Whiskey 'o clock

Programmentwurf

der Vorlesung „**A**dvanced **S**oftware **E**ngineering“

an der

Dualen Hochschule Baden-Württemberg Karlsruhe

von

Nico Holzhäuser

Abgabedatum 5. Mai 2022

Matrikelnummer	...TBD...
Kurs	TINF19B4
Bearbeitungszeitrum	5. & 6. Semester
Gutachter der Studienakademie	Mirko Dostmann

Inhaltsverzeichnis

Abbildungsverzeichnis

Liste der Algorithmen

Abkürzungsverzeichnis

API	Application Programming Interface	3
DDD	Domain Driven Design	
UML	Unified Modeling Language	ii
UC	Use Case	3

1. Domain Driven Design

1.1 Ubiquitous Language

Die „ubiquitous language“ ist ein Begriff, den Evans in seinem Buch *Domain-Driven Design: Tackling Complexity in the Heart of Software* eingeführt hat. Dieser Begriss beschreibt die allgegenwärtige Sprache, die von Softwareentwickler*innen und Fachexpert*innen gemeinsam gesprochen wird [Plöd 2022]. Sie soll der Basis für die Entwicklung des Softwaremodells sein.

1.1.1 Analyse der Ubiquitous Language

In diesem Projekt ist die „ubiquitous language“ relativ einfach gehalten, da die Fachdomäne überschaubar ist. Aufgrund dieser beschränkten Problemdomäne sollten keine schwerwiegenden Kommunikationsprobleme in der gemeinsamen Sprache auftreten. Jedoch ist es auch hier wichtig, bestimmte Thematiken und Objekttypen exakt zu spezifizieren um Problemen direkt vor deren Entstehung entgegenzuwirken.

Ubiquitous Language	
„normale“ Bezeichnung	Ubiquitous Language
Whiskeyflasche	Kleinste Entität des Systems. Einfach nur eine schöne, meist teure, Flasche Whiskey
Serie	0 .. n Whiskeyflaschen bilden zusammen eine Serie. Diese haben meist einen gemeinsamen Faktor, wie z.B. die Herkunft

1.1.2 Probleme bei der Ubiquitous Language & deren Lösung [Batista 2019]

- Übersetzen von komplexen Sachinhalten in einfache Sprache

  → Durch Wortdefinitionen eliminiert

- Unterschiedliche Bezeichnungen für ein und das Selbe

  → Begriffe aus dem Definitionspool verwenden

- Abstraktion der technischen Sachverhalte für die Domain Experten

  → Klar und deutliche Definitionen im Team und ständige Weiterentwicklung der Ubiquitous Language

- Keine Rücksichtnahme der Domain Experten bei der Entwicklung des technischen Modells

  → Mit einbeziehen alles Parteien für das bestmögliche Ergebnis

1.2 Analyse und Begründung der verwendeten Muster

1.2.1 Analyse

Eine komplette Übersicht des Domain Driven Design ist in Anhang in Abb. A.2 beigefügt.

Value Objects [Milian 2019]

Das Value Object ist ein in der Softwareentwicklung häufig eingesetztes Entwurfsmuster. Wertobjekte(engl. „Value Objects“) sind unveränderbare Objekte, die einen speziellen Wert repräsentieren. Soll der Wert geändert werden, so muss ein neues Wertobjekt erzeugt werden.

Entities [Anil 2021]

Entitäten stellen im Domänemodell Objekte da, welche nicht nur durch die darin enthaltenen Attribute definiert, sondern primär durch ihre Identität, Kontinuität und Persistenz im Laufe der Zeit definiert werden. Entitäten sind im Domänenmodell sehr wichtig, da sie die Basis eines Modells darstellen.

Aggregates [Anil 2021]

Ein Aggregat umfasst mindestens eine Entität: den sogennanten Aggregatstamm, der auch als Stammentität oder primäre Entität bezeichnet wird. Darüber hinaus kann es über mehrere untergeordnete Entitäten und Wertobjekte verfügen, wobei alle Entitäten und Objekte zusammenarbeiten, um erforderliche Verhaltensweisen und Transaktionen zu implementieren. Abb. A.1

Repositories [Šimara 2020]

Ein Repository vermittelt zwischen der Domänen- und der Datenzuordnungsebene mithilfe einer sammlungsähnlichen Schnittstelle für den Zugriff auf Domänenobjekte. Hierbei kann die Anwendung einfach vorhandene Entitäten aus der Persistenzebene laden, speichern, updaten oder löschen.

Domain Service [Gorodinski 2012]

Eric Evans beschreibt in seinem Buch *Domain-Driven Design: Tackling Complexity in the Heart of Software* einen Domänen Service wie folgt :

> When a significant process or transformation in the domain is not a natural responsibility of an ENTITY or VALUE OBJECT, add an operation to the model as standalone interface declared as a SERVICE. Define the interface in terms of the language of the model and make sure the operation name is part of the UBIQUITOUS LANGUAGE. Make the SERVICE stateless.

