
# Mise à niveau en Java

Syntaxe de base et utilisation d'objets

Guillaume Wisniewski
guillaume.wisniewski@limsi.fr
Septembre 2015

# Table des matières

# 1 Syntaxe de base

Compétences vues :
— manipulation d'Eclipse ;
— manipulation de variables et de structures de contrôle simples ;
— écriture de fonctions simples ;
— utilisation de la classe `String`.

## 1.1 Mise en bouche

Écrivez une classe `MiseEnBouche` comportant les fonctions suivantes :
— une méthode `max` prenant en paramètre un tableau d'entiers et retournant le plus grand élément de ce tableau ;
— une méthode `maxForEach` similaire à la fonction précédente, mais utilisant une boucle `foreach` ;

## 1.2 Calculs de moyenne

Écrivez les fonctions suivantes :
— une méthode `moyenne` prenant en paramètre un tableau d'entiers et retournant un `double` représentant la moyenne de ce tableau ;
— une méthode `variance` prenant en paramètre un tableau d'entiers et retournant sa variance ;
— (†) une méthode `moyenneOnLine` calculant la moyenne et la variance en *une seule passe* sur les données. On pourra s'inspirer de l'algorithme décrit ici.

## 1.3 Vérification d'adresse IP

Une adresse IP est définie par quatre nombres compris entre 0 et 255, séparés par des points. Par exemple : 212.85.150.134. L'objectif de cet exercice est de vérifier si une chaîne de caractères donnée définie bien une adresse IP. Pour cela il faut vérifier que :
— elle comporte 3 points ;
— elle commence et se termine par un nombre (.123. n'est pas une adresse IP) ;
— les caractères situés entre les points sont des chiffres compris entre 0 et 255.

Écrire une fonction permettant de vérifier tous ces éléments. On pourra utiliser la fonction le code suivant pour convertir une chaine en entier :

```
String s = "100";
Integer a = Integer.parseInt(s);
// a = 100
a = Integer.parseInt("pouet")
// a = null
```

## 1.4 Calculatrice élémentaire

On considère une chaîne de caractères, entrée par l'utilisateur, du type `"17 + 3 * 5"`. On supposera que les opérations ($+$, $-$, $/$ et $\times$) sont toujours précédées et suivies d'un unique espace. L'objectif de cet exercice est d'évaluer la valeur numérique représentée par cette expression.

La première étape de l'évaluation consiste à transformer la chaîne de caractères en un tableau d'éléments, chaque éléments étant soit un nombre, soit une des quatre opérations.

On ne considérera pas l'ordre de priorité des opérations. Ainsi, le résultat de l'évaluation de l'expression précédente sera 100 ($(17 + 3) \times 5$). Il est alors possible d'utiliser l'algorithme suivant pour évaluer une expression :

- on parcourt l'ensemble des éléments constituant l'expression à évaluer
- soit l'élément courant est une opération ; on stocke alors la valeur de cette opération dans une variable ;
- soit l'élément est un nombre. Deux cas sont alors envisageables :
  - soit on n'a pas encore lu d'opération ; le résultat courant est alors initialisé à la valeur du nombre ;
  - soit on a déjà lu une opération ; il faut alors effectuer l'opération entre ce nombre et le résultat courant

Sur l'exemple précédent (évaluation de `"17 + 3 * 5"`), les opérations effectuées sont les suivantes :

- lecture de « 17 » ; on le stocke dans la variable décrivant le résultat ;
- lecture de « $+$ » ; on le stocke dans la variable décrivant l'opération ;
- lecture de « 3 » ; comme on a déjà lu une opération, on effectue « 17 $+$ 3 » et on stocke le réultat dans la variable décrivant le résultat ;
- lecture de « $\times$ » ; on le stocke dans la variable décrivant l'opération ;
- lecture de « 5 » ; comme on a déjà lu un résultat, on effectue l'opération « résultat $\times$ 5 ». Le résultat est donc 100.

1. Écrivez le code correspondant. La méthode développée aura la signature suivante :
   ```
   public static double calculate(String arg)
   ```
   et sera dans une classe nommée `Calto`.
2. (†) On souhaite désormais adapter l'algorithme précédent pour tenir compte des priorités entre les opérations. Comment faire ? La méthode développée aura la même signature que la méthode `calculate` et s'appellera `calculateWithPriority`.

