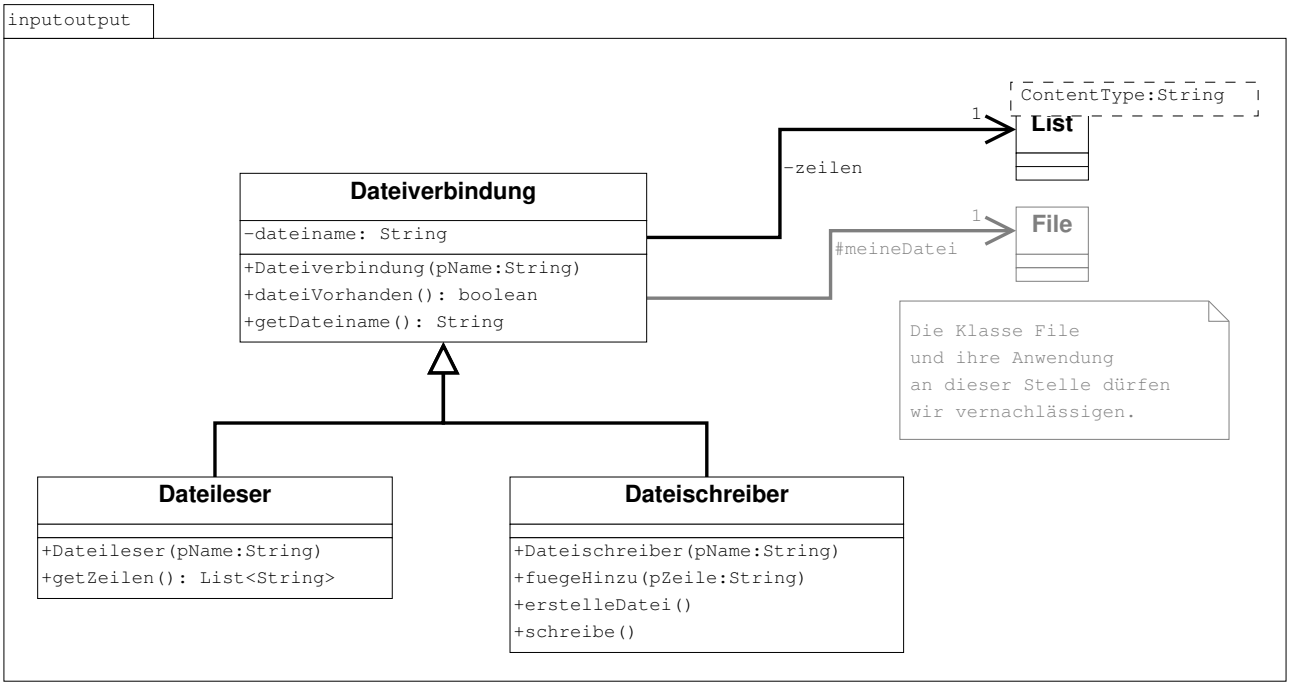
Unsere Bibliothek

Lesen und Schreiben von Dateien mit dem Paket **inputoutput**



Dokumentation der Klasse `Dateiverbindung`
Ein Objekt der Klasse **`Dateiverbindung`** stellt eine Verbindung zu einer Datei her. Man kann erfragen, ob die angegebene Datei tatsächlich existiert.
Sie dient als Oberklasse der Klassen **`Dateileser`** und **`Dateischreiber`**, die weitere Funktionen bieten.

Konstruktor

`Dateiverbindung(String pDateiname)`
Eine neue Verbindung zu der Datei mit dem Namen `pDateiname` wird erstellt. Wird nur ein Name ohne Pfad angegebenen, wird sich auf den Projektordner bezogen.

Methoden

`boolean` `dateiVorhanden()`
Liefert `true`, wenn die dem Konstruktor übergebene Datei existiert. Andernfalls wird `false` zurückgegeben.

