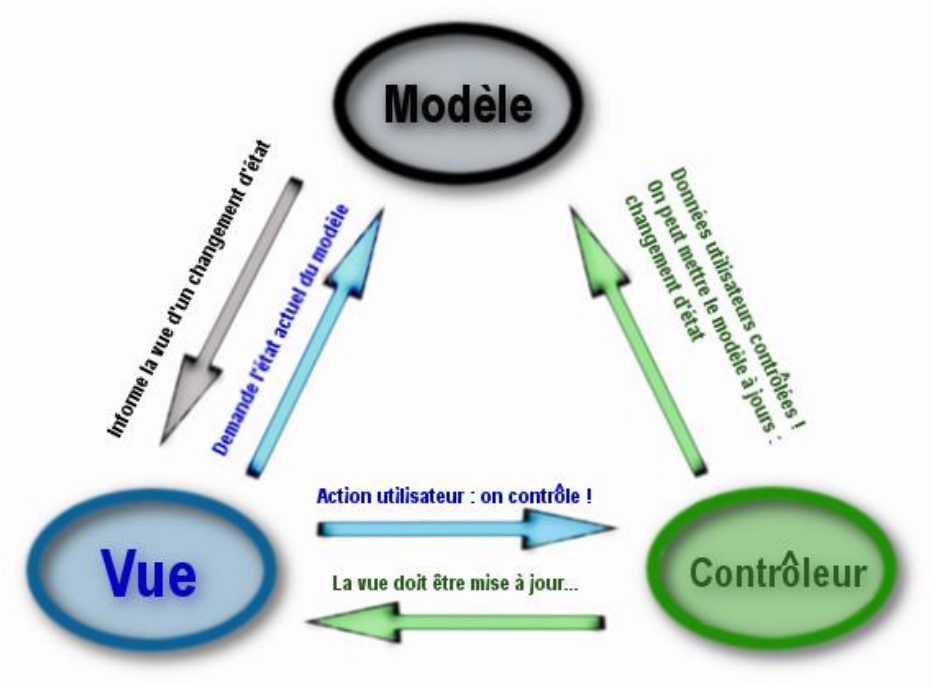
COLLÈGE MONTMORENCY

Travail pratique # 2

Présentation du modèle MVC
&
Suggestion des classes

A- Le modèle MVC ou modèle vue contrôleur

Architecture permettant de diviser le code de l'application en trois parties distinctes: modèles, vues et contrôleurs.



Openclassrooms

1. **Les modèles** : ils représentent les règles d'affaires qui régissent l'accès et la mise à jour des données. Ils représentent la couche métier (le domaine et la logique d'affaires) de l'application. Ils sont complètement indépendants de la couche présentation. Dans certains programmes très simples, le modèle contient lui même les données, mais souvent celles-ci sont stockées dans une base de données.
   - ✓ Exemple: les classes qui gèrent les tables d'une base de données.
2. **Les vues:** elles affichent les données provenant des modèles. Ce sont les interfaces utilisateurs (fenêtres, boutons, tables..). Les seuls traitements effectués par ces vues sont la gestion de la présentation et la détection d'actions de l'utilisateur (clic sur un bouton, déplacement de la souris …).
   - ✓ *Exemple*: une fenêtre qui affiche les enregistrements provenant d'une base de sonnées dans une table.

Idéalement, les modifications apportées sur les modèles n'auront aucune conséquence sur les vues et vis versa.

3. **Les contrôleurs:** ils font le lien entre les vues et les modèles.

   - Les contrôleurs traitent les évènements en provenance des vues et les transmettent au modèle pour le faire évoluer. Par exemple, au clic sur le bouton « Ajouer un employé », appeler une méthode pour l'ajout de l'employé dans le modèle.
   - Les contrôleurs peuvent aussi traiter les évènements en demandant à la vue de modifier son aspect visuel (changer la couleur d'un panneau, désactiver un bouton, ou ouvrir une autre vue…).
     - ✓ Exemples : les classes qui s'occupent des traitements d'un évènement particulier ou de plusieurs évènements en relation (menu ou groupe de boutons).

- L`application peut contenir d'autres classes **utilitaires.**

B- Suggestion des classes pour la réalisation de l'application du Tp#2

1. Les classes graphiques (vues) : différentes fenêtres demandées dans le document « Énoncé du TP2 »,
2. Les classes **Album** et **Artiste (**modèles**):** chaque classe doit avoir les mêmes attributs que la table qu'elle représente. Ajoutez les constructeurs et accesseurs nécessaires.
3. La classe **ControleConnexion** (utilitaire) : elle se charge d'établir la connexion à la BDD, de fermer la connexion et de retourner la connexion.
   - Voir l'exemple donné ci-dessous de la classe ControleConnexion qui établit une connexion à une Base de données Access des employés en utilisant le driver JDBC-ODBC bridge.
4. Les classes **GestionAlbums** et **GestionArtistes (**modèles**):** elles comportent toutes les méthodes de consultation et de mise à jour de la base de données.
   - Voir le code ici-bas de la classe GestionEmploye qui comporte quelques méthodes de gestion des employés d'une BDD Access.
5. Les classes de modèles de données des tables (modèles).
6. Les classes Renderer (vues) pour le rendu des tables.
7. Les classes de gestions d'évènements des utilisateurs (contrôleur).

