Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik III
Prof. Dr. A.B. Cremers

Aufgabenblatt 9

— Veröffentlicht am 30.06.2010 —

Abgabe bis 06.07.2010, 13:00 per SVN

Aufgabe 1 *(Arrays, Listen, Stapel - 2 Punkte)*

Überlegen Sie sich für jede Aussage ein Argument, mit dem Sie eindeutig bestimmen können, ob die Aussage wahr oder falsch ist. Für jede richtig angekreuzte Antwort werden 0,5 Punkte angerechnet, für jede falsch angekreuzte Antwort 0,5 Punkte abgezogen. Nicht angekreuzte Antworten geben keinen Punktabzug.

wahr	falsch	
		Folgendes ist eine zulässige Array-Initialisierung: `String[] weekDay = {"So","Mo","Di","Mi","Do","Fr","Sa"};`
		Das Attribut `length`, über das auf die Größe eines Arrays zugegriffen werden kann, ist eine `public final int`-Variable, deren Wert entweder positiv oder Null ist.
		Der Zugriff auf ein spezielles Element erfolgt in einer Liste schneller als in einem Array.
		Stapel arbeiten nach dem FIFO-Prinzip.

Aufgabe 2 *(Manuelles Testen - 9 Punkte)*

Bei der Software-Entwicklung sind häufige Tests unerlässlich. Daher versucht man, alle entwickelten Klassen häufig und möglichst automatisch zu testen.

a) Entwickeln Sie eine Klasse `Calculator`. Diese soll für Ganzzahlwerte die öffentlichen Klassenmethoden `add` (Addition), `sub` (Subtraktion), `mul` (Multiplikation), und `div` (Division) unterstützen. Implementieren Sie die Methoden, bauen Sie dabei in die Funktion `div` einen logischen Fehler ein.

b) Schreiben Sie nun eine Klasse `CalculatorTest`, mit welcher Sie die eben entwickelte Klasse testen (eine so genannte *Testfall-Klasse*). Verwenden Sie dazu folgende Vorlage:

```
public class CalculatorTest {

  private void assertEquals(int value, int expected) {
    if (value != expected)
      throw new AssertionError("Value is " + value + ", expected was "
          + expected);
  }

  public void testAdd() { // Testet die Calculator.add()-Methode
    assertEquals(Calculator.add(5, 7), 12); // erstes Testbeispiel
    ... // Zweites Testbeispiel
  }

  public void testSub() {
    ...
  }

  ...
}
```

Ergänzen Sie die Vorlage so, dass in vier *Testmethoden* die entsprechenden Methoden der `Calculator`-Klasse an jeweils mindestens drei Beispielen getestet werden. Verwenden Sie dabei zum Vergleichen von Ist- und Sollzustand die Methode `assertEquals`. Decken Sie besonders auch Spezialfälle ab.

c) Schreiben Sie nun ein Programm, welches vier Instanzen der *Testfall-Klasse* `CalculatorTest` erstellt und darauf jeweils eine der *Testmethoden* aufruft. Geben Sie zu jeder *Testmethode* an, ob deren Ausführung erfolgreich war. Fangen Sie dabei auftretende `AssertionError`s ab und geben Sie die Fehlermeldung aus.

Aufgabe 3 *(Reflection - 8 Punkte)*

Manuelles Testen erweist sich als umständlich und fehleranfällig, da für jeden neu entwickelten *Testfall* das Test-Programm modifiziert werden muss. Mit Hilfe von *Reflection* lässt sich das Ausführen von Testfällen automatisieren.

a) Entwickeln Sie ein kleines Programm, das automatisch vordefinierte Tests in Java-Klassen ausführt. Der Name der Klasse, indem sich die *Testmethoden* befinden, wird dem Programm beim Aufruf als Parameter mitgegeben. Die *Testmethoden* sind dadurch identifizierbar, dass sie *öffentlich* sind, *keine Parameter* erwarten, und *ihr Name mit `"test"` beginnt*. Nutzen Sie zur Programmierung die *Java Reflection API*. Hinweise dazu finden Sie auf diesen Webseiten:

`http://wiklet.javacore.de/index.php/Reflection_-_Die_Java_Reflection_API`
`http://java.sun.com/developer/technicalArticles/ALT/Reflection/index.html`

Das Programm soll folgende Schritte durchführen:

- Ein `Class`-Objekt mit dem Namen des ersten Programmarguments instantiieren.
- Jede Methode dieser Klasse daraufhin untersuchen, ob sie eine *Testmethode* ist.
- Jedes Mal, wenn eine Methode als *Testmethode* identifiziert wurde, soll ein Objekt der Klasse erstellt werden und die Methode auf diesem Objekt ausgeführt werden.

Gehen Sie bei der Entwicklung von dieser Vorlage aus:

```
import java.lang.reflect.*;

public class AutomaticTester {
  public static void main(String args[]) {
    try {
      Class classToTest = ...
      System.out.println("Testing " + classToTest);

      ...

      }
    } catch (Throwable e) {
      System.err.println("An error occured: " + e);
    }
  }

  private static void invokeTestMethod(Class classToTest, Method method) {
    System.out.print("Exectuting " + method.getName() + ": ");
    try {

      ...

      System.out.println("ok");
    } catch (Throwable e) {
      System.out.print("failed: ");
      if (e.getCause() instanceof AssertionError) {
        System.out.println(e.getCause().getMessage());
      } else {
        e.printStackTrace();
      }
    }
  }
}
```

b) Testen Sie das Programm aus Teilaufgabe a) mit der *Testfall-Klasse* `CalculatorTest` aus der vorangegangenen Aufgabe.

Aufgabe 4 *(Speicherbilder und Listen - 8 Punkte)*

In der Klasse `IntList`, die einfach verkettete Listen von Ganzzahlen (Datentyp `Integer`) repräsentiert und in der die Methoden der Klasse `TList` der Vorlesung entsprechend implementiert sind, sei die `main`-Methode wie folgt implementiert:

```
static public void main (String[] args) {
  IntList s1, s2;
  s1 = new IntList();
  s1.insertFirst(7);
  s1.insertFirst(8);
  s2 = s1.reverseListCon();
  // Stelle 1

  s1.reverseList();
  // Stelle 2

  s1.insertFirst(2);
  s2.insertFirst(3);
  // Stelle 3
  }
}
```

Skizzieren Sie den Zustand des Speichers an den Stellen 1, 2 und 3!