Wrap-Up

Einleitung:

Sie haben es fast geschafft. Dies ist das letzte Praktikumsblatt für dieses Semester. Es wird nicht testiert. Sie haben aber die Möglichkeit, Termine in der letzten Vorlesungswoche während der Praktikumszeit bei den Betreuern zu „buchen“.

Das Aufgabenblatt dient zu Übungszwecken und umfasst sowohl Wiederholungsaufgaben als auch Aufgaben zu Themen, die nicht Teil bisheriger Aufgabenblätter waren. Ich empfehle Ihnen dennoch, die Aufgaben analog P1-P9 gewissenhaft zu bearbeiten.

Hinweis zur Klausurrelevanz: Alle Themen dieses Aufgabenblattes sind klausurrelevant.

Aufgabe 1 abstrakte Klasse

In dieser Aufgabe nutzen wir für das Einlesen von der Tastatur die Klasse IO. Diese enthält statische Methoden. Die Datei IO.java finden sie in ILIAS.

Es soll eine Prüfung durchgeführt werden. Die Prüfung besteht aus einem Quiz, das eine Menge an Prüflingen zu absolvieren haben. Ein Quiz besteht aus einer Menge an Fragen. Bei den Fragen handelt es sich um Fragen unterschiedlichen Typs (Wahr/Falsch, MultipleChoice, ...). Bei einer richtigen Antwort bekommt der Prüfling eine der Frage zugeordneten Punktzahl. Eine Prüfung läuft so ab, dass zunächst das Quiz vorbereitet wird, d.h. es wird eine Menge an Fragen eingegeben. Anschließend wird die Prüfung durchgeführt, d.h. die Prüflinge müssen der Reihe nach die Fragen beantworten. Danach wird eine Rangliste nach den erreichten Punkten erzeugt und die Ergebnisse der Prüfung werden ausgegeben.

Das folgende Programm realisiert eine solche Prüfung

```
abstract class Frage {

	String text; // Fragetext

	int punkte; // zu erreichende Punktzahl

	Frage(String text, int punkte) {
		this.text = text;
		this.punkte = punkte;
	}

	// Frage auf den Bildschirm ausgeben
	void frageStellen() {
		IO.println(this.text);
	}

	// Frage beantworten durch Prüfling, Antwort auswerten
	// und Punkte vergeben
	abstract void frageBeantworten(Pruefling person);

	int getPunkte() {
		return this.punkte;
	}
}
```

Wrap-Up

Oben und in einer früheren Übung wurde ein Java-Programm vorgestellt, mit dem eine Prüfung simuliert werden kann. An Fragetypen unterstützt dieses Programm Wahr/Falsch- und MultipleChoice-Fragen. Hierzu wurden entsprechende Klassen von einer abstrakten Klasse Frage abgeleitet und die daraus resultierende Polymorphie ausgenutzt.

a) Erweitern Sie das Programm so, dass als weiterer Fragetyp die **Mehrfachauswahl** unterstützt wird, d.h. zu einer Frage werden mehrere Antworten gegeben, von denen keine, eine oder auch mehrere korrekt sind. Leiten Sie eine entsprechende Klasse von der Klasse Frage ab und erweitern Sie die Methode frageErzeugen der Klasse Pruefung, so dass auch Fragen des Typs "Mehrfachauswahl" gestellt werden können.

b) Erweitern Sie das Programm so, dass als weiterer Fragetyp der **Lückenfrage** unterstützt wird, d.h. in einer Aussage fehlt ein Wort, das ein Prüfling zu ergänzen hat. Leiten Sie eine entsprechende Klasse von der Klasse Frage ab und erweitern Sie die Methode frageErzeugen der Klasse Pruefung, so dass auch Fragen des Typs "Lückenfrage" gestellt werden können.

c) Erweitern Sie das Programm so, dass als weiterer Fragetyp **Zahlenfolgenerweiterungen** unterstützt werden, d.h. es werden *n* Zahlen einer Zahlenfolge bekannt gegeben und der Prüfling muss daraus die zugrunde liegende Zahlenfolge identifizieren und die *n+1*-te Zahl der Zahlenfolge eingeben. Leiten Sie eine entsprechende Klasse von der Klasse Frage ab und erweitern Sie die Methode frageErzeugen der Klasse Pruefung, so dass auch Fragen des Typs " **Zahlenfolgenerweiterungen** " gestellt werden können.

