OBJEKTORIENTIERTE PROGRAMMIERUNG

Abstrakte Klassen

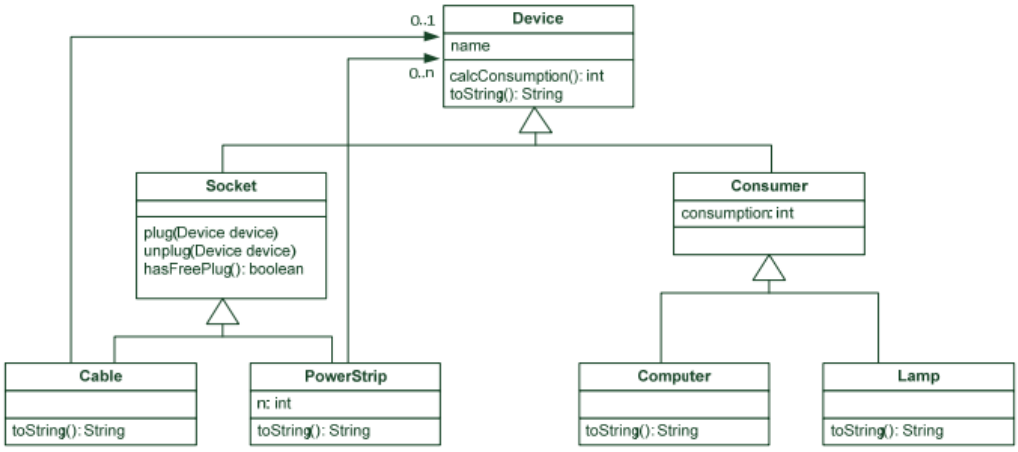
Beispiele

Stromversorgung

Unser Gebäudetechniker hat immer wieder das Problem, dass durch eine Überlast die Stromversorgung in unseren Labors zusammenbricht. Grund sind immer die Labormitarbeiter, die zu viele Geräte an einer Stromversorgung anschließen. In dieser Übung wollen wir daher ein Programm schreiben, mit der man die Belastung in einem Stromnetzwerk berechnen kann.

Das System ist auf unterschiedlichen Einheiten (Devices) aufgebaut (siehe Abbildung):

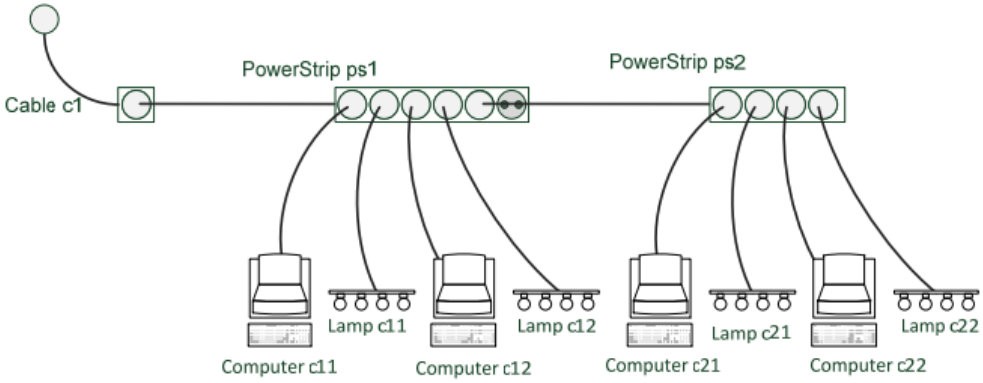
- Basisklasse des Klassensystems ist die Klasse Device. Jedes Device hat einen Namen und kann den Stromverbrauch berechnen (Methode calcConsumption).
- Die Klasse Socket ist eine Basisklasse für alle Elemente mit Steckplätzen. Sie definiert die Methoden plug(Device device) zum Anstecken von Devices, unplug(Device device) zum Abstecken und boolean hasFreePlug(), um zu prüfen, ob noch ein freier Steckplatz vorhanden ist.
- Die konkrete Unterklasse Cable stellt eine Kabel dar und kann genau ein Device anstecken.
- Die konkrete Unterklasse PowerStrip modelliert ein Verteilerkabel und hat eine bestimmte im Konstruktor eingestellte maximale Anzahl von Steckplätzen.
- Consumer ist die Basisklasse für die Verbraucher und hat eine bestimmte im Konstruktor eingestellten Stromverbrauch (consumption).
- Computer und Lamp sind konkrete Unterklassen von Consumer.



