IUT nancy Charlemagne Département Informatique

Conception et programmation objet avancées
2018-2019

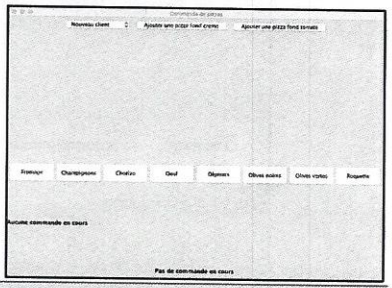
TP Noté

Jeudi 8 novembre 2018

Durée 3h45 – sortie non autorisée avant 2h –
Documents non autorisés dans la première partie (conception)

COMMANDE DE PIZZAS SELON L'ARCHITECTURE MVC

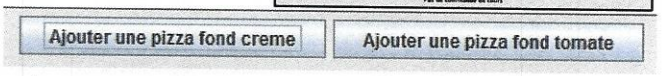
Un premier document papier est à rendre au bout de 45 minutes, pendant cette période les ordinateurs ne seront pas utilisés. Le reste du devoir sera réalisé sur machine et l'ensemble des fichiers devra être déposé sur arche dans trois fichiers archives.



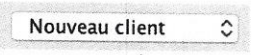
L'objectif de cet exercice est de programmer une application visuelle de commande de pizzas selon l'architecture MVC dont l'interface graphique sera similaire à celle présentée ci-contre.

DEROULEMENT DE LA COMMANDE :

A l'ouverture de l'interface, cinq zones sont visibles :



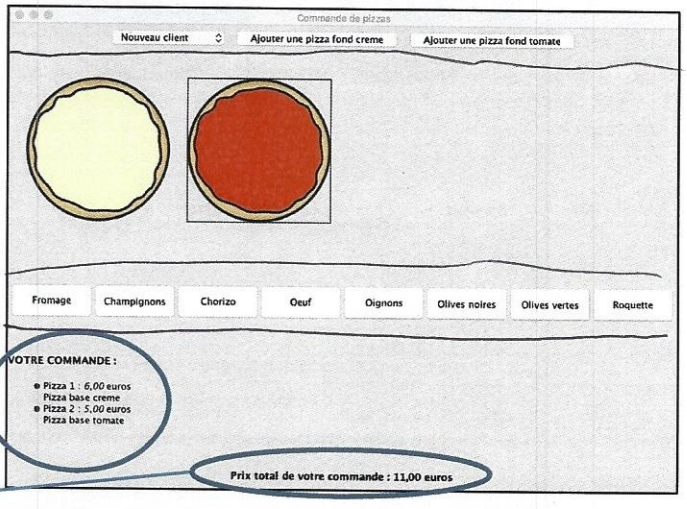
- Zone 1 : Une liste de choix (JComboBox) et deux boutons dans un JPanel situé en haut de l'interface pour ajouter des pizzas (4 maximum) avec une base crème ou tomate. La liste de choix permet d'indiquer en terme de fidélité, le statut du client qui passe la commande : « Nouveau client », « Client avec carte », « Client adhérent ». A chacun de ces choix correspond un taux de réduction du prix de la commande : pas de réduction pour un nouveau client, 10% de réduction pour un client avec carte et 30% pour un client adhérent.

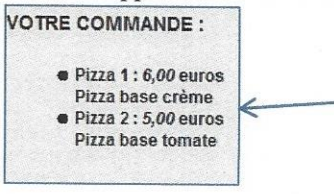


- Zone 2 : Une zone vide (JPanel) qui contiendra les images des pizzas commandées.
- Zone 3 : Un JPanel avec 8 boutons correspondant aux ingrédients pouvant être ajoutés à la pizza base crème ou tomate :

