Dateien

Zur Arbeit mit Dateien gibt es in Java zwei wichtige Klassen:

java.io.File
java.io.RandomAccessFile

Mittels der Klasse *File* können Dateipfade oder Dateien verwaltet werden. Mittels der Klasse *RandomAccessFile* können Daten in eine Datei geschrieben bzw. aus einer Datei gelesen werden.

Die Klasse *File*

Ein Objekt der Klasse *File* kann eine Datei, einen Pfad oder eine Datei mit Pfadangabe repräsentieren. Zur Erzeugung eines File-Objektes gibt es unterschiedliche Konstruktoren. Im Folgenden seien die wichtigsten Konstruktoren genannt:

1. File(String path_and_file)
2. File(String path, String file)
3. File(File path, String file)

Dem ersten Konstruktor wird nur ein Objekt des Datentyps *String* übergeben. Dabei kann es sich nur um einen Pfadnamen, nur um einen Dateinamen oder um die Kombination eines Pfad- und eines Dateinamens handeln.

Dem zweiten Konstruktor werden der Pfadname und der Dateiname jeweils als String separat übergeben.

Dem dritten Konstruktor werden der Pfad als File-Objekt und der Dateiname als String übergeben.

Beispiel: Dateiname mit relativer Pfadangabe

```
String dirName = "bsp";

String fileName = "Info.dat";

File newInfo = new File(dirName + File.separator + fileName);
```

aktuelles Verzeichnis: c:\projects ⇒ c:\projects\bsp\Info.dat

File.separator ist ein statisches Attribut der Klasse *File* vom Datentyp *String*. Er trennt unterschiedliche Pfadnamen und den Dateinamen und ist in Windows als Backslash (\) und z.B. in Unix als Slash (/) definiert.

Sollte die Datei Info.dat nicht existieren (kann mit der Methode exists() überprüft werden), kann sie mittels createNewFile() erzeugt werden.

Beispiel: Dateiname mit absoluter Pfadangabe

```
String sep = File.separator;
String dirName = "c:" + sep + "projects" + sep + "bsp";
String fileName = "Info.dat";
File newInfo = new File(dirName + sep + fileName);
```

Methoden der Klasse File:

Alle nachfolgenden Operationen beziehen sich auf die vom File-Objekt repräsentierte Datei (evtl. mit Pfad) bzw. den repräsentierten Pfad.

String getName()	Datei- bzw. Pfadname
String getParent()	Name des übergeordneten Verzeichnisses
String getPath()	Pfadname
String getAbsolutePath()	absoluter Pfadname
File[] listRoots()	z.B. C:, D:
boolean canRead()	lässt sich aus der Datei lesen?
boolean canWrite()	kann man in die Datei schreiben?
boolean exist()	existiert der Pfad bzw. die Datei?
boolean isDirectory()	ist es ein Verzeichnis?
boolean isFile()	ist es eine Datei?
boolean isHidden()	ist die Datei versteckt?
long lastModified()	Datum der letzten Änderung
long length()	Dateilänge in Bytes

boolean createNewFile() throws IOException

boolean delete()	löscht die Datei oder das Verzeichnis
boolean mkdir()	erzeugt ein Verzeichnis
boolean renameTo(File dest)	Umbenennung der Datei

Eine detaillierte Beschreibung dieser Methoden entnehmen Sie bitte dem API.

Beispiel:

```
import java.io.*;
class DirFileOperations {
    public static void main(String[] args) {
        String separator = File.separator;
        String dirName = separator + "JavaBsp";
        File dir = new File(dirName);
        if (dir.exists()) {
            System.out.println(dir.getPath() + "existiert schon.");
            }
        else {
            dir.mkdir();
            System.out.println(dir.getPath() + "wurde neu erstellt.");
            }
        String fileName = dirName+separator+"Bsp.txt";
        File file = new File(fileName);
        if (file.exist()) {
            System.out.println(file.getName() + "existiert schon".);
            }
        else {
            try {
                file.createNewFile();
                }
            catch (IOException ex) { . . . }
        .
        .
        .
        }
    }
```

