Conception et programmation orientées objet, rappels

Sébastien Jean

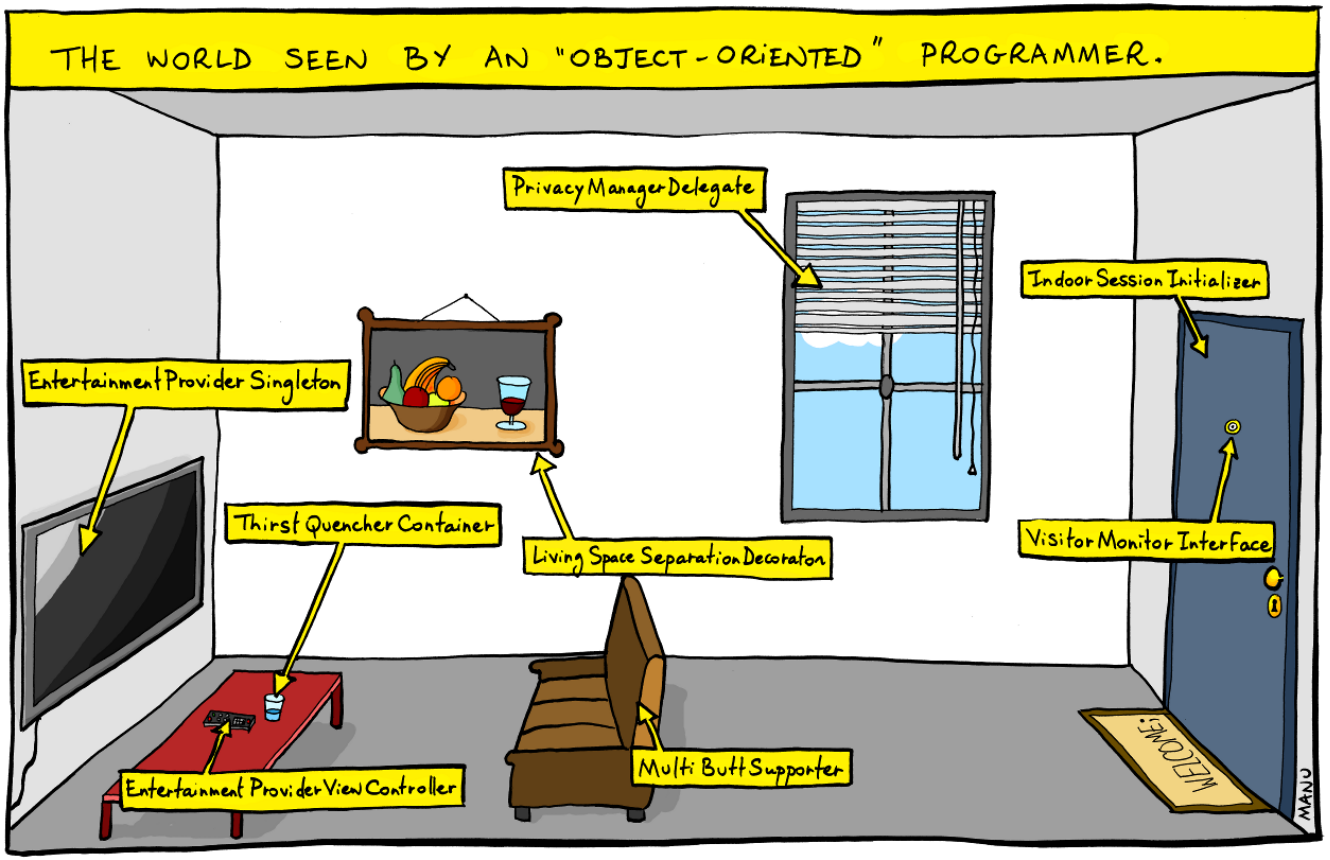
IUT de Valence
Département Informatique

v3.3, 2 septembre 2024

IUT Valence
Université Grenoble Alpes

Objectifs de ce cours

- Revoir les **notions fondamentales objets** . . .
  - Classe, objet, abstraction, . . .
- . . .à travers le prisme de la COO (UML) et de la POO (Java)



