1.6 Eingaben

Bisher haben wir Programme betrachtet, in denen mehr oder weniger komplizierte Berech-nungen durchgeführt wurden und die das Ergebnis ausgaben. Der Anwender konnte während

des Ablaufs dieser Programme die Berechnung in keiner Weise beeinflussen. Eine wesentliche Eigenschaft benutzerfreundlicher Programmme ist es jedoch, durch Eingaben während der Laufzeit ihren Ablauf zu bestimmen. In der Regel wird der Benutzer durch Dialogfenster aufgefordert, notwendige Daten einzugeben. Wir werden in diesem Abschnitt mehrere Möglichkeiten kennenlernen, Eingaben in einer Applikation zu verarbeiten.

1.6.1 Eingaben über die Konsole mit `System.console().readLine()`

Eine der einfachsten Möglichkeiten, in Java Eingaben einzulesen, ist der Befehl

```
String eingabe = System.console().readLine("Eingabe: ");
```

Er bewirkt beim Start des Programms von der Konsole, dass die Meldung des Strings (hier: „Eingabe: “) erscheint und der Programmablauf der Applikation stoppt und auf eine Eingabe des Anwenders wartet, bis die Return-Taste gedrückt wird. Die Tastatureingaben werden als String eingelesen, also dem allgemeinsten Datentyp, der Tastatureingaben umfasst, und können in eine Variable gespeichert und dann weiterverarbeitet werden. Ein einfaches Beispiel ist das folgende Programm:

```
1 public class ErsteEingaben {
2    public static void main(String[] args) {
3       String text = System.console().readLine("Gib einen Text ein: ");
4       System.out.println("Du hast eingegeben: \"" + text + "\"");
5    }
6 }
```

Hier wird die Benutzereingabe in Zeile 4 in der Variablen `text` gespeichert und in der nächsten Anweisung einfach ausgegeben. (Beachten Sie in Zeile 5 dabei die Umklammerung des Eingabetextes durch Anführungszeichen mittels der Escape-Sequenz `\"`!)

1.6.2 Eingaben über die Konsole mit der `Scanner`-Klasse

Leider ist die Anwendung von `System.console().readLine()` nicht immer möglich. Insbesondere in einer professionellen Entwicklungsumgebung (IDE) macht dieses Vorgehen leider manchmal Probleme. Deswegen gibt es noch eine zweite Variante Eingaben einzulesen, und zwar mit der `Scanner`-Klasse. Dazu erzeugen wir einen neuen Scanner (Was das genau heißt wird in Kapitel 4 noch genau erklärt) und greifen auf dessen Methoden zu. Das vorhergende Programm finden Sie hier nochmal mit der Scanner-Klasse:

```
1 public class ErsteEingabenScanner {
2    public static void main(String[] args) {
3       java.util.Scanner sc = new java.util.Scanner(System.in);
4       System.out.print("Gib einen Text ein: ");
5       String text = sc.nextLine();
6       System.out.println("Du hast eingegeben: \"" + text + "\"");
7    }
8 }
```

1.6.3 Eingabefenster mit einem Textfeld

Eingaben können etwas bedienungsfreundlicher als über die Konsole in Java auch über Dialogfenster programmiert werden. Eine einfache Möglichkeit dazu bietet die Methode `showInputDialog` der Klasse `JOptionPane` aus dem Paket `javax.swing`, die mit der Anweisung

```
String eingabe = javax.swing.JOptionPane.showInputDialog("... Eingabeaufforderung ...");}
```

aufgerufen wird (falls `eingabe` vorher als `String` deklariert wurde). Die Wirkung dieser Anweisung ist hierbei, dass sie den weiteren Ablauf des Programms solange anhält, bis der Anwender mit der Maus auf `OK` oder `Abbrechen` geklickt oder die `return`-Taste gedrückt hat. Ein einfaches Beispiel für diese Art der Eingabe ist das folgende Programm:

```
1 import javax.swing.*;
2
3 public class EingabenMitFenster {
4    public static void main(String[] args) {
5       String eingabe = JOptionPane.showInputDialog("Gib einen Text ein:");
6       System.out.println("Du hast eingegeben: \"" + eingabe + "\"");
7    }
8 }
```

(Hierbei kann die Klasse `JOptionPane` auch vor der Klassendeklaration per `import javax.swing.JOptionPane;` importiert werden.) Die Variable, in die die Eingabe gespeichert wird, heißt hier im Unterschied zu dem obigen Programm nun `eingabe`, nicht `text`. Drückt man bei der Eingabe auf `Abbrechen`, so wird die Eingabe gar nicht eingelesen und die Ausgabe unseres Programms lautet:

```
Du hast eingegeben: "null"
```

Probieren Sie es einmal aus!

