HOCHSCHULE RUHR WEST
UNIVERSITY OF APPLIED SCIENCES
INSTITUT INFORMATIK

12.03.2024

Klausur

Programmierung 2

Wintersemester 2023/24

Studiengänge: AI, MTI

Prüfer: Prof. Dr. Malte Weiß

Persönliche Angaben

Nachname	Vorname	Matrikelnummer

Bewertung

Frage	Punkte	Erreicht
1	16	
2	20	
3	9	
4	11	
5	6	

Frage	Punkte	Erreicht
6	9	
7	5	
8	16	
9	8	
Gesamt:	100	

Ablauf der Prüfung

- **Zeit für die Lösung dieser Klausur**: 120 Minuten.
- **Erlaubte Hilfsmittel**: Referenz am Ende der Aufgabenblätter (eine Seite). Handgeschriebener, ein- oder beidseitig beschriebener Notizzettel im DIN-A4-Format.
- **Nichterlaubte Hilfsmittel**: Mobiltelefon, Kommunikation (E-Mail, Web, SMS, etc.). Die Verwendung eines nicht erlaubten Hilfsmittels wird als Täuschungsversuch gewertet.
- **Stift und Papier**: Schreiben Sie ausschließlich auf dem Papier der Aufgabenblätter. Sollten die Zwischenräume nicht ausreichen, können Sie zusätzliches Papier von der Aufsicht erhalten. Beschriften Sie dieses sofort mit ihrem Namen und Ihrer Matrikelnummer. Eigenes Papier ist nicht zulässig. Schreiben Sie ausschließlich mit dokumentenechten schwarzen oder blauen Stiften.
- **Vollständigkeit des Aufgabenblatts**: Die Klausur besteht aus den Aufgabenseiten 5 – 19 und einer Funktionsreferenz auf Seite R1. Die Aufgabenblätter sind doppelseitig bedruckt. Prüfen Sie, ob alle Blätter vorhanden sind.
- **Fragen**: Melden Sie sich, wenn Sie eine Frage haben. Die Aufsicht kommt zu Ihnen, verlassen Sie nicht unaufgefordert Ihren Platz.
- **Zwischenschritte**: Geben Sie Zwischenschritte an. So können Sie selbst bei falschem Endergebnis einen Teil der Punkte erreichen.

Grundlagen

1. (a) Erklären Sie, was eine überladene Methode in Java ist. Nennen Sie ein Beispiel. (4)

(b) Die Programmiersprache *Java* gilt als sicherer als *C*. Erläutern Sie eine Eigenschaft der Sprache, die dies bestätigt. (4)

(c) Welche Funktion hat das Schlüsselwort `var`? Nennen Sie ein Beispiel. (4)

(d) Was ist eine `anonyme Klasse` in Java. Nennen Sie ein Beispiel. (4)

Objektorientierte Programmierung

2. (a) Schreiben Sie eine Klasse `Apartment`, die eine Wohnung modelliert. Eine Wohnung besitzt die folgenden Parameter: (6)

   - Name der Straße
   - Hausnummer
   - Höhe der Miete (in Euro)

   Implementieren Sie die Klasse mit einem einzigen Konstruktor, der alle Attribute setzt, mit Gettern und mit einer toString-Methode, die alle Attribute ausgibt. Andere Konstruktoren oder Setter sollen *nicht* implementiert werden.

(b) Leiten Sie eine Klasse `HolidayApartment` von `Apartment` ab. Die abgeleitete Klasse modelliert eine Ferienwohnung, die zusätzlich zur Oberklasse noch ein Attribut für die maximale Mietdauer in Tagen enthält. (5)

Implementieren Sie auch hier einen Konstruktor, der alle Attribute setzt, Getter und eine toString-Methode, die alle Attribute ausgibt.

(c) Schreiben Sie eine Klasse `ApartmentManager`. Sie ermöglicht es, eine Liste von Wohnungen zu verwalten und bietet folgende Methoden: (9)

- Das Hinzufügen einer Wohnung.
- Das Löschen einer Wohnung.
- Die Ausgabe alle Wohnungen.
- Die Rückgabe der Durchschnittsmiete. Ist die Liste aller Wohnungen jedoch leer, soll die Methode eine IllegalStateException werfen.

Tipp: Referenz am Ende der Klausur.

