TD3 : Manipulation des bases de données avec java

On veut développer une petite application qui permet de représenter le jeu « Le Pendu ».

La règle du jeu est simple :

- Un joueur principal propose un mot dans son dictionnaire au hasard et l'affiche mais en remplaçant les lettres qui le composent par des étoiles '*'. L'intérêt est que l'on connaisse le nombre de lettres et que le mot soit gardé **secret** !
  Les autres joueurs de leur mieux essayent de trouver le mot secret en proposant des lettres qui pourraient y figurer. Une et une seule lettre est autorisée à la fois par joueur pour chaque essai!
- **Si une lettre est correcte**
  - donc, une lettre correspond à une ou plusieurs lettres du mot secret, on l'affiche à la place des '*' ou des '*' qui la remplaçaient.
    - Imaginons pour cela que le mot secret est "canard", au départ, on affiche cela : * * * * * *
    - Si un joueur propose '**a**', on affiche : * **a** * **a** * *
- **Si une lettre est fausse :**
  - on ne fait rien mais en compte qu'un essai est effectué.
- Si, le nombre d'essais dépasse dix sans trouver le mot secret alors les joueurs perdent le jeu. Si non ils ont trouvé le mot secret donc ils ont gagné.

Pour notre application, Le joueur principal est la machine et les autres joueurs c'est l'utilisateur de l'application. Afin de modéliser correctement notre jeu, on a besoin de construire plusieurs classes qui vont construire le squelette de notre application.

La première étape

Elle consiste à construire une fenêtre d'authentification qui est représentée par la classe ***JDialogueIdentification*** et qui permet d'identifier un joueur membre ou d'ajouter un nouveau joueur qui n'existe pas dans la base de données de notre application. Cette classe utilise d'autres classes qui sont : ***Joueur, ConnectionBase*** et ***RequetesBaseDeDonnees***.

→La classe ***Joueur*** modélise les informations « identifiant, nom et score » et les méthodes associées à un joueur.
→La classe ***ConnectionBase*** cette classe permet de créer un singleton afin de gérer la communication avec la base avec un seul objet de type Connection.
→La classe ***RequetesBaseDeDonnees*** est la classe qui offre l'ensemble des requêtes à utiliser afin de communiquer avec la base de données et de manipuler les données de tous les joueurs de notre application.

- Cette classe contient l'ensemble des méthodes qui gèrent l'ensemble de requêtes qu'il faut lancer pour manipuler la base de données "JeuPendu_bd" en utilisant un singleton instance de la classe ConnectionBase.

Les codes des classes *Joueur* et *ConnectionBase* « qui permet de définir un singleton de connexion à la base de donnée dont le nom est **JeuPendu_bd** » sont données.

Le squelette de la classe *JDialogueIdentification* est donné par

public class **RequetesBaseDeDonnees** {

private Connection connection = ConnectionBase.getConnection( );// singleton de connexion	
private static RequetesBaseDeDonnees requete_bd;	Singleton qui sera utilisé pour exécuter toutes les méthodes de cette classe dans les autres classes.
private RequetesBaseDeDonnees()	Constructeur de la classe qui est vide.
public static RequetesBaseDeDonnees getRequetesBaseDeDonnees( )	Méthode permettant de retourner une instance de la classe RequetesBaseDeDonnees qui est l'unique instance de la classe afin d'exécuter l'ensemble des requêtes. Méthode donnée
public void createBaseDeDonnees()	Méthode permettant de construire la base de données **JeuPendu_bd** avec un driver de type MySQL. Méthode donnée
public void creerTable()	Méthode permettant de créer la table **Joueur** dans la base de données **JeuPendu_bd**. Méthode donnée
public Joueur rechercherJoueurViaId(int Id)	Méthode permettant de rechercher un joueur via son identifiant. S'il existe, on retourne l'objet joueur correspondant, si non cette méthode retourne l'objet null.
public Joueur insererNouveauJoueur(String nom, int score)	Méthode permettant d'insérer un nouveau joueur dans la table Joueur avec une clé auto-incrémente. Cette méthode retourne l'objet joueur nouvellement crée si non s'il y a un problème cette méthode retourne un objet null.
public void updateScore(int Id, int score)	Méthode permettant de faire une mise à jour de la valeur du score d'un joueur dans la table Joueur. Le joueur est identifié via son Id.
public int valeurDuDernierIdentifiant()	Méthode permettant de retourner la valeur du dernier identifiant enregistré dans la table Joueur. S'il y a un problème alors la méthode retourne 0.
public void afficherInfosJoueur(int Id)	Méthode permettant d'afficher les infos d'un joueur via son identifiant s'il existe dans la table Joueur.

