
Aufgaben zur Klausur **Objektorientierte Programmierung** im WS 2008/09 (BInf 211, BTInf 211, BMinf 211, BWInf 211)

Zeit: 75 Minuten

erlaubte Hilfsmittel: keine

Bitte tragen Sie Ihre Antworten und fertigen Lösungen ausschließlich an den gekennzeichneten Stellen in das Aufgabenblatt ein. Ist ihre Lösung wesentlich umfangreicher, so überprüfen Sie bitte nochmals Ihren Lösungsweg.

Sollten Unklarheiten oder Mehrdeutigkeiten bei der Aufgabenstellung auftreten, so notieren Sie bitte, wie Sie die Aufgabe interpretiert haben.

Viel Erfolg!

Diese Klausur besteht einschließlich dieses Deckblattes aus 9 Seiten.

Aufgabe 1:

Lesen Sie bitte vor der Bearbeitung die gesamte Aufgabe durch. Es sind in der Aufgabenstellung Vorwärtsreferenzen enthalten.

Zweidimensionale geometrische Figuren (Kreise, Rechtecke, ...) können auf sehr unterschiedliche Arten in einem System implementiert werden. Eine sehr allgemeine und in manchen Anwendungen sehr speicherplatzeffiziente Art ist die, Figuren durch Funktionen zu repräsentieren, z.B. durch folgende Schnittstelle in Java:

```
interface Figure {  
    public boolean contains(Point p);  
}
```

Die so repräsentierten Figuren können beliebig groß sein, zum Beispiel einen ganzen Quadranten darstellen, im Extremfall sogar die gesamte Ebene. Das Visualisieren solcher Figuren kann implementiert werden, indem der gesamte darzustellende Bereich *abgetastet* wird, d.h. es wird punktweise getestet, ob eine Koordinate in einer Figur liegt oder außerhalb. Dieses wird beispielhaft in dem Testprogramm am Ende der Aufgabe gemacht.

Einige häufig verwendete Figuren sind in einer Schnittstelle gesammelt:

```
interface SimpleFigures {  
  
    // useful simple figures  
  
    public static final Figure unitSquare =  
        new Square(Point.org,1.0);  
  
    public static final Figure unitCircle =  
        new Circle(Point.org, 1.0);  
  
    public static final Figure halfPlaneX =  
        new Figure() {  
            public boolean contains(Point p) {  
                return  
                    p.x >= 0;  
            }  
        };  
}
```

In der Schnittstelle sind drei Figuren vordefiniert, das Einheitsquadrat, ein Kreis um den Ursprung mit dem Radius 1.0 und die Halbebene rechts der y-Achse (alle Werte mit positiver x-Koordinate).

Erweitern Sie die Schnittstelle *SimpleFigures* um eine neue Figur, *halfPlaneD*, die die abgeschlossene Halbebene oberhalb der Hauptdiagonalen repräsentiert.

```
public static final Figure halfPlaneD =
```

.....

.....

.....

.....

.....

.....

.....

.....

Die Klasse *Circle* ist wie folgt definiert:

```
public class Circle implements Figure {  
  
    private Point org;  
    private double radius;  
  
    public Circle(Point o, double r) {  
        org = o;  
        radius = r;  
    }  
    public boolean contains(Point p) {  
        double dx = p.x - org.x;  
        double dy = p.y - org.y;  
        return  
            Math.sqrt(dx*dx + dy*dy) <= radius;  
    }  
}
```

Entwickeln Sie analog dazu eine Klasse *Square*

```
public class Square implements Figure {  
  
    .....  
  
    .....  
  
    public Square( ..... ) {  
  
        .....  
  
        .....  
    }  
    public boolean contains(Point p) {  
  
        .....  
  
        .....  
  
        .....  
  
        .....  
    }  
}
```

Die Koordinaten, die Punkte, werden in diesem Beispiel durch folgende Klasse realisiert:

```
final
public class Point {
    final public double x;
    final public double y;

    public Point(double x1, double y1) {
        x = x1; y = y1;
    }
    static final public Point org = new Point(0.0, 0.0);
    static final public Point x1 = new Point(1.0, 0.0);
    static final public Point y1 = new Point(0.0, 1.0);
    static final public Point xy = new Point(1.0, 1.0);
}
```

In der Schnittstelle sind einige häufig verwendete Punkte vordefiniert.

