Manipulation de classes (String, HashMap)

Alice Jacquot
jacquot@lri.fr

septembre-octobre 2020

Objectifs :

- réviser la syntaxe java ;
- découvrir ou réviser les méthodes de manipulation des chaînes de caractères.
- manipuler des classes fournies

Les parties 1 et 3 sont à réaliser dans une première séance de TP, la partie 4 pour la semaine suivante.

1 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 ;
- le premier nombre ne peut pas être 0.

① Écrire une méthode permettant renvoyant un `boolean` permettant de savoir si une chaine de caractères représente une IP valide ou non.

La méthode suivante permet de s'assurer que la méthode teste bien tous les cas :

```
public static void testCheckIP() {

  String[] validIp = {"127.0.0.1",
                      "127.231.1.1",
                      "1.2.3.4"};
```

```
        String[] invalidIp = {"12.2.3",
                              "12.3.213.123.123",
                              "1231.12.2.3",
                              ".1.2.3",
                              "1.2.3.",
                              "1.2.3.4.",
                              "1.2..3",
                              "0.1.2.3"};


        for (String ip : validIp) {
            if (!check(ip)) {
                System.out.println("erreur: " + ip);
            }
        }

        for (String ip : invalidIp) {
            if (check(ip)) {
                System.out.println("erreur: " + ip);
            }
        }

    }
```

On pourra utiliser les instructions suivantes :

— `int Integer.parseInt(String s)` pour convertir une chaîne de caractères en nombre (si et seulement si la chaîne de caractères correspond à un nombre) ;

— `bool isInteger(String s)` pour vérifier si chaîne de caractère représente un nombre (ce code sera expliqué lors du cours sur les exceptions) :

```
public static boolean isInteger(String s) {
  try {
    Integer.parseInt(s);
    return true;
  } catch (NumberFormatException nfe) {
    return false;
  }
}
```

— la méthode `split` de la classe `String` dont le fonctionnement est décrit ci-dessous :

```
public class ExempleSplit {

    public static void main(String[] a) {
        String s = "Bonjour.les.amis";
        // Attention aux anti-slashs devant le point !
```

```
            String separateur = "\\.";

            String[] mot = s.split(separateur);
            for (String m : mot) {
                System.out.println(m);
            }
        }

    }
```

La sortie du programme précédent est :

```
Bonjour
les
amis
```

2 Filtre par langue (partiel 2021)

Dans cet exercice, nous allons trier des textes dans différentes langues, en utilisant la classe `java.util.Locale`.

Les objets de cette classe servent à désigner une région ou une langue, dans ses différentes variantes possibles. Nous nous intéressons ici aux caractéristiques *language* (la langue, par exemple l'anglais, critère principal utilisé), *country* (le pays, par exemple US), et *variant* (la variante régionale).

Un extrait de sa javadoc, ainsi que de celles de ArrayList et HashMap, se trouvent en annexe.

Problème : On veut dans cet exercice traiter des textes formatés de la manière suivante :

- Ils sont sous forme de chaîne de caractères (String).
- La première ligne désigne la langue du texte sous la forme suivante :
  - la langue, sous forme ISO 639 (2 ou 3 caractères alphanumériques) suivie d'un espace
  - puis, éventuellement, le pays sous forme ISO 3166 (2 lettres ou 3 chiffres) suivi d'un espace
  - puis, éventuellement et uniquement dans le cas où le pays est indiqué, la variante sous une forme compatible avec la propriété correspondante de `Locale` suivie d'un espace
  - et finit par un saut de ligne (`'\n'`), toujours présent.
- La suite de la chaîne de caractère est le contenu du texte.

Vous pouvez trouver de (brefs) exemples de tels textes dans la fonction main du code joint.

Dans toute la section, on supposera les textes bien formés (pas de gestion d'erreur).

Rappels de méthodes utiles :

— la méthode `static void System.out.println(String str)` permet l'affichage de str sur la sortie standard

— la méthode `String[] split(String regexp)` de la classe String permet le découpage d'une chaîne autour de chaque sous-chaîne `regexp` rencontrée. Elle prend une *expression régulière* en paramètre permettant l'utilisation de caractères spéciaux pour correspondre à divers cas. Pour cet exercice, vous n'avez pas besoin d'utiliser de caractères spéciaux ou de les échapper. (Ignorez le fait qu'il s'agit d'une expression régulière, considérez qu'elle prend une chaîne de caractère fixe en paramètre).

Le but, à travers les question de cette section, est de compléter la classe `TextFilterByLanguage` dont le squelette est donné sur le site. Vous pouvez (et êtes encouragés à) consulter les javadoc des classes utilisées.

② Écrire la méthode `public static Locale readLang(String text)` qui prend un texte dont la première ligne indique sa langue comme indiqué en introduction et renvoie la locale correspondant à son language (on ignore le pays et la variante éventuellement présents).

③ Écrire la méthode `public static Locale readLocale(String text)` qui prend un texte dont la première ligne indique sa langue, et éventuellement un pays et une variante, comme indiqué en introduction et renvoie la locale correspondante.

④ Écrire la méthode `displayLanguage(String text)` qui prend un texte et affiche sur la sortie standard la partie *language* de sa langue, dans la langue du système de l'utilisateur du programme.
Indice : On utilise le terme *display* pour indiquer un affichage adapté ; et par défaut les méthodes correspondantes de Locale ne prenant pas d'argument utilisent la langue du système de l'utilisateur.

⑤ Écrire la méthode `public static ArrayList<String> filterByLang(ArrayList<String> textCollection, String lang)` qui prend une collection de textes, sous la forme indiquée en introduction, et une langue (dans la langue du système l'utilisateur du programme, telle qu'elle serait affichée par défaut, par exemple "anglais") et renvoie les textes de la collection qui sont dans la langue indiquée, en-tête comprise.

⑥ Écrire la méthode `public static HashMap<String, ArrayList<String» groupByLang( ArrayList<String> textCollection)` qui prend un ensemble de textes, et crée un tableau associatif (*HashMap*), associant une langue (dans la langue du système l'utilisateur du programme, telle qu'elle serait affichée par défaut, par exemple "anglais") et l'ensemble des textes en cette langue, en-tête comprise.

3 Bonus sur les String (facultatif)

⑦ Écrire une fonction `removeSpaces` qui supprime tous les espaces d'une chaine de caractères. Si l'entrée est `a b c`, la fonction renverra `abc`.

⑧ Écrire une fonction `inverse` qui renvoie la chaine de caractères passée en paramètre « inversée » : si l'entrée est `abca`, la fonction renverra `acba`

⑨ Écrire une fonction qui prend en entrée une chaine de caractères de type `xxNxx` où `x` est un caractère et `N` un nombre et qui renvoie une chaine de caractères dans laquelle tous les nombres ont été remplacés par N - 1 fois le caractère précédent.

Voici quelques exemples de sorties attendues de la fonction :

- abc → abc
- abc2 → abcc
- ab2c → abbc
- 3abc → abc (le 3 suit une chaine vide)
- a12bc → aaaaaaaaaaaabc

La signature de la fonction sera :

```
String expandNumber(String input)
```