T.P. N° 1

```
1  import java.io.*;
2
2  public class JTP1
2  {
3       public static void main (String[] args)
4            throws IOException
4       {
5            System.out.print("Tapez votre nom : ");
6            byte bArray[] = new byte[80];
             System.in.read(bArray);
             String strNom = new String(bArray);
7            System.out.print("Bonjour ");
             System.out.println(strNom);
4       }
2  }
```

COMMENTAIRES

1 Au début du programme, Tous les *packages* utilisés doivent être déclarés par l'instruction **import**. On utilise les flux d'entrées/sorties **System.in** et **System.out** du package **java.lang** .ainsi que la classe **Object** dont hérite par défaut la classe **JTP1** de notre application. Comme ce *package* est quasiment incontournable, le compilateur que nous avons utilisé ici effectue l'importation implicitement. Il n'est donc pas nécessaire de la déclarer. Par contre, la fonction **read** de la classe **InputStream** utilisée pour la lecture au clavier génère une exception de classe **IOException**. Cette classe se trouve dans le *package* **java.io** qu'il faut donc importer.

2 En java, on ne peut créer de fonction qu'à l'intérieur d'une classe. Au niveau le plus bas, il faut au moins une classe appelée classe application dont on peut librement choisir le nom (ici **JTP1**). Cette classe doit être déclarée **public**.

3 Avec la commande **`JAVA JTP1 <enter>`**, la machine virtuelle java va appeler la classe **JTP1** et y rechercher une fonction **main**, cette fonction constituant le point d'entrée du programme. Cette fonction doit être déclarée **public** et **static**. Elle reçoit en argument dans le tableau de chaînes de caractères **args**, les éventuels paramètres passés à la machine virtuelle java. Par exemple si la commande est la suivante :

```
C:\>JAVA JTP1 PARAM1 PARAM2 PARAM3 <enter>
```

args[0] contient **"JTP1"**, **args[1]** contient **"PARAM1"**, **args[2]** contient **"PARAM2"**, **args[3]** contient **"PARAM3"**, etc.

4 La fonction **main** fait appel à la fonction **read** de la classe **InputStream**. Cette fonction génère une exception de la classe **IOException**. Cette exception, n'étant pas assumée par la fonction **main** (le traitement des exception dans les fonctions fera l'objet de plusieurs T.P. ultérieurement), doit être transmise à la fonction appelante (ici la machine virtuelle java) par l'instruction **throws**. La classe **IOException** est définie dans le *package* **java.io** qui doit être importé (voir 1).

5 Le flux d'affichage à l'écran **System.out** est une instance de la classe **PrintStream**. On peut donc utiliser la fonction membre **print** de cette classe pour afficher le prompt de saisie.

6 Le flux de lecture au clavier **System.in** est une instance de la classe **InputStream**. Il n'existe pas de fonction membre pour lire directement des chaînes de caractère de type **String**. La fonction membre **read** de la classe **InputStream** copie les caractères tapés au clavier dans un tableau de caractères. Pour résoudre ce problème on utilise l'astuce suivante :

- On déclare un tableau de **byte** (ici **bArray**) pour lequel on réserve 80 octets.
- On utilise la fonction **read** pour lire le clavier dans **bArray**. Cette fonction génère une exception de classe **IOException**. Comme cette exception n'est pas traitée ici, il faut la transmettre automatiquement à la fonction appelante par l'instruction **throws** (voir 4).
- On déclare une variable **strNom** de type **String** que l'on construit à partir du tableau de caractères pour en recopier le contenu.

7 On utilise la fonction membre **print** de la classe **PrintStream** pour afficher **"Bonjour "** et la fonction membre **println** pour afficher le contenu de la variable **strNom**. Les fonctions **print** et **println** fonctionne pareillement, à la différence que **println** ajoute un retour de chariot pour passer à la ligne.

