Technische Universität Braunschweig
Institut für Anwendungssicherheit

Prof. Dr. Martin Johns
Marius Musch
Maximilian von Unwerth

# Programmieren I

*05. Übungsblatt*

**Aufgabe 1** Schreiben Sie ein Programm, das quadratische Gleichungen der Form

$$ax^2 + bx + c = 0$$

mit $a \neq 0$ löst. Zuerst soll Ihr Programm die Koeffizienten $a$, $b$ und $c$ einlesen und anschließend die Lösungen gemäß der Formel

$$x_{1,2} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}$$

mit $p = \frac{b}{a}$ und $q = \frac{c}{a}$ berechnen und ausgeben. Wenn keine reelle Lösung existiert, soll eine entsprechende Meldung erfolgen. Die Wurzel einer Zahl können Sie mit der Methode `Math.sqrt` berechnen.

**Aufgabe 2** In der Informatik werden häufig Zufallszahlen benötigt, z. B. um Simulationen durchzuführen. Einer der ersten Algorithmen zur Erzeugung von *Pseudozufallszahlen* wurde von John von Neumann in der 1940er Jahren entwickelt.
Dieser Algorithmus wird *Mid-Square-Methode* genannt. Er beginnt mit einer Zahl, die als erste Zufallszahl genommen wird. Die Zahl wird quadriert. Anschließend werden die mittleren Ziffern der Quadratzahl als nächste Zahl verwendet. Falls die Quadratzahl nicht doppelt so lang ist, müssen vorne führende Nullen eingefügt werden. Als Beispiel betrachten wir die ersten Schritte des Algorithmus mit der Anfangszahl 4637:

```
1  4637 --> 21501769 = 4637*4637
2  5017 --> 25170289 = 5017*5017
3  1702 --> 02896804 = 1702*1702
4  8968 --> 80425024 = 8968*8968
5  4250 --> 18062500 = 4250*4250
6   625 --> 00390625 =  625* 625
7  3906 --> ...
```

Meistens wiederholen sich nach relativ wenigen Schritten die Zufallszahlen. Aus diesem Grund wird der Algorithmus heute kaum noch verwendet.
Mit der Anfangszahl 4637 erzeugt die Mid-Square-Methode die folgenden Pseudozufallszahlen

$$4637, 5017, 1702, 8968, 4250, 625, 3906, \ldots, 6100, 2100, 4100, 8100, 6100, \ldots$$

Dann wiederholen sich die Zahlen 6100, 2100, 4100, 8100. Inklusive der Anfangszahl (4637) besteht die Folge aus 47 verschiedenen Zufallszahlen.
Schreiben Sie ein Java-Programm, das den Algorithmus auf alle vierstelligen Zahlen anwendet. Das heißt auf alle Anfangszahlen aus dem Bereich $1000, \ldots, 9999$. Ihr Programm soll ausgeben, wie viele verschiedene Zufallszahlen maximal von einer Anfangszahl erzeugt werden können. Außerdem soll Ihr Programm diese Folge ausgeben.
Ob Sie die Anfangszahl zu den Zufallszahlen mitzählen oder nicht, bleibt Ihnen überlassen. Falls es mehrere Anfangszahlen geben sollte, die die maximale Anzahl der Zufallszahlen erzeugen, dürfen Sie eine auswählen.

**Aufgabe 3** In Java gibt es sogenannte *checked Exceptions*, welche von `Exception` erben, sowie *unchecked Exceptions*, welch von `RuntimeException` erben.
Machen Sie sich mit den Unterschieden vertraut.
Ausführliche Dokumentation zu Exceptions in Java finden Sie beispielsweise unter: `https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html`.

**Aufgabe 4** Das Programm in Abbildung 1 bricht während der Ausführung mit einem Fehler ab.
Erklären Sie die Fehlerursache und überlegen Sie sich, wie man diese beheben könnte.

```
1  class DownTo {
2    static void downToZero(int n) {
3      if(n == 0) {
4        return;
5      }
6      downToZero(n-1);
7    }
8
9    public static void main(String[] args) {
10     int[] bounds = { 10, 100, 1000, 10000, 100000, 1000000 };
11     for(int bound : bounds) {
12       System.out.println("Testing for: " + bound);
13       downToZero(bound);
14     }
15   }
16 }
```

Abbildung 1: Erklären Sie den zur Laufzeit auftretenden Fehler.

