Programmentwurf
Laufschrift Konsole

Name: Meßmer, Simon
Martrikelnummer: 9319615
Name: Fassbinder, Lea
Martrikelnummer: 8455846

Abgabedatum: 31.05.2023

Allgemeine Anmerkungen:

- *Gesamt-Punktzahl: 60P (zum Bestehen mit 4,0 werden 30P benötigt)*
- *die Aufgabenbeschreibung (der blaue Text) und die mögliche Punktanzahl muss im Dokument erhalten bleiben*
- *es darf nicht auf andere Kapitel als alleiniger 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)*
- *das Dokument muss als PDF abgegeben werden*
- *es gibt keine mündlichen Nebenabreden / Ausnahmen – alles muss so bearbeitet werden, wie es schriftlich gefordert ist*
- *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 sondern 0,5P Abzug für das fehlende Negativ-Beispiel***
  - *Beispiel*
    - *"Nennen Sie jeweils eine Klasse, die das SRP einhält bzw. verletzt." (2P)*
      - *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: 2P ODER falls im Code mind. eine Klasse SRP verletzt: 0,5P*
- *verlangte Positiv-Beispiele müssen gebracht werden – im Zweifel müssen sie extra für die Lösung der Aufgabe implementiert werden*
- *Code-Beispiel = Code in das Dokument kopieren (inkl. Syntax-Highlighting)*
- *falls Bezug auf den Code genommen wird: entsprechende Code-Teile in das Dokument kopieren (inkl Syntax-Highlighting)*

- *bei UML-Diagrammen immer die öffentlichen Methoden und Felder angegeben – private Methoden/Felder nur angeben, wenn sie zur Klärung beitragen*
- *bei UML-Diagrammen immer unaufgefordert die zusammenspielenden Klassen ergänzen, falls diese Teil der Aufgabe sind*
- *Klassennamen/Variablennamen/etc im Dokument so benennen, wie sie im Code benannt sind (z.B. im Dokument nicht anfangen, englische Klassennamen zu übersetzen)*
- *die Aufgaben sind von vorne herein bekannt und müssen wie gefordert gelöst werden – z.B. ist es keine Lösung zu schreiben, dass es das nicht im Code gibt*
    - *Beispiel 1*
        - *Aufgabe: Analyse und Begründung des Einsatzes von 2 Fake/Mock-Objekten*
        - *Antwort: Es wurden keine Fake/Mock-Objekte gebraucht.*
        - *Punkte: 0P*
    - *Beispiel 2*
        - *Aufgabe: UML, Beschreibung und Begründung des Einsatzes eines Repositories*
        - *Antwort: Die Applikation enthält kein Repository*
        - *Punkte*
            - *falls (was quasi nie vorkommt) die Fachlichkeit tatsächlich kein Repository hergibt: volle Punktzahl*
            - *falls die Fachlichkeit in irgendeiner Form ein Repository hergibt (auch wenn es nicht implementiert wurde): 0P*
    - *Beispiel 3*
        - *Aufgabe: UML von 2 implementierte unterschiedliche Entwurfsmuster aus der Vorlesung*
        - *Antwort: es wurden keine Entwurfsmuster gebraucht/implementiert*
        - *Punkt: 0P*

Kapitel 1: Einführung (4P)

Übersicht über die Applikation (1P)

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

Die Applikation ermöglicht es, eingegebene Texte auf (individuell konfigurierbare) Art und Weise als Lauftext anzeigen zu lassen. Konfigurationen und Texte lassen sich lokal als Textdatei im XML Format speichern und wiederverwenden. Die Lauftexte werden in der Konsole von rechts nach links abgespielt, dafür sind verschiedene einprogrammierte Ascii-Art-Schriftarten verfügbar. Zweckmäßig dient dies als Spaß-Tool, ähnlich zum Ascii-Train oder Ascii-Aquarium.

Starten der Applikation (1P)

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

Das .NET Framework muss installiert sein. Zum Starten kann entweder unter Windows die .exe Datei gestartet werden oder mittels Kommandozeile mit dem Befehl „dotnet TextTicker.dll“.

Technischer Überblick (2P)

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

- .NET wurde eingesetzt, weil wir beide damit bereits mehr Erfahrung sammeln konnten, als mit Java. Damit konnte die Entwicklung effizienter und zügiger voranschreiten. Anhand der Objektorientierung und Plattformunabhängigkeit steht .NET Java mit nichts hinten an.
- XML wurde gewählt, da dafür eine native Implementierung zur (De-) Serialisierung in .NET verfügbar ist und auf externe Bibliotheken verzichtet werden kann. Außerdem ist es ein etabliertes und auch menschenlesbares Format.

Kapitel 2: Clean Architecture (8P)

Was ist Clean Architecture? (1P)

[allgemeine Beschreibung der Clean Architecture in eigenen Worten]

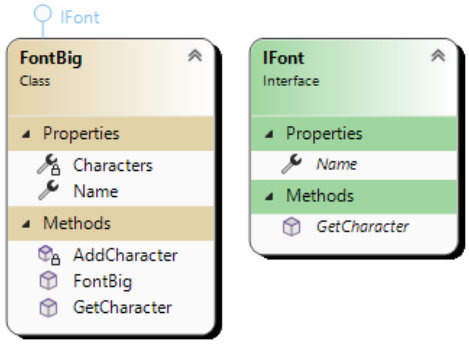
Die Clean Architecture beschreibt einen Architekturstil, der vom Aufbau her aus verschiedenen Schichten besteht. Es gibt einen Kern, der die grundlegendsten abstrakten Funktionalitäten enthält. In den Zwischenschichten sind diverse Vermittler, bis ganz außen die „Kommunikation“ mit der Außenwelt geschieht. Die Schichten sind ausschließlich von außen nach innen abhängig. Auf keinen Fall andersrum.

