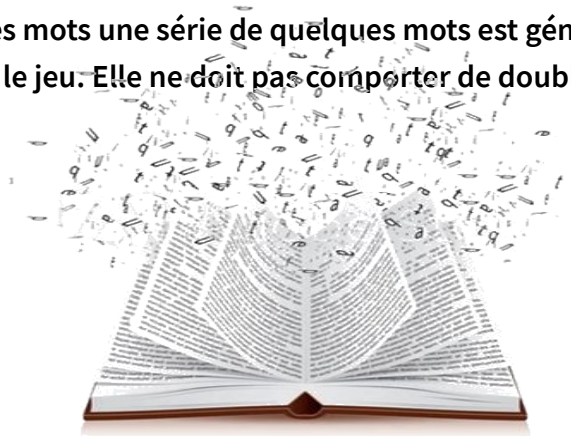
Mini-projet «Jeux de Mémoire »

1. *Objectif*

- L'objectif de ce mini-projet est la création d'un jeu permettant d'entraîner sa mémoire.
- Le jeu est composé de différents exercices de mémorisation simples.

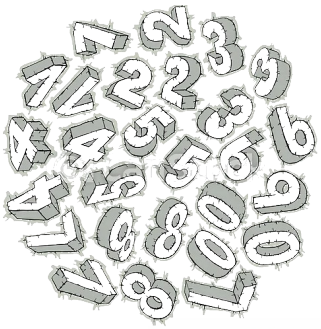
2. *Contexte*

- 3 exercices sont proposés :
  - Série de de mots
    - Règle : Une série de mots est générée aléatoirement et affichée au joueur. On lui présente ensuite le premier mot et il dispose du temps qu'il veut pour mémoriser. Quand il est prêt il le restitue. Puis on lui présente les 2 premiers mots qu'il doit restituer dans l'ordre après les avoir mémoriser. Puis on lui présente les 3 premiers et ainsi de suite jusqu' au bout de la série.
    - Génération des mots : les mots proposés au joueur sont issu d'un texte (Voir *Outils*). Ils en seront extraits, triés et nettoyés pour enlever les ponctuations et les doublons.
      A partir de ces mots une série de quelques mots est générée pour être utilisée dans le jeu. Elle ne doit pas comporter de doublons.



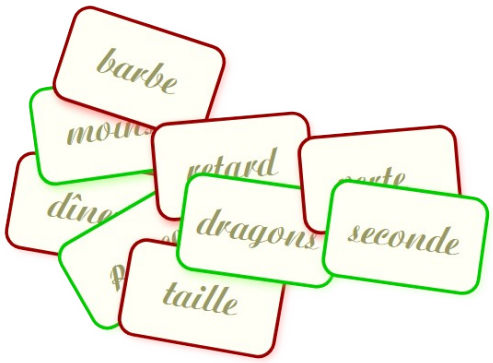
    - Difficulté de jeux : Le joueur doit pouvoir choisir la difficulté de jeux qui dépend de :
      - Nombre de mots
      - Longueur minimum des mots
      - Longueur maximum des mots

- **Série de nombres**



  - Règle : même règle que la liste de mots avec des nombres au lieu des mots
  - Génération des nomres: Une série de nombres est générées aléatoirement (Voir *Outils*). Elle ne comporte pas de doublons.
  - Difficulté de jeux : Le joueur doit pouvoir choisir la difficulté de jeux qui dépend de :
    - Nombre de nombres
    - Longueur minimum des nombres (nombres de chiffres)
    - Longueur maximum des nombres (nombre de chiffres)

- **Liste de paires de mots**



  - Règle : Une série de paires mots est générée aléatoirement et affichée au joueur. Celui-ci dispose du temps qu'il veut pour la mémoriser. Ensuite, quand il est prêt, on lui propose aléatoirement et successivement chaque premier mot des paires et il doit donner le mot correspondant. Tous les premiers mots sont présenté et ils ne le sont qu'une fois.
  - Génération des mots : les mots proposés au joueur sont issus de la même source que pour la série de mots. Une série de paires de mots est générée à partir de cette source. Un même mot ne peut s'y trouver qu'une fois quelque soit sa position dans la paire.
  - Difficulté de jeux : Le joueur doit pouvoir choisir la difficulté de jeu qui dépend de :
    - Nombre de mots
    - Longueur minimum des mots
    - Longueur maximum des mots

  - Options de jeux : 2 autres sous-jeux sont possibles en fonction de l'ordre de présentation des mots de la paire
    - Inverse : présenter le second mot de la paire et demander la restitution du premier par le joueur.

