Université De Montpellier
Faculté Des Sciences


**Niveau :** Master 2

**Module :** Évolution et restructuration des logiciels

**HAI913I**

---

# Rapport du TD / TP N°3 : Opérations de refactoring sous Eclipse

---

*Supervisé par :*
M. Abdelhak-Djamel Seriai
M. Marianne Huchard

*Réalisé par :*

Ahmed Kaci
Yanis Allouch

2021/2022

# Table des matières

Lien du dépot gitlab du projet.

# 1 Personne1

Le schéma illustre la conception d'un programme qui à pour objectif de modéliser une version simplifier d'une base de données de films. Ce programme pas très bien faits et présente des défauts à corriger grâce à une des opérations de refactoring fournis par Eclipse.

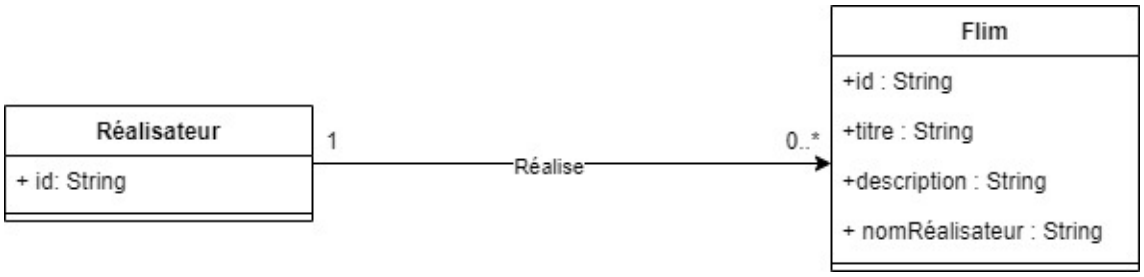



FIGURE 1 – Première version du diagramme de classe *Film*

## 1.1 Opération nécessitant un refactoring

En vérifiant le diagramme présenté ci-dessus, on remarque une incohérence dans la représentation des informations concernant le réalisateur dans la classe Film. en effet, cette dernière possède un lien vers la classe Réalisateur ainsi qu'un attribut décrivant le nom de ce réalisateur.
De ce fait, il serait judicieux d'utilisé la méthode de refactor **move** fournit par éclipse dans le but de déplacé le nom et les méthodes concerant le réalisateur dans la classe adéquate.

— *Move* : *Moves the selected elements and (if enabled) corrects all references to the elements (also in otherfiles).*

## 1.2 Application des refactoring par la personne 2

L'opération de refactor effectué est *move*. Elle permet de déplacer le nom et les méthodes concernant le réalisateur dans la classe Director.

### Capture d'écran de la réalisation

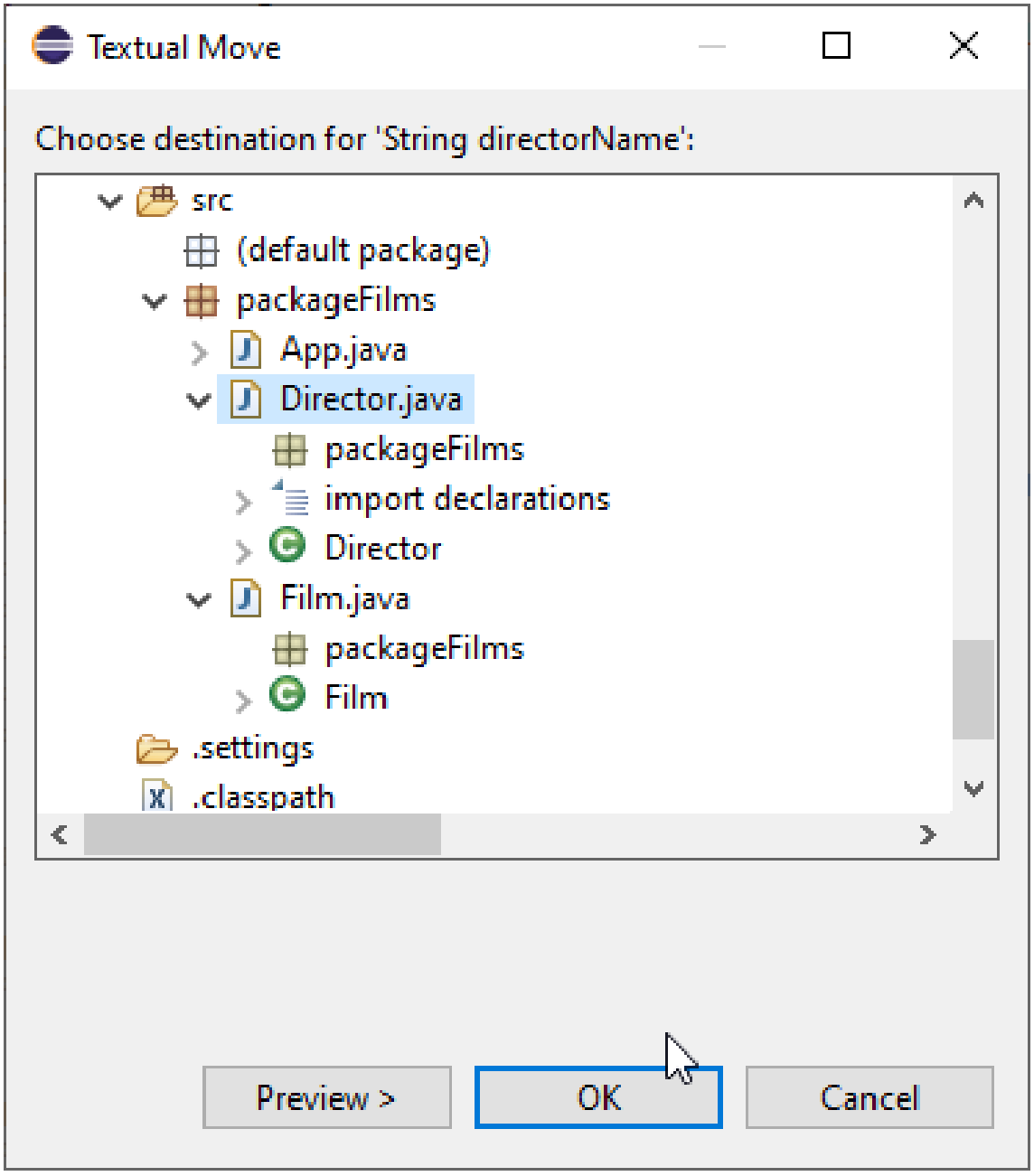



FIGURE 2 – interface obtenue après l'utilisation du menu contextuel refactor move

