Programmation par Objets - Printemps 2010
Frédéric Mallet & Jean-Paul Roy
L1 Info&Math - Faculté des Sciences de Nice

http://deptinfo.unice.fr/~roy

Les traitements du texte

Rappel : les chaînes de caractères (String)

• Les **chaînes** sont les objets de la classe `String` qui est importée automatiquement puisque dans le package `java.lang`

• Les **chaînes** sont constituées de **caractères** : lettres, chiffres, ponctuations. Nous connaissons déjà :

- la **concaténation** de deux chaînes avec l'opérateur +

- la **longueur** `length()` d'une chaîne (nombre de caractères) :

```
String str = ”Hello world !”;
System.out.println(”Longueur : ” + str.length());
```

```
Longueur : 13
```

- l'extraction `substring(i,j)` d'une **sous-chaîne** (indices dans `[i,j[`) :

```
String subStr = str.substring(6,11);
System.out.println(”Extraction : ” + subStr);
```

```
Extraction : world
```

Construction d'une chaîne, égalité de deux chaînes

• En règle générale, on n'utilise PAS le constructeur de la classe `String` pour initialiser une chaîne :

~~`String str = new String("Hello world !");`~~

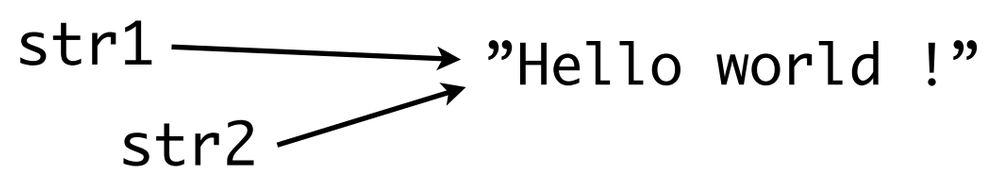
```
String str = "Hello world !";
```

• Ceci tient au fait que Java, pour optimiser la mémoire, s'efforce de **ne pas dupliquer** les chaînes qui comportent les mêmes caractères :

```
String str1 = "Hello world !";
String str2 = "Hello " + "world !";
boolean test = (str1 == str2);
System.out.println(test);
```

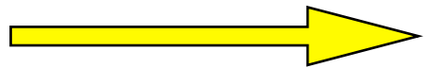
→ `true`

• D'après la signification de == sur les objets (**comparaison de références**, cours 6 page 17), `str1` et `str2` pointent sur le même objet en mémoire !

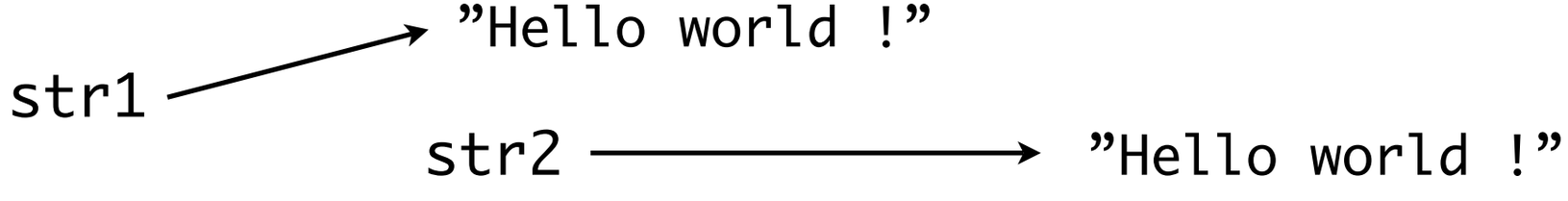


- Si l'on utilisait le constructeur `String`, avec un `new`, cela créerait un **nouvel exemplaire** de la même chaîne en mémoire, ce qui est un gaspillage sauf si l'on a d'excellentes et profondes raisons !

```
String str1 = ”Hello world !”;
String str2 = new String(”Hello ” + ”world !”);
boolean test = (str1 == str2);
System.out.println(test);
```



`false`



- Pour **tester si deux chaînes sont égales** (mêmes caractères), on utilisera donc plutôt la méthode d'instance `equals` :

```
String str1 = ”Hello world !”;
String str2 = new String(”Hello ” + ”world !”);
boolean test = str1.equals(str2);
System.out.println(test);
```

`true`

- Il est clair que `str1 == str2` implique `str1.equals(str2)`, mais la réciproque est fausse…

Le type primitif *caractère* (char)

- Avec les type `int`, `float`, `double` et `boolean`, le type `char` n'est pas une classe. **Un caractère n'est pas un objet !** La classe enveloppante se nomme `Character`.