*Ce programme se trouve dans le fichier **JTP1.java**.*

T.P. N° 2

```
1   public class JTP2
    {
        public static void main (String[] args)
2           throws Exception
        {
            System.out.print("Valeur du plus grand entier :");
            byte bArray[] = new byte[80];
            System.in.read(bArray);
            String  strLine = new String(bArray);
            strLine = strLine.substring(0, strLine.indexOf("\r\n"));
            Integer  intFin = new Integer(strLine);
            int nFin = intFin.intValue();
3           for (int i = 1; i < nFin; i++)
            {
4               System.out.print(i);
                System.out.print("\t" + i * i);
                System.out.println("\t" + Math.sqrt(i));
            }
        }
    }
```

COMMENTAIRES

1 La fonction **main** fait appel à la fonction **read** de la classe **InputStream**. Cette fonction génère une exception de la classe **IOException**. Cette exception, n'étant pas assumée par la fonction **main** (le traitement des exception dans les fonctions fera l'objet de plusieurs T.P. ultérieurement), doit être transmise à la fonction appelante (ici la machine virtuelle java) par l'instruction **throws**. Comme la nature de l'exception n'a pas d'importance dans ce TP (puisqu'on ne les gère pas), on transmet toutes les exceptions en utilisant la classe générique **Exception** de laquelle est dérivée **IOException**. Cette classe est définie dans le *package* **java.lang** qui est importé par défaut. Ce qui évite d'avoir à importer le package **java.io**.

2 Il n'existe pas de fonction simple qui permette de lire au clavier un nombre. On procède donc par étape :

- On déclare un tableau **bArray** de 80 **byte**s
- On utilise la fonction **read** de la classe **InputStream** (le flux **System.in**, associé au clavier, est une instance de cette classe) pour lire au clavier une série de caractères dans **bArray**.
- On déclare une chaîne de caractères **strLine** que l'on initialise à partir du tableau de caractère **bArray**. On verra plus tard, quand on abordera les concepts de la programmation orientée objet, que la variable strLine ne contient qu'une référence qui doit être instanciée (construite) à l'aide de l'instruction **new** pour avoir une existence réele.
- **strLine** contient 80 caractères (la longueur du tableau de **bytes** utilisé pour sa construction). On ne garde que les caractères qui précèdent le retour de chariot (**"\r\n"**). Les méthodes de la classe String qui permettent de manipuler les chaînes de caractères feront l'objet du TP suivant.

substring permet d'extraire une partie d'une chaîne de caractères. **indexOf** permet de repérer la position du chaîne dans une autre.

- On déclare un entier **intFin** qui contiendra la valeur numérique correspondant à ce qui a été saisi au clavier. **intFin** n'est en fait qu'une référence d'**Integer** qui doit être instanciée (construite) à partir de la chaîne **strLine**.

- On peut enfin convertir **intFin** dans une variable (**nFin**) de type numérique **int** en utilisant la méthode **intValue** de la classe **Integer**.

On peut se poser la question de la nécessité d'obtenir une valeur de type **int** plutôt que de ce satisfaire d'une valeur de type **Integer**. La classe **Integer**, du fait qu'elle soit intégrée dans la hiérarchie de classe issue de **Object**, offre de nombreux avantages, notamment l'interopérabilité avec toutes les classes issues **Object**. Ce que ne permet pas le type **int**. Cela est particulièrement utile en programmation orientée objet. En contre partie, la classe **integer** ne permet pas le calcul arithmétique et l'utilisation des opérateurs, ce qu'en revanche permet le type **int**. Dans notre cas, des calculs arithmétiques sont envisagés, donc il faut pousser la conversion jusqu'à **int**.

3 En JAVA, on déclare les variables au moment où on en a besoin. L'itérateur **i** peut être déclaré dans l'instruction **for**. **i** est de type **int** et est immédiatement initialisé à 1.

4 Il n'est pas prévu en JAVA de fonction pour gérer l'affichage selon un format élaboré. On utilise le caractères **"\t"** pour générer une tabulation ce qui permet un semblant d'affichage en colonne. L'écriture de fonctions plus élaborées fera l'objet de TP ultérieurs.