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]

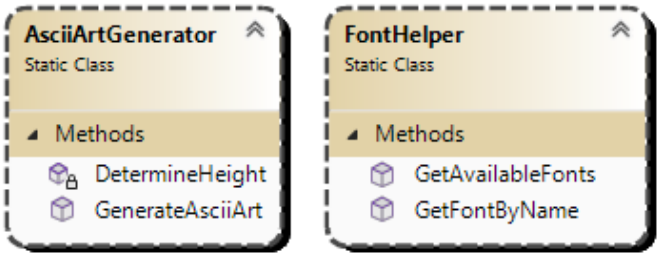
Positiv-Beispiel: Dependency Rule

FontBig implementiert das ‚IFont'-Interface, was bedeutet, dass es von diesem Interface abhängt. Diese Abhängigkeit stellt jedoch keine Verletzung der Dependency Rule dar, da sie spezifische Details darstellt, die für die Implementierung der FontBig-Klasse notwendig sind.
Von der anderen Seite sind die einzigen Abhängigkeiten direkt vom Interface IFont und nicht von der FontBig Klasse.



Negativ-Beispiel: Dependency Rule

AsciiArtGenerator hängt von der klasse FontHelper ab, da die Methode FontHelper.GetFontByName verwendet wird. Um die Dependency Rule nicht zu verletzten, wäre es nötig, eine Abstraktion zu verwenden.

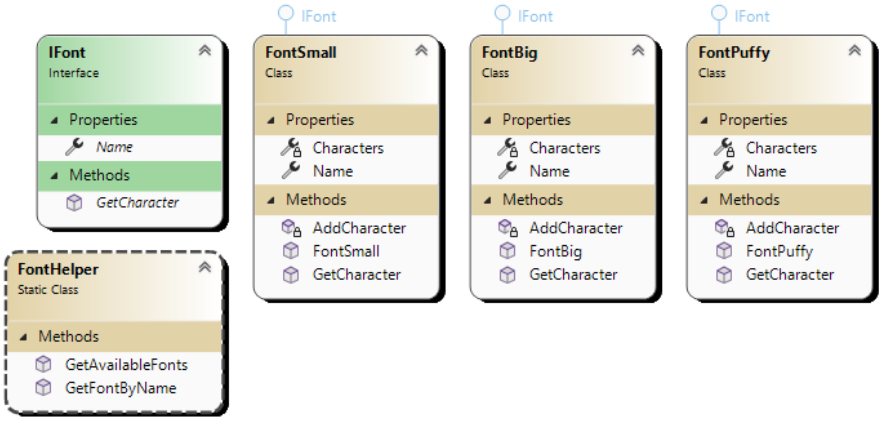


Analyse der Schichten (4P)

[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]

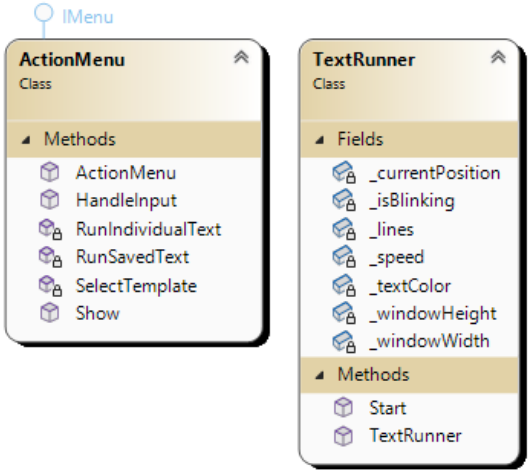
Schicht: [Adapters]

FontHelper stellt Methoden zur Verfügung, um eine Schriftart anhand ihres Namens zu erhalten und eine Liste aller verfügbaren Schriftarten zu erstellen. Dafür wird Reflection benutzt, um dynamisch Instanzen der Klassen zu erstellen, die das IFont-Interface implementieren. Die Klasse stellt eine Schnittstelle zwischen den Schriftarten und den Klassen, die diese Daten verwenden, zur Verfügung.



Schicht: [Application Code]

TextRunner hat die Aufgabe, eine Textanimation auf der Konsole anzuzeigen. Dabei werden Optionen wie Geschwindigkeit, Farbe und ob der Text blinkt oder nicht beachtet. Die Logik zur Steuerung der Textanimation, einschließlich Berechnung der Position und Durchführung des Schreibvorgangs wird in der Klasse implementiert.
Gehört zur Applikationsschicht, weil direkt das gewünschte Verhalten der Anwendung umgesetzt wird.



