Programmierschnittstelle des Rahmenprogramms: "EinAusRahmen"

Alle Aufgaben sollen unter Verwendung des Rahmenprogramms EinAusRahmen gelöst werden, um in einfacher Weise ein Fenster für die Ein- und Ausgabe auf dem Bildschirm sowie für die Datei-Eingabe und -Ausgabe zu realisieren. Zusätzlich unterstützt EinAusRahmen die syntaktische Zerteilung von Eingabe-Strings. Das Programm wurde unter Verwendung des jdk1.6 der Firma SunSoft entwickelt und auf den Plattformen "Windows XP" und "SunOS 5.5.1" getestet. Prinzipiell sollte es auf allen jdk1.2 basierten virtuellen Maschinen ablauffähig sein. (Ausnahme: Systeme die mit monochromen Monitoren ausgestattet sind.)

EinAusRahmen befindet sich in der Bibliothek "keb". EinAusRahmen ist in einem Java-Programm mit einer public-Klasse X wie folgt zu nutzen:

```
import keb.*;
public class X extends EinAusRahmen { ... }
```

Dadurch wird EinAusRahmen zur Oberklasse von X. In der Klasse X (oder einer Unterklasse von X) ist das Hauptprogramm main zu realisieren, das zuerst die Methode Anfang aufrufen muss. Angenommen, es sollen 25 Ausgabe- und 3 Statuszeilen eingerichtet werden:

```
public static void main (...)
{ Anfang (new X (), "X", 25, 3); ...
}
```

Die Summe von Ausgabe- und Statuszeilen darf einen bestimmten Wert nicht überschreiten, der je nach verwendeter Plattform (z.B. SunOS, Windows NT) variieren kann. Wenn dieser Wert überschritten wird, dann können die einzelnen Bedienelemente des vom EinAusRahmen geöffneten Fensters nicht dargestellt werden.

Die Methoden Hilfe, Antwort und Ende sind zu überschreiben, etwa wie folgt:

```
void Hilfe ()
{ ... // zeige_einen_Hilfstext
}

void Antwort (String K)
{ ... // Reaktion auf das in einer Zeile eingegebene Kommando K
}

void Ende ()
{ ... // Abschließende Berechnung
}
```

Die Methode main muss in der Regel nur die Methode Anfang aufrufen und sonst keine weitere Arbeit verrichten. Allenfalls könnten noch einige anwendungsabhängige Variablen initialisiert werden. Mit der Parametrisierung von Anfang wird ein neues Objekt der public-Klasse X erzeugt und ein Fenster für die Ein- und Ausgabe geöffnet.

Das Fenster enthält einige Menüs und Schaltflächen, eine Zeile für Eingaben, einen mehrzeiligen Ausgabebereich sowie (ganz unten) einige Zeilen für Statusmeldungen. Der zweite Parameter gibt die Beschriftung des Fensters in der Kopfzeile an, der dritte die Anzahl der Zeilen im Ausgabebereich und der vierte Parameter schließlich die Anzahl der Zeilen für Statusmeldungen unterhalb des Ausgabebereichs.

Die Methode Hilfe wird aufgerufen, wenn im Eingabefenster der Menüpunkt „Hilfe zum Anwendungsprogramm" mit der Maus ausgewählt wird. Sinnvollerweise gibt die Methode Hilfe dann einen Text aus, der über die Benutzung des Anwendungsprogramms informiert

Die Methode Antwort wird aufgerufen, wenn eine Eingabe in der Eingabezeile durch Drücken der Return-Taste oder durch Anklicken der Schaltfläche „starte Berechnung“ abgeschlossen wird. Die Eingabezeile wird als String-Parameter an die Methode Antwort übergeben. Sodann muss das in Antwort realisierte Programm auf die Eingabe reagieren, indem (je nach Eingabe) die gewünschte Funktion ausgeführt wird. In der Methode Antwort (und den von Antwort aufgerufenen Unterprogrammen) wird also die Hauptarbeit für die Anwendung verrichtet.

Die Methode Ende wird aufgerufen, wenn im Eingabefenster der Menüpunkt „Programm-Ende“ mit der Maus ausgewählt wird. Es können abschließende Berechnungen durchgeführt werden, was aber oft nicht nötig ist. Die Ergebnisse der abschließenden Berechnung könnten ohnehin nicht auf dem Bildschirm angezeigt werden, weil das Programm unmittelbar danach verlassen wird und das Programm-Fenster vom Bildschirm verschwindet. Wenn Datei-Ausgabe eingestellt ist, werden aber die Ergebnisse noch in die Datei geschrieben.

