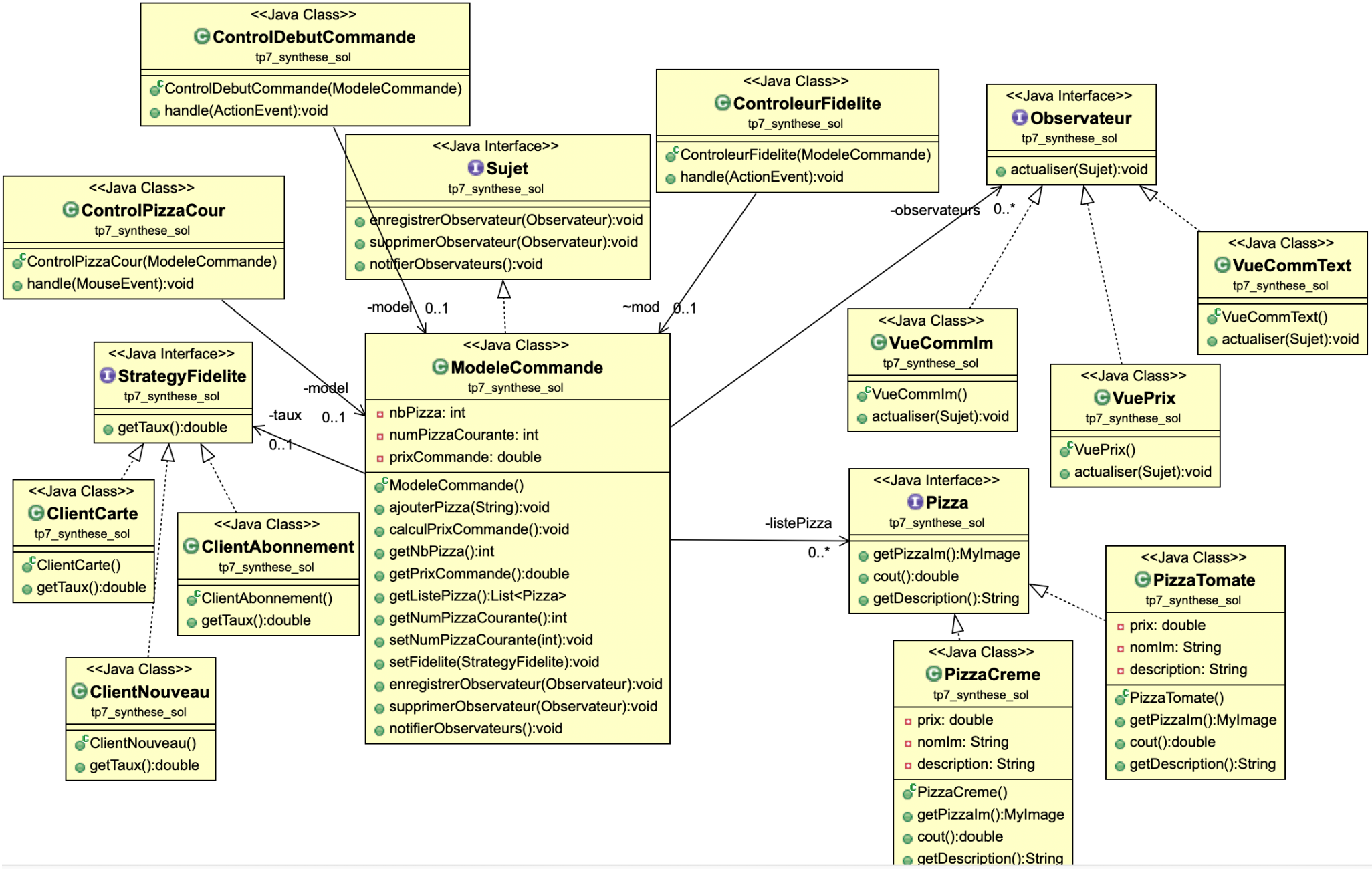
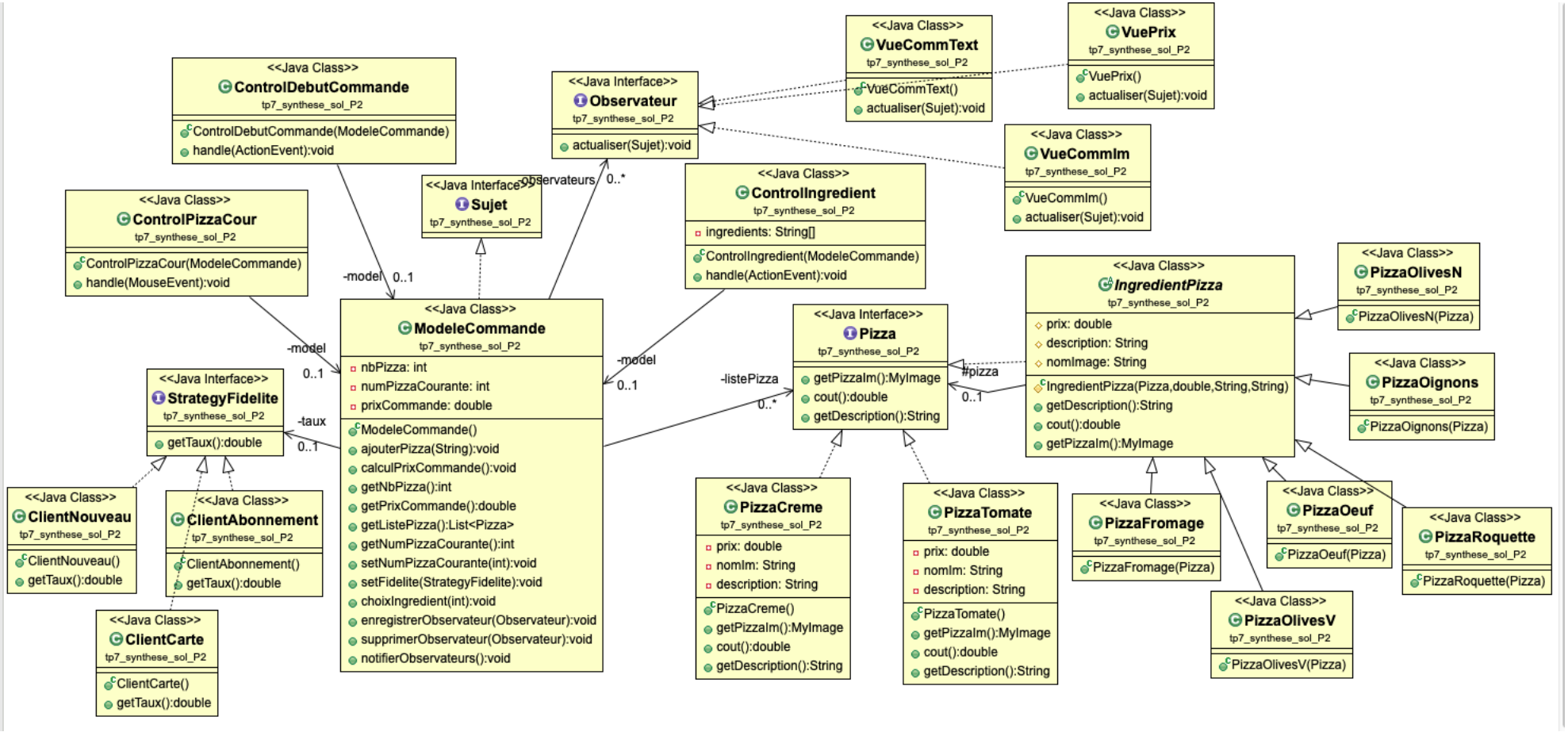

