Utilisation d'objets: la classe String

Maria Virginia Aponte

CNAM-Paris

Année 2012/2013

Les objets en Java

- déclarés à partir d'un nom de classe (ou d'interface). Ex: `String` est un nom de classe;

```
String s;
```

- crées (souvent) avec `new` + le nom de la classe;

```
Compte c = new Compte();
```

- on dit de l'objet `c` :
  - qu'il est une instance de la classe `Compte`,
  - et aussi, que son type est `Compte`
- on peut appliquer des méthodes de la classe sur un objet

Appliquer des méthodes sur un objet

Les objets contiennent (en général) des méthodes *non statiques*:

- définies dans la classe (type) de l'objet;
- applicables sur l'objet, avec une syntaxe particulière:

```
MaClasse c = new MaClasse();
// La classe MaClasse contient la méthode non statiqu
c.m(...);
```

- la syntaxe suivante n'est pas admise (si m est non statique):

```
MaClasse c = new MaClasse();
// La classe MaClasse contient la méthode non statiqu
m(c);
```

Le type String

- modélise les suite de caractères
- syntaxe: caractères entourées entre guillements doubles,
- constantes: "Bonjour", "189GH7?"
- représentés avec pointeurs (comme tout objet et aussi comme les tableaux).
- caractères accessibles par position à partir de 0 (comme les tableaux).

Différences avec les tableaux

- chaîne $\neq$ tableau caractères!
- syntaxe dédiée pour les constantes: "Abc" $\neq$ `{ 'A', 'b', 'c'}`
- l'accès par position se fait via une méthode: `s.charAt(0)` $\neq$ `tc[0]`

```
char [] tc = {'a', 'b', 'c'};
String s= "Abc";
char c = s.charAt(0);
char c1 = tc[0];
```

- les caractères d'une chaîne ne sont pas modifiables,
- la longueur d'un tableau est un attribut: `t.length`
- la longueur d'une chaîne s'obtient via une méthode (sans paramètres): `s.length()`

Déclaration et création des Strings

- avec constantes: "Bonjour", "189GH7?"
- avec `new`. Par exemple:

```
char [] tc = {'a', 'b', 'c'};
String s = new String(t);
String s2 = new String(new char []{'a', 'b', 'c'});
```

- en résultat d'une concaténation et/ou conversion:

```
String s = "" + 35 + 'a';
```

- en résultat d'une méthode qui renvoie un String;

```
String s = tc.toString();
String s2 = s.toUpperCase();
```

Variables et initialisation

`String` est un type référence:

- référence = pointeur;
- une variable de type référence déclarée et non initialisée contient l'adresse `null`:

```
String s;     // s contient null
s.length(); // provoque une erreur
```

- si une variable contient `null`:
  - l'objet référencé par la variable n'existe pas;
  - impossible d'invoquer des méthodes sur cette variable;
  - invocation sur `null` $\Rightarrow$ erreur exécution

Affectation entre variables référence

- Possible à condition que les types de ces variables soient compatibles.
  Ex: entre deux tableaux de int, entre deux Strings, etc.

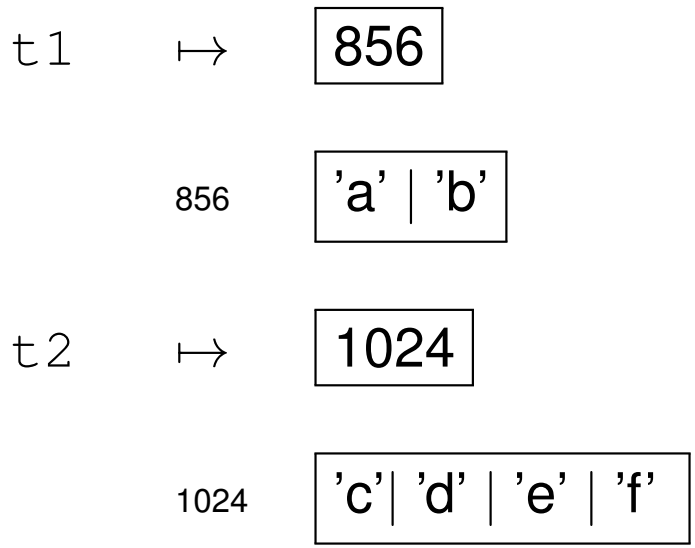
- Quelle est le résultat d'une telle affectation?

```
String s1, s2, s3;
s1 = "ab";
s2 = "cdef";
s1 = s2;
```

$\Rightarrow$ On copie le contenu d'une variable dans l'autre.
$\Rightarrow$ Ce contenu est **une adresse**.

