| **NACHNAME:** | **SEMESTER:** ☐ **M6** ☐ **M7** ☐ M4 ☐ M5 ☐ M8 |
|---|---|
| **VORNAME:** | **VERTIEFUNG:** ☐ **FV** ☐ **IM** |

**HILFSMITTEL:**

- Ausdruck des vorab bekannt gemachten Quelltextes (16 Seiten = 8 Blätter) mit eigenen, handschriftlichen Ergänzungen

**UNBEDINGT BEACHTEN:**

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

**GENERELLE VORGABEN:**

- Der Quelltext der Klassen muss vollständig angegeben werden. Insbesondere sollen auch das `package` und benötigte `import`s enthalten sein.
- Es sind keinerlei Kommentare verlangt, weder `javadoc`-Kommentare noch andere.
- Programmier-Richtlinien (insbesondere Checkstyle) sind zu beachten. Bei der Testklasse dürfen aber *magic numbers* verwendet werden.

## *Aufgabe 1: (18 Punkte)*

In dieser Aufgabe geht es um allgemeine Zusammenhänge.

**a)** Welche der folgenden Aussagen sind richtig?

| | ja | nein |
|---|---|---|
| Nur öffentliche Methoden können überschrieben werden. | | ✓ |
| Die Sichtbarkeit einer Methode bleibt beim Überschreiben gleich. | | ✓ |
| Die Sichtbarkeit kann beim Überschreiben eingeschränkt werden. | | ✓ |
| Innerhalb des gleichen Pakets sind `protected` Methoden sichtbar. | ✓ | |
| `protected` Methoden sind nur im gleichen Paket sichtbar. | | ✓ |
| `protected` Methoden gehören mit zur Klassen-Schnittstelle. | ✓ | |

**b)** Welche der folgenden Aussagen sind richtig?

| | ja | nein |
|---|---|---|
| Die `clone` Methode kann eine ungeprüfte `Exception` werfen. | ✓ | |
| Die `clone` Methode kann eine geprüfte `Exception` werfen. | ✓ | |
| Eine Klasse kann mehr als ein `interface` implementieren. | ✓ | |
| Ein `interface` kann ein anderes `interface` erweitern. | ✓ | |
| Ein `interface` kann eine Klasse erweitern. | | ✓ |
| Ein `interface` ohne Methoden ist nutzlos. | | ✓ |

**c)** Welche der folgenden Empfehlungen sollten möglichst befolgt werden?

| | ja | nein |
|---|---|---|
| Eine geprüfte `Exception` ist einer ungeprüften vorzuziehen. | | ✓ |
| Möglichst sollten eigene `Exception` Klassen verwendet werden. | | ✓ |
| Statt einem leeren Array sollten Methoden besser `null` zurückgeben. | | ✓ |
| Die Annotation `@SuppressWarnings` sollte benutzt werden, um unverständliche Warnungen auszuschalten. | | ✓ |
| Das Überschreiben von Methoden sollte generell durch eine Annotation gekennzeichnet werden. | ✓ | |

**d)** Welche der folgenden Empfehlungen sollten möglichst befolgt werden?

| | ja | nein |
|---|---|---|
| Unveränderbare Klassen sollten `Cloneable` sein. | | ✓ |
| Für veränderbare Klassen sollte es ein Zustandsdiagramm geben. | ✓ | |
| Für alle Daten sollte es Getter und Setter geben. | | ✓ |
| Klassen sollten möglichst klein sein, im Schnitt nur 5 Methoden. | | ✓ |
| Methoden sollten möglichst kurz sein, im Schnitt nur 5 Anweisungen. | ✓ | |

## *Aufgabe 2: (34 Punkte)*

Schreiben Sie eine konkrete Dekorierer-Klasse `SpiegelHorizontal` für die Spiegelung einer `IKurve` an einer vertikalen Geraden `x = a`.

