Einstieg in die Programmierung von Computern – Teil VII

Im Teil 7 beschäftigen wir uns mit der Strukturierung der Java-Software mittels „Pakete“ und „CLASSPATH“ für Compiler und Java-Interpreter. Außerdem beginnen wir mit dem Thema „grafische Oberflächen“ in Java. Dazu werden wir ein erstes kleines Java-Programm schreiben.

Pakete

Bislang haben wir in der Artikelserie alle Java-Sourcen, also die Quellcode-Dateien mit der Endung „.java“ in einem gemeinsamen Verzeichnis gespeichert. Bei ganz kleinen Programmen ist das prinzipiell auch kein Problem, bei größeren Softwareprojekten wird jedoch eine Strukturierung der Software benötigt. Die Programmiersprache Java unterstützt die Softwareentwickler zum Glück auch in der Entwurfsphase und ermöglicht mit dem Feature „Pakete“ eine optimale Strukturierung der Programmeinheiten. Was sind nun genau Programmeinheiten in Java?

- Klassen
- Interfaces (Schnittstellen)
- Threads (wird in einem späteren Artikel genau erklärt) und
- Pakete

Die Programmeinheiten (auch als „Module“ bezeichnet) sind quasi die logischen Bausteine des Quellcodes einer Java-Applikation und sind somit für den Java-Compiler kompilierbar. Java-Pakete sind somit auch „Klassenbibliotheken“, die aus einzelnen Quellcode-Dateien bestehen. Abbildung 1 erklärt die Zusammenhänge.
Als größte Strukturierungseinheit in Java bieten Pakete die Möglichkeit für eine optimale Strukturierung der Quellcode-Dateien, einen Namensraum (jedes Paket bekommt einen Namen) und somit auch einen

„com.oemus.itkolumne.controller“. In diesem Paket sollen also alle Controller-Klassen organisiert werden. In der zweiten Klasse wird das Interface verwendet. Da das Interface „DataInterface“ in einem anderen Paket liegt, muss dieses Paket importiert werden, um es zu verwenden. Um in einer Java-Datei den dazugehörigen Paketnamen bekannt zu machen, wird die „package-Anweisung“ verwendet:
package com.oemus.itkolumne.data;

Der Import einer Programmeinheit aus einem anderen Paket kann mit der „import-Anweisung“ durchgeführt werden:
import com.oemus.itkolumne.data.DataInterface;
bzw.
import com.oemus.itkolumne.data.*;

oder im Code mit dem qualifizierten Namen (inkl. Paketnamen)
com.oemus.itkolumne.data.DataInterface;

C:\java\itkolumne**com****oemus****itkolumne****Controller**\MainController.java

Im Kasten 2 wird keine „import-Anweisung“ in der Klasse „MainController“ verwendet, also muss voll qualifiziert zugegriffen werden.
Beachte: Der voll qualifizierte ist der Name einer Klasse mit vorangestelltem Paketnamen. In unserem Beispiel also **com.oemus.itkolumne.data.**DataInterface

Das Kompilieren der beiden Quellcode-Dateien in einer Paketstruktur geht mit dem folgenden Befehl in der Windows-Kommando-Konsole:

1. Wechsel in das Basisverzeichnis. Auf meinem Rechner z. B. auf dem Laufwerk „C:\“
> cd java\itkolumne

2. Komilieren der beiden Quellcode-Dateien mit den Befehlen
> **javac** com/oemus/itkolumne/data/ DataInterface.java
> **javac** com/oemus/itkolumne/controller/MainController.java

3. Starten der Applikation mit
> **java** com/oemus/itkolumne/controller/MainController

Wahl von Paketnamen

Prinzipiell kann ein beliebiger Paketname gewählt werden. Es hat sich in der Java Softwareentwicklung jedoch eingebürgert, umgekehrte Domänennamen zu verwenden. Wenn also die Website-Adresse z. B. **http://www.oemus.com** ist, dann ist der Paketname „**com.oemus**“. Umgekehrte Domainnamen gewährleisten, dass Klassen weltweit eindeutig bleiben. Außerdem ist zu beachten, dass der Paketname vollständig klein zu schreiben ist.

