Kurzeinführung in die Dateiarbeit in Java

1. Allgemeines

Serialisierung:
Konvertierung eines Objektes einer Anwendung im Hauptspeicher in ein Format, dass in einer Datei gespeichert oder über ein Netzwerk transportiert werden kann.

Persistenz:
Ist die dauerhafte Speicherung von Daten auf einem externen Datenträger.

2. Einfache Variante des Schreibens von Objekten in eine Datei

mit Hilfe der Klasse ObjectOutputStream
Konstruktor:

- *public ObjectOutputStream(OutputStream out) throws IOException*

Der übergebene OutputStream ist das Ziel der Ausgabe.

Um in eine Datei zu schreiben, wird an dieser Stelle eine Instanz der Klasse FileOutputStream verwendet.
Konstruktor:

- *public FileOutputStream(String name) throws FileNotFoundException*

Der übergebene String entspricht dem Dateinamen oder auch dem kompletten Pfad. Existiert dieser nicht, so wird er angelegt.

Methoden der Klasse ObjectOutputStream:
zum Schreiben von Objekten

- *public final void writeObject(Object obj)*

zum Schreiben primitiver Datentypen stehen einige Methoden zur Verfügung, wie zB.:

- *public void writeInt(int data) throws IOException*

Beispiel:

```
try {
    FileOutputStream fileOut = new FileOutputStream( "object.ser" );
    ObjectOutputStream dataOut = new ObjectOutputStream( fileOut );
    dataOut.writeObject(new Object( ));
    dataOut.close();
}//try
catch (IOException e) {
      System.out.println( e );
}//catch
```

Die so angelegte Datei trägt den Namen „object.ser“ . Das Kürzel „.ser“ weist darauf hin, dass in der Datei serialisierte Objekte existieren.

3. Lesen eines Objektes aus einer Datei

Serialisierte Objekte können wieder rekonstruiert werden.
Die Klasse ObjectInputStream macht dies möglich.
Konstruktor:

- *public ObjectInputStream(InputStream in)*

An dieser Stelle kann analog zum Schreiben von Objekten eine Klasse namens FileInputStream verwendet werden.
Konstruktor:

- *public FileInputStream(String name) throws FileNotFoundException*

Methoden:
analog zu ObjektOutputStream gibt es Methoden wie

- *public final Object readObject( )*
- *public int readInt( ) throws IOException* usw.

Beispiel:

```
try {
        FileInputStream fileIn = new FileInputStream( "object.ser" );
        ObjectInputStream dataIn = new ObjectInputStream( fileIn );
        Object obj;
        obj = dataIn.readObject( );
} //try
catch ( ClassNotFoundException e ) {
        System.out.println( e );
} /catch/
catch ( IOException e ) {
        System.out.println( e );
} //catch
```

Oftmals werden Objekte rekonstruiert, die eine Erweiterung der Klasse *Object* sind.
Dann ist an dieser Stelle ein Type Cast notwendig.
Angenommen, eine Instanz der Klasse Vector wurde bereits serialisiert. Die Funktion readObject( ) liefert den Typ *Object* anstatt Vector zurück.

➔ *Vector v = (Vector) dataIn.readObject ( )*

4. Sonstiges

Objekte, die auf diese Weise serialisiert und wieder rekonstruiert werden können sollen, müssen das methodenlose Interface Serializable implementieren.
Die Funktion writeObject durchsucht ein übergebenes Objekt nach Membervariablen, die vom Typ static werden nicht serialisiert. ➔ sie gehören nicht dem Objekt sondern der Klasse des Objekts. Auch überprüft sie, ob das besagte Interface implementiert wird.

Literatur:

Guido Krüger: GoTo Java 2 - Handbuch der Java-Programmierung, 2. Auflage, Addison-Wesley