Implementieren Sie dieses Klassensystem wie folgt:

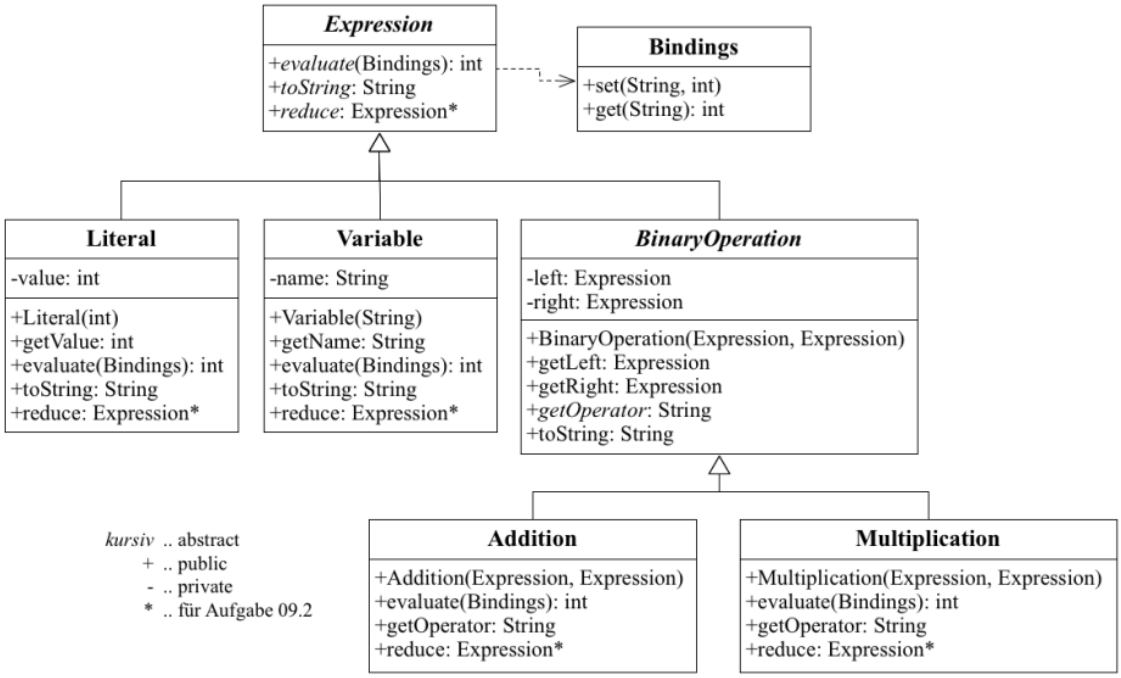
- Schreiben Sie Methoden calcConsumption, plug, unplug, hasFreePlug, etc. wie oben beschrieben.
- Überschreiben Sie die Methode toString(), um einer ansprechende textuelle Darstellung der Elemente zu erreichen.

Test:
Bauen Sie ein Netzwerk wie in der folgenden Abbildung gezeigt auf und geben Sie die Ergebnisdaten (Netzaufbau und Stromverbrauch) formatiert aus. Verwenden Sie dazu die toString-Methoden.



