Lernziele

Studierende sind in der Lage:

- eine neue Klasse anzulegen
- ein einfaches Java Programm zu schreiben und zu kompilieren
- die main-Methode zu beschreiben
- die Parameter der main-Methode auszulesen
- eine andere Klasse einzubinden

Aufgabenstellung

Es soll eine Software-Struktur entwickelt werden, mit der Punkte der zweidimensionalen Ebene dargestellt werden können.

Modellierung

Zur Darstellung der Punkte der zweidimensionalen Ebene entwickeln wir eine **Klasse `Punkt`.**

Ein **`Punkt`** der Ebene ist durch seine **Koordinaten `x` und `y`** bestimmt - dies werden die **Attribute** der **`Punkt`-Klasse**.

Zum **Erzeugen von `Punkt`-Objekten** sollen zwei **Konstruktoren** bereitgestellt werden.

- Ein Konstruktor soll als Parameter die Koordinaten-Werte des zu erzeugenden Punktes übernehmen,
- Der andere Konstruktor soll parameterlos sein und den Punkt im Koordinatenursprung erzeugen.

Zwei **Zugriffsmethoden** sollen die **Koordinatenwerte** eines Punktes zurückgeben.

Durch zwei weitere Methoden sollen sich Punkte versetzen bzw. verschieben lassen:

- **versetzen** besteht im Zuordnen neuer Koordinatenwerte,
- **verschieben** bedeutet, Differenzwerte zu den bisherigen Koordinatenwerten zu addieren.

Außerdem sind **zwei allgemeine `Object`-Methoden** an die **`Punkt`-Klasse** anzupassen:

- die **`toString()`**-Methode soll eine geeignete Zeichenkettendarstellung für Punkte liefern,
- die **`equals()`**-Methode soll testen, ob zwei **`Punkt`-Objekte** den gleichen Punkt repräsentieren.

In **UML-Notation** lässt sich die **`Punkt`-Klasse** übersichtlich beschreiben:

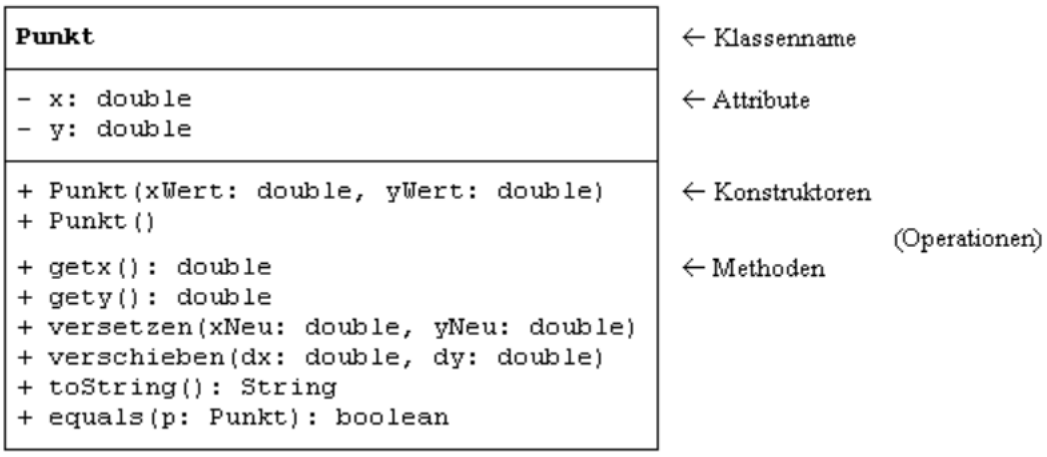

Abbildung 1 Klassendiagramm Punkt

Implementierung

Damit die Klasse `Punkt` allgemein benutzbar ist, wird sie als **öffentlich** vereinbart:

```
public class Punkt{
 //innerhalb dieses Klammern-Paares wird die Klasse beschrieben
}
```

Erzeugen Sie diese Klasse unter dem erforderlichen Dateinamen in Eclipse.

Die **Attribute** der Klasse sind die **x-** und **y-Koordinaten** der Punkte, deren Werte beliebige **Dezimalzahlen** sein können. Sie werden nach dem Geheimnisprinzip als privat vereinbart und damit dem direkten Zugriff von außen entzogen:

```
public class Punkt{
        private double x;//x-Koordinate
        private double y;//y-Koordinate
}
```

Speichern Sie diese Klasse, um festzustellen, ob sie syntaktisch korrekt ist.

