

Interface avec le système

Université de Nice - Sophia Antipolis

Version 2.10.2 – 29/1/12

Richard Grin

- Cette partie du cours décrit comment un programme Java peut s'insérer dans l'environnement dans lequel il tourne, et échanger des informations avec cet environnement

R. Grin

Interface système

2

Plan

- Classes **System** et **Runtime**
- Fichiers JAR
- Propriétés (variables d'environnement)
- Ressources
- Exécution d'un programme externe
- Utilisation de la barre de tâches du bureau
- Préférences de l'utilisateur
- Interface avec les autres langages (C, C++, assembleur, etc.)

R. Grin

Interface système

3

Classes **System** et **Runtime**

R. Grin

Interface système

4

Classe **java.lang.System**

- Cette classe n'est jamais instanciée ; toutes les méthodes sont **public static**
- Cette classe fournit
 - la méthode **exit** pour arrêter l'exécution de la JVM
 - des références aux voies standard d'entrées-sorties
 - des méthodes pour lire et écrire des propriétés (étudiées plus loin dans cette section)
 - des méthodes diverses pour installer un gestionnaire de sécurité (voir cours sur la sécurité), charger des librairies, lire l'heure système, lancer le ramasse-miettes, copier des tableaux en blocs,...

R. Grin

Interface système

5

Voies d'entrées-sorties standard

- 3 variables **public** correspondent aux 3 voies standards :
public static final PrintStream in, out, err
- 3 méthodes **public static** permettent de rediriger ces voies :
 - **void setOut(PrintStream out)**
 - **void setIn(InputStream in)**
 - **void setErr(PrintStream err)**

R. Grin

Interface système

6

Quelques méthodes de **System**

- `void exit(int statut)` ; le statut est le code retour renvoyé par java (0 indique un déroulement normal)
- `native void arraycopy(Object src, int positionSrc, Object dst, int positionDst, int longueur)`
- `native long currentTimeMillis()`
- `void gc()`

R. Grin

Interface système

7

Quelques méthodes de **System**

- `String getenv(String nomVariable)` récupère la valeur d'une variable d'environnement du système d'exploitation ; il vaut mieux utiliser l'option `-D` de java (voir les propriétés plus loin dans ce cours)
- `Map<String, String> getenv()` récupère les valeurs de toutes les variables d'environnement
- `void setSecurityManager(SecurityManager s)`

R. Grin

Interface système

8

Classe **java.lang.Runtime**

- Chaque programme Java est associé à une instance unique de la classe **Runtime**, obtenue avec la méthode
`public static Runtime getRuntime()`
- Cette instance donne au programme un accès au système d'exploitation dans lequel il s'exécute
- La classe **System** fait appel à cette instance pour exécuter plusieurs de ses méthodes

R. Grin

Interface système

9

Quelques méthodes de **Runtime**

- `exec` permet de lancer un programme externe (étudié dans la section suivante)
- `long freeMemory()` retourne une approximation du nombre d'octets disponibles
- `long totalMemory()` retourne la mémoire totale occupée par la JVM
- `void gc()` demande à la JVM d'essayer de libérer de la place avec le ramasse-miette
- `void traceInstructions(boolean on)` demande à la JVM d'indiquer les instructions exécutées
- `void traceMethodCalls(boolean on)` demande à la JVM d'indiquer les appels de méthodes

R. Grin

Interface système

10

Fichiers **.jar**

R. Grin

Interface système

11

Fichiers **.jar**

- Les fichiers **.jar** (*Java ARchive*) sont utilisés pour conserver en un seul fichier, plusieurs fichiers utilisés par des programmes Java :
 - fichiers `.class` Java,
 - images,
 - son,
 - ...

R. Grin

Interface système

12

Format des fichiers .jar

- « Repliage » de répertoires et compression des fichiers au format *zip*
- Ce fichier zip contient un fichier particulier à l'emplacement (attention, la casse des lettres peut varier !)
META-INF/MANIFEST.MF
- Ce fichier **MANIFEST.MF** contient des informations sur le fichier jar et les fichiers contenus dans le fichier jar

R. Grin

Interface système

13

Avantages des fichiers .jar (1)

- Rapidité de chargement des fichiers, spécialement si on utilise le protocole HTTP (avec des applets)
- Compression des données (moins de place sur disque)
- Format standard de distribution des applications Java (donc portabilité)

R. Grin

Interface système

14

Avantages des fichiers .jar (2)

- Grâce au fichier MANIFEST.MF, on peut
 - signer des fichiers (par exemple, pour avoir des *applets* sécurisées)
 - gérer des versions
 - donner des informations utiles pour exécuter l'application (classe principale, *classpath*)
 - protéger des paquetages contre l'ajout de classes

R. Grin

Interface système

15

Utilitaire jar pour fichiers .jar

- La syntaxe ressemble à la commande tar d'Unix
- 3 modes de travail :
 - création (option **c**)
 - extraction (option **x**)
 - listing (option **t**)
 - modification (option **u**)

R. Grin

Interface système

16

Utilitaire jar

```
jar {c|t|x|u} [-f fichier-archive.jar] [fichiers...]
```

- **c** pour créer un fichier archive à partir des *fichiers*
 - si un des *fichiers* est un répertoire, toute l'arborescence est archivée (désarchivée pour les options **x** ou **t**)
- **x** (resp. **t**) pour extraire (resp. lister) tous les fichiers de l'archive *fichier-archive.jar*
 - si on donne une liste de *fichiers*, seuls ces *fichiers* de l'archive sont extraits (resp. listés)
- **f** *fichier-archive.jar* indique avec quel fichier archive travailler
 - par défaut, *jar* travaille avec l'entrée ou la sortie standard

R. Grin

Interface système

17

Utilitaires pour fichiers .jar (2)

- **u** (*update*) pour modifier un fichier archive à partir des *fichiers*. On peut ainsi ajouter des fichiers ou remplacer des fichiers par une nouvelle version
- **m** *fichier-manifest* permet d'indiquer un fichier qui sera le fichier **MANIFEST.MF** du fichier jar
- **v** (*verbose*) fait afficher plus d'informations sur l'opération pendant son exécution, ou sur le listing

