Aufgabe 1: Programmierung vs. Software Engineering

1. Einen Fehler im Code einer Webseite beheben, der dazu führt, dass Bilder nicht angezeigt werden:
   Programmierung → keine Planung, Management, Architektur-/Infrastrukturänderung nötig, beschränkt sich auf rein technische Aufgabe
2. Eine Funktion in JavaScript schreiben, die prüft, ob ein Passwort die Sicherheitsanforderungen erfüllt, bevor es im Anmeldeformular einer Webseite akzeptiert wird:
   Programmierung → Problem beschränkt sich die Implementierung einer Funktion
3. Ein Großprojekt planen, das über mehrere Jahre läuft und von verschiedenen Teams in unterschiedlichen Ländern bearbeitet wird:
   Software Engineering → Problem beinhaltet Planung, Wartung, Management z.B. Koordination/Kommunikation zwischen Teams.
4. Ein Backup-Skript schreiben, das täglich die Datenbank sichert und bei Fehlern eine Benachrichtigung aussendet:
   Programmierung → Problem beschränkt sich auf Implementierung einer Funktion / eines Skripts.
5. Die Architektur einer Cloud-basierten Anwendung entwerfen, die mit einer variablen Anzahl von Benutzern umgehen kann, ohne an Performance zu verlieren:
   Software Engineering → Problemstellung benötigt Planung und Design der skalierbaren Architektur.
6. Ein Python-Skript entwickeln, das CSV-Dateien einliest und daraus eine Zusammenfassung der Daten als PDF generiert:
   Programmierung → Problem umfasst nur die Implementierung eines klar definierten Ablaufes.
7. Eine Kostenschätzung und einen Zeitplan für die Integration einer neuen Technologie in ein bestehendes Software-System erstellen:
   Software Engineering → Problem beinhaltet Planung und Management
8. Eine Automatisierungsroutine erstellen, die in einer Datenbank nach Duplikaten sucht und diese automatisch bereinigt:
   Programmierung → Problem umfasst Implementierung einer technischen Lösung, die wohldefiniert ist.

Aufgabe 1: Richtig oder Falsch?

1. Das Agile Manifest schätzt „Prozesse und Werkzeuge“ höher als „Individuen und Interaktionen“:
   Falsch → „Individuals and interactions over processes and tools“
2. Im Wasserfallmodell ist der Kunde hauptsächlich in den Phasen der Anforderungsanalyse und beim Testen involviert.
   Richtig → Der Kunde ist in der Anforderungsanalyse involviert, um Spezifikationen / ein Lastenheft festzulegen. Später beim Testen wird das Produkt vom Kunden wieder geprüft, ob es seinen Erwartungen / dem Lastenheft entspricht.

3. Kanban vermeidet feste Zeitrahmen für die Fertigstellung von Arbeitspaketen und fokussiert stattdessen auf kontinuierlichen Durchfluss und Pull-Prinzipien.
   Richtig → Kanban setzt nicht wie Scrum auf Sprints, also feste Zeitrahmen. Stattdessen wird auf einen kontinuierlichen Arbeitsfluss und das Pull-Prinzip gesetzt, sofern Kapazitäten frei sind.

4. Das V-Modell erlaubt eine flexible Anpassung der Projektphasen und -ergebnisse während des Entwicklungsprozesses.
   Falsch → Das V-Modell ist weiterhin ein klassisches Modell. Es erweitert das Wasserfall-Modell um den Punkt der Qualitätssicherung und leitet Testfälle aus den Phasen der Analyse, des Entwurfs und der Implementierung ab.

5. Agile Modelle wie Scrum erfordern, dass der Kunde nur zu Beginn und am Ende des Projekts involviert ist.
   Falsch → Scrum bindet den Kunden kontinuierlich ein, z.B. indem dieser den Product-Owner stellt oder durch Teilnahme an Reviews nach Sprints.

6. Kanban erfordert, dass Arbeitspakete immer in der Reihenfolge abgeschlossen werden, in der sie begonnen wurden.
   Falsch → Kanban limitiert nur die maximale Anzahl an Arbeitspaketen, die zu einem Zeitpunkt in Bearbeitung sein dürfen. Eine feste Reihenfolge wird nicht festlegt.

Aufgabe 2: Agile vs. Klassische Vorgehensmodelle

1. Startup: **agiles Vorgehensmodell** → Die im Text dargestellten Anforderungen sind häufige Updates und schnelle Anpassungen an Nutzerfeedback, welche für die agilen Methoden sprechen. Da es sich um ein Startup handelt, würden agile Methoden außerdem den Vorteil bieten, dass man sich schnell an Personaländerungen und/oder Kapitalveränderungen anpasse könnte.

2. Bank: **klassisches Vorgehensmodell** → Es gibt einen festen Zeitplan / Zeitlimit sowie ein festes Budget, diese erfordern Planung zur Einhaltung. Die Anwendung unterliegt hohen Sicherheits-Standards, dies erfordert sorgfältige Prüfung.

3. Smart-Home: **agiles Vorgehensmodell** → Es wird Flexibilität auf einen sich schnell ändernden Markt gefordert. Agile Methoden erlauben inkrementelle Entwicklung, neue Funktionen / neue Gerätetypen können schrittweise integriert werden. Dies erfordert vermehrte Deployments.

Aufgabe 3: Projektdurchführung nach Wasserfallmodell und Scrum

3.1:

Anforderungsdefinition:

- Klärung mit dem Hausbesitzer, welche Räume / Bereiche renoviert werden sollen

- Festlegung eines Budgets und Zeitplans
- Prüfung rechtlicher Vorgaben: Werden Baugenehmigungen benötigt? Welche Vorschriften müssen eingehalten werden?
- **Abschlusskriterien:** Dokumentation der Änderungen & Abnahme dieser durch Besitzer, Architekten / Bauunternehmen

Entwurf:

- Erstellung von Architektenplänen
- Erstellung von Materiallisten
- Auswahl von Bauunternehmen
- **Abschlusskriterien:** Freigabe alle Pläne und Bauunterlagen

Entwicklung:

Umsetzung der Arbeiten basierend auf den Plänen

- Abbrucharbeiten abgeschlossen
- neue Wände, Böden, Kabel, Rohre, Dämmung, Fenster eingebaut
- Abschlusskriterien: Bauabnahme durch Architekten/Bauleitung

Installation & Test:

Einbau und Testen aller Systeme z.B. Heizung, Elektrik, Wasser

- Mängelprotokoll erstellt und behoben
- Abschlusskriterien: Haus ist bewohnbar

Wartung:

- Wartung von Heizung, Elektrik und Wasser
- ggf. Garantien beanspruchen o. Reparaturen beauftragen

3.2:

Sprint 1:

Sprint-Ziel: Erstellung der Planungsgrundlage für die Renovierung
Ausgewählte Items:

- Besprechung mit dem Hausbesitzer über gewünschte Änderungen (z.B. Raumaufteilung, energetische Sanierung).
- Architektenpläne erstellen → Bestandsaufnahme des Gebäudes und erste Entwürfe für Grundrisse und Innenraumgestaltung erstellen.
- Materialplanung → Auswahl von Materialien und deren Zulieferer eingrenzen

Daily Scrum:

- Team bespricht Fortschritte bei der Erstellung der Pläne und der Analyse der Anforderungen.
- Hindernisse werden identifiziert (z. B. fehlende Genehmigungen oder unklare Anforderungen).

