

Standard-Dialoge

Die Klasse `JOptionPane` besitzt die Methoden:

```
public static void showMessageDialog(Component parentComponent, Object message)
public static String showInputDialog(Component parentComponent, Object message)
public static int showConfirmDialog(Component parentComponent, Object message)
```

Bedeutung der Parameter:

`parentComponent`: Vater-Fenster (das Frame)
`message`: Text der Nachricht (String)

Mögliche Rückgabewerte von `showConfirmDialog()`:

- `JOptionPane.YES_OPTION`
- `JOptionPane.NO_OPTION`
- `JOptionPane.CANCEL_OPTION`

Fehlerbehandlung

```
try {
    // Überwachter Code
} catch (Exception e) {
    System.out.println(e.getMessage()); // Ausgabe der Fehlermeldung
    e.printStackTrace();               // Ausgabe der Fehlerzeile
}
```

Hilfsfunktionen

Umwandlung von Strings in Zahlen

```
int zahl = Integer.parseInt(text); // text: Variable vom Typ String
double zahl2 = Double.parseDouble(text);
```

Wenn `text` keine gültige Zahl enthält, wird eine `NumberFormatException` erzeugt.

Hilfsfunktionen der Klasse `Character`

Methode	Bedeutung
<code>static boolean isDigit(char ch)</code>	Prüft, ob das angegebene Zeichen eine Ziffer zwischen '0' und '9' ist.
<code>static boolean isLetter(char ch)</code>	Prüft, ob das angegebene Zeichen ein Buchstabe ist.
<code>static boolean isSpaceChar(char ch)</code>	Prüft, ob das angegebene Zeichen ein Leerzeichen ist.
<code>static boolean isLowerCase(char ch)</code>	Prüft, ob das angegebene Zeichen ein Kleinbuchstabe ist.
<code>static boolean isUpperCase(char ch)</code>	Prüft, ob das angegebene Zeichen ein Großbuchstabe ist.
<code>static char toLowerCase(char ch)</code>	Wandelt das angegebene Zeichen in einen Kleinbuchstaben um.
<code>static char toUpperCase(char ch)</code>	Wandelt das angegebene Zeichen in einen Großbuchstaben um.

Hilfsfunktionen der Klasse `String`

Methode	Bedeutung
<code>char charAt(int index)</code>	Gibt das Zeichen an einer bestimmten Position zurück (das erste Zeichen hat die Nummer 0)
<code>boolean contains(String s2)</code>	Ist <code>s2</code> im gegebenen <code>String</code> -Objekt enthalten?
<code>boolean equals(String s2)</code>	vergleicht den <code>String</code> mit dem Inhalt von <code>s2</code>
<code>boolean equalsIgnoreCase(String s2)</code>	vergleicht den <code>String</code> mit dem Inhalt von <code>s2</code> ohne Berücksichtigung der Groß- und Kleinschreibung.
<code>int indexOf(int ch)</code>	Ermittelt den Index (= die Positionsnummer) eines Zeichens. Falls das Zeichen nicht existiert, wird -1 zurück gegeben.
<code>int length()</code>	Gibt die Länge eines Strings zurück.
<code>String substring(int beginIndex, int endIndex)</code>	Gibt die Zeichenkette von der Position <code>beginIndex</code> bis zur Position (<code>endIndex - 1</code>) zurück.
<code>String toUpperCase()</code>	Gibt den <code>String</code> komplett in GROSSBUCHSTABEN zurück
<code>String toLowerCase()</code>	Gibt den <code>String</code> komplett in kleinbuchstaben zurück

Zufallszahlen

```
import java.util.*;
```

Man muss ein Objekt der Klasse Random erzeugen (der Konstruktor besitzt keine Parameter).
Anschließend kann man mit folgender Methode der Klasse eine Zufallszahl erzeugen:

```
public int nextInt(int n)    // gibt eine Zahl zwischen 0 und (n-1) zurück
```

Zugriff auf Dateien

```
import java.net.URL;
```

```
import java.io.*;
```

Klassen **FileInputStream**, **FileOutputStream**, **InputStreamReader**, **OutputStreamWriter**

Datei öffnen zum lesen

```
public FileInputStream(String fileName) throws FileNotFoundException
```

```
public InputStreamReader(FileInputStream fileIn, String charsetName)  
    throws UnsupportedOperationException
```