**Indication :** On peut remarquer qu'il est possible de déterminer si une opération doit être effectuée ou non à partir du moment où l'on connaît l'opération suivante.

# 2 Tests unitaires et manipulation de listes

Compétences vues :
— utilisation d'`ArrayList` ;
— conception d'algorithme ;
— écritures de tests unitaires

## 2.1 Harry Potter [1]

Un libraire en ligne veut faire des promos de Noël sur la collection « Harry Potter ». Il y a actuellement 5 tomes différents disponibles, identifiés par les codes 0, 1, 2, 3, et 4. Chaque tome vendu séparément coûte 8 euros. Si un client achète deux tomes différents en même temps, il bénéficie d'une réduction de 5% sur ces deux tomes. S'il achète trois tomes différents, la réduction est de 10% sur ces tomes. Quatre tomes, réduction de 20% ; et si le client achète le grand total (5 tomes différents), il bénéficie royalement de 25% de réduction sur ces 5 tomes.

Il est possible de bénéficier de plusieurs offres de réduction pour un même achat groupé, mais chaque livre acheté ne contribue qu'à une seule réduction à la fois. Lorsque deux combinaisons de réduction sont possibles — par exemple, lorsque le client achète 5 tomes différents, on peut soit appliquer la réduction pour les 5 tomes, soit la réduction pour 2 tomes et la réduction pour 3 tomes — c'est celle qui offre la plus grande réduction globale qui est retenue. Dans l'exemple cité au dessus, on choisira donc la réduction s'appliquant à 5 tomes différents.

Il s'agit d'implémenter la fonction `montantTotal` qui, prenant en entrée une `ArrayList` contenant les identifiants des livres qu'un client souhaite acheter, calcule le montant total à payer.

Cet exercice parait, au premier abord, simple, mais sa résolution dans le cas général est un problème d'optimisation combinatoire (relativement) complexe. Pour vous convaincre de la complexité du problème, essayez de calculer le prix que devra payer une personne achetant 2 copies du premier, du second et du troisième livre et 1 copie du quatrième et du cinquième livre. [2]

Plutôt que de chercher un algorithme suffisamment « intelligent » pour résoudre le problème général, on propose de résoudre ce problème de manière incrémentale en considérant des cas particuliers de complexité croissante : achat composé d'aucun livre, d'un

1. d'après Harry Potter Kata
2. réponse : 51,20€

seul livre, de deux livres identiques, de deux livres différents, ... En développant les tests au fur et à mesure, on peut assurer que les modifications de l'algorithme nécessaire aux traitements des cas les plus complexes ne « casseront » pas la résolution des cas les plus simples.

Dans la suite du sujet on vous demande de modifier la fonction `montantTotal` de la manière *la plus simple possible* permettant de passer le test unitaire donné (et tous les tests précédents).

1. teste le cas d'un achat sans réduction :

| résultat attendu | panier |
|---|---|
| 0 | `{}` |
| 8 | `{0}` |
| 8 | `{1}` |
| 8 | `{2}` |
| 8 | `{3}` |
| 8 | `{4}` |
| $8 \times 2$ | `{0, 0}` |
| $8 \times 3$ | `{1, 1, 1}` |

2. teste le cas des réductions simples :

| résultat attendu | résultat obtenu |
|---|---|
| $8 \times 2 \times 0.95$ | `{0, 1}` |
| $8 \times 3 \times 0.9$ | `{0, 2, 4}` |
| $8 \times 4 \times 0.8$ | `{0, 1, 2, 4}` |
| $8 \times 5 \times 0.75$ | `{0, 1, 2, 3, 4}` |

3. teste les cas simples de réductions combinées :

| résultat attendu | résultat obtenu |
|---|---|
| $8 + (8 \times 2 \times 0.95)$ | `{0, 0, 1}` |
| $2 \times (8 \times 2 \times 0.95)$ | `{0, 0, 1, 1}` |
| $8 \times 4 \times 0.8) + (8 \times 2 \times 0.95)$ | `{0, 0, 1, 2, 2, 3}` |
| $8 + (8 \times 5 \times 0.75)$ | `{0, 1, 1, 2, 3, 4}` |

