8 Charts erstellen

In diesem Kapitel werden wir Charts erstellen. Während die Standard-Swing-API noch über keine Chart-Komponente verfügt, hat das Entwicklerteam JavaFX eine sehr reichhaltige Standardbibliothek spendiert. Charts sind dabei nichts anderes als spezialisierte Nodes, die dem Layout hinzugefügt werden. Momentan kann man aus neun verschiedenen Charts auswählen.

Sie werden in diesem Kapitel lernen, wie man einfache Charts erzeugt, welche verschiedenen Chart-Typen zur Verfügung stehen, wie Sie das Erscheinungsbild anpassen können, wie Charts kontinuierlich dynamische Daten darstellen können und sogar wie man Charts erweitern kann, um eigene Animationen einzufügen.

8.1 Ein Diagramm anzeigen

Zeigen wir zunächst einmal ein einfaches `LineChart` an. Anschließend werden wir uns den Code etwas genauer ansehen:

```
NumberAxis xAxis = new NumberAxis("Zeit", 0, 40, 5);
NumberAxis yAxis = new NumberAxis("Preis", 0, 80, 10);
ObservableList<XYChart.Series<Double,Double>> lineChartData =
FXCollections.observableArrayList(
   new LineChart.Series<Double,Double>("Orakel",
            FXCollections.observableArrayList(
     new XYChart.Data<Double,Double>(0.0, 10.0),
     new XYChart.Data<Double,Double>(10.0, 34.4),
     new XYChart.Data<Double,Double>(20.0, 31.9),
     new XYChart.Data<Double,Double>(30.0, 42.3),
     new XYChart.Data<Double,Double>(40.0, 57.7)
   )),
   new LineChart.Series<Double,Double>("Essape",
             FXCollections.observableArrayList(
     new XYChart.Data<Double,Double>(0.0, 57.7),
     new XYChart.Data<Double,Double>(10.0, 42.3),
     new XYChart.Data<Double,Double>(20.0, 33.9),
```

```
        new XYChart.Data<Double,Double>(30.0, 21.7),
        new XYChart.Data<Double,Double>(40.0, 17.3)
      ))
);
LineChart chart = new LineChart(xAxis, yAxis, lineChartData);
```

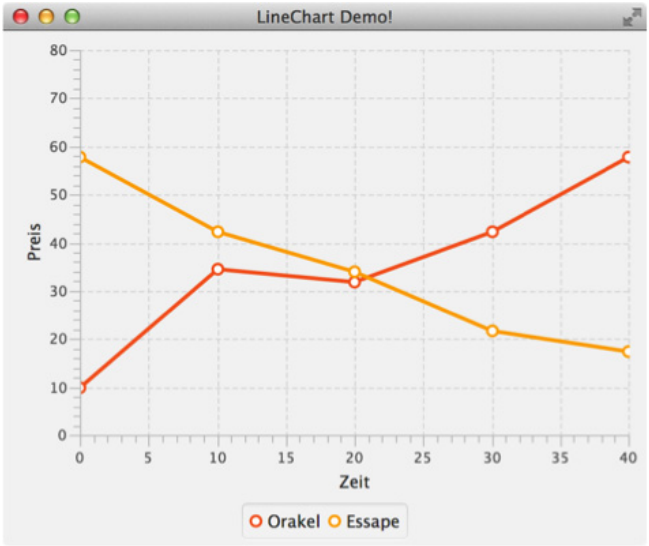

Abb. 8–1 *Ein einfaches LineChart*

Am Anfang definieren wir die beiden Achsen jeweils als `NumberAxis`, wobei wir dem Konstruktor Beschriftung, Ober- und Untergrenze sowie den Abstand der Markierungsbezeichner übergeben. Die einzelnen Datenpunkte einer Datenreihe werden dann jeweils in ein `XYChart.Series` gepackt. Anschließend werden diese beiden Reihen in eine `ObservableList` gesteckt. Diese dient dann als Datenmodell für das `LineChart`.

8.2 Aufbau der API

Alle Charts leiten von der Basisklasse `Chart` ab, die vor allem für das Zeichnen von Hintergrund, Legende und Titel verantwortlich ist. `XYCharts` erweitern diese Klasse, um die Fähigkeit, zwei Achsen darzustellen. Mit Ausnahme des `PieChart`, das keine Achsen benötigt und nur eindimensionale Daten darstellt, leiten alle anderen Klassen davon ab.

