Fakultät IV - Elektrotechnik und Informatik
Algorithmen und Datenstrukturen SoSe 2016
Brock / Deimel / Foucard / Mabrouk / Sieverling

Aufgabenblatt 1

letzte Aktualisierung: 29. April, 18:07 Uhr (revUnversioned directory)

Ausgabe: 01.05.2016 19:59
Abgabe: 11.05.2016 19:59

Thema: *Klassen, Objekte, Methoden, Iteratoren, Listen, Unit testing*

Wichtige Ankündigungen

- Bestimmte Unit tests in dieser Aufgabe sind Teil der Abgabe. Diese dürfen nicht geteilt oder weitergegeben werden!! Unit tests die nicht Teil der Abgabedateien sind, dürfen geteilt werden.
- Aufgrund des Feiertages fallen die Donnerstags-Tutorien aus. Ersatz-Tutorien findest du in ISIS!
- Das Abgabe-Repository steht unter `https://teaching.inet.tu-berlin.de/tubitauth/svn/algodat-sose2016` mittels eurem tubit-account zur Verfügung,
- Erinnerung: Registriere dich (für die meisten: in QISPOS) falls du das noch nicht gemacht hast!
- Dieses Jahr gibt es keine Gruppenabgaben in den Übungen.

Abgabe

Die folgenden Dateien müssen für eine erfolgreiche Abgabe im svn Ordner `Tutorien/txx/Studierende/deinname@TU-BERLIN.DE/Abgaben/` eingecheckt sein:

Geforderte Dateien:	
`Blatt01/src/Sphere.java`	Aufgabe 2.6
`Blatt01/src/ArrayCheck.java`	Aufgabe 3.3
`Blatt01/tests/ArrayCheckTest.java`	Aufgabe 3.4

Als Abgabe wird die nur neueste Version im svn gewertet.

1. Aufgabe: Einführung

1.1. (Tut) Arbeiten mit Eclipse Importiert das Hausaufgaben-Projekt in Eclipse. Geht dazu wie folgt vor:

1. Entpackt das Vorgabenarchiv und öffnet Eclipse.
2. Klickt mit der rechten Maustaste in den Package Explorer und klickt auf Import...
3. Wählt unter General Existing Projects into Workspace.
4. Wählt unter Select root directory den Ordner aus, in welchen ihr das Vorgabenarchiv entpackt habt.
5. Klickt auf Finish.

In dem Eclipse-Projekt findet ihr nun eine Programmskelett als Vorgabe, in welche ihr eure Lösung der Hausaufgabe schreibt. Seht euch die Dateien im Ordner `src/` an. Öffnet die Datei `Point.java`, welche in der nächsten Teilaufgabe eine Rolle spielen wird.

1.2. (Tut) Java erklären Was bedeuten die folgenden Begriffe oder Anweisungen?

1. `.java Datei vs. .class Datei`
2. `Klasse vs. Objekt/Instanz`
3. `public static void main(String[] arguments)`
4. `Point p = new Point(1, 2, 3);`
5. `p.y = p.y + 1;`
6. `p.y + 1;`

2. Aufgabe: Objektorientierung, Java-Namenskonventionen

2.1. (Tut) Namenskonventionen Warum gibt es Namenskonventionen? Welche Namenskonventionen in Java kennt ihr? Diskutiert Vor- und Nachteile von Namenskonventionen.

Hinweis: Eine Liste der wichtigsten Namenskonventionen wird auf ISIS verfügbar sein.

Hinweis: Mehr über die *Java Code Conventions* findet ihr außerdem unter `http://www.oracle.com/technetwork/java/codeconv-138413.html`.

2.2. (Tut) Klasse Quadrat programmieren Implementiert eine Klasse, welche Quadrate repräsentiert. Ein Quadrat ist durch seine Seitenlänge eindeutigt bestimmt.

Beachtet besonders die Java-Namenskonventionen!

2.3. (Tut) Eigenschaften und Konstruktor Schreibt einen Konstruktor für die Quadrat-Klasse. Dem Konstruktor soll als Argument die Seitenlänge des Quadrats übergeben werden.

2.4. (Tut) Methoden Schreibt eine Methode `scaleBy`, mit welcher man das Quadrat vergrößern oder verkleinern kann. Dazu wird der gesuchten Methode ein Faktor s als Parameter übergeben, welcher mit der Seitenlänge multipliziert wird.

2.5. (Tut) Objekte der Quadrat-Klasse instanziieren Erstellt eine Klasse `SquareObjects`; in der ihr zwei Objekte der Quadrat-Klasse instanziiert und eines der Quadrate um den Faktor 2 vergrößert.

2.6. Klasse `Sphere` implementieren (30 Punkte) Im Eclipse-Projekt zur Hausaufgabe findet ihr die Klasse `Sphere`, welche verschiedene leere Methoden enthält. Eure Aufgabe ist es, die folgenden Methoden zu implementieren:

1. `Sphere(int x, int y, int z, double r)` ist ein Konstruktor, dem x-, y-, z-Koordinate des Mittelpunkts sowie der Radius r der Kugel übergeben werden.

2. `Sphere(Point c, double r)` ist ein zweiter Konstruktor, dem der Mittelpunkt als Point-Objekt übergeben wird, sowie der Ŕadius r.
3. **`double`** `getX()` gibt die x-Koordinate der Kugel zurück.
4. **`double`** `getY()` gibt die y-Koordinate der Kugel zurück.
5. **`double`** `getZ()` gibt die z-Koordinate der Kugel zurück.
6. **`double`** `getRadius()` gibt den Radius der Kugel zurück.
7. **`double`** `calculateDiameter()` berechnet und gibt den Durchmesser der Kugel zurück.
8. **`double`** `calculatePerimeter()` berechnet und gibt den Umfang der Kugel zurück.
9. **`double`** `calculateVolume()` berechnet und gibt das Volumen der Kugel zurück.

