



Évolution et restructuration des logiciels

TP Opérations de refactoring sous eclipse

Ibrahim BERKANE

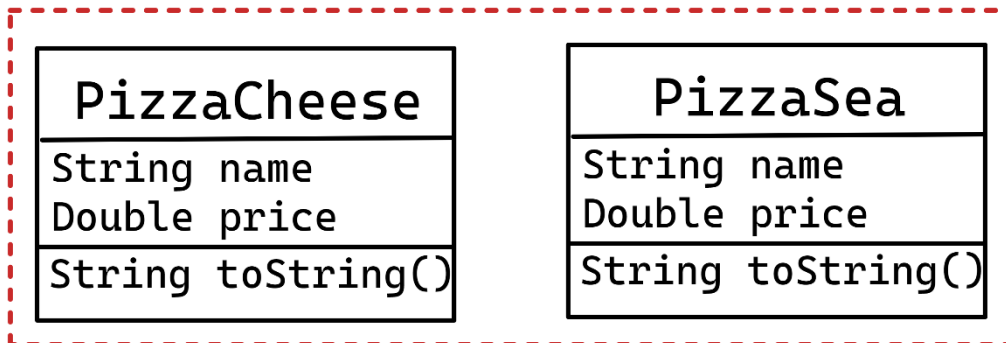
Mahi BEGOUG

Master 2 - Génie Logiciel

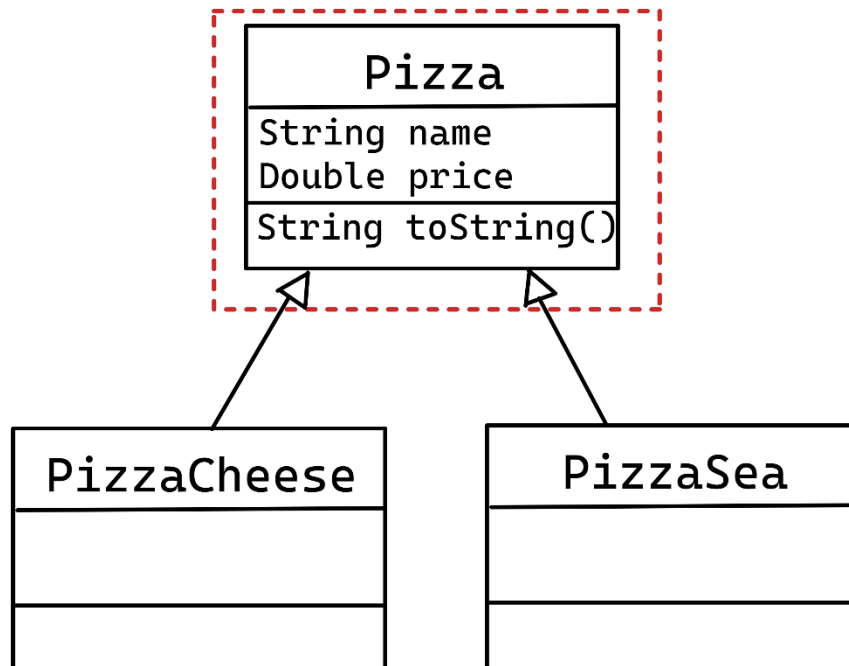
Refactoring des programmes

1 Premier programme

- Nous avons deux classes avec des propriétés et méthodes communes.



- Résultat attendu après refactoring



La première classe :

