Haute École Bruxelles-Brabant
École Supérieure d'Informatique
Bachelor en Informatique

2019 – 2020
DEV2

DEV2 – Laboratoires Java II

TD01 – Rappels de dev1

Le jeu Memory

Résumé

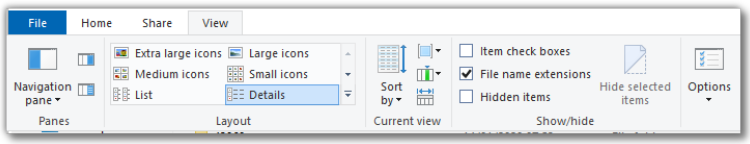
Ce TD va vous aider à revoir tout ce que vous avez fait en DEV1 : les tableaux, la *javadoc*, les tests unitaires et GIT. Les premiers rappels sont sous la forme d'un tutoriel. Le temps de travail estimé se compose de 2h de travail en classe et de 2h de travail à domicile.

1 Les outils

L'ensemble de vos exercices de JAVL seront conservés dans un dossier appelé `DEV2-JAVL`. Afin d'avoir une sauvegarde de votre travail, ce dossier sera historisé sur le serveur GITLAB de l'école. Le tutoriel suivant vous permet de parcourir les étapes à effectuer pour historiser le dossier `DEV2-JAVL` et pour créer le projet NETBEANS du premier exercice.

Tutoriel 1 Utilisation de git et Netbeans.

- Rendez-vous sur le serveur `https://git.esi-bru.be` et identifiez-vous en cliquant sur le bouton : `ESI Google apps login` ;
- créez sur GITLAB un **dépôt** qui va pouvoir accueillir votre travail et donnez à ce **dépôt** le nom : `DEV2-JAVL` ;
- créez un nouveau dossier appelé `DEV2-JAVL` dans le dossier `Mes documents` de votre disque `C:` ;
- dans une fenêtre de l'explorateur de dossiers, dans le menu `View`, activez l'option `File name extensions` afin de rendre visible les extensions des fichiers ;

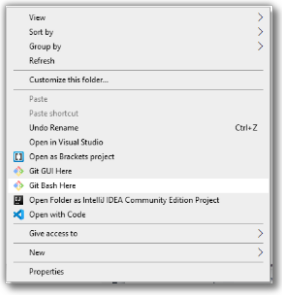


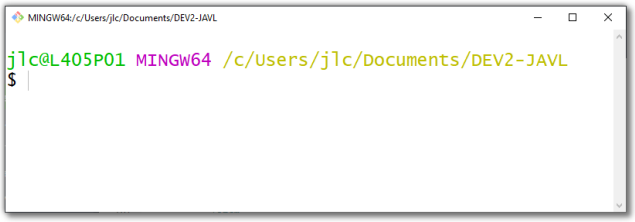
- créez dans le dossier `DEV2-JAVL` un fichier appelé `README.md` ;
- complétez ce fichier avec la description du futur contenu du dossier `DEV2-JAVL`, mettez à jour le nom et le matricule de l'auteur ;

```
# Contenu

Ce repository contient les différents exercices de JAVL.
On y trouve les dossiers suivants :
- TD01 : Rappels de dev1

## Auteur

- **G12345 Juste Leblanc**
```


Version du document
3 février 2020

- à l'intérieur de votre dossier `DEV2-JAVL` effectuez un clic droit et sélectionnez l'option `Git Bash Here` ;

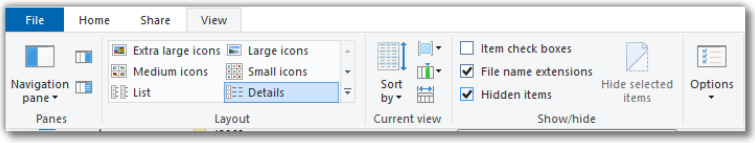




- dans la console maintenant ouverte, vérifiez que vous être bien dans le dossier `DEV2-JAVL` ;
- demandez à GIT de suivre *localement* le dossier `DEV2-JAVL` via la commande suivante [1] :

```
$ git init
```

- dans une fenêtre de l'explorateur de dossiers, dans le menu `View`, activez l'option `Hidden items` afin de rendre visible les fichiers cachés ;