Beispiel (Eingaben in grün):

```
Frage 1 (Zahlenfolgen ergaenzen)
Anzahl an vorgegebenen Zahlen: 4
Zahl 0: 1
Zahl 1: 2
Zahl 2: 3
Zahl 3: 4
Korrekte Folgezahl: 5
Erreichbare Punkte: 10

...

Identifizieren Sie die Zahlenfolge und geben Sie die nächste Zahl der Folge
ein!
1 2 3 4 5
Richtige Antwort: 10 Punkte
```

d) Erweitern Sie das Programm so, dass als weiterer Fragetyp die **Ordnungsfrage** unterstützt wird. Bei der Ordnungsfrage wird dem Prüfling eine Aufgabe und eine Menge mit n Antworten gegeben, die der Benutzer dann in die korrekte Reihenfolge bringen muss (bekannt durch die Spielerauswahlfrage bei „Wer-wird-Millionär?“). Leiten Sie eine entsprechende Klasse von der Klasse Frage ab und erweitern Sie die Methode frageErzeugen der Klasse Pruefung, so dass auch Fragen des Typs "Ordnungsfrage" gestellt werden können.

Beispiel (Eingaben in grün):

```
Frage 1: Ordnen Sie die folgenden deutschen Städte von Nord nach Süd!
Anzahl an Antworten: 4
Antwort 0: Oldenburg
Antwort 1: Kiel
Antwort 2: München
Antwort 3: Frankfurt
1. Index: 1
2. Index: 0
3. Index: 3
4. Index: 2
Erreichbare Punkte: 20

...
Ordnen Sie die folgenden deutschen Städte von Nord nach Süd!
(0): Oldenburg
(1): Kiel
(2): München
(3): Frankfurt
1. : 1
2. : 0
3. : 3
4. : 2
Richtige Reihenfolge: 20 Punkte
```

Aufgabe 2 (Exceptions und Testen)

Diese nachfolgende Aufgabe ist für die Entwicklungsumgebung Eclipse beschrieben. Vielleicht probieren Sie diese einmal aus. Alternativ kann die Realisierung auch in einer anderen Programmierumgebung erfolgen. Die Dokumentation für IntelliJ finden Sie unter https://www.jetbrains.com/help/idea/tests-in-ide.html.

Schreiben Sie eine kleine Klasse Taschenrechner mit den Methoden addiere(int, int), dividiere(int, int), subtrahiere (int,int) und multipliziere(int,int). Schreiben Sie eine weitere Klasse TaschenrechnerTest, die nur eine main-Methode enthält, eine Instanz erzeugt und einige Rechenoperationen ausführt.

a) Ergänzen Sie Ihre Lösung der Taschenrechneraufgabe um sinnvolle Ausnahmebehandlung für die Methoden dividiere(int, int) und modulo(int,int).

b) Das Testen der Methoden einer Klasse kann man sehr schön an ein vorhandenes Tool übergeben. Müssen wir ja nicht selber alles machen. Ein solches Tool ist JUnit.

   Verwenden Sie das Eclipse-Projekt für den Taschenrechner und erzeugen Sie im default-Package eine TestCase-Klasse. Führen Sie hierzu einen Rechtsklick auf Projektname im Package-Explorer durch und selektieren New -> JUnit Test Case (wenn nicht vorhanden: Other und unter Java -> JUnit -> JUnit Test Case auswählen).

