Tp2 : Design patterns de création

Exercice1 :

1\* on vient d’utiliser un design pattern qui permet la gestion de familles de véhicules(scooter et camion )-🡪 c’est **AbstractFactory**

La classe abstraite Scooter a deux sousclasses concrètes ScooterÉlectricité et ScooterEssence.

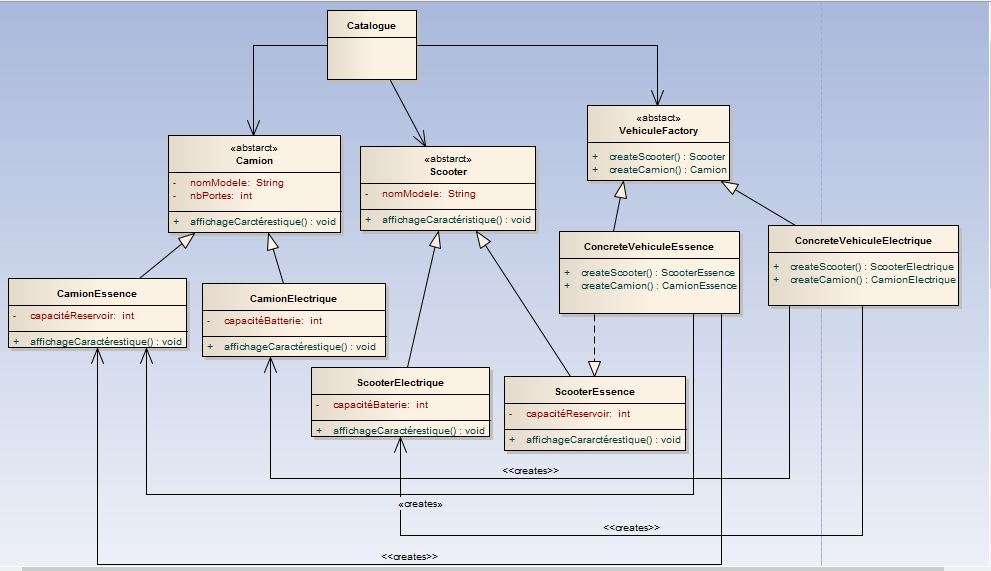
La classe abstraite Camion a deux sousclasses concrètes CamionÉlectricité et CamionEssence.

AbstarctProduct : Scooter et Camion

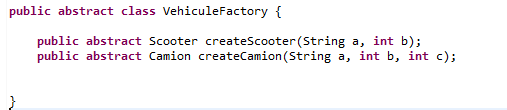
ConcreteProduct : ScooterÉlectricité, ScooterEssence, CamionÉlectricité, CamionEssence.

AbstractFactory :déclare l’interface pour les opérations qui créent des objets abstraits(scooter et Camion)🡪VehiculeFactory

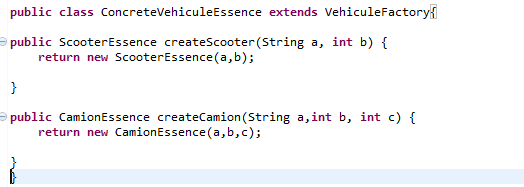
ConcreteFactory=implémente les opérations qui créent les objets concrets(ConcreteVehiculeEssence et ConcreteVehiculeElectrique)

