# INFO0201

## Introduction à la programmation orientée objet

## Partie 3

## Interface et implémentation

## Javadoc

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

**Equipe pédagogique :** J. Jonquet, Ch. Jaillet,, P. Delisle, C. Rabat, C. Besançon

# Plan

- Interface et implémentation
  - Mode d'emploi
  - Diagramme de classes
  - Réalisation en java
  - Contrat de comportement
  - Classification
  - Exemples
  - Typage dynamique
- Javadoc

# Interface : Mode d'emploi d'une classe

- Encapsulation

  Les données et les méthodes sont rassemblées au sein de l'objet

- Interface [partie publique]

  Ensemble de méthodes accessible depuis l'extérieur de la classe. Le monde extérieur n'a accès aux données que par l'intermédiaire de cet ensemble de méthodes.

- Implémentation [partie privée]

  C'est le détail [non visible] de la représentation des données et du codage des méthodes

**=> dans l'interface, les détails de l'implémentation sont cachés**

L'interface peut-être vue comme le **mode d'emploi** d'une classe.

# Interface : Mode d'emploi d'une classe

## Répartition des rôles

- **Interface**

  => Juste la signature des méthodes publiques (leur déclaration)

- **Implémentation**

  => Les attributs privés

  => La définition de toutes les méthodes (leur code)

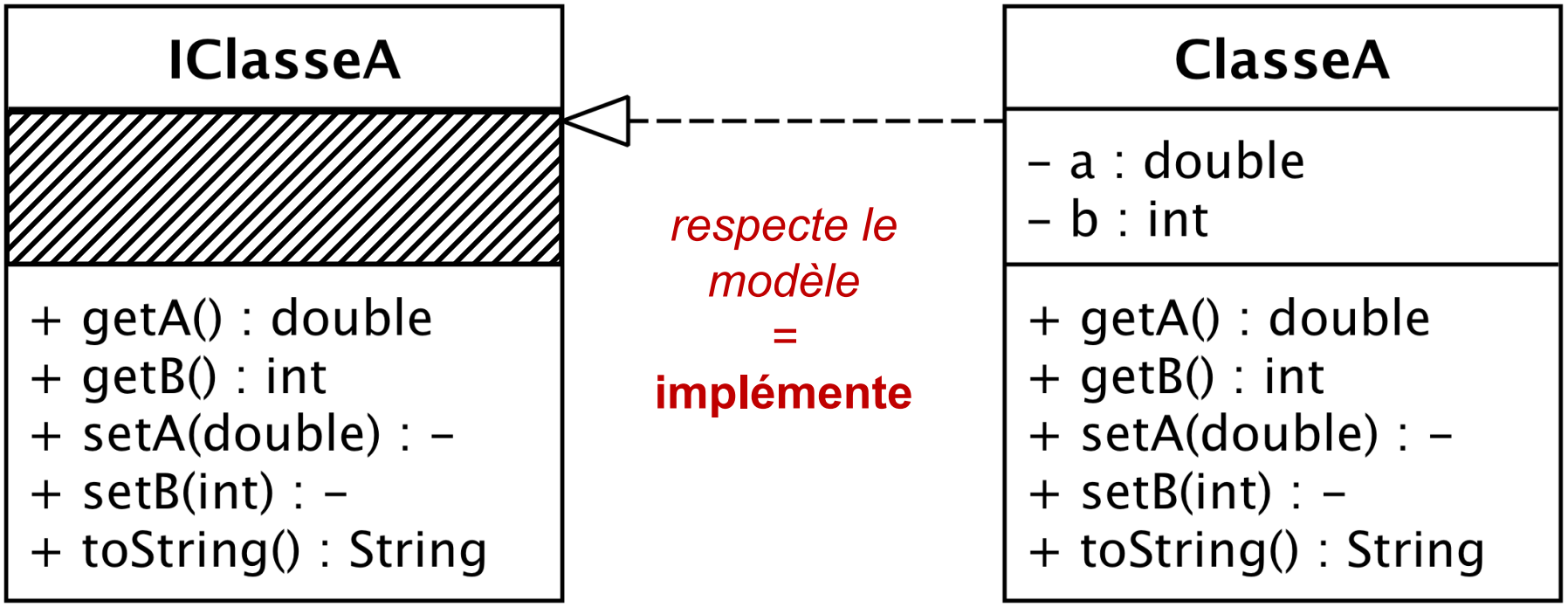
La classe **implémente** l'interface :

