Programmentwurf
Food Picker

Name: Magori, Bence
Martrikelnummer: [7722965]

Name: Hübner, Nick
Matrikelnummer: [2760116]

Abgabedatum: [09.06.2024]

Allgemeine Anmerkungen:

- *es darf nicht auf andere Kapitel als Leistungsnachweis verwiesen werden (z.B. in der Form "XY wurde schon in Kapitel 2 behandelt, daher hier keine Ausführung")*
- *alles muss in UTF-8 codiert sein (Text und Code)*
- *sollten mündliche Aussagen den schriftlichen Aufgaben widersprechen, gelten die schriftlichen Aufgaben (ggf. an Anpassung der schriftlichen Aufgaben erinnern!)*
- *alles muss ins Repository (Code, Ausarbeitung und alles was damit zusammenhängt)*
- *die Beispiele sollten wenn möglich vom aktuellen Stand genommen werden*
    - *finden sich dort keine entsprechenden Beispiele, dürfen auch ältere Commits unter Verweis auf den Commit verwendet werden*
    - *Ausnahme: beim Kapitel "Refactoring" darf von vorne herein aus allen Ständen frei gewählt werden (mit Verweis auf den entsprechenden Commit)*
- *falls verlangte Negativ-Beispiele nicht vorhanden sind, müssen entsprechend mehr Positiv-Beispiele gebracht werden*
    - *Achtung: werden im Code entsprechende Negativ-Beispiele gefunden, gibt es keine Punkte für die zusätzlichen Positiv-Beispiele*
    - *Beispiele*
        - *"Nennen Sie jeweils eine Klasse, die das SRP einhält bzw. verletzt."*
            - *Antwort: Es gibt keine Klasse, die SRP verletzt, daher hier 2 Klassen, die SRP einhalten: [Klasse 1], [Klasse 2]*
            - *Bewertung: falls im Code tatsächlich keine Klasse das SRP verletzt: volle Punktzahl ODER falls im Code mind. eine Klasse SRP verletzt: halbe Punktzahl*
- *verlangte Positiv-Beispiele müssen gebracht werden*
- *Code-Beispiel = Code in das Dokument kopieren*

Kapitel 1: Einführung

Übersicht über die Applikation

Git-Repo: https://github.com/WachtelHD/food-picker

Bei der Applikation handelt es sich um einen Rezept-Finder, wenn man dem großen Hunger nicht weg-kommt. Dabei kann man entweder eine Essensrichtung, ein bestimmtes Gericht angeben oder sogar

welche vorschlagen lassen, um sich inspirieren zu lassen. Dabei wird jeweils eine API aufgerufen, die auf externe Listen mit Rezepten und Nährwerten zugreift.

Die Hauptkomponenten der Applikation umfassen verschiedenen Services und Klassen, die miteinander interagieren, um die gewünschten Informationen bereitzustellen. Der Benutzer kann über eine einfache Konsoleingabe die gewünschten Optionen auswählen, und die Anwendung ruft dann die entsprechenden Daten von den APIs ab.

Die `ApiClient`-Klasse bildet das Herzstück der Kommunikation mit den APIs. Sie übernimmt das Senden von HTTP-Anfragen und das Verarbeiten der Antworten. Durch die Implementierung eines generischen `ApiClient`-Interfaces wird sichergestellt, dass die Anwendung flexibel bleibt und leicht an verschiedene APIs angepasst werden kann, wenn nötig.

Der `EssenService` und der `NaehrwertService` sind spezialisierte Klassen, die die API-Anfragen für spezifische Datenkategorien wie Rezepte und Nährwertinformationen verwalten. Diese Services abstrahieren die Sachen der API-Kommunikation und stellen Methoden zur Verfügung, die direkt in der Anwendung verwendet werden können.

Die `SpielFunktionen`-Klasse enthält die Hauptlogik für die Benutzerinteraktionen. Sie bietet verschiedene Methoden an, um Benutzeranfragen zu verarbeiten, wie beispielsweise das Abrufen von Rezepten nach Kategorie oder das Vorschlagen von zufälligen Gerichten. Diese Klasse interagiert eng mit den Services und dem `ApiClient`, um die gewünschten Daten abzurufen und dem Benutzer anzuzeigen.

Zusätzlich gibt es eine `BenutzerEingabe`-Klasse, die für das Entgegennehmen und Verarbeiten von Benutzereingaben zuständig ist. Diese Klasse stellt sicher, dass die Eingaben des Benutzers korrekt verarbeitet und an die entsprechenden Funktionen weitergeleitet werden.

Die Anwendung nutzt auch eine `JsonMapper`-Klasse, die für die Konvertierung der JSON-Daten, die von den APIs abgerufen werden, in die definierten Domänenobjekte verantwortlich ist. Dies ermöglicht eine saubere Trennung zwischen den JSON-Daten von der API und der Logik der Anwendung.

Die Nährwerte API: https://api.api-ninjas.com/v1/recipe?query= nur eine begrenzte Anzahl an Requests erlaubt, wird diese für die Entwicklung auskommentiert. Zum Testen kann sie gerne wieder verwendet werden.

Wie startet man die Applikation?

Zum Start wird die Datei Starter.java im Projekt 0-cleanprojekt-plugin-main aufgerufen. Hier wird man aufgefordert eine von fünf Optionen zu wählen. Nach Eingabe wird eine der Optionen angeboten:

1. Informationen zu einem Gericht aufrufen
2. Gerichtsvorschläge einer spezifischen Richtung anbieten
3. Zufälliges Essen anzeigen
4. Essensvorschläge erhalten
5. Essensvorschläge mit spezifischer Richtung erhalten

Wie testet man die Applikation?

Je nachdem welche Option man auswählt wird ein Rezept ausgegeben. Dazu gerne die Kommentare mit der API entfernen und die Starter.java klasse starten.

1. Hierbei kann man die Nährwerte zu einem bestimmten Gericht, wie zum Beispiel „1lb chicken and fries“ oder „5lb blueberries“ anzeigen lassen.
2. Bei dieser Option gibt man zum Beispiel „seafood“ ein und kriegt ein Gericht mit Rezept und Zutaten aus dieser Richtung.

3. Hier wird ein zufälliges Essen ausgegeben.
4. Bei dieser Option werden zwei oder mehr Gerichte vorgeschlagen und je nach Auswahl ausgegeben. Man wird aufgefordert eine Option zu wählen und als Ergebnis wird das Gericht, wie 1-3 ausgegeben.
5. Wie 4., aber hier kann man auch die Essensrichtung, wie in 3. Angeben. Das Ergebnis soll dasselbe sein wie in 4.

