SAE 01.1 - Implémentation jeu "Timeline"

Ce sujet présente la SAE. Il se basera sur le TP10 (pour la gestion des tableaux avec des objets) et sur le TP11 (pour la gestion d'un paquet de cartes).

Consigne

- **Durée** : 12h de travail étudiant.
- **Format** : travail en binôme.
- **Rendus attendus** :
  - un code commenté qui compile ;
  - des classes de test ;
  - un rapport qui présente votre réalisation.

Avant toute chose, nous vous conseillons de lire la section 7 décrivant le contenu attendu du rapport pour ne pas oublier des éléments lors de votre réalisation.

Consigne

N'oubliez pas de sauvegarder régulièrement votre projet (sauvegarde sur votre espace personnel, envoi par mail entre vous, utilisation d'outil de partage, ...) afin de ne pas perdre votre travail entre deux séances.

1 Présentation du jeu Timeline (rappel)

L'objectif de la SAE va être de programmer une version solitaire du jeu "Timeline" [1].

Le jeu est constitué de cartes représentant des monuments, des inventions, etc. Au verso de chaque carte est indiquée une date représentée par une année (cf figure 1).

FIGURE 1 – Exemple de carte (recto à gauche, verso à droite).

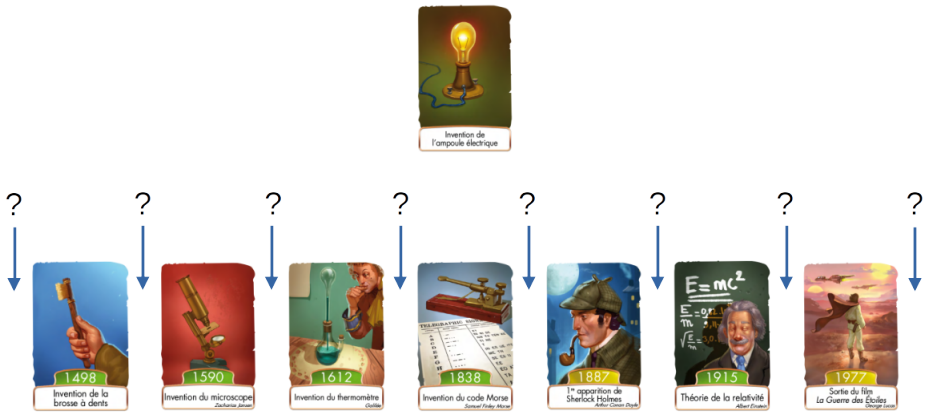

FIGURE 2 – À quelle position dans la frise insérer l'invention de l'ampoule ?

Le joueur tire un nombre de cartes au hasard dans sa main mais ne voit que les événements sans connaitre les dates. Il doit alors insérer les cartes une par une pour former une frise chronologique (cf figure 2).

A chaque fois qu'une carte est posée, on vérifie si l'année de la carte posée est bien située entre les années des cartes à gauche et à droite :

— Si c'est le cas, la carte est ajoutée à la frise chronologique ;
— sinon, la carte est défaussée et le joueur en pioche une nouvelle dans la pioche.

Dans cet exemple, l'invention de l'ampoule se situe en 1879 ; le joueur peut donc poser sa carte s'il avait prévu de la placer entre l'invention du morse (1838) et la première apparition de Sherlock Holmes (1887). Dans les autres cas, il a échoué à poser sa carte, doit défausser sa carte et en piocher une nouvelle.

L'objectif pour le joueur est de poser toutes ses cartes en se trompant le moins souvent.

2 Organisation de la SAE

La SAE est organisée par étapes. Chaque étape consiste à écrire certaines fonctionnalités et les tests associés.

Consigne

Les étapes seront progressivement validées. Ne passez à l'étape suivante que quand une étape est validée (code complet et commenté et classe de test écrite et validée).

1. (édité par Asmodée - `https://print-and-play.asmodee.fun/fr/timeline/`)

3 Étape 1 : Classe `Carte`

3.1 Descriptif des cartes

Désormais, le cartes sont légèrement plus complexes que pour le TP11. Chaque carte est décrite par

- un nom représentant l'événement (par exemple, "Invention de la brosse à dents") ;
- un entier représentant la date de l'événement (par exemple, 1498) ;
- un booléen qui précise si la face affichée correspond au recto ou au verso (date visible) de la carte.

