

Rapport

▼ Consignes

▼ Notes

Lors qu'on exécute → page d'accueil → infos

↳ On appuie sur "Shop now" et ça confirme

On va dans l'onglet Shop en haut

On a les produits que notre magasin vend

Expliquer le rôle de chaque bouton

Partie commande → "ticket de caisse"

À la fin de l'achat → ticket de caisse dans un fichier texte dans la racine du projet

Le bouton "Shop now" termine le processus d'achat

Notre projet

Dans le cadre de ce devoir de fin de semestre, nous avons réalisé un système complet de gestion de magasin. Ce programme, codé en Java, permet de :

- Créer de nouveaux produits, les ajouter au catalogue
- Parcourir le catalogue de produits
- Attribuer un caissier pour une commande
- Gérer le stock
 - Ajouter, supprimer des produits
 - Modifier les quantités
- Interagir avec l'utilisateur grâce à une interface graphique (JavaFX)

- Récupérer les informations produit depuis un fichier texte
- Proposer au client un ticket de texte dans un fichier texte

Ce programme représente une boutique en informatique généraliste, proposant à la fois des livre, du matériel et des vêtements en rapport avec son domaine.

Après chaque commande passé le caissier(employé) de notre magasin obtient des gains en fonction de son contrat de travail (CDI, Stage, CDD,)

Shop S5

Accueil Shop Personnel

Dauphine | PSL

Shop Paris DAUPHINE PSL

-Bienvenue dans notre magasin créé lors s'un projet Java du S5.
 -L'onglet "Shop" est dédié pour nos clients afin qu'ils puissent effectuer leurs achats dans notre magasin DAUPHINE PSL.
 -L'onglet "Manager" est dédié pour notre personnel afin qu'il puisse gérer notre stock de magasin.

Nom

Prenom

Date de naissance

Adresse

Code postal

Mode de paiement ☐ CB ☐ Espece ☐ Chèque

Bonjour ! Saisissez vos informations personnelles avant de commencer et appuyez sur le bouton 'Shop now'

Shop now


By AIT LAHCEN Simane, ETIENNE Loïc, BOURDON Maxime

Shop S5








Accueil

Shop

Personnel



Shop now

	PS5 : 455,45 euros	0	-	+	D
	Sweat : 40,00 euros	0	-	+	D
	Switch : 339,0 euros	0	-	+	D
	Livre Java pour les null : 35 euros	0	-	+	D
	Livre "Apprendre JavaFX" : 35 euros	0	-	+	D
	Livre SQL : 35 euros	0	-	+	D
	Livre : Algorithme dans les graphes : 35 euros	0	-	+	D

Vous pouvez commencer par voir les produits de notre magasin et pour plus de détails appuyez sur l'icone D pour telecharger un fichier dans la racine du dossier Shop.

Apres avoir lu la description de chaque article, vous pouvez desormais ajouter des articles dans votre panier ou supprimer quelques uns.

Infos personnelles

Votre panier (infos commande) :

Ticket

Nom :

NON RENSEIGNE

Num de commande :

0

Prenom :

NON RENSEIGNE

Caissier :

Loic, ETIENNE

Date de naissance :

NON RENSEIGNE

Magasin :

Daphine Shop

Adresse :

NON RENSEIGNE

Date de commande :

Code postal :

