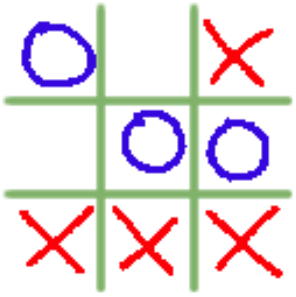
Examen final

Durée de l'épreuve : 2 h 30

Tout document interdit, de même que l'usage de la calculatrice.

Contexte :



N.B. Le texte qui suit ainsi que l'image sont extraits de Wikipedia (`http://fr.wikipedia.org/wiki/Tic-tac-toe`)

Le Tic-tac-toe se joue sur une grille carrée de 3x3 cases. Deux joueurs s'affrontent. Ils doivent remplir chacun à leur tour une case de la grille avec le symbole qui leur est attribué : O ou X. Le gagnant est celui qui arrive à aligner trois symboles identiques, horizontalement, verticalement ou en diagonale.

Etape 1 : Modélisation objet

Dans un premier temps vous réaliserez le jeu du *Tic-tac-toe* de manière simple :

- Une classe `Coordinates` représentant un couple de coordonnées (non modifiable une fois construit)
- Une classe `TicTacToe` représentant une partie de *Tic-tac-toe*.

  Vous n'y ferez pas intervenir la notion de joueurs, ceux-ci seront simulés par un tirage aléatoire de coordonnées où jouer à chaque tour. Vous afficherez à chaque tour sur la console quel joueur a joué (et où) ainsi que l'état final de la grille (en ASCII-art). Vous afficherez également sur la console le résultat à la fin de la partie (si un joueur a gagné, et lequel, ou si la partie est nulle).

  Vous isolerez de la boucle principale du jeu les méthodes :

  - `setMarkAt` affectant un pion à une case (et soulevant une exception si les coordonnées sont invalides)
  - `checkCoordinates` testant si des coordonnées sont bien sur la grille
  - `checkVictory` retournant un booléen indiquant si la partie est gagnée
- Une classe `Main` créant une nouvelle partie et la démarrant.

Un exemple de sortie console durant la partie (à respecter) est donné en annexe 1.

Les méthodes suivantes pourront être utiles :

- Classe `Random` du paquetage `java.util`
  - `public Random()`
    - crée une nouvelle instance d'un générateur de nombres aléatoires
  - `public int nextInt(int i)`
    - renvoie un nombre entier choisi aléatoirement et compris entre 0 (inclus) et `i` (exclu)
- Classe `String` du paquetage `java.lang`
  - `public String replace(String s, String d)`
    - retourne une chaîne de caractères obtenue par remplacement de toutes les occurrences de la chaîne `s` par la chaîne `d`

Etape 2 : Interfaces

Dans un deuxième temps, vous travaillerez à extraire de la boucle principale du jeu l'obtention du choix du joueur (coordonnées) et l'affichage des éléments du jeu.

Vous considèrerez donc que les joueurs ne sont plus simulés mais qu'à chaque joueur (classe `Player`) est désormais associé :

- un mécanisme permettant d'obtenir son choix à chaque tour (i.e. les coordonnées où il souhaite jouer)
- un mécanisme permettant de lui afficher les éléments du jeu (grille, résumé du tour, messages divers)

Ces mécanismes seront abstraits par des interfaces respectivement nommées `DecisionMaker` et `Display` et implémentées :

- pour `Display` :
  - sous la forme d'un affichage console (`ConsoleDisplay`)
  - sous la forme d'un affichage vide, où rien ne s'affiche nul part (`VoidDisplay`)
- pour `DecisionMaker` :
  - sous la forme d'un tirage aléatoire (`RandomDecisionMaker`)
  - via le clavier (`ConsoleDecisionMaker`), où l'utilisateur renseigne les coordonnées sur une seule ligne de texte sous la forme `x-y` (ex. `3-4`)

Les éléments du jeu (grille et résumé du tour) seront transmis au mécanisme d'affichage via une seule méthode nommée `displayTurn`. Le résumé du tour est composé du numéro du joueur ayant joué, des coordonnées où il a joué, et du résultat du tour (partie non terminée, partie gagnée, partie nulle), l'ensemble étant rassemblé dans une instance de la classe `TurnInfo`. Une méthode d'affichage de message doit également être prévue, notamment pour prévenir l'utilisateur qu'il doit jouer.

