Projet JavaFX : WomenShop

Rapport de réalisation

TOMCZYK Dayan BERREHILI Saber

ESILV – M1 Cloud Computing
Cybersecurity

26/11/2025

1 Introduction

Ce projet a été réalisé dans le cadre du module de programmation orientée objet en Java. L'objectif était de reprendre l'exercice *Women's Store* et de le transformer en une application de bureau complète, en utilisant JavaFX pour l'interface graphique et MySQL pour la persistance des données.

L'application **WomenShop** est une application de gestion de stock. Elle permet au responsable de magasin de consulter les produits disponibles, de gérer les stocks, d'appliquer des remises et de suivre des indicateurs financiers de base (capital, revenus, coûts).

Le sujet impose notamment :

— l'utilisation de JavaFX et du pattern MVC pour l'interface,
— l'utilisation d'une base de données MySQL avec un pattern DAO,
— la gestion de plusieurs catégories de produits (vêtements, chaussures, accessoires),
— la mise en place d'opérations d'achat et de vente qui impactent le stock et les indicateurs financiers,
— l'application de remises fixes selon la catégorie de produit.

2 Architecture générale de l'application

Nous avons organisé le code en plusieurs packages afin de respecter une architecture de type MVC (Model–View–Controller) couplée à un DAO pour l'accès aux données.

Package `model`

Ce package contient toutes les classes du *modèle métier* :

— **Product** : classe abstraite représentant un produit générique du magasin. Elle contient les attributs communs à tous les produits (identifiant, nom, prix d'achat, prix de vente, prix remisé, quantité en stock) ainsi que des attributs statiques pour le capital, le coût total et le revenu total.
— **Clothes**, **Shoes**, **Accessories** : trois sous-classes de `Product` représentant respectivement les vêtements, les chaussures et les accessoires. Elles ajoutent le cas échéant un attribut spécifique (taille de vêtement, pointure, etc.) et imposent des règles métier (contraintes sur les tailles).
— **Discount** : interface définissant les opérations liées aux remises. Elle fixe des constantes pour les pourcentages de réduction (30 pour les vêtements, 20 pour les chaussures et 50 pour les accessoires) et les deux méthodes `applyDiscount()` et `unApplyDiscount()` implémentées dans les sous-classes.

— **DBManager** : classe utilitaire d'accès aux données (DAO) qui se charge de la connexion à MySQL, du chargement des produits et de la mise à jour de la base (stock, prix remisés, suppression d'un produit).

Package `controller`

Ce package regroupe les contrôleurs JavaFX :
— **StoreControl** : contrôleur principal associé à la vue graphique. Il gère le `TableView` des produits, les filtres, les tris, les boutons d'achat et de vente, l'application des remises ainsi que l'affichage des indicateurs financiers.

Package `app`

Ce package contient les classes de lancement :
— **Main** : classe qui étend `Application`. Au démarrage, elle charge les produits depuis la base de données via `DBManager`, puis instancie la scène JavaFX à partir du fichier FXML.
— **Launcher** : classe avec une méthode `main` simple, utilisée pour lancer l'application plus facilement selon la configuration JavaFX (Gluon).

Package `view`

Ce package contient les fichiers FXML :
— **Mainview.fxml** : définition de la fenêtre principale. La vue est organisée avec un `BorderPane` :
  — en haut, une barre d'indicateurs (capital, revenus, coûts, profit),
  — au centre, un `TableView` affichant la liste des produits et leurs propriétés,
  — à gauche, un ensemble de boutons pour filtrer et trier les produits, ainsi que pour les opérations d'édition et de suppression,
  — en bas, la zone de contrôle des quantités avec les boutons d'achat, de vente et de gestion des remises.

3 Modèle objet et règles métier

3.1 Classe abstraite `Product`

La classe `Product` représente la base commune à tous les produits :
— identifiant interne auto-incrémenté,
— nom du produit,
— prix d'achat,
— prix de vente,
— prix remisé (initialement nul),
— quantité en stock.
Cette classe contient également des attributs statiques pour :
— le capital initial (30000),
— le coût cumulé de tous les achats,
— le revenu cumulé de toutes les ventes.
Le capital est calculé dynamiquement via la formule :

$$capitalcourant = capitalinitial + income - cost$$

La classe impose plusieurs règles :
— les prix doivent être strictement positifs ;
— le prix de vente ne peut pas être inférieur au prix d'achat ;
— la quantité en stock ne peut pas être négative ;

— une vente ne peut pas dépasser le stock disponible ;
— un achat ne peut pas dépasser le capital disponible, sinon une erreur est levée.