3.2 Chargement de fichiers

Les cartes d'un jeu sont décrites dans des fichiers texte fournis sur arche. Un tel fichier contient une carte par ligne et les lignes sont structurées sous la forme suivante `<EVENEMENT>:<DATE>`.

Ainsi, le début du fichier `timeline.txt` contient les lignes suivantes :

```
L'apparition de la ceramique:-9000
L'invention du papier:-200
La fondation du theoreme de Pythagore:-548
L'invention du morse:1838
```

La classe **fournie sur arche** `LectureFichier` permet de lire un fichier texte. Elle contient

- un constructeur `LectureFichier(String nom)` qui prend en paramètre un nom de fichier et vérifie si le fichier est bien présent. Si le fichier est absent, le constructeur renvoie une erreur et le programme s'arrête.
- une méthode `String[] readFile()` qui lit le fichier passé à la construction de l'objet et retourne un tableau de `String`. Chaque `String` correspond à une ligne du fichier lu.

Utilisé pour lire le fichier `timeline.txt`, le tableau retourné possède 60 cases car le fichier contient 60 lignes. La chaîne située dans la première case vaut `"L'apparition de la ceramique:-9000"`, correspond à la première ligne du fichier et à une carte du jeu.

3.3 Création d'une carte à partir d'une chaine

On souhaite pouvoir construire une carte en passant une chaîne de caractères de la forme `<EVENEMENT>:<DATE>`.

Pour aborder cette question, on pourra parcourir la chaîne de caractères et utiliser les méthodes suivantes :

— la méthode `char charAt(int i)` de la classe `String` qui retourne le caractère à la position `i` de la chaine sur laquelle on appelle la méthode ;
— la méthode `int Integer.parseInt(String s)` qui convertit la chaine `s` en son entier correspondant.

Question 1
Proposer un constructeur dans la classe `Carte` qui permette de faire cela. Vous supposerez que la chaîne passée en paramètre est bien formée et vous écrirez d'abord un algorithme permettant de séparer la chaine en deux en fonction de la position du caractère `':'`.

3.4 Affichage d'une carte

Une carte s'affiche différemment selon qu'elle est sur son recto ou sur son verso.
— Si la carte est sur le verso, sa date n'est pas visible et la carte doit s'afficher sous la forme `"??? -> <EVENEMENT>"`. Par exemple,

```
??? -> L'apparition de la ceramique
```

— Si la carte est sur le verso, sa date est visible et la carte doit s'afficher sous la forme `"<DATE> -> <EVENEMENT>"`. Par exemple,

```
-9000 -> L'apparition de la ceramique
```

Question 2
Écrire la méthode `toString` correspondante.

3.5 `MainCarte` pour valider

Afin de valider vos méthodes, écrire un `main` dans la classe `MainCarte` qui

1. charge le fichier `timeline.txt` et stocke le tableau de `String` ;
2. puis, pour chaque chaîne de ce tableau,
   — construit une carte,
   — l'affiche,
   — la change de côté (recto / verso),
   — l'affiche à nouveau.

Question 3
Écrire la classe `MainCarte` et toutes les méthodes utiles de la classe `Carte`.

3.6 Tests

Question 4
Compléter la classe de test `TestCarte.java` chargée de vérifier que les différentes méthodes écrites fonctionnent.

Consigne
Valider complètement cette partie avant de passer à la suite.

4 Étape 2 : Gestion de la pioche et de la main d'un joueur

Vous disposez maintenant de cartes. L'objectif de cette partie est de pouvoir disposer de paquet de cartes pour représenter la pioche et la main de cartes du joueur.

4.1 Constructeur, ajout et suppression de carte

Le TP11 vous a proposé d'écrire une classe destinée à gérer un paquet de cartes. Cette classe doit possèder

- un constructeur vide ;
- un constructeur à partir d'un tableau de cartes ;
- une méthode `ajouterCarteFin` ;
- une méthode `retirerCarte`.