`String` `getDateiname()`
Liefert den Namen der verbundenen Datei, unabhängig davon, ob sie tatsächlich existiert.

Dokumentation der Klasse `Dateileser`
Dies ist eine Unterklasse von **`Dateiverbindung`**.
Ein Objekt der Klasse **`Dateileser`** stellt eine Verbindung zu einer Datei her. Man kann damit die Zeilen aus einer Textdatei auslesen.

Konstruktor

`Dateileser(String pDateiname)`
Eine neue Verbindung zu der Datei mit dem Namen `pDateiname` wird erstellt. Wird nur ein Name ohne Pfad angegebenen, wird sich auf den Projektordner bezogen.

Methoden

`List<String> getZeilen()`
Liefert eine Liste, die die Zeilen der verbundenen Datei als Strings enthält. Sollte die verbundene Datei nicht existieren, wird eine entsprechende Meldung als einziger String in der Liste zurückgegeben.

Beispielanwendung

```
import datenstrukturen.List;
import inputoutput.Dateileser;

public class BeispielDateileser {

    public static void main(String[] args) {

        // Die Datei Demo.txt befindet sich im Projektordner.
        Dateileser oeffner = new Dateileser("Demo.txt");

        // Die Liste enthaelt alle Zeilen der Textdatei Demo.txt.
        List<String> text = oeffner.getZeilen();

        // Mit einer Schleife werden die Zeilen ausgegeben.
        text.toFirst();
        while (text.hasAccess()) {
            System.out.println(text.getContent());
            text.next();
        }

    }

}
```

Dokumentation der Klasse `Dateischreiber`
Dies ist eine Unterklasse von **`Dateiverbindung`**.
Ein Objekt der Klasse **`Dateischreiber`** stellt eine Verbindung zu einer Datei her. Man kann damit Strings in eine Textdatei schreiben. Die Strings werden zunächst intern in einer Liste abgelegt. Erst bei Ausführung der Methode `schreibe()` wird tatsächlich versucht, sie in eine Textdatei zu schreiben.

Konstruktor

`Dateischreiber(String pDateiname)`
Eine neue Verbindung zu der Datei mit dem Namen `pDateiname` wird erstellt. Wird nur ein Name ohne Pfad angegebenen, wird sich auf den Projektordner bezogen.

Methoden

`void` `fuegeHinzu(String pZeile)`
Fügt der Liste mit den zu schreibenen Zeilen den String `pZeile` als weitere Zeile hinzu.

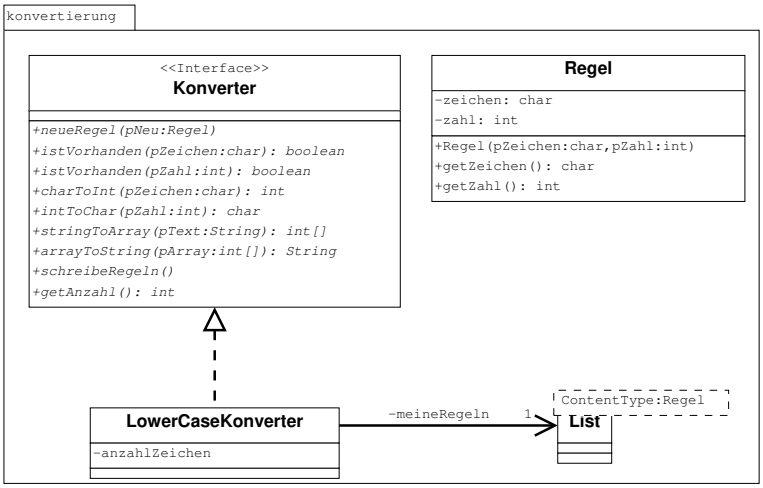
`boolean` `erstelleDatei()`
Falls die Datei mit dem angegebenen Dateinamen noch nicht existiert, wird versucht sie anzulegen. Bei Erfolg wird zur Bestätigung `true` geliefert, andernfalls `false`.

