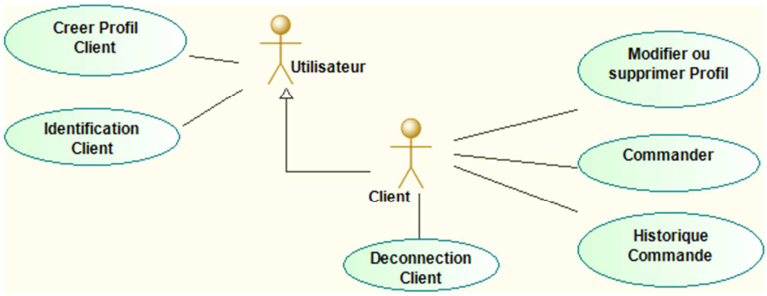
De la conception UML vers son implémentation JAVA

IHM graphique (Swing et AWT)

Dans ce TP nous allons nous intéresser aux IHM graphiques concernant le client. Les différents cas accessibles aux clients ont été donnés dans le diagramme de cas ci-dessous.

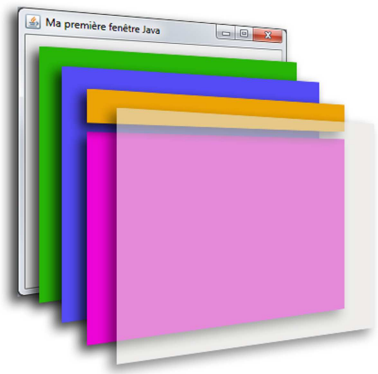


Nous allons développer ensemble l'IHM concernant le cas « Commander ». Une organisation du code est donnée en dernière page.

1. Les Frames

Concept

Une fenêtre graphique est composée de différentes couches.



- la fenêtre ;
- le *RootPane* (en vert), le conteneur principal qui contient les autres composants ;
- le *LayeredPane* (en bleu), qui forme juste un panneau composé du conteneur global et de la barre de menu (MenuBar) ;
- la *MenuBar* (en orange), la barre de menu, quand il y en a une ;
- le *ContentPane* (en rose) : c'est dans celui-ci que nous placerons nos composants ;
- le *GlassPane* (en transparence), couche utilisée pour intercepter les actions de l'utilisateur avant qu'elles ne parviennent aux composants.

Illustration tirée du cours « openclassrooms »[1]

Dans ce TP nous apprendrons à utiliser la fenêtre, le menuBar et le content pane. Ce content pane contiendra un JPanel : composant de type conteneur dont la vocation est d'accueillir d'autres objets de même type ou des objets de type composant (boutons, cases à cocher…).

Réalisation de la fenêtre

Créer un paquetage *vuegraphique*. Créer une classe « FrameClient ».
Depuis le fichier OrganisationCode.txt copier / coller tous les commentaires. Modifier le nom du constructeur « FrameNomActeur » par « FrameClient ».
Cette classe héritera de la classe « JFrame ». Cliquer sur le warning produit par éclipse (ampoule à gauche de la signature de la classe), il vous propose de générer un identifiant. Accepter.
Cette frame n'étant active qu'a la connexion du client, il faudra passer le numéro du client qui c'est authentifié au constructeur de la frame et le placer en tant qu'attribut de la frame **numClient** de type int.
Placer les instructions précédentes sous les commentaires :

```
//Les attributs métiers
//paramètres pour l'initialisation des attributs métier
//initialisation des attributs métiers
```

La classe « FrameClient » héritant de la classe « JFrame » hérite de méthodes dont *setTitle* et *setSize*.
Dans le constructeur, sous le commentaire // mise en forme de la frame (titre, dimension, ...) :

- donner comme titre : BurgerResto (`setTitle("BurgerResto");`)
- donner comme taille 900 par 400.
- Afin que le processus se termine lorsqu'on clique sur la croix rouge de la fenêtre écrire le code suivant :

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

La dernière action que vous ferez dans le constructeur sera de rendre la fenêtre visible avec la méthode *setVisible*.

Créer le paquetage *testgraphique* et à l'intérieur créer la classe « TestEcranClient ».
Dans un main écrire : `new FrameClient(1);`
Lancer le main.

[1] http://fr.openclassrooms.com/informatique/cours/apprenez-a-programmer-en-java/notre-premiere-fenetre

2. *Les Panels*

Concept

Voici le résultat de ce que nous voulons concevoir :

- Un espace où le client peut :
  - passer ses commandes par le menu « Commander »,
  - modifier son profil « ModifierProfil »,
  - consulter l'historique de ses commandes « Historique »,
  - se deconnecter « Deconnection ».

Nous allons créer un panel (**panContents**) qui contiendra les 3 espaces que peut choisir l'utilisateur. Ces trois espaces seront eux-mêmes implémentés par des panels qui seront ajoutés au **panContents**.

Nous allons placer ce dernier dans le « content pane » (en rose dans la figure de la page I).

La méthode *setBackground* permet de modifier la couleur d'un panel qui sera donné en paramètre (ex. : *Color.ORANGE*).

Implémentation

Toujours dans le paquetage *vuegraphique* nous allons créer les 3 Panels en tant que classe afin de ne pas alourdir la frame :
Créer les classes « PanCommander », « PanModifierProfil », et « PanHistorique ».
Depuis le fichier OrganisationCode.txt copier / coller tous les commentaires d'un panel.
Pour chacun des panels modifier :

- le nom du constructeur « PanNomCas » par « PanCommander », « PanModifierProfil », et « PanHistorique ».
- le nom de la méthode *nomCas* par *commander*, *modifierProfil* et *historique* ;

Ces trois classes :

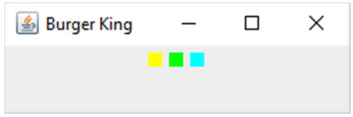
- héritent de la classe « Jpanel ». Cliquer sur le warning produit par éclipse (ampoule à gauche de la signature de la classe), il vous propose de générer un identifiant. Accepter.
- possède la méthode *initialisation*. Pour l'instant compéter cette méthode en coloriant le fond des trois panels respectivement en jaune, vert et cyan (commentaire : // mise en forme du panel (couleur, ...)).

Dans la classe « FrameClient »,

- sous le commentaire // Déclaration et creation des differents panels :
  - déclarer un attribut **panContents** et créer l'objet (de la classe « JPanel »),
  - déclarer les 3 attributs **panCommander**, **panModifierProfil** et **panHistorique** et créer les objets.
- dans le constructeur :
  - Sous le commentaire : // initialisation des differents panels : appel a leur methode d'initialisation
    - Initialiser les 3 panels avec la méthode *initialisation* que vous venez d'implémenter dans les panels,
  - Sous le commentaire : // ajout des pannels dans le ContentPane de la Frame
    - Ajouter les 3 panels au **panContents** :
      `panContents.add(panCommander, "COMMANDER");`
    - Récupérer le « content pane » de la fenêtre et ajouter au « content pane » de la fenêtre le **panContents**. (`getContentPane().add(panContents);`)

Attention la méthode *setVisible* doit rester la dernière instruction du constructeur.

Testez, vous devez obtenir cette fenêtre :
Pas de soucis nous arrangerons tout cela plus tard.



