TP R2.01 : Faire un choix de Design …

*Ce TP commence en mode TD c-a-d **en mode déconnecté** !*
*Laissez vos ordinateurs éteints pour le moment et prenez une **feuille de papier** !*

1. Phase d'analyse :

Vous connaissez le jeu de puissance 4 (**connect four**) puisque vous avez déjà travaillé au développement de ce jeu dans un projet précédent 😉

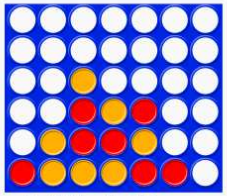
Puissance 4 (appelé aussi parfois **4 en ligne**) est un jeu de stratégie combinatoire abstrait, commercialisé pour la première fois en 1974 par la Milton Bradley Company, plus connue sous le nom de MB et détenue depuis 1984 par la société Hasbro.

Sommaire [masquer]

Règles du jeu [modifier | modifier le code]

Le but du jeu est d'aligner une suite de 4 pions de même couleur sur une grille comptant 6 rangées et 7 colonnes. Chaque joueur dispose de 21 pions d'une couleur (par convention, en général jaune ou rouge). Tour à tour, les deux joueurs placent un pion dans la colonne de leur choix, le pion coulisse alors jusqu'à la position la plus basse possible dans la dite colonne à la suite de quoi c'est à l'adversaire de jouer. Le vainqueur est le joueur qui réalise le premier un alignement (horizontal, vertical ou diagonal) consécutif d'au moins quatre pions de sa couleur. Si, alors que toutes les cases de la grille de jeu sont remplies, aucun des deux joueurs n'a réalisé un tel alignement, la partie est déclarée nulle.

Puissance 4
jeu de société



Un début de partie de Puissance 4.

Données clés

Auteur	Howard Wexler
Éditeur	MB / Hasbro
Date de 1re édition	1974
Formats	grande boîte jeu de voyage
Mécanisme	alignement
Joueur(s)	2
Âge	à partir de 6 ans
Durée annoncée	env. 10 minutes

Extrait Wikipédia : https://fr.wikipedia.org/wiki/Puissance_4

2. Phase de conception : focus sur le matériel

Pour pouvoir jouer, vous savez donc qu'il faut disposer du matériel suivant :

- Un « plateau de jeu » composé d'une grille de 6 rangées et 7 colonnes (**gameboard** as a seven-column, six-row vertically suspended grid)
- 42 pions de deux couleurs différentes pour que chaque joueur puisse disposer de 21 pions d'une même couleur chacun (en principe de couleur jaune et rouge : **yellow disc** and **red disc**)

⇨ Modélisez le matériel nécessaire pour pouvoir jouer au puissance 4 dans un diagramme de classes (**uniquement le matériel**, pas les joueurs, ni les règles du jeu)
Comparez vos différentes modélisations et justifiez vos différents choix de conception.
Vous devriez voir apparaître plusieurs conceptions possibles autour de la classe `Gameboard` et donnez les implémentations correspondantes. Justifier vos choix.

Passer en mode connecté, pour comparer différentes implémentations pour choisir au mieux une structure de données répondant au problème et facilement manipulable en fonction de votre expérience

Vous n'allez pas implémenter à nouveau le puissance 4, vous allez juste jouer avec différentes structures de données pour vous aider à choisir quelle structure de données vous auriez pu choisir si vous aviez dû implémenter le jeu puissance 4 avec un paradigme Orienté Objet.
Deux structures de données paraissent possibles :

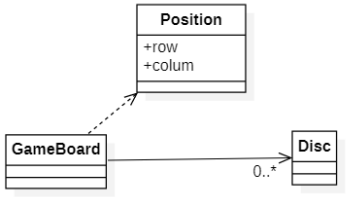
- **Un tableau 2D de `discs`** puisqu'il paraît logique de penser : « *qu'un gameboard peut contenir un ou plusieurs disc* »

