

Rapport de projet :  
E-SALAF  
Pour le module : Programmation  
Orientée Objet en Java

**Réalisé par :**

KHOUSI Imane

**Encadré par :**

Elaachak Lotfi

EN-NAIMI El Mokhtar

# Remerciement

*Avant tout développement, nous souhaitons exprimer nos sincères remerciements à notre Professeur Lotfi EL AACHAK et à Monsieur Ennaimi Mokhtar, qui nous ont donné l'opportunité de réaliser ce projet pratique en JavaFX. Leurs conseils, leur expertise et leur accompagnement ont été précieux tout au long de ce travail. Nous avons été très motivés pour atteindre le niveau d'excellence attendu de notre part et nous espérons que ce projet sera à la hauteur de leurs attentes. Encore une fois, nous tenons à remercier chaleureusement notre professeur et notre encadrant pour leur soutien et leur confiance*

## *Interface Graphique*

GUI (Graphical User Interface) en anglais, est un moyen visuel et interactif pour les utilisateurs d'interagir avec un programme informatique ou un système d'exploitation. Elle permet de naviguer à travers les fonctionnalités d'un logiciel, de cliquer sur des boutons, de saisir du texte et de visualiser des images ou des données sous forme graphique.

L'interface graphique se compose généralement de fenêtres, d'icônes, de menus déroulants, de barres d'outils et de boutons. Elle est conçue pour rendre l'utilisation du logiciel plus facile et plus intuitive, en fournissant une représentation visuelle des tâches que l'utilisateur peut accomplir.

Les interfaces graphiques sont très courantes dans les systèmes d'exploitation modernes, les applications bureautiques, les jeux vidéo et les applications mobiles.

## *Interface graphique en Java*

En Java, il est possible de créer une interface graphique à l'aide de deux bibliothèques : Swing et JavaFX

En Java, l'interface graphique peut être créée à l'aide de la bibliothèque Swing, qui fournit un ensemble de composants graphiques prêts à l'emploi tels que des boutons, des étiquettes, des champs de texte, des listes et des tableaux.

En Java, l'interface graphique peut également être créée à l'aide de JavaFX, une bibliothèque graphique plus récente que Swing, qui offre des fonctionnalités plus avancées et une meilleure intégration avec les technologies modernes telles que CSS et les applications mobiles

## IntelliJ IDEA :

IntelliJ IDEA est un environnement de développement intégré (IDE) pour le développement de logiciels en Java et dans d'autres langages de programmation tels que Kotlin, Groovy, Scala et Python. Il a été développé par JetBrains et est disponible sous licence Apache 2.0.



## JavaFX :

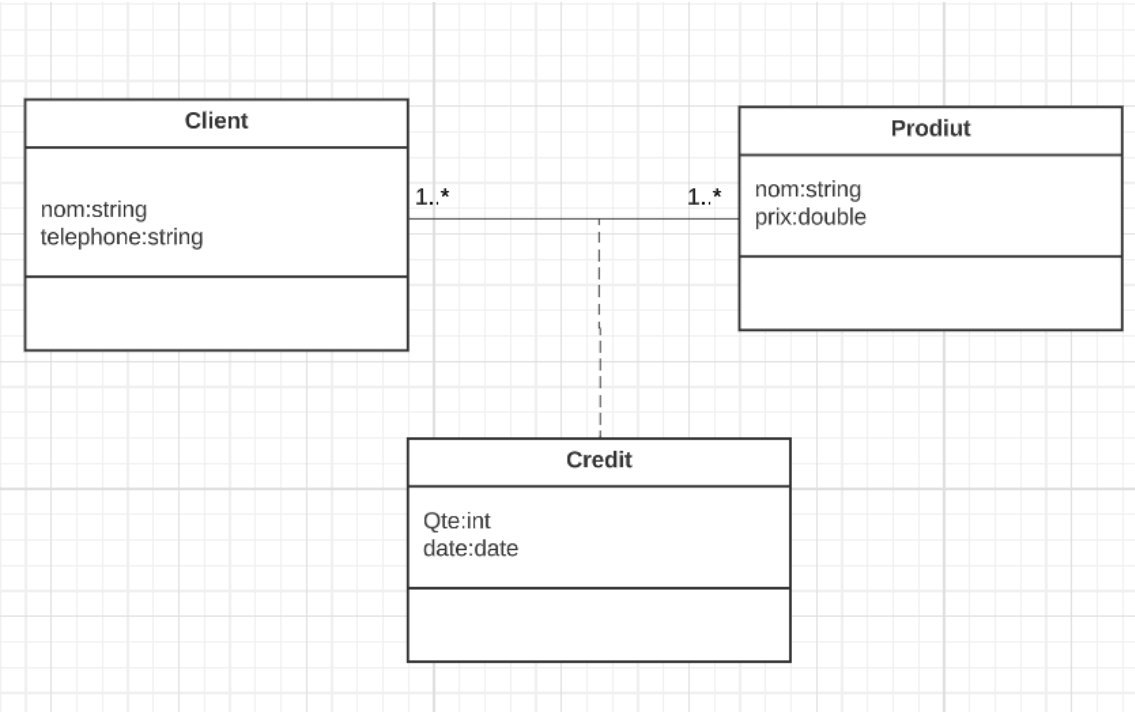
JavaFX est une plate-forme de développement d'interface utilisateur (UI) utilisée pour créer des applications de bureau et mobiles multi-plateformes. Elle est développée par Oracle et fournit une bibliothèque de composants graphiques, des outils pour la mise en page, la gestion d'événements, la liaison de données et la gestion des ressources. JavaFX utilise la langue de programmation Java pour la mise en œuvre de l'interface utilisateur et peut être intégré avec d'autres technologies Java telles que Swing et Java 2D. JavaFX est souvent utilisé pour développer des applications riches et interactives avec une interface utilisateur attrayante et moderne.

