

# Examen pratique de juin 2024

## SHIFUMI OU PIERRE-FEUILLE-CISEAUX

L'application à réaliser doit permettre de jouer une partie de *Shifumi* (ou *Chifoumi*) contre une IA simplifiée.

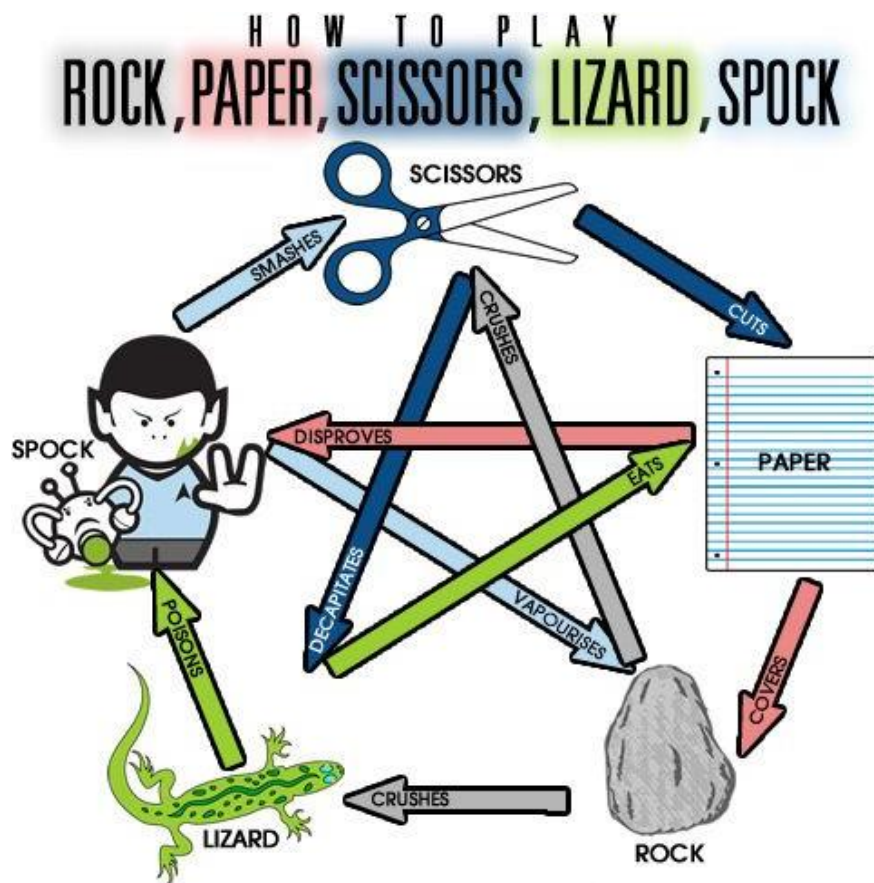
Pour rappel, le **Shifumi** est un jeu opposant 2 joueurs qui doivent, chacun, former simultanément une arme à l'aide de l'une de ses mains. Chacune des armes (pierre, feuille et ciseaux) possède un ascendant sur l'une des autres : la pierre émousse les ciseaux, les ciseaux coupent la feuille, le feuille enveloppe la pierre. Si les armes choisies sont les mêmes, il y a égalité.



**Fig. 1** – Les formes des armes, dans cet ordre : pierre, feuille et ciseaux.

La victoire revient au premier joueur qui remporte 3 manches.

Pour cette application, vous devrez programmer une variante de ce jeu qui a été popularisée par la série américaine « *The Big Bang Theory* ». Il s'agit de **Pierre-Feuille-Ciseaux-Lézard-Spock**. Ici, les règles classiques s'appliquent, mais il faut ajouter que le lézard mange le papier, empoisonne Spock, est écrasé par la pierre et est décapité par les ciseaux. Spock vaporise la pierre, casse les ciseaux et est discrédité par le papier.



**Fig. 2** – Illustration des règles de la variante Pierre-Feuille-Ciseaux-Lézard-Spock.

## DESCRIPTION DE L'APPLICATION

*Référez-vous aux exemples d'exécution donnés en fin d'énoncé afin de mieux comprendre le résultat attendu.*

Comme indiqué précédemment, l'application doit permettre à l'utilisateur de jouer une partie de *Pierre-Feuille-Ciseaux-Lézard-Spock* contre une IA. Celui qui remporte en premier 3 manches est désigné vainqueur de la partie.