*Ce programme se trouve dans le fichier **JTP2.java**.*

T.P. N° 3

```
public class JTP3
{
     public static void main (String[] args)
1         throws Exception
     {
2         System.out.println("Taper une chaine de caracteres :");
          byte bArray[] = new byte[80];
          System.in.read(bArray);
          String  strKB = new String(bArray);
          String strLine = strKB.substring(0, strKB.indexOf("\r\n"));

3         System.out.println("No du champ à extraire :");
          System.in.read(bArray);
          strKB = new String(bArray);
          strKB = strKB.substring(0, strKB.indexOf("\r\n"));
          Integer intChamp = new Integer(strKB);
          int nChamp = intChamp.intValue();

4         int nDebut = 0;
          String strChamp = "";
5         for (int i = 0; i < nChamp; i++)
          {
6              int nFin = strLine.indexOf(",", nDebut);
               if (nFin == -1)
               {
7                   if (nDebut < strLine.length())
                    {
                         strChamp = strLine.substring(nDebut);
                         nDebut = strLine.length();
                    }
8                   else
                    {
                         strChamp = "";
                    }
               }
9              else
               {
                    strChamp = strLine.substring(nDebut, nFin);
                    nDebut = nFin + 1;
               }
          }
10        System.out.println(  "Champ No" + nChamp +
                               " : [[" + strChamp + "]]");
     }
}
```

COMMENTAIRES

1 La fonction **main** fait appel à des fonctions qui génèrent des exceptions. Ces exceptions, n'étant pas assumée par la fonction **main** (le traitement des exception dans les fonctions fera l'objet de plusieurs T.P. ultérieurement), doit être transmise à la fonction appelante (ici la machine virtuelle java) par l'instruction **throws**. Comme la nature de l'exception n'a pas d'importance dans ce TP (puisqu'on ne les gère pas), on transmet toutes les exceptions en utilisant la classe générique **Exception**. Cette classe est définie dans le *package* **java.lang** qui est importé par défaut.

2 Saisie d'une chaîne de caractères au clavier. On passe par les étapes standards, à savoir, le tableau de **byte**, puis la construction de la chaîne à partir du tableau lu au clavier.

3 Saisie du numéro de champ. On passe par les étapes standard, à savoir le tableau de **byte**, puis la construction de la chaîne de caractère à partir du tableau lu au clavier, la construction d'un **Integer** à partir de cette chaîne et enfin la conversion dans le type **int**. A cette étape, **strLine** contient la chaîne et **nChamp** contient le numéro de champ.

4 La boucle qui va suivre va compter les champs. On initialise deux variables, **nDebut** qui contiendra toujours la position du premier caractère du champ en cours et **strChamp** qui contiendra en tant que résultat la chaîne de caractères composant le champ trouvé. **strChamp** est initialisé à une chaîne vide, ce qui correspond à un N° de champ inférieur à 1.

5 La boucle **for** va utilisé un itérateur **i** et explorer la chaîne jusqu'à trouver le numéro de champ correct.

6 On utilise la méthode **indexOf** de la classe **String** pour repérer la position de la virgule qui sépare les champs. **nFin** contient la position de la virgule qui suit la position **nDebut** donc la virgule qui termine le champ en cours. Si aucune virgule n'est trouvée, **indexOf** renvoie -1. Cela arrive dans deux cas : quand le N° de champ demandé correspond au dernier de la chaîne, ou quand le N° de champ demandé est supérieur au nombre de champs qui composent la chaîne.

7 Dans le cas ou le champ demandé est le dernier de la chaîne, on utilise la méthode **substring** de la classe **String** pour extraire la fin de **strLine** à partir de la position **nDebut**. **nDebut** doit être positionné à la fin de la chaîne pour les itération suivante (correspondant à un N° de champ supérieur au nombre de champ de la chaîne).

8 Dans le cas ou le N° de champ demandé est supérieur au nombre de champs composant la chaîne, le résultat est une chaîne vide.

9 Dans tous les autres cas (cas normal), on utilise la méthode **substring** de la classe **String** pour extraire le champ et le ranger dans **strChamp**. nDebut est positionné sur le caractère qui suit la virgule poue les itérations suivantes.

10 Quand la boucle est terminée, **strChamp** contient le champ demandé ou bien une chaîne vide si le N° de champ est inférieur à 1 ou s'il est supérieur au nombre de champs qui composent la chaîne.