Vous ne réécrirez pas entièrement le code des méthodes de la classe `TicTacToe` lorsqu'elles sont strictement identiques à celles de l'étape 1, vous vous contenterez simplement de reporter leur signature et la mention « identique à l'étape 1 » entre les accolades. Si vous jugez pertinent de changer la signature de certaines méthodes de cette classe, vous êtes libres de le faire.

Vous considérerez également que les références des mécanismes concrets d'affichage et de décision sont initialisés par l'application (`Main`), de même que les joueurs. Vous réécrirez donc entièrement `Main`, avec comme hypothèse que le joueur 1 est caractérisé par un tirage aléatoire et un affichage vide, et que le joueur 2 est caractérisé par un choix obtenu au clavier et un affichage console.

Enfin, vous considérerez que le mécanisme d'obtention du choix du joueur renverra `null` en cas d'erreur, et vous en tiendrez compte dans la boucle de jeu.

Un exemple de sortie console durant la partie (à respecter) est donné en annexe 2.

Les méthodes suivantes pourront être utiles :

- Classe `Integer` du paquetage `java.lang`
  - `public static int parseInt(String s)`
    - retourne l'entier représenté par la chaîne de caractères `s`. Soulève une exception non contrôlée, sous-classe de `RuntimeException`, si `s` ne représente pas un entier
- Classe `String` du paquetage `java.lang`
  - `public int indexOf(String s)`
    - retourne l'indice de la première occurrence de la sous-chaîne `s`, `-1` si la sous-chaîne n'est pas présente
  - `public String substring(int s, int e)`
    - retourne la sous-chaîne contenant les caractères situés entre l'indice `s` inclus et `e` exclu. Soulève une exception non contrôlée, sous-classe de `RuntimeException`, si les indices ne sont pas valides
  - `public String substring(int s)`
    - retourne la sous-chaîne contenant les caractères situés à partir de l'indice `s` inclus. Soulève une exception non contrôlée, sous-classe de `RuntimeException`, si l'indice n'est pas valide
- Classe `BufferedReader` du paquetage `java.io`
  - `public BufferedReader(Reader r)`
    - crée une instance d'un flux permettant de lire du texte en s'appuyant sur le flux de lecture de caractères `r`
  - `public String readLine()`
    - lit et retourne (sans le caractère de fin de ligne) une ligne de texte. Soulève une exception contrôlée de type `IOException` si la lecture échoue, retourne `null` s'il n'y a plus de caractère à lire
- Classe `InputStreamReader` du paquetage `java.io`
  - `public InputStreamReader(InputStream i)`
    - crée une instance d'un flux permettant de lire des caractères en s'appuyant sur le flux de lecture binaire `i`

Etape 3 : Héritage

Réalisez maintenant une classe *TicTacToe4* où la taille de la grille passe à 4x4 et où il faut aligner 4 pions pour gagner.

Si vous jugez pertinent de changer la signature de certaines méthodes ou la déclaration de certains attributs de la classe TicTacToe*, vous êtes libres de le faire. Vous ferez en sorte de pouvoir réutiliser entièrement la boucle principale de la classe* `TicTacToe`*. Vous pouvez modifier l'implémentation écrite à l'étape précédente, mais il vous est interdit d'y ajouter des attributs (il suffit de vous rappeler que* `t.length` *donne le nombre de cases du tableau* `t`*).*

Vous réécrirez `RandomDecisionMaker` *de sorte que cette même classe puisse être indépendamment utilisée avec la version 3x3 ou 4x4 du jeu.*

Vous pourrez décliner l'affichage console en une version 4x4 que vous appellerez `ConsoleDisplay4` *(si vous jugez pertinent de changer l'implémentation de* `ConsoleDisplay`*, vous êtes libre de le faire).*

Enfin, vous réécrirez `Main` *de sorte de faire jouer les mêmes joueurs que précédemment mais sur une grille 4x4.*

Un exemple de sortie console durant la partie (à respecter) est donné dans l'annexe 3.