Affectation entre variables String

```
String s1, s2, s3;
s1 = "ab";
s2 = "cdef";
s1 = s2;
```

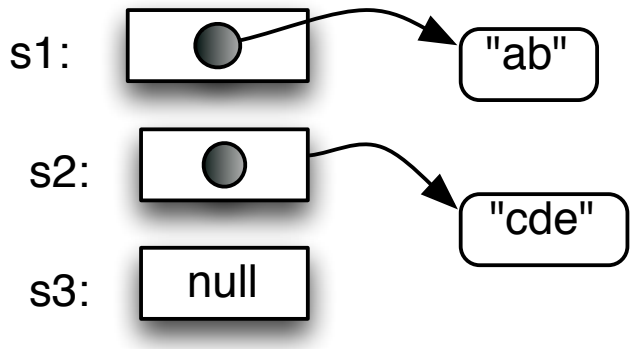


On recopie le contenu d'une variable dans l'autre. On recopie une adresse.

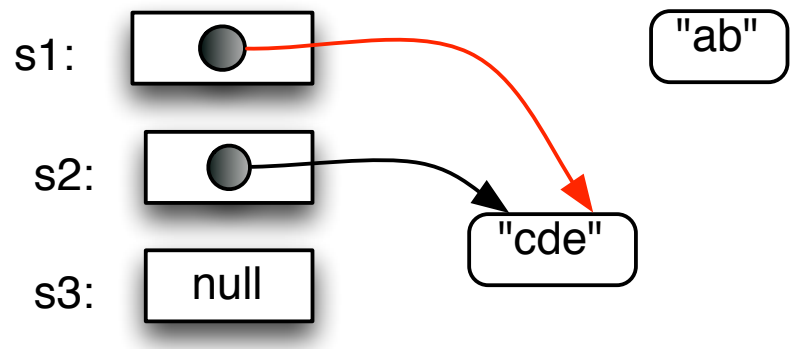
$\Rightarrow$ s1 et s2 contiennent la **même adresse**.

Affectation entre variables String

```
String s1, s2, s3;
s1 = "ab";
s2 = "cdef";
s1 = s2;
```



Avant affectation



Après affectation

Comparer tableaux, Strings

- Strings et tableaux sont des types référence (pointeurs);
- `s1 == s2` compare leurs adresses $\Rightarrow$ pointent-ils vers le même emplacement mémoire?
- pas la bonne méthode si l'on veut comparer leur contenu, c.a.d, si leurs valeurs internes sont identiques.
- $\Rightarrow$ utiliser ou écrire des méthodes qui comparent une à une chacune des composantes internes.

Méthodes de la classe String

- `s.length()` renvoie la longueur de la chaîne s;
- `s.charAt(N)` renvoie le Nth caractère de la chaîne s.
- `s.toCharArray()` renvoi un tableau de char contenant tous les caractères de la chaîne s.

```
String s = "Salut";
char [] tc = s.toCharArray();
 // tc = {'S', 'a', 'l', 'u', 't'}
```

- `s1.equals(s2)` renvoie true si s1 contient la même suite de caractères que s2. Permet de comparer deux chaînes par égalité de leur contenus et non de leurs adresses.
- `s1.equalsIgnoreCase(s2)` comme la méthode précendente mais en ignorant la différence entre majuscules et minuscules.

Méthodes de String (2)

- `s.substring(N,M)` renvoie la sous-chaîne de s allant de positions N, N+1,..., jusqu'à M-1 compris.
- `s1.indexOf(s2)` renvoie un entier. Si s2 est une sous-chaîne de s1, la valeur retournée est la position de son premier cacatère dans s1. Sinon, retourne -1. Peut-être employé avec un caractère ch: s1.indexOf(ch),
- `s1.compareTo(s2)` compare s1 et s2 et renvoie un entier. S'ils sont égaux, renvoie 0, si $s1 < s2$, renvoie une valeur négative, si $s1 > s2$, renvoie une valeur positive. L'ordre considéré est l'ordre alphabétique.

