Les Dionysies

Qualité de développement

IUT 45 — Informatique

2023-2024

Résumé

Cette feuille vous initie à l'utilisation de bibliothèques en Java.

1 Les classes de base : Auteur, Style et Journée

Les Dionysies étaient des festivités religieuses annuelles dédiées au dieu Dionysos dans la Grèce antique. Au cours de leurs célébrations, les participants étaient appelés à concourir en agônes, c'est-à-dir des concours de pièces de théâtres. Ces festivités dédiées au dieu du vin et de l'ivresse [1] étaient passablement moins guindées qu'une soirée au théâtre aujourd'hui. Naturellement, pour satisfaire un public ne demandant qu'à s'amuser et à s'oublier dans le meilleur de la fiction du moment, il faut de la part des organisateurs la plus grande rigueur. C'est pourquoi, en modélisant ce concours dans ce TP, vous vous attacherez à respecter les bonnes pratiques vues la semaine dernière : utilisation de git et Test Driven Development.

1.1 Préparations

Avant de commencer, nous allons appliquer les bonnes pratiques vues lors du dernier tp. Pour ce faire, vous allez créer un projet git sur Github appelé `dionysies` et vous inviterez votre voisin à participer au projet.

Une fois le projet créé, récupérez les sources du tp sur Célène et associez ces sources à votre projet distant. Vous pouvez vous référer au tp1 ou aux directives de Github.

> Sauf mention contraire, après chaque question, il est **impératif de faire un commit** avec un message descriptif. La forme conseillée pour ces messages est : «Question X : ajout des tests pour YYY», ou bien «Question X : implémentation de la fonctionnalité Z».
>
> Il peut arriver à certaines questions que vous reveniez sur une fonctionnalité déjà implémentée précédemment ; dans ce cas, votre message de commit sera de la forme «Correction : dans la fonction X, changement pour ZZZ».

1. et du plaisir charnel

1.2 La classe Auteur

Nous allons commencer par représenter les dramaturges, avec leurs citations les plus croustillantes !

Question 1 Ouvrez le fichier `Main.java` que vous avez récupéré sur Célène. Ce fichier contient une méthode `public static void main()` qui teste le fonctionnement du constructeur de `Auteur`. Faites en sorte que la compilation et l'exécution de ce fichier réussisse, en créant une classe `Auteur`, son constructeur et sa méthode `getQualitéTragédie`.

On implémente un constructeur qui correspond à l'item de backlog[2] suivant :

> On veut pouvoir représenter des auteurs avec :
>
> — un nom
> — leur performance en tragédie, avec une citation et un nombre entre 0 et 100 qui représente la qualité
> — la même chose pour la comédie,
> — idem pour le drame

Question 2 Pour pouvoir afficher des instances de `class Auteur`, il faut ajouter à cette classe une méthode `public String toString()` qui renvoie le nom de l'auteur précédé de `"L'honorable "`. N'oubliez pas le `@Override`.

Question 3 Quand vous compilez votre code, `javac` crée des fichiers `.class`. Créer un fichier `.gitignore` pour ignorer ces fichiers.

Question 4 Ajoutez de même des assertions et l'implémentation de `getCitationTragédie`.

Question 5 Complétez de même les méthodes pour avoir une citation et la qualité pour la comédie et le drame.

On veut maintenant pouvoir déterminer le style (comédie, tragédie ou drame) dans lequel chaque auteur est le meilleur.

Question 6 Récupérez sur Célène le fichier `Style.java`. Ce fichier définit simplement trois constantes `Style.COMÉDIE`, `Style.TRAGÉDIE` et `Style.DRAME` qui nous permettent de distinguer les trois styles.

Question 7 Pour implémenter dans la classe `Auteur` la fonctionnalité «obtenir le style dans lequel il est le meilleur». Il faudra pour cela implémenter une méthode `pointFort()`.

Indiquez ci-dessous les étapes nécessaires pour le faire conformément à la méthodologie *Test Driven Development* :

—

2. Un *backlog* est une liste de tâches à implémenter avec des niveaux de priorité

—

—

Question 8 Mettre en oeuvre le plan défini à la question 7!

Question 9 Ajouter dans la classe Auteur, une méthode `qualitéStyle(Style s)` qui renvoie la qualité de l'auteur dans le style `s`.

Question 10 Ajoutez dans la classe Auteur une méthode `citationStyle(Style s)` qui renvoie la citation de l'auteur dans le style `s`.

1.3 La classe Journée

Représente une journée avec trois épreuves : matin, après-midi et soirée. Chaque épreuve est associée à un style. Le score sur une journée est la somme des scores des trois épreuves. Le score d'un auteur à une épreuve est le produit du nombre de spectateurs de l'épreuve par la qualité de l'auteur dans le style concerné.