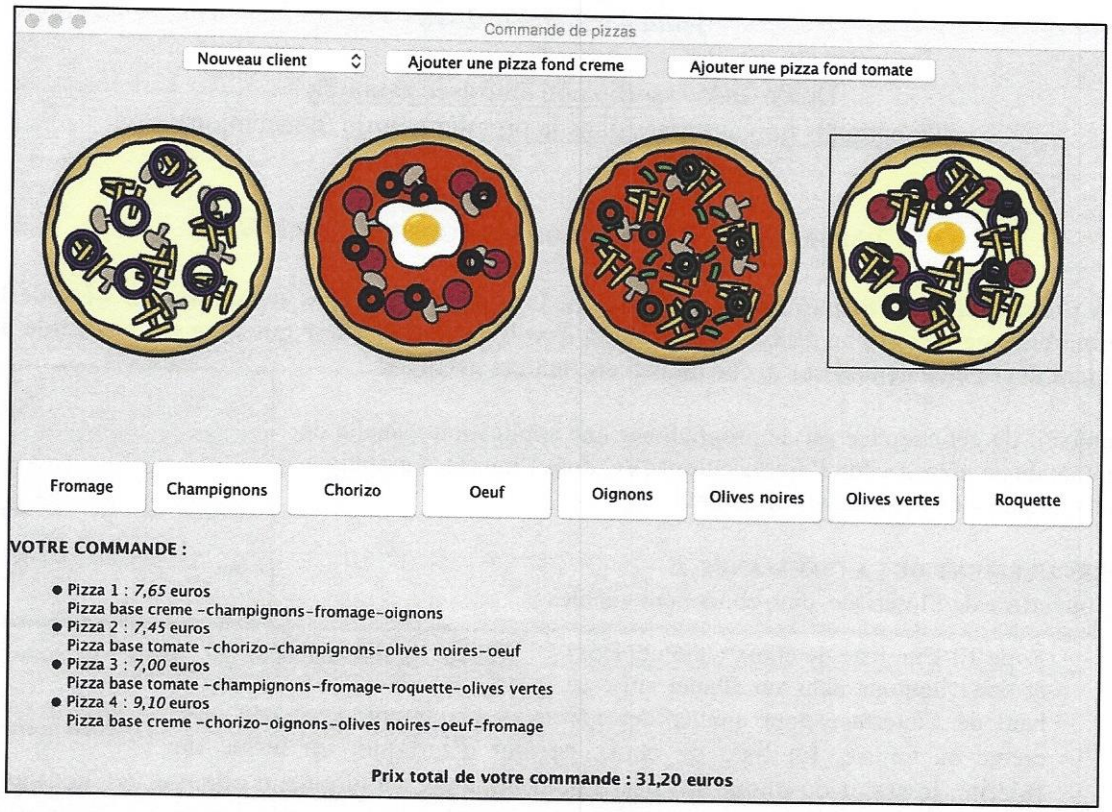
- Zone 4 : Un JLabel contenant les informations de la commande.
- Zone 5 : Un JLabel affichant le prix de la commande et à l'ouverture le texte « Pas de commande en cours ».

Ensuite, les pizzas sont ajoutées par des clics sur les boutons de la zone 1 et les images des pizzas de base choisies (**au maximum 4**) apparaissent dans la zone 2. Les informations et le prix associés aux pizzas de base apparaissent dans les zones 4 et 5.

