NACHNAME:	**SEMESTER:** ☐ **M5** ☐ **M6** ☐ M3 ☐ M4 ☐ M7
VORNAME:	**VERTIEFUNG:** ☐ **FV** ☐ **IM**

VERWENDETE KLASSEN:

- Als **Anlage** erhalten Sie einen Ausdruck des vorab bekannt gemachten Quelltextes von verschiedenen Klassen.
- Die direkt in diesen Aufgabenblättern gezeigten Code-Auszüge sind sinnvoll ergänzt zu denken, insbesondere durch geeignete `import`-Anweisungen.

UNBEDINGT BEACHTEN:

- **Bevor** Sie mit der Bearbeitung beginnen, müssen die Angaben zur Person auf dieser Seite **vollständig** ausgefüllt sein.
- Es sind **keinerlei Hilfsmittel** zugelassen, auch kein zusätzliches Konzeptpapier.
- Die zusammengehefteten Blätter dürfen **nicht getrennt** werden.

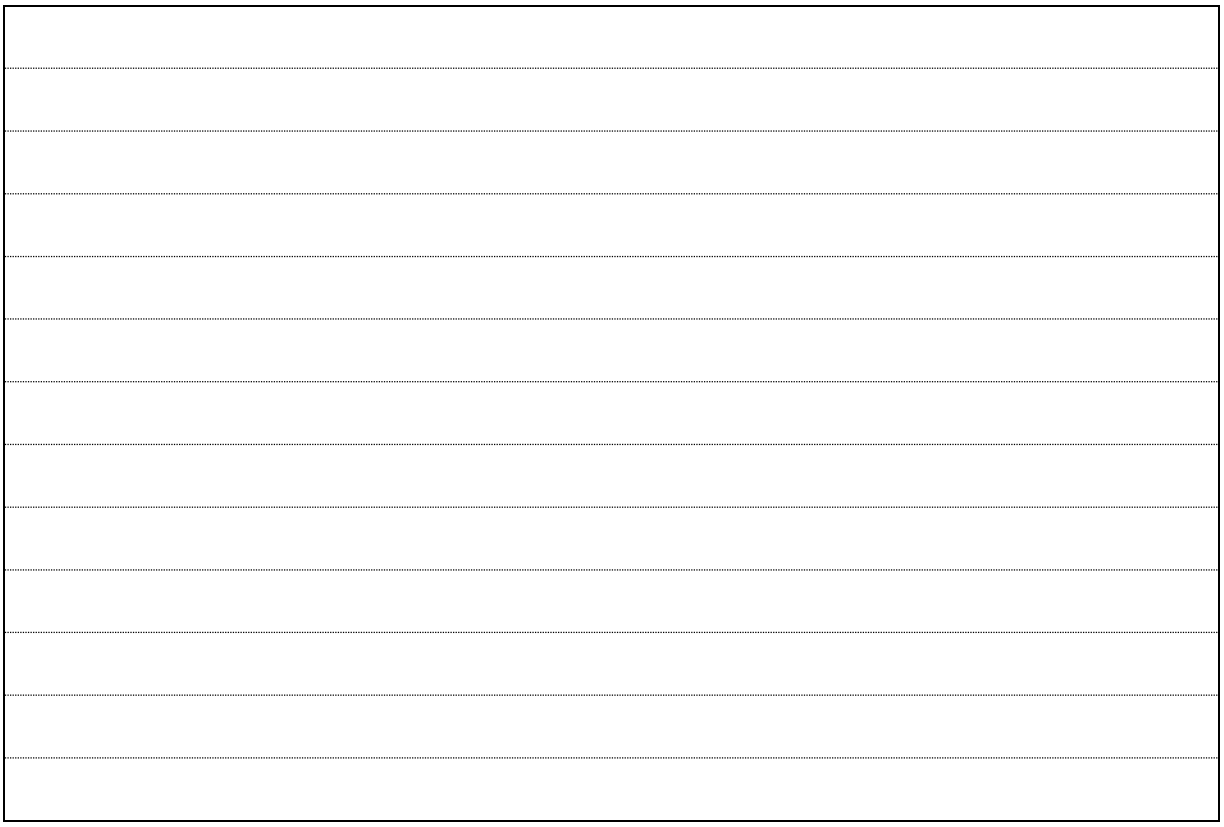
GENERELLE VORGABEN:

- Es sind keinerlei Kommentare verlangt, weder `javadoc`-Kommentare noch andere.
- Methoden und Klassen müssen vollständig, auch mit Annotationen, angegeben werden. Etwaige `import`–Anweisungen werden aber nicht verlangt.
- Die einzelnen Zeichen – auch Groß- oder Klein-Schreibung – müssen klar erkennbar sein. Abkürzungen sind nicht erlaubt.
- Programmier-Richtlinien (insbesondere Checkstyle) sind zu beachten. Bei Testklassen dürfen aber *magic numbers* verwendet werden.

