Mit Dateien arbeiten

Das Package `java.nio.file`

In den Versionen vor Java 7 stand für die Arbeit mit Dateien und Verzeichnissen das Package java.io zur Verfügung. Dieses ist eines der ersten Packages der Version Java 1.0 und besitzt deshalb im Vergleich zu anderen Packages einige Inkonsistenzen. Die Arbeit mit den Klassen des Packages unterscheidet sich deshalb zum Teil von der mit anderen Packages. So ist beispielsweise die Behandlung der Exception nicht einheitlich bzw. nicht vollständig umgesetzt.

Um hier eine bessere und zeitgemäßere Umsetzung zu bieten, besitzt Java seit der Version 7 ein neues Package: `java.nio.file`. Dieses bietet weitaus umfangreichere Möglichkeiten als die Vorgängerversion und entspricht konsequent den Anforderungen hinsichtlich des Exceptionhandlings. Für die Arbeiten im Zusammenhang mit der Dateiverwaltung stehen drei Einstiegsklassen zur Verfügung:

- `FileSystem`: Diese Klasse dient der Arbeit mit einem Dateisystem. Dabei kann es sich sowohl um ein typisches Betriebssystem-Dateisystem als auch um ein Zip-Archiv handeln.
- `Path`: Objekte dieser Klasse stehen für einen Pfad zu einer Datei oder einem Verzeichnis. Sie bietet keine zusätzlichen Informationen über die Datei und unterstützt keine Dateimanipulationen.
- `Files`: Diese Klasse besitzt unter anderem Methoden zum eigentlichen Dateihandling, wie dem Anlegen, Löschen, Kopieren und Verschieben von Dateien sowie dem Auslesen von Dateiinformationen.

Zur Unterstützung der Weiterverwendung von Programmcode der Vorgängerversionen beinhaltet die Klasse `File` des Packages `java.io` eine Methode `toPath()`.

Die Klasse `Path`

Dateisysteme

In aktuellen Dateisystemen werden Dateien in hierarchischen oder Baumstrukturen organisiert und gespeichert. Die Speicherung erfolgt im Regelfall auf Festplatten. Das Hauptkriterium dabei ist das schnelle Auffinden der einzelnen Dateien. Unter einem oder mehreren Wurzeleinträgen befinden sich die einzelnen Verzeichnisse (in Windows Ordner genannt) und Dateien. Jedes Verzeichnis kann wiederum Dateien und Unterverzeichnisse enthalten. Dies setzt sich über viele Ebenen hinweg fort.

Dateien werden durch die Angabe des Pfades zu ihrer Speicherposition in der Hierarchie eindeutig beschrieben. Der Pfad umfasst dabei den gesamten Weg ausgehend von dem Wurzelelement bis zum Unterverzeichnis, in dem sich die Datei befindet.

Informationen zu Pfad- und Dateinamen

Jedes Betriebssystem besitzt bestimmte Zeichen, die zum Trennen von Datei- und Verzeichnisnamen verwendet werden. Microsoft Windows verwendet den Backslash \,während in Linux der Schrägstrich / genutzt wird. Um den Separator des aktuellen Systems zu ermitteln, stehen Ihnen zwei Möglichkeiten zur Verfügung:

- die Verwendung der Methode `getSeparator()` auf das aktuelle Dateisystem ①,
- die Nutzung des statischen Attributs `separator` ② der Klasse `File`.

```
String separator = FileSystems.getDefault().getSeparator(); ①
String separator = File.separator; ②
```

Weiterhin kann ein Pfad ausgehend von der Wurzel beschrieben werden oder als relative Pfadangabe, die in Java vom aktuellen Benutzerverzeichnis aus interpretiert wird. Während die absolute Pfadangabe alle notwendigen Informationen zum Auffinden einer Datei enthält, wird bei einer relativen Pfadangabe eine zweite Pfadangabe benötigt, bevor die Datei im Dateiverzeichnis gefunden werden kann.