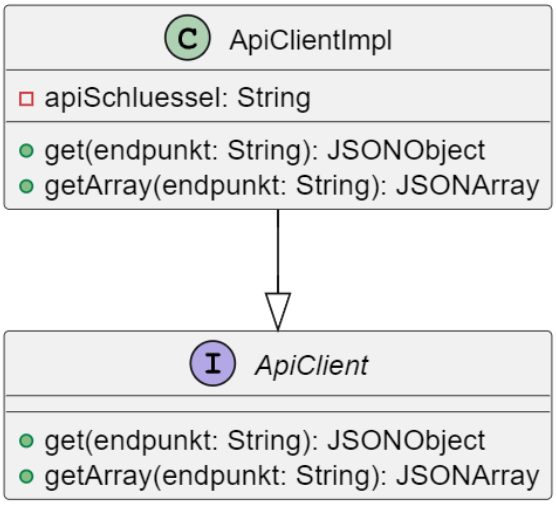
Kapitel 2: Clean Architecture

Was ist Clean Architecture?

Clean Architecture bedeutet einen übersichtlichen, leicht erweiterbaren Aufbau der Software. Dazu gehört auch die leichte Verständlichkeit von den Klassen, Methoden, Variablen und weiteren Benennungen. Ein Clean Architecture erlaubt dadurch die Wartbarkeit für die Zukunft, da die Klassen (optimalerweise) eine lose Kopplung haben und dadurch stets erweitert und optimiert werden können.

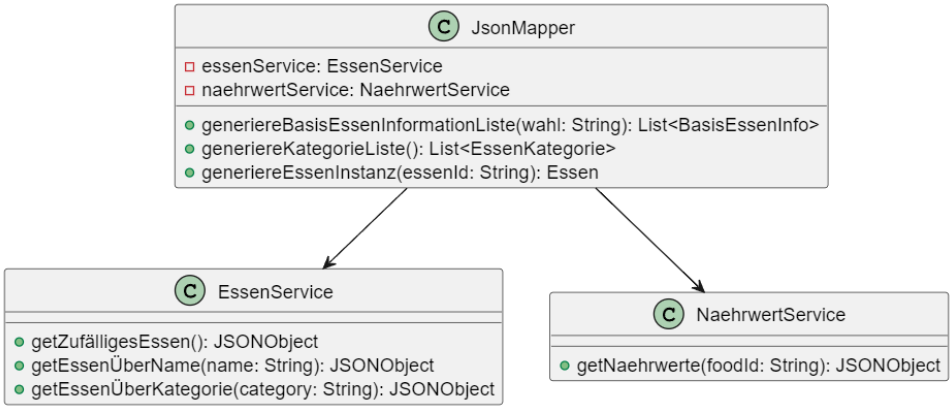
Analyse der Dependency Rule

Positiv-Beispiel: Dependency Rule



- `ApiClientImpl` hängt vom Interface `ApiClient` ab. Dies ermöglicht, verschiedene Implementierungen von `ApiClient` zu verwenden, ohne den Code von `ApiClientImpl` zu ändern.

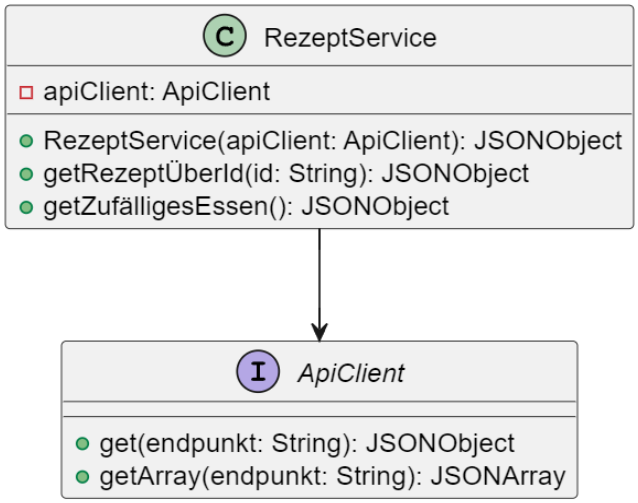
Negativ-Beispiel: Dependency Rule



- `JsonMapper` hängt direkt von den konkreten Implementierungen `EssenService` und `NaehrwertService` ab. Dies bedeutet, dass jede Änderung an diesen Implementierungen Auswirkungen auf `JsonMapper` haben kann. Das könnte also zu größeren Auswirkungen führen im Code.

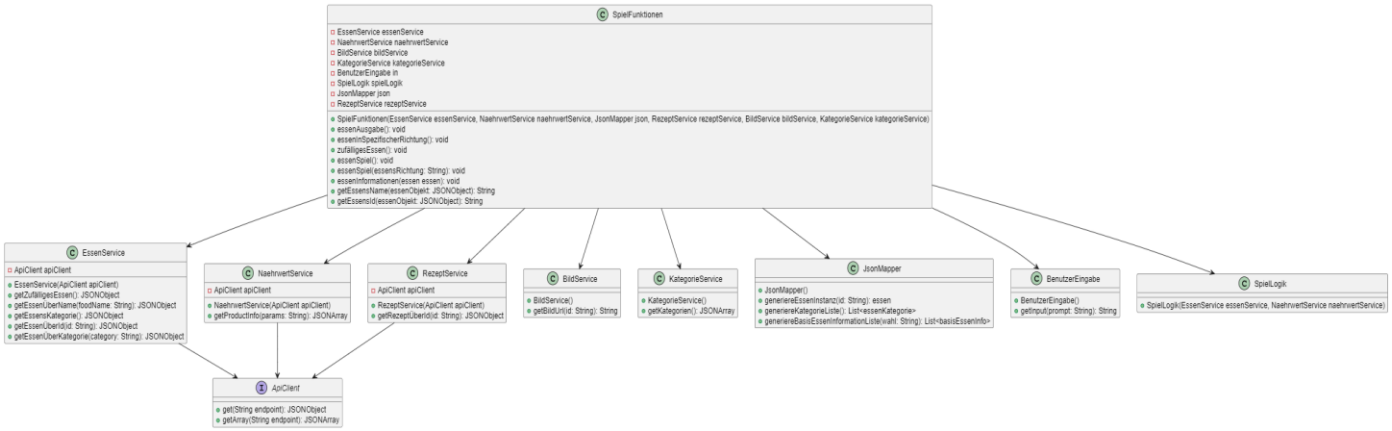
Analyse der Schichten

Schicht: [Infrastruktur]



- `ApiClientImpl` ist verantwortlich für die Kommunikation mit externen Systemen und APIs. Sie enthält Implementierungsdetails, die für den Zugriff auf externe Ressourcen notwendig sind. Deswegen gehört sie zur Infrastruktur-Schicht.

