Thread / Pattern Observer

Au niveau du BurgerResto

Nous souhaitons mettre en place deux fonctionnalités :

- Chaque fois que le client fait une nouvelle commande elle s'affiche sur l'ensemble des écrans des cuisiniers.
- A minuit toutes les commandes qui n'ont pas été récupérées sont retirées.

Au niveau des objectifs Java

Durant cette séance de TP nous travaillerons sur :

- Le pattern Observer,
- Les threads Java.

Au niveau de l'implémentation Java

Dans un premier temps nous testerons le travail fait en séance de TD dans la version console, puis nous pourrons valider l'aspect graphique avec la frame du cuisinier.

Diagramme de séquence détaillé

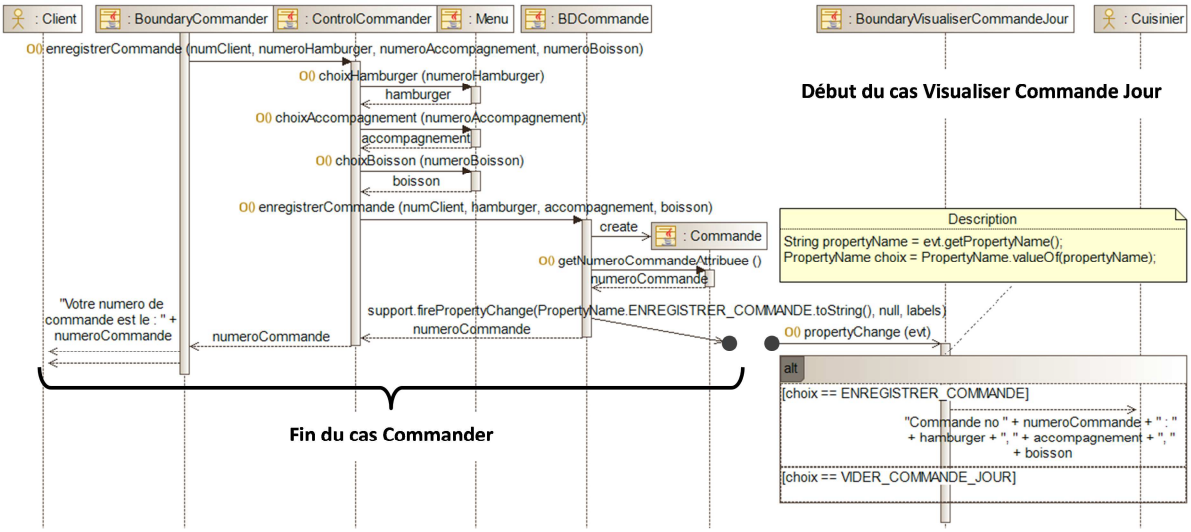
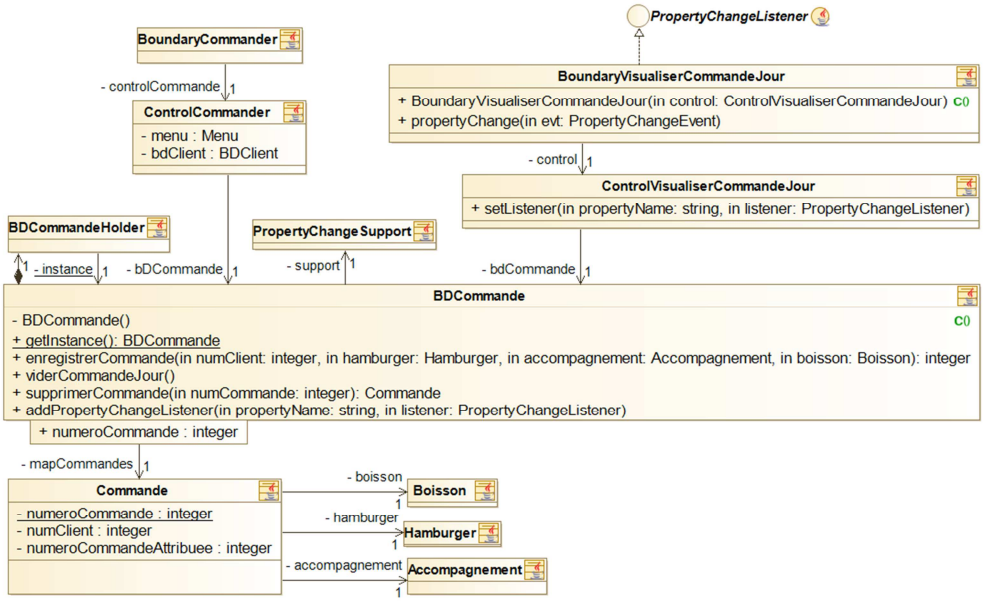


Diagramme de classes participantes



Le pattern Observer

1. *Affichage dans la console*

Vérifier que votre cas « commander » fonctionne en téléchargeant la classe « TestCommander » depuis Moodle (section Séance 2 de TP) sous le paquetage *vueconsole*.