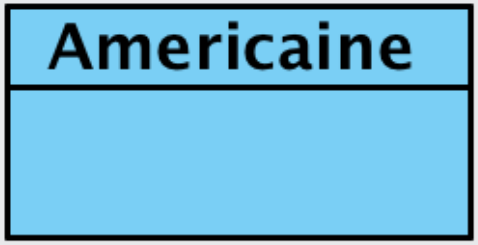
source : www.bonkersworld.net/object-world

IUT Valence Université Grenoble Alpes

Notion de classe

- Une **classe** **encapsule** des données (**attributs**) et des traitements (**méthodes**) communs à un ensemble d'**objets**, et définit un **type**

Notation UML



Code Java

```
public class Americaine
{
}
```

- Les objets sont appelés **instances** de classe

IUT Valence Université Grenoble Alpes

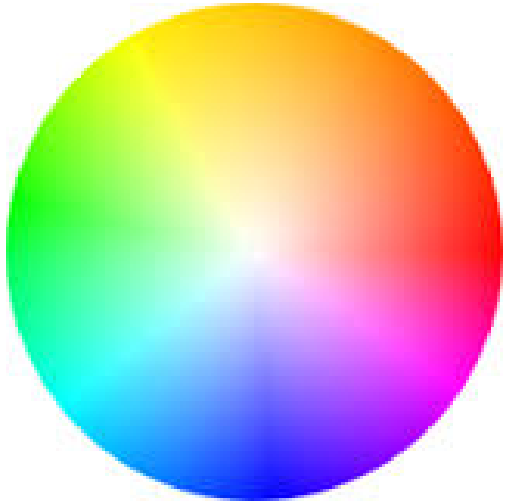
Attributs

- Les **attributs** sont des « données » élémentaires
  - L'ensemble des attributs constitue un **état**
- Les attributs sont **nommés** et **typés**
  - Type primitif ou défini par une classe
- Leurs **valeurs** peuvent être **uniques** ou **multiples** (collections), et rendues **constantes**
- Les attributs peuvent être **masqués** ou non à l'extérieur de la classe
- Les attributs peuvent être :
  - Partagés entre toutes les instances → **attributs de classe**
    - Ils sont alors dits **statiques**
  - Propres à chaque instance → **attributs d'instance**

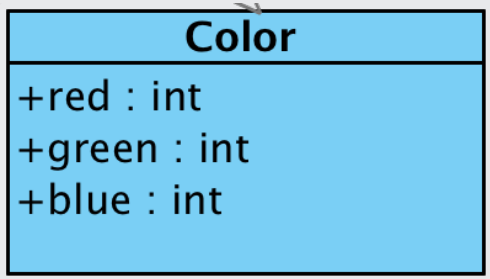
IUT Valence Université Grenoble Alpes

Exemple

La couleur d'un pixel est caractérisée par des niveaux RVB



Notation UML



Code Java

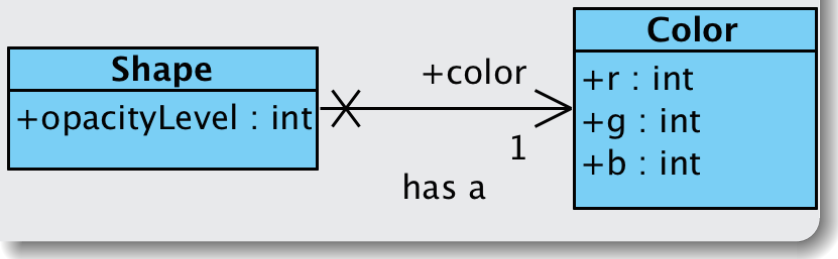
```
public class Color
{
  public int red;
  public int green;
  public int blue;
}
```

IUT Valence Université Grenoble Alpes

Association

- **Dépendance forte** entre classes
  - Relation de type « ***utilise un*** » ou « ***a un*** »
  - Nommage, cardinalité, navigabilité

Notation UML



Code Java

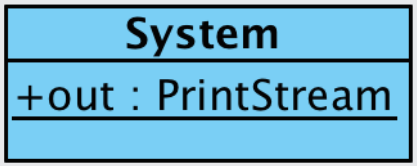
```
public class Shape
{
   public int opacityLevel;
   public Color color;
}
```

- N.B. : la croix (optionnelle) renforce l'expression de la navigabilité lorsqu'il n'y a qu'une seule direction. Il est aussi possible de décrire la nature de la relation si c'est pertinent