Hinweis:
- Die Methoden `Math.pow(double a, double b)` (Exponent) und `Math.PI` (die Zahl π) sind eure Freunde.
- Java unterscheidet zwischen Integer und Fließkomma-Division (Überprüft beispielsweise das Ergebnis von $1/2$ und $1.0/2.0$.). Überzeugt Euch vom Unterschied und verwendet den richtigen Datentyp.
- Verwendet für die Bearbeitung dieser Aufgabe die Klasse `Point.java` aus den Vorgaben.
- Die Implementierung darf Null Pointer in Methodenparametern als Ausnahmefehler behandeln, d.h. es ist keine explizite Fehlerbehandlung notwendig.

2.7. (Tut) Testen mit jUnit Öffnet die Datei `tests/SphereTest.java`. Die Datei enthält einfache Tests, die euer Programm auf jeden Fall bestehen muss. Die Tutoren haben eine Datei mit vielen verschiedenen Tests, und je mehr Tests euer Programm besteht, desto wahrscheinlicher bekommt ihr die Punkte für die Abgabe.

Führt die Tests mit Eclipse aus und betrachtet die Anzeige.

Hinweis: Ihr könnt auch selbst weitere Tests in der Klasse `SphereTest.java` hinzufügen, um Euch die Entwicklung der Sphere-Klasse zu erleichtern.
Es gibt verschiedene `assert`-Anweisungen in jUnit, wovon die einfachste `assertTrue` ist, es gibt z.B. noch `assertFalse`, `assertEquals`, `assertNull` u.v.m. Ihr findet weitere Informationen zu jUnit unter http://junit.sourceforge.net/.

Hinweis: Implementierung und Testerstellung läuft bei professioneller Softwareentwicklung in der Regel schrittweise und gleichzeitig ab, d.h. die Erweiterung der Funktionalität der Software sowie ihre Abdeckung durch Tests ist ein kontinuierlicher Prozess. Falls ihr an dem Thema Testen von Software interessiert seid, sucht bei Google mal nach „Test-driven development" (TDD).

3. Aufgabe: Arrays

Als Arrays (dt. „Felder"), werden zusammenhängende Speicherbereiche gleichen Typs bezeichnet, auf einzelne Elemente kann direkt über Indices zugegriffen werden.
Hinweis: Während z.B. in C Arrays prinzipiell eine fix Größe haben, besitzen andere Programmiersprachen wie Java Implementierungen, deren Größe sich zur Laufzeit automatisch anpasst.

3.1. (Tut) Einführung Initialisiert ein ArrayList von Zahlen und weist ihm Werte zu.

3.2. (Tut) Ablaufen eines Arrays Besprecht die Methode `calcExamAverage`, die ein Array von Klausurnoten bekommt, die Durchschnittsnote errechnet und diese zurück gibt.

3.3. Reihen als Arrays (30 Punkte) In dieser Aufgabe sollen Arrays benutzt werden, um Reihen darzustellen und auf diesen bestimmte Tests durchzuführen. Es ist die Klasse `ArrayCheck` zu implementieren, welche die Testmethoden auf den Arrays enthalten soll.

Genauer soll `ArrayCheck` dabei folgende 3 Methoden enthalten (nutzt hier die Vorgabe von `ArrayCheck`):

1. `boolean allDivisibleBy(ArrayList<Integer> arr, int divisor)`, die überprüft, ob alle Zahlen im übergebenen Array (Reihe) `arr` ganzzahlig durch die übergebene Zahl `divisor` teilbar sind. Hinweise:
   - Ist das übergebene Array nicht existent oder leer, soll die Methode `false` zurückgeben.
   - 0 ist durch alle Zahlen teilbar.
   - Keine Zahl ist durch 0 teilbar.
2. `boolean isFibonacci(ArrayList<Integer> arr)`, die überprüft, ob es sich bei dem übergebenen Zahlen-Array um einen Ausschnitt aus der Fibonacci-Reihe handelt. Das Array muss mindestens drei Elemente enthalten.

   Die Fibonaccireihe ist wie folgt definiert:

$$\begin{aligned} f_0 &= 0 \\ f_1 &= 1 \\ f_n &= f_{n-1} + f_{n-2} \quad \text{für} \quad n \geq 2 \end{aligned}$$

   Beispiel:

1	1	2	3	5	8

   Hinweis:
   - Für ein nicht-existentes Array soll die Methode `false` zurückgeben.
   - Für ein array mit weniger als 3 Elementen soll die Methode `false` zurückgeben.
   - Die Methode ignoriert die Generalisierung der Fibonacci-Reihe für negative Zahlen und tested nur die klassische Definition.

3.4. JUnit-Test für ArrayCheck (40 Punkte) Erweitert die JUnit-Testklasse `ArrayCheckTest`, die Unit-tests für die Klasse `ArrayCheck` enthält. Schreibt für jede Methode von `ArrayCheck` genau einen Unit-test. Achtet darauf, dass eure Tests weder fehlerhafte Implementierungen von `ArrayCheck` akzeptieren, noch korrekte Implementierungen verwerfen.

Haltet euch bei der Implementierung der Unit-tests an die zu testendende Verhaltensbeschreibung in der Vorgabe!

Die Vorgabe findet ihr in `Aufgabe/Blatt01/Vorgaben/tests/`. Euch steht natürlich frei, neben den für die Abgabe geforderten Unit-tests für auch weitere zu schreiben!