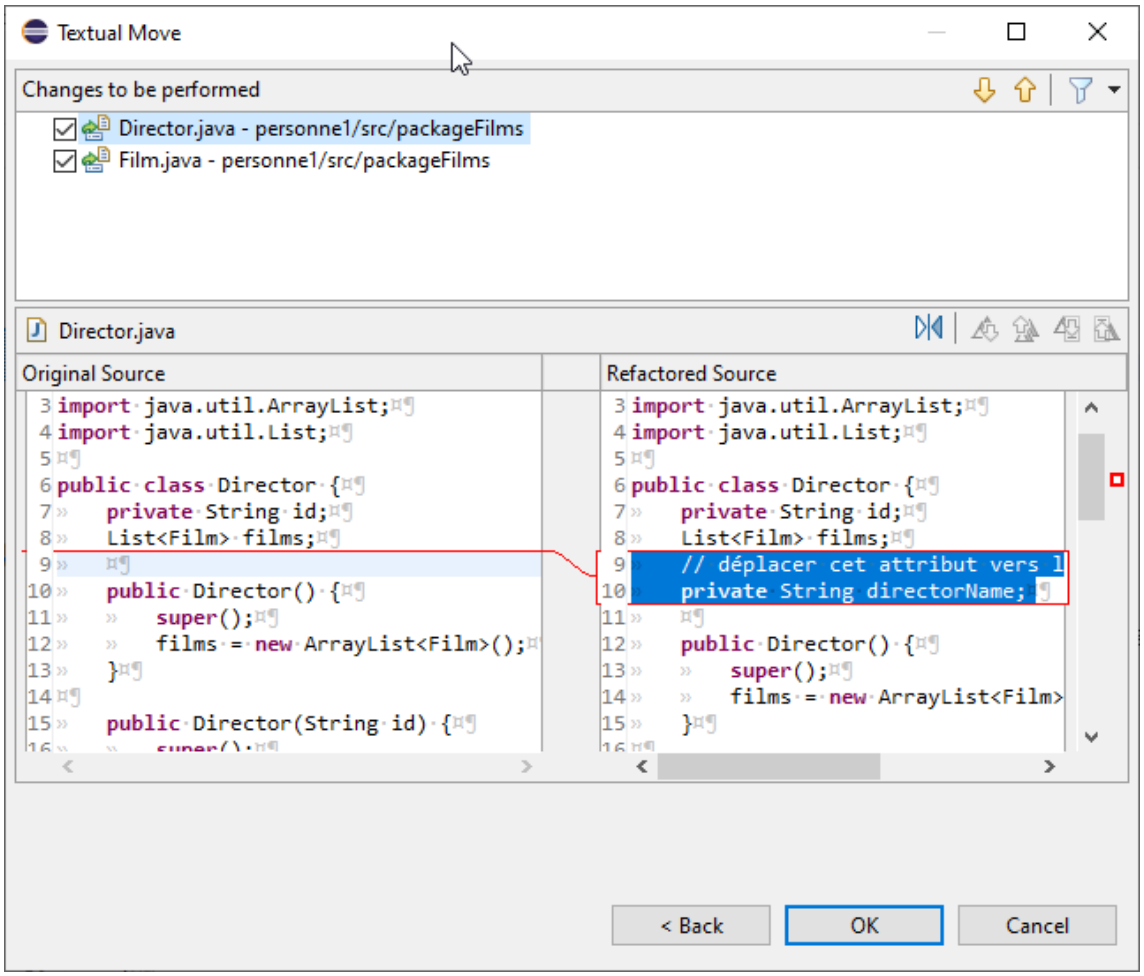



FIGURE 3 – Interface obtenu après avoir appuyé sur le bouton preview

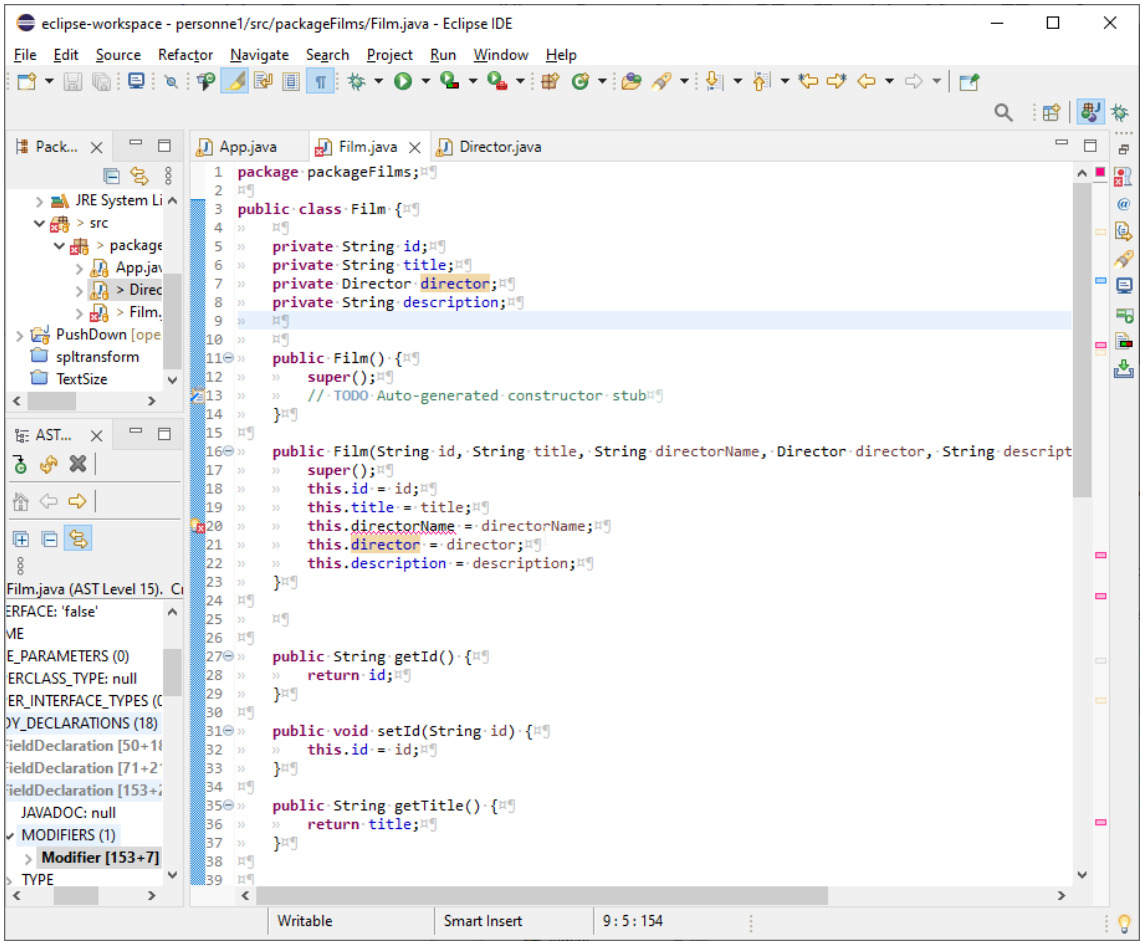



FIGURE 4 – classe Film après le refactor

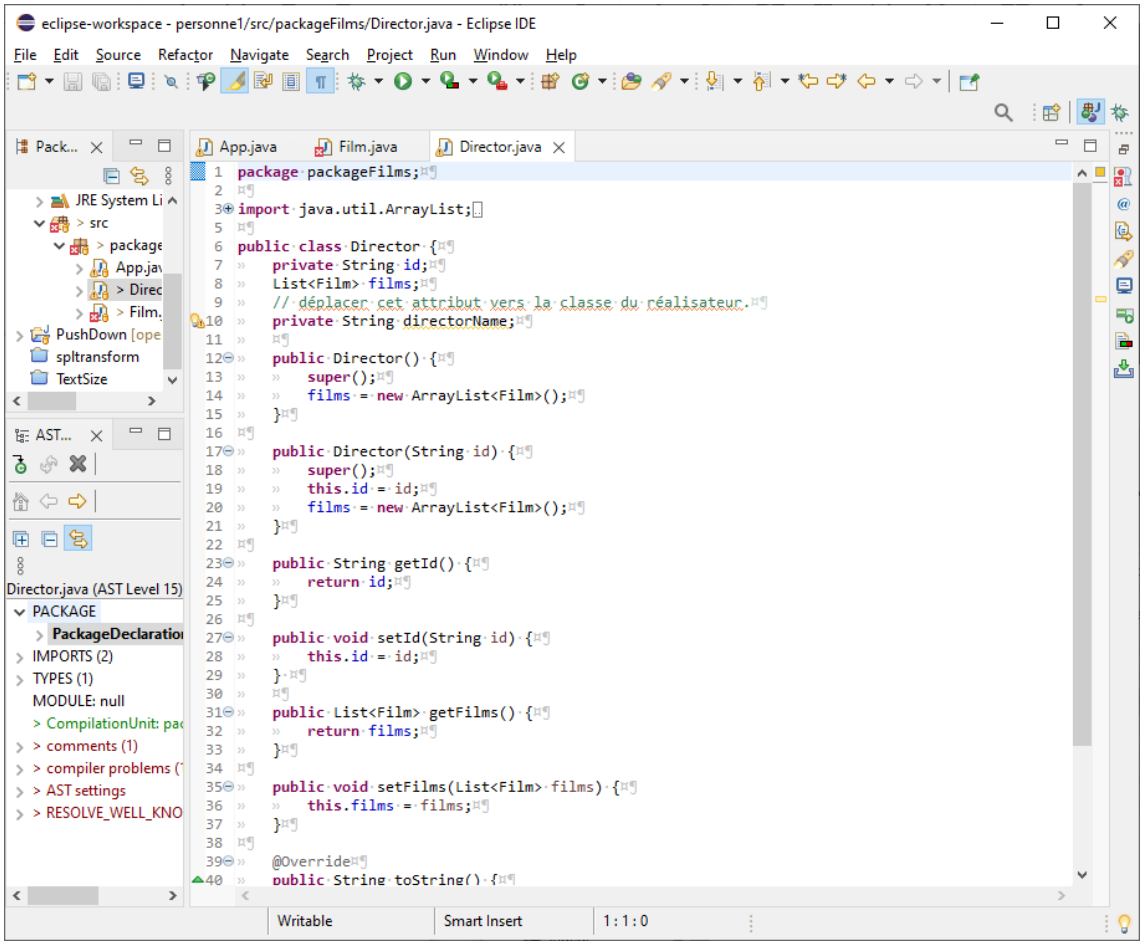



FIGURE 5 – Classe Director après le refactor

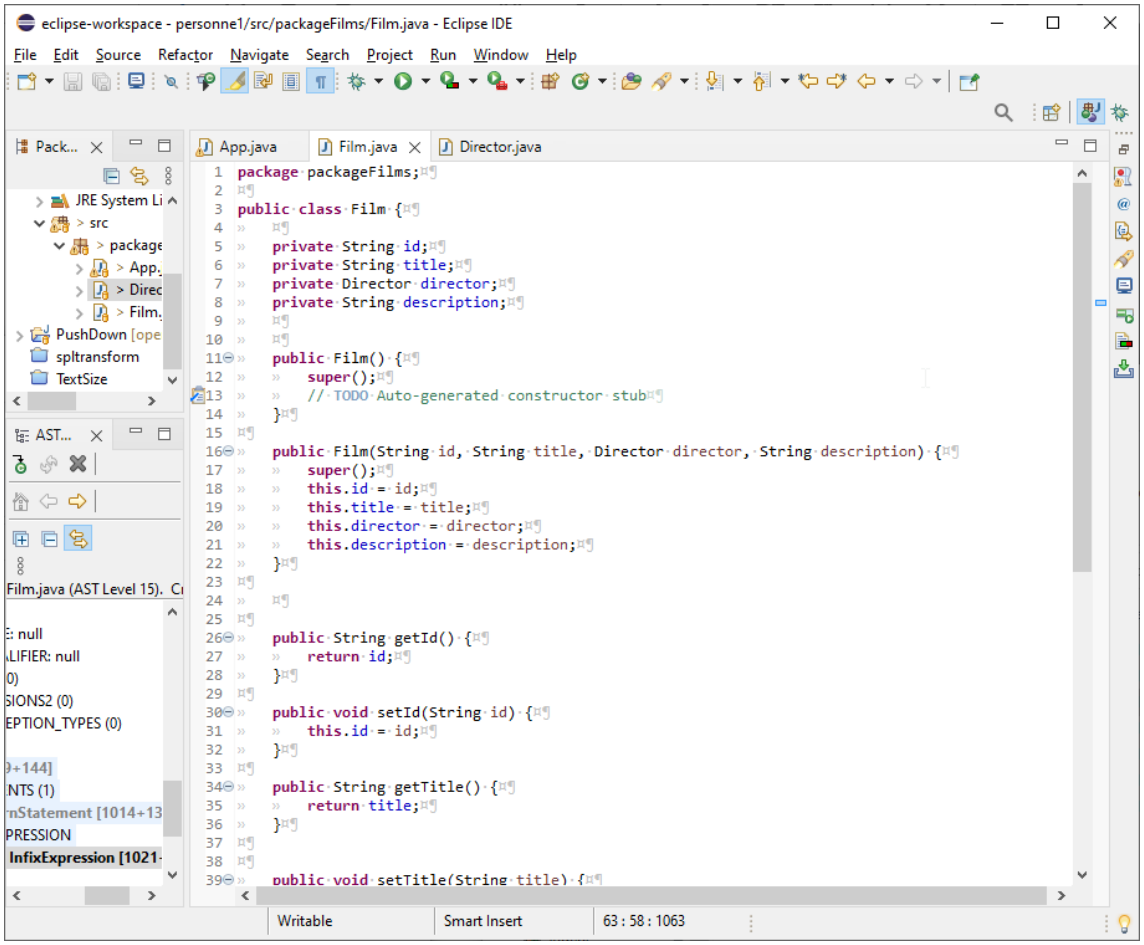



FIGURE 6 – Classe Film après la correction manuel des bugs

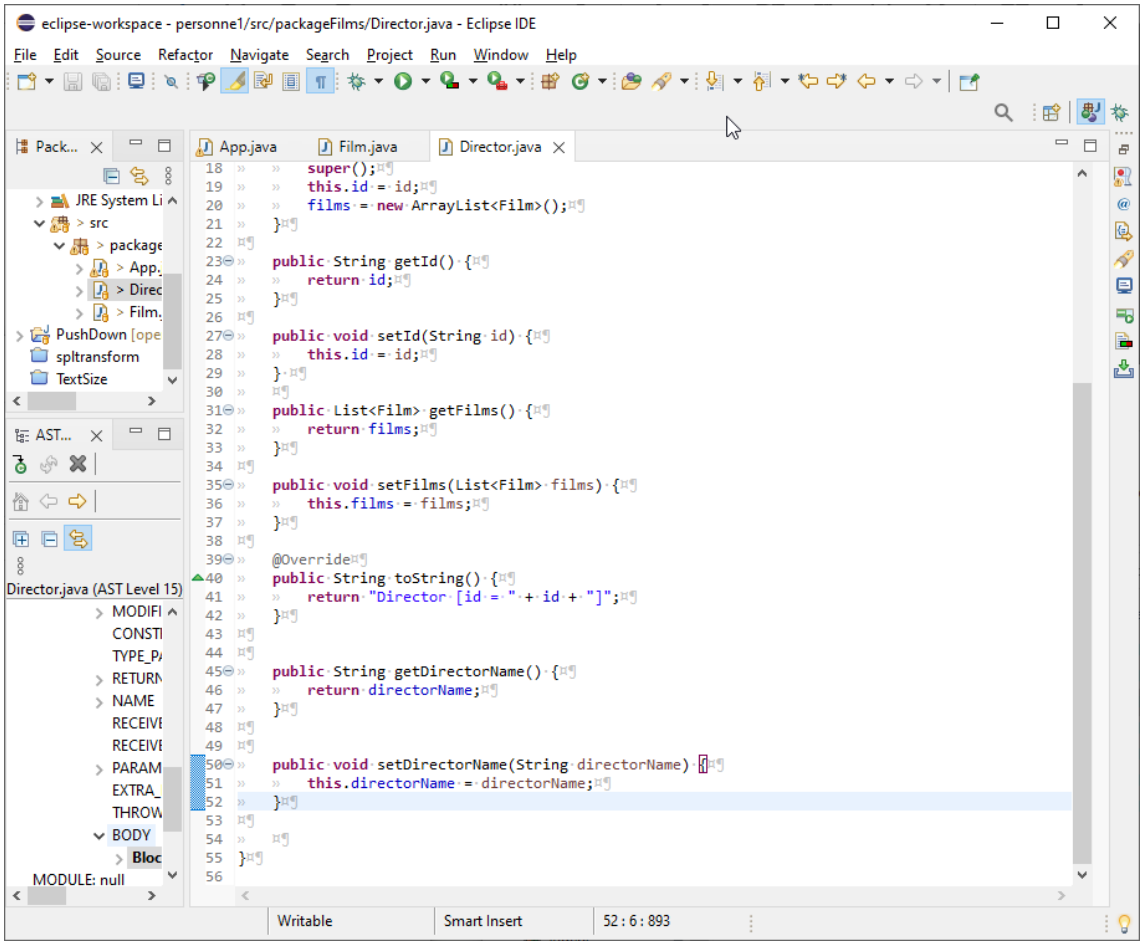



FIGURE 7 – Classe Director après la correction manuel des bugs

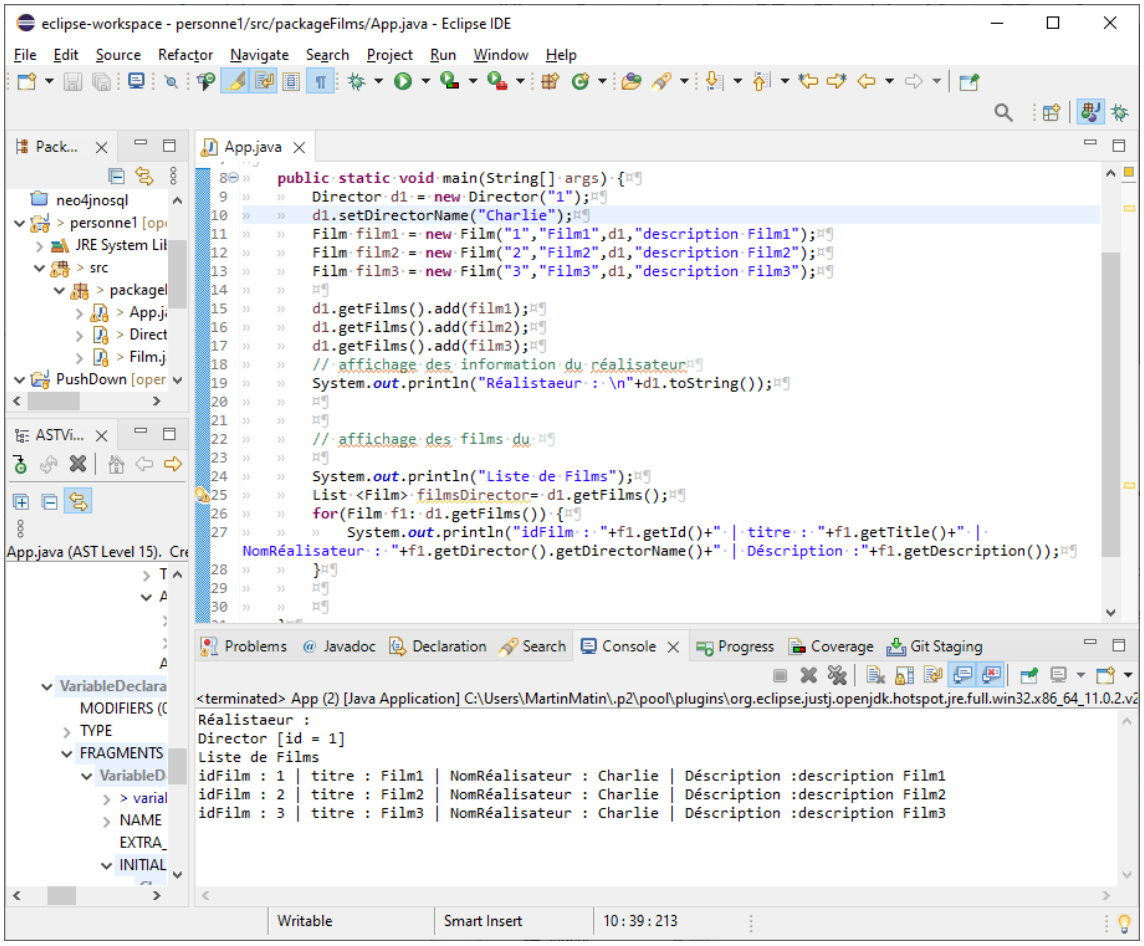