Les méthodes d'achat et de vente mettent à jour à la fois le stock et les indicateurs financiers. Lors d'une vente, le prix effectivement utilisé est le prix remisé s'il est strictement positif, sinon le prix de vente normal.

3.2 Sous-classes `Clothes`, `Shoes`, `Accessories`

Clothes Les vêtements possèdent un attribut de taille. Nous imposons que :
— la taille soit comprise entre 34 et 54,
— la taille soit paire, conformément à l'énoncé de l'exercice d'origine.

En cas de valeur invalide, une exception est levée. La méthode de remise fixe le prix remisé à 30 de réduction sur le prix de vente de base.

Shoes Les chaussures possèdent un attribut de pointure. La règle métier impose une pointure comprise entre 36 et 50 (bornes incluses). En dehors de cet intervalle, une erreur est levée. La remise appliquée est de 20.

Accessories Les accessoires ne possèdent pas de champ de taille spécifique. La remise appliquée est de 50 sur le prix de vente.

3.3 Interface `Discount`

L'interface `Discount` définit les pourcentages de réduction par catégorie et les deux opérations :
— application de la remise : calcul du nouveau prix remisé en fonction du type de produit,
— suppression de la remise : remise du prix remisé à zéro, ce qui fait revenir le produit à son prix de vente normal.

Chaque sous-classe de `Product` implémente cet interface en fonction de son pourcentage de réduction.

4 Base de données MySQL

4.1 Schéma utilisé

Nous avons choisi d'utiliser une base MySQL nommée `store`. Pour simplifier l'implémentation, nous utilisons une seule table `products` contenant :
— un identifiant auto-incrémenté,
— le nom du produit,
— le prix d'achat,
— le prix de vente,
— le prix remisé (initialement nul),
— la quantité en stock,
— un champ *type* qui indique la catégorie (Clothes, Shoes, Accessories),
— un champ *size* qui stocke soit la taille de vêtement, soit la pointure, soit - pour les accessoires.

Ce choix permet de charger facilement tous les produits avec une seule requête et de reconstruire côté Java l'objet métier correspondant à chaque ligne de la table.

4.2 Accès aux données

La classe `DBManager` encapsule les opérations suivantes :

— connexion à la base de données (URL, utilisateur, mot de passe),
— chargement de tous les produits de la table et reconstruction des objets `Clothes`, `Shoes` ou `Accessories` selon le champ *type*,
— mise à jour d'un produit après une opération d'achat, de vente ou de remise (mise à jour du stock et du prix remisé),
— mise à jour en masse de plusieurs produits lors de l'application ou de la suppression globale des remises,
— suppression d'un produit lorsque le stock est nul et que l'utilisateur confirme la suppression.

Lors du chargement, si une ligne de la base contient des données incohérentes par rapport aux règles métier (par exemple, une taille de chaussure invalide), le produit est ignoré et une suppression automatique peut être effectuée pour garder une base saine.

4.3 Limites du schéma

Le sujet suggère une base « bien conçue », avec éventuellement plusieurs tables distinctes pour chaque catégorie de produit. Pour des raisons de simplicité, nous avons retenu une seule table `products` avec un champ *type*. Cette simplification fonctionne pour le projet, mais limite la normalisation et l'extensibilité du schéma.

5 Interface graphique et interaction utilisateur

5.1 Fenêtre principale

La fenêtre principale est définie dans le fichier `Mainview.fxml`. Elle se compose des éléments suivants :

— une barre supérieure affichant le capital courant, le revenu total, le coût total et le profit,
— un tableau central (`TableView`) affichant la liste des produits avec les colonnes :
  — nom,
  — type,
  — prix d'achat,
  — prix de vente,
  — prix remisé,
  — quantité en stock,
  — taille ou pointure.
— une colonne de boutons à gauche pour filtrer les produits par catégorie, trier par prix, éditer la taille d'un produit ou supprimer un produit,
— une barre inférieure avec :
  — un champ de saisie de quantité,
  — des boutons Buy et Sell pour acheter ou vendre des articles du produit sélectionné,
  — des boutons pour appliquer ou supprimer les remises sur tous les produits.

Des boîtes de dialogue sont utilisées pour afficher les erreurs (par exemple quantité invalide, absence de sélection) ou pour confirmer la suppression d'un produit.