⇒ ce sera la **conception choisie** pour implémenter la classe ***GameBoard2D***

Remarque : Les *lignes* et ***colonnes*** du tableau ne seront autre que la `position` du `disc`,
La classe `Position` ne sera donc utilisée que dans les paramètres d'entrée de certaines méthodes de la classe **`GameBoard`** comme par exemple :

```
public boolean isDiscAt(Position position)
public boolean put(Position position, Disc disc)
```

Si la classe **`Position`** est simplement **utilisée par** la classe `GameBoard`, la relation entre la classe `GameBoard` et la classe **`Position`** est **une relation de dépendance (flèche ouverte en pointillée** dans la notation **UML**).
Il est à noter que PlantUML ne représente pas par défaut les dépendances pour ne pas alourdir la lecture 😉



Notez également que dans le cas d'une implémentation s'appuyant sur un tableau 2D, il ne serait pas forcément nécessaire de faire apparaître une classe `Position`.
Faire apparaître cette classe améliore en fait la qualité de code.

En effet, cette abstraction permet de corriger un ***smell code*** appelé **`primitive obsession`** qui apparaît lorsque tous les types des paramètres d'entrée d'une méthode sont des types primitifs… Il est alors temps de se dire qu'une petite abstraction serait nécessaire 😉

En effet, en faisant apparaître `row` et `column` comme paramètres d'entrées entiers, il est plus facile de se tromper et d'inverser par étourderie la valeur de ces paramètres lors de l'appel d'une méthode nécessitant une position. En passant directement un **objet de type `Position`**, le code passé en paramètre est « plus sûr » (dans le sens mieux contrôlé par le compilateur) et donc de meilleure qualité 😉

- **Un hashmap de `discs`** qui contiendra seulement les `discs` présents sur le `gameboard`.
Un élément de type `Hashmap` étant caractérisé par un couple clé-valeur, dans le contexte du `gameboard` la clé correspondra à la `Position` et la valeur à l'objet `Disc` posé sur cette position 😉.
Cette implémentation est souvent utilisée dans le cas d'une modélisation avec un ***qualificateur*** : *« dans le contexte d'un gameboard, pour une position donnée, il ne peut y avoir que 0 ou 1 disc sur la gameboard à cette position»* (refaites un petit tour dans le cours sur le diagramme de classes, si vous avez oublié la notion de qualificateur : notez bien que la multiplicité entre cette modélisation et la modélisation précédente a été réduite en faisant apparaître le qualificateur 😉)

⇒ ce sera la **conception choisie** pour implémenter la classe ***GameBoardHasMap***

Pour effectuer cette comparaison, vous allez récupérer sur Github le projet **`gameboard`** en suivant la procédure habituelle :

1. Récupérer le projet `gameboard` sur le remote (`Github`)

… suivre la procédure habituelle…

Sur le remote :
❑ Rendez-vous sur github à l'adresse suivante : https://github.com/iblasquez/gameboard
❑ Cliquez sur le bouton vert **Code** et copiez le lien **https** du dépôt en cliquant sur l'icône des 2 carrés à droite de l'adresse.

En local :
❑ Ouvrez **Git Bash**
❑ Placez-vous dans le **workspace `javabut1`** (les slashs des chemins sont inversés en linux et Windows)
❑ Clonez le dépôt **`gameboard`** via la **ligne de commande `git clone`** et de l'adresse **`https`** copiée.

Sous Eclipse :
❑ Lancez Eclipse avec le workspace **`javabut1`**
❑ Importez un ***projet Maven existant*** et cliquez sur **Next**.
❑ Cliquez sur **Browse** et sélectionnez le projet **`gameboard`** de manière à ce que dans le cadre **Projects** vous voyez bien un `pom.xml` Si tel est le cas, cliquez alors sur **Finish**

2. Familiariser-vous avec ce projet existant …

Pour vous familiariser avec ce projet :

❑ consultez **l'historique des versions** (**`Team→Show In History`**)
⇨ Restez uniquement sur la branche `master` pour le moment !!!

