Conception objet en Java avec BlueJ
une approche interactive

9. Héritage – notions avancées

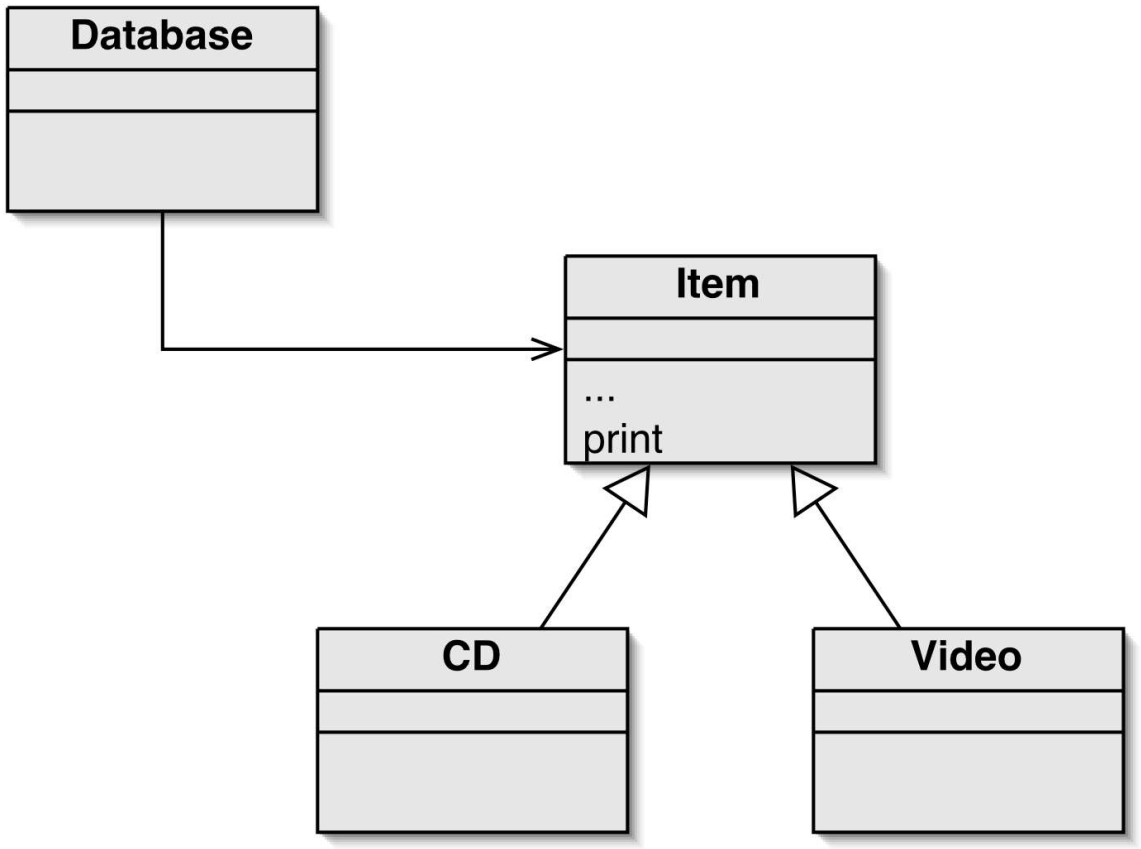
-- Polymorphisme --

David J. Barnes, Michael Kölling
version française: Patrice Moreaux

Principaux concepts abordés

- Polymorphisme de méthode
- Type statique et dynamique
- Surcharge
- Recherche dynamique de méthode
- Accès protégé

La hiérarchie d'héritage



Conflit d'affichage

Ce que nous voulons

```
CD: A Swingin' Affair (64 mins)*
    Frank Sinatra
    tracks: 16
    mon album préféré de Sinatra

video: The Matrix (136 mins)
    Andy & Larry Wachowski
    à voir si l'on aime la réalité virtuelle!
```