Hierbei ist grob gesagt, dass ein signifikanter Prozess oder eine Transformation, die über die Verantwortung einer Entität hinaus geht, als eine eigenständige Operation des Modells als Interface mit dem Namen „Service“ hinzugefügt werden muss. Somit beschreibt ein Service einen Prozess, der über die Verantwortung einer Entität oder eines Wertobjektes hinaus geht.

1.2.2 Begründung

Value Objects

In diesem Projekt sollen Wertobjekte dazu dienen, die Übertragung zwischen Backend und Frontend zu gewährleisten.Meist werden Sie für Read-,Update- oder Modifizierungsaufrufe verwendet. Hierbei sind diese „DTO“ Objekte unveränderlich und haben meist die Inhalte einer Entität. So können Entitäten in einer Serialisierten Form ausgetauscht werden.

Entities

In diesem Projekt sind vier Entitäten vorhanden, die die Basis des Domänenmodells bilden vorhanden :

- »Country« - Land
- »Manufacturer« - Hersteller/Abfüller
- »Bottle« - Flasche
- »Serie« - Flaschenserie

Diese werden durch ihre Identität selbst beschriebene, werden persistiert und haben veränderliche Attribute.

Aggregates

Aggregate sind in diesem Projekt im eigentlichen Sinne nicht vorhanden. Jedoch sind nach der Definition in Abschnitt 1.2.1 auch einzelne Entitäten ein Aggregatstamm und samit ein Aggregate im jeweiligen Bereich. Folgen wir diesem Prinzip gibt es in diesem Projekt **vier** Aggregate.

Repositories

In diesem Projekt sind Repositories im Backend zu finden. Hierbei werden sie durch Implementierungen des Interfaces »JpaRepository<Object,PrimaryKey>« umgesetzt. Mit der Hilfe von Hibernate wird so die Persistierungsschicht mit der Applikationsschicht verbunden und Methoden geschaffen, um Objekte aus der Datenbasis zu filtern.

Domain Service

Service's finden in diesem Projekt ebenfalls im Backend ihre Anwendung. Hierbei wird die Logik in sogenannte Services ausgelagert, die bestimmte Entitäten oder Beziehungen anhand eines Use Case (UC) bearbeiten. Hierbei sind in Spring Boot oft die Application Programming Interface (API) Aufrufe einzelne gekapselte Logiken, welche durch einen entsprechenden Serviceaufruf umgesetzt werden.
Desweiteren sind Security oder Transaktionale Komponenten denkbare Services für eine Spring Anwendung.

