

Support de cours

Cours:

Introduction à la programmation orientée objet (en Java)

Vidéo:

W15-03-interfacesintro-JAVA-pt1

Concepts (extraits des sous-titres générés automatiquement) :

Classe abstraite. Besoin d'une méthode evolue. Joueur de cette classe abstraite. Notion d'interface. Méthode abstraite evolue. Travers de classes. Classe interactif. Interfaces. Héritage multiple. Contenu commun. Besoin de balles. Idée de l'utilité des interfaces. Concept d'interface. Super-classes. Représentation graphique.



[vers la recherche de séquences vidéo](#)

(dans Introduction à la programmation orientée objet (en Java).)



[vers la vidéo](#)

Center for Digital Education. Plus de matériel de soutien pédagogique ici :

<https://www.epfl.ch/education/educational-initiatives/cede/educational-technologies-gallery/boocs-en/>

Interfaces

(Partie 1)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s





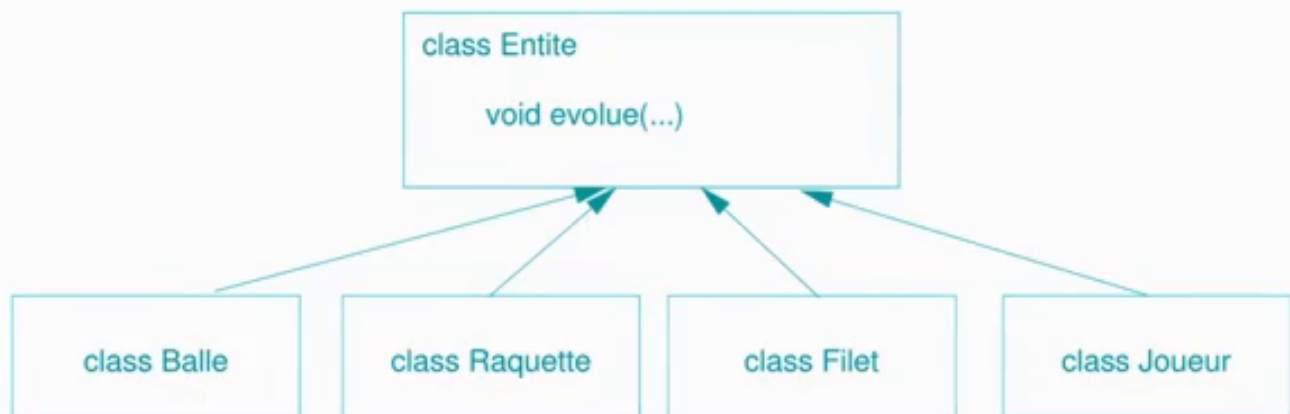
Dans cette séquence vidéo et la suivante

notes

résumé

0m 1s





Et supposons que chacune de ces entités ait besoin d'une méthode evolue pour les faire évoluer dans le jeu. Naturellement la façon de concevoir ceci en programmation orientée objets

notes

résumé

0m 21s



Si l'on analyse de plus près les besoins du jeu, on réalise que :

- ▶ certaines entités doivent avoir une représentation graphique
(Balle, Raquette, Filet)
- ▶ ... et d'autres non (Joueur)
- ▶ certaines entités doivent être interactives
(on veut par exemple pouvoir les contrôler avec la souris) :
Balle, Raquette
- ▶ ... et d'autres non : Joueur, Filet

🗨 Comment organiser tout cela ?

serait de faire une classe abstraite Entite qui aurait donc la méthode abstraite evolue et de faire hériter la balle, les raquettes, le filet, le joueur de cette classe abstraite Entite. Mais admettons que l'on pousse un peu plus loin l'analyse du jeu, et qu'on se rende compte que certaine de ces entités, donc par exemple, les balles les raquettes, les filets, devraient avoir une représentation graphique qu'on les voit effectivement, sur l'écran du jeu, et que par exemple, on ne voit pas le joueur. Donc à ce moment là on aurait à dissocier les joueurs des balles, des raquettes, et des filets du point de vue, d'avoir une représentation graphique. On pourrait aussi imaginer que certaines entités devraient être interactives, donc par exemple pour pouvoir les contrôler au clavier ou à la souris, comme donc une balle ou une raquette. Et d'autres que l'on ne pourrait pas faire bouger donc par exemple, le filet et le joueur.

