Conception de logiciels intranet : intergiciels à objets répartis (NSY102)

Objets et Valeurs

Tristan Crolard

Laboratoire CEDRIC
Equipe « Systèmes Sûrs »

tristan.crolard@cnam.fr

cedric.cnam.fr/sys/crolard

Les objets, les classes et les méthodes

- Un *objet* possède une identité propre
  (penser aux objets du monde réel : chaise, table ...)
- Ils sont regroupés en *classes* (distinguer la notion de “chaise” d'une chaise particulière)
- Les classes décrivent les attributs et les méthodes applicables (comment décrire une chaise ? que peut-on faire avec ?)

Exemple – classe Person

```
class Person {
  String firstName;
  String lastName;

  Person(String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  public String toString() {
    return firstName + lastName;
  }
}
```

Exemple – instances

Utilisation de la classe `Person` :

```
Person pierre = new Person("Pierre", "Dupond");
Person pierre2 = new Person("Pierre", "Dupond");
```

Attention, l'égalité teste l'identité d'objet. Par exemple, l'égalité suivante :

```
pierre == pierre2
```

renvoie `false`.

La classe String

Attention, une String est aussi un objet. Faites le test suivant :

```
String happyString = "Happy";
String birthdayString = "Birthday";
String firstString = happyString + "␣" + birthdayString;
String secondString = happyString + "␣" + birthdayString;
System.out.println(firstString == secondString);
System.out.println(firstString);
System.out.println(secondString);
```

Remarque. Pour comparer des String, il faut donc utiliser la méthode equals. L'égalité "==" ne donne le résultat attendu que pour les types atomiques (int, bool, ...)

Les méthodes *non* statiques

- Dans une classe `C`, toute méthode *non* statique `f` possède un paramètre supplémentaire implicite, appelé `this`, de type `C`.
- Un appel à `f`, se note par exemple : `o.f(arg)`
  Si `f` était statique, on écrirait `f(o, arg)`
  Un appel de la forme `f(arg)` correspond en fait à `this.f(arg)`

