TP 1

⚠ *Ce TP présente des manipulations (compilation, exécution, etc.). Vous devez les comprendre et vous les approprier. Dans les prochains TP (et dès la fin de celui-ci), vous **devez être capables** de les appliquer, sans aide. Pour les codes à écrire, vous devez vous aider des codes fournis et du travail fait en TD.*

Mise en place de l'espace de travail Créez un dossier dans votre espace de travail, par exemple `poo/tp1`. Récupérez sur le portail le fichier `fichiers-tp1.zip` qui accompagne décompressez cette archive dans ce dossier. Les fichiers à utiliser dans ce TP se trouvent désormais dans `poo/tp1`.

Exercice 1 : Premier contact avec le code

Dans cette classe on va travailler avec une classe `Tv` qui permet de représenter des objets représentant des télévisions.

Q 1 . Dans le dossier `poo/tp1/tv/src`, ouvrez dans un éditeur le fichier `Tv.java`.
Q 2 . Etudiez le code et identifiez

- le ou les constructeurs ;
- les attributs, quels sont leurs types ?
- les méthodes, combien y en a-t-il ?

⚠ *Dans le code source, les parties commentées encadrées de « `/**` » et « `*/` » correspondent à la documentation de l'élément qu'elles précèdent. Leur lecture fournit donc des informations sur ces éléments. Nous reviendrons d'ici peu sur l'écriture et l'exploitation de cette documentation.*

Q 3 . *(sur feuille)* On suppose que l'on définit les variables tv1 et tv2 ainsi :

```
Tv tv1 = new Tv();
Tv tv2 = new Tv();
```

Q 3.1. Quelles sont les valeurs des attributs de tv1 ? de tv2 ?
Q 3.2. Quel est le résultat de l'appel tv1.currentChannel() ?
Q 3.3. Quel est le résultat de l'appel tv1.on() ?
Quel est l'effet de cet appel ?
Quelles sont maintenant les valeurs des attributs de tv1 ?
Q 3.4. Quels sont les résultats et effets de l'appel tv1.volumeUp() ?
Quelles sont maintenant les valeurs des attributs de tv1 ?
Q 3.5. Même question avec tv2 à la place de tv1 ?
Q 3.6. Vous êtes sûr(e) de votre réponse précédente ? (Lisez bien le code de la méthode `volumeUp()`)

Exercice 2 : Premières compilations

Un code java doit être compilé pour pouvoir être utilisé dans un programme.
La commande qui permet la compilation s'appelle `javac` (comme « *java* compiler »). Cette commande prend en paramètre le nom du fichier contenant la classe à compiler.

Q 1 . Dans un terminal, placez-vous dans le dossier `poo/tp1/tv/src`. Consultez le contenu du dossier (commande « `ls` ») et exécutez la commande

```
.../poo/tp1/tv/src$ javac Tv.java
```

Consultez à nouveau le contenu du dossier `poo/tp1/tv/src`, que constatez-vous ?
Q 2 . Ouvrez le fichier `Tv.class` dans un éditeur de texte. Qu'observez-vous ?
Q 3 . Supprimez le fichier `Tv.class`[1].
Puis, dans le fichier `Tv.java`, ajoutez le code suivant :

[1] Rappel du stage unix : la commande « `rm Tv.class` » permet de faire cela depuis le terminal

```
/**
* turn this tv off
*/
public void off() {
  this.on = False
}
```

Q 3.1. Compilez le fichier `Tv.java` comme ci-dessus (on peut aussi dire « *compilez la classe Tv* »).
Q 3.2. Que constatez-vous dans le terminal ?
Q 3.3. A quelle ligne se situe l'erreur ? Qu'indique le message d'erreur ?
Q 3.4. Constatez qu'aucun fichier `Tv.class` n'a été créé.
Corrigez le problème indiqué, sauvegardez le fichier, puis essayez à nouveau de compiler la classe Tv.
Q 3.5. Corrigez et compilez à nouveau.
Qu'indique l'exécution de la commande de compilation ?

> ⚠ **Bonne pratique** ⚠
> Pour une meilleure organisation des fichiers d'un projet, une bonne pratique consite à ne pas mélanger dans un même dossier les codes sources (fichiers `.java`) et les codes compilés (fichiers `.class`).

Cette bonne pratique peut-être facilement appliquée en adaptant la commande compilation. La commande `javac` peut prendre en option :

- `-sourcepath` pour lui indiquer où se trouve les fichiers source à compiler ;
- `-d` pour lui indiquer où ranger les fichiers générés par la compilation (« d » comme « *d*estination »).

Q 4. On veut, par exemple, conserver les fichiers source dans le dossier `src/` (comme c'est le cas actuellement) et placer les fichiers compilés dans un dossier `classes/`.