*Ce programme se trouve dans le fichier **JTP3.java**.*

T.P. N° 4

```
public class JTP4
{
      public static void main (String[] args)
1           throws Exception
      {
            System.out.println("Taper une chaine de caracteres :");
2           String strLine = getString();

            System.out.print("No du champ à extraire :");
3           int nChamp = getInteger();

4           String strChamp = fieldOf(strLine, nChamp);
7           System.out.println(  "Champ No" + nChamp +
                                 " : [[" + strChamp + "]]");

            System.out.print("Calcul de la factorielle de :");

5           long lN = getLong();
6           long lF = factorielle(lN);
7           System.out.println("Fact(" + lN + ") = " + lF);
      }


2     public static String getString()
            throws Exception
      {
            byte bArray[] = new byte[80];
            System.in.read(bArray);
            String  strResult = new String(bArray);
            return strResult.substring(0, strResult.indexOf("\r\n"));
      }


3     public static int getInteger()
            throws Exception
      {
            String strKB = getString();
            Integer intResult = new Integer(strKB);
            return intResult.intValue();
      }


5     public static long getLong()
            throws Exception
      {
            String strKB = getString();
            Long lResult = new Long(strKB);
            return lResult.longValue();
      }


8     public static double getDouble()
            throws Exception
      {
            String strKB = getString();
            Double dResult = new Double(strKB);
            return dResult.doubleValue();
      }

4     public static String fieldOf(String strLine, int nField)
      {
            return fieldOf(strLine, nField, ',');
      }
```

9

```
        public static String fieldOf(String strLine, int nField, int sep)
        {
              int nDebut = 0;
              String strResult = "";
              for (int i = 0; i < nField; i++)
              {
                    int nFin = strLine.indexOf(sep, nDebut);
                    if (nFin == -1)
                    {
                          if (nDebut < strLine.length())
                          {
                                strResult = strLine.substring(nDebut);
                                nDebut = strLine.length();
                          }
                          else
                          {
                                strResult = "";
                          }
                    }
                    else
                    {
                          strResult = strLine.substring(nDebut, nFin);
                          nDebut = nFin + 1;
                    }
              }
              return strResult;
        }
```

6

```
        public static long factorielle(long n)
        {
              if (n == 1)
              {
                    return 1;                              //  1er théorème
              }
              else
              {
                    return n * factorielle(n - 1);  // 2ème théorème
              }
        }
  }
```

COMMENTAIRES

1 La fonction main fait appel à des fonctions (**getString**, **getInteger**, **getLong**) qui sont susceptibles de générer des exceptions. N'étant pas assumées pour le moment, elles sont transmises au programme appelant (la machine virtuelle) par l'instruction **throws**.

2 Lecture d'une chaîne de caractères à l'aide de la fonction **getString**. La fonction **getString** est déclarée **public** et **static** pour qu'elle puisse être utilisée plus tard dans d'autres classes. Ces concepts seront détaillés lors de T.P. ultérieurs. Cette fonction est de type **String**, c'est à dire que son résultat, renvoyé au programme appelant par l'instruction **return**, doit être de ce type. C'est le cas, cas le résultat de la fonction **substring** affectée à la variable **strResult** est de ce type. Bien que cette fonction ne reçoive pas de paramètre, il faut préciser les parenthèses (vide). **getString** utilise la méthode **read** de la classe **InputStream** qui génère une exception de type **IOException**. Cette exception doit être transmise au programme appelant par l'instruction **throws**. On utilise **Exception**, la classe générique de toutes les exceptions, pour ne pas avoir à importer le *package* **java.io** dans lequel cette classe est définie.

3 Lecture d'un entier à l'aide de la fonction **getInteger**. La fonction **getInteger** est déclarée **public** et **static** pour qu'elle puisse être utilisée plus tard dans

d'autres classes. Ces concepts seront détaillés lors de T.P. ultérieurs. Cette fonction est de type **int**, c'est à dire que son résultat, renvoyé au programme appelant par l'instruction **return**, doit être de ce type. Pour obtenir ce résultat, on procède par les étapes décrites dans les T.P. précédents, à savoir la conversion de la chaîne saisie au clavier en objet **Integer** puis en **int** grâce à la méthode **intValue** de cette classe. Bien que cette fonction ne reçoive pas de paramètre, il faut préciser les parenthèses (vide). **getInteger** utilise la fonction **getString** (définie ci-dessus) qui génère une exception de type **IOException**. et la construction d'un **Integer** à partir d'une **String** qui génère une exception de type **NumberFormatException**. Ces exceptions doivent être transmises au programme appelant par l'instruction **throws**. On utilise **Exception**, la classe générique de toutes les exceptions, pour ne pas avoir à importer de *package* particulier dans lequel ces classes seraient définies.

