Learning Assignment

In the lecture, you have just met the decorator pattern. In this learning assignment, you will compare this new pattern with other design patterns and OO mechanisms. Try to find answers to the following questions. Consult your script and books while working on these questions.

Decorator versus Inheritance

The decorator pattern is often praised as an alternative to inheritance. Compare the decorator pattern with inheritance and create a list of advantages and disadvantages of both approaches.

If you have no ideas, then make the following considerations:

- You have the task to program soccer players for a game. The base class is `SoccerPlayer`, which has the usual characteristics of a soccer player (can dribble, can juggle a ball on the head, overturns itself and cries miserably if an outstretched leg appears within a distance of 1 meter).
  Now you have to model different players: forward, back, left and right outer, goalkeeper. They all are, of course, soccer players (inheritance), but what happens, when a player needs to change its position (e.g. from back to forward)?

Decorator versus State

How an object can dynamically change its behavior, this is something we already met when we discussed the state pattern. What is the difference between "changing the behavior" in the two patterns State and Decorator?

Once again, a thought-provoking impulse to activate your brain:

a) The draw tools in *JDraw* are modeled with the state pattern. The active state is changed by clicking on the corresponding button in the tool bar. You could also say that by changing the tool the behavior of the draw view (or, to be more precise, the behavior of the mouse) is changed. Could the draw tools also be implemented as decorators?

b) With a border decorator, you can draw a frame around any figure. How would the solution look like if the State pattern were used?

Try to derive from these two examples the difference between the State and the Decorator pattern.
Also, consider sense or nonsense of such alternative implementations.