Le polymorphisme

Cours 6 / GL / LP DA2I

Cédric Lhoussaine

2019-2020

Outline

Les usages des interfaces

Les interfaces servent

- à formuler un contrat sur le comportement d'un objet
- à fournir un type commun pour rassembler divers objets ayant un comportement proche
- à exprimer divers points de vue sur un même objet

Notion de contrat

`class Rectangle implements Forme...`

- `Rectangle` possède au moins toutes les méthodes de `Forme`
- il faut écrire dans `Rectangle` le code des méthodes de `Forme`
- `Rectangle` peut avoir ses propres méthodes (publiques ou privées)
- la structure (attributs) de `Rectangle` peut être quelconque

$\rightarrow$ spécifie ce que tout Rectangle doit savoir faire *en tant que* `Forme`.

Définition d'un type commun

```
public interface Recyclable {
    void recycler() ;
}
class Verre implements Recyclable {
    public void recycler() { this.fondre(500) ; }
    private void fondre(int temperature) { ... }
}
class Papier implements Recyclable {
    public void recycler() { this.fairePate() ; }
    private void fairePate() { ... }
}
```

Usages: type commun

- `Verre` et `Papier` sont deux *sortes* d'objets `Recyclable`
- si on veut stocker ensemble du papier et du verre, on peut utiliser comme type commun `Recyclable`:

```
Verre v = ... ;
Papier p = ... ;
Recyclable [] poubelle = new Recyclable[10] ;
poubelle[0] = v ;
pouvelle[1] = p ;
```

Usages: point de vue

```
Chat c = new Chat("Tom") ;
c.jouer() ;
// à la maison
Compagnon comp = c ;
comp.jouer() ;
...
// chez le vétérinaire
    Mammifere m = (Mammifere) comp ;
// attention, comp pourrait être un chien
// mais aussi un poisson rouge !
System.out.println(m.poids()) ;
```

Le polymorphisme

Définition

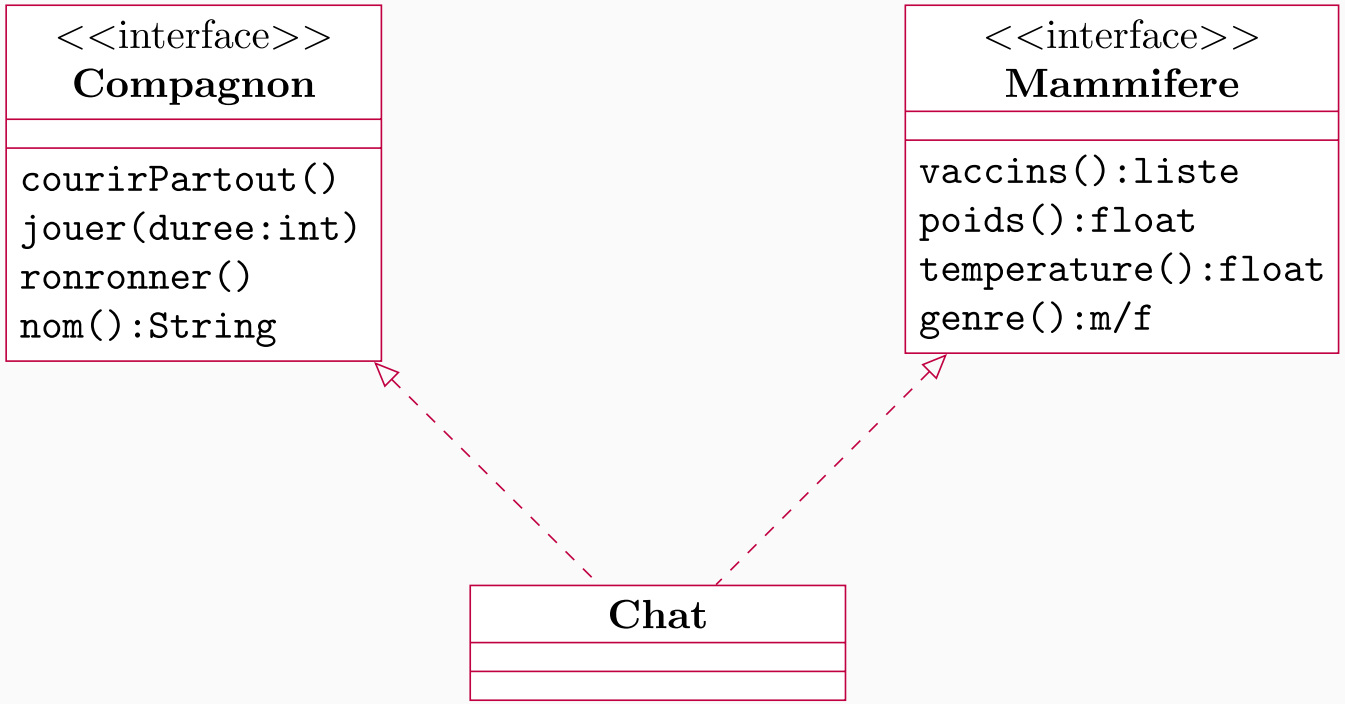
Les interfaces aident à réaliser des abstractions grâce à la notion de polymorphisme.