}

public class **JDialogueIdentification** extends JDialog implements ActionListener {

JLabel nom_Joueur, id_Joueur;	Le premier label représente le nom du nouveau joueur. Le deuxième label représente l'identifiant du joueur membre.
JTextField nomJoueur_Field, id_Joueur_Field;	→ Le premier champText permet d'entrer le nom du nouveau joueur dont le score est nul et dont l'identifiant est récupéré dynamiquement par auto incrémentation de la clé depuis la base de données. → Le deuxième champText permet d'identifier un membre existence via le numéro saisi dont ce champ en utilisant la base de données.
JButton validerNouveau, validerMembre, quitter;	Les deux premiers boutons permettent de valider soit l'ajout d'un nouveau joueur, soit d'identifier un membre déjà inscrit. → Lors du click sur le bouton « **validerNouveau** » Les informations du nouveau joueur sont stockées dans la base de données à condition de la validité du nom saisi dans le champText nomJoueur_Field. → Lors du click sur le bouton « **validerMembre** », on récupère le numéro saisie dans le champ id_Joueur_Field, puis on teste sa validité puis on teste son existence ou non dans la base de données. Si le numéro n'existe pas dans la base de données ou s'il y a un problème de validité alors on affiche un message d'avertissement. -) S'il n'y a pas de problème, lors du click sur l'un des deux premiers boutons, alors on affiche la fenêtre principale dont le nom est **JeuPenduSimple** et dont les informations en haut à droite représentent le nom de l'utilisateur courant et son score actuel. →Le bouton « quitter » permet de fermer l'application complètement
Box BoxNouveauJoueur, BoxJoueurMembre;	Les boxs permettent de définir chaque zone de chaque type de joueur, soit membre soit

	nouveau.
public JDialogueIdentification( )	Le constructeur de la classe permettant de créer l'interface associée à cette classe.
public void actionPerformed(ActionEvent e)	Méthode permettant la gestion des cliques sur les boutons et leurs actions sur la valeur entrée dans les JTextField afin d'insérer un nouveau joueur ou d'identifier un joueur membre et de commencer le jeu ou simplement de quitter l'application.
public boolean ValeurEntierValide(String val, JTextField champ)	Cette méthode permet de vérifier si un string représente bien un entier ou pas. Si c'est non, elle affiche un message via un JOptionPane relatif au texteField qui contient le string à tester. **Le code de cette méthode est donné**.

}

La deuxième étape

Elle consiste à la création de l'interface principale qui permet de faire jouer l'utilisateur de l'application après avoir était identifié lors de la première étape.

Cette étape est modélisée par la classe **JeuPenduSimple**. Cette classe permet de simuler le jeu « Le Pendu » simplifié. Le squelette de cette classe est donné par :

public class **JeuPenduSimple** extends JFrame {

public Joueur joueurCourant;	L'utilisateur courant de l'application
int compteurEssai;	Permet de compter le nombre d'essai avant de trouver le motCaché ou de perdre la partie, le nombre d'essai max est 10
public String motA_trouver;	Représente le mot caché qui est récupéré aléatoirement depuis la liste des mots du **dictionnaire** (fichier texte donné dans le répertoire des images).
public static String Dictionnaire = "DocumentPourExam/dictionnaire.txt";	Défini l'emplacement du fichier dictionnaire qui contient l'ensemble des mots à utiliser dans ce jeu.
public ArrayList<String> listDesMotsDuDictionnaire;	Liste complète des mots du dictionnaire
public JLabel nomJoueur, scoreJoueur, motCache, entrerUneNouvelleLettre;	- Le premier et le deuxième label donnent le nom et le score du joueur courant. - Le troisième label est associé au champText contenant le mot caché. - Le quatrième label est associé au champText contenant la zone de saisie d'une nouvelle lettre qu'on doit tester si elle figure dans le mot caché ou non.
public JTextField champFieldMotCache, ChampFieldLettre;	- Le premier champ contient le mot caché, - Le deuxième est la zone de saisie de la nouvelle lettre qu'on doit tester si elle figure dans le mot caché ou non. - Les deux champs sont désactivés si on n'a pas commencé le jeu, c-a-d si on n'a pas encore cliqué sur le bouton « start »
public JButton start, rejouer, jouer_Essai;	- Le bouton « **start** » permet de commencer le jeu. Lorsqu'on clique sur ce bouton, il devient désactivé puis le champ ChampFieldLettre est activé. De plus, le champ champFieldMotCache contient le mot caché sou format d'un ensemble d'étoiles dans la longueur est égale à la longueur du vrai mot caché et enfin le bouton jouer_Essai est activé en présentant un numéro d'essai.