Méthodes de String (3)

- `s.toUpperCase()` renvoie une nouvelle chaîne égale à s, où toutes les minuscules sont changées en majuscules.
- `s.toLowerCase()` comme avant mais pour le changement de majuscules en minuscules.
- `s.trim()` renvoie une nouvelle chaîne égale à s1, où tous les caractères blancs ou tabulations ont été supprimés du début et de la fin de s.
- `s.split(String sp)` découpe la chaîne en plusieurs morceaux en utilisant la chaîne `sp` comme séparateur. Ex:

```
String s = "␣un␣**deux**trois␣";
String [] res = s.split("**");
 // res = {" un ", "deux", "trois "}
```

Exemples

```
public static void main(String [] args){
  String s = "Il␣rencontre␣un␣chien␣et␣un␣chat";
  int k; String t;
  for (int i = 0; i <s.length(); i++){
    Terminal.ecrireStringln(i + "␣-->␣" + s.charAt(i)  );
  }
}
```

```
0 --> I
1 --> l
2 -->
3 --> r
4 --> e
5 --> n
6 --> c
7 --> o
...
```

Exemples

```
Terminal.ecrireString("la␣sous␣chaine␣entre␣7␣et␣11␣est␣:
Terminal.ecrireStringln(s.substring(7,11) );

Terminal.ecrireString("entrer␣un␣mot␣:" );
t = Terminal.lireString();
k=s.indexOf(t);
if (k==-1){
  Terminal.ecrireStringln(t + "␣n'est␣pas␣dans␣" + s );
}
else{
  Terminal.ecrireStringln
    ("la␣premiere␣position␣de␣" + t +"␣est␣:␣" + k );
}
```

```
la sous chaine entre 7 et 11 est :ontr
entrer un mot :un
la premiere position de un est : 13
```

Exemples

```
k=s.lastIndexOf(t);
if (k==-1){
  Terminal.ecrireStringln(t + "␣n'est␣pas␣dans␣"+ s );
}
else{
  Terminal.ecrireStringln
    ("la␣derniere␣position␣de␣" + t +"␣est␣:␣" + k );
}
```

```
la derniere position de un est : 25
```

Paramètre de la méthode main

- La méthode `main` possède un paramètre de type `String []`, qui est tableau de chaînes de caractères.
- Il est initialisé avec des informations saisies directement dans la ligne de commande lance l'exécution du programme.
- Cela permet de communiquer des informations au programme depuis la ligne de commande, par exemple, un nom de fichier, une date, etc.

Paramètre de la méthode main: exemple

Exemple: ce programme affiche les chaînes passées dans la ligne de commande.

```
public class LigneCommande{
 public static main(String [] args){
    for (int i=1; i<args.length; i++){
       Terminal.ecrireStringln(args[i]);
    }
 }
}
```

```
java LigneCommande lundi mardi mercredi
lundi
mardi
mercredi
```

Conversion `String` → autres types

Il existe des classes dédiées aux types primitifs:

- `Integer` pour le type primitif `int`,
- `Double` pour le type primitif `double`,
- `Character` pour le type primitif `char`, ...

Ces classes donnent une version "objet" des types primitifs, mais contiennent également des nombreuses méthodes utiles:

- pour conversion en `int`, on utilise `Integer.parseInt`
- pour conversion en `double`, on utilise `Double.parseDouble`
- pour conversion en `boolean`, on utilise `Boolean.parseBoolean`
- ...

Conversion `String` → autres types

Exemple: convertir la chaîne `"12"` vers `int`.
Nous employons `Integer.parseInt`

```
public class ConvertString{
 public static main(String [] args){
    int x;
    String s = "12";
    x = Integer.parseInt(s);
    x = x+2;
    Terminal.ecrireIntln(x);
  }
}
```

Conversion autres types $\rightarrow$ `String`

- conversion type primitif vers chaîne: le plus simple est d'utiliser l'opérateur de concaténation,
- concaténer la chaîne vide à la valeur à convertir: une conversion implicite est opérée automatiquement par Java.

Exemple: convertir l'entier `12` en `String`.

```
public class ConvertString{
 public static main(String [] args){
    int x = 12;
    String s = ""+x;
   Terminal.ecrireStringln(x);
  }
}
```

Autre possibilité: méthodes `toString` des classes dédiées aux types primitifs: `Integer.toString(int)`, `Boolean.toString(int)`, etc.

Méthodes de la classe Character

Méthodes de test:

- `Characater.isLetter(c)`
- `Characater.isDigit(c)`
- `Characater.isUpperCase(c)`
- `Characater.isLowerCase(c)`

et des méthodes de conversion:

- `Characater.toUpperCase(c)`
- `Characater.toLowerCase(c)`
- `Characater.toString(c)`