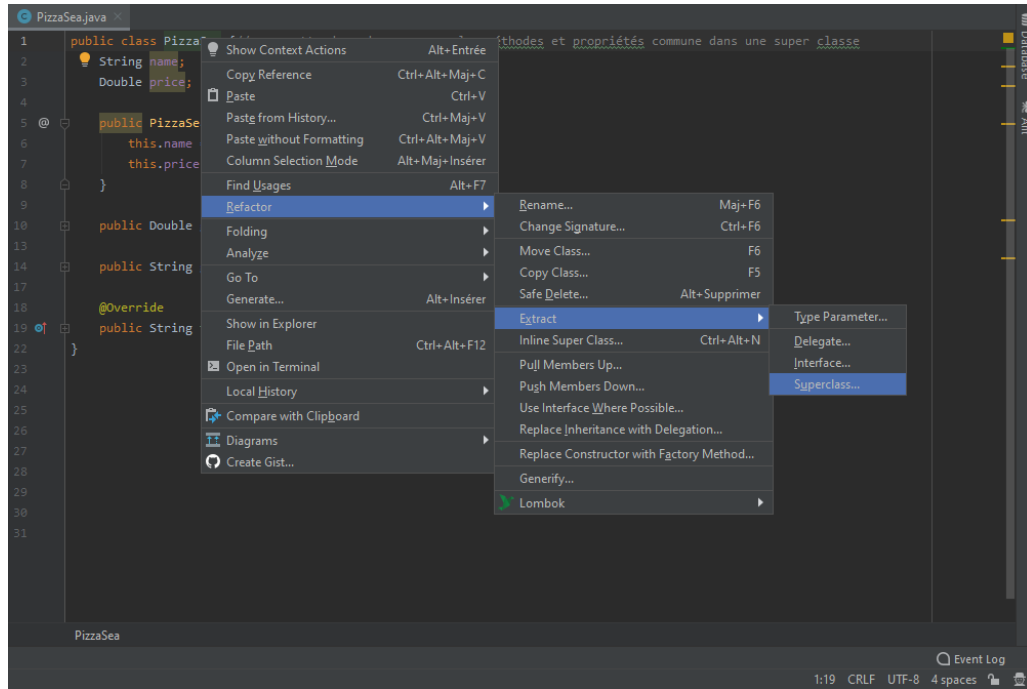
```
PizzaCheese.java x
1 public class PizzaCheese { // nous attendons de regrouper les méthodes et propriétés commune dans une super classe
2     String name;
3     Double price;
4
5     public PizzaCheese(String name, Double price) {
6         this.name = name;
7         this.price = price;
8     }
9
10    public Double getTotal() { return price*0.10 + price; }
13
14    public String getName() { return this.name; }
17
18    @Override
19    public String toString() { return name+" "+price.toString(); }
22
23 }
```

La deuxième classe :

