Dosyalar ve Kaynaklar

Giriş

- Değişkenler ve diziler bilgileri geçici olarak saklar
  - Garbage collection tarafından silinirler veya programın çalışması kesildiğinde kaybolurlar
- Uzun dönem bilgi saklamak için dosyalar (files) kullanılır
  - Sürekli kalıcı bilgi (persistent data)
- Bu bölümde:
  - Sıralı erişimli dosyalar (Sequential-access files)
  - Rastgele erişimli dosyalar (Random-access files)
  - Dosya işlem özellikleri (File processing features)
  - Kaynak giriş/çıkış özellikleri (Stream-input/output features)

Veri Hiyerarşisi

- Veri hiyerarşisi:
  - Bilgiler boyutu küçükten büyüğe gidildikçe karmaşıklaşır:
    - Bit: bir veya sıfır
      - Bütün bilgiler bitlerle ifade edilir
    - Byte: Sekiz bitten oluşur
    - Karakter: C#'ta iki byte boyutundadır
      - Karakter kümesi (Character set): program için kullanılan tüm karakterler ve bilgileri gösteren karakterler
    - Alan (Field): bir anlam taşıyan karakterler birleşimidir
    - Kayıt (Record): çok sayıdaki ilişkili alanın birleşimidir
    - Dosya (File): ilişkili kayıtların oluşturduğu gruptur
      - Record key: belirli bir giriş için kayıtları tanımlar
      - Sequential file: record-key bilgisine göre sıralı kayıtları saklar

Veri Hiyerarşisi

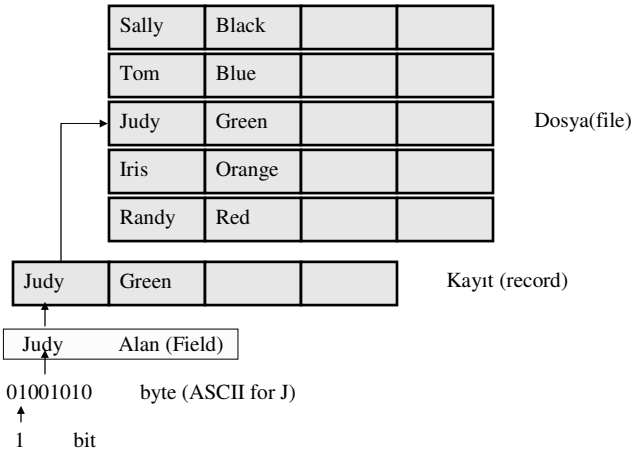


Fig. 17.1 Veri Hiyerarşisi.

Dosyalar ve Kaynaklar (Files and Streams)

- Dosyalar byte'ların sıralı kaynaklarıdır
  - Bir dosya sonu işaretçisi (end-of-file) veya belirli bir byte ile bitirilir
- C#' ta bir dosya açıldığında:
  - Bir nesne (object) oluşturulur
  - Bu nesne bir kaynakla ilişkilendirilir
    - Üç tane stream nesnesi vardır:
      - Console.In: standard giriş stream nesnesini döndürür
      - Console.Out: standard çıkış stream nesnesini döndürür
      - Console.Error: standard hata stream nesnesini döndürür
- Namespace **`System.IO`** dosya işlemleri için gereklidir

Dosyalar ve Kaynaklar (Files and Streams)

- BinaryFormatter: nesneleri serialize ve deserialize yapmak için kullanılır
  - Serialize: bir nesnenin bilgi kaybolmadan dosyaya yazılması için bir formata dönüştürülmesi
  - Deserialize: orijinal nesnenin dosyadan okunan formatlı text ile yeniden yapılandırılması
- **`System.IO.Stream`**: stream'lerin gösteriminin bitlerle yapılmasını sağlar
  - **`FileStream`**: sıralı erişimli (sequential-access) ve random erişimli (random-access) dosyaya yazma ve okuma
  - **`MemoryStream`**: bilginin doğrudan hafızaya aktarılması
  - **`BufferedStream`**: hafızaya bilgi aktarımında tampon bellek (buffer) kullanılır

Dosyalar ve Kaynaklar (Files and Streams)

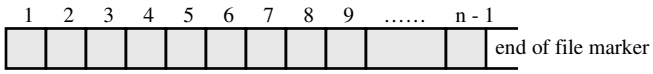


Fig. 17.2 C#'ta *n-byte* dosyanın görünümü.

Dosya (File) ve Dizin (Directory) Sınıfları

- Bilgi dosyalarda saklanır
  - Dosyalar dizinlerde organize edilir
    - **`Directory`** sınıfı dizinlerle işlemler yapmak için kullanılır
  - **`File`** sınıfı dosyalarla işlemler yapmak için kullanılır
    - Sadece static metodlar vardır, **`File`** nesneleri örneklenemezler

Dosya Sınıfı

`static` Method	Description
`AppendText`	Returns a `StreamWriter` that appends to an existing file or creates a file if one does not exist.
`Copy`	Copies a file to a new file.
`Create`	Creates a file and returns its associated `FileStream`.
`CreateText`	Creates a text file and returns its associated `StreamWriter`.
`Delete`	Deletes the specified file.
`GetCreationTime`	Returns a `DateTime` object representing the time that the file was created.
`GetLastAccessTime`	Returns a `DateTime` object representing the time that the file was last accessed.
`GetLastWriteTime`	Returns a `DateTime` object representing the time that the file was last modified.
`Move`	Moves the specified file to a specified location.
`Open`	Returns a `FileStream` associated with the specified file and equipped with the specified read/write permissions.
`OpenRead`	Returns a read-only `FileStream` associated with the specified file.
`OpenText`	Returns a `StreamReader` associated with the specified file.
`OpenWrite`	Returns a read/write `FileStream` associated with the specified file.