Client :utilise les interfaces déclarées par VehiculeFactory , Scooter et Camion

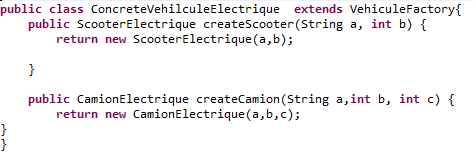
VehiculeFactory



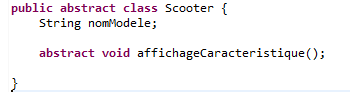
ConcreteVehiculeEssence



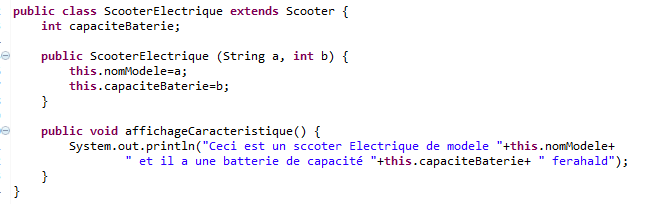
ConcreteVehiculeElectrique



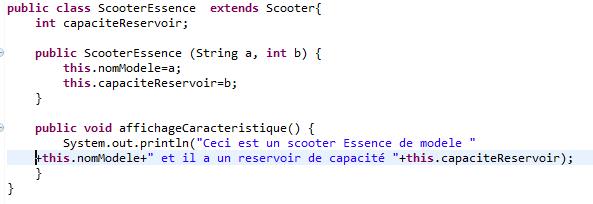
Scooter



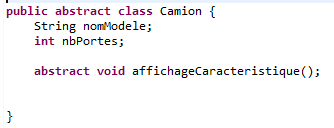
ScooterElectrique



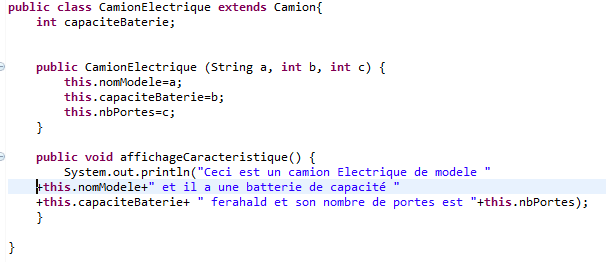
ScooterEssence



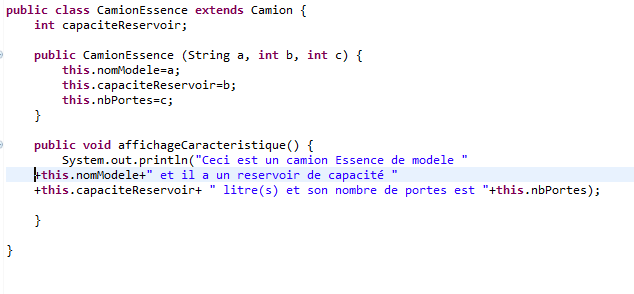
Camion



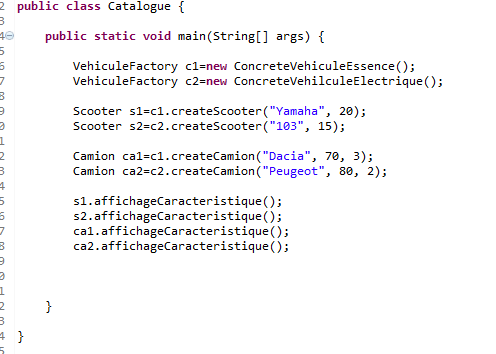
CamionElectrique

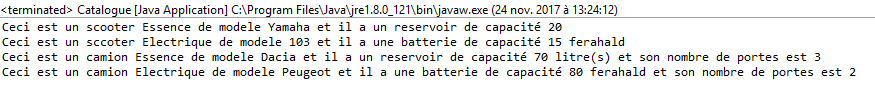


CamionEssence



Catalogue :





Exercice2 :

Un repas complexe est construit à partir d’un hamburger et une boisson froide.Le Burger pourrait être soit un hamburger végi ou un hamburger de poulet. La boisson froide peut être une coke ou un pepsi.

On suppose :

Repas1=Hambuerger Poulet+coke

Repas2=Hambuerger Poulet+Pepsi

Repas3=Hambuerger végi+coke

Repas4=Hambuerger végi+pepsi

On suppose que pour le repas1 , la coke est de type <0% sucre>

Et pour le repas2,la coke est de type <medium>

\*type est un attribut qui se trouve seulement pour coke.