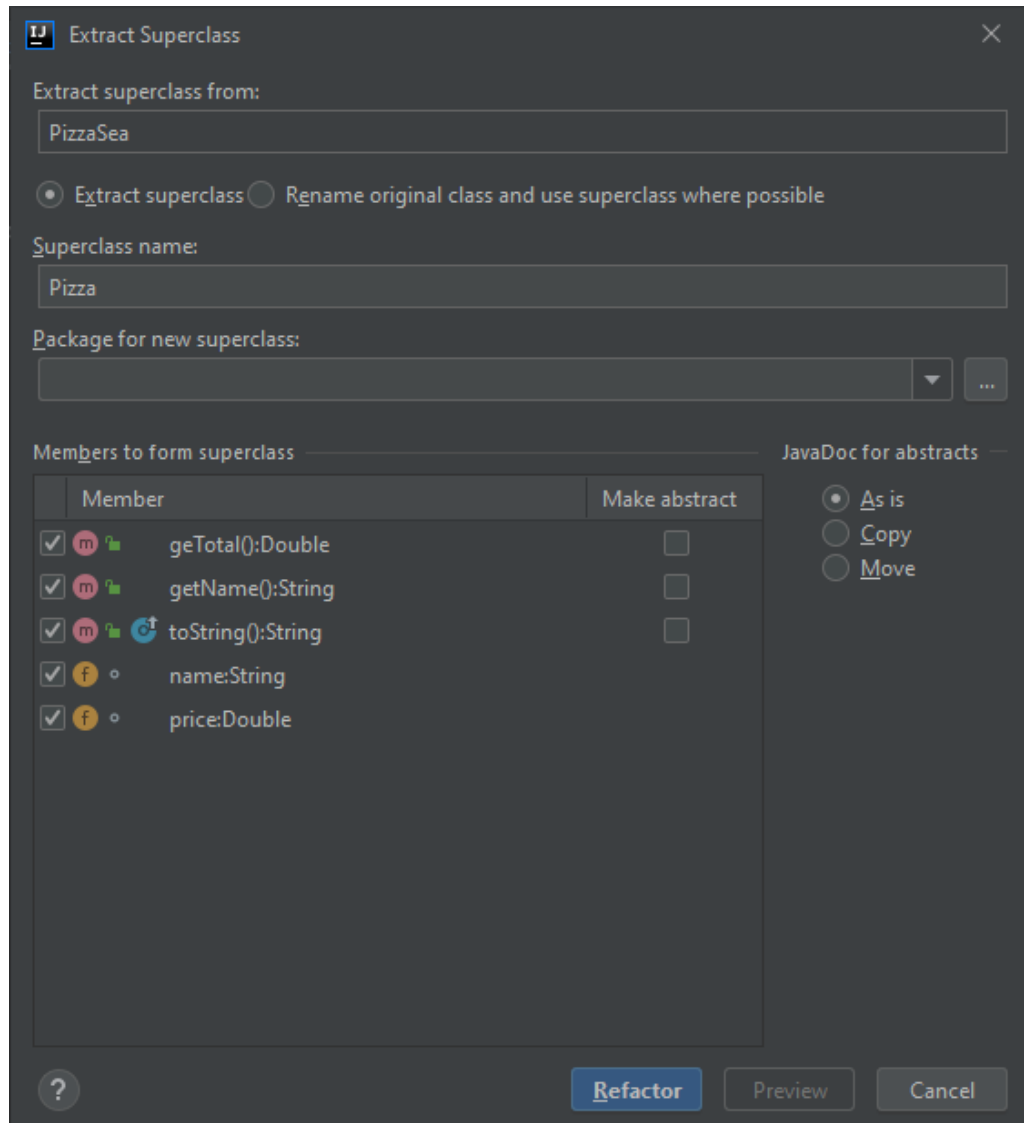
```
PizzaSea.java x
1 public class PizzaSea { // nous attendons de regrouper les méthodes et propriétés commune dans une super classe
2     String name;
3     Double price;
4
5     public PizzaSea(String name, Double price) {
6         this.name = name;
7         this.price = price;
8     }
9
10    public Double getTotal() { return price*0.10 + price; }
13
14    public String getName() { return this.name; }
17
18    @Override
19    public String toString() { return name+" "+price.toString(); }
22
23 }
```

- L'opération réalisé est l'extraction d'une classe supérieure

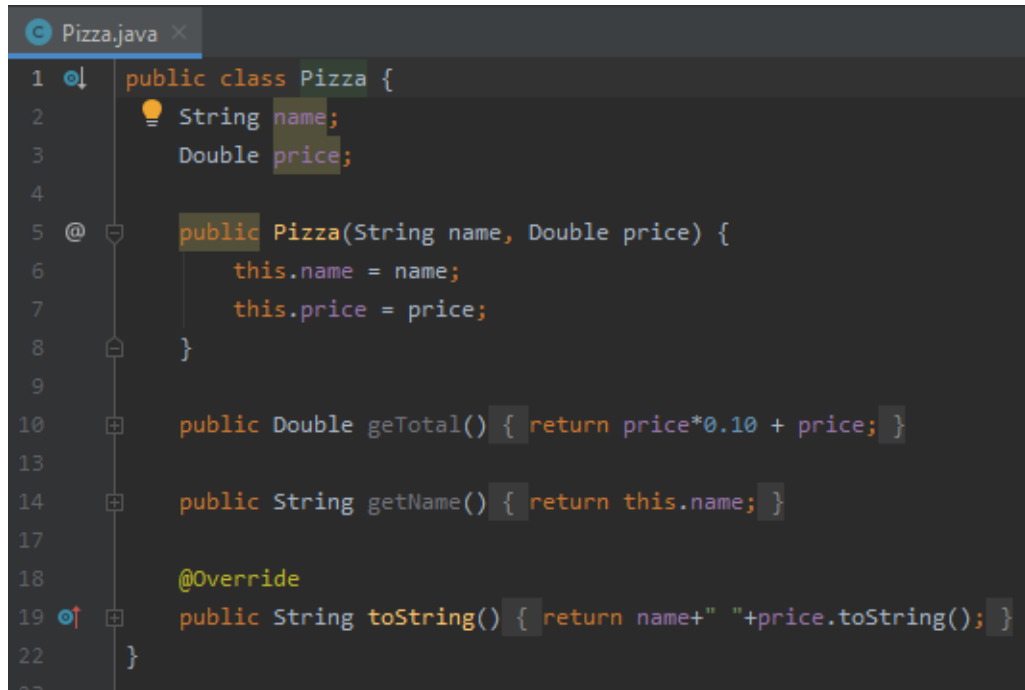
Nous avons choisi Extract SuperClass :