Fig. 17.3 `File` class methods (partial list).

Dizin Sınıfı

`static` Method	Description
`CreateDirectory`	Creates a directory and returns its associated `DirectoryInfo`.
`Delete`	Deletes the specified directory.
`Exists`	Returns `true` if the specified directory exists; otherwise, it returns `false`.
`GetLastWriteTime`	Returns a `DateTime` object representing the time that the directory was last modified.
`GetDirectories`	Returns a `string` array representing the names of the subdirectories in the specified directory.
`GetFiles`	Returns a `string` array representing the names of the files in the specified directory.
`GetCreationTime`	Returns a `DateTime` object representing the time that the directory was created.
`GetLastAccessTime`	Returns a `DateTime` object representing the time that the directory was last accessed.
`GetLastWriteTime`	Returns a `DateTime` object representing the time that items were last written to the directory.
`Move`	Moves the specified directory to a specified location.

Fig. 17.4 `Directory` class methods (partial list).

```
1    // Fig 17.5: FileTest.cs
2    // Using classes File and Directory.
3
4    using System;
5    using System.Drawing;
6    using System.Collections;
7    using System.ComponentModel;
8    using System.Windows.Forms;
9    using System.Data;
10   using System.IO;
11
12   // displays contents of files and directories
13   public class FileTestForm : System.Windows.Forms.Form
14   {
15      private System.Windows.Forms.Label directionsLabel;
16
17      private System.Windows.Forms.TextBox outputTextBox;
18      private System.Windows.Forms.TextBox inputTextBox;
19
20      private System.ComponentModel.Container components = null;
21
22      [STAThread]
23      static void Main()
24      {
25         Application.Run( new FileTestForm() );
26      }
27
28      // Visual Studio .NET generated code
29
30      // invoked when user presses key
31      private void inputTextBox_KeyDown(
32         object sender, System.Windows.Forms.KeyEventArgs e )
33      {
```

Textbox for input of file or directory name

Event handler for keystrokes

```
34         // determine whether user pressed Enter key
35         if ( e.KeyCode == Keys.Enter )
36         {
37            string fileName; // name of file or directory
38
39            // get user-specified file or directory
40            fileName = inputTextBox.Text;
41
42            // determine whether fileName is a file
43            if ( File.Exists( fileName ) )
44            {
45               // get file's creation date,
46               // modification date, etc.
47               outputTextBox.Text = GetInformation( fileName );
48
49               // display file contents through StreamReader
50               try
51               {
52                  // obtain reader and file contents
53                  StreamReader stream = new StreamReader( fileName );
54                  outputTextBox.Text += stream.ReadToEnd();
55               }
56               // handle exception if StreamReader is unavailable
57               catch( IOException )
58               {
59                  MessageBox.Show( "File Error", "File Error",
60                     MessageBoxButtons.OK, MessageBoxIcon.Error );
61               }
62            }
63
64            // determine whether fileName is a directory
65            else if ( Directory.Exists( fileName ) )
66            {
67               // array for directories
68               string[] directoryList;
```

Test if key typed was enter

Set fileName to what user typed

See if fileName is an existing file

If an existing file, get and output file information

Create StreamReader to read text from file

Call method ReadToEnd

Test if fileName is existing directory

```
69
70              // get directory's creation date,
71              // modification date, etc.
72              outputTextBox.Text = GetInformation( fileName );
73
74              // obtain file/directory list of specified directory
75              directoryList = Directory.GetDirectories( fileName );
76
77              outputTextBox.Text +=
78                 "\r\n\r\nDirectory contents:\r\n";
79
80              // output directoryList contents
81              for ( int i = 0; i < directoryList.Length; i++ )
82                 outputTextBox.Text += directoryList[ i ] + "\r\n";
83           }
84           else
85           {
86              // notify user that neither file nor directory exists
87              MessageBox.Show( inputTextBox.Text +
88                 " does not exist", "File Error",
89                 MessageBoxButtons.OK, MessageBoxIcon.Error );
90           }
91        } // end if
92
93     } // end method inputTextBox_KeyDown
94
95     // get information on file or directory
96     private string GetInformation( string fileName )
97     {
98        // output that file or directory exists
99        string information = fileName + " exists\r\n\r\n";
100
101       // output when file or directory was created
102       information += "Created: " +
103          File.GetCreationTime( fileName ) + "\r\n";
```