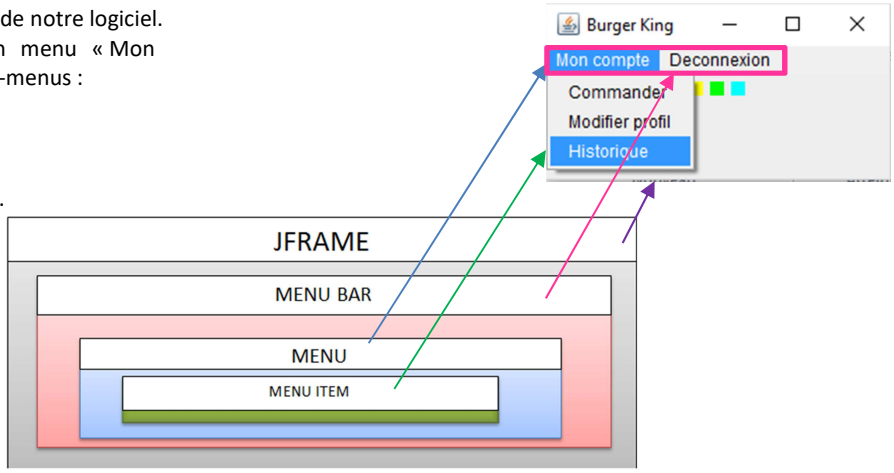
3. *Les MenuBar*

Concept

Nous allons réaliser le Menu de notre logiciel.
Le résultat attendu est un menu « Mon compte » comportant 3 sous-menus :

- Commander
- Modifier profil
- Consulter historique

et un menu « Déconnexion ».



Implémentation

Dans la classe « FrameClient »,

- sous le commentaire // Déclaration et creation de la barre de menu (MenuBar)
  Déclarer un attribut **barreMenu** et créer l'objet (de la classe « MenuBar »).
- Compéter la méthode privée initialisationMenu. Dans cette méthode :
  - Créer les trois sous-menus (de type « MenuItem »),
    ex : `MenuItem commander = new MenuItem("Commander");`
  - Créer le menu du compte : `Menu menuMonCompte = new Menu("Mon compte ");,`
  - Ajouter les trois sous-menus au menu,
  - Créer le menu de déconnexion : `Menu menuDeconnexion = new Menu("Deconnexion");,`
  - Ajouter les menus à la barre de menu **barreMenu**.
- Dans le constructeur sous le commentaire // mise en place du menu
  - Initialiser le menu : `initialisationMenu();`
  - Ajouter la barre de menu à la frame : `setMenuBar(barreMenu);`

  Attention la méthode *setVisible* doit rester la dernière instruction du constructeur.
- Exécuter le main pour vérifier que le menu est bien créé.

4. *Les Cartes*

Concept

Pour ranger les différents espaces à l'intérieur du panel **panContents** nous utiliserons des cartes.
Le principe est de pouvoir gérer nos conteneurs (panels **panCommander**, **panModifierProfil** et **panHistorique**) comme un tas de cartes (les uns sur les autres), et basculer d'un contenu à l'autre en passant par le menu.

Le principe est d'assigner des conteneurs au layout « CardLayout ». La carte visible au départ est la première ajoutée au layout.

Implémentation

Dans la classe « FrameClient »

- Sous le commentaire // Déclaration et creation du gestionnaire des cartes (CardLayout) : Déclarer un attribut **cartes** et créer l'objet (de la classe « CardLayout »).
- Dans le constructeur de la frame, juste sous le commentaire :
  // ajout des pannels dans le ContentPane de la Frame (avant d'ajouter les différents panels au panContents) ajouter au **panContents** le layout de cartes (`panContents.setLayout(cartes);`).

A partir de là, si on lance l'exécution du main de la classe « TestEcranClient », l'affichage obtenu est celui du pan **panCommander**.

5. *Accueil*

Lorsque le client entre dans son espace nous souhaitons afficher un écran d'accueil.
Dans la classe « FrameClient » :

- Sous le commentaire // Déclaration et creation des differents panels, ajouter l'attribut :
```
private JPanel panAccueil = new JPanel();
```
- Compéter la méthode privée initialisationAcceuil avec le code ci-dessous et faire les imports nécessaires :
```
private void initialisationAcceuil(){
        panAccueil.setBackground(Color.ORANGE);
        JLabel texteAccueil = new JLabel("Bienvenu à Burger Resto");
        texteAccueil.setFont(new Font("Calibri", Font.BOLD, 24));
        panAccueil.add(texteAccueil);
        panAccueil.setVisible(true);
        panContents.add(panAccueil, "ECRAN_ACCUEIL");
        cartes.show(panContents, "ECRAN_ACCUEIL");
}
```
- Dans le constructeur, sous le commentaire // mise en page : mises en place des cartes, appeler la méthode initialisationAccueil :
```
initialisationAcceuil();
```

A partir de là, si on lance l'exécution du main de la classe « TestEcranClient »,, l'affichage obtenu est celui du panel d'accueil.

6. *Les Listener*

Concept

Les événements utilisateurs sont gérés par plusieurs interfaces EventListener.
Les interfaces EventListener permettent de définir les traitements en réponse à des événements utilisateurs générés par un composant. Une classe doit contenir une interface auditrice pour chaque type d'événements à traiter :

- ActionListener : clic de souris ou enfoncement de la touche Entrée,
- ItemListener : utilisation d'une liste ou d'une case à cocher,
- MouseListener : événement de souris,
- WindowListener : événement de fenêtre.

Nous allons donc ajouter à chacun des sous-menus un actionListener pour capturer l'évènement que l'utilisateur lui clique dessus.

Implémentation

Pour pouvoir passer d'une carte à l'autre en utilisant le menu nous allons reprendre la méthode *initialisationMenu* de la classe « FrameClient ». Sous chacune des créations des sous-menus (MenuItem commander, historique et modifierProfil) nous allons ajouter un ActionListener (un écouteur d'action).

Voici le code pour le sous-menu "Commander" :

```
commander.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent event) {
      cartes.show(panContents, "COMMANDER");
    }
  });
```

Attention : faire les imports suivants :

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

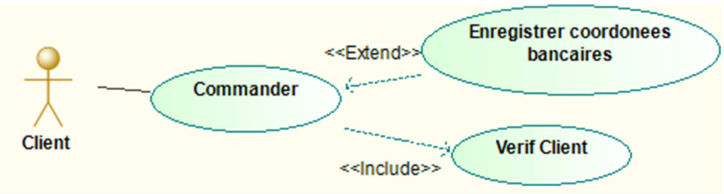
Faire de même pour les sous-menus "historique" et "modifierProfil".
L'action que je dois effectuer lorsque je clique sur le sous-menu "Commander" est de demander au layout **cartes** de montrer la carte contenant le panel "COMMANDER". Veuillez adapter ce code pour les deux autres sous-menus.

Si on lance l'exécution du main de la classe « TestEcranClient », le menu est opérationnel : il permet de changer de cartes grâce au menu.

