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: (9 Punkte)

Es soll eine Möglichkeit geschaffen werden, für eine vorgegebene `Eigenschaft` den durchschnittlichen Wert der acht Planeten abzufragen, also beispielsweise die mittlere Temperatur auf den Planeten oder ihre durchschnittliche Masse.

Schreiben Sie dazu eine Hilfsklasse (utility class) `Planeten` mit einer Methode `mittelwert(Eigenschaft e)`. Geben Sie die gesamte Klasse an:

Aufgabe 2: (9 Punkte)

Nach dem Newtonschen Gravitationsgesetz gilt für die Fallbeschleunigung *g* auf der Oberfläche eines Planeten näherungsweise

$$\boldsymbol{g = \frac{G \cdot m}{r^2}} \qquad \text{mit } G \approx 6{,}6738 \cdot 10^{-11} \frac{m^3}{kg \cdot s^2} \text{ (Gravitationskonstante)}$$

m ist die Masse des Planeten und *r* sein Radius.

a) Erweitern Sie die Klasse `Konstanten` der Anlage um die Gravitationskonstante. Geben Sie nur die neue Zeile an:

b) Schreiben Sie eine JUnit-Test-Methode, welche überprüft, ob die in den 8 Planeten abgelegten Daten (Fallbeschleunigung in m/s^2, Masse in kg und Durchmesser in km) der obigen Formel entsprechen. Der gespeicherte Wert der Fallbeschleunigung darf um höchstens 5% vom berechneten abweichen.

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

}

Aufgabe 3: (16 Punkte)

a) Eine Liste von Planeten wird wie folgt sortiert:

```
Comparator<Himmelskoerper> vergleicher =
                    Splitter.untenOben(TEMPERATUR, ERDE);
List<Planet> liste = Arrays.asList(VENUS, ERDE, SATURN,
                                   MERKUR, ERDE, JUPITER);
Collections.sort(liste, vergleicher);
System.out.println(liste);
```

Geben Sie die genaue Ausgabe (also insbesondere ohne Abkürzungen) an.

Bestimmen Sie die Ausgabe, wenn die oben markierte Initialisierung ersetzt wird durch `Splitter.obenUnten(TEMPERATUR, ERDE);`.

Wie sieht die Ausgabe aus, wenn `Splitter.untenOben(TEMPERATUR, 0);` zur Initialisierung von `vergleicher` verwendet wird?

Verwenden Sie abschließend `Splitter.obenUnten(TEMPERATUR, 0);` zur Initialisierung.

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.

```
Himmelskoerper[] merk = { VENUS, DIE_EINZIGE, MARS, VENUS };
Himmelskoerper[] feld = merk.clone();
Splitter v1 = Splitter.untenOben(MASSE, 0);
Arrays.sort(feld, v1);
```

```
feld = merk.clone();
Splitter v2 = v1.fuer(VENUS);
Arrays.sort(feld, v2);
```

```
feld = merk.clone();
Comparator<Himmelskoerper> v3 = reverseOrder(v2);
Arrays.sort(feld, v3);
```

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

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

```
feld = merk.clone();
Arrays.sort(feld, v3);
```

Aufgabe 4: (31 Punkte)

Es soll ein neuer `Comparator` für `Himmelskoerper` entstehen, der anhand einer (im Konstruktor) vorgegebenen `Eigenschaft` vergleicht. Beispielsweise muss beim Vergleich zweier Himmelskörper anhand der Durchschnittstemperatur der kältere als kleiner angesehen werden. Die Klasse soll `EigenschaftVergleicher` heißen.

a) Schreiben Sie zunächst einen zugehörigen JUnit-Test. Dieser Test soll eine Liste aller Planeten erstellen, nach jeder der vorhandenen Eigenschaften sortieren und jeweils überprüfen, ob in der sortierten Liste der Wert der Eigenschaft tatsächlich stets wächst.

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

}

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

Aufgabe 5: (9 Punkte)

Die Aufgabe bezieht sich auf den `EigenschaftVergleicher` 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 anhand einer (im Konstruktor) vorgegebenen Eigenschaft vergleicht. Beispielsweise muss beim Vergleich zweier Himmelskörper anhand der Durchschnitts-temperatur der kältere als kleiner angesehen werden. Die Klasse soll EigenschaftVergleicher heißen.

An geeigneter Stelle werden einige Comparatoren definiert:

```
Comparator<Himmelskoerper> s1 = obenUnten(ROTATIONS_PERIODE, NEPTUN);
Comparator<Himmelskoerper> s2 = new EigenschaftVergleicher(MASSE);
Comparator<Himmelskoerper> v1 = untenOben(ROTATIONS_PERIODE, TAG);
Comparator<Himmelskoerper> v2 = new SekundaerVergleicher<>(v1, s1);
Comparator<Himmelskoerper> v3 = new SekundaerVergleicher<>(v2, s2);
```

Sie sollen zur Sortierung einer Liste aus allen Planeten benutzt werden:

```
List<Planet> liste = Arrays.asList(Planet.values());
```

Geben Sie `liste.toString()` nach der Sortierung mit `v1` an:

Was ergibt sich bei Sortierung der **ursprünglichen** Liste mit `v2`?

Sortieren Sie die **ursprüngliche** Liste nun mit `v3`:

Aufgabe 6: (16 Punkte)

Vervollständigen Sie die folgenden Aussagen über Klassen und Methoden der Anlage.

a) Autoboxing kommt in der Klasse _____ vor.
Im Statement _____
wird ein _____ in ein _____ eingewickelt.
Unboxing erfolgt etwa in den Anweisungen _____
und _____ .

b) Ein schlimmer Fehler in der Klasse _____, nämlich, dass die
Klassenvariable _____ nicht _____ ist,
kann bei einigen Planeten, etwa _____ und _____<
zu falschen Werten der Membervariablen _____ führen.

c) Ein `Himmelskoerper` _____ keine `Map`.
Die Klasse _____ verletzt daher in besonders drastischer Weise
die wichtige Design-Regel „_____
_____“.

d) Keine Klassenvariablen sind _____
und _____, obwohl ihre Schreibweise dies anzeigt.
Die Verwendung kann deshalb nur über ein _____ erfolgen.

e) Ein veränderlicher `Himmelskoerper` mit unbekannter Umlaufzeit entsteht etwas durch `Himmelskoerper h =` _____.
Beim Aufruf der Methode `umlaufZeit` für diesen `Himmelskoerper h` wird dann eine _____ geworfen. Richtig wäre eine _____.

f) Zur erweiterbaren Definition mathematischer Funktionen wurde in der Vorlesung das _____-Muster verwendet. Eine Entsprechung findet sich hier in der Anlage. Dem _____ `Funktion` entspricht _____, einer konkreten Funktion wie `Sinus` entspricht hier _____ und dem verallgemeinerten _____ `MittelbareFunktion` entspricht _____.