A chaque manche, l'utilisateur saisit le nom de son arme et l'IA se voit attribuer aléatoirement l'une des 5 armes disponibles.

En fin de partie, l'application doit afficher des statistiques portant sur l'entièreté des manches jouées. Plus précisément, celles-ci consistent en l'affichage du nombre d'utilisations de chaque arme, que ce soit par le joueur ou par l'IA.

## AVANT-PROPOS

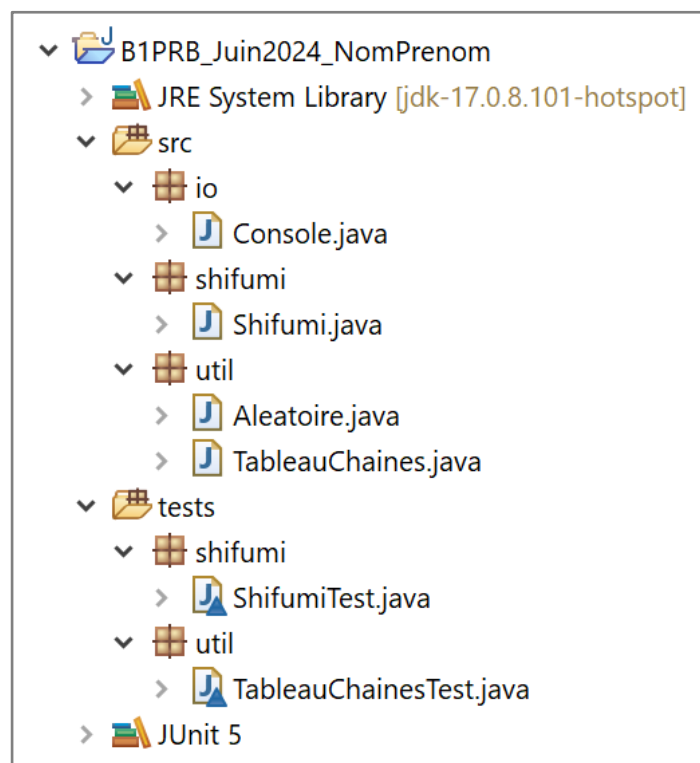
À deux exceptions près, les types **String** et **PrintStream**, vous ne pouvez pas utiliser le paradigme orienté objet pour coder cette application. N'oubliez pas de qualifier toutes vos fonctions de **static** !

## PRÉPARATION

1. Dans Eclipse, créez un projet Java nommé **B1PRB\_Juin2024\_NomPrenom**, où **NomPrenom** doit être remplacé par vos nom et prénom.

N'oubliez pas de vérifier que l'encodage utilisé pour les fichiers est l'**UTF-8**. Pour ce faire, il faut suivre le chemin : *Window > Preferences > General > Workspace > Text file encoding*.

2. Reproduisez exactement l'organisation suivante :



Les classes **Console** et **Aleatoire** sont à télécharger au début de la page *HELMo Learn* du cours. Les autres classes doivent être créées par vos soins. La fonction *main* doit être déclarée dans la classe **Shifumi**.

*Le projet déposé sur l'espace de cours doit être complet et exécutable !*

## FONCTIONNALITÉS INDISPENSABLES


Les fonctionnalités suivantes doivent être opérationnelles, tout en veillant à respecter les consignes données dans leur description :

- Les fonctions *ajouterElement* et *valeurAleatoire* de la classe **TableauChaines** doivent être correctement réalisées, testées et utilisées.
- Dans la classe **Shifumi**, un tableau de chaînes de caractères à une dimension doit être correctement créé (voir la section « Les structures de données ») et initialisé avec les **noms des 5 armes**.
- Dans la fonction *main* de la classe **Shifumi**, un tableau de chaînes de caractères à deux dimensions doit être correctement créé et initialisé afin d'y répertorier **l'historique des armes utilisées par les 2 joueurs à chaque manche** (voir la section « Les structures de données »).
- La fonction *main* doit permettre de jouer autant de manches que souhaité, et ce, sans détermination du vainqueur, ni calcul des scores (voir l'exemple d'exécution « MINIMUM REQUIS » en fin d'énoncé). A chaque manche, le joueur humain doit pouvoir saisir une arme et l'IA doit se voir attribuer une arme aléatoirement à l'aide de la fonction **TableauChaines.valeurAleatoire**. Les noms des armes utilisées doivent être ajoutés à l'historique des manches, c'est-à-dire au tableau de chaînes de caractères à deux dimensions cité au point précédent.