Schicht: [Anwendungslogik]

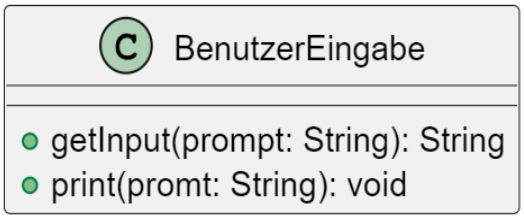


- `SpielFunktionen` ist verantwortlich für die Verarbeitung der Hauptlogik und die Koordination der Iteraktionen zwischen Benutzereingaben und Services. Sie enthält keine direkten Implementierungsdetails der Infrastruktur und bleibt somit Unaabhängig von technischen Details.

Kapitel 3: SOLID

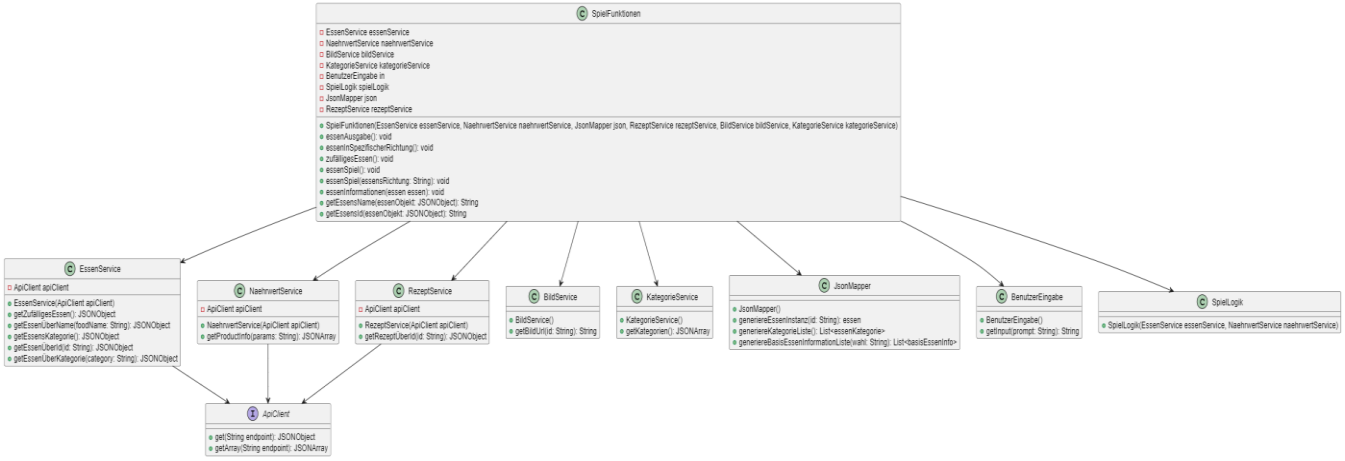
Analyse Single-Responsibility-Principle (SRP)

Positiv-Beispiel



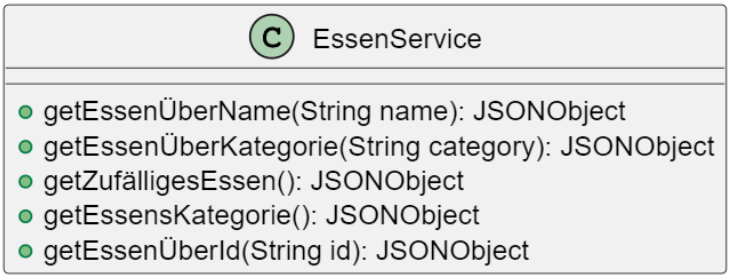
- Die Klasse `BenutzerEingabe` ist dafür verantwortlich, dass die Eingaben vom Benutzer angenommen werden. Sie hat keine weiteren Aufgaben oder Logik, die die Verantwortlichkeit erhöht.

Negativ-Beispiel



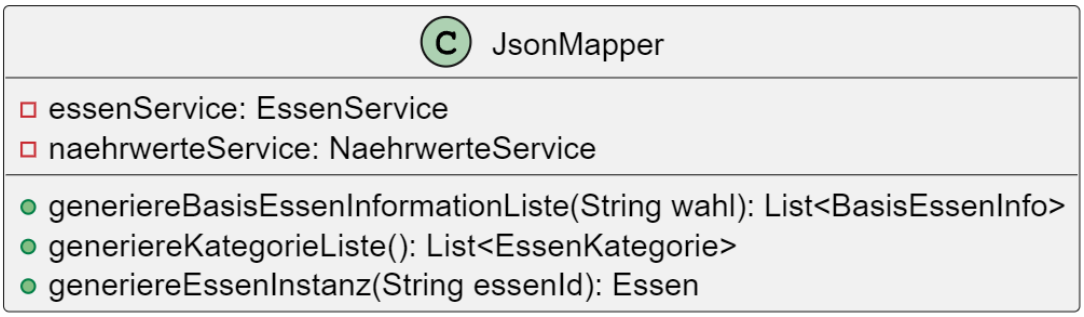
Analyse Open-Closed-Principle (OCP)

Positiv-Beispiel



- Neue API-Aufrufe oder Methoden zur Datenverarbeitung können einfach hinzugefügt werden, ohne die bestehenden Methoden zu ändern.

