2

Objektorientiertes RPG

Das erste Kapitel wies auf die Ähnlichkeiten im Aufbau eines RPG-Programms und einer Java-Klasse hin. Aber wenn diese Strukturen bei der Erstellung von Java-Anwendungen verwendet werden, gefolgt von den Java-Design-Strategien, dann beginnen die Ähnlichkeiten zwischen RPG und Java zu verschwinden. Java ist eine objektorientierte Sprache, während RPG strukturell ist. Bei Java repräsentiert eine Klasse ein Objekt und die Verarbeitung ist ereignisbezogen. Im Gegensatz dazu öffnet ein einzelnes RPG-Programm oft ein Dutzend Dateien. Tatsächlich warteten viele AS/400-Softwarehäuser ungeduldig auf ILE-RPG nicht wegen der modularen Programmierfähigkeiten, sondern wegen der Unterstützung von mehr als 50 Dateien. Die RPG-Programmiersprache wurde mit dem einzigen Ziel entwickelt, relationale Datenbanken zu verarbeiten, während die Basis der Java-Sprache relationale Datenbanken noch nicht einmal kennt.

Das Design von RPG-Anwendungen und Programmen basiert auf algorithmische Zerlegung, welche in einer Hierarchie von Systemen und Programmen resultiert. RPG ist eine prozedurale Sprache: Jede Hauptroutine eines Programms steuert die Verarbeitung von Objekten, die in DB2/400 gespeichert sind, in dem, was oft die *große Schleife* genannt wird. Im Allgemeinen bilden Java-Klassen keine Schleifen; sie sind eher Ereignis-gesteuert als Daten-bezogen. Bei Java ist das Verhalten eines Objektes eng mit seinen Daten in einer Java-Klasse verbunden. Die Schnittstelle der Objekte einer Datenbankdatei wird durch die Funktionen der Java-Klasse abgerufen, die geschaffen wurde, um diese Objekte zu repräsentieren. Die Java-Klasse definiert den Datenbank-Zugriff durch ihre Liste von Funktionen, während sie die Komplexität der Implementation dieser Klasse versteckt.

RPG++

Es ist schwierig genug eine neue Sprache zu lernen, ohne eine ganz neue Philosophie von Anwendungsdesign zu erlernen. Um diesen beispielhaften Richtungswechsel zu überbrücken, führt Sie dieses Kapitel in die Konzepte des objektorientierten Programmierens ein unter Verwendung eher der RPG-Programmiersprache als Java. Weil RPG nicht als objektorientierte Sprache geschaffen wurde, musste der Zweck von RPG geändert werden, um eine wirkungsvolle Darstellung von objektorientierten Prinzipien durch RPG-Beispiele zu erreichen. Ich erfand einige hypothetische Erweiterungen zu RPG und nannte diese neue Sprache RPG++; wie C++ eine objektorientierte C-Sprache ist, ist RPG++ ein objektorientiertes RPG. Ich zog es in Betracht, diese

erweiterte Sprache Postum zu nennen als Wortspiel zu Java, da Postum ein Kaffeeersatz ist – er ist nicht „echt“ und genau so wenig sind es meine hypothetischen Erweiterungen zu RPG.

Die erste große Veränderung bei der RPG++-Sprache liegt darin, dass jedes Programm ein Objekt bearbeitet. Jeder Programmanruf arbeitet zum Beispiel an einem Kunden – keine Schleifenbildung. „Wie soll das denn nun funktionieren?“, werden Sie sich fragen. Haben Sie noch ein wenig Geduld. Java-Klassen, wie sie im ersten Kapitel behandelt wurden, haben einen *Constructor*, der den Zustand des Feldes der Klasse initialisiert. Eine Customer-Klasse in Java kann eine Konstruktorfunktion haben, die ein integeres Argument enthält, das der Konstruktor als Schlüssel verwendet, um die Werte eines Kundendatensatzes aus einer relationalen Datenbankdatei abzurufen.

RPG benutzt schon eine Initialisierungsunterroutine, die vom Konzept her einem Konstruktor ähnlich ist. *INZSR akzeptiert jedoch keine Parameter. Die *INZSR der RPG++-Sprache akzeptiert Parameter. Das Kundenprogramm, das in Bild 2.1 gezeigt ist, schließt eine Initialisierungsunterroutine mit einem Parameter für die Kundennummer ein. Dieser Kundennummerparameter wird dann als Schlüssel verwendet, um Informationen über diesen Kunden aus der Kundenhauptdatei abzurufen.

```
FCUSTMAST U F
 * code omitted
C     *INZSR    BEGSR
```