FIGURE 8 – Méthode main après le refactor

Cette opération de refactor apporte un gain de temps lors du développement. Cependant, son implémentation sous Eclipse peut parfois générer artéfacts, car les modifications ne s'étendent pas au contexte. Grâce à cette expérimentation, on distingue des refactors locaux (par exemple move) et des refactors globaux (par exemple Generalization) [1].
Dans ce cas, la modification de la méthode main a été nécessaire, car les modifications apportées par le refactor ont eu un impact sur l'objet Film et Director utilisé.

# 2 Personne2

Le schéma suivant illustre la conception d'un programme qui à pour objectif de le fait qu'une personne peut éventuellement être un étudiant. Ce programme pas très bien faits présente des défauts à corriger grâce à une des opérations de refactoring fournis par Eclipse.

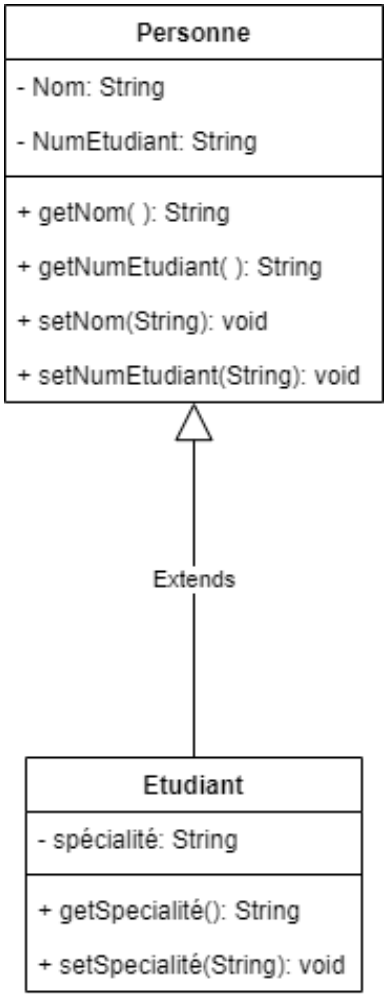



FIGURE 9 – Diagramme de classe

## 2.1 Opération nécessitant un refactoring

— *Push down* : *Moves a set of methods and fields from a class to its subclasses*,

— L'attribt *numEtudiant* et les méthodes *getNumeroEtudiant* et *setNumeroEtudiant* doivent être «pousser vers le bas» dans la classe ***Etudiant***.

