HOCHSCHULE RUHR WEST
UNIVERSITY OF APPLIED SCIENCES

19.03.2021

Klausur

Programmierung 2

Prüfer: Prof. Dr. Malte Weiß

Vorname	Nachname	Matrikelnummer

Bewertung:

1	2	3	4	**Gesamt**	**Note**
37	34	11	18	100	

Zum Ablauf der Prüfung

- **Zeit für die Lösung:** 4 Stunden
- **Beginn:** 14:00 Uhr, **Ende:** 18:00 Uhr am 19.03.2021
- **Lösung *wo* hochladen:** in E-Learning im Kursraum der Vorlesung
- **Lösung *wie* hochladen (Form):** Zip-Datei im Format *prog2-exam-VORNAME-NACHNAME-MATRIKELNUMMER.zip*.
  Beispiel: Student *Mathias Bauer* mit Matrikelnummer *10001234* gibt eine Zip-Datei prog2-exam-Mathias-Bauer-10001234.zip ab.
- **Ordnerstruktur:** Jede Aufgabe liegt in einem entsprechenden Unterordner in der Zip-Datei. Siehe aufgabenspezifische Konventionen.
- **Nicht erlaubte Hilfsmittel/Hilfe:** Kommunikation digital oder persönlich, direkt oder indirekt mit anderen Menschen (**keine** SMS, WWW-Kommunikation, Telefon, E-Mail, Chat, etc.)
- **Kompilierbarkeit:** Lösungen *müssen* kompilieren, ansonsten werden 50 % der Aufgabenpunkte abgezogen.
- **Fragen:** Für Fragen während der Klausurzeit nutzen Sie bitte das Forum im E-Learning Bereich der Veranstaltung. Preisgabe von Lösungen oder Ansätzen führt zu Nichtbestehen der Veranstaltung inkl. Fehlversuch.
- Die Klausur gilt als **nicht angetreten**, wenn Sie bei Beginn der Klausur nicht die vorgeschriebene eidesstattliche Versicherung unterschrieben haben.

Aufgabe 1: Klassenhierarchie (37 Punkte)

Die Lösung der Aufgabe muss im ZIP-Archiv im Ordner A1 gespeichert werden.

Gegeben ist folgende Klassenhierarchie, die Objekte mit finanziellem Wert modelliert:

- Die abstrakte Klasse **ValuableObject** bildet die abstrakte Oberklasse aller weiteren Klassen. Sie beschreibt ein Artefakt, das einen finanziellen Wert (in Euro) besitzt. Sie besitzt als Attribute einen Namen `name` (String) und eine abstrakte Methode `getValue()`, die den Wert des Artefakts als double zurückgibt.
- Die abstrakte Klasse **TechnicalDevice** stellt ein technisches Gerät dar und erbt von **ValuableObject**. Sie besitzt als Attribute eine Spannung `voltage` (double), Stromstärke `electricity` (double) sowie eine antizipierte Lebensdauer `lifeSpan` (int) in ganzen Jahren.
- Die abstrakte Klasse **Art** stellt Kunst dar und erbt von **ValuableObject**. Sie besitzt als Attribute ein Herstellungsdatum `manufacturingDate` (java.util.Date) sowie einen Hersteller `manufacturer` (String).
- Die konkrete Klasse **Computer** stellt einen Computer dar und erbt von **TechnicalDevice**. Sie besitzt als Attribut den long-Wert `cpuClocking`, welcher die CPU-Taktung des Computers repräsentiert. Der Wert eines Computers entspricht der Hälfte des cpuClocking-Attributs.
- Die konkrete Klasse **ElectricCar** stellt ein Elektroauto dar und erbt ebenfalls von **TechnicalDevice**. Sie besitzt ein Attribut `range` vom Typ int, welches die Reichweite des elektrischen Fahrzeugs in km beschreibt. Der Wert eines Elektroautos entspricht der zehnfachen Reichweite des Fahrzeugs.
- Die konkrete Klasse **Painting** stellt ein Gemälde dar und erbt von **Art**. Ein Gemälde benutzt einen Aufzählungstypen **ColorType**, um den Farbtyp des Gemäldes zu beschreiben. Gemälde können aus Ölfarbe (`OIL`), Acrylfarbe (`ACRYLIC`) oder Aquarellfarbe (`WATERCOLOR`) hergestellt sein.
  Ölgemälde kosten pauschal 500 Euro, Acrylgemälde 200 Euro, Aquarellbilder kosten 20 Euro. Diese Kosten sollen als Attribut in den Aufzählungstyp **ColorType** integriert sein. Beispiel: **ColorType.ACRYLIC.getCost()** liefert den Wert 200. Der Wert eines Gemäldes entspricht den Kosten multipliziert mit dem Faktor 1,1.
- Die konkrete Klasse **Sculpture** stellt eine Skulptur dar und erbt ebenfalls von **Art**. Eine Skulptur besitzt als Attribute ein Gewicht `weight` (double) in kg und eine Höhe `height` (double) in m. Der Wert einer Skulptur entspricht dem Produkt aus Gewicht und Höhe (`weight * height`).
- Für jede Klasse soll die Methode **toString()** implementiert werden. Sie gibt wie üblich alle Attribute der jeweiligen Klasse menschenlesbar auf dem Bildschirm aus. Achten Sie darauf, dass auch Attribute der Oberklassen ausgegeben werden sollten.
- Die Methode `agingFactor()` gibt für ein technisches Gerät zurück, wie schnell das jeweilige Gerät altert. Sie wird als Methode in der Klasse **TechnicalDevice** implementiert.
  Die Alterungsrate eines technischen Geräts entspricht dem Produkt von Spannung