*Dans la version minimale, il n'est pas nécessaire de vérifier la validité des données saisies par l'utilisateur.*

*Si l'une de ces fonctionnalités n'est pas observée, cela entraîne automatiquement l'échec pour l'examen. Attention, la réussite de ces fonctionnalités seules ne garantit pas l'obtention d'une note supérieure ou égale à 10 / 20 ! Ne vous contentez pas de ces fonctionnalités et veillez à soigner les tests unitaires et la javadoc des fonctions.*

## APPROCHE RECOMMANDÉE

Dans un premier temps, **ne réalisez que les fonctions indispensables**. Ces dernières sont indiquées dans la section « *Fonctionnalités indispensables* » et sont marquées d'un point d'exclamation  au niveau de leurs descriptions.

Réalisez ensuite la fonction **main** sur base des directives données dans la section « *Fonctionnalités indispensables* » et de l'exemple d'exécution **MINIMUM REQUIS**.

Il est recommandé de ne réaliser la suite du programme que lorsque ces deux premières étapes sont terminées.

## LES STRUCTURES DE DONNÉES

### ENREGISTRER LES NOMS DES ARMES

Utilisez un **tableau de chaînes de caractères à une dimension** pour y spécifier les 5 noms des armes, en respectant l'ordre donné ci-dessous :

|           |           |          |          |          |
|-----------|-----------|----------|----------|----------|
| "Ciseaux" | "Feuille" | "Pierre" | "Lézard" | "Spock"  |
| <i>0</i>  | <i>1</i>  | <i>2</i> | <i>3</i> | <i>4</i> |

*Exceptionnellement pour ce programme, la référence de ce tableau doit être mémorisée dans une constante privée de la classe **Shifumi**, plus précisément sous la forme :*

```
public class Shifumi {
    private static final String[] ARMES = ... ;
    ...
}
```

*De cette manière, les fonctions **main** et **comparer** peuvent toutes deux accéder à son contenu (en lecture uniquement).*

**ENREGISTRER L'HISTORIQUE DES MANCHES**

Utilisez un **tableau de chaînes de caractères à deux dimensions** pour enregistrer les noms des armes utilisées par les 2 joueurs à chaque manche.

Initialement, ce tableau ne comporte que 2 lignes vides, une par joueur :

|   |  |
|---|--|
| 0 |  |
| 1 |  |

Supposons maintenant que deux manches aient été jouées, que le joueur humain ait saisi à chaque manche l'arme *Spock* et que l'IA se soit vue attribuer, dans cet ordre, les armes *Ciseaux* (manche 1) et *Feuille* (manche 2). Dans ce cas, le tableau suivant est obtenu :

|   |           |           |
|---|-----------|-----------|
| 0 | "Spock"   | "Spock"   |
| 1 | "Ciseaux" | "Feuille" |

*N'oubliez que les noms des armes doivent être ajoutés à ce tableau à l'aide de la fonction `TableauChaines.ajouterElement`.*

**ENREGISTRER LES INTERACTIONS ENTRE LES ARMES**

Dans la version idéale de l'application, il est attendu d'afficher les armes en spécifiant leur interaction propre, par exemple : « *Spock est discrédité par Feuille* », « *Spock vaporise Ciseaux* » ...