Sprint Review:

- Präsentation der ersten Entwürfe der Architektenpläne.
- Feedback vom Hausbesitzer einholen und Anpassungen dokumentieren.

Sprint 2:

Sprint-Ziel: Abschluss der Planungsphase / Vorbereitung auf die Bauarbeiten
Ausgewählte Items:

- Finalisierung der Architektenpläne - Anpassung basierend auf Feedback aus Sprint 1
- Genehmigungen einholen
- Bauunternehmen, Materialien und Zulieferer festlegen

Daily Scrum:

- Fortschrittskontrolle zur Planungsanpassung und Genehmigungsabwicklung.
- mögliche Verzögerungen (z. B. langsame behördliche Prozesse) besprechen

Sprint Review:

- Präsentation der finalen Architektenpläne für die Bauarbeiten.
- Verbindliche Beauftragung der Firmen.

Aufgabe 1: Passendes Vorgehensmodell für große Projekte

Die Auswahl des Vorgehensmodells ist essentielle für Projekte, da diese unterschiedliche Vor- und Nachteile bieten. Verschiedene Modelle haben andere Anforderungen an das Team, dessen Struktur sowie die Erfahrung der Mitglieder und deren Selbstorganisation. Außerdem beeinflussen diese das Produkt, welches ausgeliefert wird und wann beziehungsweise wie es ausgeliefert wird.

Sofern die Anforderungen eines Projektes vollständig bekannt und als stabil beziehungsweise als unveränderlich angenommen werden können, eignet sich das Wasserfallmodell. Es handelt sich um ein lineares und vorhersehbares Verfahren, dadurch ist es gut geeignet in stark regulierten Umgebungen und Situationen in denen eine ausgiebige Dokumentation notwendig ist. Das Wasserfallmodell legt dabei feste zeitliche Abläufe und Fristen fest. Es ist dabei weniger auf Zusammenarbeit zwischen den Teams ausgelegt, da es sehr hierarchisch strukturiert ist und die Abläufe fest, folgend aufeinander, definiert sind. Das Produkt wird beim Wasserfallmodell am Ende ausgeliefert, was Rückmeldungen erst sehr spät erlaubt. Außerdem verhindert es frühzeitig auf Änderungswünsche eingehen zu können.
Scrum eignet sich für komplexe Projekte, dessen Anforderungen noch nicht vollständig eindeutig sind. Es erlaubt dabei jedoch die inkrementelle Verbesserung des Produktes. Dazu arbeitet Scrum in festgelegten zeitlichen Perioden, den Sprints. Scrum fördert dabei die Zusammenarbeit unter Teams durch Austausch in Daily Standups und Sprint Retrospektiven. Das Arbeiten in Sprints erlaubt eine Auslieferung des Produktes beziehungsweise Inkrementen dessen. So kann Feedback früh gesammelt und einen Einfluss auf die Entwicklung haben.
Kanban erlaubt ebenfalls die kontinuierliche Verbesserung beziehungsweise Abarbeitung von Items, ist jedoch nicht auf fest vorgelegte zeitliche Perioden beschränkt. Es erlaubt dadurch Flexibilität und fokussiert sich auf Lead Time und Cycle Time und zeigt damit Bottlenecks auf. Dadurch kann es die Arbeit effizienter machen. Kanban erfordert jedoch ein Team, welches sich gut selbst organisiert.

Aufgabe 2: Angepasste Stacey-Matrix

Simple: Auslieferung von Template-Produkten

Es wird auf einen eigenes Website-Template zurückgegriffen, das nur leicht auf den Kunden angepasst werden muss. Die Anforderungen sind klar und wiederholen sich über verschiedene Kunden ebenfalls, z.B. Logo- und Corporate Identity des Kunden hinzufügen. Dies lässt sich sehr leicht standardisieren und entsprechend in Prozessen abarbeiten und später überprüfen.

Complicated: Skalierung einer Datenbank-Architektur

Die Skalierung einer Datenbank auf ein festgelegtes Ziel an Requests/Zeit gilt als „Complicated“, da das Ziel wohldefiniert ist. Jedoch ist die technische Lösung noch nicht geklärt. Es stehen verschiedene Optionen zur Verfügung (z.B. vertical scaling, sharding, read replicas), welche gegeneinander abgewogen werden müssen.

Complex: Entwicklung einer KI-Chatbot-App

Das Ziel ist eindeutig, jedoch verbleiben Punkte welche geklärt werden müssen. Es ist zum Beispiel noch nicht klar, wie die Architektur aussehen wird um eine variable Anzahl an Nutzern zu verarbeiten. Es ist ebenfalls zu klären, welches KI-Modell verwendet wird, dies hat Auswirkungen auf die benötigte Rechenleistung und beeinflusst die Zufriedenheit der Nutzer.

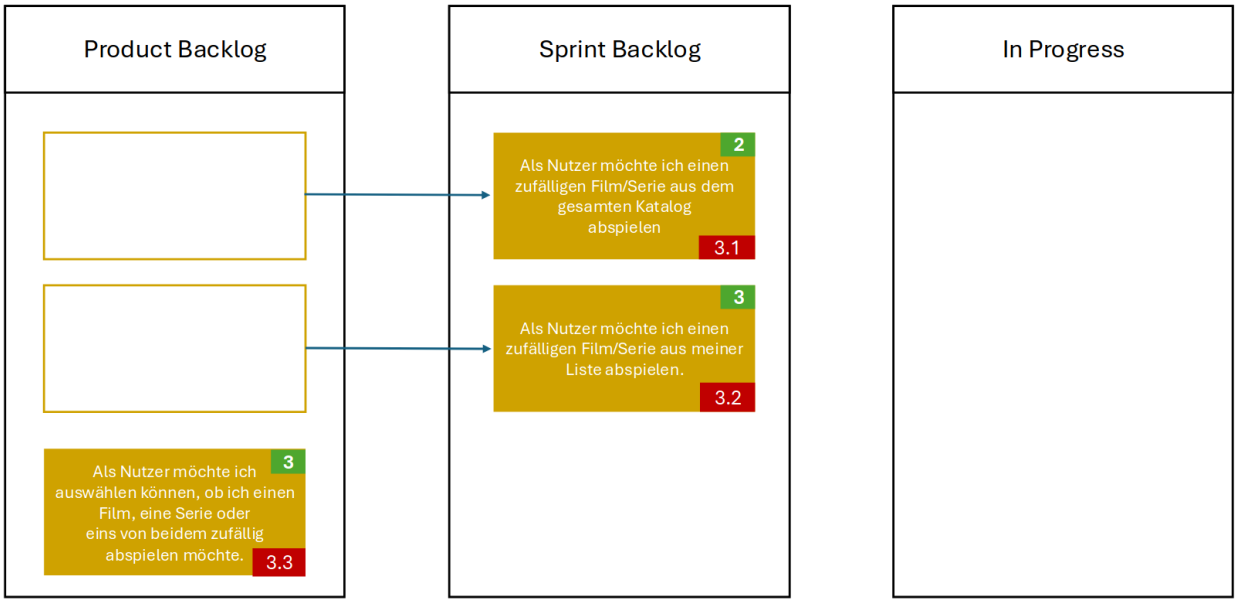
Chaotic: Systemausfall

Die Situation ist meist chaotisch, Anforderungen und Lösungswege sind zunächst nicht klar, da der Grund für den Ausfall erstmal nicht genau bestimmt werden kann. Ziel ist vorerst den Ausfall zu beheben durch Alternativpläne oder Notfalllösungen. Danach kann analysiert und geplant werden, diesen Ausfall aufzuarbeiten.