   Legen Sie ein JUnit 4 TestCase an. Was ist überhaupt ein TestCase? Eine spezielle Klasse, die von JUnit zum Testen einer anderen Klasse verwendet wird. In dem TestCase werden

verschiedene Tests in Form von Methoden zusammengefasst werden (sehen wir gleich;-)).

Als TestCase-Namen verwenden wir „TaschenrechnerTest“ und als Class-under-Test die Klasse „Taschenrechner“. Das war es auch schon. Eclipse generiert den TestCase und wir können anfangen, die Methoden der Klasse Taschenrechner zu verwenden.

Da alle Tests ein Objekt der Klasse Taschenrechner nutzen, legen wir eine Objektvariable für die Klasse TaschenrechnerTest an: Taschenrechner rechner = new Taschenrechner();

Jetzt erstellen wir neue Tests. Aber wie gehen wir vor?

Bei einem Test wollen wir häufig den berechneten Wert gegen ein bekanntes Ergebnis

vergleichen. Wir treffen bspw. die Annahme, dass die Addition der Werte 5 und 3 als Ergebnis den Wert 8 liefert. Die Addition soll von der Methode addiere(int a, int b) der Klasse Taschenrechner durchgeführt werden. Als Annahme gilt, dass bei Übergabe der Werte 5 und 3 der Wert 8 geliefert wird.

Zur Information: Annahmen heißt im englischen Assertion (oder kurz assert).

Unser Test sieht jetzt für die Addition wie folgt aus:

```
@Test
 public void testAddiere() {
   int a=5;
   int b=3;
   assertEquals(8, rechner.addiere(a, b));
}
```

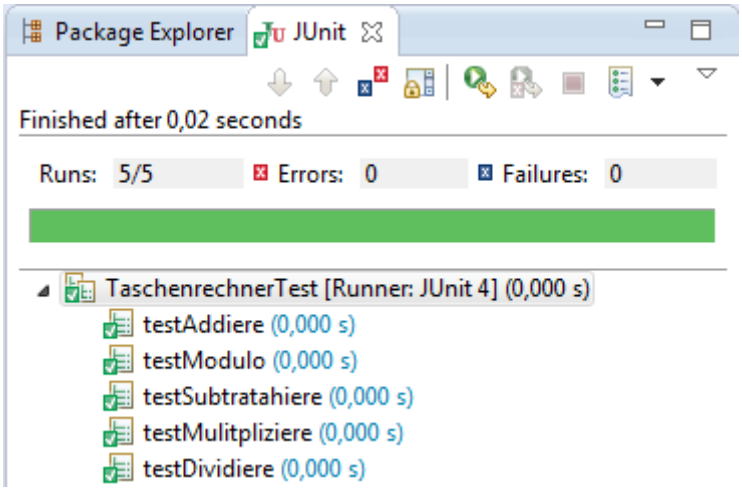
Interessant und wirklich einfach: wir testen die Methode addiere einfach, indem wir die Annahme aufschreiben: assertEquals(8, rechner.addiere(a, b)); Wir nehmen an, dass die Zahl 8 und der Rückgabewert der Methode addiere(5,3) gleich sind (=> assertEquals).

Wir lassen den Test jetzt über JUnit ausführen:

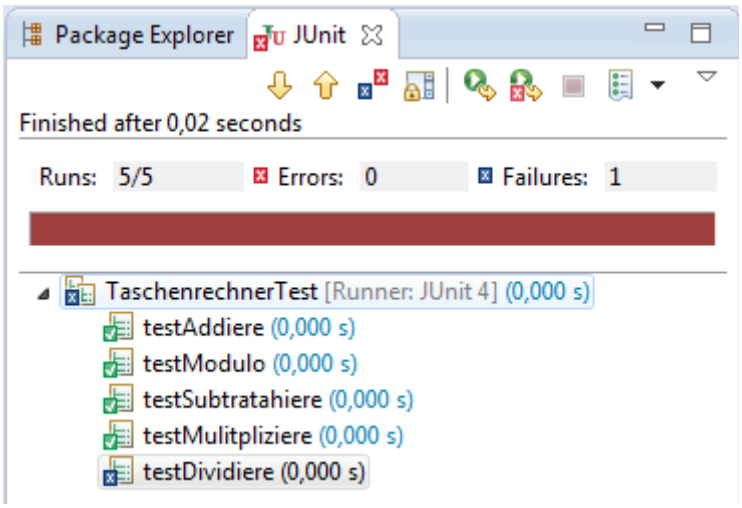
- Im Menü Run und hier den Menüpunkt Run as -> JUnit Test aufrufen
- Auf den Pfeil klicken und Run As -> JUnit Test verwenden
- Als Shortcut: Alt, Ctrl + Shift + X,T
- Rechtsklick auf das Projekt und RunAs -> JUnit Test auswählen

Wrap-Up

Nachdem JUnit für den Test gestartet wurde, werden alle Tests ausgeführt. Wenn Sie für jede Methode Tests entwickelt haben, sieht es im besten Fall so aus:



Sie waren erfolgreich!! Alle Tests sind fehlerfrei durchgelaufen. Ist ein Fehler aufgetreten, wird der fehlerhafte Test rot markiert. Doppelklicken Sie auf diesen Test und der Test wird dargestellt (kann ja sein, dass der Test selbst einen Fehler enthält; prüfen Sie dies immer zuerst).



Hier sieht man, dass der Test testDividiere() nicht durchläuft. Sehen wir uns den Test an (Doppelklicken auf testDividiere):

```
@Test
public void testDividiere() {
    assertEquals(1, rechner.dividiere(a, b));
}
```

Der Test ist korrekt. Markieren Sie die Methode dividiere und klicken Sie F3 (alternativ: Rechtsklick-> Open Declaration). Eclipse zeigt die Methode an. Tatsächlich hier ist der Fehler. Dieser wird behoben und der Test solange durchlaufen, bis alles auf Grün steht.

Aufgabe: Realisieren für jede Methode der Klasse Taschenrechner mindestens einen Test. Lassen Sie die Tests über JUnit durchlaufen und machen Sie einen Screenshot des Ergebnisses.

Aufgabe 3 verkettete Liste

Schreiben Sie Klassen, die Eisenbahnzüge repräsentieren. Ein Eisenbahnzug besteht aus einer Lokomotive und einer beliebigen Anzahl Wagen, möglicherweise auch überhaupt keinen. Lokomotiven und Wagen haben die folgenden Eigenschaften (alle ganzzahlig):

Lokomotive (locomotive)

- Länge (Meter)
- Typ (irgendeine Zahl)

Wagen (car)

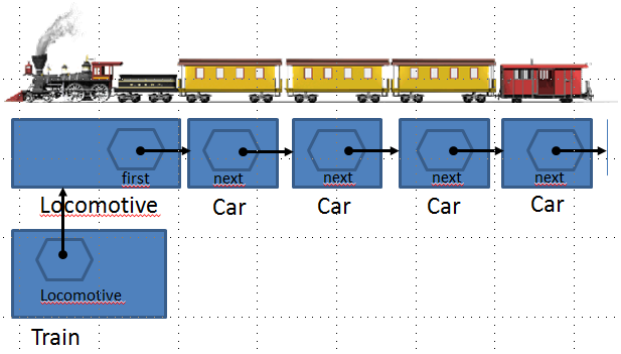
- Länge (Meter)
- Passagierkapazität (Anzahl Personen)

Definieren Sie die Klassen Locomotive und Car, jeweils mit sinnvollen Methoden. Die oben genannten Eigenschaften sind unveränderlich. Das interessante Problem ist das Zusammenstellen eines Zuges aus den Einzelteilen. Der erste Wagen hängt direkt an der Lokomotive. Geben Sie der Klasse Locomotive deshalb ein Element first vom Typ Car, dazu eine Getter und eine Setter-Methode.