## Maven :

Maven est un outil de gestion de projet logiciel open-source, développé par la Fondation Apache. Il fournit une structure de projet standardisée, une gestion des dépendances, un cycle de vie de construction, une gestion de la documentation, ainsi que des plugins pour faciliter la compilation, les tests, l'emballage et le déploiement de projets Java. Maven utilise un fichier de configuration appelé "pom.xml" (Project Object Model) pour décrire le projet, ses dépendances, ses plugins et ses étapes de construction. En utilisant Maven, les développeurs peuvent gérer efficacement les projets logiciels de grande envergure, automatiser les tâches de construction, améliorer la qualité du code et faciliter le partage et la réutilisation de code entre différents projets.



Diagramme UML



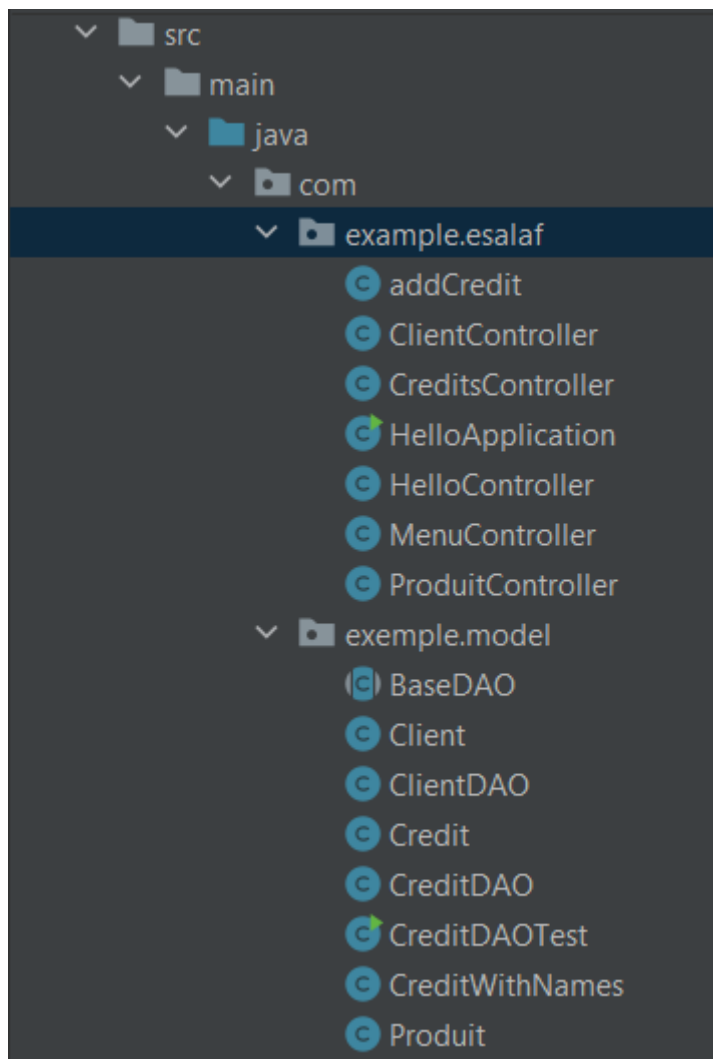
## **Création de la base de données :**

Table	Action	Lignes	Type	Interclassement	Taille	Perte
<input type="checkbox"/> client	★ Parcourir Structure Rechercher Insérer Vider Supprimer	14	InnoDB	utf8mb4_general_ci	16,0 kio	-
<input type="checkbox"/> credit	★ Parcourir Structure Rechercher Insérer Vider Supprimer	2	InnoDB	utf8mb4_general_ci	48,0 kio	-
<input type="checkbox"/> produit	★ Parcourir Structure Rechercher Insérer Vider Supprimer	5	InnoDB	utf8mb4_general_ci	16,0 kio	-

## **Connection a la base de données :**

```
public abstract class BaseDAO<T> {  
  
    protected Connection connection ;  
  
    protected Statement statement ;  
  
    protected PreparedStatement preparedStatement ;  
  
    protected ResultSet resultSet ;  
  
    private String url = "jdbc:mysql://127.0.0.1:3306/esalaf" ;  
    private String login = "root";  
    private String password = "";  
  
    public BaseDAO() throws SQLException {  
        this.connection = DriverManager.getConnection(url , login ,  
password);  
    }  
    public abstract void save(T object) throws SQLException ;  
  
    public abstract void update(T object) throws SQLException;  
  
    public abstract void delete(T object) throws SQLException;  
  
    public abstract T getOne(Long id) throws SQLException;  
  
    public abstract List<T> getAll() throws SQLException;  
}
```

## Les classes déclarées



### Table client :

<div><div><div></div><div></div><div></div></div></div>				id_client	nom	telephone
<input type="checkbox"/>		Éditer	 Copier	 Supprimer	1 Mohamed	0666089345
<input type="checkbox"/>		Éditer	 Copier	 Supprimer	2 yassine	0798553432
<input type="checkbox"/>		Éditer	 Copier	 Supprimer	3 Amina	079935321
<input type="checkbox"/>		Éditer	 Copier	 Supprimer	4 Anas	0699342517
<input type="checkbox"/>		Éditer	 Copier	 Supprimer	5 Ahmad	067745367
<input type="checkbox"/>		Éditer	 Copier	 Supprimer	6 Ali	0631246788

