Programmentwurf
QuizGame

Name: Ketterer, Marvin
Martrikelnummer: 2387051

Abgabedatum: 31.05.2023

Kapitel 1: Einführung (4P)

Übersicht über die Applikation (1P)

[Was macht die Applikation? Wie funktioniert sie? Welches Problem löst sie/welchen Zweck hat sie?]

Bei dieser Applikation handelt es sich um ein konsolenbasiertes Quiz-Spiel. Man kann entweder alleine oder mit bis drei anderen spielen und zwischen drei Schwierigkeitsgraden, welche die Schwierigkeit der Quiz-Fragen bestimmen, wählen. Auch können die Statistiken der Spielen, sowie drei Arten von Highscore-Listen, angesehen werden. Das Spiel wird standardmäßig auf Englisch gestartet, jedoch kann man bei laufender Applikation zwischen Deutsch und Englisch wählen und die Sprache dementsprechend einstellen.

Starten der Applikation (1P)

[Wie startet man die Applikation? Was für Voraussetzungen werden benötigt? Schritt-für-Schritt-Anleitung]

Es muss das .NET 6.0 Framework installiert sein.

Man kann das Spiel über die “QuizGame”-Datei im Verzeichnis “QuizGame\bin\Release\net6.0\linux-x64” starten. (Oder auch für Windows in dem Ordner “QuizGame\bin\Release\net6.0\windows-x64”.)

Beim Ausführen der “Quiz-Game”-Datei öffnet sich der Home-Screen und von dort aus können Spieler hinzugefügt, Statistiken angezeigt, Highscores angezeigt, die Sprache gewählt, die Spiel-Konfiguration geöffnet oder auch die Applikation geschlossen werden.

Technischer Überblick (2P)

[Nennung und Erläuterung der Technologien (z.B. Java, MySQL, …), jeweils Begründung für den Einsatz der Technologien]

.NET 6.0 und C#

Ich habe schon ein wenig Erfahrung mit der Entwicklung in C# mit .NET, was das Entwickeln erleichtert. Zudem ist die Plattformunabhängigkeit ein großer Vorteil. Das .NET-Framework bietet auch eine robuste Laufzeitumgebung, die eine effiziente Ausführung von Konsolenanwendungen ermöglicht. Die Just-in-Time-Kompilierung und der optimierte Code generieren schnelle und performante Anwendungen und die Objektorientiertheit der Programmiersprache C# macht diese gut für das Programmieren dieser Anwendung geeignet.

Kapitel 2: Clean Architecture (8P)

Was ist Clean Architecture? (1P)

[allgemeine Beschreibung der Clean Architecture in eigenen Worten]

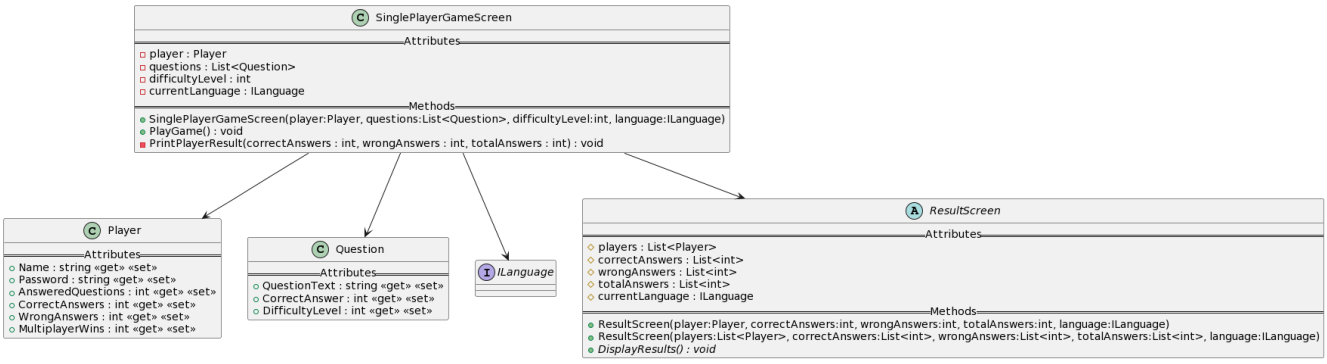
Clean Architecture ist ein Architekturmuster für Software, welches dazu dient, die Abhängigkeiten zwischen den verschiedenen Schichten einer Anwendung zu minimieren. Die Schichten sind Zwiebel-artig aufgebaut und deren Abhängigkeiten sind von außen nach innen gerichtet. Das ermöglicht es, eine sehr modulare und gut testbare Anwendung zu erstellen.

Analyse der Dependency Rule (3P)

[1 Klasse, die die Dependency Rule einhält und 1 Klasse, die die Dependency Rule verletzt; jeweils UML (mind. die betreffende Klasse inkl. der Klassen, die von ihr abhängen bzw. von der sie abhängt) und Analyse der Abhängigkeiten in beide Richtungen (d.h., von wem hängt die Klasse ab und wer hängt von der Klasse ab) in Bezug auf die Dependency Rule]

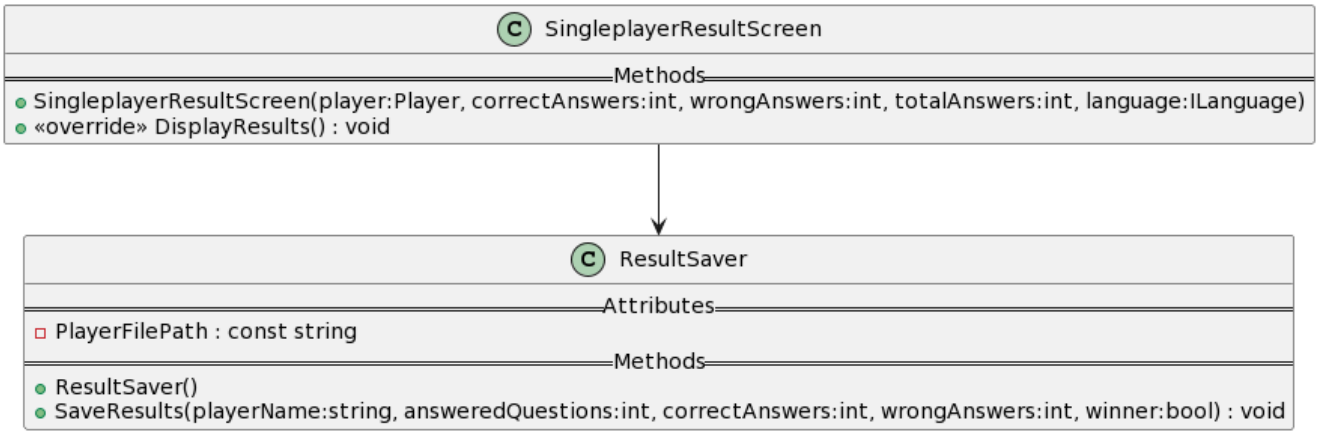
Positiv-Beispiel: Dependency Rule

In dem gegebenen Codebeispiel hält die Klasse SinglePlayerGameScreen die Dependency Rule ein. Diese Klasse ist von anderen Abhängigkeiten wie Player, Question, ILanguage und ResultScreen abhängig, jedoch werden diese Abhängigkeiten über die Konstruktorinjektion bereitgestellt. Dadurch wird die Abhängigkeit von konkreten Implementierungen gelöst und ermöglicht eine flexible und austauschbare Komponentenstruktur.



Negativ-Beispiel: Dependency Rule

Eine Klasse, die die Dependency Rule verletzt, ist die Klasse ResultSaver. Diese Klasse wird innerhalb der SingleplayerResultScreen-Klasse instanziiert, was zu einer starken Kopplung führt. Die SingleplayerResultScreen-Klasse sollte jedoch nicht direkt von der konkreten Implementierung ResultSaver abhängig sein, sondern von einer abstrakten Schnittstelle oder einem Interface, um die Abhängigkeiten zu entkoppeln und die Austauschbarkeit zu ermöglichen.

