Support de cours

Cours:

Introduction à la programmation orientée objet (en Java)

Vidéo:

W15-01-attributstat-JAVA-pt2

Concepts (extraits des sous-titres générés automatiquement) :

Variable d'instance. Cas d'une classe. Variable de classe. Variable statique. Création de toute instance. Âge officiel de départ. Travers du nom de la classe. Objet de type. Exécution du code suivant. Création d'une instance. Attributs de la classe. Exemple concret. Raison de cette ligne. Première ligne de code. Valeur de la variable d'instance.

vers la recherche de séquences vidéo
(dans Introduction à la programmation orientée objet (en Java).)

vers la vidéo

EPFL

Attributs statiques
(Partie 2)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

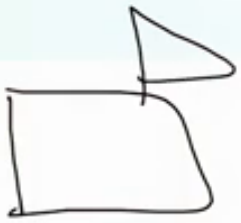
...

notes

résumé

0m 0s

Variable statique : exemple



```
class Abc {
   public static void main(String[] args) {
      A.c++;
      A v1;
      v1 = new A();
      v1.modifier();
   }
}
class A {
   int b = 1;          // variable d'instance
   static int c = 10; // variable de classe
   void modifier() {
      b++;
      c++;
   }
}
```

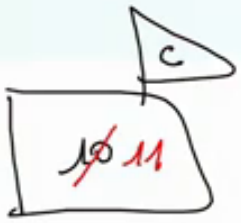
Voyons ceci sur un exemple concret. Nous avons le cas d'une classe « A » qui contient une variable d'instance, un attribut comme nous l'avons toujours utilisé jusqu'ici, et une variable de classe, c'est-à-dire un attribut dont la déclaration est précédée du mot réservé « static ». La classe « A » contient également une méthode modifiée, laquelle peut accéder à tous les attributs de la classe, qu'ils soient statiques ou non statiques. Examinons maintenant pas à pas ce qui se passe lors de l'exécution du code suivant. En fait, en amont même de l'exécution, dès que la classe « A » est utilisée dans un programme, étant donné qu'elle contient une variable statique « c », il y aura réservation d'un emplacement mémoire pour cette variable statique « c »,

notes

résumé

0m 1s

Variable statique : exemple




```
class Abc {
   public static void main(String[] args) {
      A.c++;
      A v1;
      v1 = new A();
      v1.modifier();
   }
}
class A {
   int b = 1;          // variable d'instance
   static int c = 10; // variable de classe
   void modifier() {
      b++;
      c++;
   }
}
```

laquelle sera initialisée à 10 en raison de cette ligne. Si l'on examine cette première ligne de code, la première chose que l'on peut remarquer, est que la variable de classe est accessible indépendamment de la création de toute instance, on peut accéder à l'attribut « c » au travers du nom de la classe uniquement, sans passer par la création d'une instance. Lorsque l'on exécute cette ligne de code, on va incrémenter la variable de classe « c », variable de classe de la classe « A », laquelle a déjà été initialisée, nous allons donc nous trouver dans cette situation, la valeur de « c » va passer de 10 à 11. Les deux lignes suivantes déclarent une instance de « A », et l'initialisent au moyen du constructeur par défaut. Chaque instance de « A » dispose de ses propres variables d'instance,

notes

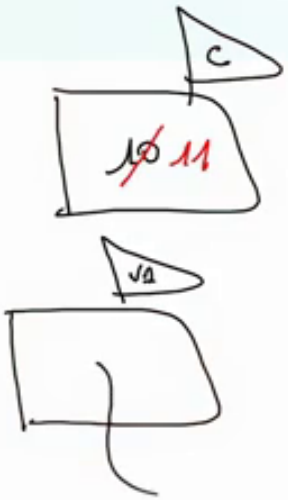
résumé

0m 49s

Variable statique : exemple

EPFL

```
class Abc {
   public static void main(String[] args) {
      A.c++;
      A v1;
      v1 = new A();
      v1.modifier();
   }
}
class A {
   int b = 1;          // variable d'instance
   static int c = 10; // variable de classe
   void modifier() {
      b++;
      c++;
   }
}
```



ici en l'occurrence, la variable d'instance « b ”, ce qui veut dire qu'au terme de l'exécution de cette ligne, la situation en mémoire sera la suivante, nous aurons donc une variable « v1 »

