Fragen, Antworten, Kommentare zur aktuellen Vorlesung

Frage (Erinnerung): Wie orientiere ich mich in den Veranstaltungsunterlagen?

Generell erfolgt der Einstieg über die passende Lernnotiz, in der u. a. das Datum oder die Daten der Vorlesung stehen zu denen sie gehört. Die passende Lernnotiz kann auch über den Veranstaltungskalender gefunden werden, der auf der Veranstaltungsseite verlinkt ist. Lesen Sie die Lernnotiz und führen Sie die genannten Schritte durch. Sie werden zunächst sich Videos mit programmierten Beispielen ansehen, die Sie mitarbeiten können. Danach werden die zugehörigen Folien in Videos vorgestellt. Das bei der Lernnotiz platzierte Material ist optional und enthält Teile der besprochenen Programme zur eigenen Analyse. Nach jeder Vorlesung wird ein „Fragen und Antworten"-Dokument hochgeladen, dass auf aktuelle Fragen aus der Veranstaltung und den Praktika eingeht, aber auch weiterführende Informationen enthält. Auch diese Dokumente sind verpflichtende Literatur. Die Lernnotiz enthält weiter die Information über das zu bearbeitende Aufgabenblatt, dass typischerweise ohne Google mit dem Wissen aus der Veranstaltung zu bearbeiten ist. Beachten Sie ebenfalls die Forderung der Lernnotiz für sich selbst zu prüfen, ob die genannten Lernziele erreicht wurden.

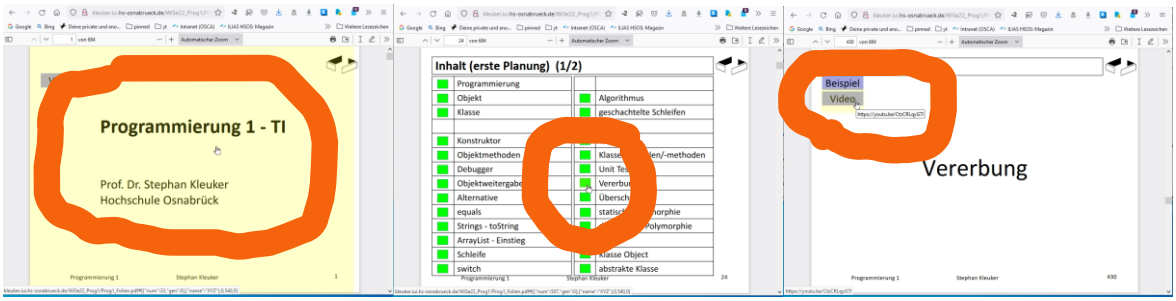
Frage oder Bemerkung: Es gibt Hinweise auf die hohe Stundenbelastung durch die Praktikumsaufgaben.

Antwort: Ja die Belastung ist hoch, aber im üblichen Rahmen eines Studiums. Beachten Sie, dass es sich um eine Veranstaltung mit 10 Leistungspunkten handelt, grob 300 Arbeitsstunden für strukturiert arbeitende Studierende (zentrale Anforderung der Studierfähigkeit) im Semester, aber noch ohne Programmierwissen. Dies ergibt pro Woche im Semester ca. 14 Stunden Arbeit für Programmierung 1. Das sind 3 Stunden Vorlesung, 2 Stunden Praktikum (30+90 Minuten) und 7 Stunden zur Vor- und Nachbereitung und Bearbeitung von Praktikumsaufgaben (damit im Umfang von mindestens 5 Stunden). Sollten Sie bei einer Aufgabe sehr lange brauchen, sollten Sie sie schieben und mit anderen Personen (Prof, Mitarbeiter, andere Studis) später reden. Sollten Sie mehrfach deutlich länger brauchen, wissen Sie, dass Sie den aktuell gestellten Anforderungen momentan nicht entsprechen. Hier müssen Sie dann überlegen, ob Sie mehr Zeit investieren wollen und können oder z. B. Programmierung mit ihrem erreichten Vorwissen im nächsten Semester nochmal versuchen. (Meine Erfahrung zeigt, dass ein dritter Versuch sinnlos ist.)

