisir le type d'affichage des dates</li> </ul>
	<p><u>Aide</u></p> <ul style="list-style-type: none"> <li>♦ courte explication sur le démarrage des applications</li> <li>♦ présentation de(des) l'auteur(s) des applications - du type:</li> </ul>



Pour obtenir l'affichage des items de menus à droite, le plus simple est d'ajouter dans le constructeur de la fenêtre (si jMenuBar1 désigne la barre de menu) :

```
jMenuBar1.add(Box.createHorizontalGlue()); //Espace horizontal pour aligner à droite
```

puis d'ajouter manuellement à cette barre les items de menus et sous-menus, donc du genre :

```
jMenuParametres = new JMenu("Paramètres");
jMenuBar1.add(jMenuParametres);
jMenuItemSys = new JMenuItem("Infos système");
jMenuParametres.add(jMenuItemSys);
...
```

Pour gérer les événements, la fenêtre implémentera ActionListener et se fera enregistrer :

```
jMenuItemSys.addActionListener(this);
...
```

## **12. Les threads : amélioration de l'impression de l'addition**

Lorsque l'on décidera de faire imprimer l'addition d'une table (ce qui sera ici simplement simulé), un thread affichera, durant le temps d'impression simulé, la boîte de dialogue suivante :



A remarquer que le format d'affichage de la date dépend des paramètres choisis dans le menu Paramètres → Paramètres de date-heure.

## VI. Les Java Beans

### 14. Une chaîne de beans dans la cuisine

A ce stade, les deux applications fonctionnent parfaitement. Mais on va imaginer une deuxième version de ApplicationCuisine dont l'architecture permettra une maintenance plus facile dans la suite (même si nous n'aurons pas le temps d'en faire plus dans ce laboratoire). Pour ce faire, on va mettre en place une chaîne de trois beans selon le mode de fonctionnement suivant.

Quand un plat est mis en préparation en cochant la checkbox correspondante (détecté par un TableModelEvent) :

- ◆ le bean GetRecipeBean est prévenu, lit les ingrédients du plat désigné (qu'il trouve dans un fichier dont le format est à choisir : sérialisé ? texte ?), les place, avec le nombre de plats à préparer, dans un IngredientsEvent puis lance cet événement à tous les IngredientsListener (une seule méthode : ingredientsReceived())
- ◆ le bean TimeComputingBean est un tel listener : il reçoit les ingrédients et estime le temps de préparation (de manière simpliste : somme des ingrédients / 3 \* 10s); ce bean possède une propriété liée instance de la classe simplissime PlatAPreparer (qui encapsule simplement le nom du plat et temps calculé);
- ◆ le bean DishReadyBean est un PropertyChangeListener pour le bean précédent: il est prévenu de la faisabilité du plat et prévient par une boîte de dialogue que la préparation du plat a été lancée, avec affichage du temps de préparation.

***Bon travail !***

s: CV, CC & JMW.

