Programmierung 1

PD Dr. Kaspar Riesen

Zusammenfassung & Musterlösungen der Serien

HS 2023

Lukas Batschelet

16-499-733

Sämtliches Material ist auch auf GitHub abgelegt: https://github.com/lbatschelet/23HS_P1

Dieses Werk ist lizenziert unter einer Creative Commons “Namensnennung – Nicht-kommerziell – Weitergabe unter gleichen Bedingungen 4.0 International” Lizenz.

Inhaltsverzeichnis

Teil I

Zusammenfassung

Kapitel 1-3

1.1 tl;dr Kapitel 1 bis 3

- *Programmieren* := Lösen von Problemen mit Software
- *Programmiersprache* := Wörter und Regeln um *Programmieranweisungen* zu definieren
- *Java* ist eine *weit verbreitete, vielfältig einsetzbare, plattformunabhängige, objektorientierte* Programmiersprache
- Java Programme werden mit *Klassen* erstellt
- Klassen enthalten *Methoden* (Verhalten) und *Variablen* (Eigenschaften)
- Die Methode `main` ist der Startpunkt eines jeden Java Programmes
- *Kommentare* erläutern, *weshalb* oder *wozu* Sie etwas tun:
  - `//` oder `/* */` oder `/** */`

```
public class Quote {
        /**
        * Gibt ein Zitat von Steve Jobs aus
        */
        public static void main(String[] args) {
            System.out.println("Steve Jobs:");
            System.out.println("Es ist besser, ein Pirat zu sein, "
                        + "als der Marine beizutreten.");
    }
}
```

- *Bezeichner* gehören zu einer der drei Kategorien:
  - Wörter, die für einen *bestimmten Zweck reserviert* sind (`class`, `int`, ...)
  - Wörter, die etwas aus *diesem* Programm bezeichnen (*eigene* Methode oder *eigene* Variable)
  - Wörter, die etwas aus dem *Java API* bezeichnen (`System`, `main`, `println`, ...)
- *Konventionen* für Bezeichner:
  - Klassen: `Student` oder `StudentActivity`
  - Methoden: `start` oder `findMin`
  - Variablen: `grade` oder `nextItem`
  - Konstanten: `MIN` oder `MAX_CAPACITY`
- *Java Quellcode* wird mit `javac` in *Bytecode übersetzt* (kompiliert)
- *Java Bytecode* wird mit `java` ausgeführt (*interpretiert*)
- *Fehler*:
  - Fehler beim Kompilieren (*Kompilier-* oder *Syntaxfehler*)
  - Fehler beim Interpretieren (*Laufzeitfehler*)

  - Fehler in der Semantik (*Logische Fehler*)
- Zeichen innerhalb doppelter Anführungszeichen sind *Zeichenketten*: `"Hallo Java"`
- Zeichenketten sind Objekte der Klasse `String`
- Zeichenketten können mittels *Konkatenation* miteinander «verklebt» werden: `"Hallo" + " " + "Java"`
- Gewisse Sonderzeichen erfordern *Escape-Sequenzen*: `"\n"` oder `"\t"`

```
System.out.println("Es gibt unendlich viele Primzahlen. Ein System, "
        + "welche Zahlen Primzahlen sind, ist nicht bekannt.");

System.out.println("\"Die Anzahl der Dummheiten übersteigt die der "
        + "Primzahlen.\nGibt es nicht unendlich viele "
        + "Primzahlen?\"\n\tGregor Brand");

System.out.println("Daher zählt die " + 1 + " nicht zu den Primzahlen.");
```

- *Variable* := Speicherort für einen Wert oder ein Objekt
- Variablen müssen mit *Datentyp* und *Bezeichner* deklariert werden
- Mit dem *Zuweisungsoperator* werden deklarierten Variablen Werte zugewiesen: `int i = 17;`
- Definierte Variablen können gelesen (*referenziert*) werden

```
int pages;
pages = 256;
int figures = 46, tables;
tables = 17;

System.out.println("Anzahl Seiten des Buches: " + pages);
System.out.println("Anzahl Abbildungen: " + figures
+ "; Anzahl Tabellen: " + tables);
```

- Konstanten werden mit `final` modifiziert: `final int MIN = 0;`
- Java kennt acht *primitive* Datentypen: (`byte`, `short`, `int`, `long`, `float`, `double`, `char`, `boolean`)
- Wechsel von «kleinen» zu «grossen»:

```
int count = 17;
double num = count; // num = 17.0
```

- Wechsel von «grossen» zu «kleinen» via *Cast*:

```
double num = 12.34;
int count = (int) num; // num = 12.34; count = 12
```

- *Ausdruck* := Kombination von einem oder mehreren *Operanden* und *Operatoren*
- Operanden sind Werte, Variablen oder Konstanten
- *Arithmetische Ausdrücke*:

```
double grade = (double) points / MAX_POINTS * 5 + 1;
```

- Lesen verändert Variablen niemals: `MAX_POINTS * 5`
- *Zuweisungsoperatoren* und das *Inkrement/Dekrement* machen das Leben einfacher:

```
points = points * 2;
points *= 2;

points = points + 1;
points++;
```

- *Boolesche Ausdrücke* sind entweder `true` oder `false`
- Boolesche Ausdrücke oder Boolesche Variablen können kombiniert und negiert werden

```
boolean smaller = hours < MAX;
boolean decision = (hours < MAX || hours > MIN) && !complete;
```

- Die `if`-*Anweisung* ist eine «Verzweigung», die auf einem Booleschen Ausdruck basiert:

```
if (hours < MAX) {
    hours += 10;
    System.out.println("10 Stunden hinzugefügt.");
    } else
    System.out.println("ACHTUNG: Maximum erreicht!");
```

- Das *Java API* besteht aus verschiedenen *Packages*, welche Klassen beinhalten, die Lösungen für *häufige Aufgaben* bereitstellen
- Sie kennen verschiedene Klassen: `String`, `Scanner`, `Random`, `DecimalFormat`.
- Der `new`-Operator *instanziiert* mit dem Aufruf des *Konstruktors* ein Objekt aus einer *Klasse* (Datentyp einer Objektvariablen := Klasse)

```
String str = new String("Hallo Welt");
Scanner scan = new Scanner(System.in);
Random rand = new Random();
```

- Methoden können mit dem *Punkt-Operator* auf instanziierten Objekten aufgerufen werden

```
int length = str.length();
int number = scan.nextInt();
double randomNumber = rand.nextFloat();
```

