Technische Universität Braunschweig
Institut für Anwendungssicherheit

Prof. Dr. Martin Johns
Marius Musch
Maximilian von Unwerth

Programmieren I

03. Übungsblatt

Melden Sie sich bitte für die Veranstaltung bei Stud.IP (http://studip.tu-bs.de) an und tragen sich in einer der kleinen Übungen ein. (`https://studip.tu-braunschweig.de/dispatch.php/course/details?sem_id=d08ea3cd66c59967be1218a540bdefa7`)

Verspätete, nicht hochgeladene Abgaben, Abgaben, die per Mail getätigt werden, sowie Plagiate führen zur Bewertung der Aufgabe mit 0 Punkten.

*Insgesamt können Sie für das Lösen der Pflichtaufgabe **15 Punkte** erhalten. Berücksichtigt werden ausschließlich Ergebnisse, die bis zum **13.12.2020 um 23:59** in Ihr Git Repository auf `https://programming.ias.cs.tu-bs.de/` gepusht wurden.*

Ihre Lösung muss in den bereits für dieses Übungsblatt angelegten Ordner (blatt2) im Git Repository gespeichert werden, damit diese gewertet wird.

Nur compilierende Programme gelten als Lösung! Ein Programm das Teillösungen beinhaltet aber nicht compiliert oder aus anderen Gründen nicht ausführbar ist erhält 0 Punkte.

Aufgabe 1: 2007 wurde die *Internationale Standard-Buchnummer* (ISBN) 13-stellig eingeführt.
Beispielsweise ist 978-3-8274-1631-5 eine gültige 13-stellige ISBN. Die erste Zifferngruppe ist 978 oder 979. Die zweite Gruppe gibt die Sprachgruppe an. Der Ziffer 3 können wir entnehmen, dass das Buch im deutschsprachigen Raum erschienen ist. Die dritte Zifferngruppe 8274 gibt den Verlag an. Die vierte Zifferngruppe, hier 1631, identifiziert das betreffende Buch. Die letzte Ziffer ist eine sogenannte Prüfziffer. Mithilfe der Prüfziffer können z. B. einzelne fehlerhafte Ziffern oder zwei vertauschte Ziffern erkannt werden.
Die Prüfziffer wird wie folgt berechnet: Man multipliziert die erste Ziffer mit 1, die zweite mit 3, die dritte mit 1 und abwechselnd weiter mit 3, 1, 3, 1, 3, 1, 3, 1 und 3. Die so gewonnenen Produkte werden addiert. Die Differenz zur nächsten durch 10 teilbaren Zahl ist die Prüfziffer. Für das obige Beispiel ergibt sich die Prüfziffer also wie folgt:

$$105 = 9 + 3 * 7 + 8 + 3 * 3 + 8 + 3 * 2 + 7 + 3 * 4 + 1 + 3 * 6 + 3 + 3 * 1$$
$$5 = 110 - 105$$

Ziel dieser Aufgabe ist es ein Prüf-Programm für ISBN zu schreiben. Im ersten Schritt soll das Tool eine ISBN als Kommandozeilen Parameter erhalten. Im zweiten Schritt

soll geprüft werden, ob dies eine gültige 13-stellige ISBN darstellt. Wenn die Prüfziffer fehlerhaft ist, soll die ISBN mit korrekter Prüfziffer ausgegeben werden.

Beispiele:

```
javac Isbn.java

java Isbn 978-3-8274-1631-5
978-3-8274-1631-5 ist eine gültige ISBN.

java Isbn 978-3-446-44073-2
978-3-446-44073-2 ist eine fehlerhafte ISBN.
Gültig wäre 978-3-446-44073-9
```

Hinweise: Die Methode `char charAt(int index)` der Klasse `String` liefert das Zeichen, das an der Stelle `index` des Strings ist. Ein Beispiel ist `("Test".charAt(0)=='T')==true`. Durch den Typcast `(int)` erhält man das Zeichen als `int`-Wert. Die Zeichen `'0',...,'9'` haben die `int`-Werte `48,...,57`. Wegen der Codierung (s. Vorlesung) besitzen die Zeichen `'0',...,'9'` nicht die `int`-Werte `0,...,9`. Das Zeichen `'-'` besitzt den `int`-Wert `45`.