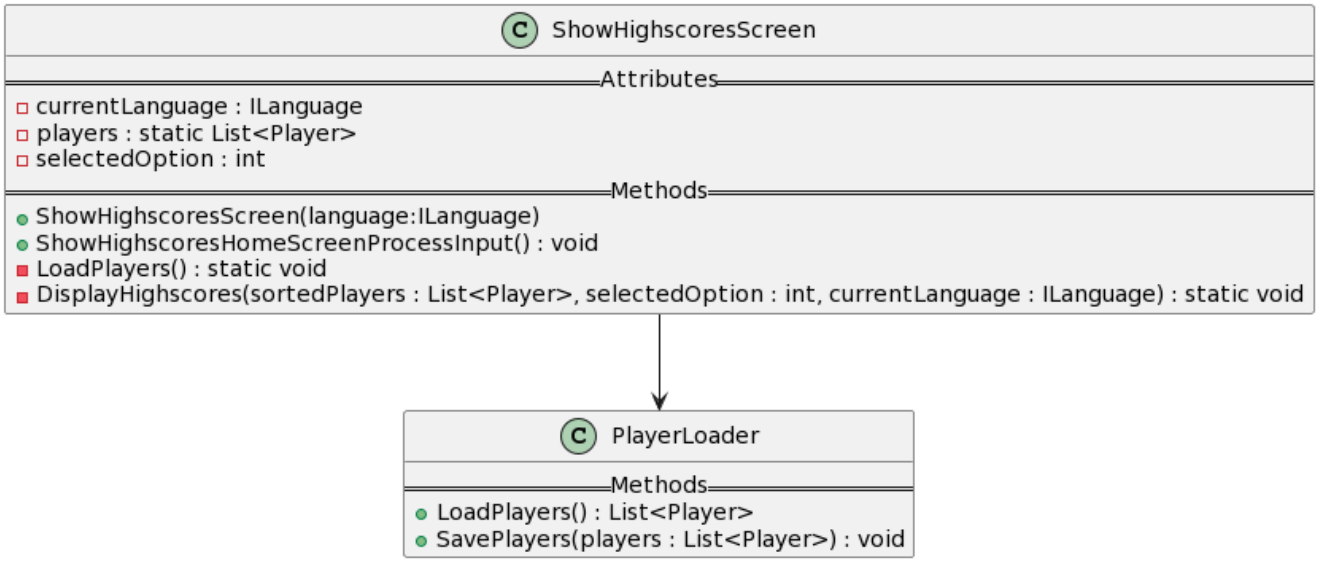


Analyse der Schichten (4P)

[jeweils 1 Klasse zu 2 unterschiedlichen Schichten der Clean-Architecture: jeweils UML (mind. betreffende Klasse und ggf. auch zusammenspielenden Klassen), Beschreibung der Aufgabe, Einordnung mit Begründung in die Clean-Architecture]

Schicht: Screens

Screens beschreibt die äußere Schicht, welche unter anderem dem Benutzer bzw. den Benutzern seine bzw. ihre aktuellen Interaktionsmöglichkeiten mit der Applikation anzeigt und Benutzereingaben entgegennimmt und weiterverarbeitet. Bsp.: ShowHighscoresScreen

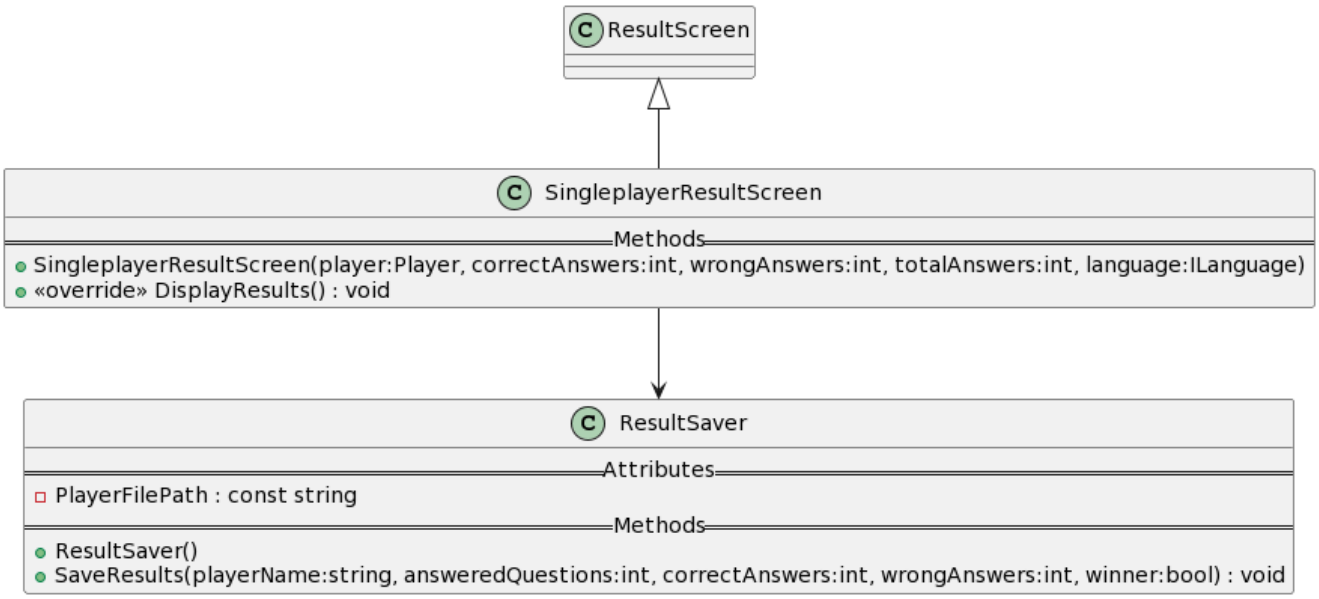


Diese Klasse ist lediglich dazu da, die Auswahl der Highscore-Kategorien aufzuzeigen und die gewählte Highscore-Liste anzuzeigen. Somit regelt die Klasse das Anzeigen von Informationen für den

Benutzer und die nimmt dessen Eingabe entgegen. Die Methode LoadPlayers, welche die Spielerliste zum einordnen in die Highscore-Liste liefert, ruft dafür die Klasse PlayerLoader in der BackgroundLogic-Schicht auf, da dies eine Hintergrund-Logik ist und nicht in ihren Zuständigkeitsbereich gehört. Der PlayerLoader wird in ShowHighscoresScreen initialisiert und instanziiert, wodurch eine Abhängigkeit des PlayerLoaders gegenüber des ShowHighscoresScreens entsteht, also eine Abhängigkeit von “außen” nach “innen”.

Schicht: BackgroudLogic

BackgroundLogic ist die innere Schicht, welche die internen Prozesse, wie bspw. das Breitstellen der Quizfragen (Question), das Mischen der Quizfragen usw. verantwortlich ist. Bsp.: ResultSaver



Der ResultSaver wird bspw. in SinglePlayerResultScreen initialisiert und instanziiert und ist somit von SinglePlayerResultScreen, einer Klasse der äußeren Schicht, abhängig. Er ist dazu da, die Ergebnisse eines Spiels nach dessen Abschluss zu speichern, indem SinglePlayerResultScreen dessen Methode SaveResults verwendet.

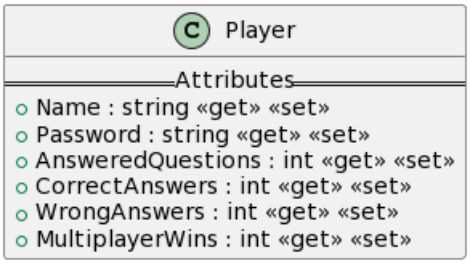
Kapitel 3: SOLID (8P)

Analyse SRP (3P)

[jeweils eine Klasse als positives und negatives Beispiel für SRP; jeweils UML und Beschreibung der Aufgabe bzw. der Aufgaben und möglicher Lösungsweg des Negativ-Beispiels (inkl. UML)]

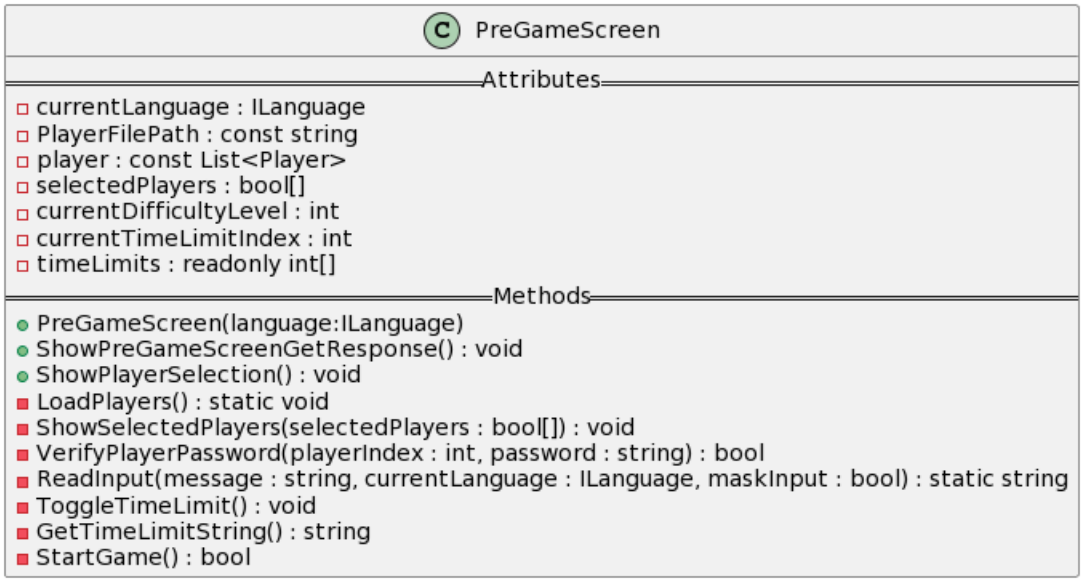
Positiv-Beispiel

Eine Klasse, in der das SRP angewendet wird, ist die Klasse Player. Diese Klasse ist dafür verantwortlich, Informationen über einen Spieler zu speichern. Dazu zählen der Name, das Passwort und Statistiken, wie die Anzahl beantwortete Fragen, die Anzahl korrekte Antworten, die Anzahl falscher Antworten und die Anzahl der Siege. Somit hat die Klasse hat eine einzige Verantwortung, nämlich das „Halten“ der Spielerdaten.

