0.1 Decorator

- Problem
  - Objekt(nicht eine ganze Klasse) soll mit zusätzlichen Verantwortlichkeiten versehen werden.
- Lösung
  - Decorator, der dieselbe Schnittstelle hat wir das ursprüngliche Objekt, wird vor dieses geschaltet. Decorator kann nun jeden Methodenaufruf entweder
    1. selber bearbeiten,
    2. ihn an das ursprüngliche Objekt weiterleiten oder
    3. eine Mischung aus beidem machen.
- Hinweise
  - Strukturell identisch mit dem Proxy Design Pattern. Hat aber andere Absicht.
  - Identisch mit Composite Design Pattern wenn Anzahl Elemente 1 ist. Hat aber andere Absicht.

0.2 Observer

- **Problem**
  - Objekt soll ein anderes Objekt benachrichtigen, ohne Typ des Empfängers zu kennen.
- **Lösung**
  - Interface definieren, dient nur dazu, Objekt über eine Änderung zu informieren. Interface vom Observer implementiert. Observable Objekt benachrichtigt ale registrierten Observer über eine Änderung.
- **Hinweise**
  - (Observer-Observable) = (Publisher-Subscriber)= (Listener-Observable)
  - Observable kennt nur Observer aber nicht wahren Typ
  - 2 Phasen:
    1. Registrierung vom Observer
    2. Benachrichtigung vom Observable
- **Erweiterung**
  - Mithilfe Mediatior Pattern kann ein Objekt zwischen Observer und Observable vermitteln. Sowohl Objekt wie auch Observable registrieren sich bei diesem Objekt. Beide benennen eine Eventquelle, die Observer abonniert und Observable mit Events bedient.
  - Beispiel:
    - IoT mit Netzwerkprotokoll MQTT- Server/Client verwendet dieses Prinzip

0.3 Strategy

- **Problem**
  - Algorithmus soll einfach austauschbar sein
- **Lösung**
  - Algorithmus in eine eigene Klasse verschieben, die nur eine Methode mit diesem Algorithmus hat.
  - Interface für diese Klasse definieren, das von alternativen Algorithmen implementiert werden muss.

- **Hinweise**
  - Motivation: technische oder fachspezifische Gründe zur Austauschung
  - Interface:
    * Nur eine Methode
    * Parameter: Alle Daten übergeben welche Algorithmus benötigt. Parameter heisst Context.

0.4 Composite

- **Problem**
  - Menge von Objekten haben dasselbe Interface und müssen für viele Verantwortlichkeiten als Gesamtheit betrachtet werden.
- **Lösung**
  - Composite definieren, das dasselbe Interface implementiert und Methoden an die darin enthaltenen objekte weiterleitet.
- **Hinweise**
  - Hierarchische Struktur of vom Fachgebiet gegeben
  - Nicht alle Methoden delegieren einfach auf enthaltenen Elemente. vor- und Nachbearbeitung ist üblich. Gewisse Methoden müssen ganz anders implementiert werden.

0.5 State

- **Problem**
  - Verhalten eines Objekts ist abhängig von seinem inneren Zustand
- **Lösung**
  - Objekt hat ein darin enthaltenes Zustandsobjekt
  - Alle Methoden, deren Verhalten vom Zustand abhängig sind, über Zustandsobjekt geführt.
- **Hinweise**
  - Zustands-Klassen implementieren Zustand-Interface
  - Zustands-Objekte sind nichts anderes als Strategy Objekte und können Singletons sein.
  - Zustandsobjekt hat entweder direkt den Code (als innere Klasse) oder delegiert an eine Methode des Objekts weiter.
- **Allgemein**
  - In Geschäftsanwendungen State Pattern selten. Häufig in technischen Anwendungen wie Protokollhandler oder Maschinensteuerungen.
- Forum(Inklusive Code)
  - Verschiedene Stufe der Änderungsmöglichkeiten

0.6 Visitor

- **Problem**
  - Klassenhierarchie um Verantwortlichkeiten (weniger wichtige) erweitert werden, ohne dass viele neue Methoden hinzukommen
- **Lösung**
  - Klassenhierarchie mit einer Visitor-Infrastruktur erweitern
  - Alle weiteren neuen Verantwortlichkeiten mit spezigischen Visitor-Klassen realisiert
- **Hinweise**
  - Widerspruch zum Information Expert. Darum wiichtige Methoden weiterhin direkt der Klasse hinzufügen
  - Oft Auswertungen an Visitor-Klassen delegieren
  - Frage bei mehrstufigen Objekthierarchie. wer die darin enthaltenen Elemente aufruft.

0.7 Facade

- **Problem**
  - Einsatz kompliziertes Subsystem mit vielen Klassen. Wie Verwendung vereinfachen, so dass alle Team-Mitglieder es korrekt und einfach verwenden?
- **Lösung**
  - Facade Klasse diefinieren,die vereinfachte Schnittstelle zum Subsystem anbietet und die meisten Anwendungen abdeckt.
- **Hinweise**
  - Facade vs Adapter: Fassade kapselt Subsystem nicht vollständig. Erlaubt dass Methoden der Facade Parameter und Rückgabewerte haben, die Bezug auf das Subsystem nehmen.
  - Oft vom Ersteller eines Frameworks entwickelt