|  |
| --- |
| COMPTE RENDU JAVAFX |

|  |
| --- |
|  |

Logo

Description automatically generated with low confidenceLogo

Description automatically generated

Université Abdelmalek Essaâdi

Faculté des Sciences et Technique

de Tanger

|  |  |
| --- | --- |
|  | Réalisé par :   * Mohamed SERBOUT * Groupe : 1   Encadré par :   * Pr. El Mokhtar EN-NAIMI * [Lotfi EL AACHAK](https://elearning.fstt.ac.ma/user/view.php?id=80&course=1) |



Objectife

Mise en place d'une application desktop java , basée sur JDBC et JavaFX, l'application doit gérer les credit, les commandes , les produits, les clients, ainsi elle donne un tableau de board générique.

Outils : JavaFX, JDBC , Mysql , Java.

|  |
| --- |
| INTRODUCTION |

Qu'est-ce que JavaFx ?

JavaFX est un Framework et une bibliothèque d'interface utilisateur issue du projet OpenFX, qui permet aux développeurs Java de créer une interface graphique pour des applications de bureau, des applications internet riches et des applications smartphones et tablettes tactiles.

Créé à l'origine par Sun Microsystems, puis développé par Oracle après son rachat et ce, jusqu'à la version 11 du JDK, c'est depuis lors à la communauté OpenJFX que revient la poursuite de son développement1.

Cette bibliothèque a été conçue pour remplacer Swing et AWT, qui ont été développés à partir de la fin des années 90, pour pallier les défauts de ces derniers et fournir de nouvelles fonctionnalités (dont le support des écrans tactiles).

Scene Builder: l'outil de conception d'interfaces :

Scène Builder est un outil interactif de conception d'interface graphique pour JavaFX. Il permet de créer des interfaces utilisateurs rapidement et sans avoir besoin de coder: il en résulte des fichiers au format FXML qui sont ensuite chargés par le programme pour afficher une interface graphique à ses utilisateurs.

A picture containing text, screenshot, indoor, electronics

Description automatically generatedDéveloppé initialement par Oracle et sous le nom JavaFX Scène Builder5, son code source a été publié en open source à partir de sa version 2.06.

Code source :

A screenshot of a computer

Description automatically generated with medium confidence

Class : CustomerData.ja

Cette classe représente les données d'achat d'un client. La classe a 11 membres de données privées.

La classe a un constructeur avec des paramètres qui prend des valeurs pour tous les membres de données privées et les initialise. La classe possède également des méthodes getter et setter pour chaque membre de données privées, qui permettent à l'utilisateur de récupérer et de modifier respectivement les valeurs des membres de données privées.

Dans l'ensemble, cette classe offre un moyen pratique de représenter et de manipuler les données d'achat des clients.

Graphical user interface, application

Description automatically generatedClass : DataBase.java

Cette classe "*Database*" fournit une méthode statique nommée "*connectDb()"* qui renvoie un objet de connexion à une base de données MySQL.

La méthode utilise le pilote *JDBC* pour établir une connexion avec la base de données MySQL exécutée sur la machine locale. L'URL de la base de données est "*jdbc:mysql://localhost:/projet-11*", où "*projet-11*" est le nom de la base de données. Le nom d'utilisateur et le mot de passe pour la base de données MySQL sont respectivement définis sur "root" et une chaîne vide ("").

La méthode intercepte toutes les exceptions qui pourraient se produire pendant le processus de connexion et imprime la trace de la pile sur la console. Si la connexion réussit, elle renvoie l'objet de connexion, sinon, elle renvoie *null*.

Dans l'ensemble, cette classe fournit un moyen pratique d'établir une connexion avec une base de données MySQL à partir d'une application Java.

Class : ProductData.java

Cette classe "*ProductData*" qui contient des champs et des méthodes pour représenter les données d'un produit. Les champs de la classe sont :

* *productId* : un entier représentant l'identifiant unique du produit.
* *nameProduct* : une chaîne représentant le nom du produit.
* *prix* : un Double représentant le prix du produit.
* *status* : une chaîne représentant l'état du produit.
* *date* : un objet Date représentant la date à laquelle le produit a été ajouté.

La classe a un constructeur qui prend les valeurs des champs ci-dessus comme paramètres et les initialise.

La classe a également des getters et des setters pour chacun des champs, qui permettent aux autres classes d'accéder et de modifier les valeurs de ces champs.

Class : HelloController.java

A screenshot of a computer

Description automatically generatedCette classe de contrôleur qui contrôle le comportement de l'interface graphique d'une application. L'interface graphique est conçue à l'aide de fichiers FXML, qui définissent la disposition et les composants de l'interface. Cette classe de contrôleur interagit avec le fichier FXML pour gérer les événements utilisateur, tels que cliquer sur des boutons ou saisir du texte dans des champs de texte.

Le code commence par importer les classes JavaFX et SQL nécessaires. Ensuite, il définit la classe contrôleur *HelloController* qui implémente l'interface *Initializable*. Cette interface nécessite l'implémentation d'une méthode d'initialisation, qui n'est pas utilisée dans ce code.