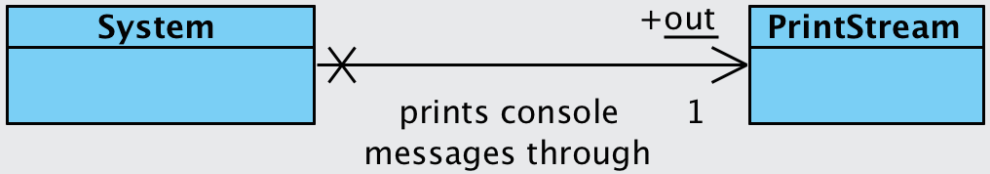
IUT Valence Université Grenoble Alpes

- Attributs **statiques**

Notation UML 1



Notation UML 2



Code Java

```
public class System
{
  public static PrintStream out;
}
```

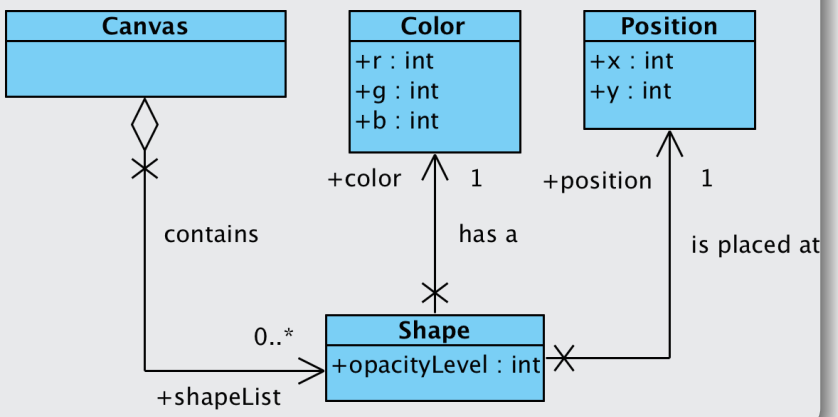
- N.B. : un attribut de type objet peut être représenté de 2 manières, il faut privilégier l'expression « fléchée » des dépendances lorsque la classe cible appartient aussi au diagramme

IUT Valence Université Grenoble Alpes

Aggregation

- **Association forte**
  - Relation de type « ***possède*** »
  - Partage possible

Notation UML



Code Java

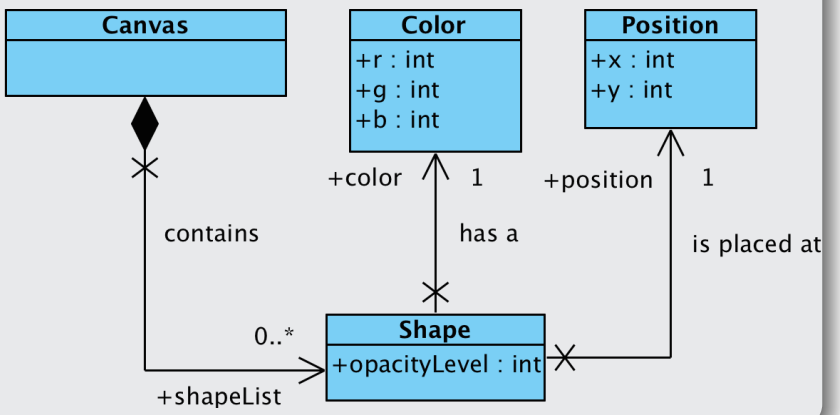
```
public class Canvas
{
   public List<Shape> shapeList;
}
```

IUT Valence Université Grenoble Alpes

Composition

- **Aggrégation forte**
  - Le « contenu » disparait (souvent) avec le « contenant »

Notation UML



Code Java

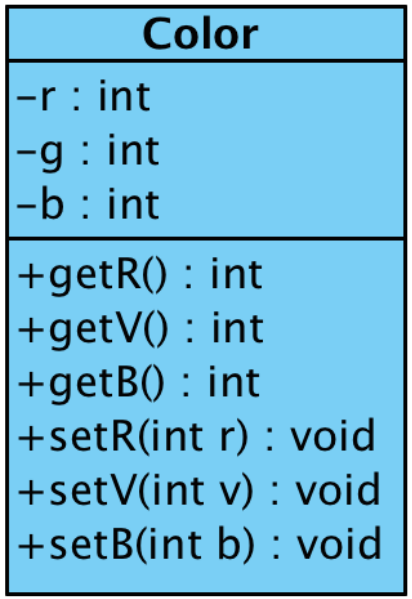
```
public class Canvas
{
   public List<Shape> shapeList;
}
```