Beispiel (die zu öffnende Datei muss bereits existieren!):

```
URL url = getClass().getResource("datei.txt");           // import java.net.URL  
InputStream fileIn = new FileInputStream(url.getFile());  
InputStreamReader in = new InputStreamReader(fileIn, "UTF-8");
```

Datei öffnen zum schreiben

```
public FileOutputStream(String fileName, boolean append) throws IOException
```

```
public OutputStreamWriter(FileOutputStream fileOut, String charsetName)  
    throws UnsupportedOperationException
```

Beispiel (die zu öffnende Datei muss bereits existieren!):

```
URL url = getClass().getResource("datei.txt");           // import java.net.URL  
OutputStream fileOut = new FileOutputStream(url.getFile());  
OutputStreamWriter out = new OutputStreamWriter(fileOut, "UTF-8");
```

Einlesen von Textdaten (InputStreamReader)

```
public int read() throws IOException    // -1 kennzeichnet das Dateiende
```

Ausgabe von Textdaten (OutputStreamWriter)

```
public void write(int zeichen) throws IOException
```

```
public void write(String text) throws IOException
```

```
public void flush() throws IOException
```

Datei schließen (FileInputStream, FileOutputStream)

```
public void close() throws IOException
```

Zeilenumbruch

```
String zeilenumbruch = "\r\n";
```

oder besser (funktioniert dann nicht nur unter Windows):

```
String zeilenumbruch = System.lineSeparator();
```

Fertige Komponenten

Klasse JButton

Methode	Bedeutung
<code>void setEnabled(boolean b)</code>	Aktiviert oder deaktiviert den Button.

Klasse JTextField

Methode	Bedeutung
<code>String getText()</code>	Text aus dem TextField auslesen
<code>void setEditable(boolean b)</code>	Legt fest, ob der Benutzer in das TextField schreiben darf (<code>true/false</code>)
<code>void setText(String t)</code>	Schreibt einen String in das TextField

Klasse JCheckBox

Methode	Bedeutung
<code>boolean isSelected()</code>	Prüfen, ob die Checkbox selektiert ist oder nicht (<code>true/false</code>)

Klasse JTextArea

Methode	Bedeutung
<code>void append(String t)</code>	String an den vorhandenen Text anhängen
<code>String getText()</code>	Gibt den Inhalt der TextArea als String zurück
<code>void setText(String t)</code>	String in die TextArea schreiben (alter Inhalt wird überschrieben)

Klasse JList<String>

Methode	Bedeutung
<code>String getSelectedValue()</code>	Liest den markierten String aus. Liefert <code>null</code> , falls keine Zeile markiert ist.
<code>int getSelectetIndex()</code>	Liefert den Index des aktuell ausgewählten Eintrags als Integer-Wert zurück.

Klasse DefaultListModel<String>

Methode	Bedeutung
<code>void add(int pos, String t)</code>	Fügt eine neue Zeile an der angegeben Position in die Liste ein.
<code>void addElement(String t)</code>	Eine neue Zeile an die Liste anhängen.
<code>void clear()</code>	Löscht den gesamten Inhalt der Liste.
<code>String remove(int pos)</code>	Löscht das Listenelement an der angegebenen Position (und gibt es als Rückgabewert zurück).
<code>int size()</code>	Liefert die Anzahl der Elemente in der Liste zurück.

Klasse DefaultComboBoxModel<String> (JComboBox<String>)

Methode	Bedeutung
<code>void addElement(String t)</code>	Fügt das Element am Ende der Liste ein.
<code>String getSelectedItem()</code>	Liest das markierte Element aus.
<code>void insertElementAt(String t, int pos)</code>	Fügt das Element an der angegebenen Position ein.
<code>void removeAllElements()</code>	Löscht den gesamten Inhalt der Liste.
<code>void removeElementAt(int pos)</code>	Löscht das Element an der angegebenen Position.
<code>void setSelectedItem(String t)</code>	Ändert den Wert des aktuell gewählten Elements

Schreiben in DefaultListModel-Objekt aus einem Thread heraus

```
void add2list(String text) {
    EventQueue.invokeLater(new Runnable() {
        @Override
        public void run() {
            listModel.addElement(text);
            list.ensureIndexIsVisible(listModel.size() - 1);
        }
    });
}
```

Wobei listModel das DefaultListModel-Objekt und list das JList-Objekt ist.