- vérifiez la présence du dossier caché `.git` dans le dossier `DEV2-JAVL` ;
- ajoutez le fichier `README.md` à la zone de travail en exécutant la commande suivante dans la console GIT BASH :

```
$ git add README.md
```

- créez votre premier commit en exécutant la commande :

```
$ git commit -m "TD01 : ajout du README"
```

- ajoutez l'adresse de votre dépôt sur le serveur via la commande :

```
$ git remote add origin <votre url git>
```

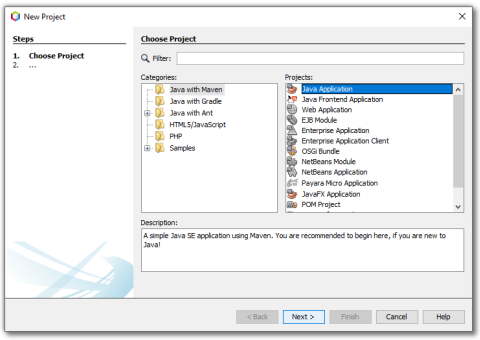
  l'url ressemble à `https://git.esi-bru.be/12345/DEV2-JAVL.git` ;
- envoyez sur le serveur votre premier commit via la commande :

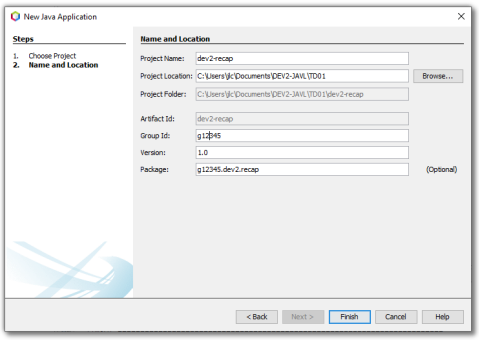
```
$ git push -u origin master
```

- consultez les fichiers de votre dépôt sur le serveur via le menu `Files`, le contenu de votre fichier `README.md` est automatiquement interprété ;
- créez dans le dossier `DEV2-JAVL`, le dossier `TD01` ;

1. Remarquez que nous utilisons GIT via des lignes de commandes car elles sont explicites. Vous pouvez obtenir le même résultat via les menus présents dans NETBEANS. Il faut cependant prêter attention au dossier utilisé par NETBEANS pour enregistrer les données GIT.

- créez un projet via NETBEANS en utilisant MAVEN dans le dossier TD01
  - ▷ Nom du projet : `dev2-recap`.
  - ▷ Project location : `DEV2-JAVL/TD01`.
  - ▷ GroupId : *`<votreLogin>`*.
  - ▷ Version : `1.0`.
  - ▷ Package : *`<votreLogin>`*`.dev2.recap`.





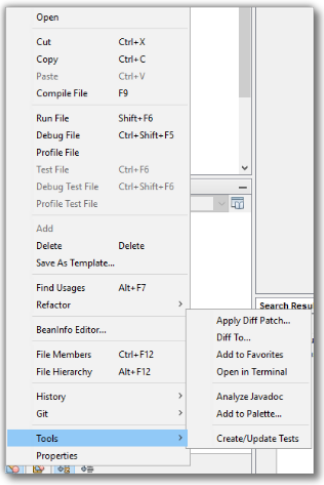
- créez le package `g12345.dev2.recap.utils` ;
- créez dans le package `g12345.dev2.recap.utils` la classe `ArrayUtil` ;
- complétez cette classe avec le code présent sur PoESI ;
- consultez la `javadoc` de la méthode `swap` pour comprendre son utilité ;
- compilez votre projet ;

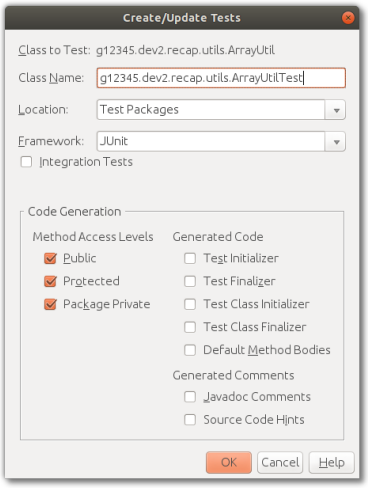
> Lors de la compilation, vous pouvez voir dans la barre des tâches une action d'indexation en cours.
>
> 
>
>
> En résumé MAVEN est en train de récupérer d'un serveur les mises à jour des librairies Java populaires et de les indexer. Cette tâche peut prendre une dizaine de minutes. La configuration de la fréquence de ces indexations peut être modifiée via les options `Tools > Option > Java > Maven > Index`.

