Decorator

The decorator pattern facilitates the augmentation of object behavior by combining objects into a flexible and extensible structure. This pattern is commonly used to add or alter functionalities of individual objects dynamically. For example, the decorator pattern can enhance the behavior of graphical components, text processing, or input/output operations. It allows clients to treat both individual objects and decorated objects uniformly, promoting code flexibility and reusability.

The key participants in this pattern include:

The component: Declares a common interface for all concrete components. Defines default behavior for all classes.

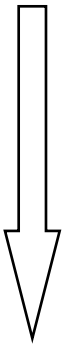
Concrete Component: Represents the original object that can be decorated.

Decorator: Serves as the base class for all decorators. Implements the common interface and maintains a reference to a component.

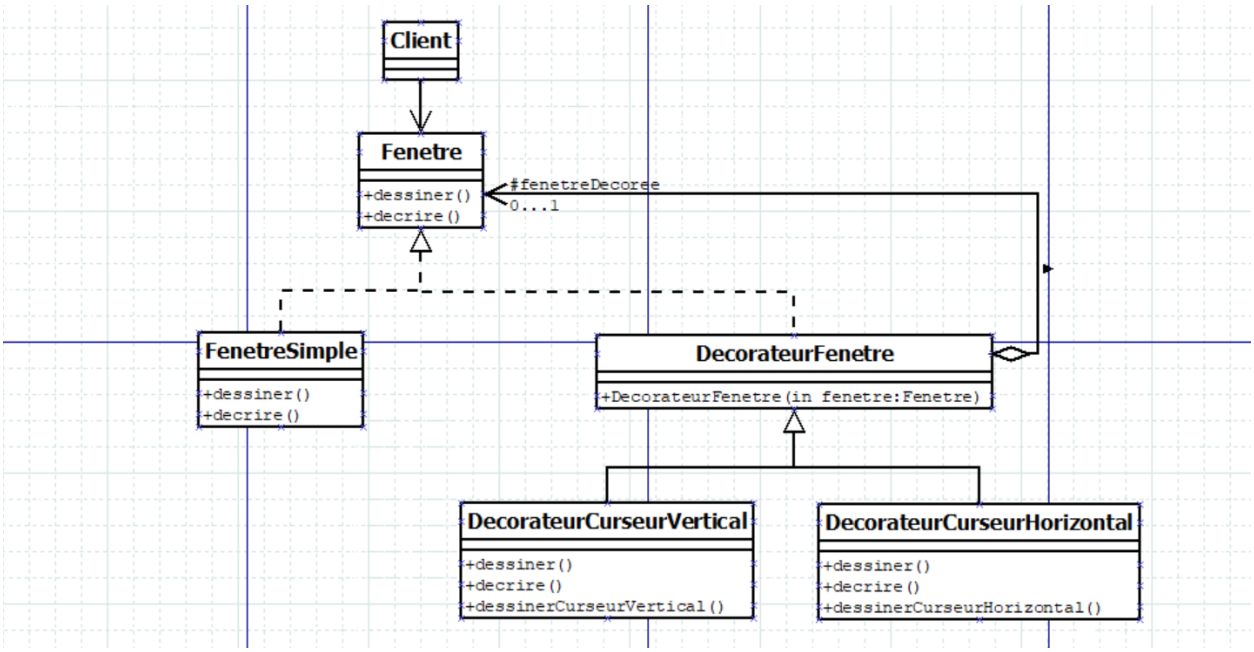
Concrete Decorator: Adds specific functionalities to the components. It extends the functionality of the original component.

The client: Interacts with the components and decorators. If the receiver is a basic component, the request is directly handled. Otherwise, the decorator forwards it to its decorated component.

Below is the implementation using what was given in the pdf which we treated in class. It resolves the exercise which was given in class.



Model


Client
Fenetre
+dessiner()
+decrire()
#fenetreDecoree
0...1
FenetreSimple
+dessiner()
+decrire()
DecorateurFenetre
+DecorateurFenetre(in fenetre:Fenetre)
DecorateurCurseurVertical
+dessiner()
+decrire()
+dessinerCurseurVertical()
DecorateurCurseurHorizontal
+dessiner()
+decrire()
+dessinerCurseurHorizontal()