Tutorium - OOP 2

Input / Output

Agenda

- Ein- und Ausgabe
- Beispiele
- Aufgaben

Ein- und Ausgabe

Ein- und Ausgabeströme

- Zum Verwenden von Datenströmen (Ein- und Ausgabe) muss das Paket `java.io` importiert werden
  - `import java.io.*;`

- Standarddatenströme
  - `System.in` (z.B. Tastatureingabe)
  - `System.out` (z.B. Bildschirmausgabe)
  - `System.err`

- Datenströme müssen nach Verwendung **immer** geschlossen werden!
  - Das funktioniert mit dem `.close()` Befehl!

Ein- und Ausgabe – Basisklassen

- Java trennt bei der Verarbeitung von Ein- und Ausgabeströmen zwischen byteorientierter und zeichenorientierter Verarbeitung
  - byteorientiert: z.B. Verarbeitung von PDF- oder MP3-Dateien
  - zeichenorientiert: z.B Verarbeitung von HTML- oder .txt-Dateien

→ Es gibt jeweils eine Basisklasse für die Ein-/Ausgabe. Ein mal für Bytes, ein mal für Zeichen.

Basisklasse für	Bytes	Zeichen
Eingabe	`InputStream`	`Reader`
Ausgabe	`OutputStream`	`Writer`

Byteweise Ein- und Ausgabe

- Zugriff auf Dateien (file)
  - **File***InputStream* / **File***OutputStream*
- Zugriff auf Eingaben zur Laufzeit (mit internem Puffer)
  - **Buffered***InputStream* / **Buffered***OutputStream*
  - Puffer zur Optimierung der Schreib- und Lesevorgänge
- Lesen und Schreiben von elementaren Werten (`int, float, char …`)
  - **Data***InputStream* / **Data***OutputStream*

Zeichenweise Ein- und Ausgabe

- Streams verarbeiten:

- Bytes aus einem `InputStream` auslesen und in Unicode-Zeichen umwandeln
  - **`Input`**`Stream`***`Reader`***

- Textzeichen aus einer Quelle aus Unicode-Darstellung in Bytefolge umwandeln und in `OutputStream` schreiben
  - **`Output`**`Stream`***`Writer`***

Zeichenweise Ein- und Ausgabe

- Zugriff auf Dateien
  - **File***Reader* / **File***Writer*
- Zugriff mit internem Puffer
  - **Buffered***Reader* / **Buffered***Writer*
- Werte verschiedener Datentypen im Textformat ausgeben
  - **Print***Writer*

Dateien einlesen / ausgeben

```
BufferedReader input =
        new BufferedReader(new FileReader("dateiname.endung"));
```

```
PrintWriter output =
        new PrintWriter(new FileWriter("dateiname.endung"));
```

- Der `FileReader` wird in einen `BufferedReader` "gewrapped" (verpackt)
  - Zweck: Performanteres Verarbeiten des Lesevorgangs
- Der `FileWriter` wird in einen `PrintWriter` verpackt
  - Zweck: Die Ausgaben werden durch den `PrintWriter` als Datentyp-Elemente in die Datei im `FileWriter` geschrieben.

Serialisierung

- Objekte ein- und auslesen
  - **`Object`***`InputStream`* / **`Object`***`OutputStream`*
  - Zweck: Zustand von Objekten extern ablegen und bei Laufzeitstart wieder einlesen

- Damit man ein Objekt einer Klasse auslesen und speichern kann, muss die Klasse das Interface `Serializable` implementieren:

```
public class Konto implements Serializable {
…
}
```