	- Le bouton « **rejouer** » permet de recommencer le jeu avec l'état initial de la fenêtre principale, c-a-d : le champFieldMotCache contient la phrase « *Je suis le mot caché.* », le bouton « start » est activé, le champ ChampFieldLettre est vide et désactivé puis le bouton jouer_Essai est désactivé avec un titre égal « Jouer_Essai_1 » - Le bouton « **jouer_Essai** » permet de faire des essais lors de saisie des lettre dans ChampFieldLettre puis si la nouvelle lettre se trouvant dans ChampFieldLettre est présente dans le mot caché alors on la place dans toute les position où elle se trouve dans le mot caché et en affiche cela correctement dans champFieldMotCache (voir début énoncés). →Si on arrive à trouver le mot caché, avant 10 essais, on affiche un message via un JOptionPane affirmant que le joueur à gagné puis en rafraichi la valeur du score du joueur. →Si non, si le joueur à fait les 10 essais alors il a perdu, on affiche un message qu'il a perdu puis on rafraichi le score du joueur → On rafraichi le score du joueur en utilisant les règles de comptage des points qui sont données en bas, ceci après le click sur le bouton « ok » du JOptionPane.
Final[ ] int tabRegleNotes = {100, 50, 35, 25, 15, 10, 5};	Tableau représentant les points de comptage en fonction du nombre d'erreurs. Le nombre d'erreurs est égal au nombre d'étoile restant, dans le champ texte **champFieldMotCache** , lors de la fin du jeu
public JeuPenduSimple(Joueur joueurCourant)	Constructeur de la classe permettant de construire l'interface de la fenêtre principale et de gérer les événements du click sur les boutons.
private ArrayList<String> enregistreDictionnaireDansUnArrayList()	Méthode permettant d'initialiser le champ listDesMotsDuDictionnaire, en faisant la lecture ligne par ligne du fichier dictionnaire. **le code de cette Méthode est donné**.

}

Règle de comptage des points :

Mot trouvé sans erreur : ….. 100 pts
Mot trouvé avec 1 erreur : ….. 50 pts
Mot trouvé avec 2 erreurs : ….. 35 pts
Mot trouvé avec 3 erreurs : ….. 25 pts
Mot trouvé avec 4 erreurs : ….. 15 pts
Mot trouvé avec 5 erreurs : ….. 10 pts
Mot trouvé avec 6 erreurs : ….. 5 pts
Mot trouvé avec plus de 6 erreurs : ….. 0 pts

Les interfaces associées aux deux classes **JDialogueIdentification** et **JeuPenduSimple** sont donnée par :

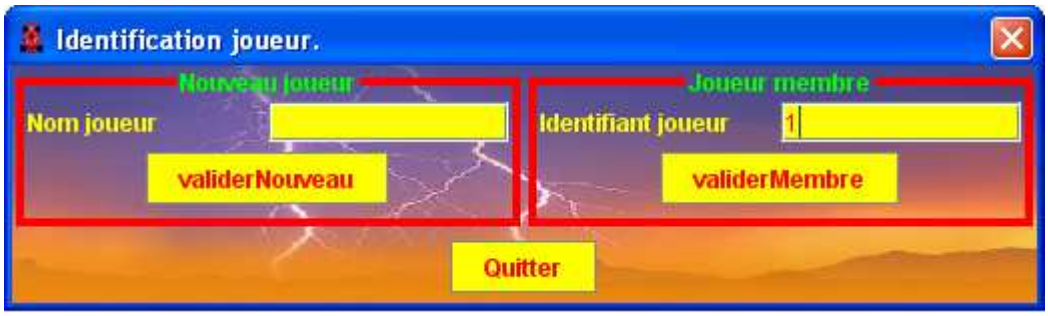

Figure 1 : Interface d'authentification d'un joueur membre.

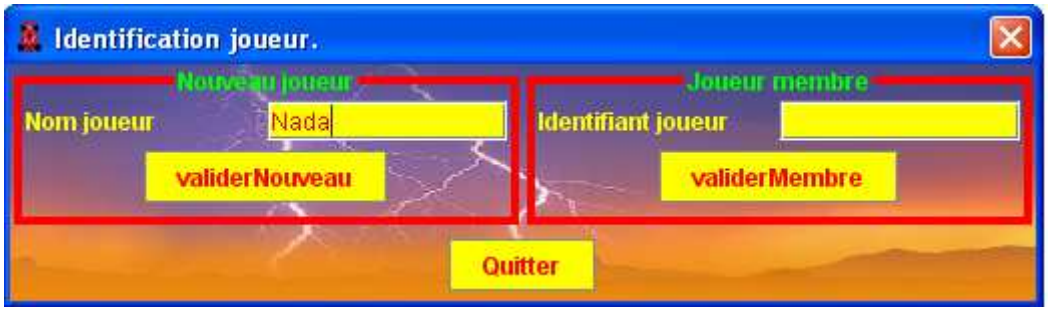

Figure 2 : Interface d'authentification pour l'ajout d'un nouveau joueur.

Figure 3 : Première interface du jeu « Le Pendu » après validation d'authentification du joueur ***Ahmed*** avec un score actuel égal à ***350***

Figure 4 : Interface obtenue après click sur le bouton « start » et la saisie de la lettre '**a**' dans le champ « nouvelle lettre » avant le lancement de l'essai 1

Figure 5 : Interface obtenue après lancement de l'essai 1

Figure 6: Interface obtenu après le dépassement de 10 essais sans gagner la partie.
en cliquant sur ok, on revient à l'interface de début donnée comme la figure 1.

Figure 7 : Interface obtenue après clique sur le bouton « ok », l'utilisateur Ahmed à pour score 360 et non pas 350 du début du jeu