- Exemples de caractères :

`'a'` `'A'` `'3'` `'!'` `'+'` `' '` *l'espace* `'.'`

- Les caractères américains (*dits internationaux*) sont numérotés de 0 à 127, c'est le **code ASCII** (*American Standard Code for Information Interchange*). Par exemple le code ASCII de `'a'` est égal à 97 mais on n'a pas besoin de le savoir...

• Des extensions multilingues sont proposées entre 128 et 255, c'est le **code ANSI**, connu sous le nom **ISO-Latin 1**. Par exemple le code de `'é'` est égal à 233, celui du `'ñ'` espagnol est 241.

• Pour faire du **calcul sur les caractères**, il faut les transformer en entiers : leur code.

• EXEMPLE : je veux **transformer une lettre** `c` **en minuscule** :

① si `c` n'est pas une lettre majuscule, rien à faire ! Sinon :

② soit `n` le code de la lettre majuscule `c`

③ soit `dist` la distance entre `'A'` et `'a'`

④ soit `m` l'entier `n + dist`

⑤ alors le résultat est le caractère `d` dont le code est `m`

• Un programme de **traitement de texte** a pour but d'effectuer des calculs et transformations sur des textes formés de caractères.

Le type char de Java et l'UNICODE

- Le codage ASCII, même étendu, est insuffisant pour certaines langues (arabe, chinois, hébreu, russe, etc).

- Le consortium UNICODE (`www.unicode.org`) a mis au point une numérotation des caractères qui représente la totalité des langues du monde, ainsi que les symboles spéciaux (musique, maths, etc).

- Un caractère est codé par un entier de type `int` (sur 16 bits). Mais Java fait la différence entre un caractère (type `char`) et son code (type `int`).

```
char c;
c = 'A';
c = '\u00e9';
```

⟶ *le numéro Unicode du caractère* `'é'` *est* `00e9` *en hexadécimal (base 16)*

N.B. *Pour compacter la numérotation, Unicode utilise un codage en base 16, qui tient sur 4 chiffres (0..9,a..f). Exemple :* `26` *en décimal s'écrit* `1a` *en hexa.*

Conversions entre int et char : du caractère à son code !

- Il est essentiel de savoir passer d'un caractère à son code, et inversement.

❶ Dans une opération arithmétique, un caractère vaut son code.

```
char c = 'A';
int n = c + 1;          // c est remplacé par 65, le code de 'A'
System.out.println("c = " + c + " et n = " + n);
```

```
c = A et n = 66
```

❷ A partir du code d'un caractère (un entier), on procède à un **forçage de type** (*type cast*) pour obtenir le caractère correspondant :

```
int n = 66;             // int n = 'A' + 1;
char c = (char) n;      // forçage de int vers char
System.out.println("n = " + n + " et c = " + c);
```

```
n = 66 et c = B
```

N.B. *Il est donc équivalent d'écrire* `int n = c;` *et* `int n = (int) c;`

Application : transformation Majuscule ➟ minuscule

• Mettons en forme le calcul de la page 6. Puisqu'un caractère vaut son code dans une opération arithmétique, **l'ordre des caractères** est induit par leur numérotation :

Version Processing

```
char maj2min(char c) {
  // si c n'est pas une lettre majuscule, rien à faire !
  if ((c < 'A') || (c > 'Z')) {
    return c;
  }
  // donc c est bien une lettre majuscule
  int decalage = 'a' - 'A';      // distance entre majuscules et minuscules
  int n = c + decalage;          // n est le code du caractère minuscule
  return (char) n;
}

void setup() {
  System.out.println("E en minuscule : " + maj2min('E'));
}
```

```
E en minuscule : e
```

• En Processing, la fonction `setup()` joue le rôle de méthode de test.