Beipiel eines Programms, das EinAusRahmen benutzt, um bei Eingabe von z.B. „Sum <x> <y>“ die Summe und bei Eingabe von „Diff <x> <y>“ die Differenz der Zahlen <x> und <y> zu berechnen und auszugeben:

```
import keb.*;
public class Berechnung extends EinAusRahmen
{
  public static void main (String [] Aufrufparameter)
  { Anfang (new Berechnung (), "Summe und Differenz", 30, 1);
  }

  public void Hilfe ()
  { spAus    ("Sum  <x> <y>   liefert die Summe,    ");
    zeigeAus ("Diff <x> <y>   liefert die Differenz.");
  }

  public void Antwort (String K)
  { int a = liesInt (1, 2), b = liesInt (1, 3);
    if      (Wort (1, 1).equals ("Sum" ))  zeigeAus ("Summe " + (a + b));
    else if (Wort (1, 1).equals ("Diff"))  zeigeAus ("Differenz " + (a - b));
    else                                   zeigeAus ("Unzulässige Eingabe");
  }
}
```

In diesem einfachen Programm liefert Wort (1, 1) das zuerst eingegebene Wort als String, liesInt (1, 2) die als zweites und liesInt (1, 3) die als drittes Wort eingegebene ganze Zahl. Die Methode zeigeAus schreibt einen String in den Ausgabebereich.

Eine Übersicht über alle Methoden von EinAusRahmen, die in Hilfe, Antwort und Ende aufgerufen werden können gibt die folgende Aufstellung:

- Methoden zum Lesen der Eingabe:

```
public int      SatzAnz      (String K)
public int      WortAnz      (String K, int s)
public String   Wort         (String K, int s, int w)
public int      liesInt      (String K, int s, int w)
public long     liesLong     (String K, int s, int w)
public float    liesFloat    (String K, int s, int w)
public int      SatzAnz      ()
public int      WortAnz      (int s)
public String   Wort         (int s, int w)
public int      liesInt      (int s, int w)
public long     liesLong     (int s, int w)
public float    liesFloat    (int s, int w)
public boolean  ungueltig    ()
```

- Methoden zur Steuerung der Verarbeitung:

```
public boolean  abgebrochen  ()
public void     brichab      ()
```

- Methoden zum Schreiben der Ausgabe:

```
public void     Echo         (boolean ein)
public String   Spalte       (String Z, int a, int b, String S)
public void     zeigeEin     (String Z)
public void     spAus        (String Z)
public void     spAusZ       (String Z)
public void     zeigeAus     (String Z)
public void     zeigeZahl    (int z,    int StatusZeile)
public void     zeigeZahl    (float z,  int StatusZeile)
public void     zeigeZahl    (String Z, int StatusZeile)
public void     zeigeMeld    (String M, int StatusZeile)
```

Für einfache Programme ohne Besonderheiten werden oft nur wenige dieser Methoden benötigt – und zwar folgende:

```
public int      SatzAnz      ()
public int      WortAnz      (int s)
public String   Wort         (int s, int w)
public int      liesInt      (int s, int w)
public long     liesLong     (int s, int w)
public float    liesFloat    (int s, int w)
public void     zeigeAus     (String Z)
```

Die Eingabe-Methoden gehen davon aus, dass jede Eingabe, die zum Aufruf von Antwort führt, in genau einer Zeile eingegeben wird. Für eine Datei-Eingabe bedeutet dies, dass Antwort für jede Zeile genau einmal aufgerufen wird. Eine Eingabezeile wird auch als Kommando bezeichnet. Jedes Kommando wird als String aufgefasst, der syntaktisch wie folgt in Sätze und Wörter aufgeteilt sein kann:

```
<Kommando> ::=  <Satz>   |  <Satz> , <Kommando>  |  <Satz> ; <Kommando>
<Satz>     ::=  <frei>   |  <xWortx>  |  <xWortx> <leer> <Satz>
<xWortx>   ::=  <frei> <Wort> <frei>
<Wort>     ::=  <wZeichen>  |  <wZeichen> <Wort>
<wZeichen> ::=  <Zeichen_außer_Leerzeichen_Komma_Semikolon>
<frei>     ::=    |  <leer> <frei>
<leer>     ::=  <Leerzeichen>
```