4. teste les cas plus compliqués de réductions combinées :

| résultat attendu | résultat obtenu |
|---|---|
| $2 \times (8 \times 4 \times 0.8)$ | `{0, 0, 1, 1, 2, 2, 3, 4}` |
| $3 \times (8 \times 5 \times 0.75) + 2 \times (8 \times 4 \times 0.8)$ | `{0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4}` |

# 3 Utilisation d'objets

Compétences vues :
— lecture de fichier ;
— écriture de fonctions ;
— utilisation de la classe `ArrayList` et `HashMap`.

## 3.1 Compter les mots

On souhaite, dans cet exercice, écrire une méthode `countWords` qui prend en paramètre une chaîne de caractères représentant un texte et qui retourne un dictionnaire associant à chaque mot apparaissant dans le texte son nombre d'occurences. On supposera que les mots sont séparés par des espaces.

## 3.2 Molécules [1]

Une entreprise de nanotechnologie `Molecule` vient de vous engager pour écrire un programme capable de gérer l'inventaire de l'entreprise. Ce programme doit :
— lire dans un fichier la liste des atomes disponibles dans les stocks de l'entreprise ;
— lire dans un autre fichier les « recettes » indiquant quels atomes entrent dans la composition des différentes molécules fabriquées par l'entreprise ;
— produire la liste des molécules que l'entreprise peut produire.
Le fichier décrivant les molécules a le format suivant :

```
# Comments start with '#' and go to the end of the line.
# Blank lines (like the one below) are allowed.

helium : He 1               # molecule name, colon, atom type, number of atoms
ammonia : N 1 H 3           # molecules may contain many types of atoms
salt : Na 1 Cl 1            # atom names may be one or two characters long
lithium hydride : Li 1 H 1  # molecule names may be several words long
```

Le fichier donnant la liste des atomes disponibles dans les stocks de l'entreprise à le format suivant :

```
# Once again, comments and blank lines are allowed.

He 3                        # atom name, number available
```

1. sur une idée originale de G. Wilson

```
H 2
N 5
Li 1
```

Avec ces deux fichiers, la sortie serait :

```
helium
lithium hydride
```

La résolution de ce problème se fera grâce à quatre fonctions définies dans une classe `Molecule` :

1. une fonction `readInventory(String fileName)` qui analyse le contenu d'un fichier décrivant l'inventaire et retourne un dictionnaire dont les clés sont des atomes et les valeurs associées le nombre d'atomes de ce type disponibles. Pour l'exemple ci-dessus, ce dictionnaire aurait pour clés `He, H, N` et `Li`, les valeurs associées étant respectivement 3, 2, 5 et 1.
2. une fonction `readFormulas(String fileName)` qui, à partir du contenu d'un fichier, renvoie un dictionnaire dont les clés sont les noms des molécules et dont les valeurs sont des dictionnaires décrivant combien d'atome de chaque sorte entre dans la composition de la molécule. Pour le fichier donné en exemple ci-dessus, le dictionnaire à retourner serait :
   ```
   {'helium' : {'He' : 1},
    'ammonia' : {'N' : 1, 'H' : 3},
    'salt' : {'Na' : 1, 'Cl' : 1},
    'lithium hydride' : {'Li' : 1, 'H' : 1}}
   ```
3. une fonction `makeable(inventory, formulas)` [2] qui a partir des deux dictionnaires précédents renvoie l'ensemble des molécules qui peuvent être fabriquées.

**Remarque** : essayez de partir du format de fichier le plus simple (une seule ligne, pas de commentaires) et de le complexifier petit à petit.

2. Les types des paramètres sont à déterminer

# 4 Définition d'objets et héritage

On souhaite développer une application permettant de gérer une médiathèque qui contient des livres, des CDs et des DVDs. Chacun de ces média sera naturellement modélisé par une classe.