⇒ signifie qu'elle **respecte** le modèle

⇒ possède **au minimum** la définition des méthodes de l'interface

# Diagramme de classes

## Exemple



# Réalisation en java

## Exemple

```
//fichier IClasseA.java
interface IClasseA {
    public double getA() ;
    public int getB() ;
    public void setA(double a) ;
    public void setB(int b) ;
    public String toString() ;
}
```

```
/fichier ClasseA.java
class ClasseA implements IClasseA {
    //attributs
    private double a;
    private int b;
    //constructeurs
    …
    //methodes
    public double getA(){return a;}
    public int getB(){return b;}
    public void setA(double a){
                this.a = a;
    }
    public void setB(int b){
                this.b = b;
    }
    …
}
```

# Interface : Contrat de comportement

- Interface
  - Déclinaison d'un ensemble de comportements (méthodes)
  - Pas d'attribut d'instance
  - Eventuellement des constantes publiques de classe

- *class + implements*

  Une classe utlise une interface en définissant les méthodes déclarées dans l'interface

  ⇒ Obligation de développement (respect du contrat)

**L'interface impose un contrat à respecter.**

# Classification

## Catégories de comportement

- Une classe peut implémenter plusieurs interfaces
- Chaque interface représente une catégorie de comportement à suivre

**=> Classification suivant des comportements**

## Exemple : types numériques

3 classes : Rationnel, Complexe et Vecteur

3 comportements : - Normé : `norme()`
- Signé : `positif()` et `negatif()`
- Ordonné : `croissant()`, `decroissant()`, `constant()`

| | Rationnel | Complexe | Vecteur |
|---|---|---|---|
| **Normé** | | X | X |
| **Signé** | X | | X |
| **Ordonné** | | | X |

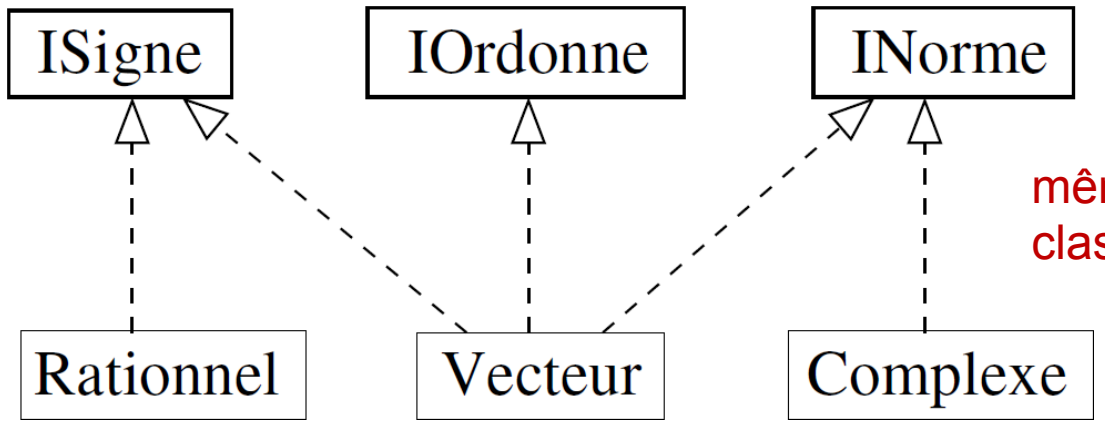
# Classification

- Catégories de comportement

  Exemple : types numériques

| | Rationnel | Complexe | Vecteur |
|---|---|---|---|
| **Normé** | | X | X |
| **Signé** | X | | X |
| **Ordonné** | | | X |

