Technische Hochschule Ingolstadt

Probeklausur Architektur und Entwurfsmuster Lösung

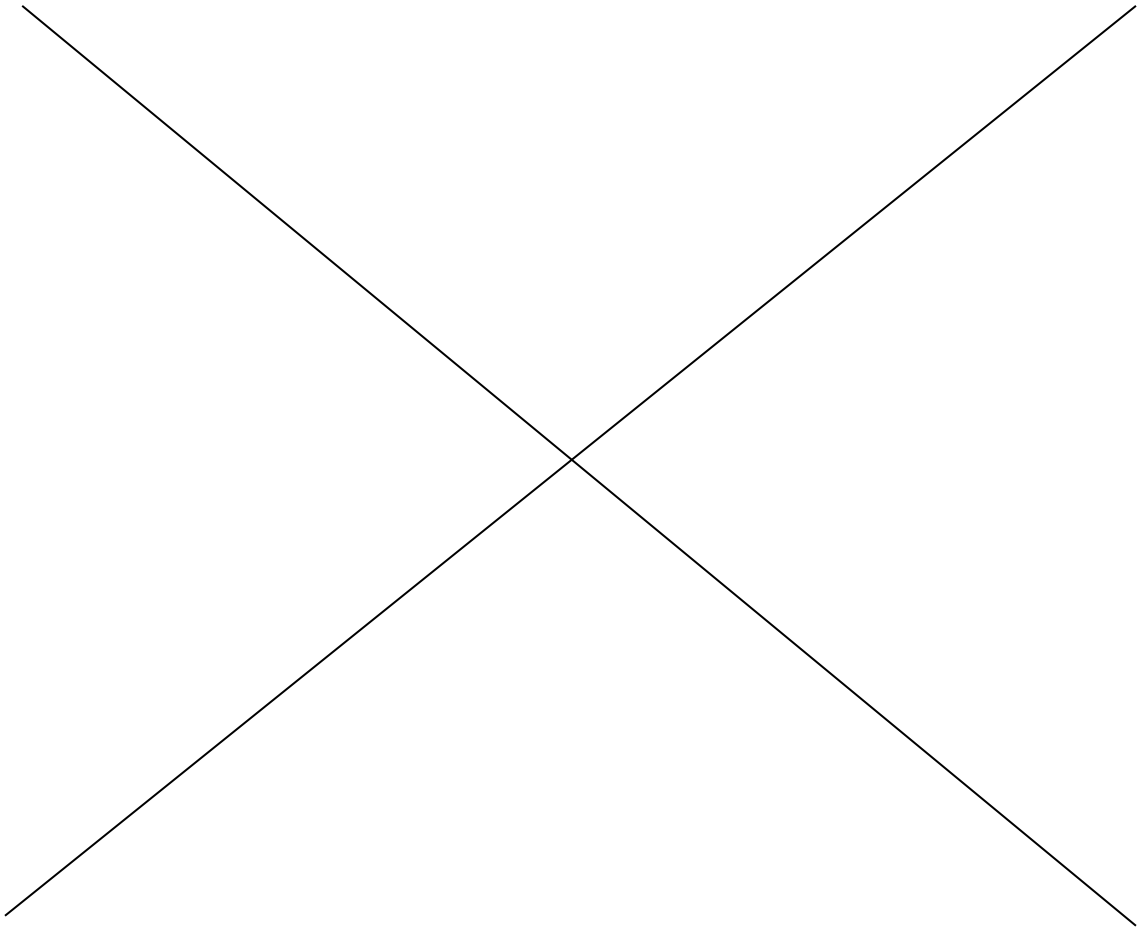
2018

Ausgangssituation

Als Mitarbeiter eines Projektteams werden Sie mit der Entwicklung eines Online-Systems für den Support von IT-Systemen beauftragt. Die Grundaufgabe des Systems besteht darin, Kunden bei auftretenden IT-Problemen zu unterstützen. Hierzu werden Störmeldungen (=Ticket) erfasst und von qualifizierten Supportmitarbeitern bearbeitet.