Pour ce faire, utilisez un **tableau de chaînes de caractères à deux dimensions** pour y spécifier de la manière suivante toutes les interactions possibles entre les différentes armes :

|   | 0                     | 1                     | 2                     | 3                     | 4                     |
|---|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
| 0 | "ne font rien contre" | "coupent"             | "sont émoussés par"   | "décapitent"          | "sont cassés par"     |
| 1 | "est coupée par"      | "ne fait rien contre" | "enveloppe"           | "est mangée par"      | "discrédite"          |
| 2 | "émousse"             | "est enveloppée par"  | "ne fait rien contre" | "écrase"              | "est vaporisée par"   |
| 3 | "est décapité par"    | "mange"               | "est écrasé par"      | "ne fait rien contre" | "empoisonne"          |
| 4 | "casse"               | "est discrédité par"  | "vaporise"            | "est empoisonné par"  | "ne fait rien contre" |

Les indices des lignes et colonnes correspondent aux indices utilisés dans le tableau *ARMES*. Autrement dit, si les armes utilisées lors d'une manche sont respectivement « *Spock* » (située à l'indice 4 dans *ARMES*) et « *Feuille* » (située à l'indice 1 dans *ARMES*), alors l'interaction correspondante est située aux indices (4 ; 1) du tableau à deux dimensions et vaut « est discrédité par ».

## LES CLASSES `TableauChaines` ET `TableauChainesTest`

Toutes les fonctions de la classe `TableauChaines` doivent être documentées avec des commentaires javadoc et testées à l'aide de JUnit 5.

Dans la classe `TableauChaines`, déclarez :



→ Une fonction nommée `ajouterElement` qui retourne une copie des éléments d'un tableau de chaînes de caractères à une dimension augmentée d'un nouvel élément :

```
public static String[] ajouterElement(String[] t,
                                     String element)
```

Le paramètre `t` est le tableau dont les éléments doivent être copiés.

Le paramètre `element` est l'élément à ajouter aux éléments du tableau `t`.

*Exemple : si `t` est le tableau `["Feuille", "Spock"]` et `element` la chaîne `"Spock"`, alors le tableau retourné est `["Feuille", "Spock", "Spock"]`.*

[SUGGESTION] Aidez-vous de la fonction `copyOf` de la classe `Arrays`.

[PRÉCONDITION] La référence réceptionnée par le paramètre `t` est supposée valide (ne vaut pas `null`).



→ Une fonction nommée `valeurAleatoire` qui retourne une chaîne de caractères choisie aléatoirement parmi toutes celles présentes dans un tableau de chaînes de caractères à une dimension :

```
public static String valeurAleatoire(String[] t)
```

Le paramètre `t` est le tableau dans lequel la valeur doit être choisie.

*Exemple : si `t` est le tableau `["Ciseaux", "Feuille", "Pierre", "Lézard", "Spock"]`, alors la fonction pourrait retourner `"Feuille"`.*

Méthodologie conseillée pour les tests :

Assurez-vous que la chaîne retournée est bien l'une des chaînes présentes dans le



*tableau.*

*Effectuez 3 appels consécutifs de la fonction et assurez-vous qu'au moins une des chaînes retournées est différente des deux autres.*

**[CONTRAİNTE]** Vous devez utiliser la fonction *aleatoire* de la classe **Aleatoire** pour obtenir une position aléatoire au sein du tableau.

**[PRÉCONDITIONS]** La référence réceptionnée par le paramètre *t* est supposée valide (ne vaut pas *null*) et la longueur du tableau est supposée supérieure ou égale à 1.

→ Une fonction nommée *commencePar* qui détermine la position d'une chaîne de caractères au sein d'un tableau de chaînes de caractères à une dimension, et ce, sans tenir compte de la casse (par exemple, la chaîne *"ciseaux"* est considérée comme équivalente à la chaîne *"Ciseaux"*) :

```
public static int commencePar(String[] t, String prefixe)
```

Le paramètre *t* est le tableau contenant les éléments parmi lesquels la recherche doit être effectuée.

Le paramètre *prefixe* est le début (ou l'entièreté) de la chaîne de caractères recherchée.

*La recherche dans le tableau doit s'effectuer de gauche à droite, à partir de l'indice 0 !*

Exemple : si *t* est le tableau [*"Ciseaux"*, *"Feuille"*, *"Splash"*, *"Lézard"*, *"Spock"*] et *prefixe* la chaîne *"sp"*, alors la fonction retourne l'entier 2 (*"Splash"* étant la première chaîne trouvée qui débute par *"sp"*).