**=> Classification suivant des comportements**



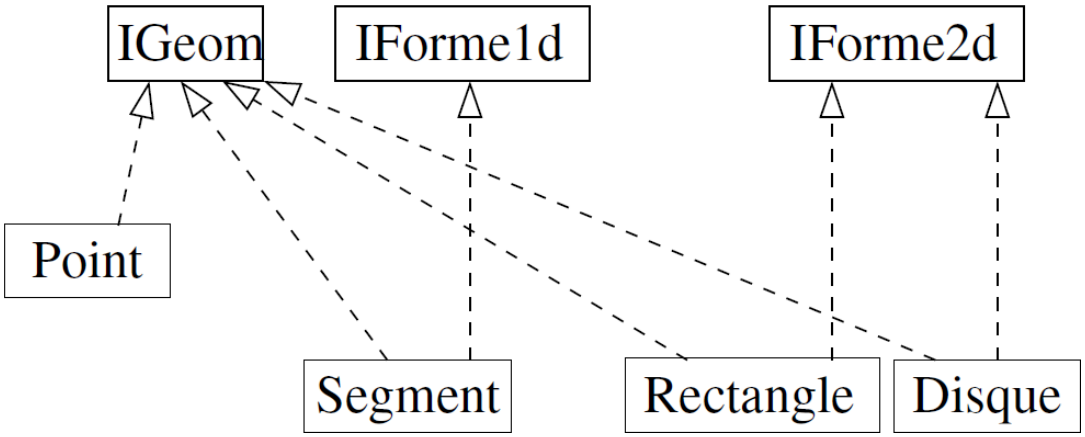

même pour des classes sans relation

# Classification

## Autre exemple : géométrie

| | déplacement | distance à l'origine | longueur | surface | périmètre |
|---|---|---|---|---|---|
| Point | X | X | | | |
| Segment | X | X | X | | |
| Disque | X | X | | X | X |
| Rectangle | X | X | | X | X |

IGeom IForme1d IForme2d



# Classification

## Autre exemple : géométrie

```
interface IGeom {
    public void deplacement(double a, double b) ;
    public double distanceALOrigine() ;
}
```

```
class Point implements IGeom {
    … // attributs et constructeurs
    public void deplacement(double a, double b){
            x += a;    y += b;
    }
    public double distanceALOrigine(){
            return Math.sqrt( x * x + y * y );
    }
    … //autres methodes de la classe
}
```

```
class Segment implements IGeom, IForme1d {
    … // plus d'obligations (celles des deux interfaces)
}
```

# Typage dynamique

- On peut déclarer des variables du type d'une interface
- On peut affecter à une référence du type de l'interface un objet qui implémente cette interface

## Exemples pour géométrie

```
class TestGeom {
    public static void main (String[ ] args){
        Point p = new Point(…);
        Segment s = new Segment(…)
                                                    IGeom ref;
                                                    ref = s;
                                                    ref.deplacer(3.5, -2.6);
                                                    double x = ref.longueur();
                                                }
                                            }
```

**impossible pas dans IGeom**

```
IGeom ref;
…
Ref = new Rectangle(…);
Ref.deplacer(3.5, -2.6);
double x = ref.surface();
…
```

**impossible pas dans IGeom**

# Typage dynamique

## Intérêt

Permettre la généricité

Par exemple : rassembler dans un tableau des objets de type différents

Un tableau ne peut stocker QUE des références du même types

=> Possible s'il existe un type de référence commun

```
IGeom[ ] tab = new IGeom[3];

tab[0] = new Rectangle(…);
tab[1] = new Disque(…);
tab[2] = new Rectangle(…);

for(int i = 0 ; i < 3 ; I++){
    tab[i].deplacer(1.0, 2.0);
    S.o.p.( tab[i].surface() );
}
…
```

**Impossible : existe dans Rectangle et disque**
**Mais PAS dans IGeom**

# Javadoc

## Outil **javadoc**

- **javadoc** est le 3e programme essentiel fourni avec le SDK (javac, java)
- Outil qui sert à générer la documentation au format Sun (API)
- Permet à un tiers d'utiliser vos programmes et vos bibliothèques de manière simple et efficace
- Permet aussi au programmeur de s'y retrouver plus rapidement

  ⇒ Maintien de votre documentation à jour

Si vous changez le nom d'une classe, d'une fonction ou même modifier la hiérarchie de votre programme, une simple nouvelle génération de la Javadoc appliquera automatiquement les changements

- Outil fondamental pour programmer en Java

# Javadoc

## Outil **javadoc**

- Génère des pages HTML contenant au minimum
  - La liste des classes
  - La liste des méthodes
  - La liste des variables
- Possible d'ajouter des informations, des commentaires

  ⇒ Génère une véritable documentation exhaustive et facilement lisible
- Avantage du format HTML : les liens hypertextes pour une navigation facilité
- Une classe **correctement commentée** permet de générer une documentation précise et complète