3. Betrachten Sie die folgende Klassenhierarchie:

```
public class Top {
    private int value;

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

    public int getValue() {
        return value;
    }

    @Override
    public String toString() {
        return "value = " + getValue();
    }
}

public class Middle extends Top {
    private int factor;

    public Middle(int value, int factor) {
        super(value);
        this.factor = factor;
    }

    public int getFactor() {
        return factor;
    }

    public int getValue() {
        return getFactor() * super.getValue();
    }
}

public class Bottom extends Middle {
    public Bottom(int value, int factor) {
        super(value, factor);
    }

    public int getFactor() {
        return 2 * super.getFactor();
    }
}
```

(Fortsetzung auf der nächsten Seite)

Welche Ausgaben produzieren die folgenden Codeausschnitte?

(a) (3)

```
Top obj = new Top(2);
System.out.println(obj);
```

Ausgabe:

(b) (3)

```
Top obj = new Middle(2, 4);
System.out.println(obj);
```

Ausgabe:

(c) (3)

```
Top obj = new Bottom(2, 4);
System.out.println(obj.getValue());   // getValue-Aufruf beachten!
```

Ausgabe:

Enums

4. (a) Entwickeln Sie einen Aufzählungstyp `Planet` für Planeten. (7)
Ein Planet besitzt einen Namen und seine Distanz zur Sonne.

ID	Name	Entfernung zur Sonne (in Millionen km)
MERCURY	Merkur	57,9
VENUS	Venus	108,2
EARTH	Erde	149,6
MARS	Mars	228

Implementieren Sie das Enum für die vier genannten Planeten in Java. Für jede Ausprägung sollen der Name und die Distanz zur Sonne zurückgegeben werden können (siehe Tabelle).

Bitte lösen Sie die Aufgabe auf der nächsten Seite.

(b) Ergänzen Sie das folgende Hauptprogramm, so dass die erzeugte Liste der Planeten (`planets`) nach der jeweiligen Distanz zur Sonne sortiert wird. Nutzen Sie dafür entweder eine anonyme Klasse oder einen Lambda-Ausdruck. (4)

```
public static void main(String[] args) {

    List<Planet> planets = new ArrayList<>();
    planets.add(Planet.MERCURY);
    planets.add(Planet.MARS);
    planets.add(Planet.JUPITER);
```

```

}
```

Arrays

5. Schreiben Sie eine Methode `countElements`. Sie enthält ein 2D-Array von `int`-Werten und gibt die Gesamtanzahl der darin enthaltenen Elemente zurück. (6)

Datenstrukturen

6. Schreiben Sie eine Methode `createGradeHistogram`. Sie erhält ein Array von Noten als `double`-Werte und gibt ein Histogramm (`Map`) zurück, das zählt, wie oft welche Note im Array vorkommt. (9)

   Beispiel: Das Array {`1.1, 3.5, 2.3, 2.3, 1.1, 2.3`} ergibt das Histogramm `1.1` $\rightarrow$ `2, 2.3` $\rightarrow$ `3, 3.5` $\rightarrow$ `1`.

   Tipp: Referenz am Ende der Klausur.

Exception

7. Erstellen Sie eine einfache Ausnahme-Klasse `UnknownZipCodeException`. Es handelt sich um eine ungeprüfte Ausnahme, die geworfen werden kann, wenn eine ungültige Postleitzahl verwendet wird. (5)

   Die Postleitzahl sollte als Attribut im Exception-Objekt hinterlegt werden.

Streams

8. Gegeben sind die folgenden Typen. Der Auszählungstyp `Gender` repräsentiert ein Geschlecht:

```
public enum Gender {
    MALE, FEMALE, DIVERSE
}
```

Ein Spieler wird durch die Klasse `Player` repräsentiert:

```
public class Player {
    private String name;      // Name
    private Gender gender;    // Geschlecht
    private long score;       // Punktzahl

    public Player(String name, Gender gender, long score) {
        this.name = name;
        this.gender = gender;
        this.score = score;
    }

    public String getName() {
        return name;
    }
    public Gender getGender() {
        return gender;
    }
    public long getScore() {
        return score;
    }
}
```

Ein Team enthält eine Liste von Spielern und wird durch die Klasse `Team` repräsentiert:

```
public class Team {
    private List<Player> players = new ArrayList<>();

    public Team(List<Player> players) {
        this.players = new ArrayList<>(players);
    }

    public List<Player> getPlayers() {
        return players;
    }
}
```

Implementieren Sie nun die folgenden Methoden mit Hilfe von **Streams**.