R. Grin

Interface système

18

Noms des entrées du jar

- Correspondent au chemin donné pour désigner ce qu'il faut mettre dans le jar
- Si on tape par exemple
`jar uvf truc.jar rep/fichier truc`
on aura 2 entrées `rep/fichier` et `truc`
- Si on veut plutôt une entrée `fichier`, il faut utiliser l'option `-C` qui fait un changement temporaire de répertoire (par rapport au répertoire courant d'exécution) pendant la compression :
`jar uvf truc.jar -C rep fichier`

R. Grin

Interface système

19

Les noms des entrées sont relatifs

- Les noms des entrées d'un jar sont relatifs et pas absolus
- Si on tape par exemple
`jar uvf truc.jar /usr/bin/ls`
on aura une entrée `usr/bin/ls` dans le jar
- Sous Windows, on peut cependant avoir des entrées du type
`C:/rep/machin`

R. Grin

Interface système

20

Exemple de création de fichier jar

- Création d'un fichier `f1.jar` contenant tous les fichiers placés dans l'arborescence du répertoire classes et le fichier `MANIFEST.MF` placé sous le répertoire `src` :
`jar cvfm f1.jar src\MANIFEST.MF -C classes .`
- Remarquez l'option `-C` qui permet de ne pas placer le répertoire `classes` lui-même dans le fichier jar

R. Grin

Interface système

21

Autre exemple de création

- Création d'une archive `f1.jar` (`v` est l'option « verbeuse ») contenant tous les fichiers `.class` du répertoire courant et tous les fichiers de l'arborescence du répertoire `images`
`jar cvf f1.jar *.class images`

R. Grin

Interface système

22

Exemples de modification

- Ajout du fichier `truc.class`
`jar uvf f1.jar truc.class`
- Remplacement du fichier `truc.class` de l'archive par une nouvelle version
`jar uvf f1.jar truc.class`

R. Grin

Interface système

23

Exemples d'extraction de fichiers depuis un fichier jar

- Lister les fichiers de l'archive `f1.jar`
`jar tvf f1.jar`
- Extraire tous les fichiers de l'archive (attention, cette commande écrase tous les fichiers éventuels de même nom qu'un fichier extrait)
`jar xvf f1.jar`
- Extraire certains fichiers de l'archive
`jar xvf f1.jar META-INF/MANIFEST.MF`
`jar xvf f1.jar Truc.java images/iml.gif`

R. Grin

Interface système

24

Modifier le fichier **MANIFEST**

- Modifier le fichier **MANIFEST.MF** avec le fichier `rep/manifest` ; dans l'exemple suivant, **m** étant placé avant **f** dans la commande, on place le fichier « manifest » avant le fichier jar

```
jar umvf rep/manifest f1.jar
```
- Autre commande équivalente (**f** et **m** dans un autre ordre) :

```
jar ufmv f1.jar rep/manifest
```

R. Grin

Interface système

25

Fichier **MANIFEST** par défaut

- Une entrée **META-INF/MANIFEST.MF** est créée par défaut si on ne donne pas d'option **m** lors de la création du fichier *jar*
- Ce fichier **MANIFEST.MF** ne contient aucune information spéciale sur le fichier jar, si ce n'est la version du format jar et celle du java qui l'a créé :

```
Manifest-Version: 1.0  
Created-by: 1.4.1 (Sun Microsystems Inc.)
```

R. Grin

Interface système

26

Fichier **MANIFEST**

- Un fichier **MANIFEST.MF** peut contenir les informations suivantes :
 - version du format *jar*
 - classe principale d'une application, répertoires liés à une *applet*, etc.
 - *checksum* pour vérifier les fichiers contenus dans le fichier *jar* (depuis la version Java 2, ce *checksum* est dans le fichier *manifest* seulement pour les fichiers *jar* signés)

R. Grin

Interface système

27

Pour pouvoir disposer des classes d'un fichier jar

- Dans le cas d'une application, il suffit d'ajouter le fichier jar dans le *classpath*
- On utilise pour cela les options **-cp** ou **-classpath** de la commande *java* (on peut aussi utiliser la variable **CLASSPATH**)
- Plus rarement, on place le fichier jar dans le répertoire des extensions
- Voir aussi plus loin la notion de fichier jar d'extension (quand l'application est dans un jar)

R. Grin

Interface système

28

Exécution d'un fichier *jar* (1)

- La commande *java* a une option **-jar** qui permet de donner un nom de fichier **.jar** au lieu du nom d'une classe comme classe initiale d'exécution
- Par exemple,

```
java -jar application.jar
```

 lance l'exécution de la méthode **main** de la classe déclarée comme classe principale dans le fichier *manifest* du fichier **application.jar**

R. Grin

Interface système

29

Exécution d'un fichier *jar* (2)

- Pour déclarer la classe principale de l'application il faut ajouter dans le fichier *manifest* une entrée **Main-Class**
- Par exemple :

```
Main-Class: fr.unice.TestClass
```

Pas d'espace avant le « : » !

R. Grin

Interface système

30

Jar exécutable sous Windows

- Sous Windows les fichiers Jar sont associés à l'exécutable `javaw -jar` lors de l'installation de Java
- Si le fichier Jar a une entrée `Main-Class:`, il peut donc être lancé en faisant un double clic sur son icône

R. Grin

Interface système

31

Classpath avec l'option -jar

- Quand on lance une application avec l'option `-jar`, le `classpath` se limite au « répertoire » racine placé dans le fichier jar (sauf s'il y a une entrée `Class-Path` dans le jar ; voir transparents suivants)
- Même une option `-classpath` sera ignorée !

