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

2021
DEV2 – JAVL

DEV2 – JAVL – Laboratoire Java

TD 1 – 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 Utiliser Netbeans et git

Git en mode console

Dans le tutoriel, nous vous montrons comment utiliser GIT **au travers des menus** de NETBEANS. Pour votre information (et parce que ça peut être pratique en cas de problème, pour vous dépanner à distance), nous vous indiquerons également comment réaliser les mêmes opérations en **mode commande**. Si vous voulez taper les commandes chez vous, il faut vous un GIT indépendant de NETBEANS :

- ▷ Sous LINUX ou MACOS vous l'avez probablement déjà installé pour vos laboratoires d'environnement en DEV1.
- ▷ Sous WINDOWS, vous pouvez en instaler un ici : `https://git-scm.com/downloads`.

Tutoriel 1 Utilisation de git et Netbeans.

Dans ce tutoriel, vous serez amenés à créer un dépôt sur GITLAB, créer un petit projet et le sauver sur le dépôt. Nous ferons également un bref rappel des tests unitaires.

Création du dépôt et partage avec votre professeur

- Rendez-vous sur le serveur `https://git.esi-bru.be` et identifiez-vous.
- Créez sur GITLAB un **dépôt** qui va pouvoir accueillir votre travail et donnez-lui le nom : `dev2-javl`.
- On vous demande d'ajouter votre professeur en tant que membre de votre projet. Cela va lui permettre de mieux vous suivre et vous aider.
  - ▷ Choisissez le menu `Settings/Members`.
  - ▷ Ajoutez votre professeur comme membre au niveau `Reporter`.

Version du document
1er février 2021

Création du projet et suivi par git

Nous allons créer un projet que vous utiliserez pour tous vos tds de DEV2-JAVL.

- Créez un projet via NETBEANS en utilisant MAVEN.
  - Nom du projet : `dev2-javl`.
  - Project location : où vous voulez ;)
  - GroupId : *<votreLogin>*.
  - Version : `1.0`.
  - Package : *<votreLogin>*`.dev2`.
- Demandez à GIT de suivre *localement* le dossier via un clic-droit sur le projet puis menu `Versioning/ Initialize Git Repository...`

```
En mode commande, placez-vous dans le dossier du projet
et tapez la commande suivante :
$ git init
```

- Effectuez votre premier commit via un clic-droit sur le projet menu `Git/Commit...`
  - Message : `Création du projet pour dev2-javl`.

```
En mode commande, vous obtenez le même effet avec :
$ git add .
$ git commit -m "Création du projet pour dev2-javl"
```

- Envoyez le commit sur le serveur via le menu `Git/Remote/Push...`
  - L'url *https* de votre projet est à trouver sur GITLAB.
    Il ressemble à `https://git.esi-bru.be/12345/DEV2-JAVL.git`.
  - Entrez votre `username` sur GITLAB (votre numéro d'étudiant).
  - Ainsi que votre `password` sur GITLAB.
  - Vous pouvez cochez la demande de sauvegarde du mot de passe.

```
En mode commande, vous obtenez le même effet avec :
$ git remote add origin <votre url git>
$ git push -u origin master
```

- Vérifiez sur le dépôt que le commit est présent.

Nous vous demandons d'utiliser ce projet pour **tous** vos tds de DEV2-JAVL.

Tests unitaires

- Créez la classe `ArrayUtil` dans le package `g12345.dev2.td01`.
- Complétez cette classe avec le code présent sur PoESI.
- Consultez la `javadoc` de la méthode `swap` pour comprendre son utilité.
- Synchronisez votre projet avec le serveur GIT via un *commit* puis un *push*.
  - Message : `td01 - tuto`.
  - Vous pouvez choisir `Push to upstream` au lieu de `Push...` pour éviter de redonner les mêmes informations.

```
En mode commande :
$ git add .
$ git commit -m "td01 - tuto"
$ git push
```

- Cliquez sur la classe `ArrayUtil` et choisissez l'option `Tools > Create/Update test`.
- Dans la fenêtre qui s'affiche, décochez toutes les options.
- Modifiez le nouveau fichier `ArrayUtilTest` et ajoutez la méthode suivante :

```java
@Test
public void testSwapGeneralCase() {
    //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);
}
```

> 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.
>
> Dans des cas simples, vous pouvez aussi condenser l'écriture en une seule ligne, par exemple : `assertEquals(4, Util.max(3,4));`.

- Cliquez sur le fichier `ArrayUtilTest` et choisissez l'option `Test File`.
- Vérifiez que l'exécution du test est en succès.
- Modifiez le nouveau fichier `ArrayUtilTest` et ajoutez la méthode suivante :

```java
@Test
public void testSwapOutsideArrayNegative() {
    int[] array = {10, 11, 12};
    int pos1 = -1;
    int pos2 = 0;
    Assertions.assertThrows(IllegalArgumentException.class, () -> {
        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.
- Synchronisez votre projet avec le serveur GIT via un *commit* puis un *push*.

  ▷ Message : `td01 - tuto - add tests`

```
En mode commande :
$ git add .
$ git commit -m "td01 - tuto - add tests"
$ git push
```

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.td01` 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.