SIT232 - OBJECT ORIENTED DEVELOPMENT

Session 11. Files and Generics

Outline

- Session 11. Files and Generics
  - Objectives
  - Files
  - File Types
  - Files Access Modes
  - Sequential Access Text Files
  - Random Access Binary Files
  - Serialisation
  - Generics

SESSION 11. FILES AND GENERICS

Objectives

- At the end of this session you should:
- How *file input and output can be used* to add persistence to the data in an application;
- The *difference between text files and binary files*;
- The *difference between sequential access and random access* and how they can/cannot be applied to text files and binary files;
  - Understand the ***concept of serialisation***;
  - Understand the ***concept of generics***;
  - Be able to implement ***file input/output, serialisation, generic methods, and generic classes*** in your programs.

Files

- Files are used regularly in programming for storing data beyond the life of a single program
  - Main memory is volatile
  - Secondary storage is non-volatile
    - Floppy disc
    - Hard disc
    - Optical disc
    - USB storage
    - etc.

File Types

- There are two options:
  - **Text files**
    - Usually *human readable/modifiable*
    - *Not very efficient*
      - Student ID “90123456” takes 4 bytes in RAM vs 8-16 bytes on disk
      - Costly to convert between numeric/textual formats
    - Usually lines of *variable length*
      - Difficult to do random access
  - **Binary files**
    - Usually *not human readable/modifiable*
    - *More efficient*
      - Data is stored the same as it is in memory
    - Usually employ a *fixed sized structure*
      - Enables random access

File Access Modes

- There are two options:
  - Sequential access
    - Reads/writes a file from start to finish
    - Required for *text files*
      - Seeking to the start/end of a file is common however
    - *Optional for binary files* but may be required
  - Random access
    - Can jump to different locations in a file to read/write
    - Usually requires a *fixed record size*
      - *Record is represents an object stored* on disk (roughly)
      - *Difficult when considering string types*

Sequential Access Text Files

- Opening a Text file:

  FileStream *variable_name* **= new FileStream(***path***,** *mode***,** *access***);**

  - Or

  FileStream *variable_name* **= File.Open(***path***,** *mode***,** *access***);**

- Path:
  - Specifies filename
    - Verbatim string literal, e.g., `@"C:\myFile.txt"`
    - Variable storing a file name, e.g., `openFileDlg.FileName`

Sequential Access Text Files

- Opening a file:

  FileStream *variable_name* **= new FileStream(***path***,** *mode***,** *access***);**

  - Or

  FileStream *variable_name* **= File.Open(***path***,** *mode***,** *access***);**

- Mode:
  - **File.Append** – (for writing) append to end of file
  - **File.Create** – truncate existing/create new file
  - **File.CreateNew** – create new file or throw exception
  - **File.Open** – open existing file or throw exception
  - **File.OpenOrCreate** – open existing or create new file
  - **File.Truncate** – truncate existing file

Sequential Access Text Files

- Opening a file:

  FileStream *variable_name* **= new FileStream(***path***,** *mode***,** *access***);**

  - Or

    FileStream *variable_name* **= File.Open(***path***,** *mode***,** *access***);**

- Access:
  - **`FileAccess.Read`** – read only
  - **`FileAccess.Write`** – write only
  - **`FileAccess.ReadWrite`** – read and write

Sequential Access Text Files

- Then encapsulate the FileStream into something more useful:

  StreamReader *variable_name* **= new StreamReader(***filestream_variable***);**

  StreamWriter *variable_name* **= new StreamWriter(***filestream_variable***);**

- Input and output is similar to Console
  - **StreamReader** has **Read()** and **ReadLine()**
  - **StreamWriter** has **Write()** and **WriteLine()**

- Lastly, don't forget to **Close()!**
  - Flushes any buffer and safely closes files
  - Prevents data loss

Sequential Access Text Files

```
FileStream fs = File.Open(@"test.txt", FileMode.Create, FileAccess.Write);
StreamWriter sw = new StreamWriter(fs);

Console.Write("Text or 'END' to terminate: ");
string text = Console.ReadLine();
while (text.ToUpper() != "END")
{
    sw.WriteLine(text);
    Console.Write("Text or 'END' to terminate: ");
    text = Console.ReadLine();
}

sw.Close();
```

Creating and writing to a text file

```
FileStream fs = File.Open(@"test.txt", FileMode.Open, FileAccess.Read);
StreamReader sr = new StreamReader(fs);

Console.WriteLine("File contents:");
while ((text = sr.ReadLine()) != null)
    Console.WriteLine(text);

sr.Close();
```

Opening and reading from a text file

Random Access Binary Files

- There is a need to have a fixed length record

...	Record A	Record B	Record C	Record D	Record E	Record F	Record G	...

