CM 9 : POO

Protected

> ⚠ Warning
>
> Attention, comportement bizarre venant de ce code

```java
public class Mammifere {
    private int nbLegs = 4;
    public int getNbLegs() {
        return this.nbLegs;
    }
}

public class Cetacee extends Mammifere {
    private int nbLegs = 0; // Pas bien

    public Cetacee() {
        super();
        this.nbLegs = 0; // Pas bien non plus
    }
    public void confusing() {
        S.o.p(this.nbLegs); // 0
        S.o.p(this.getNbLegs()); // 4
    }
}
```

En gros : pas de lookup sur les attributs

> 💡 Tip
>
> On ne redéfinit pas les attributs :D

On a un nouveau modificateur (`protected`) qui fonctionne ainsi :

- Comme `public` pour les sous-classes
- Comme `private` pour les autres

Notre situation devient alors :

```java
public class Mammifere {
    protected int nbLegs = 4;
    public int getNbLegs() {
        return this.nbLegs;
    }
}

public class Cetacee extends Mammifere {

    public Cetacee() {
        super();
        this.nbLegs = 0; // Il ne gueule plus
    }
}
```

Super

Super est aussi une référence vers la superclasse.

On a :

- `super == this`
- `super` modifie le mécanisme de lookup

`super` fait commencer dans la super classe de la classe définissant la méthode utilisant `super`, mais il NE fait PAS commencer la recherche de méthode dans la super classe de l'objet

La référence `super` ne doit pas être utilisé à la place de `this`, sauf si nous voulons "contourner" le lookup

Séparer ce qui change de ce qui ne change pas

On a ce genre de choses :

```java
// Dans Parcel
public String toString() {
    return "colis pour ...."
}

// Dans ExpressParcel
public String toString() {
    return "colis express pour  ... "
}


// InsuredParcel
public String toString() {
    return "colis assuré pour"
}
```

Au lieu de redéfinir `toString` à chaque fois, on peut créer une méthode `description`

```java
// Parcel.java

public String toString() {
    return this.description() + " pour  ... ";
}

protected String description() {
    return "colis";
}

// ExpressParcel.java
protected String description() {
    return "colis express";
}

...
```

Cette méthode n'est pas dans le cahier des charges, mais elle est nécessaire à la bonne conception du code

Autre exemple

Imaginons que nous voulons afficher un bordereau de livraison en différent types de colis, dans notre classe `DeliveryTruck` :

```java
public void BadDeliveryForm() {
    for (Parcel p : this.parcels) {
        if (p instanceof Parcel) {
            PRINTER.black(p);
        }
         ...
    }
}
```

Très mauvais, viole le principe ouvert-fermé

Ce n'est pas au `DeliveryTruck` de gérer cela, laissons gérer la mécanique de lookup en reportant la responsabilité directement dans `Parcel`

```java
// Parcel.java

public void print() {
    PRINTER.black(this.toString());
}


// ExpressParcel.java

public void print() {
    PRINTER.red(this.toString());
}


// FragileParcel.java

public void print() {
    PRINTER.orange(this.toString());
}
```

Avec cela, nous avons dans `DeliveryTruck`

```java

public void goodDeliveryForm() {
    for (Parcel p : this.parcels) {
        p.print();
    }
}
```

> Tip
>
> TOUJOURS préférer le report de responsabilité, pour ne pas devoir faire le boulot du lookup

Mort d'un objet

Pas besoin de s'en occuper, le garbage collector s'en occupe comme un grand