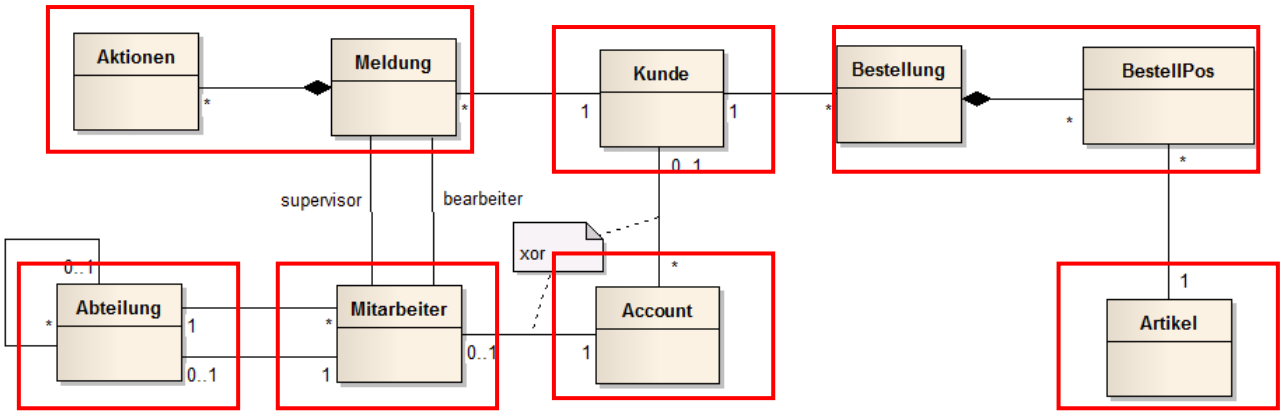
Produktvision:

- Ein Kunde soll für auftretende Probleme Störmeldungen erfassen können.
- Die Supportmitarbeiter bearbeiten die Meldungen gemäß einer festzulegenden Dringlichkeit.
- Die Kunden können zusätzlich über das System Artikel, wie z.B. Monitore und Drucker, bestellen.
- Damit ein Benutzer eine Funktion ausführen kann muss er sich am System anmelden.
- Das System soll sowohl über einen Web-Browser, als auch für gängige Smartphones angeboten werden.
- Bei der Umsetzung ist auf eine möglichst hohe Wiederverwendbarkeit zu achten.



1. Domain-driven Design

1a) Im Rahmen des Designs wenden Sie nun die Methode „Domain-driven Design“ auf das nachstehende Produktmodell an. Analysieren Sie das nachfolgende Diagramm und kennzeichnen Sie alle Klassen in diesem Diagramm. Stellen Sie folgende Kernelemente innerhalb des Diagramms heraus: Aggregate, Entity, Root-Entity, Value-Objects. Verwenden Sie falls notwendig Stereotypen.



1b) Wozu wird ein Bounded-Context sowie eine Context-Map benötigt? Begründen Sie warum diese beiden Konzepte sinnvoll sind und wie diese zusammenspielen. Punkte _5__

Problem: In großen Projekte werden verschiedene Modelle verwendet Wird der Code (basierend auf verschiedenen Modellen) kombiniert wird die Software fehleranfällig, unzuverlässig und schwer verständlich
Bounde-Context: Definieren Sie explizit den Kontext in welchem ein Modell Anwendung findet. Weisen Sie explizite Grenzen zu. Das Modell muss konsistent innerhalb dieser Grenzen sein
Context-Map: Beschreibe die Berührungspunkte zwischen den Modellen, Bilde die relevanten Bereiche der Modelle aufeinander ab, Definieren einen Transformationservice

1c) Erläutern Sie den Lebenszyklus eines Domain-Objekts und erläutern Sie die Auswirkung auf die Implementierung von Assoziationen. Wie findet sich dieses Konzept im Zusammenspiel mit Aggregaten wieder?

- Domänenobjekte existieren nicht nur im Hauptspeicher sondern auch in der Datenbank. Sie haben unterschiedliche Repräsentationen
- Der Lebenszyklus, wie ihn eine Programmiersprache definiert, ist für Domänenobjekte zu klein gefasst

Bei der Implementierung von Assoziationen müssen unterschiedliche Lebenszyklen berücksichtigt werden.
Assoziationen zwischen Aggregaten können nicht durch Objektreferenzen abgebildet werden. Innerhalb eines Aggregates können direkte Referenzen benutzt werden.

2. Software Architektur & Design Prinzipien

2a) Beschreiben Sie die Aufgabe des Konfigurationsmanagers innerhalb einer Komponentenarchitektur. Begründen Sie warum der Konfigurator essentiell für die Umsetzung der Losen Kopplung ist.