2. Clean Architecture

2.1 Schichtenarchitektur

Im [] Abb. A.3

2.1.1 Planung

2.1.2 Entscheidung anhand von Kriterien

3. Programming Principles

3.1 SOLID

3.1.1 Analyse

3.1.2 Begründung

3.2 GRASP (insb. Kopplung/Kohäsion)

3.2.1 Analyse

3.2.2 Begründung

3.3 DRY

3.3.1 Analyse

3.3.2 Begründung

4. Refactoring

4.1 Identifizieren von Codesmells

4.1.1 Code Smell 1

Begründung

Fix

4.1.2 Code Smell 2

Begründung

Fix

5. Entwurfsmuster

5.1 Begründung des Einsatzes

5.2 UML Vorher

5.3 UML Nachher

A. Anhang

A.1 Aggregate Visualisierung

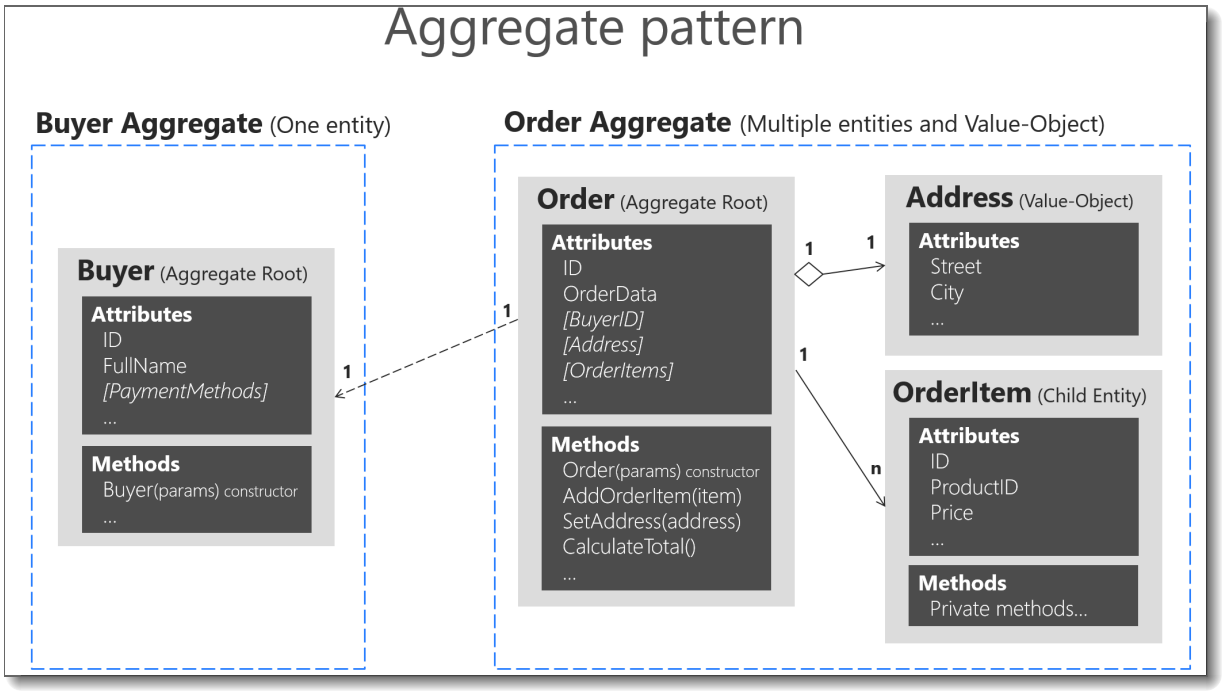

Abbildung A.1: Aggregate Visualisierung [Anil 2021]

A.2 Domain Driven Design Visualisierung

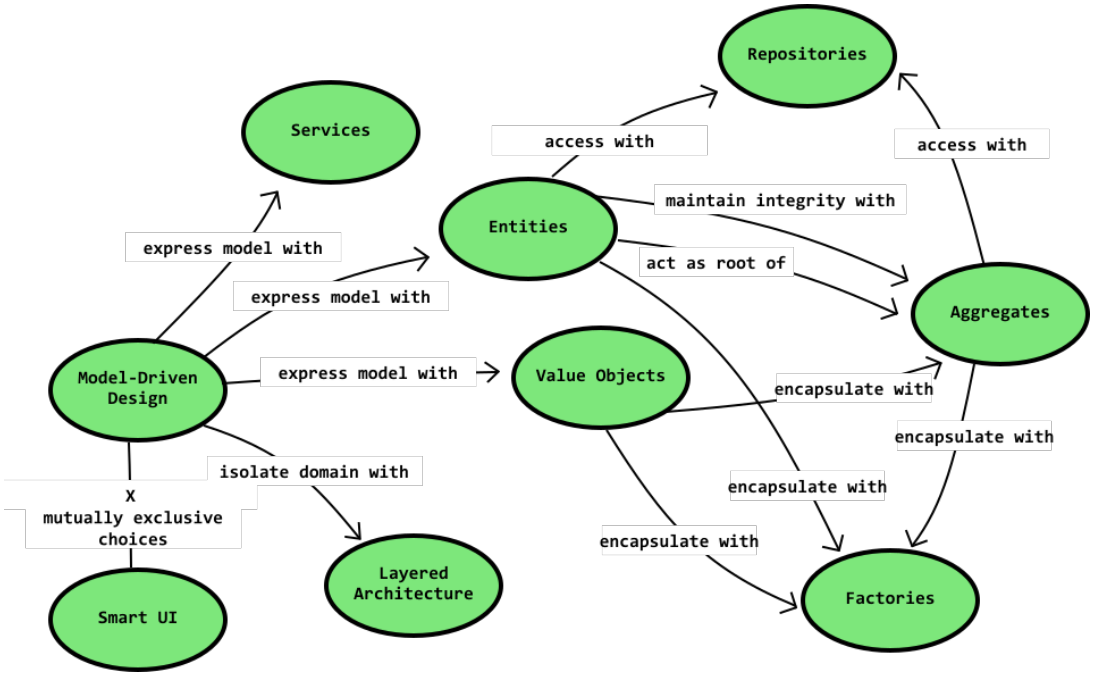

Abbildung A.2: Domain Driven Design Visualisierung [Stemmler 2019]

