

Proxy, State et Logger

Exercice 1 - Proxy Pattern

Voici quelques classes permettant de gérer un album de photo `td4photo.zip` (`td4photo.zip`) . On cherche pour déboguer à afficher un message lorsque l'on entre ou sort d'une méthode particulière d'un objet.

- 1 Créer un objet `java.util.logging.Logger` et afficher un message avec le niveau (Level) `ALL`.
Afficher en utilisant la méthode `log` un message de niveau `WARNING`
- 2 Rappeler le principe du design pattern `proxy`
Écrire la méthode `createPhotoProxy(Photo photo)` dans la classe `PhotoFactory` qui prend un objet de type `Photo` et un logger et qui renvoie un objet de type `Photo`.
Si on appelle une des méthodes de cet objet proxy, celui-ci affichera sur le logger lors de l'entrée dans une méthode le message "enter method" suivi du nom de la méthode et lors de la sortie de la méthode le message "exit method" suivi du nom de la méthode.
Pour tester le proxy, changer le code de la méthode `createPhoto(File file)` pour que si l'on crée la factory avec un logger on utilise le proxy.
- 3 On souhaite écrire un proxy générique qui affiche les messages d'entrée et de sortie quelque soit l'objet.
Pour cela nous allons utiliser la classe `java.lang.reflect.Proxy`.
Écrire une méthode `createProxy()` générique et correctement typé créant un proxy générique.
- 4 Modifier votre code pour que ne soit affiché uniquement les méthodes ayant l'annotation `Log`.

Exercice 2 - Prototype Pattern

Nous allons maintenant écrire un générateur de formes en utilisant les classes `td4vignette.zip` (`td4vignette.zip`) .

Les classes du paquetage `fr.uml.v.g1.td4.vignette.factory` sont modifiables alors que celles de `fr.uml.v.g1.td4.vignette` ne le sont pas.

- 1 Implanter la méthode `create()` de la classe `PrototypeVignetteFactory` en utilisant le design-pattern prototype.
- 2 Écrire un main permettant de créer douze carrés rouges sur fond vert avec une épaisseur de trait de 3 pixels et de les stocker dans douze fichiers PNG différents. Puis ajouter douze carrés roses sur fond vert avec une épaisseur de trait de 3 pixels.
Regarder la classe `java.awt.geom.Rectangle2D.Double`.
- 3 On souhaite pouvoir modifier des propriétés de la forme lors de la création. Pour cela implanter la méthode `create(Map...)` de la classe `PrototypeVignetteFactory`.
Pour cela, écrire le type `Properties` de façon à ce que le code écrit n'utilise ni `switch` ni `if`.

Exercice 3 - State Pattern

Nous allons maintenant ré-écrire notre propre logger. Télécharger le squelette de la classe `Logger` [td4log.zip](#) (`td4log.zip`).

1 Compléter la classe `Logger` en implantant les méthodes `warning()`, `info()` et `debug()`.

N'implantez pas la méthode `log` pour l'instant.

2 On souhaite maintenant ré-écrire les méthodes `warning()`, `info()` et `debug()` sans effectuer de `if` mais en déléguant au level le soin d'afficher ou non le message en utilisant le design pattern `state`.