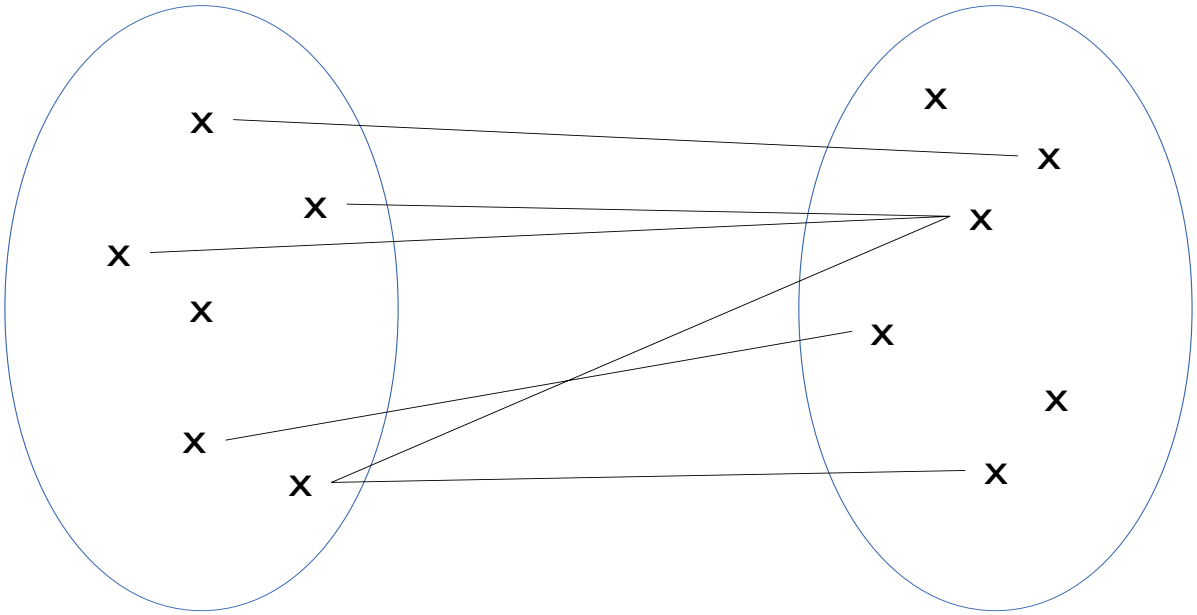
La syntaxe de Java

- Structures de contrôle (boucle, conditionnelle, ...) héritées du C
- Syntaxe pour les tableaux aussi héritée du C
- Nouveautés :
  - Notion de package
  - Notion de classe
  - Méthodes non statiques, `this` et notation « pointée »

Modélisation Orientée Objet – historique

- « Le monde est fait d'objets qui interagissent »
- Modèle Entité-Association (années 70 - méthode MERISE)
  Généralisation/spécialisation (relation « is a »)
- Modélisation Orientée Objet (OMT)
  Diagrammes de classes (notation UML)
  Notions d'aggrégation et de composition

Rappel sur les relations binaires



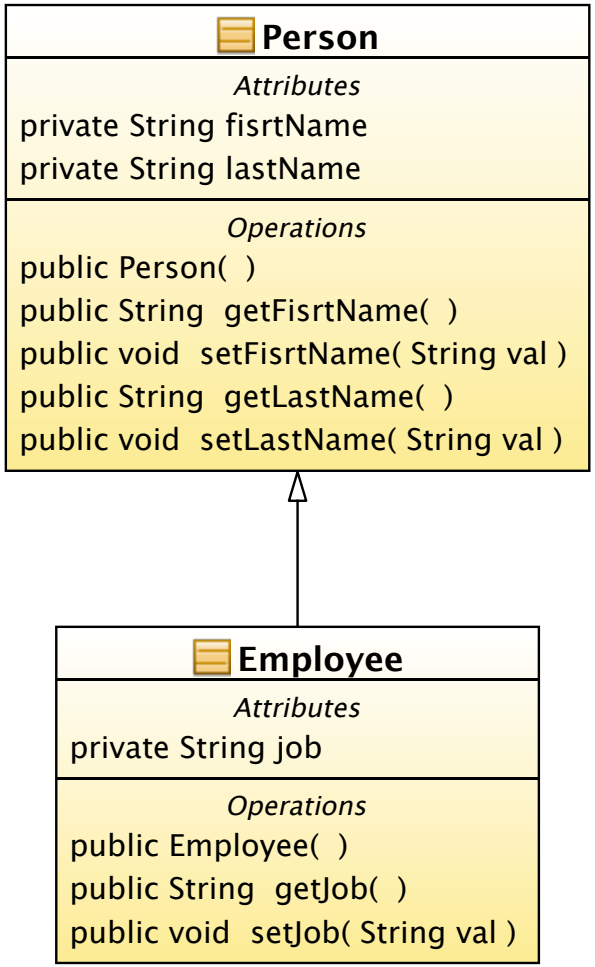
Modélisation Orientée Objet

- Le monde est fait d'objets qui interagissent
- Les objets sont regroupés en classes. Ces classes décrivent les attributs (propriétés) et les méthodes (opérations) communes à ces objets.
- L'interaction entre les objets est modélisé sous forme de relation binaires, qui sont décrites au niveau des classes sous forme d'associations.
- Ces associations sont spécifiées : navigation et multiplicité (qui indiquent comment les implanter).
- Certaines associations sont très particulières : généralisation/spécialisation et aggrégation (ou composition).

Généralisation/spécialisation et héritage

- Une classe Java peut spécialiser une autre classe
- Elle hérite alors des attributs et des méthodes de sa super-classe.
  On parle donc d'héritage dans les langages objet.
- La conversion de type entre une classe et sa super-classe est implicite.
- Les méthodes et attributs visibles sont donnés par les types des variables ou des expressions.
- Les méthodes peuvent être redéfinies.
- La methode executée et celle de la classe de l'objet (celle a été utilisée au moment de la création avec `New`).