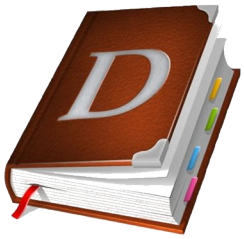
- **Aléatoire : Présenter aléatoirement soit le premier mot de la paire soit le second mot de la paire et demander la restitution de l'autre.**

- **Règles générales**
  - **Dans tous les jeux, dès que joueur se trompe je jeux s'arrête.**
  - **Quand le jeu s'arrête :**
    - **On indique le niveau atteint par le joueur (Quantité de mots ou de nombres restitués correctement).**
    - **On propose au joueur de recommencer le même jeu :**
      - **Avec les mêmes données**
      - **Avec d'autres données**

3. *Résultat attendu*

- **Vous devrez écrire un programme Java permettant de proposer au joueur les 3 exercices de mémoire décrits.**
  Votre programme devra comporter les fonctionnalités suivantes :

1. **Création de la base de mots**
   - ***Méthode* : la base de mots sera créée à partir d'un texte contenu dans un fichier texte.**



- **Un texte extrait de « Bilbo le Hobbit » vous est proposé, vous pouvez utiliser un autre texte si vous le souhaitez.**
- **Une classe « Lecteur » vous est fournie. Celle-ci permet de lire le contenu d'un fichier texte et de l'obtenir dans une variable de type String. (*Voir Outils*)**
- **Quelques méthodes Java de la classe String pouvant vous être utiles sont expliquées dans la partie *outils*.**

   - **A partir de cela vous enregistrerez les mots dans une structure de votre choix utilisable dans la suite du programme.**

2. **Niveaux de difficulté**
   - **Chaque jeu pourra se jouer à plusieurs niveaux de difficulté à vous de décider combien (au moins 3).**
   - **Le niveau de difficulté change le nombre de mots des séries, la longueur minimale et la longueur maximale des mots/nombre. Par Exemple :**
     - **Niveau 1 : 10 mots de longueur 3 à 5**
     - **Niveau 2 : 20 mots de longueur 4 à 7**
     - **Niveau 3 : 30 mots de longueur 5 à 10**

Vous pouvez créer autant de niveau que vous souhaitez en jouant sur ces 3 paramètres.

-

3. Dynamique générale du programme
   - Au démarrage vous proposerez au joeur le choix d'un jeu parmi les 3 jeux.
   - A la fin de chaque jeu vous lui proposerez de recommencer le même jeu (avec les mêmes données ou non) . S'il refuse il revient au départ où il peut encore choisir de jouer à l'un des 3 jeux
   - Le joueur pourra jouer plusieurs fois au même jeu ou à des jeux différents. Tant qu'il ne sort pas du programme, les résultat de ces jeux seront conservés.
   - Lorsque le joueur quitte le programme vous lui affichez les statistiques de ses parties :
     - Nombre de parties jouées pour chaque jeu par niveau difficulté,
     - Meilleur résultat pour chacun des jeux joués par niveau de difficulté (Plus haut niveau atteint)
     - Moyenne des résultats pour chacun des jeux joués par niveau de difficulté.