Si votre cas commander ne fonctionne toujours pas :
- Créer un project java TP5 Prototype, avec 6 paquetages : *vueconsole*, *vuegraphique*, *control*, *model*, *testconsole* et *testgraphique*.
- Télécharger depuis Moodle les classes « BDCommande » et « Commande » dans le paquetage *model*.
- Copier dans ce même paquetage vos classes « Aliment », « Hamburger », « Accompagnement », « Boisson » et l'énuméré « ProfilUtilisateur ».
- Télécharger la classe « ProtoTestVisualiserCommandeJour » dans le paquetage *testconsole*. C'est ce test que vous utiliserez durant ce TP à chaque fois que je demanderais de tester.

Si votre cas commander fonctionne :
- Télécharger la classe « TestVisualiserCommandeJour » dans le paquetage *testconsole*. C'est ce test que vous utiliserez durant ce TP à chaque fois que je demanderais de tester.

Implémenter le cas « Visualiser commande du jour » vu en TD :
- Créer l'énuméré PropertyName contenant ***ENREGISTRER_COMMANDE***, ***SUPPRIMER_COMMANDE***, ***VIDER_COMMANDE_JOUR***,
- Implémenter le contrôleur « ControlVisualiserCommandeJour » ainsi que le boundary « BoundaryVisualiserCommandeJour ».
- Modifier la classe « BDCommande » afin qu'elle puisse être observée par le boundary « BoundaryVisualiserCommandeJour »,

Tester.

2. *Affichage dans un fichier*

Pour la partie suivante nous devrons effacer les écrans des cuisiniers en fin de journée. Or nous ne pouvons pas effacer le contenu de la console. Nous allons donc faire les affichages dans un fichier nommé ecranCuisinier.txt et nous supprimerons le contenu à une heure précise (nous n'attendrons pas minuit pour faire le test !).

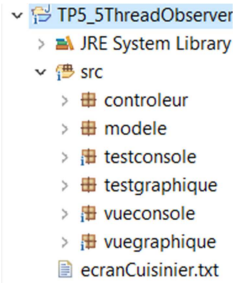
Pour cela :
- Télécharger la classe « Fichier » depuis Moodle vers le paquetage *vueconsole*. Cette classe permet d'écrire ou d'effacer le contenu d'un fichier grâce aux méthodes "static" :
  - *ecrire(String texte)*,
  - *effacer()*.

- Remplacer la sortie écran dans la méthode propertyChange de la classe « BoundaryVisualiserCommandeJour » :

```
System.out.println("Commande no "+ numeroCommande + " : "
                + nomHamburger + ", " + nomAccompagnement + ", " + nomBoisson);
```

Par :

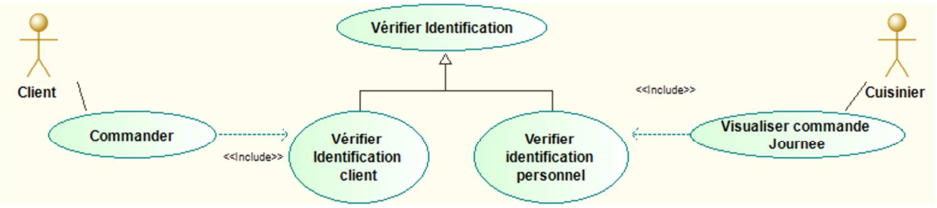
```
Fichier.ecrire("Commande no "+ numeroCommande + " : "
                            + nomHamburger + ", " + nomAccompagnement + ", " +
nomBoisson);
```



Lancer le test. Faire un refresh **sur le projet**. Apparait le fichier « ecranCuisinier.txt » (tout en bas !). Double cliquer et visualiser les commandes. Attention si vous voulez relancer le test il faut d'abord supprimer le fichier.

3. *Cas « Visualiser commande Journee »*

Attention : Afin de pouvoir visualiser la commande de la journée le cuisinier doit être connecté.