Question 5
Adapter et finir le TP11, pour disposer d'une classe `Paquet` permettant de manipuler un paquet de cartes. Vous penserez à changer la classe `Carte` utilisée.

4.2 Constructeur à partir d'un fichier

On souhaite pouvoir construire un paquet de cartes directement à partir d'un nom de fichier et de la classe `LectureFichier`.

Question 6
Proposer un constructeur supplémentaire capable de créer un paquet de cartes en donnant uniquement un nom de fichier et en chargeant les cartes décrites dans ce fichier.

4.3 Piocher une carte

On souhaite disposer d'une méthode permettant de piocher une carte aléatoirement dans un paquet. Cette méthode ne prend aucun paramètre, retourne la carte tirée au hasard et la supprime du paquet. Si le paquet est vide, la méthode retourne simplement `null`.

Afin de sélectionner une carte au hasard, on pourra utiliser la classe JAVA `Random`. Cette classe :

- nécessite d'importer le package `java.util.Random` tout en haut de la classe `Paquet` ;
- possède un constructeur vide ;
- possède une méthode `int nextInt(int bound)` qui retourne un entier choisi de manière aléatoire et compris entre `0` et `bound-1` (compris).

Le descriptif complet de la classe est disponible à l'adresse `https://docs.oracle.com/javase/8/docs/api/java/util/Random.html`.

Question 7
Ajouter une méthode `piocherHasard` à la classe `Paquet`.

4.4 Méthode toString

Question 8
Compléter la méthode `toString` proposée dans le `TP11` pour pouvoir afficher toutes les cartes d'un paquet de cartes.

4.5 Tests

Question 9
Compléter la classe de test `TestPaquet.java` chargée de vérifier que les différentes méthodes écrites fonctionnent.

Consigne
Valider complètement cette partie avant de passer à la suite.

5 Étape 3 : Gestion de la frise chronologique

Vous disposez de la classe `Carte` et de la classe `Paquet`. On souhaite désormais écrire la classe `Frise` censée représenter la frise chronologique contenant les cartes placées par le joueur.

La classe `Frise` possède aussi un tableau de cartes en attribut, mais les cartes de la classe `Frise` sont **visibles et triées** dans l'ordre chronologique.

Ainsi, la classe `Frise` possède les trois méthodes suivantes :

— la méthode `ajouterCarteTrie` qui consiste à insérer une carte à la bonne place dans le tableau de cartes triées. Cette méthode prend en paramètre une carte et doit insérer cette carte à la bonne place dans le tableau de cartes supposé déjà trié.

— la méthode `verifierCarteAprés` qui prend en paramètre une carte `c` et une place `p` et vérifie (sans rien faire) si la carte `c` peut s'insèrer **derrière** la place `p` tout en respectant l'ordre chronologique de la frise. Si la carte est correctement placée, la méthode `verifierCarteAprés` retourne `true`, sinon la méthode retourne `false`.

— la méthode `insererCarteApres` qui prend en paramètre une carte `c` et une place `p`. Cette méthode vérifie si la carte `c` s'insère bien **derrière** la place `p` en respectant l'ordre chronologique de la frise et l'ajoute si c'est bien le cas. Elle retourne un boolean valant `true` si et seulement si l'insertion a eu lieu.

Vous penserez à ajouter une méthode `toString` dans la classe `Frise`.

Question 10
Déclarer et écrire la classe `Frise` qui permet d'insérer des cartes en vérifiant l'ordre chronologique.

Question 11
Compléter la classe de test `TestFrise` qui vérifie que la classe `Frise` fonctionne correctement.

Consigne
Valider complètement cette partie avant de passer à la suite.

6 Étape 4 : Mise en place et exécution du jeu

Vous disposez désormais de tous les éléments pour écrire la classe `Jeu` chargée d'exécuter le jeu.

La classe `Jeu` contient

— un paquet de cartes représentant la main du joueur ;

— un paquet de cartes représentant la pioche des cartes ;

— une frise chronologique initialement vide.