Ce que nous avons actuellement

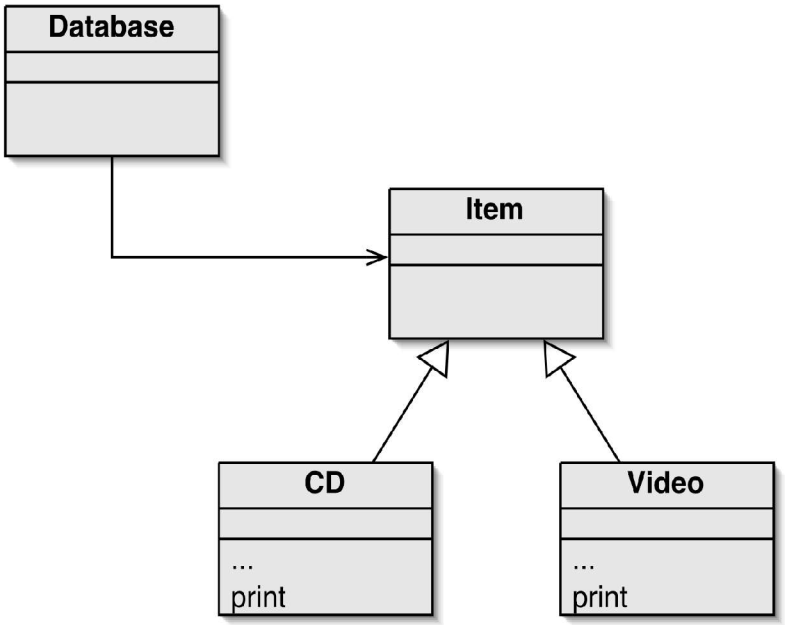
```
title: A Swingin' Affair (64 mins)*
    mon album préféré de Sinatra

title: The Matrix (136 mins)
    à voir si l'on aime la réalité virtuelle!
```

Le problème

- La méthode **`print`** de **`Item`** affiche seulement les champs communs.
- L'héritage est « à sens unique »:
  - une sous-classe hérite les champs de la superclasse.
  - la superclasse ne sait rien des champs de la sous-classe.

Une tentative de solution…



- Placer **`print`** là où elle peut accéder à l'information dont elle a besoin.
- Chaque sous-classe possède sa propre version.
- Mais les champs d'**`Item`** sont privés...
- **`...Database`** ne trouve pas de méthode **`print`** dans **`Item`**.

Type statique et type dynamique (1)

- Il nous faut de nouveaux concepts pour gérer des hiérarchies de types plus complexes.
- Un peu de terminologie:
  - type statique
  - type dynamique
  - Sélection/recherche de méthode

Type statique et type dynamique (2)

Quel est le type de c1?

```
Car c1 = new Car();
```

Quel est le type de v1?