Unter Windows ist beispielsweise Folgendes eine absolute Pfadangabe: *C:\Customer\Info.dat.*

- Die Wurzel ist durch *C:*,
- der Verzeichnispfad ausgehend von der Wurzel durch *\Customer* und
- der Dateiname durch *Info.dat* gekennzeichnet.

Unter Linux z. B. wird in */Customer/Info.dat* die Wurzel durch das Zeichen */* dargestellt.

Ein Objekt der Klasse `Path` erstellen

Bedingt durch die Besonderheiten der verschiedenen Betriebssysteme, ist ein Objekt der Klasse `Path` immer von der Plattform, auf der das Programm läuft, abhängig. Nach der Erstellung des Objekts können Sie es manipulieren, indem Sie Teile des Pfades ausschneiden, den Pfad konvertieren oder ihn mit einem anderen Pfad vergleichen. Um zu prüfen, ob die dem Objekt zugeordnete Datei wirklich existiert, diese gegebenenfalls zu erstellen oder andere Dateioperationen mit ihr durchzuführen, verwenden Sie ein Objekt der Klasse `Files`.

Zum Erstellen eines Objektes der Klasse `Path` verwenden Sie die Methode `get()` der Hilfsklasse `Paths`. Diese ist die Kurzfassung der Methode `getDefault().getPath()` der Klasse `FileSystems`.

```
Path path1 = Paths.get(String pathname) ①
Path path2 = Paths.get(URI.create(String uriname)) ②
```

- Bei Verwendung einer Zeichenkette geben Sie diese direkt oder als Parameter an ①.
- Eine andere Möglichkeit ist die Angabe der Methode `URI.create()` und des entsprechenden Parameters `uriname` ②.

Methoden der Klasse `Path`

In der folgenden Tabelle sind einige häufig verwendete Methoden der Klasse `Path` zusammengestellt.

`Path getFileName()`	Die Methode gibt den Namen oder das Verzeichnis des bestimmten Pfades als `Path`-Objekt zurück.
`FileSystem getFileSystem()`	Der Rückgabewert ist das Dateisystem, welches das betreffende Objekt erstellt hat.
`Path getName(int index)`	Diese Methode gibt in Abhängigkeit vom verwendeten Index einen Namen des Pfades als ein `Path`-Objekt zurück.
`int getNameCount()`	Die Methode ermittelt die Anzahl der Namenelemente des Pfades.
`Path getRoot()`	Die Methode gibt die Wurzelkomponente der Pfadangabe zurück. Besitzt diese keine, ist der Rückgabewert `null`.
`Path getParent()`	Als Rückgabewert wird der Elternpfad der aktuellen Pfadbezeichnung zurückgegeben. Ist diese nicht vorhanden, ist der Rückgabewert `null`.
`boolean isAbsolute()`	Die Methode liefert eine Aussage, ob es sich um eine absolute Pfadangabe handelt.
`File toFile()`	Die Methode gibt entsprechend der Pfadangabe ein Objekt der Klasse `File` zurück.
`String toString()`	Der Rückgabewert ist die Darstellung der aktuellen Pfadangabe als Zeichenkette.

Beispiel: *com\herdt\java9\kap16\PathOperations.java*

Das Beispiel zeigt die Anwendung einiger Methoden der Klasse `File`.

```
package com.herdt.java9.kap16;

import java.nio.file.*;

class PathOperations
{
  public static void main(String[] args){
    Path path = Paths.get("C:\\java9\\uebungen\\kap04");

    System.out.format("getFileName: %s%n", path.getFileName());
    System.out.format("getFileSystem: %s%n",
      path.getFileSystem());
    System.out.format("getName(1): %s%n", path.getName(1));
    System.out.format("getNameCount: %d%n", path.getNameCount());
```

```
    System.out.format("getRoot: %s%n", path.getRoot());
    System.out.format("getParent: %s%n", path.getParent());
    System.out.format("isAbsolute: %b%n", path.isAbsolute());
    System.out.format("toString: %s%n", path.toString());
  }
}
```

Programmausgabe

```
getFileName: kap04
getFileSystem: sun.nio.fs.WindowsFileSystem@22f71333
getName(1): uebungen
getNameCount: 3
getRoot: C:\
getParent: C:\java9\uebungen
isAbsolute: true
toString: C:\java9\uebungen\kap04
```

Programmausgabe

Die Klasse `Files`

Was ist die Klasse `Files`?

