Einleitung

Obwohl die C-Programmiersprache die objektorientierte Programmierung nicht unterstützt, kann man trotzdem das Konzept teilweise umzusetzen. Das Ziel dieser Übung is sich mit der objektorientierten Programmierung in C vertraut zu machen und die Grenzen kennen zu lernen.
In der Aufgabe geht es darum geometrische Objekte (z.B. Linien, Kreise, Rechtecke, ...) in einem Array zu speichern und sie dann an Funktionen zu übergeben, die den gesamten Flächeninhalt und gesamten den Umfang der Objekte ermitteln. Das Java-Beispiel in Anhängen A to C soll die Aufgabenstellung verdeutlichen.

1. Einfache Klassen

Implementiere eine Klasse `point_t`, zur Repräsentation von 2-dimensionalen Punkten. Desweiteren, die Klasse soll folgende Funktionen beinhalten.

- **`double`** `point_length(point_t* this)`
  - Memberfunktion, die die Entfernung von (0,0) bis (`x`, `y`) berechnet.
- `point_t point_diff(point_t* a, point_t* b)`
  - Statische Funktion zur Berechnung der Differenz von zwei Punkten
- **`double`** `point_dist(point_t* a, point_t* b)`
  - Statische Funktion zur Berechnung der Distanz zwischen zwei Punkten

Hinweise zur Lösung dieser Aufgabe finden Sie im Anhang A.

2. Dreierregel

Schreiben Sie eine erweiterte Punktklasse `point_adv_t` zusätzlich zu den Koordinaten eine Membervariable `label` beinhaltet, die den Namen des Punktes speichert. Implementieren Sie einen geeigneten Konstruktor. Desweiteren, soll nach Dreierregel auch den Kopiertkonstruktor, Zuweisungsoperator, Destruktor implementiert werden.

- Kopierkonstruktor
  - Ein Kopierkonstruktor wird auf uninitialisierte Objekte angewendet. Er weist eine Kopie des Quellobjektes einem uninitialisierten Zielobjekt zu.

```
1 point_adv_t p1;
2 point_adv_t p2;
3 point_adv_constructor(&p1, 4, 5, "A")
4 point_adv_copy(&p1, &p2);
5 /* ... */
```

- Zuweisungsoperator
  - Ein Zuweisungsoperator wird auf bereits initializierte Objekte angewendet. Das Zielobjekt wird zuerst bereinigt. Danach wird eine Kopie aus dem Quellobjekt erzeugt und dem Zielobjekt zugewiesen. So entstehen keine Speicherlecks.

```
point_adv_t p1;
point_adv_t p2;
point_adv_constructor(&p1, 4, 5, "A")
point_adv_constructor(&p2, 6, 7, "B")
point_adv_assign(&p1, &p2);
/* ... */
```

- Destruktor
  - Ruf Destruktoren von allen Membervariablen des Objektes.

Zeigen Sie anhand von Beispielen die typische Anwendung von den Funktionen.

3. Vererbung

Hinweise für die Lösung dieser Aufgabe finden Sie im Anhang B und Anhang C.

3.1. Basisklasse

Erstellen Sie eine abstrakte Klasse `shape_t` mit den virtuellen Funktionen

- `double (*area)(shape_t*)`
  - Flächeninhalt
- `double (*extent)(shape_t*)`
  - Umfangberechnung

Eine Klasse kann als abstrakt angesehen werden, wenn mindestens eine Funktion nicht implementiert ist. Für unsere Zwecke kann man die Tabelle auf die virtuelle Funktionen mit `NULL` initialisieren.

```
static class_vtbl_t class_vtbl = {NULL};
```

3.2. Klassenhierarchie

- Leiten Sie von der Klasse `shape_t` die Klasse `line_t`, die eine Linie repräsentiert und eine **eine** weitere Klasse ihrer Wahl ab (z.B. Kreis `circle_t`, Rechteck `rectangle_t` oder Dreieck `triangle_t`). Schreiben Sie die entsprechenden Konstruktoren, Destruktoren und implementiere das `shape_t`-Interface.
- Leiten Sie von der Klasse `line_t` die Klasse `arrow_t` ab.

3.3. Berechnung vom Gesamtumfang und Flaecheninhalt

- `double sum_area(shape_t** shape, size_t size)`
  - Summe aller Flaecheninhalte in `shape`
- `double sum_extent(shape_t** shape, size_t size)`
  - Summe aller Umfaenge in `shape`

3.4. Testlauf

Erzeuge einige Objekte und speichere sie in einem Array

```
size_t size = 4;
shape_t* objects[size];
objects[0] = (shape_t*) &circle;
objects[1] = (shape_t*) &rectangle;
objects[2] = (shape_t*) &triangle;
objects[3] = (shape_t*) &circle2;

printf("%f\n", sum_area(objects, size));
printf("%f\n", sum_extent(objects, size));
```

Anhang A Point-Klasse

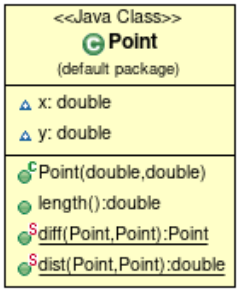

Abbildung 1: Point-Klasse

```java
import java.lang.Math;

public class Point {
    double x;
    double y;

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }


    public double length() {
        return Math.sqrt(x * x + y * y);
    }

    public static Point diff(Point a, Point b) {
        return new Point(a.x - b.x, a.y - b.y);
    }

    public static double dist(Point a, Point b) {
        return Point.diff(b, a).length();
    }

}
```

