

# Programmation Orientée Objet

## Les chaînes de caractères

Frédéric Mallet

<http://deptinfo.unice.fr/~fmallet/>

# Objectifs

## ❑ Les chaînes de caractères

- Classe String
- Relations avec les tableaux de caractères
- Conversions vers les types numériques primitifs
- Classes StringBuilder et StringBuffer

## ❑ Aperçu sur les expressions régulières

# Les chaînes du caractères

## ❑ Trois classes du paquetage java.lang

- **String** – Chaînes de caractères immutables
  - Ex:

```
String chaine1 = "Bonjour";
String chaine2 = "Bonjour";
String chaine3 = new String("Bonjour");
```
  - Immutables:  
`chaine1.toUpperCase()` créé une **nouvelle** chaîne
- **StringBuilder** ou **StringBuffer** pour les chaînes variables
  - **StringBuffer** – classe historique (*Thread-safe*)
  - **StringBuilder** – seulement depuis JDK1.5 (pas *Thread-safe*)

# Concaténation : +

- ❑ L'opérateur + est surchargé
  - `"Bonjour " + "tout le monde!"`
- ❑ Les valeurs (de type primitif) sont traduites par le compilateur
  - `int x = 5;`
  - `String s = "Valeur de x = " + x;`
- ❑ Les références
  - non-nulles sont traduites par l'appel de la méthode `toString()`
  - nulles => "null"

# Egalité de chaînes

## □ Ce sont des références

- Attention au ==
- Utilisez plutôt la méthode `equals(Object)` redéfinie depuis la classe `Object`
  - `String s1 = "Bonjour ";`
  - `String s2 = "tout le monde";`
  - `(s1+s2).equals("Bonjour tout le monde");`
- Utiliser aussi `equalsIgnoreCase()` pour la casse !

# Comparaison de chaînes

## ❑ String réalise l'interface Comparable<String>

- Méthode `compareTo(String s) : int`
  - Implante l'ordre lexicographique sur les chaînes de caractères
    - "antoine".compareTo("antonin") <0
    - Mais "antoine".compareTo("Antonin") >0
- Méthode `compareToIgnoreCase(String s): boolean`
  - Ordre lexicographique qui ignore la casse
    - "antoine".compareToIgnoreCase("Antoine") ==0
    - "antoine".compareToIgnoreCase ("Antonin") <0

# Accès aux caractères

□ On peut lire un caractère à une position donnée

- Le premier caractère est à la position 0
- Le dernier caractère est à la position `s.length()-1`
- Méthode `char charAt(int i)`
  - `i`ème `char` (pas nécessairement `i`ème caractère – `codePointAt`)
    - "bonjour".length() => 7
    - "bonjour".charAt(0) => 'b'
    - "bonjour".charAt(6) => 'r'
- Depuis Java 5: méthode `int codePointAt(int i)`
  - Char est sur 16 bits (ne permet pas l'unicode 32 bits)
  - Si le caractère est plus long que 16 bits (*surrogate*) alors `i` doit référencer le premier `char` du couple.

# Recherche

## ❑ Chercher dans une chaîne

- `int indexOf(char c)`
  - Cherche la première position du caractère c
  - Ex: "bonjour".indexOf('o') => 1, "bonjour".indexOf('O') => -1
- `int indexOf(char c, int pos)`
  - Cherche la première position du caractère c à partir de pos.
- `int lastIndexOf(char c), int lastIndexOf(char c, int pos)`
  - Dernière position du caractère c
  - Ex: "bonjour".lastIndexOf('o') => 4

# Sous-chaînes

## □ Deux méthodes

- `substring(int début, int fin)` et `substring(int début)`
- sous-chaîne entre la position `début` inclue et la position `fin` exclue
  - "bonjour".`substring(3, 7)`; renvoie la chaîne "jour"
  - "bonjour".`substring(3)`; renvoie aussi la chaîne "jour"

# Comparer des sous-chaînes

## □ Deux méthodes regionMatches

- `regionMatches(int d1, String chaine, int d2, int l);`
  - Compare la chaîne `this` à partir de la position `d1` avec la chaîne `chaine` à partir de la position `d2` sur une longueur `l`.
- `regionMatches(boolean c, int d1, String chaine, int d2, int l);`
  - Variante : si `c` vaut `true`, on ne tient pas compte de la casse !
- Exemples:
  - `"bonjour".regionMatches(3, "jour", 0, 4)?`
  - `"bonjour".regionMatches(1, "jambon", 4, 2)?`
  - `"BONJOUR".regionMatches(true,1, "jambon", 4, 2)?`

