Qualité de développement

CM3-3: Java, polymorphisme

Mickaël Martin Nevot

 $\overline{\text{V2.2.1}}$



Cette œuvre de Mickaël Martin Nevot est mise à disposition sous licence Creative Commons Attribution - Utilisation non commerciale - Partage dans les mêmes conditions.

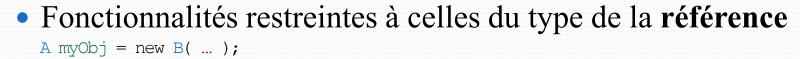
Qualité de développement

- Prés.
- Java bas. П.
- III. Obj.
- IV. Hérit.
- V. POO
- Excep. VI.
- VII. Poly.
- Thread VIII.
- IX. Java av.
- X. Algo. av.
- APP XI.
- GL XII.

B

Polymorphisme

- Surclassement (à la compilation) :
 - Vu comme un objet du type de la **référence**

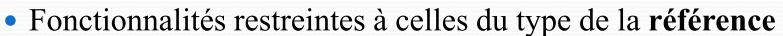


- Liaison dynamique (à l'exécution) :
 - Méthode de la classe effective de l'objet qui est exécuté myObj.meth1(...);
- Downcasting:
 - Libère les fonctionnalités restreintes par le surclassement ((B) myObj).meth2(...);

B

Polymorphisme

- Surclassement (à la compilation) :
 - Vu comme un objet du type de la **référence**



```
A myObj = new B( ... ); \leftarrow
```

- A: référence • Liaison dynamique (à l'exécution) : B: classe effective
 - Méthode de la classe effective de l'objet qui est exécuté

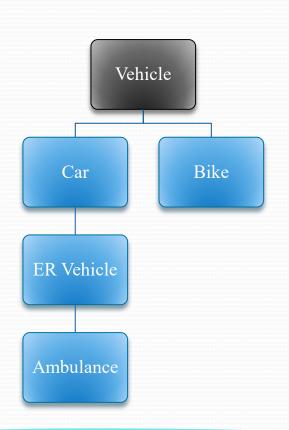
```
myObj.meth1( ... );
                         meth1 (...) est une méthode de A, redéfinie par B:
• Downcasting: c'est celle de B qui est exécutée!
```

• Libère les fonctionnalités restreintes par le surclassement

```
((B) myObj).meth2( ... );
        Transtypage
```

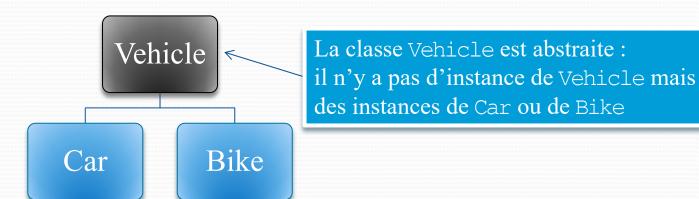
Polymorphisme

```
public class Vehicle {
    void move() { System.out.println("Avec deux ou quatre roues !"); }
public class Bike extends Vehicle {
    void move() {System.out.println("Avec deux roues !"); }
    void lean() {System.out.println("Je me penche !"); }
}
public static void main(String[] args) {
    Vehicle bike = new Bike( ... ); // Surclassement.
    bike.move(); // Liaison dynamique.
                  // Affichage : Avec deux roues !
    bike.lean(); // Erreur!
                  //(Vehicle n'a pas de méthode lean()).
    ((Bike) bike).lean(); // Downcasting.
                          // Affichage : Je me penche !
```



Classe abstraite

- Ne peut pas être instanciée (mais constructeur[s] possible[s])
- Si une seule méthode est abstraite, la classe l'est aussi
- Mot clef abstract :
 - Classe: public abstract class MyClass { ... }
 - Méthode : Pas de corps public abstract void meth1(...); ←



Interface

- Une interface donne son type aux classes l'implémentant
- Mot clef interface (pas abstract): public interface MyInterface { ... };
- Mot clef implements:

```
public class MyClass implements MyInterface1 { ... }
public class MyClass1 implements MyInterface1, MyInterface2 ... { ... }
public class MyClass2 extends MySuperClass implements MyInterface1 ... { ... }
```

- Les interfaces peuvent se dériver (mot clef extends) public interface MyInterface2 extends MyInterface1 { ... };
- Méthode par défaut : ← Trait public default void foo() { System.out.println("Default implementation of foo()");

Classes interne/anonyme

• Classe locale ou interne :

```
public class MyClass {
    class MyLocalClass { ... }
```

• Classe anomyme:

```
MyAnonymousClass myObj = new MyAnonymousClass() {
                     Attention: il s'agit d'une instruction!
```

- Bytecode:
 - Classe: MyClass.class
 - Interne: MyClass MyLocalClass.class
 - Anonyme: MyClass\$1.class

Génériques

• Polymorphisme paramétrique de type «

Classes typées à la compilation

- Comportement unique pour des types polymorphes
- Un peu comme les *templates* C++:
 - Une seule copie du code : compilé une fois pour toutes
- Notation: <Type1>, <Type2, Type3>, etc.

```
MyClass<String> myObj = new MyClass<String>();
public class MyList<B extends A, C>
MyList<MyClass, Date> list = new MyList<MyClass, Date>();
```

Génériques

• Wildcard:?

```
void myMeth1(List<? extends MyClass> a) { 
    for(MyClass p : a) {
       myMeth12(p);
```

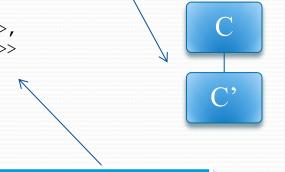
• Variance (limite de portée) : &

```
final class MyClass<A extends Comparable<A> & Cloneable<A>,
                    B extends Comparable<B> & Cloneable<B>>
                    implements Comparable < MyClass < A, B >> ,
                                Cloneable<MyClass<A,B>> {
```

• Méthode générique :

```
public <T> T addTo(T e, Collection<T> c) {
    c.add(e);
    return e;
```

Si C' hérite d'une classe C et G est un générique de paramètre C alors: il est faux de dire que G<C'> hérite de G<C>



Il ne peut y avoir que des interfaces après le premier &

Crédits