Aufgabe 2: Die Wahrscheinlichkeit $p(n)$, dass mindestens zwei Personen aus einer Gruppe von n Menschen am gleichen Tage Geburtstag haben, beträgt

$$p(n) = 1 - \frac{365}{365} \cdot \frac{364}{365} \cdot \frac{363}{365} \cdot \ldots \cdot \frac{365 - n + 1}{365}.$$

Schreiben Sie ein Java-Programm, um die Wahrscheinlichkeiten $p(2)$, ..., $p(60)$ zu berechnen und auszugeben. Wie viele Personen müssen der Gruppe mindestens angehören, damit die gesuchte Wahrscheinlichkeit größer als 50 % ist?

Aufgabe 3: Die Dezimalziffern einer natürlichen Zahl n, $n \geq 1$, werden einzeln quadriert und addiert. Anschließend wird mit der entstandenen Summe genauso verfahren. Die Ausgangszahl n ist *fröhlich*, wenn man bei diesem Vorgehen schließlich auf die Zahl 1 stößt, ansonsten ist sie *traurig*. Beispielsweise ist 7 eine *fröhliche* Zahl:

$$\begin{array}{rll} 7: & 7^2 & = 49 \\ 49: & 4^2 + 9^2 & = 97 \\ 97: & 9^2 + 7^2 & = 130 \\ 130: & 1^2 + 3^2 + 0^2 & = 10 \\ 10: & 1^2 + 0^2 & = 1 \end{array}$$

Schreiben Sie ein Java-Programm, das alle fröhlichen Zahlen zwischen einer Unter- und einer Obergrenze berechnet und die zugehörigen Folgen ausgibt. Der Dialog soll folgendermaßen ablaufen:

```
1         Bitte geben Sie die untere Grenze ein: 5
2         Bitte geben Sie die obere Grenze ein: 30
3         7 -> 49 -> 97 -> 130 -> 10 -> 1
4         10 -> 1
5         13 -> 10 -> 1
6         19 -> 82 -> 68 -> 100 -> 1
7         23 -> 13 -> 10 -> 1
8         28 -> 68 -> 100 -> 1
9
```

Hinweis: Für traurige Zahlen führt die Folge schließlich auf die Zahl 4.

Pflichtaufgabe 8: Autos und ihre Eigenschaften - (5 Punkte) Zum einfachen Herantasten an das Konzept der Objektorientierung wollen wir uns in dieser Aufgabe mit einer ersten einfachen zusätzlichen Klasse neben unserer Hauptklasse, in der wir immer arbeiten, befassen und lernen, wie man ein größeres Java-Projekt mit mehreren Dateien aufbauen kann. In der zweiten Aufgabe erweitern wir dann unser Verständnis von weiteren Konzepten und schreiben einen kleinen Objekte-Zoo.

Wir beginnen mit der Erstellung von zwei Java-Klassen. Für gewöhnlich erstellen wir für jede Klasse eine eigene .java-Datei. Erstellen Sie eine Datei **Auto.java** und **Main.java**. Erstellen Sie in der **Main.java** eine Klasse **Main** mit einer main-Methode. Legen Sie in der **Auto.java**-Datei eine Klasse **Auto** an.

Fügen Sie zu der Klasse **Auto** folgende Dinge hinzu: (2 Punkte)

- Ein privates Attribut *farbe*, dass die Farbe eines Autos repräsentieren soll
- Ein öffentliches Attribut Geschwindigkeit, dass den Wert der Geschwindigkeit speichert.
- Einen Konstruktor, dem Sie Farbe und Geschwindigkeit als Parameter übergeben können.
- Eine Methode *getFarbe()*, die die Farbe des Autos zurückgibt.

Fügen Sie der Klasse **Main** folgende Dinge hinzu: (1.5 Punkte)

- Erstellen Sie zwei Autos (*auto1* und *auto2*) mit unterschiedlichen Eigenschaften.
- Geben Sie nach der Erstellung aus, welches der beiden Autos schneller ist, indem Sie die Geschwindigkeitswerte miteinander vergleichen.
- Überprüfen Sie, ob beide Autos die gleiche Farbe besitzen.

Sie haben bereits den `==`-Operator und bei Strings die `.equals()`-Methode kennengelernt. Geben Sie sich nun in der **Main** das Ergebnis von `auto1 == auto1` aus. Sie werden erkennen, dass hier `true` ausgegeben wird. Setzen Sie nun Geschwindigkeit und Farbe ihrer beiden Autos auf den gleichen Wert. Vergleichen Sie jetzt beide Autos (auto1 und auto2) mit `==` und Sie werden feststellen, dass es sich zwar um das Gleiche Auto, aber nicht um das Selbe handelt. Ein kleiner aber feiner Unterschied. Genau, wie es bei Strings der Fall ist müssen wir, um festzustellen, ob zwei Instanzen zwei verschiedene Instanzen mit den gleichen (oder selben, oje...) Attributwerten sind, eine sogenannte `equals()`-Methode erstellen. Diese vergleicht dann die Werte von Geschwindigkeit und Farbe und gibt *true* oder *false* zurück.