# Javadoc

## Commentaires javadoc : conventions générales

Il existe une grande quantité de conseils pour la mise en page des commentaires Javadoc. Voici les principaux :

- La première phrase Javadoc de la classe (ou de l'interface) doit être une description de la classe (ou de l'interface)
- Ce conseil vaut aussi pour les méthodes, variables, …
- Utilisez la troisième personne pour commenter une méthode
- Détaillez le fonctionnement des méthodes si besoin
- Utilisez les *tags* (*cf.* page suivante)
- Les *tags* @param et @return doivent être systématiquement indiqués (sauf méthodes sans paramètres ou méthodes sans retour *void*)

# Javadoc

## Commentaires et *tags*

```
/**
* Commentaire pour la documentation javadoc
* Avec utilisation des tags
*/
```

Il existe des commentaires spécialement conçus pour la Javadoc, avec des tags précis permettent de détailler des informations sur chaque classe, chaque méthode, … En voici une liste non-exhaustive :

- @param sert à renseigner le ou les paramètres de la méthode. Derrière le tag, il faut renseigner le nom du paramètre (son type sera inclus automatiquement dans la Javadoc).
- @return sert à décrire l'objet ou la valeur retourné par la méthode.
- @throws et @exception indique la présence d'une exception qui sera propagée si elle se lève. Il faut bien indiquer le type de l'exception, et la raison de l'exception.
- @author renseigne le nom de l'auteur de la classe. Si il y a plusieurs auteurs, il faut mettre plusieurs balises. Ce tag est un tag de classe ou d'interface uniquement.
- @version indique le numéro de version de la classe. Ce tag ne peut être utilisé que pour une classe ou une interface, jamais pour une méthode.
- @see permet de faire une référence à une autre méthode, classe, … Concrètement, cela se symbolisera par un lien hypertexte dans la Javadoc. C'est donc un des tags les plus importants.
- @inheritDoc recopie la documentation de la classe mère. Utile pour éviter des copier/coller.

# Javadoc

## Exemple avec les personnes

```
/**
* La classe Personne modelise une personne.
* Elle sert pour …
* @see NegativeAgeException
* @author J. Jonquet
* @version 1.0
*/
class Personne {
        /**
        * le nom de la personne, une simple chaine de caractere
        */
        private String nom;
        /**
        * l'age de la personne, doit etre positif
        */
        private int age;
        … // la meme chose pour les autres attributs
```

# Javadoc

## Exemple avec les personnes

```
    /**
    * Constructeur par initialisation pour la classe Personne
    *
    * @param n le nom de la personne
    * @param p le prenom de la personne
    * @param a l'age de la personne >= 0
    * @param ad l'adresse de la personne
    * @throws NegativeAgeException si age < 0
    */
    public Personne(String n, String p, int a, String ad) throws NegativeAgeException {
        nom = n;
        prenom = p;
        adresse = ad;
        this.setAge(a); //gere l'exception, on verra cela dans la prochaine partie
    }
…
```

# Javadoc

## Exemple avec les personnes

```
/**
* Cree une representation sous forme de String de l'objet Personne
*
* @return une chaine de caracteres
*/
public String toString(){
        return prenom + " " + nom + " a " + age + "ans et vit à " + adresse + ".";
}
/**
* Compare une personne passee en parametre a l'objet courant
* sachant que 2 personnes sont identiques si elles ont le meme age et la meme adresse
*
* @param ref la personne a compare
* @return vrai si les personnes sont identiques, faux sinon
*/
public boolean egalA(Personne ref){
        return age == ref.age && adresse.equals(ref.adresse);
}
…
```

# Javadoc

## Utilisation du programme **javadoc**

En ligne de commande

- option `-sourcepath` : racine des sources (quand utilisation des packages)
- option `-d` : répertoire destination
- arguments : noms de package et/ou noms de fichiers .java

Plusieurs niveaux de visibilité possibles

- Option `-public` : seuls les membres *public*
- Option `-protected` : seuls les membres *public/protected*
- Option `-package` : seuls les membres *public/protected/package*
- Option `-private` : tout !

```
javadoc –public *.java –d docs //fichiers .java du repertoire courant
javadoc –private –sourcepath src –d docs cours.javadoc //package cours.javadoc
```

# Prochaine partie

**Enumération**
**Exception**
**Package**