`void` `schreibe()`
Versucht, die in der internen Liste gespeicherten Strings in die verbundene Textdatei zu schreiben. Falls diese Datei nicht existiert, kann dies nicht durchgeführt werden.

Beispielanwendung

```
import inputoutput.Dateischreiber;

public class BeispielDateischreiber {

    public static void main(String[] args) {

        // Die Datei muss noch nicht existieren.
        Dateischreiber schreiber = new Dateischreiber("Demo.txt");

        // Die Zeilen werden zunaechst zwischengespeichert.
        schreiber.fuegeHinzu("Hallo!");
        schreiber.fuegeHinzu("Das ist eine Demo.");

        // Existiert die Datei noch nicht, wird sie nun erzeugt.
        if (!schreiber.dateiVorhanden()) {
            schreiber.erstelleDatei();
        }

        // Die Zeilen werden in die Datei geschrieben.
        schreiber.schreibe();
    }

}
```

Konvertieren von Text in Zahlenwerte mit dem Paket **konvertierung**



Dokumentation des Interfaces `Konverter`
Ein Konverter wandelt nach vorher festgelegten Regeln `char`- in `int`-Werte bzw. `int`- in `char`-Werte um. Außerdem wandelt er nach denselben Regeln Strings in eindimensionale `int`-Arrays bzw. eindimensionale `int`-Arrays in Strings um.

Methoden

`void`	`neueRegel(Regel pNeue)` Fügt den Konvertierungsregeln die Regel `pNeue` hinzu, sofern noch keine Regel mit dem `char`- oder dem `int`-Wert von `pNeue` existiert.
`boolean`	`istVorhanden(char pZeichen)` Liefert `true`, falls bereits eine Regel mit dem Schriftzeichen `pZeichen` hinzugefügt wurde. Andernfalls wird `false` geliefert.
`boolean`	`istVorhanden(int pZahl)` Liefert `true`, falls bereits eine Regel mit der Zahl `pZahl` hinzugefügt wurde. Andernfalls wird `false` geliefert.
`int`	`charToInt(char pZeichen)` Liefert den laut den gespeicherten Regeln zu `pZeichen` passenden `int`-Wert.
`char`	`intToChar(int pZahl)` Liefert das laut den gespeicherten Regeln zu `pZahl` passende Schriftzeichen.
`int[]`	`stringToArray(String pText)` Liefert das laut den gespeicherten Regeln zu `pText` passende Array.
`String`	`arrayToString(int[] pArray)` Liefert den laut den gespeicherten Regeln zu `pArray` passenden String.
`void`	`schreibeRegeln()` Gibt alle Regeln in der Konsole aus.
`int`	`getAnzahl()` Liefert die aktuelle Anzahl von Umwandlungsregeln.

Dokumentation der Klasse `LowerCaseKonverter`
Dies ist eine Implementierung des Interfaces **`Konverter`**.