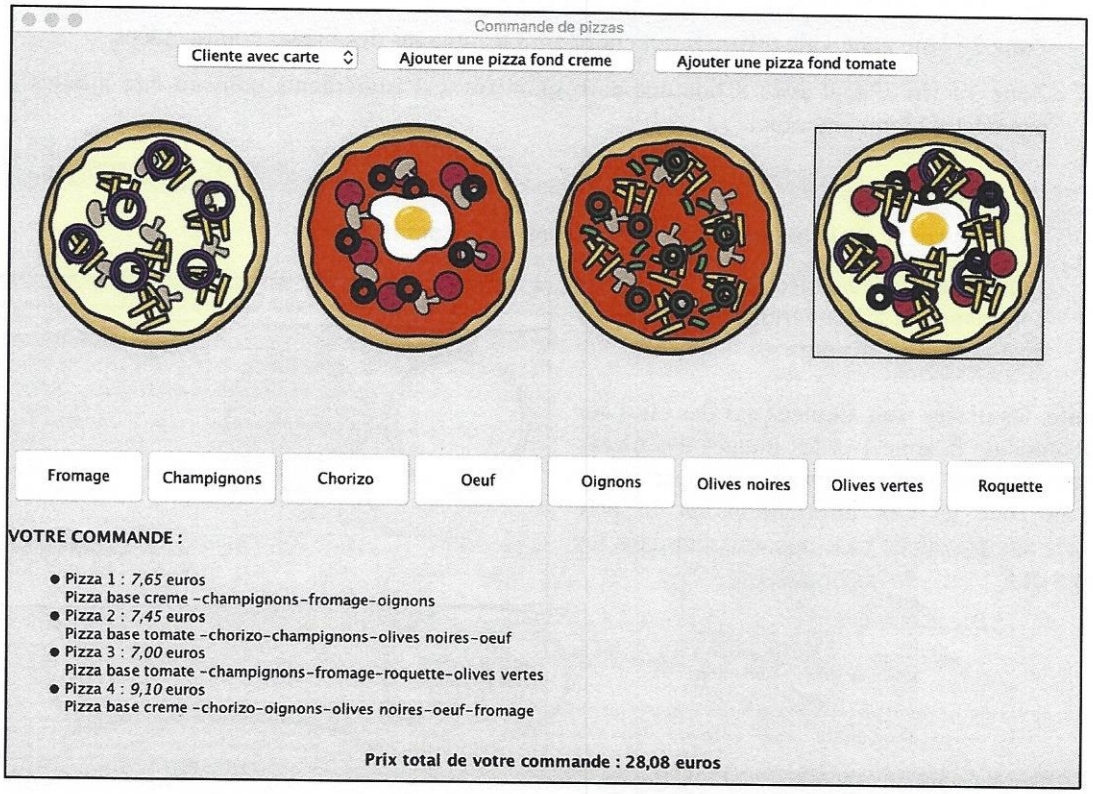




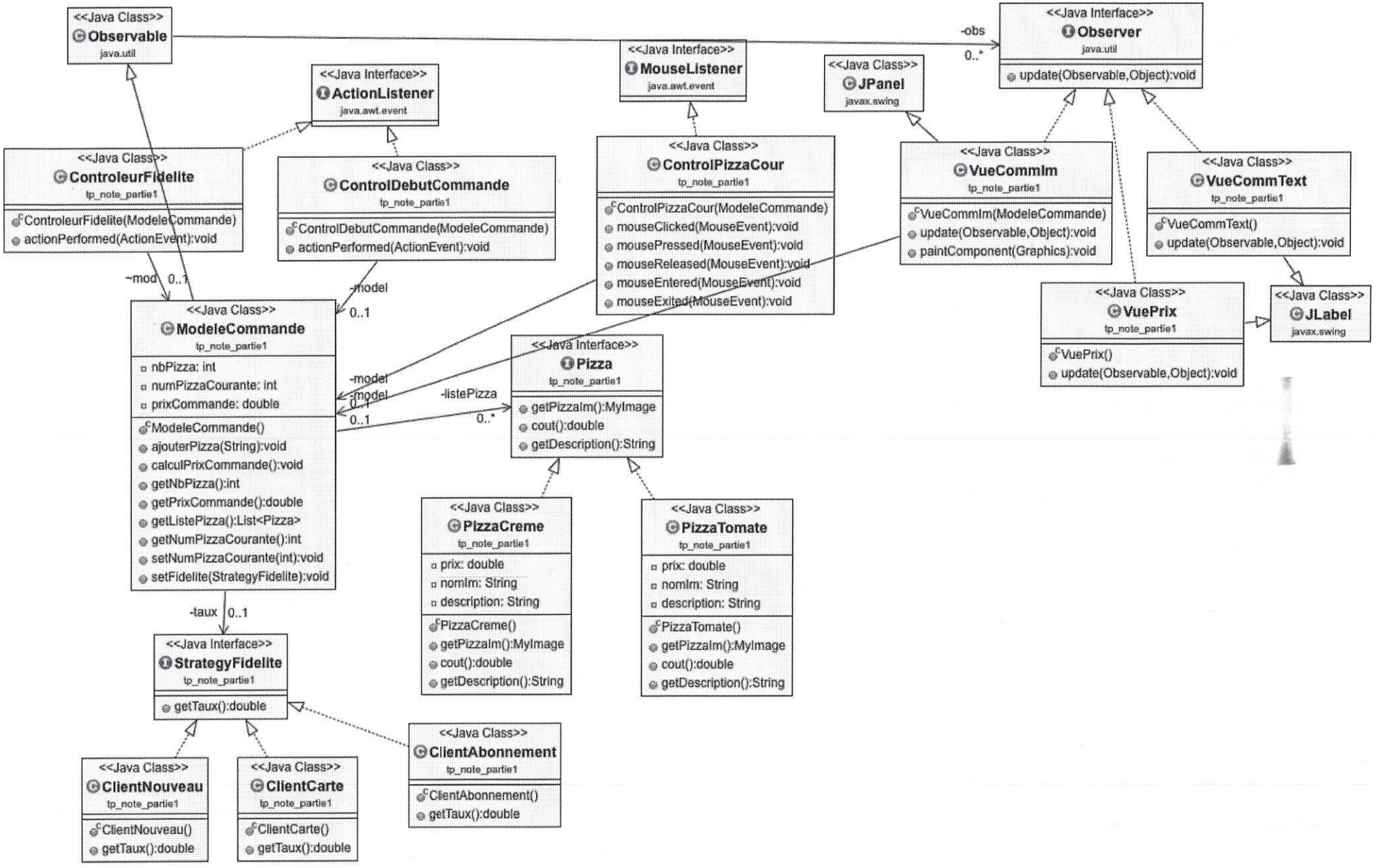
Puis, pour chacune des pizzas, différents ingrédients peuvent être choisis : il faut d'abord sélectionner la pizza à garnir avec un clic de souris sur l'image de la pizza dans la zone 2. Les ingrédients peuvent alors être choisis avec les boutons de la zone 3.