4 Extraction d'un champ d'une chaîne de caractères. Il s'agit ici de la première version de **fieldOf** qui ne reçoit que 2 paramètres, un paramètre de type **String** (la chaîne de laquelle il faut extraire un champ), et un paramètre de type **int** (le N° du champ demandé). Cette fonction est définie pour ne pas avoir à préciser le séparateur de champ par défaut (ici la virgule). Cette fonction appelle l'autre version (voir 9) qui comprend un troisième paramètre : le séparateur.

5 Lecture d'un entier à l'aide de la fonction **getLong**. La fonction **getLong** est déclarée **public** et **static** pour qu'elle puisse être utilisée plus tard dans d'autres classes. Ces concepts seront détaillés lors de T.P. ultérieurs. Cette fonction est de type **long**, c'est à dire que son résultat, renvoyé au programme appelant par l'instruction **return**, doit être de ce type. Pour obtenir ce résultat, on procède par les étapes décrites dans les T.P. précédents, à savoir la conversion de la chaîne saisie au clavier en objet **Long** puis en **long** grâce à la méthode **longValue** de cette classe. Bien que cette fonction ne reçoive pas de paramètre, il faut préciser les parenthèses (vide). **getLong** utilise la fonction **getString** (définie ci-dessus) qui génère une exception de type **IOException**. et la construction d'un **Long** à partir d'une **String** qui génère une exception de type **NumberFormatException**. Ces exceptions doivent être transmises au programme appelant par l'instruction **throws**. On utilise **Exception**, la classe générique de toutes les exceptions, pour ne pas avoir à importer de *package* particulier dans lequel ces classes seraient définies.

6 Calcul de factorielle. Cette fonction n'est indiquée ici que pour fournir un exemple de fonction récurcive :
- 1! = 1
- n! = n x (n-1)!

7 Affichage des résultats. Bien que la fonction **println** n'accepte qu'un seul paramètre de type **String**, on utilise la faculté de tous les types de données de JAVA de se convertir implicitement en **String** pour afficher des données complexes à l'aide de l'opérateur de concaténation +.

8 Lecture d'un entier à l'aide de la fonction **getDouble**. La fonction **getDouble** est déclarée **public** et **static** pour qu'elle puisse être utilisée plus tard dans d'autres classes. Ces concepts seront détaillés lors de T.P. ultérieurs. Cette

fonction est de type double, c'est à dire que son résultat, renvoyé au programme appelant par l'instruction **return**, doit être de ce type. Pour obtenir ce résultat, on procède par les étapes décrites dans les T.P. précédents, à savoir la conversion de la chaîne saisie au clavier en objet **Double** puis en **double** grâce à la méthode **doubleValue** de cette classe. Bien que cette fonction ne reçoive pas de paramètre, il faut préciser les parenthèses (vide). **getDouble** utilise la fonction **getString** (définie ci-dessus) qui génère une exception de type **IOException**. et la construction d'un **Double** à partir d'une **String** qui génère une exception de type **NumberFormatException**. Ces exceptions doivent être transmises au programme appelant par l'instruction **throws**. On utilise **Exception**, la classe générique de toutes les exceptions, pour ne pas avoir à importer de *package* particulier dans lequel ces classes seraient définies.

9 Extraction d'un champ d'une chaîne de caractères. Il s'agit ici de la deuxième version de **fieldOf** qui reçoit que 3 paramètres, un paramètre de type **String** (la chaîne de laquelle il faut extraire un champ), un paramètre de type **int** (le N° du champ demandé) et un paramètre de type **char** (le caractère de séparation des champs). On utilise ici, le concept de surcharge qui permet de créer plusieurs fonctions qui ont le même nom. Le compilateur JAVA sait faire la différence car la signature des deux fonctions **fieldOf** diffère par leur nombre respectif de paramètres (2 pour l'une, 3 pour l'autre). Pour des précisions sur l'algorithme utilisé pour l'extraction d'un champ dans une chaîne de caractères, se référer au T.P. précédent.

