# Entrées sorties formatées en Java

## 1 OBJECTIF

La programmation d'entrées sorties implique souvent la conversion de données en un format compréhensible par l'homme.

Java fournit la classe **Scanner** (du paquetage **java.util** ) pour les entrées. Elle décompose les entrées en éléments simples (String ou type simple tel que entier, réel, char...) permettant la lecture de données de différents types. Le délimiteur permettant de différentier les données est le caractère espace (ou tabulation ou le passage à la ligne suivante.

La sortie peut également être formatée pour une présentation agréable des données. La méthode **format** permet de présenter les données à l'utilisateur à la manière de la fonction **printf** du langage C.

## 2 LES ENTRÉES

### 2.1 Décomposer les entrées

Cette classe du paquet **java.util** permet d'effectuer des saisies de données de manière très flexible. Il suffit d'en créer une instance directement à partir d'un flux d'entrée, d'une socket...

Par exemple supposons que nous fournissions au clavier les données suivantes :

```
Guillaume Tell 36 1,85
```

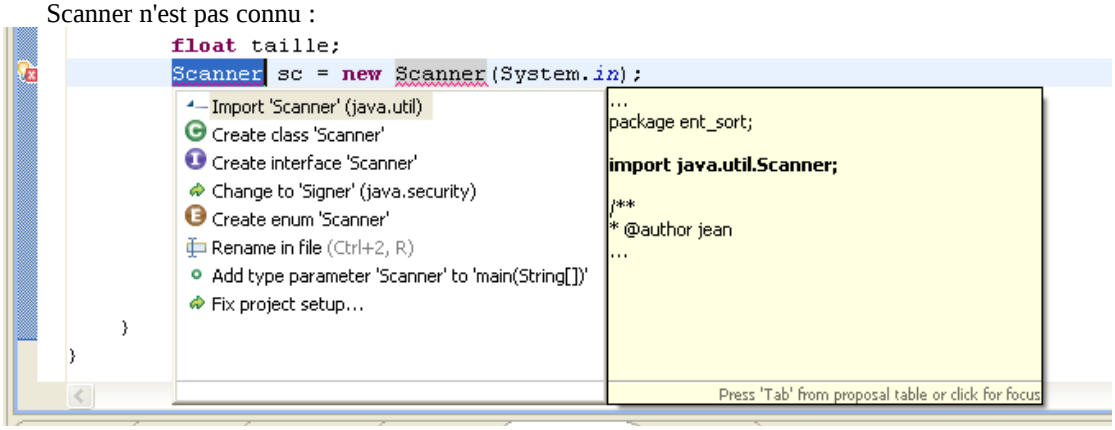
Pour saisir l'identité de Guillaume Tell, son âge et sa taille

```
package ent_sort;

/**
 * @author jean
 * programme de compréhension de la classe Scanner
 */
public class TestES {
   /**
    * @param args argument de la ligne de commande
    */
   public static void main(String[] args) {
          String nom;
          String prenom;
          int age;
          float taille;
          Scanner sc = new Scanner(System.in);
          prenom = sc.next();
          nom = sc.next();
          age = sc.nextInt();
          taille = sc.nextFloat();
          System.out.println(prenom);
          System.out.println(nom);
          System.out.println(age);
          System.out.println(taille);
          sc.close();
   }

}
```

Lors de l'écriture de la ligne de déclaration de l'instance du Scanner, l'aide nous indique que Scanner n'est pas connu :




il faut donc importer le paquetage **java.util.scanner** tout simplement en double cliquant dessus.

A noter que NetBeans vous fournit des aides pour résoudre les erreurs (symboles Croix-Rouge dans la marge. Il faut choisir avec réflexions une des propositions.

L'en tête du fichier est maintenant le suivant :

```
package ent_sort;
import java.util.Scanner;
```

A l'exécution, nous obtenons :

tout d'abord les entrées clavier

```
guillaume tell 36 1,87
```

puis les sorties écran :

```
guillaume
tell
36
1.87
```

Nous avons donc différentes méthodes pour décomposer les entrées, elles sont bâties sur **next** :

| Méthode de la classe Scanner | description |
|---|---|
| next() | un objet de type **String** . La saisie est terminée sur le caractère délimiteur habituel (espace, tabulation ou fin de ligne) |
| nextInt() | un **entier**. |
| nextFloat() | un réel simple précision |
| nextDouble() | un réel double précision |
| nextXxx() | un élément de type xxx (parmi les types simples définis pour java **sauf pour le type char**) |
| nextLine() | un objet de type String qui contient toue la ligne saisie donc jusqu'au caractère de fin de ligne |

Remarque : la saisie se fait selon les paramètres **régionaux** par contre la sortie n'est pas formatée convenablement.

## 2.2 Contrôler les saisies

Voici un autre code un peu plus complet :définit dans le **main() :**

```
Scanner sc = new Scanner(System.in);
String chaine;
while (sc.hasNext()) {
        chaine = sc.next();
        System.out.println(chaine);
}
```

voici les entrées

```
bonjour monsieur du corbeau
```

et la sortie correspondante :

```
bonjour
monsieur
du
corbeau
```

Ici, nous contrôlons qu'il y a des données présentes avec la méthode **hasNext()**. Elle est de type booléen, renvoie vrai si des données sont présentes faux dans le cas contraire.