Bild 2.1: Das hypothetische RPG++-Kundenprogramm bearbeitet nur einen Kunden zur gleichen Zeit. (Teil 1 von 2).

```
C                   PARM                    CustNo    5 0
C       Cust        Chain   CustRec
C                   ENDSR
 *
C       getLSTNAM   BEGSR
C                   Return                  LSTNAM
C                   ENDSR
*
C       getBALDUE   BEGSR
C                   return                  BALDUE
C                   ENDSR
 *
C       putBALDUE   BEGSR
C                   PARM                    NewBal    5 0
C                   z-add   NewBal          BALDUE
C                   ENDSR
 *
C       putLSTNAM   BEGSR
C                   PARM                    NewNam 20
C                   move    NewNam          LSTNAM
C                   ENDSR
 *
C       update      BEGSR
C                   Update  CustRec
C                   ENDSR
 *
C       write       BEGSR
C                   Write   CustRec
C                   ENDSR
 *
C       next        BEGSR
C                   READ    CustMast
C                   MOVE    CSTNUM          custNo
C                   NEW     'Customer'      aCust
C                   PARM                    custNo
C                   RETURN  aCust
C                   ENDSR
 *
C       *INZSR      BEGSR
C                   PARM                    CustNo       5 0
C       Cust        Chain   CustRec
C                   ENDSR
```

Bild 2.1: Das hypothetische RPG++-Kundenprogramm bearbeitet nur einen Kunden zur gleichen Zeit. (Teil 2 von 2).

Der eigentliche Zweck des Anrufs des Kundenprogramms ist es, den Kunden zu bearbeiten, der mit der Nummer zugeordnet war, die seiner Initialisierungsunterroutine übergeben wurde. Aber wie übergibt man Parameter der Initialisierungsunterroutine? Meine RPG++-Sprache hat einen neuen OpCode namens NEW. Wenn andere RPG-Programme Informationen über einen Kunden abrufen wollen, öffnen sie nicht die Kundenhauptdatei zur Aktualisierung, sondern stattdessen verwenden sie den NEW-OpCode mit dem Namen des Kundenprogramms in Faktor 2 und einem Parameter, der die Kundennummer enthält:

```
C                   NEW   'Customer'   aCust
C                   PARM  148          custNo
```

Der NEW-OpCode ruft nicht die Hauptroutine des Kundenprogramms auf und der Kundennummerparameter wird nicht der *ENTRY PLIST übergeben. Der neue OpCode löst die *INZSR des Kundenprogramms aus und der Kundennummerparameter wird verwendet, um den zugeordneten Kundenhauptdatensatz mit einer Chain-Operation zu erhalten. Wie jede RPG-Initialisierungsunterroutine, so erstellt die *INZSR des RPG++ Kundenprogramms ihre eigene Process Access Group (PAG) für diesen neuen Programmaufruf. Jede Verwendung des NEW-OpCode von RPG++ über das Kundenprogramm löst einen erneuten Programmaufruf aus mit seiner eigenen getrennten PAG.

Beachten Sie den Variablennamen im Ergebnisfeld, der dem NEW-Aufruf des Kundenprogramms folgt. Der Datentyp dieser Variablen, ACUST, ist das Kundenprogramm. Kunde ist nun ein Datentyp bei Operationen, genau wie ein gepackter Dezimaldatentyp die Operationen ADD, SUBTRACT, MULTIPLY, DIVIDE und ASSIGNMENT hat. Das Kundenprogramm ist ein abstrakter

Datentyp mit den Operationen UPDATE, WRITE, NEXT und einer Auswahl von GET und PUT-Operationen für den Abruf und die Änderung der Attribute eines Objektes. Das Kundenprogramm, das in Bild 2.1 gezeigt ist, besitzt eine Anzahl von Unterroutinen, die direkt von anderen RPG-Programmen aus aufgerufen werden können. Bild 2.2 zeigt eine Liste dieser extern aufrufbaren Unterroutinen.

Die Unterroutinen des Kundenprogramms können von einer Variablen aufgerufen werden, die einen Bezug zu dem Kundenprogrammaufruf erhält, wie die vorher geschaffene ACUST-Variable. Diese ACUST-Variable, die in einem anderen Programm als dem Kundenprogramm erklärt ist, wird verwendet zusammen mit dem Punkt-Operator, um einen Funktionsnamen des Kundenprogramms zu qualifizieren:

```
C                   EVAL bal = aCust.getBALDUE ()
C                   EVAL aCust.putBALDUE(bal)
C                   EVAL aCust.update()
```

```
C     getLSTNAM  BEGSR
C     getBALDUE  BEGSR
C     putBALDUE  BEGSR
C     putLSTNAM  BEGSR
C     update     BEGSR
C     write      BEGSR
C     next       BEGSR
```

Bild 2.2: Das Kundenprogramm besitzt eine Anzahl von Unterroutinen, die von anderen RPG-Programmen aufgerufen werden können.