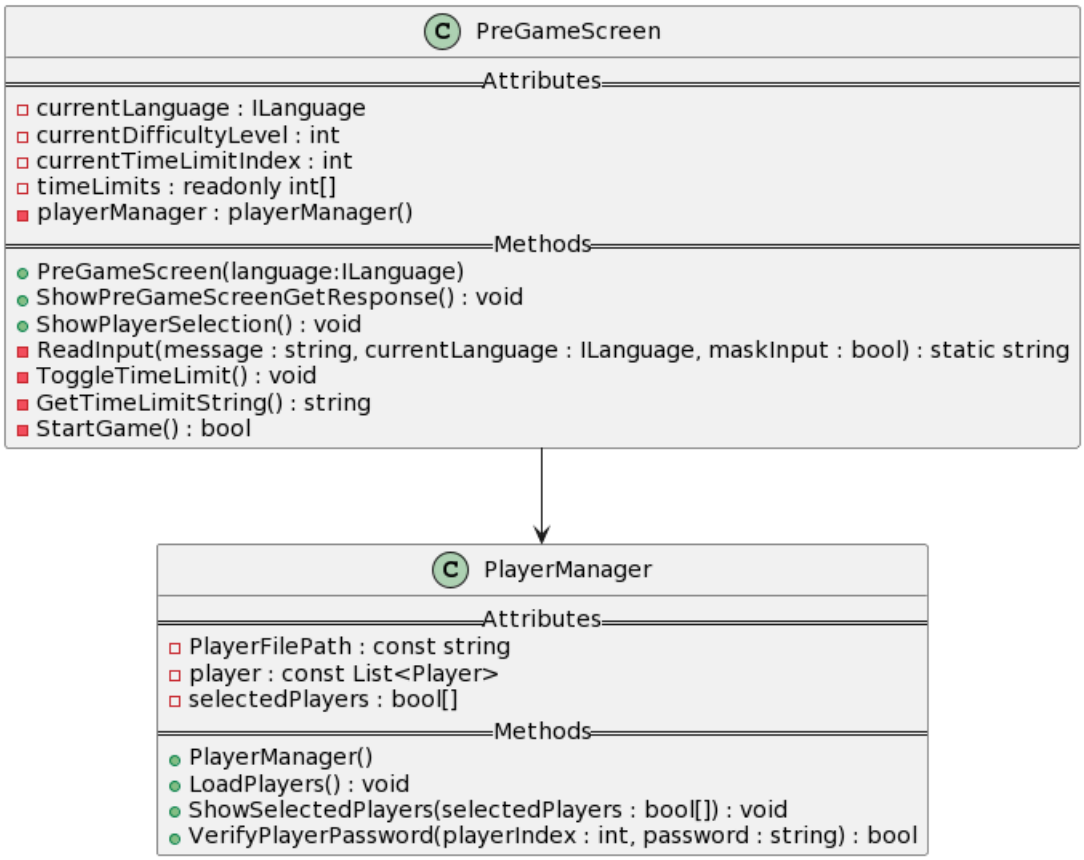


Negativ-Beispiel

Die Klasse PreGameScreen verletzt das SRP, da sie mehrere Verantwortlichkeiten hat, die unabhängig voneinander geändert werden könnten. Sie verwaltet die Benutzeroberfläche, indem sie dem Benutzer Informationen zum Konfigurieren eines Spiels anzeigt und darauf folgende Eingaben verarbeitet.
Die Klasse verwaltet jedoch auch die Spielerdaten, das Laden der Spieler aus der players.csv-Datei und das Überprüfen von Passwörtern. Das könnte durch bspw. Die Implementierung einer Klasse wie z.B. „PlayerManager“ verbessert werden, welche diese Logik auslagert. Somit würde die Klasse lediglich als Benutzerschnittstelle dienen.



Mögliche Lösung:



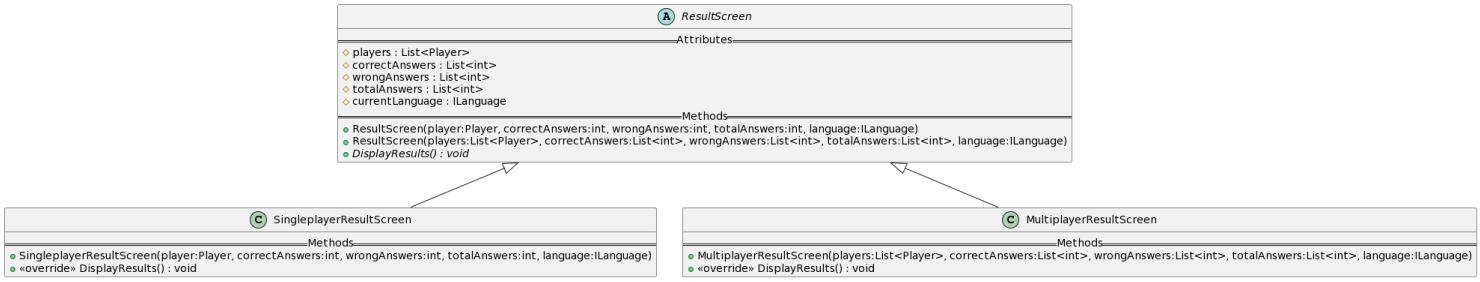
Analyse OCP (3P)

[jeweils eine Klasse als positives und negatives Beispiel für OCP; jeweils UML und Analyse mit Begründung, warum das OCP erfüllt/nicht erfüllt wurde – falls erfüllt: warum hier sinnvoll/welches Problem gab es? Falls nicht erfüllt: wie könnte man es lösen (inkl. UML)?]

Positiv-Beispiel

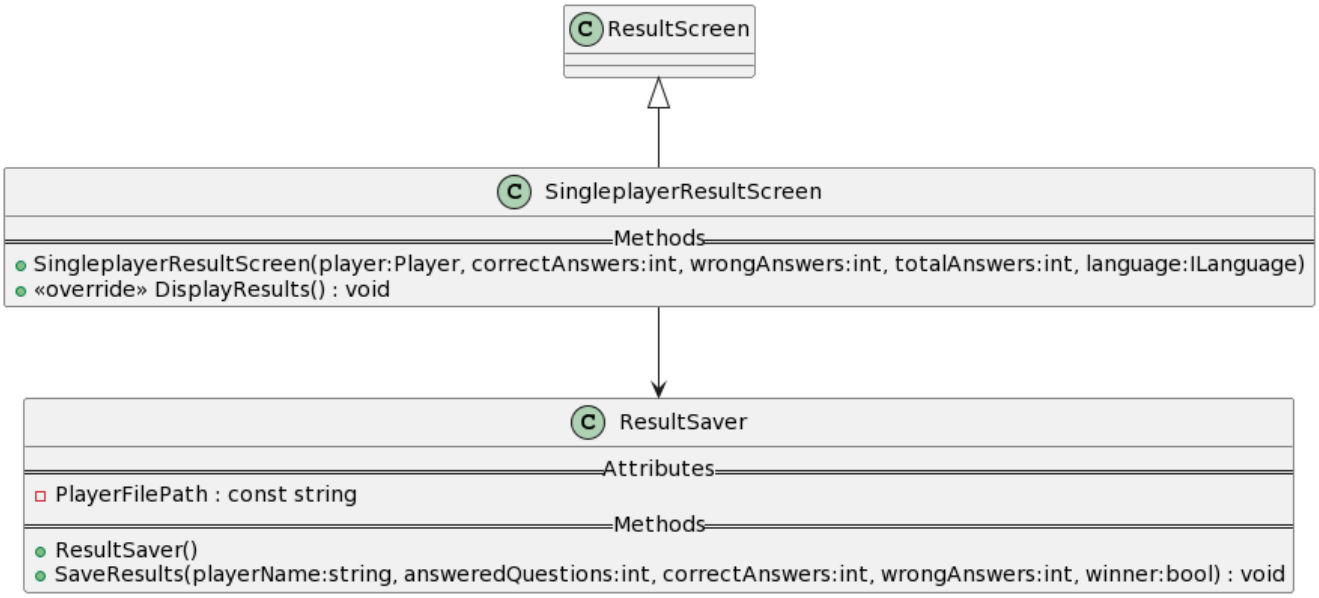
ResultScreen ist eine abstrakte Basisklasse und definiert die abstrakte Methode DisplayResults, die von der abgeleiteten Klasse SinglePlayerResultScreen implementiert wird, welche für die Darstellung des Spielergebnisses von Singleplayer-Spielen dient.

