Grundlagen der Programmierung II (Java)

Lösungshinweise zum 4. Praktikum

18. Mai 2010

Grafische Realisierung

Die Anwendung enthält ein Raster mit drei Zeilen:

Zeile 1 (gridy=0) Hier befinden sich zwei Elemente, das Label „Suche:“ und das Textfeld `textSuche`.

Da das Label nicht wachsen soll und sich auch nicht über mehrere Zellen erstreckt, reichen hier die Defaultwerte nach dem Instanzieren eines `GridBagConstraints`-Objekts.

Für das Textfeld werden folgende Constraints gewählt: gridx=RELATIVE (nächstes Element in der Zeile), gridwidth=REMAINDER (erstreckt sich über alle restlichen Spalten), fill=HORIZONTAL (wächst horizontal), weightx=1 (Wert größer Null).

Zeile 2 (gridy=1) In dieser Zeile befindet sich die in einen `JScrollPane` gepackte Auswahlliste. Diese erstreckt sich über alle Spalten (gridwidth=REMAINDER), wächst horizontal und vertikal (fill=BOTH, weightx=weighty=1).

Zeile 2 (gridy=1) Hier sind drei Elemente, das Label „Name:“, das Textfeld `textNeu` und der Button `btnNeu`. Hierbei wächst das Textfeld horizontal (fill=HORIZONTAL, weightx=1), weiterhin wird gridwidth=RELATIVE gewählt (was bedeutet: erstreckt sich über alle noch verfügbaren Spalten außer der letzten – beim dreispaltigen Grid wie hier entspricht das im Endeffekt nur einer Spalte, aber so bleibt man flexibel gegenüber Änderungen im Layout).

Datenmodell

Das Datenmodell wird hier als innere Klasse `MyListModel` realisiert. Dies geschieht eher aus Bequemlichkeitsgründen, letztlich wird nicht auf Attribute der äußeren Klasse zugegriffen. Die Datenmodell-Klasse erbt von `AbstractListModel`, die darin enthaltenen abstrakten Methoden `getSize` und `getElementAt` müssen also implementiert werden.

Für die Verwaltung der Namen werden zwei Collections als Attribute definiert: ein `SortedSet` für alle Namen (sortiert), und eine `List` für die anhand des Suchstrings ausgefilterten Namen, die dann in der Auswahlliste angezeigt werden sollen.

Der Konstruktor von `MyListModel` erwartet eine |Collection<String>| als Parameter, die die Namen enthält, die von Anfang an in der Auswahlliste erscheinen sollen. Diese Collection wird dem Konstruktor von `TreeSet<String>` übergeben und so das Attribut `alleNamen` initialisiert. Anschließend wird mit `setFilter("")` der Filter auf den leeren String gesetzt, d. h. alle Namen sollen angezeigt werden.

In der Methode `setFilter(String anfang)` sollen alle Namen aus `alleNamen` in die List `passendeNamen` überführt werden. Hierzu wird eine Instanz von `ArrayList<String>` erzeugt, deren Konstruktor `alleNamen.subset(anfang,anfang+"\377")` übergeben wird. Oktal 377 entspricht hierbei dem Zeichen 255, also dem letzten Zeichen in der ASCII-Tabelle, so dass für jeden String lstinline|s|, der mit den Zeichen aus `anfang` beginnt, die Bedingung $\texttt{anfang} \leq \texttt{s} < \texttt{anfang}$ erfüllt ist. Im trivialen Fall des leeren Filter-Strings wird der komplette `SortedSet` dem Konstruktor der `ArrayList` übergeben. Am Ende wird `fireContentsChanged(this,0,Integer.MAX_VALUE)` aufgerufen, d. h. es wird der `JList`, die das Datenmodell darstellt, signalisiert, dass dieses sich von Zeile 0 bis zur maximal möglichen Zeilennummer geändert hat, also alles neu dargestellt werden muss.

Die Methode `add` fügt einen neuen Namen zu `alleNamen` hinzu, da danach `setFilter` mit leerem String aufgerufen wird, werden alle Namen danach in die Liste übernommen.

Die Methoden `getSize` und `getElementAt` werden einfach auf die Methode `size` bzw. `get` der List `passendeName` abgebildet.

Event-Logik

Um während des Tippens im Suchfeld dynamisch die passenden Namen in der Auswahlliste anzuzeigen, wird beim Textfeld für die Suche ein `KeyListener` angemeldet. Dieser wird als anonyme Klasse implementiert, die von `KeyAdapter` erbt. Hier wird die Methode `keyReleased` überladen, die beim Loslassen der Taste aufgerufen wird. Dabei wird jeweils des Inhalt des Suchfelds ausgelesen und der Methode `setFilter` des Datenmodells übergeben.

Beim Textfeld für einen neuen Namen wird auch eine `KeyListener` angemeldet, der analog als anonyme Kindklasse von `KeyAdapter` implementiert ist. Hier wird `keyReleased` so implementiert, dass der Button `btnNeu` in Abhängigkeit davon, ob das Namenstextfeld leer ist, aktiviert bzw. deaktiviert wird.

Der Button `textNeu` wird mit einer `Action` als Parameter instanziert, die das Namensfeld ausliest und den Inhalt an die Methode `add` des Datenmodells übergibt. Dieselbe `Action` wird auch als `ActionListener` beim Namenstextfeld angemeldet, so dass der Name auch mit der Enter-Taste in die Auswahlliste übernommen werden kann.

Bei der Auswahlliste wird ein `ListSelectionListener` angemeldet, der beim Klicken auf ein Element in der Liste dieses ins Suchfeld übernimmt und den Filter-String des Datenmodells entsprechend setzt.