Anhang B Objekt-Klassen

```java
public abstract class Shape {
    public abstract double area();
    public abstract double extent();
}
```

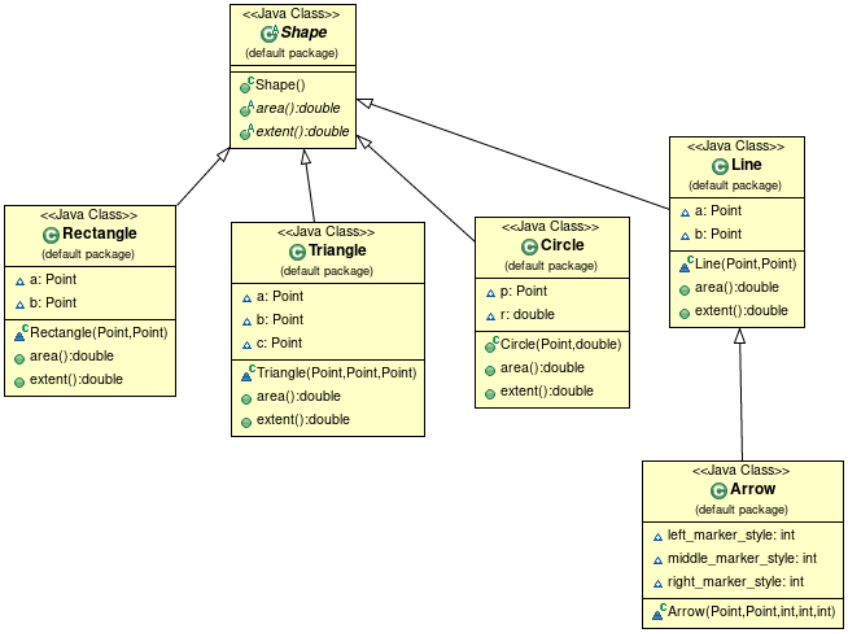

Abbildung 2: Übersicht über die Abhängigkeiten

```java
public class Circle extends Shape {
    Point p;
    double r;

    public Circle(Point p, double r) {
        this.p = p;
        this.r = r;
    }

    @Override
    public double area() {
        return 2 * r * 3.1415;
    }

    @Override
    public double extent() {
        return r * r * 3.1415;
    }
}
```

```java
public class Rectangle extends Shape {
    Point a;
    Point b;

    Rectangle(Point a, Point b) {
        this.a = a;
        this.b = b;
    }

    @Override
    public double area() {
        return (b.x - a.x) * (b.y - a.y);
```

```
    }

    @Override
    public double extent() {
        return 2 * ((b.x - a.x) + (b.y - a.y));
    }

}
```

```
public class Triangle extends Shape {
    Point a;
    Point b;
    Point c;

    Triangle(Point a, Point b, Point c) {
        this.a = a;
        this.b = b;
        this.c = c;
    }

    @Override
    public double area() {
        return (a.x * (b.y - c.y) + b.x * (c.y - a.y) + c.x * (a.y -
            ↪ b.y)) / 2;
    }

    @Override
    public double extent() {
        return Point.dist(a, b) + Point.dist(b, c) + Point.dist(b, c);
    }

}
```

```

public class Line extends Shape {
    Point a;
    Point b;

    Line(Point a, Point b) {
        this.a = a;
        this.b = b;
    }

    @Override
    public double area() {
        return 0;
    }

    @Override
    public double extent() {
        return Point.diff(a, b).length();
    }
```

```
20
21 }
```

```
1
2  public class Arrow extends Line {
3      int left_marker_style;
4      int middle_marker_style;
5      int right_marker_style;
6
7      Arrow(Point a, Point b, int lms, int mms, int rms) {
8          super(a, b);
9          this.left_marker_style = lms;
10         this.middle_marker_style = mms;
11         this.right_marker_style = rms;
12     }
13 }
```

Anhang C Testlauf

```
1  public class Run {
2      public static double sumExtent(Shape[] objs) {
3          double sum = 0;
4          for (int i = 0; i < objs.length; ++i) {
5              sum += objs[i].extent();
6          }
7          return sum;
8      }
9
10     public static double sumArea(Shape[] objs) {
11         double sum = 0;
12         for (int i = 0; i < objs.length; ++i) {
13             sum += objs[i].area();
14         }
15         return sum;
16     }
17
18     public static void main(String[] args) {
19         Shape[] objects = new Shape[4];
20         objects[0] = new Circle(new Point(4.1, 5.3), 4.5);
21         objects[1] = new Triangle(new Point(5.1, 8), new Point(5.22,
               ↪ 8.2), new Point(1, 4));
22         objects[2] = new Rectangle(new Point(9, 8), new Point(11, 12));
23         objects[3] = new Circle(new Point(4, 6), 8);
24
25         System.out.println(sumArea(objects));
26         System.out.println(sumExtent(objects));
27     }
28
29 }
```