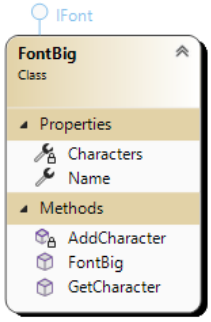
Kapitel 3: SOLID (8P)

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)]

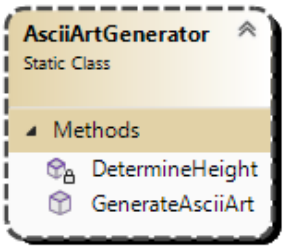
Positiv-Beispiel

FontBig hat als einzige Verantwortung eine bestimmte Schriftart bereitzustellen. Die Zeichen werden weder angezeigt noch verarbeitet, sondern lediglich zur Verfügung gestellt.



Negativ-Beispiel

AsciiArtGenerator hat die Aufgabe, Ascii-Art aus einem gegebenen Text zu generieren. Die Klasse hat jedoch noch eine weitere Verantwortlichkiet: Die Texthöhe bestimmen. Dies ist eher eine Art Hilfs-funktion.



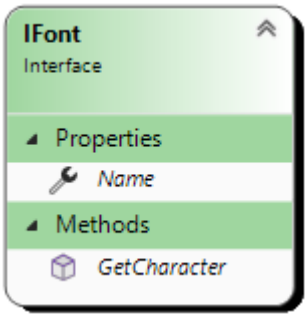
Zur Lösung könnte beispielsweise eine weitere Klasse AsciiArtMetrics erstellt werden, die diese Auf-gabe übernimmt.

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)?]

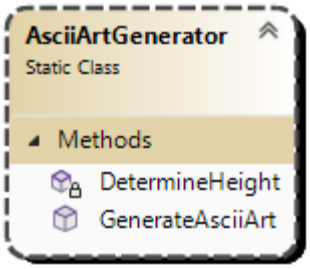
Positiv-Beispiel

IFont mit der Implementierung in Klassen wie FontBig. Das Interface definiert die Eigenschaft Name und die Methode GetCharacter. Jede Klasse, die das Interface implementiert, muss diese Eigenschaft bereitstellt. Das Prinzip ist hier erfüllt, da die Schnittstelle offen für Erweiterungen ist. Weitere Schriften können problemlos hinzugefügt werden.

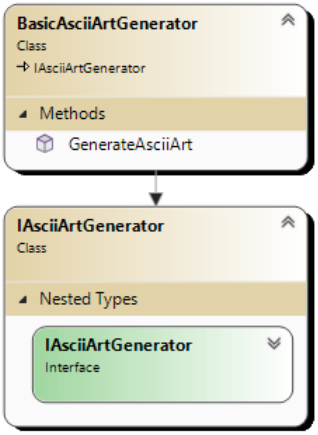


Negativ-Beispiel

AsciiArtGenerator verstößt mit der Methode GenerateAsciiArt gegen das Prinzip. Es wird ein String und ein TextTemplate entgegengenommen und daraus Ascii-Art generiert. Sollte sich die Art und Weise der Generierung ändern, muss die Methode direkt geändert werden.



Eine mögliche Lösung könnte darin bestehen, ein Interface zur Generierung dazwischenzuschalten. Konkrete Klassen implementieren dann das Interface und stellen diverse Arten der Generierung bereit.



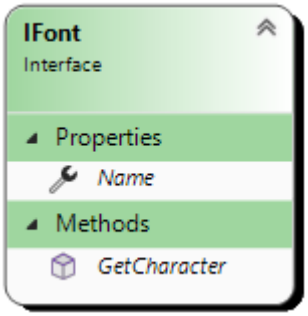
Analyse [LSP/ISP/DIP] (2P)

[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]

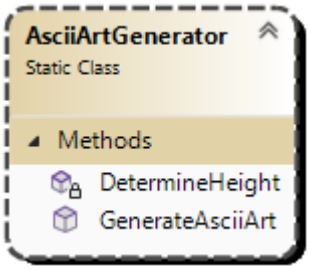
Positiv-Beispiel DIP

IFont hält das DIP ein. High-Level-Module wie die FontBig-Klasse hängen von der IFont-Abstraktion ab und nicht von einer spezifischen Implementierung. Dadurch können verschiedene Schriftarten implementiert werden, ohne dass sich High-Level-Module um Details der spezifischen Implementierungen kümmern müssen.

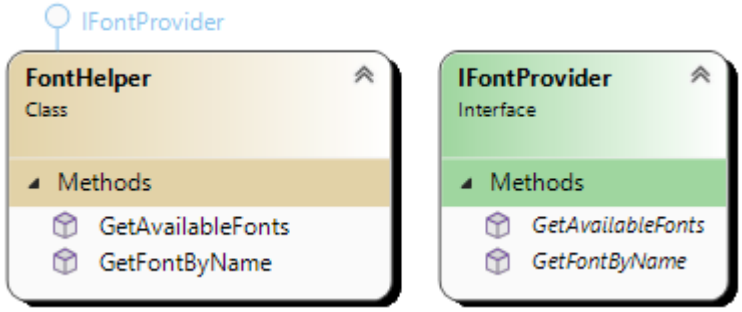


Negativ-Beispiel DIP

AsciiArtGenerator stellt ein negatives Beispiel dar, da direkt auf die Klasse FontHelper zugegriffen wird, um Schriftarten zu holen. Hier hängt das High-Level-Modul AsciiArtGenerator von einem Low-Level-Modul FontHelper ab.



Um dies zu beheben, könnte ein Interface eingefügt werden, das von FontHelper implemenitert wird. AsciiArtGenerator sollte dann von diesem Interface abhängen.