On cherche un design pattern tels que même processus de construction permette de créer différentes représentations de l’objet Repas(repas1 ,repas2,repas3,repas4)

🡪design pattern utilisé : **Builder**

IBuilder: classe abstraite qui crée les différentes

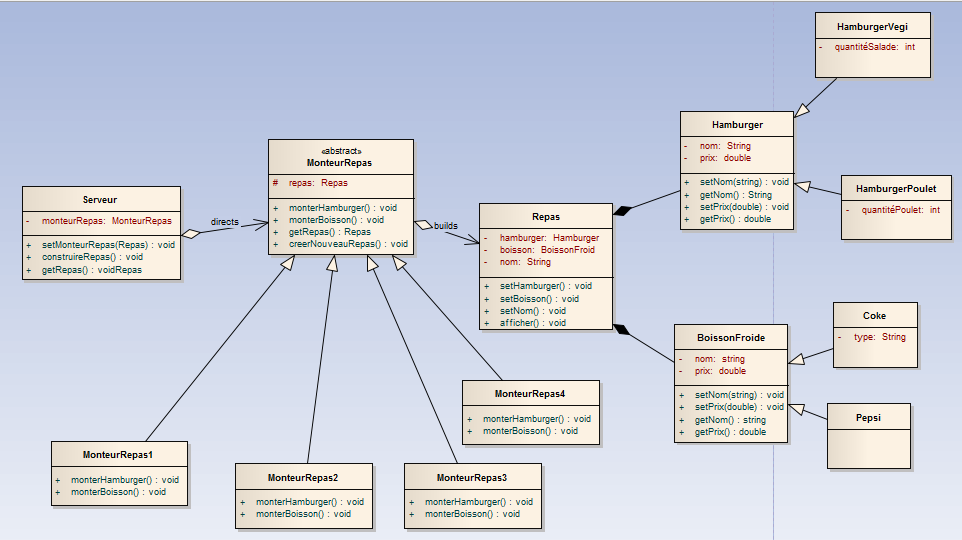
parties de l’objet complexe :MonteurRepas

ConcreteBuilder: Classe qui implémente MonteurRepas :

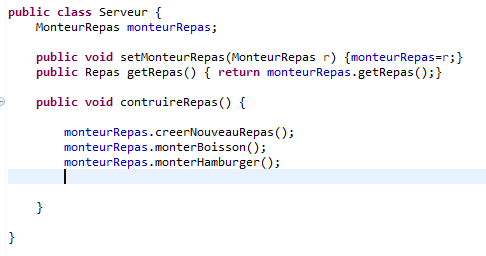
MonteurRepas1, MonteurRepas2, MonteurRepas3, MonteurRepas4

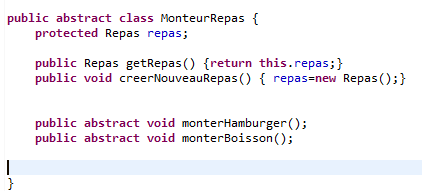
Product: Objet complexe à monter :Repas

Director: Classe qui initie le processus de montage du repas en utilisant l’interface MonteurRepas :Serveur