Objekte werden durch **Konstruktoren** erzeugt. Sie **initialisieren** die neuen Objekte, d.h., sie stellen einen definierten Anfangszustand her. Der **Name eines Konstruktors** ist prinzipiell der **Klassenname**:

```
public class Punkt{

 private double x;//x-Koordinate
 private double y;//y-Koordinate

 public Punkt(double xWert, double yWert){
  x = xWert;
  y = yWert;
 }
}
```

Praktischer Hinweis:

Kompilieren (Eclipse macht dies im Hintergrund automatisch) Sie beim Entwickeln von Klassen so oft wie möglich - am besten jedes Mal, wenn Sie eine neue Anweisung geschrieben haben. Auf diese Weise lassen sich mögliche Fehler leicht eingrenzen. Nutzen Sie auch die Hilfe von Eclipse.

Test

Um die Verwendung der Klasse `Punkt` zu testen, legen Sie eine zweite **Klasse** **`PunktTester`** an, die lediglich über die **Startmethode** **`main()`** verfügt:

```
public class PunktTester{
  public static void main(String[] args){
    Punkt p1 = new Punkt(5,7);
    System.out.println("Punkt 1 erzeugt");
  }
}
```

Speichern und **starten** Sie diese Klasse

Zur Kontrolle der korrekten Arbeit des Konstruktors sollen die Attributwerte des neuen `Punkt`-Objekts ausgegeben werden. Dazu braucht man **Zugriffsmethoden** für die Attributwerte. Vereinbaren Sie **in der Klasse `Punkt`** eine **öffentliche Methode `getx()`** zum Holen des `x`-Wertes:

```
public double getx(){
  return x;
}
```

Vereinbaren Sie eine entsprechende Methode für **gety()**.

Fügen Sie im `PunktTester` nach dem Erzeugen des 1. Punkts folgende Zeilen ein:

```
System.out.println("x = " + p1.getx());
System.out.println("y = " + p1.gety());
```

Speichern und **starten** Sie diese Klasse erneut.

Wenn Objekte einer Klasse in unterschiedlicher Weise erzeugt werden sollen, so muss die Klasse dazu mehrere Konstruktoren haben.

Vereinbaren Sie für **`Punkt`** einen **zweiten Konstruktor**, der keine Parameter hat; er soll den Punkt im Koordinaten-Ursprung erzeugen.

Lassen Sie im `PunktTester` einen zweiten Punkt mit diesem Konstruktor erzeugen und geben Sie dessen Koordinaten aus.

In Java ist für alle Objekte eine Methode **`toString()`** vorgegeben; sie soll eine Zeichenkette als **Textdarstellung für die Objekte** liefern.
Wir passen diese Methode für die `Punkt`-Klasse so an, dass sie für jeden Punkt die übliche Darstellung als **Koordinaten-Paar `"(xWert, yWert)"`** liefert:

```
public String toString(){
   return "(" + x + ", " + y + ")";
}
```

Anschließend können Sie das Anzeigen von Punkten in `PunktTester` einfacher realisieren:

```
System.out.println("2. Punkt: " + p2.toString());
```

Man kann einen **Punkt** an eine andere Stelle **versetzen**, indem man seinen Koordinaten neue Werte zuweist; vereinbaren Sie in der Klasse `Punkt` eine entsprechende Methode:

```
public void versetzen(double xNeu, double yNeu){

}
```

Weiterhin kann man einen **Punkt** an eine andere Stelle **verschieben**, indem man zu den vorhandenen Koordinatenwerten bestimmte Differenzwerte addiert. Vereinbaren Sie auch dafür eine entsprechende Methode:

```
public void veschieben(double dx, double dy){

}
```

Testen Sie beide Methoden in `PunktTester`:

- versetzen Sie zunächst den zweiten Punkt nach `(3, 3);`
- verschieben Sie ihn danach um `(2, 4).`

Lassen Sie nach jeder Operation die Koordinaten ausgeben.

Der erste und der zweite Punkt sollten nun die gleichen Koordinaten haben.

