
M2105: INTERFACES HOMME-MACHINE

Feuille de TP n°3

Modèle-Vue-Contrôleur

Cette feuille a pour objectif d'appréhender les concepts vus en cours sur Modèle-Vue-Contrôleur.

Exercice 1 *La liste des meilleurs scores*

On vous demande de créer une application qui permet de gérer la liste des meilleurs scores d'un jeu. La vue de cette application est représentée ci-dessous.

Tableau des scores	
1. Batman	40452
2. Robin	39521
3. Batman	38412
4. Superman	37541

Une fois lancée l'application s'affiche avec une liste de meilleurs scores vide. L'utilisateur entre le nom du joueur et le score obtenu. Lorsqu'il valide en cliquant sur le bouton, le nouveau score doit s'insérer dans le tableau des scores et les deux champs de saisie sont vidés. On note que le tableau des scores a un nombre de places limité (5 par exemple) et donc lorsque le tableau a atteint sa limite :

- si un score entré par l'utilisateur est moins bon que tous les scores du tableau, celui-ci n'est pas ajouté
- si un score entré par l'utilisateur est meilleur que l'un des scores du tableau, le score est ajouté à la bonne place et le dernier score est éjecté.

On vous demande d'implémenter cette application en respectant les concepts du Modèle-Vue-Contrôleur.

A. Le modèle

Vous devez implémenter en Java une classe qui gère la liste des meilleurs scores. Cette classe devra avoir

1. un constructeur qui permet de créer une liste de meilleurs scores vide ayant une taille maximum passée en paramètre du constructeur.
2. Une méthode `getNbPlaces()` qui retourne le nombre de places disponibles dans le tableau des scores.
3. Une méthode `getNbScores()` qui retourne le nombre de scores actuellement contenus dans la liste.
4. Une méthode `getScore(int place)` qui retourne sous la forme d'un chaîne de caractères ("Robin 39521") le score à la place indiquée.

5. Une méthode `ajouterScore(String nom, int nbPoints)` qui permet d'ajouter un nouveau score dans la liste au bon endroit. Cette méthode doit aussi gérer l'expulsion d'un score de la liste lorsque la limite de place est atteinte. Cette méthode retourne un booléen indiquant si le score a réellement été ajouté.

Notez que les méthodes `getXXX` sont des observateurs qui seront les seuls utilisables par la vue.

B. La Vue

Elle est représentée par la maquette ci-dessus. Cette vue devra fournir les méthodes : `getNomJoueur()`, `getNbPoints()` et `getTableau()` qui permettront au contrôleur de manipuler ces objet graphique. Notez que la vue devra créer le modèle dans la phase d'initialisation de l'application.

C. Le Contrôleur

Il s'agira ici du gestionnaire d'événements action du bouton *Valider*. Ce contrôleur devra avoir accès à la vue et au modèle (donc son constructeur prendra en paramètre une vue et un modèle). Lorsque l'utilisateur clique sur le bouton, le contrôleur devra

1. Vérifier que la zone de texte contenant les points est bien un nombre entier. Si ce n'est pas le cas un message est affiché et la saisie n'est pas validée.
2. Ajouter le nouveau score au modèle.
3. Vider les deux zones de texte du nom et du nombre de points.
4. Mettre à jour l'affichage de la liste des scores de la vue.

Notez que si un score n'a pas été ajouté car il n'était pas assez bon pour entrer dans la liste, un message d'information est affiché à l'utilisateur.