If it exists, get and output directory information

Get subdirectories

Output subdirectories

If user input is not existing file or directory output error message

Tell user file exists

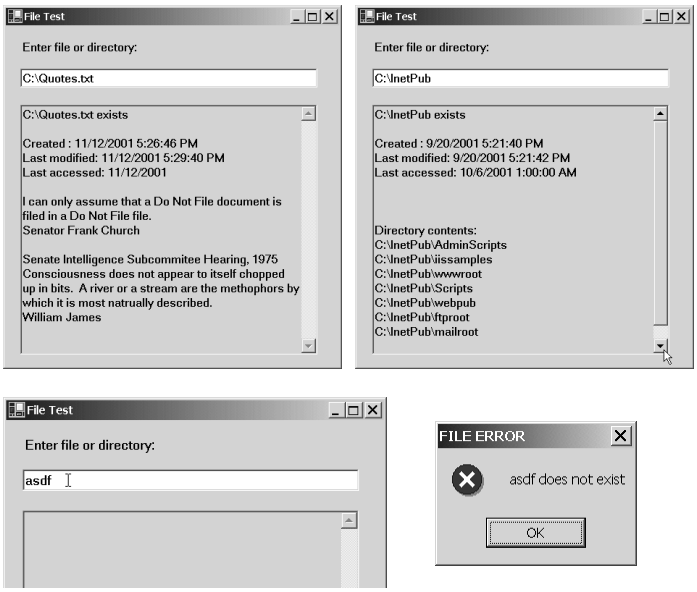
Get file creation information

```
104
105       // output when file or directory was last modified
106       information += "Last modified: " +
107          File.GetLastWriteTime( fileName ) + "\r\n";
108
109       // output when file or directory was last accessed
110       information += "Last accessed: " +
111          File.GetLastAccessTime( fileName ) + "\r\n" + "\r\n";
112
113       return information;
114
115    } // end method GetInformation
116
117 } // end class FileTestForm
```

Get last time file was modified

Get last time file was accessed

Return file information



Sequential-Access Dosya Oluşturma

- Programcılar uygulamaların gereksinimlerini karşılayacak şekilde dosyaları yapılandırmak zorundadırlar

```
1    // Fig 17.7: BankUI.cs
2    // A reusable windows form for the examples in this chapter.
3
4    using System;
5    using System.Drawing;
6    using System.Collections;
7    using System.ComponentModel;
8    using System.Windows.Forms;
9    using System.Data;
10
11   public class BankUIForm : System.Windows.Forms.Form
12   {
13      private System.ComponentModel.Container components = null;
14
15      public System.Windows.Forms.Label accountLabel;
16      public System.Windows.Forms.TextBox accountTextBox;
17
18      public System.Windows.Forms.Label firstNameLabel;
19      public System.Windows.Forms.TextBox firstNameTextBox;
20
21      public System.Windows.Forms.Label lastNameLabel;
22      public System.Windows.Forms.TextBox lastNameTextBox;
23
24      public System.Windows.Forms.Label balanceLabel;
25      public System.Windows.Forms.TextBox balanceTextBox;
26
27      // number of TextBoxes on Form'
28      protected int TextBoxCount = 4;
29
        public BankUIForm()
        {
           //
           // Required for Windows Form Designer support
           //
           InitializeComponent();
        }
```

Textboxes

Labels

```
30      // enumeration constants specify TextBox indices
31      public enum TextBoxIndices
32      {
33         ACCOUNT,
34         FIRST,
35         LAST,
36         BALANCE
37
38      } // end enum
39
40      [STAThread]
41      static void Main()
42      {
43         Application.Run( new BankUIForm() );
44      }
45
46      // Visual Studio .NET generated code
47
48      // clear all TextBoxes
49      public void ClearTextBoxes()
50      {
51         // iterate through every Control on form
52         for ( int i = 0; i < Controls.Count; i++ )
53         {
54            Control myControl = Controls[ i ]; // get control
55
56            // determine whether Control is TextBox
57            if ( myControl is TextBox )
58            {
59               // clear Text property (set to empty strng)
60               myControl.Text = "";
61            }
62         }
63
64      } // end method ClearTextBoxes
```

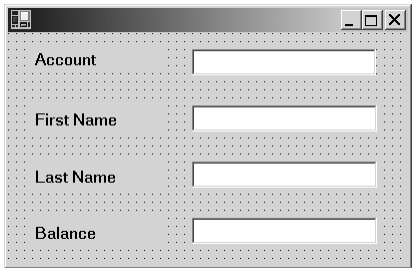
Method to clear textboxes

```
65
66      // set text box values to string array values
67      public void SetTextBoxValues( string[] values )
68      {
69         // determine whether string array has correct length
70         if ( values.Length != TextBoxCount )
71         {
72            // throw exception if not correct length
73            throw( new ArgumentException( "There must be " +
74               (TextBoxCount + 1) + " strings in the array" ) );
75         }
76
77         // set array values if array has correct length
78         else
79         {
80            // set array values to text box values
81            accountTextBox.Text =
82               values[ ( int )TextBoxIndices.ACCOUNT ];
83            firstNameTextBox.Text =
84               values[ ( int )TextBoxIndices.FIRST ];
85            lastNameTextBox.Text =
86               values[ ( int )TextBoxIndices.LAST ];
87            balanceTextBox.Text =
88               values[ ( int )TextBoxIndices.BALANCE ];
89         }
90
91      } // end method SetTextBoxValues
92
93      // return text box values as string array
94      public string[ ] GetTextBoxValues()
95      {
96         string[ ] values = new string[ TextBoxCount ];
97
```

