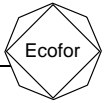




BTree2.dll

Realisiert mit Microsoft Visual Studio



INHALT

1. Allgemein	3
2. Class BTree2	3
3. Beschreibung BTree2	4
4. Anwendungsbeispiel	5

1. Allgemein

Die Klasse **BTree2** realisiert Tabellen deren interne Struktur ein ausgeglichener Baum ist (AVL balanced Tree)¹. Die Realisierung von solchen Bäumen basiert auf rekursiven Methoden welche sehr effizient sind. Beim rekursiven Durchlaufen (Apply) ist kein Aufsetzen nach einem Unterbruch möglich. Im Unterschied zur Klasse **BTree1** entfällt hier diese Einschränkung, indem die Elemente zusätzlich verkettet, und diesbezüglich gleichzeitig mit der Balancierung aktuell sind. Sie hat deshalb 4 zusätzliche Funktionen (Back, Fore, Open, Read) und eine Klasse Cursor.

BTree2 kapselt die beiden Klassen Data und Table. **Data** bildet die Schnittstelle für die Anwenderdaten welche **Table** verwaltet. Sie muss deshalb geerbt, und deren Funktion **Compare** überschrieben werden. Nur der implementierenden Instanz sind ihre Daten bekannt. Die Ordnung der Data-Elemente wird von Compare determiniert. Dieser Sachverhalt wird bei den einzelnen Funktionen nicht mehr erwähnt, kann jedoch von Bedeutung sein, wenn durch Umkehrung des Funktionswertes (-1 | 0 | +1) absteigend sortiert wird.

2. Class BTree2

```
Public Class Cursor
    Public EOF As Boolean
End Class ' Cursor

Public MustInherit Class Data
    Public MustOverride Function Compare(ByRef d As Data) As Short
    Public Overridable Sub Apply(Optional ByRef handle As Object = Nothing)
End Class ' Data

Public Class Table
    Public Sub New()
    Public Sub Add(ByRef d As Data)
    Public Sub Apply(Optional ByRef handle As Object = Nothing)
    Public Sub Delete(f As Data)
    Public Sub Dispose()
    Public Function Empty() As Boolean
```

¹ <http://de.wikipedia.org/wiki/AVL-Baum>

```
Public Function Exists(ByRef f As Data, sq As String) As Boolean
Public Function Find(f As Data, sq As String) As Data
Public Function First() As Data
Public Function Last() As Data
Public Sub Replace(ByRef d As Data)

Public Function Back() As Data
Public Function Fore() As Data

Public Sub Open(fore As Boolean, ByRef curs As BTree2.Cursor)
Public Sub Read(curs As BTree2.Cursor, ByRef d As Data)
End Class ' Table
```

3. Beschreibung BTree2

Parameter sq: | < | = | > | <= | >= | für Qualität des gesuchten, im Verhältnis zum angegebenen Key. Zweiwertige Symbole dürfen vertauscht werden.

Add: Fügt dem Baum das Daten-Element d hinzu.

Apply: **Table.Apply** durchläuft **Data.Apply** rekursiv (auf- bzw. absteigend, je nach Compare-Implementation beim Einfügen).

Back: Liefert im Funktionswert das vorherige Daten-Element bzw. Nothing.

Compare: z.B. Znr (Integer) in der Daten-Klasse „**Eline**“, aufsteigend sortiert.

Delete: Entfernt das in f gesuchte Daten-Element.

Dispose: Entfernt alle Elemente.

Empty: Liefert TRUE im Funktionswert wenn keine Elemente vorhanden sind.

Exists: Liefert TRUE im Funktionswert wenn das in f gesuchte und in sq qualifizierte Daten-Element vorhanden ist.

Find: Liefert im Funktionswert das in f gesuchte und in sq qualifizierte Daten-Element bzw. Nothing.

```
If Znr < CType(d, Eline).Znr Then : Return -1
Elseif Znr > CType(d, Eline).Znr Then : Return +1
Else : Return 0
End If
```

First: Liefert im Funktionswert das erste Daten-Element bzw. Nothing.

Fore: Liefert im Funktionswert das nächste Daten-Element bzw. Nothing.

Last: Liefert im Funktionswert das letzte Daten-Element bzw. Nothing.

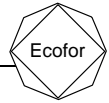
New: Konstruktor (der Destruktor ist implizit realisiert).

Open / Read : Open stellt einen Cursor für das sequentielle Lesen der Elemente zur Verfügung, mit fore=TRUE vorwärts, andernfalls rückwärts. Open positioniert einzig den Cursor, ohne das Element zu lesen, und behandelt EOF. Es kann eine beliebige Anzahl Cursor aktiv sein. Solange curs.EOF nicht erreicht ist stellt Read bei jedem Aufruf ein Element zur Verfügung.