## 2.2 Application des refactoring par la personne 1

L'opération de refactor effectué est le *push down*. Elle permet de déplacer l'attribut NumEtudiant et la méthode getNumEtudiant vers la sous classe Etudiant.

### Capture d'écran de la réalisation

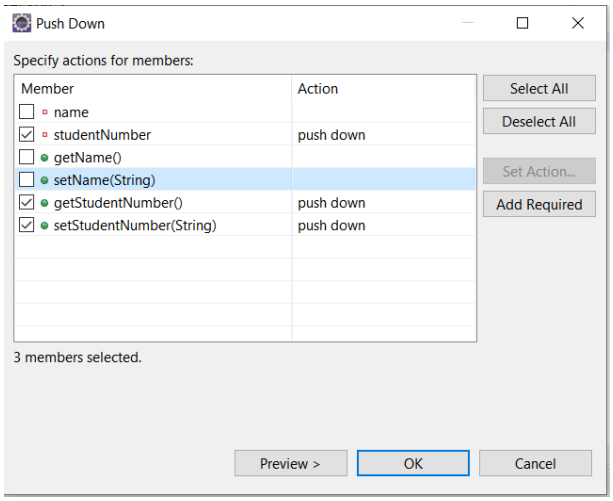



FIGURE 10 – interface obtenue après l'utilisation du menu contextuel refactor push down

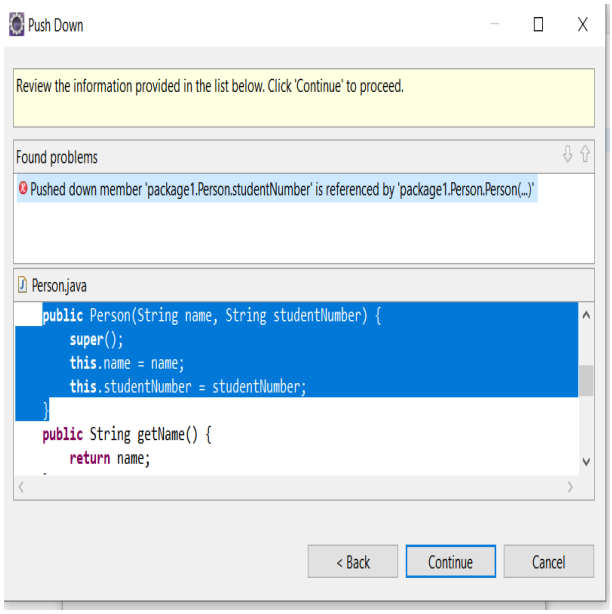
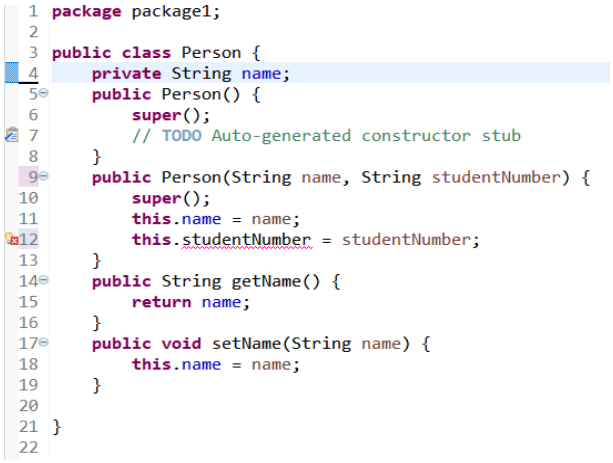
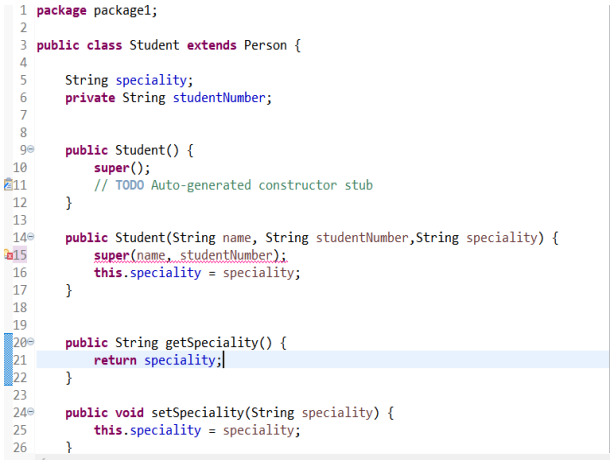