Remarque :

Si à l'étape 1 vous aviez par excès de zèle rendu les implémentations de `checkCoordinates` *et* `checkVictory` *indépendantes de la taille de la grille, faites la déclinaison différemment :*

- *Déportez tout le code commun aux versions 3 et 4 cases dans une classe* `CommonTicTacToe`
- *Réimplémentez* `TicTacToe` *et implémentez* `TicTacToe4` *de sorte que ces classes ne contiennent que ce qui est spécifique et en considérant que leurs constructeurs ne prennent pas en paramètre la taille de la grille*

Etape 4 : Collections, entrées/sorties

Vous devez maintenant réaliser un pseudo-affichage stockant l'historique de la partie sous la forme :

- d'une collection (`TurnInfoCollectionDisplay`), où chaque tour est représenté par un objet `TurnInfo`.
  - cette classe fournira une méthode `getTurns` permettant de récupérer la collection une fois la partie terminée
- d'un fichier texte (`TurnInfoFileDisplay`), où chaque tour est représenté par une ligne de texte contenant le numéro du joueur, les coordonnées où il a joué et le résultat à la fin du tour (`NF` lorsque la partie continue, `W` lorsque la partie est gagnée, `D` lorsque la partie est nulle). Un exemple de sortie (à respecter) est donné dans l'annexe 4.
  - le constructeur, prenant en paramètre le chemin d'accès au fichier, pourra propager `IOException` si le fichier ne peut être créé.
  - le fichier destination sera ouvert/écrit/refermé à chaque tour, vous ne chercherez pas à manifester d'erreur si l'écriture échoue.

Les méthodes suivantes pourront être utiles :

- Classe `File` du paquetage `java.io`
  - `public File(String path)`
    - Crée une représentation du chemin d'accès contenu dans `path`
  - `public boolean createNewFile()`
    - Crée un fichier, retourne `true` si le fichier n'existait pas auparavant, `false` sinon. Soulève `IOException`, si le fichier n'existait pas et n'a pas pu être créé
- Classe `FileOutputStream` du paquetage `java.io`
  - `public FileOutputStream(File f, boolean append)`
    - Crée un flux d'écriture binaire dans le fichier dont le chemin d'accès est représenté par `f`. Le booléen `append` permet de savoir si on souhaite ajouter en fin de fichier (`true`) ou écraser le contenu (`false`). Soulève `FileNotFoundException` si le fichier est introuvable.
- Classe `PrintStream` du paquetage `java.io`
  - `public PrintStream(OutputStream o)`
    - Crée une instance d'un flux permettant d'écrire du texte en s'appuyant sur le flux d'écriture binaire `o`
  - `public void println(String s)`
    - écrit la chaîne de caractères `s` et passe à la ligne

Vous écrirez également deux classes permettant de rejouer des parties :

- `TurnInfoCollectionDecisionMaker`, rejouant une collection de tours passée en paramètre
- `TurnInfoFileDecisionMaker`, rejouant un fichier de tour passé en paramètre
  - Le constructeur, ouvrant le fichier, pourra propager `FileNotFoundException`

– Vous ne vous soucierez pas de la fermeture du fichier (sauf si vous trouvez la seule façon de le faire proprement au ramassage de l'objet)

Les méthodes suivantes pourront être utiles :

– Classe `FileInputStream` du paquetage `java.io`
  – `public FileInputStream(File f)`
    – Crée un flux de lecture binaire dans le fichier dont le chemin d'accès est représenté par `f`. Soulève `FileNotFoundException` si le fichier est introuvable.