Configuration de la nouvelle classe supérieure :

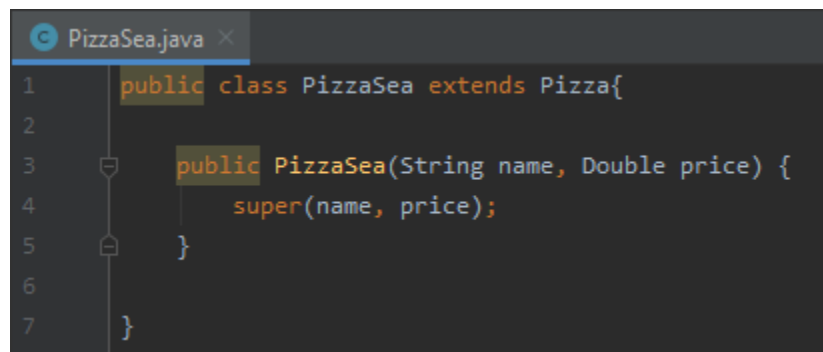


Voici la classe générée :



```
1 public class Pizza {
2     String name;
3     Double price;
4
5     public Pizza(String name, Double price) {
6         this.name = name;
7         this.price = price;
8     }
9
10    public Double getTotal() { return price*0.10 + price; }
13
14    public String getName() { return this.name; }
17
18    @Override
19    public String toString() { return name+" "+price.toString(); }
22 }
```

Et la classe PizzaSea:

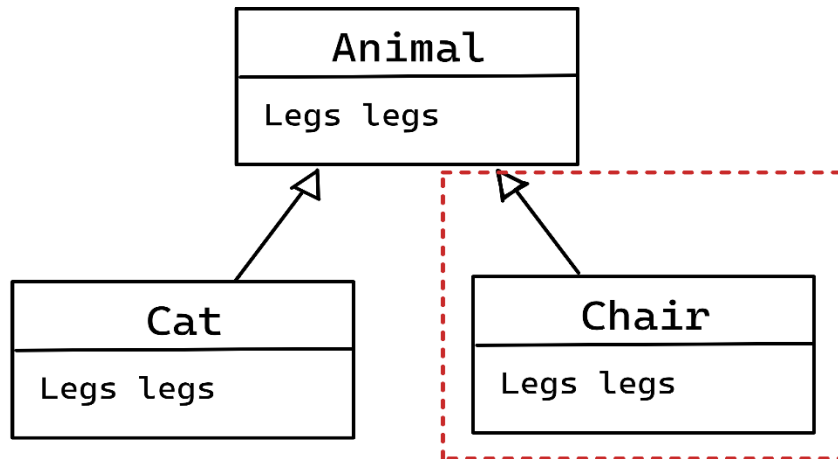


```
1 public class PizzaSea extends Pizza{
2
3     public PizzaSea(String name, Double price) {
4         super(name, price);
5     }
6
7 }
```

- Le refactoring sous IntelliJ est bien réalisée, et le programme de test marche directement sans modification.