**[CAS PARTICULIER]** Si aucune chaîne du tableau *t* ne commence par le préfixe spécifié, alors la fonction retourne -1.

**[CAS PARTICULIER 2]** La fonction doit pouvoir gérer la présence de références *null* au sein du tableau.

Exemple : si *t* est le tableau [*"Ciseaux"*, *"Feuille"*, *null*, *"Lézard"*,

*"Spock"] et **prefixe** la chaîne "sp", alors la fonction retourne l'entier 4 ("Spock" étant la première chaîne trouvée qui débute par "sp").*

**[SUGGESTION]** Aidez-vous des fonctions *startsWith*, *toLowerCase* et/ou *toUpperCase* de la classe **String**.

**[PRÉCONDITIONS]** Les paramètres réceptionnés par la fonction sont supposés valides (références différentes de **null**).

→ Une fonction nommée **compter** qui détermine le nombre d'apparitions d'une chaîne de caractères au sein d'un tableau de chaînes de caractères à une dimension, et ce, en tenant compte de la casse (par exemple, la chaîne **"ciseaux"** est considérée comme différente de la chaîne **"Ciseaux"**) :

```
public static int compter(String[] t, String chaine)
```

Le paramètre **t** est le tableau contenant les éléments parmi lesquels le dénombrement doit être effectuée.

Le paramètre **chaine** est la chaîne de caractères à dénombrer.

*Exemple : si **t** est le tableau ["Feuille", "Lézard", "Spock", "Pierre", "Feuille", "Feuille"] et **chaine** la chaîne "Feuille", alors la fonction retourne l'entier 3 ("Feuille" apparaît aux indices 0, 4 et 5).*

**[CAS PARTICULIER]** La fonction doit pouvoir gérer la présence de références **null** au sein du tableau.

*Exemple : si **t** est le tableau ["Feuille", "Lézard", "Spock", "Pierre", **null**, "Feuille"] et **chaine** la chaîne "Feuille", alors la fonction retourne l'entier 2 ("Feuille" apparaît aux indices 0 et 5, la référence **null** ne provoque pas d'erreur).*

**[PRÉCONDITIONS]** Les paramètres réceptionnés par la fonction sont supposés valides (références différentes de **null**).

## LES CLASSES SHIFUMI ET SHIFUMITEST

À l'exception de la fonction *main*, toutes les fonctions de la classe **Shifumi** doivent être documentées avec des commentaires javadoc. Cependant, seule la fonction *comparer* doit être testée à l'aide de JUnit 5.

Dans la classe **Shifumi**, déclarez :

- Une fonction nommée *selectionnerChaine* qui demande à l'utilisateur de sélectionner une chaîne de caractères parmi celles qui lui sont proposées. Pour ce faire, l'utilisateur doit saisir une chaîne de caractères qui correspond, sans tenir compte de la casse, au début (ou à l'entièreté) de l'une des chaînes de caractères présentes dans le tableau spécifié :

```
public static String selectionnerChaine(
                                String[] chainesAdmises)
```

Le paramètre `chainesAdmises` est un tableau contenant les chaînes de caractères proposées.

*La question affichée à l'utilisateur doit être constituée à partir des chaînes de caractères présentes dans le tableau.*

*L'acquisition **doit être répétée** tant que la saisie ne correspond pas au début (ou à l'entièreté) de l'une des chaînes présentes dans le tableau spécifié.*

Exemple : si `chainesAdmises` est le tableau `["Ciseaux", "Feuille", "Pierre", "Lézard", "Spock"]`, alors l'exécution de la fonction pourrait produire l'affichage suivant en console :

```
Ciseaux, Feuille, Pierre, Lézard, Spock ? ↵
Ciseaux, Feuille, Pierre, Lézard, Spock ? Papier↵
Ciseaux, Feuille, Pierre, Lézard, Spock ? Pierrot ↵
Ciseaux, Feuille, Pierre, Lézard, Spock ? pi↵
```

*L'acquisition est ici répétée 4 fois car la 1<sup>ère</sup> saisie est une chaîne vide et les 2 suivantes ne sont les préfixes d'aucune des chaînes du tableau. La 4<sup>e</sup> saisie est correcte car elle*

correspond au préfixe de la chaîne *"Pierre"* étant donné que la casse n'est pas prise en compte.

