0.1 Liste der Design Patterns für die Lerneinheit

- Adapter
- Simple Factory
- Singleton
- Dependency Injection
- Proxy
- Chain of Responsibility

0.1.1 Adapter

- Problem
  - Einsatz einer Klasse ist inkompatibel mit bereits definierten domänenspezifischen Interface
- Lösung
  - Eigene Adapter Klasse dazwischen schalten
- Hinweise
  - Oft so ein externer Dienst in eigene Anwendung integriert, insbesondere wenn Dienst austauschbar sein soll
  - Target Interface bewusst für Domänenlogik optimiert, während Adaptee von extern bezogen wird.
  - Falls Adaptee einen externen Dienst, dann im Adapter allenfalls Kommunikation integrieren.

0.1.2 Simple Factory

- Problem
  - Das Erzeugen eines neuen Objekts ist aufwändig
- Lösung
- Eine eigene Klasse für das Erzeugen eines neuen Objekts wird geschrieben
- Hinweise
  - Erzeugung oft abhängig von Konfiguration
  - Auch möglich create() mit Parametern zu ergänzen
  - Delegation Implementieren von Interfaces z.B
  - Factory kann allenfalls erzeugten Objekte zwischenspeichern und wiederverwenden
- Alternativen
  - GoF Abstract Factory: Erzeugung einer Familie von verwandten Objekte
  - GoF Factory Method: Basisklasse abstrakte Methode definiert, Objekt eines bestimmten Interfaces zu erzeugen. In abgeleiteten Klassen Methode überschrieben und erzeugt gewünschtes Objekt.

0.1.3 Singleton

- Problem
  - Benötigt von einer Klasse eine einzige Instanz
  - Instanz global sichtibar sein
- Lösung
  - Klasse mit statischen Methode, liefert immer dasselbe Objekt zurück
  - Statische Methode public deklarieren
- Hinweise
  - Globale Sichtbarkeit kritisch
  - Lazy Creation für Instanz möglich, dann aber getInstance() synchronisiert werden.
- Allgeimen
  - Singletons dann wichtig, wenn einen zentralen Ort braucht um Ressourcen zu verwalten
  - Speicherplatz wird gespart mit nur einem Objekt
  - Java Enum Instanzen sind ebenfalls Singletons
  - Globale Sichtbarkeit eher problematisch

0.1.4 Dependency Injection

- Problem
  - Klasse braucht Referenz auf ein anderes Objekt. Dieses Objekt muss ein bestimmtes Interface definieren, je nach Konfiguration mit einer anderen Funktionalität.
- Lösung
  - Anstelle Klasse abhängige Objekt selber erzeugt, Objekt von aussen (Injector) gesetzt
- Hinweise
  - Ersatz für Factory Pattern
  - Direkter Widerspruch zu GRASP Creator Prinzip
  - Unterstüzt von vielen Frameworks kann auch ohne angewendet werden
  - Erleichtert schreiben von Testfällen insbesondere Gebrauch von Mocks
- Varianten
  - DI über setter Methoden
  - Service bei Constructor vom Client übergeben werden
  - DI initialisiert ganze Anwendung
- Frameworks verwenden Annotationen um anzuzeigen welche Attribute über DI gesetzt werden sollen.

0.1.5 Proxy

- Problem
  - Objekt ist nicht oder noch nicht im selben Adressraum verfügbar
- Lösung
  - Stellvertreter Objekt mit demselben Interface anstelle des richtigen Objekts verwendet
  - Proxy Objekt leitet alle Methodenaufrufe zum richtigen Objekt weiter
- Einsatz als (Strucktur ist dieselbe!)
  - Remote Proxy = Proxy für Objekt in einem anderen Adressraum und übernimmt Kommunikation mit diesem
  - Virtual Proxy = verzögert Erzeugen des richtigen Objekts auf das 1. Mal, dass dieses benutzt wird
  - Protection Proxy = Kontrolliert Zugriff auf das richtige Objekt
- Hinweise
  - Unterschied zu Adapter: Ädapteeïn diesem Fall dasselbe Interface implementiert wie Ädapter"
  - Vom Aufbau identisch mit Decorator Pattern hat aber einen anderen Zweck
  - Persistenzframeworks verwenden Virtual Proxy Objekte, um das Erzeugen von Objekten und damit das Herunterladen der Daten zu verzögern, bis die Daten wirklich gebraucht werden.

0.1.6 Chain of Responsibility

- Problem
  - Für Anfrage potentiell mehrere handler, aber richtigen Handler im Vornherein nicht möglich herauszufinden
- Lösung
  - Handler in einer einfach verketteten Liste hintereinandergeschachtelt.
  - Jeder Handler entscheidet, ob der die Anfrage selber beantworten möchte oder sie an den nächsten Handler weiterleitet.
- Hinweise
  - Variante davon:Jeder Handler leitet Anfrage an den nächsten, unabhängig dabon, ob er sie selber behandelt oder nicht
  - Könnte auch sein, dass gar kein Handler Anfrage behandelt
- Allgemein
  - Bei allen hierarchischen Stufen: Wenn Referenz auf Parent Node vorhanden, Anfrage zum Parent weiterleiten wenn dies Node selber nicht beabeiten kann.