Java et les “Properties”

1 Introduction

Il est courant pour une application de posséder un certain nombre de paramètres dictant son fonctionnement, ces paramètres pouvant généralement être modifiés au gré de l'utilisateur. Prenons par exemple, au hasard, le cas d'un jeu où le joueur évolue dans une ville : on peut vouloir laisser à ce joueur le choix de son nom. Une fois ce choix effectué, le jeu le sauvegarde dans un fichier afin de pouvoir le ré-utiliser lors de l'exécution suivante, sans avoir à interroger le joueur de nouveau.
Ce mécanisme étant fréquemment requis, l'objet `java.util.Properties`, qui facilite sa mise en place, est proposé par `Java` dans sa bibliothèque standard. Ce sont les fonctionnalités de cet objet que nous allons étudier dans ce sujet. Pour obtenir davantage d'informations sur les méthodes utilisées, commencez par lire la documentation de la classe `Properties`.
Les fichiers dont il est question dans ce sujet sont disponibles sur le portail.

2 Premier pas

Une propriété est un couple {clé,valeur} où clé et valeur sont de type `String`. Les clés identifient de manière unique les propriétés et à chacune est associée une valeur.
Question 1. Compilez puis exécutez le programme `Heros.java` qui vous a été fourni. Dans cet exemple, un objet `Properties` vide est créé. Puis, les propriétés `nom`, `surnom` et `force` sont ajoutées en utilisant la méthode `setProperty(String key, String value)`. Enfin celles-ci sont affichées grâce à la méthode `String getProperty(String key)`.

Question 2. Afficher maintenant les propriétés initiales du Héros. Que constatez-vous ? Pour être sûr de toujours obtenir une valeur valide, même lorsque l'on demande la valeur d'une propriété qui n'existe pas, la méthode `getProperty()` peut prendre un second paramètre qui sert dans ce cas de valeur par défaut.

Question 3. A tous les appels à `getProperty()`, ajoutez comme second paramètre la chaîne `''Non défini''`. Que se passe-t-il cette fois-ci ?

Question 4. Maintenant, afin de garder les propriétés du Héros entre chaque exécution du programme, nous allons les sauvegarder dans un fichier. Afin de faire cela, l'objet `Properties` dispose de la méthode `store`. Elle prend en premier paramètre un flux de sortie et en deuxième paramètre un commentaire qui peut être `null`.

Indice 1. Voir le constructeur `java.io.FileOutputStream(String name)`. Le nom de fichier `heros1.properties` peut par exemple être choisi. N'oublier pas de fermer le flux avec la méthode `close` après son utilisation.

Question 5. Ouvrez le fichier de propriétés créé dans la question précédente avec un éditeur de texte, et voyez comment sont stockées les propriétés.

Question 6. Nous allons maintenant charger le fichier de propriétées afin de récupéré les propriétés initiales du Héros. Pour faire cela, l'objet `Properties` dispose de la méthode `load`. Elle prend en paramètre un flux d'entrée.

Indice 2. Voir le constructeur `java.io.FileInputStream(String name)`. N'oublier pas de fermer le flux avec la méthode `close` après son utilisation.

3 Properties et XML.

Java 5 permet de créer des fichiers de propriétés au format XML.

- La méthode `storeToXML` s'utilise de la même façon que la méthode `store`, mais le fichier de propriétés généré est un fichier XML au lieu d'un fichier texte.
- La méthode `loadFromXML` s'utilise de la même façon que la méthode `load`, mais elle charge un fichier de propriétés au format XML.

Question 7. Modifier le programme afin de sauvegarder les propriétés dans un fichier XML. Le nom heros1.xml pourra être choisi comme non de fichier.

Question 8. Ouvrez le fichier de propriétées créé dans la question précédente avec un éditeur de texte, et voyez comment sont stockées les propriétés.

Question 9. Modifier le programme afin de charger les propriétés au format XML.

