NACHNAME:	`Alice`	**SEMESTER:**	☐ **M5** ☐ **M6** ☐ M3 ☐ M4 ☐ M7
VORNAME:	`Stemmt`	**VERTIEFUNG:**	☐ **FV** ☐ **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 **geschlossenes**, ebenes Polygon aus zwei oder mehr Punkten beschreiben.
- Darüber hinaus wird die ebenfalls vorab bekannt gemachte Klasse `Punkt` der letzten Prüfung (Sommer-Semester 2012) verwendet. Die Klasse wird (wie damals) 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.

Alle Aufgaben beziehen sich auf die Konstanten des folgenden Interfaces:

```
package w12pruefung;
import w12pruefung.v2.Polygon;
public interface TestKonstanten {
    Punkt A = new Punkt(4, 3);
    Punkt B = new Punkt(-4, 3);
    Punkt C = new Punkt(-4, -3);
    Polygon P = new Polygon(A, B, C);
    Object Q = new Polygon(A, B, C, A);
    Polygon R = new Polygon(A, C, B);
    Polygon S = new Polygon(A, C, A, B, A);
    Polygon T = new Polygon(B, C, A);
    Object U = new Polygon(C, A, B);
}
```

Aufgabe 1: (30 Punkte)

a) Geben Sie an, welchen Wert dann die in der linken Spalte angegebenen Ausdrücke jeweils haben. Abkürzungen (etwa T und F) sind nicht erlaubt. (Der benötigte statische Import soll als vorhanden vorausgesetzt werden.)

	v1	v2	v3	v4
`P.equals(Q)`	**false**	**true**	**true**	**true**
`P.hashCode() == Q.hashCode()`	**true**	**true**	**true**	**false**
`R.equals(S)`	**true**	**false**	**true**	**true**
`P.equals(T)`	**true**	**false**	**false**	**true**
`T.equals(R)`	**false**	**false**	**false**	**true**
`U.equals(T)`	**false**	**false**	**false**	**true**

b) In einer Methode 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
`p.add(P);`	**1**	**1**	**1**	**1**
`p.add((Polygon) Q);`	**2**	**1**	**1**	**2**
`p.add(R);`	**3**	**2**	**1**	**3**
`p.add((Polygon) S);`	**4**	**3**	**1**	**4**
`p.add(T);`	**5**	**4**	**2**	**5**
`p.add((Polygon) U);`	**6**	**5**	**3**	**6**

Aufgabe 2: (20 Punkte)

Es sollen Tests für eine Methode `umgedreht` entstehen, welche ein `Polygon` mit gleichem Anfangspunkt, aber **umgedrehtem Durchlaufsinn** liefert. Das Objekt soll durch die Methode nicht verändert werden.

Eine schon vorhandene Testklasse `PolygonTest` befinde sich im package `w12pruefung`. und enthalte die Zeile

```
import static w12pruefung.TestKonstanten.*;
```

Gehen Sie davon aus, dass die `equals`-Methode von `Polygon` einen semantischen Vergleich vornimmt und den allgemeinen Vertrag einhält.

a) Ergänzen Sie die Testklasse um eine Testmethode `testeUmdrehen1`, welche prüft, ob beim Umdrehen des Polygons `U` der Anfangspunkt und die Anzahl der Punkte gleich bleiben.

```
@Test
public void testeUmdrehen1() {
    Polygon u = (Polygon) U;
    Polygon uI = u.umgedreht();
    assertEquals(u.anfangsPunkt(), uI.anfangsPunkt());
    assertEquals(u.anzahlPunkte(), uI.anzahlPunkte());
}
```

b) Durch Umdrehen von `R` entsteht ein anderes Polygon des Interfaces `TestKonstanten`. Schreiben Sie eine weitere Testmethode `testeUmdrehen2`, die das möglichst einfach überprüft. Testen Sie in dieser Methode weiter, ob durch nochmaliges Umdrehen wieder `R` entsteht.

```
@Test
public void testeUmdrehen2() {
    assertEquals(P, R.umgedreht());
    assertEquals(R, P.umgedreht());
}
```

Aufgabe 3: (20 Punkte)

a) Implementieren Sie für die Version `v1` von Polygon die in der vorigen Aufgabe beschriebene Methode `umgedreht`.

```
public Polygon umgedreht() {
    Polygon erg = new Polygon();
    erg.mPunkte.add(anfangsPunkt());
    for (int i = anzahlPunkte() - 1; i > 0; i--)
        erg.mPunkte.add(mPunkte.get(i));
    return erg;
}
```

b) Der **Durchmesser** eines Polygons ist der größte Abstand, den zwei Polygonpunkte haben können. Beim Polygon `P` ist das der Abstand von `A` und `C`, also 10.

Schreiben Sie für die Version `v2` von Polygon unter konsequenter Verwendung von foreach-Schleifen eine Methode, welche diesen Durchmesser ermittelt.

```
public double durchmesser() {
    double erg = 0.;
    for (Punkt p1 : mPunkte)
        for (Punkt p2 : mPunkte)
            erg = Math.max(erg, p1.abstandZu(p2));
    return erg;
}
```

Aufgabe 4: (20 Punkte)

a) Ergänzen Sie (in den eckigen Klammern):

In der Version 1 von Polygon wird die Methode equals nicht [überschrieben], sondern [überladen]. Bei Version 2 wird durch einen Konstruktor-Aufruf mit zwei Punkten eine [ArrayIndexOutOfBoundsException] geworfen. Bei Version 3 von Polygon wird equals letztlich von [Punkt] geerbt und vergleicht deshalb nur die [Anfangspunkte]. Version 4 verletzt den allgemeinen Vertrag der Methode [hashCode], weshalb die Verwendung in einem [HashSet] scheitert.

b) Betrachten Sie die Version `v3`. An geeigneter Stelle wird folgender Code fehlerfrei compiliert:

```
Polygon p = new Polygon(A, B, C);
p.schliessen();
```

Begründen Sie, warum der Compiler in der zweiten Zeile keinen Fehler liefert. Erläutern Sie dann genau, was zur Laufzeit passiert und weshalb.

Die Methode schliessen der Klasse PolygonPunkt ist package-private
und deshalb im gleichen package sichtbar,
insbesondere in Polygon.
Da Polygon von PolygonPunkt abgeleitet ist
kann die Methode schliessen auf ein Polygon angewendet werden.

Zur Laufzeit ergibt sich eine Endlosschleife.
Nachdem die Methode schliessen im Konstruktor von Polygon
gerufen wurde ist die Verweiskette geschlossen, das heiszt
fuer keinen PolygonPunkt der Kette ist mVoriger == null.
Die while-Schleife in schliessen bricht daher niemals ab.