- Neben dem allgemeinen Konstruktor soll es einen weiteren, spezielleren für die Spiegelung an der y-Achse geben.
- Alle Methoden der Schnittstelle sollen vollständig und korrekt implementiert werden.
- `System.out.println` soll eine sinnvolle Ausgabe liefern.

```
package pruefung2;


import pflichtuebung1.korrekt.Intervall;
import studienarbeit.*;


public final class SpiegelHorizontal implements IKurve {
	private final IKurve mKurve;
	private final double mX;

	public SpiegelHorizontal(IKurve k, double x) {
		mKurve = k;
		mX = x;
	}
	public SpiegelHorizontal(IKurve k) {
		this(k, 0.);
	}

	@Override
	public double y(double t) {
		return mKurve.y(t);
	}
```

```
        @Override
        public Intervall y(Intervall urbild) {
            return mKurve.y(urbild);
        }

        private double spiegeln(double x) {
            double d = mX - x;
            return mX + d;
        }
        private Intervall spiegeln(Intervall ix) {
            double x1 = spiegeln(ix.obereGrenze());
            double x2 = spiegeln(ix.untereGrenze());
            return new Intervall(x1, x2);
        }

        @Override
        public double x(double t) {
            double x = mKurve.x(t);
            return spiegeln(x);
        }
        @Override
        public Intervall x(Intervall urbild) {
            Intervall i = mKurve.x(urbild);
            return spiegeln(i);
        }
```

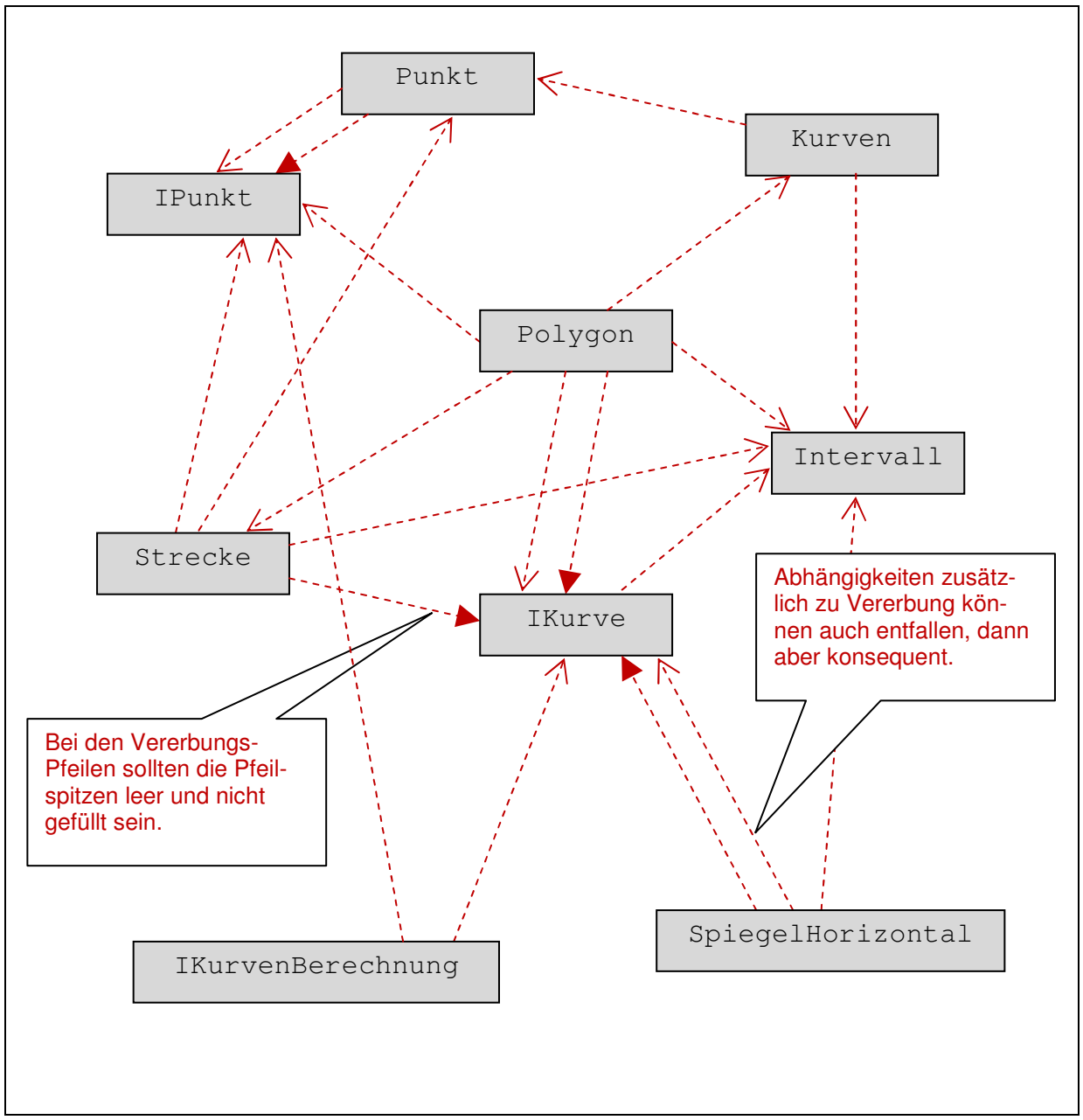
```
        @Override
        public String toString() {
              return "Gespiegelt[" + mKurve + ", x = "
                                   + mX + "]";
        }

  }
```

