1 Datenbanken-Streams-Dateien

1.1 Datenbanken

Eine Datenbank ist eine organisierte Sammlung von strukturierten Informationen, die in einem Computer oder einem anderen elektronischen System gespeichert sind. Datenbanken werden verwendet, um große Mengen von Daten effizient zu verwalten und darauf zuzugreifen.

Es gibt verschiedene Arten von Datenbanken, die je nach ihrer Struktur und Funktionalität kategorisiert werden können. Hier sind einige der gängigsten Datenbanktypen:

Relationale Datenbanken: Dies ist der am häufigsten verwendete Typ. In relationalen Datenbanken werden Daten in Tabellen organisiert, die aus Zeilen (Datensätzen) und Spalten (Attributen) bestehen. Die Beziehungen zwischen den Tabellen werden durch Schlüssel hergestellt. SQL (Structured Query Language) wird verwendet, um Daten in relationalen Datenbanken abzufragen und zu manipulieren.

Objektorientierte Datenbanken: Hier werden Daten in Form von Objekten gespeichert, die aus Eigenschaften (Attributen) und Methoden bestehen. Objektorientierte Datenbanken ermöglichen eine effiziente Speicherung und Manipulation von komplexen Datenstrukturen.

Kommunikation mit Datenbank in Java

Um zwischen Java und einer Datenbank kommunizieren zu können, wird ein JDBC-Treiber (Java Database connectivity) benötigt.

Um die Kommunikation mit einer Datenbank zu erleichtern, gibt es bereits fertige Klassen in Java.

Treibermanager: Stellt die Verbindung zur Datenbank her, dies erfolgt meistens über einen URL. Zuerst muss aber der korrekte JDBC Treiber heruntergeladen werden.

Connection: Die Connection-Klasse repräsentiert eine Verbindung zu einer Datenbank. Sie wird mithilfe der DriverManager.getConnection()-Methode erstellt und ermöglicht das Ausführen von SQL-Anweisungen auf der Datenbank.

Statement: Die Statement-Klasse wird verwendet, um SQL-Anweisungen an die Datenbank zu senden. Sie ermöglicht das Ausführen von statischen SQL-Anweisungen ohne Parameter.

ResultSet: Man erhält ein Objekt des Typen ResultSet wenn man einen SELECT Befehl sendet, ResultSet besitzt die methode .next() welche zurückliefert ob es noch ungelesene Daten gibt. Danach können die aktuellen Daten mit den einzelnen Getter-Methoden und dem richtigen Schlüssel (zb. ID, Name, etc..) abgeholt werden.

1.2 Streams

InputStream/OutputStream

- Abstrakte Klasse ➔ read/write müssen überschrieben werden
- Read/write ➔ Byteweise Lesen/Schreiben
- Verwendung bei Netzwerkkommunikation und Dateien

Reader/Writer

- Abstrakte Klasse ➔ Schreiben/Lesen von Text (Character)-Streams
- BufferedReader/BufferedWriter
    - Die zu lesenden bzw. zu schreibenden Daten werden in einem Buffer zwischengespeichert. Dadurch wird das Lesen bzw. Schreiben von einzelnen Zeichen, Arrays und Strings effizienter.
    - newLine() ➔neue Zeile wird begonnen. Die Verwendung dieser Methode ist besser als das direkte Schreiben des Charakters ‚\n', da dieser nicht auf allen Plattformen zum Beenden einer Zeile verwendet wird.
- FileReader/FileWriter
    - Komfort-Klasse zum Lesen bzw. Schreiben von Dateien
- Brückenklassen
    - InputStreamReader: ByteStreams ➔ TextStreams
    - OutputStreamWriter: TextStreams ➔ ByteStreams
    - Für eine höhere Effizienz sollten diese Klassen werden verwendet werden.

1.3 Dateiformate

CSV

- Comma (oder Character) Seperated Values
- Datensätze werden durch einen Zeilenumbruch getrennt
- Datenfelder werden durch einen Character (meist Komma oder Strichpunkt) getrennt
- Kann z.B. mit Excel in eine Tabelle umgewandelt werden.
- Mit ‘ # ‘ können Kommentare eingefügt werden

Beispiel: Aus Filme Übung

- 1;Heidi; N.N; 1952……
- 2;Rambo; Geiler Hawi; 1999; …..

Nachteile:

- Codierung ist unbekannt
- Keine Identifikation, schwierige Erweiterung

Vorteile:

- Sehr einfach (kleinster gemeinsamer Nenner)

INI

- Gleich wie CSV veraltet
- Über eckige Klammern werden Sektionen eingeteilt
- In jeder Sektion gibt eine beliebige Anzahl an key-value Paaren
  - z.B. anzahl=50 wobei „anzahl“ der key und 50 der value ist
- Jede Sektion darf nur einmal vorkommen
- Jeder Schlüssel darf pro Sektion nur einmal vorkommen
- Schlüssel ignorieren Groß-klein Schreibung

Beispiel:

```
Filme.ini
[filme]
Anzahl=99

[film_0]
Id=1
titel=Heidi
hauptdatsteller=…
…

[film_1]
Id=2
titel=Independance Day
hauptdarsteller=Jeff Goldblum
….
….
//Codierung bleibt ungewiss
//Erweiterung leichter möglich
// Kommentare starten mir ;
```

JSON

- Hierarchische Datenablegung ➔ Daten können geschachtelt werden
- Objekte
  - Beginnen mit ‘{‘ und enden mit ‘}‘. Es enthält durch Kommas geteilte ungeordnete Listen von Eigenschaften. Objekte ohne Eigenschaften sind zulässig
- Array
  - Beginnt mit ‘[‘ und endet mit ‘]‘. Es enthält eine durch Kommas geteilte ungeordnete Liste von Werten gleichen oder verschiedenen Typs. Leere Arrays sind zulässig.
- Eigenschaft
  - Besteht aus einem Schlüssel und einem Wert, getrennt durch einen Doppelpunkt (Schlüssel: Wert)

Beispiel:

```
{
  "Herausgeber": "Xema",
  "Nummer": "1234-5678-9012-3456",
  "Deckung": 2e+6,
  "Waehrung": "EURO",
  "Inhaber":
  {
    "Name": "Mustermann",
    "Vorname": "Max",
    "maennlich": true,
    "Hobbys": ["Reiten", "Golfen", "Lesen"],
    "Alter": 42,
    "Kinder": [],
    "Partner": null
  }
}
```

XML

- Hierarchische Datenablegung ➔ Daten können geschachtelt werden
- Elemente
  - Jedes Element startet mit einem Start-Tag <Tagname> und End-Tag </Tagname>
  - Leer-Tag <Tagname/>
- Attribute
  - Zusätzliche Eigenschaften eines Elements
  - Aufbau: Attributname=“Attributwert“
  - Anwendung: <Tagname Attributname=“Attributwert“ />

Beispiel:

```
<?xml version=“1.0“ encoding=“UTF-8“ standalone=”yes”?>
<filme>
  <film>
    <id>1</id>
    <title>Heidi</titel>
  </film>
  <film id=“2“ titel=“independence Day”/>
</filme>
```