8.2.1 Achsen

Für die `XYCharts` werden eine x- und eine y-Achse definiert. Wir werden dabei vor allem mit der `NumberAxis` zu tun haben, deren Daten als Zahlen definiert sind, und der `CategoryAxis`, deren Wertzuordnung anhand von Strings erfolgt. Jeder String,

den wir der `CategoryAxis` übergeben, definiert dabei einen Markierungspunkt auf der Achse. In unserem ersten Beispiel verwenden wir nur `NumberAxis`. Wir haben diesen im Konstruktor einen Titel, Minimalwert, Maximalwert und den Abstand der Markierungsstriche (Tickmarken) übergeben. Verwenden wir stattdessen den parameterlosen Konstruktor und setzen keine Minimal- und Maximalwerte, so ermittelt die Achse diese selbst per »Autoranging« anhand der Daten. In unserem Beispiel wäre das:

```
NumberAxis xAxis = new NumberAxis();
NumberAxis yAxis = new NumberAxis();
```

Die einzelnen Properties der Achse lassen sich dann einfach über Setter-Methoden konfigurieren:

```
xAxis.setLabel("Zeit");
xAxis.setTickUnit(5);
```

Zusätzlich können wir auch noch angeben, dass trotz Autoranging die Null auf jeden Fall angezeigt werden soll:

```
xAxis.setForceZeroInRange(true);
```

Wen später neue Werte hinzukommen, die außerhalb des durch Autoranging bestimmten Bereichs liegen, passen sich die Achsen automatisch an. Dabei wird der Übergang animiert dargestellt, sodass der Benutzer die Veränderung leichter nachvollziehen kann. Diese Animation lässt sich jedoch auch abstellen. Um das zu testen, können Sie Ihrem Design einen Button hinzufügen, der einen neuen Wert außerhalb des bisherigen Wertebereichs einfügt:

```
Button button = new Button("Wert hinzufügen");
button.setOnAction(new EventHandler<ActionEvent>() {

   @Override
   public void handle(ActionEvent event) {
     lineChartData.get(0).getData().add(new XYChart.Data<Double,
       Double>(100.0, 17.3));
   }
});
```

Testen Sie nun das Verhalten bei Knopfdruck und deaktivieren Sie dann die Animation:

```
chart.setAnimated(false);
```

Vor allem wenn wir Livedaten betrachten, die sich häufig ändern, wie etwa einen Aktienkurs, bietet es sich an, diese Animation abzuschalten, da sie störend wirken kann.

Manche Chart-Typen, wie zum Beispiel die `BarCharts`, erwarten eine `ValueAxis` kombiniert mit einer `CategoryAxis`. Das ist erst mal verwirrend, weil egal ist, welche der Achsen von welchem Typ ist. Dadurch kann der Compiler einen Feh-

ler nicht erkennen, und eine falsche Kombination, wie zum Beispiel zwei numerische Achsen, wird erst zur Laufzeit in Form einer Exception gemeldet.

8.2.2 Daten

Die Daten eines `XYChart` sind in `XYChart.Series` organisiert. Das `XYChart` verwaltet die Series in einer `ObservableList`. Die Daten einer Serie oder Datenreihe sind in der Darstellung meist einheitlich gekennzeichnet, sodass die Zusammengehörigkeit erkennbar ist. In der Serie stecken dann die einzelnen Datenpunkte in Form von `XYChart.Data`. Wie der Name nahelegt, können wir einen x-Wert und einen y-Wert definieren. Ein zusätzlicher »extraValue« wird in `BubbleChart` verwendet, um die Größe der Blase anzugeben.

Neben den Daten kann dem Datenpunkt auch ein Node mitgegeben werden. Dieser Node wird dann in der Darstellung anstatt des diagrammspezifischen Standard-Nodes benutzt.

8.3 Welche Diagrammtypen gibt es?

Momentan gibt es sieben verschiedene Diagrammtypen, die jedoch um eigene Typen erweitert werden können: `LineChart`, `PieChart`, `AreaChart`, `StackedAreaChart`, `BarChart`, `BubbleChart` und `ScatterChart`.

