## Aufgabe 8.1 (Ü) Kreise

In dieser Aufgabe soll eine Klasse für Kreise entworfen werden.

a) Schreiben Sie dazu zunächst eine Klasse Vector2d, die einen zweidimensionalen Vektor repräsentiert. Er soll die Koordinaten `x` und `y` vom Typ `double` als Objektvariablen enthalten. Diskutieren Sie mit Ihrem Tutor, welche Objektmethoden hilfreich sein könnten, um in den folgenden Teilaufgaben mit Vektoren rechnen zu können und implementieren Sie diese.

b) Schreiben Sie anschließend eine Klasse für Kreise (Circle). Überlegen Sie sich hierfür zunächst, welche Eigenschaften diese Objekte haben. Diskutieren Sie, welche Eigenschaften als Objektvariablen und welche als Objektmethoden implementiert werden sollten.

c) Schreiben Sie anschlie§end die Objektmethoden:

```
public boolean isIncluded(Circle c)

public boolean includes(Circle c)
```

zu der Klasse Circle die testet, ob der jeweilige Kreis in einem anderen enthalten ist, bzw. ob er einen anderen Kreis enthält.

Testen Sie Ihre Implementierung in einer geeigneten Testklasse mit einer `main`-Methode mit verschiedenen geeigneten Ein- und Ausgaben.

## Lösungsvorschlag 8.1

```
class Vector2d {

    private double x;
    private double y;

    // Konstruktor
    public Vector2d(double x, double y) {
        this.x = x;
        this.y = y;
    }

    // Addition
    public Vector2d add(Vector2d v) {
        return new Vector2d(x + v.x, y + v.y);
```

```
    }

    // Substraktion
    public Vector2d subtract(Vector2d v) {
        return new Vector2d(x - v.x, y - v.y);
    }

    // Berechnet die Laenge eines Vektors
    public double length() {
        return Math.sqrt(x * x + y * y);
    }

    // Gibt informationen zum Vektor zurueck
    public String toString() {
        return x + "," + y;
    }

    // Setzt die Koortinaten des Vektors
    public void set(double x, double y) {
        this.x = x;
        this.y = y;
    }

    // Gibt x zurueck
    public double getX() {
        return x;
    }

    // Gibt y zurueck
    public double getY() {
        return y;
    }
}

class Circle {

    private double radius;
    private Vector2d center;

    // Konstruktor
    public Circle(Vector2d center, double radius) {
        this.center = center;
        this.radius = radius;
    }

    // Gibt Informationen zum Kreis als String zurueck
    public String toString() {
        return "Circle with center (" + center.toString() + ") and
            radius "
        + radius + ".";
    }

    // Ueberprueft ob der Kreis einen weiteren Kreis c enthaelt.
    public boolean includes(Circle c) {
        return c.center.subtract(center).length() + c.radius < radius;
    }
```

```
    // Ueberprueft ob der Kreis in einem anderen Kreis c enthalten ist.
    public boolean isIncluded(Circle c) {
        return c.includes(this);
    }

    // Gibt den Radius zurueck.
    public double getRadius() {
        return radius;
    }

    // Setzt den Radius
    public void setRadius(double radius) {
        this.radius = radius;
    }

    // Gibt den Kreismittelpunkt zurueck
    public Vector2d getCenter() {
        return center;
    }

    // Setzt den Kreismittelpunkt
    public void setCenter(Vector2d center) {
        this.center = center;
    }
}

public class CirclesTest {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Vector2d circleCenter = new Vector2d(5, 5);
        Circle circle = new Circle(circleCenter, 5);
        //erzeuge ein kreis innerhalb circle
        Vector2d inCircleCenter = new Vector2d(6, 6);
        Circle inCircle = new Circle(inCircleCenter, 1);
        // erzeuge einen Kreis ausserhalb circle
        Vector2d outOfCircleCenter = new Vector2d(9, 5);
        Circle outOfCircle = new Circle(outOfCircleCenter, 2);
        System.out.println(circle);
        System.out.println(inCircle);
        System.out.println(outOfCircle);
        System.out.println(circle.includes(inCircle));
        System.out.println(circle.includes(outOfCircle));
        System.out.println(inCircle.isIncluded(circle));

    }
}
```

**Aufgabe 8.2 (Ü) Einfache verkettete Liste**

Einfach verkettete Integer-Listen bestehen aus einer Kette von Elementen. Jedes Element besteht aus einer Zahl (`info`) und der Referenz (`next`) auf den Rest der Liste. Die leere Liste wird durch `null` repräsentiert.

a) Erstellen Sie eine Klasse `IntList`. Implementieren Sie einen passenden Konstruktor,

der die Attribute `info` und `next` initialisiert.

b) Implementieren Sie eine Methode `append(int info)`, die eine neue Liste zurückgibt, die eine Kopie der aktuellen Liste ist, an deren Ende ein neues Element mit Zahl `info`, angefügt wurde.

c) Implementieren Sie eine Methode `public String toString()`, die alle Elemente der Liste als String zurückgibt.

**Aufgabe 8.3 (H) Einfache verkettete Liste (Fortsetzung)**

Die Hausaufgabe setzt Aufgabe 8.3 fort.

a) Implementieren Sie eine Methode `sum()`, die die Summe aller Listenelemente zurückgibt.

b) Implementieren Sie eine Methode `reverse()`, die eine neue Liste zurückgibt, welche die Zahlen der aktuellen Liste in umgekehrter Reihenfolge enthält.
**Hinweis:** Überlegen Sie sich zunächst eine Lösung für einelementige Listen. Danach überlegen Sie sich, wie Sie mit Listenelementen verfahren, auf die noch weitere Elemente folgen.

**Lösungsvorschlag 8.3**

```
public class IntList {

    private int     info;       // die info des Listenelementes
    private IntList next;       // Der Rest der Liste


    public IntList(int info, IntList next) {
        this.info = info;
        this.next = next;
    }



    public IntList append(int info) {
        if (next == null)
            return new IntList(this.info, new IntList(info, null));
        else
            return new IntList(this.info, next.append(info));
    }

    // Berechnet die Summe aller Elemente
    public int sum() {
        if (next == null)
            return info;
        else
            return info + next.sum();
    }

    // Rekursive Hilfsmethode zum invertieren der Liste
    private IntList reverseAux(IntList acc) {
        if (next == null)
            return new IntList(info, acc);
        else
            return next.reverseAux(new IntList(info, acc));
    }
```

```
    // Gibt invertierte Liste zurueck
    public IntList reverse() {
        return reverseAux(null);
    }

    // Gibt eine String Representation der Liste zurueck
    public String toString() {
        if (next == null)
            return "" + info;
        else
            return info + "," + next;
    }

    public static void main(String[] args) {
        IntList lst = new IntList(1, null);
        for (int i = 2; i < 10; i++)
            lst = lst.append(i);
        System.out.println(lst);
        System.out.println(lst.reverse());
        System.out.println(lst.sum());
    }
}
```