## 4.1 Médias

Tous les médias sont décrits par un titre et un nom d'auteur. Les utilisateurs peuvent donner leur avis sur la qualité d'un média en leur attribuant une note comprise entre 0 et 5. Il est possible :

— de représenter un média par une chaine de caractères (méthode `toString`). Le format de cette représentation est :

```
"Titre" par Auteur
```

— de donner son avis sur le média grâce à la méthode `vote(int note)`, `note` devant être comprise entre 0 et 5.

— d'obtenir la moyenne des votes reçus par la méthode `moyenneNotes()`. Si le média n'a reçu aucune note, cette méthode renvoie 0.

Les DVD possède une caractéristique supplémentaire permettant la gestion des zones (une zone est décrite par un entier compris entre 0 et 8, la zone 0 indique que le DVD est lisible dans toutes les zones). La méthode `readable(int[] zones)` renvoie `true` si le DVD est lisible dans une des zones passées en paramètre, `false` sinon.

La description du CD comporte également le format du contenu décrit par une chaine de caractères (`"CD musical"`, `"OGG"`, ou `"MP3"`). La méthode `toString` est modifiée pour indiquer le format et renvoie, par exemple :

```
"Some Kind Of Trouble" par James Blunt [CD musical]
```

La description des livres ne comporte aucun élément particulier.

1. Donnez le code de la classe `Livre`
2. On souhaite maintenant développer la classe `CD`. Comme cette classe comportera de nombreuses méthodes communes avec la classe `Livre`, on souhaite introduire une classe mère « générique », `Media`.
   — Modifiez la classe `Livre` pour qu'elle hérite d'une classe `Media` et réorganisez votre code de manière à tirer profit de cette nouvelle organisation.
   — Écrivez le code de la classe `CD`.
3. Donnez le code de la classe `DVD`.

## 4.2 Médiathèque

Dans une classe `Mediatheque`, la base de données de la médiathèque est une `ArrayList` de `Medium`.

1. Écrivez une méthode `add(Media m)` qui permet d'ajouter un média à la collection.
2. Écrivez une méthode `filtre(String critere, String valeur` qui retourne un `ArrayList<Media>` contenant l'ensemble des média vérifiant le critère passé en paramètre. Ce critère peut porter :
   — sur l'auteur ;
   — sur le titre ;
   — sur le type de média (on pourra utiliser l'instruction `nomObjet instanceof nomClasse` qui retourne `true` si l'objet nomObjet est une instance de la classe nomClasse ou de l'une de ses classes filles)

   Par exemple : `m.filtre("titre", "toto")` renverra l'ensemble des média dont le titre est toto et `m.filtre("media", "CD")` renverra l'ensemble des CD de la collection.

# 5 Extraction de lexique bilingue

## 5.1 Extraction de lexique

L'objectif de ce TP est d'extraire automatiquement un lexique bilingue d'un ensemble de textes. Un lexique bilingue est un dictionnaire (aussi bien au sens informatique qu'au sens « général ») qui associe à un mot d'une langue source la liste de ses traductions possibles dans une langue cible. La Table 5.1 donne un extrait d'un lexique français–anglais.

| | |
|---|---|
| cuisine | cooks, cook, cooking, kitchen, food... |
| comprendre | understand, understood, understanding, realize, ... |
| être | human, being, be, ... |
| économie | free-market, Economy, economics, market, economy, ... |

TABLE 5.1 – Extrait d'un lexique français–anglais

L'extraction sera *non supervisée* : la méthode utilisée prendra en entrée uniquement un ensemble de paire de phrases (une phrase en français et sa traduction en anglais). On appelle « corpus parallèle » un tel corpus et on qualifie de « parallèles » des phrases qui sont traductions l'une de l'autre. La Table 5.2 donne un exemple de corpus parallèle.

