TP de JAVA – Série 10
Interfaces et polymorphisme

Université de Mons-Hainaut
2004 - 2005

1. Résumé du chapitre

- Une interface Java déclare un ensemble de méthodes avec leurs signatures. Contrairement à une classe, elle n'est pas implémentée.

1. Résumé du chapitre

- Une interface Java déclare un ensemble de méthodes avec leurs signatures. Contrairement à une classe, elle n'est pas implémentée.
- Pour « réaliser » une interface, une classe doit fournir (implémenter) toutes les méthodes que l'interface déclare.

1. Résumé du chapitre

- Une interface Java déclare un ensemble de méthodes avec leurs signatures. Contrairement à une classe, elle n'est pas implémentée.
- Pour « réaliser » une interface, une classe doit fournir (implémenter) toutes les méthodes que l'interface déclare.
- Les interfaces peuvent réduire le couplage entre les classes (moins de dépendances).

1. Résumé du chapitre

- Une interface Java déclare un ensemble de méthodes avec leurs signatures. Contrairement à une classe, elle n'est pas implémentée.
- Pour « réaliser » une interface, une classe doit fournir (implémenter) toutes les méthodes que l'interface déclare.
- Les interfaces peuvent réduire le couplage entre les classes (moins de dépendances).
- On peut convertir depuis un type de classe vers un type d'interface, pourvu que la classe réalise l'interface.

1. Résumé du chapitre

- Une interface Java déclare un ensemble de méthodes avec leurs signatures. Contrairement à une classe, elle n'est pas implémentée.
- Pour « réaliser » une interface, une classe doit fournir (implémenter) toutes les méthodes que l'interface déclare.
- Les interfaces peuvent réduire le couplage entre les classes (moins de dépendances).
- On peut convertir depuis un type de classe vers un type d'interface, pourvu que la classe réalise l'interface.
- On a besoin d'un cast pour convertir dans l'autre sens : type interface $\longrightarrow$ type classe.

1. Résumé du chapitre

- Une interface Java déclare un ensemble de méthodes avec leurs signatures. Contrairement à une classe, elle n'est pas implémentée.
- Pour « réaliser » une interface, une classe doit fournir (implémenter) toutes les méthodes que l'interface déclare.
- Les interfaces peuvent réduire le couplage entre les classes (moins de dépendances).
- On peut convertir depuis un type de classe vers un type d'interface, pourvu que la classe réalise l'interface.
- On a besoin d'un cast pour convertir dans l'autre sens : type interface $\longrightarrow$ type classe.
- L'opérateur `instanceof` teste si un objet est d'un certain type.

- « Polymorphisme » signifie que le comportement peut dépendre du type actuel de l'objet.

- « Polymorphisme » signifie que le comportement peut dépendre du type actuel de l'objet.
- Une classe interne est déclarée à l'intérieur d'une autre classe. De telles classes peuvent être appelées «tactiques», elles ne sont pas visibles ailleurs dans le programme.

- « Polymorphisme » signifie que le comportement peut dépendre du type actuel de l'objet.
- Une classe interne est déclarée à l'intérieur d'une autre classe. De telles classes peuvent être appelées «tactiques», elles ne sont pas visibles ailleurs dans le programme.
- Les méthodes d'une classe interne peuvent accéder à des variables qui sont hors de la portée de la classe.

- « Polymorphisme » signifie que le comportement peut dépendre du type actuel de l'objet.
- Une classe interne est déclarée à l'intérieur d'une autre classe. De telles classes peuvent être appelées «tactiques», elles ne sont pas visibles ailleurs dans le programme.
- Les méthodes d'une classe interne peuvent accéder à des variables qui sont hors de la portée de la classe.
- Les variables locales qui peuvent être accédées par une méthode d'une classe interne doivent être déclarées de type `final`.

2. Exs. de révision

R10.1 : Supposez que `C` soit une classe qui réalise les interfaces `I` et `J`. Lesquelles des assignations suivantes nécessitent un cast ?

```
C c = ... ;
I i = ... ;
J j = ... ;
c = i ; //1
j = c ; //2
i = j ; //3
```

2. Exs. de révision

R10.1 : Supposez que `C` soit une classe qui réalise les interfaces `I` et `J`. Lesquelles des assignations suivantes nécessitent un cast ?

```
C c = ... ;
I i = ... ;
J j = ... ;
c = i ; //1
j = c ; //2
i = j ; //3
```

R10.2 : Supposez à nouveau que `C` soit une classe qui réalise les interfaces `I` et `J`. Lesquelles des assignations suivantes vont lancer une exception ?

```
C c = new C();
I i = c; //1
J j = (J)i; //2
C d = (C)i; //3
```