Der Konfigurator (= Software welches die Konfiguration vornimmt)

- sieht sowohl die Schnittstellen der Komponenten
- als auch die Implementierung der Komponenten

Die Konfigurator realisiert das Dependency-Inversion-Principle. Die Komponenten sind nicht für die Erzeugung abhängiger Komponenten zuständig. Diese Aufgabe wird an die Umgebung/Konfigurator verlagert.

Komponenten müssen nichts über die Implementierung wissen.

2b) Konfiguratoren können auf Basis des Prinzips „Dependency Injection" implementiert werden. Erläutern Sie das Prinzip und begründen Sie warum dieses Prinzip für eine maximale Entkopplung der Software-Komponenten und dem Konfigurator sorgt.

Die Umgebung injiziert die Abhängigkeit durch set-/Construckter/Attribute-Injection in die jeweilige Komponente

Dadurch müssen die Komponenten nichts über die Implementierungen wissen und sind nicht abhängig vom Konfigurator. Dies garantiert eine maximale Entkopplung

2c) Bewerten Sie folgende Aussage: „Die Implementierung einer Komponenten-architektur ist äußerst aufwändig und kompliziert. Die direkte Kopplung ohne Konfigurationsmanager ist einfacher. Komponentenarchitektur hat keinen Nutzen"

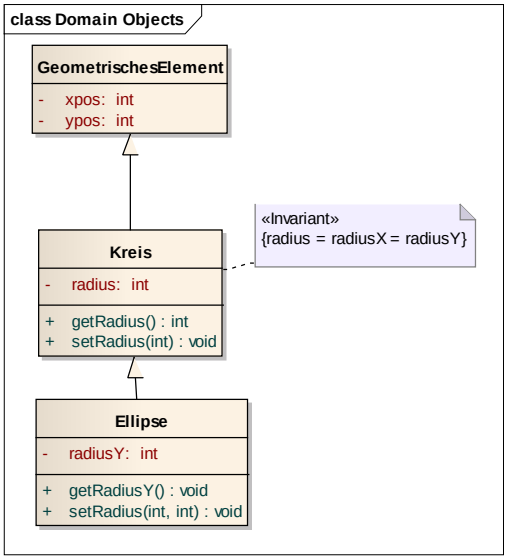
- Dieses Strukturierungsprinzip benötigt etwas mehr overhead da schnittstellen und Komponenten defineirt werden müssen
- Dieser Overhead ist gering wenn es sich um ein größeres Projekt handelt.
- Dieser Ansatz garantiert höhere Flexibilität und bessere Erweiterbarkeit
- Beugt Fehler durch die Reduktionv on Kopplung vor.

2d) Beschreiben Sie das „**Dependency Inversion Principle**“. Begründen Sie warum der Einsatz dieses Prinzips sinnvoll ist. Was wird durch dieses Prinzip erleichtert?

- **Dependency Inversion Principle**: „A. Module hoher Ebenen sollten nicht von Modulen niedriger Ebenen abhängen. Beide sollten von Interfaces abhängen. B. Interfaces sollten nicht von Details abhängen. Details sollten von Interfaces abhängen.“
    - Damit ist sichergestellt, dass die Abhängigkeitsbeziehungen immer in eine Richtung verlaufen, von den konkreten zu den abstrakten Modulen, von den abgeleiteten Klassen zu den Basisklassen
    - Damit werden die Abhängigkeiten zwischen den Modulen reduziert und insbesondere zyklische Abhängigkeiten vermieden.

- Vorteile:
    - In Kombination mit Dependency Injection Containern sehr einfach anwendbar.
    - Sehr universell, da auf praktisch allen Strukturierungs-Ebenen gültig. Dadurch breiter Anwendungsbereich.
- Nachteile:
    - erhöhter Entwurfs- und Umsetzungsaufwand
    - weniger konkret, dadurch weniger unmittelbar verständlich

2e) Definieren Sie das **Liskovsches Substitutionsprinzip**. Erstellen Sie eine Skizze welches das Problem veranschaulicht.



