# 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 débogguer 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 appel 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 créé 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 - State Pattern

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

1 Completer la classe `Logger` en implantant les méthodes `info()`, `warning()` et `debug()`.
N'implantez pas la méthode `log` pour l'instant.

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

3 On souhaite enrichir les messages pour que ne soit affichée pas uniquement le message mais aussi la méthode appelante, la classe de celle-ci, le fichier dans lequel se trouve la méthode ansi que sont numéro de ligne.
Exemple

```
fr.umlv.gl.td4.log.Main.main: message d'info
  at Main.java:12
```

Pour faire cela, on utilise une astuce. On crée une exception et l'on récupère la bonne ligne dans la pile d'appel (stack trace) (cf `Throwable.getStackTrace()`).

4 Enfin, implanter la méthode `Logger.log` toujours en utilisant le design pattern `state`. Attention, il y a un petit piège avec le `stack trace`.