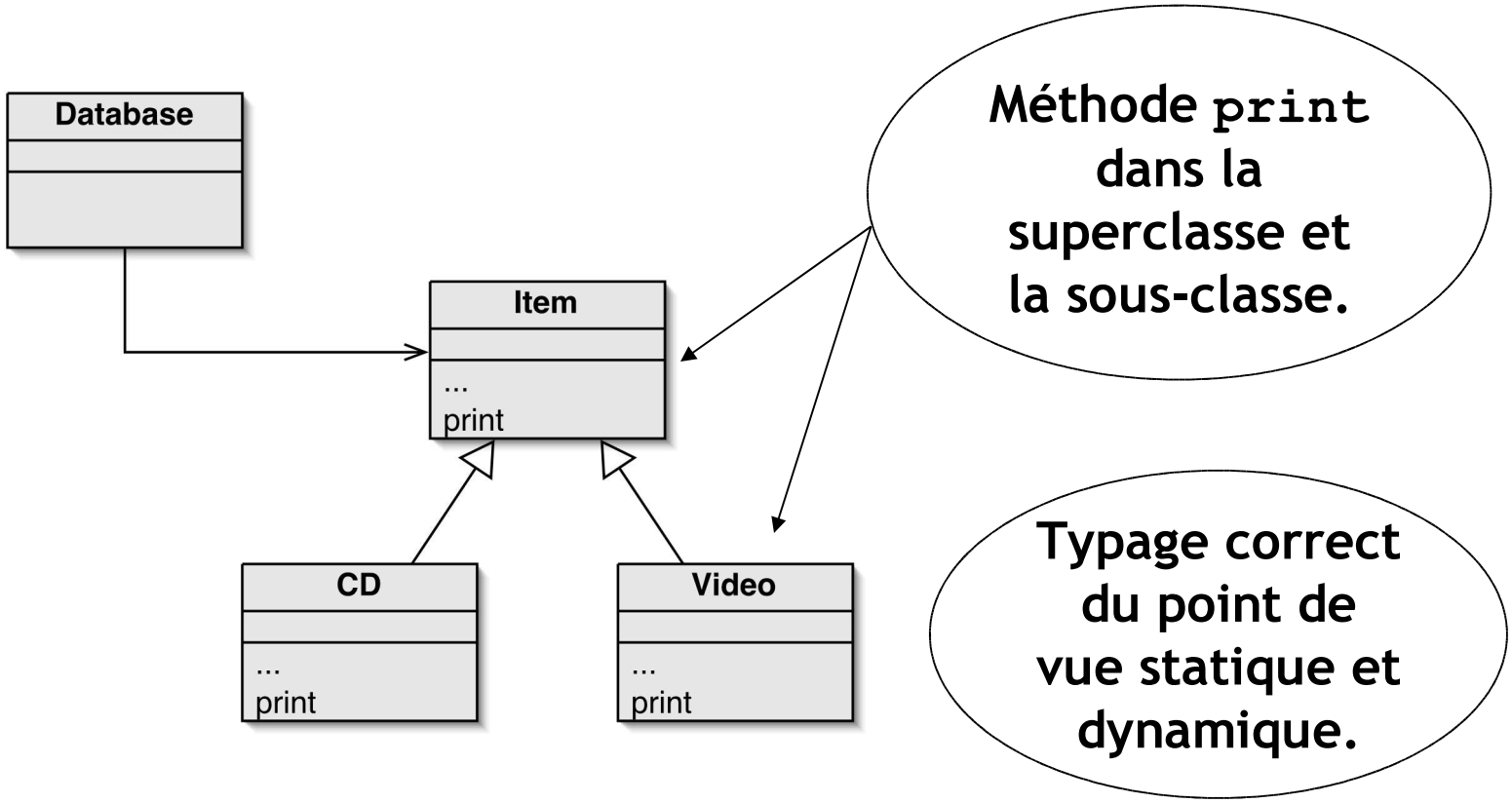
```
Vehicle v1 = new Car();
```

Type statique et type dynamique (3)

- Le type déclaré d'une variable est son *type statique*.
- Le type de l'objet que référence une variable est le *type dynamique* de cette variable.
- Le compilateur contrôle les violations de type statique.

```
Item item = (Item) iter.next();
item.print(); // erreur de compilation.
```