Aufgabe 3: Scrum

3.1:

User Story #2 „Film oder Serie zufällig abspielen“
User Story 3.1 - 2 Story Points: „Als Nutzer möchte ich einen zufälligen Film/Serie aus dem gesamten Katalog abspielen.“
User Story 3.2 - 3 Story Points: „Als Nutzer möchte ich einen zufälligen Film/Serie aus meiner Liste abspielen.“
User Story 3.3 - 3 Story Points: „Als Nutzer möchte ich auswählen können, ob ich einen Film, eine Serie oder eins von beidem zufällig abspielen möchte.“



3.2:

Akzeptanzkriterien sind Bedingungen, die es zu erfüllen gilt, damit ein Item / eine User Story als abgeschlossen gilt. Sie dienen als klare, überprüfbare Definition vom „Fertig“-Status. Sie werden definiert, damit das Team eine transparente Definition hat und helfen bei der Testbarkeit / Überprüfbarkeit bei Abschluss des Items.
User Story 3.1:

- Knopf zum Aufrufen der Zufalls-Schaltfläche ist sichtbar auf der Hauptseite.
- Knopf besitzt ein eindeutiges Icon.

– Der Knopf ist klickbar, ruft Zufalls-Schaltfläche korrekt auf und erlaubt einen zufälligen Film oder Serie abzuspielen.

User Story 3.2:

– Zufalls-Schaltfläche besitzt eine Option zur Auswahl zwischen allen oder gespeicherten Titeln in der eigenen Liste.

– Schaltfläche reagiert visuell auf Input, hebt aktuell gewählte Option deutlich vor.

– Bei Klick wird ein Film/Serie basierend auf Auswahl gestartet.

User Story 3.3:

– Zufalls-Schaltfläche besitzt eine Option zur Auswahl zwischen Serien, Filmen oder beidem.

– Schaltfläche reagiert visuell auf Input, hebt aktuell gewählte Option deutlich vor.

– Die Schaltfläche ist klickbar, reagiert visuell bei Interaktion und spielt einen zufälligen Film, Serie oder eins von beidem ab.

3.3:

User Story 3.1: Simple, da die Anforderung und Lösungen klar sind. Es handelt sich um eine einfach einzublendende UI, sowie ein simpler Zufallsalgorithmus zum Abspielen der Inhalte.
User Story 3.2: Complicated - technische Umsetzung geht über Zufallsalgorithmus hinaus. Nun werden Daten und deren Abruf benötigt, welche Inhalte der Nutzer gespeichert hat. Die Schaltfläche muss um einen 2-Way Switch erweitert werden.
User Story 3.3: Complicated - nur leicht komplizierter als 3.2, Anforderungen gestalten sich ähnlich. Es werden zusätzlich Daten benötigt, welcher Inhalt als Film oder Serie klassifiziert wird. Zusätzlich benötigt die UI einen 3-Way Switch um drei Optionen auswählbar zu machen.

Aufgabe 1: User Stories und Akzeptanzkriterien

User Story 1: Als User möchte ich mich in meinen Account anmelden.
Akzeptanzkriterien:

– Der User kann sich mit gültigen Anmeldedaten in seinen Account einloggen.

– Eine „Passwort vergessen“-Funktion erlaubt ein zurücksetzen des Passworts per E-Mail.

– Ungültige Anmeldedaten werden nicht akzeptiert.

User Story 2: Als User möchte ich passende Vorschläge bekommen, basierend auf aktuell beliebten, bereits gesehenen und/oder Inhalten, die mir gefallen haben.
Akzeptanzkriterien:

– Das RecommenderSystem schlägt neue Inhalte vor.

– Das System kann aktuell beliebte Inhalte weiterempfehlen.

– Das System kann basierend auf gesehenen/gefallenen Inhalten weitere Inhalte mit gleichem Genre, gleiche Schauspieler oder anderen Ähnlichkeiten vorschlagen.

User Story 3: Als User möchte ich Einstellungen über Lautstärke und Klang haben.
Akzeptanzkriterien:

– Der Nutzer kann zwischen 0% und 100% Lautstärke auf seinem Gerät auswählen.

– Bei Veränderung der Lautstärke wird dies, mit dem dann aktuellen Wert, angezeigt.

– Die Höhe der Lautstärke wird zwischen Sitzungen gespeichert.

Aufgabe 2: Weiterentwicklung des Streamingdienstes

Aufgabe 2.1. Authentifizierungsprüfung

```
public boolean checkAuth(String user, String password) {
      for (String[] credentials : this.userCredentials) {
         final boolean isUser = credentials[0].equals(user);
         if (!isUser)
            continue;

         final boolean correctPassword = credentials[1].equals(password);
         return correctPassword;
      }

      return false;
   }
```

```
public class StreamingApplicationTest {
@Test
public void testCheckAuthValidUser() {
   StreamingApplication sa = new StreamingApplication();

   Assertions.assertTrue(sa.checkAuth("User1", "PasswortA"));
   Assertions.assertTrue(sa.checkAuth("User2", "PasswortB"));
   Assertions.assertTrue(sa.checkAuth("User3", "PasswortC"));
   Assertions.assertTrue(sa.checkAuth("User4", "PasswortC"));
}

@Test
```

```
13 public void testCheckAuthInvalidUser() {
14     StreamingApplication sa = new StreamingApplication();
15 
16     Assertions.assertFalse(sa.checkAuth(”User”, ”PasswortA”));
17     Assertions.assertFalse(sa.checkAuth(”User2”, ”PasswortD”));
18     Assertions.assertFalse(sa.checkAuth(””, ”PasswortC”));
19     Assertions.assertFalse(sa.checkAuth(”\t\n#124124”, ”PasswortC”));
20 }
21 }
```

