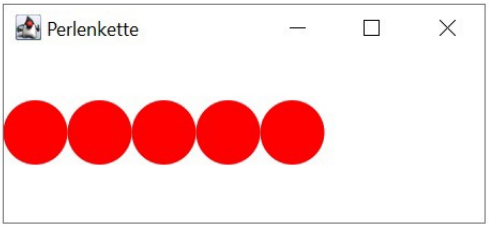
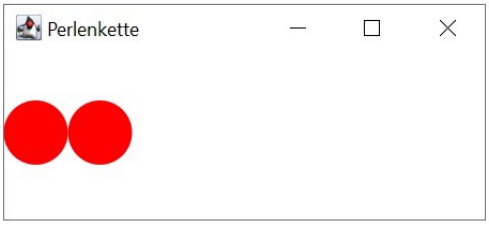
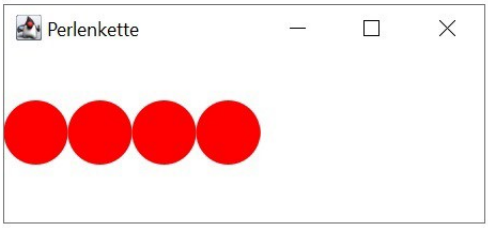
9 Umsetzung von 1:* Beziehungen: Arrays (Felder)

In diesem Abschnitt soll nun die Umsetzung der 1:* Beziehung genauer betrachtet werden. Dafür soll mithilfe der Klasse `Kreis` eine Perlenkette aus roten Kreisen gezeichnet werden:





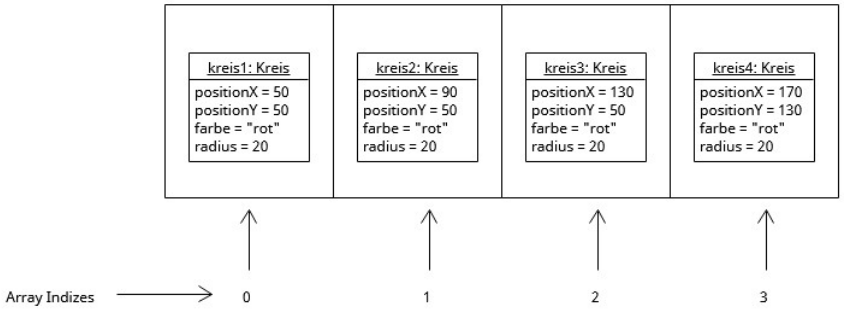


Ein erstes Klassendiagramm könnte dabei folgendermaßen aussehen:

Damit verschiedene Objekte der Klasse `Perlenkette` unterschiedlich viele Objekte der Klasse `Kreis` speichern können, benötigt man für die Deklaration des Referenzattribut ein so genanntes **Array (Feld)**:

```
public class Perlenkette{
  private Kreis[] kette;
}
```

Ein Array kann man sich dabei wie eine Art „Schubladensystem“vorstellen, in dessen einzelnen Schubladen ein Element gespeichert werden kann. Der Zugriff erfolgt dabei über einen so genannten Index:



Bevor man nun das Array mit Elementen befüllen kann, muss es noch initialisiert werden. Dies ist allgemein durch folgende Anweisung möglich:

arrayName = new DatentyDesArrays[gewünschteLänge];

Nach einer Initialisierung kann dann das Array dann mit Elementen befüllt werden. Dafür ist häufig die Verwendung einer `for`-Schleife hilfreich. Dies könnte am Beispiel der Klasse `Perlenkette` im Konstruktor wie folgt aussehen:

```
public class Perlenkette{
  private Kreis[] kette;

  public Perlenkette(int laenge, String farbe){
    kette = new Kreis[laenge];
    final int radius = 20;
    final int posY = 50;
    for(int i = 0; i<laenge;i++){
      kette[i] = new Kreis(radius + 2*i*radius, posY, radius, farbe);
    }
  }
}
```

Aufgaben:

1. Erweitern Sie im Folgenden die Klasse `Perlenkette` um weitere Methoden.
   (a) Einfache Erweiterungen
      i. `public void farbeSetzen(int index, String farbe):`
      Beim Kreis des übergebenen Index soll der Farbwert entsprechend gesetzt werden. Werden ungültige Werte für den Index übergeben, soll eine entsprechende Meldung auf der Konsole ausgegeben werden.

      ii. `public void zeichne(Leinwand l):`
      Alle Kreise sollen auf der übergebenen Leinwand gezeichnet werden.

      iii. `public void farbeSetzen(String farbe):`
      Alle Kreise sollen den übergebenen Farbwert besitzen.
      Warum nennt man die Methode `farbeSetzen` nun **überladen**?
      Hinweis für Experten: Sie können für die Schleife auch eine so genannte for-each-Schleife verwenden.

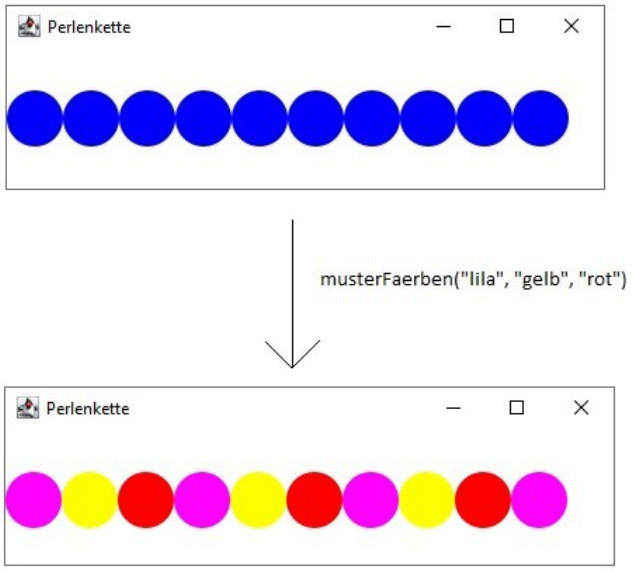
iv. `public int laengePerlenkette()`,
`public int groessterPerlenRadius()`:
Da es im nächsten Aufgabenblock möglich sein soll, den Radius einzelner Kreise zu ändern, soll die Methode `laengePerlenkette` die Länge der Perlenkette (entspricht der Summe aller Kreisdurchmesser) berechnen und zurückgeben. Die Methode `groessterPerlenradius` soll dabei den größten Radius eines Kreises in der Perlenkette ermitteln und zurückgeben.

(b) Komplexere Erweiterungen

i. `public void positionSetzen(int xPos, int yPos)`:
Die Position des ersten Kreises soll mit dieser Methode neu gesetzt werden können. Alle anderen Kreise sollen anschließend ebenfalls verschoben werden, so dass sich alle Kreise wieder berühren.

ii. `public void radiusSetzen(int index, int neuerRadius)`:
Am übergebenem Index soll der Radius des Kreises auf den neuen Radius gesetzt werden. Anschließend sollen sich alle Kreise wieder berühren. Bei ungültigen Werten für Index oder Radius soll eine entsprechende Konsolenausgabe erfolgen.

iii. `public void farbeSetzen(String... muster):`
Alle Kreise sollen gemäß des Musters eingefärbt werden. Informieren Sie sich, wofür die Schreibweise `String...` steht. Ein beispielhafter Aufruf soll dabei folgendes bewirken:



(c) Erweiterungen für Experten:

i. `public void perlenTauschen(int index1, int index2):`
Die beiden Kreise an den Positionen `index1` und `index2` sollen deren Position tauschen. Achten Sie darauf, dass sich danach wieder alle Kreise berühren. Wird ein ungültiger Index übergeben, so soll eine entsprechende Meldung auf der Konsole ausgegeben werden.

ii. `public void spiegeln():`
Der Kreis an der ersten Position soll mit dem Kreis an der letzten Position getauscht werden, der zweite Kreis mit dem vorletzten Kreis,... usw.

iii. `public void anordnen(Anordnung a)`:
Erstellen Sie eine entsprechende Enum `Anordnung`, in der sie jeweils einen Eintrag für eine horizontale, vertikale und diagonale Anordnung speichern. Nach dem Aufruf mit einem entsprechenden Parameter sollen sich die Kreise horizontal, vertikal bzw. diagonal (im 45° Winkel ausgehend von der linken oberen Ecke) anordnen.

