ISEN ALL IS DIGITAL! OUEST | yncréa

Java

Classes abstraites, Interfaces

Les classes abstraites

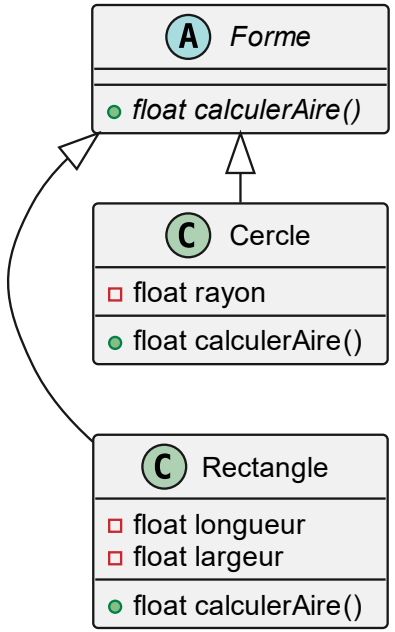
Classe abstraite

- Une classe abstraite est une classe dont la définition est incomplète : certaines méthodes, qualifiées d'abstraites, n'ont pas de corps.
- Le corps est fourni par des sous classes, par surcharge.
- Par conséquent une classe abstraite :
  - est forcément une super classe
  - ne peut pas être instanciée
- Les éléments abstraits sont identifiés avec le mot clé abstract

Pourquoi certaines classes sont abstraites ?

- L'implémentation des méthodes abstraites nécessite des informations spécifiques (notamment des attributs) qui ne sont pas présentes dans la classe abstraite.
- Exemple : Je peux calculer l'aire d'une forme, la classe `Forme` possède donc une méthode `calculerAire()`, mais son implémentation dépend du type de forme :
  - Pour un cercle, `calculerAire()` renverrait $\pi \times \text{rayon}^2$
  - Pour un rectangle, `calculerAire()` renverrait longueur x largeur
  - Mais pour une forme en général, on ne peut rien dire !

Exemple



Italique = abstrait

```
public abstract class Forme
{
        ...
        public abstract float calculerAire();
}
```

```
public class Cercle extends Forme
{
        ...
        public float calculerAire(){
                return Math.PI * this.rayon * this.rayon;
        }
}
```

```
public class Rectangle extends Forme
{
        ...
        public
        float calculerAire(){
                return this.longueur * this.largeur;
        }
}
```

Pourquoi certaines classes sont abstraites ?

- L'implémentation des méthodes abstraites nécessite des informations spécifiques (notamment des attributs) qui ne sont pas présentes dans la classe abstraite.
- Exemple : Je peux calculer l'aire d'une forme, la classe `Forme` possède donc une méthode `calculerAire()`, mais son implémentation dépend du type de forme :
  - Pour un cercle, `calculerAire()` renverrait $\pi \times \text{rayon}^2$
  - Pour un rectangle, `calculerAire()` renverrait longueur x largeur
  - Mais pour une forme en général, on ne peut rien dire !

À quoi servent les classes abstraites ?

- Grace aux classes abstraites, on peut implémenter des algorithmes génériques, indépendant des spécificités d'implémentation.
- Par exemple, si on veut calculer l'aire totale couverte par un ensemble de formes, on peut appeler calculerAire() sur chaque forme : on ne connaît pas le calcul effectué dans cette méthode mais cela n'est pas utile.

```
public abstract class Forme
{
        ...
        public static float calculerAireTotale (Forme[] formes) {
                float aireTotale = 0;
                for(Forme forme : formes) {
                        aireTotale += forme.calculerAire
                }
                return aireTotale
        }
}
```

Exemple : calcul d'aire totale

```
Forme[] formes = new Forme[4];
formes[0] = new Rectangle(...);
formes[1] = new Cercle(...);
formes[2] = new Cercle(...);
formes[3] = new Rectangle(...);
float aireTotale = Forme.calculerAireTotale(formes);
```

```
public abstract class Forme
{
        ...
        public static float calculerAireTotale (Forme[] formes) {
                float aireTotale = 0;
                for(Forme forme : formes) {
                        aireTotale += forme.calculerAire
                }
                return aireTotale
        }
}
```

Les interfaces

Interface

- Une interface est une classe ne contenant que des méthodes abstraites
- Les méthodes d'une interface sont implicitement qualifiées de public et abstract
- Exemple :

```
public interface Telephone
{
        void decrocher();
        void raccrocher();
        void numeroter(int numero);
}
```

- Les interfaces sont très utilisées dans l'API Java, notamment dans :
  - Les collections
  - Les interfaces graphiques (pour les écouteurs d'événements)

Implémentation

- Au lieu d'hériter d'une classe, on ***implémente*** une interface, en fournissant un corps à **toutes** ses méthodes.

```
public class
IPhone implements Telephone
{
        public void decrocher() {
                // implémentation de la méthode
        }
        public void raccrocher(){
                // implémentation de la méthode
        }
        public void numeroter(int numero) {
                // implémentation de la méthode
        }
}
```

Implémentations multiples

- Une classe peut implémenter plusieurs interfaces
- C'est un moyen de contourner l'absence d'héritage multiple.

```
public class
IPhone implements Telephone, GPS,Camera
{
        ...
        public void localiser(){
                // implémentation d'une méthode de GPS
        }
        public void filmer(){
                // implémentation d'une méthode de Camera
        }
}
```

Les implémentations par défaut (Java 8+)

- A partir de Java 8, on peut définir une implémentation par défaut pour certaines méthodes, avec le mot clé default
- Utilisations possibles :
  - Eviter à un utilisateur de devoir implémenter beaucoup de méthodes avec un corps vide (ex : java.awt.MouseListener dans les interfaces graphiques)
  - Fournir une implémentation fonctionnelle mais pas forcément optimisée. Par exemple :

```
public interface Collection {
        ...
        int size();
        default boolean empty(){ return size() == 0;}
        ...
}
```