

Support de cours

Cours:

Introduction à la programmation orientée objet (en Java)

Vidéo:

W14-01-polymabstract-JAVA-pt2

Concepts (extraits des sous-titres générés automatiquement) :

Méthode abstraite. Classe abstraite. Entête du mot. Façon générale. Mot clé. Classe. Début de l'entête de la classe. Sous-classes concrètes de figures. Périmètre d'une figure. Méthode. Niveau de la classe. Nouveau guerrier. Exemple du jeu. Constructeur de figures fermées. Nouveau jeu.



[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/>

Classes et méthodes abstraites

(Partie 2)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s



Une méthode *abstraite*, est *incomplètement spécifiée* :

- ▶ elle n'a *pas de définition* dans la classe où elle est introduite (pas de corps)
- ▶ elle sert à imposer aux sous-classes (non abstraites) qu'elles **doivent définir** la méthode abstraite héritée
- ▶ elle doit être contenue dans une classe abstraite *abstraite*

Syntaxe :

```
abstract Type nom_methode(liste d'arguments);
```

Exemple :

```
abstract class Personnage {  
    // ...  
    public abstract void afficher();  
    // ...  
}
```

Voyons maintenant comment tout ceci s'écrit concrètement en java. Pour avoir une méthode abstraite en java on fait simplement précéder son entête du mot réservé « abstract » et on termine cet entête tout simplement avec un point virgule (;) sans mettre de corps, sans mettre de définition parce qu'une méthode abstraite n'a pas de définition dans la classe où elle a été introduite, elle sert simplement à imposer aux sous-classes que l'on ne veut plus être abstraites, on reviendra sur ce point dans un instant, de redéfinir justement cette méthode abstraite héritée et elle doit être contenue dans une classe abstraite sur laquelle nous allons revenir dans un instant, et une classe abstraite c'est une classe qui aura aussi le mot clé « abstract » dans son entête.

notes

résumé

0m 1s



```
abstract class FigureFermee {  
    public abstract double surface();  
    public abstract double perimetre();  
  
    // On peut utiliser une méthode abstraite :  
    public double volume(double hauteur) {  
        return hauteur * surface();  
    }  
}
```

Figure Fermee
↗

Donc par exemple ici, la classe « Personnage » on veut que ce soit une classe abstraite donc on va rajouter le mot clé « abstract » comme ça dans l'entête, au début de l'entête de la classe et elle contiendra donc ici, une méthode abstraite, on a rajouté donc ici « abstract afficher » par exemple. Si je reprends donc notre autre exemple avec les figures fermées je vous rappelle que l'idée c'était d'avoir des figures fermées pour définir une méthode « surface » mais nous ne savons pas définir cette méthode « surface » de façon générale au niveau de la classe « FigureFermee » et donc cette méthode « surface » devient une méthode abstraite et la classe « FigureFermee » devient une classe abstraite et on imagine donc avoir des sous-classes concrètes

notes

résumé

0m 49s



```
abstract class FigureFermee {  
    public abstract double surface();  
    public abstract double perimetre();  
  
    // On peut utiliser une méthode abstraite :  
    public double volume(double hauteur) {  
        return hauteur * surface();  
    }  
}
```

Figure Fermee
↑
Cercle ...

de figures fermées comme par exemple un cercle. Et donc la classe « FigureFermee » c'est une classe abstraite

notes

résumé

1m 37s



```
abstract class FigureFermee {  
    public abstract double surface();  
    public abstract double perimetre();  
  
    // On peut utiliser une méthode abstraite :  
    public double volume(double hauteur) {  
        return hauteur * surface();  
    }  
}
```

Figure Fermee
Cercle

on rajoute le mot clé « abstract » ici qui contient donc deux méthodes, on en a rajouté une autre de plus, ici, abstraite. La méthode « surface » dont nous parlions tout au début dans notre exemple à laquelle on a donc rajouté le mot clé « abstract » et, admettons, on pourrait aussi imaginer

notes

résumé

1m 44s



```
abstract class FigureFerme {
    public abstract double surface();
    public abstract double perimetre();

    // On peut utiliser une méthode abstraite :
    public double volume(double hauteur) {
        return hauteur * surface();
    }
}
```