Définition du polymorphisme

- capacité d'un objet (= classe) à prendre plusieurs formes (= types)
- capacité d'une forme (= type) à englober plusieurs objets (= classes)

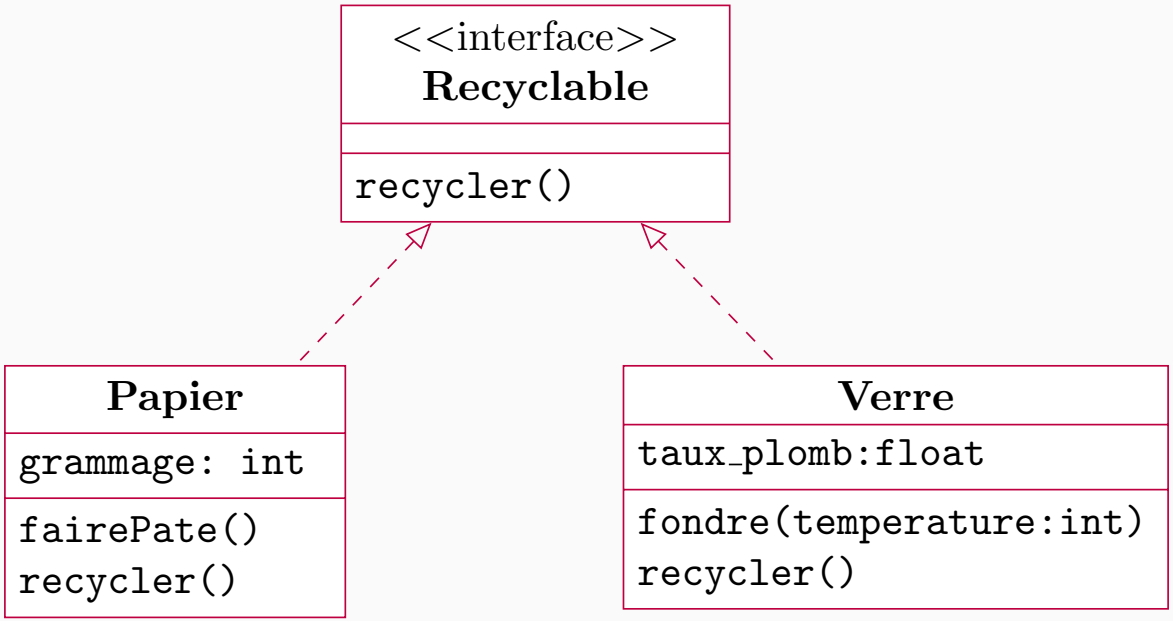
Exemple de polymorphisme

un objet → plusieurs formes



Exemple de polymorphisme

plusieurs objets → une forme



Variantes de polymorphisme

Variantes

- polymorphisme paramétrique: *surcharge* un nom de méthode avec plusieurs paramètrages

Variantes

- polymorphisme paramétrique: *surcharge* un nom de méthode avec plusieurs paramètrages
- polymorphisme d'affectation: un type de variable instanciable par plusieurs classes

```
Forme f = new Rectangle(5, 3);
```