❑ procédez à une **rétro-conception du code existant** **(à l'aide de plantUML par exemple sous Eclipse) pour vous permettre de bien identifier les composants (classes) de votre système..**

❑ **promenez-vous un peu dans le code** et **déployez la vue Explorer** pour comprendre le code et son architecture. Vous constaterez qu'il y a déjà du code implémenté et qu'il reste des **`//TODO`** !
Revenons plus particulièrement sur l'architecture de ce projet et notamment sur le code de production (**`src/main/java`**) qui est actuellement décomposé en 3 paquetages :

- Le paquetage **`gameboard.model.material`** où se trouvent les classes `Color, Constants, Disc Position` que vous avez sûrement identifiées dans la première partie pour une abstraction optimale de l'application. On y trouve également l'interface `IGameBoard` qui expose tous les services qu'un *gameboard* serait susceptible de proposer pour répondre aux besoins de notre application.

  ⇒ Ces classes ont déjà été implémentées :
  vous ne devez en aucun cas pour ce TP toucher au code de ce paquetage 😉

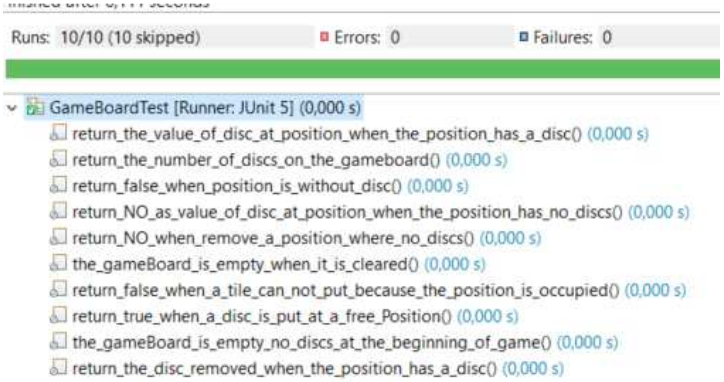
- Le paquetage **`gameboard.model.material.array2d`** qui est censé contenir l'implémentation d'un `IGameBoard` en ayant choisi comme structure de données un **tableau d'entiers 2D**. En réalité, c'est vous qui allez devoir implémenter ces service qui ne sont composés pour l'instant que de **`//TODO`** !
  Attendez un peu avant de vous lancer dans l'implémentation, nous allons vous expliquer dans quelques lignes comment procéder pas à pas, lentement mais sûrement 😉

- Le paquetage **`gameboard.model.material.hashmap`** qui est censé contenir l'implémentation d'un `IGameBoard` en ayant choisi comme structure de données une **collection de type HashMap**, avec des //TODO également 😉

❑ **lancez les jeux d'essais** : pour l'instant, il n'y a pas de paquetage `application`, donc pas de méthode `main` et pas de jeux d'essais manuels à lancer**, par contre de nombreux tests automatisés ont été écrits !!!**

❑ **consultez et lancez les tests automatisés écrits dans `GameBoardTest.`**
Que constatez-vous ?

JUnit est **VERT**, bizarre car les classes de type `IGameBoard` à tester n'ont pas été implémentées 😉



En réalité, si vous consultez plus précisément le code de test de la classe **`GameBoardTest,`** vous remarquerez que chaque test a été précédé de l'annotation **`@Disabled`**.

Exemple du premier test :

```
@Disabled("delete this annotation, check test is red and write code to pass test ! ")
@Test
void return_false_when_position_is_without_disc() {
	assertFalse(gameBoard2D.isDiscAt(FIRST_POSITION));
	assertFalse(gameBoardHashMap.isDiscAt(FIRST_POSITION));

	assertFalse(gameBoard2D.isDiscAt(_2_4));
	assertFalse(gameBoardHashMap.isDiscAt(_2_4));

	assertFalse(gameBoard2D.isDiscAt(LAST_POSITION));
	assertFalse(gameBoardHashMap.isDiscAt(LAST_POSITION));
}
```

Comme l'indique la javadoc du framework JUnit5, l'annotation **`@Disabled`** sert à désactiver un test : ***ce qui signifie que pour l'instant aucun test n'est réellement lancé.***

`@Disabled`	Used to disable a test class or test method; analogous to JUnit 4's `@Ignore`. Such annotations are not *inherited*.