Arithmetische Ausdrücke

Implementieren Sie Klassen für die arithmetische Ausdrücke Literal, Variable, Addition und Multiplikation (siehe Klassendiagramm unten). Die Klasse *Expression* ist die abstrakte Basisklasse für alle Ausdrücke. Davon abgeleitet ist die Klasse *Literal* für ganze Integer-Zahlen, die Klasse *Variable* für benannte Variablen, und die Klassen *Multiplication* und *Addition*. Ausdrücke haben folgende Methoden:
- *int evaluate(Bindings bindings)* berechnet den Wert des Ausdrucks für gegebene Variablenbindungen. Implementieren Sie dazu die Hilfsklasse *Bindings* um Variablenwerte setzen und lesen zu können.
- *String toString()* liefert einen formatierten String für den Ausdruck, zB "((x+y) * (1+z))".



Folgendes Beispielprogramm erzeugt den Ausdruck "((x * 1) + 0)", gibt den formatierten String des Ausdrucks aus und gibt den mit x=3 berechneten Wert des Ausdrucks aus.

```
Multiplication m = new Multiplication(new Variable("x"), new Literal(1));
Addition a = new Addition(m, new Literal(0));

Bindings bindings = new Bindings();
bindings.set("x", 3);

Out.println(a.toString() + " = " + a.evaluate(bindings));
```

Ausgabe:
```
((x * 1) + 0) = 3
```

Erweiterung

Erweitern Sie das Programm aus Aufgabe „Arithmetische Ausdrücke“, damit die arithmetischen Ausdrücken vereinfacht werden können. Fügen Sie dazu folgende Methode hinzu:

• *Expression reduce()* wendet nachstehende Regeln an und liefert den vereinfachten Ausdruck zurück.

a + 0 = a ➔ eine Addition eines Ausdrucks a mit 0 ergibt den Ausdruck a

a * 0 = 0 ➔ eine Multiplikation eines Ausdrucks a mit 0 ergibt 0

a * 1 = a ➔ eine Multiplikation eines Ausdrucks a mit 1 ergibt den Ausdruck a

3 * 4 = 12 ➔ eine Multiplikation von Konstanten ergibt das Produkt als neue Konstante

1 + 2 = 3 ➔ eine Addition von Konstanten ergibt die Summe als neue Konstante

Wir ergänzen das Beispielprogramm aus Aufgabe „Arithmetische Ausdrücke“ wie folgt:

```
Expression a_reduced = a.reduce();
Out.println(a_reduced.toString() + " = " + a_reduced.evaluate(bindings));
```

und erhalten folgende Ausgabe mit dem vereinfachten Ausdruck:
x = 3

Zeichenprogramm – Teil 1

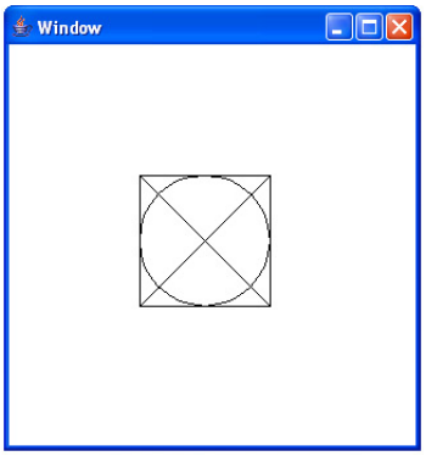
Die Klasse *Window.java* aus der Vorgabedatei zeichnet grafische Objekte in ein Fenster. Die Methode *Window.open()* öffnet das Fenster und folgende Methoden zeichnen eine Linie, ein Rechteck oder einen Kreis.:

- *drawLine*(int x1, int y1, int x2, int y2)
- *drawRectangle*(int x, int y, int w, int h)
- *drawCircle*(int x, int y, int r)

Implementieren Sie ein einfaches Zeichenprogramm, mit dem man Linien (L=*Line*), Rechtecke (R=*Rectangle*) und Kreise (C=*Circle*) zeichnen kann. Folgender Bildschirmdialog zeigt wie das Programm die Art der zu zeichnenden Figur und deren Koordinaten einliest. Die Eingabe eines Punktes beendet das Programm.

