



## Ausgabe Juni 2006

Einstieg in die Objekt-Orientierung für RPG-Programmierer

### Objektorientierung in RPG.NET

Der Umstieg von der prozeduralen in die objektorientierte Programmierung ist eine beachtliche Hürde. Da diese Technologie eine große Umstellung in allen Belangen eines Projekts verlangt, ist bei Einführung von Objekt-Orientierung (OO) Vorsicht geboten. Durch die Menge an neuen Verfahren, Regeln und Notwendigkeiten gibt es auch viele Möglichkeiten, Projekte zum Scheitern zu bringen. Bei erfolgreicher Anwendung von OO-Konzepten werden Sie aber in der Lage sein, wiederverwendbare Programme zu erstellen, die Ihnen die Wartung und Erweiterung von Projekten erleichtern.

#### Nur keine Euphorie

Wenn Sie in einer Umgebung programmieren, die OO-Techniken ermöglicht, heißt das noch lange nicht, dass Sie objektorientiert programmieren. Wenn Sie Objekte nur benutzen, programmieren Sie noch nicht objektorientiert. Wirklich objektorientiert programmieren Sie erst dann, wenn Sie OO-Sprachmittel – wie Klassen, Vererbung, Kapselung etc. – in Ihren Programmen einsetzen. Mit OO lassen sich Projekte toll umsetzen, wenn man richtig an die Sache herangeht. Allerdings kann man das nicht von heute auf morgen, auch erfahrene Programmierer sollten für den Einstieg genug Zeit einplanen.

#### Kleine Schritte

Zuerst die Theorie und dann die Praxis! Nur kontinuierliches Lernen mit aufbauenden Übungen bringt Sie im Thema weiter. Kaum ein RPG-Programmierer wird Zeit haben, an monatelangen Schulungen teilzunehmen oder vor einem Projekt eine akademische Ausbildung in OO zu absolvieren. Deshalb sollten Sie für den Start von OO-Projekten externe Kompetenz zu Rate ziehen und das richtige Werkzeug wählen. Eine gut geeignete Umgebung wie RPG.NET begleitet Sie vom Einstieg bis zum professionellen Umgang mit der OO-Technologie. In RPG.NET entscheiden Sie die Komplexität der Konzepte selbst mit und damit auch, wie Sie Ihre Projekte abwickeln und wieviel Zeit Sie für Schulung einplanen.

#### OO ist nicht gratis

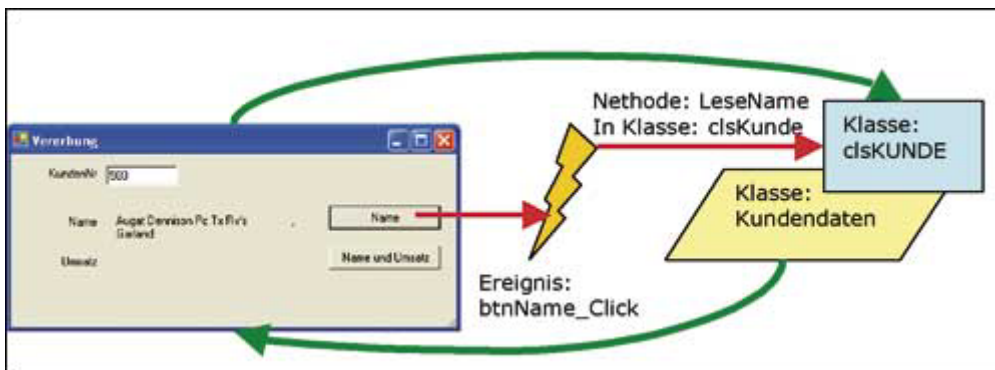
Planen Sie Ihren Einstieg in diese moderne Technologie nach Ihren Möglichkeiten und Bedürfnissen. Bedenken Sie, dass OO-Technologien in der Planung wesentlich aufwändiger sind als herkömmliche Projekte – der ROI kommt erst später. Bekanntlich betragen die Wartungs- und Änderungskosten über den gesamten Lebenszyklus gesehen mehr als 80 Prozent der Projekt-Gesamtkosten. Das ist Grund genug, um sich mit OO zu beschäftigen. Machen Sie nur nicht den Fehler, Ihrem Vorgesetzten vor Start ihres ersten OO-Projekts von zu erwartenden Einsparungen zu erzählen, die Sie in der Anfangsphase sicher nicht haben werden. Rechnen Sie mit Investitionen in die Entwicklungsumgebung und mit Schulungskosten für Ihr Team.