8.3.1 LineChart und AreaChart

Das `LineChart` haben wir gerade in unserem ersten Beispiel verwendet. Es verbindet Daten einer Serie mit einer Linie. So können Trends leicht nachvollzogen werden. Eine klassische Anwendung für diesen Diagrammtyp ist das Aktien-Chart. `LineCharts` haben eine x- und eine y-Achse. In unserem Beispiel haben wir eine `NumberAxis` verwendet. Mit einer `CategoryAxis` können Sie statt der Zahlen andere Werte als »Tickmarken« anzeigen. Tauschen Sie dazu die x-Achse im Beispiel aus:

```
CategoryAxis xAxis = new CategoryAxis();
NumberAxis yAxis = new NumberAxis("Preis", 0, 80, 10);
ObservableList<XYChart.Series<String,Double>> lineChartData =
FXCollections.observableArrayList(
   new LineChart.Series<String,Double>("Orakel",
      FXCollections.observableArrayList(
     new XYChart.Data<String,Double>("Jan", 10.0),
     new XYChart.Data<String,Double>("Feb", 34.4),
     new XYChart.Data<String,Double>("Mär", 31.9),
     new XYChart.Data<String,Double>("Apr", 42.3),
     new XYChart.Data<String,Double>("Jun", 57.7)
   )),//... Analog für die zweite Datenreihe
);
```

Wie Sie sehen, müssen wir noch die Datentypen im Datenmodell anpassen, dann übernimmt das Chart die Bezeichnungen für die Tickmarken aus dem Datenmodell.

Das `AreaChart` unterscheidet sich von dem `LineChart` dadurch, dass die Flächen zwischen x-Achse und den Verbindungslinien zwischen den Datenpunkten einer Zeitreihe farbig ausgefüllt sind. Man unterscheidet zwischen gestapelten und überlappenden Flächendiagrammen. Das überlappende Flächendiagramm dient demselben Zweck wie das Liniendiagramm und macht es einfacher, Trends zu verfolgen, da zusammengehörige Datenpunkte durch eine gleichmäßig eingefärbte Fläche dargestellt werden. Löschen wir in unserem Beispiel die Zeile, in der das LineChart aufgebaut wird:

```
LineChart chart = new LineChart(xAxis, yAxis, lineChartData);
```

Danach ersetzen wir diese durch ein `AreaChart`:

```
AreaChart chart = new AreaChart(xAxis, yAxis, lineChartData);
```

So erhalten wir ein überlappendes Flächendiagramm.

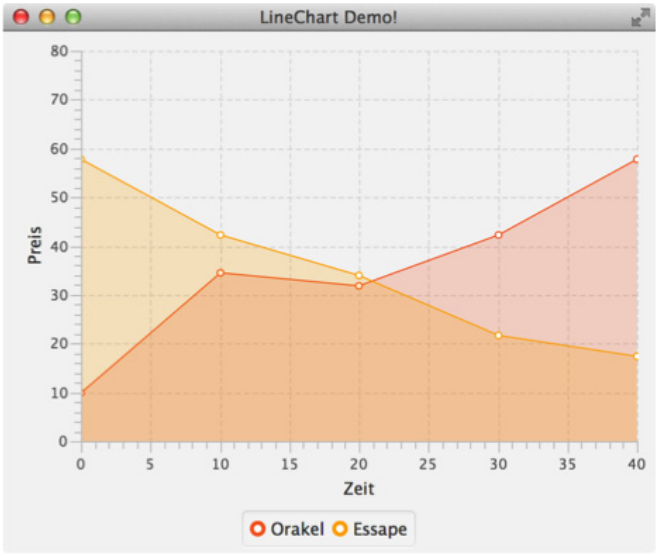

Abb. 8–2 *Die gleichen Daten wie in Abbildung 8–1, jetzt als AreaChart dargestellt*

Der Verlauf der Linien ist gleich, lediglich die Flächen darunter sind nun mit transparenter Farbe ausgefüllt. Wo sich die Flächen überlappen, entsteht ein Mischton. Um ein gestapeltes Diagramm zu erhalten, verwenden wir stattdessen folgenden Code:

```
StackedAreaChart chart = new StackedAreaChart(xAxis, yAxis, lineChartData);
```