### Table des credits :

<div><div></div><div></div><div></div></div>						id_credit	id_client	id_produit	date	qte	
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	Éditer	<div><div></div><div></div><div></div></div>	Copier	<div><div></div><div></div><div></div></div>	Supprimer	1	1	1	2023-03-27	10
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	Éditer	<div><div></div><div></div><div></div></div>	Copier	<div><div></div><div></div><div></div></div>	Supprimer	2	1	1	2023-03-27	10
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	Éditer	<div><div></div><div></div><div></div></div>	Copier	<div><div></div><div></div><div></div></div>	Supprimer	3	6	5	2023-04-03	6

### Table Produit :

						id_produit	nom	prix	
<input type="checkbox"/>		Éditer		Copier		Supprimer	1	Lait	8.00
<input type="checkbox"/>		Éditer		Copier		Supprimer	3	Sucre	6.00
<input type="checkbox"/>		Éditer		Copier		Supprimer	4	Farine	20.00
<input type="checkbox"/>		Éditer		Copier		Supprimer	5	Sel	10.00
<input type="checkbox"/>		Éditer		Copier		Supprimer	6	Café	60.00



## Le menu

# E-salaf

Managing your credits with ease and security.

Clients

Products

Credits

```
public class MenuController {

    @FXML
    private Label menuLabel;

    @FXML
    private MenuBar menuBar;

    @FXML
    private Menu clientsMenu;

    @FXML
    private MenuItem viewClientsMenuItem;

    @FXML
    private Menu productsMenu;

    @FXML
    private MenuItem viewProductsMenuItem;

    @FXML
    private Menu creditsMenu;

    @FXML
    private MenuItem viewCreditsMenuItem;

    @FXML
    void viewClients(ActionEvent event) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(getClass().getResource("clients.fxml"));
        Parent root = fxmlLoader.load();
```

```

        Stage stage = new Stage();
        stage.setScene(new Scene(root));
        stage.show();
    }

    @FXML
    void viewProducts(ActionEvent event) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(getClass().getResource("produits.fxml"));
        Parent root = fxmlLoader.load();
        Stage stage = new Stage();
        stage.setScene(new Scene(root));
        stage.show();
    }

    @FXML
    void viewCredits(ActionEvent event) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(getClass().getResource("credits.fxml"));
        Parent root = fxmlLoader.load();
        Stage stage = new Stage();
        stage.setScene(new Scene(root));
        stage.show();
    }
}

```

C'est classe qui gère une barre de menu. Cette classe comporte plusieurs champs annotés avec "@FXML", qui correspondent aux contrôles de la barre de menu créés dans le fichier FXML associé. Ces champs sont :

- menuLabel : une étiquette affichant le titre du menu.
- menuBar : la barre de menu principale.
- clientsMenu : le menu "Clients" dans la barre de menu.
- viewClientsMenuItem : l'élément de menu "Voir les clients" dans le menu "Clients".
- productsMenu : le menu "Produits" dans la barre de menu.
- viewProductsMenuItem : l'élément de menu "Voir les produits" dans le menu "Produits".
- creditsMenu : le menu "Crédits" dans la barre de menu.
- viewCreditsMenuItem : l'élément de menu "Voir les crédits" dans le menu "Crédits".

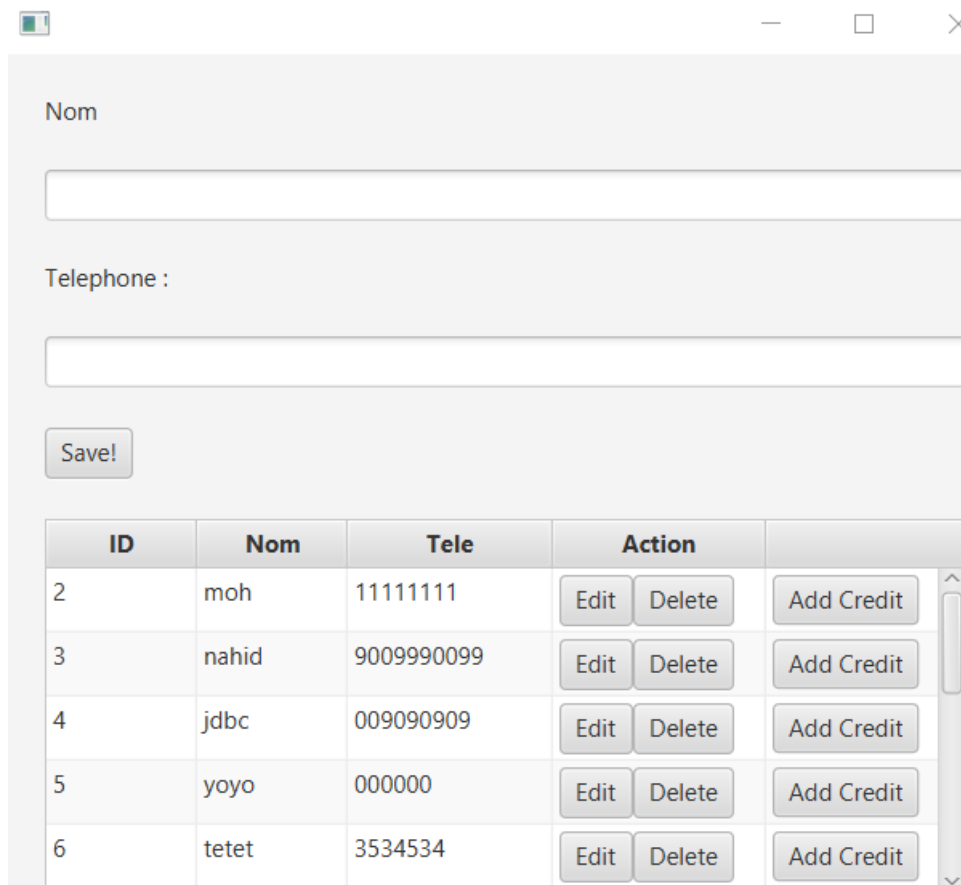
La classe comporte également trois méthodes annotées avec "@FXML", qui sont appelées lorsque l'utilisateur sélectionne un élément de menu particulier. Chacune de ces méthodes utilise la classe FXMLLoader pour charger un fichier FXML spécifique et créer une nouvelle fenêtre d'application à partir de son contenu. Les méthodes sont :