5.2 Exemples de captures d'écran

Dans cette section, nous insérons des captures d'écran de l'application :

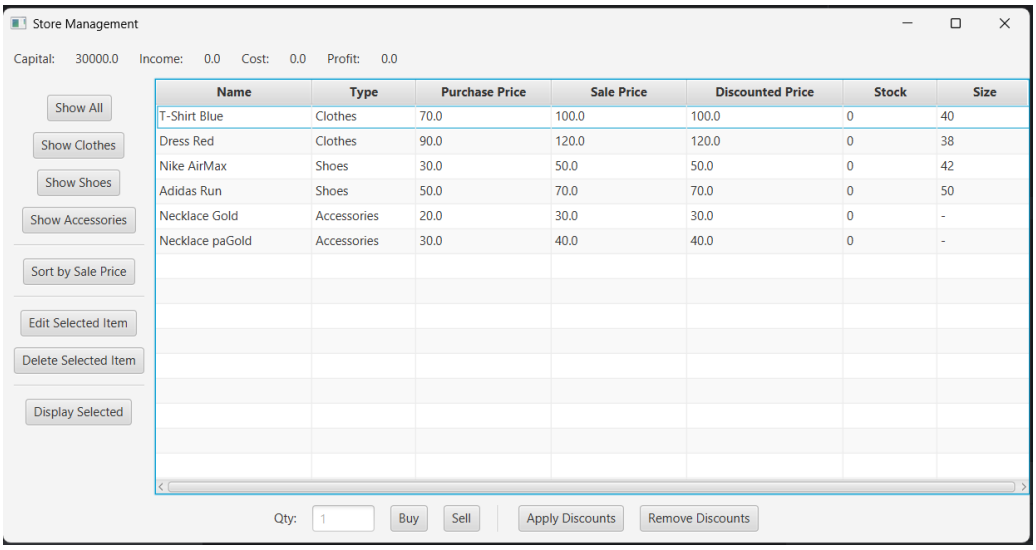

FIGURE 1 – Fenêtre principale de l'application WomenShop.

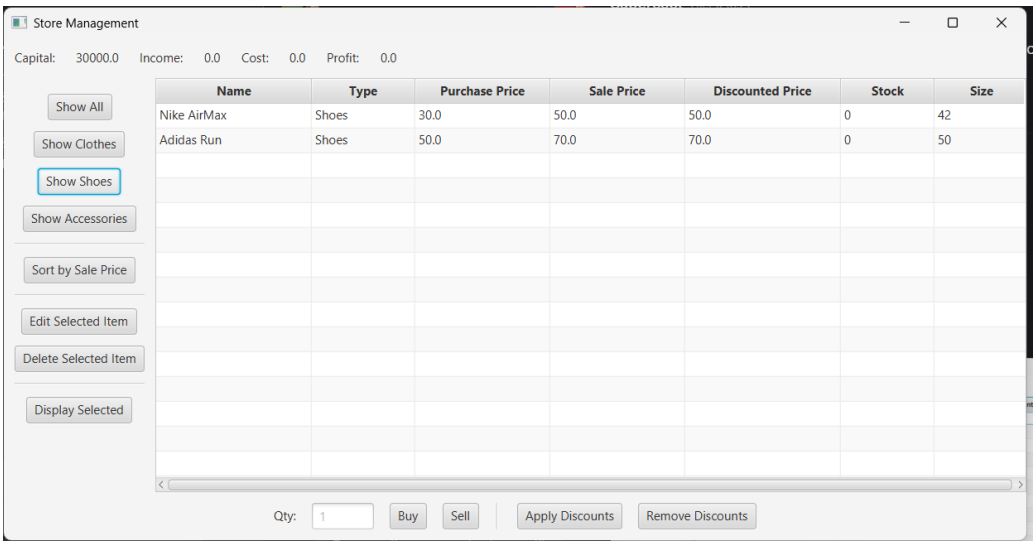

FIGURE 2 – Filtrage par catégorie et application des remises.

6 Fonctionnalités implémentées

Le sujet liste plusieurs fonctionnalités à réaliser. Nous détaillons ci-dessous celles qui sont implémentées dans notre application.

1. **Affichage des produits et du stock** :
   Tous les produits de la table `products` sont chargés au démarrage et affichés dans un tableau avec leur quantité en stock. L'utilisateur peut sélectionner un produit pour effectuer des opérations.
2. **Filtrage par catégorie** :
   Des boutons dédiés permettent de filtrer la liste sur les vêtements, les chaussures ou les accessoires. Un bouton supplémentaire permet de revenir à l'affichage de tous les produits.
3. **Tri par prix** :
   Un bouton Sort by Sale Price trie les produits affichés par prix de vente croissant.