An jedem Wagen hängt der jeweils nächste Wagen oder gar nichts beim letzten Wagen. Definieren Sie in der Klasse Car ein Element next des gleichen Typs Car, wieder mit Gettern und Settern. Diese Objektvariable speichert ein anderes Objekt derselben Klasse oder null beim letzten

Wagen.

Die folgende Abbildung zeigt die Idee.



Definieren Sie schließlich eine Klasse Train, die den ganzen Zug repräsentiert. Ein Train-Objekt speichert "seine" Lokomotive, aber nicht die Wagen. Diese können, einer nach dem anderen, auf dem Weg über die Lokomotive erreicht werden. Die Klasse Train bietet die folgenden Methoden:

- Konstruktor der Train-Konstruktor erwartet eine Lokomotive und baut einen ziemlich kurzen Zug, der nur aus der Lokomotive, noch ohne Wagen besteht.
- add hängt in diesen Zug einen gegebenen Wagen ein. Es spielt keine Rolle, wo im Zug der Wagen eingefügt wird.
- print gibt eine Liste dieses Zuges mit allen Bestandteilen aus.
- getPassengers liefert die gesamte Passagierkapazität dieses Zuges, das heißt die Summe der Passagierkapazitäten aller Wagen.
- getLength liefert die Gesamtlänge dieses Zuges, d. h. die Summe der Längen der Lokomotive und aller Wagen.
- removeFirst hängt den ersten Wagen aus diesem Zug aus und liefert den ausgehängten Wagen als Ergebnis zurück. Die restlichen Wagen rücken nach vorne. Falls es keinen Wagen gibt, ist das Ergebnis null.
- relink akzeptiert als Parameter einen anderen Zug und hängt alle Wagen des anderen Zuges in der gleichen Reihenfolge an diesen Zug an. Im anderen Zug bleibt nur die Lokomotive zurück. Nutzen Sie für diese Methode geschickt die vorher definierten Methoden.
- revert dreht die Abfolge der Wagen in diesem Zug um, das heißt, der vorher letzte Wagen wird zum ersten und umgekehrt.

Schreiben Sie schließlich eine Anwendung, die Folgendes abwickelt:

1. Eine Lokomotive "Big Chief" mit der Nummer 5311 und der Länge 23m wird erzeugt.
2. Ein Zug namens "Santa Fe"mit der Lokomotive "Big Chief" wird erzeugt.
3. An "Santa Fe" werden drei Wagen mit den Längen 12m, 15m, 20m und den Passagierkapazitäten 50, 75, 100 Personen angehängt.
4. Eine Lokomotive "Steel Horse" mit der Nummer 5409 und der Länge 21m wird erzeugt.
5. Ein Zug namens "Rio Grande Express" mit der Lokomotive "Steel Horse" wird erzeugt.
6. An den "Rio Grande Express" werden zwei Wagen mit den Längen 13m und 18m sowie den Passagierkapazitäten 60 und 80 Personen angehängt.
7. Alle Wagen von "Santa Fe" werden in den "Rio Grande Express" übernommen.
8. Die Wagenreihenfolge im "Rio Grande Express" wird umgedreht.

Aufgabe 4 Vermischtes

Ein Freund von mir (Dr. Dietrich Boles, Lehrender an der Uni Oldenburg) hat eine umfangreiche Aufgabensammlung zum Erlernen der Programmiersprache Java zusammengestellt.

https://www.boles.de/programmierkurs-java/

Überlegen Sie, wo Sie Schwächen haben. Wählen Sie nun je 2 Aufgaben aus jeder Kategorie, in der Ihre Schwächen liegen und lösen Sie diese OHNE KI und idealerweise zunächst auf Papier. UE23 (Generics) ist für uns nicht relevant.

Wir wünschen Ihnen eine schöne Weihnachtszeit.