notes

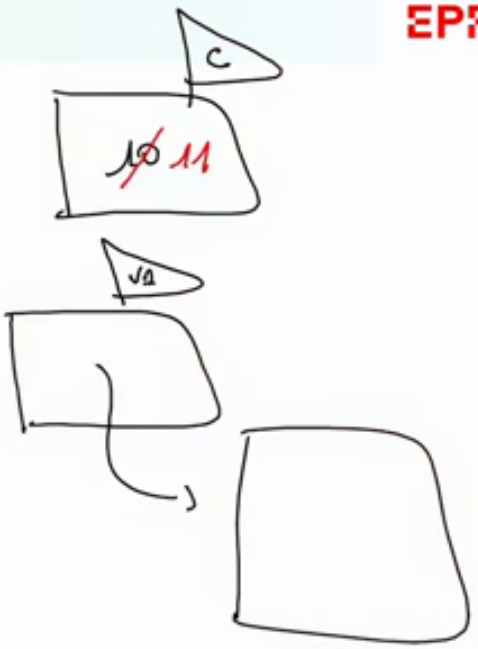
résumé

1m 37s

Variable statique : exemple

EPFL

```
class Abc {
   public static void main(String[] args) {
      A.c++;
      A v1;
      v1 = new A();
      v1.modifier();
   }
}
class A {
   int b = 1;          // variable d'instance
   static int c = 10; // variable de classe
   void modifier() {
      b++;
      c++;
   }
}
```



contenant une référence vers un objet de type « A »,

notes

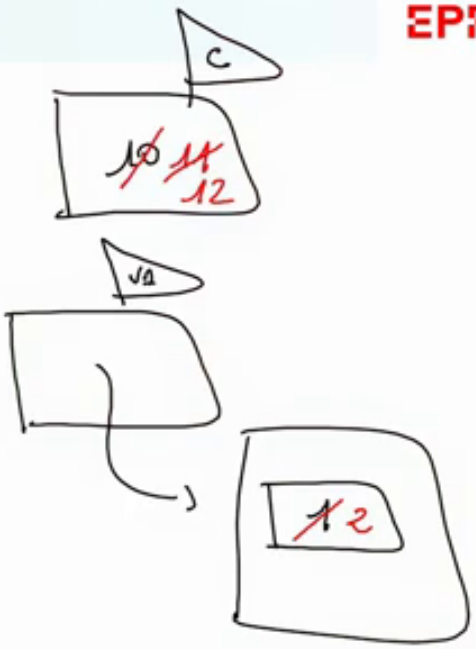
résumé

1m 49s

Variable statique : exemple

EPFL

```
class Abc {
   public static void main(String[] args) {
      A.c++;
      A v1;
      v1 = new A();
      v1.modifier();
   }
}
class A {
   int b = 1;          // variable d'instance
   static int c = 10; // variable de classe
   void modifier() {
      b++;
      c++;
   }
}
```



qui dispose d'un attribut « b », lequel est initialisé grâce à cette ligne, à 1. La méthode modifiée est ensuite appelée sur l'instance « v1 », cette méthode incrémente la valeur de la variable d'instance « b », nous allons donc nous trouver dans cette situation, au terme de l'exécution de cette ligne. La méthode modifiée incrémente ensuite la variable de classe « c ¢, vous noterez au passage que les méthodes de la classe « A » peuvent accéder aux attributs statiques, de la même façon exactement qu'ils accèdent aux attributs non statiques, aux variables d'instance. L'exécution de cette seconde ligne de la méthode modifiée va donc résulter dans une incrémentation supplémentaire

notes

résumé

1m 51s

Valeur commune

EPFL

Exercice : intégrons à une classe `Employe` le fait que 65 ans est l'âge officiel de départ à la retraite

Considérons les deux versions suivantes :

- avec une variable d'instance `ageRetraite`
- avec une variable statique `ageRetraite`

de la variable de classe « c », laquelle va passer à 12. On a donc pu ici modifier la valeur de la variable de classe de l'attribut statique aussi bien en passant uniquement par le nom de la classe, qu'en passant par une instance, donc ici au travers de l'instance « v1 », on a pu modifier la valeur de la variable de classe « c ». La variable modifiée, que ce soit au travers du nom de la classe, ou au travers d'une instance, est une zone mémoire unique. Nous avons donc vu au travers de cet exemple que lorsque l'on modifie une variable d'instance, la valeur change uniquement pour l'objet courant, par contre, lorsqu'on modifie une variable de classe, la valeur change pour tous les objets de la classe. En effet, la variable de classe est unique et accessible par tous les objets. Comme nous venons de le voir sur l'exemple précédent, une variable statique peut être utilisée au travers du nom de la classe sans passer par la création d'un objet, on peut donc utiliser des membres statiques en se passant complètement des objets, ce qui va à l'encontre de l'approche « orienté objet ». Utiliser des membres statiques pour contourner l'approche « orienté objet » est évidemment une mauvaise raison d'utiliser des variables statiques. Une bonne raison par contre est de représenter une valeur commune à tous les objets d'une classe. Étudions ceci sur un exemple concret. Imaginons que dans un programme, nous ayons à utiliser une classe « Employe », et que nous devions représenter le fait que 65 ans est l'âge officiel de départ à la retraite pour tous les employés. Et considérons deux versions possibles pour mettre en place cela. Donc une version où on utilise une variable d'instance pour modéliser l'âge de la retraite,