FIGURE 11 – Interface obtenu après avoir appuyé sur le bouton ok



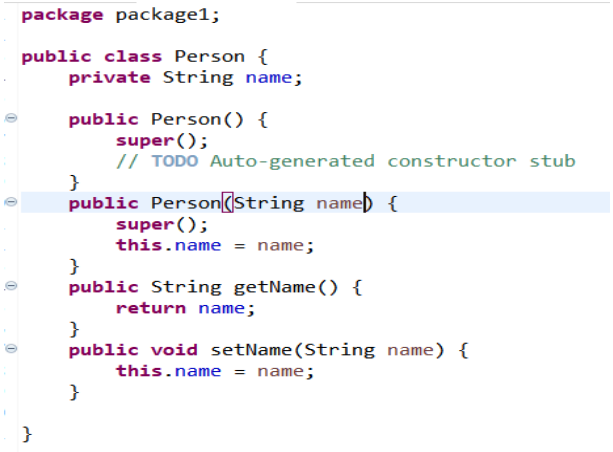
```java
package package1;

public class Person {
    private String name;
    public Person() {
        super();
        // TODO Auto-generated constructor stub
    }
    public Person(String name, String studentNumber) {
        super();
        this.name = name;
        this.studentNumber = studentNumber;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

}

```

FIGURE 12 – classe Personne après le refactor



```java
package package1;

public class Student extends Person {

    String speciality;
    private String studentNumber;


    public Student() {
        super();
        // TODO Auto-generated constructor stub
    }

    public Student(String name, String studentNumber,String speciality) {
        super(name, studentNumber);
        this.speciality = speciality;
    }


    public String getSpeciality() {
        return speciality;
    }

    public void setSpeciality(String speciality) {
        this.speciality = speciality;
    }
```

FIGURE 13 – Classe Student après le refactor



```java
package package1;

public class Person {
    private String name;

    public Person() {
        super();
        // TODO Auto-generated constructor stub
    }
    public Person(String name) {
        super();
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

}
```

FIGURE 14 – Classe Personne après la correction manuel des bugs

```java
package package1;

public class Student extends Person {

    String speciality;
    private String studentNumber;


    public Student() {
        super();
        // TODO Auto-generated constructor stub
    }

    public Student(String name, String studentNumber,String speciality) {
        super(name);
        this.speciality = speciality;
    }


    public String getSpeciality() {
        return speciality;
    }

    public void setSpeciality(String speciality) {
        this.speciality = speciality;
    }
```

FIGURE 15 – Classe Etudiant après la correction manuel des bugs

```java
package package1;

public class App {

    public static void main(String[] args) {
        Student st1 = new Student();
        st1.setName("Charlie");
        st1.setStudentNumber("001");
        st1.setSpeciality("GL");

        System.out.println(st1.getName());
        System.out.println(st1.getStudentNumber());
        System.out.println(st1.getSpeciality());

    }

}
```

```
Console  Problems  Debug Shell  Git Staging  Coverage
<terminated> App (3) [Java Application] C:\Program Files\Java\jdk-16.0.2\bin\javaw.exe  (12 oct. 2021, 20:13:07
Charlie
001
GL
```

FIGURE 16 – Méthode main avant et après le refactor

Cette opération de refactor apporte un gain de temps lors du développement. Cependant, son implémentation sous Eclipse peut parfois générer certains bugs, car les modifications ne s'étendent éventuellement pas dans tous les endroits nécessaires. Dans ce cas, la modification de la méthode main n'a pas été nécessaire, car les modifications apportées par le refactor n'ont pas eu un impact sur l'objet étudiant utilisé. Cependant, si on avait utilisé le refactor *Move* vers la sous classe *Student* avec un objet de type *Personne* il aurait fallu mettre à jour la méthode main.

# 3 Comparaison de 2 catalogues de refactorings

Nos recherches, nous ont montré que certains refactoring proposé dans Eclipse sont **dépendants** du contexte d'ouverture du menu de refactoring. De plus certains refactoring sont cachés dans la fonctionnalité Clean-up d'Eclipse.

Dans un premier temps, nous avons recherché dans le catalogue des refactoring, un refactoring non présent dans les différents menus contextuels des diverses fonctionnalités proposées par Eclipse. Nous avons conclu que le refactoring Substitue Algorithm n'est pas mis en place. En effet, ce dernier ce base sur le refactoring Extract Method dont le résultat peut être intégrée manuellement pour implémenter de nouveaux algorithmes.

Dans un second temps, nous avons recherché dans les différents menus contextuels des diverses fonctionnalités proposées par Eclipse, un refactoring non présenté dans le catalogue des refactoring. Nous avons conclu que le refactoring Migrate JAR File n'est pas présent dans ce catalogue de Martin Fowler.

# 4 Étude de l'opération de refactoring 'Extract Interface'

## 4.1 Extract interface

Nous avons suivi la démarche indiquée dans le sujet de TP pour extraire une interface d'un corpus de classe ou d'une seule classe. Nous présentons nos observations dans la liste suivante :

1. Nous avons observé tout d'abords, que l'extraction d'interface sur l'ensemble des classes en même temps, n'est pas possible. Voir Figure 17,
2. En outre, nous avons observé que l'extraction d'une seule classe proposée dans le menu contextuel des refactoring d'Eclipse, permet de choisir ce que nous souhaitons extraire dans l'interface. Voir Figure 18,
3. De plus, nous avons aussi observé que l'opération *Extract interface* ne factorise pas les signatures des méthodes communes. En effet, cette action, ne c'est pas rendu compte de l'existence de méthodes communes entre les deux implémentations. Autrement l'annotation *override* aurait été ajouté aux méthodes communes. Ainsi, nous pouvons conclure que le refactoring *Extract interface*, agit localement.
4. Après avoir réalisé manuellement la hiérarchie d'interface représentée en Figure 19, nous concluons que la factorisation proposée par l'analyse formelle de concept (voir Figure 20) a permis d'obtenir une factorisation maximale différente de celle que nous avons produites. En effet, nous avons pensé à des factorisations supplémentaires en ajoutant des interfaces qui pourraient être éventuellement implémentés par de nouvelle classe non fournie dans le contexte. Par exemple, l'interface IListTableau pourrait être implémenté par une classe supplémentaire ListTableauDynamique, ListTableauArbres, etc.
   Pour finir, l'utilisation du résultat produit par l'analyse formelle de concepts pourrait être très utiles pour effectuer un refactoring *Extract Interface*, de qualité en un temps record, en particulier sur des programmes complexes. (Voir le chapitre 7 de la section 3 à la section 4, de la référence [1]).