Simuler une journée : afficher les citations pour chaque période pour les deux candidats. Calculer / Afficher le volume d'applaudissements pour chacun. Calculer le score total. Afficher le gagnant et renvoyer son indice (0 ou 1).

Question 11 Implémenter tout cela en respectant les règles du TDD.

2 Utilisation d'une bibliothèque : le tournoi

Avec les auteurs, les styles et les journées, tout est prêt pour passer à l'organisation des dionysies [3]. Elles prennent la forme d'un tournoi composé de plusieurs journées; qui est invité à concourir à chaque journée dépend des résultats de chacun aux journées précédentes. À l'issue du tournoi, le vainqueur est désigné.

Toutes ces fonctionnalités ont été implémentées par vos enseignants sous forme d'une *bibliothèque* au format jar.

À retenir

Une bibliothèque est un ensemble de classes et de méthodes qui peuvent être utilisées par un programme et offrant une certaine API (Application Programming Interface), c'est à dire un ensemble de méthodes et de classes avec des profils précis que le programmeur peut utiliser.
Les bibliothèques java sont généralement fournies sous forme d'archives `jar` (Une archive zip contenant des fichiers `.class` et un fichier `MANIFEST.MF` contenant des métadonnées sur la bibliothèque)

3. vous assurerez en autonomie de la partie débauche

Vos enseignants vous fournissent une classe `Tournoi`. Il va falloir la récupérer dans un jar, puis l'utiliser depuis votre programme pour organiser tout un tournoi !

Question 12 Récupérer le fichier `tournoi.jar` sur Célène, et le placer dans le même dossier que votre projet.

Attention

Le fichier `tournoi.jar` qui vous est fourni a été compilé avec Java 17. Si vous voulez compiler ou exécuter votre code depuis VSCode, il faut lui indiquer d'utiliser cette version de Java. Il faut pour cela aller en bas de l'onglet `Explorer`, dans `Java Projects`, cliquer sur les points de suspension (`More Actions`), puis «Configure Java Runtime» et vérifier que la colonne «Java Version» indique bien 17.

La documentation de la bibliothèque tournoi se trouve dans Celene (Télécharger l'archive avant de consulter les pages html).

Question 13 Repérez dans la documentation les classes et méthodes permettant de :

A représenter un tournoi,

B donner le nom d'un tournoi,

C créer un tournoi vide,

D ajouter un participant à un tournoi,

E afficher tous les participants d'un tournoi.

Par défaut, le compilateur `javac` et la commande `java` recherchent les classes auxquelles votre code fait référence dans le répertoire courant, ainsi que dans la bibliothèque standard de java. Pour avoir accès à la classe `class Tournoi` qui est définie dans le fichier `tournoi.jar`, il faut indiquer au compilateur `javac` et à la commande `java` de considérer aussi le fichier jar comme un endroit susceptible de contenir des classes utiles à notre programme. La liste des endroits où rechercher des classes s'appelle le *classpath*; si on a besoin d'y ajouter d'autres emplacements que le répertoire courant, il faut appeler `java` et `javac` avec l'option `-cp`. Cette option est suivie par les éléments du classpath, séparés par des ':'.

De plus, pour utiliser une classe provenant d'une bibliothèque, il faut l'importer dans votre fichier source à l'aide du mot-clé `import`.

Question 14 Ajoutez en haut de votre fichier principal la ligne suivante :

```
import dionysies.Tournoi;
```

Question 15 Ajouter dans votre main les instructions nécessaires pour créer un tournoi entre les trois auteurs qui y sont définis.

À retenir

Pour utiliser une classe contenue dans un fichier jar, il faut :

— lire et comprendre sa documentation ;

— ajouter un `import package.Class;` dans les fichiers pertinents ;

— ajouter l'option `-cp .:fichier.jar` aux commandes `javac` et `java`.

Question 16 Quels sont les participants de la journée 1 du tournoi ?

Question 17 Comment obtenir les résultats de la journée 1 ?

Question 18 Comment obtenir les participants de la journée 1 du tournoi ?

Question 19 A l'aide d'une boucle apropriée, afficher tous les participants du tournoi.

3 Des tests plus sérieux avec JUnit

Nous allons maintenant étudier l'outil standard pour écrire des tests en Java : JUnit.

3.1 Installation de JUnit

Pour pouvoir écrire des tests en Java, nous avons besoin de la bibliothèque JUnit (et d'une bibliothèque auxiliaire appelée hamcrest). Les bibliothèques Java sont distribuées sous forme de fichiers `jar`. Quand elles sont sous licence libre, on peut généralement les récupérer depuis `https://search.maven.org/`.

