CDCover

Angabe

zu Schreiben ist ein Programm CDCover, welches beim Aufruf den Titel einer Audio-Cd und einen oder mehrere Track-Namen als Kommandozeilen-Parameter erwartet. Der angegebene Titel und die Track-Namen sollen wie im folgenden Beispiel ersichtlich formatiert ausgegeben werden:

```
c:\java> java CDCover Bochum Bochum Männer "Flugzeuge Im Bauch" Alkohol
Amerika "Für Dich Da" "Jetzt Oder Nie" Fangfragen Erwischt Mambo
                 BOCHUM

01 ............................. Bochum
02 ............................. Männer
03 .................. Flugzeuge Im Bauch
04 ............................. Alkohol
05 ............................. Amerika
06 ......................... Für Dich Da
07 ....................... Jetzt Oder Nie
08 ........................... Fangfragen
09 ............................ Erwischt
10 .............................. Mambo

c:\java>
```

Der erste der angegebenen Kommandozeilen-Parameter ist der Titel der CD, der soll in Grossbuchstaben und zentriert auf einer Breite von 40 Zeichen ausgegeben werden. Danach folgt eine Leerzeile, und dann durchnnumeriert sämtliche angegebenen Titel, rechtsbündig auf dieselbe Breite, mit Punkten gepadded.

Der Einfachheit halber soll kein Titel länger als 35 Zeichen sein.

Wird das Programm ohne oder mit zuwenigen Parameter(n) aufgerufen, soll eine entsprechende Usage-Meldung ausgegeben werden:

```
c:\java> java CDCover nurEinTitelaberKeinTrack
Usage: java CDCover Title Track1 [Track2 ...]

c:\java>
```

Tipps

So, da muss ich jetzt ein bisschen weiter ausholen, vor allem weil ich bei einer der Dienstags-Gruppen mit dem Stoff nicht so weit gekommen bin wie ich dachte. Den fehlenden Teil fasse ich euch hier kurz zusammen.

In den bisherigen Beispielen haben wir eigene Klassen programmiert und Klassen aus der Java Klassenbibliothek (KB) benutzt, bekannt sind unter anderem die Klassen java.io.File, java.util.Random, java.lang.System und so weiter.

Bei dem vorliegenden Beispiel sind Klassen von Drittanbietern aus dem Internet runterzuladen und zu verwenden. Was wir von diesen Drittanbietern (siehe gleich) brauchen, sind natürlich zum einen die Klassen in Bytecode-Form, und zum anderen eine Dokumentation der Schnittstellen dieser Klassen.

Am Besten laden wir uns das alles gleich mal aus dem Netz runter: wir gehen auf die Seite http://www.apache.org, dort links im Projects-Menü weiter auf "Jakarta", und dort im Subprojects-Menü auf "Commons". Bei den Jakarta Commons handelt es sich um eine Reihe von Open-Source Komponenten, die wir in unseren Programmen einsetzen können. Mit Komponenten sind hier einfach Gruppen von zusammengehörenden, nützlichen Java-Klassen gemeint, die uns bei der Bewältigung von bestimmten Problemen helfen können. Für unser Programm laden wir uns die Komponente "Lang" runter, dabei handelt es sich um ein paar nützliche Allerwelts-Klassen, die sich als Vervollständigung der Funktionalität der Java Klassenbibliothek verstehen.

Wer Lust und Interesse hat, findet bei den Jakarta Commons noch eine ganze Menge weiterer nützlicher Komponenten, z.B. zum einfachen Handling von komplexen Kommandozeilen-Parametern, zum Logging, usw usf...

Wir gehen jedenfalls erstmal im Menü links beim Menüpunkt "Download" auf "Releases (mirrored)", wählen aus der jetzt erscheinenden Liste "Commons Lang", und laden in Folge das Zip-File mit den Binaries (kompilierte Quelltexte nennt man auch Binaries) runter:

http://mirror.deri.at/apache/jakarta/commons/lang/binaries/commons-lang-2.1.zip

Wenn wir das Zip-File auspacken, finden wir darin die 2 Dinge, die wir brauchen: erstens finden wir Dokumentation im javadoc-Format, und zweitens finden wir die kompilierten Klassen in Form einer Jar Datei: commons-lang-2.1.jar.