# Inclusion de chaînes

## ❑ Premier emplacement d'une sous-chaîne

- int indexOf(String sousChaine)
- int indexOf(String sousChaine, int position)

## ❑ Dernier emplacement d'une sous-chaîne

- int lastIndexOf(String sousChaine)
- int lastIndexOf(String sousChaine, int position)

## ❑ Début et fin

- boolean endsWith(String sousChaine)
- boolean startsWith(String sousChaine)

## ❑ Enlève les espaces en début et en fin

- String trim()

# Majuscules et minuscules

## □ De minuscules vers majuscules: `toUpperCase()`

- `String s1 = "Bonjour";`
- `String s2 = s1.toUpperCase();`
- `s2` devient "BONJOUR", `s1` n'est pas modifiée

## □ De majuscules vers minuscules: `toLowerCase()`

- `String s1 = "Bonjour";`
- `String s2 = s1.toLowerCase();`
- `s2` devient "bonjour", `s1` n'est pas modifiée

# Découper une chaîne

□ On peut découper une chaîne en fonction d'un caractère de séparation avec la méthode `split`

- `String[] split(String pattern)`
  - Ex:
    - "boo:and:foo".split(":") => { "boo", "and", "foo"}
    - "boo:and:foo".split("o") => { "b", "", ":and:f"}
    - "ceci est un test".split("\s") => { "ceci", "est", "un", "test"};

□ Depuis JDK 1.4

- A préférer à `java.util.StringTokenizer`
- Expressions régulières pas étudiées dans ce cours

# Chaînes et tableaux

- ❑ Il est souvent commode de travailler sur les tableaux de caractères

- Exemple: Remplacer le  $i^{\text{ème}}$  caractère d'une chaîne par une majuscule

```
String enMaj(String s, int i) {  
    String debut = s.substring(0, i);  
    char c = s.charAt(i);  
    String fin = s.substring(i+1);  
    return debut + Character.toUpperCase(c) + fin;  
}  
  
String enMaj(String s, int i) {  
    char[] buf = s.toCharArray();  
    buf[i] = Character.toUpperCase(buf[i]);  
    return new String(buf);  
}
```

# Conversion vers les types primitifs

## ❑ Chaque type primitif à sa classe englobante

- int (Integer), byte (Byte), short (Short), long (Long)
- double (Double), float (Float)

## ❑ La classe englobante sait transformer une chaîne

- `int Integer.parseInt(String s)`
  - Convertit la chaîne s en un entier (si s représente un entier)
    - Sinon NumberFormatException
    - `Integer.parseInt("100") => 100`
- `int Integer.parseInt(String s, int radix)`
  - Convertit la chaîne s en un entier dans la base radix
  - `Integer.parseInt("100", 2) => 4`

# NumberFormatException

- On peut attraper les exceptions pour détecter une erreur

```
class Somme {  
    static public void main(String[] args) {  
        int somme = 0;  
        for(String arg : args)  
            somme += Integer.parseInt(arg);  
        System.out.println(somme);  
    }  
}
```

- Test :

- java Somme 12 25 14 => 51
- java Somme 12 pasUnNombre 14 =>  
**NumberFormatException !**

# NumberFormatException: try/catch

- On peut attraper les exceptions pour détecter une erreur

```
class Somme {  
    static public void main(String[] args) {  
        int somme = 0;  
        for(String arg : args){  
            try {  
                somme += Integer.parseInt(arg);  
            } catch (NumberFormatException nfe) {  
                System.err.println(arg+" n'est pas un nombre!");  
            }  
        }  
        System.out.println(somme);  
    }  
}
```

- Test :

- java Somme 12 pasUnNombre 14 => 26
- **pasUnNombre n'est pas un nombre!**

# Des types primitifs vers les chaînes

❑ Il suffit de faire une concaténation

- int i = 5;
- String s = i + "";

❑ On peut aussi utiliser la classe NumberFormat du paquetage java.text

- double d = 3456.78;
- NumberFormat.getInstance().format(d);
  - 3 456,78 (Java configuré en français !)
- NumberFormat.getInstance(Locale.ENGLISH).format(d);
  - 3,456.78
- NumberFormat.getCurrencyInstance().format(d);
  - 3 456,78 €

