

---

**UE Programmation Orientée Objet**


---

**TP 3**

Le but du TP est :

- ▷ de découvrir les outils `javac` et `java`
- ▷ d'apprendre à réutiliser des classes existantes en s'appuyant sur la documentation disponible. De plus, les classes que vous allez découvrir sont souvent utilisées.  
La documentation des API java, s'appelle la "javadoc" du nom de l'outil qui sert à les générer et que nous aborderons une autre fois.

**A partir de ce TP on abandonne BlueJ.**

Vous êtes libres d'utiliser l'éditeur qui vous convient (Emacs, Gedit et Kate ont l'avantage de disposer d'un "mode java" facilitant entre autres le travail d'indentation).

**Exercice 1 :** Premières compilation et exécution

Nous allons travailler dans cet exercice avec la classe `Stock` du TP de la semaine passée. Nécessairement le code source de cette classe se trouve dans un fichier qui s'appelle `Stock.java`. En JAVA il y a toujours correspondance entre le nom du fichier et le nom de la classe qu'il définit, et l'extension est nécessairement `java`.

- Q 1 .** Dans une commande shell, placez vous dans le répertoire contenant votre fichier `Stock.java`. Effacez le fichier `Stock.class` qui s'y trouve probablement.  
Pour compiler votre fichier, il faut exécuter la commande :

```
javac Stock.java
```

Nous verrons dans une prochaine séance de TP que les choses sont parfois un peu plus complexes pour la compilation.

Faites le. Si il n'y a pas d'erreur de compilation, votre répertoire contient maintenant le fichier `Stock.class` généré par la compilation. C'est ce fichier qui contient le bytecode JAVA utilisé par la machine virtuelle.

- Q 2 .** Exécuter un programme JAVA consiste à exécuter le corps d'une méthode particulière placée dans une classe. Cette méthode a **obligatoirement** et **rigoureusement** la signature :

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

Il peut y avoir (au plus) une méthode avec cette signature par fichier (définissant une classe). L'exécution d'une telle méthode contenue dans une classe `UneClasse` est provoquée par la commande (il faut que vous soyez dans le répertoire contenant le fichier `Stock.class`) :

```
java Stock
```

A nouveau, les choses seront nuancées dans le prochain TP.

Cette commande "lance" une machine virtuelle JAVA (JVM) qui exécute les lignes de code contenues dans la méthode `main` de la classe passée en argument (ici `Stock`). Le paramètre `args` de cette méthode contient le tableau des éventuels autres<sup>1</sup> arguments ajoutés à la ligne de commande.

Notez qu'il n'y a pas de ".java", ici on exécute une classe alors qu'avec `javac` on compile un fichier (qui permet de définir une classe). Pour cette commande, il est donc nécessaire de disposer du fichier `UneClasse.class`, mais pas nécessairement du source (.java).

Ajoutez à votre fichier `Stock.java` la définition :

```
public static void main(String[] args) {
    Stock unStock = new Stock();
    unStock.ajoute(10);
    System.out.println(unStock.getQuantite());
}
```

Sauvez, compilez puis exécutez.

**NB :** on peut ajouter les lignes suivantes au tout début de la méthode `main`, pour informer sur l'usage des arguments :

```
if (args.length <= 0) {
    System.out.println("usage : java Stock <unEntier>");
    System.exit(0);
}
```

- Q 3 .** Modifiez ainsi la méthode `main` de la classe `Stock` :

```
public static void main(String[] args) {
    Stock unStock = new Stock();
    unStock.ajoute(Integer.parseInt(args[0]));
    System.out.println(unStock.getQuantite());
}
```

**Commentaires :**

- "`Integer.parseInt`" prend en paramètre une chaîne (objet de type `String`) et renvoie un `int` correspondant à cette chaîne si elle représente un entier.  
exemple : `Integer.parseInt("42")` vaut l'`int` 42.

Il s'agit de l'utilisation de la méthode `static` de la classe `Integer` :

```
public static int parseInt(String s)
```

- `args[0]` est le premier argument de la ligne de commande.

Sauvez, compilez puis exécutez avec un argument supplémentaire qui devra correspondre à un entier, par exemple : `java Stock 12`.

**Exercice 2 :** La JavaDoc de l'API Java

En ouvrant dans votre navigateur le fichier `/opt/java/jdk1.6.0_02/docs/api/index.html` vous visualisez l'ensemble de la JAVADOC des paquetages fournis en standard avec le jdk<sup>2</sup> (**Ajoutez cette page à vos signets/marque-pages. En salle de TP, un lien direct est disponible dans la rubrique "Documents" sur le portail.**)

Oui, cette documentation est en anglais.

**Non**, il n'existera pas de version française. Il faut absolument vous habituer à lire et comprendre de la documentation technique en anglais.

En haut à gauche se trouve la liste des paquetages (notion présentée la semaine prochaine), en dessous la liste des classes du paquetage sélectionné (initialement toutes les classes) et dans la partie de droite la documentation de la classe sélectionnée (initialement la liste des paquetages).