Für die Gruppe, bei der ich nicht mehr dazu gekommen bin, die Jar-Dateien zu erklären, hier bitte schnell folgende Zusammenfassung: werden von irgendjemandem mehrere kompilierte Klassen zur Verfügung gestellt, die zum Teil auch voneinander abhängen und jedenfalls zusammengehören, dann geht man meistens her und kopiert nicht alle .class-Dateien einzeln herum, sondern man packt sie zusammen in eine sogenannte Java-Archiv-Datei. Ihr erkennt solche Dateien an der Dateiendung "jar" (= Java ARchive). Zum Beispiel sind sämtliche Klassen der Java Klassenbibliothek auch in einer solchen Archiv-Datei zusammengepackt; auf den Schulrechnern findet diese Datei bei Interesse unter c:\programme\jdk\jre\lib\rt.jar (rt steht für Runtime).

Vereinfacht gesagt sind Java-Archive nichts weiter als Zip-Dateien mit einer anderen Dateiendung. Ihr könnt jede Jar-Datei auch im Winzip öffnen. Das JDK kommt mit einem Kommandozeilen-Programm zum Erzeugen und Entpacken von Jar-Dateien daher, das wäre die "jar.exe". Kann man sich mal anschauen.

Wer möchte, kann mit diesem Werkzeug oder mit WinZip mal einen Blick in die "commons-lang-2.1.jar" werfen, darin findet ihr eine Menge .class-Dateien.

So, jetzt haben wir also das Java-Archiv mit den Klassen die wir benutzen wollen und die Dokumentation dazu. Lest euch mal die Dokumentation zur Klasse "org.apache.commons.lang.StringUtils" durch, darin findet ihr eine Menge nützlicher statischer Methoden, die uns bei unserem CDCover Programm helfen... ins Auge sticht zum Beispiel die Methode center(), ... oder leftPad(), ... usw...

Jetzt stellt sich noch die Frage, wie wir dem Compiler und dem Interpreter mitteilen, dass wir die Klassen aus dem runtergeladenen Java-Archiv benutzen wollen. Der Trick ist, dass wir dem Kompiler und den Interpreter beibringen, dass er auch in dem Java-Archiv nach Klassen suchen soll, die wir in unserem Programm benutzen. Das machen wir, indem wir das Java-Archiv in den Classpath mit aufnehmen (das ist NEU: bisher hatten wir im Classpath immer nur Verzeichnisse stehen, in denen wir unsere kompilierten .class Dateien hatten, wir können aber auch Java-Archive angeben, die kompilierte .class Dateien enthalten. Konzeptionell ist das aber gar nicht so verschieden: sowohl ein Verzeichnis als auch ein Java-Archiv sind Container für .class Dateien).

Wir könnten zum Beispiel erstmal folgendes einfache Programm schreiben, in dem wir die Klasse org.apache.commons.lang.StringUtils benutzen:

```
import org.apache.commons.lang.StringUtils;

class TestClass {
  public static void main(String[] args) {
    System.out.println(StringUtils.center("hallo!", 40));
  }
}
```

Das Programm ist dann folgendermaßen zu kompilieren und auszuführen:

```
c:\java> set CLASSPATH=c:\irgendwo\commons-lang-2.1.jar;c:\java

c:\java> javac TestClass.java

c:\java> java TestClass
                  hallo!

c:\java>
```

Das Programm ist zwar ziemlich sinnlos, aber wenn wir alles richtig gemacht haben, haben wir immerhin schon mal die Klasse org.apache.commons.lang.StringUtils benutzt und sind auf dem richtigen Weg.

Kurzer Recap: es war also notwendig, das Java-Archiv mit den kompilierten Klassen in den Classpath aufzunehmen, damit konnten wir in unserem Programm Klassen aus dem Archiv importieren und benutzen. Wenn wir in unseren Quelltexten Klassen importieren, werden der Kompiler und der Interpreter versuchen, diese Klassen zu finden und gehen dazu ja bekannterweise der Reihe nach alle Elemente im Classpath durch. In unserem Fall stösst der Kompiler dabei zuerst mal auf das Java-Archiv, und darin findet er auch schon die StringUtils Klasse (Nota bene: wir mussten das Java-Archiv nicht entpacken!)

So, ich glaub damit hätt ich alles gesagt, was von meiner Seite dazu zu sagen wäre...

jetzt seid ihr dran!

viel Erfolg,

lg

Fritz