Héritage (suite)

"Informatique : Alliance d'une science inexacte et d'une activité humaine faillible.

Luc Fayard (journaliste)

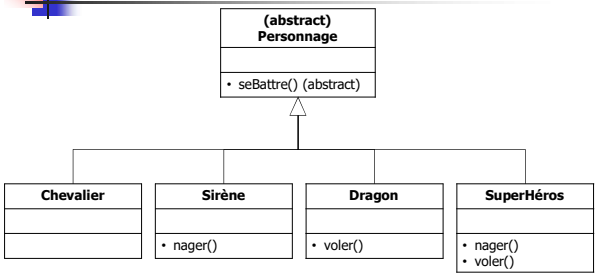
L'héritage

- Traduit le principe de Généralisation / Spécification
- Permet une classification hiérarchique des classes et objets
- Principe
  - un objet spécialisé bénéficie (ou hérite) des caractéristiques d'un objet plus général auquel il ajoute ses propres caractéristiques.

Classe abstraite

- Possibilité de déclarer une méthode sans la définir
  - on déclare son prototype
  - on ne donne pas son contenu
- La classe est alors incomplète
  - il est interdit d'instancier un objet de cette classe
- La classe devient **abstraite**

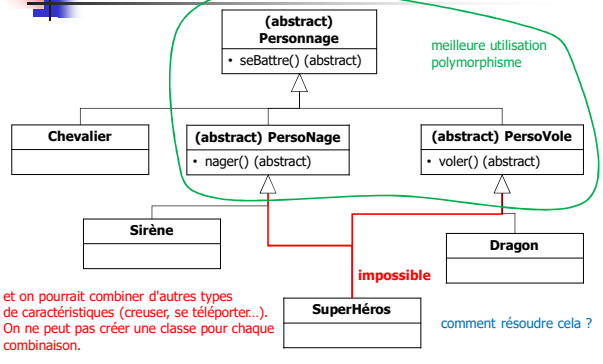
Exemple



Tous les Personnage peuvent seBattre() -> utilisation du polymorphisme (c'est ce qu'on veut en proposant cette hiérarchie)

Mais il y a d'autres caractéristiques communes à plusieurs classes, et actuellement pas de polymorphisme et rien n'est imposé aux nouvelles classes qui seraient ajoutées

Exemple: autre hiérarchie



et on pourrait combiner d'autres types de caractéristiques (creuser, se téléporter...). On ne peut pas créer une classe pour chaque combinaison.

comment résoudre cela ?

Classe abstraite vs Interface

- Classe normale: toutes les méthodes sont définies (on peut créer des instances)
- Classe abstraite: une classe pas complètement implémentée (on ne peut pas créer d'instance)
- Interface: "contrat" auquel une classe doit se conformer
  - Ensemble de méthodes qu'une classe utilisant cette interface doit redéfinir

7

Interface et Implémentation

- Redéfinition de toutes les méthodes d'une interface. La classe **implémente** l'interface.
- Classe abstraite remplacée par interface
- Classe implémentant: instruction implements
- Toutes les méthodes d'une interface sont (implicitement) public et abstract
- Tout attribut d'une interface est static et final

8

Interface: exemple (1)

```
public abstract class Piece {
    private int x,y;
    public abstract void afficher();
}
```

```
public interface Deplacement {
    public void avancer();
    public void reculer();
}
```

```
public class Pion extends Piece{
    @Override
    public void afficher() {
    }
}
```

hérite de Piece, redéfinit afficher()

```
public class Fou implements Deplacement{
    @Override
    public void avancer() {
    }
    @Override
    public void reculer() {
    }
}
```

implémente Deplacement, redéfinit avancer(),reculer()

```
public class Tour extends Piece implements Deplacement{
    @Override
    public void avancer() {
    }
    @Override
    public void reculer() {
    }
    @Override
    public void afficher() {
    }
}
```

hérite de Piece et implémente Deplacement, redéfinit afficher(), avancer(), reculer()