Dans ce cas, la fonction retourne la chaîne *"Pierre"*.

**[CONTRAINTE]** Vous devez utiliser la fonction *commencePar* de la classe **TableauChaines** pour obtenir une position aléatoire au sein du tableau.

**[SUGGESTION]** Si vous le souhaitez, vous pouvez utiliser la fonction *trim* de la classe **String** pour « nettoyer » la saisie.

**[PRÉCONDITION]** La référence réceptionnée par le paramètre *chainesAdmises* est supposée valide (ne vaut pas *null*) et la longueur du tableau est supposée supérieure ou égale à 1.

→ Une fonction nommée *comparer* qui compare deux armes entre elles afin de déterminer le résultat de la confrontation.

```
public static int comparer(String arme1, String arme2)
```

Les paramètres *arme1* et *arme2* sont les noms des deux armes qui se confrontent.

La fonction retourne :

- a) 0, dans le cas d'une égalité ;
- b) 1, si la première arme spécifiée (*arme1*) a l'ascendant sur la seconde (*arme2*) ;
- c) -1, si la seconde arme spécifiée (*arme2*) a l'ascendant sur la première (*arme1*).

#### Méthodologie conseillée pour la solution :

En observant attentivement la logique du jeu « Pierre-Feuille-Ciseaux-Lézard-Spock » illustrée dans la **figure 2** située au début de l'énoncé, il devient évident que chaque arme a l'ascendant sur l'arme qui lui succède (+1) dans le cercle, en suivant le sens horloger, et sur celle qui est située 3 positions plus loin (+3) (ou 2 positions avant (-2)).

A l'aide de la fonction **TableauChaines.commencePar** et du tableau **ARMES**, il est possible d'obtenir les positions de chacune des armes spécifiées, puis d'appliquer les conditions observées ci-dessus. Toutefois, il faut veiller à faire en sorte que le 1<sup>er</sup>

*élément du tableau succède au dernier.*

Exemple : si *arme1* est la chaîne "Spock" et *arme2* la chaîne "Feuille", alors la fonction retourne -1 ("Feuille" a l'ascendant sur "Spock").

**[CONTRAINTE]** Vous devez utiliser la fonction *commencePar* de la classe **TableauChaines** pour obtenir une position aléatoire au sein du tableau.

**[PRÉCONDITION]** Les chaînes *arme1* et *arme2* sont supposées valides (ne valent pas *null*).

## LES EXEMPLES D'EXÉCUTION

### MINIMUM REQUIS

JEU DU SHIFUMI

Ciseaux, Feuille, Pierre, Léopard, Spock ? **spock**  
 spock contre Pierre  
 (C)ontinuer, (Q)uitter ? **c**

Ciseaux, Feuille, Pierre, Léopard, Spock ? **spock**  
 spock contre Léopard  
 (C)ontinuer, (Q)uitter ? **c**

Ciseaux, Feuille, Pierre, Léopard, Spock ? **feuille**  
 feuille contre Spock  
 (C)ontinuer, (Q)uitter ? **q**

### VERSION IDÉALE

JEU DU SHIFUMI

Ciseaux, Feuille, Pierre, Léopard, Spock ? **spok**  
 Ciseaux, Feuille, Pierre, Léopard, Spock ? **spock**  
 Spock casse Pierre  
 Vous 1 - 0 IA

Ciseaux, Feuille, Pierre, Léopard, Spock ? **s**  
 Spock est empoisonné par Léopard  
 Vous 1 - 1 IA

Ciseaux, Feuille, Pierre, Léopard, Spock ? **f**  
Feuille discrédite Spock  
Vous 2 - 1 IA

Ciseaux, Feuille, Pierre, Léopard, Spock ? **léz**  
Léopard mange Feuille  
Vous 3 - 1 IA

Bravo ! Vous gagnez la partie.

#### STATISTIQUES

0 fois Ciseaux  
2 fois Feuille  
1 fois Pierre  
2 fois Léopard  
3 fois Spock