Dadurch wird der bestehende Code der SingleplayerResultScreen-Klasse nicht verändert, wenn beispielsweise eine andere Art von ResultScreen für den Singleplayer-Modus erstellt werden muss (bspw. beim einfügen eines neuen Spielmodus, welcher eine andere Darstellung des Ergebnisses fordern. Dies ermöglicht auch, dass MultiplayereResultScreen davon erbt). Stattdessen wird der Code um eine Klasse erweitert, indem eine neue Klasse, mit ResultScreen als abstrakte Basisklasse, eingeführt wird. Die neue Art von Benutzerschnittstelle kann somit erstellt werden, indem neue Klassen von der abstrakten Basisklasse erben und die abstrakte Methode DisplayResults implementieren.
Durch dies bleibt der bestehende Code unverändert und geschlossen für Modifikationen, jedoch offen für Erweiterungen.



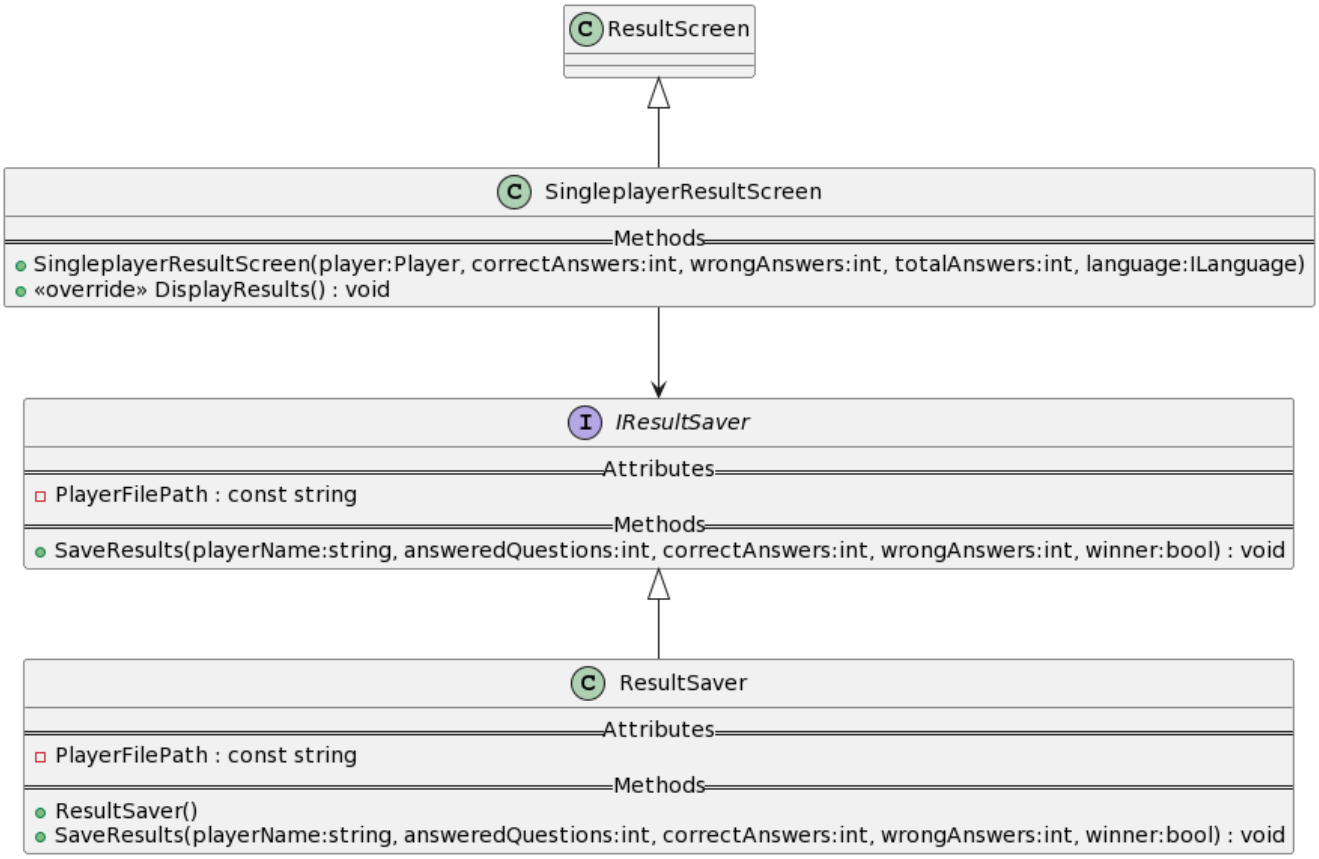
Negativ-Beispiel

Ein negatives Beispiel für das Open-Closed-Prinzip befindet sich in der Klasse ResultSaver. Die Methode SaveResults verwendet eine spezifische Implementierung zum Lesen und Schreiben von Spielerdaten aus der players.csv-Datei. Der Dateipfad zu dieser Datei ist hartkodiert. Wenn sich das Speicherformat oder die Speicherquelle ändern würde, müsste der Code der ResultSaver-Klasse geändert werden, um diese Änderungen zu berücksichtigen. Das verstößt gegen das Open-Closed-Prinzip, da der Code geändert werden müsste, anstatt die Funktionalität durch Hinzufügen neuer Klassen oder Schnittstellen zu erweitern.



Um das Open-Closed-Prinzipauf die ResultSaver-Klasse anzuwenden, könnte man eine abstrakte Basisklasse oder ein Interface einführen, das die Funktionen zum Speichern der Ergebnisse definiert. Anschließend könnten spezifische Implementierungen dieser Basisklasse oder dieses Interfaces erstellt werden, um die Ergebnisse in unterschiedlichen Formaten oder Speicherquellen zu herauszulesen und zu speichern. Auf diese Weise wäre der die Klasse flexibler und geschlossen für Modifikationen, da neue Speicherformate oder -quellen einfach hinzugefügt werden können, indem neue Klassen erstellt werden, die von der Basisklasse erben und die abstrakte Methode SaveResults implementieren.

Mögliche Lösung:



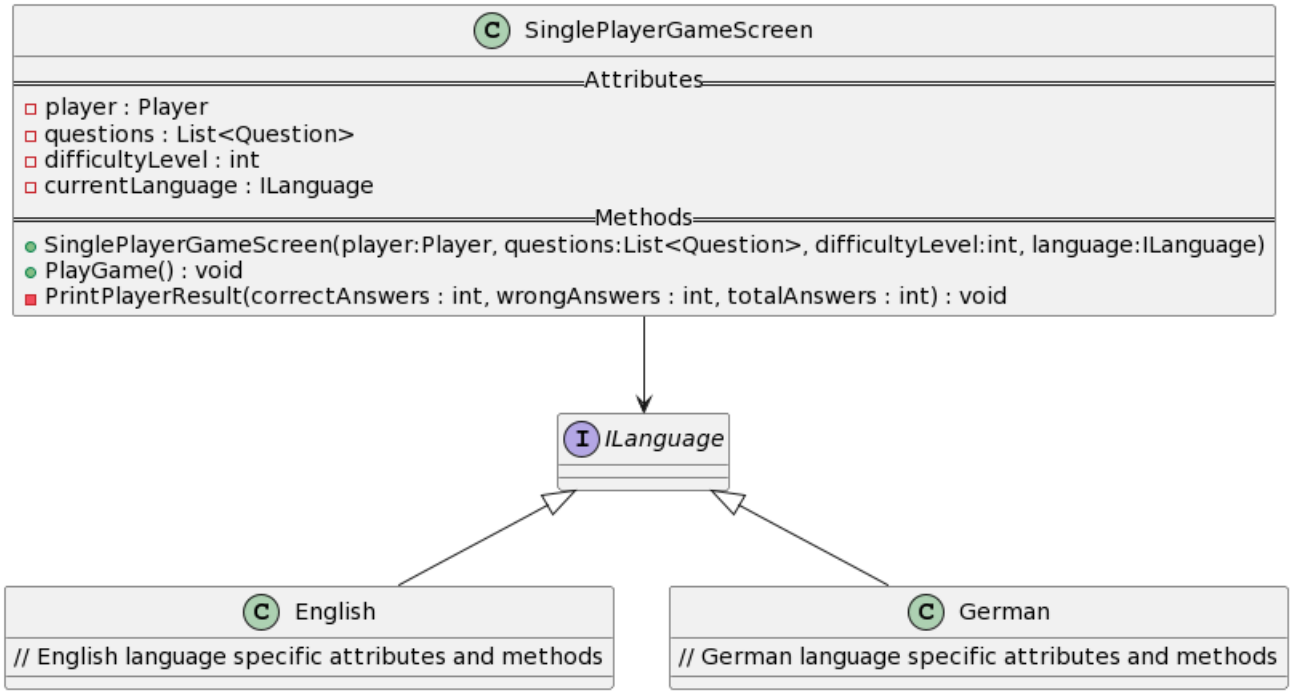
Analyse [LSP/ISP/DIP] (2P)

