Übungen zu Kapitel 22

22.1 read() in Reader und in InputStream:

In der Klasse `Reader` werden die beiden Methoden

```
public int read()
public int read( char[] cbuf )
```

mithilfe der folgenden Methode formuliert:

```
public abstract int read( char[] cbuf, int off, int len )
```

Daher müssen in einer konkreten `Reader`-Klasse die ersten beiden Methoden nicht implementiert werden und sind deshalb nicht mit `abstract` gekennzeichnet. Geben Sie eine mögliche Definition der beiden Methoden mithilfe der abstrakten an.

Im Gegensatz dazu ist in der Klasse `InputStream` die parameterlose `read()`-Methode abstrakt:

```
public abstract int read()
```

Definieren Sie:

```
public int read( byte[] b )
public int read( byte[] b, int off, int len )
```

mithilfe der parameterlosen Methode.

Anmerkung: Grundsätzlich gilt für alle `read()`-Methoden mit Parameter: Können weniger Zeichen/Bytes in das Array gelesen werden als dort hinein passen, wird die Anzahl der gelesen Zeichen/Bytes zurückgegeben. Erst der nächste Leseversuch signalisiert mit dem Rückgabewert -1 das Ende des Stroms.

22.2 Abfangen von IOException:

Ändern Sie die Klasse `CopyC`, indem der Zusatz `throws IOException` gestrichen und Ausnahmen dieser Klasse abgefangen werden.

22.3 Lesen in Arrays:

Ändern Sie die Klassen `CopyC` und `CopyB`, so dass zum Lesen und Schreiben jeweils ein `Char`- bzw. `Byte`-Array der Länge 128 verwendet wird.

Erstellen Sie zum Test für das erfolgreiche Kopieren eine Vergleichsmethode

```
static boolean fileCompare( String f1, String f2 )
```

die die Dateien `f1` und `f2` auf ihre Inhalte vergleicht. Sind beide Dateien (inhaltlich) identisch, wird `true` zurückgegeben, andernfalls `false`. Ausnahmen sollen nicht abgefangen werden, etwa wenn eine Datei unter dem angegebenen Namen nicht existiert oder nicht geöffnet werden kann.

Tipp: Öffnen Sie entsprechende Instanzen von `FileInputStream` mit `f1` und `f2` als Quelle und vergleichen Sie iterativ die gelesenen Bytes.

22.4 Schreiben und Lesen von unterschiedlichen Objekten:

Modifizieren Sie die Klasse `WriteAndReadDate`, so dass nicht nur eine Instanz von `Date`, sondern auch andere Objekte (z.B. Instanzen von `String` und von `Integer`) in den Strom geschrieben und wieder gelesen werden. Was ist bei der Schreib- und Lesereihenfolge zu beachten? Können ohne weiteres Instanzmethoden der gelesenen Objekte (etwa `intValue()` bei `Integer`) aufgerufen werden?

22.5 Persistenz bei primitiven Datentypen:

Erstellen Sie eine ausführbare Klasse `WriteAndReadPrimitives`, die beispielhaft einige Werte mit den Methoden `writeInt()`, `writeChar()`, `writeLong()`,... schreibt und mit den korrespondierenden `read`-Methoden liest. Zeigen Sie, wie man grundsätzlich auf die "primitiven" Schreib- und Lesemethoden verzichten und deren Funktionalität mit den Methoden `writeObject()` und `readObject()` bewerkstelligen kann.

22.6 Die Klasse **`File`**

Verifizieren Sie, dass man durchaus `File`-Objekte erzeugen kann, zu denen keine Datei oder Verzeichnis existiert. Dies erklärt auch, warum es eine Methode `public boolean exists()` gibt.

Erstellen Sie in einer Testklasse eine Methode

```
static void showInfo( String dirname )
```

die die wichtigsten Eigenschaften der Dateien ausgibt, die im angegebenen Verzeichnis liegen, und die rekursiv alle Verzeichnisse durchläuft. Führen Sie einige Tests durch.

Die Eigenschaften erhält man durch folgende Methoden:

```
String getName()
String getPath()
boolean canRead()
boolean canWrite()
boolean isHidden()
long length()
long lastModified()
boolean isFile()
boolean isDirectory()
```

Verwenden Sie dazu die Methode `listFiles()`.