Pour en savoir plus, rendez-vous sur https://junit.org/junit5/docs/current/user-guide/#writing-tests-disabling

Nous avons donc décidé d'utiliser cette annotation pour vous permettre d'implémenter les deux structures de données pas à pas, service par service. Pour cela vous devez suivre la procédure suivante…

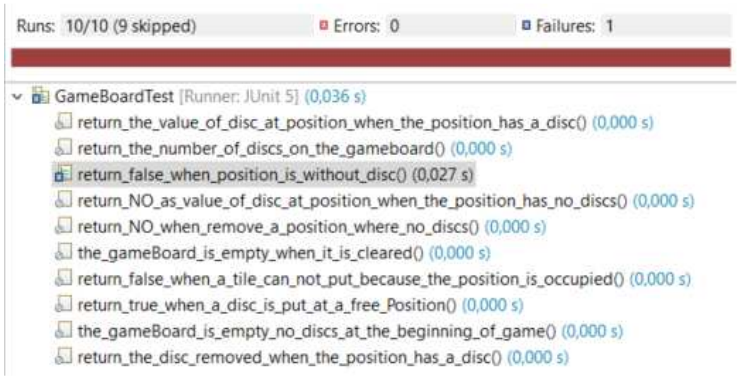
3. Implémenter pas à pas, service par service, lentement et sûrement les deux classes métiers `GameBoard2D` et `GameBoardHashMap`

Nous allons vous expliquer la procédure à suivre sur le premier test : après ce sera à vous de jouer !

1. **Effacer l'annotation `@Disabled`** au-dessus de l'annotation `@Test` du premier test.
   Il ne doit donc rester que l'annotation `@Test` au-dessus de la méthode de test.

```
@Test
void return_false_when_position_is_without_disc() {…}
```

2. **Lancer les tests (Run As → JUnit) pour vérifier que ce test échoue !**



3. Procédez à l'implémentation de la méthode **`isDiscAt`** *testée* dans ce test :
   - dans la classe **`GameBoard2D`**,
   - puis dans la classe **`GameBoardHashMap`**

   de manière que les deux implémentations de la méthode `isDiscAt` dans ces deux classes permettent à ce test passer ce test AU VERT !

4. **Une fois le test AU VERT, commitez** avec un message du genre :
   `add isDiscAt in IGameBoard hierarchy`

5. Une fois commité, vous pouvez passer au test suivant et recommencez pour ce test les étapes 1 à 4 précédentes. **Prenez bien les tests un à un dans l'ordre donné dans le fichier `GameBoardTest`** 😉

Une fois tous les tests passés au VERT, tous les services implémentés dans les deux classes GameBoard2D et GameBoardHashMap,
vous pouvez éventuellement **lancer SonarLint pour améliorer la qualité de votre implémentation !**

Quelle implémentation préférez-vous ?
Celle du *tableau 2D* ou celle de la *hashmap* ? Les deux ont des avantages et des inconvénients 😉
Si vous deviez implémenté le puissance 4 aujourd'hui, quel serait votre choix de design et pourquoi ?

4. Un affichage ascii simple pour un *gameboard*

Récupérez le fichier `GameBoardSimpleAsciiDisplayTest` sur le gist suivant https://unil.im/gbascii
Ajoutez ce fichier au code de test (**`src/test/java`**) dans le même paquetage que le fichier de tests précédent **`GameBoardTest`**

Exécutez ce nouveau fichier de test. Faîtes en sorte de faire passer le test de ce fichier au VERT.
Une fois les bonnes implémentations réalisées dans les classes `GameBoard2D` et `GameBoardHashMap`, commitez avec un message du genre :
« `add simple ascii display for gameboard` »

5. Et si vous terminiez par un petit jeu d'essai ?

Si vous consultez l'historique, vous constaterez qu'une autre équipe de développement a travaillé sur l'affichage en mode console de cette application
dans une branche **`feature-console-application`**

Merger cette branche dans votre **`master`**.
Que se passe-t-il ? Quelle est la nouvelle architecture de votre projet ?
Recherchez si une des classes mergées dispose d'un `main` et si tel est le cas lancez l'application.