Kommados werden also durch Komma oder Semikolon in Sätze unterteilt, die mehrere durch Leerzeichen getrennte Wörter enthalten können. Beispielsweise besteht das Kommando

```
Gib Gas, Gustav;   Geschwindigkeit  70 km/h, Limit 120
```

aus den 4 Sätzen „Gib Gas", „Gustav", „Geschwindigkeit 70 km/h" und „Limit 120". Der erste Satz besteht aus 2 Wörtern, der zweite aus einem Wort. Der dritte Satz besteht aus den drei Wörtern „Geschwindigkeit", „70" und „km/h", der vierte Satz aus den beiden Wörtern „Limit" und „120".

Die häufig benötigten Methoden haben folgende Bedeutung:

- **`public int SatzAnz ()`**
  gibt an, wieviele Sätze das eingegebene Kommando enthält.
- **`public int WortAnz (int s)`**
  gibt an, wieviele Wörter Satz s des eingegebenen Kommandos enthält. In dem obigen Beispiel liefert WortAnz (4) den Wert 2. Der erste Satz wird durch s = 1, nicht durch 0 (wie bei der Indizierung von Arrays), benannt.
- **`public String Wort (int s, int w)`**
  liefert Wort w in Satz s des eingegebenen Kommandos. In dem obigen Beispiel liefert Wort (1, 2) den String "Gas". Der Aufruf von Wort (4, 1) liefert "Limit", der Aufruf Wort (4, 2) liefert "120" als String. Ebenso wie die Sätze werden auch die Wörter von 1 an gezählt. Ist in der Eingabe ein Wort nicht vorhanden, so wird ein leerer String geliefert. Wort (2, 3) liefert "", weil der zweite Satz nur ein Wort besitzt.
- **`public int liesInt (int s, int w)`**
  wandelt Wort w in Satz s des eingegebenen Kommandos in eine Zahl des Typs integer und liefert diese Zahl. In dem obigen Beispiel liefert liesInt (3, 2) die Zahl 70. Wenn das durch die Parameter spezifizierte Wort nicht vorhanden ist oder keine Zahl enthält, wird der Wert –1 geliefert und gleichzeitig eine Boolesche Variable gesetzt, die über die Methode ungueltig abgefragt werden kann.
- **`public int liesLong (int s, int w)`**
  arbeitet wie liesInt, liefert jedoch eine Zahl vom Datentyp long.
- **`public int liesFloat (int s, int w)`**
  arbeitet wie liesInt, liefert jedoch eine Zahl vom Datentyp float.
- **`public void zeigeAus (String Z)`**
  schreibt den String Z in genau einer Zeile auf das Ausgabemedium. Dies kann der Ausgabebereich des Fensters auf dem Bildschirm und/oder eine Ausgabedatei sein. Das Ausgabemedium kann zur Laufzeit des Programms über die Auswahl entsprechender Menüpunkte jederzeit beliebig festgelegt werden. Wenn der Ausgabebereich voll ist, wird nach oben gescrollt. Wenn die im Folgenden beschriebenen Methoden spAus und spAusZ benutzt werden, gilt: Alle gespeicherten Strings werden, gefolgt vom String Z, im Ausgabebereich des Fensters unter dem bisherigen Inhalt des Ausgabefeldes angezeigt und/oder in die Ausgabedatei geschrieben (bei vorheriger Benutzung von spAusZ mehrere Zeilen auf einmal).

Mit den wenigen genannten Methoden lassen sich bereits viele sinnvolle Programme mit einfachem Ein- und Ausgabeverhalten erstellen. Für besondere Zwecke können zusätzlich folgende Methoden genutzt werden:

- **`public int SatzAnz (String K)`**
  gibt an, wieviele Sätze der String K enthält. Diese Methode arbeitet also wie SatzAnz (), bezieht sich jedoch nicht auf das eingegebene Kommando, sondern einen beliebigen String K.
- **`public int WortAnz (String K, int s)`**
  gibt an, wieviele Wörter Satz s des Strings K enthält.
- **`public String Wort (String K, int s, int w)`**
  liefert Wort w in Satz s des Strings K.
- **`public int liesInt (String K, int s, int w)`**
  wandelt Wort w in Satz s des Strings K in eine Zahl des Typs integer und liefert diese Zahl.
- **`public int liesLong (String K, int s, int w)`**
  wandelt Wort w in Satz s des Strings K in eine Zahl des Typs long und liefert diese Zahl.
- **`public int liesFloat (String K, int s, int w)`**
  wandelt Wort w in Satz s des Strings K in eine Zahl des Typs float und liefert diese Zahl.
- **`public boolean ungueltig ()`**
  gibt an, ob bei den letzten Aufrufen von liesInt (...), liesLong (...) oder liesFloat (...) ein Fehler aufgetreten ist. Die Boolesche Variable, die dies angibt, wird in jedem Fall auf false zurückgesetzt. Nach ein- oder mehrmaligem Aufruf von beispielsweise liesInt (...) sollte ein Programm mit `if (ungueltig ()) ...` zu einer Fehlerbehandlung verzweigen, um dort etwa eine Fehlermeldung auszugeben.
- **`public boolean abgebrochen ()`**
  gibt an, ob während der Ausführung der Methode Antwort in die Schaltfläche zum Abbruch der Berechnung geklickt wurde. Für schnell arbeitende Antwort-Programme von unter einer Sekunde Bearbeitungsdauer ist diese Abfrage nicht zu empfehlen. Bei längerer Bearbeitungsdauer nach einer Eingabe muss jedoch damit gerechnet werden, dass der Benutzer evtl. keine Lust hat lange zu warten. Er kann auf die Schaltfläche „Berechnung abbrechen" klicken, wodurch er eine Boolesche Variable auf true setzt. Diese Variable kann mit der Methode abgebrochen abgefragt werden. Es empfiehlt sich in oft ausgeführten Hauptschleifen oder Rekursionen `if (abgebrochen ()) ...` zu fragen und, falls dies true liefert, die Methode Antwort so schnell wie möglich zu beenden.

Anmerkung zur Schaltfläche "Berechnung abbrechen":

Die von einer Anwendung produzierten Ausgaben können zeitverzögert im Ausgabefeld erscheinen. Wenn nun eine Anwendung mittels einer Eingabedatei mit Kommandos versorgt wird, so kann der Eindruck entstehen, dass die Schaltfläche "Berechnung abbrechen" die Abarbeitung weiterer Kommandos nicht unterbindet. Tatsächlich

aber werden in diesem Fall nur noch Ausgaben von Kommandos in das Ausgabefeld geschrieben, die bereits vom Anwendungsprogramm produziert wurden, bevor die Schaltfläche betätigt wurde.

- **`public void brichab ()`**
  ermöglicht einem Programm, das weitere Lesen von Eingabezeilen einer Datei abzubrechen. Der Aufruf von brichab wird empfohlen, wenn eine fehlerhafte Eingabe festgestellt worden ist, die das Lesen nachfolgender zugehöriger Eingaben sinnlos macht.
- **`public void Echo (boolean ein)`**
  ermöglicht einem Programm festzulegen, ob alle gelesenen Eingabezeilen in die Ausgabe geschrieben werden sollen. Ein solches sog. Echo wird mit dem Parameterwert true eingeschaltet und mit dem Parameterwert false ausgeschaltet. Bei Programmstart ist das Echo eingeschaltet. Jeder Eingabezeile, die in die Ausgabe geschrieben wird, wird eine Kennung vorangestellt, die angibt, woher die Eingabe stammt. Es bedeuten

`Tast>>>`	Die Eingabe wurde aus der Eingabezeile des Bildschirm gelesen. Die Eingabe wurde durch Drücken der Return-Taste auf der Tastatur abgeschlossen.
`Maus>>>`	Die Eingabe wurde aus der Eingabezeile des Bildschirm gelesen. Die Eingabe wurde durch Mausklick auf die Schaltfläche „starte Berechnung“ abgeschlossen.
`Dat.>>>`	Die Eingabe wurde aus einer Eingabedatei gelesen.