Threads

a) Thread-Klasse

```
public class MeinThread extends Thread {

    public void run() {
        // Code, der im Thread ausgeführt wird
        ...
    }
}
```

b) Thread erzeugen

```
MeinThread thread = new MeinThread();
thread.start();
```

Monitor

```
synchronized (Objekt-Variable) {
    // Anweisungen, die nicht
    // unterbrochen werden dürfen
}
```

Sockets

```
import java.net.*;
import java.io.*;
```

Client-Socket (Klasse Socket)

```
public Socket(String host, int port) throws UnknownHostException, IOException
```

Server-Socket (Klasse ServerSocket)

```
public ServerSocket(int port) // legt einen Haupt-Socket mit der angegebenen Port-Nummer an
public Socket accept() // wartet auf eingehende Verbindungen
```

Streams (Klasse Socket)

```
public InputStream getInputStream() throws IOException
public OutputStream getOutputStream() throws IOException
```

Zeichenbasierte Streams

```
public InputStreamReader(InputStream is, String charsetName)
    throws UnsupportedEncodingException
public OutputStreamWriter(OutputStream os, String charsetName)
    throws UnsupportedEncodingException
```

Beispiel zum Öffnen eines zeichenbasierten Input- und Outputstream:

```
Socket s = new Socket ("localhost", 235);
InputStreamReader in = new InputStreamReader(s.getInputStream(), "UTF-8");
OutputStreamWriter out = new OutputStreamWriter(s.getOutputStream(), "UTF-8");
```

Einlesen von Daten (Klasse **InputStreamReader**)

```
public int read() throws IOException // -1 => die Verbindung wurde getrennt
```

Ausgabe von Daten (Klasse **OutputStreamWriter**)

```
public void write(int zeichen) throws IOException
public void write(String text) throws IOException
public void flush() throws IOException
```

Beispiel für die Verwendung:

```
out.write("Hallo");
out.flush();
```

Verbindung beenden (Klasse **Socket**)

```
public void close() throws IOException
```

Datenbankzugriffe

SQL-Package

```
import java.sql.*;
```

Verbindung zur Datenbank aufbauen

```
Connection conn = DriverManager.getConnection(
    "jdbc:mysql://localhost/MeineDatenbank"
    + "?serverTimezone=UTC&useSSL=false&allowPublicKeyRetrieval=true",
    "root", "root");
Statement stmt = conn.createStatement();
```

Datenbankabfragen

Datenbankabfragen kann man mit der Methode `executeQuery()` des `Statement`-Objekts stellen:

```
public ResultSet executeQuery(String sql) throws SQLException
```

Als Parameter übergibt man einen String mit einer SQL-Select-Abfrage. Zurückgegeben wird ein Objekt vom Typ `ResultSet`, das die Ergebnis-Datensätze enthält. Beispiel:

```
ResultSet rs = stmt.executeQuery("SELECT * FROM tier WHERE geschlecht='w'");
```

Das `ResultSet`-Objekt besitzt eine Methode `next()`, mit der alle Datensätze der Ergebnismenge schrittweise durchlaufen werden können:

```
boolean next()
```

Nachdem man sich mit `next()` einen Datensatz besorgt hat, kann man mit `getXXX()`-Methoden, auf die Spalten des Datensatzes zugreifen. Es gibt eine `getXXX()`-Methode für alle verschiedenen Datentypen, z.B.:

```
getBoolean(), getInt(), getString(), usw.
```

Beispiel:

```
String name = rs.getString(2);          // der Name steht in der zweiten Spalte
int tierId = rs.getInt("tier_id");       // holt den Inhalt der Spalte tier_id als int
```

Datenbankänderungen

Für Datenbankänderungen mit den SQL-Anweisungen `INSERT INTO`, `UPDATE` oder `DELETE FROM` besitzt das `Statement`-Objekt die Methode `executeUpdate()`:

```
public int executeUpdate(String sql) throws SQLException
```

`executeUpdate()` erhält genau wie `executeQuery()` die SQL-Anweisung als Parameter vom Typ `String`. Im Erfolgsfall wird die Anzahl der geänderten Datensätze zurückgegeben. Beispiel:

```
int anzahl = stmt.executeUpdate("DELETE FROM tier WHERE name='Mausi'");
```