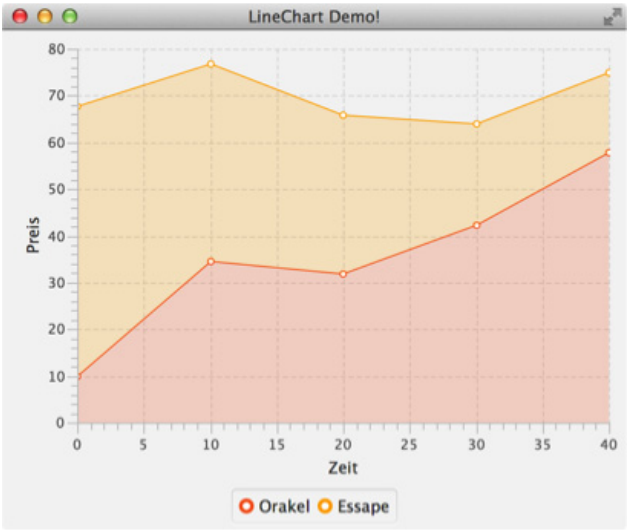

Abb. 8–3 *Das StackedAreaChart addiert die Werte.*

Die Datenkurve für unsere fiktive Firma »Orakel« ist gleich geblieben. Die Datenwerte für die Firma »Essape« haben sich jedoch verändert. Der y-Wert der einzelnen Datenpunkte ist nun nicht mehr der übergebene Wert. Stattdessen wurde der übergebene Wert zum Wert der zuvor hinzugefügten Kurve hinzuaddiert. Zum Vergleich unserer »Aktienkurse« ist dieser Diagrammtyp damit ungeeignet. Man kann ihn hingegen sehr gut dafür verwenden, um die zeitliche Entwicklung der Zusammensetzung einer Gesamtmenge zu verdeutlichen. Die oberste Linie repräsentiert die kumulierten Werte der einzelnen Datenpunkte, wie in folgendem Beispiel, das fiktive Verkaufszahlen darstellt:

```
NumberAxis xAxis = new NumberAxis("Zeit", 1981, 1985, 1);
NumberAxis yAxis = new NumberAxis("Verkaufte Einheiten", 0, 160, 10);
yAxis.setTickLabelFormatter(new
NumberAxis.DefaultFormatter(yAxis,null,"k")));
ObservableList<XYChart.Series<Integer, Double>> lineChartData =
      FXCollections.observableArrayList(
    new LineChart.Series<Integer, Double>("C64",
         FXCollections.observableArrayList(
      new XYChart.Data<Integer, Double>(1981, 0.0),
      new XYChart.Data<Integer, Double>(1982, 10.0),
      new XYChart.Data<Integer, Double>(1983, 34.4),
      new XYChart.Data<Integer, Double>(1984, 31.9),
      new XYChart.Data<Integer, Double>(1985, 42.3)
    )),
    new LineChart.Series<Integer, Double>("ZX Spectrum",
         FXCollections.observableArrayList(
      new XYChart.Data<Integer, Double>(1981, 0.0),
      new XYChart.Data<Integer, Double>(1982, 57.7),
      new XYChart.Data<Integer, Double>(1983, 42.3),
```

```
    new XYChart.Data<Integer, Double>(1984, 33.9),
    new XYChart.Data<Integer, Double>(1985, 21.7)
  ))
  // weitere Daten...
  }
);
StackedAreaChart chart = new StackedAreaChart(xAxis, yAxis,
                    lineChartData);
chart.setTitle("Heimcomputer-Verkaufszahlen in Deutschland");
```

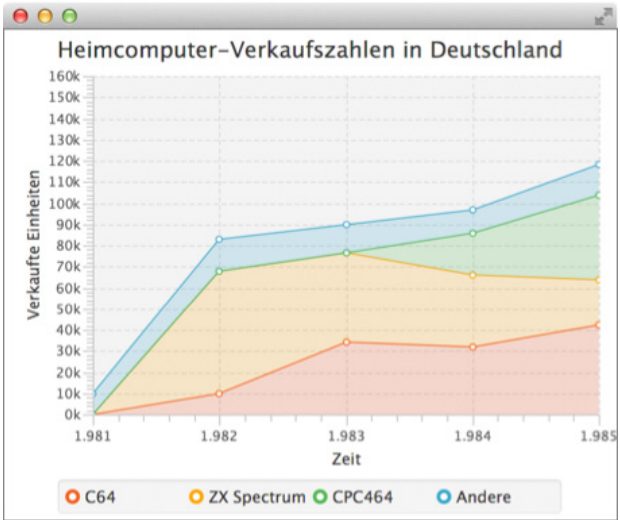

Abb. 8–4 *Das StackedAreaChart verdeutlicht die Zusammensetzung einer Gesamtmenge.*

In diesem Diagrammtyp lassen sich also gleichzeitig die Gesamtentwicklung der Verkaufszahlen aller Geräte und der Beitrag der einzelnen Modelle darstellen. Mit `setTitle` haben wir dem Chart zusätzlich eine Überschrift gegeben, und mittels eines `StringConverter` wird dem rohen y-Wert ein »k« angehängt, um zu verdeutlichen, dass es sich um Tausender handeln soll. Dazu bringt die Klasse `NumberAxis` praktischerweise einen DefaultFormatter mit, dem wir ein Präfix und ein Suffix übergeben können.

