Lernaufgaben

Sie haben das Decorator-Pattern kennengelernt. Hier geht es nun darum, das gelernte in Zusammenhang mit anderen Pattern und OO-Mechanismen zu bringen. Erarbeiten Sie Antworten auf folgende Fragen. Nehmen Sie dazu Ihre Kursunterlagen sowie auch Bücher zur Hilfe.

Decorator versus Inheritance

Das Decorator-Pattern wird oft als Alternative zur Vererbung angepriesen. Erstellen Sie eine Tabelle mit Vor- und Nachteilen des Decorator-Patterns und der Vererbung.

Wenn Ihnen nicht spontan etwas dazu einfällt, dann machen Sie doch mal folgende Überlegungen:

- Sie hätten die Aufgabe Fussballspieler für ein Game zu Programmieren. Die Basisklasse sei `SoccerPlayer`, welche die üblichen Eigenschaften aufweist (kann trippeln, kann Ball auf Kopf jonglieren, überschlägt sich und weint kläglich, wenn ein ausgestrecktes Bein < 1m Abstand vom Player vorhanden ist).
  Nun sollen die unterschiedlichen Spieler modelliert werden: Stürmer, Verteidiger, Links- und Rechts-Aussen, Torwart. Sie alle sind natürlich auch Soccer-Players (Vererbung), was geschieht aber, wenn ein Spieler einmal die Position wechseln möchte (Sturm statt Verteidigung)?

Decorator versus State

Wie ein Objekt dynamisch das Verhalten ändern kann, das haben wir doch schon beim State-Pattern kennengelernt. Worin unterscheidet sich „das Verhalten ändern“ in den beiden Pattern State und Decorator?

Auch hier wieder ein Denkanstoss an welchem Sie die Aufgabe erarbeiten können:

a) Die DrawTools in JDraw sind als State-Pattern implementiert. Das Umschalten der Zustände erfolgt durch den Benutzer indem er auf einen Tool-Button klickt. Dadurch wird das Verhalten der DrawView verändert (exakter gesagt, das Mausverhalten ändert sich). Könnten die DrawTools auch als Decorator implementiert werden?

b) Mit Decorators haben Sie einen Rahmen um beliebige Objekte zeichnen können. Wie sähe Ihre Lösung mit einem State-Pattern aus?

Versuchen Sie aus diesen beiden Beispielen den Unterschied State vs. Decorator abzuleiten.
Überlegen Sie sich auch Sinn bzw. Unsinn solcher alternativen Implementierungen.