- *Primitive Datentypen*: Kopien von Variablen sind *unabhängig*

```
int num1 = 17;
int num2 = num1;
num2 = 99;
System.out.println(num1); // 17
System.out.println(num2); // 99
```

- *Objektvariablen*: Kopien von Variablen sind *abhängig* (*Aliase*)

```
Integer num1 = new Integer(17);
Integer num2 = num1;
num2.setValue(99);
System.out.println(num1); // 99
System.out.println(num2); // 99
```

Kapitel 4

2.1 tl;dr Kapitel 4

- Klassen enthalten *Variablen* und *Methoden* (Eigenschaften und Verhalten)
- *Sichtbarkeitsmodifikatoren* (`public`/`private`) bestimmen, was extern oder nur intern referenziert werden kann
- Variablen sollten `private` deklariert werden. Methoden können `private` oder `public` deklariert werden (je nach Zweck)

```
public class Integer {

    private int value;

    public Integer(int value) {
    this.value = value;
    }

    public String toString() {
    return this.value + "";
    }

    public void setValue(int value) {
    this.value = value;
    }
}
```

- Methoden bestehen aus *Methodenkopf* und *Methodenrumpf*
- Methodenkopf: (1) *Sichtbarkeit* (2) *Datentyp* der Rückgabe oder `void` (3) *Bezeichner* (4) *Formale Parameter* in Klammern
- Konstruktoren besitzen *keinen* Rückgabetyp und heissen immer gleich wie die zugehörige Klasse

```
public Integer(int value) {
    this.value = value;
}

public String toString() {
    return this.value + "";
}

public void setValue(int value) {
    this.value = value;
}
```

- Konstruktoren instanziieren Objekte aus der Klasse und geben eine *Referenz* auf das Objekt zurück

```
Integer num1 = new Integer(17);
```

- *Tatsächlicher Parameter*: Wird beim Aufruf an die Methode mitgegeben

```
num1.setValue(99);
```

- *Formaler Parameter*: Bezeichner, in den der tatsächliche Parameter kopiert wird

```
public void setValue(int value) {
    this.value = value;
}
```

- Methoden können mit einer `return`-Anweisung «etwas» zurückgeben
- Der Rückgabetyp und der Datentyp der Rückgabe müssen übereinstimmen

```
public String toString() {
    return this.value + "";
}
```

```
String value = num1.toString();
```

Kapitel 5-6

3.1 tl;dr Kapitel 5-6

- Die Klasse `Math` bietet mathematische Funktionen als *statische Methoden* an
- Statische Methoden können «direkt» ohne instanziiertes Objekt aufgerufen werden: `Math.sqrt(3);`
- Statische Methoden können auch verschachtelt werden:
  - `Math.sqrt(1 - Math.pow(Math.sin(alpha), 2));`
- Für jeden primitiven Datentyp existiert im Java API eine entsprechende *Wrapper Klasse*
- Hauptaufgabe dieser Klassen ist es, einen primitiven Datenwert zu umhüllen
  - `Double d = 4.567;`
- Die Wrapper bieten zudem hilfreiche statische Methoden und Konstanten an:
  - `Integer.MAX_VALUE`
  - `Double.parseDouble("4.567");`
  - `Double.POSITIVE_INFINITY`
  - `Boolean.toString(true)`
- `while`-Anweisungen erlauben es, gewisse Anweisungen mehrfach auszuführen, ohne diese mehrfach programmieren zu müssen

```
Random rand = new Random();
    System.out.println("1 : " + rand.nextInt(100));
    System.out.println("2 : " + rand.nextInt(100));
    System.out.println("3 : " + rand.nextInt(100));
    System.out.println("4 : " + rand.nextInt(100));
    System.out.println("5 : " + rand.nextInt(100));
    System.out.println("6 : " + rand.nextInt(100));
    System.out.println("7 : " + rand.nextInt(100));
    System.out.println("8 : " + rand.nextInt(100));
    System.out.println("9 : " + rand.nextInt(100));
    System.out.println("10 : " + rand.nextInt(100));
```

```
Random rand = new Random();
    int i = 1;
    while (i <= 10) {
        System.out.println(i + " : " + rand.nextInt(100));
        i++;
    }
```

- Mit *Wächterwerten* können wir ein Programm kontrollieren:

```
Scanner scan = new Scanner(System.in);
int input = 1;
while (input != 0) {
    System.out.print("Mit 0 Beenden Sie den Prozess. ");
    input = scan.nextInt();
}
System.out.println("--ENDE--");
```

- `while`-Schleifen können auch zur Kontrolle von Eingaben verwendet werden:

```
Scanner scan = new Scanner(System.in);
System.out.print("Alter eingeben: ");
int age = scan.nextInt();
while (age < 0) {
    System.out.println("Ungültiger Wert.");
    System.out.print("Alter eingeben: ");
    age = scan.nextInt();
}
```

- `while`-Schleifen können auch verschachtelt werden:

```
int counter = 2;
while (counter <= 20) {
    System.out.print("Teiler von " + counter + ":\t");
    int divisor = 1;
    while (divisor <= counter / 2) {
        if (counter % divisor == 0)
            System.out.print(divisor + " ");
        divisor++;
    }
    System.out.println();
    counter++;
}
```

- Wir können Klassen *generisch* machen:

```
public class Rocket<T> {

    private T cargo;

    public Rocket(T cargo) {
    this.cargo = cargo;
    }

    public void set(T cargo) {
    this.cargo = cargo;
    }

    public T get() {
    return this.cargo;
    }
}
```

- Um eine generische Klasse zu instanziieren, müssen wir sie zusammen mit einem *Typargument* instanziieren:
  - `Rocket<Integer> intRocket = new Rocket<Integer>();`
- Die *Typvariable* `T` wird nun überall mit dem Typargument `Integer` ersetzt

- Die Klasse `ArrayList` erlaubt es, Sammlungen von Objekten des Typs `T` anzulegen.
- Objekte dieser Klasse werden bei der Instanziierung *parametrisiert*:

```
ArrayList<String> names= new ArrayList<String>();
ArrayList<PlayerCard> cards = new ArrayList<PlayerCard>();
ArrayList<Integer> numbers = new ArrayList<Integer>();
```

- Listen passen Ihre Grösse dynamisch an:

```
names.add("Keanu");
names.add("Kevin");
System.out.println(names); // [Keanu, Kevin]
names.add("Karl");
System.out.println(names); // [Keanu, Kevin, Karl]
names.remove(1);
System.out.println(names); // [Keanu, Karl]
```