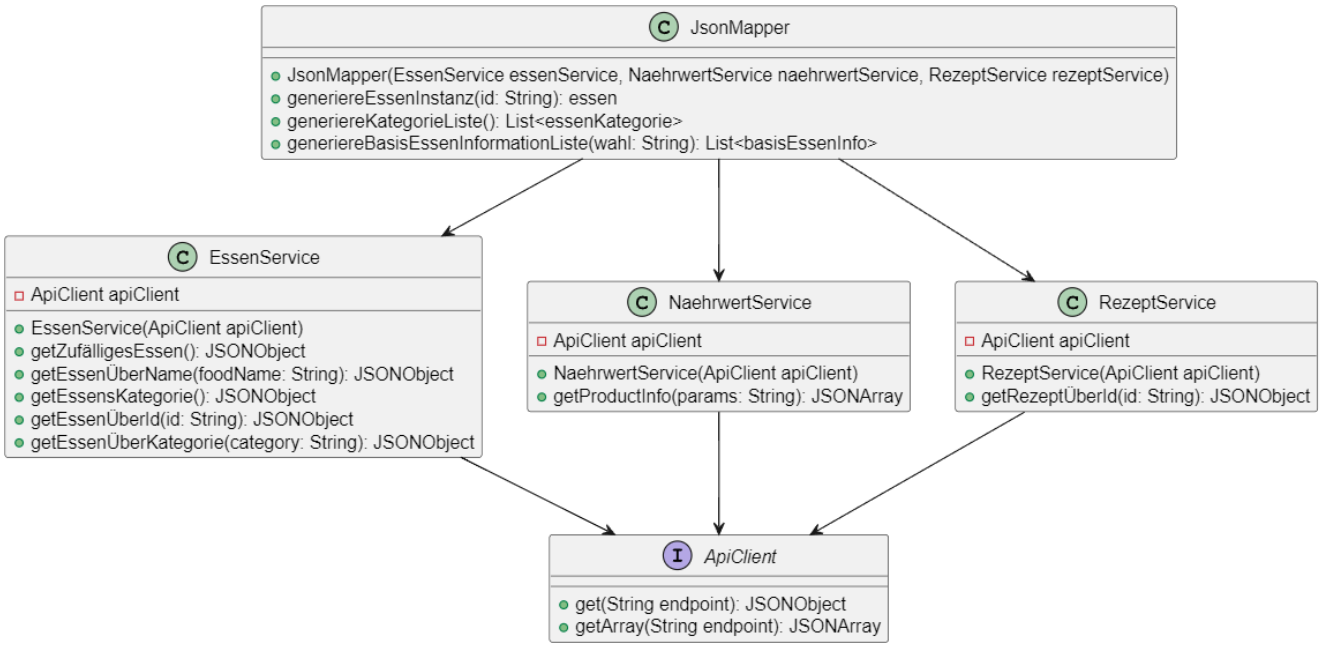
Negativ-Beispiel



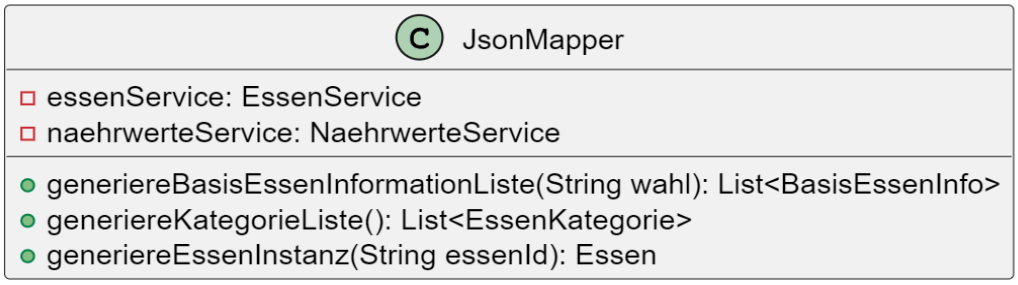
- Änderungen in der API oder neuen Datenstrukturen erfordern Änderungen in den bestehenden Methoden. Zum Beispiel muss in dem Fall BasisEssenInfo angepasst warden, sodas es die neuen API Responses annimmt.

Analyse Liskov-Substitution- (LSP), Interface-Segreggation- (ISP), Dependency-Inversion-Principle (DIP)

Positiv-Beispiel (DIP)



- In der Klasse `JsonMapper` werden Abhängigkeiten wie `EssenService`, `NaehrwerteService` und `RezeptService` über den Konstruktor injiziert.Negativ-Beispiel (DIP)

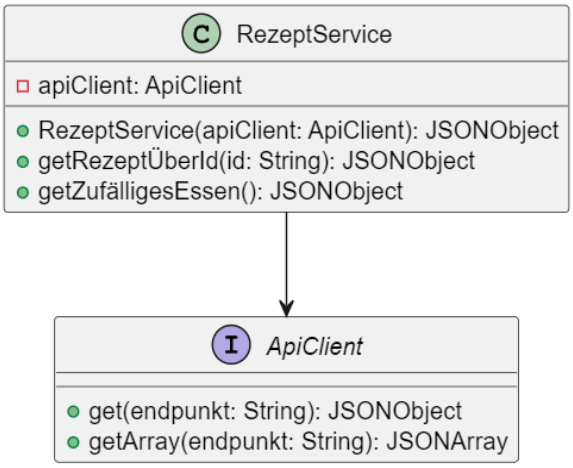


- Die Klasse hängt direkt von konkreten Implementierungen ab, was die Flexibilität und Testbarkeit reduziert.

Kapitel 4: Weitere Prinzipien

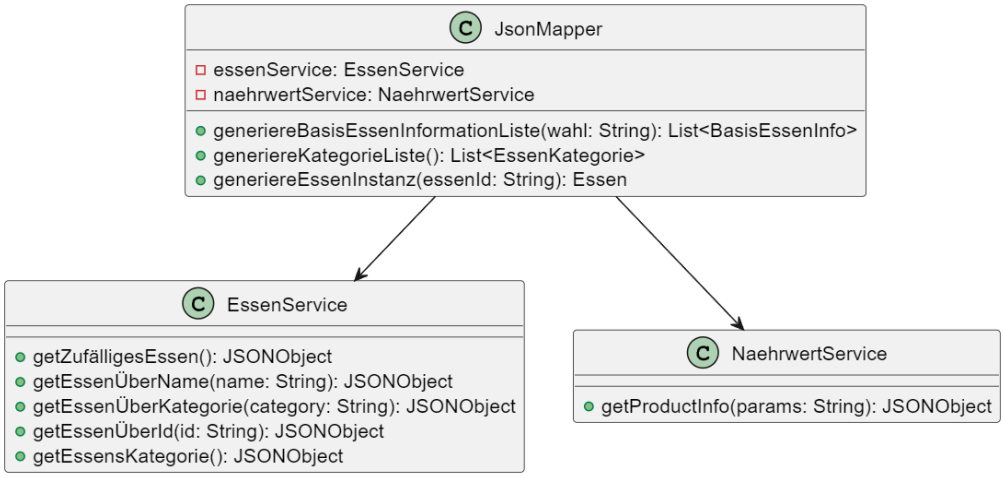
Analyse GRASP: Geringe Kopplung

Positiv-Beispiel



- `RezeptService` hängt nur von der Interface `ApiClient` ab, nicht von einer konkreten Implementierung. Dies ermöglicht eine einfache Erweiterbarkeit des `ApiClient`, ohne dass Änderungen am `RezeptService` gemacht warden müssen. Dadurch bleibt die Klasse flexibel und testbar.
- Die Aufgabe von `RezeptService` ist die Bereitstellung von Rezepten über die Id oder die Bereitstellung der Zutaten über das Rezept

Negativ-Beispiel

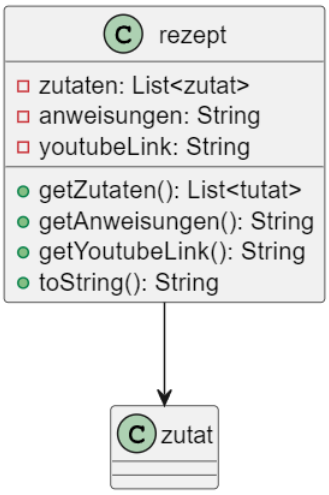


- Verwendet die direkte Implementierung von `EssenService` und `NaehrwerteService`, wodurch eine hohe Kopplung entsteht.

Lösung

Man könnte die Klassen `EssenService` und `NaehrwerteService` in Interfaces auflösen und nur die Implementierungen von den beiden verwenden. Dadurch können Änderungen vorgenommen werden, ohne dass sie größere Auswirkungen auf `JsonMapper` habne.

