

Programmation Orientée Objet - Licence TIS CM9/9

Lancelot PECQUET

Lancelot.Pecquet@math.univ-poitiers.fr



Poitiers, le 20/03/2006

Rappel sur la séance précédente

La fois précédente, nous avons vu :

- ④ fichiers
- ② sérialisation
- ③ threads
- ④ applets

Aujourd'hui, nous voyons :

- ① Principe de la séance d'aujourd'hui
- ② Révision autour de String
- ③ Comment obtenir l'immuabilité ?
- ④ Session interactive de questions de révisions

Comprendre ce qu'on ne comprend pas

- autour de la classe `String` qui vous est familière
- réviser les différents concepts de POO
- prendre conscience de ce que vous n'avez pas compris
- prétexte pour vous inciter à redécouvrir notes de cours et de TD ainsi que vos programmes

De Java en particulier à la POO en général

- Java a été notre langage support pour ce cours
- dans ce qui suit, beaucoup de réponses proviennent de choix spécifiques de Java
- les mêmes problèmes se posent dans tous les langages OO
- l'héritage traduit la relation "est un"
- abstraction OO \implies on ne sait plus toujours ce que ce "est" signifie (d'où égalité, transtypage, clonage...)
- cette polysémie au niveau abstrait engendre beaucoup d'erreurs de programmation et de conception
- l'objectif est de comprendre les questions qui se posent et de discuter les choix qui ont été faits

Opérateur == et equals()

```
1 String s1 = "toto";  
2 String s2 = s1;  
3 System.out.println(s1 == s2);
```

- true car...
- la sémantique de == est l'égalité de références et que s1 et s2 sont des références à la même adresse

```
1 System.out.println(s1.equals(s2));
```

- true car...
- sémantique de equals() est l'égalité caractère par caractère et que s1 et s2 sont identiques

Opérateur == (suite)

```
1 String u1 = "toto";  
2 String u2 = "toto";  
3 System.out.println(u1 == u2);
```

- true car...
- la chaîne toto est initialement partagée

```
1 u1 = "titi";  
2 System.out.println(u1 == u2);
```

- false car...
- désormais, u1 et u2 ont des valeurs différentes

```
1 u1 = "to" + "to";  
2 System.out.println(u1 == u2);
```

- true car...
- la chaîne toto est reconstituée et comparée aux chaînes existantes (compteur de références)

Constructeur de String

```
1 String u = new String("toto");  
2 String v = new String("toto");  
3 System.out.println(u == v);
```

- false car...
- l'invocation des constructeurs crée deux instances distinctes de la chaîne toto dans la mémoire

Mutation d'une String

```
1 String s1 = "toto", s2 = s1;  
2 s1 += "!";  
3 System.out.println(s1);
```

- toto !

```
1 System.out.println(s1 == s2);
```

- false...

```
1 System.out.println(s1.equals(s2));
```

- false

```
1 System.out.println(s2);
```

- toto car...
- String étant immuable, l'opération `s1 += "!"` a consisté à créer une nouvelle instance de String contenant la chaîne toto ! et à faire pointer s1 dessus : aussi s2 n'est-elle pas modifiée

Conséquence de l'immuabilité

- comment créer la chaîne a^{5000} ?

```
1 String s = ""; int n = 5000;  
2 for (int i=0; i<n; i++) {s += "a";}
```

- complexité en espace ?
- $1 + 2 + \dots + n = O(n^2)$
- il faut utiliser un StringBuffer pour avoir du $O(n)$

Et les char[] ?

```
1 char[] v = {'a', 'b', 'c', 'd', 'e'};  
2 System.out.println(v);
```

- abcde...

```
1 int[] z = {1, 2, 3, 4, 5};  
2 System.out.println(z);
```

- [I@1f6a7b9 car...
- int[] ne redéfinit pas le toString() d'Object

Et les char[] (suite) ?

```
1 String s = new String(v); // constructeur a partir d'un char[]
2 char[] u = s.toCharArray();
3 System.out.println(s == u);
```

- incomparable types: java.lang.String and char[]

```
1 System.out.println(u == v);
```

- false car...
- u et v sont deux références distinctes

Et les char[] (suite) ?

```
1 System.out.println(u.equals(v));
```

- false car...
- le equals() d'Object n'a pas été redéfini dans char[]

```
1 System.out.println(u.equals(s));
```

- false car...
- on est dans le même cas que précédemment

```
1 System.out.println(s.equals(u));
```

- false car...
- le x.equals(y) avec x : String renvoie false si on n'a pas y : String
- nb : String est final donc ?...
- le problème de l'égalité avec les héritières ne se pose pas

Accès au i -ième caractère

```
1 String s = new String("toto");  
2 System.out.println(s[1]);
```

- Main.java: array required, but java.lang.String found
- comment obtenir le caractère en position 1 ?
- `char c = s.charAt(1)` : accesseur \implies encapsulation

```
1 s.charAt(1) = 'a';
```

- Main.java : unexpected type
required : variable - found : value
- comment transformer toto en titi ?
- on ne peut pas !

```
1 System.out.println(s.charAt(10));
```

- `StringIndexOutOfBoundsException`

Transtypages

- comment obtenir la *chaîne de caractères* 123 à partir de l'*entier* 123 ?

```
1 String un_deux_trois = String.valueOf(123);
```

- comment obtenir l'*entier* 123 à partir de la *chaîne de caractères* 123 ?

```
1 int cent_vingt_trois = Integer.parseInt(un_deux_trois);
```

```
1 String s = "1" + 2 + "3";  
2 System.out.println(s);
```

- 123 car...
- le 2 est transtypé automatiquement en String par + dont l'opérande gauche est de type String (surprenant : cela marcherait avec `s = 1 + "2" + 3` par autoboxing)

StringBuffer pour la manipulation des chaînes

```
1 StringBuffer t1 = new StringBuffer("toto");  
2 StringBuffer t2 = t1;  
3 System.out.println(t1 == t2);
```

- true

```
1 t1.append("!");  
2 System.out.println(t1 == t2);
```

- true

```
1 System.out.println(t1);
```

- toto !

```
1 System.out.println(t2);
```

- toto !

Immuabilité

```
1 class personne{...  
2   String nom; int age; int[] tel;  
3   ...}
```

- comment obtenir l'immuabilité ?
- mettre les champs en private et, pour getAge() ?

```
1 final int getAge(){return x;} // type primitif passe par valeur (final)
```

- pour getNom() ?

```
1 final String getNom(){return nom;} // type immutable (final)
```

- pour getTel() ?

```
1 final int[] getTel(){return tel.clone();} // il faut cloner (final)
```

- sinon on peut faire...

```
1 p.getTel()[0] = 1;
```

- il faut aussi dans le constructeur : this.tel = tel.clone()

Questions ?



Session interactive de questions de révisions

- ④ classes, constructeurs, encapsulation, destruction
- ② héritage simple, polymorphisme, Object
- ③ classes abstraites, héritage multiple, interfaces
- ④ interruptions, paquetages, javadoc, collections, (Java 1.5)
- ⑤ transtypage, égalité, comparabilité, clonage
- ⑥ UML, conception OO, projet
- ⑦ fichiers, sérialisation, (threads, applets)

Conclusion

Aujourd'hui, nous avons vu :

- 1 Principe de la séance d'aujourd'hui
- 2 Révision autour de `String`
- 3 Comment obtenir l'immuabilité ?
- 4 Session interactive de questions de révisions

Bon courage pour les révisions et le projet.