



Le langage Java

Approche orientée objet – Le polymorphisme et les interfaces

Programme détaillé ou sommaire

Le polymorphisme

- Définition

- Un objet vu sous plusieurs formes

- Exemple

- Downcasting

Les interfaces

- Définition

- Exemple

- Respect de l'interface

- Classes d'implémentation

- Héritage multiple

- Héritage d'interfaces

Annexe: blocs notions avancées

- Bloc static

- Bloc d'instance

- Exemple

Chapitre 1

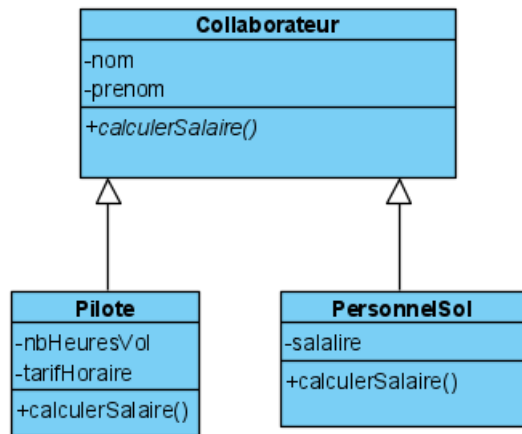
Le polymorphisme

Définition, exemple et downcasting

Polymorphisme : définition

Une fonction (i.e. méthode) peut prendre différentes formes grâce à la redéfinition de méthodes

↖



```
Pilote pilote = new Pilote();
pilote.calculerSalaire();
```

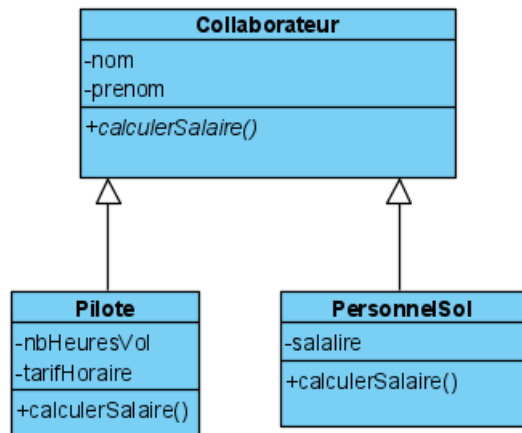
Quelle méthode **calculerSalaire()** est invoquée ?

Réponse : celle de l'objet Pilote.

Polymorphisme : autre exemple

Une fonction (i.e. méthode) peut prendre en Java différentes formes grâce à la redéfinition de méthodes

↖



```
Collab collab = new Pilote();
collab.calculerSalaire();
```

Quelle méthode **calculerSalaire()** est invoquée ?

Réponse : celle de l'objet Pilote (objet créé avec l'opérateur new).

Exemple

Implémentation :

- Si une méthode prend en paramètre un `Collaborateur`, je peux lui passer une instance de `Pilote` ou `PersonnelSol` en paramètre

```
public static void main(String[] args) {  
    PersonnelSol unPersonnel = new PersonnelSol();  
    editerPdfSalaire(unPersonnel);  
  
    Pilote unPilote = new Pilote();  
    editerPdfSalaire(unPilote);  
}  
  
public static void editerPdfSalaire(Collaborateur collab){  
    long salaire = collab.calculerSalaire();  
    String texte = "attestation officielle. Montant du salaire : "+ salaire;  
    System.out.println(texte);  
}
```

Suivant le cas, ce sera la méthode **calculerSalaire()** de **Pilote** ou de **PersonnelSol** qui sera appelée

instanceof

Ce mot clé permet de tester le type d'un objet

Exemple

```
public static void editerPdfSalaire(Collaborateur collab){  
  
    if(collab instanceof Pilote){  
        System.out.println("Statut: Pilote");  
    }  
    if(collab instanceof PersonnelSol){  
        System.out.println("Statut: Personnel au sol");  
    }  
  
}
```

Downcasting

En cas de besoin de convertir un objet dans le contexte d'un héritage

opérateur ()

En cas d'erreur : `ClassCastException`

Exemple

```
public static void editerPdfSalaire(Collaborateur collab) {  
  
    if (collab instanceof Pilote) {  
        System.out.println("Statut: Pilote");  
        Pilote pilote = (Pilote) collab;  
        int nbHeures = pilote.getNombreHeuresVol();  
    }  
}
```


Chapitre 2

Les interfaces

Définition, contrat, classes d'implémentation, héritage

Définition

Interface : contrat que doit respecter une classe

- Signature de méthodes, documentation

- Pas d'implémentation

- Des constantes

Mise en œuvre

- Déclaration : `interface` au lieu de `class`

- Tous les attributs sont implicitement `public static final`


- Toutes les méthodes sont implicitement `public` **et** `abstract`



Nom de l'interface : souvent suffixe en "able" : Comparable, Pilotable...