Analyse GRASP: Hohe Kohäsion



Don’t Repeat Yourself (DRY)

Commit: **7e102d6**
Hier wurden die API-Aufrufe in eine Adapter-Klasse verschoben und in essenService nur noch über diese Klasse aufgerufen. Das hat den Code sowie den Aufwand des API-Calls deutlich reduziert.

10 Unit Tests

[Nennung von 10 Unit-Tests und Beschreibung, was getestet wird]

Unit Test	Beschreibung
testZufälligesEssen	Testet, ob ein zufälliges Essen ausgegeben werden kann
testGetEssensName	Testet, ob der Name eines Essens korrekt zurückgegeben wird
testGetEssensId	Testet, ob die Id eines Essens korrekt zurückgegeben wird
testRezeptServiceGetRezeptById	Testet, ob ein Rezept mit der gegebenen Id zurückgegeben werden kann
testGetProductInfo	Testet, ob die Methode getProductInfo im NaehrwertService die korrekten Nährwertdaten für ein gegebenes Produkt abruft.
testGetProductInfoWithEmptyResponse	Testet, wie die Methode getProductInfo im NaehrwertService mit einer leeren Antwort umgeht.
testGetZufälligesEssen	Testet, ob die Methode getZufälligesEssen im EssenService die korrekten Daten für ein zufälliges Essen abruft.
testGetEssenÜberName	Testet ob die Methode getEssenÜberName im EssenService die korrekten Daten für ein Essen anhand des Namen abruft.
testGetEssensKategorie	Testet ob die Methode getEssensKategorie im EssenService die

	korrekten Kategoriedaten abruft.
testGetEssenÜberId	Testet, ob die Methode getEssenÜberId im EssenService die korrekten Daten für ein Essen anhand der id abruft.

ATRIP: Automatic

Mit Mockito lassen sich Mocks der einzelnen Klassen erstellen und automatiseren. So sind die Tests wiederholbar. Außerdem wurden die Assertions für die Sicherstellung der Ergebnisse verwendet und dadurch konnten die Tests größtenteils automatisiert durchlaufen.

ATRIP: Thorough

```
@Test
public void testGetEssenÜberName() throws ApiException {
   String foodName = "Pizza";
   JSONObject mockResponse = new JSONObject();
   mockResponse.put("meals", new JSONArray().put(new JSONObject().put("strMeal",
"Pizza").put("idMeal", "123")));

   when(mockApiClient.get("https://www.themealdb.com/api/json/v1/1/search.php?s=" +
foodName)).thenReturn(mockResponse);

   JSONObject result = essenService.getEssenÜberName(foodName);

   assertNotNull(result);
   assertTrue(result.has("meals"));
   assertEquals("Pizza", result.getJSONArray("meals").getJSONObject(0).getString("strMeal"));

   verify(mockApiClient, times(1)).get("https://www.themealdb.com/api/json/v1/1/search.php?s=" +
foodName);
}
```

- Der Test prüft, ob die Methode getEssenÜberName die korrekten Daten zurückgibt.
- Dabei werden mehrere Assertions verwendet und eine Mock-Verifikation. Dabei werden die relevanten Aspekte der Methode geprüft und sicherhgestellt, dass die Rückgabewerte korrekt sind.

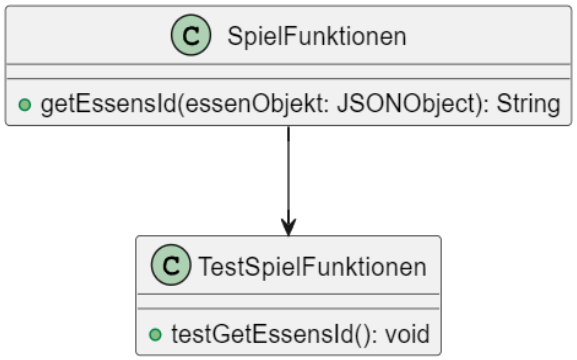
ATRIP: Professional

Code Coverage

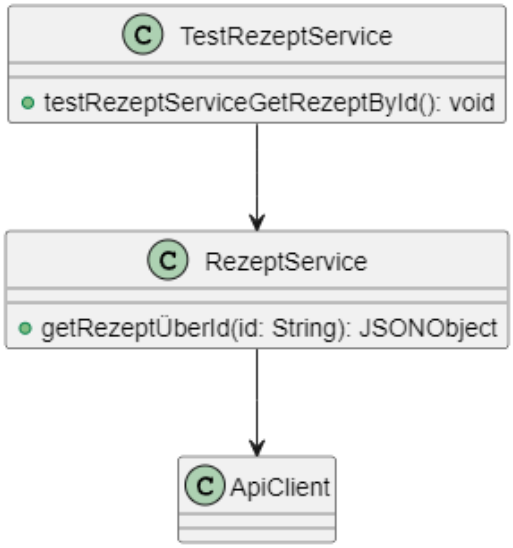
4.48% - Zeit hat nicht gereicht für mehr Tests / Domänen wurden gar nicht und Adapter nur wenig getestet.