Vous écrirez également deux applications :

– `MainCollectionOutCollectionIn` qui
  – joue une première partie sur une grille 4x4 avec
    – un joueur 1 utilisant `RandomDecisionMaker` et `TurnInfoCollectionDisplay`
    – un joueur 2 utilisant `ConsoleDecisionMaker` et `ConsoleDisplay4`
  – rejoue la première partie avec
    – un joueur 1 utilisant `VoidDisplay`
    – un joueur 2 utilisant `ConsoleDisplay4`
– `MainFileOutFileIn` qui
  – joue une première partie sur une grille 4x4 avec
    – un joueur 1 utilisant `RandomDecisionMaker` et `TurnInfoFileDisplay`
    – un joueur 2 utilisant `ConsoleDecisionMaker` et `ConsoleDisplay4`
  – rejoue la première partie avec
    – un joueur 1 utilisant `VoidDisplay`
    – un joueur 2 utilisant `ConsoleDisplay4`

N.B Puisque les collections ou le fichier contiennent les choix des deux joueurs, les références du mécanisme d'obtention de choix de chacun des joueurs devront donc pointer sur la même instance.

Annexe 1 : exemple de sortie à l'étape 1 :

```
Player 1 places his mark on (2,2)
 --- --- ---
|   |   |   |
 --- --- ---
|   | X |   |
 --- --- ---
|   |   |   |
 --- --- ---

Player 2 places his mark on (2,1)
 --- --- ---
|   |   |   |
 --- --- ---
| O | X |   |
 --- --- ---
|   |   |   |
 --- --- ---

Player 1 places his mark on (1,2)
 --- --- ---
|   | X |   |
 --- --- ---
| O | X |   |
 --- --- ---
|   |   |   |
 --- --- ---

Player 2 places his mark on (3,2)
 --- --- ---
|   | X |   |
 --- --- ---
| O | X |   |
 --- --- ---
|   | O |   |
 --- --- ---

Player 1 places his mark on (2,3)
 --- --- ---
|   | X |   |
 --- --- ---
| O | X | X |
 --- --- ---
|   | O |   |
 --- --- ---

Player 2 places his mark on (1,3)
 --- --- ---
|   | X | O |
 --- --- ---
| O | X | X |
 --- --- ---
|   | O |   |
 --- --- ---

Player 1 places his mark on (1,1)
 --- --- ---
| X | X | O |
 --- --- ---
| O | X | X |
```

```
 --- --- ---
|   | O |   |
 --- --- ---

Player 2 places his mark on (3,3)
 --- --- ---
| X | X | O |
 --- --- ---
| O | X | X |
 --- --- ---
|   | O | O |
 --- --- ---

Player 1 places his mark on (3,1)
 --- --- ---
| X | X | O |
 --- --- ---
| O | X | X |
 --- --- ---
| X | O | O |
 --- --- ---

Draw game
```

