Calculs

Dans cet exercice on s'intéresse à des calculs (sur n'importe quel type de données) qui prennent une entrée, produisent une sortie, et peuvent se décomposer en plusieurs actions. Par exemple :

- le calcul qui à un entier x associe $3x^2$ peut être vu en deux actions : d'abord élever x au carré, puis multiplier par 3
- le calcul qui à une chaîne s associe s + "toto" + s + "toto" peut être vu en deux actions : d'abord concaténer s et "toto", puis re-concaténer le résultat avec lui-même

Le but de cet exercice est de modéliser ce type de calcul. Chaque action prend donc en paramètre un objet d'un certain type, et redonne un objet du même type. On modélise donc une action et un calcul de la façon suivante :

```
public interface Action<T> {
    public T execute(T x);
}
```

```
import java.util.ArrayList;

public class Calcul<T> {
    private ArrayList<Action<T>> liste;

    public Calcul() {
        // crée un calcul sans actions
    }

    public void add(Action<T> action) {
        // ajoute "action" à la fin du calcul
    }

    public T execute(T x) {
        // exécute sur x toutes les actions dans l'ordre et retourne le résultat final
    }
}
```

Q1 Compléter tout le code de la classe `Calcul<T>`.

Q2 On souhaite recréer l'exemple ci-dessus sur les chaînes en utilisant l'interface précédente. Ecrire deux classes `MachineDouble` et `MachineAjouteToto` qui implémentent l'interface `Action` pour les `String` de la façon suivante : `MachineDouble` doit concaténer l'entrée avec elle-même, `MachineAjouteToto` doit ajouter "toto" à la suite de l'entrée.

Q3 Ecrire une méthode principale `main(String args[])` qui crée un objet de type `Calcul` sur des chaînes (composé d'une `MachineAjouteToto`, puis d'une `MachineDouble`), et qui execute ce calcul sur l'entrée "hello".

Q4 On souhaite maintenant que toute machine sur les `String` (`MachineDouble`, `MachineAjouteToto` ou d'autres dans le futur) compte le nombre de fois où elle est utilisée. En respectant les différents principes de programmation, modifiez en conséquence le code précédent en respectant la

suggestion d'organisation suivante (vous devez compléter la suggestion en ajoutant ce qui vous semble utile : attributs, héritage, ..) :

- ajouter une classe `MachineString` qui implémente l'interface Action sur les chaînes
- ajouter à `MachineString` un constructeur sans paramètre (en déduire la spécification) et une méthode `public void compter()` qui sera appelée à chaque fois que la machine est utilisée (en déduire la spécification)
- modifier si besoin les classes `MachineDouble` et `MachineAjouteToto`