

1 Filtres

Supposons que nous ayons la classe suivante :

```
class BadFilter {
    public static final int ODD = 0;
    public static final int LEQ = 1;
    private final int type;
    private int value;

    public BadFilter(int type) { this.type = type; }

    public BadFilter(int type, int value) {
        this.type = type;
        this.value = value;
    }
    public List<Integer> apply(List<Integer> list) {
        List<Integer> result = new ArrayList<>();
        if(type == ODD) {
            for (int i : list) {
                if (i % 2 == 1) result.add(i);
            }
        } else if (type == LEQ) {
            for (int i : list) {
                if (i <= value) result.add(i);
            }
        }
        return result;
    }
}
```

Question 1 : Qu'affiche le code suivant ?

```
List<Integer> list = new ArrayList<Integer>();
for (int i = 0; i < 10; i++) list.add(i);
BadFilter filterOdd = new BadFilter(BadFilter.ODD);
BadFilter filterLeq = new BadFilter(BadFilter.LEQ, 6);
List<Integer> result = filterLeq.apply(filterOdd.apply(list));
for (int i : result){
    System.out.println(i);
}
```

Question 2 : Que doit-on modifier dans le code de la classe `BadFilter` pour ajouter un filtre qui sélectionne

tous les entiers supérieurs à une valeur ? Quel(s) principe(s) SOLID ne sont pas respectés ?

On considère qu'on a à notre disposition l'interface `IntPredicate` suivante :

```
public interface IntPredicate {  
    /**  
     * Evaluates this predicate on the given argument.  
     *  
     * @param value the input argument  
     * @return {@code true} if the input argument matches the predicate,  
     * otherwise {@code false}  
     */  
    boolean test(int value);  
}
```

Question 3 : En supposant que les classes `Odd` et `Leq` implémentent l'interface `IntPredicate`, donnez le diagramme de classes d'une nouvelle organisation du code qui corrige le défaut de conception évoqué à la question 2. Vous devez faire en sorte que le code suivant ait le même comportement que celui donné pour la question 1 :

```
List<Integer> list = new ArrayList<>();  
for (int i = 0; i < 10; i++) list.add(i);  
Filter filterOdd = new Filter(new Odd());  
Filter filterLeq = new Filter(new Leq(6));  
List<Integer> result = filterLeq.apply(filterOdd.apply(list));  
for (int i : result) System.out.println(i);
```

Question 4 : Donnez le code des classes `Filter` et `Leq` du diagramme de la question précédente.

Question 5 : Proposez une implémentation de la classe `And` de sorte que le code suivant ait le même comportement que les codes des questions précédentes tout en ne parcourant qu'une seule fois la liste. Vous ne devez pas modifier les classes précédemment écrites.

```
List<Integer> list = new ArrayList<>();  
for (int i = 0; i < 10; i++) list.add(i);  
Filter filter = new Filter(new And(new Odd(), new Leq(6)));  
List<Integer> result = filter.apply(list);  
for (int i : result) System.out.println(i);
```