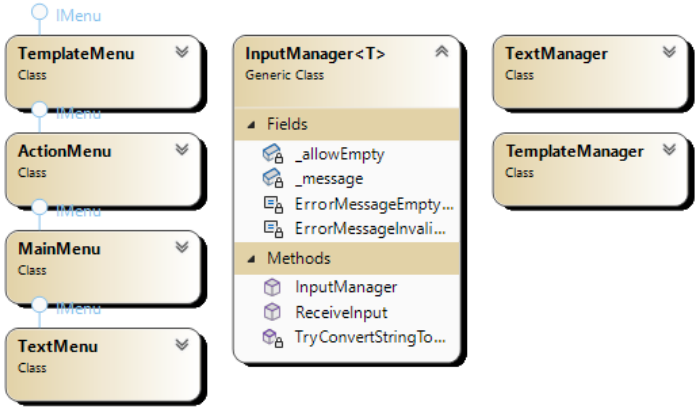


Kapitel 4: Weitere Prinzipien (8P)

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]*

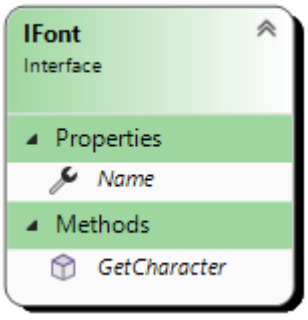
InputManager weist eine geringe Kopplung auf, da sie unabhängig von anderen Klassen agiert. Sie verwendet einen generischen Typ, was bedeutet, dass sie mit jedem Datentyp arbeiten kann. Ihre Aufgaben sind auf die Handhabung der Benutzereingabe beschränkt, sie ist nicht verantwortlich für andere Aspekte der Anwendung.



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]

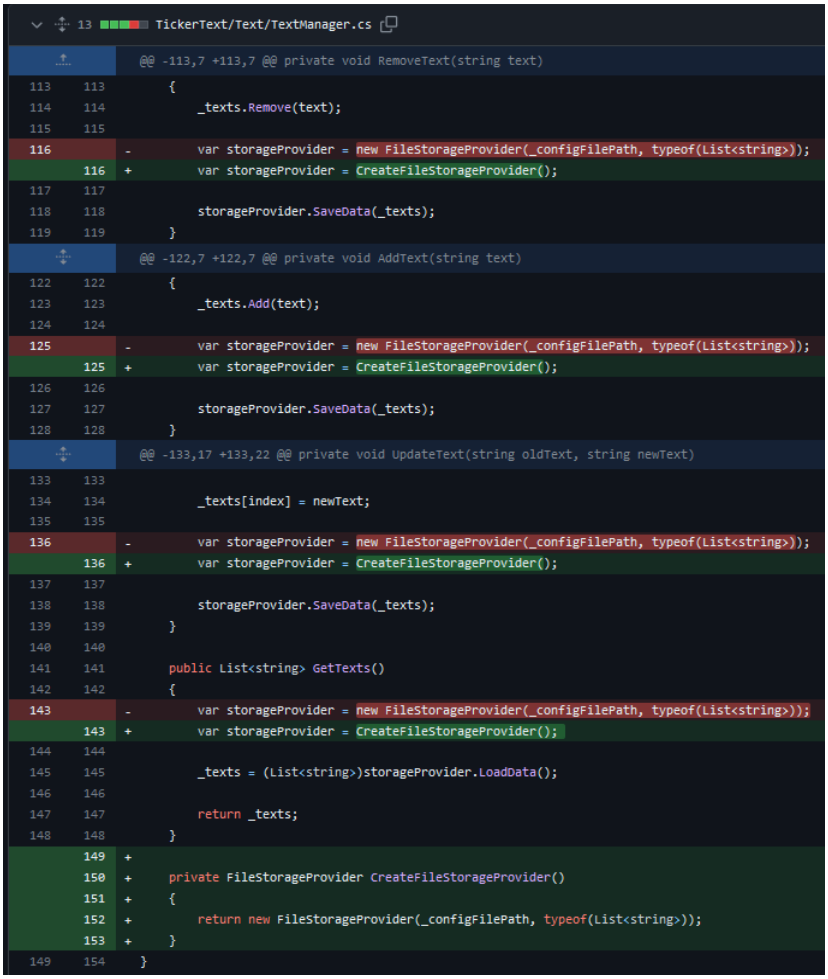
IFont und die jeweiligen Implementierungen der einzelnen Schriftarten sind ein gutes Beispiel für Polymorphismus. Durch die Verwendung des Interfaces, kann mit vielen verschiedenen Schriften gearbeitet werden, ohne die konkrete Implementierung kennen zu müssen. Außerdem wird dadurch die Implementierung weiterer Schriften sehr leicht gemacht.



