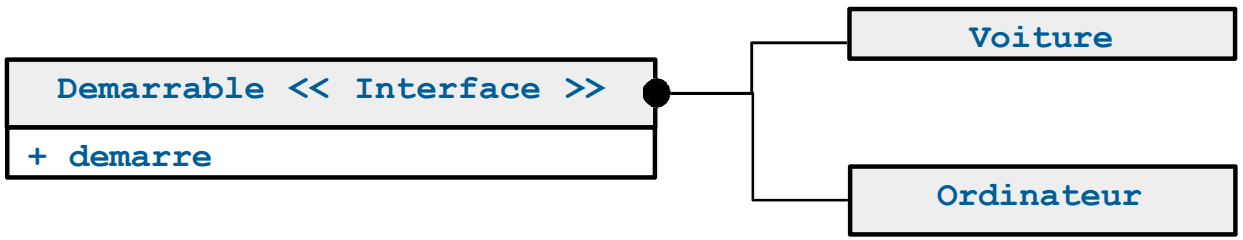
Object-Oriented Programming: Application to the Java Language

Interfaces

Concept of interface

- An interface is a model for a class
  - When all the methods of a class are abstract and there are no attributes, we arrive at the concept of an interface
  - It defines the signature of methods that must be implemented in the classes that adhere to this model
  - Any class that implements the interface must implement all the methods defined by the interface
  - Any object that is an instance of a class implementing the interface can be declared as the type of that interface
  - Interfaces can inherit from other interfaces
- Example
  - Things that are *Demarrable* must have a method *demarre*()



Concept of interface and Java

- Implementing an interface
  - The definition of an interface is similar to that of a class. The keyword **interface** is used instead of **class**

```
public interface NomInterface {
    ...
}
```

Interface: do not confuse with graphical interfaces

  - When defining a class, we can specify that it implements one or more given interface(s) by using the keyword **implements** once.

```
public class NomClasse implements Interface1,Interface3, ... {
    ...
}
```

  - If a class inherits from another class, it can also implement one or more interfaces.

```
public class NomClasse extends SuperClasse implements Inter1, ... {
    ...

}
```

Concept of interface and Java

- Implementing an interface
  - An interface does not have attributes
  - An interface can have constants

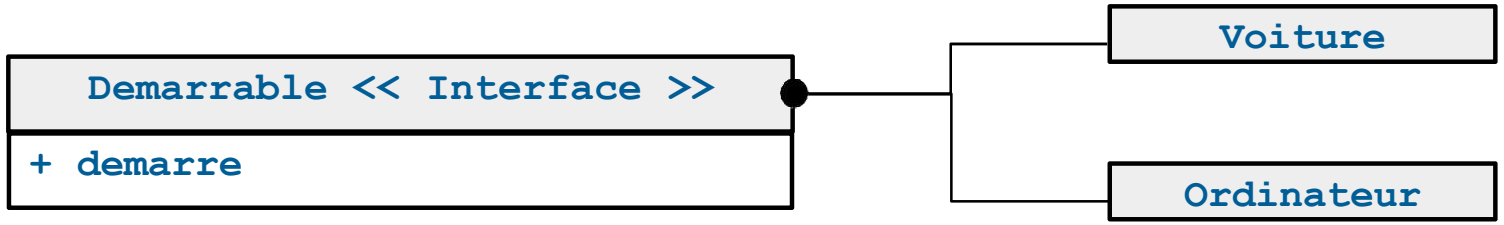
```
public interface NomInterface {
    public static final int CONST = 2;
}
```

  - An interface does not have the **abstract** keyword
  - Interfaces are not instantiable (Same reasoning as with abstract classes)

```
NomInterface jeTente = new NomInterface(); // Erreur!!
```

Concept of interface and Java

- Any class that implements the interface must implement all the methods defined by the interface.




```
public interface Demarrable {
    public void demarre();
}
```

Must implement all the methods of the interfaces.

```
public class Voiture implements Demarrable {
    ...

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

```
public class Ordinateur implements Demarrable {
    ...

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

A *Voiture* is *Demarrable*

An *Ordinateur* is *Demarrable*

Concept of interface and Java

- Any object that is an instance of a class implementing the interface can be declared as the type of that interface.

```
public class Test {
    public static void main (String[] argv) {
        // Déclaration d'un objet de type Demarrable
        Demarrable dem1;
        // Création d'un objet Voiture
        dem1 = new Voiture();

        // Déclaration et création d'un objet Personne
        Personne pers1 = new Personne(dem1);
        pers1.mettreEnRoute();

        // Déclaration d'un objet de type Demarrable
        Demarrable dem2;
        // Création d'un objet Ordinateur
        dem2 = new Ordinateur();

        // Déclaration et création d'un objet Personne
        Personne pers2 = new Personne(dem2);
        pers2.mettreEnRoute();
    }
}
```

A *Personne* can start all *Demarrable* objects.

Concept of interface and Java

- Example: a *Voiture* and a *Ordinateur* are *Demarrable* objects.