R. Grin

Interface système

32

Fichier *jar* d'extension (1)

- Le fichier *manifest* du fichier JAR peut contenir une entrée `Class-Path` qui indique les URL d'autres fichiers *jar* (ou zip) ou de répertoires (suffixes « / » obligatoire) qui pourront être utilisés pour trouver les classes
- Les classes et ressources seront donc recherchés dans le jar et aussi dans les endroits indiqués par les entrées de `Class-Path`

R. Grin

Interface système

33

Fichier *jar* d'extension (1)

- Les fichiers doivent être dans le système de fichiers local
- Les URL relatives doivent être relatives au répertoire du fichier jar qui les contient
- Attention, il s'agit d'URL donc le séparateur de noms de fichiers est « / », même sous Windows et les noms absolus Windows sont de la forme `/C:/rep1...`

R. Grin

Interface système

34

Fichier *jar* d'extension (2)

- Si c'est le jar d'une applet, le navigateur devra les charger en même temps que l'archive de l'*applet*
- Un fichier manifest peut contenir plusieurs entrées dans la ligne `Class-Path:`, séparées par un ou plusieurs espaces
- Par exemple :
`Class-Path: rep/ rep2/truc2.jar`

R. Grin

Interface système

35

Dépendance avec d'autres jars

- Mettre un jar dans un jar ne sert à rien ; ça ne marche pas !
- Si les classes d'un jar dépendent de bibliothèques contenues dans des jars, 2 solutions :
 - utiliser une entrée `Class-Path` dans le manifest (solution la plus utilisée)
 - décompacter les bibliothèques et les recompacter dans un seul jar, avec le jar de l'application principale

R. Grin

Interface système

36

Dépendance avec d'autres jars

- La 1^{ère} solution (**Class-Path**) est la plus souple car elle permet d'utiliser facilement les nouvelles versions des bibliothèques utilisées
- Mais cette solution a l'inconvénient de ses avantages car les nouvelles versions des bibliothèques peuvent créer des incompatibilités avec l'application principale
- La 2^{ème} solution fige les versions des bibliothèques, ce qui a ses avantages et ses inconvénients

R. Grin

Interface système

37

Problèmes avec le fichier MANIFEST

- Pour éviter des problèmes fréquemment rencontrés :
 - **pas d'espace avant les « : » des entrées du jar (Main-Class: par exemple)**
 - terminer la dernière ligne du fichier par un passage à la ligne
 - attention aux majuscules/minuscules dans les noms de fichiers et de répertoires

R. Grin

Interface système

38

Information sur le contenu

- Le fichier manifest d'un jar peut contenir des informations sur le contenu du jar
- On peut ainsi indiquer un numéro de version pour la spécification ou l'implémentation

R. Grin

Interface système

39

Exemple

```
Manifest-Version: 1.0
Created-By: 1.5.0_01 (Sun Microsystems)
Name: java/util/
Specification-Title: Utility Classes
Specification-Version: 1.2
Specification-Vendor: Sun Microsystems
Implementation-Title: java.util
Implementation-Version: build57
Implementation-Vendor: Sun Microsystems
```

R. Grin

Interface système

40

Signature d'un fichier jar

- En Java 2, l'outil *jarsigner* permet de signer les fichiers jar (voir cours sur la sécurité)

R. Grin

Interface système

41

Exécution d'une *applet* contenue dans un fichier *jar*

```
<applet
  code="fr.unice.pl.UneClasse.class"
  archive="jar/fichier.jar"
  width=120 height=120>
</applet>
```

R. Grin

Interface système

42

Ressources placées dans un fichier jar

- On peut situer une ressource placée dans un fichier jar avec la méthode « `URL getResource(String)` » de la classe `Class`
- On peut alors utiliser la ressource après avoir récupéré son URL
- Cette méthode fonctionne avec les applets, les applications, qu'elles soient placées dans un fichier jar ou non
- Plus de précisions dans la section « ressources » de la suite de ce support

R. Grin

Interface système

43

Jar et images

- Le cours sur les interfaces graphiques indique comment utiliser les images placées dans un fichier jar (on utilise le chargeur de classes pour la recherche de ressources) :

```
URL url =
    getClass().getResource(nomFichier);
ImageIcon icone = new ImageIcon(url);
```

- On peut ainsi, par exemple, charger les icônes utilisées par les barres d'outils

R. Grin

Interface système

44

Code pour lire un fichier jar (1)

```
import java.net.JarURLConnection;
import java.util.zip.ZipEntry;
import java.util.jar.JarFile;
. . .
URL url =
    new URL("jar:file:/chemin/f.jar!/");
JarURLConnection jarConnection =
    (JarURLConnection)url.openConnection();
JarFile jf = conn.getJarFile();
// ou jf = new JarFile("truc.zip");
```

R. Grin

Interface système

45

Code pour lire un fichier jar (2)

```
// Lecture séquentielle des entrées :
Enumeration e = jf.entries();
while (e.hasMoreElements()) {
    JarEntry entree =
        (JarEntry)e.nextElement();
    String nomEntree = entree.getName();
    String nom = entry.getName();
    // Lecture du contenu d'une entrée
    InputStream is =
        jf.getInputStream(entry);
    . . .
}
```

R. Grin

Interface système

46

Code pour lire un fichier jar (3)

```
// Ou accès direct à une entrée :
JarEntry entry =
    jf.getJarEntry("rep\\f1.txt");
InputStream is =
    jf.getInputStream(entry);
// On lit le contenu de l'entrée
. . .
```

R. Grin

Interface système

47

Récupérer les informations du jar

- La classe `java.lang.Package` permet de récupérer les informations du jar :

```
Package pkg =
    Package.getPackage("fr.truc");
String nom =
    pkg.getSpecificationVersion();
```

R. Grin

Interface système

48

Classe `JarURLConnection`

- Classe de base abstraite pour représenter une connexion à un jar ou une entrée d'un jar
- `url.getConnection()` (classe `URL`) renvoie une instance de ce type si l'URL est celui d'un jar
- Elle contient des méthodes qui facilitent le travail avec un jar, par exemple, pour accéder aux attributs du fichier **MANIFEST**
- Elle se trouve dans le paquetage `java.net`