Variantes

- polymorphisme paramétrique: *surcharge* un nom de méthode avec plusieurs paramètrages
- polymorphisme d'affectation: un type de variable instanciable par plusieurs classes

```
Forme f = new Rectangle(5, 3);
```

- polymorphisme de méthode: une méthode $\rightarrow$ plusieurs codes selon la classe

```
Forme f = ... ; float x = f.surface() ;
```

Polymorphisme paramétrique ou surcharge

surcharge → méthodes de même nom

- elles retournent le *même type*
- leurs paramètres diffèrent en nombre ou en type
- possible aussi pour les constructeurs en Java

```
float moyenne(int a, int b) ;
float moyenne(float a, float b) ;
float moyenne(int a, float b) ;
float moyenne(float [] t) ;
```

Polymorphisme paramétrique ou surcharge

surcharge → méthodes de même nom

- elles retournent le *même type*
- leurs paramètres diffèrent en nombre ou en type
- possible aussi pour les constructeurs en Java

```
float moyenne(int a, int b) ;
float moyenne(float a, float b) ;
float moyenne(int a, float b) ;
float moyenne(float [] t) ;
```

- la méthode utilisée est choisie en fonction du nombre ou du type des paramètres
- nombre : pas d'ambiguïté
- type : attention aux conversions possibles !

Rappel: lien avec le typage

Les classes qui implémentent des interfaces définissent des sous-types de ceux définis par les interfaces

```
Rectangle r = new Rectangle(5,3) ;
Cercle c = new Cercle(1) ;
Forme f1 = r, f2 = c ; // polymorphisme d'affectation
f1.surface() // -> 15.0 polyporphisme de méthode
f2.surface() // -> 3.14159 polyporphisme de méthode
```

Rappel: lien avec le typage

Les classes qui implémentent des interfaces définissent des sous-types de ceux définis par les interfaces

```
Rectangle r = new Rectangle(5,3) ;
Cercle c = new Cercle(1) ;
Forme f1 = r, f2 = c ; // polymorphisme d'affectation
f1.surface() // -> 15.0 polyporphisme de méthode
f2.surface() // -> 3.14159 polyporphisme de méthode
```

L'appel de la "bonne" méthode dépend de la classe d'appartenance de l'objet (et non de son type !) → LIAISON RETARDÉE

Fonctionnement

Comment le code peut-il correctement s'exécuter ?

- transtypages explicites entre objets: autorisés à la compilation

```
Truc x = (Truc) (new Machin());
```

- transtypage implicite d'un type quelconque vers un type plus général: toujours possible

```
Object x = new Truc();
```

$\rightarrow$ comment ensuite appeler la bonne méthode ?

Mécanismes mis en œuvre

- vérification des types (compilation):

```
Truc x = ... ;
```

  seuls les appels de méthodes ou d'attributs définis pour la classe `Truc` sont autorisés;
- vérification des classes (exécution): on ne peut mettre dans `x` que des instances de `Truc` ou d'une *sorte de* `Truc`;
- liaison retardée (exécution): le code est choisi dynamiquement en fonction de la classe de l'instance référencée par `x`.

Types abstraits de données

Les listes

```
public interface Liste<T> {
    boolean estDernier() ;
    boolean contient(T c) ;
    T contenu() ;
    Liste<T> elementSuivant() ;
    void placerALaFin(T c);
    ...
}
```

→ `public class ElementListe<T> implements Liste<T>` en remplaçant partout `int` par `T` et `ElementListe` par `Liste`.

Utilisation avec un type commun

type abstrait de données + interfaces $\rightarrow$ large variété d'utilisations !

```
// déclaration d'une variable de type Liste
Liste<Recyclable> poubelle ;
// instanciation à partir de la classe ElementListe
poubelle = new ElementListe<Recyclable>();
// ajout d'éléments implémentant Recyclable
poubelle.ajouterALaFin(new Papier()) ;
poubelle.ajouterALaFin(new Verre()) ;
...
```

Emacs 26.3 (Org mode 9.1.9)