Le cas « Vérifier Identification personnel » ne nécessite pas l'intervention d'un acteur donc le contrôleur du cas « Visualiser commande Journée » communiquera directement avec le contrôleur du cas « vérifier identification personnel ».

Si vous n'avez pas implémenté le cas « Vérifier Identification personnel » utiliser le contrôleur « ProtoControlVerifierIdentificationPersonnel » disponible sous moodle et à placer sous le paquetage « controleur ».

Concepts

Le cuisinier ne peut consulter les commande que s'il est connecter, donc nous n'enregistrerons pas le boundary « BoundaryVisualiserCommandeJour » dans le constructeur, mais dans la méthode commander une fois que l'identification sera vérifier.

Instructions

Dans la classe « BoundaryVisualiserCommandeJour » supprimer du constructeur la ligne :

```
this.control.setListener(PropertyName.ENREGISTRER_COMMANDE.toString(), this);
```

Dans le contrôleur « ControlVisualiserCommandeJour » :

- créer un l'attribut controlVerifierIdentification de type « ControlVerifierIdentification ». Cet attribut sera initialisé par le constructeur,
- créer la méthode : *verifierIdentification* prenant en paramètre d'entrée le numéro de profil du cuisinier et le profil utilisateur (ProfilUtilisateur) et qui retourne un booléen indiquant si la personne en question est connectée ou non,

Dans la classe « BoundaryVisualiserCommandeJour » créer la méthode de lancement du cas *visualiserCommandeJour*. Prenant en paramètre d'entrée le numéro du cuisinier. Cette méthode :

- teste si le cuisinier est bien connecté (appel de la méthode *verifierIdentification* du contrôleur « ControlVisualiserCommandeJour » avec le numéro du cuisinier souhaitant visualiser les commandes du jour et le profil utilisateur PERSONNEL)
- si le cuisinier est bien connecté, lie l'observateur (boundaryVisualiserCommandeJour) à l'observé (bdCommande) : méthode *setListener* du contrôleur « ControlVisualiserCommandeJour » :

```
this.control.setListener(PropertyName.ENREGISTRER_COMMANDE.toString(), this);
```

Dans la classe de test modifier la ligne :

```
// creation pour la visualisation de la commande
ControlVisualiserCommandeJour controlVisualiserCommandeJour =
    new ControlVisualiserCommandeJour();
```

Par :

```
ControlVisualiserCommandeJour controlVisualiserCommandeJour =
    new ControlVisualiserCommandeJour(controlVerifierIdentification);
);
```

Dé-commenter la partie suivante :

```
boundaryVisualiserCommandeJour.visualiserCommandeJour(numCuisinier);
boundaryVisualiserCommandeJour.visualiserCommandeJour(numCuisinier2);
```

Lancer le test, les commandes sont écrites en double 1 pour chacun des écrans des cuisiniers.

Les threads Java

1. Affichage dans la console

À minuit toutes les commandes qui n'ont pas été récupérées sont retirées.

Instructions

- Mettre en place le thread « ThreadViderCommande » vue en TD.
- Dans le fichier de test dé-commenter les lignes suivantes

```
import modele.ThreadViderCommandeJour;

// Thread
    ThreadViderCommandeJour threadViderCommande = new ThreadViderCommandeJour();
    threadViderCommande.start();
```

- Tester. Quand les minutes correspondent à celles que vous avez entrées dans la classe « ThreadViderCommande », alors apparait à l'écran la chaine « `Vider commande` ». Attention la vérification ne se fait que toutes les 60 secondes, patienter un peu ! Au niveau de la console d'Eclipse appuyer sur le carré rouge pour stopper le thread « ThreadViderCommande », puis sur la croix grise.

2. Suppression dans le fichier pattern Observer

La suppression des commandes sur l'écran d'affichage des cuisiniers correspond à la suppression des commandes de la map mapCommandes dans la classe « BDCommande ».

Instructions

- **dans la classe « BDCommande »** ajouter la méthode *viderCommandeJour*. Cette méthode :
  - vide la map mapCommandes
  - suivant l'implémentation choisie pour numéroter les commandes vous devrez peut-être ajouter du code.
    Par exemple : si vous utiliser un attribut static numeroCommande dans la classe « Commande » vous devrez le remettre à 0 en ajoutant dans cette classe une méthode (static void initialiserNumeroCommande).
    Si vous utilisez la taille de la map comme numéro il n'y aura rien à faire.
- **dans la classe « ThreadViderCommande »** modifier la méthode *run* afin qu'elle appelle la méthode *viderCommandeJour* au lieu d'afficher à l'écran la chaine « `Vider commande` ».

