Projet de TP

NB : Le zip du projet doit être envoyé, au plus tard, le samedi 18 avril par mail à l'adresse poo.usthb@gmail.com. Il doit contenir un import du projet Eclipse Eclipse (code source), ainsi qu'un rapport de projet expliquant la conception suivie. Le code doit être lisible et bien commenté. Le projet doit être nommé avec les noms des 2 binômes : GestionEUREKA_nombinôme1_nombinôme2

Enoncé

On s'intéresse à la réalisation d'un jeu de culture générale sous-forme du jeu Pendu revisité, baptisé **EUREKA**. L'idée est de poser une question à l'utilisateur, puis, à lui de trouver la réponse et proposer la réponse en tapant des caractères dans une zone de saisie. Si un des caractères est faux, une partie d'un dessin de pendu est au fur et à mesure dessiné. En arrivant à 8 échecs (caractères saisis ne faisant pas partie de la réponse), le dessin du pendu est complété, puis la partie de jeu est terminée avec un échec.

L'application devra donc :

1. Permettre à un utilisateur de s'inscrire en donnant un nom d'utilisateur et mot de passe, puis d'accéder à une partie de jeu une fois son mot de passe reconnu.
2. Proposer à l'utilisateur un ensemble de thèmes, chaque thème sera constitué d'une suite de questions/images.
3. Une fois le thème choisi, une question ou une image sera affichée à l'utilisateur, ainsi que deux zones de saisie (**JTextField**) : une première zone utilisée pour la saisie de l'utilisateur proposant la solution en tapant un caractère, et une seconde zone initialisée avec des étoiles (*) pour contenir les caractères trouvés par l'utilisateur au fur et à mesure qu'ils sont tapés durant le jeu. Une dernière zone est utilisée pour afficher une partie du dessin du pendu et sa potence.
4. Lorsque l'utilisateur tape un caractère sur la première zone, il est affiché à sa (ses) position(s) dans la seconde zone s'il appartient au mot de la réponse. Sinon, une partie du dessin du pendu et sa potence est affiché dans la dernière zone, et le nombre de tentatives restant est affiché à l'utilisateur.
5. Il doit être possible de connaître son score, et d'afficher les règles du jeu via un bouton **Help (?).** Un bouton **A propos** doit également afficher la version de votre logiciel, ainsi que l'auteur et la date de développement.

Questions

I. Pour réaliser ce projet, il est demandé de définir les classes suivantes avec constructeur dans un package **com.usthb.modeles** :

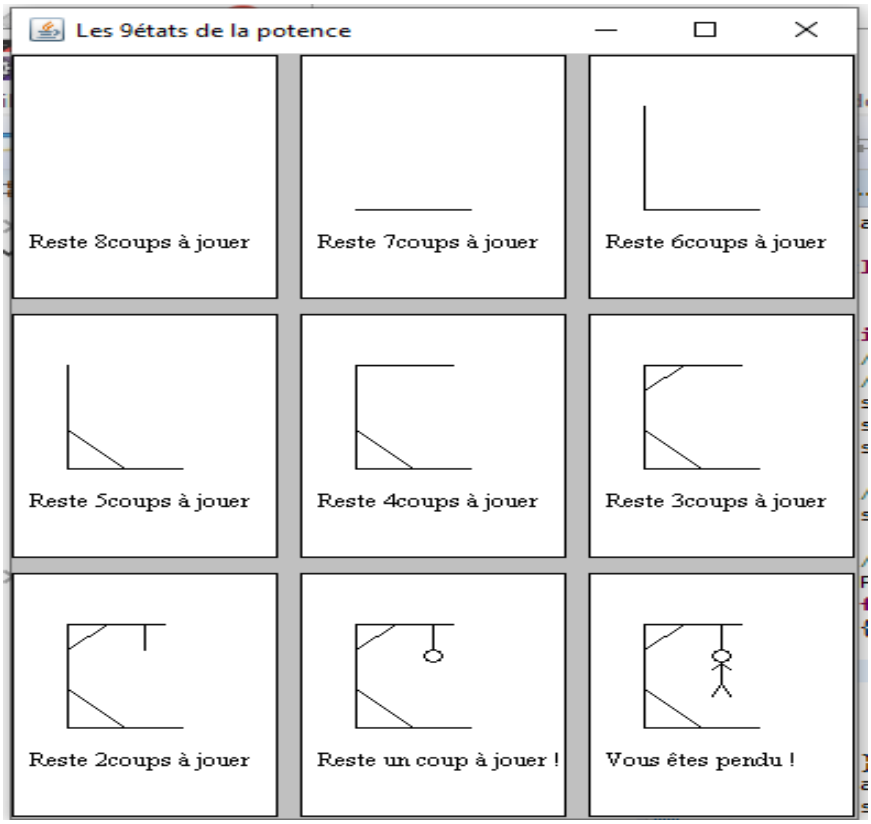
1. La classe **Joueur** caractérisée par un numéro séquentiel automatique, un nom, un prénom, une date de naissance, un mot de passe et un attribut dernier niveau atteint.
2. La classe **ThèmeJeu** caractérisée par un type (énumération : HISTOIRE, GEOGRAPHIE, SANTE, CULTUREGENERALE, ISLAM), un libellé, un coefficient et une liste de **questions** (de type **LinkedList**).
3. La classe **Question** caractérisée par un libellé, un numéro composé 3 caractères (soit HIS / GEO/ SAN/ CUL/ ISL selon le type du thème associé) suivi par un numéro séquentiel automatique, une image (nom du fichier image), une réponse (String) et un **niveau** (de 1 à 5). Une méthode **getNBPoints**() permet de retourner le nombre de points associés à la question comme suit : 5 points pour le niveau 1, 10 pour le niveau 2, 18 pour le niveau 3, 28 pour le niveau 4 et 40 pour le niveau 5.
4. La classe **PartieJeu** caractérisée par un numéro, le thème sélectionné, le numéro de question, un attribut réponse en cours (de type StringBuffer) désignant la réponse saisie par le joueur (1 seul caractère saisi) et un score des points cumulés. La réponse est initialisée, à l'instanciation d'une partie, par un nombre de caractères * égal à la taille de la réponse associée à la question.

