Patrick GIRARD
University of Poitiers
2009-2014©

Object-Oriented Programming in Java™

Advanced Inheritance

Chapter 6 - Section 2

Table of contents

Polymorphism

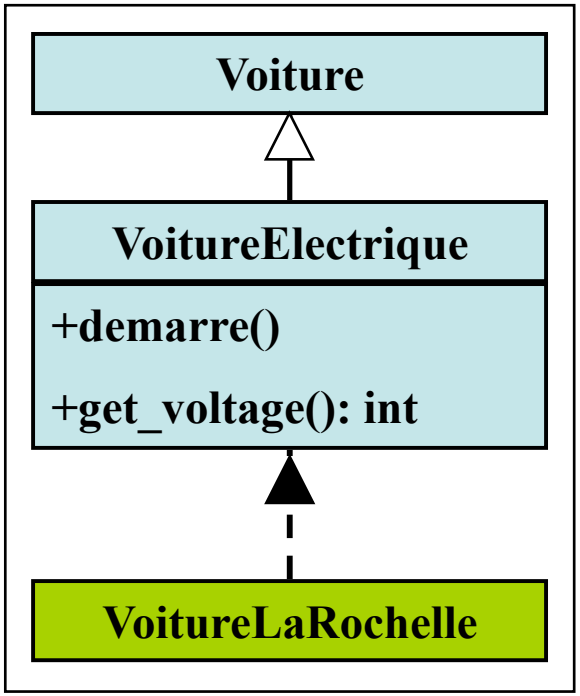
- Definition
  - Polymorphism in the context of object-oriented programming, is the ability to create a variable, a function, or an object that has more than one form
  - Operator overloading of the numeric operators (+, -, *, and /) allows polymorphic treatment of the various numerical types. Another common example is the use of the "+" operator which allows similar or polymorphic treatment of numbers (addition), strings (concatenation), and lists (attachment).

Polymorphism

- A language is defined as polymorphic if it allows perceiving an object at different levels in the inheritance tree
- An object from class B that inherits from A may be considered as an instance of A, if special characteristics of B are not valuable

Polymorphism

- VoitureLaRochelle
  - is an ElectricCar...
  - but also is a Car



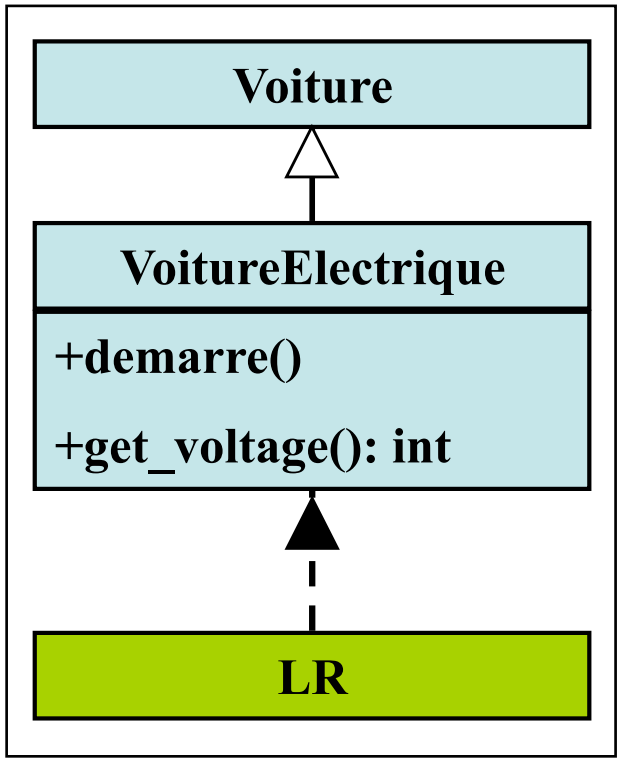
Java is polymorphic

- Assignment of child objects to parent objects is possible
  - Ex. an electric car can be assigned to a car object
- This is called upcasting

Java is polymorphic

- Upcasting

```
public class Test {
   public static void main (String[] a) {
      VoitureElectrique LR = new
            VoitureElectrique();
      Voiture voit = LR ;
      …
   }
}
```



Object restriction

- At compile time
- An upcasted object is seen as an object of the class of its reference
- Functionalities are restricted

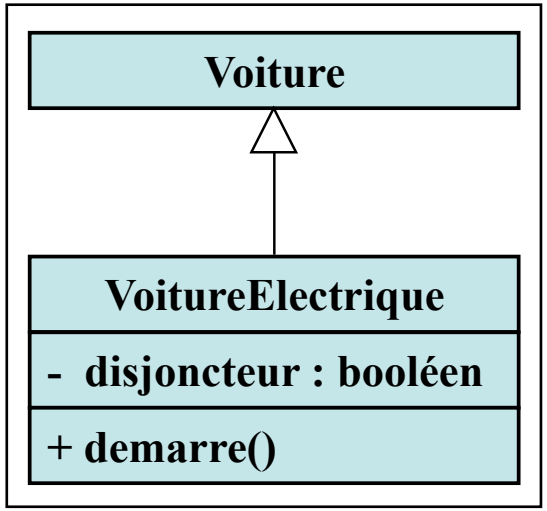
Object restriction

- Example

```
public class Test {
   public static void main (String[] a) {
      VoitureElectrique LR = new
            VoitureElectrique();
      Voiture voit = LR ;

      voit.demarre(); // ok
      System.out.println(voit.getVoltage()); // Erreur
      …
   }
}
```

GetVoltage is not available ! Why ?




```
public class VoitureElectrique
                        extends Voiture {
   private boolean disjoncteur;

   public void demarre() {
      disjoncteur = true;
   }
}
```

```
public class VoitureElectrique
                        extends Voiture {
   private boolean disjoncteur;

   public void demarre() {
      super.demarre();
      disjoncteur = true;
   }
}
```

Dynamic polymorhism

- One question

```
public class Test {
   public static void main (String[] a) {
      VoitureElectrique LR = new
            VoitureElectrique();
      Voiture voit = LR ;

      voit.demarre(); // ok
      System.out.println(voit.getVoltage()); // Erreur
      …
   }
}
```

What method is used?

Polymorphism, a dynamic link

- At runtime
  - Execution of the method of the actual class
  - The actual class cannot be known at compile time
- Different names
  - Late binding
  - Run-time binding
  - Dynamic binding

Polymorphism: Summary

- Upcasting (at compile time)
  - An upcasted object can only access the method of its reference
- Late or dynamic binding (at run-time)
  - In case of overriding, the more convenient method is used, depending on the actual class of the object

Interests of Polymorphism

- No need for explicit programming
- Easy extension
- Fast development
- Simpler and better organization
- Easier maintenance

Downcasting

- Opposite of Upcasting
- Goal
  - Specialize an object
- Interest
  - Using extension when possible
- Needs
  - Using an explicit conversion (cast)

Downcasting

```
public class Test {
   public static void main (String[] a) {
      VoitureElectrique LR = new
            VoitureElectrique();
      Voiture voit = LR ;

      voit.demarre(); // ok
      if (voit instanceof VoituerElectrique)
        System.out.println(
            (VoitureElectrique)voit.getVoltage());
      …
   }
}
```

cast

Do not forget...

- Main characteristics of OOP
  - Data Hiding (encapsulation)
  - Inheritance
  - Polymorphism