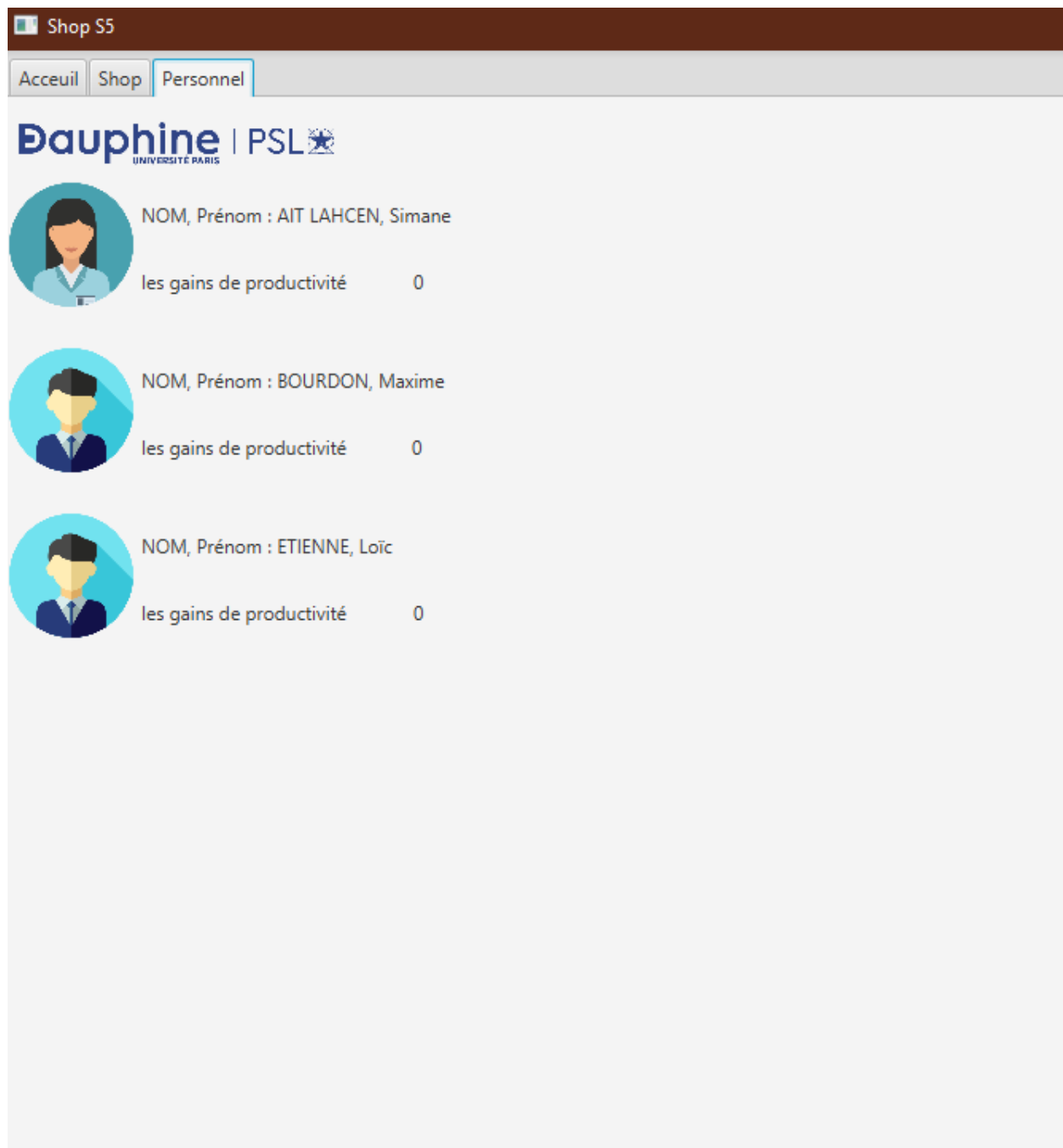
NON RENSEIGNE

Mode de paiement :

espece

Prix Total :

0.0

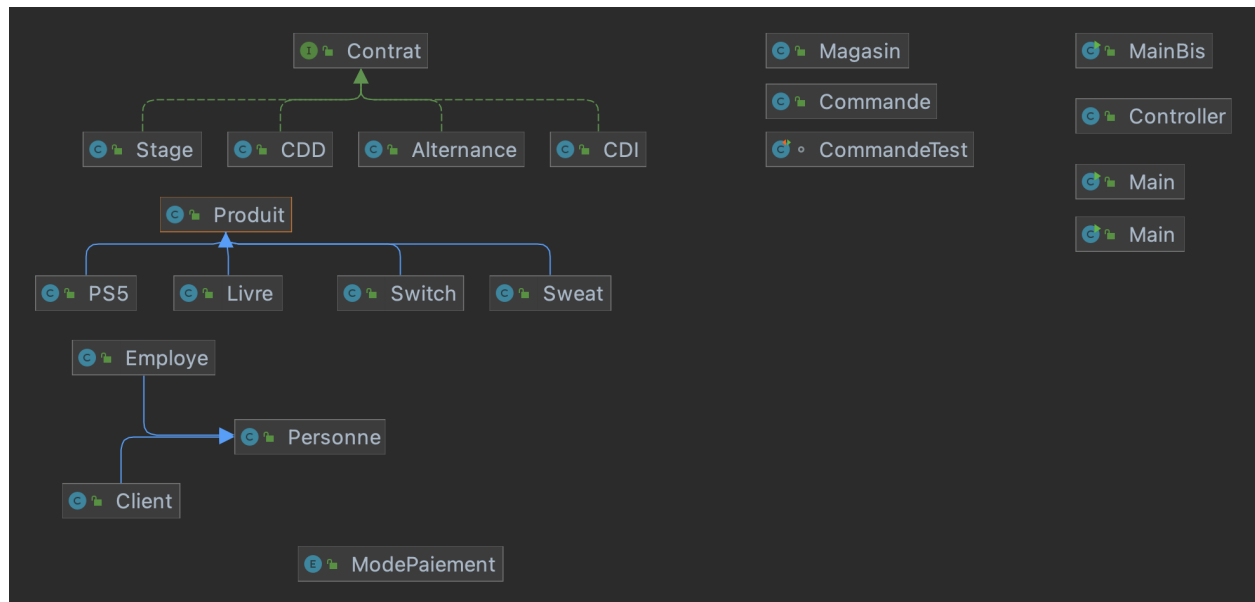


Lien GIT :

[SimaneAL/Shop\(github.com\)](https://github.com/SimaneAL/Shop)

Le code :

1. Le diagramme de classe :



2. Présentation du code :

Lors du début de l'écriture du code au cours de ce projet, nous avons utilisé Maven afin de créer une arborescence pour gérer au mieux les classes composant notre magasin.

