

# Les classes abstraites

Nous avons vu que l'héritage est un moyen de mutualiser du code dans une classe parente. Parfois cette classe représente une abstraction pour laquelle il n'y a pas vraiment de sens de créer une instance. Dans ce cas, on peut considérer que la généralisation est *abstraite*.

Par opposition, on appelle *classe concrète* une classe qui n'est pas abstraite.

## Déclarer une classe abstraite

Si nous reprenons notre exemple de la classe Vehicule :

```
package com.cgi.udev.conduite;

public class Vehicule {

    private final String marque;
    protected float vitesse;

    public Vehicule(String marque) {
        this.marque = marque;
    }

    public void accelerer(float deltaVitesse) {
        this.vitesse += deltaVitesse;
    }

    public void decelerer(float deltaVitesse) {
        this.vitesse = Math.max(this.vitesse - deltaVitesse, 0f);
    }

    // ...

}
```

Cette classe peut avoir plusieurs classes filles comme *Voiture* ou *Moto*. Finalement, la classe *Vehicule* permet de faire apparaître un type à travers lequel il sera possible de manipuler des

instances de *Voiture* ou de *Moto*. Il y a peu d'intérêt dans ce contexte à créer une instance de *Vehicule*. Nous pouvons très facilement l'empêcher en déclarant par exemple le constructeur avec une portée **protected**. En Java, nous avons également la possibilité de déclarer cette classe comme abstraite (**abstract**).

```
package com.cgi.udev.conduite;

public abstract class Vehicule {

    private final String marque;
    protected float vitesse;

    public Vehicule(String marque) {
        this.marque = marque;
    }

    public void accelerer(float deltaVitesse) {
        this.vitesse += deltaVitesse;
    }

    public void decelerer(float deltaVitesse) {
        this.vitesse = Math.max(this.vitesse - deltaVitesse, 0f);
    }

    // ...

}
```

Le mot-clé **abstract** ajouté dans la déclaration de la classe indique maintenant que cette classe représente une abstraction pour laquelle il n'est pas possible de créer directement une instance de ce type.

```
Vehicule v = new Vehicule("X"); // ERREUR DE COMPILATION : LA CLASSE EST ABSTRAITE
```

En Java, il n'est pas possible de combiner **abstract** et **final** dans la déclaration d'une classe car cela n'aurait aucun sens. Une classe abstraite ne pouvant être instanciée, il faut nécessairement qu'il existe une ou des classes filles.

## Déclarer une méthode abstraite

Une classe abstraite peut déclarer des méthodes abstraites. Une méthode abstraite possède une signature mais pas de corps. Cela signifie qu'une classe qui hérite de cette méthode doit la redéfinir pour en fournir une implémentation (sauf si cette classe est elle-même abstraite).

Par exemple, un véhicule peut donner son nombre de roues. Plutôt que d'utiliser un attribut pour stocker le nombre de roues, il est possible de faire du nombre de roues une propriété abstraite de la classe.

```
package com.cgi.udev.conduite;

public abstract class Vehicule {

    private final String marque;
    protected float vitesse;

    public Vehicule(String marque) {
        this.marque = marque;
    }

    public abstract int getNbRoues();

    // ...

}
```

Toutes les classes concrètes héritant de *Vehicule* doivent maintenant fournir une implémentation de la méthode *getNbRoues* pour pouvoir compiler.

```
package com.cgi.udev.conduite;

public class Voiture extends Vehicule {

    public Voiture(String marque) {
        super(marque);
    }

    @Override
    public int getNbRoues() {
        return 4;
    }

}
```

```
// ...  
  
}
```

```
package com.cgi.udev.conduite;  
  
public class Moto extends Vehicule {  
  
    public Moto(String marque) {  
        super(marque);  
    }  
  
    @Override  
    public int getNbRoues() {  
        return 2;  
    }  
  
    // ...  
  
}
```

Une méthode abstraite peut avoir plusieurs utilités. Comme dans l'exemple précédent, elle peut servir à gagner en abstraction dans notre modèle. Mais elle peut aussi permettre à une classe fille d'adapter le comportement d'un algorithme ou d'un composant logiciel.

```
package com.cgi.udev.tableur;  
  
public abstract class Tableur {  
  
    public void mettreAJour() {  
        tracerLignesEtColonnes();  
        int premiereLigne = getPremiereLigneAffichee();  
        int premiereColonne = getPremierColonneAffichee();  
        int derniereLigne = getDerniereLigneAffichee();  
        int derniereColonne = getDerniereColonneAffichee();  
  
        for (int ligne = premiereLigne; ligne <= derniereLigne; ++ligne) {  
            for (int colonne = premiereColonne; colonne <= derniereColonne; ++colonne) {  
                String contenu = getContenu(ligne, colonne);  
            }  
        }  
    }  
  
}
```

```

        afficherContenu(ligne, colonne, contenu);
    }
}

protected abstract String getContenu(int ligne, int colonne);

private void afficherContenu(int ligne, int colonne, String contenu) {
    // ...
}

private int getDerniereColonneAffichee() {
    // ...
}

private int getDerniereLigneAffichee() {
    // ...
}

private int getPremierColonneAffichee() {
    // ...
}

private int getPremiereLigneAffichee() {
    // ...
}

private void tracerLignesEtColonnes() {
    // ...
}
}

```

Dans l'exemple ci-dessus, on imagine une classe *Tableur* qui permet d'afficher un tableau à l'écran en fonction des lignes et des colonnes visibles. Il s'agit d'une classe abstraite et les classes qui spécialisent cette classe doivent fournir une implémentation de la méthode abstraite *getContenu* afin de fournir le contenu de chaque cellule affichée par le tableur.