R. Grin

Interface système

49

Format d'URL pour une entrée de jar

- `jar:<url du jar>!/{entrée du jar}`
- Par exemple,
`jar:http://truc.fr/rep/f.jar!/fr/truc/Classe.class`
- Un fichier jar :
`jar:http://truc.fr/rep/f.jar!`
- Un répertoire dans un fichier jar (se termine par un « / »):
`jar:http://truc.fr/rep/f.jar!/fr/truc/`
- Un répertoire local (sur la même machine)
`jar:file:rep1/rep2/`

R. Grin

Interface système

50

Propriétés

R. Grin

Interface système

51

Propriété

- Un programme Java s'exécute dans un environnement connu par le programme sous la forme de propriétés
- Une propriété est caractérisé par
 - un nom (une `String`)
 - une valeur (d'un type non primitif)
- Le nom d'une propriété est souvent hiérarchique. Par exemple, `java.class.version`

R. Grin

Interface système

52

Types de propriétés

- Les propriétés système décrivent l'environnement du système dans lequel s'exécute le programme Java ; exemple : `user.name`, `line.separator`, `file.separator`, etc.
- Le programme (ou l'utilisateur) peut avoir ses propres propriétés qui ne sont utilisés que par lui-même ; exemple : `editeur.son` (pour indiquer si le programme émettra des sons ou non)

R. Grin

Interface système

53

Les propriétés

- La classe `java.util.Properties` (classe fille de `Hashtable<Object, Object>`) a été conçue pour
 - manipuler des ensembles de propriétés
 - les conserver entre 2 invocations d'un programme, en les enregistrant dans des fichiers
- Elle représente un ensemble de propriétés identifiées par une `String` (le nom de la propriété)

R. Grin

Interface système

54

Constructeurs de **Properties**

- Le constructeur sans paramètre crée un ensemble de propriétés vide
- Il est possible de passer en paramètre un ensemble de propriétés qui seront les propriétés par défaut ; souvent on passe les propriétés système (voir exemples quelques transparents plus loin)

R. Grin

Interface système

55

Méthodes de **Properties**

- **Object** `setProperty(String nomProp, String valeur)` : met la valeur d'une propriété (retourne l'éventuelle ancienne valeur)
- **String** `getProperty(String nomProp)` : retourne la valeur de la propriété (**null** si la propriété n'existe pas)
- **String** `getProperty(String nomProp, String valeur)` : idem mais retourne **valeur** si la propriété n'existe pas

R. Grin

Interface système

56

Propriétés standard du système

- La JVM récupère automatiquement des attributs standard du système :

```
java.version      java.class.version
java.vendor       java.vendor.url
java.home         java.class.path
os.name          os.arch           os.version
file.separator    line.separator     path.separator
user.name         user.home          user.dir
user.language     user.region        user.timezone
file.encoding     file.encoding.pkg
```

- A utiliser pour la portabilité des applications :
`file.separator`, `line.separator` (fins de ligne)
- `user.dir` est le répertoire courant

R. Grin

Interface système

57

Récupérer les propriétés standard du système

- On utilise 2 méthodes **static** de la classe **System** :
- **String** `getProperty(String nomPropriete)` renvoie la valeur d'une propriété système sous forme de **String** (renvoie **null** s'il n'existe pas une propriété avec ce nom)
- **Properties** `getProperties()` renvoie toutes les propriétés système

R. Grin

Interface système

58

Valeurs de propriétés données sur la ligne de commande

- On peut donner d'autres valeurs de propriétés système au lancement de l'interprète Java, avec l'option **-D** de **java** :

```
java -Dediteur.prop1=25 -Dediteur.son=off Edit
```

R. Grin

Interface système

59

Format des fichiers de propriétés (1)

- Un programme Java peut aussi lire des valeurs de propriétés depuis des fichiers de propriétés :

```
# Propriétés pour l'éditeur Java
editeur.son = off
# La valeur peut contenir des espaces
editeur.prop1 = mot1 mot2
...
```

- Le séparateur entre le nom et la valeur peut être « = », « : » ou tabulation
- Les lignes de commentaires commencent par **#** ou **!**

R. Grin

Interface système

60

Format des fichiers de propriétés (2)

- Une ligne par propriété ; elle peut se continuer à la ligne suivante si elle se termine par \
- Il faut placer un \ avant les caractères #, !, = et : s'ils n'ont pas leur signification habituelle
- Un simple \ doit être doublé : « \\ » ; par exemple, pour désigner un fichier « C:\repl\fichier » sous Windows :
fichier = C:\\repl\\fichier

R. Grin

Interface système

61

Format des fichiers de propriétés (3)

- Il est possible d'utiliser le codage \uxxxx (valeur hexadécimale) pour représenter des caractères
- Attention, les espaces de fin de ligne risquent d'être inclus dans les valeurs des propriétés ; il faut donc éviter d'en mettre

R. Grin

Interface système

62

Lecture de propriétés

- La méthode **load** (de la classe **Properties**) peut prendre en paramètre un **Reader** ou un **InputStream** pour lire des propriétés dans un fichier (plus généralement un flot)
- Avec un **InputStream**, le codage des caractères doit être ISO 8859-1 ; le codage peut être différent avec un **Reader** (voir cours sur les entrées-sorties)

R. Grin

Interface système

63

Écriture de propriétés

- La méthode **store** (de la classe **Properties**) peut prendre un **Writer** ou un **OutputStream** en paramètre pour écrire des propriétés dans un flot, dans un format compatible avec la méthode **load**
- Le codage est ISO 8859-1 si on utilise un **OutputStream**
- Un 2^{ème} paramètre écrit un commentaire au début du fichier de propriétés (s'il n'est pas **null**)

R. Grin

Interface système

64

Exemple d'utilisation des propriétés

```
// Crée une instance de Properties qui sera utilisée pour
// modifier les propriétés système, avec les
// propriétés système actuelles comme valeur par défaut
Properties props =
    new Properties(System.getProperties());
// Charge d'autres propriétés
try (BufferedInputStream bis =
    new BufferedInputStream(
        new FileInputStream("f.properties"))) {
    props.load(bis);
    // Positionne la combinaison des propriétés comme
    // nouvelles propriétés système
    System.setProperties(props);
}
```

