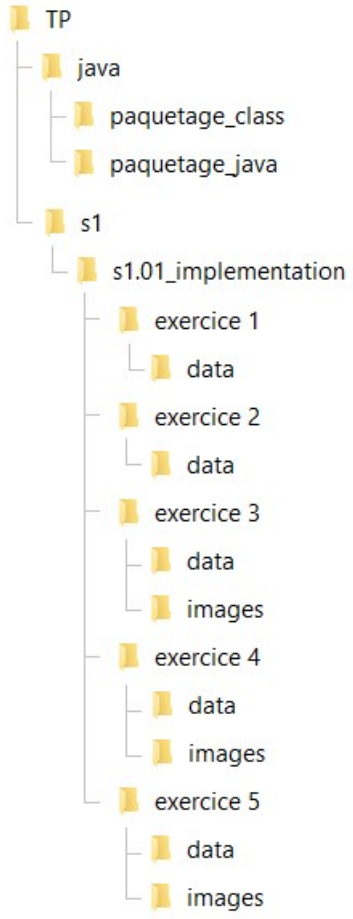
SAE 1.1

Les objectifs principaux de cette SAE sont :

- vous faire réutiliser un ensemble assez important des connaissances que vous avez normalement acquises au 1er semestre.
- Découvrir des nouvelles notions.
- Travailler en groupe
- Adapter des solutions à un nouveau besoin.
- Développer des nouvelles solutions
- Tester vos solutions
- Vous sensibiliser à des savoir-faire professionnels.
  Certaines entreprises fixent des règles très strictes sur la façon de coder,
  Vous trouverez en Annexes 1 les règles définies pour ce projet.
- Nous vous imposerons dans certains exercices, une façon de faire précise, vous devrez donc vous y tenir.

Le projet est découpé en plusieurs exercices qui vous mèneront vers une solution de plus en plus élaborée.

L'objectif sera de réaliser un jeu de plateau. Nous vous conseillons de structurer ce projet à l'aide de sous-répertoires.



Cette SAE est découpée en 2 parties :

Partie 1 : 4 exercices d'apprentissage

Partie 2 : application finale

Afin de vous aider dans la gestion du temps, nous vous indiquons à titre d'information, pour chaque exercice, une date de fin.

Il n'y aura au final qu'un seul rendu.

Nous vous donnerons de plus amples consignes dans les jours à venir.

Nous nous tenons à votre disposition, donc n'hésitez pas à venir nous solliciter.

PARTIE 1

Exercice n°1 **A finir pour jour 1 15h15**

Vous allez réaliser un mini labyrinthe. Un labyrinthe est un plateau sur lequel des Murs sont placés. Un Objet O se déplace à travers ce plateau, sans pouvoir en sortir et sans pouvoir traverser les murs.

Le labyrinthe devra être initialisé à partir du fichier **grille.data** (que vous trouverez dans les ressources)

Voici un extrait de la trace d'exécution :

```
+---+---+---+---+---+---+---+---+
| = | = | = | = | = |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   | = |   |
+---+---+---+---+---+---+---+---+
| = | = | = | = |   | = |   |   |
+---+---+---+---+---+---+---+---+
|   |   | O |   |   | = |   | = |
+---+---+---+---+---+---+---+---+
| = |   |   | = |   | = |   | = |
+---+---+---+---+---+---+---+---+
|   |   |   | = |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   | = | = | = |   | = |   | = |
+---+---+---+---+---+---+---+---+

direction ( N O S E ) : O

+---+---+---+---+---+---+---+---+
| = | = | = | = | = |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   | = |   |
+---+---+---+---+---+---+---+---+
| = | = | = | = |   | = |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   | O |   | = |   | = |
+---+---+---+---+---+---+---+---+
| = |   |   | = |   | = |   | = |
+---+---+---+---+---+---+---+---+
|   |   |   | = |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   | = | = | = |   | = |   | = |
+---+---+---+---+---+---+---+---+
```

```
+---+---+---+---+---+---+---+---+
| = | = | = | = | = |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   | = |   |
+---+---+---+---+---+---+---+---+
| = | = | = | = |   | = |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   | O |   | = |   | = |
+---+---+---+---+---+---+---+---+
| = |   |   | = |   | = |   | = |
+---+---+---+---+---+---+---+---+
|   |   |   | = |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   | = | = | = |   | = |   | = |
+---+---+---+---+---+---+---+---+

direction ( N O S E ) : N
```

Le chargement de la grille devra se faire à l'aide du fichier grille.data (contenu dans le dossier ressources).

Nous vous fournissons également un fichier TestLabyrinthe.java, que vous n'aurez théoriquement pas besoin de modifier.

Ci-contre la description UML de la classe Labyrinthe :

Labyrinthe
- posLig : int - posCol : int - grille : char[][]
+ Labyrinthe () + deplacer (char direction) + toString () : String

Travail à faire
Récupérez le fichier **TestLabyrinthe.java** que vous placerez dans le répertoire **exercice_01** Récupérez le fichier **grille.data** que vous placerez dans le répertoire **exercice_01/data** Ecrivez la classe **Labyrinthe** que vous placerez dans un fichier **Labyrinthe.java** dans le répertoire **exercice_01**

Exercice n°2 **A finir pour jour 1 18h**

Vous allez dans cet exercice enrichir la classe Labyrinthe de l'exercice précédent

LA classe Labyrinthe gérera deux fonctionnalités complémentaires :
- Les déplacements thoriques
- Une sortie matérialisée par un @

Nous souhaitons également introduire dans cet exercice
les prémices du Modèle MVC (Modèle Vue Controleur).