- Q 1 .** Dans la liste des paquetages (en haut à gauche) sélectionner le paquetage `java.lang`

- Q 2 .** Dans la liste des classes (en bas à gauche) sélectionner la classe `String`.

Dans la zone "description de classes" (cadre de droite), la documentation est toujours organisée selon la même structure :

1. description de la classe
2. résumés :
  - (a) les attributs (seuls ceux qui seraient publics)
  - (b) les constructeurs (publics)
  - (c) les méthodes (publiques)
3. détails :
  - (a) les attributs : description
  - (b) les constructeurs : présentation et description des paramètres
  - (c) les méthodes : présentation, description des paramètres et des valeurs de retour

**Exercice 3 :** Manipulation sur des mots.

Nous cherchons dans cet exercice à modéliser des mots sur lesquels nous ferons un certain nombre d'opérations.

Pour pouvoir implémenter la classe décrite ci-dessous, vous devrez utiliser les fonctionnalités de la classe `String`<sup>3</sup>.

La classe `String` permet de représenter des chaînes de caractères immuables (donc constantes), aucune méthode de la classe ne permet de modifier la chaîne manipulée.

N'oubliez pas cependant que les chaînes de caractères sont des objets!

Il est donc conseillé de jeter un oeil à la documentation de cette classe...

Pendant la documentation de cette classes est volumineuse, pour éviter que vous ne vous perdiez dans le grand nombre de méthodes disponibles, pour chaque question il sera indiqué les (nouvelles) méthodes dont vous devriez consulter la documentation avant de chercher à répondre à la question car elles devraient

<sup>2</sup>Java Development Kit.

<sup>3</sup>Un certain nombre des méthodes demandées pourraient être simplifiées si l'on utilisait les fonctionnalités offertes par la classe `StringBuffer` du paquetage `java.lang`. Cependant on ne les utilisera pas a priori dans ce TP. Vous êtes cependant fortement invités à regarder la documentation et à réfléchir à comment vous auriez pu utiliser cette classe - par exemple pour la méthode `inverse`, il y a ce qu'il faut dans `StringBuffer` pour peu que l'on comprenne - ce qui n'est pas très compliqué - comment passer d'une instance de `String` à une instance "équivalente" de `StringBuffer` et réciproquement.

<sup>1</sup>A la différence de `ocaml` où le nom du programme est repris dans le tableau d'arguments, le nom de la classe lui n'est pas repris dans ce tableau en java.

vous aider. Mais il est sans doute possible de trouver des solutions qui n'utilisent pas ces méthodes, n'en faites donc pas une fixation.

Ainsi, dans la suite, la notation (**dans NomDeClasse**) **nomMethode** signifie “consultez la documentation pour la méthode de nom *nomMethode* dans la classe *NomDeClasse*”, cela n'implique pas que vous deviez obligatoirement utiliser cette méthode pour traiter la question, mais qu'il est possible qu'elle vous soit utile pour cette question (et/ou une suivante).

**Q 1 .** Créez une classe `Mot`. Son état sera représenté par une chaîne de caractères : définissez un attribut `valeur` qui contiendra cette chaîne de caractères. Définissez le constructeur qui prendra en paramètre une chaîne de caractères pour initialiser l'attribut.

**Attention : il est important de ne pas confondre la classe `Mot` et son attribut `valeur`. Dans la suite du sujet quand on parlera d'un mot, il faudra comprendre une instance de la classe `Mot`. Cependant il est probable que la plupart des traitements se feront sur l'attribut `valeur` qui est une chaîne de caractères.**

Vous devez donc être vigilant, rigoureux et attentif et bien réfléchir à la notion que vous référez/manipulez : l'objet `Mot` ou son attribut `valeur`.

Les constantes chaînes de caractères seront notées entre “ ”.

Les méthodes demandées dans les questions qui suivent sont des méthodes de la classe `Mot`. L'écriture de certaines est triviale, pour d'autres un peu plus de réflexion sera nécessaire.

**Q 2 .** Définissez une méthode `longueur`, sans paramètre, qui a pour résultat la longueur du mot.

signature : `public int longueur()`

(dans `String`) `int length()`,<sup>4</sup>

**Q 3 .** Définissez une méthode `toString` qui renvoie une chaîne de caractères correspondant au mot. Cette méthode est automatiquement utilisée lors de l'affichage par `System.out.println`. Ainsi si `m` est une référence de type `Mot` initialisée avec une instance de cette classe, les instructions `System.out.println(m.toString());` ou `System.out.println(m);` sont équivalentes. Dans la seconde, l'invocation de la méthode `toString` est implicite et automatiquement gérée par JAVA.

**Q 4 .** Définissez une méthode `memoMot` qui renvoie `true` ssi le mot en paramètre est identique au mot courant (c-à-d celui qui invoque la méthode, c-à-d `this`).