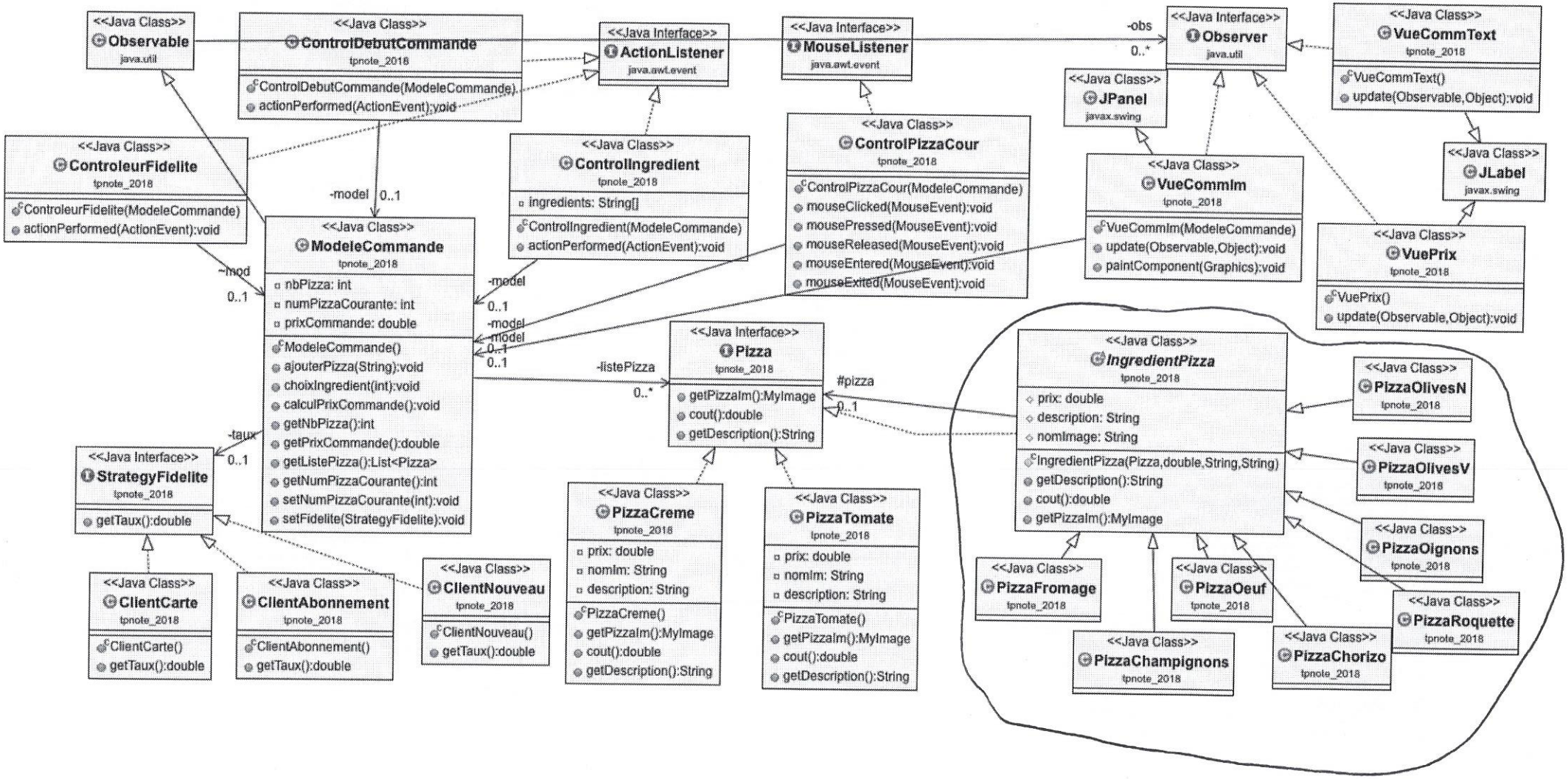
Dans la figure ci-dessous, le client a une carte et bénéficie donc d'une réduction de 10%.