Ayant par la suite implémenté une interface graphique via la librairie JavaFX, nous nous sommes renseignés en amont sur les structures de code optimales, ou du moins récurrentes, pour ce type de projet. Celles-ci convergeant globalement avec la structure Maven, première structure du projet, nous avons fait le choix de conserver cette arborescence.

Ainsi, excluant les dossiers générés par notre IDE IntelliJ, la structure du projet est la suivante :

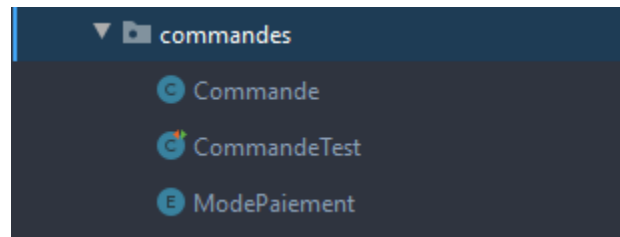
- Shop
 - bin

- out

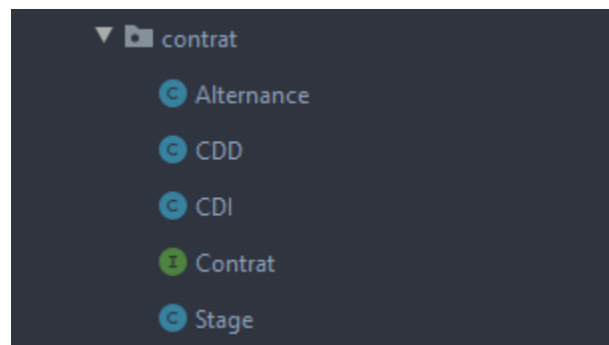
Dossiers de fichiers .class générés par IntelliJ lors de l'exécution

- src

- README.md : un fichier qui va vous montrer comment posséder à l'exécution de notre programme : IntelliJ, JavaFX...
- Commandes



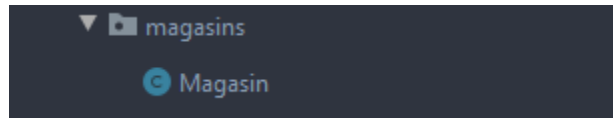
- Commande : classe permettant de gérer les commandes, ajouter, supprimer des produits, calculer le prix total de la commande courante.
 - une classe CommandeTest qui fait des tests JUnit par rapport à la classe Commande et ses méthodes.
 - ModePaiement : Enum recensant les différents modes de paiement disponibles (CB, Espece, Chèque,...)
- Contrat



- Une interface Contrat, ainsi que les différents types de contrats disponibles (stage, CDD...), avec des caractéristiques spécifiques

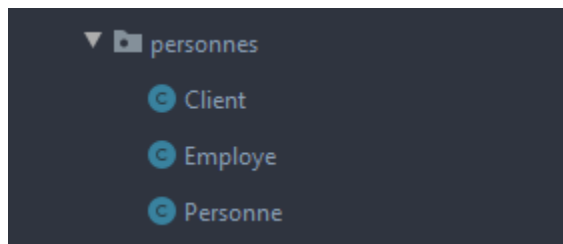
- Ces classes implémentent l'interface Contrat, et possèdent toutes un attribut de type entier `pourcentageGainsDeProductivite` qui représente le pourcentage des gains d'un employé en fonction du type de son contrat.
 - CDI : le pourcentage est de 15, le caissier gagne donc 15% du prix total associé à une commande d'un client quelconque.
 - Stage : le pourcentage est de 5.

■ magasins



- Une classe Magasin unique (comprenant l'ensemble des méthodes principales nécessaires au bon fonctionnement du magasin, comme l'attribution d'un caissier ou l'enregistrement des informations, ainsi que la gestion du stock afin de supprimer ou ajouter des produits)
 - Cette classe implémente des méthodes qui gèrent l'interaction avec le client : création des fichiers qui affichent les détails d'un produit ou alors imprimer le ticket de caisse.