1. Classe qui établit la connexion/déconnexion à une BD Access via le driver JDBC-ODBC bridge.

```
public class ControleConnexion {
/**
 * L'intérêt d'une connexion statique est de disposer d'une connexion unique durant toute
une session.
*/
private static Connection laConnexion;
private static String  url = "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb,
*.accdb)};DBQ=c:/bd/employes.accdb";

/**
* Établit la connexion à la BDD si elle n'existe pas. Attention, la connexion ne doit pas
être fermée
 */
public static void connecter() {
      try{
            if(laConnexion == null || laConnexion.isClosed()){
                  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                  laConnexion= DriverManager.getConnection(url);
                  JOptionPane.showMessageDialog(null, "Connect\u00E9 à laBD", "ALERTE",
                     JOptionPane.INFORMATION_MESSAGE);
            }
      }
      catch(ClassNotFoundException e){
          …
      }
      catch(SQLException sqle){
         …
      }
}


public static void fermerSession(){
      try{
            if (laConnexion!=null  && !laConnexion.isClosed())
            …     //fermer la connexion
      }
      catch(SQLException sqle){
            …
      }
}
public static Connection getLaConnexion() {
      …
}

}
```

Note: pour se connecter à une base de données, pas besoin de créer un objet de la classe ControleConnexion, il suffit d'appeler sa méthode statique:

```
ControleConnexion.connecter();
```

2. Classe de gestion des employés: Ajout, modification, suppression, obtention de liste des employés…

```
public class GestionEmploye {

private  Connection connection = ControleConnexion.getLaConnexion();
/**
 * Structure de données pour chercher la liste des employés de la BD
*/
private  ArrayList<Employe>  listeEmployes;

public GestionEmploye(){
      listeEmployes =obtenirListeEmployes() ;
}
public  ArrayList<Employe> getListeEmployes() {
      return listeEmployes  ;
}
private  ArrayList <Employe> obtenirListeEmployes(){
      ArrayList<Employe> liste= new ArrayList <Employe>();
      String requete= "SELECT * FROM Employes  ORDER BY nom";
      try(Statement statement= connection.createStatement();
            ResultSet jeuResultat= statement.executeQuery(requete)){
            while(jeuResultat.next()){
               String code = jeuResultat.getString("code");
               String nom = jeuResultat.getString("nom");
               String profession = jeuResultat.getString("profession");
               double salaire = jeuResultat.getDouble("salaire");
               boolean syndique = jeuResultat.getBoolean("syndique");
               liste.add(new Employe(code, nom, salaire,profession, syndique));
            }
      }
      catch (SQLException sqle) {
            JOptionPane.showMessageDialog(null,
             "Problème rencontr\u00E8 : " + sqle.getMessage(),
            "Résultat", JOptionPane.ERROR_MESSAGE);
      }
      return liste;
}
//La méthode d'ajout d'un employé retourne vrai si l'ajout dans la BD a réusi, faux si
non.
public boolean ajouterEmployeBD(Employe employe) {
      int intSyndique=0;
      boolean boolAjout =false;
      if (employe.isSyndique()){
            intSyndique=1;
      }
      String requete =
      "INSERT INTO Employes(code, nom, salaire, profession,syndique) VALUES('"
                        + employe.getCode()                  + "','"
                        + employe.getNom()                   + "',"
                        + employe.getSalaire()               + ",'"
                        + employe.getProfession()            + "',"

                        + intSyndique
                        + ")";
```

```
	try{
		Statement statement = connection.createStatement();
		statement.executeUpdate(requete);
		boolAjout=true; //L'ajout réussi

	}
	catch (SQLException sqle){
		  JOptionPane.showMessageDialog(null,
		  "Probl\u00E8me rencontr\u00E9 lors de l'enregistrement de l'employ\u00E9: "
		  + sqle.getMessage(),"Résultat", JOptionPane.ERROR_MESSAGE);
	}

	return boolAjout;
	}
	public boolean supprimerrEmployeBD(Employe employe) {
		…

	}
}
```

3. Pour ajouter un employé dans la BD à partir d'une interface graphique. On ajoute d'abord l'employé dans la BD. Et, si l'ajout est réussi, on l'ajoute en dernier dans le modèle de table.
   Exemple: ajouter un objet employé en cliquant sur le bouton "Confirmer".

```
public void actionPerformed(ActionEvent e){
	if (e.getSource()==btnConfirmer ){
	 try{
		String numero= txtNumero.getText();
		String nom= txtNom.getText();
		double salaire = Double.parseDouble(txtSalaire.getText());
		String profession= txtProfession.getText();
		boolean syndique = checkBoxSyndique.isSelected();
		Employe employe= new Employe(numero, nom, salaire, profession, syndique );

		//Ajouter l'employé à la BD puis au modèle
		GestionEmploye gestionnaire = new GestionEmploye();
		if (gestionnaire.ajouterEmployeBD(employe)){
			modeleEmploye.ajouterEploye(employe);
			viderChamps();
		}
	}
		catch (NumberFormatException  nfe){
			JOptionPane.showMessageDialog(this, "Le salaire n'est pas un nombre");
	}

	}
```