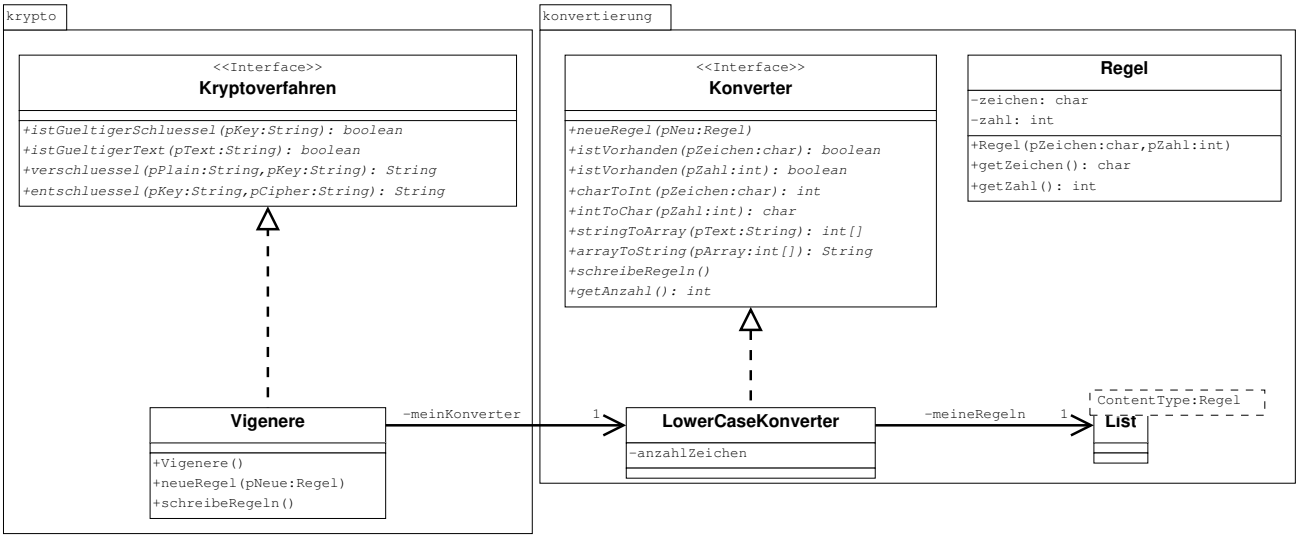
Konstruktor

`LowerCaseKonverter()`
Eine neue Instanz mit den 26 Regeln $a \leftrightarrow 0, b \leftrightarrow 1, \ldots, z \leftrightarrow 25$ wird erstellt.
Die Regeln werden intern in einem Objekt der Klasse **`List<Regel>`** verwaltet.

Beispielanwendung

```
import konvertierung.Konverter;
import konvertierung.LowerCaseKonverter;
import konvertierung.Regel;

public class BeispielLowerCaseKonverter {

    public static void main(String[] args) {

        // Der konkrete Typ des Konverters muss nur reschts angegeben werden.
        Konverter konverter = new LowerCaseKonverter();

        // Alle Konvertierungsregeln werden ausgegeben.
        konverter.schreibeRegeln();
        /* Ausgabe:
        a 0
        b 1
        c 2
        d 3
        ...
        z 25
        */

        // Eine neue Regel wird hinzugefuegt.
        konverter.neueRegel(new Regel(' ', 26));

        // Ein String wird in eine Array konvertiert.
        int[] zahlen = konverter.stringToArray("das ist ein test");

        // Das Array wird ausgegeben.
        for (int i = 0; i < zahlen.length; i = i + 1) {
            System.out.print(zahlen[i] + " ");
        }
        // Ausgabe: 3 0 18 26 8 18 19 26 4 8 13 26 19 4 18 19

        // Das Array wird in einen String konvertiert und ausgegeben.
        System.out.println(konverter.arrayToString(zahlen));
        // Ausgabe: das ist ein test

    }

}
```

Verschlüsseln mit dem Paket `krypto`



Dokumentation des Interfaces `Kryptoverfahren`
Dieses Interface gibt einige Methoden zur Umsetzung von Verfahren der Kryptographie vor.
Für mehr Flexibilität werden die Schlüssel als Strings angegeben, da bei einigen Verfahren der Schlüssel nicht nur aus einer Zahl sondern z.B. auch aus einem Wort bestehen kann.

Methoden

`boolean` `istGueltigerSchluessel(String pKey)`
Falls `pKey` ein formal gültiger Schlüssel ist, wird `true` zurückgegeben, ansonsten `false`.

`String` `verschluessel(String pPlain, String pKey)`
Der Klartext `pPlain` wird mit dem Schlüssel `pKey` verschlüsselt. Der Cipher-Text wird zurückgegeben.

