UFR de mathématiques et d'informatique UFR 27
Licence 2ème année MIASHS

Programmation orientée objet - Niveau 2

Khelifi Tesnim

Chapitre 1: Les Interfaces

CLASSES ET MÉTHODES ABSTRAITES

- L'**abstraction** des données est le processus qui consiste à masquer certains détails et à ne montrer que les informations essentielles à l'utilisateur. L'abstraction peut être réalisée à l'aide de classes abstraites ou d'interfaces.
- Le mot-clé **abstract** est un modificateur non accessible, utilisé pour les classes et les méthodes :
  - **Classe abstraite** : il s'agit d'une classe restreinte qui ne peut pas être utilisée pour créer des objets (pour y accéder, elle doit être héritée d'une autre classe).
  - **Méthode abstraite** : elle ne peut être utilisée que dans une classe abstraite et ne possède pas de corps. Le corps est fourni par la sous-classe (héritée).

- Une classe abstraite peut avoir à la fois des méthodes abstraites et des méthodes régulières.

CLASSES ET MÉTHODES ABSTRAITES

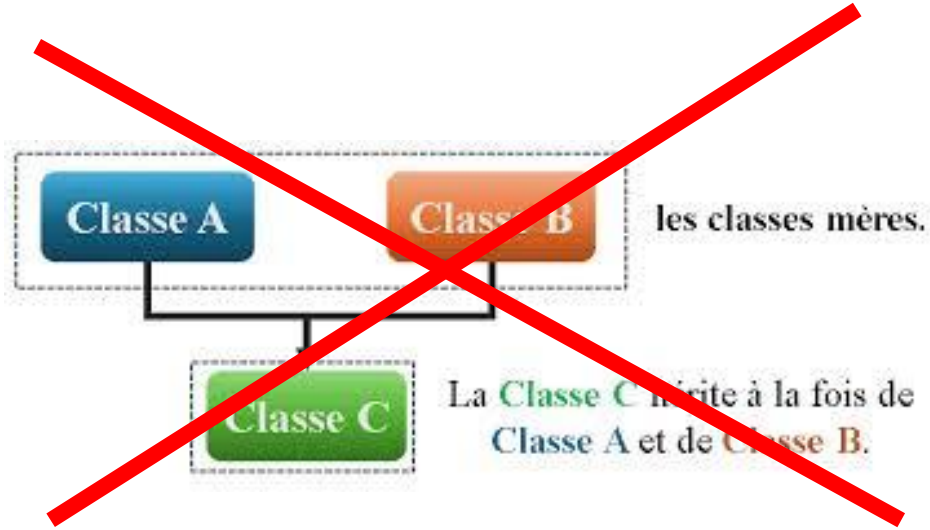
```java
// Abstract class
abstract class Animal {
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
  }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}

class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

LES INTERFACES

- Il n'y a pas **d'héritage multiple entre classes** en Java: **une classe ne peut être l'extension que d'une seule classe**.
- Par contre une classe peut implémenter plusieurs interfaces (et être l'extension d'une seule classe).



LES INTERFACES