Aufgabe 1: (6 Punkte)

Innerhalb einer Hilfsklasse (utility class) soll eine Möglichkeit geschaffen werden, diejenigen der acht Planeten abzufragen, bei denen eine vorgegebene `Eigenschaft` den angegebenen Mindestwert besitzt. Implementieren Sie die zugehörige Methode:

```
public static Set<Planet> planetenMitMindestens(
                                        Eigenschaft e, double wert) {
```



```
}
```

Aufgabe 2: (10 Punkte)

Nach dem dritten Keplerschen Gesetz verhalten sich die Quadrate der Umlaufzeiten zweier Planeten wie die dritten Potenzen der großen Bahnhalbachsen. Kepler verwendete für die Bahnachsen die mittlere Entfernung zur Sonne. Der Quotient

$$\boldsymbol{C} := \frac{\boldsymbol{T^2}}{\boldsymbol{a^3}} \qquad T = \text{Umlaufzeit},\ a = \text{Abstand zur Sonne}$$

sollte also näherungsweise für alle Planeten gleich sein. Dies soll durch einen neuen JUnit-Test in einer vorhandenen Testklasse überprüft werden. Die Berechnung des obigen Quotienten für einen vorgegebenen Planeten soll in einer Hilfsmethode der Testklasse geschehen.

a) Implementieren Sie die Hilfsmethode. Umlaufzeit und Abstand zur Sonne sollen dabei in den bei den Planeten gespeicherten Einheiten (Tage bzw. Millionen km) verwendet werden.

```
private static double keplerKonstanteFuer(Planet p) {
```

```
}
```

b) Schreiben Sie damit die JUnit-Test-Methode. Als Referenzwert (für die sogenannte Kepler-Konstante c) soll die Erde verwendet werden. Die Werte der anderen Planeten dürfen hiervon maximal um 6/1000 abweichen (bezogen auf die von der Hilfsmethode verwendeten Einheiten).

Aufgabe 3: (18 Punkte)

Ausnahmsweise dürfen die Planeten in dieser Aufgabe (und **nur** hier) durch ihre Anfangsbuchstaben abgekürzt werden.

a) Nach der Sortierung mit `Splitter.untenOben(FALLBESCHLEUNIGUNG, 99/10)` wird eine Liste von Planeten mit `[VENUS, MARS, ERDE, JUPITER]` angezeigt. Wie könnte diese Liste vor der Sortierung ausgesehen haben? Geben Sie alle Möglichkeiten an.

Wie könnte die ursprüngliche Liste ausgesehen haben, wenn sich durch die Sortierung mit dem angegebenen Splitter `[MARS, ERDE, JUPITER, SATURN]` ergibt? Geben Sie wieder alle Möglichkeiten an.

b) Der folgende Code werde mit dem Debugger von Eclipse durchlaufen. An mehreren Stellen werde der Wert der Variablen `feld` überprüft. Tragen Sie in den Kästen den jeweils aktuellen Wert ein, **genau** wie er vom Debugger angezeigt wird.

```
Universell mond = new Universell(ROTATIONS_PERIODE,
                                                 TAG * (JAHR / 12));
Himmelskoerper[] merk = {mond, NEPTUN, MARS, SATURN};
Himmelskoerper[] feld = merk.clone();
Splitter v1 = untenOben(ROTATIONS_PERIODE, (TAG * JAHR) / 12);
Arrays.sort(feld, v1);
```

```
feld = merk.clone();
Splitter s1 = untenOben(ROTATIONS_PERIODE, TAG);
Comparator<Himmelskoerper> v2 = new SekundaerVergleicher<>(v1, s1);
Arrays.sort(feld, v2);
mond.put(ROTATIONS_PERIODE, 0.);
```

```
Arrays.sort(feld, v2);
```

```
feld = merk.clone();
s1 = v1.fuer(mond);
Arrays.sort(feld, v2);
```

```
feld = merk.clone();
v1 = s1.fuer(SATURN);
Arrays.sort(feld, v1);
```

```
feld = merk.clone();
Arrays.sort(feld, v2);
```

Aufgabe 4: (30 Punkte)

Es soll ein neuer `Comparator` für `Himmelskoerper` entstehen, der einen (im Konstruktor) vorgegebenen `Himmelskoerper`, den „Boss“, an die erste Stelle setzt und den Rest, das „Fußvolk“ als gleichwertig betrachtet. Der „Boss“ ist also kleiner als alle von ihm (semantisch) verschiedenen `Himmelskoerper`. Die Klasse soll `BossVergleicher` heißen.