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]

https://github.com/Fassi2106/TickerText/commit/e99579f9d68358bd0bd0ff683c845ccfb32e2439



```
13  TickerText/Text/TextManager.cs

@@ -113,7 +113,7 @@ private void RemoveText(string text)
113 113         {
114 114             _texts.Remove(text);
115 115
116     -           var storageProvider = new FileStorageProvider(_configFilePath, typeof(List<string>));
    116 +           var storageProvider = CreateFileStorageProvider();
117 117
118 118             storageProvider.SaveData(_texts);
119 119         }
@@ -122,7 +122,7 @@ private void AddText(string text)
122 122         {
123 123             _texts.Add(text);
124 124
125     -           var storageProvider = new FileStorageProvider(_configFilePath, typeof(List<string>));
    125 +           var storageProvider = CreateFileStorageProvider();
126 126
127 127             storageProvider.SaveData(_texts);
128 128         }
@@ -133,17 +133,22 @@ private void UpdateText(string oldText, string newText)
133 133
134 134             _texts[index] = newText;
135 135
136     -           var storageProvider = new FileStorageProvider(_configFilePath, typeof(List<string>));
    136 +           var storageProvider = CreateFileStorageProvider();
137 137
138 138             storageProvider.SaveData(_texts);
139 139         }
140 140
141 141         public List<string> GetTexts()
142 142         {
143     -           var storageProvider = new FileStorageProvider(_configFilePath, typeof(List<string>));
    143 +           var storageProvider = CreateFileStorageProvider();
144 144
145 145             _texts = (List<string>)storageProvider.LoadData();
146 146
147 147             return _texts;
148 148         }
    149 +
    150 +       private FileStorageProvider CreateFileStorageProvider()
    151 +       {
    152 +           return new FileStorageProvider(_configFilePath, typeof(List<string>));
    153 +       }
149 154     }
```

Das Objekt wurde jedes Mal erneut erstellt und der Code hat sich mehrfach wiederholt. Durch die Eliminierung dieser Stellen, wurde zur Übersichtlichkeit beigetragen und zukünftige Anpassungen in der Erstellung eines FileStorageProviders fallen leichter, da sie nur an einer Stelle durchgeführt werden müssen.

Kapitel 5: Unit Tests (8P)

10 Unit Tests (2P)

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

Unit Test	Beschreibung
TemplateManagerTests#CreateTemplate_ShouldAddNewTemplate	Es wird das Hinzufügen eines Templates getestet.
TemplateManagerTests#DeleteTemplate_ShouldRemoveTemplate	Es wird das Entfernen eines Templates getestet.
TemplateManagerTests#GetTemplates_ShouldReturnNoTemplate	Testen, ob der initiale Zustand keine Templates enthält.
TemplateManagerTests#SetSelectedTemplate_ShouldSelectTemplate	Testet, ob ein Template ausgewählt werden kann.
TestTemplateBuilderTests#Build_ShouldCreateTextTemplateWithGivenValues	Testet, ob ein TextTemplate mit den gegebenen Werten erstellt wird.
TestTemplateBuilderTests#Build_WithEmptyName_ShouldThrowArgumentException	Testet, ob eine Exception geworfen wird, wenn kein Name gesetzt ist.
TestTemplateBuilderTests#Build_WithNullFont_ShouldThrowArgumentException	Testet, ob eine Exception geworfen wird, wenn keine Font gesetzt ist.
TestTemplateBuilderTests#SetName_ShouldSetTemplateName	Testet, ob der TemplateName gesetzt wird.
TestTemplateBuilderTests#SetFont_ShouldSetFontName	Testet, ob der FontName gesetzt wird.
TestTemplateBuilderTests#SetColor_ShouldSetColor	Testet, ob die Farbe gesetzt wird.

ATRIP: Automatic (1P)

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

Um eine konsequente und zuverlässige Durchführung von Build & Test, wurde ein Github Workflow angelegt. Dieser führt die Aktionen bei jedem Push auf den Main Branch durch.

ATRIP: Thorough (1P)

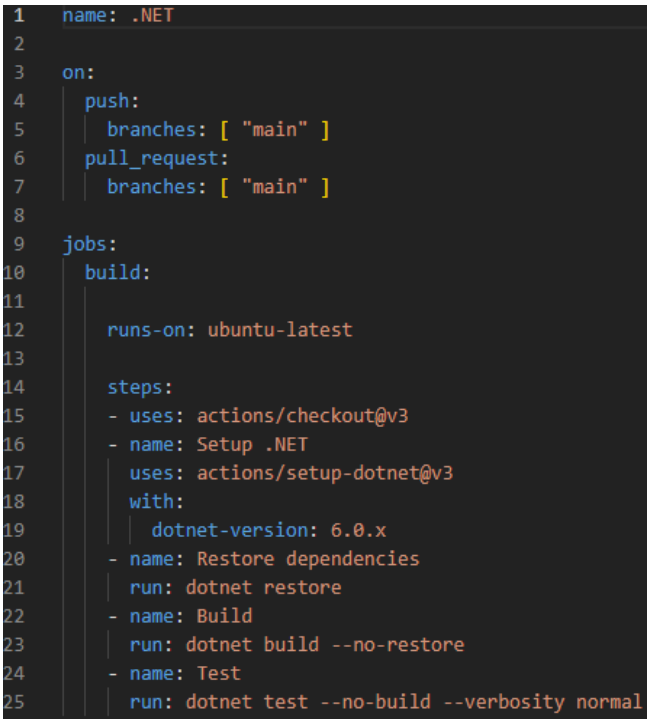
[Code Coverage im Projekt analysieren und begründen]

Die Code Coverage deckt nicht das gesamte Projekt ab, genügt aber dem Projektanspruch. Hauptsächlich werden TestTemplateBuilder und der TemplateManager getestet.