- viewClients(ActionEvent event) : cette méthode est appelée lorsque l'utilisateur sélectionne l'élément de menu "Voir les clients" dans le menu "Clients". Elle

charge le fichier FXML "clients.fxml", crée une nouvelle fenêtre d'application et affiche son contenu.

- `viewProducts(ActionEvent event)` : cette méthode est appelée lorsque l'utilisateur sélectionne l'élément de menu "Voir les produits" dans le menu "Produits". Elle charge le fichier FXML "produits.fxml", crée une nouvelle fenêtre d'application et affiche son contenu.
- `viewCredits(ActionEvent event)` : cette méthode est appelée lorsque l'utilisateur sélectionne l'élément de menu "Voir les crédits" dans le menu "Crédits". Elle charge le fichier FXML "credits.fxml", crée une nouvelle fenêtre d'application et affiche son contenu.

## CLIENT :



ID	Nom	Tele	Action	
2	moh	11111111	Edit Delete Add Credit	
3	nahid	9009990099	Edit Delete Add Credit	
4	jdbc	009090909	Edit Delete Add Credit	
5	yoyo	000000	Edit Delete Add Credit	
6	tetet	3534534	Edit Delete Add Credit	

### CLASS Client.java :

Il s'agit d'une classe Java appelée "Client" qui représente un modèle ou une entité d'un objet client. La classe a trois champs privés : "id\_client" de type Long, "nom" de type String et "telephone" de type String. Ces champs représentent l'identifiant unique du client, son nom et son numéro de téléphone, respectivement.

La classe contient trois constructeurs : un constructeur par défaut sans paramètres, un constructeur prenant en paramètres le nom et le numéro de téléphone du client, et un constructeur prenant en paramètres l'identifiant, le nom et le numéro de téléphone du client.

La classe contient également des méthodes d'accès (getters) et de modification (setters) pour les trois champs privés, ainsi qu'une méthode toString() qui retourne une représentation sous forme de chaîne de caractères de l'objet client.

## **Class ClientController :**

La classe ClientController implémente l'interface Initializable, qui permet d'initialiser le contrôleur une fois que le fichier FXML a été chargé.

Le code contient plusieurs éléments importants, notamment :

- L'utilisation de la classe Client, qui représente un client et contient des informations sur son nom, son téléphone et son ID.
- L'utilisation de la classe ClientDAO, qui gère la communication avec la base de données pour enregistrer, mettre à jour et supprimer des clients.
- La création d'une TableView, qui affiche les données des clients sous forme de tableau, avec des colonnes pour le nom, le téléphone et les actions à effectuer.
- L'utilisation de PropertyValueFactory pour lier les données des clients aux colonnes de la TableView.
- La définition d'un gestionnaire d'événements pour le bouton "Save", qui ajoute un nouveau client à la TableView et à la base de données.
- La définition d'un gestionnaire d'événements pour les boutons "Edit" et "Delete" dans la TableView, qui permettent respectivement de modifier et de supprimer un client sélectionné.
- La définition d'un gestionnaire d'événements pour le bouton "Add Credit" dans la TableView, qui ouvre une nouvelle fenêtre pour ajouter du crédit à un client sélectionné.

## **Class AddCredit :**

```
public class addCredit implements Initializable
{
    @FXML
    private TextField Nom;

    @FXML
    private TextField Tele;

    @FXML
    private Label id_cli;

    @FXML
    private TextField qte;

    private long product_id;
    private long client_id;

    public void setSelecetdClientName(String name, String Tele, Long id) {
        this.Nom.setText(name);
        this.Tele.setText(Tele);
        this.client_id = id;
        this.id_cli.setText(String.valueOf(id));
    }
}
```

```

        public void setSelectedClientId(long id) {
            this.client_id = id;
            this.id_cli.setText(String.valueOf(id));
        }

        @FXML private MenuButton produit;

        @Override
        public void initialize(URL location, ResourceBundle resources) {
            try {
                ProduitDAO produitDAO = new ProduitDAO();
                List<Produit> productList = produitDAO.getAll();
                addProductsToMenu(productList);
            } catch (SQLException ex) {
                // Handle SQL exception
            }
        }

        private void addProductsToMenu(List<Produit> productList) {
            for (Produit product : productList) {
                MenuItem menuItem = new MenuItem(product.getNom());

                // Set the product id as the id property of the MenuItem
                menuItem.setId(String.valueOf(product.getId()));
                menuItem.setOnAction(event -> {
                    // Get the selected product name
                    String selectedProduct = ((MenuItem) event.getSource()).getText();
                    // Set the text of the MenuButton to the selected product name
                    produit.setText(selectedProduct);

                    // Get the selected product id
                    String selectedProductId = ((MenuItem) event.getSource()).getId();
                    // Call a method with the selected product id
                    handleProductSelection(selectedProductId);
                });
                produit.getItems().add(menuItem);
            }
        }

        private void handleProductSelection(String productId) {
            // Do something with the selected product id
            System.out.println("Selected product id: " + productId);
            long id_prod=Long.parseLong(productId);
            product_id=id_prod;
        }

        public void onSave(){
            Credit cred = new Credit( client_id ,product_id);

            try {
                CreditDAO creditdao = new CreditDAO();

                creditdao.save(cred);

            } catch (SQLException e) {
                throw new RuntimeException(e);
            }
        }
    }
}

