PROGRAMMATION ORIENTEE OBJET

Enseignant: Ing. NGOUDJOU

ORGANISATION DU COURS

- VOLUME HORAIRE :
  - COURS MAGISTRAL : 24 H
  - TD : 32 H
  - TP : 60 H
- EVALUATION
  - Mini projet par groupe de 3 étudiants qui comptera pour le cc
  - Examen pratique
  - Examen final

RESSOURCES ET REFERENCES

- Programmer en Java, C. Delannoy, Ed. Eyrolles, 2014
- Thinking in Java, B. Eckel, Prentice Hall, 2006 Traduction française diponible en ligne

http://bruce-eckel.developpez.com/livres/java/traduction/tij2/

- Les tutoriels Oracle

https://docs.oracle.com/javase/tutorial/

- La documentation de l'API standard https://docs.oracle.com/javase/8/docs/api/

CHAPITRE 4: LES CHAINES EN JAVA

- **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

CHAPITRE 4: LES CHAINES EN JAVA

LES CHAINES DE CARACTERES

- 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()` crée une **nouvelle** chaîne

  - `StringBuilder` ou `StringBuffer` pour les chaînes variables
    - `StringBuffer` – classe historique (*Thread-safe*)
    - `StringBuilder` – seulement depuis JDK1.5 (pas *Thread-safe*)

CHAPITRE 4: LES CHAINES EN JAVA

LES CHAINES DE CARACTERES

CONCATANENATION : +

- 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"`

CHAPITRE 4: LES CHAINES EN JAVA

LES CHAINES DE CARACTERES

EGALITE DE CHAINES

- 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 ignorer la casse !

CHAPITRE 4: LES CHAINES EN JAVA

LES CHAINES DE CARACTERES

COMPARAISON DE CHAINES

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

CHAPITRE 4: LES CHAINES EN JAVA

LES CHAINES DE CARACTERES

ACCES AU CARACTERES

- 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`
    (attention aux parenthèses!)
  - 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.

CHAPITRE 4: LES CHAINES EN JAVA

LES CHAINES DE CARACTERES

RECHERCHE DANS UNE CHAINE

- 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`

CHAPITRE 4: LES CHAINES EN JAVA

LES CHAINES DE CARACTERES

LES SOUS-CHAINES

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

CHAPITRE 4: LES CHAINES EN JAVA

LES CHAINES DE CARACTERES

COMPARAISON DES SOUS-CHAINES

- 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)` ?

CHAPITRE 4: LES CHAINES EN JAVA

LES CHAINES DE CARACTERES

INCLUSION DE CHAINES

- 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()`

CHAPITRE 4: LES CHAINES EN JAVA

LES CHAINES DE CARACTERES

MAJUSCULE ET MINUSCULE

- 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

CHAPITRE 4: LES CHAINES EN JAVA

LES CHAINES DE CARACTERES

DECOUPER UNE CHAINE

- 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\nun  test".split("\\s") => { "ceci", "est", "un", "test"};`
- Depuis JDK 1.4
  - A préférer à `java.util.StringTokenizer`