ATRIP: Professional (1P)

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

Professionell ist der angelegte Github Workflow. Damit wurde einer der besten Wege verwendet, um eine konsequente und zuverlässige Durchführung von Tests zu gewährleisten. Der Workflow definiert verschiedene Schritte: Das Aufsetzen von .NET, „restore" der Dependencies, Bauen und Testen des Projekts.



```yaml
name: .NET

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3
    - name: Setup .NET
      uses: actions/setup-dotnet@v3
      with:
        dotnet-version: 6.0.x
    - name: Restore dependencies
      run: dotnet restore
    - name: Build
      run: dotnet build --no-restore
    - name: Test
      run: dotnet test --no-build --verbosity normal
```

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
FontBig	Repräsentiert eine große Schriftart.	Gängier Begriff in der Textdarstellung. Beschreibt klar Funktion und Bedeutung der Klasse.
TextRunner	Lässt den Text auf dem Bildschirm ablaufen.	Sprechender Name, der beschreibt, was die Klasse tut.
TemplateManager	Verwaltung von Textvorlagen.	Präzise Bezeichnung der Aufgabenbeschreibung.
InputManager	Verarbeitet und verwaltet Benutzereingaben.	Spechender Name, der deutlich macht, dass die Klasse für die Nutzereingaben zuständig ist.

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]

Es ist kein Repository vorhanden. Das hat mehrere Gründe:

1. Das Projekt erfordert keine komplexe Datenzugriffslogik oder Datenmanipulation.
2. Es liegt lediglich eine geringe Anzahl von Entitäten vor. Da nur das TextTemplate als Entität vorliegt und keine komplexen Beziehungen bestehten, wäre der Einsatz eines Repositories übertrieben.
3. Die Anwendungslogik ist simpel und erfordert keine komplexen Aktionen. Daher kann diese direkt auf die Datenquelle zugreifen.
4. Die Anforderungen an das Projekt werden sich nicht ändern. Repositories sind in erster Linie dazu nützlich, die Anwendungslogik von der Datenschicht zu entkoppeln und Ändernungen in der Datenstruktur zu erleichtern. Dies ist hier nicht notwendig und wäre daher nicht gerechtfertigt.

Aggregates (1,5P)

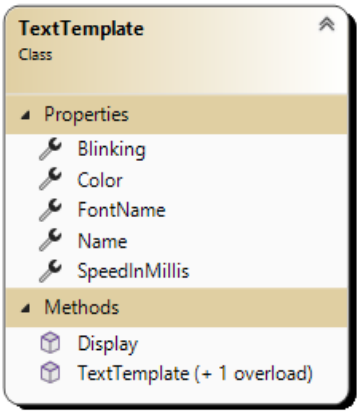
[UML, Beschreibung und Begründung des Einsatzes eines Aggregates; falls kein Aggregate vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist– NICHT, warum es nicht implementiert wurde]

Es ist kein Aggregate vorhanden. Das hat verschiedene Gründe. Das Projekt hat eine einfache Domänenlogik mit einer begrenzten Anzahl von Klassen und Funktionen. Es gibt keine komplexen Abhängigkeiten, die Aggregates erforderlich machen würden. Außerdem ist liegt keine Datenkomplexität vor und die transaktionalen Anforderungen sind sehr gering.

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]

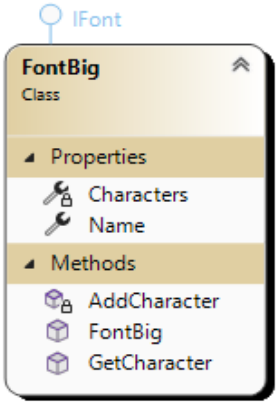
Das TextTemplate repräsentiert eine Vorlage für die Darstellung von Texten. Es enthält Attribute wie Name, FontName, Color, SpeedInMillis und Blinking, die die Eigenschaften und Parameter der Vorlage definieren.
Die Verwendung einer Entity wie dem TextTemplate ermöglicht die Strukturierung, Verwaltung und Manipulation von Vorlagen auf eine einheitliche Weise. Es ermöglicht eine klare Abgrenzung und Identifizierung der Vorlage als eigenständige Einheit und erleichtert somit die Verwaltung und Bearbeitung der Vorlagen im System.



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]

FontBig ist ein Value Object, das einen großen Schriftartstil repräsentiert. FontBig ermöglicht die einheitliche Verwendung eines spezifischen Schriftartstils in der Anwendung, verbessert die Lesbarkeit des Codes und erleichtert die Integration mit anderen Komponenten und Funktionen.