Lassen Sie in `PunktTester` folgenden Vergleich ausführen:

```
System.out.println("1. und 2. Punkt vergleichen: ");
if (p1 == p2)
{
  System.out.println("p1 und p2 zeigen auf gleiches Punkt-Objekt");
}
else
{
  System.out.println("p1 und p2 zeigen auf verschiedene Punkt-Objekte");
}
```

Warum werden `p1` und `p2` als verschiedene Punkte erkannt?

Vereinbaren Sie in der **Klasse `Punkt`** eine Methode **`equals()`**, die testet, ob ein Punkt die gleichen Koordinaten wie ein anderer Punkt hat:

```
public boolean equals(Punkt p)
{
  return (x == p.x) && (y == p.y);
}
```

Beachten Sie:

in Java kann man einer Klasse auch auf die verborgenen Attribute anderer Objekte der gleichen Klasse direkt zugreifen (auch wenn das nicht ganz konsequent im Sinne der Objekt-Orientierung ist).

Lassen Sie nun in `PunktTester` folgenden Vergleich ausführen:

```
if (p1.equals(p2))
{
     System.out.println("p1 und p2 haben gleiche Koordinaten");
}
else
{
     System.out.println("p1 und p2 haben verschiedenen Koordinaten"):
}
```

Die **Vergleichsmethode `equals()`** ist in Java einheitlich zum **Testen der "inhaltlichen Gleichheit" von Objekten** vorgesehen; was das konkret bedeutet, ist von der jeweiligen Klasse abhängig. Zu einer vollständigen Klassenbeschreibung gehört daher oftmals die Definition einer entsprechenden `equals()`-Methode. (Sonst wird die `equals()`-Methode der Oberklasse verwendet.)

Prägen Sie sich den **Unterschied** zwischen der **Vergleichsoperation `==`** und der **Vergleichsmethode `equals()`** ein.

Zusatzaufgaben

Legen Sie im `PunktTester` ein drittes `Punkt`-Objekt an, dessen Koordinaten der Benutzer eingeben soll. Benutzen Sie dazu die Methode **`readDouble()`** der Klasse **`Keyboard`**. (Wie man sie verwendet, wird in der Klasse **`KeyboardTest`** demonstriert.)

Versetzen Sie anschließend den ersten Punkt in den neuen Punkt.

Überzeugen Sie sich davon, dass beide Objekte nun den gleichen Punkt der Ebene darstellen.

Punkte in der Ebene anzeigen lassen

Zur graphischen Veranschaulichung von Punkten steht eine **Klasse `Display`** zur Verfügung. Sie stellt zwei Methoden

```
public void show(Punkt p)
public void hide(Punkt p)
```

bereit, die den übergebenen Punkt in einem Koordinatensystem darstellt bzw. wieder daraus entfernt.

Modifizieren Sie den `PunktTester` in der Weise, dass die Punkte von einem `Display`-Objekt graphisch angezeigt werden. Erzeugen Sie ein solches Objekt mit einer Größe von `400x500` Pixeln. Sehen Sie sich im Quelltext an, wie der Konstruktor der Klasse `Display` aufgebaut ist.

Fügen Sie überall dort geeignete graphische Anweisungen ein, wo bisher Textausgaben für die Punkte gemacht werden.

Um die Arbeit von `PunktTester` schrittweise verfolgen zu können, sollte an bestimmten Stellen des Programms auf eine Tastatur-Eingabe des Benutzers gewartet werden. Realisieren Sie dies mithilfe der Klasse **`Keyboard`**.

Nutzung der Klasse Punkt

Ein achsenparalleles Rechteck in der Ebene kann auf folgende Weise charakterisiert werden:

- es hat einen **Startpunkt** (seine **linke untere Ecke**), das ist ein **Punkt**;
- es hat eine **Breite**, das ist eine **Dezimalzahl**;
- es hat eine **Höhe**, das ist eine **Dezimalzahl**.

Zum Erzeugen von Rechtecken sollen zwei Möglichkeiten zur Verfügung stehen:

- der Startpunkt, die Höhe und die Breite werden vorgegeben,
- die Koordinaten des Startpunkts sowie die Höhe und die Breite werden vorgeben.