- Die `switch`-Anweisung bietet eine Alternative für (stark) verschachtelte `if`-Anweisungen:

```
if (i == 1)
    System.out.println("Eins");
else
    if (i == 2)
        System.out.println("Zwei");
    else
        if (i == 3)
            System.out.println("Drei");
        else
            if (i == 4)
                System.out.println("Vier");
            else
                if (i == 5)
                    System.out.println("Fünf");
                 else
                         System.out.println("Irgendwas anderes");
```

```
switch (i) {
    case 1: System.out.println("Eins"); break;
    case 2: System.out.println("Zwei"); break;
    case 3: System.out.println("Drei"); break;
    case 4: System.out.println("Vier"); break;
    case 5: System.out.println("Fünf"); break;
    default: System.out.println("Irgendwas anderes");
}
```

- Der *Conditional*bietet eine elegante Möglichkeit bei alternativen Zuweisungen:

```
if (points > MAX)
    points = points + 1;
else
    points = points * 2;
```

```
points = (points > MAX) ? points + 1 : points * 2;
```

- Die `do`-Anweisung ist ähnlich zur `while`-Anweisung, evaluiert aber die Boolesche Bedingung am Ende der Schleife:

```
System.out.print("Erreichte Punkte (0 bis 100): ");
int points = scan.nextInt();
while (points < 0 || points > 100) {
    System.out.print("Erreichte Punkte (0 bis 100): ");
    points = scan.nextInt();
}
```

```
int points;
do {
    System.out.print("Erreichte Punkte (0 bis 100): ");
    points = scan.nextInt();
} while (points < 0 || points > 100);
```

- Die `for`-Schleife ist gut geeignet, wenn man von Anfang an weiss, wie oft diese durchgeführt werden muss.
- Der Schleifenkopf der `for`-Schleife besteht aus drei Teilen:
  - *Initialisierung*: Wird am *Anfang* und *genau einmal* durchgeführt
  - *Boolesche Bedingung*: Wird immer *vor* dem nächsten Eintritt in die Schleife überprüft
  - *Inkrement*: Wird *immer am Ende* der Schleife durchgeführt

```
for (int i = 0; i < 10; i++)
    System.out.print(Math.pow(i, 2) + " ");
```

- Variante: `for-each`-Schleife - in jedem Durchgang zeigt die Variable auf das nächste Element einer `ArrayList`

```
for (String s : list)
    System.out.println(s);
```

- Vorsicht bei `==` auf Dezimalzahlen

```
final double TOLERANCE = 0.00000001;
if (Math.abs(num1 - num2) < TOLERANCE) {
```

- Vergleich von Zeichen basiert auf Unicode (Ziffern < Grossbuchstaben < Kleinbuchstaben)

```
char c0 = '0', c1 = 'A', c2 = 'a';
System.out.println(c0 < c1); // true
System.out.println(c1 < c2); // true
```

- Vorsicht bei `==` auf Objekten: Testet auf *Aliase*
- Verwenden/Schreiben der Methode `equals` und der Methode `compareTo`

```java
public class Integer {

    private int value;

    public Integer(int value) {
    this.value = value;
    }

    public boolean equals(Integer other) {
    return this.value == other.value;
    }

    public int compareTo(Integer other) {
    return this.value - other.value;
    }
}
```

```java
Integer i1 = new Integer(2);
Integer i2 = new Integer(17);
Integer i3 = new Integer(2);

System.out.println(i1.equals(i2)); // false
System.out.println(i1.equals(i3)); // true

System.out.println(i1.compareTo(i2)); // -15
System.out.println(i1.compareTo(i3)); // 0
System.out.println(i2.compareTo(i3)); // 15
```

Kapitel 7

4.1 tl;dr Kapitel 7-8

- *Arrays* ermöglichen das Deklarieren einer *einzigen* Variablen eines Typs, die dann *mehrere Werte* dieses Typs speichern kann.
- Arrays haben eine *feste, unveränderliche Größe* (Konstante `length`), die bei der Instanziierung angegeben werden muss.

```
int num1, num2, num3, num4, num5, num6;
```

```
int[] nums = new int[6];
```

```
int l = nums.length;
```

- Auf einzelne Elemente eines Arrays greift man mit einem *Index* innerhalb eckiger Klammern zu.

```
for (int i = 0; i < nums.length; i++) {
  System.out.println(nums[i]);
}
```

- Mit *Initialisierungslisten* können Arrays instanziiert und mit Werten gefüllt werden.

```
int[] nums = {1, 2, 3, 4};
String[] names = {"Goodbye", "Hello", "Hi", "Howdy"};
```

- Auf Methoden der in einem Array gespeicherten Objekte kann man über *Array-Referenzen* zugreifen.

```
for (int i = 0; i < names.length; i++) {
  names[i] = names[i].toUpperCase();
}
```

- Der Parameter der Methode `main` ist ein `String[]`. Dies sind *Programmparameter*, die beim Start des Programmes von „Außen“ mitgegeben werden können.

```
public static void main(String[] args) {
  String language = args[0];
  String version = args[1];
  String author = args[2];
}
```

- Methoden können mit *variablen Parameterlisten* umgehen.

```java
public static int min(int first, int ... others) {
  int min = first;
  for (int num : others) {
    min = Math.min(min, num);
  }
  return min;
}
```

```java
public class Greetings {
  private String primaryGreeting;
  private String[] greetings;

  public Greetings(String primaryGreeting, String ... otherGreetings) {
    this.primaryGreeting = primaryGreeting;
    this.greetings = otherGreetings;
  }
}
```

- Zweidimensionale Arrays sind Arrays aus Arrays.

```java
int[][] table = new int[100][5];
String[][] names = {{"Anne", "Barbara", "Cathrine"}, {"Danny", "Emilie", "Fanny"}};
```

- Für Referenzen auf Elemente in zweidimensionalen Arrays werden zwei Indizes benötigt.

```java
System.out.println(names[0][2]);

for (int row = 0; row < names.length; row++) {
  for (int col = 0; col < names[row].length; col++) {
    names[row][col] = "Hannes";
  }
}
```

4.1.1 enum

- Ein `enum` zählt *alle* zulässigen Werte eines Typs auf.

```java
public enum Category {
  Mathematik, Geographie
}
```

```java
public enum Category {
  Mathematik(10), Geographie(3);

  private int points;

  private Category(int points) {
    this.points = points;
  }
}
```

- Jedes `enum` Objekt besitzt Methoden (wie z.B. `name()`).
- Jede `enum` Klasse besitzt statische Methoden (wie z.B. `values()`).

```
Category[] categories = Category.values();
for (Category category : categories) {
  System.out.println(category.name());
}
```

