Programmation Orientee Objet

Classes Abstraites et Intefaces

Omar El Midaoui

Plan

- Classes abstraites
- Interfaces

Classes abstraites

Classes abstraites

- **Utilité :**
  - Définir des concepts incomplets qui devront être implémentés dans les sous classes
  - Factoriser le code

```
public abstract class lesFrome {
protected double x, y;
public void deplacer(double dx, double dy)
{
x += dx; y+= dy;
}
public abstract double perimetre();
public abstract double surface();
}
```

Classes abstraites

- **classe abstraite** : classe non instanciable, c'est à dire qu'elle n'admet pas d'instances directes.
  - *Impossible de faire* **new ClassAbstraite(...)**
- **opération abstraite** : opération n'admettant pas d'implémentation au niveau de la classe dans laquelle elle est déclarée, on ne peut pas dire comment la réaliser.
- Une classe pour laquelle au moins une opération abstraite est déclarée est une classe abstraite (l'inverse n'est pas vrai).
- Les opérations abstraites sont particulièrement utiles pour mettre en oeuvre le polymorphisme.

Classes abstraites

- Une classe abstraite est une description d'objets destinée à être héritée par des classes plus spécialisées.
- Pour être utile, une classe abstraite doit admettre des classes descendantes **concrètes**.
- Toute classe **concrète** sous-classe d'une classe abstraite doit concrétiser toutes les opérations abstraites de cette dernière.
- Une classe abstraite permet de regrouper certaines caractéristiques communes à ses sous-classes et définit un comportement minimal commun.
- La factorisation optimale des propriétés communes à plusieurs classes par généralisation nécessite le plus souvent l'utilisation de classes abstraites.

Interfaces

Interfaces

- Dans certaines situations, il est important que différents groupes de programmeurs soient d'accord sur un contrat qui décrit comment leurs programmes interagissent.
- Chaque groupe doit être capable d'écrire son code sans avoir les détails du code des autres groupes.
- Les interfaces de Java servent à cet objectif.

Déclaration d'une interface

- Une interface est une collection d'opérations utilisée pour spécifier un service offert par une classe.
- Une interface peut être vue comme une classe abstraite sans attributs et dont toutes les opérations sont abstraites.

Dessinable.java

```
import java.awt.*;
public interface Dessinable {
public void dessiner(Graphics g);
void effacer(Graphics g);
}
```

- Toutes les méthodes sont abstraites
- Elles sont implicitement publiques
- Possibilité de définir des attributs à condition qu'il s'agisse d 'attributs de type primitif
- Ces attributs sont implicitement déclarés comme static final

Réalisation d'une interface

- Une interface est destinée à être réalisée (implémentée) par d'autres classes (celles-ci en héritent toutes les descriptions et concrétisent les opérations abstraites).
  - Les classes réalisantes s'engagent à fournir le service spécifié par l'interface

Déclaration d'une interface

- Composants (dans l'ordre) :
  1. Modificateurs (optionnels)
  2. Mot-clé interface' suivi du nom de l'interface (obligatoire)
  3. Mot-clé extends' suivi d'une liste de noms d'interfaces (optionnel)
  4. Corps de déclaration entouré par { et }

```
public interface InterfRegroupee extends Interf1,
Interf2 {
double E = 2.718282; // base des logarithmes
void faitQqChose (int i, double x);
int faitAutreChose (String s);
}
```

Réalisation d'une interface

- De la même manière qu'une classe étend sa super-classe elle peut de manière optionnelle implémenter une ou plusieurs interfaces
  - dans la définition de la classe, après la clause extends nomSuperClasse, faire apparatre explicitement le mot clé implements suivi du nom de l'interface implémentée

```
class RectangleDessinable extends Rectangle implements Dessinable {
public void dessiner(Graphics g){
g.drawRect((int) x, (int) y, (int) largeur, (int) hauteur);
}
public void effacer(Graphics g){
g.clearRect((int) x, (int) y, (int)largeur, (int) hauteur);
}

}
```

