

Examen pratique de janvier 2025

LE PROBLÈME DES HUIT TOURS

Il convient de préciser que ce problème n'existe pas véritablement. Il s'agit en réalité d'une simplification du **problème des huit dames**.

Le **problème des huit tours** consiste à parvenir à placer huit tours (les pions du jeu d'échecs) sur un échiquier de 8×8 cases sans qu'elles puissent se menacer mutuellement, tout en sachant qu'une tour ne peut se déplacer qu'orthogonalement (horizontalement et verticalement). Autrement dit, deux tours ne peuvent se situer sur la même rangée ou sur la même colonne (voir la **figure 1**).

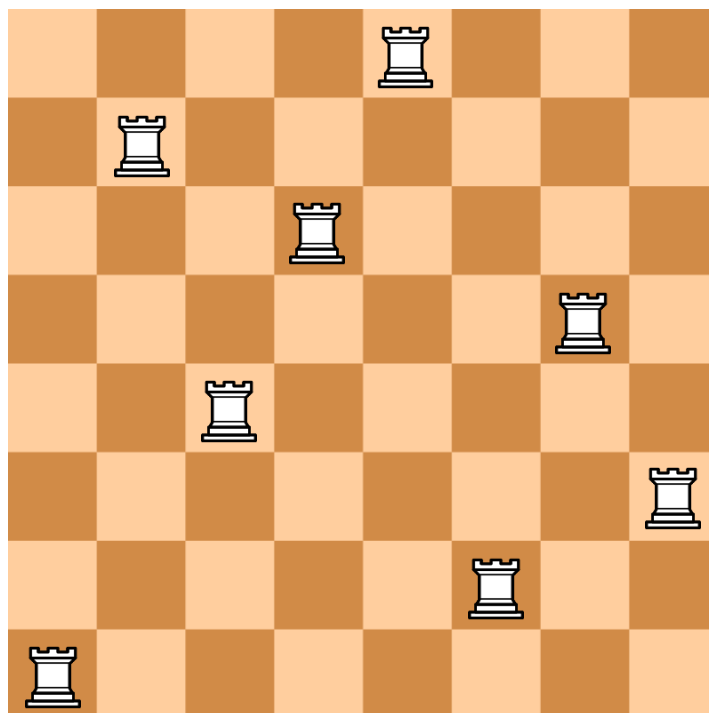


Fig. 1 – Une des solutions du problème des huit tours.

Parmi les $64!/56! = 178\,462\,987\,637\,760$ placements possibles des huit tours, il existe **1 625 702 400 solutions** au problème des huit tours.

Sur la **figure 2** de la page suivante, l'une des tours ne respecte pas les règles de placement du problème des huit tours. En effet, la tour située en haut à gauche de l'échiquier partage une ligne avec une autre tour.

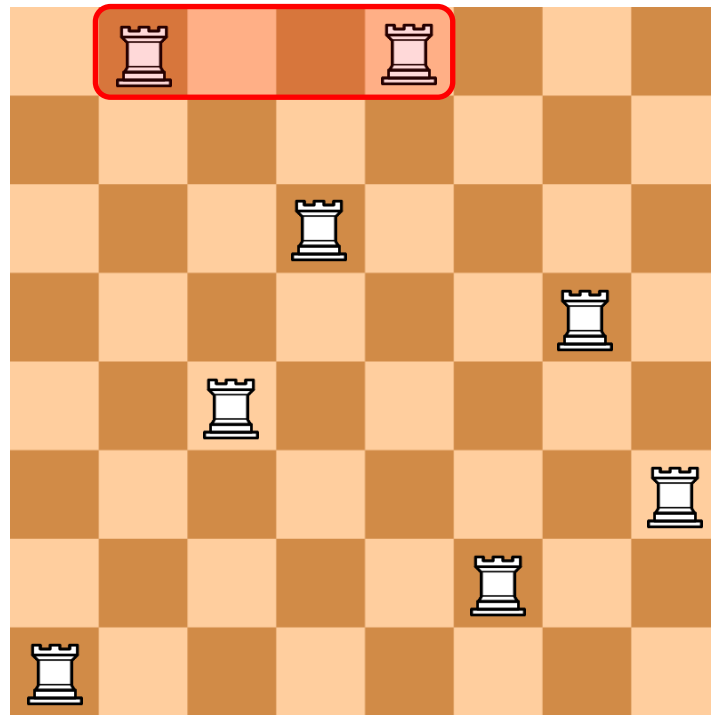


Fig. 2 – Exemple de placement des huit tours qui ne respecte pas les règles de placement.