- En programmation orientée objet (POO), une **interface** est un **type de contrat qui définit les méthodes qu'une classe doit implémenter**. Une interface ne fournit **pas d'implémentation** réelle des méthodes, mais elle fournit simplement une **liste de méthodes** que la classe qui implémente l'interface doit fournir.

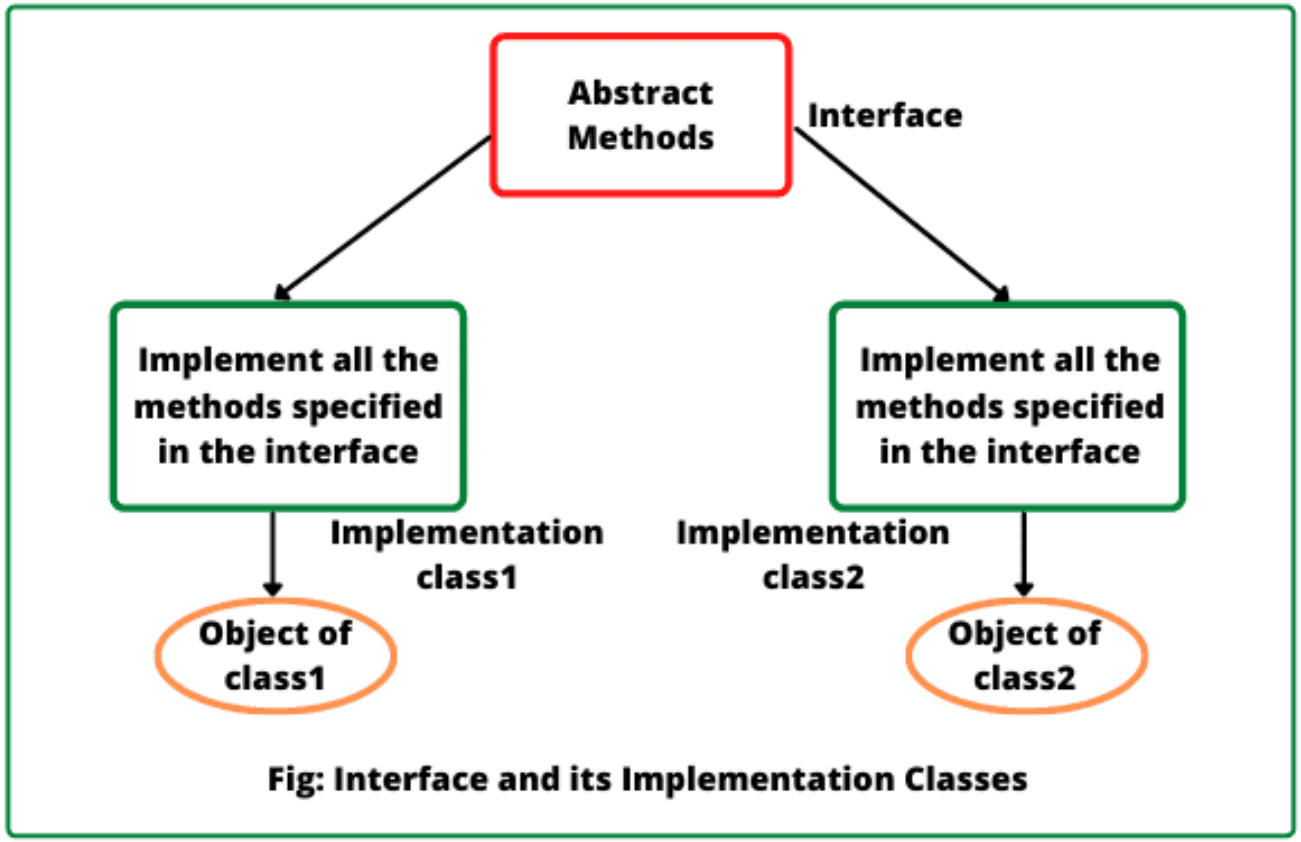


Fig: Interface and its Implementation Classes

LES INTERFACES

- En Java, le mot-clé "**implements**" est utilisé pour indiquer qu'une **classe implémente une interface**. Cela signifie que la classe fournit une implémentation pour toutes les méthodes définies dans l'interface.
- Voici un exemple de déclaration de classe qui implémente une interface en utilisant le mot clé "**implements**":

```
public class MyClass implements MyInterface {

// Code de la classe

}
```

- Dans cet exemple, "**MyClass**" est une classe qui implémente l'interface "**MyInterface**".
- La classe doit fournir des implémentations pour toutes les méthodes définies dans "**MyInterface**".

LES INTERFACES

- Une interface ne contient (essentiellement) que des déclarations de méthodes.
- Une interface est un peu comme une classe sans données membres et dont toutes les méthodes seraient **abstraites**.
- En Java, une interface est définie à l'aide du mot-clé "**interface**". Les **classes** qui implémentent une interface doivent fournir une **implémentation** pour toutes les **méthodes de l'interface**. Une **classe peut implémenter plusieurs interfaces**, ce qui permet à la classe d'avoir plusieurs types.
- Voici un exemple d'interface en Java :