R10.3 : Supposez que la classe `Sandwich` implémente l'interface `Comestible`. Lesquelles des assignations suivantes sont légales :

```
Sandwich sub = new Sandwich();
Comestible e = sub; //1
Rectangle cerealBox = new Rectangle((5,10,20,
Comestible f = cerealBox ; //2
f = (Comestible)cerealBox ; //3
sub = e ; //4
sub = (Sandwich)e; //5
sub = (Sandwich)cerealBox ; //6
```

R10.3 : Supposez que la classe `Sandwich` implémente l'interface `Comestible`. Lesquelles des assignations suivantes sont légales :

```
Sandwich sub = new Sandwich();
Comestible e = sub; //1
Rectangle cerealBox = new Rectangle((5,10,20,
Comestible f = cerealBox ; //2
f = (Comestible)cerealBox ; //3
sub = e ; //4
sub = (Sandwich)e; //5
sub = (Sandwich)cerealBox ; //6
```

R10.4 : Comment un cast comme `(BankAccount)x` diffère-t-il d'un cast de nombres comme `(int)x` ?

R10.3 : Supposez que la classe `Sandwich` implémente l'interface `Comestible`. Lesquelles des assignations suivantes sont légales :

```
Sandwich sub = new Sandwich();
Comestible e = sub; //1
Rectangle cerealBox = new Rectangle((5,10,20,
Comestible f = cerealBox ; //2
f = (Comestible)cerealBox ; //3
sub = e ; //4
sub = (Sandwich)e; //5
sub = (Sandwich)cerealBox ; //6
```

R10.4 : Comment un cast comme `(BankAccount)x` diffère-t-il d'un cast de nombres comme `(int)x` ?

R10.5 : Les classes :
- `Rectangle2D.Double`,
- `Ellipse2D.Double`

- `Line2D.Double`

réalisent l'interface `Shape`. La classe `Graphics2D` dépend de l'interface `Shape`, mais pas des classes rectangle, ellipse et line. Dessiner un diagramme UML qui représente cela.

- `Line2D.Double`

réalisent l'interface `Shape`. La classe `Graphics2D` dépend de l'interface `Shape`, mais pas des classes rectangle, ellipse et line. Dessiner un diagramme UML qui représente cela.

R10.6 : Supposez que `r` soit une réference vers `new Rectangle(5,10,20,30)`. Lesquelles de ces conditions renvoient vrai ? (Indication : vérifiez l'API pour voir quelle interface la classe `Rectangle` réalise.)

```
r instanceof Rectangle
r instanceof Shape
r instanceof Point
r instanceof Object
r instanceof ActionListener
r instanceof Serializable
```

R10.7 : Reprenez les classes de l'exercice R10.5, avec l'interface `Shape`. Celle-ci contient une méthode :

`Rectangle getBounds()`
qui retourne un rectangle contenant entièrement la forme représentée par `Shape`. Considérez l'appel de méthode suivant :

```
Shape s = .... ;
Rectangle r = s.getBounds() ;
```

Expliquez pourquoi ceci est un exemple de polymorphisme.

```
Rectangle getBounds()
```

qui retourne un rectangle contenant entièrement la forme représentée par `Shape`. Considérez l'appel de méthode suivant :

```
Shape s = .... ;
Rectangle r = s.getBounds() ;
```

Expliquez pourquoi ceci est un exemple de polymorphisme.

R10.8 : En Java, un appel de méthode tel que `x.f()` utilise la « détermination tardive » ("late binding"), i.e. que la méthode exacte à appeler dépend du type exact de l'objet auquel réfère `x`. Donnez deux formes d'appel de méthodes qui utilisent la « détermination précoce » ("early binding").

`Rectangle getBounds()`
qui retourne un rectangle contenant entièrement la forme représentée par `Shape`. Considérez l'appel de méthode suivant :

```
Shape s = .... ;
Rectangle r = s.getBounds() ;
```

Expliquez pourquoi ceci est un exemple de polymorphisme.

R10.8 : En Java, un appel de méthode tel que `x.f()` utilise la « détermination tardive » ("late binding"), i.e. que la méthode exacte à appeler dépend du type exact de l'objet auquel réfère `x`. Donnez deux formes d'appel de méthodes qui utilisent la « détermination précoce » ("early binding").

R10.9 : Supposez que vous deviez parcourir un tableau d'employés pour trouver la moyenne et le plus haut salaire. Discutez ce dont vous avez besoin pour utiliser la première implémentation de la classe `DataSet` (qui utilise des objets de type `Measurable`)

De quoi avez-vous besoin pour utiliser la deuxième implémentation ? Lequel de ces deux choix sera le plus aisé ?

De quoi avez-vous besoin pour utiliser la deuxième implémentation ? Lequel de ces deux choix sera le plus aisé ?