Q 4.1. Supprimez le fichier `Tv.class` du dossier `poo/tp1/tv/src`.
Q 4.2. Placez-vous dans le dossier `poo/tp1/tv/` et exécutez la commande [2] :

```
.../poo/tp1/tv$ javac -sourcepath src src/Tv.java -d classes
```

⚠ *On compile un fichier, on doit donc préciser le chemin relatif d'accès à ce fichier, `src`, ainsi que son extension, `.java`.*

Q 4.3. Qu'observez-vous au niveau du dossier `poo/tp1/tv/` ?
Q 4.4. Que contient le dossier `poo/tp1/tv/classes` ?
Vérifiez que le contenu du dossier `poo/tp1/tv/src` n'a pas été modifié.

⚠ *À partir de maintenant, vous appliquerez toujours cette bonne pratique, en utilisant la commande de compilation présentée ci-dessus.*

Exercice 3 : Première exécution

La commande `java` permet d'exécuter une *Machine Virtuelle Java* (*JVM*) qui fournit un environnement d'exécution de code Java compilé. Cette commande prend en paramètre un nom de classe qui doit nécessairement définir une méthode dont la signature est **obligatoirement** et **rigoureusement** :

```
public static void main(String[] args)
```

Exécuter un programme JAVA consiste à exécuter le corps de la méthode `main` de la classe passée en paramètre à la commande `java`. Cette classe devra évidemment avoir été compilée.
Pour exécuter correctement un programme Java, il est nécessaire que la JVM puisse charger les fichiers compilés définissant les classes, afin d'en connaitre leurs définitions. Le plus souvent, il est donc nécessaire de préciser à la commande `java` le dossier contenant ces fichiers compilés, ce qui peut-être fait à l'aide de l'option `-classpath`.

Q 1. Ouvrez le fichier `poo/tp1/tv/src/TvMain.java` dans un éditeur de texte et étudiez son contenu, notamment pour vérifier la présence d'une méthode « `main` ».

[2] Selon les versions de la commande `javac`, notamment sur Mac, il est possible que vous deviez créer le dossier `poo/tp1/tv/classes` avant d'exécuter cette commande, faute de quoi vous obtiendrez un message d'erreur. Au M5, cette commande est valide telle qu'elle.

⚠ *On rappelle que « `System.out.println` » permet d'afficher une chaîne de caractères dans la console. De plus, comme le montre la différence d'écriture entre les lignes 7 et 12, pour le même résultat, l'appel à la méthode `toString()` est automatiquement ajouté par le compilateur lorsqu'une donnée de type `String` est attendue et qu'une valeur de type objet est fournie (ici `tv1`). L'usage est donc, le plus souvent, de ne pas écrire explicitement l'appel à `toString()`, comme c'est le cas en ligne 12.*

Q 2 . Compilez la classe `TvMain.java`. Pour cela, placez-vous dans le dossier `poo/tp1/tv/` et exécutez :

```
.../poo/tp1/tv$ javac -sourcepath src src/TvMain.java -d classes
```

⚠ *Pour les prochaines compilations, vous devez être en mesure de produire cette commande par vous-même.*

Q 3 . Ce fichier étant compilé, on peut désormais en exécuter la méthode `main` à l'aide de la commande `java`. Toujours depuis le dossier `poo/tp1/tv/`, exécuter la commande :

```
.../poo/tp1/tv$ java -classpath classes TvMain
```

A l'aide de l'option « `-classpath classes` » on précise à la JVM où trouver les fichiers compilés nécessaires à l'exécution : `TvMain.class`, mais aussi `Tv.class`, car la classe `Tv` est utilisée par le programme.
Vous devez observer, grâce aux affichages réalisés dans la console, la trace d'exécution du programme.
Vous constatez aussi qu'à la fin de l'exécution de la méthode `main`, le programme et la JVM sont arrêtés.
Par abus de langage, on dira que l'on a « exécuté la classe `TvMain` ».

⚠ *On exécute une classe, on indique donc simplement en paramètre le nom de la classe. On peut noter la différence avec la commande de compilation : il n'y a pas ici d'extension, ni de chemin relatif d'accès.*

⚠ *Si les classes nécessaires à l'exécution se trouvent dans plusieurs dossiers, alors on sépare les différents chemins des dossiers par « `:` ». Par exemple : « `-classpath classes:other_folder:../here` ».*
Dans ce cas, sous Windows, la valeur du paramètre de l'option `-classpath` doit être encadrée par « `"` » et le séparateur à utiliser est « `;` » au lieu de « `:` » : « `-classspath "classes;other_folder,../here"` ».

⚠ Bonne pratique ⚠
Même si il est possible de définir une méthode `main` dans n'importe quelle classe, c'est une bonne pratique de définir une telle méthode dans une classe dédiée spécifique, comme c'est le cas de la classe `TvMain`.

Exercice 4 : Seconde compilation

A chaque fois que vous faites une modification dans votre code, il est nécessaire de le compiler à nouveau pour que la modification soit prise en compte à l'exécution : il faut mettre à jour les définitions des classes définies dans les fichiers `.class`, qui sont les seuls pris en compte par la JVM.