`String` `entschluessel(String pCipher, String pKey)`
Der Ciphertext `pCipher` wird mit dem Schlüssel `pKey` entschlüsselt. Der entschlüsselte Text wird zurückgegeben.

Dokumentation der Klasse `Vigenere`
Dies ist eine Implementierung von `Kryptoverfahren`. Ein Objekt dieser Klasse ver- und entschlüsselt Texte gemäß dem Vigenère-Verfahren. Zunächst können nur Texte und Schlüssel die ausschließlich aus Kleinbuchstaben bestehen verwendet werden. Es gibt aber die Möglichkeit, den Zeichensatz zu erweitern.
Ein Objekt dieser Klasse besitzt einen Konverter. Mit diesem werden die gegebenen Strings in `int`-Arrays konvertiert. Auf diesen Arrays finden dann die Rechnungen zum Ver- und Entschlüsseln statt.
Ob ein Text oder Schlüssel formal gültig ist, wird mithilfe des Konverters entschieden. Text und Schlüssel gelten als formal gültig genau dann, wenn alle enthaltenen Zeichen in den Regeln des Konverters vorhanden sind.

Konstruktor

`Vigenere()`
Eine Instanz mit Zeichensatz a, b, …, z wird erstellt.

Methoden

`void` `neueRegel(Regel pNeue)`
Dem Konverter wird die neue Regel `pNeue` hinzugefügt, sofern noch keine Regel mit dem Zeichen oder dem `int`-Wert von `pNeue` vorliegt. Dadurch kann der Zeichensatz erweitert werden.

`void` `schreibeRegel()`
Alle im Konverter vorhandenen Regeln werden in der Konsole ausgegeben.

Beispielanwendung

```
import konvertierung.Regel;
import krypto.Kryptoverfahren;
import krypto.Vigenere;

public class BeispielVigenere {
    public static void main(String[] args) {
        // Eine Instanz wird erstellt.
        Kryptoverfahren meinVerfahren = new Vigenere();

        // Es wird geprueft, ob man alibaba als Schluessel verwenden darf.
        System.out.println(meinVerfahren.istGueltigerSchluessel("alibaba"));
        // Ausgabe: true

        // Es wird geprueft, ob man Alibaba als Schluessel verwenden darf.
        System.out.println(meinVerfahren.istGueltigerSchluessel("Alibaba"));
        // Ausgabe: false

        // Ein Text wird verschluesselt.
        String cipher = meinVerfahren.verschluessel("dasisteintest", "ab");
        System.out.println(cipher);
        // Ausgabe: dbsjsuejnuett

        // Der Text wird entschluesselt.
        String plain = meinVerfahren.entschluessel(cipher, "ab");
        System.out.println(plain);
        // Ausgabe: dasisteintest

        System.out.println(meinVerfahren.entschluessel(cipher, "cd"));
        // Ausgabe: byqgqrcglrcqr

        // Eine neue Regel wird ergaenzt.
        meinVerfahren.neueRegel(new Regel('A', 26));
        meinVerfahren.schreibeRegeln();
        /*
        Ausgabe:
        a 0
        b 1
        ...
        z 25
        A 26
        */

        // Es wird geprueft, ob man Alibaba als Schluessel verwenden darf.
        System.out.println(meinVerfahren.istGueltigerSchluessel("Alibaba"));
        // Ausgabe: true

        // Ein Text wird verschluesselt.
        cipher = meinVerfahren.verschluessel("dasisteintest", "Alibaba");
        System.out.println(cipher);
        // Ausgabe: clAjsuehyafsu

        // Der Text wird entschluesselt.
        System.out.println(meinVerfahren.entschluessel(cipher, "alibaba"));
        // Ausgabe: casistehntest

        System.out.println(meinVerfahren.entschluessel(cipher, "Alibaba"));
        // Ausgabe: dasisteintest
    }
}
```