R10.10 : Que se passe-t-il si vous ajoutez un objet de type `String` à la première implémentation de `DataSet` ? Que se passe-t-il si vous ajoutez un objet `String` à un objet `DataSet` de la seconde implémentation qui utilise une classe `RectangleMeasurer` ?

De quoi avez-vous besoin pour utiliser la deuxième implémentation ? Lequel de ces deux choix sera le plus aisé ?

R10.10 : Que se passe-t-il si vous ajoutez un objet de type `String` à la première implémentation de `DataSet` ? Que se passe-t-il si vous ajoutez un objet `String` à un objet `DataSet` de la seconde implémentation qui utilise une classe `RectangleMeasurer` ?

R10.11 : Comment allez-vous réorganiser le programme de test qui utilise la classe `RectangleMeasurer` si vous avez besoin de faire `RectangleMeasurer` en une classe de plus haut niveau (i.e. non classe interne : "top-level class") ?

De quoi avez-vous besoin pour utiliser la deuxième implémentation ? Lequel de ces deux choix sera le plus aisé ?

R10.10 : Que se passe-t-il si vous ajoutez un objet de type `String` à la première implémentation de `DataSet` ? Que se passe-t-il si vous ajoutez un objet `String` à un objet `DataSet` de la seconde implémentation qui utilise une classe `RectangleMeasurer` ?

R10.11 : Comment allez-vous réorganiser le programme de test qui utilise la classe `RectangleMeasurer` si vous avez besoin de faire `RectangleMeasurer` en une classe de plus haut niveau (i.e. non classe interne : "top-level class") ?

R10.12 : Comment allez-vous réorganiser le programme de test qui utilise la classe `InterestAdder` si vous avez besoin de faire celle-ci en une classe de plus haut niveau (non classe interne) ?

R10.13 : Qu'est-ce qu'un objet «stratégique» ? Pouvez-vous imaginer un autre objet stratégique utile pour la classe `DataSet` ? (Indication : voir l'exercice P10.8).

R10.13 : Qu'est-ce qu'un objet «stratégique» ? Pouvez-vous imaginer un autre objet stratégique utile pour la classe `DataSet` ? (Indication : voir l'exercice P10.8).

R10.14 : Que se passe-t-il si une classe interne essaie d'accéder à une variable locale non-`final` ? Essayez-le et expliquez.

R10.13 : Qu'est-ce qu'un objet «stratégique» ? Pouvez-vous imaginer un autre objet stratégique utile pour la classe `DataSet` ? (Indication : voir l'exercice P10.8).

R10.14 : Que se passe-t-il si une classe interne essaie d'accéder à une variable locale non-`final` ? Essayez-le et expliquez.

R10.15 : Dans la classe de plus niveau et sa classe interne suivantes, quelles variables la méthode `f` peut accéder ?

```
public class T
{
  public void m(final int x, int y)
  {
    int a ; final int b ;
    class C implements I
    {
      public void f() {...}
```


```
    }
    final int c;
    ...
  }
  private int t;
}
```

3. Exs. programmation

P10.1 : Reprenez la classe `Die` de la série 8 précédente. Faites-la implémenter l'interface `Measurable`. Générez des dés, lancez-les, et ajoutez-les à la première implémentation de la classe `DataSet`. Affichez la moyenne.

3. Exs. programmation

P10.1 : Reprenez la classe `Die` de la série 8 précédente. Faites-la implémenter l'interface `Measurable`. Générez des dés, lancez-les, et ajoutez-les à la première implémentation de la classe `DataSet`. Affichez la moyenne.

P10.2 : Définissez la classe `Quiz` qui implémente l'interface `Measurable`. Un "quiz" a un score et une lettre (telle «B+»). Utilisez la première implémentation de la classe `DataSet` pour fabriquer une collection de "quizzes". Affichez le score moyen, ainsi que le quiz ayant le meilleur score (à la fois la meilleure lettre et le plus haut score).

3. Exs. programmation

P10.1 : Reprenez la classe `Die` de la série 8 précédente. Faites-la implémenter l'interface `Measurable`. Générez des dés, lancez-les, et ajoutez-les à la première implémentation de la classe `DataSet`. Affichez la moyenne.

P10.2 : Définissez la classe `Quiz` qui implémente l'interface `Measurable`. Un "quiz" a un score et une lettre (telle «B+»). Utilisez la première implémentation de la classe `DataSet` pour fabriquer une collection de "quizzes". Affichez le score moyen, ainsi que le quiz ayant le meilleur score (à la fois la meilleure lettre et le plus haut score).

P10.3 : Définissez une classe `Person`. Une personne a un nom, et une hauteur (en cm). Utilisez la seconde implémentation de la classe `DataSet` pour fabriquer une collection d'objets de type `Person`. Affichez la moyenne des tailles et le nom de la personne de plus grande taille.