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)]

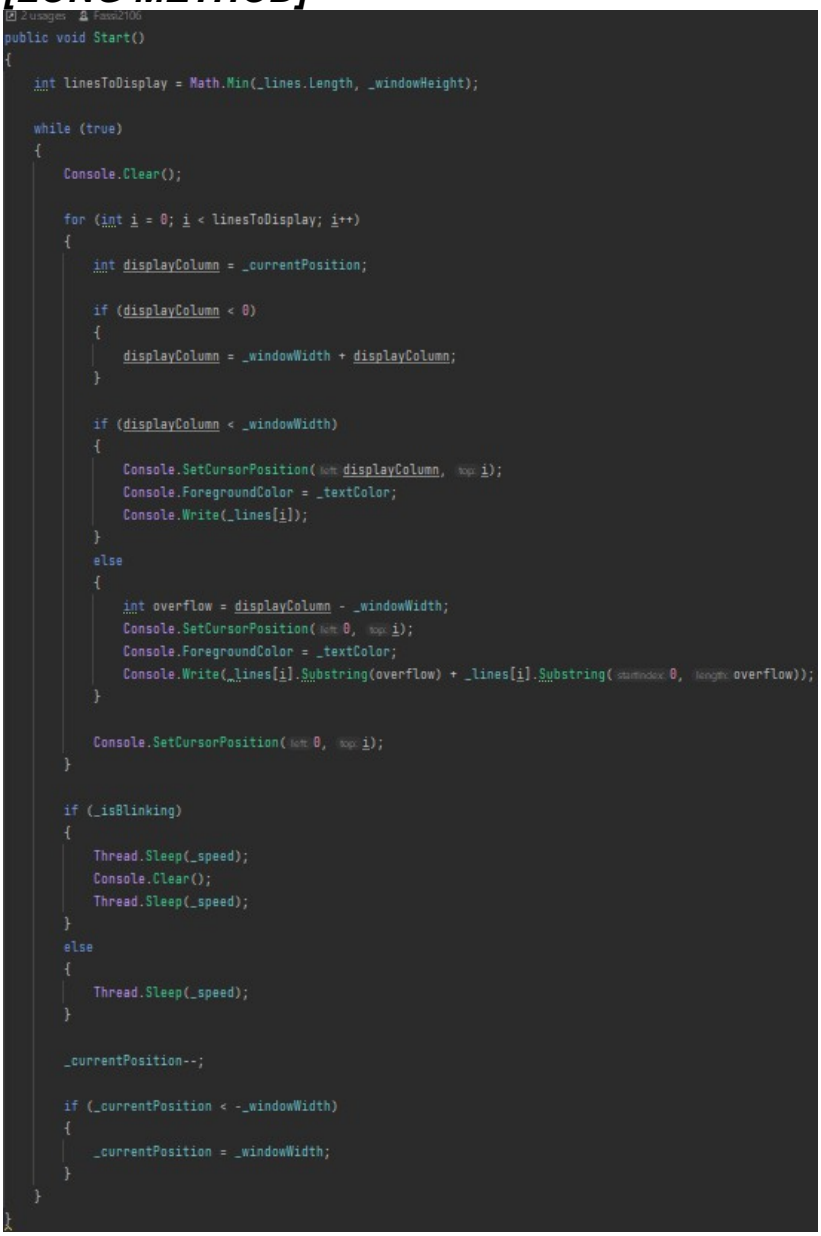
[DUPLICATED CODE]

```
3 usages  jeeez99 +1
public void RemoveTemplate(TextTemplate template)
{
    _templates.RemoveAll( match: t :TextTemplate => t.Name.Equals(template.Name));

    var storageProvider = new FileStorageProvider(_configFilePath, typeof(List<TextTemplate>));

    storageProvider.SaveData(_templates);
}

4 usages  Fassi2106 +1
public void AddTemplate(TextTemplate template)
{
    _templates.Add(template);

    var storageProvider = new FileStorageProvider(_configFilePath, typeof(List<TextTemplate>));

    storageProvider.SaveData(_templates);
}
```

Ein möglicher Lösungsweg wäre, den duplizierten in eine eigene Methode auszulagern und diese Methdoe zu verwenden.

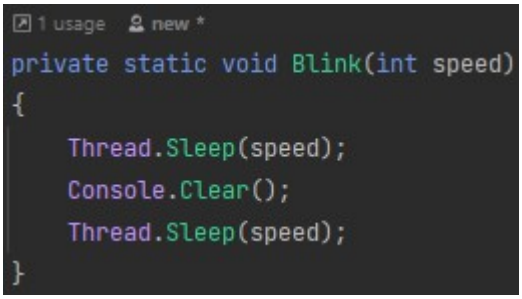
```
3 usages  jeeez99 *
public void RemoveTemplate(TextTemplate template)
{
    _templates.RemoveAll( match: t :TextTemplate => t.Name.Equals(template.Name));
    SaveData(_templates);
}

4 usages  jeeez99 +1 *
public void AddTemplate(TextTemplate template)
{
    _templates.Add(template);
    SaveData(_templates);
}

2 usages  new *
private void SaveData(List<TextTemplate> _templates)
{
    var storageProvider = new FileStorageProvider(_configFilePath, typeof(List<TextTemplate>));
    storageProvider.SaveData(_templates);
}
```

[LONG METHOD]



```
2 usages  Fassi2106
public void Start()
{
    int linesToDisplay = Math.Min(_lines.Length, _windowHeight);

    while (true)
    {
        Console.Clear();

        for (int i = 0; i < linesToDisplay; i++)
        {
            int displayColumn = _currentPosition;

            if (displayColumn < 0)
            {
                displayColumn = _windowWidth + displayColumn;
            }

            if (displayColumn < _windowWidth)
            {
                Console.SetCursorPosition( left: displayColumn,  top: i);
                Console.ForegroundColor = _textColor;
                Console.Write(_lines[i]);
            }
            else
            {
                int overflow = displayColumn - _windowWidth;
                Console.SetCursorPosition( left: 0,  top: i);
                Console.ForegroundColor = _textColor;
                Console.Write(_lines[i].Substring(overflow) + _lines[i].Substring( startIndex: 0,  length: overflow));
            }

            Console.SetCursorPosition( left: 0,  top: i);
        }

        if (_isBlinking)
        {
            Thread.Sleep(_speed);
            Console.Clear();
            Thread.Sleep(_speed);
        }
        else
        {
            Thread.Sleep(_speed);
        }

        _currentPosition--;

        if (_currentPosition < -_windowWidth)
        {
            _currentPosition = _windowWidth;
        }
    }
}
```

Die Methode könnte gekürzt werden, in dem gewisse Teile der Methode ausgelagert werden. Beispielsweise das Blinken.