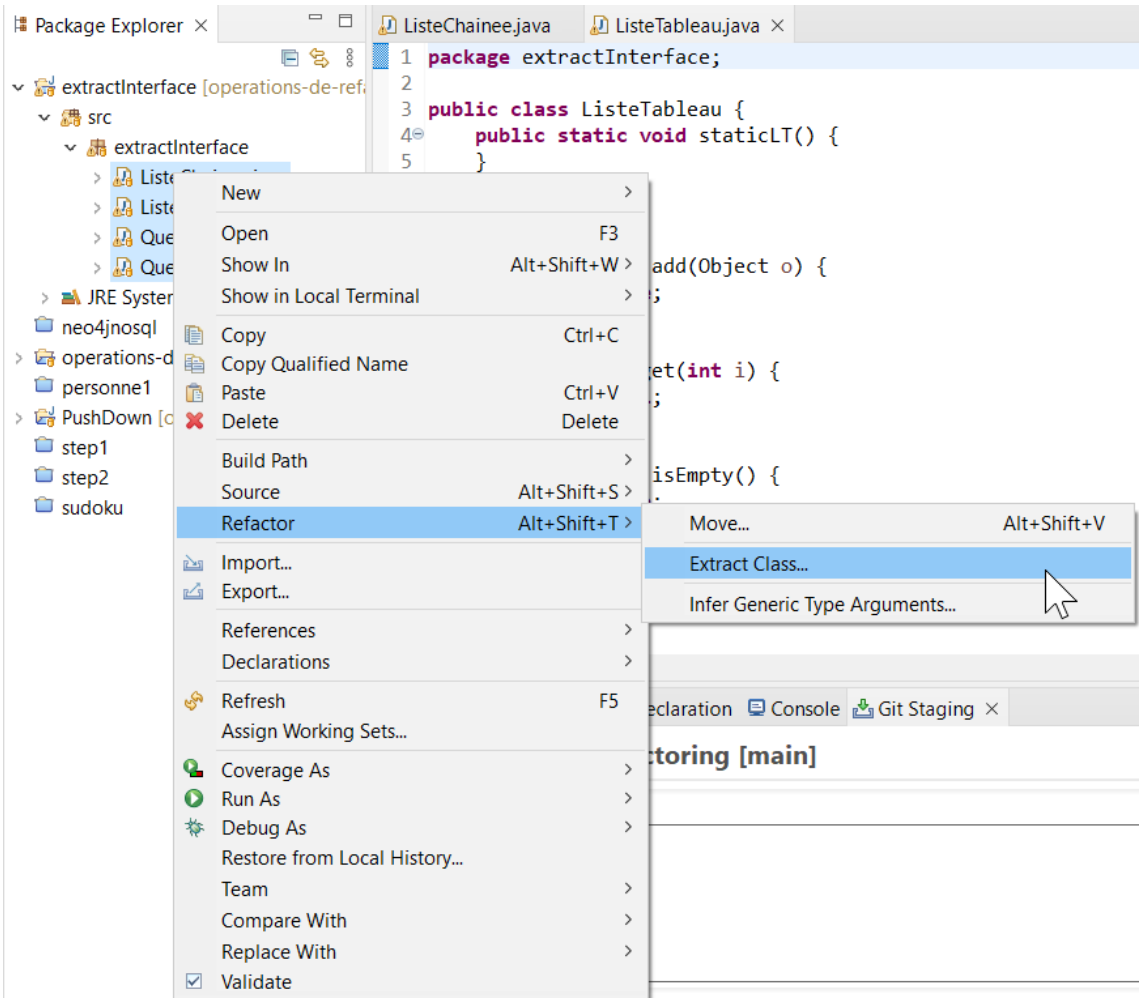



FIGURE 17 – Extract interface non disponible

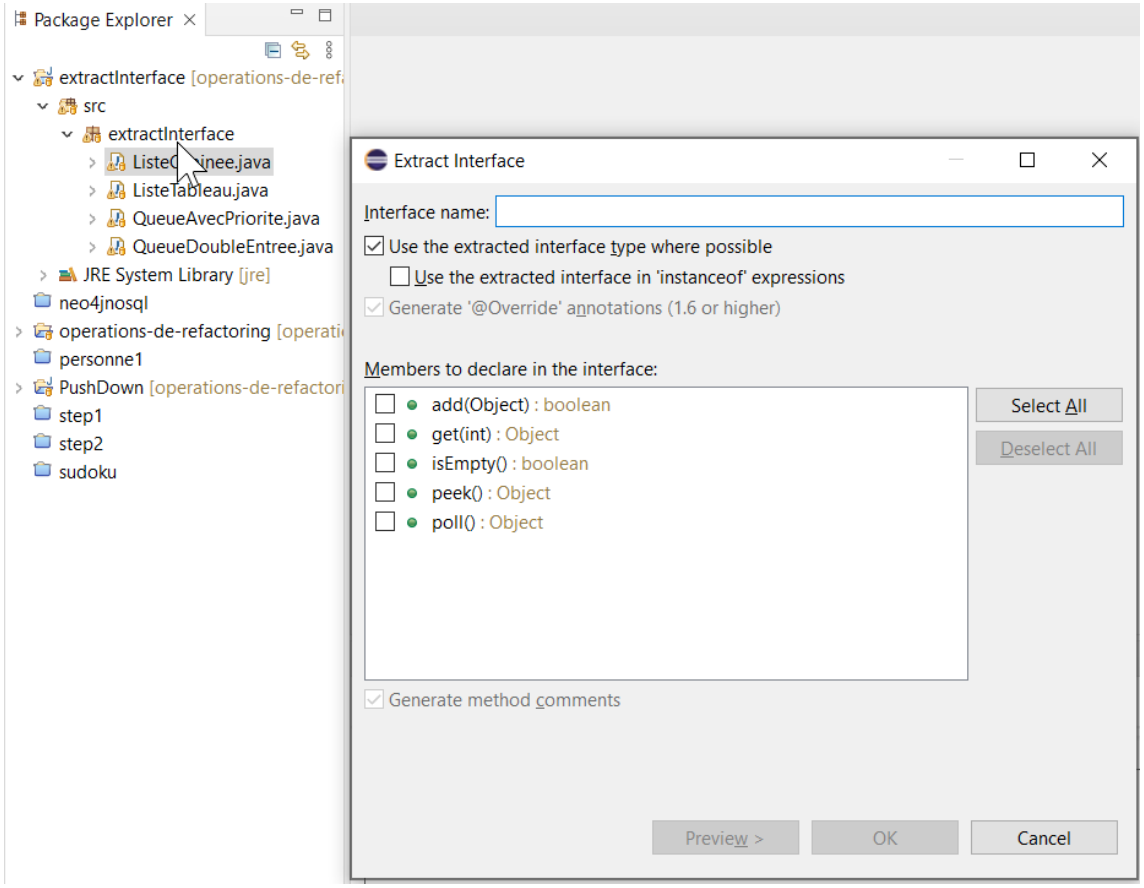



FIGURE 18 – Extract interface d'une seule classe

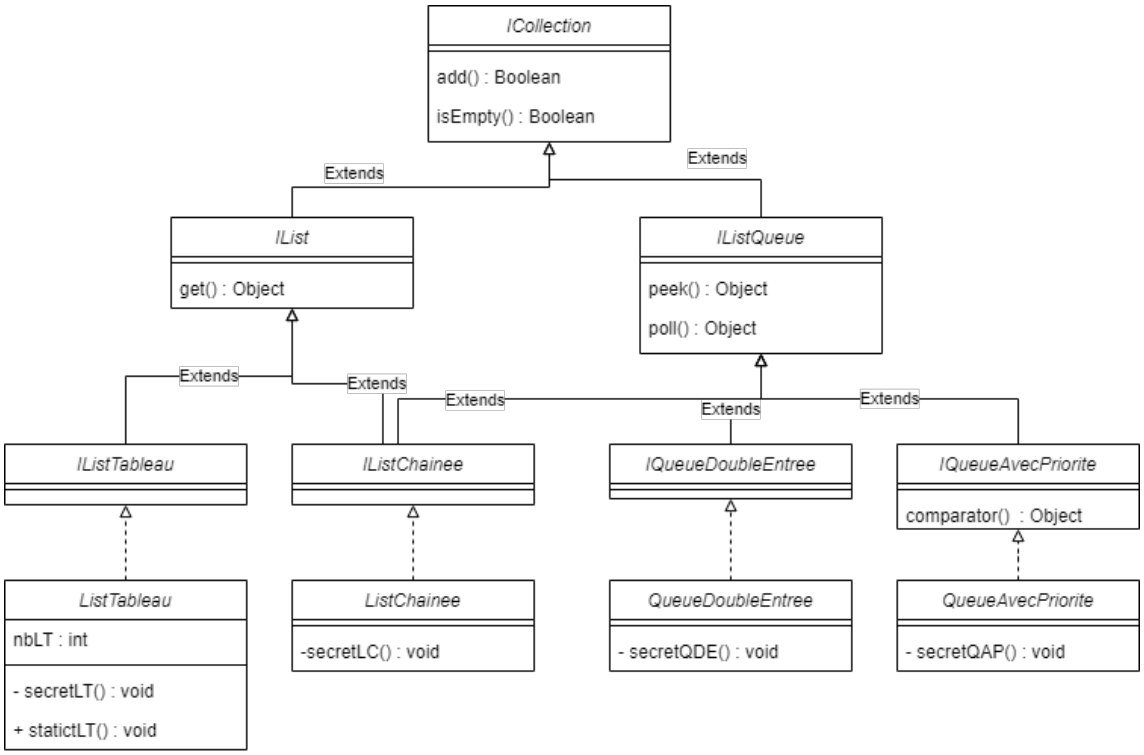



FIGURE 19 – Diagramme de classe des interfaces refactoriser manuellement