7. *MVC*

Concept

Chacun des boundary des cas étudiés correspond à un panel. Donc le cas « commander » correspond au panel « panCommander »



Lorsque le client choisit dans le menu "compte" l'item "commander" nous devrons utiliser le contrôleur du cas "commander" c'est-à-dire « ControlCommande ».
Ce cas possède le cas étendu « enregistrer coordonnées bancaires » qui doit échanger des messages avec l'utilisateur. Nous avons vu dans la version console que le boundary « BoundaryCommander » devait donc échanger des messages avec le boundary « BoundaryEnregistrerCoordonneesBancaires ». Il en va de même pour la version graphique : le panel « PanCommander » devra échanger des messages avec le panel « PanEnregistrerCoordonneesBancaires ».

Implémentation

Toujours dans le paquetage *vuegraphique* créer le panel « PanEnregistrerCoordonneesBancaire ».
Depuis le fichier OrganisationCode.txt copier / coller tous les commentaires d'un panel.
Modifier :

- le nom du constructeur « PanNomCas » par « PanEnregistrerCoordonneesBancaire »,
- le nom de la méthode *nomCas* par *enregistrerCoordonneesBancaire*

Sous le commentaire `// controleurs du cas + panel des cas inclus ou etendus en lien avec un acteur` déclarer un attribut controlEnregistrerCoordonneesBancaires de type « ControlEnregistrerCoordonneesBancaires ».

Dans les paramètres d'entrée du constructeur sous le commentaire `// parametres correspondants au controleur du cas + panels // des cas inclus ou etendus en lien avec un acteur` écrire le paramètre controlEnregistrerCoordonneesBancaires de type « ControlEnregistrerCoordonneesBancaires » constructeur.
Dans le corps du constructeur sous le commentaire `// initilaisation du controleur du cas + panels` initialiser l'attribut controlEnregistrerCoordonneesBancaires avec le paramètre d'entrée de même nom.

Dans la classe « PanCommander »
Comme donné dans le code ci-dessous, nous aurons donc deux attributs initialisés par le constructeur :

- L'attribut **controlCommande** de type « ControlCommande »,
- L'attribut **panEnregistrerCoordonneesBancaire** de type « PanEnregistrerCoordonneesBancaires».

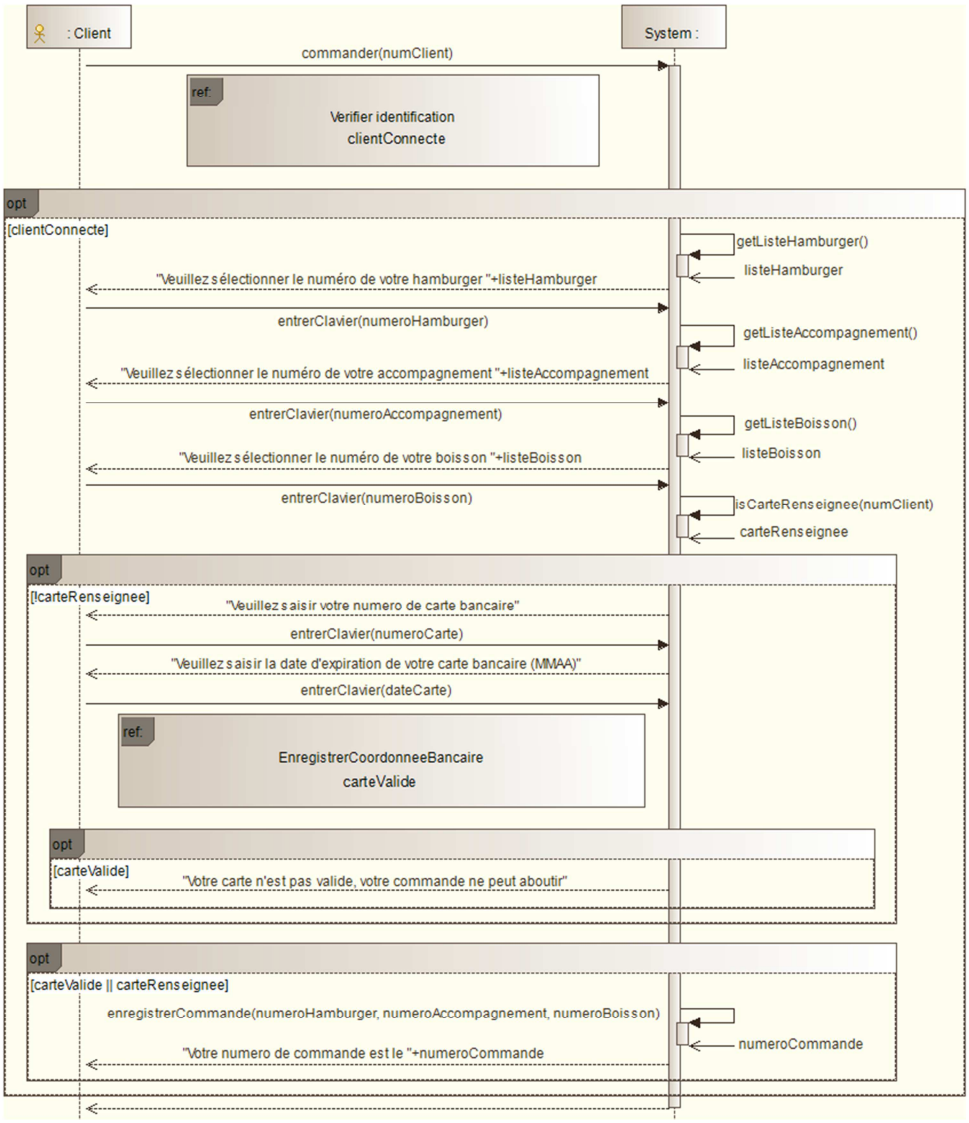
```
public PanCommander(
        // parametres pour l'initialisation des attributs metiers
        // parametres correspondants au controleur du cas + panels
        // des cas inclus ou etendus en lien avec un acteur
        ControlCommander controlCommande, PanEnregistrerCoordonneesBancaires
panEnregistrerCoordonneesBancaire) {
      // initialisation des attributs metiers
      // initilaisation du controleur du cas + panels
      // des cas inclus ou etendus en lien avec un acteur
      this.controlCommande = controlCommande;
      this.panEnregistrerCoordonneesBancaire = panEnregistrerCoordonneesBancaire;
   }
```

Dans la classe « FrameClient », supprimer la création du panel panCommander au niveau des attributs et l'ajouter dans le constructeur :

```
// Déclaration et creation des differents panels
   private PanCommander panCommander;
(…)
   // Le constructeur
   public FrameClient(
         // parametres pour l'initialisation des attributs metiers
         int numClient,
         // parametres correspondants aux controleurs des cas utiliser par
         // l'acteur en relation avec cette frame
         ControlCommander controlCommande,
         ControlEnregistrerCoordonneesBancaires controlEnregistrerCoordonneesBancaires
   ) {
(…)
      // initialisation des differents panels : appel a leur methode d'initialisation
      PanEnregistrerCoordonneesBancaires panEnregistrerCoordonneesBancaire
         = new PanEnregistrerCoordonneesBancaires(controlEnregistrerCoordonneesBancaires);
      panCommander = new PanCommander(controlCommande, panEnregistrerCoordonneesBancaire);
```

Télécharger depuis Moodle la classe « TestEcranClient » et l'adapter à votre code. Tester (Lorsque dans le menu « Mon compte » vous sélectionnez « Commander » aucune erreur ne doit s'afficher dans la console).