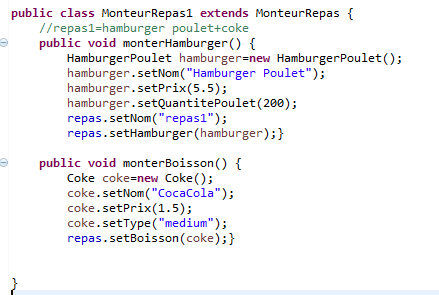


Classe Serveur

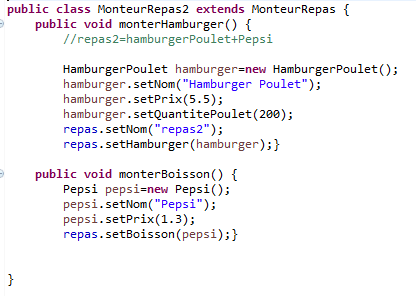




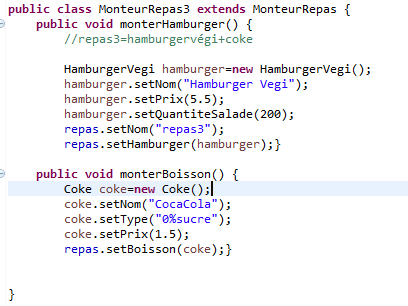
Class MonteurRepas1



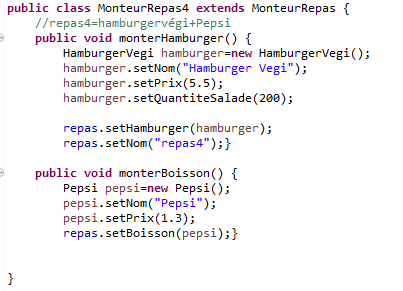
Class MonteurRepas2



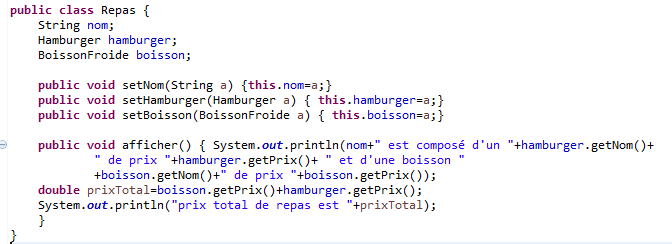
Class MonteurRepas3



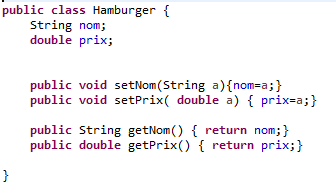
Class MonteurRepas4



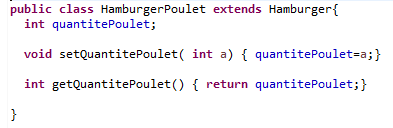
Class Repas



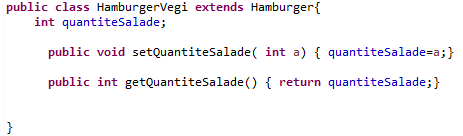
Class Hamburger



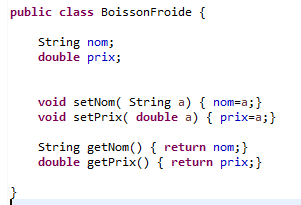
Class HamburgerPoulet

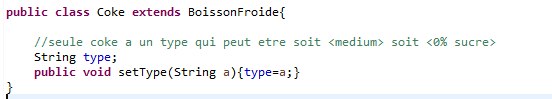


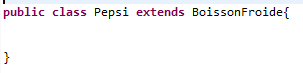
Class Hamburgervégi



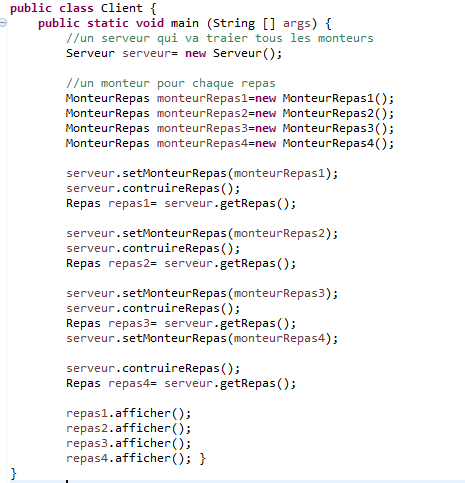
Class BoissonFroide

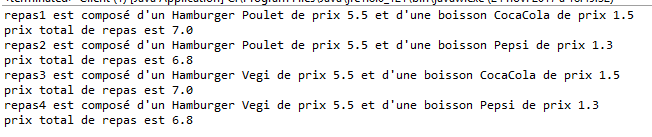






Class client





Exercice3 :

on veut ouvrir au maximum une connexion à la base de données-> avoir une unique instance

🡪utilisation de pattern :Singleton

On a crée un base de donnée avec Neatbeans ; DataBase derby(dans le cadre d’une application Web) de nom « MaBase4 »