[jeweils eine Klasse als positives und negatives Beispiel für entweder LSP oder ISP oder DIP; jeweils UML und Begründung, warum hier das Prinzip erfüllt/nicht erfüllt wird; beim Negativ-Beispiel UML einer möglichen Lösung hinzufügen]

[Anm.: es darf nur ein Prinzip ausgewählt werden; es darf NICHT z.B. ein positives Beispiel für LSP und ein negatives Beispiel für ISP genommen werden]

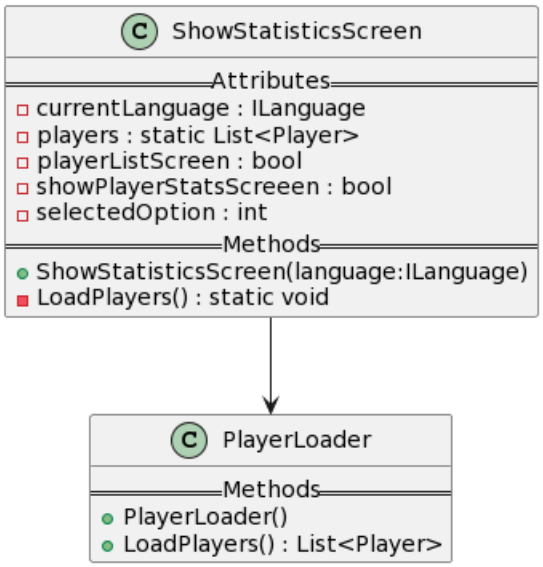
Positiv-Beispiel: Liskov Substitution Principle

Eine Klasse, die das Liskov Substitution Principle einhält, ist die Klasse SinglePlayerGameScreen. Sie erbt nicht direkt von einer Basisklasse, sondern verwendet die Schnittstelle ILanguage. Dadurch kann sie problemlos mit verschiedenen Implementierungen der Schnittstelle arbeiten, ohne den Code zu ändern.



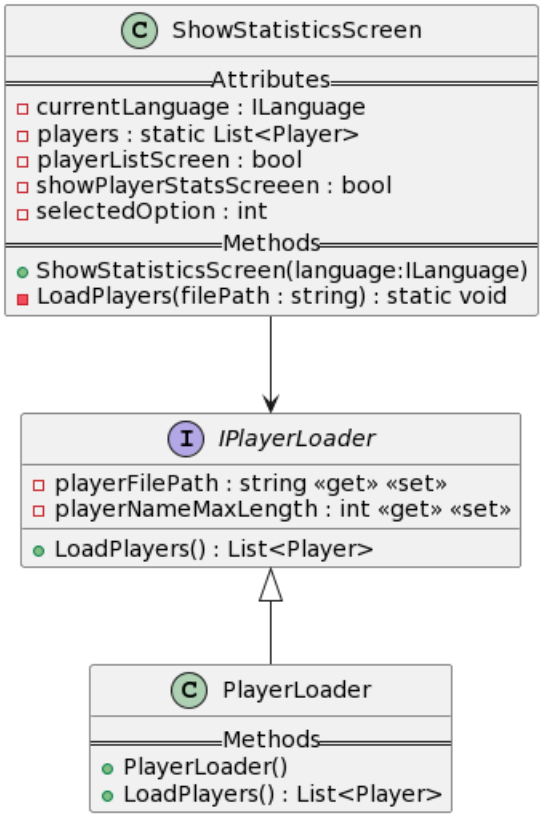
Negativ-Beispiel

Ein negatives Beispiel für das Liskov Substitution Principle ist die Klasse PlayerLoader. In der Methode LoadPlayers wird eine spezifische Implementierung verwendet, um Spielerdaten aus einer CSV-Datei zu laden. Wenn sich das Datenformat oder die Datenquelle ändern würden, müsste der Code der Klasse geändert werden, um diese Änderungen zu berücksichtigen. Es das Liskov Substitution Principle verletzt, da die Klasse nicht problemlos durch andere Implementierungen ersetzt werden kann, ohne den Code anzupassen.



Dies könnte gelöst werden, indem die Klasse ShowStatisticsScreen eine abstrakte Schnittstelle oder eine Basisklasse von PlayerLoader verwenden würde, anstatt eine spezifische Implementierung direkt zu verwenden.

Mögliche Lösung:

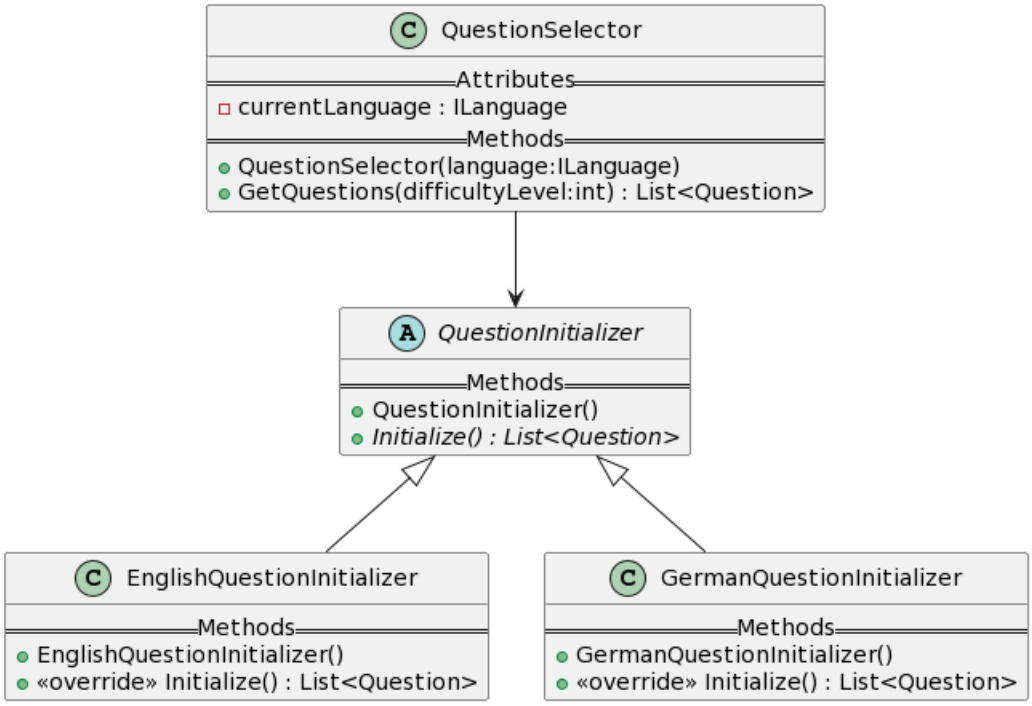


Kapitel 4: Weitere Prinzipien (8P)

Analyse GRASP: Geringe Kopplung (3P)

*[eine **bis jetzt noch nicht behandelte** Klasse als positives Beispiel geringer Kopplung; UML mit zusammenspielenden Klassen, Aufgabenbeschreibung der Klasse und Begründung, warum hier eine geringe Kopplung vorliegt]*

Eine Klasse, die das GRASP-Prinzip „Geringe Kopplung“ umsetzt, ist die Klasse EnglishQuestionInitializer. Diese Klasse ist für die Erstellung und Bereitstellung von Fragen in englischer Sprache zuständig und implementiert die abstrakte Klasse QuestionInitializer. Sie hat nur diese spezifische Verantwortung und ist unabhängig von anderen Klassen im Code. Dadurch kann die Klasse QuestionSelector, je nachdem, ob die Sprache Deutsch oder Englisch gewählt wurde, die passende, von der Basisklasse QuestionInitializer abgeleitete, Instanz erstellen und benutzen. Somit besteht eine geringe Kopplung.