Beispiele

Einlesen einer Textdatei

```java
import java.io.*;

public class BeispielInput {
    public static void main(String[] args) {
        //  -----   Pfad zur Datei, die eingelesen werden soll  -----   //
        String inputPfad = "input.txt";

        //  String in dem die aktuelle Zeile der Datei zwischengespeichert wird
        String zeile;

        //  Reader erzeugen, der Dateininhalt der Inputdatei eingelesen werden kann
        try {
            BufferedReader input = new BufferedReader(new FileReader(inputPfad));   // wirft FileNotFoundException

            while  ( (zeile = input.readLine()) != null) {      //  readLine(); wirft IOException
            //  so lange eine weitere Zeile in der Datei exisitiert gib die aktuelle Zeile der Datei in der Konsole aus.
                System.out.println(zeile);
            }

            //  Datenstrom schließen!!!
            input.close();

        //  Exceptions abfangen und "behandeln"
        } catch (FileNotFoundException fnfe) {
            System.out.println("Fehler! Datei nicht gefunden:");
            System.out.println(fnfe.getMessage());
        } catch (IOException ioe) {
            System.out.println("Fehler! Ein-/Ausgabefehler:");
            System.out.println(ioe.getMessage());
        }
    }
}
```

Schreiben einer Textdatei

```java
import java.io.*;

public class BeispielOutput {
    public static void main(String[] args) {
        //  -----   Pfad zur Datei, die eingelesen werden soll  -----   //
        String outputPfad = "output.txt";


        try {
            //  PrintWriter erzeugen, der in die Outputdatei schreibt
            PrintWriter output = new PrintWriter(new FileWriter(outputPfad));       //  wirft IO.Exception

            //  Mit Printwriter "output" in die Datei schreiben (analog zum Schreiben auf der Konsole!)
            output.println("Zeile 1");
            output.println("Zeile 2");
            int zahl = 1234;
            String text = "Ich bin ein String und nach mir kommt eine Zahl: ";
            output.println(text + zahl);

            //  Nutzer informieren und Datenstrom schließen
            System.out.println("Inhalt wurde in der Datei: " + outputPfad + "gespeichert!");
            output.close();
        } catch (IOException ioe) {
            System.out.println("Fehler! Ein-/Ausgabefehler: ");
            System.out.println(ioe.getMessage());
        }
    }
}
```

Übung

- Erstellen Sie ein Programm, das die Anzahl Zeichen und die Anzahl Zeilen einer Textdatei ermittelt.

Code Beispiele

In der IDE besprochen

Aufgaben

Aufgabe 1

- Erstellen Sie ein Programm zum Verketten von Dateien.
- Der Aufruf von
  - java DateiVerketter <Quelldatei1> [<Quelldatei2> [...]] <Zieldatei>
- soll nacheinander die angegebenen Quelldateien lesen und in die Zieldatei schreiben.

Aufgabe 2

- Die Klasse `Buch` hat die Attribute `titel` und `preis`
  - a) Erstellen Sie ein Programm, das zunächst ein Array von einigen Buch Objekten erzeugt und dann die einzelnen Objekte des Arrays in eine Datei schreibt. Benutzen Sie hierzu geeignete Methoden der Klasse `DataOutputStream`
  - b) Erstellen Sie ein Programm, das die in a) gespeicherten Daten wieder einliest und die einzelnen Objekte in eine Liste vom Typ `ArrayList` sammelt und diese dann anschließend am Bildschirm ausgibt.

Aufgabe 3

- Lösen Sie das in der letzten Aufgabe gestellte Problem, indem Sie das gesamte Array mit `ObjectOutputStream` schreiben und mit `ObjectInputStream` lesen.

Aufgabe 4

- Gegeben sei die Klasse `Konto` mit den Attributen

  ```
  private int kontonummer
  private double saldo
  ```

  sowie den üblichen `Getter und Setter Methoden`.
- Fügen Sie eine Methode hinzu, die die Werte der Instanzvariablen dieses Objekts (`this`) in einen `DataOutputStream` schreibt.
- Erstellen Sie dann einen `Konstruktor`, der die `Instanzvariablenwerte` für das neue Objekt aus einem `DataInputStream` liest.
- Testen Sie diese Methoden.

Ende für heute