Berechnung: Semester hat 26 Wochen, wir ziehen 3 Wochen Urlaub ab. Die Vorlesungszeit hat maximal 15, meist 14 bis 12 Veranstaltungswochen. Sind im Maximalfall 15*14 = 210 Stunden. Die restliche Zeit dient der Prüfungsvorbereitung oder/und Durchführung (bei Hausarbeiten) und der Nachbereitung. Sie umfasst dann 11*8 Stunden. Wird die Nachbereitung eigentlich geprüft? Indirekt ja, da nicht alle wichtigen, meist technischen Themen wie Installation und Nutzung von Git, CodeTogether und Docker, in Veranstaltungen erklärt werden. Das Wissen ist aber sehr hilfreich und teilweise notwendig. Generell sollten Sie an eigenen Programmierprojekten arbeiten, um Ihre Fähigkeiten zu trainieren und zu verbessern.

Hinweis: Mir ist unklar ob bekannt ist, dass die Kapitel des Skripts mit den Überschriften des Inhaltsverzeichnisses verlinkt sind. Weiterhin kann durch einen Klick auf die Startseite direkt zu

diesem Verzeichnis gesprungen werden. Neu ist, dass im Skript direkte Sprünge zu den Vorlesungsvideos und Beispielszenarien ergänzt wurden.



Hinweis: Unter https://youtu.be/by9k47lUlAU finden Sie die Erklärung einer Beispiellösung zu zwei Teilaufgaben der Messreihe. Dies setzt die Liste von Beispiellösungen fort, die mit der ersten komplexen Aufgabe (Säulendiagramm) angefangen wurde und für die folgenden Aufgabenblätter beispielhaft fortgesetzt wird.

Es handelt sich nebenbei um keine Musterlösungen, da bereits jetzt eine Komplexität erreicht wird, dass es verschiedene, teilweise gleich gute Lösungen gibt. Ein konkretes Beispiel ist die individuelle Überprüfung von null-werten. (Das hat für Lehrende nebenbei den positiven Seiteneffekt, dass Lösungskopien schnell erkannt werden). Musterlösungen haben zwei negative Effekte. Zunächst wie angedeutet, dass kreative Lösungen eventuell nicht als gut erkannt werden. Ein zweiter Effekt ist ein Sammlungseffekt bei Studierenden und sie nur gestapelt werden, da gehofft wird, die Aufgaben am Ende schon zu verstehen. Bedenken Sie immer, dass der Schritt von der Fähigkeit ein Programm zu lesen, die fast alle erreichen zum selbständigen Schreiben von Programmen ein riesengroßer ist, der nur durch intensive individuelle Übung machbar wird.

In den Lösungen zeige ich einen systematischen Ansatz und versuche möglichst unsere Coding Guidelines zu erfüllen, die auf der Veranstaltungsseite stehen. Mit etwas mehr Programmierer- und Software-Engineering-Erfahrungen würden die Lösungen teilweise anders aussehen, aber das ist Thema Ihrer nächsten Semester.

Denken Sie daran, dass es immer Möglichkeiten gibt, über Ihre individuellen Lösungen zu diskutieren. Dies ist u. a. Teil des zweiten Praktikumstermins. Sie können aber auch zur Vorlesungszeit vorbeischauen oder mir eine E-Mail schreiben. Meine Betreuungszeit ist während der Vorlesungszeit aktuell noch nicht ausgelastet.