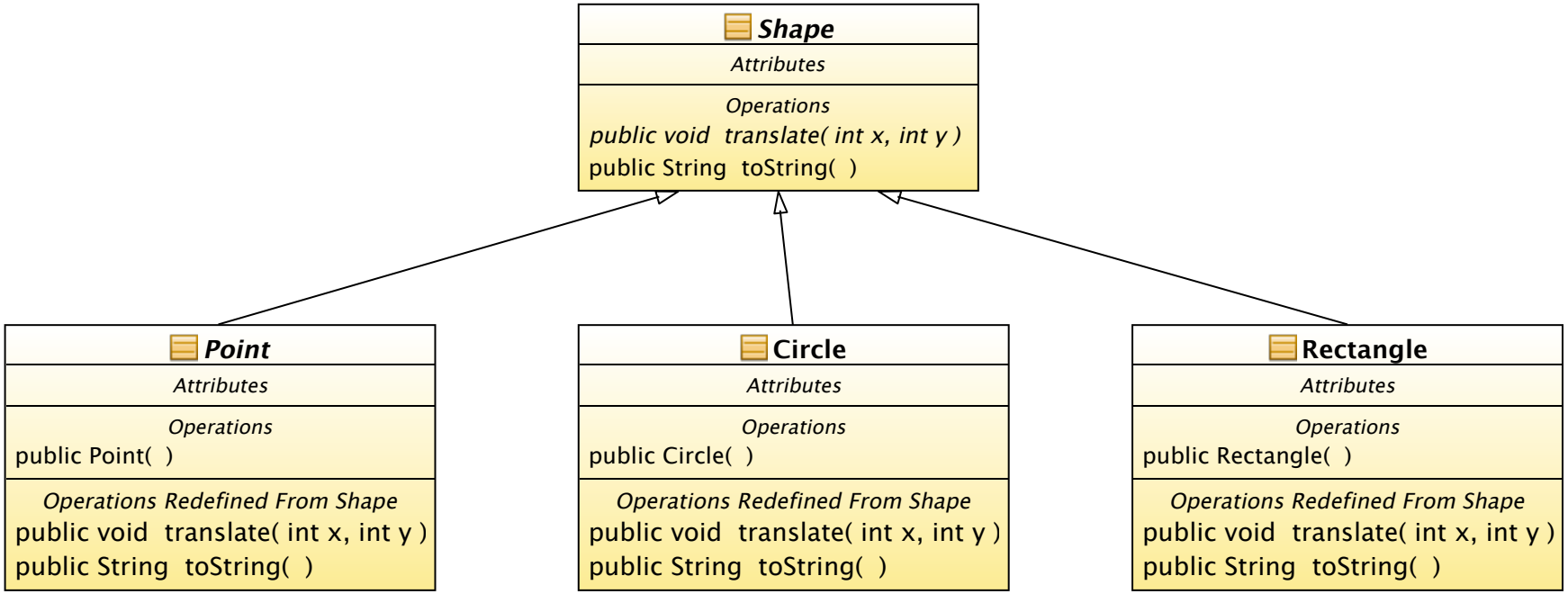
Généralisation/spécialisation – exemple



Classes abstraites et interfaces

- Il est possible de différer l'implantation d'une méthode à une sous-classe en la rendant abstraite (`abstract`).
- Une classe qui possède au moins une méthode abstraite est nécéssairement abstraite : *il n'est pas possible de créer une instance de cette classe*.
- Une classe abstraite peut n'implanter aucune méthode (elles sont toutes abtraites). Elle sert alors uniquement pour le typage. On lui préfère alors la notion d'interface (qui autorise en plus plusieurs super-interfaces en Java).
- On parlera de « type » pour désigner indifférement classe ou interface.

Exemple : figures géométriques



Polymorphisme paramétrique (generics)

- Une classe (ou interface) peut être paramétrée par un ou plusieurs types. Une telle classe est dite générique.
- La syntaxe est, par exemple, `List<E>`
  L'interface `List` de la JDK est paramétrée par le type `E` des éléments.
- On utilise une classe générique en donnant une classe (ou interface) comme paramètre effectif pour `E`. Par exemple, `List<String>`