CHAPITRE 4: LES CHAINES EN JAVA

LES CHAINES DE CARACTERES

CHAINES ET TABLEAUX

- Il est souvent commode de travailler sur les tableaux de caractères
  - Exemple: Remplacer le $i^{ème}$ 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);
}
```

CHAPITRE 4: LES CHAINES EN JAVA

LES CHAINES DE CARACTERES

CONVERSION VERS TYPE PRIMITIF

- Chaque type primitif à sa classe enveloppe
  - `int` (Integer), `byte` (Byte), `short` (Short), `long` (Long)
  - `double` (Double), `float` (Float)
  - `char` (Character), `boolean` (Boolean)
- La classe enveloppe sait analyser 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`

CHAPITRE 4: LES CHAINES EN JAVA

LES CHAINES DE CARACTERES

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 !

CHAPITRE 4: LES CHAINES EN JAVA

LES CHAINES DE CARACTERES

Des types primitifs vers les chaines

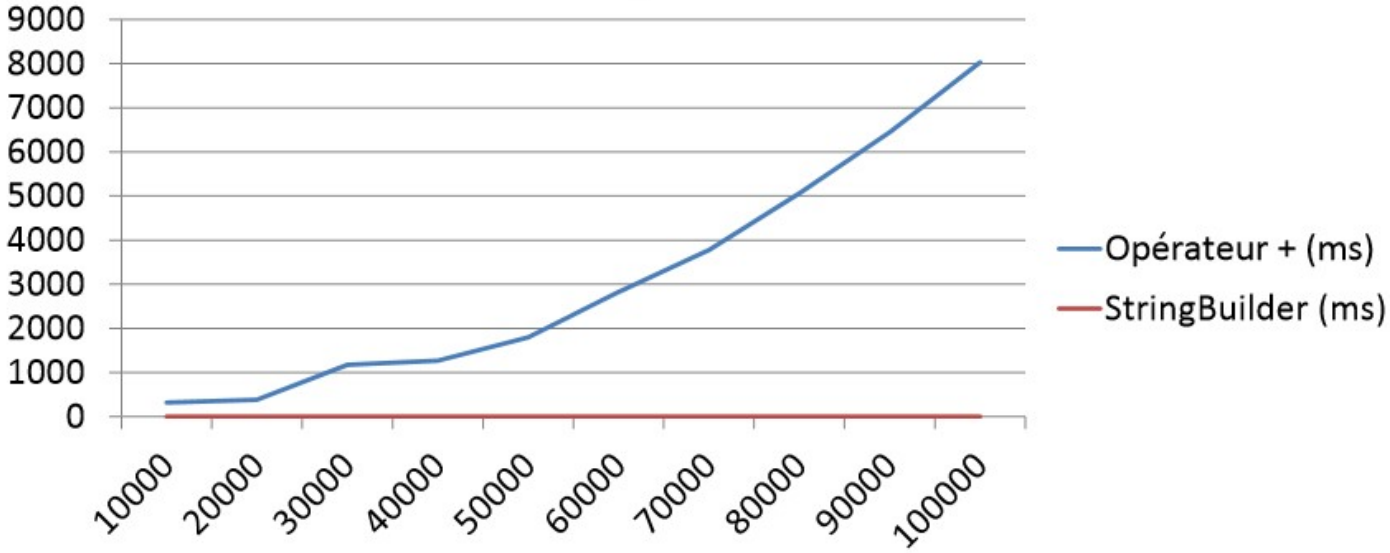
- Il suffit de faire une concaténation
  - `String s = 5 + "";`
- Ou d'utiliser la méthode statique `toString()` de la classe envelope
  - `String s = Integer.toString(5);`
- On peut aussi utiliser la classe `NumberFormat` du paquetage `java.text`
  - `double d = 3456.78;`
  - `NumberFormat.getInstance().format(d);`
    - `3 456,78` (Java configuré en français !)
  - `NumberFormat.getInstance(Locale.ENGLISH).format(d);`
    - `3,456.78` (on peut changer la localisation)
  - `NumberFormat.getCurrencyInstance().format(d);`
    - `3 456,78 €`

CHAPITRE 4: LES CHAINES EN JAVA

LES CHAINES DE CARACTERES

La classe String

- `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 +



CHAPITRE 4: LES CHAINES EN JAVA

LES CHAINES DE CARACTERES

Les classes StringBuffer et StringBuilder

- Opérations
  - Concaténation
    - `append(X caracteres), append(char[] tab)`
  - Insertion
    - `insert(int index, X caracteres)`
  - Remplacement
    - `replace(int debut, int fin, String chaine)`
  - Effacement
    - `delete(int debut, int fin), deleteCharAt(int index)`
  - Vers les chaînes
    - `toString(), substring(int debut, int fin)`
- Ne redéfinissent pas la méthode `equals(Object)` !

CHAPITRE 4: LES CHAINES EN JAVA

LES CHAINES DE CARACTERES

Les classes StringBuffer et 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 (en théorie)

CHAPITRE 4: LES CHAINES EN JAVA

LES CHAINES DE CARACTERES

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();
}
```