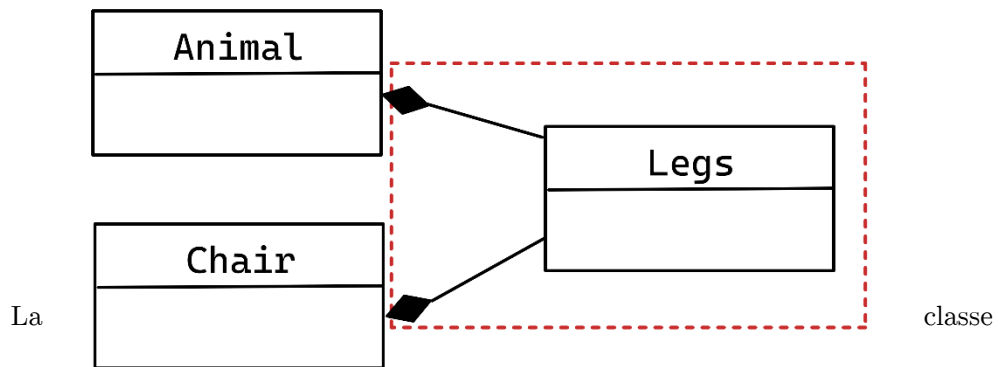
2 Deuxième programme

- Nous avons une sous-classe n'utilise que certaines méthodes et propriétés héritées de ses parents. Les méthodes inutiles peuvent simplement devenir inutilisées ou être redéfinies et générer des exceptions.



La superclasse **Animal** et la sous-classe **Chair** sont complètement différentes.

- Résultat attendu après le refactorign



```

1 public abstract class Animal {
2     String name;
3     String type;
4
5     //nous attendons l'extraction de ces deux propriété a une nouvelle classe
6     int legsNumber;
7     int legSize;
8
9     public Animal(String name, String type, int legsNumber, int legSize) {
10         this.name = name;
11         this.type = type;
12         this.legsNumber = legsNumber;
13         this.legSize = legSize;
14     }
15 }

```

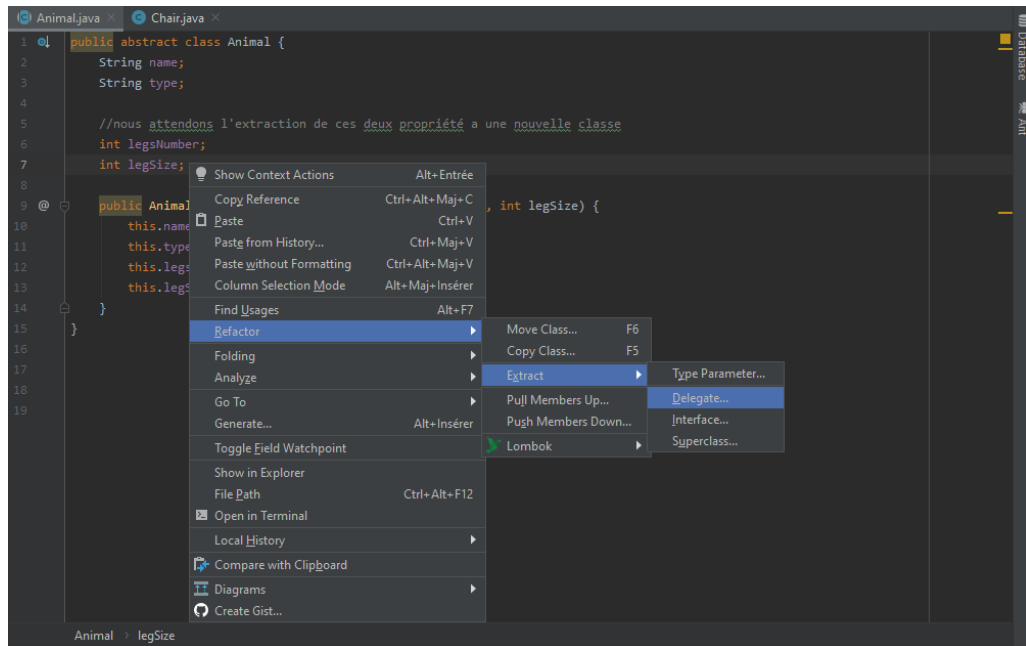
La classe Chair :

```

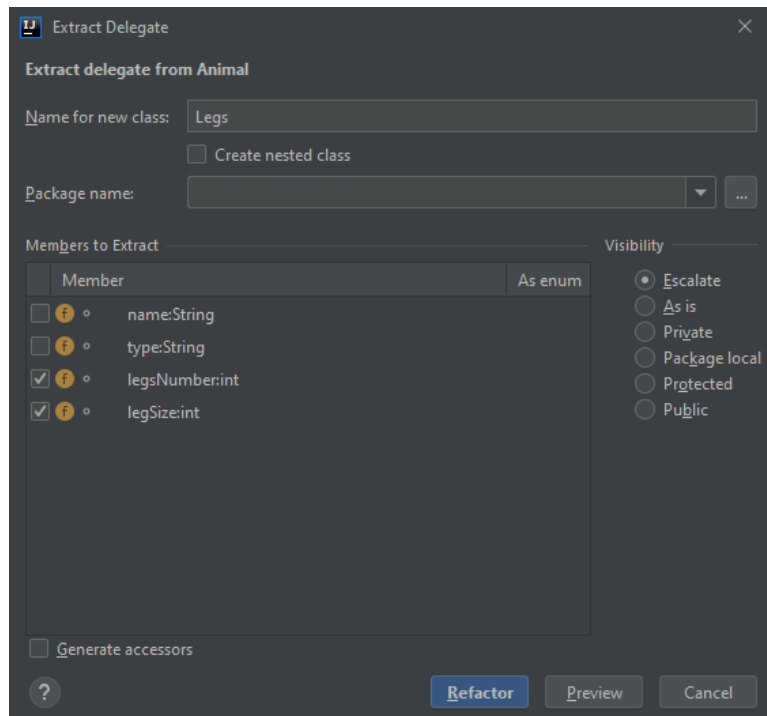
1 public class Chair extends Animal { //nous attendons elimination de l'héritage avec la classeAnimal
2
3
4     public Chair(String name, String type, int legsNumber, int legSize) { super(name, type, legsNumber, legSize); }
5
6
7     @Override
8     public String toString() {
9         return "Chair ( " + name + " " + type + " " + legsNumber + " " + legSize + " )";
10    }
11
12 }