4. **Ajout / modification / suppression de produits** :

   — *Ajout de produit* : non implémenté dans la version actuelle de l'interface.
   — *Modification* : nous permettons la modification de la taille ou de la pointure d'un produit via une boîte de dialogue.
   — *Suppression* : la suppression d'un produit est possible uniquement si son stock est nul. Une boîte de confirmation est affichée avant la suppression effective dans la base de données.

5. **Affichage du capital, des revenus et des coûts** :
   Le capital initial est fixé à 30000. Après chaque opération d'achat ou de vente, les valeurs de revenu total et de coût total sont mises à jour et affichées, ainsi que le profit (revenu moins coût). Ces indicateurs sont gérés dans le modèle via des attributs statiques de la classe `Product`.

6. **Achat et vente d'articles par produit** :
   À partir du produit sélectionné, l'utilisateur saisit une quantité puis clique sur Buy ou Sell. Les règles suivantes sont appliquées :
   — en cas d'achat, si le capital disponible n'est pas suffisant, une erreur est levée ;
   — en cas de vente, si la quantité demandée dépasse le stock, l'opération est refusée ;
   — en cas de succès, le stock, le capital, le coût et les revenus sont mis à jour, et la base de données est synchronisée.

7. **Application et suppression des remises** :
   Deux boutons permettent d'appliquer ou de retirer les remises sur l'ensemble des produits. Les remises suivent les pourcentages fixés par la spécification selon le type de produit. Le prix remisé est sauvegardé en base, ce qui permet de garder une cohérence entre l'affichage et les données.

8. **Validation des entrées et messages d'erreur** :
   Plusieurs validations sont mises en place :
   — vérification de la sélection d'un produit avant toute opération,
   — contrôle de la quantité saisie (valeur entière positive),
   — contraintes sur les tailles et pointures via les constructeurs et les accesseurs des classes `Clothes` et `Shoes`,
   — messages d'erreur affichés dans des boîtes de dialogue en cas d'entrée invalide ou d'opération impossible (stock insuffisant, budget insuffisant, etc.).

7 Fonctionnalités non réalisées ou partielles

Certaines fonctionnalités ou aspects du sujet n'ont pas été entièrement réalisés :

— **Ajout de nouveaux produits via l'interface graphique** :
  La version actuelle ne fournit pas encore de formulaire complet pour créer un produit (choix du type, des prix, de la taille, etc.) et l'enregistrer dans la base.
— **Modification complète des produits** :
  Nous ne modifions pour l'instant que la taille ou la pointure d'un produit. La modification du nom ou des prix pourrait être ajoutée dans une évolution future.
— **Persistance des indicateurs financiers** :
  Le capital, les revenus et les coûts sont maintenus en mémoire à partir des opérations réalisées pendant l'exécution. Ils ne sont pas encore stockés dans une table dédiée en base de données.
— **Schéma de base de données simplifié** :
  Le sujet suggère de ne pas se limiter à une seule table pour tous les produits. Pour simplifier le développement, nous avons choisi une table unique `products` avec un champ

de type. Une évolution naturelle serait de normaliser davantage la base (tables séparées pour les vêtements, les chaussures, etc.).

8 Conclusion

Ce projet nous a permis de mettre en pratique plusieurs concepts importants de la programmation orientée objet et de JavaFX : héritage et polymorphisme pour modéliser les différentes catégories de produits, pattern MVC pour structurer l'application, et pattern DAO pour séparer l'accès aux données de la logique métier.

L'application *WomenShop* permet aujourd'hui :

— d'afficher et filtrer les produits par catégorie,
— de trier les produits par prix,
— de gérer les opérations d'achat et de vente,
— d'appliquer et de retirer des remises selon la catégorie,
— de suivre l'évolution du capital, des revenus et des coûts.

Plusieurs pistes d'amélioration sont envisageables :

— ajout d'un écran complet de création et d'édition des produits,
— persistance des données financières dans la base,
— refonte du schéma de la base de données en plusieurs tables reliées,
— enrichissement de l'interface utilisateur avec davantage de retours visuels et de contrôles.

Malgré ces limites, le projet respecte l'essentiel des exigences fonctionnelles du sujet et fournit une base solide pour une application de gestion de stock de type WomenShop.