Annexe 2 : exemple de sortie à l'étape 2 :

```
Player 1 places his mark on (1,1)
 --- --- ---
| X |   |   |
 --- --- ---
|   |   |   |
 --- --- ---
|   |   |   |
 --- --- ---

Player 2 turn
Coordinates? (eg: 1-1)
4-3
Invalid coordinates, retry
Coordinates? (eg: 1-1)
1-1
Invalid coordinates, retry
Coordinates? (eg: 1-1)
2-1
Player 2 places his mark on (2,1)
 --- --- ---
| X |   |   |
 --- --- ---
| O |   |   |
 --- --- ---
|   |   |   |
 --- --- ---

Player 1 places his mark on (2,2)
 --- --- ---
| X |   |   |
 --- --- ---
| O | X |   |
 --- --- ---
|   |   |   |
 --- --- ---

Player 2 turn
Coordinates? (eg: 1-1)
1-3
Player 2 places his mark on (1,3)
 --- --- ---
| X |   | O |
 --- --- ---
| O | X |   |
 --- --- ---
|   |   |   |
 --- --- ---

Player 1 places his mark on (2,3)
 --- --- ---
| X |   | O |
 --- --- ---
| O | X | X |
 --- --- ---
|   |   |   |
 --- --- ---

Player 2 turn
Coordinates? (eg: 1-1)
```

```
3-3
Player 2 places his mark on (3,3)
 --- --- ---
| X |   | O |
 --- --- ---
| O | X | X |
 --- --- ---
|   |   | O |
 --- --- ---

Player 1 places his mark on (3,2)
 --- --- ---
| X |   | O |
 --- --- ---
| O | X | X |
 --- --- ---
|   | X | O |
 --- --- ---

Player 2 turn
Coordinates? (eg: 1-1)
1-2
Player 2 places his mark on (1,2)
 --- --- ---
| X | O | O |
 --- --- ---
| O | X | X |
 --- --- ---
|   | X | O |
 --- --- ---

Player 1 places his mark on (3,1)
 --- --- ---
| X | O | O |
 --- --- ---
| O | X | X |
 --- --- ---
| X | X | O |
 --- --- ---

Draw game
```

Annexe 3 : exemple de sortie à l'étape 3 :

```
Player 1 places his mark on (1,2)
 --- --- --- ---
|   | X |   |   |
 --- --- --- ---
|   |   |   |   |
 --- --- --- ---
|   |   |   |   |
 --- --- --- ---
|   |   |   |   |
 --- --- --- ---

Player 2 turn
Coordinates? (eg: 1-1)
2-1
Player 2 places his mark on (2,1)
 --- --- --- ---
|   | X |   |   |
 --- --- --- ---
| O |   |   |   |
 --- --- --- ---
|   |   |   |   |
 --- --- --- ---
|   |   |   |   |
 --- --- --- ---

Player 1 places his mark on (3,2)
 --- --- --- ---
|   | X |   |   |
 --- --- --- ---
| O |   |   |   |
 --- --- --- ---
|   | X |   |   |
 --- --- --- ---
|   |   |   |   |
 --- --- --- ---

Player 2 turn
Coordinates? (eg: 1-1)
2-2
Player 2 places his mark on (2,2)
 --- --- --- ---
|   | X |   |   |
 --- --- --- ---
| O | O |   |   |
 --- --- --- ---
|   | X |   |   |
 --- --- --- ---
|   |   |   |   |
 --- --- --- ---

Player 1 places his mark on (1,4)
 --- --- --- ---
|   | X |   | X |
 --- --- --- ---
| O | O |   |   |
 --- --- --- ---
|   | X |   |   |
 --- --- --- ---
|   |   |   |   |
```

```
 --- --- --- ---

Player 2 turn
Coordinates? (eg: 1-1)
2-3
Player 2 places his mark on (2,3)
 --- --- --- ---
|   | X |   | X |
 --- --- --- ---
| O | O | O |   |
 --- --- --- ---
|   | X |   |   |
 --- --- --- ---
|   |   |   |   |
 --- --- --- ---

Player 1 places his mark on (4,1)
 --- --- --- ---
|   | X |   | X |
 --- --- --- ---
| O | O | O |   |
 --- --- --- ---
|   | X |   |   |
 --- --- --- ---
| X |   |   |   |
 --- --- --- ---

Player 2 turn
Coordinates? (eg: 1-1)
2-4
Player 2 places his mark on (2,4)
 --- --- --- ---
|   | X |   | X |
 --- --- --- ---
| O | O | O | O |
 --- --- --- ---
|   | X |   |   |
 --- --- --- ---
| X |   |   |   |
 --- --- --- ---

Player 2 wins
```

Annexe 4 : exemple de sortie fichier à l'étape 4 :

```
1 1-1 NF
2 2-1 NF
1 3-4 NF
2 2-2 NF
1 3-1 NF
2 2-3 NF
1 1-4 NF
2 2-4 W
```