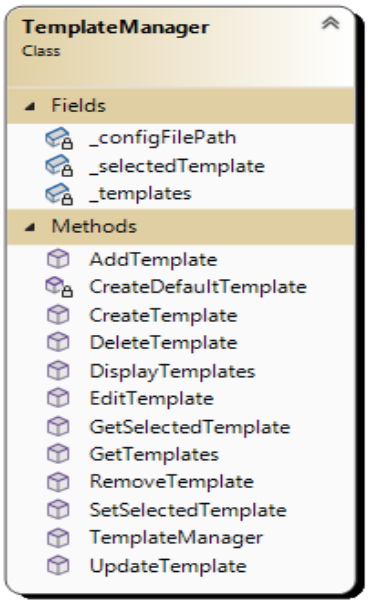


```
1 usage   new *
private static void Blink(int speed)
{
    Thread.Sleep(speed);
    Console.Clear();
    Thread.Sleep(speed);
}
```

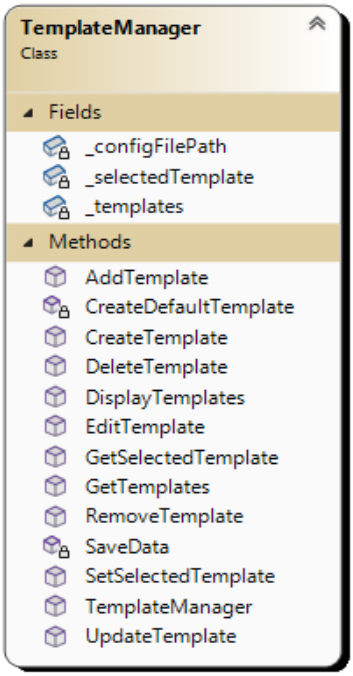
2 Refactorings (6P)

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

[Extract Method]

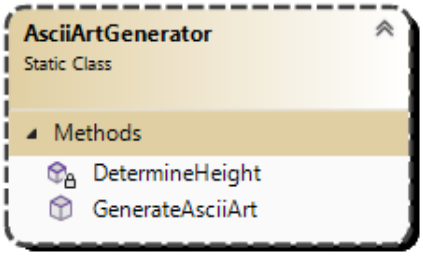


https://github.com/Fassi2106/TickerText/commit/79e0c38f9dfcfce21fd87466df534a3c433a635c

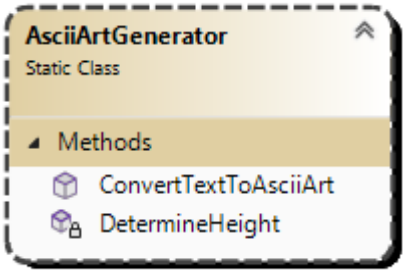


Das Extract Method Refactoring wird angewendet, um wiederholten Code zu reduzieren und dadurch die Lesbarkeit und Wiederverwendbarkeit des Codes zu verbessern. Durch das Extrahieren des Codes wird der gesamte Code kompakter und leichter verständlich. Außerdem wird das DRY-Prinzip gefördert.

[Rename Method]



https://github.com/Fassi2106/TickerText/commit/a822fd3a523f6f6f45dc0962a1460d429620513ec8



Das Rename Method Refactoring wird angewendet, um den Namen einer Methode zu mehr Klarheit abzuändern und damit die Verständlichkeit im Code zu verbessern. Durch die Verwendung von aussagekräftigen Namen wird die Absicht und Funktion der Methode deutlicher. Dadruch ist der Code für Entwickler leichert lesbar, verständlich und besser wartbar.

Kapitel 8: Entwurfsmuster (8P)

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

Entwurfsmuster: [Singleton] (4P)

Der Einsatz des Singleton-Entwurfsmusters im MenuManager ermöglicht den Zugriff auf eine einzige Instanz vom ganzen Projekt aus. Dies ist sinnvoll, da nur eine einzige Instanz des MenuManagers existieren soll, die das Konsolenmenü verwaltet.

Entwurfsmuster: [Builder] (4P)

Das Builder-Entwurfsmuster dient dazu, ein Objekt stückweise mit Informationen zu füllen und am Ende zu erstellen. In diesem Falle ist der TextTemplateBuilder sinnvoll, da die TextTempaltes in der Konsole in mehreren Schritten erstellen werden. Um eine übersichtliche und simple Erstellung zu gewährleisten, wird ein Builder Objekt verwendet. Am Ende des Erstellungsprozesses wird das Objekt gebaut und persistiert. Verläuft der Erstellungsprozess nicht erfolgreich, so kann das Builder Objekt einfach verworfen werden und es bleiben keine Daten übrig.