Die Klasse *RandomAccessFile*

Konstruktoren:

RandomAccessFile(File file, String mode)

RandomAccessFile(String fileName, String mode)

Mode: "r" → Datei wird nur zum Lesen geöffnet
"rw" → Datei wird zum Lesen und Schreiben geöffnet

Datei nicht vorhanden: bei "r" → FileNotFoundException
bei "rw" → leere Datei wird erstellt

Methoden zum Schreiben

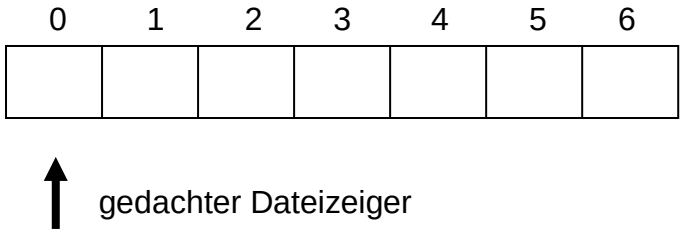
writeChar(int v)	2 Byte
writeBoolean(boolean v)	1 Byte
writeByte(int v)	1 Byte
writeInt(int v)	4 Byte
writeDouble(double v)	8 Byte
. . .	
writeBytes(String s)	Zeichenlänge: 1 Byte
writeChars(String s)	Zeichenlänge: 2 Byte
writeUTF(String s)	modified UTF-8 → zu empfehlen

Methoden zum Lesen

boolean readBoolean()	1 Byte: 0 = false
byte readByte()	1 Byte
int readInt()	4 Byte
double readDouble()	8 Byte
. . .	
String readUTF()	falls mit writeUTF geschrieben → zu empfehlen
readFully(byte[] b)	liest byteweise und schreibt byteweise ins Array b
String readLine()	liest byteweise und wandelt jedes Byte in Char um (High Byte = 0)

UTF ist ein optimiertes Format zum Speichern von Strings (s. z.B. https://de.wikipedia.org/wiki/UTF-8)

Navigieren innerhalb der Datei:



Beim Öffnen einer Datei steht der gedachte Dateizeiger am Dateianfang, also beim Byte 0. Beim Schreiben oder Lesen rückt der Zeiger um die entsprechende Anzahl von Bytes weiter. Geschrieben oder gelesen kann immer nur ab der aktuellen Position des Dateizeigers.

Es gibt in der Klasse *RandomAccessFile* Methoden, mittels derer man den Dateizeiger bewegen kann, ohne zu schreiben oder zu lesen. Weiterhin gibt es Methoden, um die aktuelle Position des Dateizeigers zu erfahren.

seek(long pos)	Setzt den Dateizeiger auf die Position pos (in Byte) ausgehend vom Dateianfang
int skipBytes(int n)	Verschiebt den Dateizeiger um n Bytes nach rechts ausgehend von der aktuellen Position (negatives n ist nicht erlaubt)
long getFilePointer()	aktuelle Zeigerposition (in Byte)

Verschieben nach links: z.B. file.seek(file.getFilePointer() – 10);

ans Dateiende: file.seek(file.length());

Beispiel:

```
RandomAccessFile file;
try {
        file = new RandomAccessFile("test.dat", "rw");
        file.writeBytes("AAAAA\nBBBBB\nCCCCC\n");
        file.seek(0);   // an den Dateianfang
        for (int i = 1; i <=3; i++) {           // 3 Zeilen lesen
                System.out.println(file.readLine());
                }
        file.writeBytes("DDDDD\nEEEEE\nFFFFF\n");
        file.seek(0);
        for (int i = 1; i <=6; i++) {           // 6 Zeilen lesen
                System.out.println(file.readLine());
                }
        file.close();
        }
catch (FileNotFoundException fnf) {
        System.out.println(fnf.getMessage());
        }
catch (IOExceotion io) {
        System.out.println(io.getMessage());
        }
```