```
public interface Animal {

public void makeSound();
public void move();

}
```

Dans cet exemple, nous avons défini une **interface** appelée "**Animal**". L'interface définit deux méthodes, "**makeSound**" et "**move**". Toutes les classes qui implémentent cette interface doivent fournir une implémentation pour ces deux méthodes.

LES INTERFACES

- Voici deux exemples de classes qui implémentent l'interface "Animal" :

```
public class Dog implements Animal {

public void makeSound() {
System.out.println("Woof!");
}

public void move() {
System.out.println("The dog is running.");
}

}
```

```
public class Cat implements Animal {

public void makeSound() {
System.out.println("Meow!");
}

public void move() {
System.out.println("The cat is running.");
}

}
```

LES INTERFACES

- Maintenant, nous pouvons utiliser ces deux classes comme des objets de type "Animal". Par exemple :

```
Animal myDog = new Dog();

Animal myCat = new Cat();


myDog.makeSound(); // affichera "Woof!"

myCat.makeSound(); // affichera "Meow!"
```

Dans cet exemple, nous avons créé **deux objets**, "**myDog**" et "**myCat**", de **type** "**Animal**". Nous avons ensuite appelé la méthode "**makeSound**" sur chacun de ces objets. Lorsque nous avons appelé "**myDog.makeSound()**", il a affiché "Woof!", et lorsque nous avons appelé "**myCat.makeSound()**", il a affiché "Meow!".

LES INTERFACES

- Les interfaces sont utiles dans de nombreuses situations, notamment lors de la création de bibliothèques de classes qui doivent être utilisées par d'autres développeurs. En utilisant des interfaces, les développeurs peuvent **garantir que toutes les classes qui implémentent l'interface ont des méthodes spécifiques**, ce qui facilite la création de programmes qui utilisent ces classes.
- Les méthodes d'interface n'ont pas de corps : celui-ci est fourni par la classe « **implement** ».
- Lors de l'implémentation d'une interface, vous devez redéfinir toutes ses méthodes.
- Les méthodes d'interface sont par défaut **abstract** et **public**.
- Les attributs d'interface sont par défaut **public**, **static** et **final**.
- Une interface ne peut pas contenir de constructeur (car elle ne peut pas être utilisée pour créer des objets).

Pourquoi et quand utiliser les interfaces ?

1) Pour garantir la sécurité : masquer certains détails et n'afficher que les détails importants d'un objet (interface).

2) Java ne prend pas en charge « l'héritage multiple » (une classe ne peut hériter que d'une seule superclasse). Cependant, cela peut être réalisé avec des interfaces, car la classe peut implémenter plusieurs interfaces. Remarque : pour implémenter plusieurs interfaces, séparez-les par une virgule (voir l'exemple).

LES INTERFACES

```
interface FirstInterface {
  public void myMethod(); // interface method
}

interface SecondInterface {
  public void myOtherMethod(); // interface method
}

class DemoClass implements FirstInterface, SecondInterface {
  public void myMethod() {
    System.out.println("Some text..");
  }
  public void myOtherMethod() {
    System.out.println("Some other text...");
  }
}

class Main {
  public static void main(String[] args) {
    DemoClass myObj = new DemoClass();
    myObj.myMethod();
    myObj.myOtherMethod();
  }
}
```

LES INTERFACES

- Une **méthode default** dans une **interface** est une méthode qui possède une implémentation (du code), contrairement aux méthodes classiques d'interface qui sont abstraites.
- Elle permet d'ajouter un comportement sans obliger les classes à réécrire la méthode.

```
public interface Animal {

   void parler();   // méthode abstraite

   default void dormir() {   // méthode default
      System.out.println("L'animal dort...");
   }
}
```

Implémentation dans une classe

```
public class Chat implements Animal {

    @Override
    public void parler() {
        System.out.println("Miaou");
    }
}
```

```
Chat c = new Chat();
c.dormir();   // Affiche : L'animal dort...
```

- La classe **n'implémente pas `dormir()`**, mais elle peut quand même l'utiliser.