Fakes und Mocks

1. mockEssen in testGetEssensName



2. mockRezept in testRezeptServiceGetRezeptById



Kapitel 6: Domain Driven Design

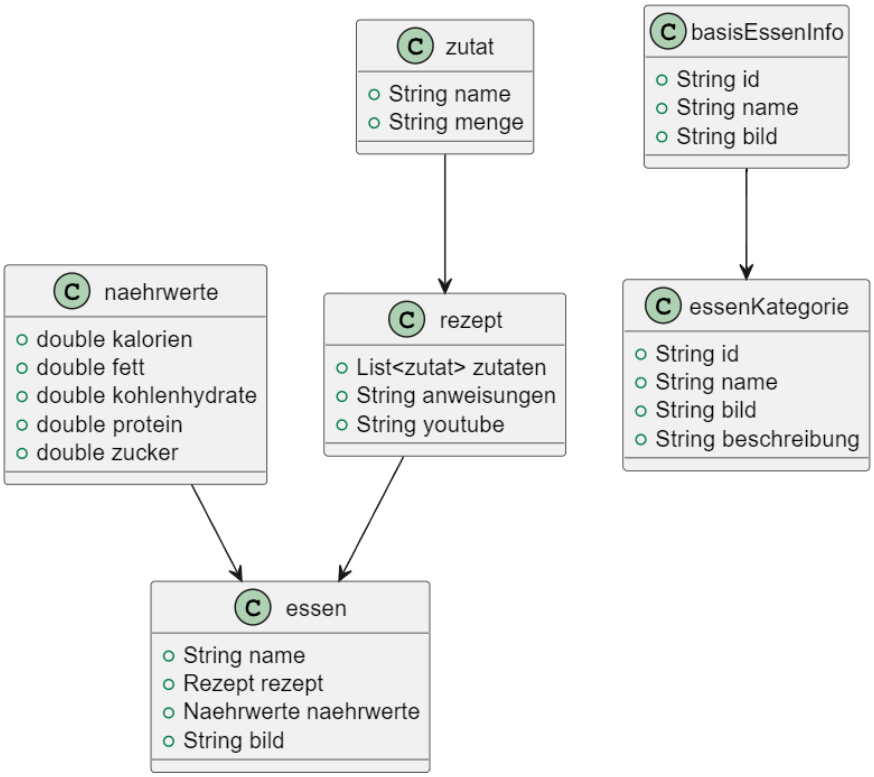
Ubiquitous Language

Bezeichnung	Bedeutung	Begründung
`Naehrwerte`	Die Werte eines Gerichts (Kalorien, Fett, Proteine, etc).	Wichtige Informationen für Nutzer, die sich gesund ernähren oder spezielle Diäten einhalten möchte.
`Essen`	Ein Gericht, das aus mehreren Zutaten zubereitet wird.	Wird in der gesamten Domäne zur Beschreibung von Gerichten verwendet, die über die API abgerufen werden.
`Rezept`	Die Anweisungen und Zutaten zur Zubereitung eines Gerichts.	Ein Element in der Domäne, das beschreibt, wie ein Gericht zubereitet wird.
`Kategorie`	Eine Gruppe von Gerichten, die eine gemeinsame Eigenschaft haben (z.B. italienisch, Dessert)	Kategorisiert die verschiedenen Gerichte und beschleunigt die Suche.

Entities

Entities repräsentieren eindeutige Identitäten in der Domäne. In diesem Projekt gibt es keine eindeutigen Entitäten, da die Daten hauptsächlich durch externe APIs bereitgestellt werden und keine komplexen Geschäftslogiken oder Zustandsveränderungen erfordern. Alle Datenobjekte warden im wesentlichen durch ihre Attribute definiert. Diese Daten müssen auch nicht persistent gespeichert warden, da sie jederzeit wieder aufgerufen werden können.

Value Objects



- `Naehrwerte`: Enthält die Werte eines Gerichts.
- `RezeptZutat`: Beschreibt eine Zutat und ihre Menge in einem Rezept.
- `Rezept`: Enthält die Zutaten und die Anweisungen zur Zubereitung eines Gerichts.
- `BasisEssenInfo`: Grundlegende Informationen zu einem Gericht.
- `EssenKategorie`: Beschreibt eine Kategorie von Gerichten.
- `Essen`: Kombiniert alle relevanten Informationen zu einem Gericht.

Diese Value Objects sind unveränderlich und repräsentieren beschreibende Aspekte dieser Domän.

Repositories

Da die Daten aus externen APIs stammen und keine komplexe Veränderung der Daten oder Persistenz erfordern, ist der Einsatz von Repositories in diesem Projekt nicht erforderlich. Die Service-Klassen (`EssenService` und `NaehrwerteService`) übernehmen die Aufgabe, die Daten aus den APIs abzurufen und an die Anwendung zu geben.

Aggregates

Die Anwendung konzentriert sich auf das Abrufen und Darstellen von Daten aus externen APIs ohne komplexe Logik oder Zustandsmanagement. Daher sind Aggregates in diesem Projekt nicht notwendig.

Kapitel 7: Refactoring

Code Smells

Code Duplicate -> APIs

- Commit: (387ab46) https://github.com/WachtelHD/food-picker/commit/387ab4643a479b8abdf4c05939f0debf546594df

```
- public class get {
-     public static void main(String[] args) {
-
-     }
-
-     public void getProductInfo(){
-         String params = "pie";
-         String APIKey = "3kTkxnTFVv/5fW3/lKu70A==fHOzKx39l17N3X0C";
-     public void getProductRange(){
-         String APIKey = "1";
-
-         try {
-             URL url = new URL("https://www.themealdb.com/api/json/v1/1/random.php");
-
-             HttpURLConnection connection = (HttpURLConnection) url.openConnection();
-             connection.setRequestProperty("accept", "application/json");
-             connection.setRequestProperty("X-Api-Key", APIKey);
-
-             BufferedReader br = null;
-             if (connection.getResponseCode() == 200) {
-                 br = new BufferedReader(new InputStreamReader(connection.getInputStream()));
-                 String strCurrentLine;
-                     while ((strCurrentLine = br.readLine()) != null) {
-                         System.out.println(strCurrentLine);
-                     }
-             } else {
-                 br = new BufferedReader(new InputStreamReader(connection.getErrorStream()));
-                 String strCurrentLine;
-                     while ((strCurrentLine = br.readLine()) != null) {
-                         System.out.println(strCurrentLine);
-                     }
-             }
-         } catch (Exception e) {
-             e.printStackTrace();
-         }
-     }
-
```

```
-       public void getProductInfo(String params){
-           String APIKey = "3kTkxnTFVv/5fW3/lKu70A==fHOzKx39l17N3X0C";
-
-
-           // recipe API
-           // https://api.api-ninjas.com/v1/recipe?query=
-
-           try {
-               URL url = new URL("https://api.api-ninjas.com/v1/nutrition?query=" + params);
-               HttpURLConnection connection = (HttpURLConnection) url.openConnection();
-               connection.setRequestProperty("accept", "application/json");
-               connection.setRequestProperty("X-Api-Key", APIKey);
-
-               BufferedReader br = null;
-               if (connection.getResponseCode() == 200) {
-                   br = new BufferedReader(new InputStreamReader(connection.getInputStream()));
-                   String strCurrentLine;
-                       while ((strCurrentLine = br.readLine()) != null) {
-                           System.out.println(strCurrentLine);
-                       }
-               } else {
-                   br = new BufferedReader(new InputStreamReader(connection.getErrorStream()));
-                   String strCurrentLine;
-                       while ((strCurrentLine = br.readLine()) != null) {
-                           System.out.println(strCurrentLine);
-                       }
-               }
-           } catch (Exception e) {
-               e.printStackTrace();
-           }
-       }
```

