1/ Interfaces

On dit qu'une classe **implémente** une interface, si elle **définit les méthodes de l'interface**.

En java on déclare qu'une classe implémente une interface avec le mot clé `implements`.

Une interface définit un type (comme une classe) et les classes qui implémentent cette interface sont donc des sous-types.

Exemple

```
interface Inflammable {
    void enflammer();
}

class Bois implements Inflammable {
        public void enflammer() {
                System.out.println("Je brule et fais des braises");
        }
}

class Dancefloor  implements Inflammable {
        public void enflammer() {
                System.out.println("♪ ♫ Youhouhou ♬ ♫ ");
        }
}

public class Main {
    static public void main(String[] args) {
        Inflammable[] tab = { new Bois(), new Dancefloor() };

        for(Inflammable i : tab)
                i.enflammer();
    }
}
```

Le mot clé `public` est implicite dans une interface.

2/ Avantage :

- Une classe peut implémenter plusieurs interfaces.
- On a tous les avantages du sous-typage comme avec l'héritage classique (notamment le polymorphisme).

Exemple

```
public interface Comparable {
    boolean greaterThan(Object o);
}
class Maximize {
    static public Comparable max(Comparable a, Comparable b) {
        if(a.greaterThan(b)) {
            return a;
        }
        return b;
    }
}

class Person implements Comparable {
    private int size;
    Person(int size) { this.size = size; }
    @Override
    public String toString() { return "size: "+size; }
    public boolean greaterThan(Object o) {
        return this.size > ((Person)o).size;
    }
}
public class Main {
    static public void main(String[] args) {
        Person p1 = new Person(157);
        Person p2 = new Person(173);
        System.out.println(Maximize.max(p1, p2));
    }
}
```

3/ Une interface peut remplacer une classe pour déclarer :

- un attribut
- une variable
- un paramètre
- une valeur de retour

Attention, on ne peut pas instancier une interface.

Exemple

```
interface Comparable {
    boolean greaterThan(Object o);
}

class Maximize {
    static public Comparable max(Comparable a, Comparable b) {
        if(a.greaterThan(b)) {
            return a;
        }
        return b;
    }
}

class ListElements {
        Comparable[] tab;
        int nbElements = 0;

        public ListElements(int maxSize) {
                tab = new Comparable[maxSize];
        }

        public void add(Comparable e) {
                tab[nbElements] = e;
                nbElements++ ;
        }
        public boolean isIncreasing() {
                for (int i = 1; i < nbElements ; i++) {
                        if (tab[i-1].greaterThan(tab[i])) return false;
                }
            return true;
        }
}

class Person implements Comparable {
    private int size;
    Person(int size) { this.size = size; }
    @Override
    public String toString() { return "size: "+size; }
    public boolean greaterThan(Object o) {
        return this.size > ((Person)o).size;
    }
}

public class Main {
    static public void main(String[] args) {
```

```
        Person p1 = new Person(157);
        Person p2 = new Person(173);
        Person p3 = new Person(175);
        ListElements l = new ListElements(10);
        l.add(p1);
        l.add(p2);
        l.add(p3);
        System.out.println(l.isIncreasing());
        l.add(p2);
        System.out.println(l.isIncreasing());
    }
}
```