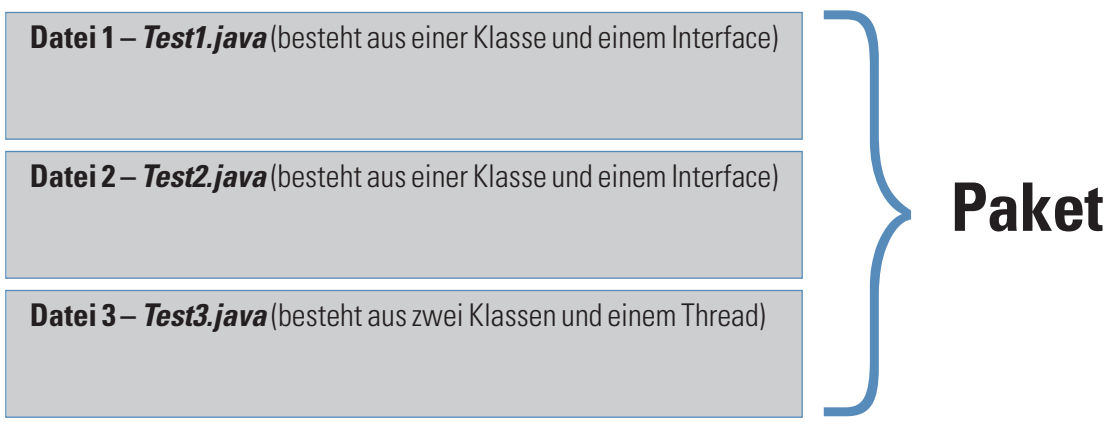

Abb. 1: Ein Java-Paket mit drei Java-Dateien.

Zugriffsschutz durch die Kapselung der Programmeinheiten. Die Programmeinheiten können nur mit ihrem Paketnamen angesprochen werden. Wie erstellt man nun Java-Pakete. Dazu wollen wir uns ein kleines Beispiel anschauen (siehe Kasten 1). Die Datei „DataInterface.java“ gehört zum Paket „com.oemus.itkolumne.data“ und hat zwei Meldungstexte als statische Variablen in einem Interface gespeichert. Das Paket „data“ soll demnach alle Daten-Interfaces enthalten. Die Klasse „MainController“ in der Datei „MainController.java“ liegt im Paket also
public class MainController implements **com.oemus.itkolumne.data.**DataInterface { …
und
System.out.println(**com.oemus.itkolumne.data.DataInterface.**MELDUNGSTEXT_1);

Beachte: Die Quellcode-Dateien, die in einem Paket organisiert sind, müssen auch in einem Verzeichnis gespeichert werden, das den Paketnamen hat. In unserem Beispiel also:
C:\java\itkolumne**com****oemus****itkolumne****Data**\DataInterface.java

CLASSPATH

Ganz allgemein gesagt ist der CLASSPATH eine Umgebungsvariable, über die der Java-Compiler bzw. der Java-Interpreter

```
// Datei: DataInterface.java
package com.oemus.itkolumne.data; // package-Anweisung

public interface DataInterface {

    public static final String MELDUNGSTEXT_1 = "Heute scheint die Sonne.";
    public static final String MELDUNGSTEXT_2 = "Heute regnet es.";
}

// Datei: MainController.java

package com.oemus.itkolumne.controller; // package-Anweisung

import com.oemus.itkolumne.data.DataInterface; // import-Anweisung

public class MainController implements DataInterface {

    public static void main(String[] args) {

        System.out.println(MELDUNGSTEXT_1);
        System.out.println(MELDUNGSTEXT_2);
    }
}
```

Kasten 1: Java-Pakete – package- & import-Anweisung.

```
// Datei: DataInterface.java
package com.oemus.itkolumne.data; // package-Anweisung

public interface DataInterface {

    public static final String MELDUNGSTEXT_1 = "Heute scheint die Sonne.";
    public static final String MELDUNGSTEXT_2 = "Heute regnet es.";
}

// Datei: MainController.java

package com.oemus.itkolumne.controller; // package-Anweisung

public class MainController implements com.oemus.itkolumne.data.DataInterface {

    public static void main(String[] args) {

        System.out.println(com.oemus.itkolumne.data.MELDUNGSTEXT_1);
        System.out.println(com.oemus.itkolumne.data.MELDUNGSTEXT_2);
    }
}
```

Kasten 2: Java-Pakete – voll qualifizierter Zugriff (ohne import-Anweisung).

```
// Datei: HelloWorld.java
package com.oemus.itkolumne; // package-Anweisung

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class HelloWorld extends Application {

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {

        primaryStage.setTitle("Hello World!");
        Button btn = new Button();
        btn.setText("Say 'Hello World'");

        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });

        StackPane root = new StackPane();
        root.getChildren().add(btn);
        primaryStage.setScene(new Scene(root, 300, 250));
        primaryStage.show();
    }
}
```