L'approche proposée est fondée sur une théorie linguistique, l'*hypothèse distributionnelle* : le sens d'un mot peut se déduire directement du contexte dans lequel celui-ci apparait. La généralisation au cas bilingue de cette hypothèse peut se formuler de la manière suivante : un mot français et un mot anglais qui *co-occurent* (apparaissent dans

| |
|---|
| Living on my own, I really miss my Mom's **cooking**. |
| Vivant seul, la **cuisine** de ma mère me manque. |
| She left the **kitchen** with the kettle boiling. |
| Elle quitta la **cuisine** avec la bouilloire. |
| Is there any coffee in the **kitchen**? |
| Y a-t-il encore du café dans la **cuisine** ? |
| **Cooking** runs in my family. |
| La **cuisine** c'est de famille. |
| Both boys and girls should take **cooking** class in school. |
| Garçons et filles devraient suivre des cours de **cuisine** à l'école. |

TABLE 5.2 – Exemple d'un corpus parallèle français–anglais

| |
|---|
| Πάω στο σπίτι μας. |
| Je vais chez nous (*lit.* dans notre maison). |
| Το σπίτι μου είναι μεγάλο. |
| Ma maison est grande |
| Το σπίτι της Έλλης είναι κοντά στην παραλία. |
| La maison d'Elli est à côté de la plage. |
| Με λένε Έλλη. |
| Je m'appelle Elli. |
| Ένα σπίτι του χωριού κάηκε |
| Une maison du village a brûlé. |
| Αγαπώ την Έλλη. |
| J'aime Elli. |

TABLE 5.3 – Exemple d'un corpus parallèle grec–français

une paire de phrases parallèles) fréquemment ont une grande chance d'être traduction l'un de l'autre. Ainsi sur l'exemple du corpus de la Table 5.2, il est naturel de supposer que *cuisine* se traduit soit par *cooking*, soit par *kitchen* puisque ce sont les seuls mots qui apparaissent dans toutes les traductions.

— En considérant le corpus français–grec, décrit Table 5.3 pouvez-vous donner (en les justifiants) les traductions en grec des mots `Elli`, `est` et `maison`.

## 5.2 Approche naïve

Vous trouverez sur le site du cours deux fichiers contenant le roman *Voyage au centre de la Terre* en anglais et en français. Les documents ont été pré-traités pour :

— être alignés au niveau des phrases : la $i^{\text{e}}$ ligne du fichier en français est la traduction de la $i^{\text{e}}$ ligne du fichier anglais.

— segmentés en mots : cette segmentation consiste à séparer certains signes (par exemple « `à_l'école.` » est ré-écrit en « `à_l'_école_.` ») et à en regrouper d'autres (« 100 000 » est récrit en « 100000 »).

L'extraction d'un lexique à partir de ce corpus parallèle repose sur la construction d'un table de co-occurences. Cette table peut-être modélisée par une variable de type `HashMap<String, <HashMap<String, Integer>` qui associe à un mot français (la $1^{\text{re}}$ clef), une `HashMap` dont les clefs sont l'ensemble des mots anglais co-occurant avec le mot français et les valeurs le nombre de paires de phrases dans lesquelles le mot français et le mot anglais apparaissent tous les deux. Par exemple, si le corpus est constitué de deux phrases :

| doc n° 1 | doc n° 2 |
|---|---|
| la vache et le veau | the cow and the calf |
| le chien et le chat | the dog and the cat |

la table extraite sera :

```
{'chat': {'and': 1, 'the': 1, 'dog': 1, 'cat': 1}),
 'chien': {'and': 1, 'the': 1, 'dog': 1, 'cat': 1}),
 'et': {'and': 2, 'calf': 1, 'cat': 1, 'cow': 1, 'dog': 1, 'the': 2},
 'la': {'and': 1, 'the': 1, 'cow': 1, 'calf': 1}),
 'le': {'and': 2, 'calf': 1, 'cat': 1, 'cow': 1, 'dog': 1, 'the': 2},
 'vache': {'and': 1, 'calf': 1, 'cow': 1, 'the': 1},
 'veau': {'and': 1, 'the': 1, 'cow': 1, 'calf': 1})}
```