Code Seam UND long class

- (Zerteilung der Klasse getFunktionen) (7e102d6) https://github.com/WachtelHD/food-picker/commit/7e102d601b428ac43a17609ff023722ac71c4946
- (Löschung dieser Klasse) (6662368) https://github.com/WachtelHD/food-picker/commit/666236860c2e884593c0713b6b1bbb94b03a14d7

```
- public class GetFunktionen {
-
-     public GetFunktionen() {
-
-     }
-     public static void main(String[] args) {
```

```
-
-       }
-
-       public JSONObject getRandomFood(){
-           String APIKey = "1";
-
-           try {
-               URL url = new URL("https://www.themealdb.com/api/json/v1/1/random.php");
-
-               HttpURLConnection connection = (HttpURLConnection) url.openConnection();
-               connection.setRequestProperty("accept", "application/json");
-               connection.setRequestProperty("X-Api-Key", APIKey);
-
-               BufferedReader br = null;
-               if (connection.getResponseCode() == 200) {
-                   br = new BufferedReader(new InputStreamReader(connection.getInputStream()));
-                   String strCurrentLine;
-                   while ((strCurrentLine = br.readLine()) != null) {
-                       // System.out.println(strCurrentLine);
-                       JSONObject zufälligesEssenJSON = new JSONObject(strCurrentLine);
-                       return zufälligesEssenJSON;
-                   }
-               } else {
-                   br = new BufferedReader(new InputStreamReader(connection.getErrorStream()));
-                   String strCurrentLine;
-                   while ((strCurrentLine = br.readLine()) != null) {
-                       System.out.println(strCurrentLine);
-                   }
-               }
-           } catch (Exception e) {
-               e.printStackTrace();
-           }
-           return null;
-       }
-
-       public JSONObject getEssenÜberName(String essenName){
-           String APIKey = "1";
-
-           try {
-               URL url = new URL("https://www.themealdb.com/api/json/v1/1/search.php?s=" +
    essenName);
-
-               HttpURLConnection connection = (HttpURLConnection) url.openConnection();
-               connection.setRequestProperty("accept", "application/json");
-               connection.setRequestProperty("X-Api-Key", APIKey);
-
-               BufferedReader br = null;
-               if (connection.getResponseCode() == 200) {
-                   br = new BufferedReader(new InputStreamReader(connection.getInputStream()));
```

```
-                String strCurrentLine;
-                while ((strCurrentLine = br.readLine()) != null) {
-                    JSONObject essenJSON = new JSONObject(strCurrentLine);
-                    return essenJSON;
-                }
-            } else {
-                br = new BufferedReader(new InputStreamReader(connection.getErrorStream()));
-                String strCurrentLine;
-                while ((strCurrentLine = br.readLine()) != null) {
-                    System.out.println(strCurrentLine);
-                }
-            }
-        } catch (Exception e) {
-            e.printStackTrace();
-        }
-        return null;
-    }
-
-    public JSONObject getEssenKategorien(){
-        String APIKey = "1";
-
-        try {
-            URL url = new URL("https://www.themealdb.com/api/json/v1/1/categories.php");
-
-            HttpURLConnection connection = (HttpURLConnection) url.openConnection();
-            connection.setRequestProperty("accept", "application/json");
-            connection.setRequestProperty("X-Api-Key", APIKey);
-
-            BufferedReader br = null;
-            if (connection.getResponseCode() == 200) {
-                br = new BufferedReader(new InputStreamReader(connection.getInputStream()));
-                String strCurrentLine;
-                while ((strCurrentLine = br.readLine()) != null) {
-
-                    JSONObject kategorienJson = new JSONObject(strCurrentLine);
-                    return kategorienJson;
-
-                }
-            } else {
-                br = new BufferedReader(new InputStreamReader(connection.getErrorStream()));
-                String strCurrentLine;
-                while ((strCurrentLine = br.readLine()) != null) {
-                    System.out.println(strCurrentLine);
-                }
-            }
-        } catch (Exception e) {
-            e.printStackTrace();
-        }
-        return null;
```

```
-     }
-
-     public JSONObject getEssenÜberId(String idString){
-         String APIKey = "1";
-
-
-
-         try {
-             URL url = new URL("https://www.themealdb.com/api/json/v1/1/lookup.php?i=" +
    idString);
-
-             HttpURLConnection connection = (HttpURLConnection) url.openConnection();
-             connection.setRequestProperty("accept", "application/json");
-             connection.setRequestProperty("X-Api-Key", APIKey);
-
-             BufferedReader br = null;
-             if (connection.getResponseCode() == 200) {
-                 br = new BufferedReader(new InputStreamReader(connection.getInputStream()));
-                 String strCurrentLine;
-                 while ((strCurrentLine = br.readLine()) != null) {
-                     JSONObject essenJSON = new JSONObject(strCurrentLine);
-                     return essenJSON;
-                 }
-             } else {
-                 br = new BufferedReader(new InputStreamReader(connection.getErrorStream()));
-                 String strCurrentLine;
-                 while ((strCurrentLine = br.readLine()) != null) {
-                     System.out.println(strCurrentLine);
-                 }
-             }
-         } catch (Exception e) {
-             e.printStackTrace();
-         }
-         return null;
-     }
-
-     public JSONObject getProductRange(String essensRichtung){
-         String APIKey = "1";
-
-         // www.themealdb.com/api/json/v1/1/filter.php?i=chicken_breast -> ingredient
-         // www.themealdb.com/api/json/v1/1/filter.php?a=Canadian -> Area
-         // www.themealdb.com/api/json/v1/1/lookup.php?i=52772 -> meal details by id
-
-         try {
-             URL url = new URL("https://www.themealdb.com/api/json/v1/1/filter.php?c=" +
    essensRichtung);
-
-             HttpURLConnection connection = (HttpURLConnection) url.openConnection();
-             connection.setRequestProperty("accept", "application/json");
```

```
-            connection.setRequestProperty("X-Api-Key", APIKey);
-
-            BufferedReader br = null;
-            if (connection.getResponseCode() == 200) {
-                br = new BufferedReader(new InputStreamReader(connection.getInputStream()));
-                String strCurrentLine;
-                while ((strCurrentLine = br.readLine()) != null) {
-                    JSONObject essenJSON = new JSONObject(strCurrentLine);
-                    return essenJSON;
-                }
-            } else {
-                br = new BufferedReader(new InputStreamReader(connection.getErrorStream()));
-                String strCurrentLine;
-                while ((strCurrentLine = br.readLine()) != null) {
-                    System.out.println(strCurrentLine);
-                }
-            }
-        } catch (Exception e) {
-            e.printStackTrace();
-        }
-        return null;
-    }
-
-    public JSONArray getProductInfo(String params){
-        String APIKey = "3kTkxnTFVv/5fW3/lKu70A==fHOzKx39l17N3X0C";
-
-
-        // recipe API
-        // https://api.api-ninjas.com/v1/recipe?query=
-
-        try {
-            URL url = new URL("https://api.api-ninjas.com/v1/nutrition?query=" + params);
-            HttpURLConnection connection = (HttpURLConnection) url.openConnection();
-            connection.setRequestProperty("accept", "application/json");
-            connection.setRequestProperty("X-Api-Key", APIKey);
-
-            BufferedReader br = null;
-            if (connection.getResponseCode() == 200) {
-                br = new BufferedReader(new InputStreamReader(connection.getInputStream()));
-                String strCurrentLine;
-                    while ((strCurrentLine = br.readLine()) != null) {
-                        //TODO: remove print
-                        System.out.print(strCurrentLine);
-
-                        JSONArray obj = new JSONArray(strCurrentLine);
-
-                        return obj;
-                    }
-            } else {
```

```
-                 br = new BufferedReader(new InputStreamReader(connection.getErrorStream()));
-                 String strCurrentLine;
-                     while ((strCurrentLine = br.readLine()) != null) {
-                         System.out.println(strCurrentLine);
-                     }
-             }
-         } catch (Exception e) {
-             e.printStackTrace();
-         }
-         return null;
-
-     }
-
- }
```