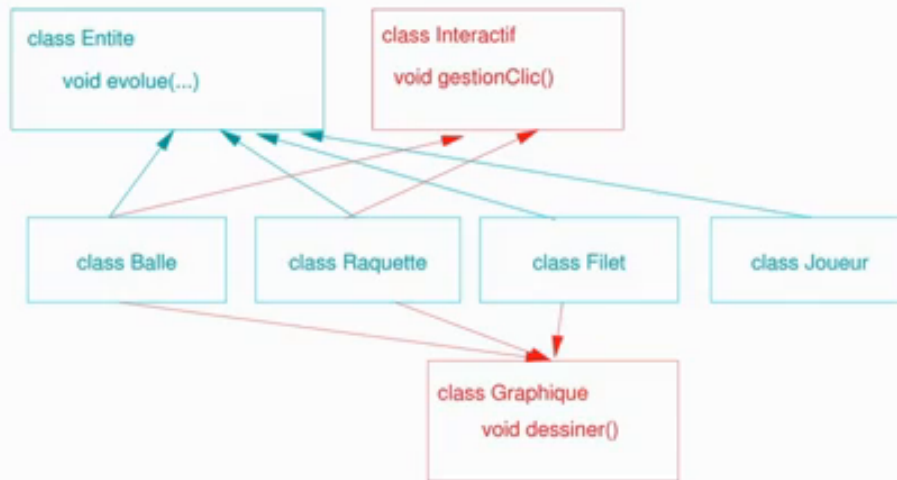
notes

résumé

0m 30s



Idealement, il nous faudrait mettre en place une hiérarchie de classes telle que celle-ci :



Mais ... **Java ne permet que l'héritage simple** : chaque sous-classe ne peut avoir qu'une seule classe parente directe !

Donc comment organiser tout ceci au niveau de notre conception orienté objet.

notes

résumé

1m 25s





Idéalement on ferait tout ça aux travers de classes avec donc, par exemple une classe interactif qui nous permettrait de gérer de façon interactive donc, la balle par exemple et la raquette. Une classe qui nous permettrait d'avoir une méthode pour dessiner graphiquement les objets graphiques dont hériteraient filet, raquette et balle. Mais ceci ferait donc ce que l'on appelle de l'héritage multiple. La balle hériterait de façon multiple d'une entité et d'interactif et de graphique, or en java on ne peut pas avoir d'héritage multiple en java on a que de l'héritage simple. Donc comment faire ? Avant d'y répondre, signalons simplement

notes

résumé

1m 32s





- ▶ Pourquoi pas d'héritage multiple en Java ?
 - ▶ Parfois difficile à comprendre (quel sens donner ?), y compris pour le compilateur (par exemple si une sous-sous-classe hérite d'une super-super-classe par différents chemins)
- ▶ Si une variable/méthode est déclarée dans plusieurs super-classes
 - ▶ Ambiguïté : laquelle utiliser, comment y accéder ?

que certains autres langages de programmation ont choisi d'avoir l'héritage multiple alors donc pourquoi il n'y a pas d'héritage multiple en java ? Simplement parce que il y a parfois des situations un peu difficile à comprendre, difficile de comprendre quel sens on veut vraiment donner, donc par exemple si on avait une classe, qui hérite multiplement d'un animal, est ce que c'est un animal ? Le même animal ? Ou est ce que c'est deux animaux ? Et ces situations sont aussi difficiles à comprendre pour le compilateur donc avec des décisions à prendre. Ceux qui voudraient s'en convaincre pourraient aller voir la séquence sur les classes virtuelles en c++ sur notre autre mooc qui se déroule en parallèle concernant le langage c++ et vous serez certainement convaincu.

notes

résumé

2m 9s



Mais en fait, que souhaitait-on utiliser de l'héritage multiple dans le cas de notre exemple de jeu vidéo ?

- ☛ **Le fait d'imposer à certaines classes de mettre en oeuvre des méthodes communes**

Par exemple :

- ▶ `Balle` et `Raquette` doivent avoir une méthode `gestionClic`;
 - ▶ mais `gestionClic` ne peut être une méthode de leur super-classe (car n'a pas de sens pour un `Joueur` par exemple).
- ☛ Imposer un contenu commun à des sous-classes en dehors d'une relation d'héritage est le rôle joué par la notion d'`interface` en Java.

