JAVA 09 : LES INTERFACES

Présentation - L'objet d'étude

Au cours de ce TP on simulera le fonctionnement individuel d'ampoules électriques de différents modèles, ainsi que le comportement collectif d'ampoules appartenant à un même luminaire.

Une ampoule électrique répond aux stimuli suivants :
- allumer (éclairage pleine puissance)
- eteindre
- intensifier (augmenter la luminosité si elle n'est pas déjà maximale)
- diminuer (diminuer la luminosité si l'ampoule n'est pas déjà éteinte)

De plus elle est capable de fournir son état :
niveau d'intensité (d'éteinte à allumée pleine puissance) ou en panne.

Un luminaire est composé de plusieurs ampoules et répond exactement aux mêmes stimuli qu'une ampoule électrique isolée.
Un luminaire fournit les informations sur leur état selon les mêmes règles que l'ampoule, à l'exception de l'état en panne qui n'est vérifié que si toutes les ampoules qui le composent sont en panne.

Pour simplifier les choses on estime qu'une ampoule électrique ne peut tomber en panne que lorsqu'elle passe de l'état éteint à un état « allumé », peu importe l'intensité.

La probabilité qu'une ampoule tombe en panne est fonction de son type. Voici les probabilités, hautement fantaisistes, de tomber en panne pour chaque type d'ampoule dans le TP :
- 1% pour une ampoule à LED
- 5% pour un tube néon
- 10% pour une ampoule à incandescence

Les tubes néons ont la particularité de ne pas pouvoir changer d'intensité : ils sont soit éteints, soit allumés.

L'interface de commande

Les stimuli auxquels répondent les ampoules et les luminaires sont identiques, de même que l'affichage de l'état. On peut considérer que l'on manipule ces éclairages au travers d'une interface normalisée qui serait un « interrupteur variateur » avec affichage de l'intensité lumineuse.
Peu importe que l'on veuille allumer éteindre ou faire varier d'intensité une ampoule isolée ou un luminaire complet, les actions sont les mêmes, seul le dispositif qui est à l'extrémité change.
Chez vous vous pouvez utiliser une ampoule basse consommation, une ampoule à LED ou une ampoule classique, un luminaire ou une ampoule isolée sans avoir à refaire toute l'installation électrique ou changer les interrupteurs.
Cette application utilise une interface très semblable à ce modèle Céliane avec diodes de chez Legrand :

En Java la définition de la commande de manipulation des ampoules est dans l'interface Eclairage.java qui permet d'allumer, éteindre, faire varier l'intensité et afficher l'état :

```
public interface Eclairage {
     public void allumer();
     public void eteindre();
     public void intensifier();
     public void diminuer();
     public int etat();
}
```

L'application va utiliser cette interface de commande pour envoyer des ordres au dispositif d'éclairage et récupérer son état tout comme vous utiliseriez l'interrupteur Céliane pour piloter un éclairage réel.

Notion d'interface

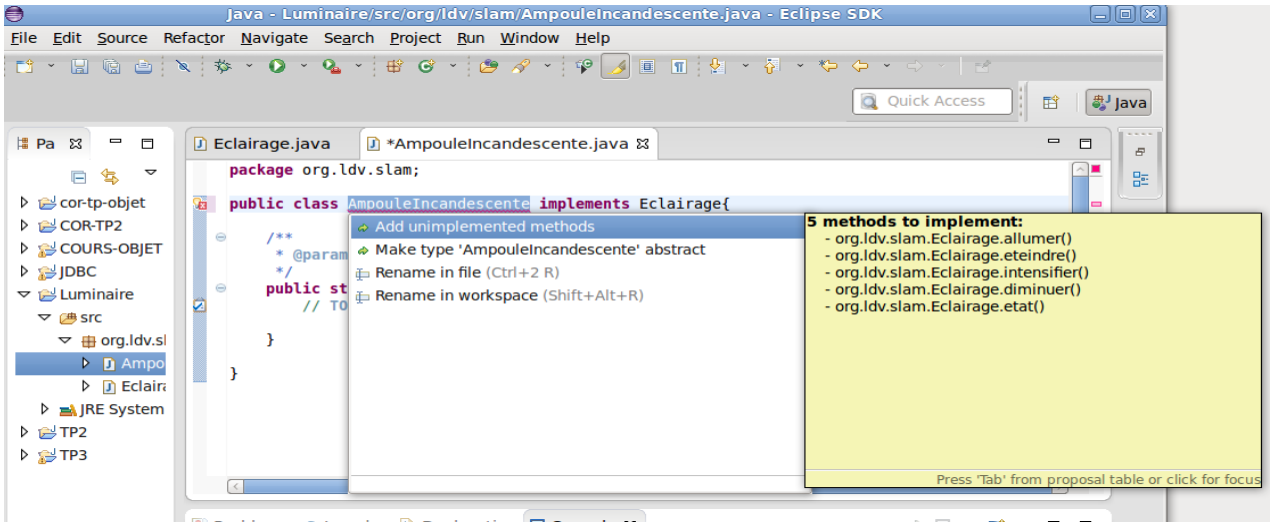
Parfois, vous voudriez que différentes classes aient un comportement similaire. Par exemple, vous voulez que toutes les ampoules aient le même comportement et les mêmes méthodes.
Les interfaces sont un moyen très efficace pour résoudre ce problème.
Une interface est une collection de définitions de méthodes (sans implémentation c'est à dire sans écrire les méthodes) et de valeur constantes. C'est une sorte de contrat avec lequel on va dire que toutes les classes qui vont implémenter l'interface « Eclairage » devront définir les différentes méthodes de cette interface.