Rappelez-vous la classe observée est la classe « BDCommande » et la classe qui observe est la classe « BoundaryVisualiserCommandeJour ».

- **dans la classe « BoundaryVisualiserCommandeJour »** :
  - dans la méthode visualiserCommandeJour, ajouter un listener sur la propriété VIDER_COMMANDE_JOUR :
    `control.setListener(PropertyName.VIDER_COMMANDE_JOUR.toString(), this);`
- **dans la classe « BDCommande »** :
  - compléter la méthode *viderCommandeJour* pour qu'elle envoie évènement à ses observateurs :
    `support.firePropertyChange(PropertyName.VIDER_COMMANDE_JOUR.toString(), null, null);`
- **dans la classe « BoundaryVisualiserCommandeJour »** :
  - Dans la méthode *properyChange* ajouter le cas où la propriété reçut est VIDER_COMMANDE_JOUR.
    Dans ce cas supprimer ce qui est écrit dans le fichier ecranCuisinier.txt (utilisation de la méthode *effacer* de la classe « Fichier »).

Tester :
- modifier dans le thread « ThreadViderCommande » les minutes pour avoir le temps d'entrer votre commande, par exemple il est 10H51 et vous avez entré pour la comparaison avec les minutes 52,
- lancer votre test. Pour ceux qui ne sont pas sur le proto, remplir les commandes des deux clients,
- cliquer droit sur votre projet et faire « Refresh » : le fichier « EcranCuisinier.txt » apparait,
- double cliquer sur le fichier : vous pouvez visualiser les deux commandes,
- attendre entre 11H52 et 11H53 : les deux commandes sont supprimées,
- appuyer sur le carré rouge de la console pour stopper le thread « ThreadViderCommande », puis sur la croix grise.

Partie Graphique

Dans le paquetage *vuegraphique*, télécharger depuis Moodle les classes :

- « FrameCuisinier ». Cette classe est complète et ne sera donc pas à modifier dans ce TP,
- « PanVisualiserCommandeJour ». Cette classe est à compléter comme indiquer ci-dessous.

Instructions

Dans la classe « PanVisualiserCommandeJour » :

- Compléter la méthode `visualiserCommandeJour(int numProfil)` : après l'instruction `initialiserPanel();` reprendre le même code que pour la méthode de même nom dans le boundary « BoundaryVisualiserCommandeJour » (copier/coller),
- Compléter la méthode *propertyChange* reprendre le même code que pour la méthode de même nom dans le boundary « BoundaryVisualiserCommandeJour » (copier/coller). Puis remplacer :

  - l'écriture du fichier :

```
Fichier.ecrire("Commande no " + numeroCommande + " : " + hamburger + ", " + accompagnement
+ ", " + boisson);
```

  par :

```
String texteCommandeJour = ("Commande no " + numeroCommande + " : " + hamburger + ", "
  + accompagnement + ", " + boisson);
JLabel label = new JLabel(texteCommandeJour);
label.setFont(policeParagraphe);
PanVisualiserCommandeJour.mapCommandeJour.put(numeroCommande, texteCommandeJour);
boxCommandes.add(label);
actualiserPanel();
```

  - l'effacement du contenu du fichier :

```
Fichier.effacer();
```

  par :

```
PanVisualiserCommandeJour.mapCommandeJour.clear();
boxCommandes.removeAll();
actualiserPanel();
```

Dans le paquetage *testgraphique* télécharger :

- la classe « TestEcranCuisinier » pour ceux qui ont le cas « commander » en version graphique qui fonctionne,
- la classe « TestEcranCuisinierMiTextuelle » pour ceux qui ont le cas « commander » qui fonctionne mais pas en version graphique,
- la classe « ProtoTestEcranCuisinier » pour ceux dont le cas « commander » en version console ne fonctionne pas.

Tester :

- Modifier dans le thread les minutes pour l'effacement des commandes sur les écrans des cuisiniers.
- Lancer le test :
  - les quatre fenêtres sont superposées : déplacer les,
  - sélectionner dans le menu des deux écrans des cuisiniers (fond gris) Mon compte > Commandes de la journée,
  - passer les commandes. Les clients sont obligés de remplir leur numéro de carte bancaire pour leur première commande, mais pas pour leurs commandes suivantes.
- Une fois le test terminé appuyer sur le carré rouge de la console pour stopper le thread « ThreadViderCommande », puis sur la croix grise.