Introduction à la programmation et au développement d'applications en Java

Relations abstraites : les interfaces

Introduction

- ❏ Qu’est-ce qu’une interface ?
- ❏ Implémenter une interface
- ❏ Implémenter plusieurs interfaces
- ❏ Déclarer une interface

Qu'est-ce qu'une interface ?

- Une **interface** définit un **contrat**
- Contrairement à une classe, ne fournit pas d'implémentation
- Une classe peut indiquer qu'elle **implémente** une interface (mot-clé `implements`) :
  - elle déclare alors se conformer au contrat en implémentant les méthodes définies dans l'interface
  - cela ne limite pas les autres aspects de l'implémentation de la classe

Exemple : `java.lang.Comparable`

- L'interface **`java.lang.Comparable`** est définie dans la JCL
- Utilisée pour déterminer un ordre relatif entre objets du même type
- Elle a une seule méthode, **`compareTo`** :
  - reçoit en argument l'objet avec lequel l'objet courant doit être comparé
  - la valeur de retour indique si, dans une séquence ordonnée, l'objet courant doit se trouver avant ou après l'objet passé :
    - négative : avant
    - positive : après
    - zéro : égalité

```
public class Passager {

  // ... Autres membres ...

  private int niveauMembre;       // 1 (silver), 2 (gold), 3 (platine)
                                  // les niveaux supérieurs sont prioritaires
  private int joursMembre;









}
```

```
public class Passager implements Comparable {

  // ... Autres membres ...

  private int niveauMembre;       // 1 (silver), 2 (gold), 3 (platine)
                                  // les niveaux supérieurs sont prioritaires
  private int joursMembre;









}
```

```
public class Passager  implements Comparable {

  // ... Autres membres ...

  private int niveauMembre;       // 1 (silver), 2 (gold), 3 (platine)
                                  // les niveaux supérieurs sont prioritaires
  private int joursMembre;

  public int compareTo(Object obj) {









  }
}
```

```
public class Passager implements Comparable {

  // ... Autres membres ...

  private int niveauMembre;       // 1 (silver), 2 (gold), 3 (platine)
                                  // les niveaux supérieurs sont prioritaires
  private int joursMembre;

  public int compareTo(Object obj) {
    Passager autre = (Passager) obj;

    if (niveauMembre > autre.niveauMembre)
      return -1;
    if (niveauMembre < autre.niveauMembre)
      return 1;
    if (joursMembre > autre.joursMembre)
      return -1;
    if (joursMembre < autre.joursMembre)
      return 1;
    return 0;
  }
}
```

```
public class Passager implements Comparable {

  // ... Autres membres ...

  private int niveauMembre;       // 1 (silver), 2 (gold), 3 (platine)
                                  // les niveaux supérieurs sont prioritaires
  private int joursMembre;

  public int compareTo(Object obj) {
    Passager autre = (Passager) obj;

    if (niveauMembre > autre.niveauMembre)
      return -1;
    if (niveauMembre < autre.niveauMembre)
      return 1;
    if (joursMembre > autre.joursMembre)
      return -1;
    if (joursMembre < autre.joursMembre)
      return 1;
    return 0;
  }
}
```

```
Passager bob = new Passager(1, 180);
Passager lea = new Passager(1, 90);
Passager zoe = new Passager(3, 730);
Passager tom = new Passager(2, 180);

Passager[] passagers =
             { bob, lea, zoe, tom };
```

```java
public class Passager implements Comparable {

  // ... Autres membres ...

  private int niveauMembre;       // 1 (silver), 2 (gold), 3 (platine)
                                  // les niveaux supérieurs sont prioritaires
  private int joursMembre;

  public int compareTo(Object obj) {
    Passager autre = (Passager) obj;

    if (niveauMembre > autre.niveauMembre)
      return -1;
    if (niveauMembre < autre.niveauMembre)
      return 1;
    if (joursMembre > autre.joursMembre)
      return -1;
    if (joursMembre < autre.joursMembre)
      return 1;
    return 0;
  }
}
```

```java
Passager bob = new Passager(1, 180);
Passager lea = new Passager(1, 90);
Passager zoe = new Passager(3, 730);
Passager tom = new Passager(2, 180);

Passager[] passagers =
            { bob, lea, zoe, tom };

// Arrays.sort va utiliser compareTo
// pour effectuer le tri
Arrays.sort(passagers);
```

```java
public class Passager implements Comparable {

  // ... Autres membres ...

  private int niveauMembre;       // 1 (silver), 2 (gold), 3 (platine)
                                  // les niveaux supérieurs sont prioritaires
  private int joursMembre;

  public int compareTo(Object obj) {
    Passager autre = (Passager) obj;

    if (niveauMembre > autre.niveauMembre)
      return -1;
    if (niveauMembre < autre.niveauMembre)
      return 1;
    if (joursMembre > autre.joursMembre)
      return -1;
    if (joursMembre < autre.joursMembre)
      return 1;
    return 0;
  }
}
```

```java
Passager bob = new Passager(1, 180);
Passager lea = new Passager(1, 90);
Passager zoe = new Passager(3, 730);
Passager tom = new Passager(2, 180);

Passager[] passagers =
            { bob, lea, zoe, tom };

// Arrays.sort va utiliser compareTo
// pour effectuer le tri
Arrays.sort(passagers);

// ordre : zoe, tom, bob, lea
```

```
public class Vol implements Comparable {

  // ... Autres membres ...

  private int tempsDeVol;

  public int compareTo(Object obj) {

    Vol autre = (Vol) obj;

    if (tempsDeVol < autre.tempsDeVol)
      return -1;
    if (tempsDeVol > autre.tempsDeVol)
      return 1;
    return 0;
  }
}
```

```
public class Vol implements Comparable {

  // ... Autres membres ...

  private int tempsDeVol;

  public int compareTo(Object obj) {

    Vol autre = (Vol) obj;

    return tempsDeVol - autre.tempsDeVol;
  }
}
```