Erstellen Sie nun also eine Methode `public boolean equals(Object obj)` wie folgt:

```java
@Override
public boolean equals(Object obj) {

}
```

Hinweis: Hiermit überladen Sie die Standard *equals*-Methode von Object. Sie können als Parameter jegliche Typen von Instanzen übergeben. Strings, Autos oder was auch immer. Wir wollen hier jedoch nur Autos mit einander vergleichen. Casten Sie daher die Variable *obj* zu einem Auto mittels: `Auto auto = (Auto) obj;`. Sorgen Sie nun dafür, dass Ihre *equals*-Methode entsprechend der Gleichheit der Attribute einen Wahrheitswert zurückgibt.

Ergänzen Sie jetzt Ihre Main-Methode um einen Vergleich von *auto1* und *auto2* mit equals. Sie sollten nun *true* erhalten, wenn beide Autos die gleiche Geschwindigkeit und die gleiche Farben haben. (1 Punkt)

Pflichtaufgabe 9: Objekte-Zoo - (15 Punkte) In dieser Aufgabe befassen wir uns ausführlich mit dem Konzept der Objektorientierung und erstellen unser erstes größere Java-Projekt.

Damit ein Projekt wie dieses stets übersichtlich bleibt, sollte darauf geachtet werden, dass alle Formatierungsrichtlinien eingehalten werden und man ausführlich dokumentiert. Zum Überprüfen, ob man sich an alle Richtlinien gehalten hat und sein Programm nicht nur macht, was es soll, sondern auch äußerlich in gutem Stil ist, gibt es Checkstyle. Mittels: `java -jar checkstyle-7.6.1-tubs.jar -c /tubs_checks.xml MyClass.java` überprüfen Sie die java-Datei MyClass, ob Sie den Regeln der Veranstaltung entspricht. Beachten Sie, dass Sie die Dateien *checkstyle-7.6.1-tubs.jar* und *tubs_checks.xml* bei den zu überprüfenden Klassen speichern oder statt *MyClass.java* einen vollständigen Pfad angeben.

Tipp: Sie können auch direkt alle *.java*-Dateien überprüfen, indem Sie in Ihrem Ordner mit den Dateien **.java* überprüfen.

Checken Sie mit Checkstyle die Dateien der **beiden** Pflichtaufgaben und sorgen Sie dafür, dass Sie alle entstandenen Fehler beheben. (2 Punkte)

Beginnen wir nun den Zoo zu erstellen. Lesen Sie bevor Sie mit dem Programmieren beginnen zuerst die Beschreibung aller Klassen, damit Sie sich ein Gesamtbild des Projektes machen können! Erstellen Sie dafür folgende Klassen:

Simulation: Im Objekte-Zoo geht es darum einen virtuellen Zoo mit ein paar Tieren auszustatten. Der Zoo verfügt über eine bestimme Menge an Essen und Trinken, womit die Tiere versorgt werden können. Unser Programm soll nun eine Simulation des Zoos durchführen und alle Tiere an jedem Tag mit Futter und Trinken versorgen. Die Simulation ermittelt dann wie lange die Verpflegung reicht und wann es nötig sein wird für Nachschub zu sorgen. Erstellen Sie hierfür zuerst eine Klasse **Simulation**, in der Sie auch die *main*-Methode anlegen. Legen Sie in dieser ein Objekt vom Typ Zoo an und lassen Sie ein paar Tiere in Ihren Zoo einfügen. Wie genau das geht, steht im Absatz *Zoo* genauer erklärt. Erstellen Sie nun eine Variable *zeit*, die verwaltet an welchem Tag sich unsere Simulation befindet. Sorgen Sie dann dafür, dass alle Tiere essen und trinken. Nutzen Sie eine *for-each*-Schleife, um das Array mit den Tieren durchzugehen. (1 Punkt) Ist nicht genügend Futter vorhanden, soll die Simulation gestoppt werden und man soll erkennen können, an welchem Tag nun neue Verpflegung organisiert werden muss. (1 Punkt)