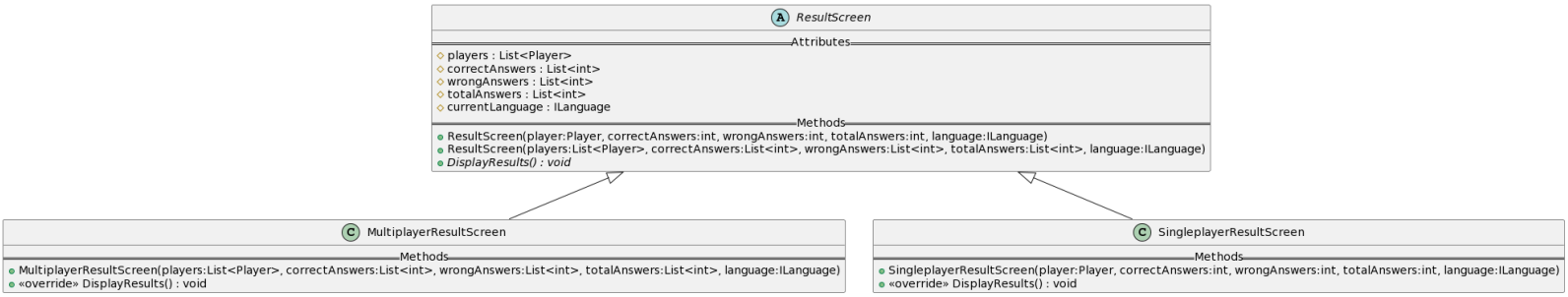
Analyse GRASP: [Polymorphismus/Pure Fabrication] (3P)

[eine Klasse als positives Beispiel entweder von Polymorphismus oder von Pure Fabrication; UML Diagramm und Begründung, warum es hier zum Einsatz kommt]

Polymorphismus:

In der Klasse ResultScreen wird die Methode DisplayResults als abstrakte Methode deklariert, die in den abgeleiteten Klassen implementiert wird. Dies ermöglicht es verschiedenen Klassen, welche das Ergebnis anzeigen, unterschiedliche Implementierungen der DisplayResults-Methode zu haben, während sie den gleichen Namen verwenden. Zudem enthält sie zwei Konstruktoren, was als Overload bezeichnet wird. Dies ermöglicht es, dass sowohl SingleplayerResultScreen als auch MultiplayerResultScreen darauf zugreiffen können.

Die Klasse SingleplayerResultScreen erbt von der abstrakten Klasse ResultScreen und überschreibt die DisplayResults-Methode mit ihrer eigenen Implementierung. Dadurch wird die Polymorphie ermöglicht, da die Methode je nach Instanz des ResultScreens unterschiedlich ausgeführt wird.
Hier wird der Polymorphismus genutzt, indem eine Instanz der abstrakten Klasse ResultScreen erstellt wird, aber tatsächlich die DisplayResults-Methode der abgeleiteten Klasse SingleplayerResultScreen aufgerufen wird, die überschrieben wurde.



DRY (2P)

[ein Commit angeben, bei dem duplizierter Code/duplizierte Logik aufgelöst wurde; Code-Beispiele (vorher/nachher) einfügen; begründen und Auswirkung beschreiben – ggf. UML zum Verständnis ergänzen]

Die Klassen ShowStatisticsScreen und ShowHighscoresScreen enthielten beide eine Methode, LoadPlayers, um die Player aus der players.csv-Datei zu laden.:

```
private const string PlayerFilePath = "players.csv";
7 references
private static List<Player> players;
```

Dies wurde mit der Klasse PlayerLoader zusammengefügt, sodass der Code sich in beiden Klassen nicht wiederholt und somit in beiden Klassen nur die Methode LoadPlayers des PlayerLoader aufgerufen wird.

```
1 reference
private static void LoadPlayers()
{
    players = new List<Player>();

    if (File.Exists(PlayerFilePath))
    {
        string[] lines = File.ReadAllLines(PlayerFilePath);
        foreach (string line in lines)
        {
            string[] parts = line.Split(',');
            Player player = new Player
            {
                Name = parts[0],
                Password = parts[1],
                AnsweredQuestions = int.Parse(parts[2]),
                CorrectAnswers = int.Parse(parts[3]),
                WrongAnswers = int.Parse(parts[4]),
                MultiplayerWins = int.Parse(parts[5])
            };
            players.Add(player);
        }
    }
}
```

Die neue Klasse PlayerLoader:

```
4 references
public class PlayerLoader
{
    2 references
    private const string PlayerFilePath = "players.csv";
    2 references
    public PlayerLoader()
    {

    }

    2 references
    public List<Player> LoadPlayers()
    {
        List<Player> players = new List<Player>();

        if (File.Exists(PlayerFilePath))
        {
            string[] lines = File.ReadAllLines(PlayerFilePath);
            foreach (string line in lines)
            {
                string[] parts = line.Split(',');
                Player player = new Player
                {
                    Name = parts[0],
                    Password = parts[1],
                    AnsweredQuestions = int.Parse(parts[2]),
                    CorrectAnswers = int.Parse(parts[3]),
                    WrongAnswers = int.Parse(parts[4]),
                    MultiplayerWins = int.Parse(parts[5])
                };
                players.Add(player);
            }
        }
        return players;
    }
```

Der Aufruf in den Klassen ShowStatisticsScreen und ShowHighscoresScreen:

```
1 reference
private static void LoadPlayers()
{
    PlayerLoader playerLoader = new PlayerLoader();
    players = playerLoader.LoadPlayers();
}
```

Commit:

https://github.com/BeardedCoffee/QuizGame/commit/1fcd88aba82fa1a9462a6c9731c38b9912791c6b

Kapitel 5: Unit Tests (8P)

10 Unit Tests (2P)

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

Unit Test	Beschreibung
TestEnglishQuestionInitializer	Überprüft, ob der EnglishQuestionInitializer eine Liste von Fragen initialisiert und stellt sicher, dass die Liste nicht leer ist.
TestGermanQuestionInitializer	Überprüft, ob der GermanQuestionInitializer eine Liste von Fragen initialisiert und stellt sicher, dass die Liste nicht leer ist.
TestAddPlayer	Fügt einen Testspieler hinzu und überprüft, ob die Spielerdaten korrekt in die CSV-Datei geschrieben wurden und stellt sicher, dass die letzte Zeile der CSV-Datei den erwarteten Spielerdaten entspricht.
TestLoadPlayers	Lädt Spieler aus einer Testspielerdatei und überprüft, ob die Spielerdaten korrekt geladen wurden und überprüft, ob die Anzahl der geladenen Spieler korrekt ist und ob ihre Eigenschaften den erwarteten Werten entsprechen.
TestGetQuestions	Ruft Fragen für einen bestimmten Schwierigkeitsgrad ab und überprüft, ob die zurückgegebene Liste nicht null oder leer ist.
TestGetSelectedQuestions_Valid	Überprüft, ob die ausgewählte Sprache basierend auf einer gültigen Option korrekt zurückgegeben wird und stellt sicher, dass die zurückgegebene Sprache vom erwarteten Typ ist.
TestGetSelectedQuestions_Invalid	Überprüft, ob null zurückgegeben wird, wenn eine ungültige Option ausgewählt wird.
TestGetAllQuestionsAndSelect	Überprüft, ob alle Fragen abgerufen und eine Frage ausgewählt wird und stellt sicher, dass die zurückgegebene Liste nicht null oder leer ist.
TestShuffle	Überprüft, ob die Reihenfolge der Fragen nach dem Mischen geändert wurde.
TestSaveResults	Speichert die Ergebnisse eines Spiels und überprüft, ob die Spielerdaten in der CSV-Datei korrekt aktualisiert wurden und überprüft, ob die gespeicherten Werte den erwarteten Werten entsprechen.

ATRIP: Automatic (1P)

[Begründung/Erläuterung, wie ‘Automatic’ realisiert wurde]

Automatic wurde mittels des Github Actions Plugins gelöst, welches den automatisch nach dem Pushen von Code auf den master-Branch das Builden und Testen des Codes ausgeführt und eine E-Mail sendet, wenn etwas fehlgeschlagen ist. Hierzu wurde die dotnet-desktop.yml-Datei aufgesetzt.