- synchronisez votre projet avec le serveur GIT en éxécutant les commandes suivantes à partir du dossier DEV2-JAVL ;

```
$ git add .
$ git commit -m "TD01 : création du projet"
$ git push
```

- cliquez sur la classe `ArrayUtil` et choisissez l'option `Tools > Create/Update test` ;
- dans la fenêtre qui s'affiche, décochez les options de génération automatique du code ;





- modifiez le nouveau fichier `ArrayUtilTest` et ajoutez la méthode suivante :

```java
@Test
public void testSwapGeneralCase() {
    System.out.println("swap general case");
    //Arrange
    int[] array = {10, 11, 12};
    int pos1 = 0;
    int pos2 = 1;
    //Action
    ArrayUtil.swap(array, pos1, pos2);
    //Assert
    int[] expected = {11, 10, 12};
    assertArrayEquals(expected, array);
}
```

- cliquez sur le fichier `ArrayUtilTest` et choisissez l'option `Test File` ;
- vérifiez que l'exécution du test est en succès ;

Comme vous pouvez le constater sur le dernier exemple, un cas de test essaye de respecter une structure en trois étapes : le `AAA pattern`.

1. *Arrange* : on prépare le test, on crée les données utiles ;
2. *Action* : on exécute la méthode que l'on souhaite tester ;
3. *Assert* : on vérifie que le résultat de l'exécution de la méthode avec les données préparées correspond à ce qu'on avait prévu.

- modifiez le nouveau fichier `ArrayUtilTest` et ajoutez la méthode suivante :

```java
@Test
public void testSwapOutsideArrayNegative() {
    System.out.println("swap position négative");
    //Arrange
    int[] array = {10, 11, 12};
    int pos1 = -1;
    int pos2 = 0;
    //Assert
    Assertions.assertThrows(IllegalArgumentException.class, () -> {
        //Action
        ArrayUtil.swap(array, pos1, pos2);
    });
}
```

- cliquez sur le fichier `ArrayUtilTest` et choisissez l'option `Test File` ;
- vérifiez que l'exécution du test est en succès ;

> La syntaxe utilisée pour les tests unitaires est peut-être différente de ce que vous avez expérimenté dans les cours de DEV1, ceux-ci utilisant la version 4 de JUNIT. En DEV2 nous utiliserons la dernière version de cette librairie, JUNIT5, qui utlise de la programmation dite fonctionnelle. Pour identifier la version de JUNIT utilisée, consultez les dépendances ajoutées à votre projet par MAVEN ou regardez les `imports` ajoutés à votre classe de test. Nous ne vous demandons par pour l'instant de maîtriser la programmation fonctionnelle. Nous reviendrons plus en détails sur ces notations dans un TD dédié à la question.

- compilez votre code ;

> Lorsque vous compilez votre projet avec MAVEN, les tests unitaires sont automatiquement exécutés. Autrement dit un projet dont les tests unitaires échouent, génère une erreur lors de la compilation.

- vérifiez que les tests unitaires ont bien été exécutés en inspectant la console `Output` de NETBEANS ;

```
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
```

- créez dans le package `g12345.dev2.recap.utils` la classe `MemoryUtil` ;
- complétez cette classe avec le code présent sur PoESI ;
- compilez votre code ;
- synchroniser votre projet avec le serveur GITLAB.

```
$ git add .
$ git commit -m "TD01 : test unitaire de ArrayUtils"
$ git push
```

Le tutoriel concernant les rappels d'utilisations de GIT et NETBEANS est maintenant terminé.

2 Memory

Voici un exercice conséquent pour pratiquer les éléments importants de DEV1.

- ▷ Vous devez écrire la *javadoc* de **toutes** les méthodes.
- ▷ Votre code doit être **robuste** (résister aux mauvaises entrées de l'utilisateur).
- ▷ Vous devez sauver votre travail sur GITLAB en faisant un commit à la fin de chaque méthode terminée.

2.1 Présentation

Nous allons implémenter une version simplifiée du jeu de Memory.

