NACHNAME:	`Poly`	**SEMESTER:**	☐ **M5** ☐ **M6** ☐ M3 ☐ M4 ☐ M7
VORNAME:	`Nom-Kenner`	**VERTIEFUNG:**	☐ **FV** ☐ **IM**

ANLAGE, HILFSMITTEL:	• Ausdruck des vorab bekannt gemachten Quelltextes

UNBEDINGT BEACHTEN:	• **Bevor** Sie mit der Bearbeitung beginnen, müssen die Angaben zur Person auf dieser Seite **vollständig** ausgefüllt sein.

GENERELLE VORGABEN:	• Es sind keinerlei Kommentare verlangt, weder `javadoc`-Kommentare noch andere. • Programmier-Richtlinien (insbesondere Checkstyle) sind zu beachten. Bei Testklassen dürfen aber *magic numbers* verwendet werden.

Aufgabe 1: (19 Punkte)

Die Aufgabe nimmt Bezug auf alle drei Versionen von `Polynom`.

Geben Sie für die drei Versionen (kurz **V1**, **V2**, **V3**) jeweils an,
ob die Anweisung in Ordnung (**OK**) ist, zu einem Compile-Fehler (**CF**) führt oder zur Laufzeit eine Exception (**E**) wirft.

a) Prüfen Sie die angegebenen Anweisungen.

	V1	V2	V3
`Polynom p1 = new Polynom();`	**OK**	**OK**	**CF**
`Polynom p2 = new Polynom(7/8);`	**OK**	**OK**	**OK**
`Polynom p3 = new Polynom(0, 0, 7);`	**OK**	**OK**	**OK**
`Polynom p4 = new Polynom(1, 1, 1, 1, 1);`	**OK**	**OK**	**E**
`Polynom p5 = new Polynom(1) { };`	**CF**	**CF**	**OK**
`Polynom p6 = new Polynom(0, 0, 7) { };`	**CF**	**CF**	**OK**

b) Prüfen Sie nun die Anweisung

```
Polynom p = new Polynom(E);
```

wobei `E` zuvor über den jeweils angegebenen statischen Import bekannt gemacht wird.

	V1	V2	V3
`import static java.lang.Math.*;`	**CF**	**CF**	**OK**
`import static pruefung.Funktionen.*;`	**OK**	**OK**	**CF**
`import static pruefung.Funktion.*;`	**OK**	**OK**	**OK**
`import static pruefung.StammFunktion.*;`	**CF**	**CF**	**CF**

Aufgabe 2: (14 Punkte)

Die Aufgabe bezieht sich auf das `Polynom` der `version1`.

a) Die Testmethode `testeEqualsEinfach` geht schief. Durch Änderung einer einzigen Methode läuft der Test durch. Geben Sie diese Methode vollständig an.

```
@Override
public String toString() {
    String erg = "Polynom";
    for (int koeff : mKoeff) erg += ";" + koeff;
    return erg;
}
// oder (mit import von java.util.Arrays)
@Override
public String toString() {
    return "Polynom" + Arrays.toString(mKoeff);
}
```

b) Auch der Test `testeUnveraenderbarkeitKonstruktor` führt zu einem Fehler. Beheben Sie diesen Fehler. Geben Sie den verbesserten Konstruktor vollständig an.

```
public Polynom(int... koeff) {
    if (koeff.length == 0)
                    throw new IllegalArgumentException();
    mKoeff = koeff.clone();
}
```

Aufgabe 3: (19 Punkte)

Die Aufgabe bezieht sich auf `version2` der Klasse `Polynom`.

Die Testmethode `testeSumme` führt zu einem Fehler. Die Ursache liegt an einem prinzipiellen Problem der Klasse, das immer dann entsteht, wenn der Koeffizient der höchsten Potenz 0 ist.

