Classes Utilitaires

Cours 6

Nombres en Java

- Problématique :
  - utiliser un nombre (**type primitif**) comme paramètre d'une méthode n'acceptant que des objets
    - Programmation répartie (RMI, Corba, Web Services)
  - utiliser un nombre dans un container (ex: List) n'est possible qu'avec des objets
  - bénéficier de méthodes manipulant un nombre (formatage, conversion, ...)

Inboxing/Outboxing

- Solution : tous les types primitifs ont un correspondant en objet :

int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
byte	Byte
short	Short
char	Character

Classe d'un type primitif

- Chaque classe permet de :
  - d'encapsuler une valeur de type primitif dans un objet
    - Ex : Integer objetI = new Integer(12);
  - d'extraire la valeur d'un objet qui l'encapsule
    - Ex : int i = objetI.intValue();
- et offre des utilitaires autour des types primitifs. Exemples :
  - int i = Integer.parseInt("15");
  - char c = Character.toUpperCase('a');

Automatisation Inboxing/Outboxing

- La conversion entre type primitif et objet de la classe correspondante est automatique :

```
Integer objetEntierI;
Object objetI;
int i;
i = 5;
objetEntierI = i;  // new Integer(i) implicite
i = objetEntierI;  // objetEntier.intValue() implicite
objetI = i;          // new Integer(i) implicite
i = (Integer) objetI;  // ((Integer) objetEntier).intValue() implicite
```

Formatage des nombres

- Par des fonctions d'impression plus évoluées : **`printf(...)`** et **`format(...)`** sur les Objets Numériques.
- Par un objet qui décrit comment formater un nombre : **`DecimalFormat`**.
- Utiliser un objet de formatage permet de :
  - mémoriser un format
  - éviter de recalculer le format à chaque impression

Fonctions mathématiques

- Les fonctions mathématique sont réunies dans une classe : Math
- Toutes les méthodes sont statiques (méthodes de classe)
- => **on n'instancie jamais la classe Math**

Caractères

- Le Java utilise couramment le type primitif **`char`**
- Le caractère Java est de l'Unicode => 2 octets par caractère (UTF16)
- Le source Java accepte les caractères étendus, contrairement au autres langages (le source Java est traité comme de l'Unicode).

`char` (Java) ≠ **`char`** (C)

2 octets ... 1 octet

La classe Character

- Le type **`char`** est doublé par un type objet **`Character`**
- La classe **`Character`** dispose de méthodes utiles d'identification et de conversion (plus de 50).
  - Exemples :
    - isLowerCase()
    - isLetter()
    - ...

Chaînes de caractères

- En Java, une chaîne est un objet (`String`).
- Une chaîne littérale, ex. "Hello world" est un appel implicite au constructeur de String
- On peut construire une chaîne de diverses manières :

```
char[] charArray = {'a', 'z', 'e', 'r'};
String s = new String(charArray);
```

Chaînes de caractères (2)

- Longueur de chaîne : length() appliqué sur une String
- Concaténation :
  - Utiliser une méthode :

```
String ch3 = ch1.concat(ch2)
```

  - Utiliser un opérateur :

```
String ch3 = ch1 + ch2;
```

Conversions implicites

- Tout objet disposant une méthode `toString()` peut être converti implicitement.
- L'expression texte d'une variable est utilisée suivant le contexte :

```
Integer i = new Integer(i);
String s =  "" + i;
```

Mémorisation de formatage

- Appliquer la méthode **`format()`** directement sur la classe **`String`** permet de mémoriser dans une variable le résultat d'un formatage.

```
String strVar =  "le résultat est ";
float flVar =  34.21e-12f;
String fs;
fs = String.format("%s %f",strVar, flVar);
System.out.println(fs);
```

Conversions texte

- La conversion texte convertit une valeur en texte et vice-versa :
  - valeur -> texte -> formatage
  - texte -> valeur -> parsing
- Le parsing est la technique qui analyse une chaîne de caractère pour y reconnaître un nombre :

`parse`*Type*(**`String`**)

Manipulations de chaîne

- Opérations de caractère :
  - pointer un caractère
  - chercher un caractère
- Opérations de sous-chaîne :
  - extraction de sous-chaîne
  - recherche de sous-chaîne
  - insertion/remplacement de sous-chaîne
- Opérations plus complexes :
  - Tokenisation (vu plus tard)

Non mutabilité de la chaîne

- Un objet String est **non mutable :** on ne peut pas le modifier.

=> un remplacement dans une chaîne produit un nouvel objet String.

=> deux objets chaîne sont égaux (equals()) si leurs séquences de caractères sont égales

```
str1.equals(str2)
str1.intern() == str2.intern()
str1 == str2 // objets égaux
```

Comparaisons de chaînes

- Méthodes de comparaisons
  - Egalité totale (equals)
  - Correspondance de préfixe (startsWith)
  - Correspondance de suffixe (endsWith)
  - Correspondance de région (regionMatches)
  - Versions casse « aware » et casse « unaware »

StringBuilder

- Effectuer une manipulation complexe d'une chaîne :
  - consomme beaucoup de mémoire
  - consomme beaucoup de mouvements de mémoire
- => la classe StringBuilder permet de construire une chaîne mutable (buffer).
- Un StringBuffer a une longueur (= String) et une capacité "réservée".

Documentation Java

- Commentaires :
  - Bloc :

    ```
    /*
                1 commentaire sur plusieurs lignes
    */
    ```

  - Ligne :

    ```
    // 1 commentaire de fin de lignes
    ```

- Documentation automatique HTML avec Javadoc (API JDK like)

  ```
  /**
   *              Commentaire Javadoc
   *                       ...
   */
  ```

Exemple Commentaire

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. The name
 * argument is a specifier that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param url an absolute URL giving the base location of the image
 * @param name the location of the image, relative to the url argument
 * @return the image at the specified URL
 * @see Image
 */
public Image getImage(URL url, String name) { … }
```

Rendu de Javadoc

getImage

```
public Image getImage(URL url,
                      String name)
```

Returns an `Image` object that can then be painted on the screen. The `url` argument must specify an absolute URL. The `name` argument is a specifier that is relative to the `url` argument.

This method always returns immediately, whether or not the image exists. When this applet attempts to draw the image on the screen, the data will be loaded. The graphics primitives that draw the image will incrementally paint on the screen.

Parameters:

`url` - an absolute URL giving the base location of the image

`name` - the location of the image, relative to the `url` argument

Returns:

the image at the specified URL

See Also:

`Image`

Balises (tags) Javadoc

- @param : paramètre de constructeur ou méthode
- @return : valeur de retour d'une méthode
- @throws / @ exception : exceptions
- @author : auteur du code
- @version : version du code
- @see : référence vers 1 autre classe/champs/constructeur/méthode/package
- {@link} : idem mais inclus dans le texte
- @since : valable depuis telle version de Java
- @deprecated : code maintenu mais à ne plus utiliser