R. Grin

Interface système

65

Exemple d'utilisation des propriétés

```
// Lit la valeur d'une propriété
String son = props.getProperty("editeur.son");
...
// Modifie la propriété
props.setProperty("editeur.son", "off");
// Sauvegarde les nouvelles propriétés
try (BufferedOutputStream bos =
    new BufferedOutputStream(
        new FileOutputStream("f.properties"))) {
    props.store(bos, "Propriétés avec son");
}
```

Insère ce commentaire (#)
en 1ère ligne du fichier

R. Grin

Interface système

66

Propriétés et ressources

- Travailler avec un nom de fichier pour conserver des propriétés n'est pas souple (voir cours sur les fichiers)
- Il vaut mieux utiliser des noms de ressources (voir section suivante)
- Le transparent suivant récrit l'exemple précédent pour la lecture des propriétés dans un fichier ; ce fichier est maintenant désigné par un nom de ressources et plus par un nom relatif

R. Grin

Interface système

67

Exemple de chargement de propriétés en utilisant un nom de ressources

```
BufferedInputStream bis =  
    new BufferedInputStream(  
        getClass().getResourceAsStream(  
            "f.properties"));  
props.load(bis);  
bis.close();
```

Le fichier de propriétés est situé dans le même répertoire que la classe qui contient ce code (car nom relatif)

R. Grin

Interface système

68

Énumérer des propriétés

- La méthode `propertyNames()` de la classe `Properties` renvoie une `Enumeration` pour énumérer tous les noms des propriétés d'une instance de `Properties`
- On peut aussi afficher toutes les propriétés sur un `PrintStream` ou un `PrintWriter` avec les méthodes `list(PrintStream ps)` ou `list(PrintWriter pw)` ; par exemple, `props.list(System.out)`

R. Grin

Interface système

69

Propriétés d'un autre type que `String`

- `System.getProperty` permet de lire la valeur d'une propriété sous forme d'une `String`
- La classe `java.awt.Font` contient la méthode `static Font getFont(String nomFonte)` qui renvoie la fonte spécifiée par la valeur de la propriété système `nomFonte`
- D'autres méthodes dans des classes diverses permettent de lire des valeurs de propriétés de type autre que `String` ; par exemple, `getColor` (de la classe `java.awt.Color`) renvoie un `Color`

R. Grin

Interface système

70

Fichiers de propriétés XML

- Les méthodes `storeToXML` et `loadFromXML` permettent d'enregistrer et de lire les propriétés dans des documents XML

R. Grin

Interface système

71

- Les fichiers qui contiennent des valeurs de propriétés sont un cas particulier de fichiers de ressources

R. Grin

Interface système

72

Ressources

R. Grin

Interface système

73

Définition

- Une ressource est un élément extérieur à une classe Java (pas contenu dans un fichier « .java »), utilisé par une application
- Nous allons étudier dans cette section le cas d'une ressource contenue dans un fichier (les ressources peuvent aussi ne pas correspondre à un fichier ; par exemple une base de données peut être une ressource utilisée par une application)

R. Grin

Interface système

74

Exemples de fichiers de ressources

- Une classe Java peut utiliser un fichier de propriétés, un fichier qui contient une image, du son, une vidéo, ou un fichier binaire quelconque
- Pour faciliter l'internationalisation d'une application il est intéressant d'enregistrer les textes présentés à l'utilisateur dans des fichiers de ressources

R. Grin

Interface système

75

Ressources et fichiers JAR

- Le plus souvent les ressources et les classes qui les utilisent sont regroupées dans un fichier JAR
- Il faut donc écrire du code qui peut retrouver les ressources associées aux classes, même si l'application est empaquetée dans un fichier JAR

R. Grin

Interface système

76

Désigner une ressource

- Dans le cours sur les entrées-sorties on montre dans l'étude de la classe `File` que les emplacements relatifs ou absolus dans le système de fichiers ont des inconvénients sérieux
- De plus ils ne permettent pas de désigner une ressource dans un fichier jar
- Nous allons étudier comment désigner et lire le contenu d'une ressource

R. Grin

Interface système

77

Emplacement de la ressource (1)

- Une ressource est associée à une classe (une des classes qui l'utilisera) ou à plusieurs classes
- Si elle est associée à plusieurs classes, son emplacement sera le plus souvent donné relativement au *classpath* de l'application

R. Grin

Interface système

78

Emplacement de la ressource (2)

- Si la ressource est associée à une seule classe, son emplacement peut aussi être donné relativement à l'emplacement de cette classe
- Les classes et les ressources peuvent se trouver
 - dans le système de fichier local
 - dans un fichier JAR
 - sur un serveur Web

R. Grin

Interface système

79

Méthode `getResource()`

- La classe **Class** contient une méthode qui permet d'obtenir une ressource en donnant un chemin « absolu » (relativement au *classpath*), ou relativement à l'emplacement de la classe qui effectue le chargement :
URL `getResource(String nom)`
- Cette méthode très utile est souvent mal utilisée ; étudions-la en détails

R. Grin

Interface système

80

Recherche de la ressource par la classe

- La classe va déléguer la recherche à son chargeur de classe
- Avant de déléguer, elle va faciliter la recherche de ressources dont l'emplacement est relatif à l'emplacement de son fichier **.class**

R. Grin

Interface système

81

Préliminaire

- En fait, la recherche dépend du chargeur de la classe ; elle s'effectue le plus souvent dans le *classpath* mais elle peut aussi s'effectuer dans une base de données ou sur le réseau
- On va décrire d'abord la recherche effectuée par les chargeurs de classes habituels (en général des `UrlClassLoaders`), qui recherchent dans le *classpath*
- On verra ensuite le cas général pour un chargeur de classe quelconque