Kasten 3: Erste JavaFX GUI-Applikation.

den Weg zu den verwendeten Bibliotheksklassen und eigenen Klassen finden kann. Werden Pakete verwendet (sollte der Normalfall sein), so ist der voll qualifizierte Weg, den der Java-Interpreter zum Finden einer Klasse benötigt, immer.

```
C:\java\itkolumne\com\oemus\itkolumne\controller
```

CLASSPATH Paketnamen

Fehlt der CLASSPATH oder der Paketname, so kann der Java-Compiler bzw. Java-Interpreter die dort befindlichen Klassen nicht auffinden und es kommt zu einem Fehler.

Der CLASSPATH im Java-Compiler

Verwendet man den Java-Kompiler, wird der Klassenpfad mit dem Attribut **classpath** gesetzt. Werden fremde Java-Bibliotheken (**.jar-Dateien**) in der Applikation benutzt, müssen diese unter einem Windows-System mit Strichpunkten (;) getrennt angehängt werden. Beispiel in der Windows-Konsole:

```
javac -classpath "c:\java\itkolumne\library1.jar;c:\java\itkolumne\library2.jar" Applikation.java
```

Erklärung:

- **javac:** ist der Befehl zum Aufrufen des Java-Kompilers.
- **-classpath:** ist die Option zum Setzen eines oder mehrerer Klassenpfade.
- **"c:\java\itkolumne\library1.jar; c:\java\itkolumne\library2.jar":** sind die einzubindenden Klassenbibliotheken.
- **Applikation.java:** ist die zu kompilierende Datei.

Grafische Oberflächen mit JavaFX

Applikationen mit einer grafischen Bedienoberfläche werden auch als „GUI (**G**raphical **U**ser **I**nterface)-Applikationen“ bezeichnet. In Java schreibt man solche Applikationen entweder mit **Swing** oder mit **JavaFX**. JavaFX wird derzeit von Oracle als Nachfolge-Technologie von Swing gesehen und stark vorangetrieben. *Wikipedia schreibt dazu:* JavaFX ist ein Framework für plattformübergreifende **Rich Internet Applications (RIA)**. Es ist eine Java-Spezifikation und von Oracle und steht in direkter Konkurrenz zu Adobe Flash bzw. Flex und Microsoft Silverlight. Die JavaFX 2.x-Laufzeitumgebung wird ab der Version Java SE Runtime 7 Update 6 mitinstalliert, kann aber auch separat heruntergeladen und installiert werden.

Das Tolle an JavaFX ist, dass es auf vielen Endgeräten wie z. B Desktop-Computern, Mobilfunkgeräten, Set-Top-Boxen ablaufen kann.

Gleich am Anfang wollen wir eine erste Beispiel-Applikation entwickeln (siehe Kasten 3), kompilieren und anschließend starten. Das HelloWorld-Programm wird auf meinem Rechner in einer Windows-Konsole so kompiliert:

```
javac -classpath "C:\Java\jre7\lib\jfxrt.jar" com\oemus\itkolumne\HelloWorld.java
```

Die „-classpath“ Option ist notwendig, da wir eine spezielle JavaFX-Bibliothek mit kompilieren müssen. Nun starten wir unsere erste JavaFX GUI-Applikation mit dem folgendem Befehl in einer Windows-Konsole:

```
java -classpath ".;C:\Java\jre7\lib\jfxrt.jar" com.oemus.itkolumne.HelloWorld
```

Danach sollten Sie auf Ihrem Bildschirm ein neues Fenster sehen:

Keine Angst! In der nächsten Ausgabe werde ich dann die Applikation im Detail erklären!

Ausblick zum Teil 8 der Serie

Im nächsten Teil werden wir uns weiter mit dem Thema JavaFX beschäftigen. Bleiben Sie also dran! ZT

ZT Adresse

Thomas Burgard Dipl.-Ing. (FH)
Softwareentwicklung & Webdesign
Bavariastraße 18b
80336 München
Tel.: 089 540707-10
Fax: 089 540707-11
info@burgardsoft.de
www.burgardsoft.de
burgardsoft.blogspot.com
twitter.com/burgardsoft

Thomas Burgard Dipl.-Ing. (FH) [Autoreninfo]