und Stromstärke (`voltage * electricity`). Elektroautos altern allerdings 10% schneller als übliche Geräte.

- Die Schnittstelle (Interface) **CanWriteToStream** schreibt eine Methode `writeToStream()` vor. Diese besitzt keinen Rückgabetyp und erhält einen OutputStream als Parameter. Eine Klasse, die diese Schnittstelle implementiert, implementiert die Methode **writeToStream** so, dass der Inhalt des Objekts menschenlesbar in den Stream geschrieben wird.
- Die Klassen **Computer** und **Sculpture** implementieren **CanWriteToStream**.

Ihre Aufgaben sind:

a) Implementieren Sie die oben dargestellte Klassenhierarchie.

Jede Klasse implementiert angemessene Konstruktoren, Getter & Setter und alle Methoden und Schnittstellen entsprechend der oben beschriebenen Klassenhierarchie. (26 P)

b) Schreiben Sie ein Hauptprogramm, das die Klassenhierarchie ausgiebig demonstriert, indem Sie Objekte anlegen, Hilfsmethoden testen und Polymorphie darstellen. (4 P)

c) Schreiben Sie eine Hilfsfunktion **writeEverythingToStream**. Sie erhält einen Ausgabestream und eine Liste von Objekten. Jedes Objekt, das **CanWriteToStream** implementiert, soll seinen Inhalt in den Ausgabestream schreiben. Testen Sie diesen Mechanismus in einem Hauptprogramm. (7 P)

Aufgabe 2: Notendatenbank (34 Punkte)

Die Lösung der Aufgabe muss im ZIP-Archiv im Ordner A2 gespeichert werden.

In dieser Aufgabe soll eine einfache Notentabelle aus einer Datei gelesen und ausgewertet werden.

Ein Studierender ist definiert als eine Datenstruktur mit folgenden Attributen:

- Name
- Matrikelnummer
- Semesterzahl

Die Noten einer Klausur sind zeilenweisen als CSV-Datei (komma-separierte Textdatei) gespeichert. Die beigefügte Testdatei `grades.csv` enthält zum Beispiel diese Daten:

```
Student name,Student number,Semester,Grade
Alice,912821,2,1.0
Bob,129123,1,2.0
Charles,91828,3,1.3
Daisy,48311,2,2.7
Edward,72123,1,5.0
Florence,110002,4,4.0
```

Hinweis: Die erste Zeile enthält Tabellenköpfe und keine Daten.

a) Erstellen Sie eine Java-Klasse, die einen Studierenden gemäß der obigen Datenstruktur repräsentiert. Die Klasse sollte über Getter & Setter, mindestens einen Konstruktor und eine toString-Methode verfügen. (4 P)

b) Erstellen Sie eine Klasse **GradeDatabase** im Paket **de.hrw.java.exam.files.student**. Die Klasse enthält als einziges Attribut eine Abbildung (Map) **studentToGradeMap** von Studenten-Objekten auf Noten, wobei Noten als Double-Objekte hinterlegt sind. Die Klasse besitzt einen Konstruktor, der ein File-Objekt übergeben bekommt. Das Objekt verweist auf eine CSV-Datei der oben genannten Struktur. Der Konstruktor liest die gegebene Datei Zeile für Zeile aus. Für jede Zeile wird ein Student-Objekt generiert und die Note extrahiert. Beides wird dann im Attribut **studentToGradeMap** hinterlegt.

   Enthält die Datei eine ungültige Note (< 1 oder > 5), wird eine von Ihnen definierte **InvalidGradeException** ausgelöst. Schlägt das Lesen der Datei aus irgendeinem anderen Grund fehl, wird stattdessen eine von Ihnen definierte **CannotReadGradeDatabaseException** geworfen.

   Nach dem Aufruf des Konstruktors wurde die Datei eingelesen und als Abbildung von Studenten auf Noten abgelegt. Schreiben Sie außerdem ein Hauptprogramm, das alles testet. (15 P)

c) Schreiben Sie eine parameterlose Methode **printStudentsGrade** ohne Rückgabetyp in der Klasse **GradeDatabase**, die alle Studierenden und ihre Noten auf der Konsole ausgibt.

