# Programmation Orientée Objets en JAVA

### **Daniele Varacca**

Departement d'Informatique Université Paris Est

2014





En Java une classe se définit par le mot clé class

```
class MyClass {
}
```

Par *convention* les noms des classes commencent par une lettre majuscule





Une classe peut contenir des *champs* Chaque champ doit avoir un type

```
class MyClass {
    int x;
    boolean b;
    String s;
    MyOtherClass o;
}
```

Un champs peut être de type primitif ou objet





Une classe peut également contenir des des méthodes

```
class MyClass {
    int x;
    boolean b;
    String s;
    MyOtherClass o;
    int f (int x) {/* corps */ }
    MyOtherClass g () {/* corps */}
    void h (String s1, String s2) {/* corps */}
}
```

Par *convention* les noms des champs et des méthodes commencent par une lettre minuscule



Le nom, les types des arguments, et le type de retour constituent la *signature* d'une méthode. L'ensemble de toutes les signatures d'une classe constitue la signature de la classe ou son *interface* 

```
class MyClass {
    int x;
    boolean b;
    String s;
    MyOtherClass o;
    int f (int x) {/* corps */ }
    MyOtherClass g () {/* corps */}
    void h (String s1, String s2) {/* corps */}
}
```

La signature d'une classe définit ce que les objets de cette classe peuvent ou ne peuvent pas pas faire

Dans le corps d'une méthode je peux mentionner les champs de la classe

```
class MyClass {
    int x;
    boolean isPositive () {
        if (x>0) return true
        else return false;
     }
}
```



Dans le corps d'une méthode je peux mentionner les champs de la classe

```
class MyClass {
     int x;
     boolean isPositive () {return (x>0);}
}
```





# Objets

Un objet se crée en utilisant le mot clé new

```
MyClass anObject = new MyClass();
```

Pour accéder aux champs et au méthodes d'un objet on utilise le point

```
int y = anObject.f(5);
anObject.b = true;
anObject.h();
```

Le compilateur rejette les accès au champs qui ne sont pas dans la classe, comme l'appel des méthodes qui ne sont pas dans la signature





# **Objets**

Dans une méthode on peut mentionner "soi même" avec le mot clé this

Cela peut enlever des ambiguïtés

```
class MyClass {
        int x;
        int f (int x) {return (x + this.x) }
}
```

Et peut aussi rendre le code plus lisible

```
class MyClass {
    int x;
    boolean compare (MyClass t) {
        return (t.x > this.x)
    }
}
```



# **Objets**

On peut donner des valeurs par défaut aux champs

```
class MyClass {
         int x=3:
         boolean b=true:
         String s="Salut";
         MyOtherClass o = new MyOtherClass();
Mais qu'est-ce qui se passe si on ne le fait pas...
MyClass t = new MyClass();
MyOtherClass z = t.o;
Quelle est la valeur z?
```



### Constructeurs

Les valeurs par défaut n'ont pas toujours du sens

```
class Person {
    int age;
    String name;
    Job job;
}

Person myself = new Person();
myself.age = 27;
/* etc */
```





### Constructeur

```
On peut définir un constructeur
```

```
class Person {
        int age;
        String name;
        Job job;
        Person (int age, String name, Job job) {
             this .age = age;
             this .name = name;
            this.job = job;
Person myself =
```

new Person (27, "Daniele", new Job (

## Classe abstraites

Les méthodes qu'on écrit sont aussi "par défaut" et parfois n'ont pas de sens

```
class RegularShape {
    int numberOfSides;
    int side;
    int area() {/* compute the area/*}
}
```

Comment on généralise les constructeurs au méthodes?





### Classe abstraites

side=s:

# Mot clé abstract abstract class RegularShape { int numberOfSides; int side; abstract int area(); /\* pas de corps!\*/ RegularShape (int n, int s) { numberOfSides=n; } }





### Classe abstraites

Quand on crée l'objet on doit aussi définir la méthode:

```
RegularShape square = new RegularShape (4,12) {
        int area () {return side*side;}
        };
RegularShape pentagon = new RegularShape (5,12) {
        int area () {
        return Math.sqrt(5*(2+Math.sqrt(5)))*side*side/4;
        }
        }
}
```



# Complements

- Mécanisme de création des objets
- ▶ La valeur null
- Surcharge des noms des méthodes
- Le ramasse miettes
- La méthode toString





# Création d'objets

Quand on crée un objet à l'aide de new.

```
MyClass o = new MyClass();
```

une portion de la mémoire est alloué pour contenir les champs est les méthode du nouvel objet

La variable o est affecté avec un pointeur vers cette mémoire





# Création d'objets

Les objets sont manipulé à travers des pointeurs.

- quand on passe un objet à une méthode, on ne crée pas de copie
- les adresses des objets ne peuvent pas être connus directement
- ▶ le mot clé pour un pointeur qui ne pointe nulle part est null
- null est la valeur par défaut des champs de type objet





### Le ramasse miettes

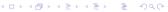
En java ce n'est nécessaire de libérer la mémoire (comme en C).

Cela est fait automatiquement par un programme qu'on appelle le *ramasse miettes* 

```
MyClass o = new MyClass();
o = new MyClass();
```

- Deux objets sont créés
- il n'y a plus moyen d'accéder au premier
- sa mémoire est libérée





# La surcharge des méthodes

L'identité d'une méthode est déterminée par sa signature:

- son nom
- les types de ses arguments
- le type de retour

Dans une classe on ne peut pas avoir deux méthodes avec la même signature

Mais on peut avoir deux méthode avec le même nom!





# La surcharge des méthodes

```
class MyClass {
   int x;
   boolean smaller (MyClass o) {
     return (this.x < o.x);
   }
   boolean smaller (MyClass o1, MyClass o2) {
     return (this.x < o1.x && this.x < o2.x);
   }
}</pre>
```

Seule règle: deux méthodes ne peuvent pas juste différer sur le type de retour

(Pour des raisons pratiques)

