Chaines de caractères

Table des matières

Conversions Chaines-Nombres

Il arrive souvent que l'on doive convertir une chaine de caractères en nombre. Nous allons revoir comment effectuer cette conversion en utilisant les classes Integer et Double du package java.lang. La chaine de caractères à convertir étant souvent fournie par un utilisateur, des erreurs qui rendent la conversion impossible peuvent se produire. Nous verrons donc aussi comment gérer ses erreurs à l'aide des exceptions.

Utiliser les classes Integer et Double

Nous avons déjà largement utilisé les types int et double. Java fournit des classes Integer et Double qui permettent de traiter les nombres comme des objets. Ces classes permettent de convertir simplement des chaines de caractères en nombre.

Utiliser les constructeurs

Les classes Integer et Double possèdent un constructeur qui utilise une chaine de caractères en paramètre. Elles disposent aussi de méthodes intValue et doubleValue qui renvoient des valeurs de type int et double. En combinant ces deux idées on peut facilement transformer une chaine de caractères en nombre.

Par exemple, pour convertir une chaine S en un entier i, on écrira:

```
Integer I=new Integer(S);
int i=I.intValue();
```

Attention, la variable I est un objet de type Integer alors que la variable i est un entier.

De même, pour convertir une chaine S en un nombre à virgule d, on écrira:

```
Double D=new Double(S);
double d=D.doubleValue();
```

Ici aussi on a une variable D qui est un objet de type Double et une variable d qui est un nombre décimal.

Utiliser la méthode statique valueOf

Les classes Integer et Double possèdent la méthode statique valueOf qui transforme une chaine de caractères en un objet. En faisant appel aux méthodes intValue ou doubleValue de cet objet on obtient un résultat sous forme de nombre.

Par exemple, pour convertir une chaine S en un entier i, on écrira:

```
int i=Integer.valueOf(S).intValue();
```

De même, pour convertir une chaine S en un nombre à virgule d, on écrira:

```
double d=Double.valueOf(S).doubleValue();
```

Gestion des erreurs

Java fournit un mécanisme de gestion des erreurs basé sur les exceptions qui sont des objets dérivés de la classe Exception.

Capture d'une exception

Lorsqu'un bloc de code peut provoquer des erreurs, on peut les capturer en le plaçant dans la structure:

```
try {
 instruction1;
 instruction2;
 ....
} catch (Exception e) {
 instruction_si_erreur;
 ....
} finally {
 ....
}
```

Cela signifie qu'on essaie d'exécuter le bloc d'instructions suivant **try** et que si une erreur représentée par l'exception e se produit, on exécute le bloc d'instructions suivant **catch**. Le bloc suivant **finally** est facultatif, il est exécuté dans tous les cas.

Pour obtenir des renseignements sur l'exception e qui a été déclenchée on peut utiliser ses méthodes getMessage et toString qui renvoient des chaines de caractères.

Mise en pratique

Ecrivons un programme qui divise deux nombres passés en paramètres en gérant les exceptions. Nous aurons à convertir les chaines fournies en paramètres en nombres, puis à effectuer la division. Deux exceptions peuvent se produire : il n'y a moins de deux paramètres fournis ou les chaines ne peuvent pas être converties.

On peut écrire ce programme de la façon suivante:

```
public class ajoute {
 public static void main(String[] args) {
  try {
   double d1=Double.valueOf(args[0]).doubleValue();
   double d2=Double.valueOf(args[1]).doubleValue();
   double r=d1+d2;
   System.out.println(d1+"+"+d2+"="+r);
  } catch (Exception e) {
   System.out.println("Erreur :");
   System.out.println(e.toString());
   System.out.println("Fin erreur.");
  }
 }
}
```

En exécutant le programme avec des paramètres incorrects on obtient des messages indiquant le type d'exception rencontrée.

Par exemple, "java ajoute 3" provoque l'affichage suivant:

```
Erreur :
java.lang.ArrayIndexOutOfBoundsException: 1
Au revoir...
```

De même, "java ajoute 5 w" provoque l'affichage de:

```
Erreur :
java.lang.NumberFormatException: w
Au revoir...
```

Amélioration

La structure try...catch... peut être complétée en tenant compte du type d'exception rencontrée. Il suffit d'introduire plusieurs clauses catch associée à des types d'exceptions moins généraux que Exception.

On peut ainsi reprendre le programme précédent pour qu'il affiche des messages d'erreur adaptés. Cela nous donne :

```
public class ajout {
 public static void main(String[] args) {
  try {
   double d1=Double.valueOf(args[0]).doubleValue();
   double d2=Double.valueOf(args[1]).doubleValue();
   double r=d1+d2;
   System.out.println(""+d1+"+"+d2+"="+r);
  } catch (ArrayIndexOutOfBoundsException e) {
   System.out.println("Entrer deux paramètres !");
  } catch (NumberFormatException e) {
   System.out.println("Les paramètres doivent être des nombres !");
  } catch (Exception e) {
   System.out.println("Erreur imprévue.");
  } finally {
   System.out.println("Au revoir...");
  }
 }
}
```

Cette fois, "java ajoute 3" provoque l'affichage suivant:

```
Entrer deux paramètres !
Au revoir...
```

Et, "java ajoute 5 w" provoque l'affichage de:

```
Les paramètres doivent être des nombres !
Au revoir...
```

Interpréter des chaines

La classe StringTokenizer

La classe StringTokenizer fait partie du package java.util. Elle permet de décomposer une chaîne de caractères en une suite de "mots" séparés par des "délimiteurs".