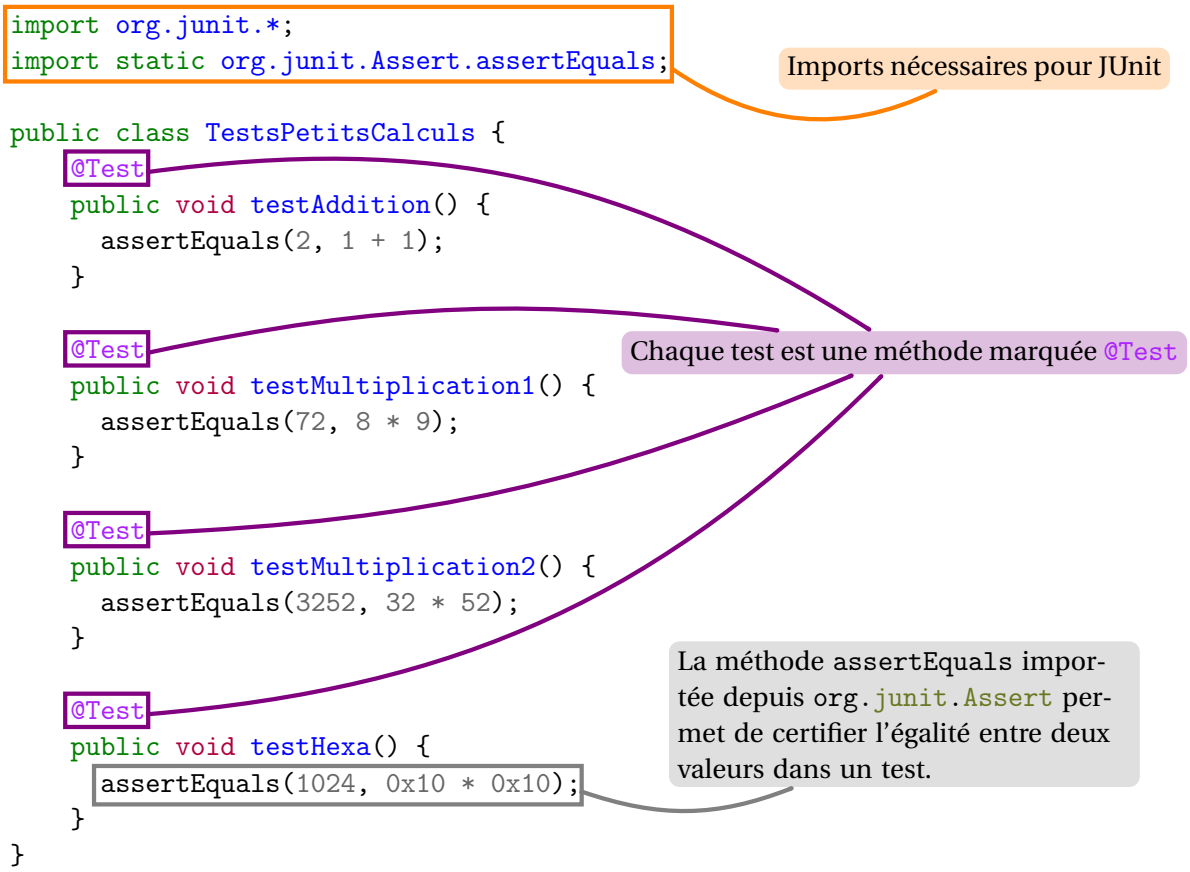
Question 20 Récupérer sur Célène les fichiers `junit-4.13.2.jar` et `hamcrest-2.2.jar`.

Astuce

Vous pouvez indiquer à `code` que vous allez utiliser ces deux fichiers `.jar` en ouvrant la palette de commandes `Ctrl-Shift-P`, puis en tapant `configure classpath`. Dans la page que vous obtenez, dans la section **Referenced Libraries**, utiliser le bouton **Add** pour ajouter chacun des deux fichiers `.jar`.

