

NACHNAME:	SEMESTER: <input type="checkbox"/> M5 <input type="checkbox"/> M6 <input type="checkbox"/> M3 <input type="checkbox"/> M4 <input type="checkbox"/> M7
VORNAME:	VERTIEFUNG: <input type="checkbox"/> FV <input type="checkbox"/> IM

**VERWENDETE
KLASSEN:**

- Als **Anlage** erhalten Sie den Ausdruck des vorab bekannt gemachten Quelltextes von vier Versionen einer Klasse `Polygon`. Ein `Polygon` soll (in allen vier Versionen) ein ebenes Polygon aus zwei oder mehr Punkten beschreiben.
 - Darüber hinaus wird die aus der Studienarbeit bekannte Klasse `Punkt` verwendet mit der einzigen Änderung, dass die Klasse **nicht** `final` ist. Die Klasse wird als bekannt vorausgesetzt.
-

**UNBEDINGT
BEACHTEN:**

- **Bevor** Sie mit der Bearbeitung beginnen, müssen die Angaben zur Person auf dieser Seite **vollständig** ausgefüllt sein.
- Es sind **keinerlei Hilfsmittel** zugelassen.

**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: (30 Punkte)

Eine Testklasse `PolygonTest` beginne wie folgt:

```
public final class PolygonTest {
    private static final Punkt A1 = new Punkt(1, 0);
    private static final Punkt A2 = new Punkt(1, 0);
    private static final Punkt B = new Punkt(1, 1);
    private static final Polygon P = new Polygon(A1, B);
    private static final Polygon Q = new Polygon(A1, B);
    private static final Polygon R = new Polygon(A1, B, A2);
    private static final Polygon S = new Polygon(A2, B);
    private static final Polygon T = new Polygon(A2, A2);
}
```

a) Prüfen Sie, ob die in der linken Spalte angegebenen **einzelnen** Tests jeweils gut gehen. Tragen Sie in diesem Falle **OK** ein, anderenfalls **F** (für Failure). (Ein Error entsteht nirgends.) Prüfen Sie bei jeder Version **alle** Tests, auch wenn schon eine Failure erkannt wurde.

	v1	v2	v3	v4
<code>assertTrue(P.equals(Q));</code>				
<code>assertTrue(S.equals(P));</code>				
<code>assertEquals(P, Q);</code>				
<code>assertEquals(P, S);</code>				
<code>assertFalse(R.equals(P));</code>				
<code>assertFalse(T.equals(P));</code>				

b) In einer anderen Testmethode der Klasse wird ein Container für Polygone angelegt:

```
Set<Polygon> p = new HashSet<Polygon>();
```

Dann werden **nacheinander** einige Polygone in diesen Container aufgenommen. Geben Sie nach jeder Operation die Größe `p.size()` des Containers bei Verwendung der angegebenen Version von Polygon an.

	v1	v2	v3	v4
<code>p.add(P);</code>				
<code>p.add(Q);</code>				
<code>p.add(R);</code>				
<code>p.add(S);</code>				
<code>p.add(T);</code>				
<code>p.add(P);</code>				

Aufgabe 2: (20 Punkte)

Es soll eine Methode `laenge` entstehen, welche die geometrische Länge eines `Polygon`s liefert.

a) Ergänzen Sie zunächst die Testklasse `PolygonTest` aus **Aufgabe 1** um eine Testmethode `testeLaenge`, welche die Länge der Polygone `P`, `R` und `T` und die Länge der zugehörigen geschlossenen Polygone überprüft.

b) Implementieren Sie die Methode `laenge` für die Klasse `Polygon` in der Version `v2`.

Aufgabe 3: (20 Punkte)

Die Aufgabe bezieht sich auf die Version `v3` von `Polygon` und die Testklasse `PolygonTest` aus **Aufgabe 1**.

a) Diese liefert teilweise ziemlich merkwürdige Ergebnisse. Prüfen Sie die Tests in der linken Spalte **unabhängig** voneinander. Geben Sie wie in Aufgabe 1 als Ergebnis **OK** oder **F** (Failure) an.

<code>assertFalse(P.istGeschlossen());</code>	
<code>assertTrue(R.istGeschlossen());</code>	
<code>assertEquals(R, P.geschlossen());</code>	
<code>assertTrue(P.geschlossen().istGeschlossen());</code>	

b) Selbst die Polygon-Punkte verhalten sich höchst eigenwillig. Der folgende Test geht schief:

```
@Test
public void testePunkte() {
    Punkt a = P.anfangsPunkt();
    Punkt b = P.endPunkt();
    double ab = a.abstandZu(b);
    double ba = b.abstandZu(a);
    assertEquals(ab, ba, 1.e-6);
}
```

Und das liegt nicht an einer zu kleinen Toleranz. Woran dann?

Aufgabe 4: (20 Punkte)

a) Bei Verwendung der Version `v4` in einer (beliebigen) anderen Klasse kann jedes Polygon in einen illegalen Zustand mit weniger als 2 Polygon-Punkten gebracht werden. Die Initialisierung eines Polygons `p` geschehe mittels

```
Punkt a = Punkt.URSPRUNG;  
Punkt b = new Punkt(10., 5.);  
Punkt c = new Punkt(-10., 0.);  
final Polygon p = new Polygon(a, b, c);
```

Ergänzen Sie diesen Code so, dass `p` am Ende illegal ist.

b) Auch bei Verwendung von Version `v2` kann ein ordnungsgemäß erzeugtes Polygon nachträglich zerstört werden. Die Initialisierung muss dabei aber anders geschehen. Erzeugen Sie ein Polygon `q` und bringen Sie es dann in einen illegalen Zustand.