Constructeurs

La classe StringTokenizer propose 3 constructeurs :

- **StringTokenizer(String str, String delim, boolean returnTokens)**
  - str est la chaîne à analyser
  - delim est une chaîne contenant les délimiteurs reconnus
  - returnTokens indique si les délimiteurs doivent être renvoyés comme parties de la chaîne.
- **StringTokenizer(String str, String delim)**
  Les paramètres définissent la chaîne à analyser et les délimiteurs. Par défaut, ceux-ci ne sont pas renvoyés comme éléments de la chaîne.
- **StringTokenizer(String str)**
  str est la chaîne à analyser; les délimiteurs sont les caractères espace, tabulation, retour chariot et changement de ligne.

Utilisation

La classe StringTokenizer fournit deux méthodes importantes qui permettent d'obtenir les différentes parties d'une chaîne l'une après l'autre : la méthode **hasMoreTokens** indique s'il reste des éléments à extraire; la méthode **nextToken** renvoie l'élément suivant.

Exemple d'utilisation :

```
StringTokenizer st = new StringTokenizer("this is a test");
while (st.hasMoreTokens()) {
 System.out.println(st.nextToken());
}
```

Ce code fournit la sortie suivante :

```
this
is
a
test
```

Application

Nous allons écrire une applet qui se contente de dessiner un disque coloré. Elle lira la couleur du fond et la couleur du disque dans les paramètres fournis par le fichier html.

Sans lecture de paramètres

Commençons par écrire l'applet sans tenir compte des paramètres fournis par le fichier html.

```
import java.applet.*;
import java.awt.*;

public class disque extends Applet {

 public void init() {
  //couleurs par défaut
  setBackground(Color.blue);
  setForeground(Color.yellow);
 }
```

```
 public void paint(Graphics g) {
  int rayon=getSize().height/4;
  int x=(getSize().width-2*rayon)/2;
  int y=(getSize().height-2*rayon)/2;
  g.fillOval(x,y,2*rayon,2*rayon);
 }

}
```

Lecture des paramètres

Les paramètres seront associés aux mots "fond" et "disque". Les couleurs seront définies par trois entier séparés par des virgules. L'appel de l'applet dans le fichier html se fera donc de la façon suivante :

```
<APPLET CODE="dique.class" WIDTH="200" HEIGHT="200">
<PARAM NAME="fond" VALUE="255,0,0">
<PARAM NAME="disque" VALUE="128,255,128">
</APPLET>
```

La méthode init de l'applet va devoir récupérer les chaînes fournies par l'attribut VALUE et les transformer en objets de type Color. Nous écrirons une méthode couleur(String S) qui se chargera de cette tâche.

```
 public void init() {
  String S;
  Color col;
  //couleurs par défaut
  setBackground(Color.blue);
  setForeground(Color.yellow);
  //lecture des paramètres
  S=getParameter("fond");
  if (S!=null) {
   col=couleur(S);
   if (col!=null) setBackground(col);
  }
  S=getParameter("disque");
  if (S!=null) {
   col=couleur(S);
   if (col!=null) setForeground(col);
  }
 }
```

La méthode couleur reçoit en paramètre une chaine contenant trois nombres entiers séparés par des virgules. Nous allons donc utiliser un objet de type StringTokenizer pour les extraire. Le résultat renvoyé par la méthode sera la couleur lorsque tout s'est bien passé et null lorsqu'une erreur s'est produite.

```
Color couleur(String S) {
 int r,g,b;
 Color col=null;
 StringTokenizer st=new StringTokenizer(S,",;");
 try {
  r=Integer.parseInt(st.nextToken());
  g=Integer.parseInt(st.nextToken());
  b=Integer.parseInt(st.nextToken());
  col=new Color(r,g,b);
```

```
        } catch (Exception e) {}
        return col;
    }
```

Exercices

1. **Opérations(1)**
   Ecrire une applet qui contient deux lignes de saisie destinées à recueillir des nombres, quatre boutons déclenchant les 4 opérations et une zone d'affichage du résultat obtenu ou d'un message d'erreur.

2. **Opérations(2)**
   Ecrire une applet qui contient une ligne de saisie destinée à recueillir un calcul du type a+b ou a-b ou a*b ou a/b, exécute ce calcul et affiche le résultat.

3. **Histogramme**
   Ecrire une applet qui dessine un histogramme à partir de données passées en paramètre dans le fichier html sous la forme
   ```
   <PARAM NAME="di" VALUE="nombre">
   ```
   où i est un numéro et nombre la valeur associée.