PARTIE 1


<<Java Class>>
Observable
java.util
-obs
0..*
<<Java Interface>>
Observer
java.util
update(Observable,Object):void
<<Java Interface>>
ActionListener
java.awt.event
<<Java Interface>>
MouseListener
java.awt.event
<<Java Class>>
JPanel
javax.swing
<<Java Class>>
ControleurFidelite
tp_note_partie1
ControleurFidelite(ModeleCommande)
actionPerformed(ActionEvent):void
<<Java Class>>
ControlDebutCommande
tp_note_partie1
ControlDebutCommande(ModeleCommande)
actionPerformed(ActionEvent):void
<<Java Class>>
ControlPizzaCour
tp_note_partie1
ControlPizzaCour(ModeleCommande)
mouseClicked(MouseEvent):void
mousePressed(MouseEvent):void
mouseReleased(MouseEvent):void
mouseEntered(MouseEvent):void
mouseExited(MouseEvent):void
<<Java Class>>
VueCommIm
tp_note_partie1
VueCommIm(ModeleCommande)
update(Observable,Object):void
paintComponent(Graphics):void
<<Java Class>>
VueCommText
tp_note_partie1
VueCommText()
update(Observable,Object):void
-mod
0..1
-model
0..1
<<Java Class>>
ModeleCommande
tp_note_partie1
nbPizza: int
numPizzaCourante: int
prixCommande: double
ModeleCommande()
ajouterPizza(String):void
calculPrixCommande():void
getNbPizza():int
getPrixCommande():double
getListePizza():List<Pizza>
getNumPizzaCourante():int
setNumPizzaCourante(int):void
setFidelite(StrategyFidelite):void
-model
0..1
-model
0..1
-listePizza
0..*
<<Java Interface>>
Pizza
tp_note_partie1
getPizzaIm():MyImage
cout():double
getDescription():String
<<Java Class>>
VuePrix
tp_note_partie1
VuePrix()
update(Observable,Object):void
<<Java Class>>
JLabel
javax.swing
-taux
0..1
<<Java Interface>>
StrategyFidelite
tp_note_partie1
getTaux():double
<<Java Class>>
PizzaCreme
tp_note_partie1
prix: double
nomIm: String
description: String
PizzaCreme()
getPizzaIm():MyImage
cout():double
getDescription():String
<<Java Class>>
PizzaTomate
tp_note_partie1
prix: double
nomIm: String
description: String
PizzaTomate()
getPizzaIm():MyImage
cout():double
getDescription():String
<<Java Class>>
ClientNouveau
tp_note_partie1
ClientNouveau()
getTaux():double
<<Java Class>>
ClientCarte
tp_note_partie1
ClientCarte()
getTaux():double
<<Java Class>>
ClientAbonnement
tp_note_partie1
ClientAbonnement()
getTaux():double

PARTIE 2


<<Java Class>>
Observable
java.util
<<Java Class>>
ControlDebutCommande
tpnote_2018
ControlDebutCommande(ModeleCommande)
actionPerformed(ActionEvent):void
<<Java Interface>>
ActionListener
java.awt.event
<<Java Interface>>
MouseListener
java.awt.event
-obs
0..*
<<Java Interface>>
Observer
java.util
<<Java Class>>
VueCommText
tpnote_2018
VueCommText()
update(Observable,Object):void
<<Java Class>>
JPanel
javax.swing
<<Java Class>>
JLabel
javax.swing
<<Java Class>>
ControleurFidelite
tpnote_2018
ControleurFidelite(ModeleCommande)
actionPerformed(ActionEvent):void
-model
0..1
<<Java Class>>
ControlIngredient
tpnote_2018
ingredients: String[]
ControlIngredient(ModeleCommande)
actionPerformed(ActionEvent):void
<<Java Class>>
ControlPizzaCour
tpnote_2018
ControlPizzaCour(ModeleCommande)
mouseClicked(MouseEvent):void
mousePressed(MouseEvent):void
mouseReleased(MouseEvent):void
mouseEntered(MouseEvent):void
mouseExited(MouseEvent):void
<<Java Class>>
VueCommIm
tpnote_2018
VueCommIm(ModeleCommande)
update(Observable,Object):void
paintComponent(Graphics):void
<<Java Class>>
VuePrix
tpnote_2018
VuePrix()
update(Observable,Object):void
-mod
0..1
<<Java Class>>
ModeleCommande
tpnote_2018
nbPizza: int
numPizzaCourante: int
prixCommande: double
ModeleCommande()
ajouterPizza(String):void
choixIngredient(int):void
calculPrixCommande():void
getNbPizza():int
getPrixCommande():double
getListePizza():List<Pizza>
getNumPizzaCourante():int
setNumPizzaCourante(int):void
setFidelite(StrategyFidelite):void
-model
0..1
-listePizza
0..*
<<Java Interface>>
Pizza
tpnote_2018
getPizzaIm():MyImage
cout():double
getDescription():String
#pizza
0..1
<<Java Class>>
IngredientPizza
tpnote_2018
prix: double
description: String
nomImage: String
IngredientPizza(Pizza,double,String,String)
getDescription():String
cout():double
getPizzaIm():MyImage
-taux
0..1
<<Java Interface>>
StrategyFidelite
tpnote_2018
getTaux():double
<<Java Class>>
PizzaCreme
tpnote_2018
prix: double
nomIm: String
description: String
PizzaCreme()
getPizzaIm():MyImage
cout():double
getDescription():String
<<Java Class>>
PizzaTomate
tpnote_2018
prix: double
nomIm: String
description: String
PizzaTomate()
getPizzaIm():MyImage
cout():double
getDescription():String
<<Java Class>>
PizzaOlivesN
tpnote_2018
<<Java Class>>
PizzaOlivesV
tpnote_2018
<<Java Class>>
PizzaOignons
tpnote_2018
<<Java Class>>
PizzaRoquette
tpnote_2018
<<Java Class>>
PizzaFromage
tpnote_2018
<<Java Class>>
PizzaOeuf
tpnote_2018
<<Java Class>>
PizzaChampignons
tpnote_2018
<<Java Class>>
PizzaChorizo
tpnote_2018
<<Java Class>>
ClientCarte
tpnote_2018
ClientCarte()
getTaux():double
<<Java Class>>
ClientAbonnement
tpnote_2018
ClientAbonnement()
getTaux():double
<<Java Class>>
ClientNouveau
tpnote_2018
ClientNouveau()
getTaux():double