1.6.4 Eingabedialoge mit mehreren Eingabefeldern

Wie kann man mehrere Dateneingaben in einem Dialogfenster programmieren? Eingaben über Dialogfenster mit mehreren Eingabefeldern zu erstellen, ist im Allgemeinen eine nicht ganz einfache Aufgabe. In Java geht es relativ kurz, man benötigt allerdings drei Anweisungen (Anmerkung (2)), den „Eingabeblock“:

```
// Eingabefelder aufbauen:
JTextField[] feld = {new JTextField(), new JTextField()};
Object[] msg = {"Erster Text:", feld[0], "Zweiter Text:", feld[1]};
// Dialogfenster anzeigen:
int click = JOptionPane.showConfirmDialog(null, msg, "Eingabe", 2);
```

(1.7)

Wir werden diese drei Anweisungen hier nur insoweit analysieren, um sie für unsere Zwecke als flexibles Dialogfenster einsetzen zu können. Um genau zu verstehen, wie sie funktionieren, fehlt uns zur Zeit noch das Hintergrundwissen. Die ersten zwei Anweisungen bauen die Eingabefelder und die dazugehörigen Texte auf. Beides sind Deklarationen mit einer direkten Wertzuweisung. Die erste Variable ist `feld`, ein so genanntes *Array*, also eine durchnummerierte Liste vom Datentyp `JTextField[]`. Dieser Datentyp ist eine Klasse, die im Paket `javax.swing` bereit gestellt wird. Das Array wird in diesem Programm mit zwei Textfeldern gefüllt, jeweils mit dem Befehl `new JTextField()`. Das erste Textfeld in diesem Array hat nun automatisch die Nummer 0, und man kann mit `feld[0]` darauf zugreifen, auf das zweite entsprechend mit `feld[1]`.

In der zweiten Anweisung wird ein Array vom Typ `Object[]` deklariert, wieder mit einer direkten Wertzuweisung. Hier werden abwechselnd Strings und Textfelder aneinander gereiht, ihr Zusammenhang mit dem später angezeigten Dialogfenster ist in Abb. 1.8 dargestellt.
Mit der dritten Anweisungszeile schließlich wird das Dialogfenster am Bildschirm angezeigt:

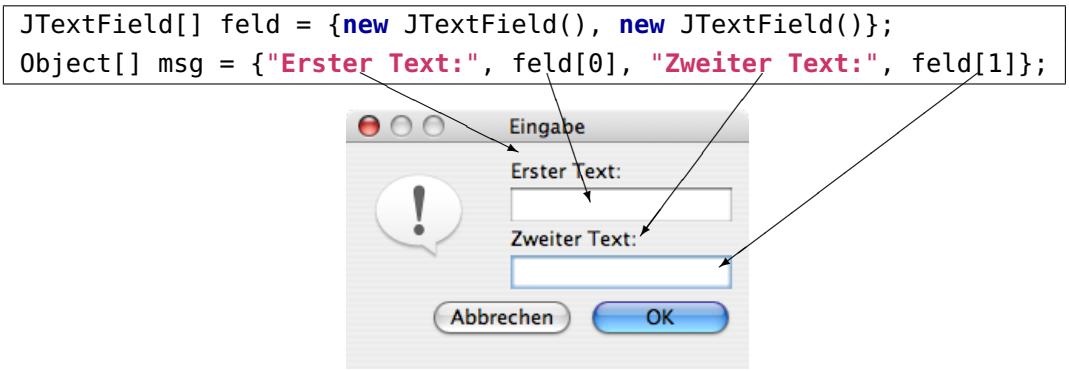

Abbildung 1.8. Das durch den „Eingabeblock“ erzeugte Dialogfenster

```
int click = JOptionPane.showConfirmDialog(null, msg, "Eingabe", 2);
```

Die Wirkung dieser Anweisung ist hier wieder, dass sie den weiteren Ablauf des Programms solange anhält, bis der Anwender mit der Maus auf `OK` oder `Abbrechen` geklickt hat. Alle weiteren Anweisungen der Applikation, also ab Anweisung (3), werden also erst ausgeführt, wenn der Anwender den Dialog beendet hat. Man spricht daher von einem „modalen Dialog“. Danach kann man mit der Anweisung

$$\texttt{feld[}n\texttt{].getText()}$$

auf den vom Anwender in das $(n+1)$-te Textfeld eingetragenen Text zugreifen. Der Text ist in Java natürlich ein `String`.