> « Le jeu de Memory se compose de paires de cartes portant des illustrations identiques. L'ensemble des cartes est mélangé, puis étalé face contre table. À son tour, chaque joueur retourne deux cartes de son choix. S'il découvre deux cartes identiques, il les ramasse et les conserve, ce qui lui permet de rejouer. Si les cartes ne sont pas identiques, il les retourne faces cachées à leur emplacement de départ. » (wikipedia)

La version que nous allons développer se joue à un seul joueur.

Exemple de déroulement du jeu

```
*** Memory ***
Avec combien de paires voulez-vous jouer ? (3 à 20): 3
*** tour 1
Les cartes: ? ? ? ? ? ?
Entrez une position de carte (0 à 5): 0
La carte en position 0 est un 2
Entrez une position de carte (0 à 5): 1
La carte en position 1 est un 1
Elles ne correspondent pas !
*** tour 2
Les cartes: ? ? ? ? ? ?
Entrez une position de carte (0 à 5): 2
La carte en position 2 est un 3
Entrez une position de carte (0 à 5): 3
La carte en position 3 est un 1
Elles ne correspondent pas !
*** tour 3
Les cartes: ? ? ? ? ? ?
Entrez une position de carte (0 à 5): 1
La carte en position 1 est un 1
Entrez une position de carte (0 à 5): 3
La carte en position 3 est un 1
Elles correspondent !
*** tour 4
Les cartes: ? 1 ? 1 ? ?
Entrez une position de carte (0 à 5): 1
Cette carte n'est plus disponible !
Entrez une position de carte (0 à 5): 4
La carte en position 4 est un 3
Entrez une position de carte (0 à 5): 2
La carte en position 2 est un 3
Elles correspondent !
*** tour 5
Les cartes: ? 1 3 1 3 ?
Entrez une position de carte (0 à 5): 5
La carte en position 5 est un 2
```

```
Entrez une position de carte (0 à 5): 0
La carte en position 0 est un 2
Elles correspondent !
Partie terminée en 5 coups.
```

Choix d'implémentation

Nous allons noter n le nombre de *paires* de cartes. Il y a $2 * n$ cartes dans le jeu. Les cartes n'ont pas *d'illustration* mais une valeur (un entier compris entre 1 et n).

Nous représenterons les cartes à découvrir par un tableau de taille $2 * n$, que nous appellerons `cards`. Par exemple le tableau (avec 4 couples cette fois)

`cards` =

2	4	4	3	1	3	1	2

représente la suite de cartes de valeur : $2, 4, 4, 3, 1, 3, 1, 2$.

Les cartes ne sont pas visibles. Le joueur donnera la position des deux cartes à retourner (0 étant la première carte, 1 la deuxième, etc). Par exemple, le joueur propose les cartes en position 3 et 5. Dans notre exemple ci-dessus, il s'agit des cartes de valeurs 3. Le joueur a découvert une paire ; il conserve donc ces 2 cartes.

Pour savoir quelles cartes le joueur a déjà découvertes et ramassées on utilise un tableau de booléens de même taille que le tableau de cartes. La i-ème case de ce tableau est *true* si le joueur a ramassé la carte en position i. Nous appellerons ce tableau `collectedCards`.

Par exemple, pour les cartes de l'exemple ci-dessus le tableau suivant :

`collectedCards` =

false	false	false	false	false	false	false	false

indique qu'il n'a ramassé aucune carte.

Par contre le tableau

`collectedCards` =

false	false	false	true	false	true	false	false

indique qu'il a ramassé les cartes en position 3 et 5 (de valeur 3).

Le jeu se déroule comme suit :

1. le programme affiche les cartes (`'?'` pour une carte pas encore ramassée) ;
2. le joueur précise la position de 2 cartes qu'il veut retourner, le programme affiche la valeur des cartes choisies ;
3. si la valeur des cartes indiquées est la même, le joueur ramasse les 2 cartes ;
4. si toutes les paires de cartes n'ont pas été trouvées, on recommence.

2.2 Initialiser les cartes

Dans le package `g12345.dev2.recap` créez une nouvelle classe appelée `Memory`. Dans cette classe `Memory`, écrivez une méthode

```
int[] initCards(int n)
```

qui crée un tableau de $2 * n$ entiers et l'initialise avec les valeurs des cartes. Chaque valeur de 1 à n apparait deux fois (dans l'ordre ; nous nous occuperons de le *mélanger* plus tard). Par exemple, si n vaut 4, le tableau sera initialisé ainsi :

1	1	2	2	3	3	4	4

Une `IllegalArgumentException` est lancée si le nombre de paires n'est pas entre 3 et 20 inclus.