<<Java Class>>
ControlDebutCommande
tp7_synthese_sol
ControlDebutCommande(ModeleCommande)
handle(ActionEvent):void
<<Java Class>>
ControleurFidelite
tp7_synthese_sol
ControleurFidelite(ModeleCommande)
handle(ActionEvent):void
<<Java Interface>>
Observateur
tp7_synthese_sol
actualiser(Sujet):void
<<Java Interface>>
Sujet
tp7_synthese_sol
enregistrerObservateur(Observateur):void
supprimerObservateur(Observateur):void
notifierObservateurs():void
<<Java Class>>
ControlPizzaCour
tp7_synthese_sol
ControlPizzaCour(ModeleCommande)
handle(MouseEvent):void
-observateurs
0..*
<<Java Class>>
VueCommText
tp7_synthese_sol
VueCommText()
actualiser(Sujet):void
-model
0..1
~mod
0..1
<<Java Class>>
VueCommIm
tp7_synthese_sol
VueCommIm()
actualiser(Sujet):void
<<Java Interface>>
StrategyFidelite
tp7_synthese_sol
getTaux():double
-model
-taux
0..1
0..1
<<Java Class>>
ModeleCommande
tp7_synthese_sol
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
enregistrerObservateur(Observateur):void
supprimerObservateur(Observateur):void
notifierObservateurs():void
<<Java Class>>
VuePrix
tp7_synthese_sol
VuePrix()
actualiser(Sujet):void
<<Java Class>>
ClientCarte
tp7_synthese_sol
ClientCarte()
getTaux():double
<<Java Class>>
ClientAbonnement
tp7_synthese_sol
ClientAbonnement()
getTaux():double
-listePizza
0..*
<<Java Interface>>
Pizza
tp7_synthese_sol
getPizzaIm():MyImage
cout():double
getDescription():String
<<Java Class>>
PizzaTomate
tp7_synthese_sol
prix: double
nomIm: String
description: String
PizzaTomate()
getPizzaIm():MyImage
cout():double
getDescription():String
<<Java Class>>
ClientNouveau
tp7_synthese_sol
ClientNouveau()
getTaux():double
<<Java Class>>
PizzaCreme
tp7_synthese_sol
prix: double
nomIm: String
description: String
PizzaCreme()
getPizzaIm():MyImage
cout():double
getDescription():String