(dans `String`) `boolean equals(Object o)`

**Q 5 .** Il faut tester les méthodes ci-dessus. La notion de test est impérative et très importante et il est indispensable que vous testiez **systématiquement** chacune de vos méthodes.

**Il faudrait en fait que vous écriviez les tests et la javadoc avant le code!** Essayez vous allez voir, cela ne complique pas les choses, au contraire, cela vous oblige à savoir précisément avant ce que vous voulez coder...

Pour réaliser des tests, ajoutez à votre classe `Mot` une méthode `main`<sup>5</sup> qui :

1. déclare une référence de type `Mot`,
2. initialise cette référence par une instance de la classe `Mot` créée à partir de la valeur du premier argument passée en ligne de commande (`args[0]`),
3. invoque la méthode `longueur()` sur cette référence et affiche le résultat (utilisez `System.out.println`).
4. affiche le résultat de l'appelle à `memoMot` avec le mot `timoleon` (obtenu par : `new Mot("timoleon")`).

Testez ensuite par : “java `Mot uneChainePourLeMotATester`” (exemple java `Mot radar` ou java `Mot timoleon`).

Pour chacune des méthodes suivantes, vous effectuerez des tests similaires en complétant petit à petit le code de la méthode `main`, mais sans effacer les test qui y sont déjà, tous les tests doivent pouvoir être refaits.

Veillez bien dans vos différents tests à contrôler les différents cas de figure et à ne pas vous limiter à ni oublier des cas particuliers.

**Q 6 .** Définissez une méthode, `nbDOccurrencesDuChar` qui calcule le nombre d'occurrences d'un caractère donné (en paramètre) dans le mot.

(dans `String`) `int indexOf(char c)`,  
(dans `String`) `int indexOf(char c, int fromIndex)`  
(dans `String`) `char charAt(int index)`,

**Q 7 .** Pensez aux tests !

**Q 8 .** Après en avoir écrit la documentation, définissez une méthode `inverse` qui retourne un objet `Mot` (pas `String`!) dont la valeur (l'attribut) est l'inverse de la valeur du mot initial (c-à-d. du mot sur lequel cette méthode a été appelée).

**Q 9 .** Définissez une méthode `contient` qui teste si un mot donné est un “sous-mot” du mot courant. Par exemple les mots correspondants à `tim`, `mole` et `leon` sont des sous-mots du mot `timoleon`. Ce n'est pas le cas des mots `ile` ou `hobbit`.

(dans `String`) `int indexOf(String str)`,  
(dans `String`) `int indexOf(String str, int fromIndex)`  
(dans `String`) `String substring(int beginIndex)`,  
(dans `String`) `String substring(int beginIndex, int endIndex)`

**Q 10 .** Définissez une méthode `rimeAvec`, qui regarde si le mot rime avec un autre mot. On dira que deux mots riment si leur 3 derniers caractères sont identiques (il faut donc au moins 3 caractères, sinon le résultat vaut forcément faux...).

(dans `String`) `boolean endsWith(String suffix)`

**Q 11 .** Et vos tests ?

**Q 12 .** Définissez une méthode `extraitAvant` qui retourne un tableau de deux mots, le premier correspond au plus petit préfixe du mot précédant la première occurrence d'un caractère donné (passé en paramètre) inclus, le second correspond au reste du mot (sans le caractère). Si le caractère n'existe pas dans le mot, le premier élément du tableau résultat est le mot vide et le second est le mot lui-même.

signature : `public Mot[] extraitAvant(char c)`

Par exemple pour le mot `timoleon` et le caractère `o`, on veut obtenir `timo` et `leon`, alors que pour le caractère `i` on obtient `ti` et `moleon`.

**Q 13 .** Définissez une méthode `estPalindrome` qui teste si le mot est un palindrome. Une telle méthode renvoie évidemment un booléen.

**Q 14 .** Définissez une méthode `estAnagramme` qui indique si un mot donné est un anagramme du mot (les deux mots ont les mêmes caractères en même nombre).

Dans vos tests pensez à vérifier par exemple que votre méthode-prédicat est bien symétrique, c-à-d que si `m1` est anagramme de `m2`, `m2` l'est aussi pour `m1`... Notamment faites le test avec les mots `abracadabra` et `crabada`.

Il est important lorsque l'on parle de tests de prévoir un jeu de tests qui couvre l'ensemble des cas de figure, dans la mesure du possible. C'est ce qui rend la construction de tels jeux de tests parfois difficile. Cette étape est néanmoins indispensable.

**Q 15 .** Définissez une méthode qui indique si un nom est un nom propre, c'est-à-dire commence par une majuscule.

(dans `Character`) `static boolean isUpperCase(char ch)`

NB : “upper-case” signifie “majuscule” en anglais...

<sup>4</sup>à ne pas confondre avec la propriété `length` des tableaux!

<sup>5</sup>Attention à la signature de cette méthode qui doit être scrupuleusement respectée!