Remarque 1. La DTD des fichiers de propriétés utilisent les *attributs*.

```
1. <?xml version="1.0" encoding="UTF-8"?>
2.   <!-- DTD for properties -->
3.
4. <!ELEMENT properties ( comment?, entry* ) >
5. <!ATTLIST properties version CDATA #FIXED "1.0">
6. <!ELEMENT comment (#PCDATA) >
7. <!ELEMENT entry (#PCDATA) >
8. <!ATTLIST entry key CDATA #REQUIRED>
```

Figure 1: DTD des propriétés Java

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE properties SYSTEM "http://
3.        java.sun.com/dtd/properties.dtd">
4. <properties>
5.   <entry key="nom">Kim</entry>
6.   <entry key="force">2</entry>
7.   <entry key="surnom">Le boulet</entry>
8. </properties>
```

Figure 2: Exemple de fichier de propriétés

Un attribut XML est une paire {clé,valeur} qui est associée à un élément. La syntaxe pour définir un attribut est la suivante :

```
<! ATTLIST Elément Attribut Type >
```

Type représente le type de donnée de l'attribut, il en existe trois :

Littéral : il permet d'affecter une chaîne de caractères à un attribut. Pour déclarer un tel type il faut utiliser le mot clé `CDATA`.

Enumération : il permet de définir une liste de valeurs possibles pour un attribut donné, afin de limiter le choix de l'utilisateur. La syntaxe de ce type d'attribut est : (la valeur pas défaut est optionnelle)

```
<! ATTLIST Elément Attribut (Valeur1 | Valeur2 )  "valeur par défaut" >.
```

Atomique : il permet de définir un identifiant unique pour chaque élément grâce au mot clé `ID`.

Enfin chacun de ces types d'attributs peut être suivi d'un mot clé particulier permettant de spécifier le niveau de nécessité de l'attribut :

#IMPLIED signifie que l'attribut est optionnel.

#REQUIRED signifie que l'attribut est obligatoire.

#FIXED signifie que l'attribut sera affecté d'une valeur par défaut s'il n'est pas défini.

Dans le cas de la DTD des propriétés Java, la ligne 5 permet de déclarer un attribut nommé `version` pour l'élément `properties`. Cet attribut est une chaine de caractères avec comme valeur par défaut "1.0". La ligne 8 permet de déclarer un attribut obligatoire nommé `key` pour l'élément `entry`. C'est une chaine de caractères qui correspond à la clé de la propriété.

4 Gestion d'ensembles de propriétés

Il est possible de créer différents ensembles de propriétés, chacun dans leur fichier, puis de charger le fichier qui convient lors de l'exécution du programme. On peut par exemple vouloir proposer plusieurs langues à l'utilisateur : on aura dans ce cas un fichier de propriétés pour chaque langue, contenant les phrases utilisées dans le jeu. Il ne reste plus ensuite qu'à charger le fichier qui convient.

Question 10. Créez un fichier `francais.xml` contenant la propriété {`greetings,Bonjour`}, puis un fichier `english.xml` contenant la propriété {`greetings,Hello`}.

Question 11. Ecrivez un programme `Multilingue` qui, en fonction d'un paramètre reçu sur sa ligne de commande, charge un des deux fichiers créés ci-dessus puis affiche la valeur de la propriété `greetings` afin de saluer l'utilisateur dans la langue qu'il aura choisie.

Question 12. Modifiez votre programme `Multilingue` de manière à lire la langue à utiliser depuis un fichier `config.xml` au lieu d'utiliser la ligne de commande. Le fichier de configuration doit aussi indiquer dans quel répertoire se trouvent les fichiers de langue.

5 Pour aller plus loin

Tout programme Java dispose de son propre ensemble propriétés système. Ces propriétés sont accessibles en utilisant `String System.getProperty(String key)`. L'ajout de propriétés système peut se faire lors du lancement de l'application Java en ajoutant à la ligne de commande de la machine virtuelle Java : `-Dclé=valeur`.

Exemple. java -Dconfig=pathToConfigFile.xml Multilingue.

```
public static void main(String [] args)
{
  String nameConfig = System.getProperty("config","defaultConfig.xml");
  //Charge le fichier de config de nom "nameConfig"
}
```

Dans le programme précédent, si aucune propriété système n'est fixée, le fichier de configuration par défaut est chargé, sinon le fichier qui sera chargé est celui référencé par la propriété.