Method to set values of textboxes

Method to get the values of textboxes

```
98          // copy text box fields to string array
99          values[ ( int )TextBoxIndices.ACCOUNT ] =
100            accountTextBox.Text;
101         values[ ( int )TextBoxIndices.FIRST ] =
102            firstNameTextBox.Text;
103         values[ ( int )TextBoxIndices.LAST ] =
104            lastNameTextBox.Text;
105         values[ ( int )TextBoxIndices.BALANCE ] =
106            balanceTextBox.Text;
107
108         return values;
109
110      } // end method GetTextBoxValues
111
112   } // end class BankUIForm
```



```
1    // Fig. 17.8: Record.cs
2    // Serializable class that represents a data record.
3
4    using System;
5
6    [Serializable]
7    public class Record
8    {
9       private int account;
10      private string firstName;
11      private string lastName;
12      private double balance;
13
14      // default constructor sets members to default values
15      public Record() : this( 0, "", "", 0.0 )
16      {
17      }
18
19      // overloaded constructor sets members to parameter values
20      public Record( int accountValue, string firstNameValue,
21         string lastNameValue, double balanceValue )
22      {
23         Account = accountValue;
24         FirstName = firstNameValue;
25         LastName = lastNameValue;
26         Balance = balanceValue;
27
28      } // end constructor
29
```

Tells compiler objects of class **Record** can be represented as a set of bits

Data to go into record

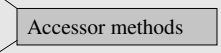
Sets members to 0

Set members to parameters

```
30      // property Account
31      public int Account
32      {
33         get
34         {
35            return account;
36         }
37
38         set
39         {
40            account = value;
41         }
42
43      } // end property Account
44
45      // property FirstName
46      public string FirstName
47      {
48         get
49         {
50            return firstName;
51         }
52
53         set
54         {
55            firstName = value;
56         }
57
58      } // end property FirstName
59
```

Accessor methods

```
60      // property LastName
61      public string LastName
62      {
63         get
64         {
65            return lastName;
66         }
67
68         set
69         {
70            lastName = value;
71         }
72
73      } // end property LastName
74
75      // property Balance
76      public double Balance
77      {
78         get
79         {
80            return balance;
81         }
82
83         set
84         {
85            balance = value;
86         }
87
88      } // end property Balance
89
90   } // end class Record
```




```
1    // Fig 17.9: CreateSequentialAccessFile.cs
2    // Creating a sequential-access file.
3
4    // C# namespaces
5    using System;
6    using System.Drawing;
7    using System.Collections;
8    using System.ComponentModel;
9    using System.Windows.Forms;
10   using System.Data;
11   using System.IO;
12   using System.Runtime.Serialization.Formatters.Binary;
13   using System.Runtime.Serialization;
14
15   // Deitel namespace
16   using BankLibrary;
17
18   public class CreateFileForm : BankUIForm
19   {
20      private System.Windows.Forms.Button saveButton;
21      private System.Windows.Forms.Button enterButton;
22      private System.Windows.Forms.Button exitButton;
23
24      private System.ComponentModel.Container components = null;
25
26      // serializes Record in binary format
27      private BinaryFormatter formatter = new BinaryFormatter();
28
29      // stream through which serializable data is written to file
30      private FileStream output;
31
```

```
32      [STAThread]
33      static void Main()
34      {
35         Application.Run( new CreateFileForm() );
36      }
37
38      // Visual Studio .NET generated code
39
40      // invoked when user clicks Save button
41      private void saveButton_Click(
42         object sender, System.EventArgs e )
43      {
44         // create dialog box enabling user to save file
45         SaveFileDialog fileChooser = new SaveFileDialog();
46         DialogResult result = fileChooser.ShowDialog();
47         string fileName; // name of file to save data
48
49         // allow user to create file
50         fileChooser.CheckFileExists = false;
51
52         // exit event handler if user clicked "Cancel"
53         if ( result == DialogResult.Cancel )
54            return;
55
56         // get specified file name
57         fileName = fileChooser.FileName;
58
59         // show error if user specified invalid file
60         if ( fileName == "" || fileName == null )
61            MessageBox.Show( "Invalid File Name", "Error",
62               MessageBoxButtons.OK, MessageBoxIcon.Error );
```

Instantiate SaveFileDialog object

Show SaveFileDialog

Test if user canceled save

Get file name to save to

```
63         else
64         {
65            // save file via FileStream if user specified valid file
66            try
67            {
68               // open file with write access
69               output = new FileStream( fileName,
70                  FileMode.OpenOrCreate, FileAccess.Write );
71
72               // disable Save button and enable Enter button
73               saveButton.Enabled = false;
74               enterButton.Enabled = true;
75            }
76
77            // handle exception if file does not exist
78            catch ( FileNotFoundException )
79            {
80               // notify user if file does not exist
81               MessageBox.Show( "File Does Not Exist", "Error",
82                  MessageBoxButtons.OK, MessageBoxIcon.Error );
83            }
84         }
85      } // end method saveButton_Click
86
87      // invoke when user clicks Enter button
88      private void enterButton_Click(
89         object sender, System.EventArgs e )
90      {
91         // store TextBox values string array
92         string[] values = GetTextBoxValues();
93
94         // Record containing TextBox values to serialize
95         Record record = new Record();
96
```