Ajoutez à la classe **Joueur** la liste des parties de jeu réalisées, ainsi qu'une méthode **getTotalScore**() qui retourne le score obtenu depuis l'inscription du joueur.

Ajoutez aux classes les getters/setters et méthodes que vous jugez nécessaires.

II. Dans un package **com.usthb.dessin**, créer une classe **Potence,** créée à partir de la classe **Component** (la définir comme suit **public class** Potence **extends** Component). Elle est caractérisée par un état (int) dont les valeurs vont de 0 à 8 désignant l'état courant de la potence dessinée pour une partie de jeu, un booléen trouvé qui sera à vrai si la réponse à la question en cours est trouvée, et un attribut dimension de type Dimension (java.awt.Dimension). Lui ajouter une méthode **incrémentEtat**() qui incrémente l'état courant et une méthode **paint(Graphics g)** qui permet de dessiner la potence (voir le code donné en annexe).

   Ajouter à la classe **PartieJeu** une méthode **checkCaractère**(char c) qui vérifie si le caractère c appartient aux caractères de la réponse de la question en cours, puis, si le caractère est correct, remplacer le caractère * de l'attribut réponse en cours par le caractère c aux positions trouvées, et vérifier si l'attribut réponse en cours est égal à la réponse. Dans ce dernier cas, l'attribut trouvé de la potence en cours est mis à Vrai et le joueur est considéré comme gagnant de la partie jouée. Dans le cas où le caractère c n'est pas correct, l'état de la potence est incrémenté puis la potence est redessinée0 (utiliser la méthode **repaint**()).



III. Dans le package **com.usthb**, réaliser l'application **EUREKA**. Elle doit contenir :

- Un attribut de type dictionnaire (**HashMap**) pour contenir l'ensemble des joueurs inscrits au jeu. La clé utilisée pour accéder à un joueur doit être son numéro.
- Un attribut de type ensemble (**HashSet)** pour contenir la liste des thèmes définis.

- Une méthode **Initialisation** qui permet d'initialiser le jeu avec un ensemble de thèmes et questions. Les thèmes et questions sont saisies via le clavier dans un premier temps.
- Une méthode **Inscription** qui permet de récupérer les informations nécessaires d'un joueur.
- Une méthode **Se connecter** qui permet de saisir le nom et mot de passe d'un joueur et lui afficher une fenêtre qui lui permet de jouer une nouvelle partie correspondant au dernier niveau atteint.
- Une méthode **getScoreCourant**() pour un joueur connecté.

Proposez les différents composants graphiques (**JTexteField, JPanel, JLabel**, etc) nécessaires à la fenêtre principale de votre application, ainsi qu'à la fenêtre correspondant à une partie de jeu.