Interfaces génériques

- Certaines interfaces requièrent des informations de type supplémentaires, qui vont en quelque sorte paramétrer l'interface
- Ces interfaces sont dites **génériques**
- Ce concept est connu sous le nom de **généricité** (existe aussi pour les classes)
- Par exemple, il existe une version générique de l'interface **`Comparable`** : **`Comparable<T>`**
- On indique par cette syntaxe que l'interface est paramétrée par un **type** qui sera fourni lors de l'utilisation de l'interface
- L'implémentation de l'interface sera alors liée au type indiqué

```
public interface Comparable<T> {

  int compareTo(T o);

}
```

```
public interface Comparable<T> {

  int compareTo(T o);

}
```

```
class Vol implements Comparable {

  // ... Autres membres ...

  public int compareTo(Object obj) {
    Vol autre = (Vol) obj;
    return tempsDeVol - autre.tempsDeVol;
  }
}
```

```
public interface Comparable<T> {

  int compareTo(T o);

}
```

```
class Vol implements Comparable<Vol> {

  // ... Autres membres ...

  public int compareTo(Object obj) {
    Vol autre = (Vol) obj;
    return tempsDeVol - autre.tempsDeVol;
  }
}
```

```
public interface Comparable<T> {

  int compareTo(T o);

}
```

```
class Vol implements Comparable<Vol> {

  // ... Autres membres ...

  public int compareTo(Vol autre) {
    Vol autre = (Vol) obj;
    return tempsDeVol - autre.tempsDeVol;
  }
}
```

```
public interface Comparable<T> {

  int compareTo(T o);

}
```

```
class Vol implements Comparable<Vol> {

  // ... Autres membres ...

  public int compareTo(Vol autre) {
    Vol autre = (Vol) obj;
    return tempsDeVol - autre.tempsDeVol;
  }
}
```

```
public class Passager implements Comparable {

  // ... Autres membres ...

  public int compareTo(Object obj) {
    Passager autre = (Passager) obj;

    if (niveauMembre > autre.niveauMembre)
      return -1;
    // etc.
  }
}
```

```
public interface Comparable<T> {

  int compareTo(T o);

}
```

```
class Vol implements Comparable<Vol> {

  // ... Autres membres ...

  public int compareTo(Vol autre) {
    Vol autre = (Vol) obj;
    return tempsDeVol - autre.tempsDeVol;
  }
}
```

```
public class Passager implements Comparable<Passager> {

  // ... Autres membres ...

  public int compareTo(Object obj) {
    Passager autre = (Passager) obj;

    if (niveauMembre > autre.niveauMembre)
      return -1;
    // etc.
  }
}
```

```java
public interface Comparable<T> {

  int compareTo(T o);

}
```

```java
class Vol implements Comparable<Vol> {

  // ... Autres membres ...

  public int compareTo(Vol autre) {
    Vol autre = (Vol) obj;
    return tempsDeVol - autre.tempsDeVol;
  }
}
```

```java
public class Passager implements Comparable<Passager> {

  // ... Autres membres ...

  public int compareTo(Passager autre) {
    Passager autre = (Passager) obj;

    if (niveauMembre > autre.niveauMembre)
      return -1;
    // etc.
  }
}
```

Implémenter plusieurs interfaces

- En Java, il est interdit d'hériter de plus d'une classe
- En revanche, une classe peut implémenter **autant d'interfaces qu'elle souhaite**
- Une classe peut donc se conformer à autant de contrats qu'elle a besoin
- Les noms des interfaces implémentées sont séparées par des virgules après le mot-clé **`implements`** :

```
class A implements Comparable<A>, Iterable<Personne> { ... }
```

Déclarer une interface

- Jusqu'à présent, on a implémenté des interfaces fournies par la JCL
- Mais on peut aussi déclarer nos propres interfaces
- Similaire à la déclaration d'une classe

Déclarer une interface

- Jusqu'à présent, on a implémenté des interfaces fournies par la JCL
- Mais on peut aussi déclarer nos propres interfaces
- Similaire à la déclaration d'une classe

```
public interface EnginVolant {

  double ALTITUDE_MAX = 60000.0d;

  void voler() ;
  double monter(double delta) ;
  double descendre(double delta) ;
}
```