```

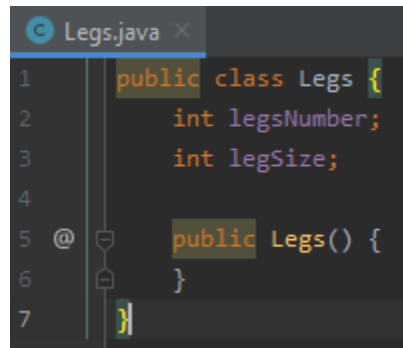
- Nous avons choisi Extract delegate



Configuration de la nouvelle classe :



Voici la classe générée :

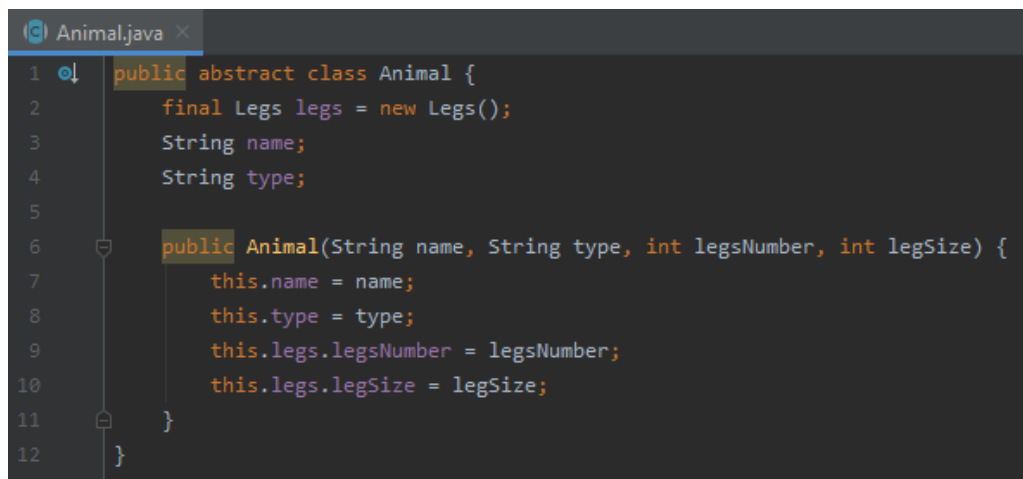


```

1 public class Legs {
2     int legsNumber;
3     int legSize;
4
5     public Legs() {
6     }
7 }