- En Java pur, nous ne pouvons pas rajouter de méthode d'instance à la classe `String`. Dans la classe courante, il suffit donc d'écrire une méthode de classe :

Version Java

```
class Truc {

  static char maj2min(char c) {
    //  si c n'est pas une lettre majuscule, rien à faire !
    if ((c < 'A') || (c > 'Z')) {
      return c;
    }
    //  donc c est bien une lettre majuscule
    int decalage = 'a' - 'A';      //  distance entre majuscules et minuscules
    int n = c + decalage;          //  n est le code du caractère minuscule
    return (char) n;
  }

  static void test() {
     System.out.println("E en minuscule : " + maj2min('E'));
  }
}
```

- **Il est essentiel de maîtriser le passage entre caractère et code !**

La classe enveloppante Character

- Le type char n'est pas une classe. Il existe pour chaque type primitif une **classe enveloppante**, ici Character.

- La classe Character a un autre intérêt : elle fournit plusieurs méthodes **statiques** que l'on aurait du mal à programmer. Par exemple... savoir si une lettre est en majuscule ! Car nous n'avons pas géré les caractères spéciaux 'Ç', 'É', ...

```
Character.isUpperCase('Ç')      → true
Character.isUpperCase('\u00C7') → true
Character.isLowerCase('a')      → true
Character.isLetter('$')         → false
Character.isDigit('8')          → true
Character.toUpperCase('a')      → 'A'
Character.toLowerCase('A')      → 'a'
```

La classe String

java.lang
Class String

- C'est une **collection numérotée de caractères** (char).
  - le premier caractère a pour indice 0.
  - l'implémentation exacte n'est pas connue.

- Les **méthodes d'instance** les plus utilisées sont :

`int length()`	*le nombre de caractères*
`char charAt(int i)`	*le caractère d'indice i*
`int indexOf(char c)`	*l'indice de la première apparition du caractère c*
`int indexOf(String str)`	*l'indice de la première apparition de la sous-chaîne str*
`String toLowerCase()`	*une copie de la chaîne en minuscule*
`String toUpperCase()`	*une copie de la chaîne en majuscule*

Les String sont inaltérables

- Cela signifie que, contrairement aux tableaux, **on ne peut pas modifier le caractère d'indice** `i` **directement**, on peut seulement construire une autre chaîne par concaténation de sous-chaînes.

- Par exemple, je veux mettre en majuscule le caractère d'indice i :

```
String enMaj(String str, int i) {
  char c = str.charAt(i);
  c = Character.toUpperCase(c);
  return str.substring(0,i) + c + str.substring(i+1);
}

void setup() {
  System.out.println("Hello world ! --> " + enMaj("Hello world !",6));
}
```

de `i+1` *jusqu'à la fin*

de `0` *à* `i-1`

```
Hello world ! --> Hello World !
```

Processing

Les tableaux de caractères

- Il est souvent plus pratique et plus efficace, pour transformer une chaîne, de passer par un tableau de caractères, pour revenir ensuite à une chaîne !

chaîne → tableau

`char[]`	**`toCharArray`**`()` Converts this string to a new character array.

tableau → chaîne

`String``(char[] value)` Allocates a new `String` so that it represents the sequence of characters currently contained in the character array argument.

- Par exemple, je veux mettre en majuscule le caractère d'indice i :

```
String enMaj(String str, int i) {
  char[] t = str.toCharArray();          // chaîne → tableau
  t[i] = Character.toUpperCase(t[i]);    // traitement du tableau
  return new String(t);                  // tableau → chaîne
}
```

Exemple - La méthode replace(...)

- BUT : Remplacer dans une chaîne `str` toutes les occurrences du caractère `cOld` par le caractère `cNew`.

- On transite par un tableau intermédiaire, que l'on transforme :