Neben der Klasse `Path` ist `Files` eine weitere Hauptklasse zur Nutzung des Packages `java.nio.file`. Die Klasse besitzt eine Vielzahl von statischen Methoden. Durch deren Verwendung können Sie Dateien und Verzeichnisse lesen, schreiben und verändern. Dafür verwenden die Methoden Instanzen der Klasse `Path`.

Bei der Anwendung der Methoden müssen Sie zwei Dinge beachten:

- Die genutzten Ressourcen müssen wieder geschlossen werden, da diese das Interface `java.io.Closeable` implementieren, welches das erfordert. Durch die Verwendung der Anweisung `try-with-ressources` erfolgt das Schließen automatisch.
- Bei der Arbeit mit I/O-Operationen können immer Exceptions auftreten. Diese müssen Sie in einem `try-with-ressources` oder einer Standard `try-catch`-Anweisung abfangen.

Die Methoden der Klasse `Files`

In den folgenden Tabellen sind einige häufig verwendete Methoden der Klasse `Files` zusammengestellt.

Datei- und Verzeichnisinformationen

`boolean exists(Path path, LinkOption... options)`	Die Methode prüft, ob eine Datei existiert.
`boolean isDirectory(Path path, LinkOption... options)`	Die Methode testet, ob es sich um ein Verzeichnis handelt.
`boolean isExecutable(Path path)`	Die Methode prüft, ob die Datei ausführbar ist.

`boolean isHidden(Path path)`	Die Methode testet, ob die Datei verborgen ist.
`boolean isReadable(Path path)`	Die Methode prüft, ob die Datei lesbar ist.
`boolean isRegularFile(Path path, LinkOption... options)`	Die Methode testet, ob es sich um eine reguläre Datei handelt.
`boolean isSameFile(Path path, Path path2)`	Die Methode prüft, ob es sich bei zwei Pfadangaben um dieselbe Datei handelt.
`boolean isWritable(Path path)`	Die Methode testet, ob es möglich ist, in die Datei zu schreiben.
`Object getAttribute(Path path, String attribute, LinkOption... options)`	Die Methode liefert den Wert eines Fileattributes.
`FileTime getLastModifiedTime(Path path, LinkOption... options)`	Die Methode gibt die Zeit der letzten Dateiänderung zurück.
`UserPrincipal getOwner(Path path, LinkOption... options)`	Die Methode liefert als Rückgabewert den Besitzer der Datei.
`boolean notExists(Path path, LinkOption... options)`	Die Methode prüft, ob eine Datei nicht existiert.
`Map<String,Object> readAttributes(Path path, String attributes, LinkOption... options)`	Die Methode liefert die Werte mehrerer Fileattribute.
`long size(Path path)`	Die Größe der Datei wird ermittelt.