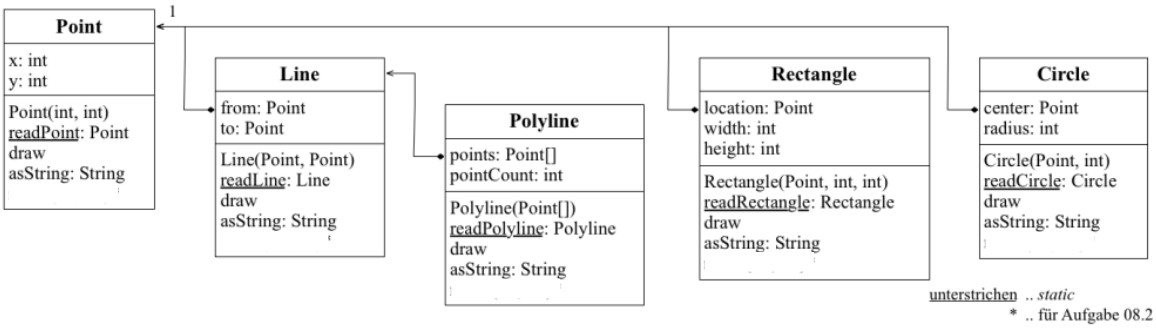
```
Figure (L|R|C): R
 Punkt (x, y): 100 100
 Breite und Hoehe (w, h): 100 100
Figure (L|R|C): C
 Punkt (x, y): 150 150
 Radius (r): 50
Figure (L|R|C): L
 Punkt (x, y): 100 200
 Punkt (x, y): 200 100
Figure (L|R|C): L
 Punkt (x, y): 100 100
 Punkt (x, y): 200 200
Figure (L|R|C): .

Rechteck (100, 100) - 100 | 100
Linie (100,200) - (200,100)
Linie (100,100) - (200,200)
Kreis (150, 150) / 50
```



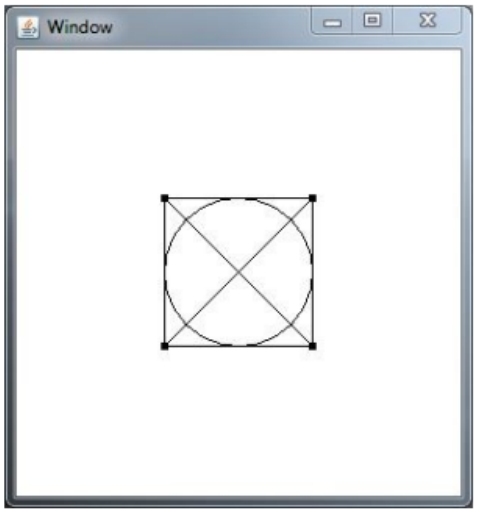
Implementieren Sie Klassen für *Point*, *Line*, *Rectangle* und *Circle*. Eine eingelesene Figur soll unmittelbar nach der Eingabe in das Fenster gezeichnet werden. Zusätzlich sollen die Figuren in Arrays vom Type *Line[ ]*, *Rectangle[ ]* und *Circle[ ]* gespeichert werden. Jedes Array soll Platz für 100 Figuren haben. Geben Sie am Programmende eine textuelle Beschreibung für alle Figuren auf der Konsole aus.

Orientieren Sie sich dabei am nachstehenden Klassendiagramm und implementieren Sie für jede Klasse:
- einen Konstruktur, zB initialisiert *Point(int x, int y)* ein Objekt der Klasse *Point*
- eine statische *read*-Methode, die mit Hilfe der *In*-Klasse ein Objekt einliest, zB erzeugt *readPoint* ein *Point*Objekt und initialisiert es mit von der Konsole gelesenen Daten
- eine *asString*-Methode, welche die Daten des Objekts als Text liefert, zB liefert *asString* der Klasse *Point* eine Zeichenkette im Format "(x=%d, y=%d)" für ein Point-Objekt



Erweiterung 1

Damit man mit Ihrem Zeichenprogramm mehr machen kann nur Kreise und Rechtecke, ergänzen Sie eine Klasse für Linienzüge (P=*Polyline*) mit den Methoden wie im Klassendiagramm dargestellt.