ATRIP: Thorough (1P)

[Code Coverage im Projekt analysieren und begründen]

ATRIP: Professional (1P)

[1 positves Beispiel zu 'Professional'; Code-Beispiel, Analyse und Begründung, was professionell ist]

Fakes und Mocks (3P)

[Analyse und Begründung des Einsatzes von 2 Fake/Mock-Objekten (die Fake/Mocks sind ohne Dritthersteller-Bibliothek/Framework zu implementieren); zusätzlich jeweils UML Diagramm mit Beziehungen zwischen Mock, zu mockender Klasse und Aufrufer des Mocks]

Fake 1:

Im Test TestEnglishQuestionInitializer wird das QuestionInitializer-Objekt mit der konkreten Implementierung EnglishQuestionInitializer initialisiert. Dies ermöglicht es, die Initialisierung von englischen Fragen zu testen, ohne von der tatsächlichen Implementierung abhängig zu sein. Dadurch wird eine Art Fake-Objekt verwendet, um die Funktionalität des Initialisierers zu testen.

Fake 2:

Kapitel 6: Domain Driven Design (8P)

Ubiquitous Language (2P)

[4 Beispiele für die Ubiquitous Language; jeweils Bezeichung, Bedeutung und kurze Begründung, warum es zur Ubiquitous Language gehört]

Bezeichung	**Bedeutung**	**Begründung**
ILanguage	Diese Klasse repräsentiert eine Schnittstelle für die Sprachunterstützung im Quizspiel. Sie enthält Eigenschaften, um die Benutzeroberflächentexte zu verwalten.	Da die Sprache im Kontext von Spielen, außer wenn sich das Spiel selbst um Sprachen dreht, ist in so gut wie jedem Spiel auswählbar ist und siedie Sprache repräsentiert, in welcher die Texte angezeigt oder gesprochen wird.
Player	Diese Klasse repräsentiert einen Spieler im Quizspiel und hält dessen Attribute, die für die individuellen Spieler wichtig sind.	Die Klasse Player kann man im Kontext von Spielen allgemein als Werte-Träger von Spielern ansehen, da alle Spieler gewisse Attribute beinhalten.
Question	Die Klasse repräsentiert die Quiz-Fragen im Spiel und hält deren Attribute, die für die individuellen Fragen wichtig sind.	Im Kontext eines Quiz-Spiels ist die Klasse Question eindeutig als Repräsentante von den Quiz-Fragen zu erkennen.
PlayerLoader	Diese Klasse ist für das instanziieren der Player-Objekte zuständig, welche sich in der players.csv-Datei befinden.	Die Klasse PlayerLoader hat auch einen selbsterklärenden Namen und ist eindeutig ihrer Funktion im Spiel zuordenbar.

Repositories (1,5P)

[UML, Beschreibung und Begründung des Einsatzes eines Repositories; falls kein Repository vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist – NICHT, warum es nicht implementiert wurde]

Aggregates (1,5P)

[UML, Beschreibung und Begründung des Einsatzes eines Aggregates; falls kein Aggregate vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist– NICHT, warum es nicht implementiert wurde]

Entities (1,5P)

[UML, Beschreibung und Begründung des Einsatzes einer Entity; falls keine Entity vorhanden: ausführliche Begründung, warum es keine geben kann/hier nicht sinnvoll ist– NICHT, warum es nicht implementiert wurde]

Value Objects (1,5P)

[UML, Beschreibung und Begründung des Einsatzes eines Value Objects; falls kein Value Object vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist– NICHT, warum es nicht implementiert wurde]

Kapitel 7: Refactoring (8P)

Code Smells (2P)

[jeweils 1 Code-Beispiel zu 2 unterschiedlichen Code Smells aus der Vorlesung; jeweils Code-Beispiel und einen möglichen Lösungsweg bzw. den genommen Lösungsweg beschreiben (inkl. (Pseudo-)Code)]

CODE SMELL 1: Duplicated Code

In PreGameScreen:

```
1 reference
private static void LoadPlayers()
{
    if (File.Exists(PlayerFilePath))
    {
        string[] lines = File.ReadAllLines(PlayerFilePath);
        foreach (string line in lines)
        {
            string[] parts = line.Split(',');
            Player player = new Player
            {
                Name = parts[0],
                Password = parts[1],
                AnsweredQuestions = int.Parse(parts[2]),
                CorrectAnswers = int.Parse(parts[3]),
                WrongAnswers = int.Parse(parts[4]),
                MultiplayerWins = int.Parse(parts[5])
            };
            players.Add(player);
        }
    }
}
```

In PlayerLoader:

```
3 references
public List<Player> LoadPlayers()
{
    List<Player> players = new List<Player>();

    if (File.Exists(PlayerFilePath))
    {
        string[] lines = File.ReadAllLines(PlayerFilePath);
        foreach (string line in lines)
        {
            string[] parts = line.Split(',');
            Player player = new Player
            {
                Name = parts[0],
                Password = parts[1],
                AnsweredQuestions = int.Parse(parts[2]),
                CorrectAnswers = int.Parse(parts[3]),
                WrongAnswers = int.Parse(parts[4]),
                MultiplayerWins = int.Parse(parts[5])
            };
            players.Add(player);
        }
    }
    return players;
}
```

Das könnte einfach gelöst werden, indem man die Methode LoadPlayers in PreGameScreen umändert, wie in ShowHighscoreScreen, sodass die Code-Redundanz beseitigt wird.:

```
1 reference
private static void LoadPlayers()
{
    PlayerLoader playerLoader = new PlayerLoader();
    players = playerLoader.LoadPlayers();
}
```

CODE SMELL 2: Long Method

Die Methode ShowStatisticsHomeScreenProcessInput in ShowStatisticsScreen ist extrem lang.:

```
1 reference
private void ShowStatisticsHomeScreenProcessInput()
{
    playerListScreen = true;
    showPlayerStatsScreen = false;

    while (true)
    {
        Console.Clear();
        Console.WriteLine(currentLanguage.PlayerStats);
        Console.WriteLine("-------------------");
        Console.WriteLine(currentLanguage.BackWithHashtag);
        Console.WriteLine(currentLanguage.ChoosePlayer);

        ConsoleKeyInfo keyInfo;

        do
        {
            if (playerListScreen)
            {
                Console.Clear();
                Console.WriteLine(currentLanguage.PlayerStats);
                Console.WriteLine("--------------------------");

                for (int i = 0; i < players.Count; i++)
                {
                    if (i == selectedOption)
                    {
                        Console.Write("-> ");
                        Console.ForegroundColor = ConsoleColor.Green;
                        Console.WriteLine(players[i].Name);
                        Console.ResetColor();
                    }
                    else
                    {
                        Console.WriteLine("   " + players[i].Name);
```

```
            }
        }
    }
    else if (showPlayerStatsScreen)
    {
        Console.Clear();
        Console.WriteLine(currentLanguage.PlayerStats);
        Console.WriteLine("--------------------");
        Player selectedPlayer = players[selectedOption];
        Console.WriteLine($"{currentLanguage.PlayerName} {selectedPlayer.Name}");
        Console.WriteLine($"{currentLanguage.AnsweredQuestions} {selectedPlayer.AnsweredQuestions}");
        Console.WriteLine($"{currentLanguage.CorrectAnswers} {selectedPlayer.CorrectAnswers}");
        Console.WriteLine($"{currentLanguage.WrongAnswers} {selectedPlayer.WrongAnswers}");
        Console.WriteLine($"{currentLanguage.MultiplayerWins} {selectedPlayer.MultiplayerWins}");
        Console.WriteLine("\n-----------------------------------");
        Console.WriteLine(currentLanguage.PressButtonToContinue);
        Console.ReadKey(intercept: true);

        playerListScreen = true;
        showPlayerStatsScreen = false;
        break;
    }

    keyInfo = Console.ReadKey(intercept: true);

    if (keyInfo.Key == ConsoleKey.Escape)
    {
        Console.WriteLine(currentLanguage.CanceledByUser);
        return;
    }
    else if (keyInfo.KeyChar == '#')
    {
        if (playerListScreen)
        {
            new HomeScreen(currentLanguage);
            return;
```

```
            }
            else if (showPlayerStatsScreen)
            {
                playerListScreen = true;
                showPlayerStatsScreen = false;
            }
        }
        else if (keyInfo.Key == ConsoleKey.UpArrow && selectedOption > 0)
        {
            selectedOption--;
        }
        else if (keyInfo.Key == ConsoleKey.DownArrow && selectedOption < players.Count - 1)
        {
            selectedOption++;
        }
        else if (keyInfo.Key == ConsoleKey.Enter)
        {
            if (playerListScreen)
            {
                playerListScreen = false;
                showPlayerStatsScreen = true;
            }
        }
    } while (true);
}
}
```