Datei- und Verzeichnisoperationen

`Path copy(Path source, Path target, CopyOption... options)`	Die Methode kopiert eine Datei.
`Path createDirectories(Path dir, FileAttribute<?>... attrs)`	Die Methode erstellt ein Verzeichnis. Fehlende Elternverzeichnisse werden dabei zuerst erstellt.
`Path createDirectory(Path dir, FileAttribute<?>... attrs)`	Die Methode erstellt ein Verzeichnis.
`Path createFile(Path path, FileAttribute<?>... attrs)`	Die Methode erstellt eine neue leere Datei. Existiert diese bereits, tritt ein Fehler auf.
`void delete(Path path)`	Die Methode löscht eine Datei.
`boolean deleteIfExists(Path path)`	Die Methode löscht eine Datei. Existiert diese nicht, wird kein Fehler erzeugt.
`Path move(Path source, Path target, CopyOption... options)`	Die Methode verschiebt eine Datei oder benennt diese um.
`Path setLastModifiedTime(Path path, FileTime time)`	Die Methode aktualisiert die Zeit des letzten Zugriffs auf eine Datei.
`Path setOwner(Path path, UserPrincipal owner)`	Die Methode aktualisiert den Besitzer der Datei.

Beispiel: *com\herdt\java9\kap16\DirFileOperations.java*

Das Beispiel zeigt die Anwendung einiger Methoden der Klasse `File`.

```
   package com.herdt.java9.kap16;

   import java.io.*;
   import java.nio.file.*;

   class DirFileOperations
   {
     public static void main(String[] args){
①      Path file =
         Paths.get("C:\\java9\\uebungen\\kap16\\uebung.txt");
       Path dir =
         Paths.get("C:\\java9\\test\\kap16\\uebung.txt");
②      if (Files.exists(file))
       {
         System.out.format(file.toString() + " existiert.");
       }
③      else
       {
④        try
         {
⑤          Files.createDirectories(file.getParent());
⑥          Files.createFile(file);
⑦          Files.createDirectories(dir.getParent());
⑧          Files.copy(file, dir);
⑨          Files.delete(file);
         }
④         catch (IOException ex) {
           System.err.println(ex);
         }
       }
     }
   }
```

① Es werden zwei `Path`-Objekte erzeugt.

② Es wird geprüft, ob die Datei existiert.

③ Existiert die Datei noch nicht, werden mit der Datei verschiedene Operationen ausgeführt.

④ Um mögliche Exceptions auffangen zu können, müssen die Methoden in einem `try-catch`-Block ausgeführt werden. Um das Beispiel übersichtlich zu halten, wird lediglich auf eine `IOException` geprüft. Jede der verwendeten Methoden kann jedoch spezielle Exceptions werfen.

⑤ Das Verzeichnis für die erste Datei wird erstellt.

⑥ Die erste Datei wird erstellt.

⑦ Das Zielverzeichnis für das Kopieren wird erstellt.

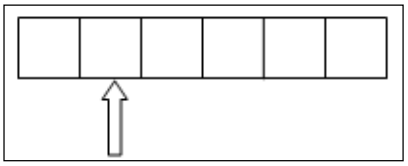
⑧ Die erste Datei wird in das Zielverzeichnis kopiert.
⑨ Die zuerst erstellte Datei wird gelöscht.

Wahlfreier Zugriff auf Dateien

Was bedeutet wahlfreier Zugriff?

Dateien, auf die ein wahlfreier Zugriff (random access) möglich ist, werden als Random Access Files bezeichnet. Für die Arbeit mit Random Access Files stellt Java seit der Version 7 das Interface `SeekableByteChannel` zur Verfügung.

Ein **Dateizeiger (Pointer)** verweist auf die aktuelle Position innerhalb der Datei. Er kann vorwärts bewegt und an eine bestimmte Position gesetzt werden, sodass eine beliebige Stelle der Datei gelesen bzw. geschrieben werden kann. Zum Bearbeiten einer Datei öffnen Sie diese, positionieren den Dateizeiger und lesen oder schreiben an der gewünschten Position.



Random Access File mit Dateizeiger

Methoden des Interface `SeekableByteChannel` für die Arbeit mit wahlfreiem Zugriff

Folgende Methoden ermöglichen Ihnen das Schreiben und Lesen von Daten an einer definierten Position:

`long position`	Die aktuelle Position des Dateizeigers wird zurückgegeben.
`SeekableByteChannel position(long)`	Die Methode setzt den Dateizeiger an die bestimmte Position.
`int read(ByteBuffer)`	Die Methode liest Bytes aus der `FileChannel`-Instanz in den Bytepuffer.
`int write(ByteBuffer)`	Die Methode schreibt den Inhalt des Bytepuffers in die `FileChannel`-Instanz.