Ihre Lösung darf **keine Schleifen** und **keine weiteren lokalen Variablen** beinhalten.

(a) Die Methode `getTotalPlayers` erhält eine Liste von Teams und gibt die Gesamtzahl aller Spieler zurück. (3)

```
public int getTotalPlayers(List<Team> teams) {
```

```

```

```
}
```

(b) Die Methode `getAnyPlayerWithName` erhält ein Team und einen Namen und gibt zurück, ob es ein Teammitglied mit dem angegebenen Namen gibt. (3)

```
public boolean getAnyPlayerWithName(Team team, String name) {
```

```

```

```
}
```

(c) `getTeamsWithMinimumTeamScore` erhält eine Liste von Teams und eine Mindestpunktzahl. Sie gibt eine Liste der Teams zurück, deren aufsummierte Spieler-Punktzahlen die Mindestpunktzahl überschreiten. (5)

```
public List<Team> getTeamsWithMinimumTeamScore(
    List<Team> teams, int minimumTeamScore) {
```

```

```

```
}
```

(d) `getPlayersWithGender` erhält eine Liste von Teams und ein Geschlecht. Die Methode gibt alle Spieler zurück, deren Geschlecht dem übergebenem Geschlecht entspricht. Tipp: `flatMap` verwenden. (5)

```
public long getPlayersWithGender(List<Team> teams, Gender gender) {
```

```

```

```
}
```

Generics

9. Entwickeln Sie einen generischen Datentyp `Box` mit einem Platzhalter `T`. Eine Box besitzt ein Inhaltsattribut vom Typ `T` und eine Liste von beliebig vielen Unterboxen desselben Typs `T`. (8)

   Die Klasse soll darüber hinaus folgendes anbieten:

   - Einen Konstruktor mit einem Inhaltsparameter, der den Inhalt Box setzt und eine leere Liste von Unterboxen anlegt.
   - Eine Methode, um eine Unterbox hinzufügen.
   - Eine Ausgabe-Methode, die den Inhalt der Box und anschließend alle Unterboxen ausgibt.

Referenz

Listen

List ist ein **Interface** für eine geordnete Sammlung von Objekten eines Typs. Elemente dürfen mehrfach vorkommen. Standard-Implementierungen von Listen: **ArrayList, LinkedList**

Wichtigste Methoden des List-Interfaces:

- **add**(element) fügt ein Element einer Liste hinzu.
- **contains**(element) prüft, ob ein Element in der Liste vorkommt.
- **remove**(element) entfernt ein Element aus einer Liste.
- **size**() liefert die Anzahl der Elemente der Liste zurück.

Maps

Map ist ein **Interface** für eine Abbildung von Schlüsseln auf Werte. Standard-Implementierung von Maps: **HashMap**

Wichtigste Methoden des Map-Interfaces:

- **put**(key, value) fügt einen Wert für einen Schlüssel hinzu
- **get**(key) gibt einen Wert für einen Schlüssel zurück. Gibt es den Wert nicht, wird null zurückgegeben.
- **containsKey**(key) gibt genau dann true zurück, wenn es einen Wert für den Schlüssel gibt.

Typische Stream-Operationen

Streams erzeugen	Zwischenoperationen	Terminale Operationen
Collection / List stream() parallelStream() **Stream** **IntStream** **LongStream** **DoubleStream** static generate() static of(..) static empty() static iterate(..) static concat(..) static builder() **IntStream** **LongStream** static range(..) static rangeClosed(..) **Arrays** static stream(..)	**BaseStream** sequential() parallel() unordered() onClose(..) **Stream** filter(..) map(..) mapToInt(..) mapToLong(..) mapToDouble(..) flatMap(…) distinct() sorted(..) peek(..) limit(..) skip(..) **IntStream** mapToObj(..) asLongStream() asDoubleStream() **LongStream** mapToObj(..) asDoubleStream() **DoubleStream** mapToObj(..)	**BaseStream** iterator() **Stream** forEach(..) toArray(..) reduce(..) collect(..) min(..) max(..) count() anyMatch(..) allMatch(..) noneMatch(..) findFirst() findAny() **IntStream** **LongStream** **DoubleStream** sum() average()