Compléments sur les interfaces

Stephane Malandain – POO - Java

I. Les interfaces

- Une **interface** est une classe abstraite qui implémente aucune méthode et aucun champ (hormis les constantes)

- Elle définit un/le **comportement** qui doit être implémenté par une classe, sans implémenter celui-ci. C'est un ensemble de méthodes abstraites et de constantes.

- Notion plus riche que la classe abstraite car :

  - Une classe peut implémenter plusieurs interfaces
  - La notion d'interface se superpose à celle de dérivation
  - Les interfaces peuvent donc se dériver
  - On peut utiliser des variables de types interface

I. Les interfaces

- Liste de méthodes dont on donne seulement la **signature**. Attention, ce n'est pas une classe !

- **Représente un "contrat", ce qu'on attend d'un objet.** Lorsqu'on utilise un objet d'une classe implémentant une certaine interface, on a la garantie que cet objet dispose de toutes les méthodes annoncées par l'interface.

- Peut être implémentée par une ou plusieurs classes qui doivent donner une implémentation pour chacune des méthodes annoncées (et éventuellement d'autres).

- Une classe peut implémenter **plusieurs** interfaces (permettant un héritage multiple).

I. Les interfaces

- Elle contient des définitions de constantes et des déclarations de méthode (prototype ou signature)
- Toutes les méthodes d'une interface sont implicitement publiques et abstraites.
- Une interface n'a pas de constructeurs
- Une interface ne peut avoir de champs sauf si ceux-ci sont statiques.
- Une interface peut être étendue par une ou plusieurs autre(s) interface(s).
- Si `Inter` est une interface, alors on peut écrire `Inter I;`

I. Interfaces : exemple 1

```
public interface Exemple1 {
  public static final MAX = 32;
  int ajouter(int i, int j);
  int soustraire (int i, int j);
}

public class classe1 implements Exemple1 {
  public int ajouter(int a, int b) {
    return a+b+10;
  }
  public int soustraire (int a, int b) {
    return a-b+20;
  }
} // classe 1

public class classe2 implements Exemple1 {
  public int ajouter(int a, int b) {
    return a+b+100;
  }
  public int soustraire (int a, int b) {
    return a-b+200;
  }
} // classe 2
```

I. Interfaces : exemple 1

```
public class test {

  private static void calculer (int i, int j, Exemple1 inter) {
    System.out.println(inter.ajouter(i,j));
    System.out.println(inter.soustraire(i,j));
}

  public void main(String[] arg) {
       classe1 c1 = new classe1();
       classe2 c2 = new classe2();
       // appels de la fonction statique calculer
       calculer(4,3,c1);
       calculer(14,13,c2);
  }
 } // classe test
```

II. Interfaces : exemple 2

```
public interface Comparable {
   // devra retourner -1, 0 ou 1 selon si l'objet concerne est considere
   // comme plus petit, egal ou plus grand que l'objet o

   public static final int MAX = 100;
   int comparerA(Object o);
}
```

L'interface `Comparable` sera implémentée par des classes d'objets qu'on souhaite comparables entre eux.

Nous allons illustrer ceci avec une gestion simplifiée de comptes bancaires.

La classe `compte` suivante permet de rendre des comptes bancaires comparables selon leur solde.

I. Interfaces : exemple 2

```
public class Compte implements Comparable {
 private int montant;
 private int numero;
 private String proprietaire;

 public Compte(String proprietaire, int numero, int montant) {
     this.proprietaire = proprietaire;
     this.numero = numero;
     this.montant = montant;
 }
 public void modifier(int somme) {
     if (this.montant + somme > 0) this.montant = this.montant + somme;
 }
 public int getMontant() { return this.montant; }
 public int getNumero() { return this.numero; }
 public String getProprietaire() { return proprietaire; }
 public String toString() {
   return "Compte numero " + this.numero + " : proprietaire " +
this.proprietaire + ", montant " + this.montant;
}

// On compare les comptes selon leurs montants
public int comparerA(Object o) {
   Compte autre = (Compte) o;
   if (this.montant < autre.getMontant()) return -1;
   if (this.montant > autre.getMontant()) return 1;
   return 0;
} }
```

II. Interfaces : exemple 2

Nous appellerons des `Comparable` tout objet d'une classe qui implémente l'interface `Comparable`.

On décide maintenant de définir des méthodes qui puissent s'appliquer à des `comparables`(s).

3 méthodes statiques :

• Une première méthode : la méthode `plusGrand(Comparable c1, Comparable C2)` qui retourne le plus grand des deux.

• Une méthode nommée `trier` servant à trier un tableau de `comparable`(s)

• Une méthode appelée aussi `trier` servant à trier un `ArrayList` de `Comparable`(s).

Remarque : les méthodes `trier` utilisent un tri par insertion

II. Interfaces : exemple 2

```
import java.util.ArrayList;
public class Ordre {

public static Comparable plusGrand(Comparable c1,Comparable c2) {
  if (c1.comparerA(c2) == 1) return c1;
  else return c2;
}

public static void trier(Comparable[] tableau) {
  int i, j;
  Comparable cle;
  for (i = 1; i < tableau.length; i++) {
    cle = tableau[i];
    j = i;
    while((j >= 1) && (cle.comparerA(tableau[j - 1]) == -1)) {
      tableau[j] = tableau[j - 1];
      j = j - 1;
    }
    tableau[j] = cle;
  }
}
```

II. Interfaces : exemple 2

```
Public static <T extends Comparable> void trier(ArrayList<T> Liste)
{
 int i, j;
 T cle;
 for (i = 1; i < liste.size(); i++) {
     cle = liste.get(i);
     j = i;
     while((j >= 1) && (cle.comparerA(liste.get(j - 1)) == -1)) {
        liste.set(j, liste.get(j - 1));
        j = j - 1;
      }
      liste.set(j, cle);  };
  }

}
```

Remarque : l'API java contient aussi une interface `Comparable` dans le paquetage `java.lang`. Cette interface déclare une méthode `compareTo` qui s'écrit ainsi :

```
public interface Comparable {
   int compareTo(T t);
}
```

I. Interfaces - exemple 2 : la classe Banque

```
import java.util.ArrayList;
import utilitaires.Ordre;
public class Banque {
   private int dernierNumero; // pour numeroter les comptes
   private ArrayList<Compte> listeComptes = new ArrayList<Compte>();
   public int ajouterCompte(String nom, int montant) {
      dernierNumero++;
      Compte nouveauCompte = new Compte( nom, dernierNumero, montant);
      listeComptes.add(nouveauCompte);
      return dernierNumero;
   }
   public void lister() {
      for (Compte compte : listeComptes)  System.out.println(compte);
      System.out.println();
   }
   /* Cherche dans listeCompte s'il existe un compte dont le numero est le numero
      indique en parametre à la methode. Retourne ce compte s'il en existe un,
      null sinon. */

   public Compte chercherCompte(int numero) {
      for (Compte compte : listeComptes)
         if (compte.getNumero() == numero) return compte;
      return null;
   }
...
```

I. Interfaces - exemple 2 : la classe Banque

```
public boolean supprimerCompte(int numero) {
   for (Compte compte : listeComptes)
      if (compte.getNumero() == numero) {
          listeComptes.remove(compte);
          return true;
      }
   return false;
}

 public void modifierCompte(int numero, int somme) {
    Compte compte = this.chercherCompte(numero);
    if (compte == null) return;
    compte.modifier(somme);
 }

 public void trierComptes() {
        Ordre.trier(listeComptes);
 }
} // fin classe banque
```