Méthodes

- Les **méthodes** sont des « actions » élémentaires
  - L'ensemble des méthodes constitue un **comportement**
  - Une partie du comportement peut être masqué (méthodes internes)
- Les méthodes sont **nommées**, et possèdent
  - (éventuellement) des **paramètres**, ordonnés, nommés et typés, permettant de décliner l'action en fonction d'informations extérieures
  - une **valeur de retour**, typée
- Les méthodes peuvent s'appliquer à :
  - une instance (dont les attributs non statiques peuvent être consultés et/ou modifiés) → **méthodes d'instance**
  - la classe (dont les attributs statiques peuvent être consultés et/ou modifiés) → **méthodes de classe**
    - Elles sont alors dites **statiques**

IUT Valence Université Grenoble Alpes

Méthodes



Code Java

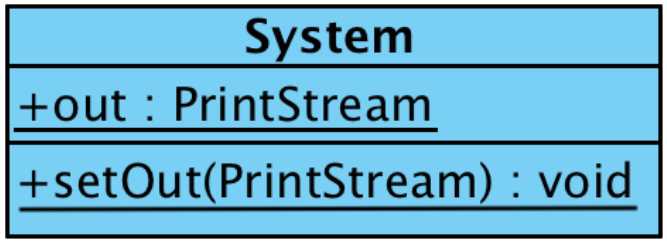
```
public class Color
{
   private int r;
   ...
   public int getR() {return this.r;}
   public void setR(int r) {this.r = r;}
   ...
}
```

- `void` dénote l'absence de valeur de retour

IUT Valence Université Grenoble Alpes

Méthodes

- Méthodes statiques



Code Java

```
public class System
{
  public static PrintStream out;
  ...
  public static void setOut(PrintStream out) {...}
  ...
}
```

IUT A Valence
Université Grenoble Alpes

Objet

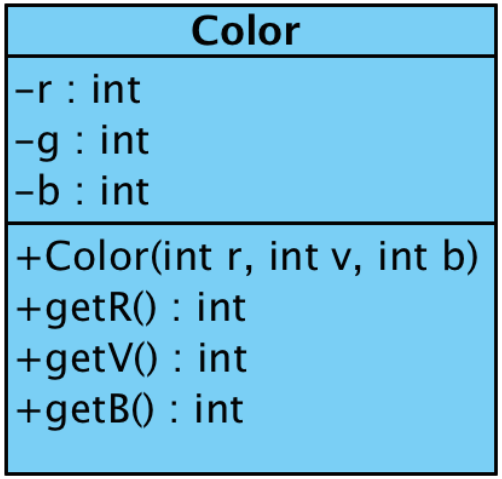
- Un **objet** est une instance de classe, il n'existe qu'à l'exécution
- La classe définit le type et la structure (attributs/méthodes) d'un objet

- Un objet possède un **état interne**, constitué de l'ensemble des valeurs des attributs d'instance
  - L'état n'évolue idéalement qu'à travers des **appels de méthodes**
- Un objet possède une **identité** propre et unique
  - En Java, cette identité s'appelle référence

IUT Valence Université Grenoble Alpes

Constructeur

- La création d'un objet (**instanciation**) passe par la définition et l'invocation d'un **constructeur**



Code Java

```
public class Color
{
   ...
   public Color(int r0, int v0, int b0)
   {
      this.r = r0;
      this.v = v0;
      this.b = b0;
   }
      ...
}
```