— Quel est l'intérêt de la segmentation en mot ?
— Écrivez une méthode `countSentencesWithWord` dont l'entrée est un nom de fichier et qui renvoie une `HashMap` associant à chaque mot du fichier le nombre de phrases dans lequel il apparait. **Attention** : on cherche à déterminer le nombre de phrases dans lequel un mot apparait, et même si un mot apparait plusieurs fois dans une même phrase, il ne faut le compter qu'une fois. Il est possible de supprimer les doublons d'une liste en utilisant l'instruction :

```
HashSet<String> uniqWord = new HashSet<String>(Arrays.asList(tab));
```

qui à partir d'un tableau de `String` (par exemple `["a", "b", "a"]`) construit une *liste* sans doublons (si on reprend l'exemple précédent `["b", "a"]`).
— Écrivez une méthode `buildContTable` qui prend en argument deux noms de fichiers décrivant un corpus parallèle et retourne la table de co-occurence de ce corpus. À nouveau, il faut penser à supprimer les doublons à l'intérieur d'une phrase avant de réaliser les comptages.
— À l'aide de la méthode `printSortedCoocTable`, afficher la table de co-occurences par ordre de fréquence décroissant.
— Interprétez le résultat obtenu : est-ce que cette méthode permet de construire un « bon » lexique ?

**Remarque** : pensez à regarder le squelette qui vous est fourni sur le site du cours.

## 5.3 Tests statistiques

L'approche de la section précédente n'est utilisable que si l'on dispose d'un moyen de distinguer les associations « porteuses de sens » (celles que l'on observe dans un corpus parce que les mots sont traduction l'un de l'autre) des associations dues au hasard (par exemple, celles que l'on observe que parce que notre corpus est trop petit ou à cause des mots fréquents).

Les *tests de significativité* sont des outils statistiques qui répondent à cet objectif. Dans la suite du TP nous proposons d'utiliser un de ces tests, le *likelihood ratio*, pour filtrer la table des co-occurences construite dans la partie précédente et ne garder que les associations les *significatives*. Ce test évalue l'intérêt d'une association entre le mot $a$ et le mot $b$ à partir de sa table de contingence. Cette table décrit :

— $n(a, b)$ le nombre de paires de phrases dans lesquelles $a$ et $b$ co-occurent ;
— $n(\neg a, b)$ le nombre de paires de phrases dans lesquelles $b$ apparait, mais pas $a$ ;

— $n(a, \neg b)$ le nombre de paires de phrases dans lesquelles $a$ apparait, mais pas $b$ ;
— $n(\neg a, \neg b)$ le nombre de paires de phrases dans lesquelles ni $a$, ni $b$ n'apparaissent.
La table 5.4 résume ces notations.

| | nbre phrase avec $a$ | nbre phrase sans $a$ | |
|---|---|---|---|
| nbre phrase avec $b$ | $n(a, b)$ | $n(\neg a, b)$ | $n(b)$ |
| nbre phrase sans $b$ | $n(a, \neg b)$ | $n(\neg a, \neg b)$ | $n(\neg b)$ |
| | $n(a)$ | $n(\neg a)$ | $N$ |

TABLE 5.4 – Table de contingence

Le *likelihood ratio* est défini par :

$$G^2 = 2 \cdot N \left[ \sum_{a? \in \{a, \neg a\}} \sum_{b? \in \{b, \neg b\}} p(a? \text{ et } b?) \cdot \log \frac{p(a? \text{ et } b?)}{p(a?) \cdot p(b?)} \right]$$

où $p(x?)$ est la fréquence de l'événement $x?$ (le rapport entre le nombre de fois où $x?$ est vrai et le nombre de fois où $x?$ est vrai *ou* faux). Par exemple, $p(a) = \frac{n(a)}{N}$ et $p(a, \neg b) = \frac{n(a, \neg b}{N}$.

— On note $n(a)$ (resp. $n(b)$) le nombre de phrases dans lesquelles $a$ (resp. $b$) apparaît et $N$ le nombre total de phrases. Exprimez $n(\neg a, \neg b)$, $n(\neg a, b)$ et $n(a, \neg b)$ en fonction de $n(a)$, $n(b)$, $n(a, b)$ et $N$.
— Écrivez une méthode qui calcule le *likelihood ratio* d'une paire mot français / mot anglais. Cette méthode prendra en paramètre :
  — le nombre de phrases du corpus ;
  — le nombre de phrases dans lesquelles le mot apparait ;
  — le nombre de phrases dans lesquelles le mot anglais apparait ;
  — le nombre de phrases parallèles dans lesquelles la paire de mot apparait.
— Écrivez une méthode qui calcule le likelihood ratio de l'ensemble des paires de mots du corpus. Cette méthode renverra une instance de `HashMap<String, Double>` qui associe à une paire de mots (obtenu, par exemple, en concaténant le mot français et le mot anglais) son score. Seules les paires dont le likelihood ratio n'est pas infini devront être ajoutées au résultat. Pour savoir si un `Double` est infini, on peut utiliser l'instruction

```
lr.equals(Double.NaN)
```

— Afficher les associations par ordre décroissant de *likelihood ratio* croissant. Que peut-on en conclure ?

