

TD7 : Classes abstraites et interfaces

Soit les deux classes suivantes :

```
import java.util.Arrays;

public class ListSum {
    private int[] elements;
    private int size;
    private int increment;

    public ListSum() {
        this(10,10);
    }

    public ListSum(int initialSize, int increment) {
        elements = new int[initialSize];
        this.increment=increment;
        size=0;
    }

    public void add(int value) {
        if (elements.length==size) {
            elements = Arrays.copyOf(elements, elements.length+increment);
        }
        elements[size] = value;
        size++;
    }

    public int eval() {
        int result = 0;
        for (int i = 0; i < size; i++)
            result += elements[i];
        return result;
    }
}
```

```
import java.util.Arrays;

public class ListProduct {
    private int[] elements;
    private int size;
    private int increment;

    public ListProduct() {
        this(10,10);
    }

    public ListProduct(int initialSize, int increment) {
        elements = new int[initialSize];
        this.increment=increment;
        size=0;
    }

    public void add(int value) {
        if (elements.length==size) {
            elements = Arrays.copyOf(elements, elements.length+increment);
        }
    }
}
```

```

    }
    elements[size] = value;
    size++;
}

public int eval() {
    int result = 1;
    for (int i = 0; i < size; i++)
        result *= elements[i];
    return result;
}
}

```

1- Classe abstraite

Ces codes ont de nombreuses ressemblances. Il serait donc judicieux de factoriser un peu cela. Proposez une solution avec une classe abstraite dont hériteront `ListSum` et `ListProduct`. Astuce pour factoriser au maximum : On peut isoler dans des méthodes abstraites l'initialisation du résultat par l'élément neutre de l'opérateur et l'opération en elle même.

2- Interfaces et délégation

Imaginons maintenant que je veuille calculer la somme puis le produit d'une même liste. La solution précédente oblige à instancier deux listes avec les mêmes éléments.

Il pourrait donc être intéressant de séparer le stockage (i.e. la liste) de l'opération.

Pour cela, on peut "déléguer" l'évaluation à une autre classe qui est l'opérateur.

On aura dans ce cas une classe concrète pour représenter la liste. Cette classe aura une méthode `eval` qui prendra en paramètre une instance du type `Operator`. Le type `Operator` est une interface pour laquelle il existe au moins deux implémentations : `SumOperator` et `ProductOperator`.

Ecrivez le code qui correspond à cette solution.