```

Le code importe les classes Credit, CreditDAO, Produit et ProduitDAO. Les classes Credit et CreditDAO sont utilisées pour accéder et manipuler les données de crédit dans une

base de données. Les classes Produit et ProduitDAO sont utilisées pour accéder et manipuler les données de produits dans une base de données.

La classe addCredit implémente l'interface Initializable de JavaFX, ce qui signifie qu'elle a une méthode initialize() qui est appelée lorsque la fenêtre est initialisée. Dans cette méthode, elle crée une instance de ProduitDAO et récupère tous les produits de la base de données à l'aide de la méthode getAll(). Elle ajoute ensuite ces produits à un MenuButton dans l'interface utilisateur en créant un MenuItem pour chaque produit et en les ajoutant au MenuButton.

La classe dispose également de méthodes pour gérer la sélection d'un produit et la sélection d'un client. Lorsqu'un produit est sélectionné dans le MenuButton, la méthode handleProductSelection() est appelée et l'ID du produit est stocké dans la variable product\_id. Lorsqu'un client est sélectionné, la méthode setSelectedClientName() est appelée et le nom, le numéro de téléphone et l'ID du client sont stockés dans les variables Nom, Tele et client\_id, respectivement.

La méthode onSave() est appelée lorsque l'utilisateur clique sur un bouton "Enregistrer" dans l'interface utilisateur. Elle crée une instance de la classe Credit à l'aide de l'ID du client et de l'ID du produit sélectionnés, puis elle utilise CreditDAO pour enregistrer cette instance dans la base de données. Si une exception SQLException est levée, elle est propagée en tant que RuntimeException.

**Add credit**

Nom

Téléphone

Produit

client ID : 2

## Produit:

id	Product name	Price	Action
1	Miel	200.0	<button>Edit</button> <button>Delete</button>
3	Product 1	100.0	<button>Edit</button> <button>Delete</button>
4	Farine	20.0	<button>Edit</button> <button>Delete</button>
5	Product 1	100.0	<button>Edit</button> <button>Delete</button>

Product name

Price

Add product

Edit Produit ×

Nom:

Prix:

Save Cancel

## Class Produit .java :

Ce code définit une classe Java nommée "Produit" dans le package "com.exemple.model". La classe a trois variables d'instance, à savoir "id" de type Long, "nom" de type String et "prix" de type double. La classe a trois constructeurs - un constructeur par défaut qui ne prend aucun argument, un constructeur prenant les arguments "nom" et "prix", et un constructeur prenant les arguments "id", "nom" et "prix". La classe a également des getters et des setters pour les trois variables d'instance. Enfin, la classe a une méthode "toString()" qui renvoie une chaîne de caractères décrivant l'objet "Produit" avec les valeurs des variables d'instance.

## Class ProduitDAO :



### Enregistrer Produit :

```
public void save(Produit object) throws SQLException {
    String req = "insert into produit (nom, prix) values (?, ?)";
    this.preparedStatement = this.connection.prepareStatement(req);
    this.preparedStatement.setString(1, object.getNom());
    this.preparedStatement.setDouble(2, object.getPrix());
    this.preparedStatement.execute();
}
```

Ce code définit une méthode "save" qui prend un objet "Produit" en paramètre et qui insère les valeurs de cet objet dans une table nommée "produit" dans une base de données via une requête SQL. La méthode utilise un objet "PreparedStatement" pour préparer la requête SQL en y incluant les valeurs de nom et de prix de l'objet "Produit" passé en paramètre. Ensuite, la méthode exécute la requête SQL avec la méthode "execute()" de l'objet "PreparedStatement". Si la requête réussit, elle insère les données dans la table "produit". Si la requête échoue, une exception "SQLException" est levée.

### Modifier Produit :

```
public void update(Produit object) throws SQLException {
    String req = "update produit set nom=?, prix=? where id_produit=?";
    this.preparedStatement = this.connection.prepareStatement(req);
    this.preparedStatement.setString(1, object.getNom());
    this.preparedStatement.setDouble(2, object.getPrix());
    this.preparedStatement.setLong(3, object.getId());
    this.preparedStatement.executeUpdate();
}
```

Ce code définit une méthode "update" qui prend un objet "Produit" en paramètre et qui met à jour les valeurs correspondantes de cet objet dans une table nommée "produit" dans une base de données via une requête SQL. La méthode utilise un objet "PreparedStatement" pour préparer la requête SQL en y incluant les nouvelles valeurs de nom, de prix et l'identifiant de l'objet "Produit" passé en paramètre. Ensuite, la méthode exécute la requête SQL avec la méthode "executeUpdate()" de l'objet "PreparedStatement". Si la requête réussit, elle met à jour les données dans la table "produit". Si la requête échoue, une exception "SQLException" est levée.