Instantiate output stream with write permission

Method to save data when user clicks enter

```
97          // determine whether TextBox account field is empty
98          if ( values[ ( int )TextBoxIndices.ACCOUNT ] != "" )
99          {
100            // store TextBox values in Record and serialize Record
101            try
102            {
103               // get account number value from TextBox
104               int accountNumber = Int32.Parse(
105                  values[ ( int )TextBoxIndices.ACCOUNT ] );
106
107               // determine whether accountNumber is valid
108               if ( accountNumber > 0 )
109               {
110                  // store TextBox fields in Record
111                  record.Account = accountNumber;
112                  record.FirstName =
113                     values[ ( int )TextBoxIndices.FIRST ];
114                  record.LastName =
115                     values[ ( int )TextBoxIndices.LAST ];
116                  record.Balance = Double.Parse( values[
117                     ( int )TextBoxIndices.BALANCE ] );
118
119                  // write Record to FileStream (serialize object)
120                  formatter.Serialize( output, record );
121               }
122               else
123               {
124                  // notify user if invalid account number
125                  MessageBox.Show( "Invalid Account Number", "Error",
126                     MessageBoxButtons.OK, MessageBoxIcon.Error );
127               }
128            }
129
```

Store TextBox fields in record

Write data to file

```
130            // notify user if error occurs in serialization
131            catch( SerializationException )
132            {
133               MessageBox.Show( "Error Writing to File", "Error",
134                  MessageBoxButtons.OK, MessageBoxIcon.Error );
135            }
136
137            // notify user if error occurs regarding parameter format
138            catch( FormatException )
139            {
140               MessageBox.Show( "Invalid Format", "Error",
141                  MessageBoxButtons.OK, MessageBoxIcon.Error );
142            }
143         }
144
145         ClearTextBoxes(); // clear TextBox values
146
147      } // end method enterButton_Click
148
149      // invoked when user clicks Exit button
150      private void exitButton_Click(
151         object sender, System.EventArgs e )
152      {
153         // determine whether file exists
154         if ( output != null )
155         {
156            // close file
157            try
158            {
159               output.Close();
160            }
161
```

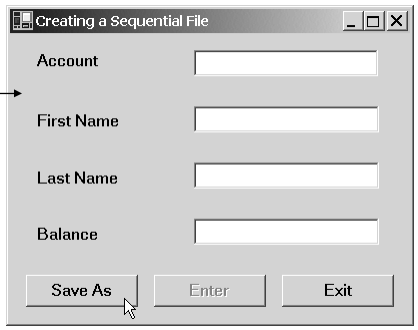
Catch block if user input invalid data

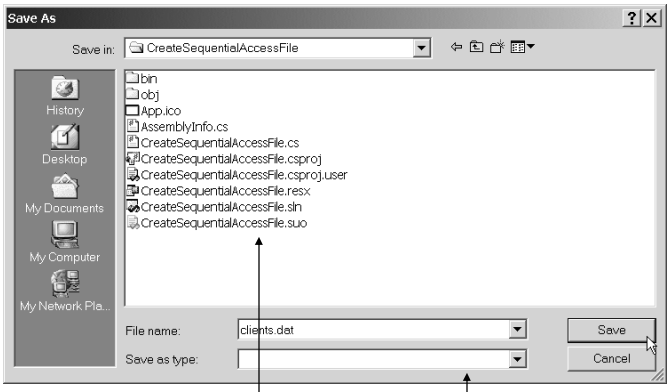
Close FileStream

```
162           // notify user of error closing file
163           catch( IOException )
164           {
165              MessageBox.Show( "Cannot close file", "Error",
166                 MessageBoxButtons.OK, MessageBoxIcon.Error );
167           }
168        }
169
170        Application.Exit();
171
172     } // end method exitButton_Click
173
174  } // end class CreateFileForm
```