8. *Diagramme de séquence système « Commander »*



Concept

Lorsque je clique sur le sous-menu "Commander" le layout **cartes** montre la carte contenant le panel "COMMANDER".
Il faut donc compléter dans le panel « PanCommander » la méthode `commander(int numClient)` qui devra suivre le diagramme de séquence System ci-dessus.

Dans le diagramme la méthode *commander* possède un paramètre d'entrée numClient qui doit être stocké dans le boundary. Il faudra aussi stocker les différentes entrées utilisateur : numeroHamburger, numeroAccompagnement et numeroBoisson.

Implémentation

Dans la classe « PanCommander » sous le commentaire `// les attributs metiers (ex : numClient)` déclarer et créer les attributs numeroHamburger, numeroAccompagnement et numeroBoisson. Les initialiser à 0.

Sous le commentaire `// Methodes privees pour le bon deroulement du cas`, créer la méthode privée `affichageMenu()` , cette méthode récupère les listes d'hamburger, d'accompagnement et de boisson.
Exemple :
List<String> : `List<String> listeHamburger = controlCommande.getListHamburger();`

Compléter la méthode `commander(int numClient)` du panel « PanCommander », cette méthode :

- Initialise l'attribut **numeroClient** avec le paramètre numClient,
- Implémenter la vérification de l'identification grâce à son contrôleur **controlCommande**,
- Si le client est bien identifié alors appelle la méthode privée affichageMenu.

Etant donné qu'il faut maintenant présenter cette liste à l'utilisateur on s'arrêtera là pour l'instant.

Dans la classe « FrameClient » dans la méthode `initialisationMenu()` de « FrameClient » juste avant le changement de carte appeler la méthode commander que l'on vient de créer :

```
MenuItem commander = new MenuItem("Commander");
   commander.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent event) {
     panCommander.commander(numClient);
     cartes.show(panContents, "COMMANDER");
    }
   });
```

Nous allons passer à la mise en page de la page de la prise de commande.

9. *Les Labels*

Concept

La classe Label sert à afficher une étiquette qui tient sur une seule ligne. On peut agir sur sa police, mettre en forme le texte (souligné, italique, gras), changer sa couleur…

Implémentation

Dans la classe « PanCommander », sous le commentaire `// Declaration et creation des polices d'ecritures`, créer les attributs :

- **policeTitre** de la classe « Font » : `"Calibri", Font.BOLD, 24,`
- **policeParagraphe** de la classe « Font » : `"Calibri", Font.HANGING_BASELINE, 16,`

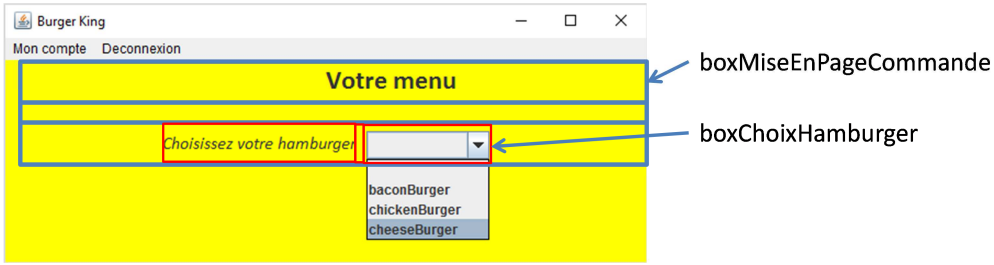
Dans la méthode initialisation, sous le commentaire `// creation des differents elements graphiques (JLabel, Combobox, Button, // TextAera ...)` , créer :

- la variable `texteCommander` de la classe « JLabel » et qui a pour texte : "Votre menu". Affecter-lui la police **policeTitre** (méthode `setFont`),
- la variable `texteHamburger` de la classe « JLabel » et qui a pour texte : "Choisissez votre hamburger". Affecter-lui la police **policeParagraphe.**

10. Les Box

Concept

Une box permet de ranger les composants à la suite soit sur une ligne, soit sur une colonne. Une box (par exemple verticale) peut contenir d'autres box (par exemple horizontale).
Dans notre panel, nous aurons les box suivantes :



Implémentation

Dans la classe « PanCommander », sous le commentaire `// Mise en page : les Box`, créer :

- l'attribut **boxMiseEnPageCommande** comme une box verticale
  `(Box boxMiseEnPageCommande = Box.createVerticalBox();).`
- l'attribut **boxChoixHamburger** comme une box horizontale.

Dans la méthode `initialisation`, sous le commentaire : `// mise en page : placements des differents elements graphiques dans des Box`, ajouter :

- le label `texteCommander` à la box **boxMiseEnPageCommande**,
- le label `texteHamburger` à la box **boxChoixHamburger**.

Pour ajouter un espace entre les deux titres il vous faut placer entre l'ajout des labels l'instruction suivante :

```
boxMiseEnPageCommande.add(Box.createRigidArea(new Dimension(0, 30)));
```

30 étant la hauteur de l'espace que vous voulez ajouter.
Sous le commentaire : `// mise en page : placements des differentes box dans une box principale`, ajouter la box **boxChoixHamburger** à la box **boxMiseEnPageCommande.**
Sous le commentaire : `// mise en page : ajout de la box principale dans le panel`, ajouter la box au pannel : `this.add(boxMiseEnPageCommande);`

Vous pouvez tester l'affichage de vos deux textes (la liste déroulante n'apparait pas encore).

11. Les ComboBox

Concept

La classe « JComboBox » est une classe qui permet d'afficher une liste déroulante. Nous allons affichez dans notre liste déroulante les différents hamburgers récupérer par le contrôleur.

On verra dans le chapitre suivant comment créer valider son choix en appuyant sur le bouton "Valider Menu".

Implémentation

Dans la classe « PanCommander », sous le commentaire `// Declaration et creation des ComboBox`, déclarer et créer les combobox de type JComboBox, exemple :

```
private JComboBox<String> comboBoxHamburger = new JComboBox<>();
```

Dans la méthode `initialisation` :

- sous le commentaire : `// creation des differents elements graphiques (JLabel, Combobox, Button, // TextAera ...)`, juste après avoir déclaré les deux labels texteCommander et texteHamburger ajouter un listener pour savoir quand l'utilisateur choisi un item avec les instructions suivantes :

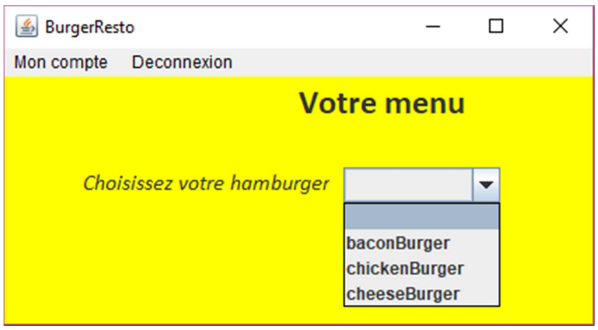
```
comboBoxHamburger.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        numeroHamburger = comboBoxHamburger.getSelectedIndex();
    }
});
```

- sous le commentaire `// mise en page : placements des differents elements graphiques dans des Box`, dans la box boxChoixHamburger ajouter la comboBox comboBoxHamburger.