Beispiel: *com\herdt\java9\kap16\RandomAccess.java*

Das Beispiel zeigt das Schreiben in eine neu erstellte Datei mittels wahlfreiem Zugriff.

```
   package com.herdt.java9.kap16;
①  import java.io.*;
   import java.nio.*;
   import java.nio.channels.*;
   import java.nio.file.*;
   import static java.nio.file.StandardOpenOption.*;

   class RandomAccess
   {
     public static void main(String[] args){
       Path file =
         Paths.get("C:\\java9\\uebungen\\kap16\\random.txt");
       String s = ". Zeile!\r\n";
       byte data[];
       ByteBuffer out;
②      if (Files.exists(file))
       {
         System.out.format(file.toString() + " existiert.");
       }
       else
       {
③        try
         {
④          Files.createDirectories(file.getParent());
           Files.createFile(file);
⑤          try (FileChannel fc =
             (FileChannel.open(file, READ, WRITE)))
           {
⑥            fc.position(0);
⑦            for (int i = 0; i  < 5; i++)
             {
⑧              data = ((i+1) + s).getBytes();
⑨              out = ByteBuffer.wrap(data);
⑩              while (out.hasRemaining())
                 fc.write(out);
               out.rewind();
             }
           }
```

```
⑤           catch (IOException x)
            {
              System.out.println("I/O Exception: " + x);
            }
        }
③       catch (IOException ex)
        {
          System.err.println(ex);
        }
    }
  }
}
```

① Die benötigten Packages werden importiert.

② Wenn die Datei bereits existiert, wird das Programm nach Ausgabe eines Hinweises beendet.

③ Die Aktionen zum Erstellen des Verzeichnisses und der Datei werden in einem `try-catch`-Block eingeschlossen, um mögliche Exceptions aufzufangen.

④ Das Verzeichnis und die Datei werden erstellt.

⑤ Die Datei wird als Instanz der Klasse `FileChannel` zum Lesen und Schreiben geöffnet. Dies erfolgt in einem `try-with-ressources`-Block, sodass die Ressource automatisch wieder geschlossen wird.

⑥ Der Dateizeiger wird an den Anfang der Datei gestellt.

⑦ Es sollen fünf Zeilen in die Datei geschrieben werden. Die Steuerung der Ausgabe erfolgt über eine `for`-Schleife.

⑧ Der Inhalt der Zeichenkette wird mit der Methode `getBytes()` in ein Byte-Array konvertiert.

⑨ Das Byte-Array wird in einen Bytepuffer übertragen.

⑩ Der Inhalt des Bytepuffers wird in die Dateiressource geschrieben.

16.2 Übung

Arbeit mit Daten

Level		Zeit	ca. 20 min
Übungsinhalte	✔ Dateien lesen und schreiben ✔ Puffer und Arrays anwenden		
Übungsdatei	*Alphabet.txt*		
Ergebnisdatei	*Exercise.java*		

1. Schreiben Sie ein Programm, das per Random Access den Inhalt der Datei *Alphabet.txt* (die Zeichenkette *abcdefghijklmnopqrstuvwxyz*) in einen Puffer einliest.
2. Positionieren Sie den Dateizeiger auf den Dateianfang.
3. Fügen Sie den Text *Das umgekehrte Alphabet* - in die Datei ein.
4. Positionieren Sie den Dateizeiger auf das Dateiende.
5. Ändern Sie unter Verwendung eines Hilfsarrays den Inhalt des Puffers so, dass die Reihenfolge der Zeichen umgekehrt wird (zu: *zyxwvutsrqponmlkjihgfedcba*).
6. Fügen Sie den Inhalt des Puffers wieder in die Datei ein.