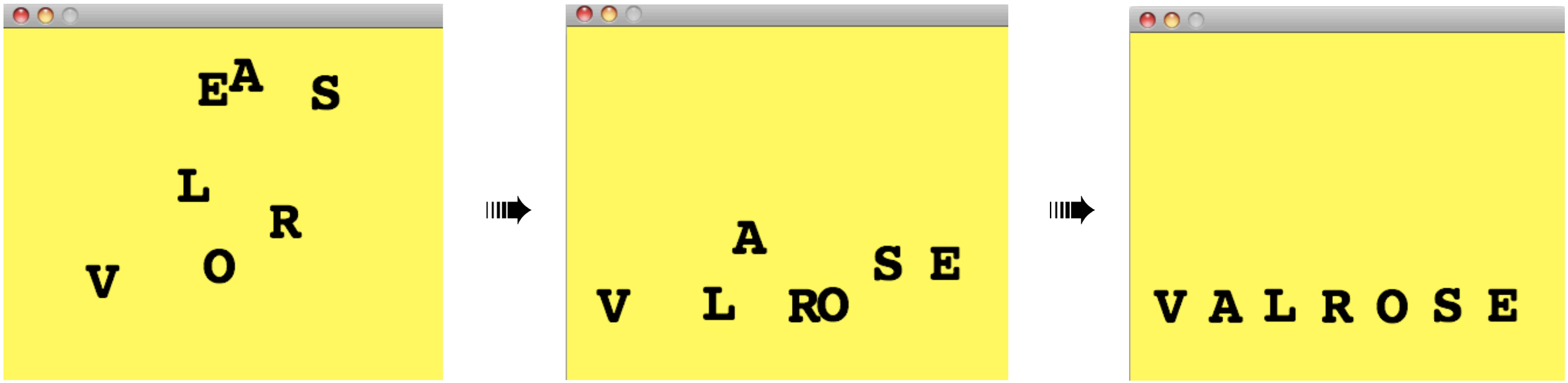
```
static
    String replace(String str, char cOld, char cNew) {
      char[] tmp = str.toCharArray();
      for(int i=0; i < tmp.length; i++) {
        if(tmp[i] == cOld) {
          tmp[i] = cNew ;
        }
      }
      return new String(tmp);
}

void setup() {
  String remp = replace("Hello World !",'l','L');
  System.out.println(remp);
}
```

```
HeLLo WorLd !
```

Exemple – Animation de caractères

- BUT : Animer des caractères placés au hasard dans un canvas, qui vont migrer avec un peu de hasard vers leur position finale...



- L'état d'un **caractère animé** est donné par :
  - le caractère lui-même, par exemple 'A'
  - la position courante (xc ; yc) du caractère dans le canvas
  - la position finale (xf ; yf) qu'il doit atteindre
  - le fait qu'il ait ou non déjà atteint sa position finale

- Nous développons donc la classe CharAnim des caractères animés :

```
class CharAnim {
  char c;                  // le caractère lui-même
  float xc, yc;            // sa position courante
  float xf, yf;            // sa position finale
  boolean fixed;           // est-il en position finale ?
  Random rand = new Random();

  CharAnim(char c, float xf, float yf, float xc, float yc) {
    this.c = c;
    this.xf = xf;
    this.yf = yf;
    this.xc = xc;
    this.yc = yc;
    fixed = false;
  }

  <méthode move>
}
```

- La méthode d'instance move() déplace un peu le caractère, soit au hasard, soit dans la direction de sa position finale :

```
void move() {
  if (!fixed) {
    int choix = rand.nextInt(2);  // au hasard ou vers la position finale ?
    if (choix == 0) {             // au hasard !
      xc = xc + rand.nextFloat();
      yc = yc + rand.nextFloat();
    } else {                      // vers la position finale !
      float dx = xf - xc, dy = yf - yc;     // la direction où aller
      xc = xc + dx / abs(dx);               // un pas unitaire
      yc = yc + dy / abs(dy);
    }
    fixed = abs(xc - xf) < 1 && abs(yc - yf) < 1;     // arrivé à destination ?
  }
}
```

- On parie que le processus converge et... il converge !

• **L'animation** proprement dite est un programme Processing. Il travaille sur une variable globale qui est un tableau de caractères animés :

```
CharAnim[] mot;
Random rand = new Random();
```

• La fonction setup() procède aux initialisations. La taille du canvas dépend du nombre de caractères et de la taille de la police :

```
void setup() {
  PFont font = loadFont("Courier-Bold-48.vlw");
  textFont(font);
  fill(0,0,0);                    // caractères dessinés en noir
  String str = "VALROSE";
  size(40+48*str.length(),300);
  mot = new CharAnim[str.length()];
  for(int i = 0; i < mot.length; i = i + 1) {
    mot[i] = new CharAnim(str.charAt(i),20+48*i,250,
                          rand.nextInt(300),rand.nextInt(200));
  }
}
```

- La fonction draw() exprime le dessin d'une image et prépare la transition vers l'image suivante en déplaçant un peu les caractères :

```
void draw() {
  background(255,255,0);
  for(int i = 0; i < mot.length; i = i + 1) {
    text(mot[i].c,mot[i].xc,mot[i].yc);
    mot[i].move();
  }
}
```

- A vous d'imaginer toutes les variantes possibles. Et souvenez-vous : **ce n'est qu'en programmant qu'on apprend à programmer !**