- si la classe est une classe concrète elle doit fournir une implémentation (un corps) à chacune des méthodes abstraites définies dans l'interface (qui doivent être déclarées publiques)

Réalisation d'une interface

- Une classe JAVA peut implémenter simultanément plusieurs interfaces
- Pour cela la liste des noms des interfaces à implémenter séparés par des virgules doit suivre le mot clé implements

```
class RectangleDessinable extends Rectangle implements Dessinable,
Comparable {
public void dessiner(Graphics g){
g.drawRect((int) x, (int) y, (int) largeur, (int) hauteur);
}
public void effacer(Graphics g){ // Méthode de l'interface Dessinable
g.clearRect((int) x, (int) y, (int)largeur, (int) hauteur);
}
public int compareTo(Object o) { // Méthode de l'interface Comparable
if (o instanceof Rectangle)
...
} }
```

Interfaces

- En Java, une interface est un type (similaire à une classe) qui contient de constantes, les signatures et la valeur de retour des méthodes (mais pas leur corps de déclaration), et les classes imbriquées.
- Interfaces ne peuvent pas être instanciées. En revanche, elles peuvent être étendues et surtout implémentées.
- Un champ d'une classe peut être de type Interface
- Une variable dans une méthode peut être de type Interface

Déclaration d'une interface

- Une interface peut étendre plus d'une interface (une sorte d'héritage multiple).
- Toutes les méthodes d'une interface sont implicitement déclarées public (donc, le modificateur public peut être omis).
- Une interface peut contenir de déclarations de constantes. Elles sont implicitement déclarées
- public static final (donc, ces modificateurs peuvent être omis.)

Déclaration d'une interface

- **Exemple**

```
public interface Comparable {
// this et other doivent être des instances
// de la meme classe
// retourne 1, 0, -1 si this est plus grand,
// egal, ou plus petit que other
public int compareTo(Object other);
}
```

- Toute classe peut implémenter Comparable, si il y a une facon de comparer la ses objets.
- Si nous savons que la classe implémente Comparable, alors nous savons que nous pouvons comparer ses objets.

Héritage d'interfaces

- De la même manière qu'une classe peut avoir des sous-classes, une interface peut avoir des "sous-interfaces"
- Une sous interface
  - hérite de toutes les méthodes abstraites et des constantes de sa "superinterface"
  - peut définir de nouvelles constantes et méthodes abstraites

```
interface Set extends Collection
{
...
}
```

- Une classe qui implémente une interface doit implémenter toutes les méthodes abstraites définies dans l'interface et dans les interfaces dont elle hérite.

Implémentation d'une interface

```
class Rectangle implements Comparable {
public int largeur = 0;
public int hauteur = 0;
...
public int getSurface() { return largeur * hauteur; }
public int compareTo(Object other) {
Rectangle autreRect = (Rectangle)other;
if (this.getSurface() < autreRect.getSurface())
return -1;
else if (this.getSurface() > autreRect.getSurface())
return 1;
else
return 0;
}
}
```

Exercice

- 1. Déclarez une classe StringInverse qui contient une string inversée en implémentant l'interface suivante :

```
public interface CharSequence {
char charAt(int index);
int length();
CharSequence subSequence(int debut, int fin);
String toString();
}
```

Réponse

```
public class StringInverse implements CharSequence {
private String chaine;
public StringInverse(String str) {
char[] tab = new char[str.length()];
int j = str.length() - 1;
for (int i = 0; i < tab.length; i++)
tab[i] = str.charAt(j--);
chaine = new String(tab);
}
```

Réponse

```
public char charAt(int index) {
return chaine.charAt(index);
}
public int length() {
return chaine.length();
}
public CharSequence subSequence(int debut, int fin) {
return (CharSequence)chaine.subSequence(debut, fin);
}
public String toString() {
return chaine;
}
}
```