#### Objekte, Eigenschaften, Ereignisse und Methoden

Zuerst geht es um die Klärung einiger Begriffe der Objekt-Orientierung. Ein Objekt – wie zum Beispiel ein Printfile – ist RPG-Programmierern nicht neu. Der Printfile hat Attribute – wie zum Beispiel eine bestimmte OUTQ (Eigenschaft). Ein Ereignis (Event) wird vom Benutzer oder der Programmlogik ausgelöst, beim Printfile wäre das die Eröffnung in einem Programm. Soweit wenden wir iSeries-Programmierer die Objekt-Orientierung bereits an. Natürlich geht das auf .NET weiter. In einem Windows-Dialogprogramm besitzt ein Button die „Click“-Eigenschaft mit einer damit verbundenen Ereignisbehandlung. Innerhalb dieser Ereignisroutine wird eine Methode eines Objekts verwendet, um Daten zur Verfügung zu stellen. Dieses Objekt baut auf ein bereits vorhandenes Objekt auf und ergänzt nur mehr die geforderten Daten. Es braucht sich nicht mehr darum zu kümmern, wie die Basisdaten zustande gekommen sind. Die Routine, die die Daten bereitstellt, wird „OO-Methode“ genannt. Methoden kann man Parameter übergeben und sie geben üblicherweise eine Klasse als Ergebnis zurück.

#### Vererbung und Kapselung

Durch den Verweis auf das Basisobjekt „erbt“ eine Klasse Eigenschaften sowie Methoden seiner Basisklasse. Innerhalb seiner Logik greift das ableitende Objekt auf Arbeitsfelder zu, die nur in der eigenen Routine existieren. Dadurch kann das Feld i (typischer Name für Schleifenzähler) in jeder Routine vorkommen, ohne dass man sich Gedanken machen muss, ob man einen Wert verändert, der anderswo verwendet wird. In iSeries-RPG sind alle Variablen für das komplette Programm gültig – sie sind also „global“. Die Möglichkeit, einen Gültigkeitsbereich (Scope) für Variablen anzugeben, sowie das „Verstecken“ der Logik, das heißt, wie die Daten zustande gekommen sind, nennt man „Kapselung“.



Click-Ereignis auf Button "Name"

```

// EreignisSubroutine wird aufgerufen wenn auf den Button "btnName" geklickt wird
BegSr btnName_Click Access(*Private) Event(*this.btnName.Click)
  DeclSrParm sender Type(*Object)
  DeclSrParm e Type(System.EventArgs)

  // deklarieren eines nur für diese Routine gültigen Arbeitsfeldes
  DeclFld zonKndNr Type(*zoned) Len(9,0)

  // aufrufen einer Subroutine die Ausgabefelder löscht
  ExSr ClearControls

  // Eingabewert aus Zeichenfeld in gezontes Feld übernehmen
  // um einen möglichen Fehler abzufangen
  Try
    zonKndNr = txtKndNr.Text
  Catch ex Exception
    // Info an Benutzer im Statusbalken
    stb.Text = "Kundennummer ungültig"
    LeaveSr
  EndTry

  // Kundendaten aus der Klasse "clsKunde" in der DLL "KundenObj"
  // einlesen, übergeben wird die gezonte Kundennummer
  DeclFld cKundenDaten Type(KundenObj.clsKunde.Kundendaten)
  cKundenDaten = cKunde.LeseName(zonKndNr)

  // wenn die Kundennummer nicht gesetzt ist
  // wird angenommen dass der Satz nicht existiert
  If cKundenDaten.KundenNr = *zero
    stb.Text = "Kunde " + zonKndNr.ToString() + " wurde nicht gefunden"
  Else
    // wenn Kundennummer gefunden wurde Daten anzeigen
    lblKunde.Text = cKundenDaten.KundenName + ", " + cKundenDaten.KundenOrt
  Endif

EndSr

```

Ereignisbehandlungsroutine für Button "Name"