Dans la méthode `afficherMenu`, après avoir récupérer la liste des hamburgers ajouter :

- une instruction supprimant tous les items existants : `comboBoxHamburger.removeAllItems();`
- un item vide : `comboBoxHamburger.addItem("");`
- les différents items (*addItem*) en faisant une boucle sur la liste des hamburgers,

Vous pouvez tester l'affichage de votre liste.

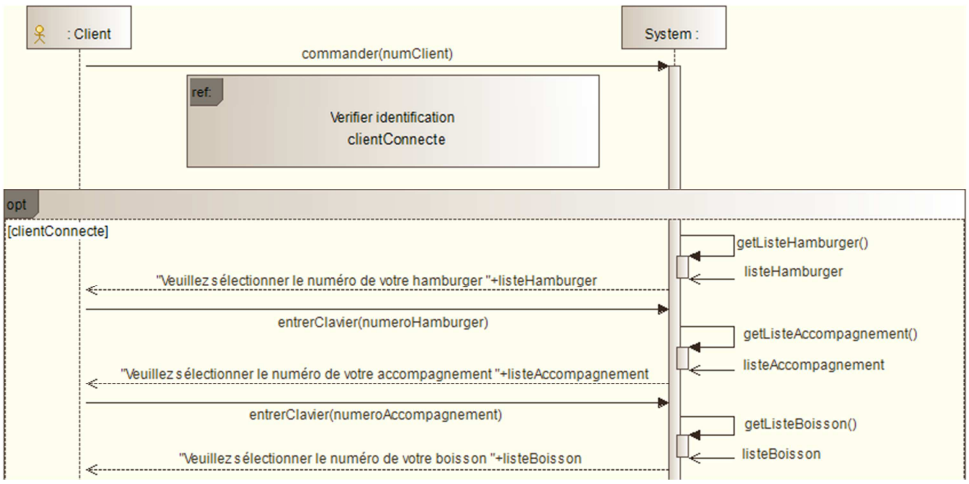


Pour ajouter un espace entre le texte et la liste déroulante il vous faut placer entre l'ajout des labels l'instruction suivante :

```
boxChoixHamburger.add(Box.createRigidArea(new Dimension(10, 0)));
```

10 étant la largeur de l'espace que vous voulez ajouter.

12. Affichages du cas commander



Suivre la même logique pour récupérer les listes des accompagnements et des boissons, et pour les afficher.

13. Les Button

Concept

La classe « JButton » permet de créer des « boutons ». Ces boutons peuvent contenir des images ou du texte.
Le bouton "Valider" permettra de confirmer les différents aliments composant le menu.

Implémentation

Les modifications de cette partie se font dans la classe « PanCommander ».

- Sous le commentaire `// Declaration et creation des Button`, créer en attribut un nouveau bouton : `private JButton validerCommande = new JButton();`
- Sous le commentaire `// Declaration et creation des Button`, créer en attribut une nouvelle Box : **boxValiderChoix**,

Dans la méthode initialisation :

- à la fin du code correspondant au commentaire `// creation des differents elements graphiques (JLabel, Combobox, Button, // TextAera ...)` :
  - ajouter à l'objet `validerCommande` le texte "Valider" (setText),
  - ajouter à l'objet `validerCommande` un listener pour savoir quand l'utilisateur cliquera sur le bouton avec les instructions suivantes :

```
validerCommande.setText("Valider");
  validerCommande.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
     if (       numeroHamburger != 0 && numeroAccompagnement != 0
                && numeroBoisson != 0) {
                System.out.println("OK");;
     }
   }
  });
```

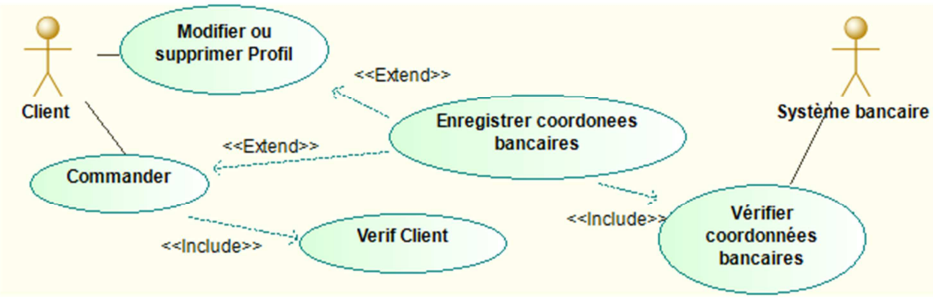
- à la fin du code correspondant au commentaire `// mise en page : placements des differents elements graphiques dans des Box` : ajouter l'objet `validerCommande` à la box **boxValiderChoix**,
- à la fin du code correspondant au commentaire `// mise en page : placements des differentes box dans une box principale` : ajouter la box **boxValiderChoix** à la box **boxMiseEnPageCommande**.

Tester : si vous appuyez sur le bouton Valider ET que votre commande est complète (sélection d'un hamburger, d'un accompagnement et d'une boisson) alors apparait dans la console la chaîne « OK », s'il manque un aliment alors rien ne s'affiche.

14. Cas « enregistrer coordonnées bancaires »

Mettons de côté le cas « commander » pour nous intéresser au cas « enregistrer coordonnées bancaires ».

Les modifications de cette partie se font **dans la classe « PanEnregistrerCoordonneesBancaires ».**



Comme pour les panels panCommander, panHistorique et panModifierProfil créer dans la classe « PanEnregistrerCoordonneesBancaires ».

Créer les attributs suivants :

```
// les attributs metiers (ex : numClient)
private int numClient;
// Mise en page : les Box
private Box boxMiseEnPageCoordonneesBancaires = Box.createVerticalBox();
private Box boxNumeroCarte = Box.createHorizontalBox();
private Box boxValiditeCarte = Box.createHorizontalBox();
private Box boxValiderCarte = Box.createHorizontalBox();
```

Pour la mise en forme du texte de ce panel, créer sous le commentaire `// Declaration et creation des polices d'ecritures` les deux attributs suivants de la classe « Font » :

- **policeTitre** de la classe « Font » : `"Calibri", Font.BOLD, 24,`
- **policeParagraphe** de la classe « Font » : `"Calibri", Font.HANGING_BASELINE, 16,`

Compléter la méthode `initialisation()` :

- sous le commentaire `// mise en forme du panel (couleur, ...)`, définir le fond du panel en jaune (`Color.YELLOW`)

- sous le commentaire `// creation des differents elements graphiques (JLabel, Combobox, Button, TextAera ...)` :

  - créer le label texteCoordonnerBancaire et lui affecter le texte "Entrer vos coordonnées bancaires",
  - lui affecter la police **policeTitre** (*setFont*),
  - l'ajouter dans la box **boxMiseEnPageCoordonneesBancaires**

  - créer le label texteNumeroCarteBancaire et lui affecter le texte "Entrer le numéro de votre carte bancaire",
  - lui affecter la police **policeParagraphe** (*setFont*),
  - l'ajouter dans la box **boxNumeroCarte**.