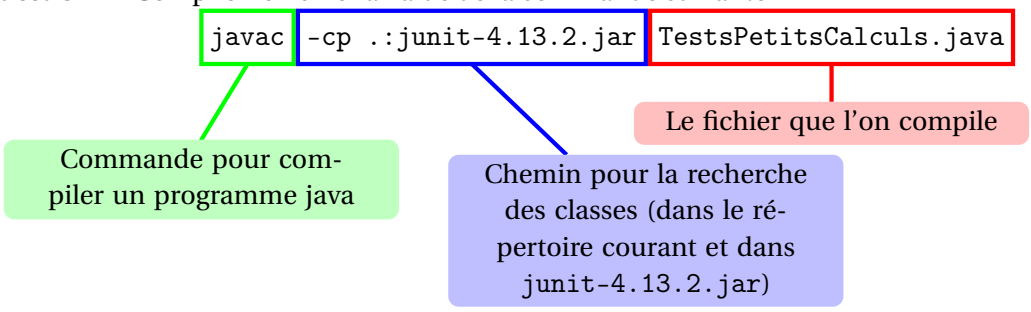
À l'aide de ces deux fichiers, nous allons pouvoir écrire notre première classe de tests ! Vérifions par exemple nos connaissances en arithmétique ...

Question 21 Recopier le fichier `TestsPetitsCalcul.java` suivant, qui contient vos premiers tests utilisant JUnit.



Ce fichier utilise les deux bibliothèques java `JUnit` et `hamcrest`. Pour le compiler et l'exécuter, il faut indiquer au compilateur java et à la jvm où trouver ces bibliothèques.

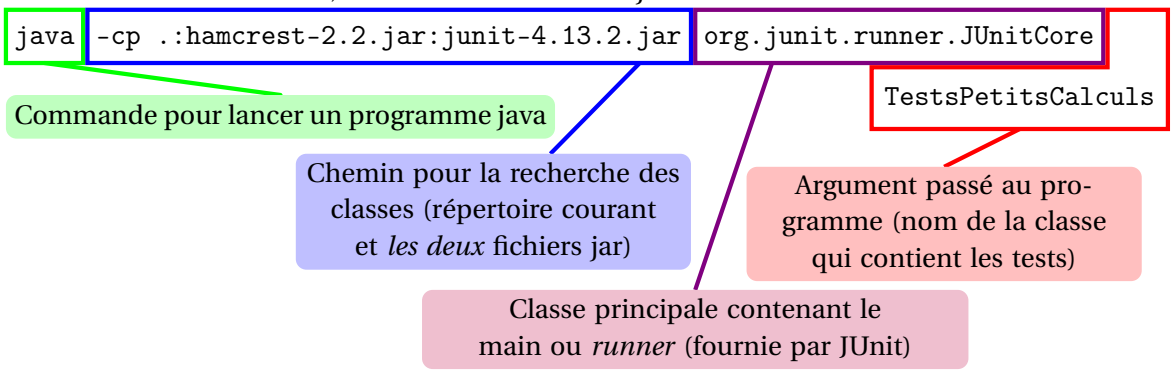
Question 22 Compiler le fichier à l'aide de la commande suivante



Notre fichier contient une classe TestsPetitsCalculs. Les méthodes de cette classe qui correspondent à des tests portent l'annotation `@Test`. Comme vous pouvez le remarquer, cette classe ne contient pas de méthode `main`. Pour lancer les tests, nous allons donc utiliser une

classe *fournie par JUnit* qui contient un main. Cette classe est un *test runner*; elle s'appelle `org.junit.runner.JUnitCore`, et est contenue dans `junit-4.13.2`. Elle fait appel à des classes contenues dans `hamcrest-2.2.jar`. Sa fonction `main()` attend comme argument le nom de la classe contenant les tests.

Pour faire tourner nos tests, nous allons donc lancer java comme suit :



Question 23 Lancer les tests de la classe qui vous a été donnée. Repérer les messages d'erreur et corriger les tests qui ne passeraient pas.

À retenir

Les tests en java s'écrivent avec JUnit, dans une classe de Test. Cette classe doit se compiler avec les fichiers `.jar` de JUnit et hamcrest. On lance les tests en exécutant la classe `org.junit.runner.Runner` et en passant le nom de la classe de tests en paramètre.

À retenir

En JUnit, les tests sont les méthodes marquées `@Test` dans la classe de tests.

À retenir

Dans un test JUnit, on peut utiliser `assertEquals` pour affirmer que deux valeurs sont égales. Ainsi, `assertEquals(2+2, 4)` permet au test de passer, tandis que `assertEquals(2+2, 5)` provoque l'échec du test qui le contient.

3.2 Vos premiers tests JUnit

Question 24 Ajoutez des tests sur des String Java dans la classe de tests. Par exemple, testez que `"abc".length()` est égal à 3. Puis écrire une fonction qui inverse une chaine de caractères et la tester avec JUnit. En déduire une fonction palindrome qui teste si une chaine de caractères est un palindrome et les tests correspondants en JUnit. Ajoutez des tests dont on sait qu'ils vont échouer. Comment les déclarer en JUnit?