Der Bildschirmdialog nach der Linienzüge-Ergänzung:

```
Figure (L|R|C|P|.): R
  Eckpunkt (x, y): 100 100
  Breite und Hoehe (w, h): 100 100
Figure (L|R|C|P|.): C
  Mittelpunkt (x, y): 150 150
  Radius r: 50
Figure (L|R|C|P|.): P
 Punkt 1 (x, y): 100 100
 Punkt 2 (x, y): 200 200
 Punkt 3 (x, y): 100 200
 Punkt 4 (x, y): 200 100
 Punkt 5 (x, y): -1 -1
Figure (L|R|C|P|.): .


 Alle Figuren:
 - Rechteck (100,100) - 100 | 100
 - Kreis (150,150) / 50
 - Polylinie {
     (100,100),
     (200,200),
     (100,200),
     (200,100)
   }
```

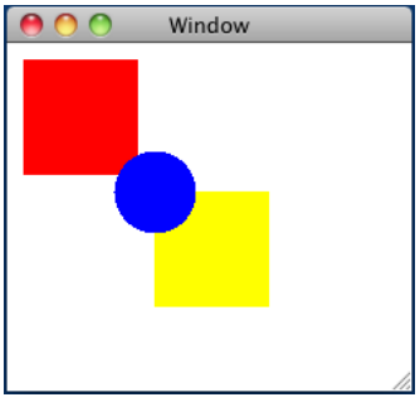
Implementierungshinweise

- Zeichnen Sie die graphischen Objekte jeweils in der *draw*-Methode mit Hilfe der Klasse *Window.java*.
- Zeichnen Sie alle Punkte als gefüllte Quadrate mit der Methode *Window.fillRectangle(int x, int y, int w, int h, java.awt.Color color)* und einer Seitenlänge von 5 Pixeln.
- Die *move*-Methode implementieren Sie erst in der nächsten Aufgabe