  - créer le label texteValiditeCarte et lui affecter le texte "Entrer la date d'expiration de votre carte bancaire",
  - lui affecter la police **policeParagraphe** (*setFont*),
  - l'ajouter dans la box **boxValiditeCarte**.

  - Ajouter le bouton de validation :

```
JButton validationCoordonneeBancaire = new JButton();
   validationCoordonneeBancaire.setText("Valider");
   validationCoordonneeBancaire.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
                System.out.println("OK");
         }
   });
```

- sous le commentaire `// mise en page : placements des differents elements graphiques dans des Box` : ajouter les différents éléments graphiques texteNumeroCarteBancaire, texteValiditeCarte et validationCoordonneeBancaire respectivement dans les box **boxNumeroCarte**, **boxValiditeCarte** et **boxValiderCarte**

- sous le commentaire `// mise en page : placements des differentes box dans une box principale` : Ajouter les box **boxNumeroCarte**, **boxValiditeCarte** et la box **boxValiderCarte** dans la box **boxMiseEnPageCoordonneesBancaires**.

- sous le commentaire `// mise en page : ajout de la box principale dans le panel` :
  Ajouter la box **boxMiseEnPageCoordonneesBancaires** dans le panel, la rendre visible mais NE PAS rendre le panel visible immédiatement car il le sera seulement quand le cas « commander » l'appellera :

```
this.add(boxMiseEnPageCoordonneesBancaires);
boxMiseEnPageCoordonneesBancaires.setVisible(true);
this.setVisible(false);
```

15. *Les TextArea*

Concept

La classe « JTextArea » est une classe qui permet d'entrer, ou d'afficher du texte. Nous allons implémenter deux aires de texte, un vide pour que l'utilisateur entre son numéro de carte bancaire et un autre où seront affichées les lettres MMAA. L'utilisateur pourra y entrer le mois et l'année de péremption de sa carte bancaire.

Implémentation

Les modifications de cette partie se font **dans la classe « PanEnregistrerCoordonneesBancaires ».**

Sous le commentaire `// Declaration et creation des polices d'ecritures` créer les deux attributs suivants de la classe « Font » :

- **policeAremplacer** : "Arial", Font.ITALIC, 12
- **policeChoixUtilisateur** : "Arial", Font.TRUETYPE_FONT, 12

Sous le commentaire `// Declaration et creation des TextArea` créer les deux attributs permettant de récupérer le numéro de la carte bancaire et sa date de validité :

```
private TextArea textAeraNumeroCarte = new TextArea();
private TextArea textAreaDateExpiration = new TextArea();
```

Dans la méthode d'initialisation :

- à la fin du code correspondant au commentaire `// creation des differents elements graphiques (JLabel, Combobox, Button, TextAera ...)` configurer l'aire de texte **textAreaNumeroCarte** en lui donnant une taille maximum (`setMaximumSize(new Dimension(120,20))`.
- sous le commentaire `// mise en page : placements des differents elements graphiques dans des Box` placer l'aire de texte **textAreaNumeroCarte** dans la box **boxNumeroCarte**.
- à la fin du code correspondant au commentaire `// creation des differents elements graphiques (JLabel, Combobox, Button, TextAera ...)` configurer l'aire de texte **textAreaDateExpiration** en lui donnant les caractéristiques suivantes :
  - la couleur grise (*setForeground*),
  - une taille maximum (`setMaximumSize(new Dimension(60,20))`
  - la police **policeAremplacer** (*setFont*).

Il faut maintenant ajouter un listener sur le clic souris pour vider l'air de texte et que l'utilisateur puisse écrire en noir et dans la police **policeChoixUtilisateur**.
Toujours dans la méthode d'initialisation écrire les lignes suivantes à la suite du code précédent :

```
textAreaDateExpiration.addMouseListener(new MouseListener() {

    public void mouseReleased(MouseEvent arg0) {}
    public void mousePressed(MouseEvent arg0) {}
    public void mouseExited(MouseEvent arg0) {}
    public void mouseEntered(MouseEvent arg0) {}
    public void mouseClicked(MouseEvent arg0) {
      textAreaDateExpiration.setText(null);
      textAreaDateExpiration.setFont(policeChoixUtilisateur);
      textAreaDateExpiration.setForeground(Color.black);

    }
  });
```

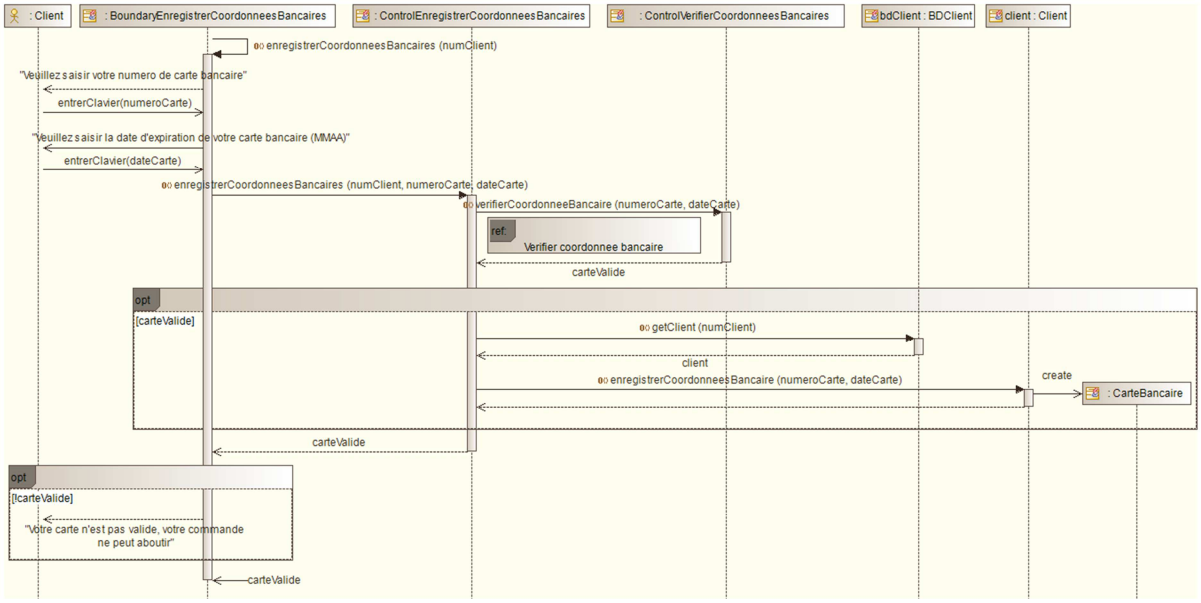
Sous le commentaire `// mise en page : placements des differents elements graphiques dans des Box` placer l'aire de texte **textAreaDateExpiration** dans la box **boxValiditeCarte**.