Figuren können auf viele unterschiedliche Arten manipuliert werden. Ein Transformationstyp ist der, dass eine punktweise Transformation, z.B. eine Verschiebung oder Drehung, vorgenommen wird. Transformationen können ebenfalls durch Objekte repräsentiert werden. Die folgende Klasse wird in diesem Beispiel verwendet:

```
interface Transform {

    public Point move(Point p);

    // useful elementary transformations

    static final public Transform mirror =
        new Transform() {
            public Point move(Point p) {
                return
                    new Point(-p.x, -p.y);
            }
        };

    static final public Transform rotate90 =
        new Transform() {
            public Point move(Point p) {
                return
                    new Point(-p.y, p.x);
            }
        };
}
```

In dieser Schnittstelle sind zwei Transformationen durch die Verwendung von anonymen Klassen realisiert, eine punktweise Spiegelung am Ursprung und eine 90-Grad-Drehung.

Mit den Methoden aus der Translationsklasse können Figuren verschoben werden.

```
public class Translation implements Transform {  
  
    private Point p;  
  
    public Translation(Point p1) {  
        p = p1;  
    }  
  
    public Point move(Point p1) {  
        return  
            new Point(p1.x - p.x, p1.y - p.y);  
    }  
}
```

Entwickeln Sie analog zur Translationsklasse eine Klasse für die Erzeugung von Skalierungen, also von Transformationen, die alle Kooordinaten mit einem festen Wert in x-Richtung skalieren, und mit einem zweiten Wert in y-Richtung.

```
public class Scaling implements Transform {  
  
    .....  
  
    .....  
  
    public Scaling( ..... ) {  
  
        .....  
  
        .....  
    }  
  
    public Point move(Point p1) {  
  
        .....  
  
        .....  
  
        .....  
  
        .....  
    }  
}
```

Eine Transformation kann auf eine Figur angewendet werden. Das Resultat ist wieder eine Figur. Dieser Prozess wird ebenfalls durch eine Klasse implementiert. Vervollständigen Sie diese Klasse:

```
public class TransformedFigure implements Figure {  
    .....  
    .....  
    public TransformedFigure( ..... ) {  
        .....  
        .....  
    public boolean contains(Point p) {  
        .....  
        .....  
        .....  
        .....  
    }  
}
```

Zwei oder mehrere Figuren können auf unterschiedliche Weise kombiniert werden, zum Beispiel sind alle Mengenoperationen (Vereinigung, Durchschnitt, Subtraktion, ...) möglich. Die zweistellige Kombination wird wieder mit Hilfe einer Klasse beschrieben:

```
abstract public class CombinedFigure implements Figure {  
    protected Figure fig1;  
    protected Figure fig2;  
  
    protected CombinedFigure(Figure f1, Figure f2) {  
        fig1 = f1; fig2 = f2;  
    }  
}
```

Entwickeln Sie eine abgeleitete Klasse für die Vereinigung zweier Figuren.

```
public class UnionFigure extends CombinedFigure {  
  
    public UnionFigure( ..... ) {  
  
        .....  
  
        .....  
    }  
  
    public boolean contains(Point p) {  
  
        .....  
  
        .....  
  
        .....  
  
    }  
}
```


Gegeben sei die folgende Mini–Anwendung:

```
class Main implements SimpleFigures {  
    public static void main(String [] args) {  
        Figure fig1 = unitCircle;  
        Figure fig2 = new UnionFigure(fig1, unitSquare);  
        Figure fig3 = new TransformedFigure(fig2, new Scaling(new Point(0.4,0.4)));  
        Figure fig4 = new TransformedFigure(fig3, new Translation(new Point(0.5,0.5)));  
        scan(fig1,10,10);  
        scan(fig2,10,10);  
        scan(fig3,10,10);  
        scan(fig4,10,10);  
    }  
    static void scan(Figure fig, int w, int h) {  
        for (int j = h-1; j >= 0; --j) {  
            for (int i = 0; i < w; ++i)  
                scan(fig.contains(new Point((double)j/h, (double)i/w)), j, i);  
            System.out.println();  
        }  
        System.out.println();  
    }  
    static void scan(boolean c, int y, int x) { System.out.print(c ? 'X' : '.'); }  
}
```

Wie viele Objekte der Klasse *Point* werden bei der Ausführung von *scan(fig1,10,10)*; erzeugt?

.....

Wie viele Objekte der Klasse *Point* werden bei der Ausführung von *scan(fig3,10,10)*; erzeugt?

.....

Wie viele unterschiedliche Objekte sind über die Variablen *fig1,fig2,fig3* und *fig4* referenzierbar?

.....

Wie viele Objekte der Klasse *TransformedFigure* oder Unterklassen dieser Klasse werden bei der Ausführung des gesamten Programms erzeugt?

.....