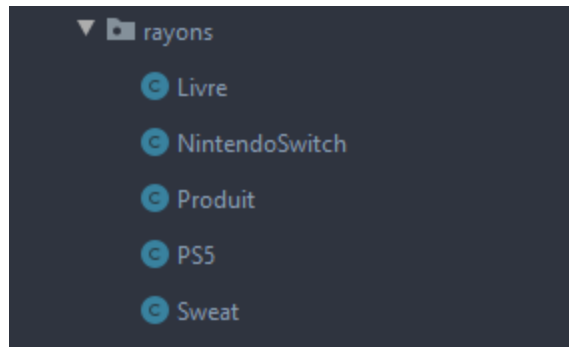
■ personnes



- Un package recensant une classe Personne (comprenant les attributs de chaque individu (nom, prenom, date de naissance...) et des getters & setters), et deux sous classes **Employe** & **Client**, avec des attributs et méthodes qui leur sont spécifiques (ces deux classes héritent de Personne) et chacune fait un boulot différent, Employe par exemple

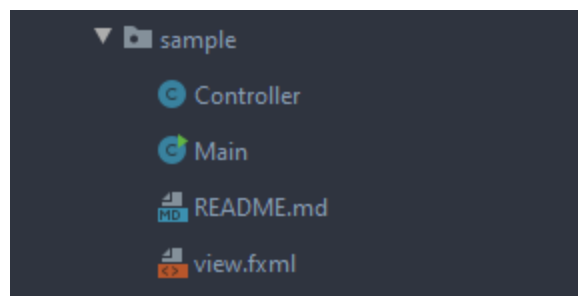
gère ses gains pour chaque commande passé par un client auquel il est associé.

- picsUn package d'images utiles à l'interface graphique
- rayons



- Une classe Produit avec attributs, getters et setters, ainsi qu'une sous classe par type de produit (par exemple, Livre.java), héritant de Produit et présentant le prix du produit et sa description.

- sample



- un fichier view.fxml dans lequel on a pu coder les éléments de notre interface graphique en utilisant le JavaFX.
- Un dossier comprenant une classe Main (démarrage avec JavaFX).
- Une classe Controller est aussi présente permettant de modéliser l'interface graphique. ainsi que la gestion des erreur :
 - exp : si un utilisateur possède au paiement avant de renseigné ses infos personnelles ou avant de choisir un article au moins, un message s'affiche afin de lui demander de les renseigner.

(sujet à évolution → @Simane)

Les contraintes

Les contraintes que nous avons à suivre pour la réalisation de ce projet étaient les suivantes :

- Il est nécessaire de réaliser au moins une classe de Test
 - ... @Maxime classe Commande
- Une documentation correcte devra être fournie
 - Une documentation javadoc est disponible à la racine du projet, présentant les classes que nous avons construit, leurs attributs et leurs méthodes, les paramètres ainsi que des description pour chacun de ces éléments.
- Il est nécessaire d'utiliser les collections
 - Notre projet utilise les collections Map, ArrayList, *enum* pour gérer les modes de paiement, les articles, le ticket de caisse et le stock.
- Deux modes de navigation (via le terminal et en utilisant les fichiers) devront être disponibles
 - Il est possible d'interagir avec notre magasin via une interface en JavaFX (classe `sample/MainBis`) implique l'utilisation de différents fichiers, comme l'impression du ticket de caisse ou la récupération des informations sur chaque article).

La difficulté

La première difficulté que nous avons rencontré fut la recherche d'un fil directeur. Nous avons créé plusieurs projets avant de se tourner vers une version définitive.

La conception fut également une étape complexe, des désaccords dans l'équipe ayant survécu à de multiples reprises lors de cette phase. C'est durant cette phase que nous discutons du nombre de classes à utiliser, des liaisons (héritage, implémentation...) entre-elles, des méthodes qu'elles devaient proposer...

Ensuite, une fois notre décision arrêtée avec un diagramme de classe complet et les premières lignes de code écrites, nous avons fait face à de multiples erreurs dont nous n'avions jamais eu connaissance auparavant.

Ces erreurs concernaient souvent les nouveaux types manipulés (Map...), la conversion des types (String en int en utilisant "Integer.parseInt" ou alors int en String en utilisant "String.valueOf")

Ainsi que des erreurs concernant des pointeurs nuls, cela était au niveau de notre interface graphique lorsqu'un utilisateur entame le shopping sans avoir démarré la partie (renseigner ses infos personnelles, ou choisir un article au moins avant de passer la commande)

Celles-ci étaient également parfois liées à la version de Java, ou à l'architecture de la machine sur laquelle nous travaillions (Intel x64 ou Apple arm64).

Ce dernier point s'est surtout vérifié lors de l'implémentation de JavaFX en complément de la navigation via le terminal, où il fut impossible d'exécuter le code sous un ordinateur Apple sous une architecture arm si la version de JavaFX (SDK) installée n'était pas prévue pour une architecture x64.

Enfin, l'utilisation de Git, avec notamment des commits mal synchronisés et une gestion des branches au début complexe, a pu être vecteur de confusions.