Enfin il nous faut compléter la méthode d'entrée dans le cas : enregistrerCoordonneesBancaires. Lorsque le cas débute rien ne doit être écrit dans l'aire où l'utilisateur doit entrer le numéro de sa carte bancaire et la chaine MMAA doit être écrite dans le champs où l'utilisateur entre le mois et l'année de la validité de sa carte.

```
public void enregistrerCoordonneesBancaires(int numClient) {
   this.numClient = numClient;
   textAeraNumeroCarte.setText("");
   textAreaDateExpiration.setText("MMAA");
   this.setVisible(true);
   this.repaint();
 }
```

Télécharger le test « TestEnregistrerCoordonnerBancaire » pour tester la « forme » de votre panel. ATTENTION ce test n'a aucune valeur de test fonctionnel du cas, d'ailleurs il se peut que vous ne pouviez pas entrer la date de péremption… Ce n'est pas grave, passer à la suite !

16. Cas « enregistrer coordonnées bancaires » suite



Le cas « Enregistrer coordonnées bancaire » appelle la méthode *enregistrerCoordonneesBancaire* de son contrôleur **controlEnregistrerCoordonneesBancaires** après que l'utilisateur ait validé ses coordonnées bancaires en appuyant sur le bouton valider. Il nous faut donc une méthode privée à appeler quand on appuie sur le bouton **validationCoordonneeBancaire**.

Les modifications de cette partie se font **dans la classe « PanEnregistrerCoordonneesBancaires »**.

Sous le commentaire `// Methodes privees pour le bon deroulement du cas`, créer la méthode privée traitementCoordonneesBancaires :

```
private void traitementCoordonneesBancaires(int numeroCarte,
    int dateCarte) {
  boolean carteValide = controlEnregistrerCoordonneesBancaires
    .enregistrerCoordonneesBancaires(numClient, numeroCarte, dateCarte);
}
```

Dans la méthode *initialisation* reprendre la configuration du bouton **validationCoordonneeBancaire** en appelant la méthode *traitementCoordonneesBancaires* que l'on vient d'écrire. Remplacer l'affichage de la chaine « OK » par la partie encadrée :

```
JButton validationCoordonneeBancaire = new JButton();
  validationCoordonneeBancaire.setText("Valider");
  validationCoordonneeBancaire.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      int numeroCarte = Integer.valueOf(textAeraNumeroCarte.getText());
      int dateCarte = Integer.valueOf(textAreaDateExpiration
          .getText());
      traitementCoordonneesBancaires(numeroCarte, dateCarte);
    }
  });
```

Nous ne prendrons pas en compte la dernière partie du diagramme de séquence c'est-à-dire la frame opt car dans notre application, ne pouvant réellement vérifier le numéro de la carte bancaire, la carte sera toujours valide.

Par contre le cas retourne le boolean **carteValide** récupéré de l'appel à méthode de son contrôleur. Nous devons donc stocker l'objet (le panel) qui à appeler le cas afin de lui retourner le boolean. Le problème est comment stocker ce panel alors que nous ne connaissons pas son type (ici il peut s'agir de « PanCommander » ou « PanModifierProfil »). Et comment lui envoyer ?
Pour cela nous allons créer une interface « IUseEnregistrerCoordonneesBancaires » que devront implémenter les panels qui utilisent le panel « PanEnregitrerCoordonneesBancaires ». Cette interface n'aura qu'une méthode permettant d'envoyer la valeur de retour du cas.

Créer l'interface :

```
public interface IUseEnregistrerCoordonneesBancaires {
        public void retourEnregistrerCoordonneesBancaire(boolean carteValide);
}
```

Sous le commentaire `// controleurs du cas + panel des cas inclus ou etendus en lien avec un acteur`, créer l'attribut permettant de stocker le panel appelant :

```
private IUseEnregistrerCoordonneesBancaires panAppelant;
```

Modifier la méthode enregistrerCoordonneesBancaires pour qu'elle stocke l'appelant :

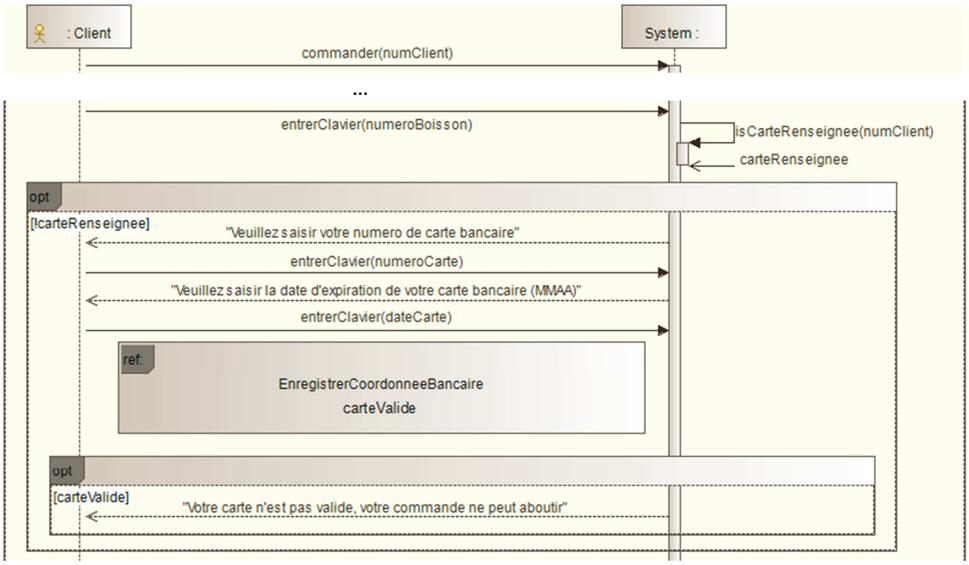
```
public void enregistrerCoordonneesBancaires(int numClient,
     IUseEnregistrerCoordonneesBancaires panAppelant) {
   this.numClient = numClient;
   this.panAppelant = panAppelant;
…
}
```

A la fin de la méthode traitementCoordonneesBancaires envoyer le booléen carteValide (retourné par le cas « enregistrer coordonnées bancaire ») à l'appelant :

```
panAppelant.retourEnregistrerCoordonneesBancaire(carteValide);
```

17. Cas Etendu « Enregistrer coordonnées bancaire »

Reprenons le cas « Commander », plus précisément le moment où l'on doit appeler le boundary « BoundaryEnregistrerCoordonneesBancaires ».



Dans la classe « PanCommander ».