4.1.2 Statische Variablen

- *Statische Variablen* werden von allen Instanzen geteilt (es existiert also nur *eine* Kopie der Variablen für alle Objekte).

```
public class Person {
  public static int globalCount = 0;
  private int id;

  public Person() {
    this.id = Person.globalCount++;
  }
}
```

- Statische Methoden werden direkt aufgerufen, ohne vorher ein Objekt zu instanzieren.

```
Functions.generateRandoms();

public static int[] generateRandoms() {
  // Macht etwas
}
```

4.1.3 Abhängigkeiten

Selbstabhängig

- Eine Klasse kann von sich selbst abhängig sein.

```
Person p1 = new Person("Emilie");
Person p2 = new Person("Ava");
Person p3 = new Person("Maya");

p1.knows(p2);
p1.knows(p3);

System.out.println(p1.getFriends()); // [Ava, Maya]
```

```java
public class Person {
  private String name;
  private ArrayList<Person> friends;

  public Person(String name) {
    this.name = name;
    this.friends = new ArrayList<Person>();
  }

  public void knows(Person other) {
    this.friends.add(other);
  }

  public String toString() {
    return this.name;
  }

  public ArrayList<Person> getFriends() {
    return this.friends;
  }
}
```

Aggregation

- Aggregation := Ein Objekt besteht z.T. aus anderen Objekten.

```java
public class Person {
    private String name;
    private Address address;

    public Person(String name, Address address) {
        this.name = name;
        this.address = address;
    }
}

public class Address {
    private String street;
    private int zipCode;

    public Address(String street, int zipCode) {
        this.street = street;
        this.zipCode = zipCode;
    }
}
```

Teil II

Serien

Serie 1

5.1 Theorieaufgaben

5.1.1 Bezeichner

Aufgabe: Finden Sie passende Bezeichner für ...

- eine Java Klasse, die eine Prüfung repräsentieren soll.
- die erreichten Punkte in einer Prüfung.
- eine Methode, welche den Durchschnittswert aller Prüfungen berechnet.
- die maximale Punktzahl, die in einer Prüfung erreicht werden kann.

Lösung:

- `public class Exam {}`
- `int points`
- `meanPoints()`
- `final int MAX_POINTS`

5.1.2 Variablen und Eigenschaften

Aufgabe: Finden Sie für die Klasse `Flight` mindestens drei Variablen (Eigenschaften) und drei Methoden (Verhalten/Funktionen), die in den Klassen modelliert werden könnten.

Mögliche Lösung:

Variablen

1. `altitude`
2. `cargoWeight`
3. `fuelRemaining`

Methoden

1. `bookSeat()`
2. `cancelFlight()`
3. `readFuelLevel()`

5.1.3 Zitat

Aufgabe: Schreiben Sie eine einzige `println` Anweisung, die die folgende Zeichenkette als Ausgabe generiert:

```
"Mein Name ist Winston Wolfe.
Ich löse Probleme!", stellte er sich vor.
```

Lösung:

```
System.out.println("\"Mein Name ist Winston Wolfe. \n" +
    "Ich löse Probleme!\", stellte er sich vor.");
```

5.1.4 Rechnung

Aufgabe: Welchen Wert enthält die Variable `result`, nachdem folgende Anweisungen durchgeführt worden sind?

```
int result = 25;
result = result + 5;
result = result / 7;
result = result * 3;
```

Lösung:

```
int result = 25;
result = result + 5; // 30
result = result / 7; // 4
result = result * 3; // 12
```

5.1.5 Rechnung

Aufgabe: Welchen Wert enthält die Variable `result`, nachdem folgende Anweisungen durchgeführt worden sind?

```
int result = 15, total = 100, min = 15, num = 10;
result /= (total - min) % num;
```

Lösung:

```
int result = 15, total = 100, min = 15, num = 10;
result /= (total - min) % num;
result = result / ((total - min) % num;
3 = 15 / (( 100 - 15) % 10;
```

5.1.6 Operationen

Aufgabe: Gegeben seien folgende Deklarationen:

```
int result1, num1 = 27, num2 = 5;
double result2, num3 = 12.0;
```

Welches Resultat wird jeweils durch folgende Anweisungen gespeichert?

```
result1 = num1 / num2;
result2 = num1 / num2;
result2 = num3 / num2;
result1 = (int) num3 / num2;
result2 = (double) num1 / num2;
```

Lösung:

```java
int result1, num1 = 27, num2 =5;
double result2, num3 = 12.0;
result1 = num1 / num2; // 5.0
result2 = num1 / num2; // 5.4
result2 = num3 / num2; // 2.4
result1 = (int) num3 / num2; // 2
result2 = (double) num1 / num2; // 5.4
```

5.1.7 Boolsche Operationen

Aufgabe: Gegeben seien folgende Deklarationen:

```java
int val1 = 15, val2 = 20;
boolean ok = false;
```

Was ist der Wert der folgenden Booleschen Ausdrücke?

```java
val1 <= val2
(val1 + 5) >= val2
val1 < val2 / 2
val1 != val2
!(val1 == val2)
(val1 < val2) || ok
(val1 > val2) || ok
(val1 < val2) && !ok
ok || !ok
```

Lösung:

```java
int val1 = 15, val2 = 20;
Boolean ok = false;
true
true
false
true
true
true
false
true
true
```

5.2 Implementationsaufgaben

5.2.1 Einfache Ausgabe - `WinterIsComing.java`

Aufgabe: Schreiben Sie ein Programm, welches den Satz "Winter is comingäusgibt (erste Version: auf einer Zeile; zweite Version: jedes Wort auf einer separaten Zeile).

Lösung:

```java
public class WinterIsComing {

    public static void main(String[] args) {

        System.out.println("\"Winter is coming\"");
        System.out.println("\"Winter \n" +
            "is \n" +
            "coming\"");
    }
}
```

5.2.2 Einfache Berechnungen - `Quotient.java`

Aufgabe: Schreiben Sie ein Programm, das vom Benutzer die Eingabe von zwei ganzzahligen Werten `a` und `b` fordert. Ihr Programm soll den Quotienten $\frac{a^2}{b}$ sowohl als Gleitkommazahl (d.h. ungerundet) als auch als ganze Zahl mit Rest berechnen und beide Ergebnisse am Bildschirm ausgeben. Testen Sie Ihr Programm mit beliebigen Zahlen.

Beobachten Sie insbesondere das Programmverhalten bei Eingabe der Zahl `0` als Divisor und versuchen Sie diesen Laufzeitfehler abzufangen.