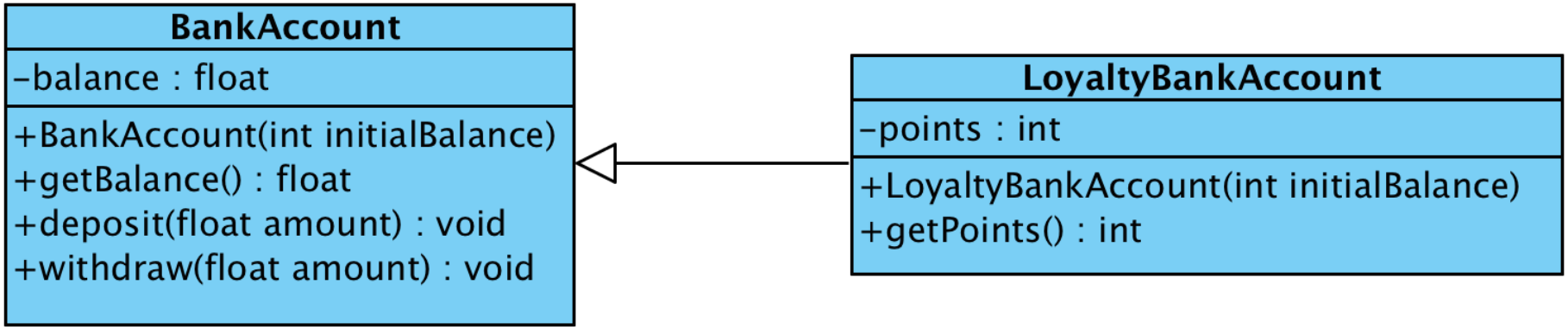
Code Java

```
Color cyan  = new Color(0,255,255);
```

IUT Valence Université Grenoble Alpes

Héritage, polymorphisme

- Réutilisation par **raffinement de concept**
  - Ajout d'attribut(s)
  - Ajout de méthode(s)
  - Redéfinition de méthode(s)



- Le **polymorphisme** est la capacité d'un objet a pouvoir être vu à travers plusieurs types

IUT Valence Université Grenoble Alpes

Héritage, polymorphisme

Code Java

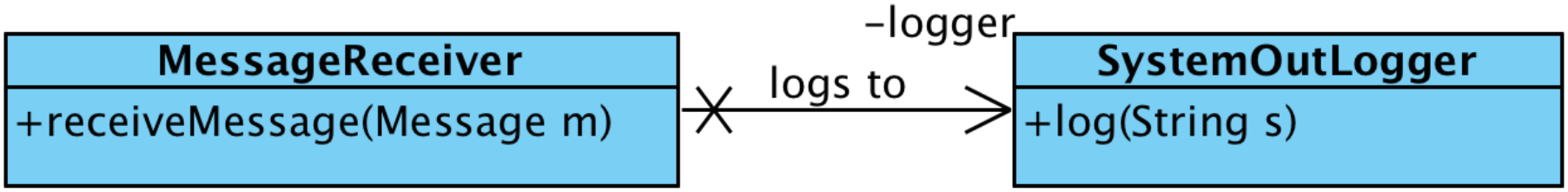
```
public class LoyaltyBankAccount extends BankAccount
{
  private int points;

  public LoyaltyBankAccount(int initialBalance)
  {
    super(initialBalance);
    this.points = 0;
  }
  ...
  public void deposit(int amount)
  {
    super.deposit(amount);
    this.points += amount / 100;
    ...
  }

}
```

Interfaces, classes abstraites

- Exemple (à ne pas forcément suivre . . .)