Aufgabe 2.2. Refactoring der Authentifizierung

```
package teaching.swe.streaming;

public class LoginManager {
   private String[][] userCredentials = {
         { ”User1”, ”PasswortA” },
         { ”User2”, ”PasswortB” },
         { ”User3”, ”PasswortC” },
         { ”User4”, ”PasswortC” }
   };

   public boolean checkAuth(String user, String password) {
      for (String[] credentials : this.userCredentials) {
         final boolean isUser = credentials[0].equals(user);
         if (!isUser)
            continue;

         final boolean correctPassword = credentials[1].equals(password);
         return correctPassword;
      }

      return false;
   }
}
```

```
package teaching.swe.streaming;

import java.util.List;

import teaching.swe.streaming.fraud.FraudDetection;
import teaching.swe.streaming.fraud.IFraudDetection;
import teaching.swe.streaming.recommender.IRecommenderSystem;
import teaching.swe.streaming.recommender.RecommenderSystem;
import teaching.swe.streaming.LoginManager;

public class StreamingApplication {

   private final IRecommenderSystem rs;
   private final IFraudDetection fd;
   private final LoginManager lm;

   private int volumeLevel = 50;
```

```

    public StreamingApplication() {
        this.rs = new RecommenderSystem();
        this.fd = new FraudDetection();
        this.lm = new LoginManager();
    }

    public StreamingApplication(IRecommenderSystem rs, IFraudDetection fd) {
        this.rs = rs;
        this.fd = fd;
        this.lm = new LoginManager();
    }

    public int setVolume(int level) {
        if (level < 0 || level > 100) {
            return -1;
        } else {
            this.volumeLevel = level;
            return level;
        }
    }

    public int getVolume() {
        return this.volumeLevel;
    }

    public LoginResponse login(LoginRequest request) {
        boolean successful = !fd.isFraud(request);
        List<String> recommendations = rs.recommend(request);

        LoginResponse response = new LoginResponse();
        response.setSuccessful(successful);
        response.setRecommendations(recommendations);

        return response;
    }
}
```

```
package teaching.swe.streaming;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

```

```java
public class LoginManagerTest {
   @Test
   public void testCheckAuthValid() {
      LoginManager lm = new LoginManager();

      Assertions.assertTrue(lm.checkAuth("User1", "PasswortA"));
      Assertions.assertTrue(lm.checkAuth("User2", "PasswortB"));
      Assertions.assertTrue(lm.checkAuth("User3", "PasswortC"));
      Assertions.assertTrue(lm.checkAuth("User4", "PasswortC"));
   }

   @Test
   public void testCheckAuthInvalid() {
      LoginManager lm = new LoginManager();

      Assertions.assertFalse(lm.checkAuth("User", "PasswortA"));
      Assertions.assertFalse(lm.checkAuth("User2", "PasswortD"));
      Assertions.assertFalse(lm.checkAuth("", "PasswortC"));
      Assertions.assertFalse(lm.checkAuth("\t\n#124124", "PasswortC"));
   }
}
```

Aufgabe 2.3. Verwendung von Mocks

Listing 1: ILoginManager

```
package teaching.swe.streaming;

public interface ILoginManager {
   public boolean checkAuth(String user, String password);
}
```

Listing 2: LoginManager implements ILoginManager

```
package teaching.swe.streaming;

public class LoginManager implements ILoginManager {
   private String[][] userCredentials = {
         { "User1", "PasswortA" },
         { "User2", "PasswortB" },
         { "User3", "PasswortC" },
         { "User4", "PasswortC" }
   };

   @Override
   public boolean checkAuth(String user, String password) {
      for (String[] credentials : this.userCredentials) {
         final boolean isUser = credentials[0].equals(user);
         if (!isUser)
            continue;

         final boolean correctPassword = credentials[1].equals(password);
         return correctPassword;
      }

      return false;
   }
}
```

Listing 3: LoginManagerMock

```
package teaching.swe.streaming;

public class LoginManagerMock implements ILoginManager {
   private boolean returnValue;

   public LoginManagerMock(boolean returnValue) {
```

```
        this.returnValue = returnValue;
    }

    @Override
    public boolean checkAuth(String user, String password) {
        return returnValue;
    }
}
```

Listing 4: Dependency Injection des ILoginManagers in SA

```
package teaching.swe.streaming;

import java.util.List;

import teaching.swe.streaming.fraud.FraudDetection;
import teaching.swe.streaming.fraud.IFraudDetection;
import teaching.swe.streaming.recommender.IRecommenderSystem;
import teaching.swe.streaming.recommender.RecommenderSystem;
import teaching.swe.streaming.ILoginManager;
import teaching.swe.streaming.LoginManager;

public class StreamingApplication {

    private final IRecommenderSystem rs;
    private final IFraudDetection fd;
    private final ILoginManager lm;

    private int volumeLevel = 50;

    public StreamingApplication() {
        this.rs = new RecommenderSystem();
        this.fd = new FraudDetection();
        this.lm = new LoginManager();
    }

    public StreamingApplication(IRecommenderSystem rs, IFraudDetection fd, ILoginManager
        lm) {
        this.rs = rs;
        this.fd = fd;
        this.lm = lm;
    }

    public int setVolume(int level) {
```

```
33          if (level < 0 || level > 100) {
34              return -1;
35          } else {
36              this.volumeLevel = level;
37              return level;
38          }
39      }
40
41      public int getVolume() {
42          return this.volumeLevel;
43      }
44
45      public LoginResponse login(LoginRequest request) {
46          boolean successful = !fd.isFraud(request);
47          List<String> recommendations = rs.recommend(request);
48
49          this.lm.checkAuth(null, null); // TODO: placeholder 2.4
50
51          LoginResponse response = new LoginResponse();
52          response.setSuccessful(successful);
53          response.setRecommendations(recommendations);
54
55          return response;
56      }
57  }
```

Listing 5: Unittests für login() (Failen, Methode wird in 2.4 angepasst)

```
1   public class StreamingApplicationTest {
2   [...]
3       @Test
4       public void testLoginValid() {
5           IRecommenderSystem rs = new RecommenderSystemMock();
6           IFraudDetection fd = new FraudDetectionMock();
7           ILoginManager lm = new LoginManagerMock(true);
8           StreamingApplication sa = new StreamingApplication(rs, fd, lm);
9
10          LoginRequest request = new LoginRequest();
11          request.setLocation("DE");
12          request.setUserName("Test User");
13          request.setPassword("Test PWD");
14
15          LoginResponse response = sa.login(request);
16
```

```
        assertTrue(response.isSuccessful());
        assertEquals(Arrays.asList("Dummy Movie A"), response.getRecommendations());
    }

    @Test
    public void testLoginInvalid() {
        IRecommenderSystem rs = new RecommenderSystemMock();
        IFraudDetection fd = new FraudDetectionMock();
        ILoginManager lm = new LoginManagerMock(false);
        StreamingApplication sa = new StreamingApplication(rs, fd, lm);

        LoginRequest request = new LoginRequest();
        request.setLocation("DE");
        request.setUserName("Test User");
        request.setPassword("Test PWD");

        LoginResponse response = sa.login(request);

        assertFalse(response.isSuccessful());
        assertTrue(response.getRecommendations().isEmpty());
    }
}
```

Aufgabe 2.4. Erweiterung und Integrationstest der Login-Methode

Listing 6: Erweiterung der login()-Methode

```
public class StreamingApplication {
[...]
   public LoginResponse login(LoginRequest request) {
      final boolean isFraud = fd.isFraud(request);
      final boolean authSuccessful = lm.checkAuth(request.getUserName(), request.
          getPassword());

      final boolean validRequest = authSuccessful && !isFraud;
      LoginResponse response = new LoginResponse();
      response.setSuccessful(validRequest);

      if (!validRequest)
         return response;

      List<String> recommendations = rs.recommend(request);
      response.setRecommendations(recommendations);

      return response;
   }
}
```