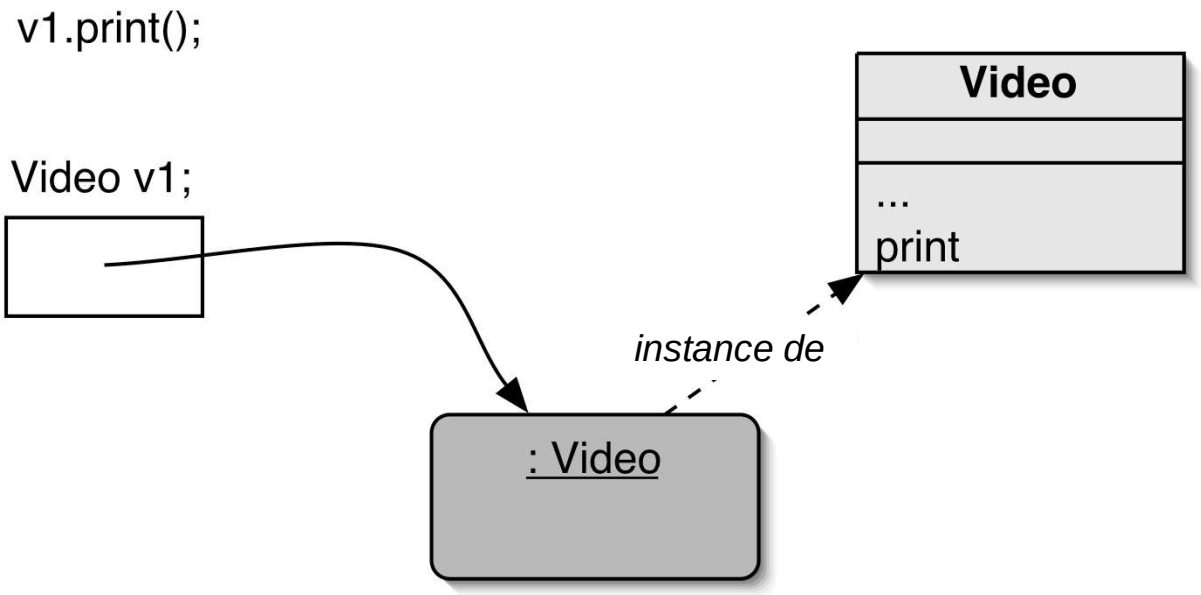
Solution: la surcharge



Surcharge

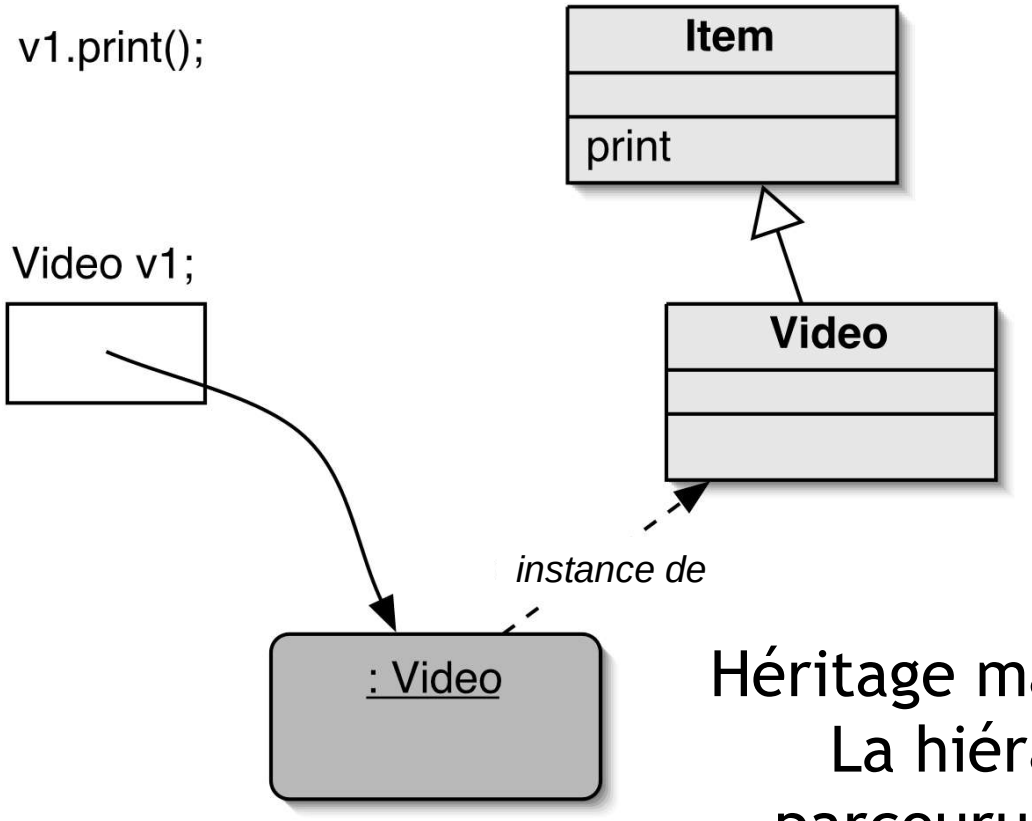
- La superclasse et la sous-classe définissent des méthodes de même signature.
- Chacune a accès aux champs de sa classe.
- La vérification statique de type porte sur la superclasse.
- La méthode de la sous-classe est appelée à l'exécution – elle *surcharge* la version de la superclasse.
- Que devient la version de la superclasse?

Recherche/sélection de méthode (1)