1.2. Patron de conception Stratégie

Le patron Stratégie est utilisé dans ce programme pour le calcul du prix de la commande selon la fidélité du client (cf. exemples sur la seconde page de l'énoncé distribué).

Coder l'interface **`StrategyFidelite`** ainsi que les trois classes concrètes associées. La méthode `getTaux` retournera un réel correspondant au nombre par lequel devra être multiplié le coût de la commande pour obtenir la bonne réduction : 1 pour un nouveau client, 0.9 pour un client avec carte (10% de réduction) et 0.7 pour un client avec abonnement (30% de réduction).

Ajouter un attribut `taux` à la classe **`ModeleCommande`** ainsi que la méthode `setFidelite`. La méthode `calculPrixCommande` devra être modifiée afin que le calcul du prix de la commande tienne compte de la fidélité du client. Par défaut, à l'ouverture de l'interface, le client est considéré comme nouveau.

La classe **`ControleurFidelite`** sera ajoutée pour la gestion du choix de fidélité du client dans le composant JComboBox.

1.3. Générer le diagramme des classes de votre programme (**enregistrer obligatoirement l'image associée sinon votre diagramme ne pourra pas être corrigé**), vérifier sa conformité par rapport au diagramme distribué de la partie 1, faites ressortir les éléments caractéristiques du MVC et du patron Stratégie.

1.4. Commenter les classes afin de pouvoir générer la documentation (javadoc) du programme.

Plus précisément et **en priorité**, les commentaires associés à la classe **`ModeleCommande`** et à ses méthodes de modification devront décrire le mécanisme du **patron de conception Observateur** mis en œuvre dans votre programme. Les commentaires associés aux classes contrôleurs devront décrire le mécanisme du **patron d'architecture MVC** mis en œuvre dans votre programme.

Enregistrer l'ensemble des fichiers réalisés pour les questions 2.1 à 2.3 dans un fichier nommé *votreGroupe_VotreNom_PARTIE1.zip*
Le déposer sur arche.

PARTIE 2- PATRON DECORATEUR ET SUITE DE MVC (4 POINTS)

Dans cette partie, on fait évoluer la commande, **le patron décorateur est introduit pour la gestion des ingrédients tout en restant dans une architecture MVC**.

Le diagramme de classes à considérer est celui intitulé Partie 2.

1.5. Coder les classes correspondant au patron décorateur (en respectant le diagramme de classes Partie2) : **`IngredientPizza`** et les différentes classes d'ingrédients (**`PizzaFromage`**, **`PizzaChorizo`**, ...). Les coûts des différents ingrédients sont donnés dans le tableau ci-après :

Fromage	Champignons	Chorizo	Oeuf	Oignons	Olives noires	Olives vertes	Roquette
0.75	0.5	1	0.7	0.4	0.25	0.3	0.45

Remarque : Pour gérer la superposition des images, vous pouvez consulter l'exemple d'utilisation de la classe **`MyImage`** dans la méthode **paintComponent** de la classe **`PanneauImages`**.