Ob der Anwender nun `OK` oder `Abbrechen` gedrückt hat, wird in der Variablen `click` gespeichert, bei `OK` hat sie den Wert 0, bei `Abbrechen` den Wert 2:

$$\texttt{OK} \Rightarrow \texttt{click = 0}, \qquad \texttt{Abbrechen} \Rightarrow \texttt{click = 2}. \tag{1.8}$$

Zunächst benötigen wir diese Information noch nicht, in unserem obigen Programm ist es egal, welche Taste gedrückt wird. Aber diese Information wird wichtig für Programme, deren Ablauf durch die beiden Buttons gesteuert wird. Ein einfaches Beispiel für diese Art der Eingabe ist das folgende Programm:

```
1  import javax.swing.JOptionPane;
2  import javax.swing.JTextField;
3
4  public class MehrereEingaben {
5     public static void main(String[] args) {
6        // Eingabefelder aufbauen:
7        JTextField[] feld = {new JTextField(), new JTextField()};
8        Object[] msg = {"Erster Text:", feld[0], "Zweiter Text:", feld[1]};
9        // Dialogfenster anzeigen:
10       int click = JOptionPane.showConfirmDialog(null, msg, "Eingabe", 2);
11       // Eingaben ausgeben:
12       System.out.println(
13          "Du hast eingegeben: \"" + feld[0].getText() + ", " + feld[1].getText() + "\""
14       );
15    }
16 }
```

1.6.5 Zwei eingegebene Strings addieren

Unsere nächste Java-Applikation wird zwei Eingabestrings über die Tastatur einlesen und sie *konkateniert* (= aneinandergehängt) ausgeben.

```
import javax.swing.*;                                          // (1)

/**
 * Addiert 2 einzugebende Strings
 */
public class StringAddition {
  public static void main( String[] args ) {
    // Eingabefelder aufbauen:
    JTextField[] feld = {new JTextField(), new JTextField()};   // (2)
    Object[] msg = {"Erster Text:", feld[0], "Zweiter Text:", feld[1]};
    // Dialogfenster anzeigen:
    int click = JOptionPane.showConfirmDialog(null, msg, "Eingabe", 2);

    // Konkatenation der eingegebenen Texte:
    String ausgabe = feld[0].getText() + feld[1].getText();    // (3)

    // Ausgabe des konkatenierten Texts:
    JOptionPane.showMessageDialog(
      null, ausgabe, "Ergebnis", JOptionPane.PLAIN_MESSAGE     // (4)
    );
  }
}
```

Das Programm gibt nach dem Start das Fenster in Abbildung 1.9 auf dem Bildschirm aus.

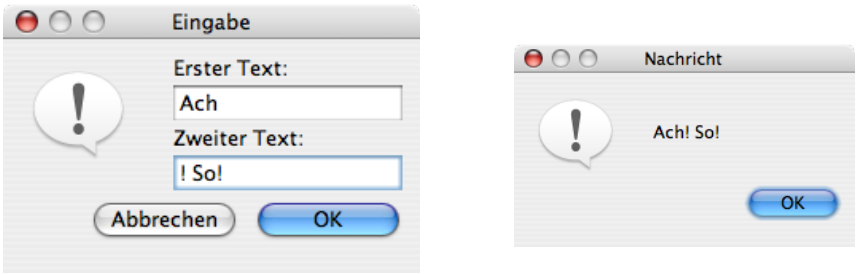

Abbildung 1.9. Die Applikation StringAddition. Links der Eingabedialog, rechts die Ausgabe.

Wir bemerken zunächst, dass diesmal mit `import javax.swing.*` das gesamte Swing-Paket importiert wird, insbesondere also die Klasse `JOptionPane`. Ferner wird die Klasse `StringAddition` deklariert, der Dateiname für die Quelldatei ist also `StringAddition.java`.