```

Et la classe Animal :

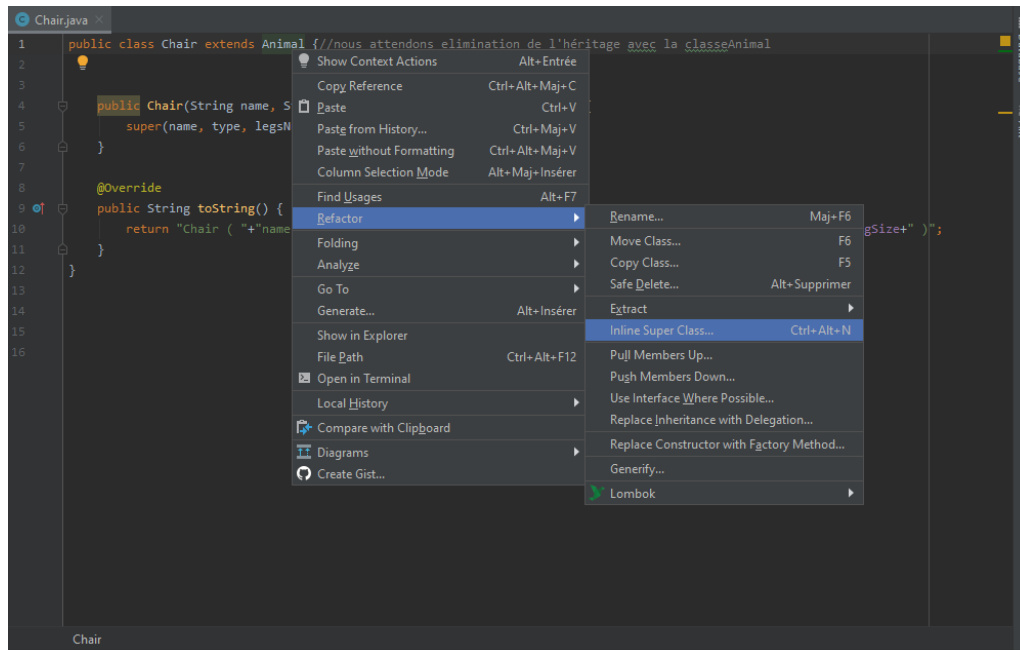


```

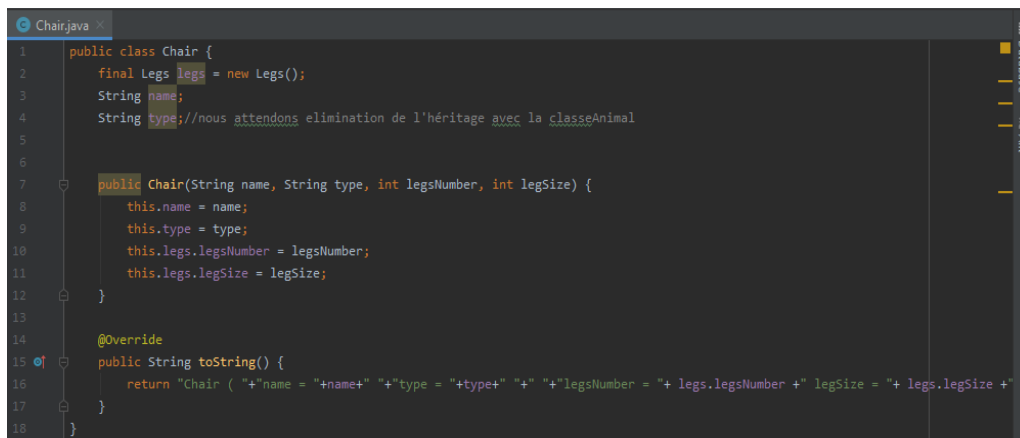
1 public abstract class Animal {
2     final Legs legs = new Legs();
3     String name;
4     String type;
5
6     public Animal(String name, String type, int legsNumber, int legSize) {
7         this.name = name;
8         this.type = type;
9         this.legs.legsNumber = legsNumber;
10        this.legs.legSize = legSize;
11    }
12 }