Exit program

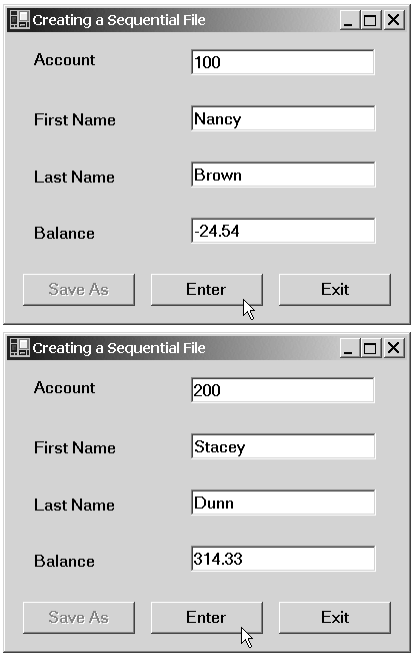
BankUI graphical user interface

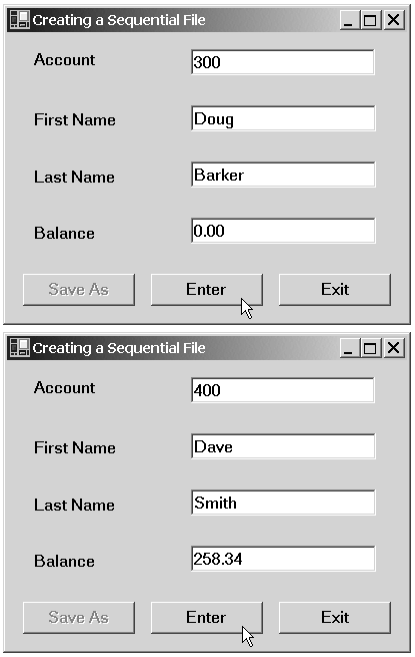


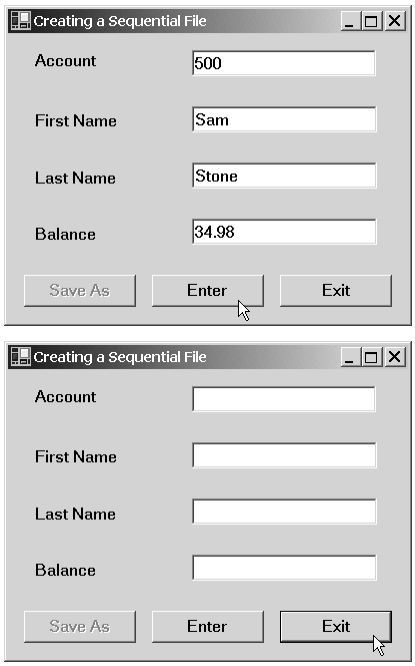


SaveFileDialogue

Files and directories


Creating a Sequential File
Account
100
First Name
Nancy
Last Name
Brown
Balance
-24.54
Save As
Enter
Exit
Creating a Sequential File
Account
200
First Name
Stacey
Last Name
Dunn
Balance
314.33
Save As
Enter
Exit


Creating a Sequential File
Account
300
First Name
Doug
Last Name
Barker
Balance
0.00
Save As
Enter
Exit
Creating a Sequential File
Account
400
First Name
Dave
Last Name
Smith
Balance
258.34
Save As
Enter
Exit



Sequential-Access Dosya Oluşturma

Account Number	First Name	Last Name	Balance
100	Nancy	Brown	-25.54
200	Stacey	Dunn	314.33
300	Doug	Barker	0.00
400	Dave	Smith	258.34
500	Sam	Stone	34.98

Fig. 17.10 Sample data for the program of Fig. 17.9.

Sequential-Access Dosyadan Bilgi Okuma

- Bilgileri dosyadan sıralı olarak oku
  - Programlar genellikle dosyanın başından başlar ve istenen bilgi bulunana kadar ardarda okuma yapar
    - Bu işlemi bazen çok defa tekrar etmek gerekir
  - Dosya pozisyonu işaretçisi (File-position pointer):
    - Okunacak veya yazılacak bilgi için dosyadaki bir sonraki byte'ı işaret eder
    - Dosyadaki başka bir noktaya yönlendirilebilir
- **`RichTextBox`**:
  - **`LoadFile`**: bir dosyanın içeriğini görüntüler
  - **`Find`**: string'leri aramak için kullanılan metoddur
  - Text bilgiyi birden çok satırda gösterebilir

```
1    // Fig. 17.11: ReadSequentialAccessFile.cs
2    // Reading a sequential-access file.
3
4    // C# namespaces
5    using System;
6    using System.Drawing;
7    using System.Collections;
8    using System.ComponentModel;
9    using System.Windows.Forms;
10   using System.Data;
11   using System.IO;
12   using System.Runtime.Serialization.Formatters.Binary;
13   using System.Runtime.Serialization;
14
15   // Deitel namespaces
16   using BankLibrary;
17
18   public class ReadSequentialAccessFileForm : BankUIForm
19   {
20      System.Windows.Forms.Button openButton;
21      System.Windows.Forms.Button nextButton;
22
23      private System.ComponentModel.Container components = null;
24
25      // stream through which serializable data are read from file
26      private FileStream input;
27
28      // object for deserializing Record in binary format
29      private BinaryFormatter reader = new BinaryFormatter();
30
31      [STAThread]
32      static void Main()
33      {
34         Application.Run( new ReadSequentialAccessFileForm() );
35      }
```

```
36
37      // Visual Studio .NET generated code
38
39      // invoked when user clicks Open button
40      private void openButton_Click(
41         object sender, System.EventArgs e )
42      {
43         // create dialog box enabling user to open file
44         OpenFileDialog fileChooser = new OpenFileDialog();
45         DialogResult result = fileChooser.ShowDialog();
46         string fileName; // name of file containing data
47
48         // exit event handler if user clicked Cancel
49         if ( result == DialogResult.Cancel )
50            return;
51
52         // get specified file name
53         fileName = fileChooser.FileName;
54         ClearTextBoxes();
55
56         // show error if user specified invalid file
57         if ( fileName == "" || fileName == null )
58            MessageBox.Show( "Invalid File Name", "Error",
59               MessageBoxButtons.OK, MessageBoxIcon.Error );
60         else
61         {
62            // create FileStream to obtain read access to file
63            input = new FileStream( fileName, FileMode.Open,
64               FileAccess.Read );
65
66            // enable next record button
67            nextButton.Enabled = true;
68         }
69
70      } // end method openButton_Click
```