- **`public String Spalte (String Z, int a, int b, String S)`**
  Füge den String S in die Spalten a bis abs (b) des Strings Z ein. Wenn b positiv ist, wird linksbündig, andernfalls rechtsbündig eingefügt. Die Nummerierung der Spalten beginnt mit 1. Der bisherige Inhalt der Spalten a bis b von Z wird überschrieben. Ist Z kürzer als a, so wird Z zunächst mit Leerzeichen verlängert. Der resultierende String wird zurückgeliefert. Diese Methode dient dem spaltenweisen Formatieren von Text, der danach ausgegeben werden soll.
- **`public void zeigeEin (String Z)`**
  Zeige die erste Zeile von Z in der Eingabezeile des Fensters an. Damit ist es möglich, bestimmte Texte als default für die Eingabe vorzugeben. Wenn etwa ein Programm verlangt, den Namen des leistungsstärksten Fußballclubs einzugeben, so kann man mit zeigeEin ("Schalke") erreichen, dass „Schalke“ in der Eingabezeile erscheint, so dass alle, die „Schalke“ eingeben möchten, nur noch auf „starte Berechnung“ klicken brauchen – und für Schalke entscheiden sich ja ohnehin die meisten.
- **`public void spAus (String Z)`**
  Speichere den String Z für die spätere Ausgabe im Ausgabebereich des Fensters. Bei einem späteren Aufruf von zeigeAus (X) werden alle gespeicherten String entsprechend ihrer Speicher-Reihenfolge, gefolgt vom String X, ausgegeben. Die Speicherung eines Strings Z ist vorteilhaft, wenn kurz nacheinander viele Ausgaben angezeigt werden sollen. Entsprechend viele Zugriffe auf den Ausgabebereich können die Programmausführung verlangsamen. Durch die Speicherung wird

dagegen erreicht, dass mehrere Zeilen mit nur einem Zugriff auf dem Bildschirm angezeigt werden. Wenn SpAus mit dem Parameter null aufgerufen wird, dann werden alle zuvor gespeicherten Strings gelöscht.

- **`public void spAusZ (String Z)`**
  arbeitet wie spAus, jedoch wird der String Z mit einem Zeilenwechsel abgeschlossen.
- **`public void zeigeZahl (int z, int StatusZeile)`**
  Zeige die Zahl z im Zahlenfeld der angegebenen Statuszeile an. Die Statuszeilen befinden sich unter dem Ausgabebereich am unteren Rand des Fensters. Jede Statuszeile ist in eine linke und eine rechte Spalte unterteilt. Jede Spalte besteht aus einem Zahlenfeld, gefolgt von einem Textfeld. Die linke Spalte einer Statuszeile x wird mit –x, die rechte mit x bezeichnet. Die erste (oberste) Statuszeile hat den Betrag 1. Um also beispielsweise in die linke Spalte der zweiten Statuszeile zu schreiben, muss der Parameter StatusZeile den Wert –2 aufweisen. Dort wird dann die Zahl z angezeigt.
- **`public void zeigeZahl (float z, int StatusZeile)`**
  arbeitet wie zeigeZahl (int z, int StatusZeile), jedoch wird eine Zahl des Datentyps float angezeigt.
- **`public void zeigeZahl (String Z, int StatusZeile)`**
  arbeitet wie zeigeZahl (int z, int StatusZeile), jedoch wird im Zahlenfeld der String Z angezeigt. Nur sehr kurze Strings passen in ein Zahlenfeld, längere werden abgeschnitten.
- **`public void zeigeMeld (String M, int StatusZeile)`**
  Zeige die Meldung M im Textfeld der angegebenen Spalte einer Statuszeile an. Die Bedeutung des Parameters StatusZeile geht aus der Beschreibung der Methode zeigeZahl (int z, int StatusZeile) hervor.

Die Anzahl der eingerichteten Statuszeilen wird als Parameter der Methode Anfang angegeben, die zu Programmbeginn in der Methode main aufgerufen werden muss (siehe oben). Die Statuszeilen werden sinnvollerweise dazu genutzt, den Rechenfortschritt in der Methode Antwort anzuzeigen. Soll beispielsweise eine Strategie zur Vorratshaltung in einem Lager simuliert werden, um zu prüfen, ob eine Mangelsituation auftritt, so könnten die Statuszeilen genutzt werden, um den momentanen Bestand der Ware A und der Ware B, die Anzahl der simulierten Zugriffe auf das Lager sowie die Anzahl aufgetretener Mangelsituationen angezeigt werden. Die Zugriffsanzahl wird wohl als schnell laufender Zähler den Programmfortschritt zeigen. Die Statuszeilen könnten in diesem Beispiel wie folgt aussehen:

```
  24 Pakete der Ware A          23055 Zugriffe auf das Lager
3006 Pakete der Ware B              3 Mangelsituationen
```