Ergänzen Sie außerdem Ihr Hauptprogramm, um diese Methode zu testen. (4 P)

d) Schreiben Sie eine parameterlose Methode **getAverageGrade** in der Klasse **GradeDatabase**. Sie gibt die durchschnittliche Note aller Studierenden zurück.

Ergänzen Sie außerdem Ihr Hauptprogramm, um diese Methode zu testen. (4 P)

e) Schreiben Sie eine parameterlose Methode **getGradeHistogram** in der Klasse **GradeDatabase**. Sie gibt ein Notenhistogramm zurück, das für jede Note im Einserbereich (>= 1 und < 2), im Zweierbereich (>= 2 und < 3), im Dreierbereich (>= 3 und < 4), im Viererbereich (>= 4 und < 5) und im Fünfer-Bereich (= 5,0) die jeweilige Anzahl der entsprechenden Noten zurückgibt.

Das Histogramm wird als Abbildung (Map) von Note auf Anzahl zurückgegeben.

Beispiel: Die oben genannte Datei gibt folgendes Histogramm zurück:

```
Note 1,x:  2
Note 2,x:  2
Note 3,x:  0
Note 4,x:  1
Note 5,x:  1
```

Ergänzen Sie außerdem Ihr Hauptprogramm, um diese Methode zu testen. (7 P)

Aufgabe 3: Innere Klassen (11 Punkte)

Die Lösung der Aufgabe muss im ZIP-Archiv im Ordner A3 gespeichert werden.

Betrachten Sie die Klassen im Paket **de.hrw.java.exam.innerclasses**. Die Klasse **OrderItem** repräsentiert die Position in einem Auftrag und besteht aus:

- Einer Beschreibung (**description**)
- Stückpreis (**pricePerItem**)
- Anzahl (**quantity**)

Der **Wert einer Position** entspricht dem Produkt aus Stückpreis und Anzahl (`pricePerItem * quantity`).

Das Hauptprogramm in dem Paket legt eine Liste von Positionen an und gibt diese aus.

a) Schreiben Sie das Hauptprogramm so um, dass die Positionsliste (**orderItems**) mit Hilfe einer inneren statischen Klasse sortiert wird. (4 P)

b) Ergänzen Sie das Hauptprogramm so, dass eine zweite Positionsliste (**orderItems2**) mit Hilfe einer anonymen Klasse sortiert wird. (4 P)

c) Ergänzen Sie das Hauptprogramm so, dass eine dritte Positionsliste (**orderItems3**) unter Verwendung eines Lambda-Ausdrucks sortiert wird. (3 P)

Aufgabe 4: Streams (18 Punkte)

Die Lösung der Aufgabe muss im ZIP-Archiv im Ordner A4 gespeichert werden.

Die beigefügte Klasse **Hamster** im Paket **de.hrw.java.exam.streams.hamster** modelliert einen Hamster.

Hamster verfügen bekanntlich über die folgenden Eigenschaften:

- **Name**
- **Geschlecht**: m/w/d
- **Alter** in Monaten
- **Typ**: Goldhamster (syrischer Hamster), Zwerghamster, chinesischer Hamster
- **Temperament**: sanguin, cholerisch, melancholisch, phlegmatisch
- **Politische Orientierung**: links, rechts, liberal

Vervollständigen Sie die Klasse **HamsterHelper** unter der ausschließlichen Verwendung von Streams. Es gelten folgende Auflagen

- Verwendung von Schleifen und if/switch sind verboten
- Es dürfen keine lokalen Variablen definiert werden. Alle Aufgaben lassen sich mit einer Zeile Code lösen.

Folgende Methoden müssen vervollständigt werden (siehe auch JavaDoc in der Klasse HamsterHelper):

- **getNumberOfLiberalSyrianHamsters** erhält eine Liste von Hamstern und gibt die Anzahl der liberal eingestellten Goldhamster zurück. (3 P)
- **anyCholericRightHamsterAboveAge** erhält ein Array von Hamstern und einen Altersschwellenwert. Die Methode gibt zurück, ob es einen politisch rechts eingestellten, cholerischen Hamster gibt, der mindestens so alt ist wie der übergebene Altersschwellenwert. (3 P)
- **getAverageDiverseHamsterAge** erhält eine Menge von Hamstern und gibt das Durchschnittsalter der darin enthaltenen diversen Hamster zurück. (3 P)
- **getMelancholicFemaleHamsters** erhält eine Menge von Hamstern und gibt die darin enthaltenen melancholischen (weiblichen) Hamsterinnen als Liste zurück. (3 P)
- **printHamstersOnly** erhält eine Liste von Objekten (Typ Object) und gibt alle Hamster dieser Liste auf der Konsole aus. Nicht-Hamster werden nicht ausgegeben. (3 P)

Implementieren Sie ein Hauptprogramm, das alle Methoden testet. (3 P)