# Objectifs

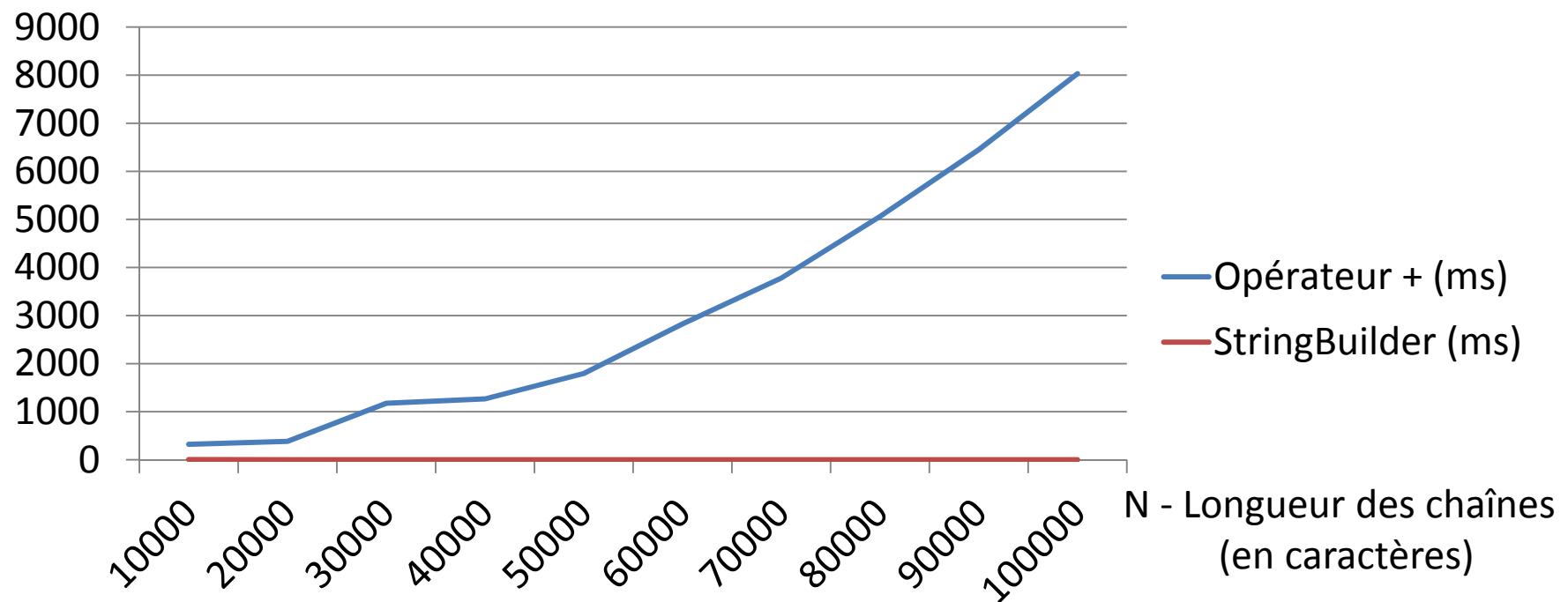
## □ Les chaînes de caractères

- Classe String
- Relations avec les tableaux de caractères
- **Classes StringBuilder et StringBuffer**

# Problème de performances

## □ String est une classe immutable

- Ses objets ne sont pas modifiables
- Chaque opération implique la création d'un nouvel objet (coûteux en temps et en mémoire)
- Création d'une chaîne de longueur n avec +



# StringBuilder et StringBuffer

## ❑ Opérations

- Concaténation
  - `append(X)`, `append(char[])`
- Insertion
  - `insert(int index, X caractères)`
- Remplacement
  - `replace(int début, int fin, String chaîne)`
- Effacement
  - `delete(int début, int fin)`, `deleteCharAt(int index)`
- Vers les chaînes
  - `toString() : String`, `substring(int début, int fin)`

## ❑ Ne redéfinissent pas la méthode `equals(Object)` !

# StringBuffer vs. StringBuilder

❑ Mêmes fonctionnalités et même noms de méthodes

❑ StringBuffer

- Existe depuis le début du langage
- *Thread-safe* (peut être utilisée sans risque quand il y a plusieurs threads – notamment graphique)
  - Les appels sont synchronisés (`synchronized`)

❑ StringBuilder

- Introduite avec Java 5.0
- Pas *Thread-safe*
- Performances légèrement meilleures

# Interface CharSequence

- ❑ Suite de char lisible (JDK 1.4)
- ❑ String, StringBuilder et StringBuffer réalisent cette interface
- ❑ Interface utile pour les expressions régulières notamment

```
interface CharSequence {  
    char charAt(int index);  
    int length();  
    CharSequence subSequence(int start, int end);  
    String toString();  
}
```

# Objectifs

## ❑ Les chaînes de caractères

- Classe String
- Relations avec les tableaux de caractères
- Classes StringBuilder et StringBuffer