Remarques :

1. **Le concept d'encapsulation doit être respecté.**
2. **Le code doit être lisible et bien commenté.**
3. **Le code doit être optimisé (décomposer votre traitement en très petits problèmes, éviter la répétition des appels de la même méthode et les mêmes paramètres si possible en passant par des variables, éviter le gaspillage mémoire, par exemple des variables locales et des objets locaux dont on peut s'en passer).**
4. **Il est souhaitable de traiter le projet en binôme.**

Annexe

1) L'utilisation d'une **ArrayList/LinkedList** se fait comme suit :
Déclaration : ArrayList<type élément> liste = new ArrayList () ;
LinkedList<type élément> liste = new LinkedList () ;
Méthodes :

- liste.**add**(e) : rajoute l'objet **e** à la fin de la liste
- liste.**get**(i) : retourne l'objet se trouvant à l'indice i de la liste.
- liste.**remove**(i) : rajoute l'objet se trouvant à l'indice i de la liste.
- liste.**size**() : retourne le nombre d'éléments de la liste.
- liste.**contains** (e) : rend true ssi l'objet e existe dans la liste.

2) L'utilisation d'une **HashMap** se fait comme suit :
Déclaration : HashMap<type index,type donnée> table = new HashMap() ;
Méthodes :

- table.**put**(cl, e) : rajoute l'objet **e** de clé **cl** à l'index
- table.**get**(cl) : retourne l'objet **e** de clé **cl** si la clé existe sinon retoune **null**.
- tale.**containsKey**(cl) : rend true ssi l'objet de clé cl existe.

3) L'utilisation d'une **HashSet** se fait comme suit :
Déclaration : HashSet<type élément> set = new HashSet () ;
Méthodes :

- set.**add**(e) : rajoute l'objet **e** à l'ensemble
- set.**remove**(e) : rajoute l'objet e de l'ensemble.
- set.**size**() : retourne le nombre d'éléments de l'ensemble.
- set.**contains** (e) : rend true ssi l'objet e existe dans l'ensemble.
- set.**iterator**() : rend un itérateur (pointeur) pour parcourir les éléments de la liste

4) Méthode **paint** de la classe **Potence**

```
// Dessin de la potence
public void paint (Graphics g)
{
	// Le dessin s'adapte à l'espace attribué
	dimenson = getSize(); // de Component
	g.clearRect(0, 0, di.width-1, di.height-1);  //effacer
	g.drawRect(0, 0, di.width-1, di.height-1);   //tracer le cadre

	// s'adapter à l'espace du composant
	int taille = 12*(di.width/120);
	if (taille <8)     taille = 8;
	g.setFont (new Font("TimesRoman", Font.PLAIN, taille));
	if (etat >= 1) g.drawLine(l(30), h(120), l(90), h(120));
	if (etat >= 2) g.drawLine(l(30), h(120), l(30), h(40));
	if (etat >= 3) g.drawLine(l(60), h(120), l(30), h(90));
	if (etat >= 4) g.drawLine(l(30), h(40), l(80), h(40));
	if (etat >= 5) g.drawLine(l(30), h(60), l(50), h(40));
	if (etat >= 6) g.drawLine(l(70), h(40), l(70), h(60));
	if (etat >= 7) g.drawOval(l(65), h(60), l(10), h(10)); // tête
	if (etat >= 8)
	{
		g.drawLine(l(70), h(70), l(70), h(85)); // corps
```

```
            g.drawLine(l(70), h(70), l(65), h(75)); // corps
            g.drawLine(l(70), h(70), l(75), h(75)); // corps
            g.drawLine(l(70), h(85), l(65), h(95)); // corps
            g.drawLine(l(70), h(85), l(75), h(95)); // corps
            }

      if (trouve) g.drawString("Bravo! vous avez trouvé", l(10), h(150));
      else if (etat == 8) g.drawString("Vous êtes pendu !", l(10), h(150));
           else if (etat == 7)
                  g.drawString("Reste un coup à jouer !", l(10), h(150));
                else // (etat >=0 && etat <7)
                  g.drawString("Reste "+(8-etat)+"coups à jouer", l(10), h(150));
      }

// Mise à l'échelle en largeur de v
int l (int v)
{
      double k = Math.min(di.width/140., di.height/160);
      return (int)(v*k);
      }

// Mise à l'échelle en hauteur de v
int h (int v)
{
      double k = Math.min(di.width/140., di.height/160);
      return (int)(v*k);
      }
```