Ce programme se trouve dans le fichier ***JTP4.java****.*

T.P. N° 5

```
1   import java.io.*;

    public class JTP5
    {
2       public static void main (String[] args)
        {
3           try
            {
                System.out.println("Taper une chaine de caracteres :");
4               String strLine = getString();

                System.out.print("No du champ à extraire :");
5               int nChamp = getInteger();

6               String strChamp = fieldOf(strLine, nChamp);
                System.out.println("Champ No" + nChamp + " : [[" +
                                   strChamp + "]]");
            }

7           catch (IOException ioe)
            {
                System.err.println(ioe);
                System.err.println("Problème lors de la lecture au" +
                                   " clavier.");
            }

8           catch (NumberFormatException nfe)
            {
                System.err.println(nfe);
                System.err.println("Le No de champ doit etre entier.");
            }

9           catch (IndexOutOfBoundsException ioobe)
            {
                System.err.println(ioobe);
            }

10          System.out.println("Fin du programme");
        }

11      public static String getString()
            throws IOException
        {
            byte bArray[] = new byte[80];
            System.in.read(bArray);
            String  strResult = new String(bArray);
            return strResult.substring(0, strResult.indexOf("\r\n"));
        }

12      public static int getInteger()
            throws IOException, NumberFormatException
        {
            String strKB = getString();
            Integer intResult = new Integer(strKB);
            return intResult.intValue();
        }


13      public static long getLong()
            throws IOException, NumberFormatException
        {
            String strKB = getString();
            Long lResult = new Long(strKB);
            return lResult.longValue();
        }


14      public static double getDouble()
            throws IOException, NumberFormatException
        {
```

```
                String strKB = getString();
                Double dResult = new Double(strKB);
                return dResult.doubleValue();
        }

15      public static String fieldOf(String strLine, int nField)
                throws IndexOutOfBoundsException
        {
                return fieldOf(strLine, nField, ',');
        }


16      public static String fieldOf(String strLine, int nField, int sep)
                throws IndexOutOfBoundsException
        {
17              try
                {
18                      if (nField < 1) throw new IndexOutOfBoundsException();
                        int nDebut = 0;
                        String strResult = "";
                        for (int i = 0; i < nField; i++)
                        {
                                int nFin = strLine.indexOf(sep, nDebut);
                                if (nFin == -1)
                                {
19                                      strResult = strLine.substring(nDebut);
                                        nDebut = strLine.length() + 1;
                                }
                                else
                                {
                                        strResult = strLine.substring(nDebut, nFin);
                                        nDebut = nFin + 1;
                                }
                        }
                        return strResult;
                }
20              catch (IndexOutOfBoundsException ioobe)
                {
21                      String msg = "No de champ inconnu dans la chaine.";
22                      IndexOutOfBoundsException e =
                                        new IndexOutOfBoundsException(msg);
                        // Ici on a le choix, soit déclencher une exception,
                        // soit renvoyer une chaîne vide.
23                      throw e;
                        // return "";
                }
        }
}
```

COMMENTAIRES

1 La plupart des classes utilisées dans notre programme sont définies dans le *package* **java.lang**. Sauf la classe d'exception **IOException** qui est définie dans la *package* **java.io**. Il faut donc importer ce *package*.

2 Comme la fonction main assume la gestion des exceptions (par **try** et **catch**), il n'est plus nécessaire de transmettre les exceptions au programme appelant.

3 Les instructions susceptibles de générer des exceptions sont surveillées dans un bloc contrôlé par l'instruction **try**.

4 La fonction **getString** peut générer des exceptions de type **IOException**

5 La fonction **getInteger** peut générer des exceptions de type **IOException** et **NumberFormatException**.

6 La fonction **fieldOf** peut générer des exceptions de type **IndexOutOfBoundException** lorsque l'on tente d'extraire un champ qui n'existe pas.

7 Les exceptions de type **IOException**, n'arrivent que lorsqu'il y a un problème de lecture dans un flux (fichier inexistant, fin de fichier atteinte,.etc). Cela n'arrive pratiquement jamais lorsqu'un flux est associé au clavier de la console.

8 Les exceptions de type **NumberFormatException** arrivent pendant l conversion de chaîne de caractères en nombre, si la chaîne comprend des caractères non numériques.

