

# 1- Aufgaben:

## 1.1- Aufgabe bis Montag, 27.10.2025


Ihr erstellt **zwei Diagramme** für euer eigenes Spiel:

### Klassendiagramm (Pflicht)

Ein **Klassendiagramm** beschreibt die **Struktur eures Programms**:

- welche **Klassen** vorhanden sind,
- welche **Attribute** und **Methoden** sie haben,
- und **wie sie miteinander verbunden sind** (Aggregation, Assoziationen, Vererbung ...).

Ihr müsst **sowohl die GUI-Klassen** (Präsentationsschicht) als auch die **Application-Schicht** (Entity- und Control-Klassen) darstellen, damit man die Verbindung zwischen Oberfläche und Logik erkennt.

 **Bewertungsschwerpunkt** liegt aber auf der **Application-Schicht**, also auf den **Entity-** und **Control-Klassen**.

### Sequenzdiagramm (Pflicht)

Das **Sequenzdiagramm** zeigt, **wie eure Klassen im Ablauf zusammenarbeiten** – also **wer wen aufruft** und **in welcher Reihenfolge** etwas passiert.

 **Bewertungsschwerpunkt:**

Ihr lernt das Thema **in Software-Engineering 1 bei Herrn Bruns**.

Ihr seid **verpflichtet**, ein Sequenzdiagramm zu machen und müsst **mindestens zeigen, dass ihr es versucht habt**.

 **Abgabe:**

Jede\*r erstellt **sein eigenes Klassendiagramm und Sequenzdiagramm** und gibt alles **als ZIP-Datei** beim **Projektleiter** ab.

 **Abgabetermin:** Montag, **27.10.2025**, 23:59 Uhr

## 1.2- Aufgabe bis Freitag, 31.10.2025

### Persönliche Zeitplanung (Pflicht)


Jede\*r erstellt eine **eigene Zeitplanung** für **das eigene Spiel**.

In **Moodle** findet ihr dazu die Folien mit dem Titel

 „Zeitplanung“

## Inhalt der PDF „Zeitplanung“

- Wie man einen Zeitplan Schritt für Schritt erstellt
- Welche Aufgaben, Zeiten und Meilensteine geplant werden sollen
- Beispiel eines 12-Wochen-Plans für ein Softwareprojekt

 **Abgabetermin: Freitag, 31. Oktober 2025, 12:00 Uhr**

## 2-Schichtenarchitektur (Hinweise)

In der Softwareentwicklung teilt man das Programm in **Schichten (Layers)** auf.  
Das hilft, das Programm **übersichtlich** zu halten und **später leichter zu erweitern**.

### Mit welcher Schicht beginnt man zuerst?

Man beginnt **immer mit der Application-Schicht** (Logik)  
und macht **danach die Präsentationsschicht (GUI)**.

### Warum?

Die **Application-Schicht funktioniert auch ohne GUI**.

Beispiel:

Ein Taschenrechner-Programm kann im Code rechnen,  
auch ohne Fenster oder Buttons.

Später kann man dann eine **GUI** hinzufügen,  
die diese Logik einfach benutzt.

**Wichtigster Layer: Application-Schicht (auch Logik- oder Anwendungsschicht)**

 Hier steckt die **eigentliche Funktion eures Spiels oder Programms**.

## 2.1 Application-Schicht

Sie besteht aus zwei Arten von Klassen:

### Entity-Klassen

- enthalten **Daten aus der realen Welt**  
(z. B. Spieler, Person, Kunde, Spielbrett)
- ändern sich selten
- speichern nur Werte (Name, Punkte, Position ...)
- keine Spielabläufe oder Steuerung

### Control-Klassen

- sind das **Herzstück des Programms**
- steuern, **was im Spiel passiert**
- rufen Methoden der Entity-Klassen auf
- enthalten **die Logik und Abläufe** (z. B. wer dran ist, was passiert bei Klick, Bewegung usw.)

## 2.2 Präsentationsschicht (GUI-Schicht)

- Zeigt die Daten aus der Application-Schicht an
- Reagiert auf Eingaben (z. B. Klicks, Tasten, Eingaben)
- Ruft Methoden aus der Application-Schicht auf
- Enthält **keine Spiellogik** – sie zeigt nur an, was passiert

## 3-Kommunikation zwischen den Software-Schichten

Diese Muster sorgen dafür, dass die Schichten **lose gekoppelt** sind – jede Schicht kann geändert werden, ohne dass alles neu geschrieben werden muss:

- **Fassade-Pattern (Facade):**  
Schmale Schnittstelle zwischen den Schichten.  
GUI ruft nur Methoden der **Fassade** in der Application-Schicht auf.
- **Observer-Pattern:**  
Wenn sich Daten (z. B. in einem Model oder einer Entity) ändern, werden die **GUI-Komponenten automatisch benachrichtigt und aktualisiert**.
- **Singleton-Pattern:**  
Stellt sicher, dass es von einer Klasse nur **ein einziges Objekt** gibt, z. B. eine zentrale Steuerung (GameController oder AppManager).