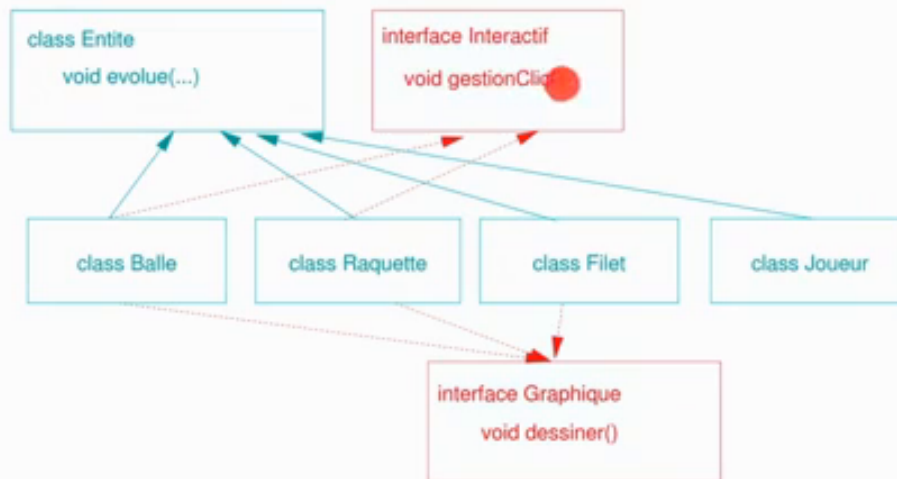
Une autre raison c'est qu'on peut avoir de l'ambiguïté lorsqu'une méthode ou une variable est déclarée dans plusieurs super-classes. Laquelle utiliser ? Comment y accéder ? Donc, pour de bonnes raisons, Java a décidé de ne pas avoir d'héritage multiple. Donc, comment faire pour bien concevoir en java, le jeu que nous avons imaginé jusqu'ici ?

notes

résumé

2m 49s





- Interface \neq Classe
- Une interface permet d'imposer à certaines classes d'avoir un contenu particulier sans que ce contenu ne fasse partie d'une classe.

Revenons pour cela à la raison qui nous avait poussé à introduire de l'héritage multiple, pourquoi voulions-nous de l'héritage multiple dans notre jeu vidéo ? Pour imposer à certaines classes de mettre en oeuvre des méthodes communes. Donc par exemple, on voulait que Balle et Raquette aient une méthode qui permette par exemple, de gérer un clique de souris. Mais la méthode gestionClic ne peut pas être une méthode pour leur super-classe, commune à tous pour la classe Entite parce qu'un joueur par exemple n'a pas de gestionClic. Donc ce que l'on veut c'est imposer un contenu commun à certaines sous-classes en dehors d'une relation d'héritage pour les différencier d'autres sous-classes, c'est ce qu'on appelle la notion justement d'interface en java. La notion d'interface en java qui est différente de la notion de classe, va nous permettre comme ça d'imposer à certaines classes d'avoir des contenus particuliers sans que ça fasse partie, à proprement parler, d'une classe donc par exemple ici les sous-classes Raquette et Balle, sous-classe de la classe Entite, vont en plus avoir l'interface, par exemple, Interaction ou Interactif

notes

résumé

3m 7s





qui permet donc de gérer, par exemple les interactions à la souris et il n'aurait été bien sûr, pas correct d'avoir cette méthode `gestionClic` dans la classe `Entite`, parce qu'alors le `filet` et le `joueur` auraient aussi eu une méthode `gestionClic` dont ils n'ont absolument rien à faire. Les sous-classes `balle`, `raquette` et `filet` de la classe `Entite` vont, en plus, avoir une interface qui les oblige à pouvoir se décider d'un point de vue graphique. Voilà donc pour le concept d'interface.

notes

résumé

4m 13s



Les interfaces jusqu'à Java 7 ne peuvent contenir que :

- ▶ des constantes
- ▶ des méthodes abstraites

Nouveautés depuis Java 8, elles peuvent aussi contenir :

1. des définitions par défaut pour les méthodes
2. des méthodes statiques

Maintenant que vous avez une idée de l'utilité des interfaces voyons à quoi cela correspond concrètement en java. Avant de poursuivre sur le contenu des interfaces,

notes

résumé

4m 44s



Syntaxe :

```
interface UneInterface { constantes ou méthodes abstraites }
```

 (Java < 8)

Exemple :

```
interface Graphique {  
    void dessiner();  
}  
interface Interactif {  
    void gestionClic();  
}
```

Il ne peut y avoir de constructeur dans une interface

👉 Impossible de faire `new` !