Zoo: Die Klasse Zoo bildet das Hauptkonstrukt ihres Projektes. Erstellen Sie 4 private Parameter. Zwei Werte, die Essen und Trinken speichern (*double*), einen Wert, der die Gehegeanzahl speichert (int) und ein Array, dass quasi die Gehege repräsentiert (Tier[]). Sorgen Sie mit einem Keyword dafür, dass Ihre Variable für die Gehegeanzahl nicht mehr verändert werden kann (1 Punkt). Erstellen Sie einen Konstruktur `public Zoo(int essen, int wasser, int gehegeAnzahl)` der alle Attribute initialisiert. Sorgen Sie dafür, dass Sie

in der Simulation einen Zoo mit geeigneten Werten erzeugen (1 Punkt). Erstellen Sie in der Klasse **Zoo** außerdem folgende Methoden:

- `public void neuesTier(Tier tier)`: Diese Methode fügt ein neues Tier in ein freies Gehege ein. Ihr Tier Array ist standardmäßig an allen Stellen mit *null* initialisiert. Suchen Sie daher in ihrem Array nach einem freien Platz für das neue Tier und speichern Sie dieses darin. Ist kein Gehege mehr frei soll eine Meldung ausgegeben werden und das Tier nicht in den Zoo einziehen (1 Punkt).
- Für Essen und Wasser sollen Sie entsprechende *Getter*-Methoden erstellen.
- Erzeugen Sie eine Methode `public boolean fuettern(Tier tier)` und eine Methode zum *traenken* der Tiere mit dem gleichen Rückgabewert und Parameter. Bei den Methode soll geprüft werden, ob der Zoo noch über ausreichend Futter bzw. Wasser verfügt. Ist dies der Fall wird das Tier versorgt und *true* zurückgegeben. Andernfalls gibt die Methode *false* zurück (1 Punkt).
- Für die Simulation können Sie außerdem einen Getter für das Tier Array gebrauchen.

Tier: In einem Zoo leben Tiere. Also brauchen wir eine Klasse **Tier**. Jedes Tier hat einen *name*, *hunger*, *durst* und eine *groesse* als Attribute. Sorgen Sie mit Hilfe einer geeigneten Sichtbarkeit dafür, dass die Attribute auch in Klassen, die von Tier erben verfügbar sind (1 Punkt). Hunger und Durst geben an, wie viel ein durschnittliches Tier der Größe 1 am Tag isst und trinkt. Erstellen Sie nun einen Konstruktor, der den Namen, den Hunger, die Größe und den Durst übergeben bekommt und setzt. Erstellen Sie außerdem einen zweiten Konstruktor, bei dem Sie keine Größe angeben müssen. Diese ruft den anderen Konstruktor mit der Größe 1 auf (1 Punkt).

Jedes Tier will essen, jedes Tier will trinken! Erstellen Sie eine Methode `public double trinken()`, die den *durst* multipliziert mit der *groesse* zurückgibt. Erstellen Sie analog eine Methode zum Trinken (1 Punkt). Jedes Tier soll außerdem einen Ruf haben. Standardmäßig soll jedes Tier beim Aufruf von `toString()` *Trö* zurückgeben. Erstellen Sie außerdem einen Getter für den Namen.

Mit Hilfe eines bestimmten Java Keyword an der Klasse können Sie dafür sorgen, dass Sie keine Instanzen einer Klasse erstellt werden können. Wir wollen niemals ein Tier von der Rasse Tier erzeugen. Sorgen Sie also dafür, dass die Klasse **Tier** diese Funktionalität erhält. (1 Punkt)

Loewe: Wir besiedeln die Tierwelt nun mit zwei verschiedenen Tierrassen. Zuerst mit Loewen. Die Klasse **Löwe** soll alle Funktionalität vom Tier erben (1 Punkt). Dadurch besitzt der Löwe alle Attribute und Methoden, die auch die Klasse Tier bereitstellt. Ergänzen Sie beim Löwen außerdem eine Methode `private String berechneRuf()`, die für die *toString()*-Methode zufällig einen String erzeugt. Entweder der Löwe hat heute einen

guten Tag, dann brüllt er (*Groooooooooaaaaaar*) oder er bleibt heute ruhig (...) (1 Punkt).

Elefant: Als zweite Tierart erstellen wir die Elefanten. Ergänzen Sie hier die `toString()`-Methode so, dass ein Elefant, der besonders groß ist (Größe >= 2) zusätzlich zu dem, was jedes Tier als Ruf hat, ein weiteres *Töröö* ausgibt (1 Punkt).