a) Schreiben Sie eine neue Testmethode (in `PolynomTest`), welche für zwei solche Polynome mit „führenden Nullen“ den Grad überprüft. Eines der Polynome soll eine führende Null und den Grad -1 haben, das andere zwei führende Nullen und den Grad 0.

```
@Test
public void testeGradBeiEntartungen() {
    Polynom p = new Polynom(0);
    assertEquals(-1, p.grad());
    Polynom q = new Polynom(0, 0, 7);
    assertEquals(0, q.grad());
}
```

b) Die beste Behebung des Fehlers ist, Polynome bereits in „Normalform“, also ohne führende Nullen zu erzeugen. Nehmen Sie diese Änderung vor. Geben Sie betroffene Methoden / Konstruktoren vollständig an.

```
public Polynom(int... koeff) {
    for (int a : koeff) mKoeff.add(a);
    while (!mKoeff.isEmpty() && mKoeff.get(0) == 0)
                                    mKoeff.remove(0);
}
// oder (noch besser)
public Polynom(int... koeff) {
    for (int a : koeff)
        if (a != 0 || mKoeff.size() != 0) mKoeff.add(a);
}
```

Aufgabe 4: (19 Punkte)

Die Aufgabe nimmt Bezug auf `version1` und `version2` von `Polynom`.

In `version2` besitzt die Klasse `Polynom` eine in vielerlei Hinsicht wesentlich bessere Methode `stammFunktion` als in `version1`.

a) Der vorhandene Test für die Stammfunktion prüft nur einzelne Funktionswerte und das sogar nur näherungsweise. Schreiben Sie zunächst eine entsprechende Testmethode `testeStammfunktionExakt`, welche die komplette Funktion exakt prüft.

```
@Test
public void testeStammfunktionExakt() {
    Polynom p = new Polynom(3, 0, 7); // 3x^2 + 7
    Polynom ist = p.stammFunktion();
    Polynom soll = new Polynom(1, 0, 7, 0);
    assertEquals(soll, ist);
}
```

b) Überschreiben Sie nun die Methode `stammFunktion` in `version1` der Klasse `Polynom` entsprechend. Obiger Test muss dann fehlerfrei durchlaufen.

```
@Override
public Polynom stammFunktion() {
    int anzahl = mKoeff.length;
    int[] stKoeff = new int[anzahl + 1];
    for (int i = 0; i < anzahl; i++) {
        int n = anzahl - i;
        stKoeff[i] = mKoeff[i] / n;
    }
    stKoeff[anzahl] = 0;
    return new Polynom(stKoeff);
```

```
  }
  // Hinweis: Die Methode stammFunktion in version2 ist zwar
  // exakt, aber wegen einer Integer-Division nicht korrekt.
  // Beim vorliegenden Test spielt das aber keine Rolle,
  // da das Polynom "guenstig" gewaehlt wurde.
  // Dies gilt dann auch fuer die entsprechende Implementierung
  // in version1.
```

Aufgabe 5: (19 Punkte)

Die Aufgabe benutzt alle drei Versionen der Klasse `Polynom`.

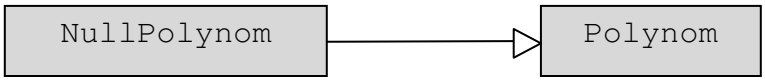
a) Die statische Variable `E` in `UeberallDefinierteFunktion` werde (statt auf 1000) auf 0 gesetzt. Die Variablen `p1`, `p2`, `p3` und `f1` seien wie folgt initialisiert:

```
Polynom p1 = new Polynom(86);
Polynom p2 = new Polynom(86);
Polynom p3 = new Polynom(1, 86);
Funktion f1 = p1;
```

Geben Sie an, welches Ergebnis (`true` oder `false`) die jeweiligen Aufrufe liefern.

	V1	V2	V3
`f1.equals(p1)`	**true**	**true**	**true**
`p2.equals(f1)`	**true**	**true**	**false**
`f1.equals(p3)`	**false**	**false**	**false**
`p3.equals(f1)`	**false**	**false**	**false**
`p3.equals(p1)`	**true**	**true**	**true**

b) Betrachten Sie folgende Beziehung:



Ergänzen Sie (in den eckigen Klammern):

Bei der Beziehung handelt es sich um eine [Vererbung]. Diese ist nur möglich bei der Version [3] von `Polynom`, weil die Klasse `Polynom` in den anderen beiden Versionen [final] ist. Die Beziehung bedeutet: Ein `NullPolynom` [ist ein] `Polynom`. Dies scheint korrekt zu sein. Ein [Nullpolynom] kann dann aber überall verwendet werden, wo ein Objekt der anderen Klasse verwendet werden kann. Dies nennt man das [Liskovsche Substitutionsprinzip]. Wenn die Klasse `Polynom` nicht [unveränderbar] ist, kann das zu einem ernsten Problem führen.