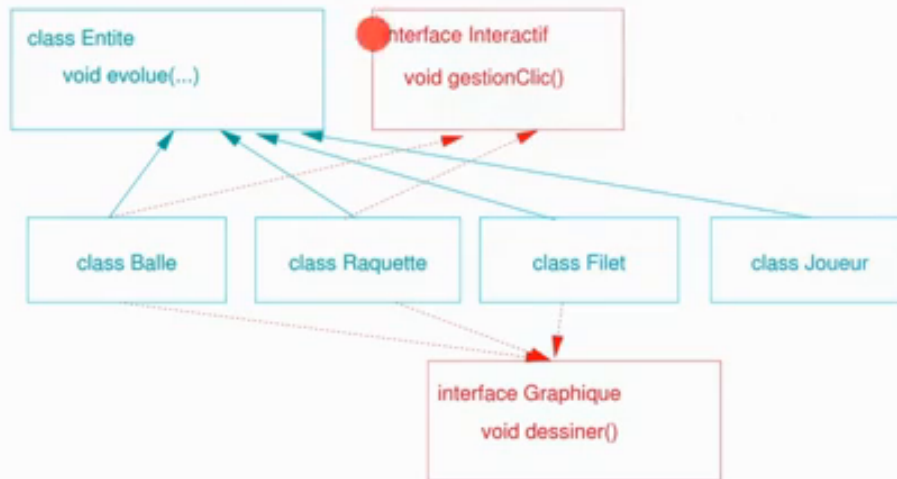
il convient de faire une remarque importante. Les interfaces jusqu'à Java 7, comme vous aurez l'occasion de le voir dans les transparents qui suivent, ne peuvent contenir que des constantes et des méthodes abstraites. De nombreuses nouveautés ont cependant été ajoutées depuis Java 8. Dans les transparent qui suivent, nous allons essentiellement exposer le contenu des interfaces telles que classiquement défini jusqu'à Java 7. Les nouveautés de Java 8 feront, elles, l'objet d'une nouvelle vidéo. En java, une interface se déclare un peu comme une classe à la différence près que on va remplacer le mot réservé classe par le mot réservé interface suivi du nom de l'interface, librement choisi, puis, une paire d'accolades ouvrantes et fermantes un peu comme pour le corps d'une classe. A la différence d'une classe, une interface ne peut comporter que des méthodes abstraites, ce qui est le cas le plus courant, ou encore, des constantes.

notes

résumé

4m 50s





- Interface \neq Classe
- Une interface permet d'imposer à certaines classes d'avoir un contenu particulier sans que ce contenu ne fasse partie d'une classe.

Voici donc à quoi pourrait correspondre le code des interfaces de notre petit exemple d'introduction. Vous vous demandez sans doute ici pourquoi les méthodes définies à l'intérieur des interfaces sont définies sans le mot réservé `abstract` alors que nous venons de voir qu'une interface ne contient que des méthodes abstraites, nous aurons l'occasion d'y revenir dans un petit moment.

notes

résumé

5m 49s



Syntaxe :

```
interface UneInterface { constantes ou méthodes abstraites }
```

Exemple :

```
abstract interface Graphique {
public void dessiner();
}
interface Interactif {
    void gestionClic();
}
```

~~Graphique g = new (-);~~

Il ne peut y avoir de constructeur dans une interface

❗ Impossible de faire `new` !

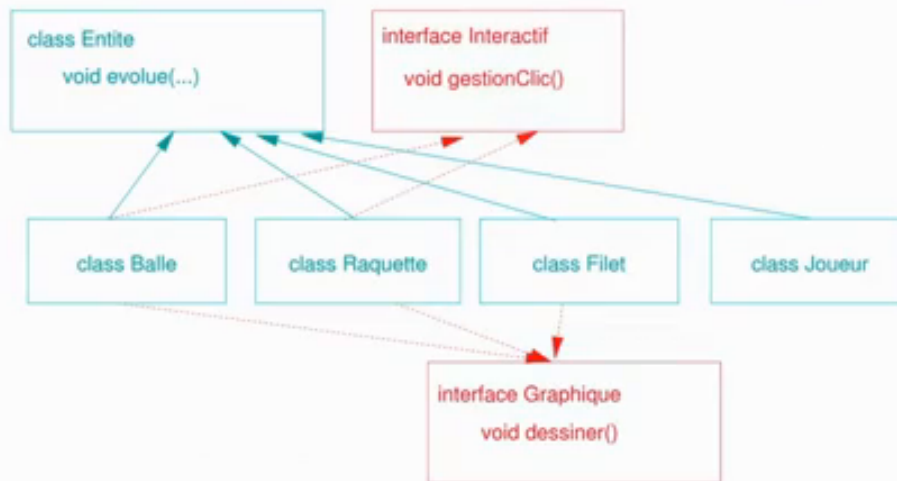
Pour rappel, l'idée était que ces interfaces imposent un certain contenu aux classes avec lesquelles elles sont en relation. Et ici le contenu imposé était typiquement un certain nombre de méthodes. Une interface, ne pouvant contenir que des méthodes abstraites, ne peut pas contenir de constructeur il est donc impossible de créer une instance d'interface. Donc ici par exemple, je ne peux pas écrire dans un programme quelque chose qui a cette allure, donc, déclarer un objet graphique et essayer de l'instancier par une tournure de cette nature. Ceci est évidemment, impossible. Mais revenons à la question qui nous a préoccupé tout à l'heure : pourquoi sommes nous ici dispensés de mettre explicitement le mot réservé `abstract`, par exemple ? Comme les méthodes d'une interface sont nécessairement abstraites, java vous dispense de le mentionner explicitement. Ce qu'il faut savoir, c'est que toute méthode déclarée dans une interface est nécessairement abstraite et nécessairement publique. Si vous essayez de déclarer une méthode d'interface