Figure Ferme
Cercle

qu'on a une méthode « perimetre »

notes

résumé

1m 59s



```
abstract class FigureFermee {  
    public abstract double surface();  
    public abstract double perimetre();  
  
    // On peut utiliser une méthode abstraite :  
    public double volume(double hauteur) {  
        return hauteur * surface();  
    }  
}
```

Figure Fermee
↑
Cercle ...

qui renvoie le périmètre d'une figure fermée et on ne sait pas plus le définir de façon générale au niveau des figures fermées donc c'est aussi une méthode abstraite. Je rappelle que l'on peut utiliser, dans une classe abstraite une méthode abstraite dans une méthode non-abstraite, donc par exemple ici, le calcul du volume engendré par la surface de la figure fermée sur une hauteur donnée en paramètre ici

notes

résumé

2m 2s



Une classe abstraite est une classe désignée comme telle au moyen du mot réservé `abstract`.

- ▶ Elle *ne peut être instanciée*
- ▶ Ses sous-classes *restent abstraites* tant qu'elles ne fournissent pas les définitions de *toutes les méthodes abstraites* dont elles héritent.

~~Figure Fermée~~ `fig = new ... ;`

Un exemple « concret »...

serait le produit de la hauteur par la valeur retournée par la méthode abstraite ici « surface », ceci est tout à fait possible. Une classe abstraite donc, qualifiée par le mot clé « `abstract` » au début de son entête : « `abstract class quelque-chose` » est une classe qui ne peut pas être instanciée et qui contient au moins, une méthode abstraite, c'est pour ça qu'on appelle ceci une classe abstraite par ce que on ne peut pas créer d'instance donc par exemple, si je voulais écrire « `FigureFermée` », déclarer une instance de figure fermée donc par exemple, « `fig = new ...` » et puis avec un constructeur de figures fermées, je ne peux pas faire ceci, le compilateur va m'interdire de faire ceci

notes

résumé

2m 25s



Une autre équipe crée la sous-classe **Guerrier** de **Personnage** et veut l'utiliser :

```
Jeu jeu = new Jeu();  
jeu.ajouterPersonnage(new Guerrier(...));
```

S'ils ont oublié de définir la méthode **afficher**,
le code ci-dessus génère une erreur de compilation
car on ne peut pas créer d'instance de **Guerrier** :

Guerrier is abstract; cannot be instantiated

parce que « **FigureFermee** » est une classe abstraite : « **abstract class** » et donc je ne peux pas créer d'instance. Et les sous-classes d'une classe abstraite restent abstraites tant qu'elles ne redéfinissent pas toutes les méthodes abstraites, c'est à dire, elles restent abstraites tant qu'il reste au moins une méthode abstraite héritée qui n'a pas été définie. Prenons un exemple, je reviens à l'exemple du jeu et des personnages, « **Personnage** » est une classe abstraite qui contient une méthode abstraite « **afficher** » et donc « **Guerrier** » est une sous-classe de « **Personnage** » et imaginons que « **Guerrier** » oublie de redéfinir la méthode abstraite « **afficher** » de « **Personnage** ». Alors, si je fais la chose suivante : donc créer un nouveau jeu et essayer d'ajouter un nouveau guerrier au jeu, si « **Guerrier** » oublie de redéfinir la méthode abstraite de « **Personnage** »

notes

résumé

3m 13s



Classes abstraites : autre exemple

abstract Figure Ferm
abstract perimetre abstract Su EPFL

```
class Cercle extends FigureFermee {  
    private double rayon;  
  
    public double surface() {  
        return Math.PI * rayon * rayon;  
    }  
    public double perimetre() {  
        return 2.0 * Math.PI * rayon;  
    }  
}
```