Konkatenation von Strings

In Anmerkung (3),

```
String ausgabe = feld[0].getText() + feld[1].getText();
```

wird die Variable `ausgabe` deklariert und ihr direkt ein Wert zugewiesen, nämlich die „Summe“ aus den beiden vom Anwender eingegebenen Texten, in Java also Strings. Natürlich kann man Strings nicht addieren wie Zahlen, eine Stringaddition ist nichts anderes als eine Aneinaderreihung. Man nenn sie auch *Konkatenation*. So ergibt die Konkatenation der Strings `"Ach"` und `"! So!"` z.B.

```
"Ach" + "! So!"   ↦   "Ach! So!".
```

1.6.6 Zahlen addieren

Unsere nächste Java-Applikation wird zwei Integer-Zahlen über die Tastatur einlesen und die Summe ausgeben.

```
import javax.swing.*;

/**
 * Addiert 2 einzugebende Integer-Zahlen.
 */
public class Addition {
  public static void main( String[] args ) {
    int zahl1, zahl2,        // zu addierende Zahlen          // (1)
        summe;               // Summe von zahl1 und zahl2
    String ausgabe;
    // Eingabefelder aufbauen:
    JTextField[] feld = {new JTextField(), new JTextField()};
    Object[] msg = {"Erste Zahl:", feld[0], "Zweite Zahl:", feld[1]};
    // Dialogfenster anzeigen:
    int click = JOptionPane.showConfirmDialog(null, msg, "Eingabe", 2);

    // Konvertierung der Eingabe von String nach int:
    zahl1 = Integer.parseInt( feld[0].getText() );            // (2)
    zahl2 = Integer.parseInt( feld[1].getText() );

    // Addition der beiden Zahlen:
    summe = zahl1 + zahl2;                                    // (3)

    ausgabe = "Die Summe ist " + summe;                       // (4)
    JOptionPane.showMessageDialog(
      null, ausgabe, "Ergebnis", JOptionPane.PLAIN_MESSAGE
    );
  }
}
```

1.6.7 Konvertierung von Strings in Zahlen

In großen Teilen ist dieses Programm identisch mit unserem Programm zur Stringaddition (Abb. 1.9). bis auf die Deklaration der `int`-Variablen. Das Einlesen der Werte geschieht hier genau wie im zweiten Programm, nur dass der Informationstext in der Dialogbox jetzt etwas anders lautet: Der Anwender wird aufgefordert, ganze Zahlen einzugeben.

Nun passiert bei der Eingabe aber Folgendes: Der Anwender gibt ganze Zahlen ein, die Textfelder `feld[0]` und `feld[1]` aber, denen der Wert übergeben wird, liefern mit `feld[`*n*`].getText()` einen Wert vom Typ String. Wie passt das zusammen?

Kurz gesagt liegt das daran, dass über Textfelder genau genommen nur Strings eingelesen werden können. Schließlich ist *jede* Eingabe von der Tastatur ein String, d.h. auch wenn jemand `5` oder `3.1415` eintippt, so empfängt das Programm immer einen String. Der feine aber eminent wichtige Unterschied zwischen `3.1415` als String und 3,1415 als Zahl ist vergleichbar mit dem zwischen einer Ziffernfolge und einer Zahl. Als Zahlen ergibt die Summe 5 + 3,1415 natürlich 8,1415 — als Strings jedoch gilt

```
"5" + "3.1415" = "53.1415"!
```

Zahlen werden durch + arithmetisch addiert (was sonst?), aber Strings werden konkateniert.

Unser Programm soll nun jedoch Integer-Zahlen addieren, empfängt aber über die Tastatur nur Strings. Was ist zu tun? Die Eingabestrings müssen *konvertiert* werden, d.h. ihr Stringwert muss in den Integerwert umgewandelt werden. Java stellt für diese Aufgabe eine Methode bereit, `Integer.parseInt`, eine Methode der Klasse `Integer` aus dem Paket `java.lang`. In der Zeile der Anmerkung (2),

```
zahl1 = Integer.parseInt( feld[0].getText() );
```

gibt die Methode `parseInt` den nach **`int`** konvertierten String aus dem Textfeld als **`int`**-Wert an das Programm zurück, der dann sofort der **`int`**-Variablen `zahl1` zugewiesen wird. Entsprechend wird in der darauf folgenden Zeile der Wert des zweiten Textfeldes konvertiert und an die Variable `zahl2` übergeben.

Die Zeile der Anmerkung (3)

```
summe = zahl1 + zahl2;
```

besteht aus zwei Operatoren, = und +. Der +-Operator addiert die Werte der beiden Zahlen, der Zuweisungsoperator = weist diesen Wert der Vaiablen `summe` zu. Oder nicht so technisch formuliert: Erst wird die Summe `zahl1 + zahl2` errechnet und der Wert dann an `summe` übergeben.