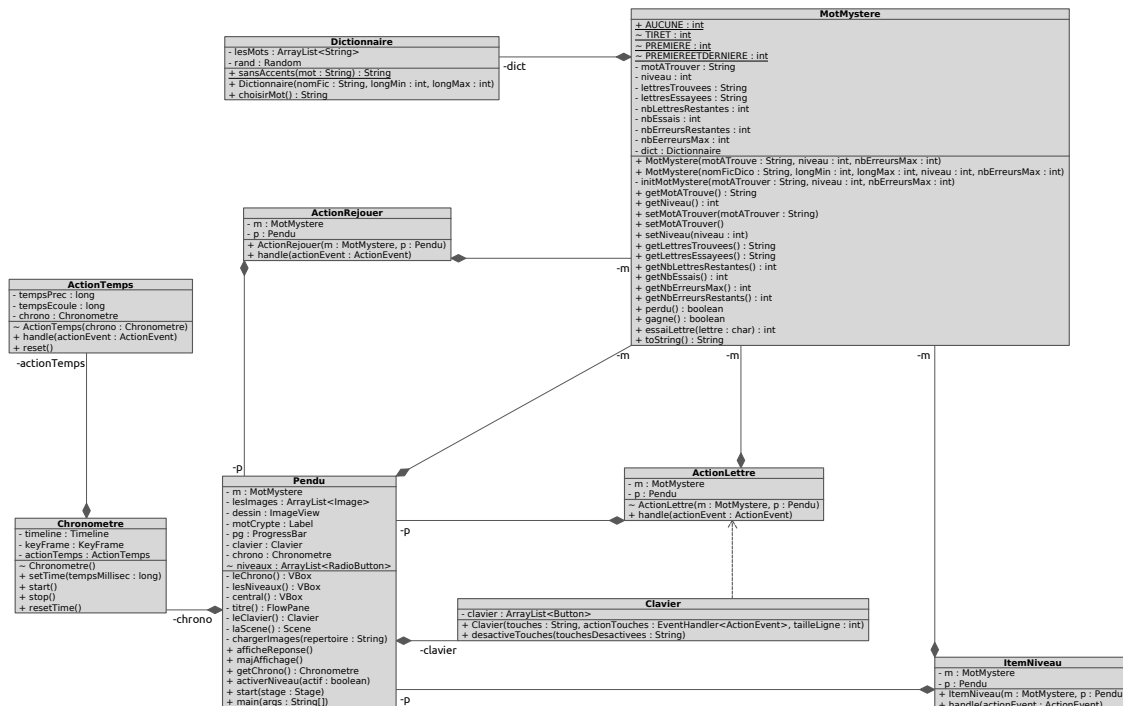
Exercice 2 Le jeu du Pendu

L'objectif est de créer une application pour jouer au jeu du pendu où il s'agit de retrouver un mot dont les lettres sont cachées. À chaque tour de jeu l'utilisateur propose une lettre si elle se trouve dans le mot, toutes les occurrences de la lettre dans le mot sont dévoilées sinon un élément supplémentaire du pendu est dessiné. Si le pendu est entièrement dessiné le joueur a perdu la partie.

L'archive `pendu.zip` contient les fichiers utiles à ce TP.

- Le répertoire `src` contient les fichiers JAVA du projet. Dans ce projet vous aurez à compléter certaines classes d'autres vous seront données. Les méthodes à implémenter sont indiquées dans les commentaires.
- Le répertoire `doc` contient la documentation JAVA (Javadoc) du projet. Vous pouvez la consulter via un navigateur web. Pour tout ce que vous aurez à faire, veuillez vous référer à cette documentation (ou aux commentaires dans les fichiers sources Java).
- Le répertoire `img` contient les images qui seront affichées pour le jeu du pendu.

Le diagramme de classes de l'application est le suivant



Dans cette application

- le *modèle* est constitué des deux classes `MotMystere` et `Dictionnaire`,
- la *vue* est constituée de trois classes `Pendu` qui est la classe principale ainsi que les classes `Clavier` qui gère le clavier de lettres et `Chronometre` qui gère l'affichage du chronomètre de la partie
- le *contrôleur* est constitué des quatre classes `ActionRejouer`, `ItemNiveau`, `ActionLettre` (associée au clavier) et `ActionTemps` (associée au chronomètre)

A. Le modèle

C'est lui qui va gérer la représentation mémoire du jeu du pendu et de l'évolution de la partie. Vous n'avez rien à implémenter dans le modèle.