6.1 Lancement du jeu

Au lancement du jeu, il est nécessaire de préciser `tailleMain` la taille de la main du joueur et le nom du fichier contenant le descriptif des cartes pour cette partie.

L'installation du jeu consiste à
— charger toutes les cartes dans la pioche à partir du nom de fichier ;
— tirer au hasard et à partir de la pioche le nombre de cartes nécessaires pour remplir la main du joueur ;
— créer une frise vide.

6.2 Déroulement de la partie

La partie se déroule en tour par tour. À chaque tour,
— le jeu affiche à l'écran la frise chronologique et les cartes dans la main du joueur ;
— le jeu demande à l'utilisateur le numéro de la carte qu'il souhaite jouer de sa main (entre 0 et `tailleMain-1`) ;
— le jeu demande **derrière** quel numéro de carte de la frise existante le joueur souhaite poser sa carte (-1 signifie qu'il souhaite la poser en première position) ;
— le jeu joue ce coup en fonction des règles du jeu, informe le joueur si la carte proposée est bien placée ou non et met à jour l'état du jeu en fonction de la réussite ou non du joueur.

6.3 Fin du jeu

Le jeu s'arrête selon une des conditions suivantes :
— le joueur a joué toutes les cartes de sa main : c'est une victoire.
— la pioche est vide : c'est une défaite.

6.4 Exemple

Voici un exemple d'exécution du jeu (le premier coup) tel qu'affiché dans la console :

Affichage console

```
--------------
frise
--------------
0. 900 -> L'invention du sablier
1. 1612 -> L'invention du thermometre
2. 1817 -> L'invention du velocipede
3. 1873 -> L'invention du jeans
4. 1918 -> La decouverte de la penicilline
--------------
main du joueur
--------------
 0. ??? -> La tour Eiffel
 1. ??? -> L'invention du coffre-fort
 2. ??? -> L'invention des lunettes
 3. ??? -> L'invention de l'arbalete
```

```
 4. ??? -> L'invention du sous-marin
 5. ??? -> L'invention du telephone
 6. ??? -> L'invention du cheque
 7. ??? -> L'invention du transistor

quelle carte de votre main ?
0
??? -> La tour Eiffel
derriere quelle carte de la frise ?
3
entre ....
   - 1873 -> L'invention du jeans
   - 1918 -> La decouverte de la penicilline
- carte jouée : 1889 -> La tour Eiffel
 !!! Une carte de placee !!!

--------------
frise
--------------
0. 900 -> L'invention du sablier
1. 1612 -> L'invention du thermometre
2. 1817 -> L'invention du velocipede
3. 1873 -> L'invention du jeans
4. 1889 -> La tour Eiffel
4. 1918 -> La decouverte de la penicilline
--------------
main du joueur
--------------
 0. ??? -> L'invention du coffre-fort
 1. ??? -> L'invention des lunettes
 2. ??? -> L'invention de l'arbalete
 3. ??? -> L'invention du sous-marin
 4. ??? -> L'invention du telephone
 5. ??? -> L'invention du cheque
 6. ??? -> L'invention du transistor
[...]
```

6.5 Travail à réaliser

Question 12

Décider des méthodes et programmer la classe `Jeu` pour pouvoir créer et exécuter une partie.

Question 13

Écrire la classe `ProgJeu` permettant de créer et de lancer un jeu. Le nom du fichier texte à utiliser pour construire la pioche sera passé en paramètre au programme (via `args` du `main`).

Consigne
Valider complètement cette partie avant de passer à la suite.

7 Rendu attendu

7.1 Version 1

La version 1 de votre application doit contenir
— votre code complet (classes, classes de test, classes main attendues) ;
— un code commenté (commentaires internes, javadoc) ;
— un petit compte-rendu qui présente votre travail.

Ce compte-rendu devra préciser
— les noms, prénoms et groupe des membres du binôme ;
— l'avancée de votre projet (à quelle partie en êtes vous) ;
— un déroulé de votre travail (combien de temps pour chacune des parties réalisées) ;
— les difficultés éventuelles rencontrées lors du travail ;
— quelques explications sur la manière dont vous avez programmé la classe `Jeu`.