Q 1 . Modifiez le code source de la classe `Tv`, par exemple en modifiant, lignes 26 et 40, la valeur initiale de l'attribut `soundVolume` de 3 à 7, dans les constructeurs.
N'oubliez pas de sauvegarder votre fichier `Tv.java`.
Eventuellement, vous pouvez exécuter à nouveau la classe `TvMain` pour vérifier que la modification n'est pas prise en compte (tant que l'on n'a pas recompilé la classe).

Q 2 . Effacez tout le contenu du dossier `classes/` puis compilez le fichier `src/TvMain.java`, comme précédemment. Que constatez-vous dans le dossier `classes/` ?

⚠ *Lors de la compilation d'une classe, ici `TvMain`, le compilateur identifie toutes les dépendances vers d'autres classes, ici `Tv`, et compile également ces autres classes, éventuellement en compilant leurs propres dépendances, le cas échéant. On dit que le compilateur identifie la « clôture transitive » des dépendances du code et en compile l'ensemble.*

Q 3 . Exécutez la classe `TvMain` pour vérifier la prise en compte de la modification suite à cette nouvelle compilation.

⚠ *Pour compiler un ensemble de fichiers source, il est également possible d'utiliser le joker « `*` » dans le nom du fichier compilé :*

```
javac -sourcepath src src/*.java -d classes
```

Exercice 5 : Un second programme : un peu d'écriture de code

On s'intéresse maintenant aux classes du dossier `poo/tp1/library`.

Q 1 . Parcourez les codes de ces classes, notamment pour identifier

- **Q 1.1.** les attributs des objets `Book`, et leurs types ;
- **Q 1.2.** la structure de données qui permet de mémoriser les différents objets `Book` dans un objet `Library` ;
- **Q 1.3.** les méthodes de la classe `Library`.

Aidez-vous des documentations fournies pour chacun des éléments de code.

Q 2 . Compilez les trois classes fournies pour s'assurer que leur code est sans erreur de syntaxe (en respectant les bonnes pratiques).

Q 3 . Définissez une classe `LibraryMain` qui possèdera une méthode `main` dont l'exécution doit :

- **Q 3.1.** créer un objet `Book` correspond au livre « Le Seigneur des Anneaux » écrit en 1954 par JRR Tolkien, né en 1892 (vous devez créer un objet `Author`) ;
- **Q 3.2.** afficher sur la console la description de ce livre.

Q 4 . Compilez et exécutez votre code.

Q 5 . Complétez votre programme pour :

- **Q 5.1.** créer un second objet `Book` correspondant au livre « Bilbo le Hobbit » écrit en 1937, lui aussi par JRR Tolkien ;
- **Q 5.2.** créer un second objet `Book` correspondant au livre « Dune » écrit en 1965 Frank Herbert, né en 1920 ;
- **Q 5.3.** ajouter ces trois livres dans un objet `Library`, que vous aurez créé ;
- **Q 5.4.** afficher le contenu de cette bibliohtèque.

Q 6 . Dans la classe `Author`, on peut légitimement considérer qu'il manque certains accesseurs : pour le prénom et pour la date de naissance notamment.
Définissez ces accesseurs, que vous pourrez appeler `getFirstname` et `getBirthYear`.

Q 7 . Ajoutez également dans cette classe une méthode qui permet d'obtenir le nom complet d'un auteur (son nom et son prénom).

Q 8 . Compilez votre code.

Q 9 . Modifiez le code de la classe `Book` pour

- **Q 9.1.** ajouter aux objets de cette classe, un attribut `nbPages` correspondant au nombre de pages du livre ;
- **Q 9.2.** ajouter les accesseur (`getNbPages`) et modificateur (`setNbPages`) pour cet attribut ;
- **Q 9.3.** qu'à la construction d'un objet `Book` l'attribut `nbPages` soit initialisé à 0 ;
- **Q 9.4.** définir un second constructeur de la classe `Book` pour qu'il soit possible de construire un objet `Book` en précisant en paramètre une valeur pour cet attribut (en plus des autres valeurs déjà fournies par le premier constructeur) ;
- **Q 9.5.** adapter la méthode `toString()` pour que la description fournie en résultat fasse apparaître le nombre de pages de l'objet `Book` si celui n'est pas 0 ; sinon la description doit rester identique à l'existant.

Q 10 . Compilez votre code.

Q 11 . Modifiez la classe `LibraryMain` pour :

- **Q 11.1.** indiquer à sa création que « Le Seigneur des Anneaux » fait 1600 pages[3] ;
- **Q 11.2.** préciser après sa création le nombre de pages de « Bilbo le Hobbit » à 408 pages[4].

Q 12 . Compilez et exécutez votre code.

Q 13 . Complétez la classe `LibraryMain` avec un code qui calcule puis affiche le nombre de livres de l'objet `Library` créé pour lequel le nombre pages est connu.

Q 14 . Compilez et exécutez votre code.

[3] Édition intégrale chez *Pocket*

[4] Édition *Le livre de Poche*