PARTIE 1


<<Java Class>>
ControlDebutCommande
tp7_synthese_sol_P2
ControlDebutCommande(ModeleCommande)
handle(ActionEvent):void
<<Java Interface>>
Observateur
tp7_synthese_sol_P2
actualiser(Sujet):void
<<Java Class>>
VueCommText
tp7_synthese_sol_P2
VueCommText()
actualiser(Sujet):void
<<Java Class>>
VuePrix
tp7_synthese_sol_P2
VuePrix()
actualiser(Sujet):void
<<Java Class>>
VueCommIm
tp7_synthese_sol_P2
VueCommIm()
actualiser(Sujet):void
<<Java Class>>
ControlPizzaCour
tp7_synthese_sol_P2
ControlPizzaCour(ModeleCommande)
handle(MouseEvent):void
<<Java Interface>>
Sujet
tp7_synthese_sol_P2
observateurs
0..*
<<Java Class>>
ControlIngredient
tp7_synthese_sol_P2
ingredients: String[]
ControlIngredient(ModeleCommande)
handle(ActionEvent):void
-model
0..1
<<Java Class>>
ModeleCommande
tp7_synthese_sol_P2
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
choixIngredient(int):void
enregistrerObservateur(Observateur):void
supprimerObservateur(Observateur):void
notifierObservateurs():void
-model
0..1
-listePizza
0..*
<<Java Interface>>
Pizza
tp7_synthese_sol_P2
getPizzaIm():MyImage
cout():double
getDescription():String
#pizza
0..1
<<Java Class>>
IngredientPizza
tp7_synthese_sol_P2
prix: double
description: String
nomImage: String
IngredientPizza(Pizza,double,String,String)
getDescription():String
cout():double
getPizzaIm():MyImage
<<Java Class>>
PizzaOlivesN
tp7_synthese_sol_P2
PizzaOlivesN(Pizza)
<<Java Class>>
PizzaOignons
tp7_synthese_sol_P2
PizzaOignons(Pizza)
<<Java Interface>>
StrategyFidelite
tp7_synthese_sol_P2
getTaux():double
-taux
0..1
<<Java Class>>
ClientNouveau
tp7_synthese_sol_P2
ClientNouveau()
getTaux():double
<<Java Class>>
ClientAbonnement
tp7_synthese_sol_P2
ClientAbonnement()
getTaux():double
<<Java Class>>
ClientCarte
tp7_synthese_sol_P2
ClientCarte()
getTaux():double
<<Java Class>>
PizzaCreme
tp7_synthese_sol_P2
prix: double
nomIm: String
description: String
PizzaCreme()
getPizzaIm():MyImage
cout():double
getDescription():String
<<Java Class>>
PizzaTomate
tp7_synthese_sol_P2
prix: double
nomIm: String
description: String
PizzaTomate()
getPizzaIm():MyImage
cout():double
getDescription():String
<<Java Class>>
PizzaFromage
tp7_synthese_sol_P2
PizzaFromage(Pizza)
<<Java Class>>
PizzaOeuf
tp7_synthese_sol_P2
PizzaOeuf(Pizza)
<<Java Class>>
PizzaRoquette
tp7_synthese_sol_P2
PizzaRoquette(Pizza)
<<Java Class>>
PizzaOlivesV
tp7_synthese_sol_P2
PizzaOlivesV(Pizza)

PARTIE 2

1. REALISATION DE L'APPLICATION DE COMMANDE DE PIZZAS

Le code de la construction de l'interface graphique se trouve dans la classe `Principale` (package tp7_synthese). La classe `PanneauImages` et la classe `MyImage` (package tp7_synthese) sont aussi fournies dans le fichier ***tp7_synthese.zip***. Les 10 images (chacune de 200 * 200 pixels) à utiliser sont dans le fichier ***images.zip*** à placer décompressé dans le répertoire ressources de votre projet IntelliJ.
Les `panneaux` sont organisés selon la description indiquée sur l'image ci-dessous, obtenue après exécution du programme java fourni :

Panneau pnord

Panneau pcentral
Contient deux composants :
- **PanneauImages visionComm** affiche les images des pizzas
- **panneau pingr, un GridPane,** contient les boutons des ingrédients

panneau psud
Contient deux Label

L'objectif de ce tp est, en tenant compte des **diagrammes distribués** après la phase de conception, 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 **`tp7_synthese`**.
Le travail demandé est décomposé en 3 parties et **un fichier archive doit être déposé sur arche pour chaque partie.**

PARTIE 1 PROGRAMMATION - PATRON D'ARCHITECTURE MVC ET PATRON DE CONCEPTION STRATEGIE

Important : dans cette première partie les boutons des ingrédients ne seront pas utilisés.
Le travail demandé correspond au premier diagramme de classes, 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 qui hérite de GridPane (et donc renommée ici en VueCommIm).
  La vue VueCommText, hérite de Label, et correspond à la zone de description de la commande.
  La vue VuePrix, hérite de Label, et correspond à la zone où figure le prix de la commande.

  Remarque : 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.

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 ComboBox.

1.3. Générer le diagramme des classes de votre programme à l'aide de votre IDE et plantUML, sous IntelliJ (**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

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`** 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

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.