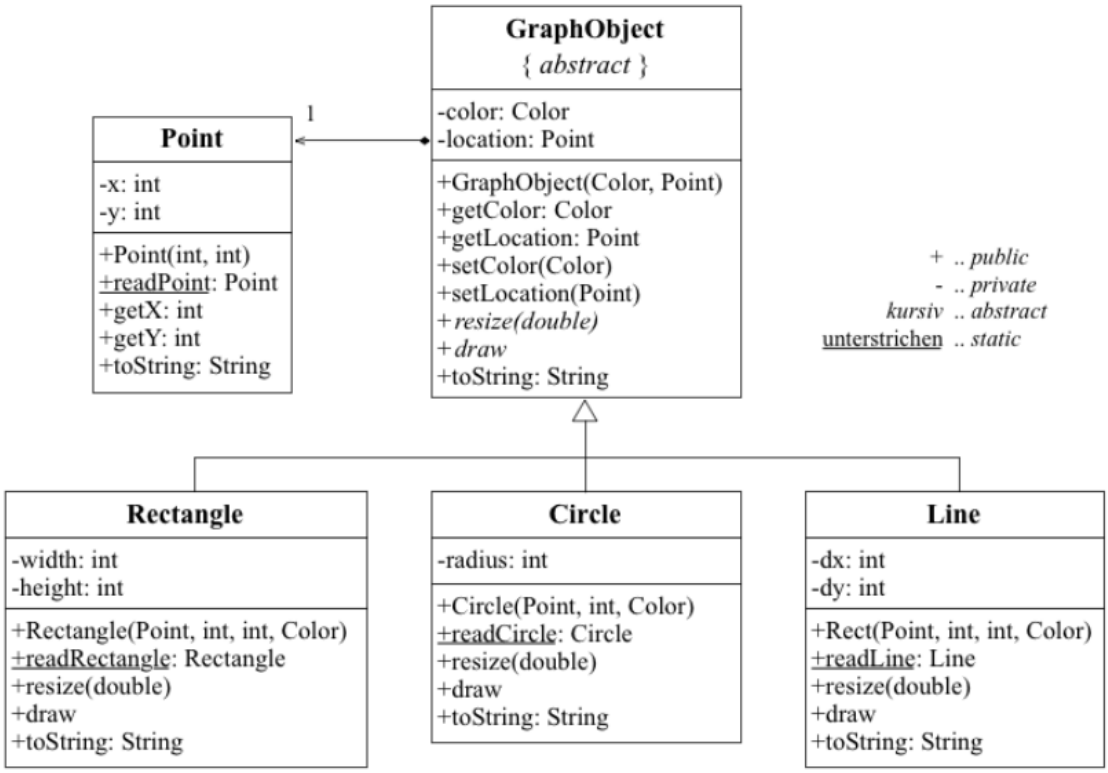
Zeichenprogramm – Teil 2

Verbessern Sie das in Aufgabe „Zeichenprogramm" beschriebene Zeichenprogramm!

```
Figure:  [L]ine, [R]ect, [C]ircle: R
Location (x, y): 10 10
Width: 70
Height: 70
Color (R G B): 255 0 0
Figure:  [L]ine, [R]ect, [C]ircle: R
Location (x, y): 90 90
Width: 70
Height: 70
Color (R G B): 255 255 0
Figure:  [L]ine, [R]ect, [C]ircle: C
Location (x, y): 90 90
Radius: 25
Color (R G B): 0 0 255
Figure:  [L]ine, [R]ect, [C]ircle .
All Figures:
 - Rectangle (10 10) - 70 | 70 - RGB=(255 0 0))
 - Rectangle (90 90) - 70 | 70 - RGB=(255 255 0))
 - Circle (90 90) - 25 - RGB=(0 0 255)
```



Führen Sie eine abstrakte Basisklasse *GraphObject* ein und leiten Sie Unterklassen *Rectangle*, *Circle* und *Line* ab. Ergänzen Sie für alle graphischen Objekte eine Farbe und verwenden Sie zum Zeichnen die Methoden für ausgefüllte Objekte, zB *fillRectangle* statt *drawRectangle*. Orientieren Sie Ihre Programmstruktur am Klassendiagramm und führen Sie bei Bedarf weitere Methoden ein.
Implementieren Sie für die Klassen *Rectangle*, *Circle* und *Line*:
- einen Konstruktor, zB initialisiert *Circle(Point p, int radius, Color c)* ein Objekt der Klasse *Circle*
- eine statische *read*-Methode, die mit Hilfe der *In*-Klasse ein Objekt einliest; zB liest *readCircle* ein *Circle*-Objekt
- eine *toString*-Methode, die den Inhalt eines Objekts als Text liefert; zB liefert *toString* der Klasse *Circle* eine Zeichenkette für ein *Circle*-Objekt
Circle (100 100) - 80 - RGB=(0 0 255)
- eine *draw*-Methode, die mit Hilfe der *Window*-Klasse die Figur zeichnet
- eine *resize*-Methode, um die Grösse der Figur zu ändern; ein Faktor größer 1 vergrößert die Figur, ein Faktor kleiner 1 verkleinert die Figur



Überarbeiten Sie auch Ihr Hauptprogramm. Dort wo sie bisher drei Arrays vom Typ *Line[ ], Rectangle[ ]* und *Circle[ ]* verwendet haben um die Figuren zu speichern, verwenden Sie ein einziges Array vom Typ *GraphObject[ ]*.