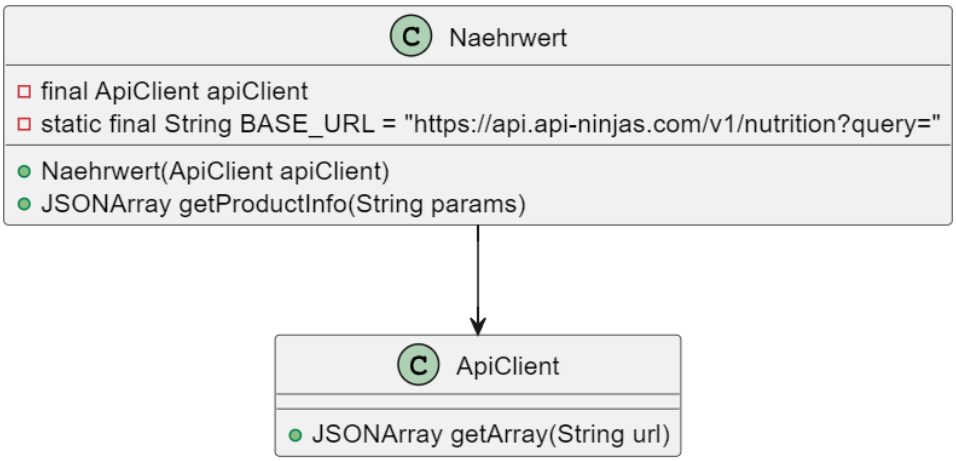
Der Genommene Lösungsweg

- Die Klasse wurde in die Klasse `ApiClient` und `ApiClientImpl` unterteilt und die Aufrufe in die Klassen `EssenService` und `NaehwerteService` ausgelagert, um die Duplikation afzulösen.

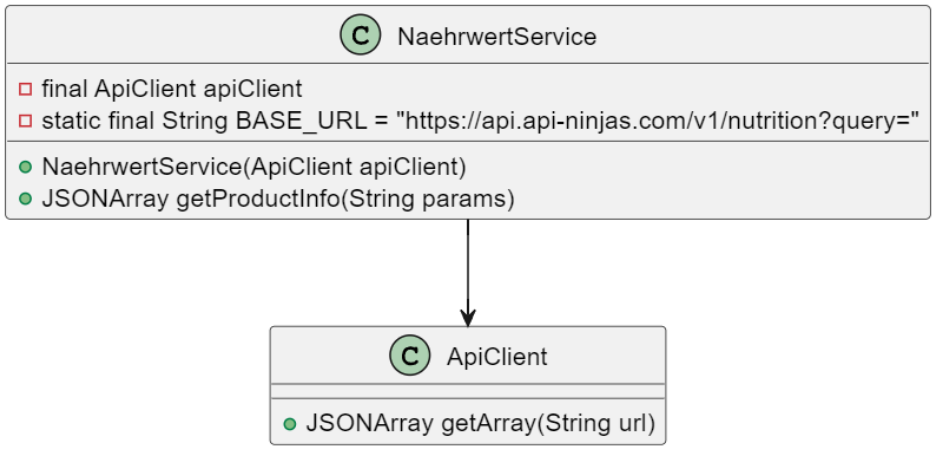
2 Refactorings

Rename Method

- Commit: (1b8568f) https://github.com/WachtelHD/food-picker/commit/1b8568fd7f104140f9f60f6896bcba60b33993f9
- vorher



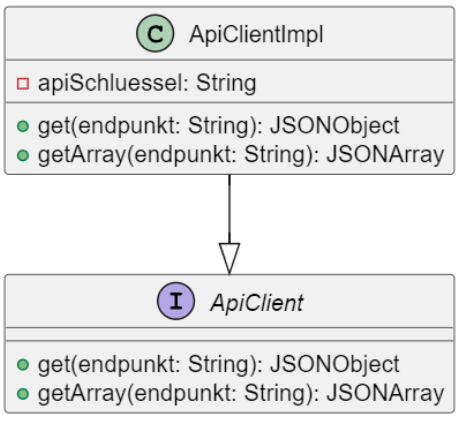
- Nachher



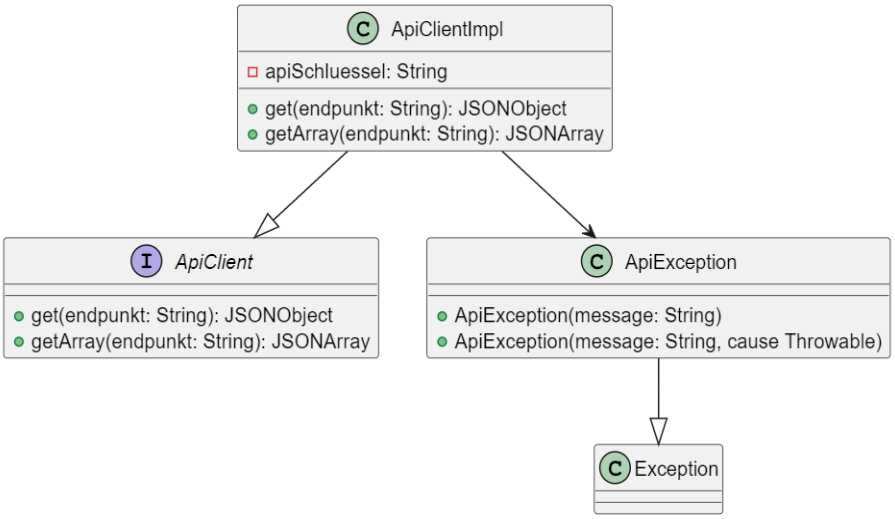
- Begründung:
    - Da die NährwerteService auf ApiClient verweist und einen bestimmten API-Call macht, kann dieser eher als Service eingestuft werden. Dazu kommen auch die weiteren Service-Klassen

Rename Method

- Commit: (8beffc8) https://github.com/WachtelHD/food-picker/commit/8beffc8dc7f836a88c0c9cdf01a4c15c58418a03
- Vorher



- Nachher



- Begründung:
    - Wenn ein Api-Fehler auftrat, dann wurde immer nur die Rückmeldung des Servers geparsed und nicht wirklich behandelt. Mit einer separaten Klasse konnte diese Rückmeldung clean gefangen und behandelt werden.

Kapitel 8: Entwurfsmuster

Entwurfsmuster: [Name]

Entwurfsmuster: [Name]