Instantiate OpenFilelDialog

Display OpenFileDialog

Create FileStream object for input with read only perimission

```
71
72      // invoked when user clicks Next button
73      private void nextButton_Click(
74         object sender, System.EventArgs e )
75      {
76         // deserialize Record and store data in TextBoxes
77         try
78         {
79            // get next Record available in file
80            Record record =
81               ( Record )reader.Deserialize( input );
82
83            // store Record values in temporary string array
84            string[] values = new string[] {
85               record.Account.ToString(),
86               record.FirstName.ToString(),
87               record.LastName.ToString(),
88               record.Balance.ToString() };
89
90            // copy string array values to TextBox values
91            SetTextBoxValues( values );
92         }
93
94         // handle exception when no Records in file
95         catch( SerializationException )
96         {\
97            // close FileStream if no Records in file
98            input.Close();
99
100           // enable Open Record button
101           openButton.Enabled = true;
102
103           // disable Next Record button
104           nextButton.Enabled = false;
105
```

Method to view next record

Deserialize and cast next record

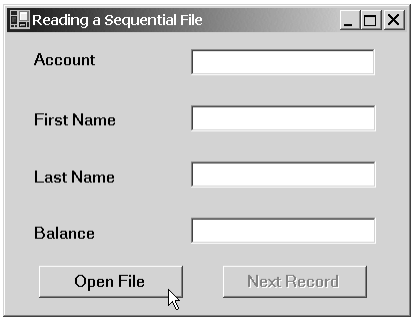
Display Record in TextBoxes

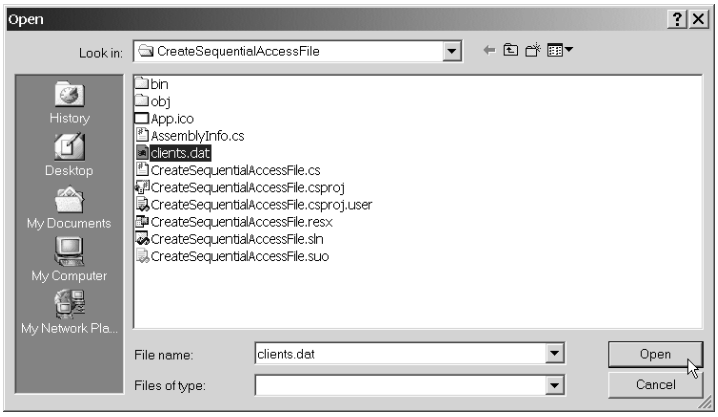
Exception thrown when there are no more records

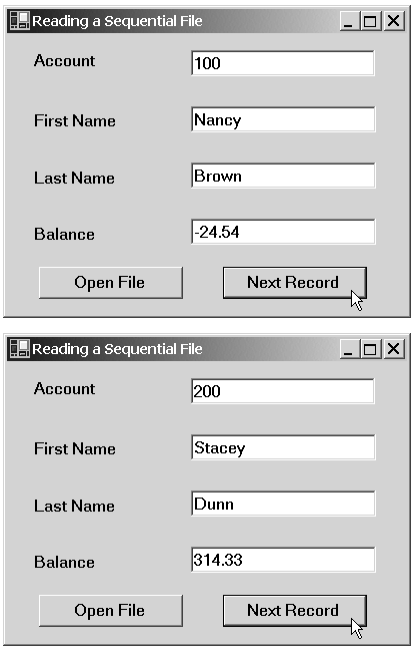
Close FileStream

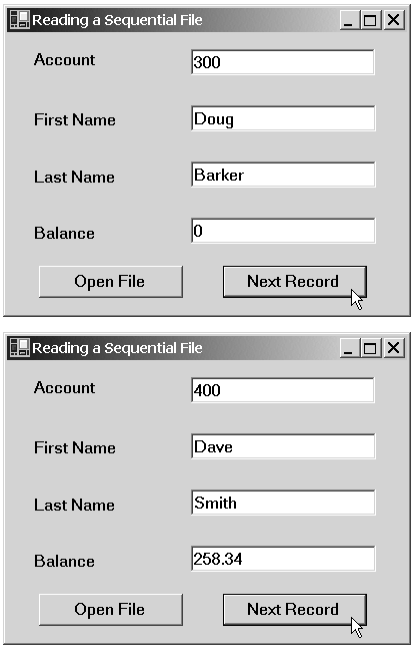
```
106           ClearTextBoxes();
107
108           // notify user if no Records in file
109           MessageBox.Show( "No more records in file", "",
110              MessageBoxButtons.OK, MessageBoxIcon.Information );
111        }
112
113     } // end method nextButton_Click
114
115  } // end class ReadSequentialAccessFileForm
```

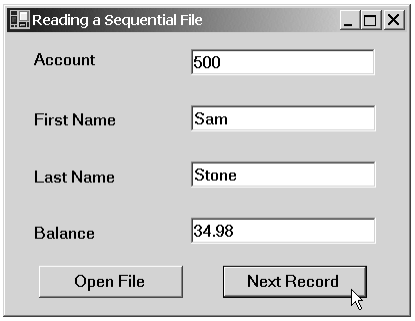
Tell user there are no more records






Reading a Sequential File
Account
100
First Name
Nancy
Last Name
Brown
Balance
-24.54
Open File
Next Record
Reading a Sequential File
Account
200
First Name
Stacey
Last Name
Dunn
Balance
314.33
Open File
Next Record