9 Les exceptions **IndexOutOfBoundException** de type arrive lorsque l'on passe un paramètre ayant le rôle d'un index et que cet index est en dehors des limites permises. Dans notre cas, cette exception est générée lorsque l'on passe à la fonction **fieldOf** un N° de champ erroné.

10 Cette instruction n'existe que pour montrer que dans tous les cas, exception ou non, ce qui est en dehors des blocs contrôlés par **try** ou **catch** st exécuté.

11 **getString** utilise la méthode **read** de la classe **InputStream** qui peut générer une exception de classe **IOException** et la méthode **substring** de la classe **String** qui peut générer une exception de classe **IndexOutOfBoundException**. Cette dernière ne peut jamais arriver. En effet, une lecture au clavier se termine toujours par un retour de chariot. De ce fait, la fonction méthode **indexOf** renverra toujours un index valide pour la méthode **substring**. Il n'y a pas de traitement particulier à effectuer à ce niveau lorsque ces exceptions arrivent. On les transmet donc au programme appelant par **throws**.

12 **getInteger** utilise la fonction **getString** qui peut générer une exception de classe **IOException** et la méthode de construction d'un **Integer** à partir d'une **String** qui peut générer une exception de classe **NumberFormatException**. Il n'y a pas de traitement particulier à effectuer à ce niveau lorsque ces exceptions arrivent. On les transmet donc au programme appelant par **throws**.

13 mêmes remarques que précédemment pour la fonction **getLong.**

14 mêmes remarques que précédemment pour la fonction **getDouble**.

15 Cette première version de la fonction **fieldOf** fait un appel à la deuxième version qui elle génère une exception de classe **IndexOutOfBoundException** lorqu'on lui passe, en paramètre, un N° de champ erroné. Il n'y a pas de traitement particulier à effectuer à ce niveau lorsque ces exceptions arrivent. On les transmet donc au programme appelant par **throws**.

16 **fieldOf** génère des exceptions de type pour signaler au programme appelant que le paramètre **nField**, correspondant au N° de Champ, ne se trouve pas dans les limites permises. Cela se produit dans deux cas : lorsque **nField** est inférieur à 1 et lorsque **nField** est supérieur au nombre de champs qui

composent la chaîne **strLine**. Ces exceptions sont transmises au programme appelant par **throws**.

17 Toutes les instructions correspondant au traitement de la fonction **fieldOf** du T.P. précédent se trouvent maintenant surveillées par **try**.

18 Lorsque le N° de champ est inférieur à 1, on provoque une exception de classe **IndexOutOfBoundException** par le rupteur **throw**. Comme ce rupteur est utilisé dans un bloc contrôlé par **try**, cela provoque une branchement inconditionnel au bloc **catch** traitant cette classe d'exception (en 20).

19 C'est cet appel de la méthode **substring** qui va provoquer l'exception **IndexOutOfBoundException** correspondant à un N° de champ supérieur au nombre de champ composant la chaîne **strline**. En effet, **nDebut** est alors supérieur à la longueur de **strLine**, donc en dehors des limites permises par **substring**.

20 Les deux cas d'exceptions signalés au points 16 provoquent un branchement inconditionnel à cet endroit. Le traitement de ces exceptions permet ici d'en générer une nouvelle avec un message d'erreur personnalisé pour la fonction **fieldOf**.

21 Composition d'un message d'erreur personnalisé pour la fonction **fieldOf**.

22 Création d'une nouvelle exception de classe **IndexOutOfBoundException** que l'on construit avec un message d'erreur personnalisé pour la fonction **fieldOf**.

23 Cette exception crée est alors transmise pour signaler au programme appelant que le N° de champ est en dehors des limites permises. On aurait pu se contenter de renvoyer comme résultat une chaîne vide. Ce qui aurait abouti à un fonctionnement identique au T.P. précédent. Le fait, ici, d'utiliser le rupteur **throw** plutôt que **return** permet de distinguer entre un champ vide (deux virgules accolées) et un N° de champ erroné. **throw** provoque un branchement inconditionnel au bloc **catch** correspondant à la capture de cette exception. Dans notre cas, le branchement se produit en 9.

*Ce programme se trouve dans le fichier **JTP5.java**.*