notes

résumé

2m 37s

Classe Employe1

EPFL

- Version avec une variable d'instance pour l'âge de la retraite :

```
class Employe1 {
  private String nom;
  private int ageRetraite;
  // ...
  public Employe1(String unNom,  int unAgeRetraite) {
    nom = unNom;
    ageRetraite = unAgeRetraite;
    // ... reste des initialisations
  }
  // ...
}
```

et une autre version où l'on utilise une variable de classe, une variable statique.

notes

résumé

4m 13s

Classe Employe2

EPFL

- Version avec une variable *statique* pour l'âge officiel de départ à la retraite

```
class Employe2 {
  // ...
  // Seule modification,  ageRetraite devient static:
  static int ageRetraite;
  // ...
}
class Entreprise {
  public static void main (String[] args) {
  //   ...
  // Remplissage du tableau comme avant.
  // Modification de l'âge de la retraite :
  // aucun parcours du tableau nécessaire
  Employe2.ageRetraite = 67;
  employes[0].ageRetraite = 67;   // alternative
  employes[250].ageRetraite = 67; // alternative
  }
}
```

Première version donc, une classe « Employe » qui modéliserait l'âge de la retraite comme une variable d'instance. Le constructeur de la classe « Employe1 » serait donc en charge d'initialiser chacun des attributs au moyen de valeurs passées en paramètres. Supposons maintenant qu'une classe « Entreprise » utilise ce type « Employe1 », notamment pour créer un tableau contenant un certain nombre d'employés, et chacun des éléments du tableau sera rempli en créant une instance d'employés. Comme l'âge officiel de départ à la retraite a été modélisée au moyen d'une variable d'instance, chaque employé va stocker une zone mémoire qui contient cet âge officiel, qui se trouve être le même pour tout le monde. Il y a donc pour chaque employé réservation d'une zone mémoire pour contenir exactement la même information. Donc si on a un milliers d'employés, on va se retrouver avec mille fois la valeur 65 stockée, ce qui est évidemment une duplication inutile. De plus, cette duplication cause de gros problèmes de maintenance, supposons par exemple que la loi change, et que l'âge officiel de départ à la retraite soit désormais de 67 ans. Il faudrait donc parcourir toutes les instances possibles pour aller modifier cet âge, de façon appropriée. Si les instances ont été stockées dans un tableau, c'est encore faisable, mais imaginez que les instances ont été créées de façon indépendante, ceci pose évidemment un gros souci de mise à jour. Dans la seconde version, où on utilise maintenant une variable statique pour l'âge officiel de départ à la retraite, donc où l'on précède la déclaration de l'attribut du mot réservé « static », on a plus du tout ce problème, en effet il existera une zone mémoire unique accessible par toutes les instances, qui contiendra cet âge de la retraite officiel, qui est le même pour tout le monde.

notes

résumé

4m 20s

Constantes : `final` et `static`

Pour les constantes communes à toutes les instances d'une classe :

- inutile de stocker une valeur pour chaque objet de la classe
- les déclarer en `final static`

```
class Planete {
   // G = constante gravitationnelle
   // Une variable G pour chaque planète :
   // Possible
   private final double G = 6.674E-8;

   // Une variable G pour toutes les planètes :
   // BEAUCOUP MIEUX !
   private final static double G = 6.674E-8;

   // ...
}
```

Cet âge de la retraite pourra être accédé via toute instance, ou via le nom de la classe, il n'y a plus de duplication d'informations, et il est facile de faire la modification. On n'a ici plus besoin de rechercher toutes les instances possibles d'employés pour y modifier l'âge de la retraite et le faire passer à 67 ans. Voilà, j'espère que ce petit exemple vous aura convaincus de l'utilité d'utiliser à bon escient des variables statiques,

notes

résumé

6m 1s

notons également sur ce chapitre que pour les constantes qui sont communes à toutes les instances d'une classe, il est inutile de stocker une valeur pour chaque objet de la classe, il faut en fait les déclarer en « final static ». Par exemple, si pour une classe « Planete », on a besoin d'utiliser G, la constante gravitationnelle, cette constante est la même pour toutes les planètes, et dans ce cas là, il vaut mieux déclarer la constante en question comme « final static », plutôt que simplement comme « final », pour éviter de dupliquer cette constante pour chaque instance de planète. pour chaque instance de planète.

notes

résumé

6m 27s