|  |
| --- |
| Dokumentation zum Praxisprojekt im Kurs Advanced Software Engineering  an der  Dualen Hochschule Baden-Württemberg Karlsruhe  von  Burak Özkan  Marius Engelmeier  Abgabedatum:  30.04.2023 |

Inhaltsverzeichnis

Inhaltsverzeichnis 2

1. Projektbeschreibung 3

2. Klassenübersicht 3

3. Clean Architecture 4

4. Programming Principles 5

4.1 SOLID-Prinzipien 5

4.1.1 Single Responsibility Principle (SRP) 5

4.1.2 Open-Closed Principle (OCP) 5

4.1.3 Liskov Substitution Principle (LSP) 5

4.1.4 Interface Segregation Principle (ISP) 5

4.1.5 Dependency Inversion Principle (DIP) 5

4.2 GRASP-Prinzipien 5

4.3 DRY-Prinzip 6

5. Entwurfsmuster 7

5.1 Beobachter 7

6. Unit Tests 7

7. Refactoring 7

# Projektbeschreibung

S.N.A.C.K steht für das **s**tilvolle **n**eue **a**ttraktive **c**oole **K**nabbersystem. Es handelt sich um einen virtuellen Verkaufsautomaten der verschiedenste Snacks und Getränke anbietet. Der Automat funktioniert wie ein normaler Verkaufsautomat, Münzen werden eingeworfen und der entsprechende Snack/Getränke wird durch ein Nummernpad ausgewählt. Zudem ist es möglich den Automaten zu bearbeiten sodass die Preise oder das Sortiment festgelegt werden können.

Das Projekt soll dazu dienen, Unerfahrene Automatenbenutzer mit dem virtuellen S.N.A.C.K auf die Benutzung von Automaten in der Realwelt vorzubereiten. Zudem können Besitzer von Automaten lernen wie dieser zu initialisieren ist.

Das Programm wurde vollständig in Python implementiert. Die Programmiersprache wurde gewählt, da wir beide bereits Erfahrung mir der Sprache haben und wir uns mit der Sprache am sichersten fühlen.

# Klassenübersicht

Das Programm besteht aus folgenden Klassen:

**VendingMachine**

Die Klasse enthält die Logik des Automaten. Sie hat Methoden zum Einwerfen von Münzen, Auswählen von Produkten und Durchführen von Transaktionen. Sie hat auch einen Observer (VendingMachineGUI-Klasse), der benachrichtigt wird, wenn sich der Status des Automaten ändert.

**CoinSlot**

Die Klasse verwaltet die eingeworfenen Münzen und berechnet den Gesamtbetrag. Sie benachrichtigt den Observer (VendingMachine-Klasse), wenn eine Münze eingeworfen wird.

**ProductList**

Die Klasse enthält eine Liste von Produkten und Methoden zum Hinzufügen, Entfernen und Abrufen von Produkten.

**Product**

Die Klasse enthält Informationen zu einem Produkt wie Name und Preis.

**CoinsDialog**

Die Klasse repräsentiert das Dialogfenster zum Hinzufügen von Münzen. Sie hat eine Methode zum Hinzufügen von Münzen und gibt die ausgewählte Münze zurück.

**ConfigDialog**

Die Klasse repräsentiert das Konfigurationsmenü zum Hinzufügen, Entfernen und Ändern von Produkten. Sie hat eine Methode zum Abrufen der aktuellen Produktliste.

**VendingMachineGUI**

Die Klasse enthält die GUI-Komponenten und die Methoden zum Hinzufügen von Produkten, Anzeigen des Gesamtbetrags und Durchführen von Transaktionen. Sie ist auch der Observer der CoinSlot-Klasse und wird benachrichtigt, wenn eine Münze eingeworfen wird.

# Clean Architecture

Das Programm verwendet mindestens zwei Schichten der Clean Architecture:

Die innerste Schicht ist die Domäne-Schicht, die die Logik des Automaten und die Entitäten wie Product und Coin enthält.

Die äußere Schicht ist die Präsentations-Schicht, die die GUI-Komponenten und die Klassen für die Interaktion mit dem Benutzer enthält. Es gibt keine direkte Abhängigkeit von der GUI zur Domäne-Schicht. Stattdessen vermittelt der Controller (VendingMachineGUI-Klasse) zwischen der Präsentations-Schicht und der Domäne-Schicht.

# Programming Principles

## SOLID-Prinzipien

### Single Responsibility Principle (SRP)

Jede Klasse hat nur eine Verantwortlichkeit, z.B. ist die ProductList-Klasse nur für das Verwalten der Produkte zuständig.

### Open-Closed Principle (OCP)

Die Logik des Automaten (VendingMachine-Klasse) ist für Erweiterungen offen, da z.B. neue Produkte hinzugefügt werden können, ohne den bestehenden Code zu ändern.

### Liskov Substitution Principle (LSP)

Die abgeleiteten Klassen (z.B. CoinsDialog) können anstelle der Basisklasse (z.B. QDialog) verwendet werden.

### Interface Segregation Principle (ISP)

Die Klassen haben nur Methoden, die sie wirklich benötigen, z.B. hat die CoinsDialog-Klasse nur eine Methode zum Hinzufügen von Münzen.

### Dependency Inversion Principle (DIP)

Die Klassen sind abhängig von Abstraktionen (Interfaces) und nicht von konkreten Implementierungen, z.B. hängt die VendingMachine-Klasse von der CoinSlot-Schnittstelle ab und nicht von der CoinSlot-Klasse.

## GRASP-Prinzipien

Controller: Die VendingMachineGUI-Klasse ist der Controller, der zwischen der Benutzeroberfläche und der Logik des Automaten vermittelt.

Information Expert: Die Logik des Automaten (VendingMachine-Klasse) hat alle Informationen, die für die Durchführung der Aufgaben benötigt werden.

Creator: Die VendingMachine-Klasse erstellt Instanzen von CoinSlot, ProductList usw.

Low Coupling: Die Klassen haben möglichst wenige Abhängigkeiten voneinander.

High Cohesion: Die Klassen haben zusammenhängende Aufgaben und nur die benötigten Methoden.

## DRY-Prinzip

Der Code ist so organisiert, dass keine Duplikationen vorhanden sind. Zum Beispiel ist die Logik für die Berechnung des Gesamtbetrags in der CoinSlot-Klasse und der VendingMachine-Klasse vorhanden, aber es gibt keine Duplikation, da beide Klassen dieselbe Methode (get\_total\_amount) verwenden.

# Entwurfsmuster

## Beobachter

Das Programm verwendet das Entwurfsmuster "Beobachter". Die CoinSlot-Klasse ist das "beobachtete" Objekt und die VendingMachine-Klasse ist der "Beobachter". Die CoinSlot-Klasse ruft die update-Methode der VendingMachine-Klasse auf, wenn eine Münze eingeworfen wird. So kann die VendingMachine-Klasse den Gesamtbetrag aktualisieren und den Status des Automaten ändern.

# Unit Tests

1. Die Unit-Tests sind in der entsprechenden Python Datei zu finden.

# Refactoring

kkkkk