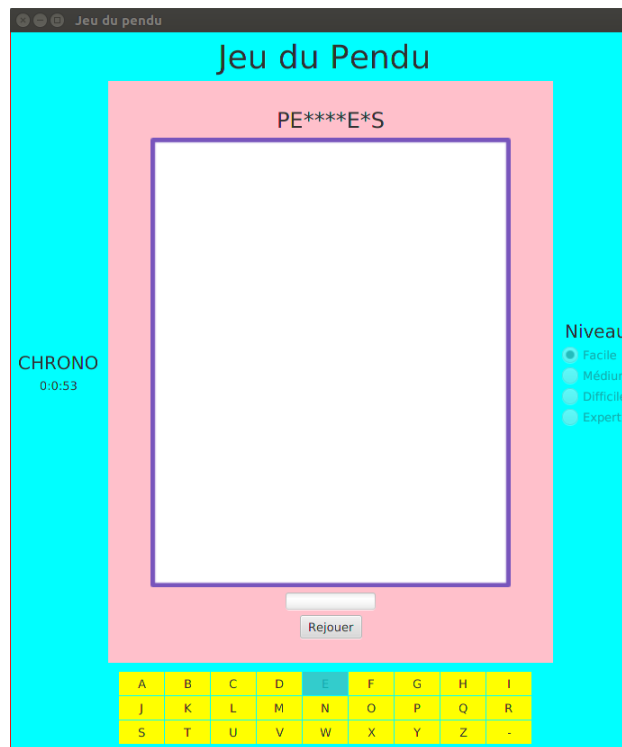
- `Dictionnaire` sert simplement à charger une liste de mots à partir d'un fichier texte.
- `MotMystere` permet de gérer une partie de pendu. Cette classe permet notamment de choisir un mot au hasard dans une liste de mots, d'essayer des lettres et de vérifier si l'utilisateur a gagné ou perdu. Les méthodes importantes dans cette classe :

- `getMotATrouver()` qui retourne le mot à trouver.
 - `getLettresTrouvees()` qui retourne que des `*` sauf aux emplacements des lettres déjà trouvées par l'utilisateur.
 - `essaiLettre(char lettre)` qui correspond à un tour du jeu du pendu
- Cette classe gère 4 niveaux de jeu qui diffèrent uniquement au démarrage du jeu en découvrant automatiquement le tiret s'il y en a un, la première lettre ou la première et la dernière du mot.

Vous devrez implémenter votre jeu en vous servant uniquement des méthodes de cette classe sans les modifier.

B. La vue

L'image ci-dessous représente la vue du jeu du pendu.



Le container principal de cette vue est un `BorderPane` avec en haut un titre, à droite le chronomètre, à gauche les radio-boutons contrôlant le niveau en bas le clavier et au milieu, le mot mystère avec les lettres déjà trouvées, le dessin du pendu, une barre de progression indiquant où en est la partie et le bouton rejouer.

Par ailleurs la vue nécessite l'existence des 11 fichiers images représentant les 11 états du pendu. Ces fichiers se trouvent dans le répertoire `img` et doivent porter le nom `pendu xx .png` où xx est le numéro de l'état du pendu.

Dans cette classe, vous aurez à construire les différents panels. Vous aurez aussi à implémenter les méthodes `majAffichage()`, `afficheReponse()` et `activerNiveau(active)`. En effet on ne pourra changer de niveau que si la partie n'est pas commencée et donc il faut pouvoir activer ou non les radio-boutons.

Vous pouvez noter que le clavier est géré par une classe particulière `Clavier` qui doit gérer la création de la vue d'un clavier sous la forme d'un `GridPane`. Les actions à faire lors du clic sur une touche seront implémentées dans le contrôleur `ActionLettre`.

De même le chronomètre est géré par la classe `Chronometre` qui hérite d'un `Label` est gère l'affichage d'un chronomètre. Cette classe gère la mise à jour de chronomètre, son démarrage, son arrêt ainsi que sa remise à zéro en utilisant une `Timeline`. Le contrôleur qui gère les événements générés à chaque fin de `KeyFrame` est de la classe `ActionTps`.

C. Le contrôleur

Il s'agira d'implémenter les contrôleurs suivants :

- **ActionLettre** : il gère les événements action sur les touches du clavier. Lorsqu'une touche est cliquée, la lettre est jouée, la touche correspondante est désactivée et la vue est mise à jour. Évidemment le joueur doit être informé s'il a gagné ou perdu.
- **ActionRejouer** : il gère les événements action sur le bouton *Rejouer*. Si une partie est en cours, il faut demander si le joueur souhaite abandonner la partie avant de réinitialiser la partie.
- **ItemNiveau** : il gère les événements action sur les radio boutons gérant le niveau. Ces boutons ne doivent être actifs que si il n'y a pas de partie en cours (c-à-d qu'aucune lettre n'a été tentée). Le changement de niveau réinitialise la partie avec un nouveau mot pour tenir compte des nouvelles modalités liées au niveau.
- **ActionTps** : il gère les événements générés par le chronomètre permettant d'afficher le temps écoulé depuis le début de la partie.