Support de cours

Cours:

Initiation à la programmation (en Java)

Vidéo:

Init-JAVA-01-3-Variables-pt4

Concepts (extraits des sous-titres générés automatiquement) :

Première chose. Exécution d'une telle instruction. Évaluation de l'expression. Notion d'affectation. Égalité mathématique. Fin de l'instruction d'affectation. Petit détail syntaxique. Genre d'expressions. Variables n. Genre de tournure. Première instruction. Cas concret. Second temps. Relation de cette nature. Cas de la seconde instruction.

vers la recherche de séquences vidéo
(dans Initiation à la programmation (en Java).)

vers la vidéo

EPFL

Variables
(Partie 4)

Initiation à la programmation (Java)

Jamila Sam, Vincent Lepetit et Jean-Cédric Chappelier

...

notes

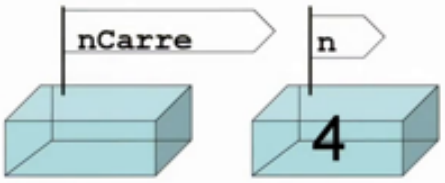
résumé

0m 0s

Affectations

L'exécution d'une affectation se décompose en deux temps :

```
nCarre = n * n;
```



Supposons que dans un programme, j'ai déclaré deux variables n et nCarré.

notes

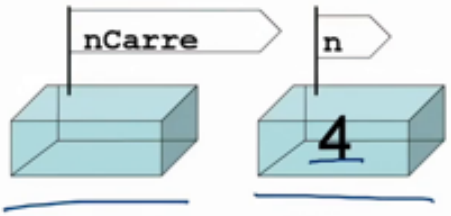
résumé

0m 1s

Affectations

L'exécution d'une affectation se décompose en deux temps :

```
nCarre = n * n;
```



N a été déclaré et initialisé à 4, alors que nCarré n'a pas été initialisé. Je souhaite mettre une valeur dans nCarré, en utilisant la notion d'affectation. Comment va se dérouler une affectation de cette nature ? Il faut savoir que l'exécution d'une telle instruction

notes

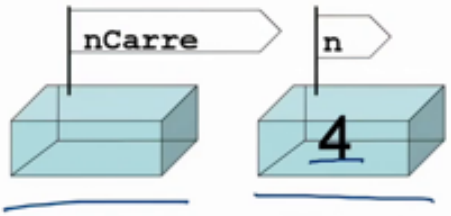
résumé

0m 6s

Affectations

L'exécution d'une affectation se décompose en deux temps :

```
nCarre = n * n;
```



va se décomposer en deux temps. La première chose qui va être réalisée,

notes

résumé

0m 25s

Affectations

De façon plus générale, une affectation suit le schéma:

```
nom_de_variable = expression;
```

Une expression calcule une valeur, qui doit être de même type que la variable.

Exemples d'expression:

- `4`
- `n * n`
- `n * (n + 1) + 3 * n - 2`

Nous reviendrons sur les expressions un peu plus loin.

c'est l'évaluation de l'expression, qui est à droite du symbole =. Donc ici, cette évaluation va me donner la valeur 16. Dans un second temps, je vais stocker la valeur ainsi évaluée dans la variable nCarré. Ce qui me donne ici, pour résultat, d'avoir dans la variable nCarré la valeur 16. Pour résumer, la syntaxe de l'affectation en java est la suivante : on donne le nom de la variable, suivi du symbole égal (=),

notes

résumé

0m 31s

Attention: Ne confondez pas une affectation avec une égalité mathématique.