## *Aufgabe 3: (18 Punkte)*

Tragen Sie in das nachstehende Diagramm die Beziehungen zwischen den **angegebenen** Schnittstellen und Klassen in **UML**-Notation ein.

- Die Beziehungen sollen gerichtet sein. Kardinalitäten sollen **nicht** angegeben werden.
- Auch schwache Abhängigkeiten, bei denen keine konkreten Objekte betroffen sind, sollen eingetragen werden.
- Unterscheiden Sie in der Darstellung **nicht** zwischen Kompositionen, Aggregationen, Assoziationen und noch schwächeren Abhängigkeiten.



## *Aufgabe 4: (20 Punkte)*

Schreiben Sie für die Klasse `SpiegelHorizontal` eine rudimentäre (**JUnit 4**) Testklasse mit einer einzigen Methode.

- Geprüft werden soll die Spiegelung des Polygons von (`x`,0) über (`x`,`y`) nach (0,0) an der y-Achse, wobei `x` = -1458 und `y` = 29/16.
- Konkret überprüft werden sollen die Bilder der Intervalle [0, ½] und [½, 1] und zwar jeweils x- und y-Koordinate.
- Wo möglich, soll auf exakte Übereinstimmung getestet werden.

```
package pruefung2.kurven;


import org.junit.Test;
import static org.junit.Assert.*;

import pflichtuebung1.korrekt.Intervall;
import studienarbeit.*;
import pruefung2.*;


public class TestSpiegel {

    @Test
    public void testePolygonEinfach() {
        final double x = -1458.;
        final double y = 29. / 16.;

        IPunkt p1 = new Punkt(x, 0.);
        IPunkt p2 = new Punkt(x, y);
        IPunkt p3 = new Punkt(0., 0.);

```

```
            IKurve poly = new Polygon(p1, p2, p3);
            IKurve kurve = new SpiegelHorizontal(poly);

            Intervall i1 = new Intervall(0., 1. / 2.);
            Intervall i2 = new Intervall(1. / 2., 1.);

            assertEquals(new Intervall(-x, -x), kurve.x(i1));
            assertEquals(new Intervall(0., -x), kurve.x(i2));
            assertEquals(new Intervall(0., y), kurve.y(i1));
            assertEquals(new Intervall(0., y), kurve.y(i2));

        }
    }
```