In Anmerkung (4) schließlich wird das Ergebnis mit einem Text verknüpft, um ihn dann später am Bildschirm auszugeben.

```
ausgabe = "Die Summe ist " + summe;
```

Was geschieht hier? Die Summe eines Strings mit einer Zahl? Wenn Sie die Diskussion der obigen Abschnitte über Strings, Ziffernfolgen und Zahlen rekapitulieren, müssten Sie jetzt stutzig werden. String + Zahl? Ist das Ergebnis eine Zahl oder ein String?

Die Antwort ist eindeutig: String + Zahl ergibt — String! Java konvertiert automatisch . . . ! Der +-Operator addiert hier also nicht, sondern konkateniert. Sobald nämlich der erste Operand von + ein String ist, konvertiert Java alle weiteren Operanden *automatisch* in einen String und konkateniert.

Gibt beispielsweise der Anwender die Zahlen 13 und 7 ein, so ergibt sich die Wertetabelle 1.5 für die einzelnen Variablen unseres Programms.

1.6.8 Konvertierung von String in weitere Datentypen

Entsprechend der Anweisung `Integer.parseInt()` zur Konvertierung eines Strings in einen **`int`**-Wert gibt es so genannte parse-Methoden für die anderen Datentypen für Zahlen, die alle

Zeitpunkt	`feld[0]`	`feld[1]`	`zahl1`	`zahl2`	`summe`	`ausgabe`
vor Eingabe	—	—	—	—	—	—
nach Eingabe von `13` und `7`	`"13"`	`"7"`	—	—	—	—
nach Typkonvertierung (2)	`"13"`	`"7"`	13	7	—	—
nach Anmerkung (3)	`"13"`	`"7"`	13	7	20	—
nach Anmerkung (4)	`"13"`	`"7"`	13	7	20	`"Die Summe ist 20"`

Tabelle 1.5. Wertetabelle für die Variablen im zeitlichen Ablauf des Programmms

nach dem gleichen Prinzip aufgebaut sind:

Datentyp`.parse`*Datentyp*`(`... *Text* ...`)`

Beispielsweise wird durch `Double.parseDouble("3.1415")` der String `"3.1415"` in die **`double`**-Zahl `3.1415` konvertiert. Hierbei sind `Integer` und `Double` sogenannte *Hüllklassen (wrapper classes)*. Zu jedem der primitiven Datentypen existiert solch eine Klasse, wie in folgender Tabelle aufgelistet.

Datentyp	**`boolean`**	**`char`**	**`byte`**	**`short`**	**`int`**	**`long`**	**`float`**	**`double`**
Hüllklasse	`Boolean`	`Character`	`Byte`	`Short`	`Integer`	`Long`	`Float`	`Double`

Datentyp	**Hüllklasse**	**Parse-Methode**
`boolean`	`Boolean`	`Boolean.parseBoolean("...");`
`byte`	`Byte`	`Byte.parseByte("...");`
`short`	`Short`	`Short.parseShort("...");`
`int`	`Integer`	`Integer.parseInt("...");`
`long`	`Long`	`Long.parseLong("...");`
`float`	`Float`	`Float.parseFloat("...");`
`double`	`Double`	`Double.parseDouble("...");`

Tabelle 1.6. Die primitiven Datentypen und ihre Hüllklassen *(Wrapper Classes)*.

1.6.9 Einlesen von Kommazahlen

Ein häufiges Problem bei der Eingabe von Kommazahlen ist die kontinentaleuropäische Konvention des Dezimalkommas, nach der die Zahl π beispielsweise als 3,1415 geschrieben. Gibt ein Anwender jedoch eine Zahl mit einem Dezimalkomma ein, so entsteht bei Ausführung der `parseDouble`-Methode ein Laufzeitfehler (eine *Exception*) und das Programm stürzt ab.

Eine einfache Möglichkeit, solche Eingaben dennoch korrekt verarbeiten zu lassen, bietet die Methode `replace` der Klasse `String`. Möchte man in einem String `s` das Zeichen `'a'` durch `'b'` ersetzen, so schreibt man

```
s = s.replace('a', 'b');
```

Beipielsweise ergibt `s = "Affenhaar".replace('a','e');` den String `"Affenheer"`. Mit Hilfe des replace-Befehls kann man eine Eingabe mit einem möglichen Komma also zunächst durch einen Punkt ersetzen und erst dann zu einem **`double`**-Wert konvertieren:

```
double x;
...
x = Double.parseDouble( feld[0].getText().replace(',', '.') );
```