

Industrial Programming

Lecture 6: C# Data Manipulation

The Stream Programming Model

- File streams can be used to access stored data.
- A stream is an object that represents a generic sequence of bytes.
- Any type of data, marked `Serializable`, can be transformed into a stream. This is called *serialisation*.
- Streams can then be used to:
 - Read/Write data from/to disk.
 - Move data between machines.
- Although streams work at the byte level, programmers don't need to work with bytes.
- *Reader* and *Writer* objects are usually used to ease the use of streams.

Manual serialisation

- Writing your own serialisation function is easy, and useful in many different contexts, eg. implementing `ToString()`.
- To serialise an object of class A:
 - Serialise all value type attributes, by directly writing the data into the result buffer
 - Serialise all reference types attributes by recursively calling serialisation on them.

Naïve serialisation

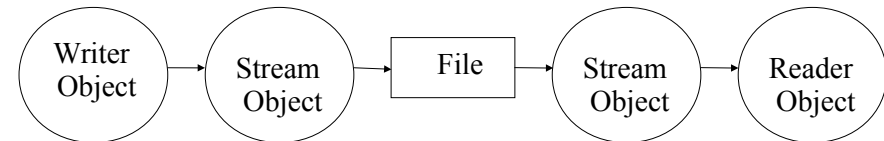
- We implement `ToString()` for our `Person/Student` example as one special case of serialisation:

```
public string ToString0() {  
    return String.Format(  
        "Name: {0} {1}\tAddress:  
{2}\nMatricNo: {3}\tDegree: {4}",  
        this.GetfName(),  
        this.GetlName(),  
        this.GetAddress(),  
        this.matricNo,  
        this.degree);  
}
```

An example of serialisation

```
public override string ToString() {  
    string base_str = base.ToString();  
    string this_str = String.Format(  
        "MatricNo: {0}\tDegree: {1}",  
        this.matricNo, this.degree);  
    return base_str+"\n"+this_str;  
}
```

Accessing Files Using Streams



C# Support for File Streams

- C# provides a number of classes in the *System.IO* namespace to access data in files including *Stream*, *TextWriter* and *TextReader*.
- The *stream* class is used to access data at the byte level.
- *TextWriter* and *TextReader* support access to readable text through using
 - *Write()* and *WriteLine()* of *TextWriter*.
 - *Read()* and *ReadLine()* of *TextReader*.

Accessing a File: Writing

- Open a file (a *FileStream* object is created).
 - A new file could be created.
 - Or an existing file opened for data to be appended to its content.
- Use a *StreamWriter* object to write text to the file.
- Close the stream.

Accessing a File: Reading

- Open a file (a `FileStream` object is created).
- Use a `StreamReader` object to read text from the file.
- Close the stream.

Example: Accessing a File

```
using System;
using System.IO;

public class FileReadWrite{
    public static void Main(){
        // Write to a file
        StreamWriter sw = new StreamWriter("test.txt");
        sw.Write("Hello World!");
        sw.Close();

        // Reading from a file
        StreamReader sr = new StreamReader("test.txt");
        Console.WriteLine(sr.ReadLine());
        sr.Close();
    }
}
```

More on File Access

- Reading more than one line from a file

```
StreamReader sr = new StreamReader("test.txt");
string inValue = "";
while((inValue = sr.ReadLine()) != null)
    Console.WriteLine(inValue);
```

- Handling file access problems with exceptions

```
try{
    StreamWriter sw = new StreamWriter("test.txt");
    sw.Write("Hello World!");
    sw.Close();
}
catch(IOException ex){
    Console.WriteLine(ex.Message); }
```

Another common pattern

```
using (StreamReader sr = new StreamReader(infile)) { // open file
    using (StreamWriter sw = new StreamWriter(outfile)) {
        string str = "";
        string str0 = "";
        while ((str = sr.ReadLine()) != null) // iterate over lines
        {
            str0 = "";
            foreach (char c in str) {
                if (Char.IsPunctuation(c)) {
                    // nothing
                } else {
                    str0 += c;
                }
            }
            sw.WriteLine(str0.ToLower());
        }
    }
}
```

C# Database Access

- C# supports database access through the ActiveX Data Objects (ADO.NET) technology.
- The ADO.NET architecture supports different databases through using *data providers* for:
 - Microsoft SQL Server (*System.Data.SqlClient*)
 - Oracle (*System.Data.OracleClient*)
 - OLE DB (*System.Data.OleDb*)
 - ODBC (*System.Data.Odbc*)

ADO.NET Data Providers

- Each provider contains a collection of classes:
 - *Connection*: setup a connection with a data source.
 - *Command*: execute an SQL statement to retrieve data from a data source.
 - *DataReader*: sequential access of data.
 - *DataAdapter*: update the database.

Creating a Connection Object

- To connect to a database hosted by SQL Server, an *SqlConnection* object need to be created.
- The *SqlConnection* constructor is used to create the object as follows:

```
SqlConnection conn = new SqlConnection(  
    "Data Source = (local);Initial Catalog=TestDatabase;  
    User ID=myId;Password=myPassword");
```

Cont. Creating a Connection Object

- The *SqlConnection* constructor takes a *string* argument which includes the following parts:
 - Data Source: identifying the server hosting the database, could be a local machine, domain name or an IP address.
 - Initial Catalog: identifying the database name.
 - User ID and Password.

Using the Connection

- We have already instantiated an *SqlConnection* object.
- We are going now to:
 - Open the connection.
 - Perform SQL operations.
 - Close the connection.

Example

```
using System;
using System.Data;
using System.Data.SqlClient;

class DatabaseConnExample{
    public static Main(){
        SqlConnection conn = new SqlConnection(
            "Data Source= (local);Initial Catalog=TestDatabase;User ID
            =myId;Password=myPassword");
        SqlDataReader dr = null;
        try{
            conn.open(); //open connection
            // pass the connection to a command object
            SqlCommand cmd = new SqlCommand("Select * from
            Students", conn);
            //get query results
            dr = cmd.ExecuteReader();
        }
```

Example (cont'd)

```
        while(dr.Read()){Console.WriteLine(dr[0]);}
    }
    finally{
        if(dr != null){dr.Close();}
        if(conn != null){conn.Close();} // Close
        // connection
    }
}
```

Example Explained

- The connection is opened using the *open()* function of the *SqlConnection* instance (*conn*).
- The *SqlCommand* object uses *conn* to perform a query on the *students* table.
- The result set is returned in a *SqlDataReader*.
- The while loop reads the value of the first field in each row of the result set.
- The *close()* function of *conn* is called to ensure that the connection is closed.

LINQ

- The Language Integrated Query (LINQ) components makes access to databases easier.
- It provides uniform access to a range of databases and data formats (e.g. XML).
- The commands are similar to SQL commands.
- When querying a database, tables can be treated like classes and columns like members.
- It makes use of advanced language features such as anonymous types, implicitly typed variables, and lambda expressions.
- From C# 3.0 onwards this is the preferred way of working with large sets of data.