## ❑ Aperçu sur les expressions régulières

# Expressions Régulières: le fond

□ Recherche d'une séquence dans une chaîne

□ Opération coûteuse => 2 phases

- Compilation d'un automate reconnaisseur

```
static Pattern java.lang.regex.Pattern.compile(String)
```

- Le paramètre est l'expression régulière à reconnaître

- Reconnaissance d'une ou plusieurs chaînes

```
Matcher Pattern.matcher(String)
```

- Le paramètre est la chaîne à comparer

□ Exemple

- ".\*\\".java"      chaîne qui termine par ".java"

# Les caractères spéciaux

- ❑ Une expression régulière est une suite de caractères (type String)
- ❑ Un certain nombre de caractères spéciaux
  - x Le caractère 'x'
  - \\  
Le caractère *backslash*
  - \n Le caractère *newline* (saut de ligne) ('\u000A')
  - \r Le caractère *retour à la ligne* ('\u000D')
  - \e Le caractère *d'échappement* ('\u001B')
  - \cx Ctrl-x

# Expressions régulières: la forme

- ❑ Une expression régulière est une suite de caractères (type String)
  - ❑ Un certain nombre de caractères spéciaux
    - \0n Le caractère de valeur octale 0n ( $0 \leq n \leq 7$ )
    - \0nn Le caractère de valeur octale 0nn ( $0 \leq n \leq 7$ )
    - \0mnn Le caractère de valeur octale 0mnn ( $0 \leq m \leq 3$ ,  
 $0 \leq n \leq 7$ )
    - \xhh Le caractère de valeur hexadécimale 0xhh
    - \uhhhh Le caractère de valeur hexadécimale 0uhhhh
    - \x{h...h} Le caractère de valeur hexadécimale 0xh...h  
(Character.MIN\_CODE\_POINT  $\leq 0xh...h \leq$   
Character.MAX\_CODE\_POINT)

# Les classes de caractères

- ❑ Certaines constructions permettent d'accepter plusieurs possibilités: classes de caractères
- ❑ Les crochets [] permettent de construire des classes
  - [abc] 'a', 'b', ou 'c' (classe simple)
  - [^abc] Un caractère sauf 'a', 'b', or 'c' (négation)
  - [a-zA-Z] de 'a' à 'z' ou de 'A' à 'Z', inclusif (domaine)
  - [a-d[m-p]] de 'a' à 'd', ou de 'm' à 'p': [a-dm-p] (union)
  - [a-z&&[def]] 'd', 'e', ou 'f' (intersection)
  - [a-z&&[^bc]] de 'a' à 'z', sauf 'b' et 'c': [ad-z] (soustraction)
  - [a-z&&[^m-p]] de 'a' à 'z', et pas de 'm' à 'p': [a-lq-z]

# Classes prédéfinies

## □ Des classes sont prédéfinies

- . N'importe quel caractère (y compris fin de ligne)
- \d Un chiffre: [0-9]
- \D Un caractère qui n'est pas un chiffre : [^0-9]
- \s Un caractère *blanc* : [ \t\n\x0B\f\r]
- \S Un caractère non *blanc* : [^\s]
- \w Un caractère d'un *mot* [a-zA-Z\_0-9]
- \W Autre qu'un caractère d'un mot : [^\w]

# Caractères frontières

## □ Les frontières des mots, lignes et données d'entrée

- `^` Le début d'une ligne
- `$` La fin d'une ligne
- `\b` La frontière d'un mot (début ou fin)
- `\B` Un caractère pas à la frontière
- `\A` Le début de l'entrée (différent de `^` si +s lignes)
- `\G` La fin de la correspondance précédente
- `\z` La fin de l'entrée

# Quantificateurs cupides

❑ Certains quantificateurs sont “cupides”

- $X?$        $X$ , une fois ou zéro
- $X^*$        $X$ , zéro ou plusieurs fois
- $X^+$        $X$ , une fois ou plusieurs
- $X\{n\}$      $X$ , exactement  $n$  fois
- $X\{n,\}$     $X$ , au moins  $n$  fois
- $X\{n,m\}$   $X$ , au moins  $n$  fois mais pas plus de  $m$  fois

# Quantificateurs réticents

## □ D'autres le sont moins

- $X??$       X, zéro ou 1 fois
- $X^*?$       X, zéro ou plus
- $X^+?$       X, une fois ou plusieurs fois
- $X\{n\}?$       X, exactement n fois
- $X\{n,\}?$       X, au moins n fois
- $X\{n,m\}?$       X, au moins n fois mais pas plus de m fois