Pour que toutes nos ampoules (LED , à incandescence, néon etc) aient un comportement identique, on va définir « un contrat » à travers une interface, dans lequel on indiquera les différentes méthodes de nos ampoules : allumer, éteindre, faire varier l'intensité (intensifier ou diminuer) et afficher l'état .

Pour implémenter une interface, il faut rajouter à la déclaration de la classe concernée le mot clé **implements** suivi du nom de l'interface. Ensuite, il faut redéfinir dans la classe toutes les méthodes déclarées dans l'interface.

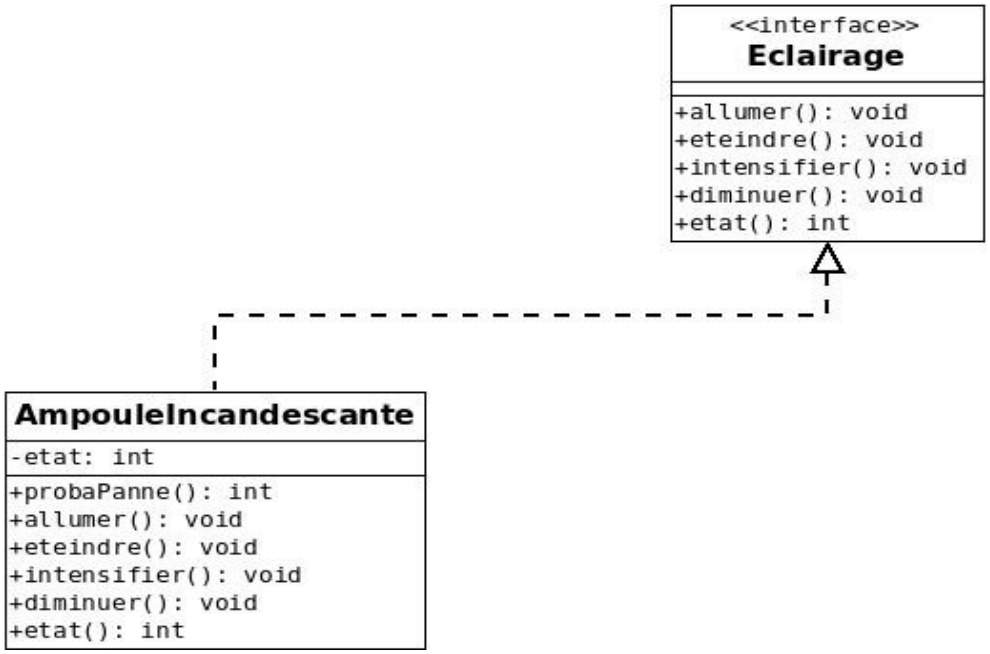
```
public class AmpouleIncandescente implements Eclairage {
  /**
   * etat de la lampe : 0 éteinte, 10 allumée pleine intensité
   */
  private int etat;
```

Lorsqu'une classe implémente une interface, elle doit redéfinir toutes les méthodes de l'interface. Avec Eclipse, vous pouvez :



Notation UML :

Un **trait discontinu** partant de la classe d'implémentation et allant vers l'interface, se terminant par une **flèche fermée**



TRAVAIL À FAIRE :

Le but de l'application est d'allumer 1000 fois un dispositif d'éclairage (ampoule ou luminaire) si il ne tombe pas en panne avant. Après chaque allumage à pleine puissance on fait varier l'intensité du maximum vers le minimum jusqu'à ce que l'éclairage s'éteigne.