Wichtige Eigenschaften nach Liskov:

- Sei q(x) eine beweisbare Eigenschaft von Objekten x des Typs T. Dann soll q(y) für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.
- Man kann ein Objekt y des Untertyps S von T so verwenden, als wäre es ein Objekt x vom Typ T.
- Alle Eigenschaften (Invarianten) der Oberklasse müssen auch/immer für alle Unterklassen gelten
- Beim Spezialisieren/Generalisieren muss dies Berücksichtigt werden
- Bedingungen in den Unterklassen dürfen Bedingungen der Oberklasse verfeinern, niemals aber erweitern
- Vermeidung schwerwiegender semantischer Probleme bei der Verwendung von Subtypen.

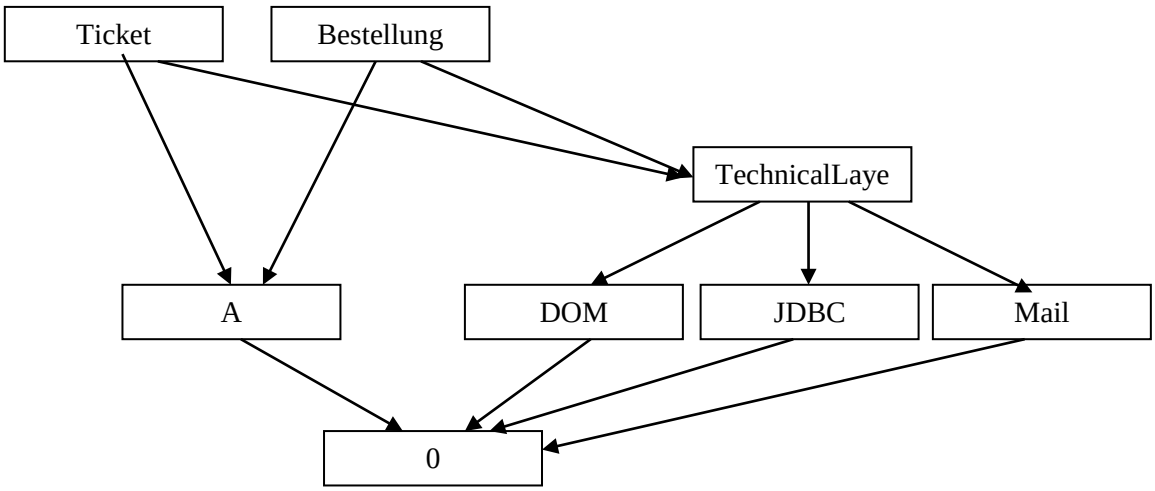
3.) Software Architektur – Die Struktursicht

Im Rahmen einer Architekturbewertung sollen Sie nun Komponenten, sowie deren Schnittstellen bewerten. Hierzu wird ihnen für das Repository des Ticketsystems folgendes Quellcodefragment vorgelegt (Schnittstellenspezifikation):

```
public interface TechnicalLayer {
 // Studiengang laden
 java.jdom.JDomDocument loadMeldung( String meldungNr );
 // Studiengang in der Datenbank speichern
 void saveArtikel(jdbc.Connection conn, Artikel artikel );
 // Mail versenden
 void sendMail( java.mail.Adress recipient, String betreff, String body );
}
```

3a) Skizzieren Sie den Software-Kategoriegraphen unter der Annahme, dass die Anwendungskomponente „Ticket-Center“, sowie die Anwendungskomponente „Bestellung“ auf das TechnicalLayer zugreifen. Darüber hinaus gelten folgende Annahmen:

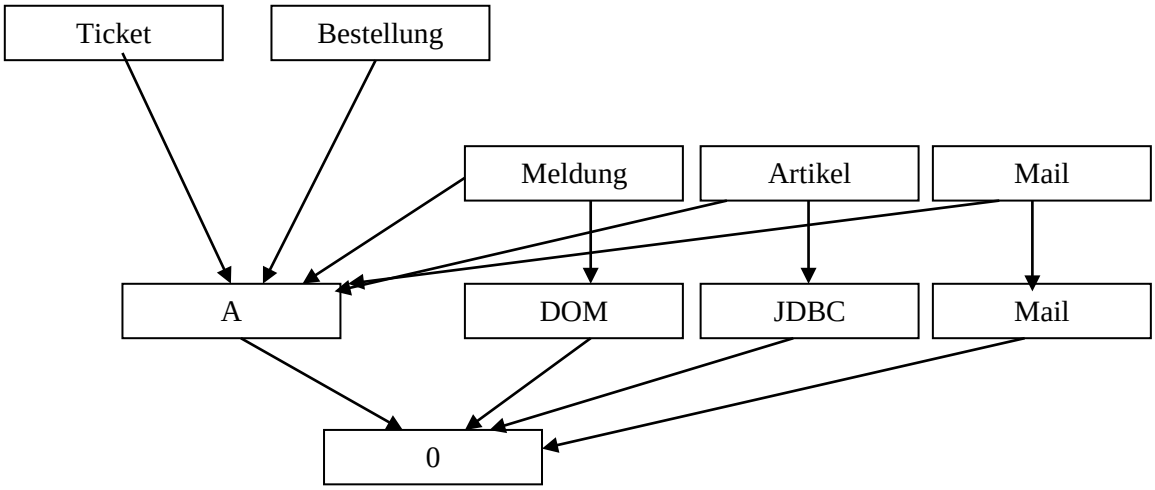
- Alle Datenbankspezifischen Klassen/Interfaces (jdbc) liegen in der Kategorie „JDBC“.
- Alle Email spezifischen Klassen/Interfaces (java.mail) liegen in der Kategorie „Mail“
- Alle XML/Dom spezifischen Klassen/Interfaces (java.jdom) liegen in der Kategorie „XML“
- Die Klassen des Produktmodells werden in der Kategorie „A“ (Anwendung) angesiedelt
- Collections liegen in Kategorie 0



3b) Begründen Sie, warum man gemäß Quasar A und T Software immer trennen sollte. Geben Sie ein negativ Beispiel und dessen Auswirkung an.

- Die technische Basis heutiger System verändert sich permanent/schnell. Deshalb ist die Trennung wichtig um Flexibilität und Erweiterbarkeit einfach zu gestalten
- Die Domain-Logik muss unabhängig von der Technologie sein
- Problem: Die Datenbank soll von JDBC auf No-SQL umgestellt werden

3c) Zeichnen Sie nun einen verbesserten Kategoriegraphen. Zerlegen Sie das Interface „TechnicalLayer“ in verschiedene Interfaces, um das Interface-Segregation-Prinzip zu gewährleisten.



3d) Definieren Sie das Konzept der R-Software. Welche besondere Einschränkung muss bei R-Software gelten. Welche Kategorien aus 3c entsprechen dieser R-Software

- Verwende Transformationssoftware um die Daten zwischen verschiedenen Welten zu „transformieren“
- R-Software übersetzt die Domänen-Welt in die Technologie
- R-Software: Software die nur transformiert aber keine fachlichen Aufgaben übernimmt

3e) Die Schnittstelle „TicketCenterIF“ wird in die Kategorie 0 gelegt. Welche Auswirkung hat diese Aussage auf die Parameter und Rückgabewerte der Interface-Methode(n). Wie können basierend auf dieser Entscheidung komplexe Objekte übergeben werden (2 Varianten)?

- Es dürfen nur Aftefakte verwendet werden welche ebenfalls in Kategori 0 liegen. Dies sind in der Regel Typen wie String und Objeckt
- Komplexe Objekte können nicht direkt übergeben werden. Entweder als „Object“ oder als z.B. HashMap.
- Dies ist Kompliziert und nicht typisiert, Fehleranfällig

4.) Prozessicht & Skalierbarkeit

4a) Erläutern Sie die Grundidee des Session Context Manager-Patterns. Begründen Sie warum der Einsatz dieses Patterns die Skalierbarkeit der Software verbessert.

Problem

- Wie kann ein Worker Multiple Request Sessions bearbeiten, ohne exklusiv einem Client zugeordnet zu sein?

Lösung:

- Ein Session Context Manager verwaltet Zustand und Daten einer Multiple Request Session
- Ein Worker bearbeitet immer genau einen Service Request, indem er den zugehörigen Kontext vom Manager anfordert, den Service Request beantwortet und den geänderten Kontext an den Manager zurück gibt.
- Zur Identifikation der Session wird eine ID verwendet
- Der Client muss die ID zusammen mit dem Service Request übertrag

4b) Beschreiben Sie die Skalierungsstrategie **Scale-Out**. Sind Anapssung an der Software notwendig um diese Strategie nutzen zu können (Begründung)?