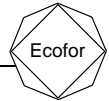
Replace: Ersetzt das Daten-Element **d** im Baum.

4. Anwendungsbeispiel

Im Beispiel auf den folgenden Seiten werden BTree1 und BTree2 verwendet. Es speichert alle Kunden und von diesen alle Monatsumsätze in Bäume (KundeTab mit KundeEle => JahrTab mit MonatEle). Kunden sind nach Name aufsteigend, und die Monate absteigend sortiert. Wenn die Bäume gefüllt sind wird pro Kunde das Umsatztotal ausgewertet und zuletzt das Gesamttotal. Danach werden alle Kunden mit „B“ bzw. „b“ an erster Stelle im Namen ausgewertet. Das Beispiel würde in der Praxis natürlich mit SQL realisiert. Zudem wurde die Abarbeitung der Problemstellung bewusst nicht geschachtelt (Umsätze laden!).



```
Module Demo
' Definition Kunden-Daten
Private Class KundeEle
    Inherits BTree2.Data
    Public Name As String
    Public KID As Integer
    Public Branche As Integer
    Public JahrTab As BTree1.Table
    Public Overrides Function Compare(ByRef d As BTree2.Data) As Short ' liefert -1, 0, +1
        Compare = String.Compare(UCase(Name), UCase(CType(d, KundeEle).Name)) ' A,A,B,B ..
        If Compare = 0 Then
            Compare = String.Compare(Name, CType(d, KundeEle).Name) ' a,A,b,B ..
            If Compare = 0 Then
                If KID > CType(d, KundeEle).KID Then : Compare = +1
                ElseIf KID < CType(d, KundeEle).KID Then : Compare = -1
                End If
            End If
        End If
    End Function
    Public Sub New()
        JahrTab = New BTree1.Table
    End Sub
    Public Overrides Sub Apply(Optional ByRef total As Object = Nothing)
        Dim Umsatz As Double = 0
        JahrTab.Apply(Umsatz)
        Debug.WriteLine(Name & ": " & Umsatz)
        total += Umsatz
    End Sub
End Class
' Definition Umsatz-Daten
Private Class MonatEle
    Inherits BTree1.Data
    Public Monat As Byte
    Public Umsatz As Single
    Public Overrides Function Compare(ByRef d As BTree1.Data) As Short
        If Monat < CType(d, MonatEle).Monat Then : Return +1 ' sortiert absteigend
        ElseIf Monat > CType(d, MonatEle).Monat Then : Return -1
        Else : Return 0
        End If
    End Function
End Class
```

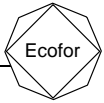


```
Public Overrides Sub Apply(Optional ByRef total As Object = Nothing)
    CType(total, Double) += Umsatz
End Sub
End Class

' Demo-Programm
Public Sub Demo()
    Dim adoc As New ADODB.Connection ' Server für Recordsets
    adoc.Open("File Name=Demo.udl;")
    Dim KundeTab As New BTree2.Table
    Dim kunde As KundeEle
    Dim rs As New ADODB.Recordset

    ' Kunden laden
    rs.Open("SELECT KID, Name, Branche FROM TKunde", adoc)
    While Not rs.EOF
        kunde = New KundeEle
        kunde.Name = rs.Fields("Name").Value
        kunde.KID = rs.Fields("KID").Value
        kunde.Branche = rs.Fields("Branche").Value
        KundeTab.Add(kunde)
        rs.MoveNext()
    End While
    rs.Close()

    ' Umsätze laden
    Dim monat As MonatEle
    Dim curs As New BTree2.Cursor
    KundeTab.Open(True, curs)
    While Not curs.EOF
        KundeTab.Read(curs, kunde)
        rs.Open("SELECT Monat, Umsatz FROM TUmsatz WHERE KID=" & kunde.KID, adoc)
        While Not rs.EOF
            monat = New MonatEle
            monat.Monat = rs.Fields("Monat").Value
            monat.Umsatz = rs.Fields("Umsatz").Value
            kunde.JahrTab.Add(monat)
            rs.MoveNext()
        End While
        rs.Close()
    End While
End Sub
```



```
' Umsätze pro Kunde und Gesamt auswerten
Dim Umsatz As Double = 0
KundeTab.Apply(Umsatz)
Debug.WriteLine("Gesamt: " & Umsatz)

' alle Kunden mit B an erster Stelle im Namen auswerten
Dim fkunde As New KundeEle
fkunde.Name = "B"
fkunde.KID = 0
kunde = KundeTab.Find(fkunde, ">=")
While Not kunde Is Nothing AndAlso UCase(Left(kunde.Name, 1)) = "B"
    Debug.WriteLine(kunde.Name)
    kunde = KundeTab.Fore()
End While
End Sub
End Module
```