### Supprimer produit :

```
public void delete(Produit object) throws SQLException {
    String req = "delete from produit where id_produit=?";
    this.preparedStatement = this.connection.prepareStatement(req);
    this.preparedStatement.setLong(1, object.getId());
    this.preparedStatement.executeUpdate();
}
```

Ce code définit une méthode "delete" qui prend un objet "Produit" en paramètre et qui supprime cet objet de la table "produit" dans une base de données via une requête SQL. La méthode utilise un objet "PreparedStatement" pour préparer la requête SQL en y incluant l'identifiant de l'objet "Produit" passé en paramètre. Ensuite, la méthode exécute la requête SQL avec la méthode "executeUpdate()" de l'objet "PreparedStatement". Si la requête réussit, elle supprime l'enregistrement correspondant dans la table "produit". Si la requête échoue, une exception "SQLException" est levée.

### Récupère un Produit a partir de son Id :

```
public Produit getOne(Long id) throws SQLException {
    String req = "select * from produit where id_produit=?";
    this.preparedStatement = this.connection.prepareStatement(req);
    this.preparedStatement.setLong(1, id);
    this.resultSet = this.preparedStatement.executeQuery();

    if (this.resultSet.next()) {
        return new Produit(this.resultSet.getLong(1),
this.resultSet.getString(2), this.resultSet.getDouble(3));
    }

    return null;
}
```

Ce code définit une méthode "getOne" qui prend un identifiant de produit en paramètre et qui renvoie l'objet "Produit" correspondant à cet identifiant dans une table nommée "produit" dans une base de données via une requête SQL. La méthode utilise un objet "PreparedStatement" pour préparer la requête SQL en y incluant l'identifiant de produit passé en paramètre. Ensuite, la méthode exécute la requête SQL avec la méthode "executeQuery()" de l'objet "PreparedStatement" qui renvoie un objet "ResultSet" contenant les résultats de la requête SQL.

Si le "ResultSet" renvoie au moins un enregistrement, la méthode crée un nouvel objet "Produit" en récupérant les valeurs de l'enregistrement courant (id, nom et prix) avec les méthodes "getLong", "getString" et "getDouble" de l'objet "ResultSet". La méthode renvoie ensuite cet objet "Produit". Si le "ResultSet" est vide, la méthode renvoie "null".

## Récupérer tous les produits :

```
public List<Produit> getAll() throws SQLException {
    List<Produit> produits = new ArrayList();
    String req = "select * from produit";
    this.statement = this.connection.createStatement();
    this.resultSet = this.statement.executeQuery(req);

    while (this.resultSet.next()) {
        Produit produit = new Produit();
        produit.setId(this.resultSet.getLong("id_produit"));
        produit.setNom(this.resultSet.getString("nom"));
        produit.setPrix(this.resultSet.getDouble("prix"));
        produits.add(produit);
    }

    return produits;
}
```

Cette méthode récupère tous les produits à partir d'une base de données et les stocke dans une liste en utilisant les étapes suivantes :

Créer une liste vide pour stocker les produits : `List<Produit> produits = new ArrayList();`.

Construire la requête SQL pour récupérer tous les produits : `String req = "select * from produit";`.

Exécuter la requête SQL et stocker les résultats dans un objet `ResultSet` : `this.resultSet = this.statement.executeQuery(req);`.

Itérer sur les résultats du `ResultSet`, créer un objet `Produit` pour chaque ligne, extraire les valeurs des colonnes `id_produit`, `nom` et `prix` et les stocker dans l'objet `Produit`, puis ajouter l'objet `Produit` à la liste de produits : `produits.add(produit);`.

## Récupérer le nom d'un Produit à partir de son id :

```
public String getProductName(Long id) {
    String req = "SELECT nom FROM produit WHERE id_produit = ?;";
    try {
        this.preparedStatement =
this.connection.prepareStatement(req);
        this.preparedStatement.setLong(1, id);
        this.resultSet = this.preparedStatement.executeQuery();
        if (this.resultSet.next()) {
            return this.resultSet.getString(1);
        }
    } catch (SQLException e) {
        // Handle any SQL exceptions that may occur
        e.printStackTrace();
    } finally {
```

```

        // Clean up resources
        try {
            if (this.resultSet != null) {
                this.resultSet.close();
            }
            if (this.preparedStatement != null) {
                this.preparedStatement.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    return null;
}

```

Cette méthode récupère le nom d'un produit à partir de son identifiant dans une base de données en utilisant les étapes suivantes :

Construire la requête SQL pour récupérer le nom du produit correspondant à l'identifiant.

Créer un objet PreparedStatement à partir de la requête SQL, définir la valeur de l'identifiant à l'aide de la méthode setLong() et exécuter la requête SQL .

Si la requête SQL retourne un résultat, extraire la valeur de la première colonne du ResultSet (dans ce cas, la colonne nom) et la retourner en tant que nom du produit. Sinon, retourner null .

### **Class ProduitController :**

Le code ci-dessus est un contrôleur JavaFX qui gère l'affichage et la manipulation des produits d'un magasin dans une table. Le code utilise une architecture de type modèle-vue-contrôleur (MVC) pour séparer les responsabilités du code. Le modèle des produits est défini dans la classe Produit et stocké dans la base de données à l'aide de la classe ProduitDAO. Le contrôleur est responsable de la vue (table et boutons) et de la liaison avec le modèle.

La méthode UpdateTable() définit les colonnes de la table, ajoute des boutons d'édition et de suppression, et remplit la table avec des données à partir du modèle. La méthode editProduit() crée une boîte de dialogue pour modifier les détails d'un produit sélectionné. La méthode onAddButtonClick() est appelée lorsqu'un utilisateur ajoute un nouveau produit à la table. Elle crée une nouvelle instance de Produit, l'ajoute à la base de données, et met à jour la table avec les données nouvellement ajoutées.