#### „Keep it simple“...

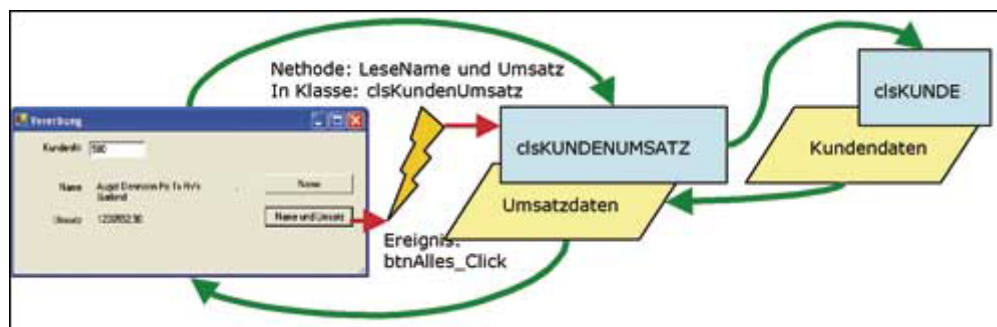
... heißt die oberste Regel. Nehmen sie sich für ihr erstes Projekt nicht allzu viel vor. Unser Beispiel in Abbildung 1 zeigt folgenden Ablauf: Der Benutzer klickt auf den Button „Name“, daraufhin wird die Ereignisroutine in Abbildung 2 ausgelöst. Wie Sie hier sehen, beschäftigt sich das Programm gar nicht mit dem Datenzugriff. Die Daten werden in der Klasse „clsKUNDE“ (Abbildung 3) eingelesen und dann als Klasse an das Programm zurückgegeben. Somit muss das aufrufende Programm weder wissen, wo die Daten zu finden noch wie sie strukturiert sind.

```

1 Using System
2
3 // Kunde als Basisklasse
4 BegClass clsKunde Access(*Public)
5
6 Deklarationen
7
8
9
10
11
12
13 // Methode zum einlesen der KundenDaten,
14 // übergeben wird eine Kundennummer
15 // zurückgegeben wird eine Klasse mit KundenDaten
16
17 BegFunc LeseName Type(KundenDaten) Access(*Public)
18     DclSrParm zonKndNr Type(*zoned) Len(9,0) // Parameter Kundennummer
19
20 // erzeugen der Klasse "KundenDaten"
21 DclFld cKundenDaten Type(KundenDaten)
22 cKundenDaten = *new KundenDaten()
23
24 // Zugriff auf den Kundenstammsatz
25 Chain CMastNewLi Key(zonKndNr) Err(*extended)
26 If NOT %Error() and %found()
27     // füllen der Eigenschaftsfelder der Klasse "KundenDaten"
28     cKundenDaten.KundenNr = CMCustNo
29     cKundenDaten.KundenName = CMName
30     cKundenDaten.KundenOrt = CMCity
31 Endif
32
33 // Klasse KundenDaten an den Aufrufer zurückgeben
34 LeaveSR cKundenDaten
35
36 Endfunc
37
38 // Klasse KundenDaten erstellen
39 BegClass KundenDaten Access(*Public)
40
41 // Felder als Eigenschaften der Klasse festlegen |
42 DclFld KundenNr Like(CMCustNo) Access(*Public)
43 DclFld KundenName Like(CMName) Access(*Public)
44 DclFld KundenOrt Like(CMCity) Access(*Public)
45
46 EndClass
47
48
49 EndClass
50

```

Klasse "Kunde" mit Methode "LeseName" und eingebetteter Klasse "KundenDaten"



Klasse "Kundenumsatz" benutzt Klasse "Kunde" als Basisklasse

```

30 // Methode zum einlesen von KundenDaten und Umsätze,
31 // übergeben wird eine Kundennummer
32 // zurückgegeben wird eine Klasse die KundenDaten und Umsätze enthält
33 BegFunc LeseNameUndUmsatz Type(UmsatzDaten) Access(*Public)
34     DclSrParm zonKndNr Type(*zoned) Len(9,0)
35
36     // erzeugen der Klasse "UmsatzDaten"
37     DclFld cUmsatzDaten Type(UmsatzDaten)
38     cUmsatzDaten = *new UmsatzDaten()
39
40     // erzeugen der Klasse "KundenDaten"
41     // und einlesen aus der Methode der Basisklasse
42     // in die Felder der Klasse die zurückgegeben wird
43     DclFld cKundenDaten Type(clsKunde.KundenDaten)
44     cKundenDaten = *base.LeseName(zonKndNr)
45
46     // wenn Kundennummer nicht gefunden wurde
47     // gar nicht erst weitersuchen
48     If cKundenDaten.KundenNr > *zero
49
50         // Werte in die "geerbten" Felder einstellen
51         cUmsatzdaten.KundenName = cKundenDaten.KundenName
52         cUmsatzdaten.KundenNr = cKundenDaten.KundenNr
53         cUmsatzdaten.KundenOrt = cKundenDaten.KundenOrt
54
55         // Zugriff auf die Umsatzzdaten und Einlesen in die Array
56         Chain CSMasterLl Key(zonKndNr) Err(*extended)
57         If NOT *Error() and *found()
58             // Summieren der Array und füllen des Summenfeldes
59             cUmsatzDaten.KundenUmsatz = %xfoot(SalesArr)
60         Endif
61
62     Endif
63
64     // Klasse Umsatzzdaten an den Aufrufer zurückgeben
65     LeaveSR cUmsatzDaten
66
67 Endfunc
68
69 // die Klasse "UmsatzDaten" erweitert die Klasse "KundenDaten"
70 BegClass UmsatzDaten Extends(clsKunde.KundenDaten) Access(*Public)
71
72     // hier brauchen nur mehr die Felder angegeben werden
73     // die zu den Feldern der Basisklasse hinzukommen
74     DclFld KundenUmsatz Type(*packed) Len(13,2) Access(*Public)
75     // die Felder Nummer, Name, Ort werden von der Basisklasse geerbt und
76     // brauchen hier nicht angegeben zu werden
77
78 EndClass