Mögliche Lösung:

```java
import java.util.Scanner;

public class Quotient {
    public static void main(String[] args) {

        System.out.println("Dieses Programm berechnet den Quotienten
                            zweier Zahlen \"a\" und \"b\".");

        Scanner scan = new Scanner(System.in);
        System.out.println("Geben Sie den Teil \"a\" ein:");
        double var1 = scan.nextDouble();

        System.out.println("Geben Sie den Teil \"b\" ein:");
        double var2 = scan.nextDouble();

        if (var2 != 0) { //Wert 0 führt zu divide by zero
            double quotientDouble = (var1 * var1) / var2;
            // int quotientInt = (int) var1 * (int) var1) / (int) var2;
            int quotientInt = (int) quotientDouble;
            System.out.println("_____________________________________\n" +
                "Der Quotient Ihrer Zahlen: \t" + quotientDouble +
                "\nUnd als \"int\":\t \t \t" + quotientInt);
        } else {
            System.out.println("Geben Sie nicht 0 ein!");
        }
        scan.close();
    }
}
```

5.2.3 Benutzerinteraktion - `HumanThermometer.java`

Aufgabe: Schreiben Sie ein Programm, das vom Benutzer die Eingabe einer Temperatur `t` fordert. Die Ausgabe Ihres Programmes definiert sich danach folgendermassen:

$$\text{Ausgabe} = \begin{cases} \text{Kalt} & \text{wenn } t < 15, \\ \text{Angenehm} & \text{wenn } 15 \leq t < 24, \\ \text{Warm} & \text{wenn } t \geq 24 \end{cases}$$

Hinweis: Verwenden Sie Konstanten für beide Temperaturgrenzen.

Mögliche Lösung:

```
import java.util.Scanner;

public class HumanThermometer {
    public static void main(String[] args) {
    final int LOWER_BOUND = 15;
    final int UPPER_BOUND = 24;

    Scanner scan = new Scanner(System.in);
    System.out.println("Die aktuelle Temperatur: ");
    int temperature = scan.nextInt();

    if (temperature < LOWER_BOUND) {
      System.out.println("Es ist kalt");
    }
    else if ((LOWER_BOUND <= temperature) && (temperature <= UPPER_BOUND)) {
      System.out.println("Es ist angenehm");
    }
    else
      System.out.println("Es ist warm");

    scan.close();
  }
}
```

Serie 2

6.1 Theorieaufgaben

6.1.1 Der `new`-Operator

Aufgabe: Der `new`-Operator zusammen mit dem Konstruktor einer Klasse erledigt zwei Dinge. Was genau?

Lösung:

- Der `new`-Operator...
  - erzeugt/instanziiert ein neues Objekt aus der Klasse
  - erstellt eine Adresse für das Objekt und gibt diese zurück
- `Scanner scan = new Scanner(System.in);`
  - Objekt vom Typ Scanner ist instanziiert und die Adresse des Objektes ist der Variablen scan zugewiesen.

6.1.2 `ArrayList` in der Java API

Aufgabe: Informieren Sie sich in der Java API Dokumentation über die Klasse `ArrayList` (welche eine Liste für Objekte repräsentiert).

- Wie instanziieren Sie eine solche Liste?
- Wie fügen Sie ein Objekt zur Liste hinzu?
- Wie greifen Sie auf ein Objekt an Position `i` zu?
- Wie löschen Sie den gesamten Inhalt der Liste?
- Wie können Sie überprüfen, ob ein bestimmtes Objekt in der Liste vorhanden ist?

Lösung:

- Wie instanziieren Sie eine solche Liste?
  - `ArrayList list = new ArrayList();`
  - Oder mit einer Zuordnung des Typs (hier mit dem Beispiel `String`)
    - `ArrayList<String> list = new ArrayList<>();`
- Wie fügen Sie ein Objekt zur Liste hinzu?
  - mit der Methode `.add()` wird ein Objekt am Ende der Liste eingefügt
    - Möglicher Parameter ist der Index vom Typ `int`, welcher angibt an welcher Position das Objekt in die Liste eingefügt werden soll
- Wie greifen Sie auf ein Objekt an Position `i` zu?
  - mit der Methode `.get(i)`
- Wie löschen Sie den gesamten Inhalt der Liste?
  - mit der Methode `.clear()`
- Wie können Sie überprüfen, ob ein bestimmtes Objekt in der Liste vorhanden ist?

- mit der Methode `.contains()` wird ein `boolean` Wert zurückgegeben, ob das Objekt welches als Parameter "mitgegeben wurdeïn der Liste ist

6.1.3 Unterschied zwischen Klassen und Objekten

Aufgabe: Erläutern Sie anhand der Klasse `String` und des Objektes `"String"` den Unterschied zwischen Klasse und Objekt.

Lösung:

- Die Klasse `String` gibt den «Bauplan» für alle verschiedenen Objekte der Klasse vor. Sie definiert das Verhalten und die Eigenschaften eines `String` (Methoden und Variablen).
- Das Objekt `"String"` ist eine konkrete «Realisierung» der Klasse `String`. Dieses Objekt besitzt die in der Klasse definierten Methoden und Variablen:
  - es hat z.B. Länge 6 (Variable)
  - wir können z.B. die Methode `substring` darauf aufrufen.
    * `"String".substring(3); // "ing"`

6.1.4 Ausgabe

Aufgabe: Welche Ausgabe erzeugen folgende Anweisungen?

```
String testString = "Think different";
System.out.println(testString.length());
System.out.println(testString.substring(0, 4));
System.out.println(testString.toUpperCase());
System.out.println(testString.charAt(7));
System.out.println(testString);
```

Lösung:

```
15
Thin
THINK DIFFERENT
i
Think different
```

6.1.5 Random

Aufgabe: Gegeben sei eine Objektvariable vom Typ `Random` mit Bezeichner `rand`. In welchen Intervallen suchen die folgenden Anweisungen eine Zufallszahl?

- `rand.nextInt(100) + 1;`
- `rand.nextInt(51) + 100;`
- `rand.nextInt(10) - 5;`
- `rand.nextInt(3) - 3;`

Lösung:

- `rand.nextInt(100) + 1;`
  - `[1, 100]`
- `rand.nextInt(51) + 100;`
  - `[100, 150]`
- `rand.nextInt(10) - 5;`
  - `[-5, 4]`

- `rand.nextInt(3) - 3;`
  - `[-3, -1]`

6.1.6 Aliase

Aufgabe: Was sind Aliase und weshalb können Aliase problematisch sein?