### **Le fichier Produit.fxml :**

Le fichier décrit une interface utilisateur qui comprend un bouton "Ajouter produit", une table de visualisation (TableView) avec quatre colonnes ("id", "Nom de produit", "Prix" et "Action"), ainsi que deux champs de texte (TextField) pour entrer le nom et le prix d'un produit.

Le fichier utilise des balises pour importer les classes JavaFX nécessaires (Button, Label, TableView, TableColumn, TextField, AnchorPane) et définit un contrôleur (ProduitController) qui est responsable de la gestion des événements associés aux éléments de l'interface utilisateur.

### Crédit :

Client Name	Product Name	Quantity	Date	Action
moh	Miel	5	2023-04-05	<button>Edit</button> <button>Delete</button>

### Class Crédit.java :

Cette classe Java définit une entité Crédit dans une application hypothétique. L'entité Crédit représente une transaction où un client achète une certaine quantité d'un produit.

La classe possède plusieurs constructeurs pour permettre différentes façons d'initialiser un objet Crédit.

Les propriétés de l'objet incluent :

"credit\_id" : un identifiant unique pour chaque transaction de crédit

"produit" : l'objet Produit acheté

"client" : l'objet Client qui a effectué l'achat

"productName" : le nom du produit acheté (utilisé dans certains constructeurs)

"clientName" : le nom du client qui a effectué l'achat (utilisé dans certains constructeurs)

"qte" : la quantité de produit acheté

"date" : la date de la transaction

La classe a également des méthodes d'accès pour chaque propriété et une méthode toString() pour afficher une représentation textuelle de l'objet Crédit.

## **Class CreditDAO :**

### **Enregistrer crédit :**

```
@Override
public void save(Credit credit) throws SQLException {
    String req = "INSERT INTO credit (id_produit, id_client) VALUES
    (?, ?, ?)";

    this.preparedStatement = this.connection.prepareStatement(req);

    this.preparedStatement.setLong(1, credit.getProduit_id());
    this.preparedStatement.setLong(2, credit.getClient_id());

    this.preparedStatement.execute();
}
```

le code est une méthode "save" qui insère une ligne dans une table "credit" en utilisant des paramètres d'objet "Credit". La requête SQL est préparée avec des espaces réservés pour les valeurs à insérer, puis les valeurs sont ajoutées aux positions correspondantes à l'aide de méthodes PreparedStatement. Finalement, la requête est exécutée à l'aide de la méthode execute(). En résumé, cette méthode permet de sauvegarder un objet Credit dans une table de base de données relationnelle.

### **Modifier crédit:**

```
@Override
public void update(Credit credit) throws SQLException {
    String req = "UPDATE credit SET id_produit = ?, id_client = ?, qte
    = ?, date = ? WHERE id_credit = ?";

    this.preparedStatement = this.connection.prepareStatement(req);

    this.preparedStatement.setLong(1, credit.getProduit().getId());
    this.preparedStatement.setLong(2,
    credit.getClient().getId_client());
```

```

        // this.preparedStatement.setInt(3, credit.getQte());
        //this.preparedStatement.setDate(4,
        java.sql.Date.valueOf(credit.getDate()));

        this.preparedStatement.executeUpdate();
    }

```

Ce code est une méthode "update" qui met à jour une ligne dans une table "credit" en utilisant des paramètres d'objet "Credit". La requête SQL est préparée avec des espaces réservés pour les valeurs à mettre à jour, puis les nouvelles valeurs sont ajoutées aux positions correspondantes à l'aide de méthodes PreparedStatement. Finalement, la requête est exécutée à l'aide de la méthode executeUpdate(). En résumé, cette méthode permet de mettre à jour un objet Credit dans une table de base de données relationnelle.

### Supprimer un crédit :

```

@Override
public void delete(Credit credit) throws SQLException {
    String req = "DELETE FROM credit WHERE id_credit= ?";

    this.preparedStatement = this.connection.prepareStatement(req);

    this.preparedStatement.setLong(1, credit.getCredit_id());

    this.preparedStatement.executeUpdate();
}

```

Ce code est une méthode "delete" qui supprime une ligne dans une table "credit" en utilisant un paramètre d'objet "Credit". La requête SQL est préparée avec un espace réservé pour la valeur à supprimer, puis la valeur est ajoutée à la position correspondante à l'aide de la méthode PreparedStatement. Finalement, la requête est exécutée à l'aide de la méthode executeUpdate(). En résumé, cette méthode permet de supprimer un objet Credit dans une table de base de données relationnelle.

### Recupere un crédit a partir de son id :

```

public Credit getOne(Long id) throws SQLException {
    String req = "SELECT credit.*, produit.nom AS produit_nom,
client.nom AS client_nom " +
                "FROM credit " +
                "INNER JOIN produit ON credit.id_produit =
produit.id_produit " +
                "INNER JOIN client ON credit.id_client = client.id_client
" +
                "WHERE credit.id_credit = ?";

    this.preparedStatement = this.connection.prepareStatement(req);

    this.preparedStatement.setLong(1, id);

    this.resultSet = this.preparedStatement.executeQuery();
}

```

```

        if (this.resultSet.next()) {
            ProduitDAO produitDAO = new ProduitDAO();
            ClientDAO clientDAO = new ClientDAO();

            Produit produit =
produitDAO.getOne(this.resultSet.getLong("id_produit"));
            Client client =
clientDAO.getOne(this.resultSet.getLong("id_client"));
            String produitNom = this.resultSet.getString("produit_nom");
            String clientNom = this.resultSet.getString("client_nom");

            Credit credit = new Credit(
                this.resultSet.getLong("id_credit"),
                produit,
                client,
                this.resultSet.getInt("qte"),
                this.resultSet.getDate("date").toLocalDate()
            );

            credit.getProduit().setNom(produitNom);
            credit.getClient().setNom(clientNom);

            return credit;
        } else {
            return null;
        }
    }
}