Tests aus 2.3 bleiben und Failen nicht mehr

Listing 7: StreamingApplication Integration Test mit LoginManager

```
[...]
public class StreamingApplicationAndLoginManagerIntegrationTest {
   private StreamingApplication streamingApplication;

   @BeforeEach
   public void setUp() {
      IRecommenderSystem rs = new RecommenderSystemMock();
      IFraudDetection fd = new FraudDetectionMock();
      ILoginManager lm = new LoginManager();
      streamingApplication = new StreamingApplication(rs, fd, lm);
   }

   @Test
   public void testLoginValid() {
      LoginRequest request = new LoginRequest();
      request.setLocation("DE");
```

```
        request.setUserName("User1");
        request.setPassword("PasswortA");
        LoginResponse response = streamingApplication.login(request);

        // response should be successful as User-Password combination is correct
        assertTrue(response.isSuccessful());
        // RecommenderSystemMock should create recommendations as the request is valid
        assertFalse(response.getRecommendations().isEmpty());
    }

    @Test
    public void testLoginInvalid() {
        LoginRequest request = new LoginRequest();
        request.setLocation("DE");
        request.setUserName("User1");
        request.setPassword("InvalidPassword");
        LoginResponse response = streamingApplication.login(request);

        // response should be unsuccessful as password is wrong
        assertFalse(response.isSuccessful());
        // recommendations should be empty as the request was invalid
        assertTrue(response.getRecommendations().isEmpty());
    }

    @AfterEach
    public void tearDown() {
        streamingApplication = null;
    }

}
```

Aufgabe 1.1: Klassen definieren und testen

Listing 8: SimpleUserManager

```
package teaching.swe;

import java.util.ArrayList;
import java.util.List;

public class SimpleUserManager {
    private List<String> users = new ArrayList<String>();

```

```
    public SimpleUserManager() {
    }

    public void addUser(String name) {
        if (name == null)
            return;

        users.add(name);
    }

    public void removeUser(String name) {
        users.removeIf(user -> user.equals(name));
    }

    public List<String> getUsers() {
        return this.users;
    }
}
```

Listing 9: SimpleUserManagerTest

```
package teaching.swe;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class SimpleUserManagerTest {
    private SimpleUserManager um;

    @BeforeEach
    void setUp() {
        this.um = new SimpleUserManager();
    }

    @Test
    void testAddNullUserDoesNotChangeList() {
        this.um.addUser(null);
        assertEquals(0, um.getUsers().size());
    }

    @Test
```

```
24    void testRemoveUserOnEmptyListDoesNothing() {
25       this.um.removeUser("non-existent-user");
26       assertEquals(0, um.getUsers().size());
27    }
28
29    @Test
30    void testRemoveNullDoesNothing() {
31       this.um.removeUser(null);
32       assertEquals(0, um.getUsers().size());
33    }
34
35    @Test
36    void testAddSingleUser() {
37       this.um.addUser("user1");
38       assertEquals(1, um.getUsers().size());
39       assertTrue(um.getUsers().contains("user1"));
40    }
41
42    @Test
43    void testAddDuplicateUsers() {
44       this.um.addUser("user1");
45       this.um.addUser("user1");
46       assertEquals(2, um.getUsers().size());
47    }
48
49    @Test
50    void testRemoveExistingUser() {
51       this.um.addUser("user1");
52       this.um.removeUser("user1");
53       assertEquals(0, um.getUsers().size());
54    }
55
56    @Test
57    void testRemoveSameUserTwice() {
58       this.um.addUser("user1");
59       this.um.removeUser("user1");
60       this.um.removeUser("user1"); // should do nothing
61       assertEquals(0, um.getUsers().size());
62    }
63
64    @Test
65    void testAddMultipleUsersAndRemoveInOrder() {
```

```
        this.um.addUser("user1");
        this.um.addUser("user2");
        this.um.addUser("user3");

        assertEquals(3, um.getUsers().size());

        this.um.removeUser("user1");
        assertEquals(2, um.getUsers().size());

        this.um.removeUser("user2");
        assertEquals(1, um.getUsers().size());

        this.um.removeUser("user3");
        assertEquals(0, um.getUsers().size());
    }

    @Test
    void testAddAndRemoveOutOfOrder() {
        this.um.addUser("user1");
        this.um.addUser("user2");
        this.um.addUser("user3");

        this.um.removeUser("user2");
        assertEquals(2, um.getUsers().size());
        assertTrue(um.getUsers().contains("user1"));
        assertTrue(um.getUsers().contains("user3"));
    }
}
```

Aufgabe 1.2: Implementierung der Audit-Logik

Listing 10: IAuditLog

```
package teaching.swe;

import java.util.List;

public interface IAuditLog {
    public void log(LogEntry entry);

    public List<LogEntry> getLogs();
}
```

Listing 11: InMemoryAuditLog

```java
package teaching.swe;

import java.util.ArrayList;
import java.util.List;

public class InMemoryAuditLog implements IAuditLog {
   private List<LogEntry> logs = new ArrayList<LogEntry>();

   @Override
   public void log(LogEntry entry) {
      this.logs.add(entry);
   }

   @Override
   public List<LogEntry> getLogs() {
      return this.logs;
   }
}
```

Listing 12: LogEntry

```java
package teaching.swe;

public class LogEntry {
   public enum LogType {
      Add,
      Remove
   };

   public LogType type;
   public String name;

   public LogEntry(LogType type, String name) {
      this.type = type;
      this.name = name;
   }
}
```

Listing 13: Änderung SimpleUserManager, damit UserManagerWithAudit Zugang zum Member users hat

```java
protected List<String> users = new ArrayList<String>();
```

Listing 14: UserManagerWithAudit

```java
package teaching.swe;
```

```
2
3  import teaching.swe.LogEntry;
4  import teaching.swe.LogEntry.LogType;;
5
6  public class UserManagerWithAudit extends SimpleUserManager {
7     IAuditLog logger;
8
9     public UserManagerWithAudit(IAuditLog logger) {
10       this.logger = logger;
11    }
12
13    @Override
14    public void addUser(String name) {
15       if (name == null)
16          return;
17
18       if (users.contains(name))
19          return;
20
21       users.add(name);
22       logger.log(new LogEntry(LogType.Add, name));
23    }
24
25    @Override
26    public void removeUser(String name) {
27       if (name == null)
28          return;
29
30       if (!users.contains(name))
31          return;
32
33       users.removeIf(user -> user.equals(name));
34       logger.log(new LogEntry(LogType.Remove, name));
35    }
36 }
```