DESCRIPTION DE L'APPLICATION

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

L'application doit permettre, dans cet ordre, les actions suivantes :

- La saisie d'un entier compris entre 2 et 8, correspondant à la taille de l'échiquier (de 2×2 cases à 8×8 cases) ainsi qu'au nombre de tours à placer (il faut, par exemple, placer 4 tours sur un échiquier de 4×4 cases) ;
- Le placement aléatoire des tours sur un échiquier de taille appropriée ;
- La vérification du respect des règles de placement des tours ;
- Les deux étapes précédentes sont répétées tant qu'une solution au problème des huit tours n'est pas obtenue. En fin d'exécution, le nombre de tentatives nécessaires est affiché.

AVANT-PROPOS

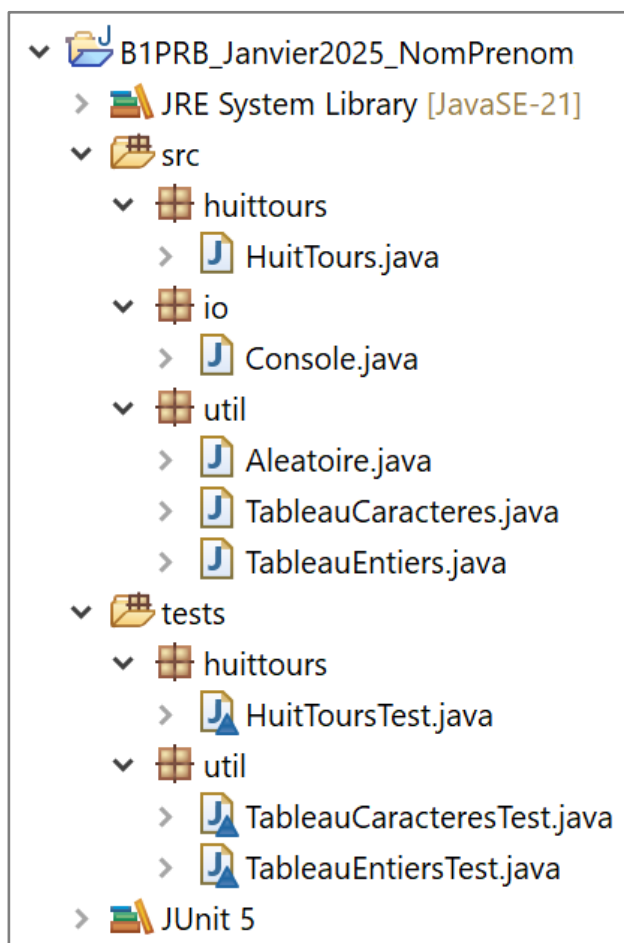
À deux exceptions près, les types `String` et `PrintStream`, vous ne pouvez pas utiliser le paradigme orienté objet pour coder cette application. N'oubliez pas de qualifier toutes vos fonctions de `static` !

PRÉPARATION

1. Dans Eclipse, créez un projet Java nommé **B1PRB_Janvier2025_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 `HuitTours`.



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 stipulées dans cet énoncé :


- La fonction `creerEchiquier` de la classe `HuitTours` doit être correctement réalisée, testée et utilisée.
- La fonction `permuter` de la classe `TableauCaracteres` doit être correctement réalisée, testée et utilisée.
- La fonction `melanger` de la classe `TableauCaracteres` doit être correctement réalisée et utilisée.  *Les tests unitaires ne sont pas ici obligatoires.*
- Dans la fonction `main` de la classe `HuitTours`, un tableau à deux dimensions de caractères doit être correctement créé et initialisé (voir la section « Les structures de données ») afin de représenter l'échiquier et le placement des tours.
- Dans sa version minimale, la fonction `main` doit effectuer la saisie de la taille de l'échiquier, la création d'un échiquier de taille appropriée avec la fonction `HuitTours.creerEchiquier` et le mélange de l'échiquier avec la fonction `TableauCaracteres.melanger` (voir l'exemple d'exécution « MINIMUM REQUIS » en fin d'énoncé).

