ÜbunGszettel TDD

# Aufgabe 1: Theorie

Arbeite dich in die Theorie zum Thema Testen bzw. Test-Driven-Development (TDD) ein. Nutze dazu die Materialien aus den verschiedenen „Input“ Abschnitten im Themenbereich „Test-Driven Development“ unseres Moodle-Kurses (https://moodle.tsn.at/course/view.php?id=24763#section-8). Nach dieser Aufgabe musst du folgende Punkte erklären können:

* Testdriven Development  
  Test werden vor dem Code geschrieben, nicht andersherum. So sind alle Tests vorher Rod, also fehlgeschlagen, bis funktionierender Code geschrieben wurde.
* Red-Green-Refactor  
  Code wird überarbeitet, damit dieser übersichtlicher ist. Es wird ein gutes Fundament für zukünftige Iterationen gelegt. Die Test zeigen, dass der Code nach wie vor funktioniert.
* FIRST-Acronym
* Fast: run (subset of) tests quickly (since you'll be running them all the time)
* Independent: no tests depend on others, so can run any subset in any order
* Repeatable: run N times, get same result (to help isolate bugs and enable automation)
* Self-checking: test can automatically detect if passed (no human checking of output)
* Timely: written about the same time as code under test (with TDD, written first!)
* Kent Beck (welche Rolle spielt er in Bezug auf TDD)  
  Erfinder von TDD.
* Testarten
  + Unit-Tests (Sociable, Solitary, Mocks)
  + Integrationstests
  + UI-Tests / End-To-End Tests / Systemtests
  + Akzeptanztests
* Testpyramide
* JUNIT (Junit5)
* Mockito (Sinn und Funktionsweise von Mocking-Bibliotheken)

# Aufgabe 2: Ausgangsprojekt

Laden Sie sich das gegebene Maven-Ausgangsprojekt („TDD Kino Demo“, siehe Moodle) herunter. Laden Sie es als Maven-Projekt in ihre IDE und schauen Sie sich an, wie das Projekt aufgebaut ist:

* pom.xml (Dependencies, Java-Version etc.)
* Gegebene Domänen-Klassen (Kinosaal, Ticket etc.)
* Gegebene Start-Junit5-Tests in test /java/at.itkolleg/AppTest

Starten Sie den Test AppTest über den grünen Pfeil und versichern Sie sich, dass alles korrekt läuft. Starten Sie auch die App (main-Methode).

# Aufgabe 3: Einarbeitung in den gegebenen Code

Arbeiten Sie sich in den gegebenen Code zur Kinoverwaltung ein. Verwenden Sie die gegebenen Klassen KinoSaal, Ticket, Vorstellung, Kinoverwaltung in der App-Klasse (main-Methode), um ein Gefühl für die Funktionsweise des Programms zu bekommen. Führen Sie folgende Punkte durch:

* Kinosäle anlegen
* Vorstellungen anlegen
* Vorstellungen über die Kinoverwaltung einplanen
* Tickets für Vorstellungen ausgeben
* etc.

# Aufgabe 4: JUNIT-Tests für KinoSaal

Testen Sie alle Methoden der Klasse KinoSaal (Testklasse TestKinoSaal).

|  |
| --- |
| @Test      void testKinosaalName() {          assertEquals("KS2", kinosaalOriginal.getName());      }      @Test      void testKinoSaalPruefePlatz() {          assertTrue(kinosaalOriginal.pruefePlatz('A', 10));          assertTrue(kinosaalOriginal.pruefePlatz('B', 9));          assertFalse(kinosaalOriginal.pruefePlatz('X', 9));          assertFalse(kinosaalOriginal.pruefePlatz('C', 20));      }      @Test      void testKinosaalEquals(){          assertTrue(kinosaalOriginal.equals(kinosaalOriginal));          assertFalse(kinosaalOriginal.equals(new Object()));      } |

# Aufgabe 5: JUNIT-Tests für Vorstellung

Testen Sie alle Methoden der Klasse Vorstellung (Testklasse TestVorstellung).

|  |
| --- |
| @Test      @DisplayName("Pruefe Film")      void testVorstellungFilm() {          assertEquals("Iron Man", vorstellungOriginal.getFilm(), "Falscher Film");      }      @Test      @DisplayName("Pruefe Konosaal")      void testVorstellungSaal() {          assertEquals(kinosaalOriginal, vorstellungOriginal.getSaal(), "Falscher Kinosaal");      }      @Test      @DisplayName("Pruefe Zeitfenster")      void testVorstellungZeitfenster() {          assertEquals(NACHT, vorstellungOriginal.getZeitfenster(), "Falsches Zeitfenster");      }      @Test      @DisplayName("Pruefe Datum")      void testVorstellungDatum() {          assertEquals(localDateOriginal, vorstellungOriginal.getDatum(), "Falsches Datum");      } |

# Aufgabe 6: JUNIT-Tests für KinoVerwaltung

Testen Sie alle Methoden der Klasse KinoVerwaltung (Testklasse TestKinoverwaltung).

|  |
| --- |
| @Test      void testExceptionBereitsEingeplant(){          IllegalArgumentException exception = assertThrows(IllegalArgumentException.class, () -> {              kinoVerwaltungOriginal.einplanenVorstellung(vorstellungOriginal);          });          assertEquals("Die Vorstellung ist bereits eingeplant", exception.getMessage());      }      @Test      void testEinplanenVorstellung(){          List<Vorstellung> vorstellungenList = new LinkedList<>();          Vorstellung vorstellungOriginal2 = new Vorstellung(kinosaalOriginal, ABEND, localDateOriginal, "Iron Man 2", 23.50F);          vorstellungenList.add(vorstellungOriginal);          vorstellungenList.add(vorstellungOriginal2);          kinoVerwaltungOriginal.einplanenVorstellung(vorstellungOriginal2);          assertEquals(vorstellungenList, kinoVerwaltungOriginal.getVorstellungen());      } |