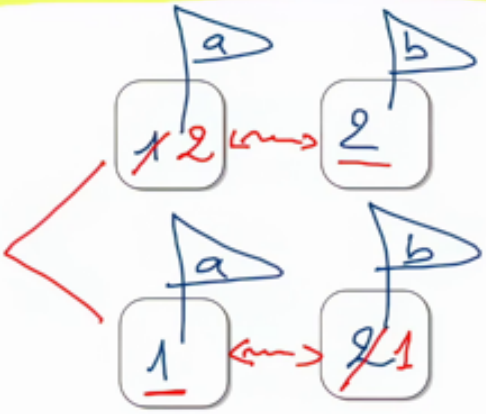
Toutes les deux utilisent le signe égal =, mais l'affectation est un mécanisme dynamique.

Par exemple, les deux instructions:

```
a = b;
b = a;
```

copie la valeur de b dans a

copie la valeur de a dans b



ne sont pas identiques.

et suivi d'une expression valide en java. L'expression peut être quelque chose de tout à fait élémentaire, comme ici, une valeur littérale, comme ça peut être quelque chose de plus complexe, comme ici. Nous aurons l'occasion de revenir plus en détails sur les expressions. La chose importante à respecter est que l'expression doit calculer une valeur, qui doit être du même type que la variable. Petit détail syntaxique à nouveau, ne pas oublier le point virgule (;) à la fin de l'instruction d'affectation. Revenons maintenant un peu sur la comparaison entre égalité mathématique et affectation. Toutes les deux vont utiliser le signe égal (=) et la notion de variables. Simplement, si en mathématique je rédige ce genre d'expressions, je signifie que les variables a et b, auront toujours les mêmes valeurs au cours du temps. Par contre, l'affectation en programmation est un mécanisme différent car il va être dynamique, donc va dépendre du temps. Prenons un cas concret. Imaginons ici que a stocke la valeur 1, et b stocke la valeur 2. Si j'exécute maintenant cette première instruction, je commence, comme on l'a vu tout à l'heure, par évaluer l'expression, qui est à droite de l'affectation, ce qui va me donner 2. Et ensuite je vais stocker cette valeur, ce 2 dans la variable a, ce qui me produit le résultat suivant. Maintenant, prenons le cas de la seconde instruction, et plaçons-nous dans les mêmes conditions de départ, avec un a stockant 1, et un b stockant 2. Si j'exécute maintenant cette instruction, comme tout à l'heure, j'évalue ce qui est à droite de l'affectation a, qui me retourne 1. Et ensuite, je stocke ce 1 dans b, ce qui me donne cette situation. Donc on voit bien qu'en programmation, dans les deux cas, a et b, ont les mêmes valeurs, mais ce ne sont pas les mêmes valeurs au cours du

notes

résumé

1m 1s

EPFL

Attention: Ne confondez pas une affectation avec une égalité mathématique.

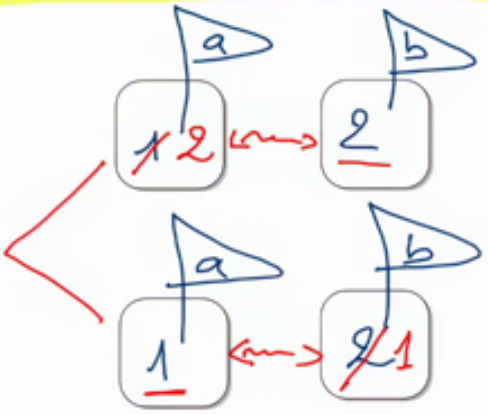
Toutes les deux utilisent le signe égal =, mais l'affectation est un mécanisme dynamique.