Listing 15: UserManagerWithAuditTest

```
1  package teaching.swe;
2
3  import static org.junit.jupiter.api.Assertions.assertEquals;
4  import static org.junit.jupiter.api.Assertions.assertTrue;
5
6  import org.junit.jupiter.api.BeforeEach;
```

```java
import org.junit.jupiter.api.Test;

public class UserManagerWithAuditTest {
   private SimpleUserManager um;
   private IAuditLog logger;

   @BeforeEach
   void setUp() {
      this.logger = new InMemoryAuditLog();
      this.um = new UserManagerWithAudit(logger);
   }

   @Test
   void testAddNullUserDoesNotChangeList() {
      this.um.addUser(null);
      assertEquals(0, um.getUsers().size());

      assertEquals(0, logger.getLogs().size());
   }

   @Test
   void testRemoveUserOnEmptyListDoesNothing() {
      this.um.removeUser("non-existent-user");
      assertEquals(0, um.getUsers().size());

      assertEquals(0, logger.getLogs().size());
   }

   @Test
   void testRemoveNullDoesNothing() {
      this.um.removeUser(null);
      assertEquals(0, um.getUsers().size());

      assertEquals(0, logger.getLogs().size());
   }

   @Test
   void testAddSingleUser() {
      this.um.addUser("user1");
      assertEquals(1, um.getUsers().size());
      assertTrue(um.getUsers().contains("user1"));

```

```
        assertEquals(1, logger.getLogs().size());
    }

    @Test
    void testAddDuplicateUsers() {
        this.um.addUser("user1");
        this.um.addUser("user1");
        assertEquals(1, um.getUsers().size());

        assertEquals(1, logger.getLogs().size());
    }

    @Test
    void testRemoveExistingUser() {
        this.um.addUser("user1");
        this.um.removeUser("user1");
        assertEquals(0, um.getUsers().size());

        assertEquals(2, logger.getLogs().size());
    }

    @Test
    void testRemoveSameUserTwice() {
        this.um.addUser("user1");
        this.um.removeUser("user1");
        this.um.removeUser("user1"); // should do nothing
        assertEquals(0, um.getUsers().size());
        assertEquals(2, logger.getLogs().size());
    }

    @Test
    void testAddMultipleUsersAndRemoveInOrder() {
        this.um.addUser("user1");
        this.um.addUser("user2");
        this.um.addUser("user3");

        assertEquals(3, um.getUsers().size());
        assertEquals(3, logger.getLogs().size());

        this.um.removeUser("user1");
        assertEquals(2, um.getUsers().size());
        assertEquals(4, logger.getLogs().size());
```

```
91
92          this.um.removeUser("user2");
93          assertEquals(1, um.getUsers().size());
94          assertEquals(5, logger.getLogs().size());
95
96          this.um.removeUser("user3");
97          assertEquals(0, um.getUsers().size());
98          assertEquals(6, logger.getLogs().size());
99      }
100
101     @Test
102     void testAddAndRemoveOutOfOrder() {
103         this.um.addUser("user1");
104         this.um.addUser("user2");
105         this.um.addUser("user3");
106
107         this.um.removeUser("user2");
108         assertEquals(2, um.getUsers().size());
109         assertEquals(4, logger.getLogs().size());
110         assertTrue(um.getUsers().contains("user1"));
111         assertTrue(um.getUsers().contains("user3"));
112     }
113 }
```

Aufgabe 1.3: Erweiterung um Protokollierung auf der Konsole

Listing 16: Erweiterung der UserManagerWithAudit Klasse um eine changeLogger Methode

```
1 public class UserManagerWithAudit extends SimpleUserManager {
2     IAuditLog logger;
3     [...]
4     public void changeLogger(IAuditLog logger) {
5         this.logger = logger;
6     }
7 [...]
```

Listing 17: ConsoleAuditLog zur Protokollierung auf der Konsole

```
1 package teaching.swe;
2
3 import java.util.List;
4
5 public class ConsoleAuditLog implements IAuditLog {
6     @Override
```

```
    public void log(LogEntry entry) {
        System.out.println("Log: " + entry.type.toString() + " User: " + entry.name);
    }

    @Override
    public List<LogEntry> getLogs() {
        return null;
    }
}
```

Listing 18: Tests für die changeLogger() Methode

```
package teaching.swe;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class UserManagerWithConsoleAndMemoryAuditTest {
    private UserManagerWithAudit um;
    private IAuditLog memoryLogger;
    private IAuditLog consoleLogger;

    @BeforeEach
    void setUp() {
        this.memoryLogger = new InMemoryAuditLog();
        this.consoleLogger = new ConsoleAuditLog();
        this.um = new UserManagerWithAudit(memoryLogger);
    }

    @Test
    void testChangeLogger() {
        this.um.changeLogger(this.consoleLogger);

        // no exception should be thrown after changing the logger and a log is caused
        this.um.addUser("test");
        this.um.removeUser("test");

        this.um.changeLogger(this.memoryLogger);
        this.um.addUser("test");
        this.um.removeUser("test");
    }
}
```

Aufgabe 1: Tests und Verbesserungen für die Binärsuche

Listing 19: 1.1: Tests

```
[...]
public class BinarySearchTest {
   @Test
   public void testOnArrayAndElementAtStart() {
      int[] a = { 1, 2, 3, 4, 5 };
      int k = 1;
      int result = BinarySearch.binarySearch(a, k);
      assertEquals(0, result);
   }

   @Test
   public void testOnArrayAndElementInMiddle() {
      int[] a = { 1, 2, 3, 4, 5 };
      int k = 3;
      int result = BinarySearch.binarySearch(a, k);
      assertEquals(2, result);
   }

   @Test
   public void testOnArrayAndElementAtEnd() {
      int[] a = { 1, 2, 3, 4, 100 };
      int k = 100;
      int result = BinarySearch.binarySearch(a, k);
      assertEquals(4, result);
   }

   @Test
   public void testOnArrayAndElementNotFound() {
      int[] a = { 1, 2, 3, 4, 5 };
      int k = 6;
      int result = BinarySearch.binarySearch(a, k);
      assertEquals(BinarySearch.NOT_FOUND, result);
   }

   @Test
   public void testOnEmptyArray() {
      int[] a = {};
      int k = 1;
      int result = BinarySearch.binarySearch(a, k);
```

```
        assertEquals(BinarySearch.NOT_FOUND, result);
    }

    @Test
    public void testOnArrayWithOneElement() {
        int[] a = { 1 };
        int k = 1;
        int result = BinarySearch.binarySearch(a, k);
        assertEquals(0, result);
    }

    @Test
    public void testOnArrayWithOneElementNotFound() {
        int[] a = { 1 };
        int k = 100;
        int result = BinarySearch.binarySearch(a, k);
        assertEquals(BinarySearch.NOT_FOUND, result);
    }

    @Test
    public void testOnArrayWithDuplicates() {
        int[] a = { 1, 2, 2, 2, 3, 4, 5 };
        int k = 2;
        int result = BinarySearch.binarySearch(a, k);
        assertEquals(3, result);
    }
}
```

Listing 20: 1.2: Verbesserung der Implementierung durch ChatGPT

```
[...]
public class BinarySearch {
    public static final int NOT_FOUND = -1;
    public static int binarySearch(int[] array, int target) {
        int low = 0;
        int high = array.length - 1;

        while (low ⩽ high) {
            int mid = low + (high - low) / 2; // prevents overflow

            if (array[mid] == target) {
                return mid; // found
            } else if (array[mid] < target) {
                low = mid + 1; // search right half
```

```
            } else {
                high = mid - 1; // search left half
            }
        }

        return -1; // NOT_FOUND
    }
}
```