# 6 Rappels d'algorithmique

## 6.1 Complexité

On considère la fonction suivante :

```
public static int something(int[] a) {
    int temp = 0;
    for (int i = 0; i < a.length; i++) {
        for (int j = i +  1; j < a.length; j++) {
            if (Math.abs(a[j] - a[i]) > temp) {
                temp = Math.abs(a[j] - a[i]);
            }
        }
    }
    return temp;
}
```

1. Que fait cette fonction ?
2. Quelle est sa complexité ?
3. Proposez une méthode plus efficace.

## 6.2 Tri par sélection

On considère la manière suivante de trier un tableau $A$ : on commence par trouver le plus petit élément de $A$ et on le permute avec $A[0]$. On trouve ensuite le deuxième plus petit élément de $A$ et on le permute avec $A[2]$. On continue de cette manière pour tous les éléments du tableau.

1. Donnez le code java mettant en œuvre le principe décrit ce-dessus.
2. Donnez la complexité de cette algorithme dans le cas le plus favorable et dans le cas le plus défavorable.

## 6.3 Méthode de Horner

Dans cet exercice on s'intéresse à l'évaluation d'un polynôme $P = a_0 + a_1 \times x + a_2 \times x^2 + ... + a_n \times x^n$ en un point donné. On supposera, dans la suite que les coefficients du polynômes sont stockés dans un tableau.

1. Écrivez le code java qui permet de calculer la valeur d'un polynôme en un point donné.
2. Quelle est la complexité de cette fonction ?
3. Il est possible de réduire cette complexité en utilisant la méthode de Horner. Cette méthode repose sur l'observation qu'un polynôme $P$ peut se factoriser en $P = a_0 + x \times (a_1 + x \times (a_2 + ... + x \times (a_{n-1} + x \times a_n)))$. Écrivez une seconde méthode d'évaluation fondée sur cette factorisation. Quelle est sa complexité ?

## 6.4 (††) Inversions

Soit $A[0..n-1]$ un tableau de $n$ nombres distincts. Si $i < j$ et $A[i] > A[j]$, on dit que le couple $(i, j)$ est une inversion de $A$.

1. Donnez les cinq inversions du tableau `2,3,8,6,1` ;
2. Quel est le tableau dont les éléments appartiennent à l'ensemble $[\![1, n]\!]$ qui a le plus d'inversions ? Combien en possède-t-il ?
3. Donnez un algorithme qui détermine en un temps $\mathcal{O}(n \times \log_2 n)$ dans le cas le plus défavorable, le nombre d'inversions présentes dans un nombre quelconque de $n$ éléments.

Indication pour la dernière question : faites le lien avec la fonction merge du tri fusion.

## 6.5 Exponentiation rapide

1. Donnez le code d'une méthode récursive permettant de calculer $x^n$
2. Quelle est la complexité de cette méthode ?
3. Une méthode d'exponentiation plus efficace repose sur l'observation que :
   — si $n$ est pair alors $x^n = (x^2)^{\frac{n}{2}}$. Il suffit alors de calculer $y^{\frac{n}{2}}$ avec $y = x^2$
   — si $n$ est impair et $n > 1$, alors $x^n = x \times (x^2)^{\frac{(n-1)}{2}}$. Il suffit de calculer $y^{\frac{(n-1)}{2}}$ avec $y = x^2$ et de multiplier le résultat par $x$.
   À l'aide de cette observation, proposez une définition récursive de la fonction puissance et donnez son implémentation.
4. Quelle est la complexité de cette nouvelle fonction ?