- **Scale-Out (horizontale Skalierung)**
  - Steigerung der Leistung des Systems durch Hinzufügen zusätzlicher Rechner/Knoten
    - Parallelbetrieb identischer Software auf mehreren Knoten
    - Elastizität: Skalierbarkeit der Anwendung während des laufenden Betriebs
  - Scale-Out ist sehr stark von der implementierten Software-Architektur abhängig.
  - Refrakturierung der Applikationen um die Effekte bei einem Scale-Out nutzen zu können.
    - Aufspalten eines Monolithen in mehrere, unabhängige einzelne Applikationen

4c) Eine Maßnahme zur Verbesserung der Skalierung lautet „Asynchrone Kommunikation“. Was wird darunter verstanden und wie kann dadurch die horizontale Skalierbarkeit besser unterstützt werden? Welche Nachteile ergeben sich dadurch?

Problem bei synchroner Kommunikation:

- Der Client/Aufrufer blockiert bis das Ergebnis des Aufrufes eintrifft.

Lösung: Asynchrone Kommunikation:

- Der Aufrufende Dienst/Client muss nicht auf das Ende eines Auftrages warten.
- Auftrag wird von Empfänger quittiert (Empfangsbestätigung)
- Kein warten bis der eigentlich Auftrag verarbeitet wurde
- Er wird evtl. am Ende von dem erfolgreichen Abschluss informiert.

Nachteile der losen Kopplung

- Programmlogik wird komplizierter
- Nachrichten können in unterschiedlichen Reihenfolge eintreffen

5) Die physische Sicht

5a) Erläutern Sie die Grundidee eines Fail-Over-Clusters mit automatic fail-over. Skizzieren Sie eine Idee wie der Ausfall/Fehlfunktion eines Knoten erkannt werden kann.

- Erkennen das ein Konten ausgefallen oder nicht voll funktionsfähig ist
  - Prüfen der Cluster-Software über das verteilte Cluster-Membership-Protokoll mit Hilfe von Heartbeat-Alive-Checks
  - Prüfen der Betriebsumgebung: durch Monitoring von Systemresourcen wie z.B. CPU-Last, Swap-Space
  - Prüfen der Anwendungen: durch Aufruf einer Monitoringschnittstelle oder z.B. Überwachen einer Prozessliste
-
- Durchführen eines Fail-Overs
  - Anwendung auf den ursprünglichen Knoten beenden
    - Stop-Skripte
  - Recover ausführen (optional)
  - Anwendung auf dem neuen Knoten starten
    - Start-Skripte

5b) Analysieren sie das folgende Programmfragment in Hinblick auf den Einsatz in einem Application-Cluster. Identifizieren Sie die problematische(n) Stellen und geben Sie jeweils eine Begründung an. Skizzieren Sie für jedes der Probleme eine mögliche Lösung.

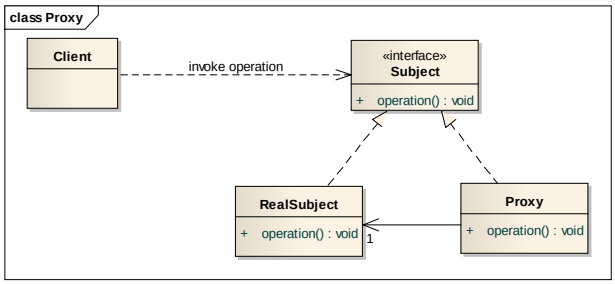
```
01 public class NumberServiceImpl implements NumberService {
02
03    Map<String,AtomicLong> numberMap = new HashMap<String,AtomicLong>();
04
05    public long getNextNumber( String category ) {
06        synchronized( this ) {// Enter this block exclusively
07            if( numberMap.containsKey( category ) == false ) {
08                numberMap.put( category, new AtomicLong( 1 ) );
09            }
10            return numberMap.get( category ).incrementAndGet();
11       }
12    }
13}
```

Problem 1: NumberMap ist an den Hauptspeicher gebunden. Beide Konten vergeben identische Nummer.
Lösung 1: Zustand der Map in einem zentralen repository speicher

Problem 2: Sperren (synchronized) ist nur lokal implementiert.
Lösung 2: Globale sperren wenn die vergabe der nummern im Cluster benutzt

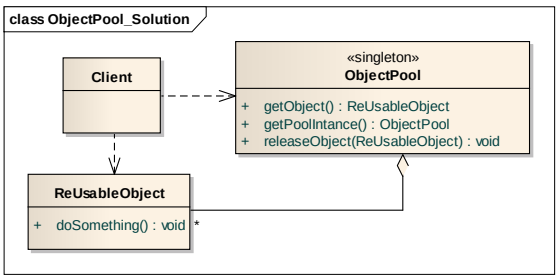
6 Design Pattern

6a) Erläutern Sie die Grundidee des Proxy-Patterns. Geben Sie ein sinnvolles Beispiel für die Verwendung dieses Patterns an.