8.3.2 PieChart

`PieCharts` oder Tortendiagramme werden meist verwendet, um den prozentualen Anteil einer Kategorie an einer Gesamtmenge zu verdeutlichen. Sie haben im Unterschied zu `LineChart` und `AreaChart` nur eine Dimension und keine Achsen. Daher sieht hier der Code ein wenig anders aus:

```
PieChart pieChart = new PieChart(FXCollections.observableArrayList(
   new PieChart.Data("Java", 19),
   new PieChart.Data("C", 16),
   new PieChart.Data("C++", 9),
   new PieChart.Data("PHP", 7),
   new PieChart.Data("C#", 6),
   new PieChart.Data("Visual Basic", 5),
   new PieChart.Data("Python", 5),
   new PieChart.Data("Other", 33)
));
pieChart.setId("BasicPie");
pieChart.setTitle("TIOBE Index");
```

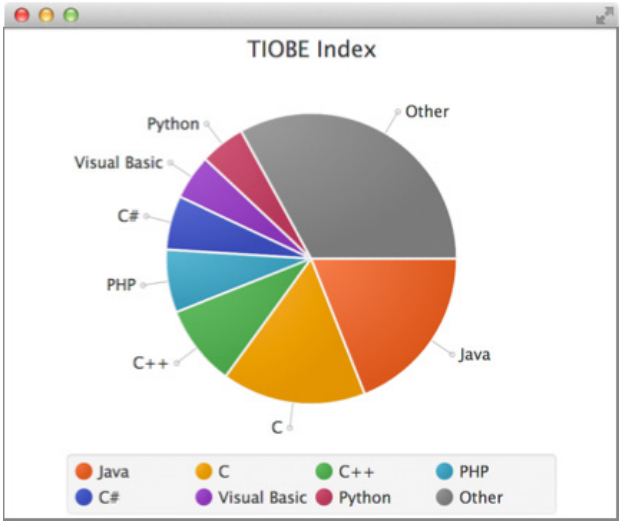

Abb. 8–5 *Tortendiagramme werden zur Anzeige eindimensionaler Daten verwendet.*

Statt der `XYChart.Data` für zweidimensionale Daten werden `PieChart.Data` verwendet, um jeweils Name und Wert eines Datensatzes festzulegen.

8.3.3 BarChart

Balken- und Säulendiagramme werden mithilfe der Klasse `BarChart` erstellt. Sie werden verwendet, um Daten verschiedener Kategorien zu vergleichen. Die Klasse erwartet eine `Number` oder `ValueAxis` kombiniert mit einer `CategoryAxis`. Übergibt man im Konstruktor zum Beispiel zwei `NumberAxis`, so bekommt man zur Laufzeit eine Exception. `BarCharts` eignen sich am besten dann, wenn nur wenige Datenpunkte von eindimensionalen oder zweidimensionalen Werten dargestellt werden sollen.

Für die Erstellung eines eindimensionalen `BarChart` übergeben wir einfach der `CategoryAxis` nur einen einzigen Wert:

```
CategoryAxis xAxis = new CategoryAxis();
NumberAxis yAxis = new NumberAxis();
BarChart<String,Number> bc = new BarChart<String,Number>(xAxis,yAxis);
xAxis.setCategories(FXCollections.<String>observableArrayList(""));
xAxis.setTickMarkVisible(false);
yAxis.setLabel("Performance");
XYChart.Series<String,Number> series1 = new XYChart
   .Series<String,Number>("JavaFX",FXCollections.
    observableArrayList(new XYChart.Data<String,Number>("",
      956)));
XYChart.Series<String,Number> series2 = new XYChart
   .Series<String,Number>("Swing",
    FXCollections.observableArrayList(new XYChart
      .Data<String,Number>("", 789)));
XYChart.Series<String,Number> series3 = new
XYChart.Series<String,Number>("JavaScript",
   FXCollections.observableArrayList(new XYChart
    .Data<String,Number>("", 312)));
```

Die Darstellung der Tickmarke ergibt an dieser Stelle wenig Sinn, deshalb stellen wir sie mit `setTickMarksVisible(false)` ab.