R. Grin

Interface système

82

Nom d'une ressource

- Un nom absolu commence par un « / » :
`/images/truc.jpg`
dans ce cas, le nom indiqué désigne un chemin par rapport à un des répertoires indiqués dans le *classpath*
- Si le nom est relatif :
`images/truc.jpg`
le chemin est relatif au répertoire du paquetage de la classe qui charge la ressource :
- Attention, le séparateur est toujours « / », quel que soit le système d'exploitation

R. Grin

Interface système

83

Pratiquement

- Lorsque l'on distribue une application (dans un jar ou non), on peut choisir 2 politiques pour les ressources :
 - placer toutes les ressources dans un répertoire à part (souvent nommé *resources*, ou *resources* en anglais) ; en ce cas, on utilisera des noms absolus
 - placer les ressources sous le même répertoire que les classes qui les utilise (souvent dans un sous-répertoire) ; on utilisera alors des noms relatifs

R. Grin

Interface système

84

Dans un jar

Une arborescence dans un jar qui contient 2 répertoires p1 et images :

- p1
 - p2
 - C1.class
 - images
 - i1.gif
 - images
 - i2.gif
- La classe C1 récupérera l'URL de **i1.gif** par :
- ```
getClass().getResource("images/i1.gif");
```
- et de **i2.gif** par :
- ```
getClass().getResource("/images/i2.gif");
```
- (le / du début fait la différence)

R. Grin

Interface système

85

Ressources dans un jar à part

- Dans le cas où on ne place pas l'application dans un jar, on peut placer les ressources dans un fichier jar ou zip
- Pour l'exécution il faudra alors placer le fichier jar ou zip dans la *classpath* et désigner les ressources par des noms absolus
- Si l'application est dans un jar, on peut faire de même en ajoutant dans le fichier Manifest du jar, le nom du fichier qui contient les ressources (entrée **Class-Path**)

R. Grin

Interface système

86

Recherche d'une ressource (1/2)

1. Obtenir l'objet **Class** de la classe qui charge la ressource

- si c'est la classe courante (si la méthode n'est pas **static**) :

```
Class c = this.getClass();
```

- si c'est une autre classe (ou une méthode **static**) :

```
Class c = NomClasse.class; // OU  
Class c = Class.forName("nomClasse");
```

R. Grin

Interface système

87

Recherche d'une ressource (2/2)

2. Obtenir l'URL du fichier de ressource :

```
URL urlRess = c.getResource(nomRess);
```

- A partir de cet URL on va voir comment obtenir un flot pour lire la ressource

R. Grin

Interface système

88

Remarque

- Comme la recherche est déléguée au chargeur de classe de la classe, on pourrait penser utiliser explicitement ce chargeur :

```
getClass().getClassLoader().getResource(...)
```
- Mais si on fait ainsi, on perd la souplesse de l'utilisation des noms relatifs pour donner le chemin des ressources par rapport à l'emplacement de la classe
- De plus, on a besoin d'une permission spéciale (pour obtenir le chargeur de classes ; voir cours sur la sécurité)

R. Grin

Interface système

89

Lire une ressource

- Quand on a l'URL, on peut lire le contenu de la ressource par

```
InputStream in = urlRess.openStream();
```
- La classe **Class** fournit aussi un raccourci qui renvoie directement un **InputStream** à partir d'un nom de ressource ; si **c** est la classe qui utilise la ressource :

```
InputStream in =  
c.getResourceAsStream(nomRess);
```

R. Grin

Interface système

90

Lire un fichier texte

- Si la ressource contient du texte, il faut utiliser `InputStreamReader` pour obtenir un `Reader` ; par exemple :

```
BufferedReader br =
    new BufferedReader(
        new InputStreamReader(in));
String ligne;
while ((ligne = br.readLine()) != null) {
```

- On peut spécifier le codage des caractères de la ressource si ça n'est pas le codage par défaut :

```
new InputStreamReader(
    in, Charset.forName("UTF-8"))
```

R. Grin

Interface système

91

Chemin d'une ressource

- On peut parfois avoir besoin du chemin (dans le système d'exploitation) du fichier de ressources
- Pour l'obtenir à partir de l'URL :

```
String nom = urlRes.getPath();
// Voir cours sur les entrée-sorties
// (sur URL) pour utilité
// de URLDecoder.decode
nomFichier =
    URLDecoder.decode(nom, "ISO-8859-1");
```

R. Grin

Interface système

92

Récupérer une ressource multimédia

- Les ressources multimédia peuvent se récupérer directement à partir de l'URL du fichier :
 - `getImage(urlRes)` de la classe `Toolkit` pour les images
 - `new ImageIcon(urlRes)` pour une icône
 - `getAudioClip(urlRes)` pour un fichier son (seulement dans une *applet* ; voir `javax.sound.sampled.AudioSystem` sinon)

R. Grin

Interface système

93

Chargeur de classes quelconque

- Le nom passé à `getResource` peut être
 - absolu (commence par un « / ») : la recherche est effectuée par le chargeur de classes avec l'URL inchangé (le mode de recherche dépend du chargeur)
 - relatif (ne commence pas par un « / ») : le chemin associé au paquetage de la classe est ajouté au début ; par exemple `/fr/unice/toto/p1/` si la classe est du paquetage `fr.unice.toto.p1`, et ensuite la recherche se fait comme avec un chemin absolu

R. Grin

Interface système

94

Noms de ressources sous Windows

- Un nom relatif : `rep1/rep2/...`
- Un nom absolu : `/C:/rep1/...`

R. Grin

Interface système

95

URL d'une ressource d'un jar

- A utiliser si les ressources sont dans un autre jar que les classes
- Format : `! / sépare l'URL et l'entrée dans le jar`
`jar:url![entrée]`
- Par exemple :

```
jar:http://www.unice.fr/~toto/f.jar!
/fr/unice/fr/toto/Truc.class
ou (désigne tout le fichier jar)
jar:http://www.unice.fr/~toto/f.jar!
```
- `jar:file:/rep1/rep2/f.jar`