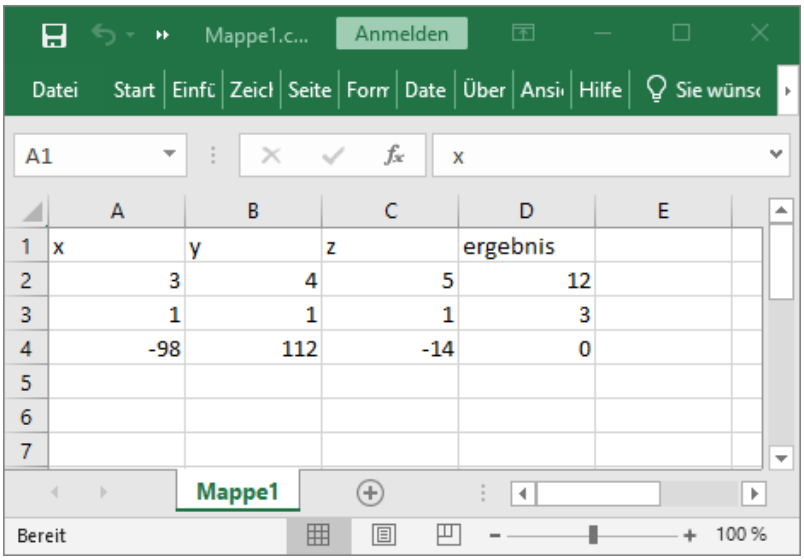
Weiterführend für Fortgeschrittenere: Die Erstellung von Tests ist eine zentrale Aufgabe während der Programmentwicklung. Tests können auch vor der Entwicklung geschrieben werden, dadurch besteht die Möglichkeit das zu programmierende Verhalten genau zu beschreiben. Wie immer bei Testfällen werden typische Fälle und jedwedes mögliche Randverhalten in Tests umgesetzt. Danach wird inkrementell die Software entwickelt und es sollten Schritt für Schritt mehr Tests grün werden. Da einige Tests laufen werden, auch wenn das zu entwickelnde Programm nichts macht, wird die Zahl der laufenden Tests nicht unbedingt kontinuierlich anwachsen. Das folgende Beispiel gibt einen kleinen unvollständigen Ausblick, was noch mit JUnit möglich ist. Beachten Sie, dass es unüblich ist Ausgaben in Tests zu programmieren, dies erfolgt hier zur Anschauung. Die zu testende Klasse hat eine Methode, die Zahlen addiert.

```java
public class Adder {
  public int sum(int x, int y, int z) {
    return x + y + z;
  }
}
```

Die Beispieltestklasse sieht wie folgt aus.

```java
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Assumptions;

import java.util.List;
import java.util.stream.Stream;

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.CsvFileSource;
import org.junit.jupiter.params.provider.MethodSource;

public class AdderTest {

  private Adder sut; // System under Test
  private Adder sut2; // kann beliebig viele Testobjekte geben

  @BeforeAll
  public static void setUpBeforeClass() {
    //Dies wird einmal am Start fuer alle Tests gemacht
  }

  @AfterAll
  public static void tearDownAfterClass() {
    //Dies wird einmal am Ende fuer alle Tests gemacht
  }

  @BeforeEach
  public void setUp() {
    //Dies wird einmal vor jedem Test gemacht
    this.sut = new Adder();
    this.sut2 = new Adder();
  }

  @AfterEach
  public void tearDown() {
    //Dies wird einmal mach jedem Test gemacht
  }

  @Test
  void testSumErwartet() {
    System.out.println("test1");
    int erg = this.sut.sum(21, 21, 0);
    Assertions.assertEquals(42, erg
```

```
        , "Erwartet wurde 42, gefunden: " + erg);
  }

  @Test
  void testSumVertauschbar() {
    System.out.println("test2");
    Assertions.assertEquals(this.sut.sum(1, 2, -1)
        , this.sut2.sum(-1, 1, 2));
  }

  @Test
  void testSumMitAnnahme() {
    System.out.println("test3");
    // Test nur ausfuehren, wenn Annahme erfuellt
    Assumptions.assumeTrue(1 + 2 + 3 == 6);
    int erg = this.sut.sum(1, 2, 3);
    Assertions.assertEquals(6, erg
        , "Erwartet wurde 42, gefunden: " + erg);
  }

  // Stellen Sie sich Stream einfach als List vor; der
  // ebenfalls mit einem Iterator alle Elemente durchlaufen kann.
  public static Stream<Arguments> daten(){
    List<Arguments> testdaten = List.of(
        Arguments.of(0, 0, 0, 0),
        Arguments.of(-2, -1, -3, -6),
        Arguments.of(6, 7, 29, 42)
    );
    return testdaten.stream();
  }


  @ParameterizedTest
  @MethodSource({"daten"})
  public void testSumParametrisiert(int a, int b, int c, int erg) {
    System.out.println("Testpar mit " + a + "," + b + "," + c);
    Assertions.assertEquals(erg, this.sut.sum(a, b, c));
  }

  @ParameterizedTest
  @CsvFileSource(resources ={"/Mappe1.csv"}, numLinesToSkip = 1)
  public void testSumMitExcelSheet(int a, int b, int c, int erg) {
    System.out.println("Testcsv mit " + a + "," + b + "," + c);
    Assertions.assertEquals(erg, this.sut.sum(a, b, c));
  }
}
```



Die resultierende Ausgabe sieht wie folgt aus.