Zweck: Kontrolliere den Zugriff auf ein Objekt mit Hilfe eines vorgelagerten Stellvertreterobjektes
Subject: Deklariert eine gemeinsame Schnittstelle von RealesSubjekt und Stellvertreter D.h. der Stellvertreter kann überall dort eingesetzt werden wo auch das RealeSubjekt engesetzt werden kann
Proxy: Verwaltet eine Referez auf das reale Objekt. Kontrolliert den Zugriff auf das reale Objekt
RealSubject: Definiert das reale Objekt welches durch den Proxy geschützt werden soll
Beispiel: netzwerkkommunikation

6b) Erläutern Sie die Grundidee/Aufbau des Objekt-Pool-Patterns. Wann wird dieses Pattern eingesetzt? Worauf muss besonders geachtet werden? Wo liegen mögliche Probleme? Geben Sie ein sinnvolles Anwendungsbeispiel an.



Zweck:

- Definiere einen Pool von Objekten
- Stelle diese Objekte einem Client zur Verfügung
- Mach die Objekte wiederverwendbar wenn es nicht mehr benötigt wird.

Vorteile:

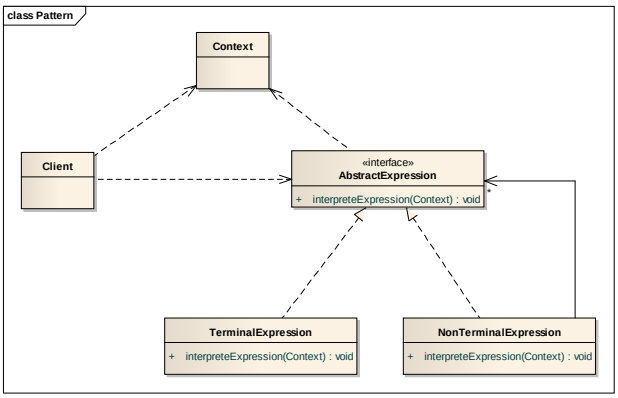
- Erzeugung von aufwendigen Objekten wird optimiert
- Systemressourcen werden geschont

Nachteil:

- Rückgabe von Objekten muss durch Client erfolgen (Programmierfehler)
- Die Wiederverwendbaren Objekte müssen sich vor Ihrer Wiederverwendung wieder in einem Initialzustand befinden
- Der Pool ist eine zentrale Stelle welche unter Umständen zu einem Deadlock führen kann.

6 Design Pattern, Verhaltensmuster

6c) Erläutern Sie die Grundidee des Interpreter-Patterns. Wie unterscheidet sich dieses Pattern von dem Visitor-Pattern. Wo liegen die Grenzen des Interpreter-Patterns. Geben Sie ein sinnvolles Anwendungs-beispiel an.



Context:

- Enthält Information für die Berechnung welche zwischen den Ausdrücken übergeben werden kann

AbstractExpression:

- Abstrakte Sicht auf einen Ausdruck. Der Ausdruck berechnet ein Ergebnis. Das wie ist den konkreten Implementierungen überlassen.

TerminalExpression:

- Ein einfacher Ausdruck welcher durch sich selbst beschrieben ist und keine weiteren Unterausdrücke benötigt.

NonTerminalExpression:

- Ein zusammengesetzter Ausdruck dessen Ergebnis sich aus den Teilergebnissen der Unterausdrücke berechnet.

Unterschied: Nur eine Art von Berechnung

6d) Wenden Sie nun das Interpreter-Muster auf die Berechnung von arithmetische Ausdrücke an. Skizzieren Sie ein Klassendiagramm mit welchem Ausdrücke mit folgenden Eingeschalten dargestellt werden können: Operatoren: *, /, Konstanten und Variablen. Erläutern sie wozu in diesem Fall der „Context“ benötigt wird.