Aufgabe 2: Code Coverage Analyse für die Binärsuche

1. int[] a = 1, 3, 5, 7, 9 und int k = 7

Sequenz der Ausführungen:

1, 2, 3, 4, 6, 7,

2, 3, 4, 5, 2, 9

Es fehlt Anweisung 8 beziehungsweise der dazugehörige Pfad

Nicht anweisungsüberdeckend o. zweigüberdeckend

2. int[] a = 1, 2, 3, 4, 5 und int k = 4

Sequenz der Ausführungen:

1, 2, 3, 4, 6, 7,

2, 3, 4, 5, 2, 9

Es fehlt Anweisung 8 beziehungsweise der dazugehörige Pfad

Nicht anweisungsüberdeckend o. zweigüberdeckend

Aufgabe 3: Risiken und Maßnahmen bei der Softwareintegration

3.1: Risiken der Entwicklung des Zahlungsmoduls im nächsten Sprint

Die Entwicklung des Zahlungsmodul mit der Abhängigkeit des extern entwickelteten Authentifizierungsmoduls birgt mehrere Risiken. Da das Modul noch nicht fertig ist oder sogar dementsprechend instabil, kann dies die eigene Entwicklung behindern. Änderungen oder Verzögerungen im Authentifizierungsmodul können direkten Einfluss auf das Zahlungsmodul haben.
Sollte noch keine Schnittstelle für das Authentifizierungsmodul definiert oder dokumentiert sein, wird dessen Integration stark verkompliziert. Unter Umständen kommt es zu Mehraufwand, da Annahmen über Funktionsweisen falsch waren.
Die fehlende Schnittstelle verhindert außerdem, dass das Modul gemockt werden kann. So sinkt die Testbarkeit des eigenen Moduls. Eventuelle Fehler, die bei der Integration beider Module entstehen könnten so erst sehr spät entdeckt werden.

3.2: Mitigation der Risiken

Als technische Maßnahme zur Mitigation sollte eine Vereinbarung zwischen beiden Teams getroffen werden, welche die Schnittstelle (Klassen-Interfaces, REST API o.ä.) verbindlich beschreibt. So kann ich die Entwicklung parallel geschehen und Abweichungen würde die vorherige Genehmigung des anderen Teams benötigen.
Als organisatorische Maßnahme sollte es Meetings zwischen beiden Teams (beziehungsweise Vertretern deren) geben. Dies schafft Transparenz zwischen den Teams bezüglich Zeitplänen und erlaubt die Klärung von technischen Fragen oder sogar möglichen notwendigen / gewünschten Änderungen.

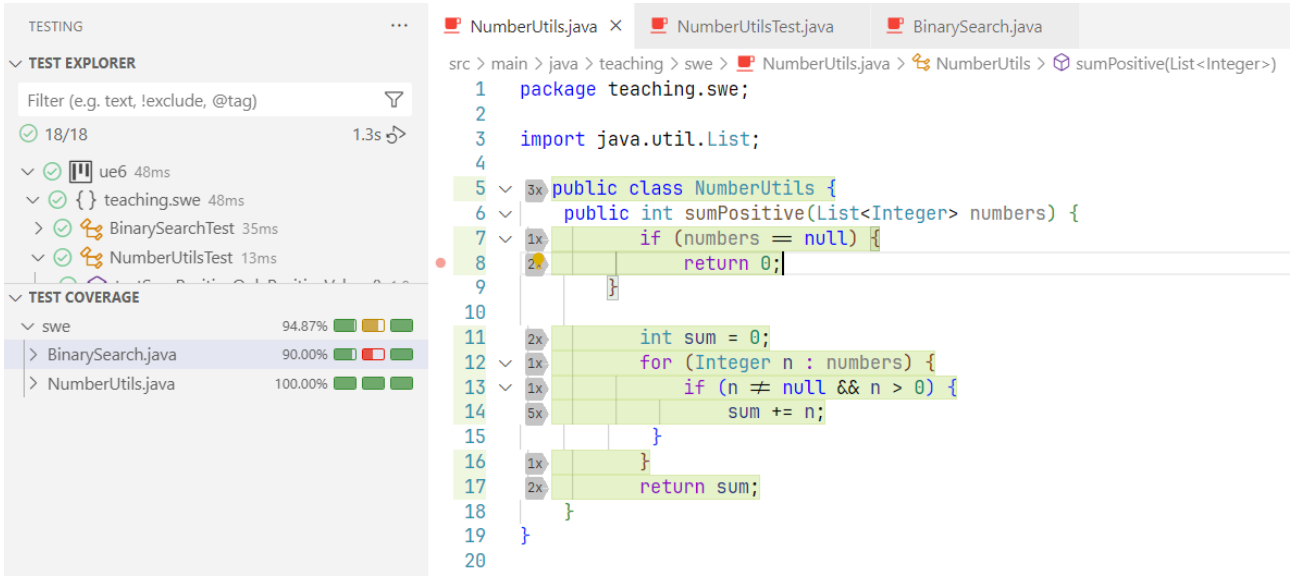
Aufgabe 4: Testen von Grenzwerten und speziellen Werten

Listing 21: 1.1: Tests

```
[...]
public class NumberUtilsTest {
   NumberUtils numberUtils;

   @BeforeEach
   public void setUp() {
      this.numberUtils = new NumberUtils();
   }

   @Test
   public void testSumPositiveOnlyPositiveValues() {
      List<Integer> numbers = List.of(1, 2, 3, 4, 5);
      int result = numberUtils.sumPositive(numbers);
      assertEquals(15, result);
   }

   @Test
   public void testSumPositiveOnlyNegativeValues() {
      List<Integer> numbers = List.of(-1, -2, -3, -4, -5);
      int result = numberUtils.sumPositive(numbers);
      assertEquals(0, result);
   }

   @Test
   public void testSumPositiveMixedValues() {
      List<Integer> numbers = List.of(1, -1, 2, -2, 3, -3);
      int result = numberUtils.sumPositive(numbers);
      assertEquals(6, result);
   }

   // boundary cases:
   @Test
   public void testSumPositiveOnEmptyList() {
      List<Integer> numbers = List.of();
      int result = numberUtils.sumPositive(numbers);
      assertEquals(0, result);
   }

   @Test
```

```
40    public void testSumPositiveOnSizeOneList() {
41       List<Integer> numbers = List.of(1);
42       int result = numberUtils.sumPositive(numbers);
43       assertEquals(1, result);
44    }
45
46    @Test
47    public void testSumPositiveOnSizeOneListNegativeValue() {
48       List<Integer> numbers = List.of(-1);
49       int result = numberUtils.sumPositive(numbers);
50       assertEquals(0, result);
51    }
52
53    // edge cases:
54    @Test
55    public void testSumPositiveNull() {
56       int result = numberUtils.sumPositive(null);
57       assertEquals(0, result);
58    }
59
60    @Test
61    public void testSumPositiveOnListWithZeroes() {
62       List<Integer> numbers = List.of(0, 0, 0);
63       int result = numberUtils.sumPositive(numbers);
64       assertEquals(0, result);
65    }
66
67    @Test
68    public void testSumPositiveOnNullList() {
69       List<Integer> numbers = Arrays.asList(null, null, null);
70       int result = numberUtils.sumPositive(numbers);
71       assertEquals(0, result);
72
73       numbers = Arrays.asList(null, null, null, 2, 4);
74       result = numberUtils.sumPositive(numbers);
75       assertEquals(6, result);
76    }
77
78    @Test
79    public void testSumPositiveWithIntMaxValue() {
80       List<Integer> numbers = List.of(Integer.MAX_VALUE, 1);
81       int result = numberUtils.sumPositive(numbers);
```

```
82        assertEquals(Integer.MIN_VALUE, result); // expecting overflow to Integer.MIN_VALUE
83    }
84 }
```