**Pflichtaufgaben** Auf diesem Aufgabenblatt finden sich die drei letzten Pflichtaufgaben. Auch wenn Sie bereits einen Großteil der zum Bestehen der Hausaufgabenleistung benötigten Punkte erreicht haben, empfehlen wir dringend die Bearbeitung der folgenden Aufgaben. In der Hausarbeit, die in diesem Jahr als Klausurersatzleistung geschrieben wird, werden Dinge wie JUnit und Gradle gebraucht, die mit diesen Aufgaben geübt werden sollen.

**Pflichtaufgabe 12: Vom Fangen und Werfen (4 Punkte)** Sicherlich ist bei Ihnen schon einige Male eine *Exception* beim Ausführen Ihres Codes aufgetreten. Diese sogenannten Ausnahmefehler beenden ein Java-Programm, welches ein unerwartetes Verhalten ausweist.Man kann sich solchejedoch auch zu Nutze machen und *behandeln*.

Erstellen Sie eine neue Klasse, die eine natürliche Zahl als Kommandozeilenparameter einliest und in einer Integer-Variable speichert. Der Wert dieser Variable soll daraufhin einfach in der Kommandozeile ausgegeben werden. Vergisst der Nutzer des Programms ein Kommandozeilenargument zu übergeben, soll nun die entstehende Exception, eine `ArrayIndexOutOfBoundsException` mit Hilfe eines `try-catch`-Blockes abgefangen und eine geeignete vereinfachte Fehlermeldung ausgegeben werden. (1 Punkt). Solche `try-catch`-Blöcke eignen sich auch dazu Variablen einen Wert zu geben, egal, ob die Eingabe gültig war oder nicht. Geben Sie der Variable in den catch-Blöcken einen solchen Default-Wert, damit die Variable auf jeden Fall initialisiert wird oder beenden Sie an der gleichen Stelle das Programm. (1 Punkt) Fangen Sie zusätzlich die Exception, die auftritt, wenn ein Nutzer einen String eingibt, der nicht zu einem Integer-Wert gecastet werden kann. (1 Punkt). Nun soll ihr Programm, wenn ein Nutzer eine Zahl, die größer als 10 ist eine eigene Exception werfen. Recherchieren Sie, wie Sie eine Exception werfen können und wählen Sie eine geeignete Java Exception, die Sie Ihr Programm bei Eingabe einer solch großen Zahl werfen lassen. (1 Punkt)

**Pflichtaufgabe 13: Testen mit JUnit (2 Punkte)** Zum systematischen Testen der Funktionalität Ihrer Programme wollen wir in dieser Aufgabe das Testing-Framework **JUnit** kennenlernen. Jedes Programm besteht aus einer ganzen Menge von Unterprogrammen, Methoden, Funktionen, ..., die alle eine bestimmte Funktionalität liefern, beziehungsweise liefern sollen. Um diesen Begriff zu verdeutlichen stellen wir uns Funktionalität so vor, dass ein bestimmter Input stets zu einem erwarteten entsprechenden Output führt. Ein sogenannter JUnit-Test überprüft stets eine Kombination von Input und Output, also ob bei einer bestimmten Eingabe der der Spezifikation entsprechende Ausgabewert erzeugt wird. Um so die vollständige Funktionalität eines Programmes zu überprüfen, sind somit mehrere oder besser viele Unit-Tests notwendig. Insbesondere Randfälle sollten dabei besonders überprüft werden. Hat man später etwas an seinem Programm verändert und will prüfen, ob dieses immernoch funktioniert oder besser “der Spezifikation entspricht”, testet man erneut mit allen Unit-Tests, um festzustellen, ob sein Programm für alle Eingaben immernoch die erwarteten Ausgaben erzeugt. Durch solch ausführliches Testen lassen sich viele Fehler identifizieren, bevor sie in der Praxis

auftreten. In vielen Bereichen, wie dem Flugzeugbau oder der Raumfahrt sind solche Tests absolut essentiell.