Pour manipuler les éclairages, vous disposez de la classe App.java qui permet d'exécuter un cycle de 1000 allumages puis diminution d'intensité sur un objet qui implémente l'interface Eclairage.

Les ampoules et les luminaires ont une méthode toString() déclarée ainsi :

```
/**
    * @return l'état de l'objet sous la forme d'une chaîne de caractères
    */
   public String toString()
```

Cette méthode doit retourner de façon claire le nom de la classe et l'état de l'ampoule ou du luminaire :

- éteinte,
- allumée, avec l'intensité
- en panne.

 un attribut *etat* de type numérique [0..10] simule l'état d'intensité de l'ampoule : 0=éteinte, 10=pleine puissance et -1=en panne

1 – Créer un nouveau projet java : Luminaire.

2 – Dans votre projet Luminaire, créer l'interface Eclairage

```
public interface Eclairage {
     public void allumer();
     public void eteindre();
     public void intensifier();
     public void diminuer();
     public int etat();
}
```

3 – Importer dans votre projet les programmes :

- App.java
- AmpouleIncandescente.java

4 – Pour que l'application soit fonctionnelle, recherchez dans le programme AmpouleIncandescente les commentaires TODO (à faire) et complétez à l'aide de l'annexe les différentes méthodes.

Remarque : avant d'effectuer une opération sur une ampoule vous devez vérifier son état et éventuellement sa probabilité de panne. Par exemple on ne peut pas augmenter l'intensité d'une ampoule éteinte ou dont l'intensité est déjà au maximum, ou allumer une ampoule déjà allumée.

Comment indiquer dans la méthode allumer() le fait qu'une ampoule s'allume si elle n'est pas déjà allumée et si sa probabilité de panne ne retourne pas -1 ?

5 – Sur le modèle de la classe AmpouleIncandescente, créer 2 nouvelles classes :

- AmpouleLed.java
- AmpouleTube.java

La différence entre ces ampoules réside dans la probabilité fantaisiste de panne qui est de :

- 1% pour une ampoule à LED
- 5% pour un tube néon
- 10% pour une ampoule à incandescence

Les tubes néons ont la particularité de ne pas pouvoir changer d'intensité : ils sont soit éteints, soit allumés.

Comment allez-vous traitez le fait que les tubes néons ne supportent pas de variation d'intensité ?

6 – Un luminaire répond exactement à la même interface de commande qu'une ampoule, mais il est composé de 4 ampoules de même type qui s'allument et s'éteignent en même temps.
Vous devez créer une classe Luminaire qui sera pilotée par App.

- la classe Luminaire doit-elle implémenter l'interface Eclairage ? Pourquoi ?

```
  private AmpouleIncandescente ampoule1;
  private AmpouleIncandescente ampoule2;
  private AmpouleIncandescente ampoule3;
  private AmpouleIncandescente ampoule4;

  /**
   * Constructeur
   * crée un Luminaire avec 4 ampoules
   */
 public Luminaire() {
     ampoule1 = new AmpouleIncandescente();
     ampoule2 = new AmpouleIncandescente();
     ampoule3 = new AmpouleIncandescente();
     ampoule4 = new AmpouleIncandescente();

  }
```

- implémentez les méthodes qui permettent de gérer le Luminaire.
  Remarque : un luminaire est composé de 4 ampoules, donc lorsque vous allumez le luminaire vous allumez les 4 ampoules, idem pour les autres méthodes.

- Testez des luminaires avec 3 types de lampes.

ANNEXE :

Calculer un pourcentage de chance.

Problème

Il y a 5% de chance (ou de malchance) qu'une voiture du modèle X tombe en panne au après 40000 km. Comment traduire cette probabilité en Java ?

Solution

La classe java.util.Random permet de créer des objets qui fournissent des nombres aléatoires de tout type. La méthode `public int nextInt(int borneSup)` d'un objet de type Random permet d'obtenir une valeur entre 0 et boneSup – 1. On peut estimer que 5% revient à tirer une valeur comprise entre 0 et 4 si borneSup est égal à 100. L'extrait de code suivant correspond à cette solution :

```
Random alea = new Random();
if (km > 40000 && alea.nextInt(100) < 5) {
   // action si 5% en panne panne
} else {
   // action autrement ça roule
}
```

Récupérer le nom d'une classe sous forme de String

Problème

Comment faire dans une application pour afficher le nom de la classe d'un objet sans avoir à le saisir en dur dans le code ?

Solution

En Java tout objet hérite implicitement de la méthode public Class getClass() qui permet de travailler sur ses méthodes et attributs. La méthode public String getName() de Class retourne une chaîne contenant le nom de la classe :

```
String nom = this.getClass().getName();
```