Lösung:

- Aliase sind Variablen, die auf dasselbe Objekt zeigen. Bei primitiven Datentypen passiert das nicht, da der Wert kopiert wird.

```
int num1 = 17;
int num2 = num1;
num2 = 99; System.out.println(num1); // 17
System.out.println(num2); // 99
```

- Bei Objektvariablen sieht es anders aus:

```
Integer num1 = new Integer(17);
Integer num2 = num1;
num2.setValue(99);
System.out.println(num1); // 99
System.out.println(num2); // 99
```

- Hier greifen beide Variablen auf das selbe Objekt zu. Das ist nicht immer wünschenswert, weil:
  - Es ist unklar, welche Variable für was zuständig ist.
  - Änderungen an einer Variable beeinflussen die andere.
  - Der Code wird unübersichtlicher.
  - Es kann sogenannter *garbage* und damit einhergehender Datenverlust entstehen.
    - Im obigen Beispiel aus der Vorlesung ist dem Objekt `num1` selber kein Wert mehr zugewiesen, sondern nur noch die Verweisung auf `num2`. Der Wert `17` wird damit zu *garbage*

6.2 Implementationsaufgaben

6.2.1 Rechteck - `Rectangle.java`

Aufgabe: Schreiben Sie ein Programm, das nach der Länge und Breite eines Rechtecks fragt und danach die Fläche und den Umfang des Rechtecks berechnet und ausgibt. Zusätzlich soll Ihr Programm feststellen, ob es sich beim definierten Rechteck um ein Quadrat handelt oder nicht und eine entsprechende Ausgabe erzeugen.

Mögliche Lösung:

```java
import java.util.Scanner;

public class Rectangle {
    public static void main(String[] args) {

        System.out.println("Geben Sie die Länge des Rechtecks ein:");
        Scanner scan = new Scanner(System.in);
        double length = scan.nextDouble();
        System.out.println("Geben Sie die Breite des Rechtecks ein:");
        double width = scan.nextDouble();
        scan.close();

        double area = length * width;
        double perimeter = 2 * (length + width);

        System.out.println("Die Fläche des Rechtecks beträgt: " + area);
        System.out.println("Der Umfang des Rechtecks beträgt: " + perimeter);
        if (length == width) {
            System.out.println("Das Rechteck ist ein Quadrat.");
        } else {
            System.out.println("Das Rechteck ist kein Quadrat.");
        }
    }
}
```

6.2.2 Zufällige Addition - `RandomAddition.java`

Aufgabe: Schreiben Sie ein Programm, das eine zufällige Additionsaufgabe mit zwei positiven Zahlen anzeigt. Die Summe der beiden Zahlen darf maximal 20 betragen. Der Benutzer soll dann ein Ergebnis eingeben können und das Programm soll überprüfen, ob die Eingabe korrekt war oder nicht und eine entsprechende Rückmeldung ausgeben.

Mögliche Lösung:

```java
import java.util.Random;
import java.util.Scanner;

public class RandomAddition {
    public static void main(String[] args) {

        Random random = new Random();
        final int MAX = 21;
        int number1 = random.nextInt(MAX);
        // Die Summe der beiden Zahlen darf maximal 20 betragen.
        int number2 = random.nextInt(MAX - number1);
        int sum = number1 + number2;

        Scanner scan = new Scanner(System.in);

        System.out.println("Die Aufgabe lautet: " + number1 + " + " + number2);
        System.out.println("Geben Sie das Ergebnis ein:");
        int guess = scan.nextInt();

        scan.close();

        if (guess == sum) {
            System.out.println("Das Ergebnis ist korrekt!");
        } else {
            System.out.println("Das Ergebnis ist falsch!");
        }
    }
}
```

6.2.3 Username und Passwort - `UsernameAndPassword.java`

Aufgabe: Schreiben Sie ein Programm, das einen Benutzer separat nach seinem Vor- und Nachnamen fragt und diese einliest. Danach soll ein Benutzername nach folgendem Muster erzeugt werden:

$$FL_1L_2L_3L_4L_5D_1D_2D_3$$

wobei

- F dem ersten Buchstaben des Vornamens entspricht.
- L_i dem i-ten Buchstaben des Nachnamens entspricht.
- $D_1D_2D_3$ einer zufälligen Zahl zwischen 000 und 999 entspricht.

Falls der eingegebene Nachname kürzer als 5 Zeichen sein sollte, werden entsprechend weniger Zeichen verwendet.

Zusätzlich erzeugen Sie ein zufälliges Passwort für den Benutzer. Das Passwort soll mit einer 7 oder 8 oder 9 starten, gefolgt von 5 zufälligen ganzen Zahlen von 0 bis 9, gefolgt von einem Bindestrich -, gefolgt von drei zufälligen Grossbuchstaben.

Geben Sie Benutzernamen und Passwort aus.

Tipp: Die Zahlen 65 bis 90 repräsentieren die Grossbuchstaben A bis Z in Unicode und Sie können bspw. die ganze Zahl 77 folgendermassen in einen Buchstaben umwandeln:

```java
char d1 = (char) 77;
```

Mögliche Lösung:

```java
import java.util.Scanner;
import java.util.Random;

public class UsernameAndPassword {
    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        System.out.println("Geben Sie Ihren Vornamen ein:");
        String firstName = scan.nextLine();
        System.out.println("Geben Sie Ihren Nachnamen ein:");
        String lastName = scan.nextLine();
        scan.close();

        int lastNameLength = lastName.length();
        final int MAX_LETTERS_OF_LASTNAME = 5;

        String firstLetter = firstName.substring(0, 1);
        // Math.min(a, b) sorgt dafür, dass auch Nachnamen mit weniger
        // als MAX_LETTERS_OF_LASTNAME Buchstaben funktionieren
        String lastNameArray = lastName.substring(0, Math.min(MAX_LETTERS_OF_LASTNAME,
                   lastNameLength));

        Random random = new Random();
        int randomNumber = random.nextInt(1000);
        // String.format("%03d", randomNumber) fügt führende Nullen hinzu,
        // falls randomNumber < 100
        String username = firstLetter.toUpperCase() + lastNameArray.toUpperCase() +
                  String.format("%03d", randomNumber);


        System.out.println("Der Benutzername lautet: " + username);

        final int UNICODE_LOWER_BOUND = 65;
        final int UNICODE_UPPER_BOUND = 91;
        // ermöglicht die zufällige Ausgabe des ganzen Zahlenteils des Passworts
        int randomPasswordNumber = random.nextInt(700000, 1000000);
        char randomPasswordLetter1 = (char) random.nextInt(UNICODE_LOWER_BOUND,
                                   UNICODE_UPPER_BOUND);
        char randomPasswordLetter2 = (char) random.nextInt(UNICODE_LOWER_BOUND,
                                     UNICODE_UPPER_BOUND);
        char randomPasswordLetter3 = (char) random.nextInt(UNICODE_LOWER_BOUND,
                                     UNICODE_UPPER_BOUND);

        String password = randomPasswordNumber + "-" + randomPasswordLetter1 +
              randomPasswordLetter2 + randomPasswordLetter3;

        System.out.println("Das Passwort lautet: " + password);
    }
}
```