notes

résumé

6m 10s





- Interface \neq Classe
- Une interface permet d'imposer à certaines classes d'avoir un contenu particulier sans que ce contenu ne fasse partie d'une classe.

avec un autre modificateur que public vous aurez droit à un message d'erreur du compilateur. Il est également possible de faire figurer dans une interface, des constantes, même si cela est moins courant que les méthodes abstraites. Par exemple, je pourrais définir une interface comme ceci qui contient donc des constantes, comme ceci, et dans ce cas là, je dois donner des valeurs aux constantes puisqu'il n'y a pas de constructeur éventuel qui pourrait attribuer des valeur initiales à ces constantes, je suis obligée de le faire à ce stade et comme pour les méthodes, il y a aussi des modificateurs implicites pour les constantes, toute constante que vous mettez dans une interface est nécessairement public, final, et static.

notes

résumé

7m 13s



Attribution d'une interface à une classe :

Syntaxe :

```
class UneClasse implements Interface1, ... , InterfaceN  
{ ... }
```

Exemple :

```
class Filet extends Entite implements Graphique {  
    public void dessiner() { ... }  
}
```

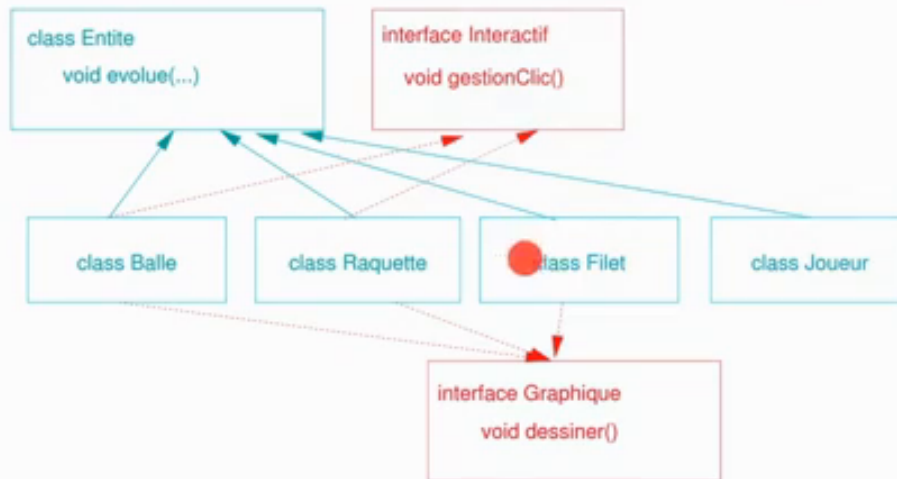
Nous savons maintenant ce que l'on peut mettre à l'intérieur d'une interface, intéressons nous à comment établir le lien entre l'interface et une classe.

notes

résumé

7m 59s





- Interface \neq Classe
- Une interface permet d'imposer à certaines classes d'avoir un contenu particulier sans que ce contenu ne fasse partie d'une classe.