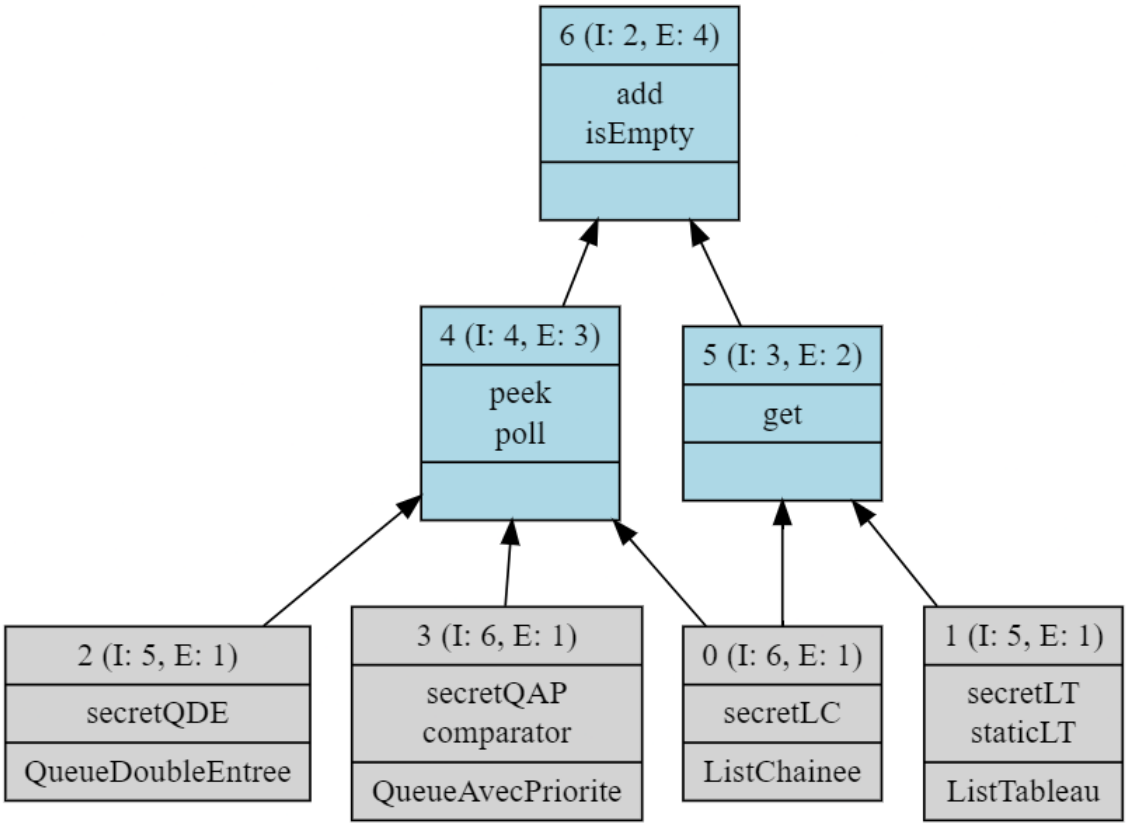



FIGURE 20 – Diagramme de classe des interfaces refactoriser par AFC

# Références bibliographiques

[1] SERIAI. *Évolution et maintenance des systèmes logiciels*. Paris : Hermès Science publications Lavoisier, 2014. ISBN : 9782746245549.