4. *Outils*

1. Simuler le hasard

- Vous aurez besoin d'introduire du *hasard* pour générer vos séries. Pour ce faire vous pourrez utiliser la méthode Math.random() disponible en Java

```
public static double random()

Renvoie une valeur double avec un signe positif, supérieur
ou égal à 0,0 et inférieur à 1,0. Les valeurs renvoyées
sont choisies de façon pseudo-aléatoire avec une
distribution (approximativement) uniforme à partir de cette
plage.
```

- Voici la méthode genererUnEntier() qui utilise la méthode Math.random() pour générer pseudo-aléatoirement un entier compris entre 1 et un maximum. Vous pouvez l'utiliser directement ou l'adapter dans votre programme :

```
public static int genererUnEntier(int max){

        return (int) (Math.random() * max + 1);
    }
```

2. Lire un fichier texte

- La classe Lecteur vous est donnée sous forme d'un fichier Lecteur.class. Vous pouvez l'utiliser pour lire un fichier texte et le récupérer dans une variable de type String, par exemple :

```
try {
    String texte = Lecteur.LireTexte("C:\\Document\\Montexte.txt");
    System.out.println(texte);
} catch (IOException ex) {
    System.out.println("Erreur de lecture du fichier");
    return "";
}
```

Attention la méthode Lecteur.LireTexte() est susceptible de lever l'exception IOException.

- Pour utiliser la méthode Lecteur.LireTexte() vous devez placer le fichier Lecteur.class dans le même répertoire que votre programme java.
- Le fichier texte utilisé doit être au encodé en UTF-8.

3. Traiter les chaînes de caractères

- Voici quelques méthodes qui pourrons vous être utiles pour manipuler les chaînes de caractères :
  - toUpperCase() : retourne la même chaîne en majuscules

```
String chaine;
chaine ="En Majuscule";
System.out.println(chaine.toUpperCase());
```

⇒ `EN MAJUSCULE`

  - toLowerCase() : retourne la même chaîne en minuscules

```
chaine ="En Minuscule";
System.out.println(chaine.toLowerCase());
```

⇒ `en minuscule`

- **endsWith()** : vérifie si une chaîne se termine par une autre chaîne

```
chaine= "Mot de la fin";
//La chaine "Mot de la fin" se termine par la chaine "fin"
System.out.println(chaine.endsWith("fin"));
//La chaine "Mot de la fin" ne se termine pas par la chaine "FIN"
System.out.println(chaine.endsWith("FIN"));
```

⇒
```
true
false
```

- **stripLeading()** : retourne une chaîne en supprimant les espaces de début

```
chaine ="  avec des espaces   ";
System.out.println(chaine.stripLeading()+".");
```

⇒
```
avec des espaces   .
```

- **stripTrailing()** : retourne une chaîne en supprimant les espaces de fin

```
chaine ="  avec des espaces   ";
System.out.println(chaine.stripTrailing()+".");
```

⇒
```
  avec des espaces.
```

- **strip()** : retourne une chaîne en supprimant les espaces de début et de fin

```
chaine ="  avec des espaces   ";
System.out.println(chaine.strip()+".");
```

⇒
```
avec des espaces.
```

- **replace()** : remplace dans une chaîne une suite de caractères par une autre

```
chaine ="  avec des espaces   ";
//Remplace  les espaces par des points
System.out.println(chaine.replace(" ","."));
//Supprime les espaces en les remplaçant par une chaîne vide
chaine ="  avec des espaces   ";
System.out.println(chaine.replace(" ",""));
```

⇒
```
..avec.des.espaces...
avecdesespaces
```

- **substring() : Extrait une partie la chaîne ⇒ à partir d'une position jusqu'à la fin ou entre 2 positions**

```
chaine ="Chaine entière";
System.out.println(chaine.substring(7)); //à partir du 8ème caractère
System.out.println(chaine.substring(7,10)); // du 8ème au 10ème caractère
```