Wenn ein RPG++-Programm mit einem anderen Kunden arbeiten muss, muss es den NEW-OpCode verwenden, um auf ein anderes Kundenprogramm zuzugreifen, weil, wie Sie sich erinnern werden, die *INZSR den Datensatz dieses Kunden aus der Kundenhauptdatei abruft.

```
C                   NEW         'Customer'      aCust
C                   PARM        148             custNo
```

Verarbeitungssätze

Dieses Szenario funktioniert bestens in einem interaktiven Programm, dass den Benutzer iterativ veranlasst, eine Kundennummer einzugeben um ein neues Kundenprogramm aufzurufen. Aber was ist, wenn das andere RPG++-Programm sequentiell alle Kunden verarbeiten will? Eine Möglichkeit, dies zu erreichen, ist das Hinzufügen einer Unterroutine, um einen neuen Kundenprogrammaufruf zu erstellen. Dies macht die Unterroutine NEXT in dem in Bild 2.3 gezeigten Kundenprogramm. Sie liest einen Datensatz aus der Kundenhauptdatei und verwendet dann die Kundennummer in dem Parameter, der dem NEW-OpCode folgt. Wie ich bereits erwähnt habe ruft der NEW OpCode die *INZSR des Kundenprogramms auf, um eine neue Programminstanz zu erstellen. Diese Kundenprogramminstanz wird dann an die anderen Programme zurückgegeben, die die NEXT-Funktion des Kundenprogramms auslösen.

```
C     next        BEGSR
C                 READ    CustMast
C                 MOVE    CSTNUM        custNo
C                 NEW     'Customer'    aCust
C                 PARM                  custNo
C                 RETURN  aCust
C                 ENDSR
```

Bild 2.3: Die NEXT-Unterroutine erstellt ein neues Kundenprogramm aus einem sequentiell abgerufenen Kundenhauptdatensatz.

Das Beispiel der DO-Schleife, das in Bild 2.4 gezeigt ist, veranschaulicht, wie ein anderes RPG-Programm in einer Schleife die Kunden verarbeiten würde. Innerhalb der DO-Schleife, verwendet das Programm die NEXT-Funktion des Kundenprogramms, um die Kunden sequentiell abzurufen. Wenn die ACUST-Variable (die eine Verbindung zu dem Kundenprogrammaufruf darstellt, der den Zustand des ersten Kunden hält) Null ist, bricht die Schleife ab; anderenfalls wird ein neuer Kontostand für diesen Kunden erstellt und die setBALDUE-Funktion des Kundenprogramms wird durch die ACUST-Variable ausgelöst. Die Änderung wird an DB2/400 durchgeführt mittels eines Aufrufes der UPDATE-Funktionen des Kundenprogramms.

```
C                   DO
C                   EVAL      aCust = Customer.next()
C         aCust     IFEQ        0
C                   LEAVE
C                   ENDIF
C* calculate balance due
C                   EVAL      aCust.setBALDUE (balance)
C                   EVAL      aCust.update()
C                   ENDDO
```

Bild 2.4: Andere RPG++-Programme können in einer Schleife Kunden abarbeiten, indem sie iterativ die NEXT-Unterroutine des Kundenprogramms aufrufen.

Geschäftsobjekte

Ob Sie in Java, C++, oder Smalltalk (oder sogar in RPG++) programmieren, sollte der Zugriff und die Manipulation Ihrer Kunden, Rechnungen, Bestellungen, Artikel und so weiter mittels einer Klasse durchgeführt werden. Mit einer gut entworfenen relationalen Datenbank, verbinden sich die Klassen direkt zu Datensatzformaten. Die Felder dieser Klasse korrespondieren mit den Felder eines Datensatzformates. Wenn eine Klasse verwendet wird, um eine Instanz einer Entität (die als Datensatz in DB2/400 gespeichert ist) zu erstellen, dann ist diese Instanz als ein Objekt bekannt.

Dieses Kapitel führte Sie in einen OpCode namens NEW ein; bei Java gibt es auch einen Operator namens NEW, den Sie verwenden können, um ein Objekt aus einer Klasse zu erstellen. Die Klasse ist ein abstrakter Datentyp, der verwendet wird, um Varia-

blen zu definieren. Bei Java findet der Zugriff auf Objekte durch die Klasse statt, die dieses Objekt repräsentiert. Nur die Java-Klasse, die geschaffen wurde, um ein Objekt zu repräsentieren, sollte die DB2/400-Datei öffnen, die den Zustand dieses Objektes speichert. Wenn andere Klassen ein Objekt abrufen, modifizieren, oder neu erstellen müssen, tun sie dies allein durch die Funktionen der Klasse dieses Objektes.

Es ist leicht, Java zu lernen und dann anfangen, strukturierte Java-Anwendungen zu erstellen, aber die Vorteile von Java werden nur von objektorientierten Programmieren kommen. Tatsächlich macht, wie Sie in Abschnitt III sehen werden, der Aufwand des Zufriffs auf DB2/400-Dateien und Felder, es beinahe unmöglich, in derselben Art und Weise zu kodieren, wie Sie dies bei RPG tun. Aber wenn Sie Ihre Java-Anwendungen unter Beachtung der grundlegenden Strategien des objektorientierten Programmierens erstellen, werden Sie durch Code-Wiederverwendung, Anwendungsanpassungsfähigkeit und verbesserte Programmierproduktivität belohnt.