Erste Schritte zum lauffähigen Java-Programm

Diese kleine Einführung ist eine Hilfe für Studenten der Vorlesung SWT I zur Meisterung der sich ergebenden Hürden bei der Erstellung eines ersten kleinen Java-Programms. Es wird sowohl auf die besonderen Bedingungen im Rechenzentrum als auch auf das Vorgehen am eigenen PC eingegangen.

1. Was wird benötigt?

Um in der Programmiersprache Java ein Programm zu schreiben, braucht man zunächst zwei essentielle Dinge:

- einen Text-Editor
- ein installiertes Java-SDK (Software Development Kit) auch JDK genannt (Java Development Kit)

Im Rechenzentrum ist natürlich beides vorhanden.

1.1 Zur Wahl des Editors

Als Editor ist z.B. Proton zu empfehlen. Es würde aber auch ein einfacheres Programm, wie z.B. Notepad tun. Wichtig bei der Wahl eines Editors ist, dass es sich um einen sog. Nur-Text-Editor handelt, der keine Formatierungen oder ähnliches, sondern nur den reinen Text in der Datei speichert. Ungeeignet wären Programme wie MS-Word oder Open-/LibreOffice. Auch unter Linux sollte es keinerlei Schwierigkeiten machen, einen geeigneten Editor zu finden (Gedit, emacs, Kate,...).

1.2 Woher bekommt man ein SDK?

Wer nicht im Rechenzentrum sondern zuhause arbeitet, sollte sicherstellen, dass eine aktuelle Java Platform – Standard Edition (Java SE) installiert ist. Java SE (häufig auch JDK genannt) enthält u.a. den Java-Compiler. Es ist nicht zu verwechseln mit der JRE (Java Runtime Environment). Die JRE erlaubt einem nur, Javaprogramme zu starten, nicht aber diese zu kompilieren. Das JDK kann man sich frei herunterladen (z.B. bei der Firma Oracle unter der Web-Adresse https://jdk8.java.net/download.html).
Dort sind Versionen für Mac OS X, Linux und Windows verfügbar. Weiterhin ist es empfehlenswert, die Java-API-Dokumentation gleich mit herunterzuladen. Sie wird einem im weiteren Verlauf noch gute Dienste leisten (spätestens im Software-Praktikum im 3. Semester).
Für diejenigen, die über eine langsame Internetanbindung verfügen der Hinweis: Das SDK + Hilfe ist nicht gerade klein (ca. 120 MB)! Aber es stehen ja auch CD-Brenner und ZIP-Laufwerke im Rechenzentrum zur Verfügung.

2 Anmerkungen zur Installation

Kompilieren
Beim Kompilieren von Java Programmen mit dem Befehl javac können folgende Fehler auftreten:

1. **Umgebungsvariable PATH ist nicht richtig gesetzt**
2. **Syntaktische Fehler im Quellcode**
3. **Fehlende Schreibrechte im Verzeichnis**

zu 1.) Damit man das Programm javac in einem beliebigen Verzeichnis ausführen kann, muss dem Betriebssystem mitgeteilt werden, dass der Pfad, in dem sich die Java-Tools befinden, unter der Umgebungsvariablen PATH gesetzt ist.
Um zu testen, ob dies der Fall ist, kann man einfach einmal `javac` ohne Parameter aufrufen. Erscheint darauf eine Hilfe zu den Parametern von `javac`, so ist die Umgebungsvariable korrekt gesetzt, und der Fehler vermutlich im Quelltext zu suchen. Erscheint jedoch eine Fehlermeldung, dass der Befehl oder Dateiname nicht gefunden wurde, so muss die Variable noch gesetzt werden.
Auch im Rechenzentrum tritt dieses Problem auf!
Zur Lösung bieten sich zwei Varianten an:

a) Setzen der Umgebungvariablen über die Eingabeaufforderung:

Der Befehl dazu lautet (für die Version 1.6.0_02):

```
set path=C:\Programme\Java\jdk1.6.0_02\bin
export PATH=$PATH:/path/to/jdk/bin

bei MAC OSX wird Umgebungsvariable automatisch gesetzt
```

Je nachdem, in welchem Verzeichnis das SDK installiert ist.
Dieser Pfad ist natürlich für das Rechenzentrum gedacht, zuhause wird der Pfad, in dem das SDK installiert ist, eventuell ein anderer sein.
Nachteil dieser Methode: Nach dem Schließen der Eingabeaufforderung geht die Einstellung wieder verloren.

b) Setzen der Umgebungsvariablen über die Systemsteuerung:

Windows:
System → Erweitert → Umgebungsvariablen…

Dort fügt man mit „Neu“ eine Umgebungsvariable mit dem Namen: path und als Wert den bin-Pfad des JDKs (Rechenzentrum siehe oben) hinzu, bzw. ergänzt den schon vorhandenen Eintrag um das zusätzliche Verzeichnis, das Trennzeichen zwischen den einzelnen Pfaden ist das Semikolon.
Diese Variante bleibt zuhause dauerhaft, im Rechenzentrum wenigstens bis zur Abmeldung erhalten.

Linux:
Um unter Linux eine Umgebungsvariable dauerhaft setzen zu können, muss diese in der ~/.bashrc im eingetragen werden. Dafür einfach auf dem Terminal

```
echo "export JAVA_HOME=/path/to/jdk" >> ~/.bashrc
echo "export PATH=$PATH:$JAVA_HOME/bin" >> ~/.bashrc
```

eingeben.

Mac OS X:
Nach der Installation des JDK von Oracle müssen keine weiteren Schritte unternommen werden. Um die Installation zu testen einfach das Programm ‚Terminal' (/Applications/Utilities/Terminal.app) ausführen und die Befehle java und javac ohne Argumente ausführen. Nun sollten beide Programme eine kurze Hilfe ausgeben.

Hat alles geklappt (startet nun das Programm javac), kann der Quelltext nun (wie oben beschrieben) kompiliert werden.

zu 2.) Häufige syntaktische Fehler sind: Semikolon vergessen, falsche Anführungszeichen, Fehler in Groß- und Kleinschreibung.
Also noch mal genau mit der Vorlage vergleichen.

zu 3.) Sollte eigentlich nicht vorkommen, da man zum Speichern von HelloWorld.java ja schon Schreibrechte benötigt.

NoClassDefFoundError beim Ausführen von Java-Programmen

Da die Umgebungsvariable CLASSPATH im Rechenzentrum nicht ordentlich gesetzt ist, wird dort der aktuelle Pfad bei der Suche nach den Klassen nicht mit einbezogen. Obwohl die Befehle javac und auch java erkannt werden, kann beim Versuch, ein kompiliertes Programm auszuführen, ein NoClassDefFoundError ausgegeben werden. Dies liegt meist daran, dass im Classpath noch ein Punkt ergänzt werden muss, der auf das aktuelle Verzeichnis verweist.

Lösung:
Setzen der „Classpath"-Variablen über die Systemsteuerung: System → Erweitert → Umgebungsvariablen – hier die Variable „Classpath" editieren, indem man ans Ende der Eintragsliste noch ein Semikolon gefolgt von einem einfachen Punkt anfügt.

3. Ein erstes kleines Beispielprogramm

Der Quelltext des ersten kleinen Beispielprogramms: Hello World (wie könnte es anders sein ;-)).

```
public class HelloWorld
{
     public static void main(String[] args)
     {
          System.out.println(“Hello World!”);
     }
}
```

Nachdem man den Quelltext unter `HelloWorld.java` gespeichert hat (man achte auf die richtige Dateiendung, die Groß- und Kleinschreibung und es dürfen auch keine Leerzeichen im Dateinamen vorkommen), muss man den Compiler aufrufen. Es ist zu beachten, dass der Name der öffentlichen (public) Klasse identisch sein muss mit dem Dateinamen.