1.6. Ajouter une nouvelle classe contrôleur, **`ControlIngredients`**, chargée de gérer les choix des ingrédients et de communiquer ces choix au modèle. Celle-ci utilisera une nouvelle méthode de **`ModeleCommande`** : **choixIngredient(int)**, que vous coderez aussi. L'entier passé en paramètre désigne alors le type d'ingrédient à ajouter à la pizza sélectionnée (en lien avec les boutons d'ajout d'ingrédient, par exemple 0=fromage, 1=champignons, ...).
Quand l'utilisateur clique sur un bouton ingrédient, l'ingrédient est ajouté à la pizza sélectionnée. L'image de l'ingrédient est superposée à celle de la pizza sélectionnée et l'ensemble des affichages textuels est mis à jour (cf. figures de la seconde page de l'énoncé distribué).

Enregistrer l'ensemble des fichiers réalisés pour cette partie dans un fichier nommé *votreGroupe_VotreNom_PARTIE2.zip*

PARTIE 3- AMELIORATIONS (2 POINTS)

1.7. Ajouter les boutons suivants et programmer leurs actions tout **en respectant l'architecture MVC** :

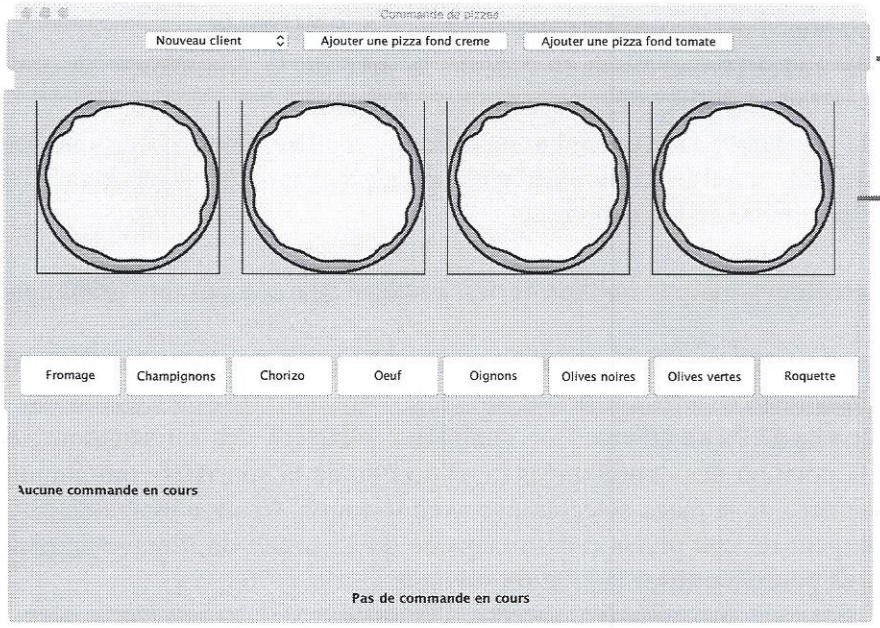
- Un bouton dans la zone 1 pour retirer la dernière pizza ajoutée
- Un bouton dans la zone 3 pour retirer le dernier ingrédient ajouté à la pizza sélectionnée.
- Un bouton pour valider la commande et réinitialiser l'interface dans la zone 5 : la liste des pizzas commandées sera affichée avec un numéro de commande spécifique dans la console et l'interface sera réinitialisée.
- Mettre en œuvre le patron **Factory** pour créer les fonds de pizza, on veut aussi pouvoir ajouter un fond fromage blanc.

Enregistrer l'ensemble des fichiers réalisés pour cette partie dans un fichier nommé *votreGroupe_VotreNom_PARTIE3.zip*

Déposer les archives sur arche.

1. REALISATION DE L'APPLICATION DE COMMANDE DE PIZZAS (15 POINTS)

Le code de la construction de l'interface graphique se trouve dans la classe `Principale` (package tp_note). La classe `PanneauImages` (package tp_note) et le package `MyImage` sont aussi fournis dans le fichier ***tp_note_2018.zip***. Les 10 images (chacune de 200 * 200 pixels) à utiliser sont dans un répertoire ***images*** à déposer à la racine de votre projet Eclipse.
Les `JPanel` sont organisés selon la description indiquée sur l'image ci-dessous, obtenue après exécution du programme java fourni :



→ **JPanel pnord**

→ **JPanel pcentral**
Contient deux composants :
- **PanneauImages visionComm** affiche les images des pizzas
- **JPanel pingr** en GridLayout(1,0), contient les boutons des ingrédients

→ **JPanel psud**
Contient deux JLabel

L'objectif de ce tp est, en tenant compte des **diagrammes distribués** après la phase de conception de 45 minutes, de modifier le code fourni de la classe `Principale` et de la classe `PanneauImages`, d'ajouter des classes, afin de programmer une application de commande de pizzas selon **l'architecture MVC et les patrons observateur, stratégie et décorateur** proposés sur les diagrammes. Vous respecterez le fonctionnement de l'application comme il est décrit sur la première page de l'énoncé. Vous travaillerez dans le package nommé `tp_note`.
Le travail demandé est décomposé en 3 parties et un fichier archive doit être déposé sur arche pour chaque partie.