Il existe une méthode hasNextXxx() par type de données simple tout comme les méthodes nextXxxx pour la saisie des types de données simple. Par ex hasNextShort() va vérifier que la donnée suivante est de type short (entier sur 16 bits)

La fin des saisies sur les entrées se fait à l'aide du caractère de fin de fichier (CTRL+Z sous WINDOWS ou CTRL+D sous LINUX, pour les types texte)

**Un objet de type Scanner est bloquant sur les entrées.**

Voici un exemple extrait du tutoriel de Sun :

```
public static void main(String[] args) {
   Scanner sc = null;
   double sum = 0;
   sc = new Scanner(System.in);
   sc.useLocale(Locale.FRANCE);
   while (sc.hasNext())
   {//il y a encore des données à lire
          if (sc.hasNextDouble())
          {// la donnée présente est de type double
                 sum += sc.nextDouble();
          } else {// en fait non, on la lit sans la stocker
                 sc.next();
          }
   }
   sc.close();
   System.out.println(sum);
}
```

avec les entrées suivantes :

```
8,5 32,767 3,14159
```

fournira le résultat suivant :

```
44.408590000000004
```

Dans cet exemple

```
sc.useLocale(Locale.FRANCE);
```

Indique que pour l'objet sc de type Scanner, les saisies se feront selon le format en vigueur en France (donc pour les réels, utilisation de la virgule). Par défaut c'est ce qui est réalisé. Il n'était donc pas nécessaire de le rappeler.

Que réalise ce programme ?

# 3 LES SORTIES FORMATÉES

Nous avons pu voir sur les exemples précédents que la sortie n'était pas des plus agréables :

- format de type américain
- pas de possibilité de définir la taille des réels lors de l'affichage, etc .

## 3.1 Les méthodes print et println

Ces méthodes du flux de sortie affichent tous les types simples et les objets de type String sur la sortie standard. De plus, println effectue un saut de ligne avec positionnement du curseur en début de ligne.

Par exemple, le code suivant :

```
public class Root {
   public static void main(String[] args) {
      int i = 2;
      double r = Math.sqrt(i);
      System.out.print("La racine carre de  ");
      System.out.print(i);
      System.out.print(" est ");
      System.out.print(r);
      System.out.println(".");
      i = 5;
      r = Math.sqrt(i);
      System.out.println("La racine carre de  " + i + " est " + r + ".");
   }
}
```

fournira en sortie :

```
La racine carre de  2 est 1.4142135623730951.
La racine carre de  5 est 2.23606797749979.
```

Les résultats bien entendu sont corrects toutefois, on attend un peu mieux de l'affichage.

## 3.2 Méthode format

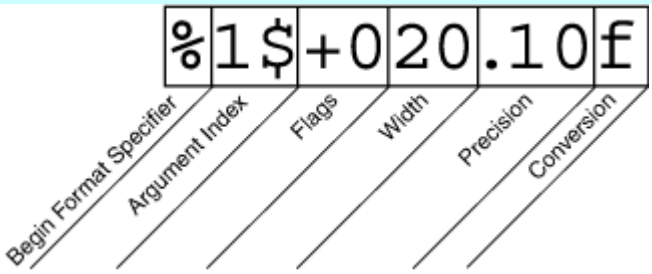
```
int i = 2;
double r = Math.sqrt(i);
System.out.println(r);
System.out.format("La racine carre de   %d est %2.4f\n", i, r);
```

et le résultat correspondant :

```
1.4142135623730951
La racine carre de   2 est 1,4142
```

Comparez bien les deux résultats :

le premier utilise le format américain, ne se préoccupe donc pas du paramètre régional tandis que la méthode **format** utilise ce paramètre. Cette méthode fonctionne comme la fonction **printf** du langage C




| Drapeau | Rôle | Exemple |
|---|---|---|
| + | Affiche le signe des nombres positifs et négatifs | +3333.33 |
| espace | Ajoute un espace avant les nombres positifs | \| 3333.33\| |
| 0 | Ajoute des zéros préalables | 003333.33 |
| - | Justifie le champ à gauche | \|3333.33\| |
| ( | Entoure le nombre négatif de parenthèses | (3333.33) |
| , | Ajoute des séparateurs de groupe | 3,333.33 |
| #(pour format f) | Inclut toujours une décimale | 3,333 |
| # (pour formatx ou o) | Ajoute le préfixe 0x ou 0 | 0xcafe |
| ^ | Transforme en majuscules | 0xCAFE |
| $ | Indique l'indice de l'argument à mettre en forme ; par exemple, %1$d %1$x affiche le premier argument en décimal et hexadécimal | 159 9F |
| < | Met en forme la même valeur que la spécification précédente ; par exemple, %d %<x affiche le même nombre en décimal et en hexadécimal | 159 9F |