Für Rechtecke sollen folgende Methoden vorhanden sein:

- Zugriffsmethoden zum Holen des Startpunkts, der Höhe und der Breite des Rechtecks,
- Berechnungsmethoden für die Fläche und den Umfang des Rechtecks,
- eine Methode zur Darstellung als Zeichenkette,
- eine Methode zum inhaltlichen Vergleich mit einem anderen Rechteck.

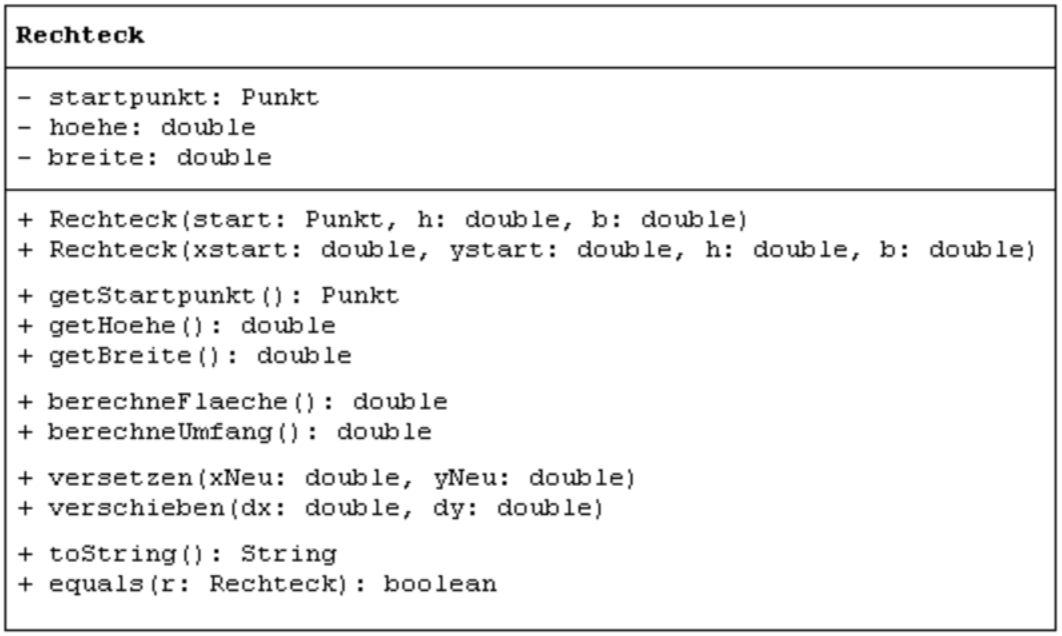

Abbildung 2 Klassendiagramm Rechteck

Entwickeln Sie nach diesem Klassendiagramm eine **Klasse** **`Rechteck`**.

Schreiben Sie eine **Klasse** **`RechteckTester`** zum Testen Ihrer Klasse **`Rechteck`**.

Graphische Darstellung von Rechtecken

Die Klasse **`Display1`** ist eine Erweiterung der Klasse **`Display`** aus der vorigen Aufgabe. Sie stellt zusätzlich eine **`show()`**- und eine **`hide()`**-Methoden für Rechtecke zur Verfügung. Wenn Sie sich bei der Implementierung der Klasse **`Rechteck`** an die Vorgaben des Klassendiagramms gehalten haben, können Sie sich die von Ihnen erzeugten **`Rechteck`**-Objekte graphisch anzeigen lassen.
Schreiben Sie eine entsprechende **Klasse** **`RechteckTester1.`**

Zusatzaufgaben

Erweitern Sie die Klasse `Punkt` um zusätzliche Methoden:

- zum **Spiegeln von Punkten** an der `x`- oder `y`-Achse,
- zur Berechnung der **Polarkoordinaten** eines Punktes:
  - Abstand zum Nullpunkt,
  - Winkel zwischen `x`-Achse und der Verbindungsgeraden des Punktes mit dem Ursprung.

Erweitern Sie die Klasse `Rechteck` um zusätzliche Methoden:

- zum Ausgeben der vier Eckpunkte des Rechtecks,
- zum Spiegeln eines Recktecks an der **`x`**- oder **`y`**-Achse,
- zum Testen, ob zwei Rechtecke gemeinsame Punkte haben.