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

REPRÉSENTER L'ÉCHIQUIER ET LE PLACEMENT DES TOURS

Utilisez un **tableau à deux dimensions de caractères** pour représenter l'échiquier. Sa taille doit correspondre à celle spécifiée par l'utilisateur au début de l'exécution.

Le tableau doit être initialisé avec des caractères espace ' ' (le symbole `␣` étant utilisé ici pour représenter un espace) afin de représenter les cases vides.

Les tours sont toutes placées sur la première rangée de l'échiquier sous la forme de lettres T majuscules 'T'.

Par exemple, si l'utilisateur spécifie une taille de 4 × 4 cases, l'échiquier obtenu est :

	0	1	2	3
0	'T'	'T'	'T'	'T'
1	' '	' '	' '	' '
2	' '	' '	' '	' '
3	' '	' '	' '	' '

*C'est la fonction **HuitTours.creerEchiquier** qui se chargera de créer l'échiquier et de l'initialiser de la sorte.*

L'ALGORITHME

La méthode permettant de vérifier si les règles de placement des tours sont respectées est décrite ci-dessous. Cette dernière doit être implémentée dans la fonction *main*.



Cette partie du programme ne fait pas partie des indispensables.

ETAPES PRINCIPALES

Pour chaque ligne de l'échiquier, obtenir l'indice de la colonne où se situe la première tour trouvée.

Trier les indices obtenus par ordre croissant.

Comparer la séquence d'indices obtenue avec celle souhaitée.

ILLUSTRATION N°1

Pour illustrer cet algorithme, nous commencerons par l'échiquier de taille 4 × 4 suivant :

	0	1	2	3
0	'T'	'T'	'T'	'T'
1	'T'	'T'	'T'	'T'
2	'T'	'T'	'T'	'T'
3	'T'	'T'	'T'	'T'

Les indices des premières tours trouvées à chaque ligne sont les suivants :

1	2	-1	0
0	1	2	3

En effet, la 1^{ère} tour de la ligne d'indice 0 est située en 1, celle de la ligne d'indice 1 est située en 2, il n'y a pas de tour présente dans la ligne d'indice 2 et, enfin, la tour de la ligne d'indice 3 est située en 0.

*C'est la fonction **TableauCaracteres.positionsParLigne** qui se chargera d'obtenir ces positions.*

Après avoir trié les indices par ordre croissant à l'aide de la fonction `Arrays.sort`, le tableau devient :

-1	0	1	2
0	1	2	3

Si les règles de placement avaient été respectées, les indices auraient été les suivants :

0	1	2	3
0	1	2	3

C'est la fonction `TableauEntiers.intervalle` qui se chargera de créer ce tableau des indices attendus.

Il suffit dès lors de comparer les deux tableaux, celui obtenu et celui attendu, à l'aide de la fonction `Arrays.equals`.

ILLUSTRATION N°2

Envisageons maintenant l'échiquier de taille 4 x 4 suivant :

	0	1	2	3
0	'L'	'L'	'T'	'L'
1	'L'	'T'	'L'	'L'
2	'L'	'L'	'L'	'T'
3	'T'	'L'	'L'	'L'

Les indices des premières tours trouvées à chaque ligne sont les suivants :

2	1	3	0
0	1	2	3

Une fois ses éléments triés par ordre croissant à l'aide de la fonction `Arrays.sort`, le tableau des indices devient :

0	1	2	3
0	1	2	3

LES CLASSES HUITTOURS ET HUITTOURSTEST

A l'exception de la fonction `main`, toutes les fonctions de la classe `HuitTours` doivent être documentées avec des commentaires javadoc. Cependant, seule la fonction `creerEchiquier` doit être testée à l'aide de JUnit 5 dans la classe `HuitToursTest`.