Le code définit quelques variables d'instance, notamment des boutons, des champs de texte et un objet de connexion à la base de données. Ces variables sont annotées avec l'annotation *@FXML* pour permettre au framework de les injecter à partir du fichier *FXML* correspondant

Graphical user interface, application

Description automatically generated

La méthode *login()* est la méthode principale exécutée lorsque l'utilisateur clique sur le bouton de connexion. Il vérifie d'abord si les champs nom d'utilisateur et mot de passe ne sont pas vides. S'ils sont vides, un message d'erreur s'affiche. S'ils ne sont pas vides, la méthode récupère le nom d'utilisateur et le mot de passe saisis dans les champs de texte, crée une instruction préparée avec une requête *SQL* pour récupérer les données de la table de base de données admin et exécute la requête. Si la requête renvoie un résultat, c'est-à-dire que le nom d'utilisateur et le mot de passe correspondent, l'utilisateur est redirigé vers le tableau de bord de l'application. Sinon, un message d'erreur s'affiche.

Lorsque l'utilisateur est redirigé vers le tableau de bord, une nouvelle étape (fenêtre) est créée avec un style transparent. L'élément racine est le nœud de niveau supérieur du graphe scénique. Les événements de souris sur le nœud racine sont gérés pour permettre de faire glisser la fenêtre lorsque l'utilisateur clique et fait glisser la souris sur l'interface.

La méthode *close()* est exécutée lorsque l'utilisateur clique sur le bouton de fermeture. Il quitte simplement l'application en appelant la méthode *System.exit().*

Dans l'ensemble, ce code implémente une interface de connexion de base pour une application qui interagit avec une base de données pour authentifier les utilisateurs. Lorsque l'utilisateur est authentifié avec succès, l'application ouvre une nouvelle fenêtre de tableau de bord.

Graphical user interface, chart

Description automatically generatedClass : DashboardController.java

Cette classe pour un contrôleur de tableau de bord qui gère différents formulaires et tableaux, tels que l'ajout, la suppression et la mise à jour de produits et de clients, l'affichage de graphiques, etc.

La classe *DashboardController* implémente l'interface *Initializable*, ce qui signifie qu'elle a une méthode *initialize()* qui initialise le contrôleur après que son élément racine a été complètement traité.

Le code contient plusieurs annotations @FXML avant les variables représentant les composants de l'interface utilisateur tels que les boutons, les champs de texte et les tableaux. Ces annotations sont utilisées par JavaFX pour injecter les composants UI définis dans le fichier FXML dans les variables correspondantes.

Par exemple, *@FXML private AnchorPane main\_form* ; injecte *AnchorPane* avec l'attribut fx:id défini sur "main\_form" défini dans le fichier FXML dans la variable *main\_form*.

Les outils de base de données tels que *connect*, *prepare*, *result* et *statement* sont utilisés pour se connecter à la base de données, préparer des instructions, exécuter des requêtes et obtenir des résultats.