Zweidimensionale `BarCharts` sind vor allem dann gefragt, wenn die Daten der einzelnen Erhebungspunkte keine Zeitreihe darstellen und die Verbindung der Datenpunkte wie etwa in einem `LineChart` keinen Sinn ergeben würde. So werden sie zum Beispiel gerne bei Performance-Vergleichen verwendet, wenn die Ergebnisse mehrerer unabhängiger Tests in einem einzelnen Diagramm dargestellt werden sollen. Wir können unsere fiktiven Performance-Testdaten ganz einfach erweitern, indem wir der `CategoryAxis` mehrere Kategorien übergeben und dafür jeweils Datenpunkte erzeugen:

```
final CategoryAxis xAxis = new CategoryAxis();
final NumberAxis yAxis = new NumberAxis();
final BarChart<String, Number> bc = new BarChart<String, Number>
    (xAxis, yAxis);
yAxis.setTickLabelFormatter(new NumberAxis.DefaultFormatter(yAxis,
    null, " parsecs"));
xAxis.setCategories(FXCollections.<String>observableArrayList(
    "BubbleTrouble", "Kessel-Run"));
xAxis.setTickMarkVisible(false);
yAxis.setLabel("Performance (less is better)");
ObservableList<XYChart.Series<String, Number>> data =
   FXCollections.observableArrayList(
    new XYChart.Series<String, Number>("JavaFX",
      FXCollections.observableArrayList(
         new XYChart.Data<String, Number>("BubbleTrouble", 17, 9),
         new XYChart.Data<String, Number>("Kessel-Run", 11, 87))),
    new XYChart.Series<String, Number>("Swing",
      FXCollections.observableArrayList(
         new XYChart.Data<String, Number>("BubbleTrouble", 29, 1),
         new XYChart.Data<String, Number>("Kessel-Run", 33, 2))),
```

```
    new XYChart.Series<String, Number>("JavaScript",
      FXCollections.observableArrayList(
          new XYChart.Data<String, Number>("BubbleTrouble", 100),
          new XYChart.Data<String, Number>("Kessel-Run", 112))));
bc.setData(data);
```

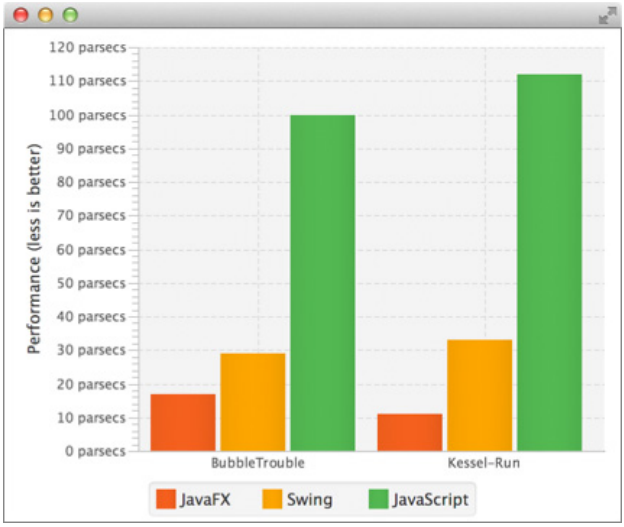

Abb. 8–6 *Ein BarChart mit zwei Kategorien*

8.3.4 ScatterChart

Streudiagramme oder `ScatterCharts` stellen Datenpunkte einer `XYChart.Series` in einer Punktwolke an der angegebenen xy-Position dar. Jeder Serie wird dabei zusätzlich ein Symbol zugewiesen, sodass die Zugehörigkeit eines Datenpunkts zu einer Serie leichter zu erkennen ist. Dieser Diagrammtyp lässt sich zur Korrelationsanalyse nutzen, um zum Beispiel Cluster zu erkennen.

```
NumberAxis xAxis = new NumberAxis();
NumberAxis yAxis = new NumberAxis();
ScatterChart<Number, Number> sc =
    new ScatterChart<Number, Number>(xAxis, yAxis);
xAxis.setLabel("X Axis");
yAxis.setLabel("Y Axis");
for (int s = 1; s < 5; s++) {
   XYChart.Series<Number, Number> series =
     new XYChart.Series<Number, Number>();
   series.setName("Data Series " + s);
   for (int i = 0; i < 30; i++) {
     series.getData().add(
       new XYChart.Data<Number, Number>(
          s % 2 == 0 ? 50 +(Math.random() * 98) / s :
             (Math.random() * 98)/ s,
```

```
            s % 2 == 0 ? 50 +(Math.random() * 98) / s :
                (Math.random() * 98)/ s));
    }
    sc.getData().add(series);
```