On trouve dans la partie :
- **Modèle**, les classes Métier, pour nous ce sera notre classe **Labyrinthe**
- **Vue**, les classes en charge de l'affichage et des saisies ce sera notre classe **IhmCui**
- **Controleur**, la classe **Controleur** permettant de faire le lien entre la partie Métier et Vue

Vous devez ajouter à votre classe Labyrinthe, les méthodes suivantes

public int getNbLignes (),	qui retourne le nombre de lignes de la grille
public int getNbColonnes (),	qui retourne le nombre de colonnes de la grille
public char getCase (int lig, int col),	qui retourne le contenu de la case lig col nous partons du principe que les valeurs fournies sont correctes cette méthode renverra un O si la case est occupée par l'Objet et un @ s'il s'agit de la case de sortie.
public boolean estSortie(),	qui indique si l'objet se trouve sur la case de sortie

Note importante, vous remarquerez qu'il n'y a aucune méthode qui retourne la grille, afin de garantir l'intégrité du Labyrinthe

Nous vous fournissons la Classe **Controleur**, ainsi que la classe **IhmCui**.
Il vous faudra compléter la classe **IhmCui**.

La classe **IhmCui** sera donc en charge de faire les lire et écrire, elle ne pourra pas directement accéder à la partie Metier, elle devra toujours passer par le **Controleur**.

Vous pouvez remarquer qu'elle possède un attribut ctrl permettant d'accéder au **Controleur**.

Vous ne devez donc plus faire appel à la méthode toString() de Labyrinthe pour afficher la grille, mais passer par les méthodes appropriées du **Controleur**.

Travail à faire

Dans un sous-répertoire **exercice_02**,
Récupérez les fichiers **Controleur.java** **IhmCui.java** **grille.data**
Ajoutez les nouvelles méthodes à votre classe **Labyrinthe**
Complétez le code de la classe **IhmCui**

Exercice n°3 **A finir pour jour 2 11h30**

Le mode console, c'est bien, mais il y a plus transcendant. Nous vous proposons dans cet exercice d'incorporer à votre labyrinthe une IHM en mode GUI.

Pour cela, rendez-vous sur la page des Fiches Complémentaires :

"Concevoir les bases d'une IHM GUI (en attendant le S2)"

Vous devrez contrôler les déplacements de votre personnage à l'aide des flèches directionnelles.

Nous vous conseillons de télécharger l'application exemple et de bien l'analyser.

Travail à faire
Dans un sous-répertoire **exercice_03**, Récupérez l'ensemble des images, placez-les dans un sous-répertoire **images**. Récupérez le fichier data de l'exercice précédent pour le placer dans le sous-répertoire **data** Récupérez la classe **Labyrinthe** de l'exercice précédent. **Aucune modification n'est nécessaire**. En vous appuyant sur la fiche complémentaire, écrire la classe **Controleur.**

Nota : les images ont une taille de 30x 30, n'oubliez donc pas de régler la largeur des images à l'aide de la méthode `setLargeurImg`.

pour fermer l'application à la fin, il vous faudra appeler la méthode `fermer` sur votre objet de type `FrameGrille`.

Exercice n°4 **A finir pour jour 2 15h**

Cet exercice est une simple adaptation de l'exercice précédent.

- Nous souhaitons tout d'abord améliorer l'esthétisme de l'affichage, en distinguant différents types de murs. Une illustration valant mieux qu'un long discours, voilà ce que nous souhaiterions obtenir :

Deux solutions s'offrent à nous :

1. Directement implémenter des symboles différents pour chaque type de murs. Vous ne mettriez plus dans la structure permettant de stocker le plateau un '=' mais « =1 », « =2 », « =3 », …
   Le souci, et lorsque l'on effectuera les déplacements, il faudra tester tous les types de Mur, ou extraire le premier caractère.
2. Ne pas implémenter différents symboles dans la structure permettant de stocker le plateau, mais uniquement fournir à IIhm une image distincte en fonction des cases adjacentes.

C'est cette dernière solution que nous retiendrons.

Attention la méthode `getCase` de **Labyrinthe** retournera toujours le contenu de la case, en effet la Partie Metier ne change pas, c'est l'interprétation graphique que l'on en fait qui change. C'est donc la Classe **Controleur** qui aura en charge de déterminer le bon type d'image en fonction du voisinage d'une Case =.

Nous vous proposons pour cela un nouvel ensemble d'images (de 40x40).

Nous souhaitons également afficher le nombre de déplacements

Travail à faire

1. Recopiez le contenu du dossier **exercice_03** dans le dossier **exercice_04**

   Etudiez la classe **Controleur** de la page "Concevoir les bases d'une IHM GUI (en attendant le S2)"

   Adaptez le **Controleur** et la classe **Labyrinthe** pour répondre à ce besoin.