```

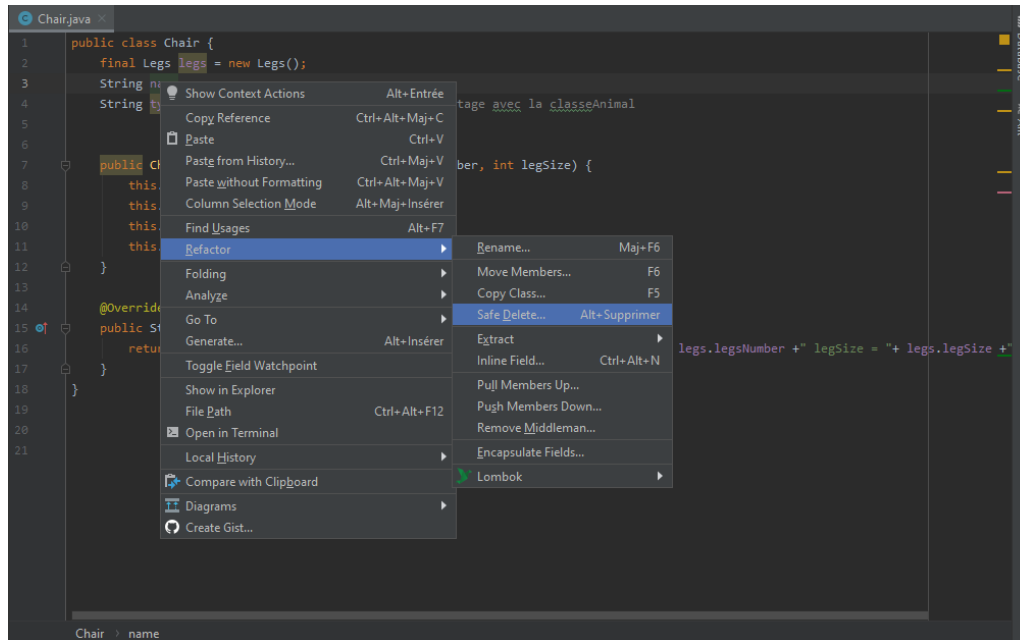
- Après nous avons choisi Inline Super Class pour la classe Chair



La classe Chair après cette opération :



- Après nous avons choisi Safe Delete pour deux propriétés inutiles



Voici la classe Chair après cette opération :



- Le refactoring sous IntelliJ est bien réalisée, et le programme de test marche directement sans modification.

Comparaison de 2 catalogues de refactorings

- L'opération Split Variable se trouve dans le catalogue et pas dans IntelliJ.
Split Variable est de supprimer les affectations aux paramètres.
- L'opération Safe Delete se trouve dans IntelliJ et pas dans le catalogue.
Safe Delete est de supprimer une propriété d'une classe et ces utilisations par tous dans le projet.

Etude de l'opération de refactoring

'Extract Interface'

- Ce n'est pas possible de sélectionner toutes les classes pour faire une extraction d'interface. Il la faire classe par classe.
- Elle n'a pas factorisé les signatures des méthodes communes.
- Nous remarquons que pour chaque ensemble de signatures de méthodes communes, une interface est créée.
- Il est différent avec notre version, nous n'avons pas créer une interface pour la méthode Object get(int i).
- À l'aide de l'AOCposet nous avons pu se rendre compte que notre version devrait utiliser une interface pour la méthode Object get(int i).