## 6.6 Palindromes

On appelle *palindrome* un mot ou une phrase qui, sans tenir compte des espaces, se lit de la même façon dans les deux sens. Par exemple, les mots `ABCBA` et `ABBA` ainsi que la phrase `esope reste et se repose` sont des palindromes.

1. Donnez le code d'une fonction récursive qui permet de tester si une chaîne de caractères est un palindrome ou non.

## 6.7 Suite de Fibonacci [1]

La suite de Fibonacci est une suite d'entiers très connue. Elle doit son nom à un mathématicien italien du XIII$^{e}$ siècle connu sous le nom de Leonardo Fibonacci qui décrit ainsi la croissance d'une population de lapins :

> « Un homme met un couple de lapins dans un lieu isolé de tous les côtés par un mur. Combien de couples obtient-on en un an si chaque couple engendre tous les mois un nouveau couple à compter du troisième mois de son existence ? »

Ce problème est à l'origine de la suite dont le $i^{e}$ terme correspond au nombre de paires de lapins au $i^{e}$ mois. Dans cette population (idéale), on suppose que :
— au (début du) premier mois, il y a juste une paire de lapereaux ;
— les lapereaux ne procréent qu'à partir du (début du) troisième mois ;
— chaque (début de) mois, toute paire susceptible de procréer engendre effectivement une nouvelle paire de lapereaux ;
— les lapins ne meurent jamais.

1. Donnez la formule de récurrence définissant la suite de Fibonacci.
2. Donnez l'implémentation d'une fonction permettant de calcule le $i^{e}$ terme de cette suite
3. Quelle est la complexité de cette fonction ?
4. À partir de l'analyse de la pile d'appels, montrer comment optimiser ce calcul pour obtenir une fonction linéaire ?
5. Il est possible de montrer que :

$$\begin{cases} \mathcal{F}_{2k} = (2\mathcal{F}_{k-1} + \mathcal{F}_k) \times \mathcal{F}_k \\ \mathcal{F}_{2k+1} = \mathcal{F}_{k+1}^2 + \mathcal{F}_k^2 \end{cases}$$

Donnez un algorithme logarithmique du calcul des termes de la suite de Fibonacci.

## 6.8 Évaluation d'un nombre écrit en chiffres romains

Les chiffres romains étaient un système de numération utilisé par les Romains de l'Antiquité pour, à partir de seulement sept lettres, écrire des nombres entiers (mais pas le zéro qu'ils ne considéraient pas comme un nombre).

La numérotation romaine repose sur quatre principes :
— toute lettre placée à la droite d'une autre figurant une valeur supérieure ou égale à la sienne s'ajoute à celle-ci ;
— toute lettre d'unité placée immédiatement à la gauche d'une lettre plus forte qu'elle, indique que le nombre qui lui correspond doit être retranché au nombre qui suit ;

1. L'explication est extraite de la page Wikipédia

— les valeurs sont groupées en ordre décroissant, sauf pour les valeurs à retrancher selon la règle précédente ;
— la même lettre ne peut pas être employée 4 fois consécutivement sauf M.

Le tableau suivant donne les valeurs des chiffres romains de base :

| I | V | X | L | C | D | M |
|---|---|---|---|---|---|---|
| 1 | 5 | 10 | 50 | 100 | 500 | 1 000 |

Pour connaître la valeur d'un nombre écrit en chiffres romains, il faut lire le nombre de droite à gauche, il suffit d'ajouter la valeur du chiffre, sauf s'il est inférieur au précédent, dans ce cas, on le soustrait. Ainsi :

— $\mathrm{XVI} = 1 + 5 + 10 = 16$ ;
— $\mathrm{XIV} = 5 - 1 + 10 = 14$, car I est inférieur à V ;
— $\mathrm{DIX} = 10 - 1 + 500 = 509$, car I est inférieur à X ;
— $\mathrm{MMMMCMXCIX} = 10 - 1 + 100 - 10 + 1\,000 - 100 + 1\,000 \times 4 = 4\,999$ ;
— $\mathrm{MMMMDCCCLXXXVIII} = 4\,888$, est le nombre romain le plus long en quantité de symboles.

Écrivez le code d'une fonction récursive qui donne la valeur numérique d'un nombre romain représenté par une chaîne de caractères.