Support de cours

Cours:

Initiation à la programmation (en Java)

Vidéo:

Init-JAVA-03-4-Boucles-pt4

Concepts (extraits des sous-titres générés automatiquement) :

Nombre négatif. Nombre supérieur. Condition d'arrêt de la boucle. Exemple introductif. Nombre de notes. Intérieur du corps de la boucle. Langue naturelle. Façon adéquate. Nouvelle condition. Cas présent. Exemple de saisie d'un nombre. Fois des valeurs. Nombre limité d'essais. Petit programme. Passage d'une formulation.

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

vers la vidéo

EPFL

Boucles
(Partie 4)

Initiation à la programmation (Java)

Jamila Sam, Vincent Lepetit et Jean-Cédric Chappelier

...

notes

résumé

0m 0s

```java
int nombreDeNotes;

do {
  System.out.println("Entrez le nombre de notes");
  nombreDeNotes = clavier.nextInt();

} while(nombreDeNotes <= 0);
```

```
Entrez le nombre de notes:
-2
il faut entrer un nombre superieur a 0
Entrez le nombre de notes:
5
```

Examinons maintenant, comment il est possible d'étoffer un petit peu notre exemple introductif en améliorant un peu ses fonctionnalités. Par exemple ici, si l'utilisateur introduit un nombre négatif ou nul, nous aimerions bien pouvoir lui afficher un message lui signifiant que ce n'est pas ce que le programme attend. Donc par exemple, si l'utilisateur introduit -2, on aimerait pouvoir afficher : "il faut entrer un nombre supérieur à 0". Cela éviterait l'entêtement qu'on a vu dans le déroulement pas à pas, où l'utilisateur introduit plusieurs fois des valeurs qui ne sont pas souhaitées. Donc comment procéder dans le cas présent? Alors ici, très clairement, ce message, doit être affiché autant de fois que l'utilisateur introduit un nombre non voulu, c'est-à-dire potentiellement, autant de fois qu'il saisit une valeur. Donc il est assez naturel de penser tout de suite à placer les instructions en charge d'afficher ce message à l'intérieur du corps de la boucle. Donc typiquement, nous allons procéder comme suit : nous

notes

résumé

0m 1s

Comment trouver la condition ?

Supposons maintenant qu'on veuille limiter le nombre de notes à 10.

On veut toujours qu'il soit supérieur à 0.

Comment trouver la nouvelle condition ?

On veut répéter la boucle **tant que** le nombre de notes est incorrect,

allons enrichir le corps de la boucle de nouvelles instructions. Donc une fois que l'utilisateur a saisi le nombre de notes, nous allons tester si ce nombre de notes correspond aux critères souhaités, et si cela n'est pas le cas, il faudra afficher le message que nous voulons transmettre à l'utilisateur. Une difficulté courante lorsqu'on commence à écrire des boucles "while" et "do while" est sur comment formuler la condition d'arrêt de la boucle de façon adéquate. Nous allons examiner sur quelques exemples simples, comment procéder. Toujours dans notre exemple de saisie d'un nombre de notes, nous imposions jusqu'ici à l'utilisateur qu'il introduise un nombre supérieur à zéro, et nous souhaitons maintenant faire en sorte que ce nombre aussi, ne dépasse pas dix notes. Donc nous ne voulons pas qu'il introduise un nombre plus grand que dix. Donc nous allons nous intéresser ici, à comment trouver cette nouvelle condition. Le plus facile pour formuler la condition, est sans doute d'essayer de l'exprimer en langue naturelle. Que veut-on ici exactement? Nous voulons répéter les traitements, donc, qui demandent à l'utilisateur d'introduire un nombre de notes, tant que ce nombre de notes est incorrect.

notes

résumé

1m 1s

Supposons qu'on veuille écrire un programme qui demande à l'utilisateur de deviner un nombre. Pour simplifier, nous supposerons que le nombre à deviner est toujours 5.

Le programme peut s'écrire ainsi:

```
int nombreADeviner = 5;
int nombreEntre;

do {
  System.out.println("Entrez un nombre entre 1 et 10");
   nombreEntre = clavier.nextInt();
} while( condition ? );

System.out.println("Trouve");
```