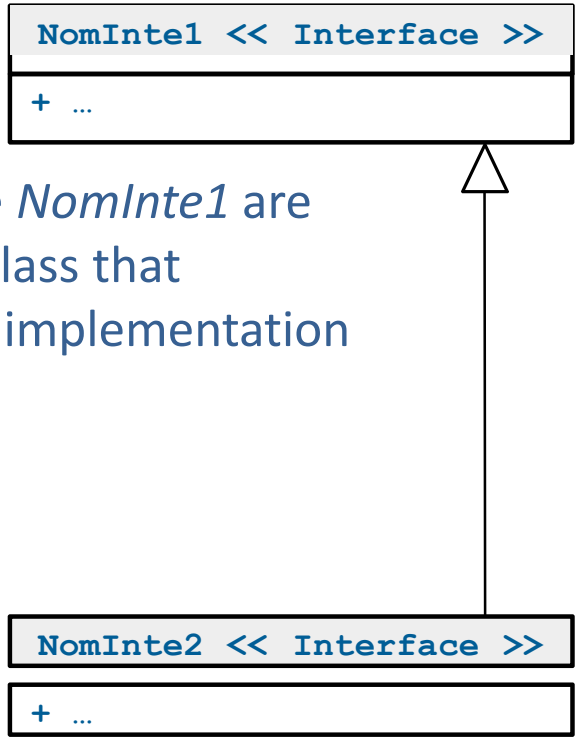
```
public class Person {

   private Demarrable

   objetDemarrable; public

   Person(Demarrable dem) {
      objetDemarrable = dem;
   }

   public void mettreEnRoute()
      {
      objetDemarrable.demarre(
      );
   }
}
```

A *Personne* can start a *Voiture* and a *Ordinateur* without knowing their exact nature.

Concept of interface and Java

- Interfaces can inherit from other interfaces
  - An interface can inherit from another interface using the keyword "extends".

- Consequences



  - The method definitions of the parent interface *NomInte1* are inherited by the child interface *NomInte2*. Any class that implements the child interface must provide an implementation for all the methods, including those inherited.

- Usage

  - When a model can be defined by several complementary sub-models.

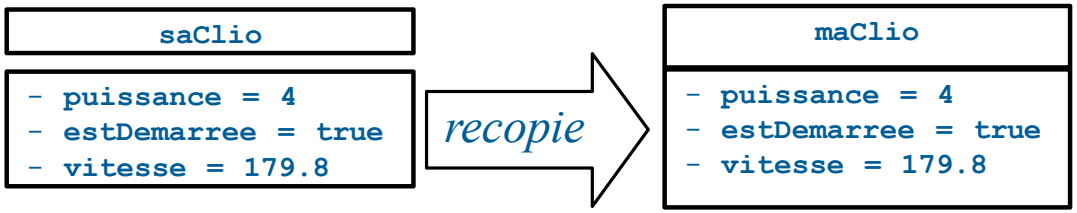
Abstract Classes versus Interfaces

- Classes
  - They are fully implemented
  - Another class can inherit from them
- Abstract Classes
  - They are partially implemented
  - Another non-abstract class can inherit from them but must provide an implementation for the abstract methods
  - Another abstract class can inherit from them without necessarily providing an implementation for all the abstract methods
  - They cannot be instantiated but can provide a constructor
- Interfaces
  - They are not implemented
  - Any class that implements one or more interfaces must implement all their (abstract) methods

The "Cloneable" interface

- Two possibilities for duplicating an object
  - Create a custom method *public MaClasse dupliquer()* that returns a copy of the object by creating a new instance and initializing the attributes (using the constructor)

`saClio != maClio`
But the content is identical



- Use the "Cloneable" interface to maintain compatibility with other Java classes
  - Implement the *protected Object clone()* method from the *Cloneable* interface

```
public class Voiture implements Demarrable, Cloneable {
    protected Object clone()
        { Voiture copie;
        copie = new Voiture(this.puissance, (Galerie)laGalerie.clone());
        return copie;
    }
}
```

The "Inner Classes"

- Basic rule in Java
  - One class per file and one file per class
- Local or inner classes
  - Defined inside other classes (Engine inside Car)
- Anonymous classes
  - Are instantiations of classes and implementations of an abstract class or an interface
  - The abstract method(s) must be implemented at the time of instantiation

```
public class Voiture {
    ...
    class Moteur {
        ...
    }
}
```

```
Demarrable uneInstance =
   new Demarrable(){
        public void demarre() {
            // Code ici
        }
   };
```

Anonymous classes are widely used for developing GUIs with Java/Swing

The "Inner Classes"

- Source code: 1 file
  - class
  - anonymous class
  - inner class
- Byte-code generation: 3 files
  - *Voiture.class*
  - anonymous *Voiture$1.class*
  - inner *Voiture$Moteur.class*

Anonymous class, implements the *Init* interface

```
public class Voiture {

    public Voiture(...) {
        monMoteur = new Moteur(...);
        Init monInit = new Init() {
            public void initialisation() {
                ...
            }
        };
    }
    class Moteur {

        public Moteur(...) {

            ...
        }
    }
}
```

Inner class

Nom	Taille	Type
Voiture$1.class	1 Ko	Fichier CLASS
Voiture$Moteur.class	1 Ko	Fichier CLASS
Voiture.class	1 Ko	Fichier CLASS
Voiture.java	1 Ko	Java Source File

The .class files that have a $ in their name are not temporary files!