# Aufgabe 7: JUNIT-Tests Advanced

Falls nicht schon in den vorhergehenden Aufgaben passiert, testen Sie folgende Punkte unter Verwendung der fortgeschrittenen Features von JUNIT 5:

1. Schreiben Sie einen Test, der validiert, dass das Anlegen einer Vorstellung korrekt funktioniert. Der Test sollte eine fachliche Bezeichnung haben und die Assertions sollten bei Validierungsfehler eine Hinweistext liefern.

|  |
| --- |
| @Test      void testEinplanenVorstellung(){          List<Vorstellung> vorstellungenList = new LinkedList<>();          vorstellungenList.add(vorstellungOriginal);          assertEquals(vorstellungenList, kinoVerwaltungOriginal.getVorstellungen());      } |

1. Schreiben Sie einen Test, der validiert, dass das Einplanen mehrerer Vorstellungen korrekt funktioniert. Stellen Sie zudem sicher, dass beim möglichen Auftreten eines Fehlers trotzdem alle Validierungen ausgeführt werden.

|  |
| --- |
| @Test      void testEinplanenVorstellungMehrere(){          ArrayList<Throwable> al = new ArrayList<>();          List<Vorstellung> vorstellungenList = new LinkedList<>();          Vorstellung vorstellungOriginal2 = new Vorstellung(kinosaalOriginal, ABEND, localDateOriginal, "Iron Man 2", 23.50F);          vorstellungenList.add(vorstellungOriginal);          vorstellungenList.add(vorstellungOriginal2);          kinoVerwaltungOriginal.einplanenVorstellung(vorstellungOriginal2);          try {              assertEquals(vorstellungenList.get(0), kinoVerwaltungOriginal.getVorstellungen().get(0));          } catch (AssertionError error) {              al.add(error);          }          try {              assertEquals(vorstellungenList.get(1), kinoVerwaltungOriginal.getVorstellungen().get(1));          } catch (AssertionError error) {              al.add(error);          }          if(al.size() > 0){              for (Throwable error : al) {                  System.out.println(error.getMessage());              }              fail();          }      } |

1. Schreiben Sie einen Test, der sicherstellt, dass ein Fehler geworfen wird, wenn eine Veranstaltung doppelt eingeplant wird.

|  |
| --- |
| @Test      void testExceptionBereitsEingeplant(){          IllegalArgumentException exception = assertThrows(IllegalArgumentException.class, () -> {              kinoVerwaltungOriginal.einplanenVorstellung(vorstellungOriginal);          });          assertEquals("Die Vorstellung ist bereits eingeplant", exception.getMessage());      } |

1. Schreiben Sie einen parametrisierten Test, der mehrere Ticketkäufe mit unterschiedlichen Parametern überprüft.

|  |
| --- |
| @ParameterizedTest      @CsvSource({              "B, 3, 100",              "B, 2, 50",              "A, 1, 20",              "A, 4, 20",              "C, 5, 50",              "C, 6, 100"               })      void paramatertestKaufeTicket(char reihe, int platz, float geld) {          Ticket ticket = new Ticket(kinosaalOriginal.getName(), NACHT, localDateOriginal, reihe, platz);          Ticket vorstellungTicket = vorstellungOriginal.kaufeTicket(reihe, platz, geld);          assertAll("tickets",                  () -> assertEquals(ticket.getSaal(), vorstellungTicket.getSaal()),                  () -> assertEquals(ticket.getZeitfenster(), vorstellungTicket.getZeitfenster()),                  () -> assertEquals(ticket.getDatum(), vorstellungTicket.getDatum()),                  () -> assertEquals(ticket.getReihe(), vorstellungTicket.getReihe()),                  () -> assertEquals(ticket.getPlatz(), vorstellungTicket.getPlatz())          );      } |

1. Schreiben Sie eine dynamische TestFactory die den Ticketkauf mit zufälligen Werten bombardiert. Der Test soll sicherstellen, dass der Ticketkauf entweder funktioniert oder nur einen der definierten Fehlermeldungen (z.B. new IllegalArgumentException("Nicht ausreichend Geld.")) ausgibt. Die Tests müssen reproduzierbar sein.

|  |
| --- |
| @TestFactory      public Collection<DynamicTest> kaufeTicketCollection() {          List<DynamicTest> testListe = new ArrayList<>();          for (int i = 0; i < 25; i++) {              Vorstellung vorstellung = kinoVerwaltungOriginal.getVorstellungen().get(i % 2);              char reihe = (char) ((i % 3) + 65);              int platz = i % 12;              int geld = i;              testListe.add(dynamicTest(vorstellung.getFilm() + ", " + reihe + platz + ", " + geld + "€",                      () -> {                          try {                              kinoVerwaltungOriginal.kaufeTicket(vorstellung, reihe, platz, geld);                          } catch (IllegalArgumentException e) {                              boolean errGeld = "Nicht ausreichend Geld.".equals(e.getMessage());                              boolean errPlatz = e.getMessage().contains("existiert nicht");                              assertTrue(errGeld || errPlatz);                          } catch (IllegalStateException e) {                              assertTrue(e.getMessage().contains("ist bereits belegt."));                          }                      }));          }          for (DynamicTest dynamicTest : testListe) {              System.out.println(dynamicTest.getDisplayName());          }          return testListe;      } |

# Aufgabe 8: Mockito Einführung

Lesen Sie sich in das Mocking-Framework Mockito ein (Links siehe Moodle im Abschitt „Input zu Mockito“).

Verwenden Sie die wesentlichen Mockito-Möglichkeiten praktisch in kleinen Programmen.