Dans la classe `HuitTours`, déclarez :



- Une fonction nommée `creerEchiquier` qui crée un tableau à deux dimensions de caractères en l'initialisant avec des caractères espace (désignent les cases vides), à l'exception de la 1^{ère} ligne qui est initialisée avec des lettres T majuscules (désignent les cases occupées par une tour) :

```
public static char[][] creerEchiquier(int taille)
```

Le paramètre `taille` désigne la taille de l'échiquier à créer.

Exemple : si `taille` est l'entier 3, alors le tableau retourné est `[['T', 'T', 'T'], [' ', ' ', ' '], [' ', ' ', ' ']]` (où `_` symbolise un espace).

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

[PRÉCONDITION] La taille réceptionnée par le paramètre `taille` est supposée valide (est comprise entre 2 et 8, bornes incluses).

- Une fonction nommée `afficherEchiquier` qui affiche un échiquier :

```
public static void afficherEchiquier(char[][] echiquier)
```

Le paramètre `echiquier` désigne l'échiquier à afficher.

Exemple : si `echiquier` est le tableau `[[' ', ' ', 'T'], ['T', ' ', ' '], [' ', 'T', ' ']]` (où `_` symbolise un espace), alors la fonction affiche en console :


```

+---+---+---+
|   |   | T |
+---+---+---+
| T |   |   |
+---+---+---+
|   | T |   |
+---+---+---+

```

[SUGGESTION] Aidez-vous de la fonction `String.repeat`.

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

→ Une fonction nommée `main` contenant la logique principale de l'application.

Pensez à effectuer une **validation de la donnée acquise**, à savoir la taille de l'échiquier. Cette dernière doit être comprise entre 2 et 8.

LES CLASSES `TableauCaracteres` ET `TableauCaracteresTest`

Toutes les fonctions de la classe `TableauCaracteres` doivent être documentées avec des commentaires javadoc. A l'exception de la fonction `meLanger`, toutes les fonctions doivent être testées à l'aide de JUnit 5 dans la classe `TableauCaracteresTest`.

Dans la classe `TableauCaracteres`, déclarez :



→ Une fonction nommée `permuter` qui effectue la permutation des caractères situés aux deux positions spécifiées d'un tableau à deux dimensions :

```
public static void permuter(char[][] t, int i1, int j1, int i2, int j2)
```

Le paramètre `t` désigne le tableau dans lequel la permutation doit être effectuée.

Le paramètre `i1` désigne l'indice de la ligne du premier caractère.

Le paramètre `j1` désigne l'indice de la colonne du premier caractère.

Le paramètre `i2` désigne l'indice de la ligne du deuxième caractère.

Le paramètre `j2` désigne l'indice de la colonne du deuxième caractère.

Exemple : si `t` est le tableau `[['T', 'T', 'T'], ['_', '_', '_'], ['_', '_', '_']]` (où `_` symbolise un espace), `i1` et `j1` les indices 1 et 0, `i2` et `j2` les

indices 0 et 2, alors le tableau `t` devient `[['T', 'T', '_'], ['T', '_', '_'], ['_', '_', '_']]`.

[PRÉCONDITIONS] La référence réceptionnée par le paramètre `t` est supposée valide (ne vaut pas `null`). De plus, les positions désignées par les couples d'indices (`i1`, `j1`) et (`i2`, `j2`) sont également supposées valides (existent au sein du tableau `t`).



→ Une fonction nommée `meLanger` qui mélange aléatoirement les caractères présents dans un tableau à deux dimensions :

```
public static void meLanger(char[][] t)
```

Le paramètre `t` désigne le tableau dont les caractères doivent être mélangés.

Exemple : si `t` est le tableau `[['T', 'T', 'T'], ['_', '_', '_'], ['_', '_', '_']]`, alors le tableau `t` pourrait devenir `[['_', 'T', '_'], ['T', '_', '_'], ['_', '_', 'T']]`.

MÉTHODOLOGIE À SUIVRE

Pour chaque case du tableau, effectuez :

Choisir aléatoirement une ligne.

Choisir aléatoirement une colonne.