Exemple

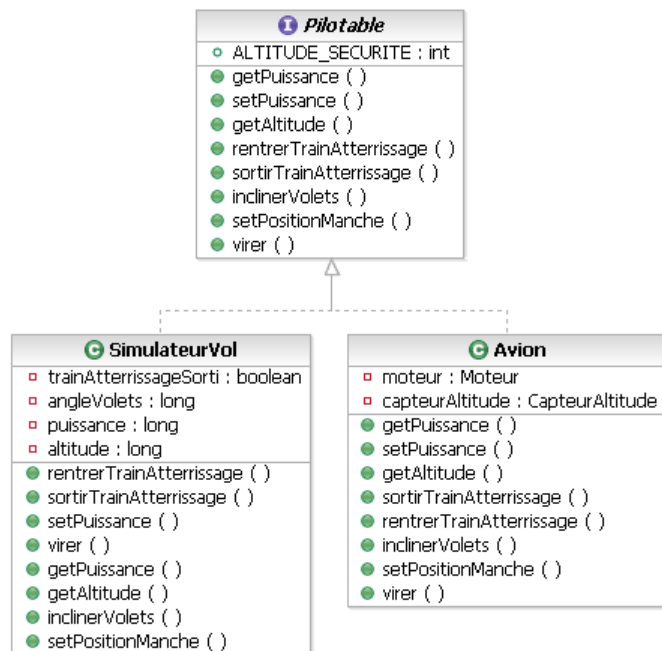
Interface Pilotable

 Pilotable
◦ ALTITUDE_SECURITE : int
● getPuissance ()
● setPuissance ()
● getAltitude ()
● rentrerTrainAtterrissage ()
● sortirTrainAtterrissage ()
● inclinerVolets ()
● setPositionManche ()
● virer ()

```
public interface Pilotable {  
    public static final int ALTITUDE_SECURITE = 100;  
  
    public long getPuissance();  
    public void setPuissance(long puissanceCible);  
    public long getAltitude();  
    public void rentrerTrainAtterrissage();  
    public void sortirTrainAtterrissage();  
    public void inclinerVolets(long angle);  
    public void setPositionManche(long position);  
    public void virer(long angle);  
}
```

Respect de l'interface

Avion et SimulateurVol respectent l'interface Pilotable



Classes d'implémentation

Mot-clé **implements**

La classe implémente toutes les méthodes de l'interface

Sinon erreur compilation

*doivent être déclarées **public***

+ éventuellement d'autres méthodes

```
public class Avion implements Pilotable {  
    private Moteur moteur;  
    private CapteurAltitude capteurAltitude;  
  
    public long getPuissance() { return moteur.getPuissance(); }  
    public void setPuissance(long puissance) { moteur.setPuissance(puissance); }  
    public long getAltitude() { return capteurAltitude.getAltitude(); }  
    public void sortirTrainAtterrissage() { /* code */ }  
    public void rentrerTrainAtterrissage() { /* code */ }  
    public void inclinerVolets(long angle) { /* code */ }  
    public void setPositionManche(long position) { /* code */ }  
    public void virer(long angle) { /* code */ }  
}
```

Exemple d'utilisation comme un type

Pilotable : simulateur ou avion

```
public void faireDecoller(Pilotable objetVolant) {  
    // code simpliste pour faire décoller l'avion  
    objetVolant.setPuissance(400);  
    objetVolant.inclinerVolets(10);  
    objetVolant.setPositionManche(3);  
  
    if (objetVolant.getAltitude() >= Pilotable.ALTITUDE_SECURITE) {  
        objetVolant.inclinerVolets(0);  
        objetVolant.rentreTrainAtterrissage();  
    }  
}
```

```
public static void main(String[] args) {  
    Pilote unPilote = new Pilote();  
    Pilotable pilotable = new SimulateurVol();  
    // autre possibilité : pilotable = new Avion();  
    unPilote.faireDecoller(pilotable);  
}
```