Erweitern Sie Ihre Klasse aus Pflichtaufgabe 12 um eine Methode `twentyBy(int value)`, die stets die Zahl 20 durch den Kommandozeilenparameter teilt und das Ergebnis zurückgibt. Diese wollen wir nun auf ihr Verhalten testen. Erstellen Sie eine Test-Klasse, die den Namen Ihrer Klasse aus Pflichtaufgabe mit dem Suffix *Test* bekommt. (1 Punkt) Erstellen Sie mit Hilfe der Annotation `@Test` einen neuen Test und testen Sie mittels mehrerer `assertEquals()`, ob Ihre Methode in jedem Fall das richtige Ergebnis zurückgibt. (1 Punkt) Wählen Sie für Ihre Tests geeignete Testwerte.

Laden Sie sich zum Ausführen Ihrer Tests die Datei `junit-platform-console-standalone-1.6.2.jar` herunter und legen Sie sie dort ab, wo sich auch Ihre Java-Datei aus Aufgabe 12 befindet und Ihre Tests befinden. Wechseln Sie im Terminal in dieses Verzeichnis und kompilieren Sie Ihre Java Dateien mit:

```
javac -cp junit-platform-console-standalone-1.6.2.jar *.java
```

Führen Sie dann Ihre Tests mit folgendem Befehl aus:

```
java -jar junit-platform-console-standalone-1.6.2.jar --class-path . --scan-class-path
```

Bitte achten Sie darauf, die *.jar*-Datei nicht mit hochzuladen.

**Pflichtaufgabe 14: Gradle (4 Punkte)** Gradle ist ein auf Java basiertes Build-Management und Automatisierungstool. Hiermit wird das "Zusammenbauen" eines Projektes, welches teilweise aus sehr vielen Abhängigkeiten oder externen Bibliotheken bestehen kann vereinfacht. Beispielsweise nutzt auch Android Gradle für das Bauen von Applikationen. Mit Gradle können auch die Tests aus der vorherigen Aufgabe automatisiert ausgeführt werden.
Es existieren drei Dateien in einem Gradle-Projekt, wovon für uns nur die *build.gradle* interessant sein soll.

- **build.gradle**: Definition des Builds mit allen Abhängigkeiten
- **settings.gradle**: Wenn mehrere Teilprojekte ein großes bilden sollen
- **gradle.properties**: Für projektspezifische Initialisierung notwendig

Der Gradle-Wrapper kann einfach aufgerufen werden. Es ist keine Installation nötig. Die notwendigen Dateien finden Sie beim Übungsblatt im Studip. In Linux führen Sie über die Kommandozeile das Programm `gradlew` aus. In Windows nutzen Sie die Datei `gradlew.bat`.

Erstellen Sie nun für Pflichtaufgabe 12 und 13 ein neues Gradle-Projekt und legen Sie

den Code im *src* Ordner in den entsprechenden Unterordnern *main* oder *test* ab. (1 Punkt) Erstellen Sie eine geeignete `build.gradle` um Ihren Java-Code zu kompilieren und bauen Sie Ihr Projekt mit dem entsprechenden Befehl zu einer *.jar*-Datei. (1 Punkt) Zum Testen unseres Codes wollen wir wieder JUnit nutzen. Fügen Sie JUnit als Abhängigkeit hinzu (1 Punkt) und führen Sie Ihre Tests aus. Welchen Befehl nutzen Sie, um alle Ihre Tests auszuführen? Schreiben Sie den notwendigen Befehl als Kommentar in die letzte Zeile der Klasse von Pflichtaufgabe 13. (1 Punkt). Machen Sie sich mit der Ordnerstruktur vertraut. In der Hausarbeit können Sie eine sehr ähnliche Struktur finden. Es wird Tests geben, die es Ihnen ermöglichen zu überprüfen, ob Ihre Programme funktionieren und `.java`-Dateien auf denen Sie Ihr Projekt aufbauen können. Viel Erfolg.

Mehr zu Gradle finden Sie auf `https://docs.gradle.org/current/samples/sample_building_java_applications.html` oder auf `https://spring.io/guides/gs/gradle/`.