Par exemple, les deux instructions:

```
a = b;
b = a;
```

copie la valeur de b dans a

copie la valeur de a dans b



ne sont pas identiques.

temps.

notes

résumé

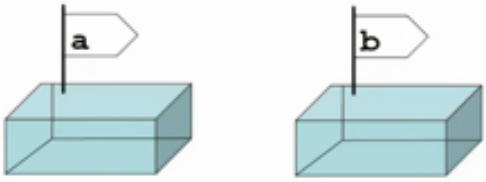
En mathématiques:

$$b = a + 1$$

signifie que tout au long des calculs, a et b vérifieront toujours cette relation. Autrement dit, quel que soit a, b sera toujours égal à $a+1$

En Java:

```
a = 5;
b = a + 1;
a = 2;
```



Qui plus est, si plus tard, l'une des deux variables est modifiée, l'autre ne l'est pas. Par exemple, si a est ensuite modifié en 3, b garde sa propre valeur, par exemple 1, dans la seconde affectation. Autre exemple, si en mathématiques,

notes

résumé

2m 49s

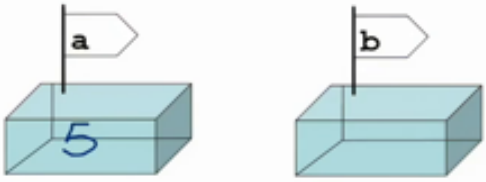
En mathématiques:

$b = a + 1$

signifie que tout au long des calculs, a et b vérifieront toujours cette relation.
Autrement dit, quel que soit a, b sera toujours égal à $a+1$

En Java:

```
a = 5;
b = a + 1;
a = 2;
```



je rédige une relation de cette nature, je signifie que a et b vont toujours vérifier cette relation au cours du temps. Par contre, si j'écris strictement la même chose en programmation, on verra que l'incidence sur a et b est différente. Donc ici, je commence par exécuter cette instruction, ce qui aura pour issue de stocker 5 dans a. Ensuite j'exécute cette instruction d'affectation,

notes

résumé

3m 1s

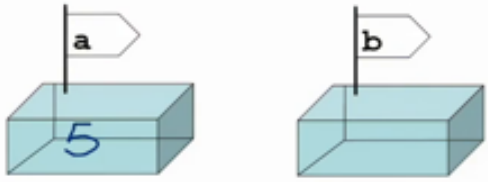
En mathématiques:

$b = a + 1$

signifie que tout au long des calculs, a et b vérifieront toujours cette relation.
Autrement dit, quel que soit a, b sera toujours égal à $a+1$

En Java:

```
a = 5;
b = a + 1;
a = 2;
```



comme on l'a vu tout à l'heure, on va donc évaluer a + 1,

notes

résumé

3m 25s

EPFL

On peut écrire aussi des affectations telles que:

```
a = a + 1;
```

Ce type d'affectation, où une variable apparaît de chaque côté du signe = permet de résoudre de nombreux problèmes.

Cette affectation signifie:

« calculer l'expression de `a + 1` et ranger le résultat dans `a`. Cela revient à augmenter de `1` la valeur de `a` »

Nous reviendrons sur ce point dans la suite.

qui nous produit 6, puis ensuite stocker ce 6 dans b, ce qui me produit ce résultat. A ce stade, on a effectivement que a et b respectent cette relation. Par contre, si je passe à l'instruction suivante, donc ici je vais modifier la valeur de a pour y stocker un 2, et à ce moment, on voit que la relation n'est plus respectée. En raison de ce que nous venons de voir, il est licite en programmation, d'écrire ce genre de tournure. C'est d'ailleurs une tournure très utile, que vous allez rencontrer de façon récurrente dans des exemples. Que se passe-t-il si j'exécute ce genre d'instructions ?

notes

résumé

3m 28s

Partons d'un cas concret, avec un a initialisé à 1. Comme nous l'avons vu tout à l'heure, on commence par évaluer cette partie de l'affectation, qui produit 2, et seulement dans un second temps, on va mettre ce 2 dans a, ce qui produit ce résultat. Donc lorsque j'écris ce genre de choses en programmation, je ne signifie pas que, comme en mathématiques, que la relation a = a + 1 doit toujours être respectée, ce qui peut me poser des problèmes en mathématiques, ici en l'occurrence. Je signifie simplement que je suis en train d'augmenter de 1 la valeur de a. d'augmenter de 1 la valeur de a.

notes

résumé

4m 1s