TD final

I. Objectifs du TD 1
II. Utilisation des classes 2
II.1 Un manager d'éléments : ElementManager 3
III. Interface, énuméré et généricité 3
III.1 Des résultats 3
IV. Piles et Listes 4
IV.1 Des traces d'expérimentation 4
IV.2 Des actions sur les éléments 5

I. Objectifs du TD

- Réviser les concepts étudiés dans ce module.

Attention, ce TD vous prépare au DS terminal, c'est son objectif principal.
A certaine étape il est possible qu'il vous manque des codes qui vous sont donnés ensuite.

Vous trouverez tous les codes, y compris des solutions possibles sous :

II. Utilisation des classes

Considérez la classe abstraite **Element** suivante :

```
/**
 * Element is the abstract class that represents the elements of the system.
 */
public abstract class Element {
    private String name;

    /**
     * Constructs an element with the given name.
     * @param name the name of the element
     */
    protected Element(String name) {
        this.name = name;
    }

    /**
     * Gets the name of the element.
     * @return the name of the element
     */
    public String getName() {
        return name;
    }

    /**
     * Perform an action on the element.
     * @param actionName the name of the action to perform
     * @return the result of the action
     */
    public abstract Result performAction(String actionName);
}
```

Des implémentations de cette classe ***WaterElement*** et ***FireElement*** ont été définies.

II.1 Un manager d'éléments : ElementManager

Q.1. Créez une classe **ElementManager**, telle que une instance de ElementManager
- a un nom,
- permet de stocker des instances d'**Element**
- et de les retrouver à partir de leur nom – Définissez la méthode **Optional<Element> getElement(String elementName)** pour permettre la récupération d'une instance d'**Element** par son nom, renvoyant **Optional.empty()** si l'élément n'est pas présent dans la collection.
- On peut ajouter un Element à une instance de ElementManager.

Q.2. Créez une instance de la classe **ElementManager** appelée "ForEver" en utilisant les méthodes dont vous avez besoin pour que ces codes soient fonctionnels.
- Ajoutez un **WaterElement** et un **FireElement** à cet ElementManager.
- Utilisez la méthode **getElement(String elementName)** pour vérifier la présence du **WaterElement** dans les éléments associés à « ForEver ». Afficher simplement si l'élément a été trouvé ou non.

III. Interface, énuméré et généricité

III.1 Des résultats

Vous avez dû remarquer que la méthode **performAction** retourne un objet de type **Result**.
Il s'agit d'une interface définie comme suit et qui doit être implémentée par les résultats spécifiques à chaque élément.

```
public interface Result<T, V extends Enum> extends Comparable<Result<T, V>>{
    //Value of the result
    T getValue();
    //Status of the result
    V getStatus();

    /**
     * Returns the code associated to this result.
     * @return the code associated to this result.
     * The code is an integer value that can be used to compare results and to
signal if a result is dangerous.
     */
    int getCode();

    /**
     * The possible values for the code associated to a result explaining this
result is dangerous.
     */
    final static int DANGER = 3;

}
```

Q.3. Implémentez la classe **WaterStatus** en tant qu'énumération pour représenter les différents états possibles de l'eau. Chaque état doit être associé à un code numérique unique et à une description. Les états et leurs codes sont les suivants :

- "Liquid" avec le code 2 : "Liquid : 0°C < T < 100°C"
- "Frozen" avec le code 3 : "Frozen : T ≤ 0°C"
- "Gas" avec le code 1 : "Gas : T > 100°C"

Q.4. Implémentez la classe **WaterResult** qui représente le résultat d'une analyse de l'état de l'eau en fonction de la température. Elle doit implémenter l'interface **Result**. Une instance de WaterResult donne la température (un entier) et l'état de l'eau (WaterStatus).

- Une méthode d'analyse de la température pour déterminer l'état de l'eau (gelé, gazeux, liquide) est également fournie qui analyse à la création d'un WaterResult la température de l'eau et affecte son état.
- **WaterResult** doit bien sûr implémenter les méthodes nécessaires de l'interface **Result**. Lors de la comparaison de deux résultats, la comparaison doit se faire uniquement sur la température.
- Deux résultats sont égaux si la température et l'état sont égaux.
- La représentation d'un WaterResult sous la forme d'une String est la suivante

IV. Piles et Listes

IV.1 Des traces d'expérimentation

Q.5. Créez une classe `ExperimentTrace`

- On stocke dans *ExperimentTrace*, les résultats des expérimentations dans une pile. *La pile préserve l'ordre des résultats obtenus.*
- Une exception personnalisée de type `ExperimentTraceException` est levée si un résultat d'expérimentation à ajouter dans la trace a un code supérieur à Result.*Danger* ou est nul.
- Une méthode permet de récupérer le dernier résultat enregistré (sans modifier la trace)
- Une méthode permet de récupérer tous les résultats enregistrés sous la forme d'une liste dont le premier élément est le dernier résultat obtenu et vide la trace.
- Une méthode permet de retirer de la trace tous les résultats qui sont égaux à un résultat donné.

A terme nous utiliserons cette trace pour gérer le suivi de nos expérimentations.

IV.2 Des actions sur les éléments

On vous donne la classe action suivante

```
public class Action<T,E extends Enum> {

    private String actionName;

    public Action(String actionName) {
       this.actionName = actionName;
    }
    public  Result<T,E> executeOn(Element e){
       return e.performAction(actionName);
    }
```

Q.6. Complétez la classe **`ElementManager`** pour inclure des méthodes d'exécution d'actions sur des éléments.

- **performAction :** Complétez la méthode **`performAction`** qui prend en paramètre un objet **`Element`** et une action **`Action`**. Cette méthode doit exécuter l'action sur l'élément et enregistrer le résultat dans la trace d'expérience (**`ExperimentTrace`**) que vous associez à présent à ElementManager.
- **performActionListOn** : Complétez la méthode performActionListOn qui prend en paramètre le nom d'un élément et une liste d'actions. Cette méthode doit obtenir l'élément correspondant au nom à partir de la map d'éléments, puis itérer sur la liste d'actions et exécuter chaque action sur l'élément. Les résultats doivent être enregistrés dans la trace d'expérience. Si l'élément n'est pas trouvé, lever une exception ElementNotFoundException

Q.7. Complétez la classe **`ElementManager`** pour nettoyer la trace d'expérimentation en éliminant tous les résultats redondants.