Héritage multiple

N'existe pas en Java

une classe ne peut hériter que **d'une seule** classe

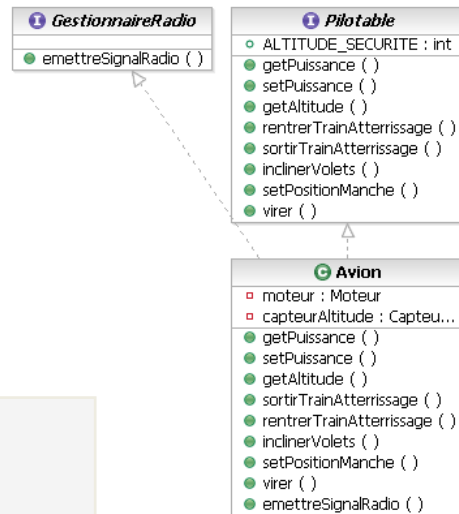
Une classe peut implémenter plusieurs interfaces

Plusieurs "casquettes"

Objet pilotable

Objet avec GestionnaireRadio

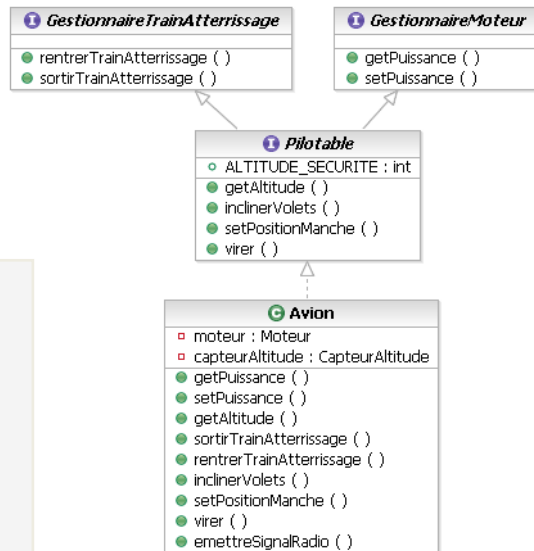
```
public class Avion implements Pilotable, GestionnaireRadio {  
    // attributs  
    // toutes les méthodes de l'interface Pilotable  
    public long getPuissance() { return moteur.getPuissance(); }  
    public void setPuissance(long puissance) { moteur.setPuissance(puissance); }  
  
    // toutes les méthodes de l'interface GestionnaireRadio  
    public void emettreSignalRadio(String message) { /* code */ }  
}
```



Héritage d'interface

Une interface peut hériter de plusieurs interfaces

```
public interface Pilotable extends GestionnaireMoteur,  
GestionnaireTrainAtterrissage {  
    public static final int ALTITUDE_SECURITE = 100;  
  
    public long getAltitude();  
    public void inclinerVolets(long angle);  
    public void setPositionManche(long position);  
    public void virer(long angle);  
}
```



Redéfinition de toutes les méthodes dans les classes d'implémentation

Annexes

Notions avancées sur les blocs

Blocs statiques et blocs d'instance

Bloc static

Le bloc statique est exécuté une seule fois, la première fois qu'une classe est utilisée
Une classe peut contenir plusieurs blocs statiques.

```
public class Chose {  
    static {  
        System.out.println("Ce bloc n'est exécuté qu'une seule fois.");  
    }  
  
    public Chose(){  
        System.out.println("La classe est instanciée.");  
    }  
}
```

Que produit ce code ?

```
public class TestChose {  
    public static void main(String[] args) {  
        Chose chose1 = new Chose();  
        Chose chose2 = new Chose();  
    }  
}
```

Bloc static

Réponse

Ce bloc n'est exécuté qu'une seule fois.
La classe est instanciée.
La classe est instanciée.

Bloc d'instance

Le bloc d'instance est exécuté à chaque instanciación avant l'appel du constructeur.
S'il y a un bloc statique, le bloc d'instance est exécuté après le bloc statique

```
public class Bidule {  
    {  
        System.out.println("Bloc d'instance exécutée.");  
    }  
  
    public Bidule(){  
        System.out.println("La classe est instanciée.");  
    }  
}
```

Que produit ce code ?

```
public class TestBidule {  
  
    public static void main(String[] args) {  
        Bidule bidule1 = new Bidule();  
        Bidule bidule2 = new Bidule();  
    }  
}
```

Bloc d'instance

Réponse

```
Bloc d'instance exécutée.  
La classe est instanciée.  
Bloc d'instance exécutée.  
La classe est instanciée.
```

Exemple

Code code est-il correct ?

```
class TestSyntaxe {  
    static {  
        {  
            System.out.println("Bloc static");  
        }  
    }  
    {  
        {  
            {  
                System.out.println("Bloc 1");  
            }  
        }  
    }  
    void hello() {  
        {  
            {  
                System.out.println("Hello");  
            }  
        }  
    }  
}
```

Atelier (TP)

OBJECTIFS : mettre en œuvre les principaux mécanismes de la programmation orientée objet

DESCRIPTION : Dans ce TP vous allez mettre en place une interface et une classe d'implémentation