- Pour que le panel « PanCommander » puisse appeler le panel « PanEnregistrerCoordonneesBancaires » il faut qu'il implémente l'interface IUseEnregistrerCoordonneesBancaires. Compléter la signature de la classe.
- Sous le commentaire `// Methodes privees pour le bon deroulement du cas`, écrire les méthodes
  - La méthode retourEnregistrementCoordonneesBancaire :

```
@Override
 public void retourEnregistrerCoordonneesBancaire(boolean carteValide) {
 }
```

  - Créer une méthode privée validationCartePayement(),
  - Créer une méthode privée enregistrerCommande(boolean carteRenseignee).
- Dans la méthode initialisation, et plus précisément dans le listener ajouté au bouton de validation de la commande, remplacer l'affichage de la chaine « OK » par l'appel à la méthode validationCartePayement.
- Compléter la méthode validationCartePayement selon le diagramme de séquence système :
  - Appel à la méthode isCarteRenseignee du contrôleur « controlCommander »,
  - Si la carte n'est pas renseignée appeler le cas étendu « PanEnregistrerCoordonneesBancaires » sinon appeler la méthode privée enregistrerCommande qui suivra la dernière partie du diagramme de séquence :

```
if (!carteRenseignee) {
     boxMiseEnPageCommande.setVisible(false);
     panEnregistrerCoordonneesBancaire.setVisible(true);
     this.repaint();
     panEnregistrerCoordonneesBancaire
        .enregistrerCoordonneesBancaires(numClient, this);
} else
     this.enregistrerCommande(carteRenseignee);
}
```

  - Traiter le retour du cas « Enregistrer coordonnées bancaires »

```
@Override
 public void retourEnregistrerCoordonneesBancaire(boolean carteValide) {
   this.panEnregistrerCoordonneesBancaire.setVisible(false);
   if(carteValide) this.enregistrerCommande(carteValide);
 }
```

Dans le constructeur de **la classe « ClientFrame »** :

- Lancer l'initialisation du panel **panEnregistrerCoordonneesBancaires** (partie encadrée) :

```
panCommander.initialisation();
panHistorique.initialisation();
panModifierProfil.initialisation();
panEnregistrerCoordonneesBancaires.initialisation();
```

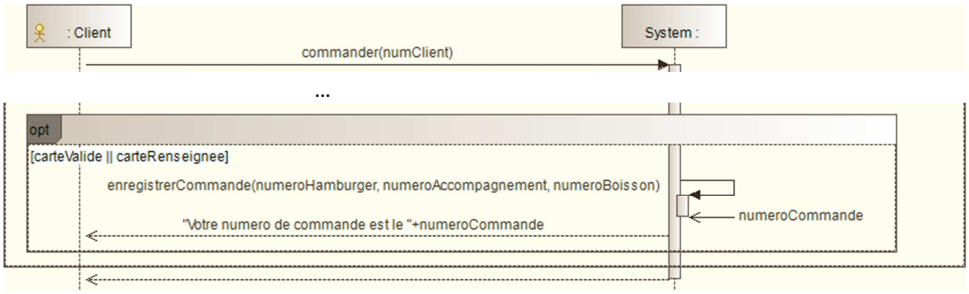
- Ajouter le panel **panEnregistrerCoordonneesBancaires** au panel **panCommander** (partie encadrée) :

```
panContents.add(panCommander, "COMMANDER");
panCommander.add(panEnregistrerCoordonneesBancaires,
            "ENREGISTRER_COORDONNEES_BANCAIRE");
panContents.add(panModifierProfil, "MODIFIER_PROFIL");
panContents.add(panHistorique, "HISTORIQUE");
```

Tester votre travail (TestEcranClient). Le cas étendu doit fonctionner (même pour la date de validité).

18. Fin du cas « Commander »

Il ne reste plus que la fin du cas du cas « commander »



Les modifications de cette partie se font **dans la classe « PanCommander »** : l'affichage du numéro de la commande.

Pour cela :

- Créer les attributs suivants :

```
// Declaration et creation des Labels
private JLabel numeroCommande = new JLabel();
// Mise en page : les Box
private Box boxMiseEnPageNumeroCommande = Box.createVerticalBox();
```

- Dans la méthode initialisation, à la fin du code correspondant au commentaire `// creation des differents elements graphiques (JLabel, Combobox, Button, // TextAera ...)`, créer le label texteNumeroCommandeTitre, mettre le texte en forme et placer le tout dans les box :

```
JLabel texteNumeroCommandeTitre = new JLabel("Votre commande");
   texteNumeroCommandeTitre.setFont(policeTitre);

   numeroCommande.setFont(policeParagraphe);

   boxMiseEnPageNumeroCommande.add(numeroCommande);
   boxMiseEnPageNumeroCommande.add(texteNumeroCommandeTitre);
   boxMiseEnPageNumeroCommande.add(Box.createRigidArea(new Dimension(0, 30)));
   numeroCommande.setFont(policeParagraphe);
   boxMiseEnPageNumeroCommande.add(numeroCommande);
   this.add(boxMiseEnPageNumeroCommande);
```

- Au tout début du la méthode commander gérer la visibilité des box :

```
 public void commander(int numClient) {
   boxMiseEnPageCommande.setVisible(true);
   boxMiseEnPageNumeroCommande.setVisible(false);
(…)
```

- Enfin compléter la méthode enregistrerCommande :

```
private void enregistrerCommande(boolean carteRenseignee) {
   if (carteRenseignee) {
     int numCommande =controlCommande.enregistrerCommande(numClient, numeroHamburger,
        numeroAccompagnement, numeroBoisson);
     numeroCommande.setText("Votre numero est : " + numCommande);
   }
   this.setVisible(true);
   boxMiseEnPageCommande.setVisible(false);
   boxMiseEnPageNumeroCommande.setVisible(true);
   this.repaint();
 }
```

19. Organisation des classes graphiques

La frame

```
public class FrameNomActeur extends JFrame {
   // Les attributs metiers (ex : numClient)
   // Declaration et creation des elements graphiques (JLabel)
   // Declaration et creation de la barre de menu (MenuBar)
   // Declaration et creation des differents panels
   // Declaration et creation du gestionnaire des cartes (CardLayout)

   // Le constructeur
   public FrameNomActeur (
         // parametres pour l'initialisation des attributs metiers
         // parametres correspondants aux controleurs des cas utiliser par
         // l'acteur en relation avec cette frame
   ) {
      // initialisation des attributs metiers
      // mise en forme de la frame (titre, dimension, ...)
      // initialisation des differents panels : appel a leur methode d'initialisation
      // ajout des pannels dans le ContentPane de la Frame
      // mise en page : mises en place des cartes
      // mise en place du menu
      // appel a la methode d'initialisation du menu
      // appel a la methode d'initialisation de la page d'accueil (optionnel)

      this.setVisible(true);
   }

   private void initialisationAcceuil(){
   }

   public void initialisationMenu() {
   }
}
```

Le Panel

```
public class PanNomCas extends JPanel {
   // controleurs du cas + panel des cas inclus ou etendus en lien avec un acteur
   // les attributs metiers (ex : numClient)

   // Les elements graphiques :
   // Declaration et creation des polices d'ecritures
   // Declaration et creation des ComboBox
   // Declaration et creation des Button
   // Declaration et creation des TextArea
   // Declaration et creation des Labels

   // Mise en page : les Box

   public PanNomCas (
         // parametres pour l'initialisation des attributs métiers
         // parametres correspondants au controleur du cas + panel des cas inclus ou etendus
         // en relation avec un acteur
      ) {
      // initialisation des attributs metiers
      // initilaisation du controleur du cas + panels
      // des cas inclus ou etendus en lien avec un acteur
   }

   //Méthode d'initialisation du panel
   public void initialisation() {
      // mise en forme du panel (couleur, …)
      // creation des différents elements graphiques (JLabel, Combobox, Button, TextAera …)
      // mise en page : placements des differents elements graphiques dans des Box
      // mise en page : placements des differentes box dans une box principale
      // mise en page : ajout de la box principale dans le panel
   }

   // Methode correspondante au nom du cas
   public void nomCas( /*parametres metiers*/ ) {
   }

   // Methodes privées pour le bon deroulement du cas
}
```