Or, ici, quand ce nombre de notes est-il incorrect? Il peut l'être dans deux situations, soit que l'utilisateur a introduit quelque chose qui ne satisfait pas ce premier critère, soit qu'il a introduit un nombre de notes, qui ne satisfait pas, ce second critère. Donc, nous nous trouvons dans la situation où le nombre de notes introduit est incorrect, dans le cas où l'utilisateur a introduit quelque chose qui est inférieur ou égal à zéro, ou qu'il a introduit quelque chose qui est supérieur à dix. Et ceci peut très naturellement se transcrire en Java. Donc, nous allons reprendre notre première phrase ici, s'il est inférieur ou égal à zéro, c'est exactement ce que nous formulons là, OU, nous trouvons notre OU ici, s'il est supérieur à dix, c'est ce que nous formulons ici. Donc, nous voyons ici, que le passage d'une formulation en langue naturelle à Java, peut se faire de façon très naturelle, une fois qu'on a clairement en tête, ce que l'on veut tester. Dans le même esprit, imaginons maintenant, que l'on souhaite écrire un petit programme qui a pour but de faire deviner un nombre inconnu à un utilisateur. Donc, ici, pour simplifier la chose, nous allons partir de l'idée qu'il doit deviner le nombre cinq.

notes

résumé

2m 1s

la boucle doit être répétée
tant que l'utilisateur n'a pas trouvé le nombre à deviner, c'est-à-dire

```
    nombreEntre = clavier.nextInt();
} while( condition ? );

System.out.println("Trouve");
```

Donc, l'idée est de répéter la saisie du nombre, tant que l'utilisateur n'a pas trouvé le nombre à deviner. Donc la question que nous posons toujours est: comment formuler la condition? Alors, comme nous avons fait dans l'exemple précédent, nous allons commencer par le formuler en langue

notes

résumé

3m 13s

Supposons qu'on veuille en plus limiter le nombre d'essais à 3.
On peut ajouter une variable qui va compter le nombre d'essais utilisés:

```
int nombreADeviner = 5;
int nombreEntre;
int nombreEssais = 0;

do {
  System.out.println("Entrez un nombre entre 1 et 10");
   nombreEntre = clavier.nextInt();
   ++nombreEssais;
} while( condition ? );

System.out.println("Trouve");
```

Comment modifier la condition pour que la boucle s'arrête quand le nombre d'essais dépasse 3 ?

naturelle : la boucle doit être répétée, tant que l'utilisateur n'a pas trouvé le nombre à deviner. Concrètement, tant que le nombre qu'il a entré, est différent du nombre à deviner. Transcrivons maintenant cette formulation en langage courant, en Java : tant que le nombre entré est différent du nombre à deviner, s'écrit en Java, comme ceci. Encore un exemple où le passage de la formulation en langue courante à Java, se fait de façon tout à fait naturelle. Sophistiquons maintenant un peu ce même exemple, imaginons que l'on souhaite toujours faire deviner un nombre à l'utilisateur, mais qu'on souhaite lui donner un nombre limité d'essais, donc s'il n'a pas trouvé au bout de tant de tentatives, on arrête, on lui dit qu'il a perdu. Donc comment modifier la condition, pour, par exemple que la boucle s'arrête si l'utilisateur dépasse trois tentatives.

notes

résumé

3m 26s

```
int nombre_a_deviner = 5;
int nombre_entre;
int nombre_essais = 0;
boolean trouve = false;
do {
  System.out.println("Entrez un nombre entre 1 et 10");
  nombre_entre = clavier.nextInt();
  ++nombre_essais;
} while(nombre_entre != nombre_a_deviner && nombre_essais < 3);
```

trouve = (nombre_entre == nombre_a_deviner);

!trouve

Si on veut afficher un message pour indiquer à l'utilisateur s'il a trouvé le nombre ou si il a épuisé ses essais, on peut ajouter après la boucle:

```
if (nombre_entre == nombre_a_deviner) {
  System.out.println("Trouve");
} else {
  System.out.println("Perdu. Le nombre etait " + nombre_a_deviner);
}
```

Comment formuler cette condition? Alors ici, l'idée est d'avoir recours naturellement, à un compteur d'essais, nous allons compter le nombre de tentatives, ce compteur, va s'incrémenter à chaque itération de la boucle, et donc nous allons essayer de formuler la condition, aussi, dépendamment de ce compteur. Toujours le même principe, on essaie de formuler cette condition en langue naturelle: on veut répéter le traitement tant que l'utilisateur n'a pas trouvé le nombre à deviner, et qu'il lui reste des essais. Donc, on affine un petit peu la description, donc en clair que veut dire cette condition? On itère, on répète les traitements, tant que le nombre entré est différent du nombre à deviner, et que, le nombre d'essais est inférieur à trois. Si j'essaye de transcrire en Java, cela se fait de façon toujours aussi naturelle. Donc ici, ma première condition: tant que le nombre entré est différent du nombre à deviner, va s'écrire tout simplement comme ceci. Mon "et", va s'écrire comme ceci, et ma seconde condition va pouvoir être rédigée sous cette forme-là, en Java. Notre utilisateur a désormais deux façons possibles de sortir de la boucle, il peut sortir de la boucle parce qu'il a trouvé le nombre à deviner, le nombre entré a la même valeur que le nombre à deviner. Dans ce cas-ci, l'évaluation de cette condition va retourner "false", comme l'expression globale implique un "et", et que, il suffit qu'un des arguments du "et" soit faux, pour que l'ensemble de l'expression retourne "false", et bien, on va sortir à ce moment-là de la boucle. Autre condition de sortie possible: l'utilisateur a effectué un nombre d'essais pour deviner le nombre, qui va dépasser ou devenir égal à trois. Donc on aimerait bien maintenant, sophistiquer un petit peu le programme, en lui indiquant, pour quelle raison il est sorti de cette boucle. Est-il sorti parce qu'il a trouvé le nombre, dans ce cas-là, on veut

