Ein System nutzt die folgende externe Schnittstelle OnlineEventService zum Suchen nach Events

```java
interface OnlineEventService {

    /**
    * Searches for events in the given location on the given date.
    * @param location must not be null
    * @param date must not be in the past
    * @return a list with the results
    */
    List<String> find(String location, LocalDate date);
}
```

Um isoliertes Unit Testing des Systems zu ermöglichen, wurde folgender Test Double implementiert.

```java
public class SimpleEventService implements OnlineEventService {
    private List<String> events;
    private boolean called;

    public void setEvents(List<String> events) { this.events = events; }

    public boolean isCalled() { return called; }

    @Override
    public List<String> find(String location, LocalDate date) {
        Assertions.assertNotNull(location);
        Assertions.assertNotNull(date);
        Assertions.assertTrue(date.isAfter(LocalDate.now()));

        called = true;
        return events;
    }
}
```

Handelt es sich bei dem bereitgestellten Test Double SimpleEventService um einen Stub oder um einen Mock?
Begründen Sie Ihre Antwort bzw. erläutern Sie den Unterschied basierend auf dem gegebenen Beispiel.

Ein Wintersportgebiet wirbt mit einer Rabattaktion. Auf Tageskarten wird deshalb 10% Rabatt, auf Wochenkarten 15% Rabatt und auf Stundenkarten 5% Rabatt vergeben.
Personen, die eine aktive Mitgliedschaft beim Alpenverein haben, erhalten beim Kauf einer Tages- bzw. Wochenkarte zusätzlich 20% Rabatt.
Bei Online-Kauf werden allen Personen zusätzlich 5% auf Tageskarten gewährt. Rabatte akkumulieren sich.

Definieren Sie anhand der gegebenen Informationen alle Äquivalenzklassen.
Geben Sie zwei wichtige Testfälle (nicht JUnit-Tests) an.

Führen Sie ein Review der nachfolgenden (**JUnit 5**) Testfälle durch.
Finden Sie 5 Fehler in der Implementierung der Testfälle. Erklären Sie die gefundenen Probleme und geben Sie Vorschläge zur Behebung.
Achten Sie im Speziellen auf **Bad Practices**!

Hinweis: Sollte ein Fehler mehrfach vorkommen, zählt dieser nur als ein Fehler.

```java
public class SimpleListTest {
    private static List<String> list;

    @BeforeAll
    public static void setUp() {
        list = new ArrayList<>();
        list.add("A");
        list.add("B");
    }

    @Test
    public void testContains_shouldFindElement() {
        list.add("C");
        boolean isFound = list.contains("C");
    }

    @Test
    public void testAdd_shouldAddElement() {
        for(int i = 0; i < 3; i++) {
            list.add(i, "C-" + i);
        }

        assertTrue(list.get(0).equals("C-0")
            && list.get(1).equals("C-1")
            && list.get(2).equals("C-2"));
    }

    @Test
    public void test_shouldFail() {
        if(!list.contains("Z")) {
            assertTrue(true);
        }
        else {
            fail();
        }
    }

    @Test
    public void testAddWrongIndex_shouldFail() throws IndexOutOfBoundsException {
        assertThrows(Exception.class, () ->
            list.add(100, "Elem-100"));
    }
}
```

Bitte hier eintragen:

Gefundene Fehler

Nummer	Zeile/n	Problembeschreibung	Vorschlag zur Behebung
1			
2			
3			
4			
5			

Bei der Klasse StringStack handelt es sich um eine stringbasierte Stack Implementierung.
Überprüfen Sie die Funktionalität der Methode pop() indem Sie für folgende zwei Szenarien einen JUnit 5 Test definieren:

a) Normalfall: Aufruf der Methode pop() auf einem befüllten Stack
b) Fehlerfall: Aufruf der Methode pop() auf einem leeren Stack

Bedenken Sie auch Best Practices bei der Erstellung von Tests.
Sie können davon ausgehen, dass die Methoden size() sowie push() korrekt funktionieren.

```java
public class StringStack {
    private String[] storage;
    private int position;

    public StringStack(int capacity) { this.storage = new String[capacity]; }

    public int size() { return position; }

    public void push(String elem) {
        storage[position] = elem;
        position++;
    }

    public String pop() throws EmptyStackException {
        if(position == 0) { throw new EmptyStackException(); }

        String elem = storage[position-1];
        storage[position-1] = null;
        position--;

        return elem;
    }
}
```

Ihr Unternehmen wurde mit der Entwicklung einer Web Applikation für Schulen beauftragt.
Die Applikation soll im ersten Schritt die Stundenpläne der einzelnen Klassen darstellen und Lehrern/Lehrerinnen die Anwesenheitskontrolle der Schüler/Schülerinnen während des Unterrichts ermöglichen.
Auf die Stundenpläne haben sowohl Lehrer/Lehrerinnen, als auch Schüler/Schülerinnen jederzeit Zugriff.
Die Erweiterung um weitere Features für e-Learning ist bereits beschlossen und befindet sich in Planung.

Nennen und beschreiben Sie drei wichtige Qualitätsfaktoren (nicht-funktionale Anforderungen) an die Applikation, die hier besonders berücksichtigt werden müssen!
Begründen Sie Ihre Antwort.

Geben Sie an, wie viele Testfälle erforderlich sind für eine vollständige

a) Anweisungsüberdeckung (Statement Coverage)
b) Zweigüberdeckung (Branch Coverage)
c) Pfadüberdeckung (Path Coverage)

Definieren Sie zusätzlich 3 konkrete Testfälle (Input-Daten, ausgeführte Statements) auf Basis von Grenzwerten.