

4. Redéfinition et surcharge

- Surcharge = plusieurs méthodes qui portent le même nom mais qui ont une signature différente ;
- Elles peuvent se trouver dans la même classe ou dans une classe et ses descendantes ;

173

```
public class A  
{ public void methodeX(int i)  
{ ... }  
}
```

Surcharge

```
public class B extends A  
{ public void methodeX(double i)  
{ ... }  
}
```

B possède 2 méthodes

174

Règle 1

- Redéfinition = possibilité de redéfinir dans une sous-classe une méthode qui existe déjà dans la super-classe (même signature) ;
- L'interpréteur cherche la méthode invoquée à partir de la classe concernée et remonte jusqu'à la première rencontrée ;

175

```
public class A  
{ public void methodeX(int i)  
{ ... }  
}
```

Surcharge

```
public class B extends A  
{ public void methodeX(double i)  
{ ... }  
}
```

B possède 2 méthodes

Redéfinition

```
public class C extends A  
{ public void methodeX(int i)  
{ ... }  
}
```

C possède 1 seule méthode

176

Exemple

- On considère 2 classes liées par héritage :
 - class Employe { ... }
 - class Representant extends Employe { ... }
- La classe représentant redéfinit la méthode calculSalaire déjà décrite pour un employé ;
-

177

5. Polymorphisme

- Vient du grec et signifie : qui peut prendre plusieurs formes.
- En POO : aptitude d'un même message à déclencher des opérations différentes, selon l'objet auquel il est destiné ;

178

- Exemple :

```
int type;
float salaire;
Employe e;

System.out.print ("Type ? (1:Employe - 2:Representant) ");
type = Clavier.lireInt();
if (type == 1)
{
    e = new Employe("Dupond", 1000);
}
else
{
    e = new Representant("Dupond",1000, 3000);
}
System.out.println ("Salaire : " + e.calculSalaire() + " euros");
```

179

- Exemple :

```
int type;
float salaire;
Employe e;           /* Objet et sa classe à déclarer */

System.out.print ("Type ? (1:Employe - 2:Representant) ");
type = Clavier.lireInt();
if (type == 1)
{
    e = new Employe("Dupond", 1000);
}
else
{
    e = new Representant("Dupond",1000, 3000);
}
System.out.println ("Salaire : " + e.calculSalaire() + " euros");
```

180

A l'exécution du programme, le code de la méthode "calculSalaire" à lancer est déterminé suivant la classe de l'objet !!! (Type réel de l'objet pas sa référence)

Remarque:

Pour utiliser le polymorphisme, il faut réunir les conditions suivantes :

- Avoir une arborescence d'héritage ;
- L'utiliser par une référence d'objet de la classe de base ;
- Avoir 2 méthodes de même signature définies l'une dans la classe de base, l'autre dans la classe dérivée ;

181

Remarque (Suite)

- Et si, pour un représentant, on veut faire ensuite appel à une méthode propre à la sous-classe ?
`e.methPropreRepr();`
- Solution : cast vers un objet de la sous-classe
`((Representant)e).methPropreRepr();`

182