9

Interface: exemple (2)

```
public abstract class Piece implements Deplacement {
    private int x,y;
    public abstract void afficher();
}
```

```
public interface Deplacement {
    public void avancer();
    public void reculer();
}
```

une classe abstraite peut implémenter une interface

```
public class Tour extends Piece {
    @Override
    public void avancer() {
    }
    @Override
    public void reculer() {
    }
    @Override
    public void afficher() {
    }
}
```

on hérite de l'implémentation

10

Interface: exemple (3)

```
public abstract class Piece implements Deplacement{
    private int x,y;
    public abstract void afficher();
    @Override
    public void avancer() {
    }
}
```

```
public interface Deplacement {
    public void avancer();
    public void reculer();
}
```

```
public class Tour extends Piece {
    @Override
    public void reculer() {
    }
    @Override
    public void afficher() {
    }
}
```

on hérite de l'implémentation et des méthodes de la classe mère: avancer() a déjà été définie dans la classe mère, on peut soit l'utiliser directement, soit la redéfinir comme dans un héritage normal

11

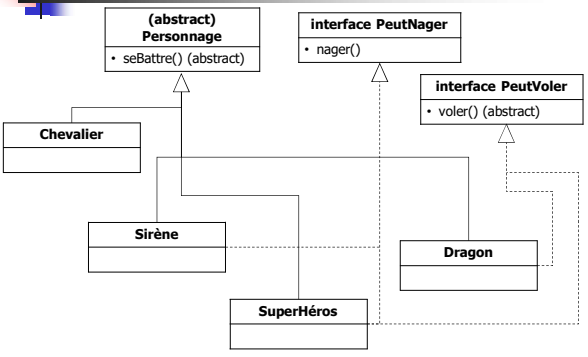
Interface ou classe abstraite ?

- Une classe n'**hérite** que d'**une** autre classe
- Une classe peut **implémenter plusieurs** interfaces
- Une interface peut étendre plusieurs interfaces

- Interface
  - modification plus facile
    - pour une classe existante afin qu'elle implémente une nouvelle interface
  - plus appropriée pour ajouter de nouvelles fonctionnalités "optionnelles" aux classes
- Classe abstraite
  - l'héritage devient un avantage s'il faut modifier la classe abstraite
    - toutes les sous-classes héritent automatiquement de la nouvelle méthode

12

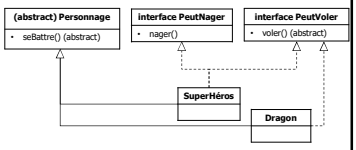
Exemple Personnage



13

Interface et polymorphisme

- Le polymorphisme fonctionne toujours avec les interfaces



Quelles instructions sont correctes ?

Dragon d = new Dragon(); **oui**
d.seBattre(); **oui**
d.voler(); **oui**

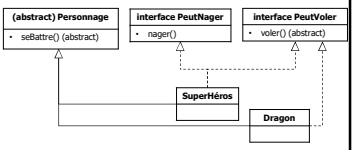
Personnage p = new SuperHeros(); **oui**
p.seBattre(); **oui**
p.voler(); **non p est une ref sur Personnage, ne voit que les méthodes de Personnage**

PeutNager x = new SuperHeros(); **oui**
x.seBattre(); **non p est une ref sur PeutNager, ne voit que les méthodes de PeutNager**
x.nager(); **oui**

14

Interface et polymorphisme

- Le polymorphisme fonctionne toujours avec les interfaces



Qu'affichent les instructions ?

Dragon d = new Dragon();
Personnage p = new SuperHeros();
PeutNager x = new SuperHeros();

System.*out*.println(d instanceof Personnage); **true**

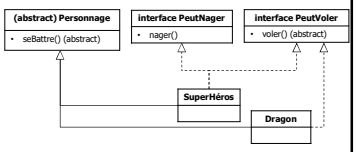
System.*out*.println(p instanceof PeutNager); **true**

System.*out*.println(x instanceof SuperHeros); **true**

15

Interface et polymorphisme

- Le polymorphisme fonctionne toujours avec les interfaces



Quelles instructions sont correctes ?

Dragon d = new Dragon();
Personnage p = new SuperHeros();
PeutNager x = new SuperHeros();

SuperHeros s = (SuperHeros)x; **oui**
s.seBattre(); **oui**
s.nager(); **oui**
s.voler(); **oui**

PeutVoler v = (PeutVoler)p; **oui**
v.seBattre(); **non**
v.nager(); **non**
v.voler(); **oui**