Permutez le caractère de la case actuelle avec celui de la case située à la position obtenue aléatoirement.

[CONTRAINTES] Pour effectuer ce traitement, vous devez utiliser l'une des fonctions `Aleatoire.aleatoire` et la fonction `TableauCaracteres.permuter`.

[PRÉCONDITIONS] La référence réceptionnée par le paramètre `t` est supposée valide (ne vaut pas `null`). De plus, la longueur de chaque ligne du tableau `t` est supposée supérieure ou égale à 1. Enfin, bien que le programme ne le requiert pas, faites en sorte que le mélange puisse également s'effectuer sur un tableau dont les longueurs de ses lignes diffèrent.

TESTS UNITAIRES

*Comme indiqué précédemment, il n'est pas obligatoire de réaliser des tests unitaires pour cette fonction. Cependant, si vous y parvenez, ces derniers vous rapporteront des **points bonus**.*

→ Une fonction nommée **positionDe** qui détermine la position de la première occurrence d'un caractère au sein d'un tableau :

```
public static int positionDe(char[] t, char c)
```

Le paramètre **t** désigne le tableau dans lequel la recherche doit s'effectuer.

Le paramètre **c** désigne le caractère à rechercher.

*Exemple : si **t** est le tableau ['_', '_', 'T', '_', 'T'] et **c** le caractère 'T', alors la fonction retourne la position 2.*

[CAS PARTICULIER] Si aucune occurrence du caractère recherché n'est trouvée, alors la fonction retourne -1.

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

→ Une fonction nommée **positionsParLigne** qui retourne un tableau répertoriant les positions de la première occurrence d'un caractère au sein de chacune des lignes d'un tableau à deux dimensions :

```
public static int[] positionsParLigne(char[][] t, char c)
```

Le paramètre **t** désigne le tableau dans lequel les recherches doivent s'effectuer.

Le paramètre **c** désigne le caractère à rechercher.

*Exemple : si **t** est le tableau [['T', '_', '_'], ['T', 'T', '_'], ['_', '_', '_']] et **c** le caractère 'T', alors la fonction retourne le tableau [0, 0, -1].*

[CONTRAINTE] Pour effectuer ce traitement, vous devez utiliser la fonction **TableauCaracteres.positionDe**.

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

LES CLASSES `TableauEntiers` ET `TableauEntiersTest`

La fonction de la classe `TableauEntier` doit être documentée avec des commentaires javadoc et testée à l'aide de JUnit 5 dans la classe `TableauEntierTest`.

Dans la classe `TableauEntier`, déclarez une fonction nommée *intervalle* qui crée un tableau énumérant les nombres entiers compris entre les deux bornes spécifiées :

```
public static int[] intervalle(int min, int max)
```

Le paramètre `min` désigne la borne inférieure inclusive de l'intervalle.

Le paramètre `max` désigne la borne supérieure inclusive de l'intervalle.

Exemple : si `min` est l'entier -2 et `max` l'entier 3, alors la fonction retourne le tableau `[-2, -1, 0, 1, 2, 3]`.

[PRÉCONDITIONS] Les bornes désignées par les paramètres `min` et `max` sont supposées valides ($\text{min} \leq \text{max}$).

LES EXEMPLES D'EXÉCUTION

MINIMUM REQUIS

Taille de l'échiquier (de 2 à 8) ? 4

Echiquier après 1 mélange :

```
[[ , , T, ], [ , , , ], [ , T, , T], [ , , T, ]]
```

L'affichage du tableau n'est pas obligatoire, mais permet de vérifier la réussite des fonctionnalités indispensables !

Aidez-vous de `Arrays.deepToString`.

VERSION IDÉALE

Taille de l'échiquier (de 2 à 8) ? 14

Taille de l'échiquier (de 2 à 8) ? 4

```
+---+---+---+---+
|   |   |   | T |   |
+---+---+---+---+
| T |   |   |   |   |
+---+---+---+---+
|   |   |   |   | T |
+---+---+---+---+
|   | T |   |   |   |
+---+---+---+---+
```

La taille admise pour l'échiquier est comprise entre 2 et 8.

63 tentatives ont été nécessaires.