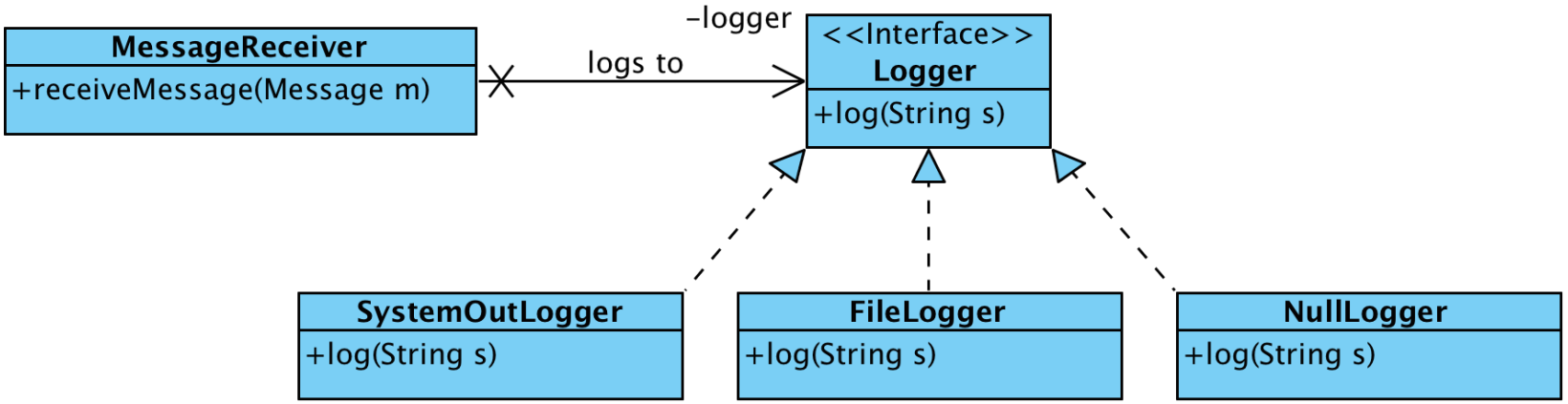


- **Dépendance vis-à-vis de l'implémentation**
  - L'instance de `MessageReceiver` possède un attribut de type `SystemOutLogger` dont la valeur est une référence vers une instance de `SystemOutLogger`

IUT Valence Université Grenoble Alpes

Interfaces, classes abstraites (suite)

- Une **interface** est un type purement abstrait, et ne définit (sauf depuis Java 8) que des signatures de méthodes
- Les interfaces sont implémentées par des classes (devant définir l'implémentation de toutes les méthodes) qui peuvent se substituer
  - **Indépendance vis-à-vis de l'implémentation**

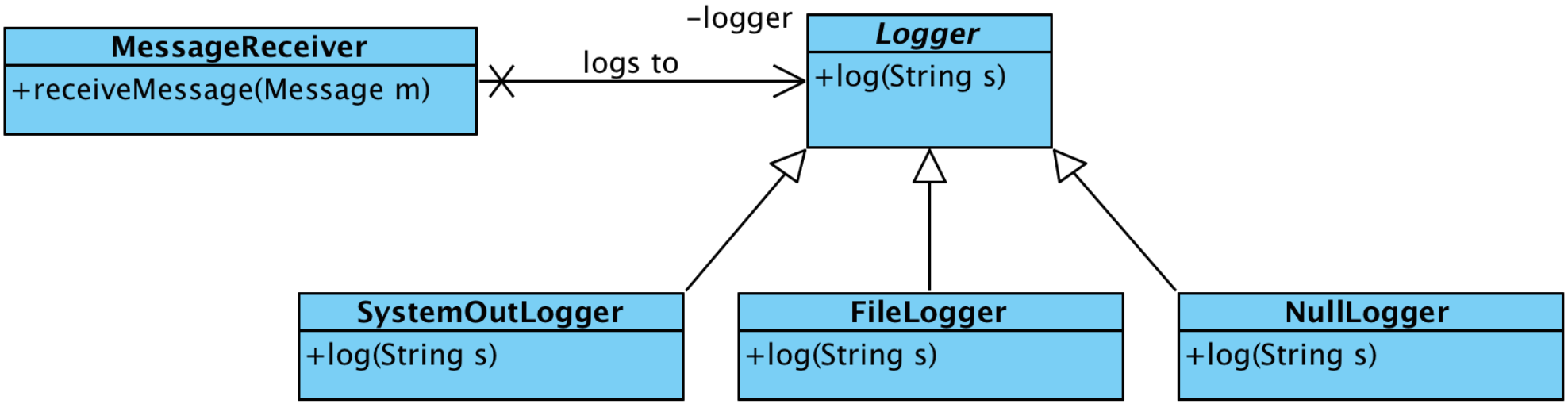


- L'instance de `MessageReceiver` possède un attribut de type abstrait `Logger` dont la valeur peut-être une référence vers une instance de `SystemOutLogger`, `NullLogger`, `FileLogger`, ...
  - Référence passée en général en paramètre du constructeur

IUT Valence Université Grenoble Alpes

Interfaces, classes abstraites (fin)

- Une **classe abstraite** permet de factoriser les attributs et méthodes communes à toutes les sous-classes



Notion de framework

- Collection de classes (concrètes/abstraites) et d'interfaces liées permettant la **conception de sous-systèmes**
  - Exemple : JDBC
- **API** (*Application Provider Interface*) : ensemble de points d'entrée pour la construction d'applications
- **SPI** (*Service Provider Interface*) : ensemble de points d'entrée pour l'extension du framework

IUT Valence Université Grenoble Alpes

Fin !

IUT Valence
Université Grenoble Alpes