```

Klasse "Kundenumsatz" erbt von Basisklasse "Kunde"

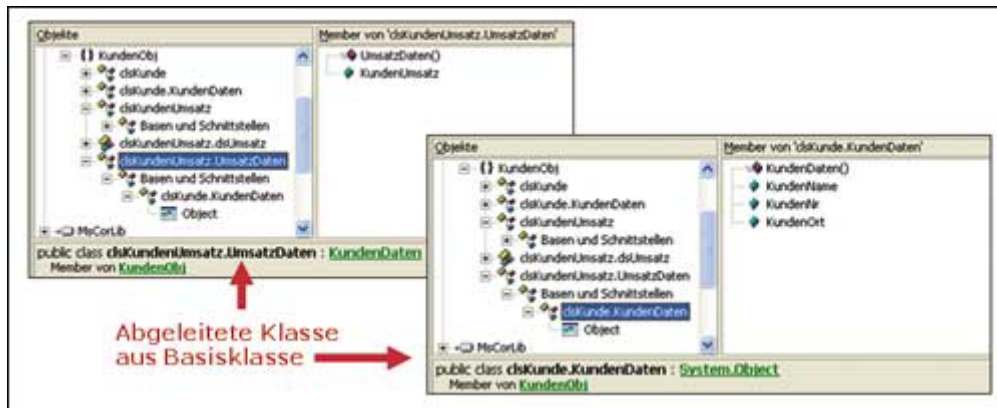
#### Wiederverwenden von Objekten

Die Klasse „Kundenumsatz“ wird von der Klasse „clsKunde“ abgeleitet, da sie nicht nur den Umsatz, sondern auch die Kundendaten an den Aufrufer zurückgeben soll. Sie bedient sich der Klasse „clsKunde“ als Basisklasse, da eine mehrfache Kodierung der Einleselogik für die Kundendaten zu vermeiden ist (siehe Abbildung 4). In Abbildung 5 finden sie den Code der Methode „LeseNameUndUmsatz“ der Klasse „Kundenumsatz“. Wie sie sehen, wird hier die Basisklasse verwendet.

#### Kommt Ihnen das bekannt vor?

Wenn Sie diese Vorgangsweise mit Ihrer gewohnten Arbeitsweise vergleichen, dann kommt Ihnen das sicher bekannt vor. In Ihren aktuellen Programmen werden sie – ohne OO – ähnlich vorgehen.

Ein wesentlicher Unterschied ist, dass sie die Schnittstelle zwischen Ereignis und aufgerufener Klasse von der Klasse selbst erben und nur Felder ergänzen (Abbildung 5: Klasse „Umsatzdaten“). In RPG haben sie ja nicht die Möglichkeit, die Felder einfach vom aufrufenden Programm zu erben.



Klassenhierarchie in Objekt-Browser

In der Objektdarstellung von VisualStudio in Abbildung 6 wird die Klasse mit ihrer Objekthierarchie dargestellt.

### Respekt ist angebracht

Auch wenn Sie nun zu dem Schluss kommen, dass dieses Beispiel nicht wirklich aufregend ist, sollten Sie bedenken, dass die Komplexität von OO-Umgebungen sehr schnell zunehmen kann. Angst oder Scheu vor der OO-Welt ist unangebracht, Respekt schon. Sie haben als RPG-Entwickler eine gute Wissensbasis, um in die objektorientierte Programmierung einzusteigen.