2. Für Experten: Zweidimensionale Arrays

   (a) Informieren Sie sich über den Aufbau von so genannten zweidimensionalen Arrays .

   (b) Für die Temperaturüberwachung eines Rechenzentrums wird ein zweidimensionales Array mit den Temperaturen der Server genutzt.
   Für jeden Server gibt es eine Spalte mit den Werten der Temperatursensoren. Wenn drei dieser Temperaturen den Wert von 40 °Celsius überschreiten soll eine Ausgabe der Servernummer erfolgen. Die Servernummer steht in der ersten Zeile des Temperaturarrays.
   Beispielaufbau des Temperatur Arrays:
   `temperaturArray[][]`

2586	**2588**	**2544**	**2576**	**2596**	**2583**	**2526**	**2593**	**2555**
23	23	35	32	35	43	26	54	25
34	23	43	23	25	23	35	43	43
21	35	43	36	43	43	42	34	34
36	27	41	43	23	32	38	39	32
23	28	36	41	25	41	39	27	32
18	36	21	23	28	31	42	45	35
43	27	35	25	36	37	43	45	43

   Dazugehörige Ausgabe:

```
Temperaturalarm ServerNr.: 2544
Temperaturalarm ServerNr.: 2526
Temperaturalarm ServerNr.: 2593
```

   Vervollständigen Sie folgendes Java-Programm, so dass die oben beschriebene Funktionalität umgesetzt

```
public class ServerTemperatur {
   public static void main(String[] args) {
      int[][] temperaturArray ={
        {2586, 2588, 2544, 2576, 2596, 2583, 2526, 2593, 2555},
        {23,   23,    35,   32,   35,   43,   26,   54,   25},
        {34,   23,    43,   23,   25,   23,   35,   43,   43},
        {21,   35,    43,   36,   43,   43,   42,   34,   34},
        {36,   27,    41,   43,   23,   32,   38,   39,   32},
        {23,   28,    36,   41,   25,   41,   39,   27,   32},
        {18,   36,    21,   23,   28,   31,   42,   45,   35},
        {43,   27,    35,   25,   36,   37,   43,   45,   43}};
      // Hier mit dem Programm beginnen:




   }
}
```

3. Für Experten: Call by value/Call by Reference
   Gegeben ist folgende Klasse **MyInt**:

```
public class MyInt{
  private int i;

  public MyInt(int i){ this.i = i; }

  public int getI(){ return i; }

  public void setI(int i){ this.i = i; }
}
```

   Diese Klasse wird in der Klasse **Verweise** verwendet. Geben Sie mithilfe der Tabelle die Werte von **x**, **y[0]** und **z.getI()** nachdem die angegebenen Zeilen in der Tabelle ausgeführt wurden.

```
1  public class Verweise{
2
3    public static void main(String[] args){
4      int x = 23;
5      int[] y = {23};
6      MyInt z = new MyInt(23);
7
8      x *= 3;
9      y[0] *= 3;
10     z = new MyInt(z.getI() * 3);
11     foo(x, y, z);
12
13   }
14
15   public static void foo(int x, int[] y, MyInt z){
16     x += 27;
17     y[0] += 27;
18     z.setI(z.getI() + 27);
19     bar(x, y, z);
20   }
21
22   public static int bar(int x, int[] y, MyInt z){
23     x /= 3;
24     y[0] /= 3;
25     z = new MyInt(z.getI() / 3);
26     return x;
27   }
28 }
```

Zeile	x	y[0]	z.getI()
4	23	-	-
5	23	23	-
6	23	23	23
8	69	23	23
9			
10			
16			
17			
18			
23			
24			
25			
19			
11			