Feux tricolores

Sujet

Un objet métier « **FeuTricolore »** possède 2 attributs. Le premier indique si le feu est à l'arrêt ou en marche. Lorsque le feu est en marche, le second indique la couleur du feu (vert, orange ou rouge). Si le feu est à l'arrêt, la sémantique du second est indéfinie. Les opérations « arreterDemarrer() » et « changer() » modifient circulairement les valeurs de ces deux attributs.

NB : Lors de sa mise (ou remise) en marche, le feu est obligatoirement rouge.

A cette classe métier sont associées deux interfaces *(au sens IHM, pas Java)* :

- L'une graphique qui représente le feu avec l'une de ses trois couleurs : vert, orange, rouge . Quand le feu est à l'arrêt la couleur affichée est le gris.
- L'autre qui est un simple panneau sur lequel figure un texte parmi : « feu arrêté», « passez », « attention » ou « stop »

Les deux interfaces possèdent chacune 2 boutons :

- l'un permettant de changer de couleur dans l'ordre habituel ;
- L'autre permettant d'arrêter ou de remettre en service le feu.

Un clic sur l'un quelconque de ces boutons modifie l'état de l'objet métier. Les 2 interfaces doivent toujours être cohérentes avec l'objet métier.

Contraintes

- Les deux interfaces (graphique et texte) sont indépendantes et ne communiquent pas entre elles
- Le Feu est indépendant des interfaces.

1ère Version

Un seul objet métier "Feu", une interface graphique et une interface texte

2e Version

Toujours un seul objet métier "Feu" ; on ajoute une interface de menu permettant d'ajouter :

- une nouvelle interface graphique,
- une nouvelle interface texte

Les interfaces doivent toujours être cohérentes avec l' état du « Feu »

3e Version

On modifie un petit peu les contraintes initiales : les vues sont des `JinternalFrame` avec un titre et rien d'autre (non `resizable`, etc.), incluses dans un `JDesktopPane` qui est le `ContentPane` de votre fenêtre principale. Bref, vous devez passer par votre menu pour fermer une vue puisqu'elle n'a pas les petites icônes qui vont bien. Conséquence : vous mettez en place un gestionnaire de vues qui est évidemment un **singleton**. Les vues affichent dans leur barre de titre quelque chose qui les identifie de manière unique.

On ajoute deux fonctionnalités.

- Un feu doit pouvoir adapter son cycle à la localisation : Strasbourg ou Kehl. Pour cela, on le munit d'un nouvel attribut codant la « stratégie » à utiliser (patron **Stratégie**). Conséquemment, vous ajoutez un item de menu pour changer de rive votre feu.
- Une IHM peut être décorée dynamiquement (via le menu) par un (ou plusieurs) affichage(s) **piéton** et/ou **tourne-à-droite** :
  - un affichage piéton est vert quand le feu maître est rouge ; sinon, il est rouge;
  - un tourne-à droite est orange clignotant (`javax.swing.Timer`) quand le feu maître est rouge ; sinon, il est éteint.

    Utiliser le patron **Décorateur** pour créer des interfaces décorées. Le décorateur remplace la vue décorée dans la liste du gestionnaire de vues et la suppression du décorateur supprime récursivement toutes les vues qu'il décore. Toutes les décorations associées à une même vue de base affichent dans leur barre de titre la même information (une identification sans ambiguïté de la vue de base).

    Par mesure de sécurité, on ne fera fonctionner les feux munis d'un affichage piéton qu'avec des cycles à quatre temps.

4e Version (facultatif)

L'interface de menu permet maintenant d'ajouter/supprimer d'autres feux tricolores indépendants les uns des autres (nouvelles occurrences de l'objet métier « Feu ») et sur chaque nouvel objet, d'ajouter les 2 types d'interfaces. Une action sur une interface ne fait changer d'état que le feu auquel cette interface est attachée. On utilisera le patron Singleton pour le gestionnaire de Feux.