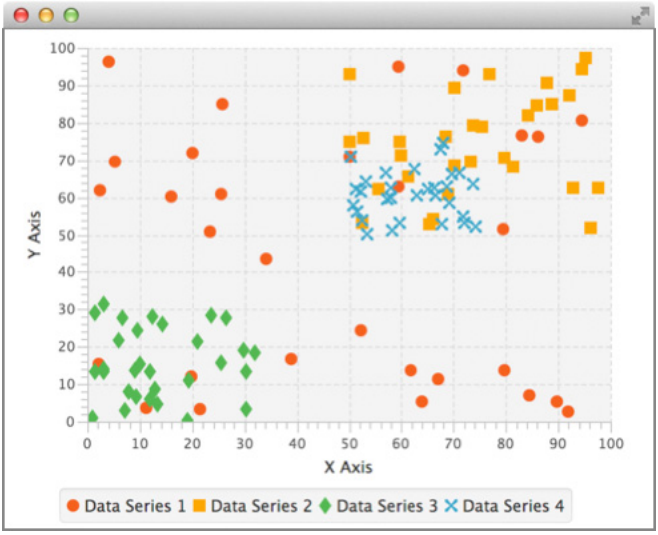

Abb. 8–7 *In ScatterCharts lassen sich gut Cluster erkennen.*

8.3.5 BubbleChart

Mithilfe von Blasendiagrammen lassen sich in einem zweidimensionalen Diagramm drei abhängige Merkmale darstellen. Die dritte Dimension wird dabei durch die Größe des Datenpunktes abgebildet. Der Wert wird als »extraValue« einfach dem `XYData`-Objekt im Konstruktor übergeben:

```
double SCALING_FACTOR = 2.5;
final NumberAxis xAxis = new NumberAxis("Entfernug zur Sonne",0, 5000, 500);
final NumberAxis yAxis = new NumberAxis("Einwohnerzahl (Millionen)",
   -500, 7000, 1000);

final BubbleChart<Number, Number> blc = new BubbleChart<Number, Number>(xAxis,
yAxis);

XYChart.Series merkur = new XYChart.Series("Merkur",
   FXCollections.observableArrayList(new XYChart.Data(58, 0,
     4.8*SCALING_FACTOR)));
XYChart.Series venus = new XYChart.Series("Venus",
   FXCollections.observableArrayList(new XYChart.Data(108 , 0,
     12.4*SCALING_FACTOR)));
XYChart.Series erde = new XYChart.Series("Erde",
   FXCollections.observableArrayList(new XYChart.Data(150, 6800,
     12.7*SCALING_FACTOR)));
XYChart.Series mars = new XYChart.Series("Mars",
```

```
    FXCollections.observableArrayList(new XYChart.Data(280, 0,
      6.8*SCALING_FACTOR)));
XYChart.Series jupiter = new XYChart.Series("Jupiter",
    FXCollections.observableArrayList(new XYChart.Data(775, 0,
      142.8*SCALING_FACTOR)));
XYChart.Series saturn = new XYChart.Series("Saturn",
    FXCollections.observableArrayList(new XYChart.Data(1440, 0,
      120.8*SCALING_FACTOR)));
XYChart.Series uranus = new XYChart.Series("Uranus",
    FXCollections.observableArrayList(new XYChart.Data(2870, 0,
      47.6*SCALING_FACTOR)));
XYChart.Series neptun = new XYChart.Series("Neptun",
    FXCollections.observableArrayList(new XYChart.Data(4500, 0,
      44.6*SCALING_FACTOR)));
```