P10.4 : Modifiez la première implémentation de `DataSet` pour aussi calculer l'élément dont les données sont minimales.

P10.4 : Modifiez la première implémentation de `DataSet` pour aussi calculer l'élément dont les données sont minimales.

P10.5 : Modifiez la seconde implémentation de `DataSet` pour aussi calculer l'élément dont les données sont minimales.

P10.4 : Modifiez la première implémentation de `DataSet` pour aussi calculer l'élément dont les données sont minimales.

P10.5 : Modifiez la seconde implémentation de `DataSet` pour aussi calculer l'élément dont les données sont minimales.

P10.6 : En utilisant un objet `Measurer` différent, parcourir un ensemble d'objets `Rectangle` pour trouver le rectangle de périmètre maximal.

P10.4 : Modifiez la première implémentation de `DataSet` pour aussi calculer l'élément dont les données sont minimales.

P10.5 : Modifiez la seconde implémentation de `DataSet` pour aussi calculer l'élément dont les données sont minimales.

P10.6 : En utilisant un objet `Measurer` différent, parcourir un ensemble d'objets `Rectangle` pour trouver le rectangle de périmètre maximal.

P10.7 : Améliorer la classe `DataSet` de façon à ce qu'elle puisse être utilisée soit avec un objet de type `Measurer` ou pour accéder à des objets de type `Measurable`. Indication : Ajoutez un constructeur par défaut qui implémente un `Measurer` qui peut accéder à des objets `Measurable`.

P10.4 : Modifiez la première implémentation de `DataSet` pour aussi calculer l'élément dont les données sont minimales.

P10.5 : Modifiez la seconde implémentation de `DataSet` pour aussi calculer l'élément dont les données sont minimales.

P10.6 : En utilisant un objet `Measurer` différent, parcourir un ensemble d'objets `Rectangle` pour trouver le rectangle de périmètre maximal.

P10.7 : Améliorer la classe `DataSet` de façon à ce qu'elle puisse être utilisée soit avec un objet de type `Measurer` ou pour accéder à des objets de type `Measurable`. Indication : Ajoutez un constructeur par défaut qui implémente un `Measurer` qui peut accéder à des objets `Measurable`.

P10.8 : Définissez une interface `Filter` comme suit :

```
public interface Filter
{
  boolean accept(Object x);
}
```

Modifiez la seconde implémentation de `DataSet` pour utiliser à la fois des objets de type `Measurer` et `Filter`. Seulement les objets que le filtre accepte peuvent être accédés. Vérifiez votre modification en procédant sur une collection de comptes `BankAccount` en filtrant les comptes dont le solde ("balance") est inférieur à 1$.

```
public interface Filter
{
  boolean accept(Object x);
}
```

Modifiez la seconde implémentation de `DataSet` pour utiliser à la fois des objets de type `Measurer` et `Filter`. Seulement les objets que le filtre accepte peuvent être accédés. Vérifiez votre modification en procédant sur une collection de comptes `BankAccount` en filtrant les comptes dont le solde ("balance") est inférieur à 1$.

P10.9 : Écrivez un programme qui utilise un timer pour afficher l'heure courante chaque seconde. Indication : le code suivant affiche l'heure courante :

```
Date now = new Date();
System.out.println(now);
```

P10.10 : Améliorer le programme précédent pour incrémenter un solde bancaire une fois par seconde, ceci par l'ajout d'un

second compte bancaire ayant une balance de 2.0$ et un timer qui l'incrémente toutes les 2 secondes.

second compte bancaire ayant une balance de 2.0$ et un timer qui l'incrémente toutes les 2 secondes.

P10.11 : Vérifiez la définition de l'interface `Comparable` dans l'API. Modifiez la classe `DataSet` pour accepter des objets de type `Comparable`. Avec cette interface, il n'y a plus de sens à calculer la moyenne. La classe `DataSet` devrait mémoriser les données de valeur minimale et maximale. Tester votre classe `DataSet` modifiée en ajoutant un nombre d'objets `String`. (La classe `String` implémente l'interface `Comparable`).

second compte bancaire ayant une balance de 2.0$ et un timer qui l'incrémente toutes les 2 secondes.

P10.11 : Vérifiez la définition de l'interface `Comparable` dans l'API. Modifiez la classe `DataSet` pour accepter des objets de type `Comparable`. Avec cette interface, il n'y a plus de sens à calculer la moyenne. La classe `DataSet` devrait mémoriser les données de valeur minimale et maximale. Tester votre classe `DataSet` modifiée en ajoutant un nombre d'objets `String`. (La classe `String` implémente l'interface `Comparable`).

P10.12 : Modifiez les classes `Coin` et `Purse` de la série 8 pour qu'elles implémentent l'interface `Comparable`.