PARTIE 1-PATRON D'ARCHITECTURE MVC ET PATRON DE CONCEPTION STRATEGIE (9 POINTS)

Important : dans cette première partie les boutons des ingrédients ne seront pas utilisés. Le travail demandé correspond au premier diagramme de classes distribué, intitulé PARTIE1.

1.1. MVC

Commencer par coder **la classe `ModeleCommande`** puis inclure l'instanciation d'un objet de la classe `ModeleCommande` dans la classe `Principale` :

- **Attributs de `ModeleCommande`** :
  - `nbPizza` est le nombre de pizzas de la commande en cours
  - `listPizza` est la liste des pizzas de la commande (ArrayList)
  - `numPizzaCourante` est le numéro de la pizza sélectionnée
  - `prixCommande` est le prix de la commande en cours

- La méthode `ajouterPizza` modifie le modèle en fonction de son paramètre String qui indique la nature de la pizza à ajouter (base crème ou tomate). Les attributs de ModeleCommande doivent évoluer en fonction de la pizza ajoutée. Attention, au plus 4 pizzas peuvent être ajoutées à la commande.
- La méthode `calculPrixCommande` calcule le prix de la commande en cours (pizza base crème : 6 euros, pizza base tomate : 5 euros) .
- La méthode `setNumPizzaCourante` modifie le modèle, plus spécifiquement modifie l'attribut numPizzaCourante, correspondant à la pizza sélectionnée, en fonction du paramètre de la méthode.

- Coder les **contrôleurs**, ne pas oublier de les associer aux composants graphiques concernés :
  - la classe `ControlDebutCommande` : ce contrôleur gère le choix des pizzas de base par l'utilisateur.
  - la classe `ControlPizzaCour` : ce contrôleur récupère des informations, en particulier la position des coordonnées (x,y) du clic de la souris et communique ensuite le numéro de la pizza sélectionnée avec la souris. Chaque image de pizza ayant une largeur de 200 pixels et étant séparée de 25 pixels de l'image de pizza suivante, vous pourrez utiliser la relation suivante :

    ```
    int numpizzaSelec = x/225;
    ```

    L'image de la pizza sélectionnée devra être entourée d'un rectangle noir (et pas les autres pizzas).

- Coder les 3 classes **vues** et modifier en conséquence la classe `Principale` ainsi que la classe `PanneauImages` qui doit être transformée en une vue (et donc renommée).
  Remarques : Afin d'afficher les listes à puces attendues, vous devrez écrire du code html dans un JLabel. La méthode `setText` peut être utilisée avec du code html en paramètre quand le JLabel est bien construit (cf. l'appel des constructeurs de JLabel dans la classe Principale).
  De plus, la méthode *`format`* de la classe String permet de formater l'affichage du contenu d'une variable de type double.
  Exemple : `String.format("%.2f", prix)` génère une chaîne de caractères correspondant au double contenu dans la variable prix avec deux chiffres derrière la virgule.

A l'issue de cette question, l'appui sur les boutons d'ajout de pizzas fond crème ou tomate permet l'affichage des images des fonds sur la partie centrale de l'interface ainsi que le descriptif et les prix de la commande comme indiqué sur la figure de la première page de l'énoncé distribué.

NOM : Costantini Prénom : Ugo Groupe : S3 A

1. **MODELE VUE CONTROLEUR - CONCEPTION – Les explications demandées dans les questions 1.1 et 1.2 ci-dessous sont à rendre 45 minutes après la distribution du sujet.**

1.1. Quels **patrons de conception** peuvent être mis en œuvre dans cette application ? Expliquer le rôle de chacun dans le cadre de ce programme.

1.2. Dans le cadre de l'implantation du **modèle d'architecture MVC** pour l'interface graphique présentée à la page précédente, identifier et décrire précisément :
- le **modèle** (donner les attributs, les méthodes envisagées en précisant leurs rôles),
- les **vues** (indiquez la nature de leur composant graphique, le(s) méthode(s) envisagée(s) en précisant leur(s) rôle(s)),
- les **contrôleurs** (indiquer les interactions avec certains composants de l'interface graphique, les relations avec le modèle, les méthodes envisagées, ...).

1.3. Faire SUR PAPIER **le diagramme des classes java** que vous envisagez pour la mise en œuvre du modèle MVC ainsi que les patrons de conceptions identifiés à la question 1.1 en précisant les liens d'**héritage**, d'**implémentation**, et d'**association**.

----Rendez un document papier avec vos réponses avant de passer à la programmation----