CHAPITRE 4: LES CHAINES EN JAVA

LES EXPRESSIONS REGULIERES

- Recherche d'un motif dans une chaîne
- Opération coûteuse => 2 phases
  - Compilation d'un automate reconnaissant le motif

    ```
    static Pattern java.lang.regexpr.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"`

CHAPITRE 4: LES CHAINES EN JAVA

LES EXPRESSIONS REGULIERES

- 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

CHAPITRE 4: LES CHAINES EN JAVA

LES EXPRESSIONS REGULIERES

- 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', ou 'c' (négation)
  - `[a-zA-Z]` de 'a' à 'z' ou de 'A' à 'Z', inclusif (domaine)
  - `[a-z&&[dEf]]` 'd' ou 'f' (intersection)
  - `[a-z&&[^bc]]` de 'a' à 'z', sauf 'b' et 'c' (soustraction)
  - `[a-z&&[^m-p]]` de 'a' à 'z', mais pas de 'm' à 'p': `[a-lq-z]`
- L'union s'obtient avec le caractère '|'
  - `[a-d]|[m-p]` de 'a' à 'd', ou de 'm' à 'p': `[a-dm-p]` (union)

CHAPITRE 4: LES CHAINES EN JAVA

LES EXPRESSIONS REGULIERES

- 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]`

CHAPITRE 4: LES CHAINES EN JAVA

LES EXPRESSIONS REGULIERES

- 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 plusieurs lignes)
  - `\G` La fin de la correspondance précédente
  - `\z` La fin de l'entrée

CHAPITRE 4: LES CHAINES EN JAVA

LES EXPRESSIONS REGULIERES

LES QUANTIFICATEURS

- Moyens d'itérer une expression régulière
  - X? X, zéro ou une fois
  - X* X, un nombre quelconque de fois (0, 1 ou plus)
  - X+ X, au moins une fois, équivalent à XX*
  - 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

```
Pattern.matches("do{2}", "dodo");    false
Pattern.matches("(do){2}", "dodo"); true
Pattern.matches("do{2}", "doo");     true
```

CHAPITRE 4: LES CHAINES EN JAVA

LES EXPRESSIONS REGULIERES

LES QUANTIFICATEURS

- Quantificateurs gloutons (par défaut), réticents (ajouter ?) et possessifs (ajouter +)

```
Pattern p = Pattern.compile(".*do");
Matcher m = p.matcher("si do la si do");
System.out.println(m.matches());
m.reset();
while (m.find())
  System.out.println("De " + m.start() + " à " + m.end());
```

```
true
De 0 à 14
```

CHAPITRE 4: LES CHAINES EN JAVA

LES EXPRESSIONS REGULIERES

LES QUANTIFICATEURS

- Quantificateurs gloutons (par défaut), réticents (ajouter ?) et possessifs (ajouter +)

```
Pattern p = Pattern.compile(".*?do");
Matcher m = p.matcher("si do la si do");
System.out.println(m.matches());
m.reset();
while (m.find())
  System.out.println("De " + m.start() + " à " + m.end());
```

```
true
De 0 à 5
De 5 à 14
```

CHAPITRE 4: LES CHAINES EN JAVA

LES EXPRESSIONS REGULIERES

LES QUANTIFICATEURS

- Quantificateurs gloutons (par défaut), réticents (ajouter ?) et possessifs (ajouter +)

```
Pattern p = Pattern.compile(".*+do");
Matcher m = p.matcher("si do la si do");
System.out.println(m.matches());
m.reset();
while (m.find())
  System.out.println("De " + m.start() + " à " + m.end());

false
```