Cercle n'est pas une classe abstraite

```
class Polygone extends FigureFermee {  
    private ArrayList<Double> cotes;  
  
    public double perimetre() {  
        double p = 0.0;  
        for (Double cote : cotes) {  
            p += cote;  
        }  
        return p;  
    }  
}
```

Polygone reste par contre une classe abstraite

alors « Guerrier » reste une classe abstraite et je ne pourrais pas, comme ça, créer une instance de cette classe abstraite « Personnage ». Si j'essaye de faire ceci, j'aurais le message : « Guerrier is abstract ; cannot be instantiated ». Je ne peux pas créer d'instance de la classe abstraite « Guerrier » qui est restée abstraite parce qu'elle n'a pas redéfini la méthode « afficher ». Si je prends un second exemple sur les figures fermées, imaginons qu'on a toujours cette classe abstraite « FigureFermee » avec les méthodes abstraites « surface » et... (sans paroles) « perimetre » qui est aussi abstraite et donc supposons qu'on fasse une classe comme ça, « Cercle » qui hérite de « FigureFermee ». Cette classe « Cercle » redéfinit

notes

résumé

4m 1s



Classes abstraites : autre exemple

abstract Figure Ferm
abstract perimetre abstract Su EPFL

```
class Cercle extends FigureFermee {  
    private double rayon;  
  
    public double surface() {  
        return Math.PI * rayon * rayon;  
    }  
    public double perimetre() {  
        return 2.0 * Math.PI * rayon;  
    }  
}
```

Cercle n'est pas une classe abstraite

Cercle c1 = new ...

```
class Polygone extends FigureFermee {  
    private ArrayList<Double> cotes;  
  
    public double perimetre() {  
        double p = 0.0;  
        for (Double cote : cotes) {  
            p += cote;  
        }  
        return p;  
    }  
}
```

Polygone reste par contre une classe abstraite



la méthode « surface » ici, de façon concrète de façon non-abstraite -- par exemple, π fois le rayon au carré -- et redéfinit aussi la méthode « perimetre » de façon concrète. Elle a donc redéfini les deux méthodes abstraites héritées de sa super-classe abstraite « FigureFermee » et donc, cette classe « Cercle » n'est plus une classe abstraite, on peut créer des instances de « Cercle », ceci est maintenant possible puisque la classe « Cercle » n'est plus abstraite. Par contre, imaginons donc une classe « Polygone » qui hérite aussi de la classe « FigureFermee » mais qui ne redéfinit que, par exemple, le périmètre, en supposant qu'on sait calculer le périmètre d'un polygone comme étant la somme des longueurs de ses différents cotés

notes

résumé

4m 49s



Classes abstraites : autre exemple

abstract Figure Ferm
abstract perimetre abstract Su EPFL

```
class Polygone extends FigureFermee {  
    private ArrayList<Double> cotes;  
  
    public double perimetre() {  
        double p = 0.0;  
        for (Double cote : cotes) {  
            p += cote;  
        }  
        return p;  
    }  
}
```



Polygone reste par contre une classe abstraite

~~Polygone p2 = new ---~~

donc ça on sait parfaitement le définir mais on ne saurait pas forcément définir de façon générale la surface d'un polygone, et donc la classe « Polygone » ne redéfinit que la méthode abstraite « perimetre » mais ne redéfinit pas la méthode abstraite « surface » et donc la classe « Polygone » reste une classe abstraite. Je ne peux pas créer d'instance de « Polygone ». Ceci n'est pas possible parce que « Polygone » est une classe abstraite puisqu'elle n'a pas redéfini la méthode abstraite « surface ».

notes

résumé

5m 37s





D'ailleurs il manque un petit quelque chose pour que le code de « Polygone » donné ici soit correct, sauriez-vous dire quoi ? Voilà, ceci conclut cette séquence sur les méthodes abstraites et les classe abstraites. La prochaine séquence s'intéressera à différents compléments sur le polymorphisme, les constructeurs, la super-classe « Object », etc. la super-classe « Object », etc.

notes

résumé

6m 10s