Écrivez des **tests unitaires** pour vérifier que la méthode retourne le bon tableau et que l'exception est lancée quand il faut. Les différents cas de tests à implémenter sont les suivants :

n°	Entrées	Résultat attendu	Note
1	4	1, 1, 2, 2, 3, 3, 4, 4	Cas général
2	3	1, 1, 2, 2, 3, 3	Taille minimale
3	20	1, 1, ..., 19, 19, 20, 20	Taille maximale
4	2	IllegalArgumentException	Trop petit
5	21	IllegalArgumentException	Trop grand

Vous devez écrire la *javadoc* de **toutes** les méthodes (hors méthodes de test).

Faites un *commit* (message : `"TD01 - Memory - Initialiser les cartes"`) avant de passer à l'étape suivante. Déposez votre travail sur le serveur GitLab via la commande *push*.

2.3 Demander une position

Écrivez une méthode

```
int askPosition(int[] cards, boolean[] collectedCards)
```

qui demande à l'utilisateur une position de carte, affiche la valeur de cette carte et retourne la position. Si la position ne correspond pas à une carte encore en jeu, un message d'erreur est affiché et la méthode repose la question jusqu'à ce que ce soit correct.

N'hésitez pas à reprendre les classes de lecture robuste que vous avez développé durant les cours de DEV1. Les tests unitaires ne sont pas demandés ici.

On vous le rappelle une dernière fois, n'oubliez pas la **javadoc**, le **commit** et la synchronisation avec le serveur GitLab.

2.4 Jouer un coup

Écrivez une méthode

```
void checkPositions(int[] cards, boolean[] collectedCards, int pos1, int pos2)
```

qui teste les deux cartes en position `pos1` et `pos2` et affiche un petit message indiquant si elles correspondent. Dans ce cas, elles sont prises par le joueur (le tableau `collectedCards` est adapté en conséquence).

La méthode doit lancer une exception si `pos1`=`pos2`.

Les tests unitaires ne sont pas demandés ici.

2.5 Vérifier si le jeu est terminé ou non

Écrivez une méthode

```
boolean isGameOver(boolean[] collectedCards)
```

qui vérifie si le jeu est terminé. Pour rappel, le jeu est terminé dès que le joueur a ramassé toutes les cartes.

Par exemple, si le tableau reçu en paramètre est :

false	false	false	true	false	true	false	false

le jeu n'est pas terminé et la méthode doit retourner `false`. Par contre, avec le tableau

true	true	true	true	true	true	true	true

toutes les cartes ont étés ramassées et la méthode doit retourner `true`.

2.6 Le jeu complet

Écrivez une méthode

```
int playMemory(int n)
```

qui implémente le jeu pour n paires et retourne le nombre de tours que le joueur a dû effectuer pour ramasser toutes les cartes.

Les étapes du jeu sont les suivantes :

1. Initialiser le paquet de cartes (`cards`) et le tableau de booléens (`collectedCards`).
2. Mélanger les cartes. Pour ce faire, vous pouvez utiliser la méthode que nous fournissons dans la classe `ArrayUtil`.
3. Répéter les étapes suivantes :
   (a) Afficher les cartes. Nous vous fournissons le code correspondant dans la classe `MemoryUtil`.
   (b) Demander une première position à l'utilisateur et afficher la valeur de la carte à cette position.
   (c) Demander une seconde position à l'utilisateur et afficher la valeur de la carte à cette position.
   (d) Jouer un coup avec ces 2 positions.
   (e) Recommencer si la partie n'est pas finie.

Il faut aussi gérer le nombre de tours pour pouvoir le retourner.

2.7 La méthode principale

Vous pouvez à présent remplacer votre (éventuelle) méthode principale par un code qui :

1. demande à l'utilisateur avec combien de paires il veut jouer ;
2. joue une partie ;
3. affiche le nombre de coups utilisés.

Le développement du `Memory` est à présent terminé. N'oubliez pas de faire un dernier commit et de déposer votre travail sur le serveur GitLab.