Serie 3

7.1 Implementationsaufgaben

7.1.1 Thermometer - `Thermometer`

Aufgabe: Programmieren Sie eine Klasse `Thermometer`, welche einen einfachen Fieberthermometer modelliert. Die Klasse soll eine Temperatur in Celsius als einzige Instanzvariable speichern. Der Konstruktor soll diese Instanzvariable standardmässig auf 37.0 Grad setzen. Schreiben Sie eine Methode `increase`, welche die Temperatur um 0.1 Grad erhöht und einen Getter für die Temperatur. Zudem definieren Sie eine Methode `reset`, welche die Temperatur wieder auf 37.0 zurücksetzt.

Schreiben Sie eine zweite Klasse `ThermometerTest`, in der Sie zwei Objekte vom Typ `Thermometer` instanziieren und deren Methoden ausführlich testen.

Mögliche Lösung:

Klasse `Thermometer.java`

```java
public class Thermometer {

    private double temperature;

    public Thermometer() {
        this.temperature = 37.0;
    }

    public void increase() {
        this.temperature += 0.1;
    }

    public double getTemperature() {
        return this.temperature;
    }

    public void reset() {
        this.temperature = 37.0;
    }
}
```

Klasse `ThermometerTest.java`

```
public class ThermometerTest {

        public static void main(String[] args) {
            Thermometer thermometer1 = new Thermometer();
            Thermometer thermometer2 = new Thermometer();
            System.out.println(thermometer1.getTemperature());
            System.out.println(thermometer2.getTemperature());
            thermometer1.increase();
            System.out.println(thermometer1.getTemperature());
            System.out.println(thermometer2.getTemperature());

            for (int i = 0; i < 10; i++) {
                thermometer2.increase();
            }
            System.out.println(thermometer1.getTemperature());
            System.out.println(thermometer2.getTemperature());
            thermometer1.reset();
            thermometer2.reset();
            System.out.println(thermometer1.getTemperature());
            System.out.println(thermometer2.getTemperature());
        }
}
```

7.1.2 `Car` -Klasse

Aufgabe: Programmieren Sie eine Klasse `Car`, welche die Marke, das Modell und den Jahrgang des Fahrzeuges modelliert. Der Konstruktor soll diese drei Instanzvariablen gemäss Parameterübergabe initialisieren – zudem schreiben Sie Getter und Setter für alle Instanzvariablen und eine `toString` Methode für eine einzeilige Repräsentation von `Car` Objekten. Schliesslich definieren Sie eine Methode `isAntique`, welche einen boolean zurückgibt, der anzeigt ob das Auto aktuell älter ist als 45 Jahre.

In einer zweiten Klasse `Garage` instanziieren Sie drei `Car` Objekte und testen alle programmierten Methoden.

Mögliche Lösung:

Klasse `Car.java`

```java
import java.time.LocalDate;

public class Car {

    private String brand;
    private String model;
    private int year;
    private boolean isAntique;
    private final int ANTIQUE_AGE = 45;

    public Car(String brand, String model, int year) {
        this.brand = brand;
        this.model = model;
        this.year = year;
        this.isAntique = checkIfAntique();
    }

    public String getBrand() {
        return this.brand;
    }

    public String getModel() {
        return this.model;
    }

    public int getYear() {
        return this.year;
    }

    public boolean getIsAntique() {
        return this.isAntique;
    }

    public void setBrand(String brand) {
        this.brand = brand;
    }

    public void setModel(String model) {
        this.model= model;
    }

    public void setYear(int year) {
        this.year = year;
        this.isAntique = checkIfAntique();
    }

    private boolean checkIfAntique() {
        int currentYear = LocalDate.now().getYear();
        if (currentYear - this.year > ANTIQUE_AGE) {
            return true;
        } else {
            return false;
        }
    }

    public boolean isAntique() {
        return this.isAntique;
    }

    public String toString() {
        return this.brand + " " + this.model + " " + this.year + " Is Antique: " + this.isAntique;
    }
}
```

Klasse `Garage.java`

```java
public class Garage {

        public static void main(String[] args) {
            Car car1 = new Car("BMW", "M3", 1900);
            Car car2 = new Car("Mercedes", "C63", 2010);
            Car car3 = new Car("Audi", "RS6", 2015);
            System.out.println("Car1 " + car1);
            System.out.println("Car2 " + car2);
            System.out.println("Car3 " + car3);

            car1.setYear(2000);
            System.out.println("Set car1 year to 2000");
            System.out.println("Car1: " + car1);

            car2.setBrand("Mercedes-Benz");
            System.out.println("Set car2 brand to Mercedes-Benz");
            System.out.println("Car2: " + car2);

            car3.setModel("RS7");
            System.out.println("Set car3 model to RS7");
            System.out.println("Car3: " + car3);

            car1.setYear(1984);
            System.out.println("Set car1 year to 1984");
            System.out.println("Car1: " + car1.getYear());
            System.out.println("Car1: " + car1.isAntique());

            car2.setBrand("Mercedes");
            System.out.println("Set car2 brand to Mercedes");
            System.out.println("Car2: " + car2.getBrand());

            car3.setModel("RS5");
            System.out.println("Set car3 model to RS5");
            System.out.println("Car3: " + car3.getModel());
        }
}
```

7.1.3 `Cargo` und `Box` -Klassen

Aufgabe: Schreiben Sie eine Klasse `Cargo`, welche ein Stückgut mit Länge, Breite, Höhe und einem Namen modelliert (z.B. 30, 44, 65, "Kaffeemaschine"). Schreiben Sie einen Konstruktor, Getter und Setter für alle Instanzvariablen und eine Methode `toString`.