Consigne
IMPORTANT : Avant de passer à la suite, sauver une version 1 de votre application dans un fichier zip et déposer cette version sur arche.

7.2 Version 2

Une fois votre version 1 déposée sur arche, copier cette version et la renommer en une version v2 pour aborder les questions optionnelles. Cette nouvelle version servira de base à la suite de votre travail pour éviter les confusions.

Si vous répondez à des questions optionnelles, vous penserez à déposer votre version 2 dans le second dépôt arche.

8 Étape 5 : Options de jeu (partie optionnelle)

Une fois la version 1 complétée, différentes extensions sont possibles pour améliorer votre jeu. Si vous avez le temps, vous pouvez développer une (ou plusieurs) de ces extensions

8.1 Gestion du score

Désormais, l'application compte en plus le nombre de coups nécessaires pour finir la partie. Ce nombre de coups correspondra au score de l'utilisateur.

À la fin d'une partie, le score de l'utilisateur est affichée à l'écran et le jeu redemande si on souhaite faire une nouvelle partie.

Question optionelle 14
Ajouter la gestion du high-score à l'application.

8.2 Gestion du meilleur score

Le jeu mémorise les 5 meilleurs scores obtenus par un joueur. Dés qu'un meilleur score est obtenu, le jeu demande le nom du joueur et associe ce nom au high-score.

À la fin de chaque partie, avant de redemander si le joueur veut lancer une nouvelle partie, le jeu affiche les high-score mémorisés.

Comme le nombre de coups dépend de la main initiale du joueur, on utilisera au début les high score que pour une main standard (6 cartes) puis on pourra stocker des high score pour chaque niveau de difficulté (taille de la main initiale)

Question optionelle 15
Pour faire cela en tirant parti de la programmation orientée objet, gérer les high-score à l'aide d'une classe dédiée nommée `HighScore`.

8.3 Sauvegarde des high-scores

On souhaite sauver et charger les high-score dans un fichier texte pour pouvoir mémoriser les meilleurs scores entre deux lancements de l'application.

La classe `EcritureFichier` est fournie sur arche et permet d'écrire dans un fichier. Elle fonctionne en plusieurs étapes

- on créée d'abord un objet `EcritureFichier` en lui donnant le nom du fichier en paramètre ;
- on ouvre ensuite le fichier avec la méthode `void ouvrirFichier()` ;
- on écrit dans le fichier ligne par ligne grâce à la méthode `void ecrireFichier(String ligne)` ;
- enfin, on ferme et on sauve le fichier avec la méthode `void fermerFichier()`.

Par exemple, le code suivant crée un fichier "test.txt" et ajoute 10 lignes à l'intérieur

```
// creation de l'objet pour ecrire
EcritureFichier fichier = new EcritureFichier("test.txt");

// ouverture du ficheir
fichier.ouvrirFichier();

// ecriture de 10 lignes
for (int i = 0; i < 10; i++){
  fichier.ecrireFichier("ligne " + i);
}

// fermeture et sauvegarde
fichier.fermerFichier();
```

Question optionelle 16
A l'aide de la classe `EcritureFichier` fournie sur arche, gérer la sauvegarde et le chargement des high-score (pensez au préalable au format du fichier).

8.4 Fusion de paquets de cartes

On souhaite pouvoir fusionner deux paquets de cartes pour mélanger des cartes issues de deux fichiers différents.

Question optionelle 17
Modifier l'application pour demander au lancement plusieurs fichiers de carte et fusionner les paquets de cartes pour n'en faire qu'une seule pioche.

8.5 Partie multijoueurs

On souhaite pouvoir lancer une partie avec plusieurs joueurs. Au début du jeu, le jeu demande le nombre de joueurs. Chaque joueur possède une main de cartes dédiée.

Le jeu demande ensuite à tour de rôle à chaque joueur de placer une carte de sa main. Le premier joueur a avoir posé toutes ses cartes a gagné. Si la pioche est vide avant qu'un joueur ait gagné, tout le monde a perdu.

Question optionelle 18
Ajouter cet aspect multi-joueur à votre jeu.