Bonusblatt 1

Aufgabe 1: Ableitung von User Stories

EmailChecker

Als **[Nutzer]** möchte ich, **[dass meine E-Mail auf Richtigkeit geprüft wird]**, damit ich **[keine falsche angebe / kontaktiert werden kann]**.
Akzeptanzkriterien:

- Leerer String wird abgelehnt
- E-Mail beinhaltet genau ein „@“-Zeichen
- E-Mail beinhaltet gültigen Teil vor dem „@“-Zeichen:
  kein leerer String
- E-Mail beinhaltet gültigen Teil nach dem „@“-Zeichen:
  kein leerer String
  beinhaltet Punkt
  endet nicht mit einem Punkt
- beinhaltet keine Leerzeichen
- E-Mail wird akzeptiert (gilt als gültig), wenn alle Kriterien erfüllt sind

PasswordChecker

Als **[Team]** möchten wir **[Anforderungen an Passwörter festlegen]**, damit wir **[unseren Nutzer eine gewisse Sicherheit bieten können]**.
Akzeptanzkriterien:

- Leere Passwörter sind nicht erlaubt TODO: sind sie anscheinend doch?
- Null wird abgelehnt
- Passwörter beinhalten min. 8 Zeichen
- Passwörter beinhalten max. 20 Zeichen
- Passwörter muss ein Großbuchstabe beinhalten
- Passwörter muss ein Kleinbuchstabe beinhalten
- Passwörter muss eine Zahl beinhalten
- Passwörter muss ein Sonderzeichen beinhalten
- Passwort wird akzeptiert, wenn es alle Kriterien erfüllt

RegistrationValidator

Als **[System]** möchte ich **[Registrierungsdaten validieren können]**, damit ich **[nur korrekte Nutzerdaten gespeichert werden].**
Akzeptanzkriterien:

- Nutzt EmailChecker
  und erfüllt dessen Kriterien
- Nutzt PasswordChecker
  und erfüllt dessen Kriterien
- Nutzt UsernameChecker
  und erfüllt dessen Kriterien
- Nur erfolgreich wenn alle drei Komponenten (Username, Email, Password) gültig sind
- Implementierungen (Subklassen) sind nicht zugänglich außer über eigene Public Methoden, welche die Methoden dieser aufrufen

UsernameChecker

Als **[registrierter Nutzer]** möchte ich **[einen gültigen Benutzernamen wählen können]**, damit ich **[mich eindeutig identifizieren kann].**
Akzeptanzkriterien:

- Null wird abgelehnt
- Username muss min. 4 Zeichen beinhalten
- Username darf max. 15 Zeichen beinhalten
- erlaubte Zeichen A-Z,a-z,0-9, Unterstrich _
- keine Leerzeichen oder Sonderzeichen (außer „_“)
- Username wird akzeptiert (gilt als gültig), wenn alle Kriterien erfüllt sind

Aufgabe 2: Testfallanalyse mittels Äquivalenzklassen und Grenzwertanalyse

EmailChecker

(a) Äquivalenzklassen

Gültige Klassen:

- Simple:
  name@adresse.de
- Mit Subdomain:
  name1.name2@example.subdomain.org
- Local-Part:
  name1.name2+tag@example.org
- Sonderzeichen (! _ - etc.):
  name!@domain.com

Ungültige Klassen:

- kein @-Zeichen:
  name1.name2.de
- mehrere @-Zeichen:
  name1@name2@de
- keine Domain:
  name1@name2
- Leerzeichen:
  name1 @name2.de

(b) Grenzwertanalyse

- Null
- Empty String
- Whitespace String

(c) Testfälle

Eingabe	Erwartetes Ergebnis
“name@domain.de”	True
“name.name@domain.de”	True
“name-name@domain.com”	True
“name!name@domain.com”	True
Null	False
Empty String	False
Whitespace String	False
“name.@@domain.de”	False
“@domain.de”	False
“name@”	False
“name@domain”	False
“name@domain.”	False
“name @domain.de”	False

PasswordChecker

(a) Äquivalenzklassen

Gültige Klassen:

- minimale Länge:
  Abcdef1!
- maximale Länge:
  Abcdefghijklmnopqr1!
- beliebige Reihenfolge:
  !Bcdefghijklmnopqr1!

Ungültige Klassen:

- Null
- minimale Länge:
  Abcd1!
- maximale Länge:
  Abcdefghijklmnopqrstuvw1!

(b) Grenzwertanalyse

- Null
- Passwort mit Whitespace
- nicht erlaubte Sonderzeichen
- Escape-Characters

(c) Testfälle

Eingabe	Erwartetes Ergebnis
Abcdef1!	True
Abcdefghijklmnopqr1!!	True
!Bcdefghijklmnopqr1!	True
Null	False
Abcd1!	False
Abcdefghijklmnopqrstuvw1!	False

RegistrationValidator

(a) Äquivalenzklassen

Gültige Klassen:

- Gültige Email, Password und Username:
  name@domain.de, Abcdef1!, validUser123

Ungültige Klassen:

- Ungültige Email:
  name1.name2.de
- Ungültige Password:
  Abcdef1!
- Ungültige Username:
  user name

(b) Grenzwertanalyse

- Null

– Mehrere Leerzeichen

(c) Testfälle

Eingabe	Erwartetes Ergebnis
name@domain.de, Abcdef1!, validUser123	True
Null	False

UsernameChecker

(a) Äquivalenzklassen

Gültige Klassen:

– Alphabeten und Zahlen:
validUser123

– Undercore:
user_name

– Nur Zahlen:
123456789

– Nur Alphabeten:
abcdefg

Ungültige Klassen:

– Null

– Name mit Leerzeichen:
user name

– Special Chars:
user@name

– minimale Länge:
abc

– maximale Länge:
thisusernameiswaytoolongtobevalid

(b) Grenzwertanalyse

– Null

– Username mit mehrere Underscores

– Username mit Lerrzeichen

– Sonderzeichen

(c) Testfälle

Eingabe	Erwartetes Ergebnis
validUser123	True
user_name	True
123456789	True
abcdefg	True
Null	False
user name	False
user@name	False