```

Ce code est une méthode "getOne" qui récupère une ligne dans une table "credit" en utilisant un paramètre "id". La requête SQL est préparée avec une jointure avec les tables "produit" et "client" pour récupérer les informations correspondantes à l'aide de leurs clés étrangères, puis la valeur est ajoutée à la position correspondante à l'aide de la méthode PreparedStatement. Finalement, la requête est exécutée à l'aide de la méthode executeQuery(). Si un résultat est renvoyé, les informations sont récupérées à l'aide de la méthode ResultSet, une instance de l'objet Credit est créée et les informations récupérées sont stockées dans cet objet. Enfin, l'objet Credit est retourné. Si aucun résultat n'est renvoyé, la méthode renvoie null. En résumé, cette méthode permet de récupérer un objet Credit avec toutes ses informations en fonction de son identifiant à partir d'une table de base de données relationnelle.

### Récupère tous les crédits avec le noms de client et de produits :

```

public List<CreditWithNames> getAllWithNames() throws SQLException {
    List<CreditWithNames> credits = new ArrayList<>();

    String req = "SELECT credit.id_credit, produit.nom, client.nom,
qte, date FROM credit " +
        "JOIN produit ON credit.id_produit = produit.id_produit "
+
        "JOIN client ON credit.id_client = client.id_client;";

    this.statement = this.connection.createStatement();
    this.resultSet = this.statement.executeQuery(req);

    while (this.resultSet.next()) {
        Long credit_id = this.resultSet.getLong(1);
    }
}

```



```

        String productName = this.resultSet.getString(2);
        String clientName = this.resultSet.getString(3);
        int qte = this.resultSet.getInt(4);
        java.sql.Date date = this.resultSet.getDate(5);

        credits.add(new CreditWithNames(credit_id, productName,
clientName, qte, date.toLocalDate()));
    }

    return credits;
}

```

Ce code est une méthode "getAllWithNames" qui récupère toutes les lignes de la table "credit" avec les noms correspondants des clients et des produits à partir des tables "client" et "produit" respectivement. La requête SQL est préparée avec une jointure avec les tables "produit" et "client" pour récupérer les informations correspondantes à l'aide de leurs clés étrangères. Ensuite, la requête est exécutée à l'aide de la méthode executeQuery(). Si un résultat est renvoyé, les informations sont récupérées à l'aide de la méthode ResultSet et stockées dans un objet CreditWithNames, puis ajoutées à une liste d'objets CreditWithNames. Enfin, la méthode renvoie la liste d'objets CreditWithNames. En résumé, cette méthode permet de récupérer une liste d'objets CreditWithNames avec toutes les informations de chaque ligne de la table "credit" et les noms correspondants des clients et des produits à partir des tables "client" et "produit" respectivement.

### **Class CreditsController :**

Le code présenté est un ensemble de classes Java pour gérer les crédits d'une entreprise. Il y a une classe pour les crédits (Credit), une classe DAO pour les crédits (CreditDAO), une classe DAO pour les clients (ClientDAO), une classe DAO pour les produits (ProduitDAO) et une classe de contrôleur pour la vue (CreditsController).

La classe CreditsController est un contrôleur de vue pour la fenêtre d'affichage des crédits. Elle contient des références aux éléments d'interface utilisateur tels que la table des crédits et les colonnes associées. Elle contient également des méthodes pour récupérer les noms des clients et des produits à partir de la base de données, mettre à jour la table des crédits et récupérer les données de crédit à afficher dans la table. La méthode initialize est appelée lors de l'initialisation de la vue et met à jour la table des crédits.

### **Le fichier Credits.fxml :**

Il s'agit également d'un fichier XML écrit en format FXML utilisé pour concevoir des interfaces utilisateur en JavaFX. Ce fichier contient une balise AnchorPane avec un TableView à l'intérieur, qui a quatre colonnes nommées "clientNameColumn", "productNameColumn", "creditAmountColumn" et "creditDateColumn". Les colonnes du TableView ont chacune un identifiant unique (fx:id), une largeur préférée et un titre

à afficher. En outre, le TableView est positionné à l'intérieur de l'AnchorPane avec une hauteur préférée et une largeur préférée. Le TableView est identifié par l'attribut `fx:id` "creditsTable", ce qui permet au programme JavaFX de le récupérer et de le manipuler. Enfin, la balise AnchorPane est associée à un contrôleur JavaFX (`fx:controller`) nommé "CreditsController".

## Conclusion

En conclusion, le projet Esalaf est une application de gestion de crédit qui utilise la technologie JavaFX pour créer une interface graphique conviviale. Le projet a été développé en suivant une méthodologie agile, avec une attention particulière portée à l'expérience utilisateur et à l'optimisation des performances.

Le projet comprend plusieurs fonctionnalités telles que la gestion des clients, des produits, des crédits et des paiements. Les données sont stockées dans une base de données MySQL, avec des classes DAO pour assurer une communication efficace entre l'application et la base de données.

Le développement du projet a permis aux développeurs de mettre en pratique leurs compétences en programmation Java, en développement d'interface utilisateur et en gestion de bases de données. De plus, le choix de la technologie JavaFX a permis d'obtenir une interface utilisateur moderne et dynamique, avec une grande flexibilité et une personnalisation facile.