A.3 Clean Architecuture in Spring Boot

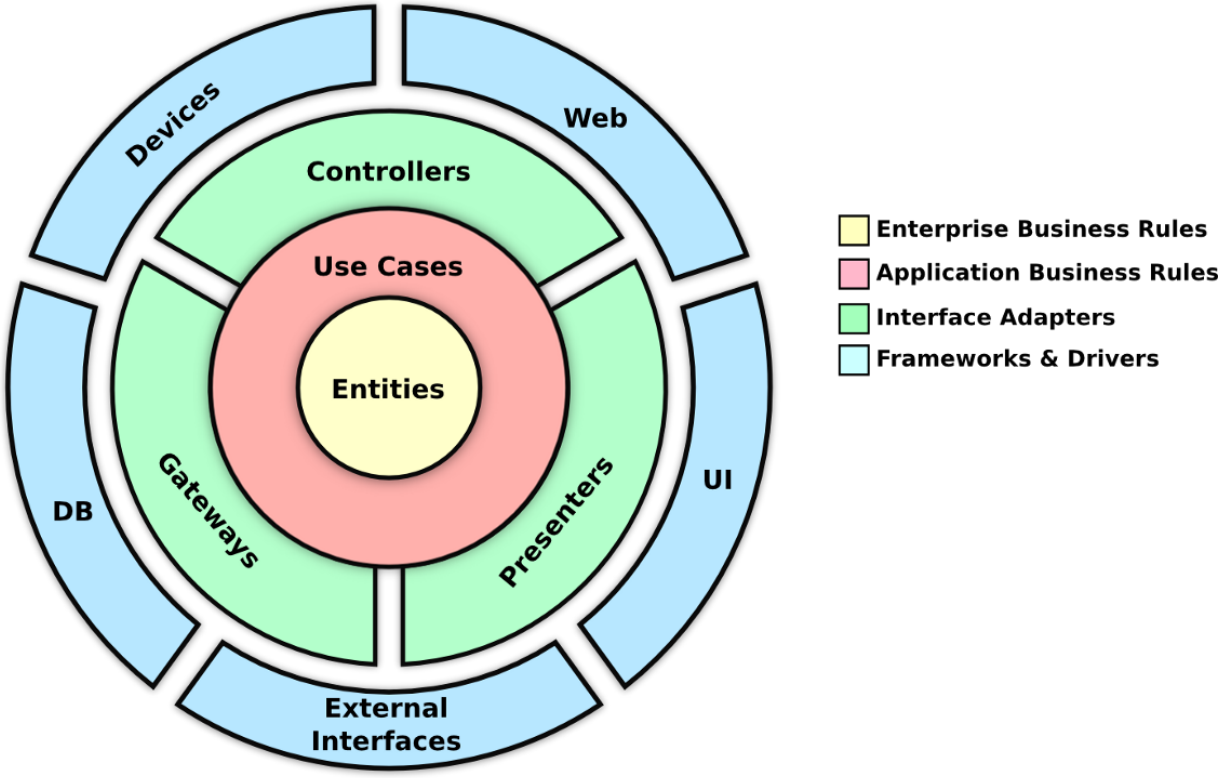

Abbildung A.3: Clean Architecuture in Spring Boot [Hewagamage 2020]

Literaturverzeichnis

Alle Quellen sind zusätzlich im Ordner Quellensicherung gespeichert !

Anil, Nish [2021]. *Entwerfen eines Microservicedomänenmodells.* URL: https://docs.microsoft.com/de-de/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/microservice-domain-model [besucht am 21. 02. 2022] [siehe S. 2, I].

Batista, Felipe De Freitas [2019]. *Developing the ubiquitous language.* URL: https://medium.com/@felipefreitasbatista/developing-the-ubiquitous-language-1382b720bb8c [besucht am 21. 02. 2022] [siehe S. 1].

Evans, Eric [2004]. *Domain-Driven Design: Tackling Complexity in the Heart of Software.* Addison-Wesley [siehe S. 1, 2].

Gorodinski, Leo [2012]. *Services in Domain-Driven Design (DDD).* URL: http://gorodinski.com/blog/2012/04/14/services-in-domain-driven-design-ddd/ [besucht am 21. 02. 2022] [siehe S. 2].

Hewagamage, Pasindu [2020]. *Clean Architecture on Spring Boot.* URL: https://medium.com/@pasinduhewagamage/clean-architecture-on-spring-boot-da3ff7bdcc7 [besucht am 22. 02. 2022] [siehe S. III].

Šimara, Svaťa [2020]. *Domain-Driven Design, part 5 — Repository.* URL: https://docs.microsoft.com/de-de/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/microservice-domain-model [besucht am 21. 02. 2022] [siehe S. 2].

Milian, Paul [2019]. *Value Objects to the rescue!* URL: https://medium.com/swlh/value-objects-to-the-rescue-28c563ad97c6 [besucht am 21. 02. 2022] [siehe S. 2].

Plöd, Michael [2022]. *Qualitätssicherung.* URL: https://entwickler.de/software-architektur/warum-ist-sprache-so-wichtig-001 [besucht am 21. 02. 2022] [siehe S. 1].

Stemmler, Khalil [2019]. *Understanding Domain Entities [with Examples] - DDD w/ TypeScript.* URL: https://khalilstemmler.com/articles/typescript-domain-driven-design/entities/ [besucht am 21. 02. 2022] [siehe S. II].