Déclarer une interface

- Jusqu'à présent, on a implémenté des interfaces fournies par la JCL
- Mais on peut aussi déclarer nos propres interfaces
- Similaire à la déclaration d'une classe

mot-clé **`interface`**

```
public interface EnginVolant {

  double ALTITUDE_MAX = 60000.0d;

  void voler() ;
  double monter(double delta) ;
  double descendre(double delta) ;
}
```

Déclarer une interface

- Jusqu'à présent, on a implémenté des interfaces fournies par la JCL
- Mais on peut aussi déclarer nos propres interfaces
- Similaire à la déclaration d'une classe

mot-clé **`interface`**

méthodes : automatiquement **`public`**

```
public interface EnginVolant {

  double ALTITUDE_MAX = 60000.0d;

  void voler() ;
  double monter(double delta) ;
  double descendre(double delta) ;
}
```

Déclarer une interface

- Jusqu'à présent, on a implémenté des interfaces fournies par la JCL
- Mais on peut aussi déclarer nos propres interfaces
- Similaire à la déclaration d'une classe

mot-clé **`interface`**

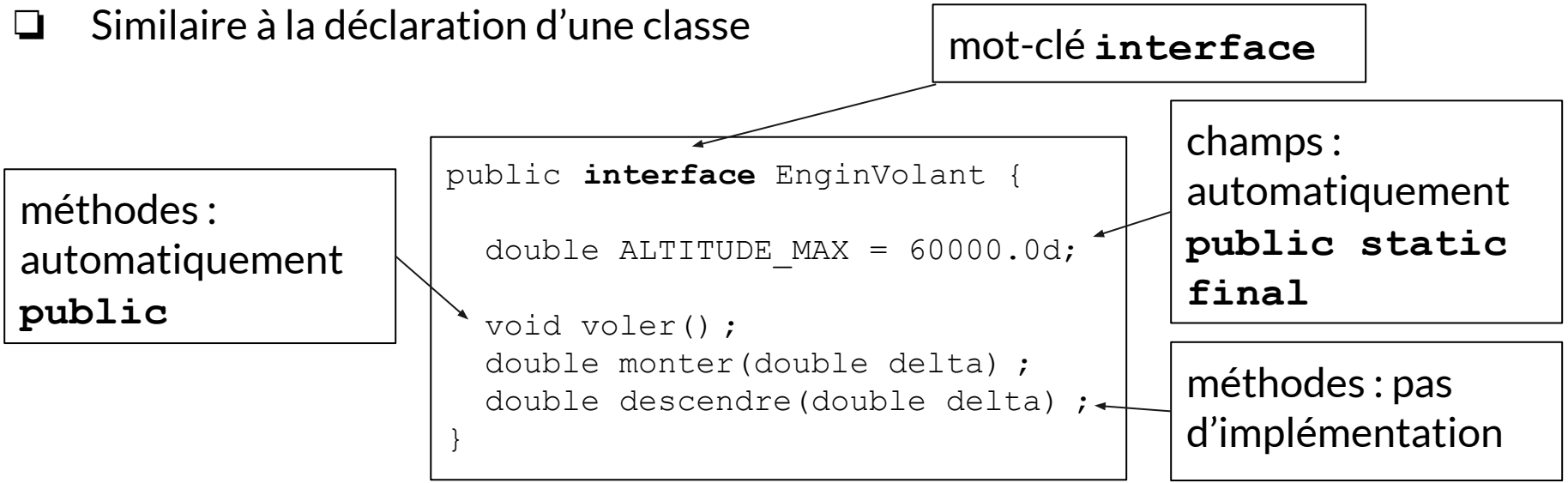
```
public interface EnginVolant {

  double ALTITUDE_MAX = 60000.0d;

  void voler();
  double monter(double delta);
  double descendre(double delta);
}
```

méthodes : automatiquement **`public`**

méthodes : pas d'implémentation

Déclarer une interface

- Jusqu'à présent, on a implémenté des interfaces fournies par la JCL
- Mais on peut aussi déclarer nos propres interfaces
- Similaire à la déclaration d'une classe



Déclarer une interface

- Une interface peut **dériver** d'une autre interface
- La classe qui l'implémente devra alors se conformer aux contrats de l'interface **et de son interface parente**

```
public interface EnginVolant extends Engin {

  double ALTITUDE_MAX = 60000.0d;

  void voler() ;
  double monter(double delta) ;
  double descendre(double delta) ;
}
```

Résumé

- Concept fondamental : une interface définit un **contrat**
- Une classe qui **implémente** l'interface déclare qu'elle est **capable** de remplir ce contrat
- Contrat : méthodes et constantes, mais pas d'implémentation
- Les classes peuvent **implémenter plusieurs interfaces**, c'est ainsi que Java lève sa limitation de ne pas proposer d'héritage multiple
- Les interfaces peuvent dériver d'autres interfaces