⇒ `entière`
`ent`

5. *Éléments de méthode*

- **Pour traiter ce problème vous pouvez le décomposer en vous posant les questions suivantes :**
  - ***De quelles fonctionnalités générales le programme doit-il disposer ?***
    - **Exemples :**
      - **Lire un texte**
      - **Extraire et stocker les mots du texte**
      - **Générer une série de mots aléatoirement à partir des mots du texte**
      - **Choisir un jeu**
      - **Jouer à un jeu**
      - **Etc ...**
  - ***Quelles actions élémentaires le programme doit-il être capable de réaliser afin de disposer des fonctionnalités identifiées ?***
    - **Exemples :**
      - **Supprimer du texte les caractères de ponctuation et autres symboles indésirables**
      - **Supprimer de la liste des mots certains mots indésirables**
      - **Demander une confirmation à l'utilisateur**
      - **Demander un choix à l'utilisateur**
      - **Générer aléatoirement des séries**
      - **Etc ...**
  - ***De quelles données le programme a-t-il besoin pour mettre en œuvre les fonctionnalités ?***
    - **La nature de ces données ?**
    - **Quels sont les types de ces données ?**
    - **Comment organiser ces données (tableau ou non) ?**
    - **Ces données doivent-elles être conservées durant toute l'exécution ou sont-elles seulement nécessaires le temps de certains traitements ?**
    - **Comment le programme obtient-il ces données : saisie, constantes, génération aléatoire … ?**

- Synthétiser le problème
  - Vous avez les outils pour traduire le problème sous forme d'algorithmes en français basé sur un enchaînement d'actions élémentaires.

- Traduisez ces actions élémentaires en code informatique
  - Beaucoup de ces actions élémentaires pourront (devront) se traduire par l'écriture de méthodes.
  - Peut-être certaines de ces actions se ressemblent-elles et peuvent donner lieu à l'écriture d'une méthode à usage général.
  - Une méthode sert à écrire une partie de programme qui pourra être appelée à plusieurs reprise dans le programme. Une méthode peut aussi servir à découper des traitements longs ou complexes en des traitements élémentaire plus simples.
    ⇒ Vous serez peut-être amené à simplifier vos méthodes en écrivant des « sous-méthodes ».

6. *Consignes générales*

- Faites simple : si un code est trop compliqué, simplifiez-le en le découpant (en méthodes par exemple). Remplacez les if … else par des switch quand c'est possible et pertinent, utilisez des tableaux quand cela est nécessaire.
- Accompagnez votre utilisateur : affichez des messages clairs. Contrôlez les saisies pour éviter les valeurs erronées.
- Soignez la présentation des affichages : Passez des lignes quand c'est nécessaire, introduisez des séparateurs, mettez des majuscules aux initiales des mots et surtout faites attention à l'orthographe.
- Écrivez un code lisible : Nommez vos variables intelligemment, respecter les conventions de nommage, commentez, indentez et aérez !
  Supprimez de votre programme final tout ce qui ne sert à rien :
  - Morceaux de code mis en commentaire lors de la mise au point,
  - Affichage d'informations utilisées pour la mise au point ,
  - Etc …
- Les commentaires doivent notamment servir à :
  - Identifier votre source : *Contexte, Nom ou Objet du programme, Auteur* et toute information pertinente qu'il vous paraît utile d'ajouter.
  - Expliquer le rôle et l'utilité de chaque méthode et la signification de ses paramètres et de sa valeur de retour,
  - Expliquer à quoi servent les variables que vous déclarez,

- Expliquer l'utilité des différentes partie d'une méthode (Boucles, instructions conditionnelles, calculs …)

- Testez et fiabilisez votre code :
  - Assurez -vous que chacune de vos méthodes fonctionne indépendamment du reste du code.
  - Si certaines parties de votre programme ne fonctionnent pas ou ne sont pas terminées au moment de le rendre, signalez le dans le dossier de programmation.
    Dans ce cas arrangez-vous pour rendre un source compilable sans erreur, par exemple en laissant le corps d'une méthode vide, en renvoyant une constante, en commentant les parties contenant des erreurs ...

7. *Attendus*

- Vous fournirez l'ensemble du code source produit.
- Vous accompagnerez votre code source d'un dossier de programmation succinct [1] dans lequel vous détaillerez notamment :
  - La méthode (au sens façon de faire) générale utilisée,
  - Les choix des types de données pour les données principales,
  - Pour chaque méthode (au sens sous-programme), son utilité et son fonctionnement (Cela ne vous empêche pas de commenter le code).
  - Optionnellement vous pouvez joindre quelques exemples de résultats d'exécution.

  Remarque : Le dossier de programmation doit être réalisé au moyen d'un traitement de texte. Soignez sa présentation.

1 *Au maximum 4 pages sans les éventuels exemples*