```
60            default: // Falsche Zahl eingegeben
61              System.out.println("Eingabefehler!");
62          }
63        } // Schleifenende
64      } // Ende des Hauptprogramms
65    } // Ende des Programms
```

Unsere neuen Objekte `adr0` und `adr1` werden hierbei in den Zeilen 19 und 20 vereinbart.[7] Die Referenz `adr` wird anfangs auf `adr0` gesetzt (Zeile 21). Das Menü wird um einen zusätzlichen Eintrag erweitert (Zeile 33), der in den Zeilen 54 bis 58 implementiert wird. Das restliche Programm stimmt mit dem von Seite 135 überein.

### 5.2.6 Felder von Klassen

Wir haben im letzten Abschnitt gelernt, dass wir mit nur wenig Mehraufwand unser Adressprogramm von der Verwaltung *einer* Adresse auf die Verwaltung von mehr als einer Adresse ausweiten konnten: durch die Verwendung von Referenzen konnten wir das allgemeine Problem *mehrerer* Adressen (`adr0` und `adr1`) auf die Verwaltung *einer* Adresse (`adr`) zurückführen. Wie können wir unser Programm nun so modifizieren, dass wir auch eine größere Zahl von Datensätzen verarbeiten?
Der Gedanke liegt nahe, hierfür auf unser Wissen aus Abschnitt 5.1 zurückzugreifen. Wir haben eine Vielzahl von Daten (z. B. zwanzig Adresseinträge), die wir etwa in Form einer Tabelle anordnen könnten. Wir nummerieren die Adresseinträge von 0 bis 19 durch und speichern sie in einem Feld namens `adressen`:

```
Adresse[] adressen= new Adresse[20];
```

Dieser einfache Ansatz führt tatsächlich bereits zu dem gewünschten Ergebnis. Es ist nämlich so, dass in Java Felder nicht nur über einfachen Datentypen (**`int`**, **`double`**,...), sondern auch über Referenzdatentypen aufgebaut werden können. Hierzu zählen sowohl Felder (siehe Abschnitt 5.1.7 und 5.1.7) als auch sonstige Klassen. Wir haben diesen Umstand unbewusst schon genutzt, indem wir für unseren Terminkalender Felder über Strings bildeten. Nun wollen wir uns diese Eigenschaft von Java jedoch auch bewusst zu Nutze machen.
Abbildung 5.16 zeigt den Zustand unseres Feldes `adressen` nach der Erzeugung mit Hilfe des **`new`**-Operators. Die Variable `adressen` umfasst insgesamt zwanzig Einträge, das heißt, sie verweist auf die Komponenten `adressen[0]` bis `adressen[19]`, die vom Typ `Adresse` sind. Da es sich hierbei um eine benutzerdefinierte Klasse handelt, also auch um einen Referenzdatentyp, stellt jeder dieser Einträge wieder eine Referenz dar. Diese zeigt zu Anfang „nirgendwohin“, d. h. sie referenziert kein spezifisches Objekt. Es handelt sich um die so genannte **Null-Referenz**, die in Java mit **`null`** bezeichnet wird.

[7] Den aufmerksamen Lesern wird hier wahrscheinlich etwas aufgefallen sein: Warum werden die Objekte hier als `adr0` und `adr1` bezeichnet, obwohl es sich doch nur um die Namen der *Referenzen* handelt? Tatsächlich „heißen“ die Objekte natürlich nicht `adr0` oder `adr1`; aus Gründen der Übersichtlichkeit wird diese Ungenauigkeit aber üblicherweise in Kauf genommen.