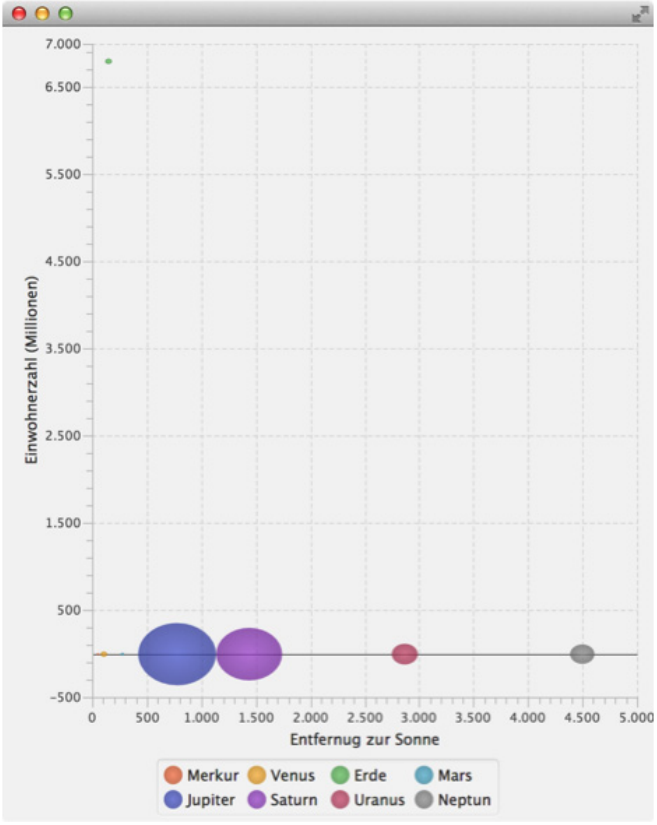

Abb. 8–8 *BubbleCharts haben die Blasengröße als dritte Dimension.*

8.4 Dynamische Charts

So wie wir die Charts bis jetzt verwendet haben, können wir sie gut für das Erstellen von Reports einsetzen. Sie sind aber genauso gut für die Darstellung veränderlicher Livedaten geeignet. Nehmen wir zum Beispiel einen fiktiven Aktienkurs:

```
NumberAxis xAxis = new NumberAxis();
final NumberAxis yAxis = new NumberAxis();
final LineChart<Number, Number> lineChart = new LineChart<Number,
Number>(xAxis, yAxis);
lineChart.setCreateSymbols(false);
lineChart.setAnimated(false);
xAxis.setForceZeroInRange(false);
XYChart.Series<Number, Number> dataSeries = new XYChart.Series<Number,
Number>();
lineChart.getData().add(dataSeries);
Timeline animation = new Timeline();
animation.getKeyFrames()
   .add(new KeyFrame(Duration.millis(100),
    new EventHandler<ActionEvent>() {
      @Override
      public void handle(ActionEvent actionEvent) {
          dataSeries.getData().add(new XYChart
             .Data<Number, Number>(i++, Math.random() * 100));
          if (dataSeries.getData().size() > 30) {
             dataSeries.getData().remove(0);
          }
      }

    }
   ));
animation.setCycleCount(Animation.INDEFINITE);
animation.play();
```

Als Datenquelle verwenden wir einen Zufallsgenerator, der über eine `Timeline`-Animation jede Zehntelsekunde einen neuen Wert einfügt. Es werden jeweils maximal gleichzeitig 30 Werte angezeigt. Sobald diese Zahl erreicht ist, werden vom Anfang der Liste in gleicher Frequenz Werte gelöscht, sodass wir einen durchlaufenden Kurs erhalten, wie wir das von Aktienkursdarstellungen kennen. Da sich das Werteintervall der x-Achse kontinuierlich ändern soll, müssen wir mit `xAxis.setForceZeroInRange(false)` dafür sorgen, dass der Nullwert nicht mit angezeigt wird. Bei Aktienkursen ist es auch unüblich, dass ein Symbol am Datenpunkt dargestellt wird. Das schalten wir im Beispiel mit `lineChart.setCreateSymbols(false);` ab. Es ist vielleicht nicht sofort verständlich, weshalb wir die Animation abschalten mussten (`lineChart.setAnimated(false)`). Probieren Sie einfach mal aus, wie sich die Darstellung ändert, wenn Sie diese Zeile auskommentieren.

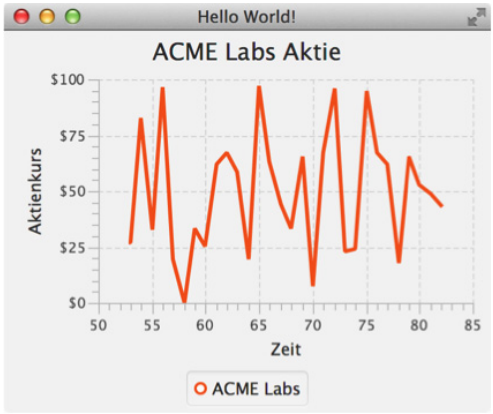

Abb. 8–9 *Ein Chart mit ständig aktualisierten Daten*