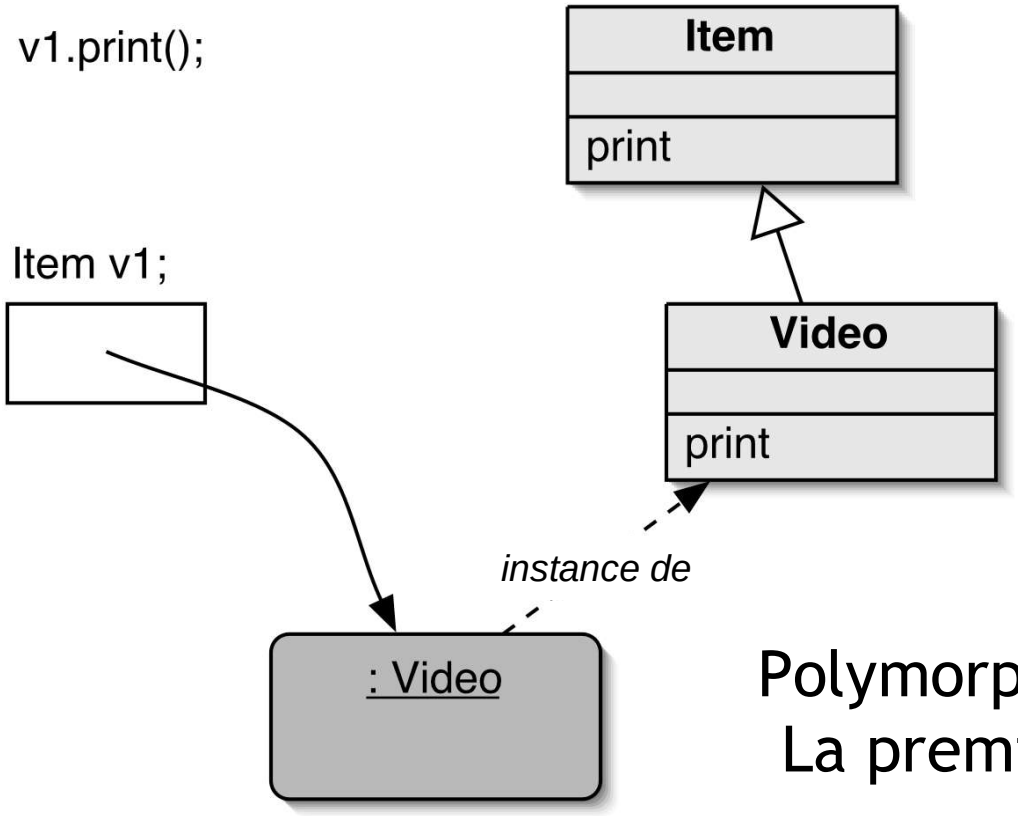
Pas d'héritage ni de polymorphisme.
La seule méthode possible est sélectionnée.

Recherche/sélection de méthode (2)



Héritage mais pas de surcharge.
La hiérarchie d'héritage est parcourue de bas en haut à la recherche de la méthode.

Recherche/sélection de méthode (3)



Polymorphisme et surcharge.
La première version trouvée est utilisée.

Recherche/sélection de méthode - résumé

- On accède à la variable.
- On trouve l'objet stocké dans la variable.
- On détermine la classe de l'objet.
- On recherche la méthode dans cette classe.
- S'il n'y en a pas, on cherche dans la superclasse.
- On recommence jusqu'à la trouver ou avoir parcouru toute la hiérarchie.
- La surcharge de méthode a priorité.

Appel `Super` dans les méthodes

- Les méthodes surchargées sont masquées ...
- ... mais on a souvent besoin de les appeler quand même.
- Une méthode surchargée peut être appelée par celle qui la surcharge.
  - `super.method(...)`
  - comparez avec l'emploi de `super` dans les constructeurs.

Appeler une méthode surchargée

```
public class CD
{
    ...
    public void print()
    {
        super.print();
        System.out.println("    " +
artist);
        System.out.println("    pistes: " +
                           numberOfTracks);
    }
    ...
}
```

Polymorphisme de méthode

- Nous avons analysé la *sélection de méthode polymorphe*.
- Une variable polymorphe peut stocker des objets de différents types pendant l'exécution.
- Les appels de méthode sont polymorphes.
  - la méthode effectivement appelée dépend du type (dynamique) de la variable.

Les méthodes de la classe `Object`

- Les méthodes de **`Object`** sont héritées par toutes les classes.
- Chacune peut être surchargée.
- La méthode **`toString`** est en général surchargée:
  - **`public String toString()`**
  - renvoie une représentation chaîne de l'objet.

Surcharge de `toString` (1)

```
public class Item
{
    ...

    public String toString()
    {
        String line1 = title +
                       " (" + playingTime + " mn)");
        if(gotIt) {
            return line1 + "*\n" + "    " +
                   comment + "\n");
        } else {
            return line1 + "\n" + "    " +
                   comment + "\n");
        }
    }
    ...
}
```