Graphical user interface, text, application

Description automatically generatedGraphical user interface

Description automatically generated

* La méthode *availbleProductAdd()* est responsable de l'ajout de nouveaux produits à la base de données lorsque le bouton Ajouter est cliqué. Il vérifie d'abord si tous les champs obligatoires sont remplis, puis vérifie si le produit avec le même ID existe déjà dans la base de données. Si le produit n'existe pas, il insère les nouvelles données du produit dans la base de données.
* La méthode *availbleProductReset()* qui réinitialise les valeurs de certains éléments de l'interface graphique. Plus précisément, il définit le texte de *available\_productID* et *available\_nameProduct* sur une chaîne vide, efface la sélection de *available\_status*, définit le texte de *available\_price* sur une chaîne vide et définit la valeur de *available\_date* sur *null*.
* Cette méthode *availbleProductUpdate()* qui met à jour un enregistrement dans une table de base de données en fonction de l'entrée de l'utilisateur via une interface graphique. La méthode vérifie si tous les champs obligatoires sont remplis par l'utilisateur et affiche un message d'erreur si l'un d'entre eux est vide. Si tous les champs sont remplis, la méthode affiche un message de confirmation demandant à l'utilisateur de confirmer la mise à jour. Si l'utilisateur confirme, la méthode met à jour l'enregistrement dans la base de données et affiche un message de réussite. Enfin, la méthode met à jour les éléments de l'interface graphique avec les nouvelles données et les réinitialise à leurs valeurs par défaut.
* Cette méthode "*availbleProductDelete()"* qui supprime un enregistrement de produit d'une table de base de données nommée "produit". La méthode effectue les opérations suivantes :
  + - Construit une requête SQL pour supprimer un enregistrement de la table "*product*" en fonction de la valeur "*product\_id*" extraite d'un champ de texte nommé "*available\_productID*".
    - Vérifie si tous les champs obligatoires (ID produit, nom, statut, prix et date) sont remplis et affiche un message d'erreur si l'un des champs est vide.
    - Affiche une boîte de dialogue de confirmation pour demander à l'utilisateur s'il est sûr de vouloir supprimer le produit, puis exécute la requête de suppression si l'utilisateur confirme.
    - Affiche un message de réussite après la suppression du produit, met à jour une vue de tableau avec les données actuelles et réinitialise les champs.
    - Cette méthode fournit une fonctionnalité pour supprimer un enregistrement de produit d'une table de base de données et garantit également que les champs obligatoires sont remplis avant la suppression
* Cette méthode *availbleProductData* qui renvoie une *ObservableList* d'objets *ProductData*. La méthode se connecte à une base de données, exécute une requête SQL pour sélectionner toutes les lignes de la table *product*, puis parcourt le jeu de résultats pour créer des objets *ProductData* et les ajouter à *ObservableList*.  
  Les objets *ProductData* sont créés à l'aide des valeurs de plusieurs colonnes du jeu de résultats, telles que *product*\_id, name\_*product*, *status*, *price* et *date*.
* Cette *method* *availableBShowProductData* qui remplit un *TableView* avec des données et permet de rechercher/filtrer les données en fonction de l'entrée de l'utilisateur. Le *TableView* est rempli avec une liste d'objets *ProductData* obtenus à partir de la méthode availbleProductData.
* Méthode productSIdList() : récupère tous les produits disponibles dans la base de données et remplit une liste déroulante (productSelect\_ID) avec leurs identifiants.
* Méthode *productSNameProductList()*: récupère tous les produits disponibles dans la base de données et remplit une autre liste déroulante (*productSelect\_nameProduct*) avec leurs noms.
* Méthode *productSTypeList()* : crée une liste de chaînes de statut (*statusList*), la convertit en une liste observable et remplit une liste déroulante (*productSelect\_type*) avec les statuts.
* Méthode *QuantityList()* : crée une liste de chaînes représentant des quantités de 1 à 40, la convertit en une liste observable et remplit une liste déroulante (*productSelect\_quantity*) avec les quantités.
* Méthode productSelected() : récupère les données de plusieurs éléments de l'interface graphique (tels que le prénom, le nom et la quantité du produit), interroge la base de données pour obtenir le prix du produit sélectionné, calcule le prix total et renseigne plusieurs éléments de l'interface graphique (tels que une étiquette affichant le prix total). Il Graphical user interface, application

  Description automatically generatedGraphical user interface

  Description automatically generatedaffiche également une boîte d'alerte avec un message de réussite.
* Méthode *productSelectedReset()* : réinitialise certains des éléments de l'interface graphique renseignés par la méthode *productSelected().*
* La méthode *productSelectedPay()* récupère les données de divers champs de saisie, interroge une base de données pour obtenir le prix d'un produit sélectionné, calcule le coût total en fonction de la quantité et du type de produit sélectionnés et insère les données de paiement dans une table de base de données. Il affiche également une boîte de dialogue de confirmation et réinitialise les champs de saisie une fois le paiement effectué.
* Graphical user interface, application

  Description automatically generatedLa méthode *productSelectedPayReset()* réinitialise les champs de saisie à leurs valeurs par défaut.
* La méthode *customersDataList()* interroge une base de données et crée une *ObservableList* d'objets *CustomerData()* en fonction du résultat. Chaque objet *CustomerData* représente une ligne dans la table des clients de la base de données.
* La méthode, *customersShowDataList(),* appelle *customersDataList()* pour obtenir la *ObservableList* des objets *CustomerData*, puis utilise cette liste pour remplir un *TableView*. Il configure également un champ de recherche et filtre le *TableView* en fonction de la requête de recherche de l'utilisateur.
* La méthode *dashboardDisplayAP()* interroge la base de données pour obtenir le nombre de produits disponibles et l'affiche dans un composant d'interface utilisateur appelé *dashboard*\_availble*()*.
* La méthode *dashboardDisplayIT()* interroge la base de données pour obtenir le revenu total gagné aujourd'hui et l'affiche dans un composant d'interface utilisateur appelé *dashboard*\_ *incometoday()*.
* La méthode *dashboardDisplayTA()* interroge la base de données pour obtenir le montant total gagné de tous les clients et l'affiche dans un composant d'interface utilisateur appelé *dashboard*\_totalAmount.
* La méthode *displayUsername()* affiche le nom d'utilisateur de l'utilisateur actuellement connecté dans un composant d'interface utilisateur appelé *username*.
* La méthode *dashboardChart()* affiche un graphique des revenus gagnés chaque jour au cours des 9 derniers jours.
* Le programme comprend également des méthodes qui basculent entre différents formulaires dans l'interface utilisateur du système POS.
* La méthode *switchForm()* est appelée lors d'un clic sur un bouton et affiche le formulaire correspondant tout en masquant les autres formulaires.
* La méthode *availbleBSelectProductData()* affiche les détails d'un produit disponible sélectionné dans les composants de l'interface utilisateur.
* La méthode *defaultBtn* définit le style par défaut des boutons de navigation dans l'interface utilisateur. Enfin, la méthode *switchForm()* affiche des données différentes selon le formulaire en cours d'affichage.

Graphical user interface, text, application

Description automatically generated

J'espère que vous aimez le travail