Eine ausführliche Erklärung dazu finden Sie in den **Folien**

👉 *Tutorial: Software-Entwicklungsprozess / -architektur*

## 4- Klassendiagramm (Pflicht)

**Was ist das? (aus Prog2 bekannt)**

Ein **Klassendiagramm** zeigt die **Struktur** eurer Anwendung:  
welche **Klassen** es gibt, welche **Attribute und Methoden** sie haben  
und **wie sie miteinander verbunden sind** (Aggregation, Assoziationen, Vererbung ...).

- Klassen (Name, wichtige Attribute, zentrale Methoden)
- Beziehungen (Assoziation, Aggregation/Komposition, Vererbung)
- Multiplizitäten (1, 0..1, 1..\*, \*), Navigierbarkeit (Pfeilrichtung)

⚙️ **Wichtiger Schwerpunkt**

Am wichtigsten ist, dass die **Application-Schicht** richtig modelliert ist.

- Ihr müsst auch GUI-Klassen (Präsentationsschicht) im Diagramm anlegen, damit man die Verbindung zur Logik sieht.
- Bewertungsschwerpunkt liegt jedoch auf der Application-Schicht, also auf Entity- und Control-Klassen.

### Erklärung und Beispiel

Eine genaue Erklärung findet ihr in Moodle unter:

📁 „Von der Anforderungsanalyse zum Klassenmodell“

und auch in den Folien

📁 Tutorial: Software-Entwicklungsprozess / -architektur

Beispiel: Kontakt-Datenbank – Klassendiagramm (Folie Nr. 47–48)

## 5- Sequenzdiagramm (Pflicht)

### Was ist ein Sequenzdiagramm?

Ein **Sequenzdiagramm** zeigt, wie **Objekte (aus dem Klassendiagramm)** in einem bestimmten **Ablauf (Use Case)** miteinander **kommunizieren**.

Es stellt also **den Ablauf im System Schritt für Schritt dar** – **wer ruft wen auf**, und **in welcher Reihenfolge** etwas passiert.

### Ziel

Das Ziel ist, zu zeigen,  
**wie ein Use Case im System tatsächlich umgesetzt wird.**  
Man sieht also:

- welche **Objekte beteiligt** sind,
- welche **Methoden** aufgerufen werden,
- und **in welcher Reihenfolge** die Aktionen ablaufen.

💡 So verbindet das Sequenzdiagramm eure **Use Cases** mit dem **Klassendiagramm**:

Use Case = *Was passiert fachlich?*

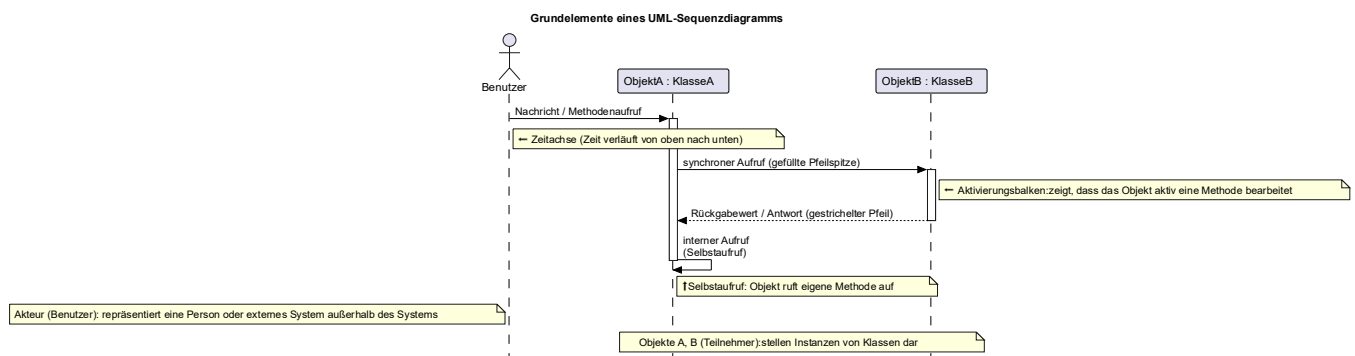
Klassendiagramm = *Welche Klassen gibt es?*

Sequenzdiagramm = *Wie diese Klassen zusammenarbeiten, um den Use Case auszuführen.*

## 5-1 Notation: (es ist nicht vollständig)

Dies ist **nicht die vollständige UML-Notation**, sondern nur eine **vereinfachte Übersicht** der wichtigsten Elemente, damit ihr die Grundidee der **Sequenzdiagramme** versteht.

Ziel ist nicht, alle Symbole auswendig zu lernen, sondern **selbstständig mit Hilfe der offiziellen UML-Dokumentation oder Online-Quellen** (z. B. Visual Paradigm, Lucidchart, PlantUML oder [uml-diagrams.org](http://uml-diagrams.org)) die vollständige Notation zu entdecken und richtig anzuwenden.