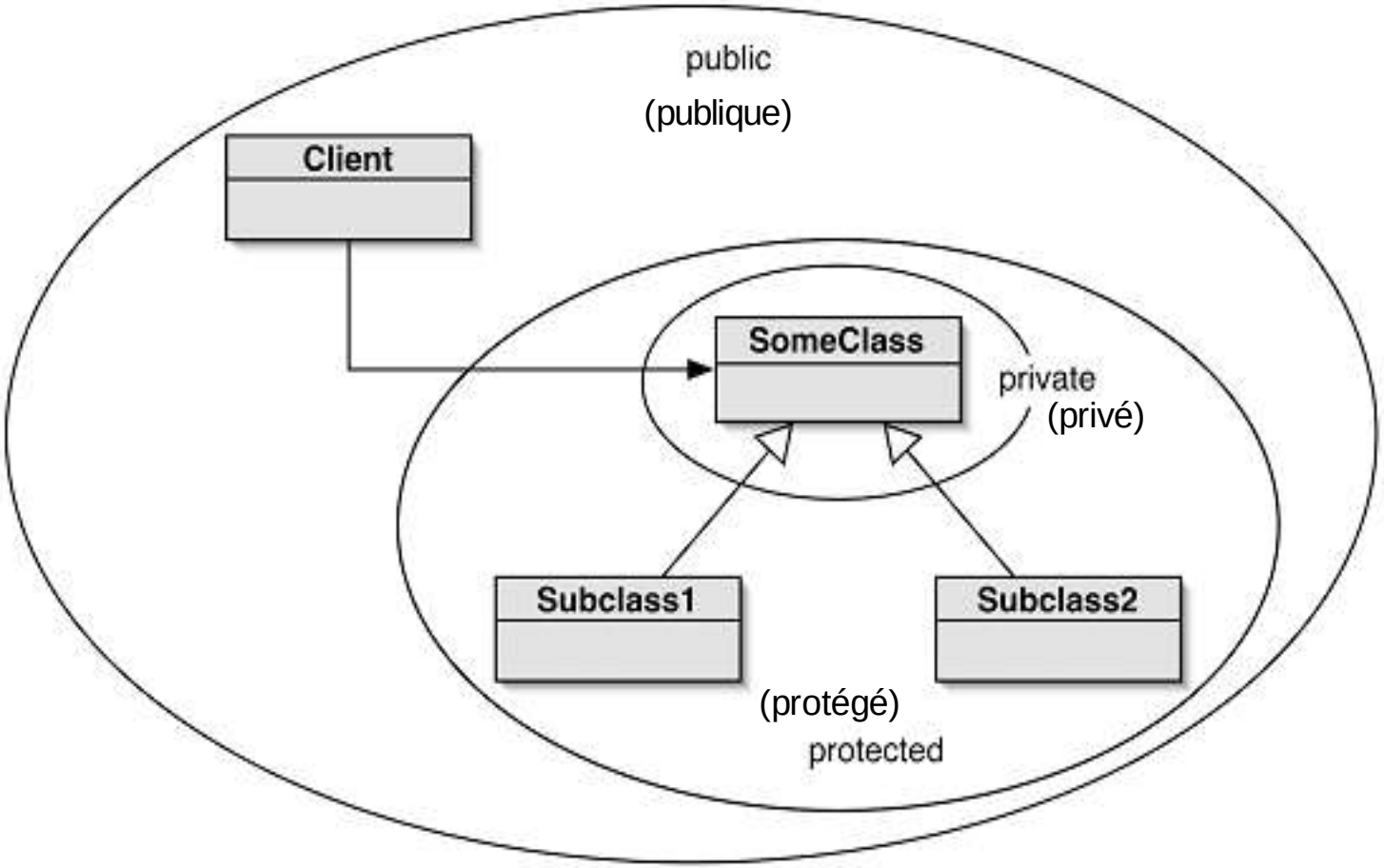
Surcharge de `toString` (2)

- On peut souvent se passer d'une méthode **`print`** explicite dans une classe:
  - `System.out.println(item.toString());`
- Les appels à **`println`** avec un simple objet génèrent des appels à **`toString`** de la classe de l'objet:
  - `System.out.println(item);`

Accès protégé

- L'accès privé (**`Private`**) dans la superclasse peut être trop restrictif pour une sous-classe.
- La relation d'héritage la plus proche est prise en compte par l'*accès protégé*.
- L'accès protégé est plus restrictif que l'accès publique.
- Nous recommandons cependant de conserver les champs privés (**`private`**).
  - et de définir des accesseurs et modificateurs protégés.

Niveaux d'accès



Résumé

- Le type déclaré d'une variable est son type statique.
  - les compilateurs vérifient les types statiques.
- Le type de l'objet qu'elle contient est son type dynamique.
  - les types dynamiques sont utilisés à l'exécution.
- Les méthodes peuvent être surchargées dans une sous-classe.
- La recherche de méthode commence avec le type dynamique.
- L'accès protégé prend en compte l'héritage.

Sommaire général