Man könnte den Code hier in kleinere Methoden runterbrechen und ihn somit leserlicher machen:

```
private void ShowStatisticsHomeScreenProcessInput()
{
    playerListScreen = true;
    showPlayerStatsScreen = false;

    while (true)
    {
        Console.Clear();
        Console.WriteLine(currentLanguage.PlayerStats);
        Console.WriteLine("--------------------");
        Console.WriteLine(currentLanguage.BackWithHashtag);
        Console.WriteLine(currentLanguage.ChoosePlayer);

        ConsoleKeyInfo keyInfo = Console.ReadKey(intercept: true);

        if (keyInfo.Key == ConsoleKey.Escape)
        {
            Console.WriteLine(currentLanguage.CanceledByUser);
            return;
        }
        else if (keyInfo.KeyChar == '#')
        {
            if (playerListScreen)
            {
                HandleBackOption();
                return;
            }
            else if (showPlayerStatsScreen)
            {
                ResetScreen();
            }
        }
        else if (keyInfo.Key == ConsoleKey.UpArrow && selectedOption > 0)
        {
            MoveSelectionUp();
        }
        else if (keyInfo.Key == ConsoleKey.DownArrow && selectedOption < players.Count - 1)
        {
            MoveSelectionDown();
        }
        else if (keyInfo.Key == ConsoleKey.Enter)
        {
            if (playerListScreen)
            {
                SwitchToPlayerStatsScreen();
            }
        }
    }
```

```
}

private void HandleBackOption()
{
    new HomeScreen(currentLanguage);
}

private void ResetScreen()
{
    playerListScreen = true;
    showPlayerStatsScreen = false;
}

private void MoveSelectionUp()
{
    selectedOption--;
}

private void MoveSelectionDown()
{
    selectedOption++;
}

private void SwitchToPlayerStatsScreen()
{
    playerListScreen = false;
    showPlayerStatsScreen = true;
    DisplayPlayerStats();
}

private void DisplayPlayerStats()
{
    Console.Clear();
    Console.WriteLine(currentLanguage.PlayerStats);
    Console.WriteLine("--------------------");
    Player selectedPlayer = players[selectedOption];
    Console.WriteLine($"{currentLanguage.PlayerName} {selectedPlayer.Name}");
    Console.WriteLine($"{currentLanguage.AnsweredQuestions} {selectedPlayer.AnsweredQuestions}");
    Console.WriteLine($"{currentLanguage.CorrectAnswers} {selectedPlayer.CorrectAnswers}");
    Console.WriteLine($"{currentLanguage.WrongAnswers} {selectedPlayer.WrongAnswers}");
    Console.WriteLine($"{currentLanguage.MultiplayerWins} {selectedPlayer.MultiplayerWins}");
    Console.WriteLine("\n-----------------------------------");
    Console.WriteLine(currentLanguage.PressButtonToContinue);
    Console.ReadKey(intercept: true);

    ResetScreen();
}
```

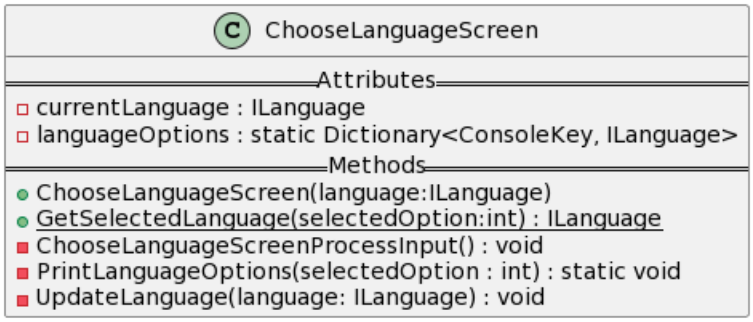
2 Refactorings (6P)

[2 unterschiedliche Refactorings aus der Vorlesung anwenden, begründen, sowie UML vorher/nachher liefern; jeweils auf die Commits verweisen]

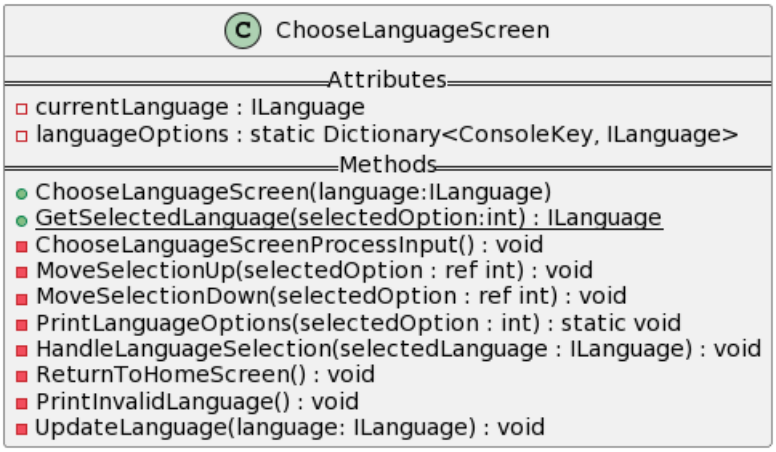
Refactoring 1: Extract Method

Die Methode ChooseLangageScreenProcessInput in ChooseLanguageScreen war extrem lang. Daher wurden Teilinhalte in kleiner Methoden unterteilt, um bessere Wiederverwendbarkeit und bessere Leserlichkeit zu erlangen:

Vorher:



Nachher:

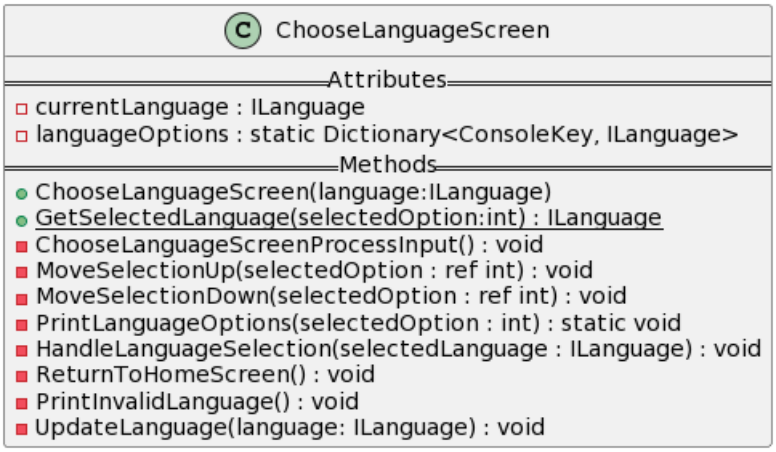


Commit:
https://github.com/BeardedCoffee/QuizGame/commit/cc6d9defc33a5aad8825c5318b590c5d4b74a488

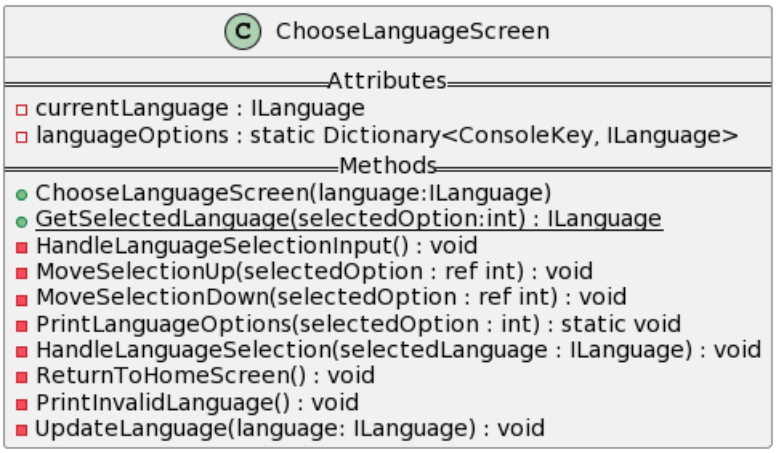
Refactoring 2:

Der Name “ChooseLanguageScreenProcessInput” war nicht aussagekräftig und leicht verwirrend. Daher wurde dieser in “HandleLangageSelectionInput” geändert.

Vorher:



Nachher:



Commit:
https://github.com/BeardedCoffee/QuizGame/commit/6665fd3ab84a18c957b4bdd7ca46034d4598c0d5

Kapitel 8: Entwurfsmuster (8P)

[2 unterschiedliche Entwurfsmuster aus der Vorlesung (oder nach Absprache auch andere) jeweils sinnvoll einsetzen, begründen und UML-Diagramm]

Entwurfsmuster: [Name] (4P)

Entwurfsmuster: [Name] (4P)