Reading a Sequential File
Account
300
First Name
Doug
Last Name
Barker
Balance
0
Open File
Next Record
Reading a Sequential File
Account
400
First Name
Dave
Last Name
Smith
Balance
258.34
Open File
Next Record



Random-Access Dosyalar

- Bilgiye doğrudan erişime izin verir
  - İstenen bilgiye diğer kayıtların içerisinden arama yapmadan doğrudan ulaşılabilir
- Bütün kayıtlar aynı boyutta olursa kolaydır
- Kayıtlar dosyanın tamamı yeniden yazılmadan güncellenebilir
- Serialize yapılmaz

Random-Access Dosyalar

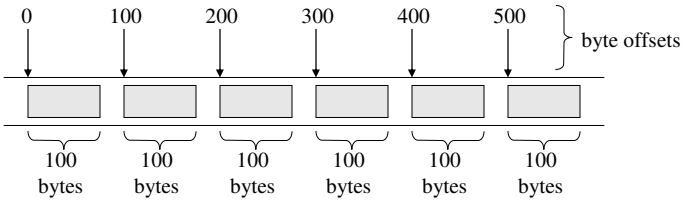


Fig. 17.13 Sabit Boyuttaki Kayıtlarla Random-access Dosya.

```
1    // Fig. 17.14: RandomAccessRecord.cs
2    // Data-record class for random-access applications.
3
4    using System;
5
6    public class RandomAccessRecord
7    {
8       // length of firstName and lastName
9       private const int CHAR_ARRAY_LENGTH = 15;
10
11      private const int SIZE_OF_CHAR = 2;
12      private const int SIZE_OF_INT32 = 4;
13      private const int SIZE_OF_DOUBLE = 8;
14
15      // length of record
16      public const int SIZE = SIZE_OF_INT32 +
17         2 * ( SIZE_OF_CHAR * CHAR_ARRAY_LENGTH ) + SIZE_OF_DOUBLE;
18
19      // record data
20      private int account;
21      private char[] firstName = new char[ CHAR_ARRAY_LENGTH ];
22      private char[] lastName = new char[ CHAR_ARRAY_LENGTH ];
23      private double balance;
24
25      // default constructor sets members to default values
26      public RandomAccessRecord() : this( 0, "", "", 0.0 )
27      {
28      }
29
```

Constant to hold record length

Private members for data storage

Set members to default value

```
30      // overloaded counstructor sets members to parameter values
31      public RandomAccessRecord( int accountValue,
32         string firstNameValue, string lastNameValue,
33         double balanceValue )
34      {
35         Account = accountValue;
36         FirstName = firstNameValue;
37         LastName = lastNameValue;
38         Balance = balanceValue;
39
40      } // end constructor
41
42      // property Account
43      public int Account
44      {
45         get
46         {
47            return account;
48         }
49
50         set
51         {
52            account = value;
53         }
54
55      } // end property Account
56
57      // property FirstName
58      public string FirstName
59      {
60         get
61         {
62            return new string( firstName );
63         }
64
```

Set records to parameter values

Account accessor methods

First name accessor methods

```
65         set
66         {
67            // determine length of string param
68            int stringSize = value.Length;
69
70            // firstName string representation
71            string firstNameString = value;
72
73            // append spaces to string parameter if too sho
74            if ( CHAR_ARRAY_LENGTH >= stringSize )
75            {
76               firstNameString = value +
77                  new string( ' ', CHAR_ARRAY_LENGTH - stringSize );
78            }
79            else
80            {
81               // remove characters from string parameter if too long
82               firstNameString =
83                  value.Substring( 0, CHAR_ARRAY_LENGTH );
84            }
85
86            // convert string parameter to char array
87            firstName = firstNameString.ToCharArray();
88
89         } // end set
90
91      } // end property FirstName
92
93      // property LastName
94      public string LastName
95      {
96         get
97         {
98            return new string( lastName );
99         }
```

Set accessor assures length is only 15 characters

Get length of string

Add spaces if length is less then 15 characters

Remove characters if length is more then 15 characters

Last name accessor methods

```
100
101         set
102         {
103            // determine length of string parameter
104            int stringSize = value.Length;
105
106            // lastName string representation
107            string lastNameString = value;
108
109            // append spaces to string parameter if too short
110            if ( CHAR_ARRAY_LENGTH >= stringSize )
111            {
112               lastNameString = value +
113                  new string( ' ', CHAR_ARRAY_LENGTH - stringSize );
114            }
115            else
116            {
117               // remove characters from string parameter if too long
118               lastNameString =
119                  value.Substring( 0, CHAR_ARRAY_LENGTH );
120            }
121
122            // convert string parameter to char array
123            lastName = lastNameString.ToCharArray();
124
125         } // end set
126
127      } // end property LastName
128
```

Set accessor assures length of 15 characters

Get name length

Add spaces if length is less then 15 characters

Remove characters if length is more then 15 characters

```
129      // property Balance
130      public double Balance
131      {
132         get
133         {
134            return balance;
135         }
136
137         set
138         {
139            balance = value;
140         }
141
142      } // end property Balance
143
144   } // end class RandomAccessRecord
```

Balance accessor methods