## 6-beispiel: von Use Case -> Klassen Diagramm -> Sequenzdiagramm

Use Case: Spiel starten

**Name:** Spiel starten

**Primärer Akteur:** Spieler

**Ziel:** Ein Spieler gibt Name und Alter an; das System prüft die Altersfreigabe und startet das (einzige) Spiel.

**Vorbedingungen:**

- System ist gestartet und bereit.
- Ein (einziges) Spiel mit festem Mindestalter ist im System vorhanden.

**Auslöser:**

- Der Spieler klickt im Spiel-Dialog auf „Start“.

**Hauptszenario (Basic Flow, ohne Alternativen):**

1. Der Spieler öffnet den Spiel-Dialog.
2. Der Spiel-Dialog fordert **Name** und **Alter** an; der Spieler gibt beides ein.
3. Der Spieler klickt **Start**.
4. Der Spiel-Dialog übergibt **Name** und **Alter** an die **DarkIt-Fassade**.

5. Die DarkIt-Fassade ruft den **GameController** mit einem **Spieler**-Objekt auf.
6. Der GameController prüft, ob der Spieler das **Mindestalter** des Spiels erfüllt.
7. Bei erfüllter Bedingung startet der GameController das Spiel.
8. Der Spiel-Dialog zeigt die Bestätigung „Spiel gestartet“.

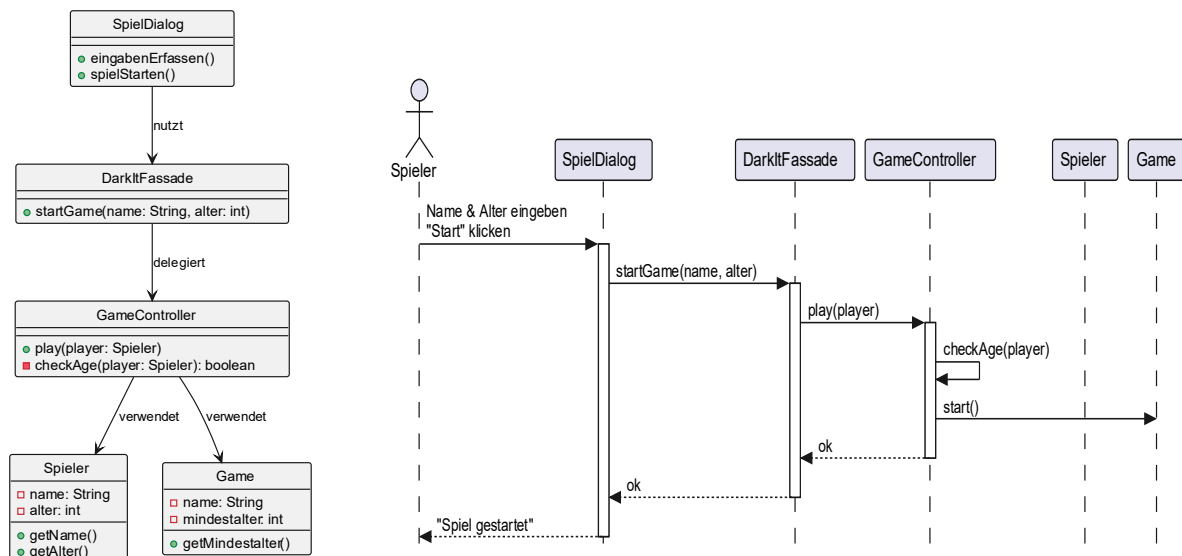
#### Nachbedingungen (Erfolg):

- Das Spiel ist gestartet und läuft.
- Der Start wurde dem Spieler in der GUI bestätigt.

#### Beteiligte Systemkomponenten (Kontext): (nur damit euch klar seid)

- **GUI:** SpielDialog
- **Fassade:** DarkItFassade
- **Anwendung:** GameController, Spieler (Entity), Game (Entity)

-Klassen diagramm / Sequenzdiagramm (in Voll Größe am Ende diese Pdf )



### Hinweis:

Normalerweise werden die **Schichten** (GUI, Fassade, Anwendung) **nicht als Packages** dargestellt. Hier wurde die Einteilung **nur zur Veranschaulichung** ergänzt, damit erkennbar ist, welche Klasse zu welcher Schicht gehört.

In professionellen Diagrammen wird die Trennung meist **durch horizontale Linien** oder **durch die Platzierung der Klassen** verdeutlicht.

Die **Fassade** ist **keine eigene Schicht**, sondern dient lediglich als **Schnittstelle** zwischen der **Präsentationsschicht (GUI)** und der **Anwendungsschicht (Application)**.

Ebenso wird im **Sequenzdiagramm** normalerweise **nicht jede Klasse nach Schicht gruppiert**; die farbliche oder räumliche Aufteilung wurde hier **nur zur Verdeutlichung** hinzugefügt.