4. Kompilieren von Java Programmen

Bei dem Java-Compiler handelt es sich um ein Kommandozeilen-Tool namens `javac`.
Für den Aufruf muss man unter Windows die Eingabeaufforderung öffnen (Start → Programme → Zubehör → Eingabeaufforderung).
Unter Mac OS/Linux benötigt man eine Shell/Terminal (bash, zsh).

Danach wechselt man in das Verzeichnis, in dem die Datei `HelloWorld.java` gespeichert wurde. Prinzipiell würde nun der Aufruf des Compilers folgendermaßen aussehen:

```
javac HelloWorld.java
```

Sind keine syntaktischen Fehler vorhanden, so sollte am Ende im gleichen Verzeichnis, in dem sich auch die Quelldatei befindet, eine Datei `HelloWorld.class` existieren.

5. Das Programm ausführen

Java-Quellcode wird nicht durch `javac` in direkt ausführbaren Code übersetzt, sondern in maschinenunabhängigen Bytecode (die `class`-Dateien). Zur Ausführung des Programms wird ein Interpreter für diesen Bytecode benötigt. Dabei handelt es sich um das Programm `java`.

Der Aufruf des kleinen Testprogramms wäre dann:

```
java HelloWorld
```

bzw. im Rechenzentrum (Punkt im classpath nicht gesetzt):

```
java –classpath . HelloWorld
```

oder auch kürzer:

```
java –cp . HelloWorld
```

Wichtig: Wieder auf Groß- und Kleinschreibung achten und die Endung `.class` weglassen.

Als Ergebnis der Ausführung sollte ein „Hello World!“ auf der Konsole ausgegeben werden.

Für Fragen ist das Forum, das unter der SWT-Seite zur Lehrveranstaltung verlinkt ist, sicher eine geeignete Anlaufstelle.

https://auditorium.inf.tu-dresden.de/en/groups/110406004

6. Ergänzungen für die spätere Verwendung

Es sei darauf hingewiesen, dass jede öffentliche Klasse in einer separaten Datei mit dem gleichen Namen wie die öffentliche Klasse und der Endung `.java` abgelegt sein muss.

6.1 Kompilieren mehrerer Quellcodedateien

alle Dateien im Verzeichnis kompilieren: `javac *.java`

alle Dateien kompilieren, die in der Datei `liste.txt` stehen: `javac @liste.txt`

Wenn Klassen referenziert werden, die sich in Unterverzeichnissen befinden (bedingt durch Paketstrukturen), sollte es keine Probleme geben.

Werden jedoch Quelldateien in externen Verzeichnissen referenziert, so müssen deren Pfade über eine Compiler-Option bekannt gemacht werden.

Beispiel:
Windows:

```
javac –classpath .;c:\extp1 test.java
```

Mac OS/Linux:

```
javac -classpath .:/home/user/dev/ test.java
```

Diese Option sagt dem Compiler, er soll neben dem aktuellen Verzeichnis auch im Verzeichnis c:\extp1 (bzw. /home/user/dev) nach notwendigen Dateien schauen.
Bitte beachten: Unter Linux ist das Trennzeichen der Doppelpunkt und nicht das Semikolon.

6.2 Probleme mit Packages

Klassen lassen sich in sog. Packages strukturieren. Dies geschieht durch das Schlüsselwort `package` mit darauf folgendem Bezeichner für das Paket (abgeschlossen durch Semikolon) am Anfang des Quelltextes. Will man eine Klasse aus einem solchen Paket verwenden, so geschieht dies durch das Schlüsselwort `import`.
Wichtig ist hier, dass die Ordnerstruktur, in der die Quelltexte liegen, ebenfalls die Paketstruktur widerspiegelt.

Beispiel: Befindet sich die Klasse `Test1` im Paket `phaupt.pteil`, so muss die relative Adresse der Datei `./phaupt/pteil/Test1.java` sein.