Schreiben Sie eine Klasse `Box`, die Instanzvariablen für die Länge, Breite und Höhe einer Box enthält. Zusätzlich enthält die Klasse `Box` eine Instanzvariable `full` vom Typ boolean, die angibt, ob die Box gefüllt ist oder nicht, sowie eine Instanzvariable `cargo` vom Typ `Cargo`. Der Konstruktor setzt die Länge, Breite und Höhe einer Box gemäss Parametern – neu instanziierte `Box` Objekte sollen standardmässig leer sein. Definieren Sie einen zweiten Konstruktor ohne Parameter, der eine Standard-Box mit Länge, Breite und Höhe 1 generiert. Zusätzlich definieren Sie eine Methode `getCapacity`, die das Volumen der Box berechnet und zurückgibt.

Schliesslich schreiben Sie eine Methode `addCargo`, welche ein Objekt vom Typ `Cargo` als Parameter entgegennimmt. Falls dieses Stückgut gemäss Länge, Breite und Höhe in die Box passt, passen Sie die Variable `full` und die Instanzvariable `cargo` an und geben `true` zurück (andernfalls `false`).

Testen Sie die Klasse `Box`, indem Sie in einer weiteren Klasse `BoxTest` drei `Box` Objekte instanziieren, manipulieren und ausgeben.

Mögliche Lösung:

Klasse `Cargo.java`

```java
public class Cargo {

    private String name;
    private int length;
    private int width;
    private int height;

    public Cargo(String name, int length, int width, int height) {
        this.name = name;
        this.length = length;
        this.width = width;
        this.height = height;
    }

    public String getName() {
        return this.name;
    }

    public int getLength() {
        return this.length;
    }

    public int getWidth() {
        return this.width;
    }

    public int getHeight() {
        return this.height;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setLength(int length) {
        if (length > 0) {
            this.length = length;
        }
    }

    public void setWidth(int width) {
        if (width > 0) {
            this.width = width;
        }
    }

    public void setHeight(int height) {
        if (height > 0) {
            this.height = height;
        }
    }

    public String toString() {
        return "Cargo: \tName: " + this.name + " Dimensions: " + this.length + "x" + this.width + "x"
    }
}
```

Klasse `Box.java`

```java
import java.util.ArrayList;
import java.util.Collections;


public class Box {

    private Cargo cargo;
    private boolean isFull;
    private int length;
    private int width;
    private int height;

    public Box(int length, int width, int height) {
        this.length = length;
        this.width = width;
        this.height = height;
        this.isFull = false;
    }

    public Box() {
        this(1, 1, 1); // refers to the other constructor
    }

    public int getCapacity() {
        return this.length * this.width * this.height;
    }

    public int getLength() {
        return this.length;
    }

    public int getWidth() {
        return this.width;
    }

    public int getHeight() {
        return this.height;
    }

    public boolean getIsFull() {
        return this.isFull;
    }

    public Cargo getCargo() {
        return this.cargo;
    }

    public void setLength(int length) {
        if (length > 0) {
            this.length = length;
        }
    }

    public void setWidth(int width) {
        if (width > 0) {
            this.width = width;
        }
    }

    public void setHeight(int height) {
        if (height > 0) {
            this.height = height;
        }
```

Klasse BoxTest.java

```
public class BoxTest {

    public static void main(String[] args) {

        Cargo cargo1 = new Cargo("Kaffeemaschine", 30, 44, 65);
        Cargo cargo2 = new Cargo("Kühlschrank", 60, 60, 180);
        Cargo cargo3 = new Cargo("Küchentisch", 80, 120, 75);
        Cargo cargo4 = new Cargo("Küchenstuhl", 40, 40, 90);

        System.out.println("Cargo1: " + cargo1);
        System.out.println("Cargo2: " + cargo2);
        System.out.println("Cargo3: " + cargo3);
        System.out.println("Cargo4: " + cargo4);

        cargo1.setName("Espressomaschine");
        System.out.println("Set cargo1 name to Espressomaschine");
        System.out.println("Cargo1: " + cargo1.getName());

        cargo2.setLength(70);
        System.out.println("Set cargo2 length to 70");
        System.out.println("Cargo2: " + cargo2.getName() + " Length: " + cargo2.getLength());

        cargo3.setWidth(100);
        System.out.println("Set cargo3 width to 100");
        System.out.println("Cargo3: " + cargo3.getName() + " Width: " + cargo3.getWidth());

        cargo4.setHeight(100);
        System.out.println("Set cargo4 height to 100");
        System.out.println("Cargo4: " + cargo4.getName() + " Height: " + cargo4.getHeight());

        Box box1 = new Box(50, 50, 50);
        Box box2 = new Box(100, 100, 100);
        Box box3 = new Box();

        System.out.println("Box1: " + box1);
        System.out.println("Box2: " + box2);
        System.out.println("Box3: " + box3);

        box1.setLength(100);
        System.out.println("Set box1 length to 100");
        System.out.println("Box1: " + box1.getLength());

        box2.setWidth(50);
        System.out.println("Set box2 width to 50");
        System.out.println("Box2: " + box2.getWidth());

        box3.setHeight(100);
        System.out.println("Set box3 height to 100");
        System.out.println("Box3: " + box3.getHeight());

        System.out.println("\n" + cargo1);
        System.out.println( box1);
        System.out.println("Placing cargo1 in box1 (NOTE: Dimensions don't match up, Box has to be tu
        box1.addCargo(cargo1);
        System.out.println("Box1: " + box1.getCargo());

        System.out.println("Trying to add another cargo into the same Box");
        box1.addCargo(cargo2);

        System.out.println("\nEmpty Box1");
        box1.removeCargo();
        System.out.println(box1);
```

7.1.4 `Book` -Klasse

Aufgabe: Auf ILIAS (Übungen → Serie 3) finden Sie eine Datei `Book.java`. Ihre Aufgabe ist es die darin implementierte Klasse `Book` wie folgt zu erweitern:

1. Schreiben Sie einen Konstruktor sowie Getter und Setter für alle Instanzvariablen.
2. Implementieren Sie die Methode `age`, welche das Alter eines Buches (Anzahl Tage seit Erscheinungsdatum) berechnet und zurückgibt.
3. Implementieren Sie die Methode `toString`, die alle Informationen eines `Book` Objekts als String zurückgibt. Beispiel: 123, Die Blechtrommel, Günter Grass, 1.1.1959
4. Vervollständigen Sie die Methode `input`, welche die Werte für `id`, `title`, `author` und `dateOfPublication` von der Kommandozeile vom Benutzer einliest und im jeweiligen `Book` Objekt abspeichert. Ungültige Eingaben müssen Sie nicht abfangen.

Testen Sie sämtliche Methoden der Klasse `Book` in einer zusätzlichen Klasse `BookShelf`.

Mögliche Lösung:

Klasse `Book.java`