notes

résumé

4m 13s

```
int nombre_a_deviner = 5;
int nombre_entre;
int nombre_essais = 0;
boolean trouve = false;
do {
  System.out.println("Entrez un nombre entre 1 et 10");
  nombre_entre = clavier.nextInt();
  ++nombre_essais;
  trouve = ( nombre_entre == nombre_a_deviner );
} while(!trouve && nombre_essais < 3);
```

Si on veut afficher un message pour indiquer à l'utilisateur s'il a trouvé le nombre ou si il a épuisé ses essais, on peut ajouter après la boucle:

```
if (nombre_entre == nombre_a_deviner) {
  System.out.println("Trouve");
} else {
  System.out.println("Perdu. Le nombre etait " + nombre_a_deviner);
}
```

lui signifier qu'il a gagné, ou est-il sorti en état d'échec? Il n'a pas trouvé le nombre en ayant dépassé le nombre de tentatives permises. Ce n'est en fait qu'une fois qu'on est sorti de la boucle qu'on peut essayer de voir pourquoi on en est sorti, donc on va placer l'instruction qui teste pourquoi on est sorti de la boucle, après la fin de la boucle "do while". Donc ici, on va simplement tester est-on sorti de la boucle, alors qu'on a trouvé le nombre à deviner, dans ce cas-là on peut signifier à l'utilisateur qu'il a bel et bien trouvé. Et bien, dans l'autre cas, il faudra lui dire simplement qu'il a perdu. Notez ici, que nous avons, à deux reprises, l'évaluation de la même condition : est-ce qu'on a trouvé le nombre ou est-ce qu'on ne l'a pas trouvé? Dans deux situations différentes, une fois dans le "if" et une fois dans le "while". Il aurait été judicieux, en guise de bonnes pratiques, de regrouper ces deux conditions en une seule variable. Par exemple ici, je pourrais déclarer, une variable booléenne "trouvee", initialisée à l'origine à "false", à l'intérieur de la boucle, donner pour valeur à la variable : "le nombre entré est le même que le nombre à deviner", et à ce moment-là, je peux remplacer cette condition-là, simplement par l'expression !trouvee. Tant que l'on n'a pas trouvé, et que le nombre d'essais est inférieur à trois, on continue à boucler.

notes

résumé

```
int nombreADeviner = 5;
int nombreEntre;
int nombreEssais = 0;

do {
  System.out.println("Entrez un nombre entre 1 et 10");
   nombreEntre = clavier.nextInt();
   ++nombreEssais;
} while(nombreEntre != nombreADeviner && nombreEssais < 3);
```

Attention, si on avait utilisé `nombreEssais < 3` comme condition:

```
if (nombreEssais < 3) {
  System.out.println("Trouve");
} else {
  System.out.println("Perdu. Le nombre etait " + nombreADeviner);
}
```

le programme afficherait "`Perdu. ...`" quand l'utilisateur trouve au troisième essai.

De même, on pourrait remplacer ici, cette condition, par l'évaluation de "if (trouve)". Et à ce moment-là on établit un lien entre deux conditions qui sont les mêmes, alors que à l'origine, ce lien n'existait pas, ce qui est moins bon.

notes

résumé

7m 25s

Alors il faut être attentif à la façon de formuler ces tests. On peut les formuler de façon maladroite et rencontrer, donc des erreurs d'exécution. Ici, par exemple, imaginez que pour tester qu'on a bien deviné le nombre, on teste que l'on n'a pas dépassé le nombre d'essais de cette façon-là, et bien, ce qui va se passer ici, c'est que le cas où l'utilisateur a trouvé le nombre au bout de trois tentatives, va simplement désormais passer pour un échec, puisque, cette condition-ci ne va pas être vérifiée, alors qu'on a quand même trouvé le nombre au bout de la troisième tentative, et donc on va signifier à l'utilisateur qu'il a perdu, alors qu'en réalité il a fini par trouver. Donc ceci pour dire qu'il est important de bien réfléchir à comment formuler la condition, et le fait de la formuler de façon explicite, comme tout à l'heure, par rapport au nombre à deviner, permet d'éviter l'écueil que nous venons de décrire dans l'exemple précédent. précédent.

notes

résumé

7m 38s