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

Aufgabe 1: (14 Punkte)

WICHTIG: Achten Sie in **allen** Aufgabenteilen darauf, dass keine illegalen Objekte entstehen. `javadoc`-Kommentare sind hingegen **nicht** verlangt.

Die Klasse `Monom` beschreibt eine Funktion der Form

$$f(x) = a \cdot x^n \qquad (a \in \mathbb{R}\backslash\{0\},\ n \in \mathbb{N}_0)$$

Eine Implementierung beginne so:

```
public final class Monom {
    private final double mKoeffizient;
    private final int mHochzahl;
```

a) Ein elementarer Test prüft einen der Konstruktoren von `Monom`:

```
@Test
public void testeParabel() {
    final int grad = 2;
    Monom p = new Monom(1., grad);
    assertEquals(grad, p.grad());
}
```

Geben Sie eine mögliche Implementierung des im Test verwendeten Konstruktors an. Andere im Test benötigte Methoden werden als bereits vorhanden angenommen.

```
public Monom(double a, int n) {
    if (a == 0.) throw new IllegalArgumentException();
    if (n < 0) throw new IllegalArgumentException();
    mKoeffizient = a;
    mHochzahl = n;
}
```

b) Ein anderer Test prüft die Methode `ableitung`:

```
@Test
public void testeAbleitungsGrad() {
    final int startGrad = 86;
    Monom m = new Monom(1., startGrad);
    assertEquals(startGrad, m.grad());
    for (int grad = startGrad; grad > 0; grad--) {
        m = m.ableitung();
        assertEquals(grad - 1, m.grad());
    }
}
```

Wie könnte die Implementierung der Methode `ableitung` aussehen?

```
public Monom ableitung() {
    if (mHochzahl == 0)
            throw new UnsupportedOperationException();
    return new Monom(mKoeffizient * mHochzahl,
                                    mHochzahl - 1);
}
// schlechter, aber akzeptabel: IllegalStateException
```

c) Sinnvollerweise sollte nicht nur der Grad der Ableitung getestet werden, sondern die ganze Funktion, etwa so:

```
@Test
public void testeAbleitungsfunktion() {
    final int grad = 5;
    Monom p = new Monom(1., grad);
    Monom ps = new Monom(grad, grad - 1);
    assertEquals(ps, p.ableitung());
}
```

Was muss in der Klasse `Monom` getan sein, damit dieser Test funktioniert?

```
Sie Methode equals muss überschrieben sein,
und zwar entsprechend des allgemeinen Vertrags.
```

Aufgabe 2: (10 Punkte)

Die Aufgabe bezieht sich auf die in **Aufgabe 1** definierte Klasse `Monom`:

a) In der Klasse `Monom` werde versucht, die Deklaration von `mHochzahl` um eine explizite Initialisierung zu ergänzen:

```
public final class Monom {
    private final double mKoeffizient;
    private final int mHochzahl = 0;
```

Was passiert dann im Konstruktor `Monom(double, int)` und weshalb?

Da mHochzahl final ist, kann der Wert nicht mehr
Verändert werden.
Das Setzen der Hochzahl im Konstruktor Monom(double,int)
führt zu einem Compile-Fehler.

b) In einem JUnit-Test werde versucht, das `Monom m` mittels

```
System.out.println("" + m.mKoeffizient
                      + "x^" + m.mHochzahl);
```

auszudrucken. Was passiert und weshalb?

Da die Daten private sind, sind sie nur innerhalb der
Klasse Monom sichtbar.
In der Testklasse kann auf die Daten nicht zugegriffen
werden. Ergebnis: Compile-Fehler.

c) In einer Methode der Klasse Monom werde versucht, auf die Daten eines anderen Objekts der Klasse zuzugreifen:

```
public Monom mal(Monom anderes) {
    double a = mKoeffizient * anderes.mKoeffizient;
    int n = mHochzahl * anderes.mHochzahl;
    return new Monom(a, n);
}
```

Was passiert und weshalb?

Gar nichts. Das ist erlaubt und wird oft benötigt
(insbesondere meist in der Methode equals).
Das Ergebnis ist aber falsch. Die Hochzahlen müssen
addiert, nicht multipliziert werden.

Aufgabe 3: (16 Punkte)

Eine Klasse `StringTango` enthalte folgende Methode:

```
public static List<String> tuWasMit(List<String> texte) {
    List<String> erg = new ArrayList<String>();
    for (String text : texte) {
        if (!text.equals("Cha")) erg.add(text);
    }
    if (texte.size() - erg.size() > 2) {
        texte = erg;
    } else if (erg.size() < 2) {
        erg = texte;
    }
    texte.add("tanzen");
    return erg;
}
```

An anderer Stelle werde diese Methode aufgerufen:

```
List<String> parameter = …
List<String> ergebnis = tuWasMit(parameter);
```

Prüfen Sie welches Ergebnis (`ergebnis`) bei dem jeweils angegebenen Argument (`parameter`) erzielt wird und wie sich gegebenenfalls das Argument dabei verändert.

WICHTIG: Zur syntaktischen Vereinfachung werden als Strings nur einzelne Wörter verwendet und die Listen als Sätze dargestellt. In der ersten auszufüllenden Zeile besteht die Liste also aus den beiden Strings „`Niemals`“ und „`Tango`“.

`parameter` vor Aufruf	`ergebnis`	`parameter` nach Aufruf
`Niemals Tango`	**`Niemals Tango`**	**`Niemals Tango tanzen`**
`Niemals Cha Cha`	**`Niemals Cha Cha tanzen`**	**`Niemals Cha Cha tanzen`**
`Ich mag Cha Cha`	**`Ich mag`**	**`Ich mag Cha Cha tanzen`**
`Ich mag Cha Cha Cha`	**`Ich mag tanzen`**	**`Ich mag Cha Cha Cha`**