R. Grin

Interface système

96

Programmes externes

R. Grin

Interface système

97

Exécuter un programme externe

- La classe `java.lang.Runtime` contient des méthodes `exec()` pour exécuter un programme externe à la JVM :
 - `Process exec(String commande)`
 - `Process exec(String[] commande)` : la commande sous la forme d'un tableau de chaînes
 - `Process exec(String commande, String[] env)` et `Process exec(String[] commande, String[] env)` : on peut passer des variables d'environnements sous la forme `var=valeur`

R. Grin

Interface système

98

Classe `java.lang.Process`

- Les méthodes `exec` de `Runtime` renvoient un objet de type `Process`
- Les méthodes de la classe `Process` offrent une interface minimum avec le processus lancé :
 - `exitValue()` récupère le code retour du processus (erreur si le processus n'est pas terminé)
 - `waitFor()` attend la fin du processus et récupère le code retour
 - `destroy()` supprime le processus

R. Grin

Interface système

99

Classe `java.lang.Process`

- On peut aussi agir sur les 3 voies standard du processus :
 - `get{Input|Output|Error}Stream()` récupère les 3 voies standard du processus (sortie, entrée et erreur)
 - Attention ! `getInputStream` se branche sur la sortie standard du processus (on se place du point de vue du programme Java)
 - Il est conseillé de les récupérer rapidement ces voies pour éviter le blocage de certains processus

R. Grin

Interface système

100

Exemple de processus externe

```
Process p = Runtime.getRuntime().exec("who");
BufferedReader br = new BufferedReader(
    new InputStreamReader(p.getInputStream()));
ArrayList utilisateurs = new ArrayList();
String utilisateur;
while ((utilisateur = br.readLine()) != null) {
    utilisateurs.add(utilisateur);
    System.out.println(utilisateur);
}
```

Attention, risque de blocage de `readLine` avec fenêtre DOS de Windows

R. Grin

Interface système

101

Échanges avec le programme externe

- Si on a lancé un programme qui accepte des entrées par la voie standard d'entrée (un *shell* Unix par exemple), on peut lui envoyer des messages en utilisant `getOutputStream` :

```
PrintWriter pw = new PrintWriter(
    new OutputStreamWriter(
        p.getOutputStream()));
pw.println(message);
pw.flush();
...
```

R. Grin

Interface système

102

ProcessBuilder

- Elle a été introduite par le JDK 5 pour offrir plus de possibilités que celles offertes par la méthode `exec` de `Runtime`
- Sa méthode `start()` renvoie une instance de `Process`

R. Grin

Interface système

103

ProcessBuilder

- On peut
 - modifier le répertoire courant,
 - rediriger la voie standard des erreurs vers la voie de sortie standard,
 - récupérer l'environnement du processus (les valeurs des variables) et le modifier (par l'intermédiaire d'une Map)
- Cette classe n'est pas protégée contre les accès multiples (par plusieurs threads)

R. Grin

Interface système

104

Exemple de base

```
Process p =
    new ProcessBuilder("ipconfig", "/all")
        .start();
InputStream is = p.getInputStream();
BufferedReader br = new BufferedReader(
    new InputStreamReader(is));
String line;
while ((line = br.readLine()) != null) {
    System.out.println(line);
}
```

R. Grin

Interface système

105

Exemple avec modification de l'environnement

```
ProcessBuilder pb =
    new ProcessBuilder("uneCommande",
        "arg1", "arg2");
// Modification de l'environnement
Map<String, String> env = pb.environment();
env.put("VAR1", "myValue");
env.remove("VAR2");
env.put("VAR2", env.get("VAR1") + "suffix");
pb.directory("unRep"); // répertoire courant
Process p = pb.start();
```

R. Grin

Interface système

106

Classe Desktop

- Introduite avec le JDK 6, elle permet d'ouvrir un fichier avec l'application qui lui est liée dans le système, par exemple, d'ouvrir un fichier « .doc » avec Open Office ou Word
- Une fois l'application lancée, le code Java ne peut plus communiquer avec elle

R. Grin

Interface système

107

Commandes du shell

- Si on veut lancer une commande du langage de commande du shell (par exemple `cd`), il faut lancer l'exécution du shell avec la commande en argument
- Exemple sous Windows :

```
Runtime runtime =
    Runtime.getRuntime();
runtime.exec(new String[] {
    "cmd.exe", "/C", "dir" });
```

R. Grin

Interface système

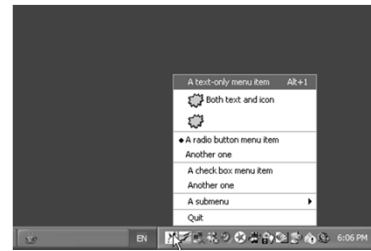
108

Interface avec la barre des tâches du bureau

R. Grin

Interface système

109



Barre de tâche sous Windows

R. Grin

Interface système

110

SystemTray

- Depuis Java SE 6, la barre des tâches est représentée par la classe `java.awt.SystemTray`
- Pour la portabilité entre différents systèmes d'exploitation, il vaut mieux vérifier si cette fonctionnalité est supportée, avec la méthode `static SystemTray.isSupported()`
- Il faut tout d'abord obtenir une instance avec la méthode `static getSystemTray()`

R. Grin

Interface système

111

TrayIcon

- Ensuite, il faut placer une icône pour représenter l'application en utilisant la classe `java.awt.TrayIcon`
- Cette classe contient une méthode `setPopupMenu` qui permet d'associer à l'icône un menu popup qui apparaîtra dans la barre de tâches en cliquant sur l'icône

R. Grin

Interface système

112

Exemple (1/2)

```
if (SystemTray.isSupported()) {
    SystemTray tray =
        SystemTray.getSystemTray();
    Image image = ...; // Récupérer image
    PopupMenu popup = new PopupMenu();
    ... // Construire menu popup
    final TrayIcon trayIcon =
        new TrayIcon(image, "Tray Demo", popup);
}
```