a) Schreiben Sie zunächst einen zugehörigen JUnit-Test für den `BossVergleicher` mit dem „Boss“ `ERDE`. Getestet werden soll die Liste aller Planeten in vielen, zufälligen Sortierungen.

```
@Test
public void testeBossErdeSortierung() {
```

}

b) Schreiben Sie nun eine entsprechende Klasse `BossVergleicher`. Serialisierbarkeit und den semantischen Vergleich ihrer Objekte braucht die Klasse nicht zu unterstützen. Geben Sie die gesamte Klasse an:

Aufgabe 5: (8 Punkte)

Die Aufgabe bezieht sich auf den `BossVergleicher` aus der vorigen Aufgabe, kann aber unabhängig bearbeitet werden. Die Ergebnisse sind nämlich so anzugeben, wie sie bei korrekter Implementierung anfallen würden.

Benötigt wird von **Aufgabe 4** damit nur die Vorgabe, die hier nochmals wiederholt sei:

Es soll ein neuer Comparator für Himmelskoerper entstehen, der einen (im Konstruktor) vorgegebenen Himmelskoerper, den „Boss“, an die erste Stelle setzt und den Rest, das „Fußvolk“ als gleichwertig betrachtet. Der „Boss“ ist also kleiner als alle von ihm (semantisch) verschiedenen Himmelskoerper. Die Klasse soll BossVergleicher heißen.

An geeigneter Stelle werden eine Variable `fuszball` und einige Comparatoren definiert:

```
Himmelskoerper fuszball = new Universell(MASSE, 0.43);
Comparator<Himmelskoerper> b1 = new BossVergleicher(fuszball);
Comparator<Himmelskoerper> b2 = new BossVergleicher(VENUS);
Comparator<Himmelskoerper> s = untenOben(DURCHMESSER, 0.);
```

Sie sollen zur Sortierung folgender Liste benutzt werden:

```
List<Planet> liste = Arrays.asList(ERDE, MARS, VENUS, fuszball);
```

Tragen Sie in die Kästen jeweils `liste.toString()` nach der Sortierung **dieser (ursprünglichen)** Liste mit dem angegebenen `Comparator` ein. Wird bei der Sortierung eine `Exception` geworfen, geben Sie stattdessen den genauen Typ dieser `Exception` an.

Sortieren Sie mit `new SekundaerVergleicher<>(s, b1)`:

Sortieren Sie stattdessen mit `new SekundaerVergleicher<>(b1, s)`:

Sortieren Sie stattdessen mit `new SekundaerVergleicher<>(b2, s)`:

Sortieren Sie stattdessen mit `new SekundaerVergleicher<>(b2, b1)`?

Sortieren Sie stattdessen mit `new SekundaerVergleicher<>(b1, b2)`?

Aufgabe 6: (18 Punkte)

Vervollständigen Sie die nachstehenden Aussagen über Klassen und Methoden der Anlage. Die letzte Teilaufgabe bezieht sich zusätzlich auf die Klasse `BossVergleicher` aus **Aufgabe 4**.

a) Beim Aufruf `mond.put(ROTATIONS_PERIODE, 0.)` findet _____ statt. Der Aufruf `mond.put(ROTATIONS_PERIODE, 0)` führt zu _____ _____, weil ein _____ nicht in ein _____ eingewickelt werden kann.

b) Die Klasse `Splitter` sollte unbedingt _____ sein. Vor allem müsste dazu die Methode _____ entfernt werden oder als _____ deklariert werden. Dadurch könnte die Variable _____ wie empfohlen als _____ deklariert werden. Schließlich sollte die Klasse noch _____ sein, damit auch in _____ _____ Klassen kein Unfug angestellt werden kann.

c) Ein `Planet` ist ein `Himmelskoerper`. Nach dem _____ _____ können Planeten daher mit einem `Comparator<`_____`>` sortiert werden. Dies führt bisweilen zu dem Fehlschluss, dass einem `Comparator<`_____`>` ein `Comparator<`_____`>` zugewiesen werden kann.

d) Ein veränderlicher `Himmelskoerper` mit einer Umlaufzeit von 100 Tagen entsteht etwa durch `Himmelskoerper h =` _____ _____. Durch den Methodenaufruf `h.` _____ _____ kann die Umlaufzeit nachträglich auf 1 Jahr gesetzt werden.

e) Die Klasse `BossVergleicher` sollte besser _____ sein wie die Klasse _____ der Anlage. Sie könnte dann zur Sortierung beliebiger Objekte verwendet werden. Beispielsweise könnte mittels `Comparator<` _____ `> = new BossVergleicher<>(` _____ `)` ein `BossVergleicher` für die Klasse `String` definiert werden.