Interfaces abstraites

Mise en œuvre en Java

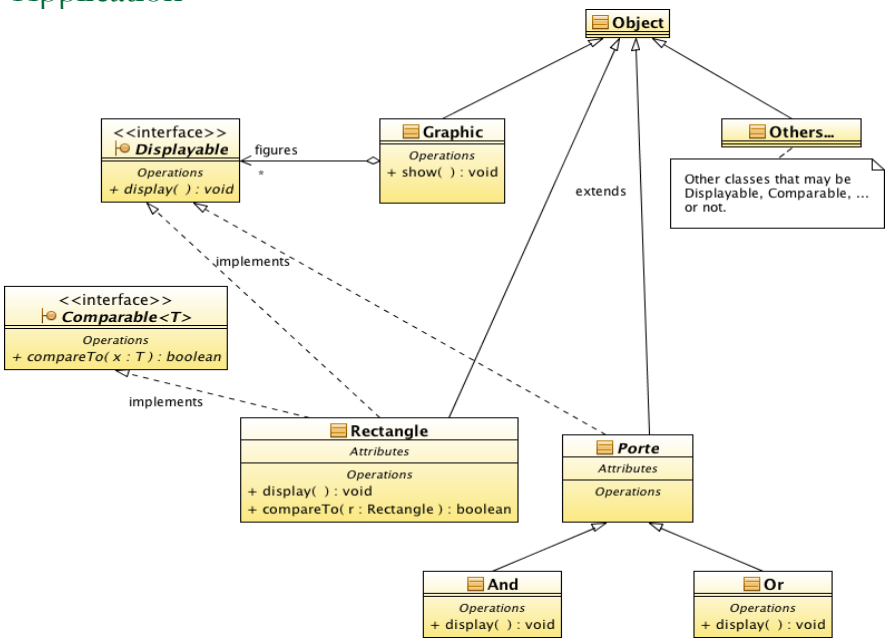
La notion d'interface abstraite

- A l'extrême des classes abstraites
- « pures protocoles » (pas d'implémentation)
  - pas de structure (pas de variables d'instance)
  - que des méthodes abstraites
- Peuvent être génériques, comme les classes
- Spécifient des « capacités » transversales aux classes qu'elles implémentent:
  - relation **`implements`** (n-aire)
  - en plus de **`extends`** (unaire)

- Exemple

```
interface Displayable {
   void display();
   void displayAt(Point p);
   …
}
```

Application



Implémentation d'interface

Implémentation d'interfaces: par les classes

```
public class Rectangle implements Displayable {
   public void display() {//implantation...}
   public void displayAt(Point p) {//implantation...}
...}

public class And
   extends Porte
   implements Displayable
   // implantation de display et displayAt...

public class Or
   extends Porte
   implements Displayable
   //implantation de display et displayAt...
```

Exemples du langage et de bibliothèques

- `Comparable<E>` (pour `java.util`)
  - des objets qui offrent une relation d'ordre (méthode `compareTo`)
- `Iterable<E>`
  - Des séquences d'objets (tableaux, listes, …) : `next()/hasNext()`
- `Cloneable`
  - objets dont on peut obtenir une copie par la méthode `clone()`
- `Serializable` (pour `java.io`)
  - objets que l'on peut sauvegarder/lire sur fichier par :
    `readObject(ObjectInputStream in)`
    `writeObject(ObjectOutputStream out)`
- Protocoles d'écoutes d'évènements des composants d'interfaces graphiques (`java.awt, javax.swing`)

```
public interface ActionListener extends EventListener{
 void actionPerformed(ActionEvent e);}
```


Interface comme type

- Tout comme une classe, une interface définit un type d'objets
- Typage souple : un objet est du type de l'interface si sa classe l 'implémente (directement ou par héritage)
- Les règles du polymorphisme s'applique

```
// interface abstraite => typage souple
Rectangle r;
And a;
Displayable x;

//affectation polymorphe
x=r;
x=a;
x=pick();
```

Interface comme type

- Comme type d'éléments de tableaux (et de collections)

```
public class Graphic {
   protected Displayable[] figures;
   public void add(Displayable fig) {…}
   public void show() {
     for (Displayable fig : figures) fig.display();
   }
}
```

- Comme type de paramètre

```
Graphic g;

g.add(r);
g.add(a);
g.add(x);
```


Interface comme type

```
public class Rectangle implements Displayable,
   Comparable<Rectangle> {
 public double surface() {…}
 public void display() {…}
 public int compareTo(Rectangle r) {
   return (int)(this.surface() - r.surface());
 }
}
public class Pavage {
 protected Rectangle[] rectangles;
 public void sort() {
   Arrays.sort(rectangles); // <= Comparable
 }
}
```

Interface et classe abstraite

- Une classe
  - doit implémenter tout le protocole de l'interface
  - sauf si elle est abstraite :
    elle peut alors laisser abstraites certaines méthodes

```
public abstract class Porte implements Displayable {
   // display() remains abstract
}
public class And extends Porte {
     public void display() { //code…}
}
public class Or extends Porte {
     public void display() { //code…}
}
```


Hiérarchie de classes et d'interfaces

- Une classe est sous-classe d'une seule classe mais peut implémenter plusieurs interfaces ("héritage multiple" de classes abstraites en C++)

```
public class ArrayList<E>
 extends AbstractList<E>
 implements List<E>, Cloneable, Serializable
```

- Une interface peut étendre (`extends`) une ou plusieurs interfaces

```
public interface MouseInputListener //AWT
   extends MouseListener, MouseMotionListener {...}
```