R. Grin

Interface système

113

Exemple (2/2)

```
ActionListener al = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        trayIcon.displayMessage("...", "...",
            TrayIcon.MessageType.INFO);
    }
};
trayIcon.addActionListener(al);
try { tray.add(trayIcon); }
catch (AWTException e) {
    System.err.println("Impossible ajouter icône");
}
```

Message popup affiché temporairement au-dessus de l'icône

R. Grin

Interface système

114

Gestion des préférences de l'utilisateur

R. Grin

Interface système

115

Description

- Sauvegarder et récupérer les préférences pour un utilisateur particulier ou pour tous les utilisateurs, entre 2 sessions différentes
- Cette API est apparue avec la version 1.4 (paquetage `java.util.prefs`)
- Auparavant on utilisait les propriétés
- La façon de sauvegarder les préférences entre 2 sessions d'utilisation dépend du système d'exploitation ; le développeur n'a pas besoin de savoir comment ça se passe

R. Grin

Interface système

116

Exemples d'utilisation

- Conserver le chemin d'un répertoire où un utilisateur garde certaines informations ou récupère des fichiers
- Conserver le nom d'un fichier ou la couleur préférée d'un utilisateur

R. Grin

Interface système

117

Concepts de base

- Les préférences sont des couples clé-valeur
- Elles sont rangées dans des nœuds de préférences
- La convention est de faire correspondre ces nœuds aux paquetages
- 2 types de préférences correspondant à 2 types de nœuds :
 - utilisateur (concerne un seul utilisateur)
 - système (partagé par tous les utilisateurs)

R. Grin

Interface système

118

Noeud

- Un nœud correspond à un ensemble de préférences, qui est entreposé de façon persistante dans l'application
- Chaque nœud contient une *map* dont chaque entrée est le nom d'une propriété et la valeur est la valeur de la propriété
- Chaque propriété correspond à une préférence de l'utilisateur, par exemple un nom de fichier ou une couleur

R. Grin

Interface système

119

Nom d'un nœud

- Structure hiérarchique
- Un nœud a un nom, un chemin absolu, un nom relatif (par rapport à un nœud ancêtre)
- Le nœud racine de l'arborescence a le nom « / »
- Les noms absolus sont du type des noms de fichiers Unix : `/nœud1/nœud2/...`
- Un nom relatif est du type `nœud1/nœud2/...`
- Le nom relatif d'un nœud par rapport à lui-même est la chaîne vide

R. Grin

Interface système

120

Nom d'un nœud

- Pour éviter qu'un nom de nœud mal écrit ne provoque une erreur à l'exécution, l'usage est de donner à un nœud le nom du paquetage d'une classe (objet de la classe **Class<T>**)
- Le plus simple, si ça convient est d'associer au nœud le nom du paquetage de la classe qui utilise la préférence
- Si plusieurs applications utilisent le même paquetage, on peut distinguer le nom des propriétés en préfixant par le nom de l'application qui l'utilise (ou par un autre moyen)

R. Grin

Interface système

121

Classe Preferences

- La classe **Preferences** correspond à un nœud de préférences
- On récupère un nœud avec des méthodes **static** (modèle de conception « fabrique ») de la classe **Preferences** ; elles renvoient une instance de **Preferences** :
userNodeForPackage(Class<?>)
systemNodeForPackage(Class<?>)
- Si le nœud n'existe pas, il est créé

R. Grin

Interface système

122

Exemple

- **Preferences prefs =**
userNodeForPackage(this.getClass());
- Si la classe est **fr.unice.Classe**, le nom du nœud de la préférence sera le nom du paquetage, en remplaçant les **“.”** par des **“/”** :
/fr/unice

R. Grin

Préférences

page 123

Propriétés

- Quand on a un nœud, on peut ajouter ou lire des préférences comme dans une *map*
- Les clés qui identifient les préférences sont des **String**
- Les valeurs des préférences sont de type **String, boolean, byte[], double, float, int, long**
- Les entrées de type **String** sont limitées à 8192 caractères ; la taille des tableaux d'octets ne doit pas dépasser 6144

R. Grin

Interface Système

124

Lire et écrire une propriété

- Des méthodes **getXXX** permettent de récupérer la valeur d'une préférence
- Des méthodes **putXXX** permettent de ranger la valeur d'une préférence
- Les méthodes sont **putInt, putBoolean, putFloat, putLong, putByteArray**, et les méthodes **get** correspondantes
- **get()** et **put(String)** correspondent à une préférence de type **String**

R. Grin

Interface Système

125

Sauvegarder une préférence

```
// Récupère le nœud
Preferences prefs =
    Preferences.userNodeForPackage(
        getClass());
// Mets la valeur de la préférence pref1
prefs.put("pref1", "v11");

// Pour les nœuds « système » :
Preferences prefs =
    Preferences.systemNodeForPackage(
        getClass());
```

R. Grin

Interface système

126

Lire une préférence

- On doit donner une valeur par défaut ; cette valeur sera retournée si la préférence ne peut être lue :

```
String valeurPropriété =  
    prefs.get("pref1", "defaut");
```

R. Grin

Interface système

127

Supprimer une préférence

- La méthode `remove(String)` supprime la préférence du nom passé en paramètre
- `clear()` supprime toutes les préférences du nœud ; elle peut lancer une exception contrôlée par le compilateur (`BackingStoreException`)

R. Grin

Interface système

128

Sauvegarde des préférences

- La sauvegarde des préférences entre les sessions de travail est automatique et dépend du système dans lequel on travaille
- Sous Windows les préférences sont sauvegardées dans le registre
- Sous Unix, dans le système de fichiers
- On peut aussi importer ou exporter des préférences dans un fichier XML avec des `{In|Out}putStream`

R. Grin

Interface système

129

Interface avec un autre langage que Java

- Nous n'étudierons pas cette fonctionnalité du langage Java
- *Java Native Interface* (JNI) du JDK fournit une API pour appeler dans un programme Java des méthodes écrites dans un autre langage (C le plus souvent)

R. Grin

Interface système

130