Ce lien s'établit au moyen d'un mot réservé particulier : le mot réservé implements lorsque l'on déclare la classe on va indiquer que cette classe implémente un certain nombre d'interfaces. Il peut y en avoir plusieurs, séparées par des virgules.

notes

résumé

8m 9s



Attribution d'une interface à une classe :

Syntaxe :

```
class UneClasse implements Interface1, ... , InterfaceN  
{ ... }
```

Exemple :

```
class Filet extends Entite implements Graphique {  
    public void dessiner() { ... }  
}
```

Par exemple, pour exprimer le fait que notre classe filet étend la classe Entite mais implémente l'interface Graphique, ce qui va nous permettre de voir ce filet comme un objet dessinable,

notes

résumé

8m 24s



Une classe peut implémenter plusieurs interfaces (mais étendre une seule classe)

- Séparer les interfaces par des virgules

Exemple :

```
class Balle extends Entite implements Graphique, Interactif {  
    // code de la classe  
}
```

On peut déclarer une hiérarchie d'interfaces :

- Mot-clé `extends`
- La classe qui implémente une interface reçoit aussi le type des super-interfaces

```
interface Interactif { ..}  
interface GerableParSouris extends Interactif { ... }  
interface GerableParClavier extends Interactif { ... }
```

on va écrire en java, la classe Filet étend la classe Entite et implémente l'interface Graphique. Lorsqu'une classe implémente un certain nombre d'interfaces il est impératif, si on veut qu'elle soit instantiable qu'elle redéfinisse toutes les méthodes déclarées dans les interfaces. Toutes les méthodes abstraites doivent être redéfinies concrètement dans la classe qui les implémente, pour que celles-ci soient instanciables. C'est de cette façon qu'une interface impose un contenu à une classe qui l'implémente, dès l'instant où on établit le lien entre la classe et l'interface, alors la classe, si on veut pouvoir l'instancier doit redéfinir toutes les méthodes spécifiées dans l'interface. Donc comme nous l'avons vu,

notes

résumé

8m 37s



Une classe peut implémenter plusieurs interfaces (mais étendre une seule classe)

- ▶ Séparer les interfaces par des virgules

Exemple :

```
class Balle extends Entite implements Graphique, Interactif {  
    // code de la classe  
}
```

On peut déclarer une hiérarchie d'interfaces :

- ▶ Mot-clé `extends`
- ▶ La classe qui implémente une interface reçoit aussi le type des super-interfaces

```
interface Interactif { ..}  
interface GerableParSouris extends Interactif { ... }  
interface GerableParClavier extends Interactif { ... }
```

une classe peut parfaitement implémenter plusieurs interfaces, par exemple, la balle est un objet cliquable et est un objet dessinable, donc elle implémente deux interfaces, l'interface Interactif et l'interface Graphique. Si on veut pouvoir créer des instances de bases ce qui est probablement le cas, alors la classe Balle doit impérativement donner une définition concrète de la méthode gestionClic et de la méthode dessiner . Notez qu'en java, il est tout à fait possible de déclarer une hiérarchie d'interfaces. On dira qu'une interface en étend une autre et ceci s'exprime au moyen du mot réservé extends comme pour les classes.

notes

résumé

9m 19s



Attribution d'une interface à une classe :

Syntaxe :

```
class UneClasse implements Interface1, ... , InterfaceN  
{ ... }
```

Exemple :

```
class Filet extends Entite implements Graphique {  
    public void dessiner() { ... }  
}
```

Par exemple ici nous avons deux interfaces GerableParSouris et GerableParClavier qui étendent toutes deux une super-interface Interactif. Nous avons bel et bien ici, une hiérarchie d'interfaces avec une super-interface Interactif, et deux sous-interfaces, GerableParSouris et GerableParClavier.

notes

résumé

10m 1s



Avant de poursuivre, revenons à notre premier exemple de classe implémentant une interface et ce, pour mentionner un point, qui peut avoir son importance. Nous avons vu précédemment que dans une interface les méthodes étaient de facto, publiques. Que se passe-t-il si une classe qui implémente une interface, définit la méthode présente dans l'interface mais avec des droits différents que le droit public ? Donc imaginez par exemple qu'ici je mette le droit d'accès `protected`. Et bien ceci sera tout bonnement refusé par le compilateur parce qu'en java, lorsque vous redéfinissez une méthode existante vous avez le droit d'élargir les droits mais jamais de les restreindre. Notez que ceci est également valable pour la redéfinition dans le cadre de l'héritage. Si j'ai par exemple une super-classe A dont hérite une sous-classe B, si A définit une méthode `m` comme `public` alors la redéfinition de cette méthode ne peut pas restreindre les droits. Ceci sera refusé. Ceci sera refusé.

notes

résumé

10m 27s