- *Allows us to calculate the location of a record*
  - $A = (n-1) \times L$

- But which record to seek to? Options...
  - *Indexing* – small subset of data and record number and read into memory
  - *Hashing* – algorithm applied to data to calculate record number

Random Access Binary Files

```
static void Main(string[] args)
{
    FileStream testFile = new FileStream(@"students.dat", FileMode.Create, FileAccess.ReadWrite);

    // Student names taken from Top 100 singles on Australian charts for 2000
    // Source: http://www.aria.com.au/pages/aria-charts-end-of-year-charts-top-100-singles-2000.htm
    Student.ToStream(testFile, new Student(1001, "Anastacia",       1991,  1,  1));
    Student.ToStream(testFile, new Student(1002, "Wheatus",         1992,  2,  2));
    Student.ToStream(testFile, new Student(1003, "Bomfunk MCs",     1993,  3,  3));
    Student.ToStream(testFile, new Student(1004, "Madonna",         1994,  4,  4));
    Student.ToStream(testFile, new Student(1005, "Destiny's Child", 1995,  5,  5));
    Student.ToStream(testFile, new Student(1006, "Bardot",          1996,  6,  6));
    Student.ToStream(testFile, new Student(1007, "*N Sync",         1997,  7,  7));
    Student.ToStream(testFile, new Student(1008, "Madison Avenue",  1998,  8,  8));
    Student.ToStream(testFile, new Student(1009, "Spiller",         1999,  9,  9));
    Student.ToStream(testFile, new Student(1010, "Mary Mary",       2000, 10, 10));
    Student.ToStream(testFile, new Student(1011, "Mandy Moore",     2001, 11, 11));
    Student.ToStream(testFile, new Student(1012, "Chris Franklin",  2002, 12, 12));

    Random rnd = new Random();
    Student s;
    Console.Write("Press ENTER or type anything and ENTER to quit: ");
    string input = Console.ReadLine();
    while (input == "")
    {
        int which = rnd.Next(0, 12); // random number from 0-11
        testFile.Seek(which * Student.RecordLength, SeekOrigin.Begin);
        Student.FromStream(testFile, out s);
        Console.WriteLine("{0,2}: {1}", which, s);
        Console.Write("Press ENTER or type anything and ENTER to quit: ");
        input = Console.ReadLine();
    }

    testFile.Close();
}
```

Reading and writing to a file using random access

Random Access Binary Files

```
public static void ToStream(Stream stream, Student student)
{
    BinaryWriter bw = new BinaryWriter(stream, Encoding.Unicode);

    bw.Write(student._ID);              // Write out _ID

    // Write out the length of _Name
    int nameLength = student._Name.Length;
    if (nameLength > MAX_NAME_LENGTH)
        nameLength = MAX_NAME_LENGTH;
    bw.Write(nameLength);

    // Prepare to write out _Name
    char[] name = student._Name.ToCharArray();
    int length = Math.Min(student._Name.Length, MAX_NAME_LENGTH);
    if(name.Length != MAX_NAME_LENGTH)
    {
        char [] tmp = new char[MAX_NAME_LENGTH];
        int i = 0;
        while(i < length)
        {
            tmp[i] = name[i];
            i++;
        }
        while(i < MAX_NAME_LENGTH)
        {
            tmp[i++] = '\0';
        }
        name = tmp;
    }
    bw.Write(name, 0, name.Length);     // Write out _Name

    bw.Write(student._BirthYear);       // Write out _BirthYear
    bw.Write(student._BirthMonth);      // Write out _BirthMonth
    bw.Write(student._BirthDay);        // Write out _BirthDay
}
```

Writing data from an object to a binary file

Serialisation

- *Serialisation* is the conversion of an object into a sequence of bytes to be written to a file
- *Deserialisation* is the conversion of a sequence of bytes back into an object

- Three types:
  - **Binary serialisation**
  - **SOAP serialisation**
  - **XML serialisation**
    - Only considers public elements

Serialisation

- **Binary serialisation:**

```
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
…
FileStream fs = File.Open(@"filename", FileMode.Create, FileAccess Write);
BinaryFormatter bf = new BinaryFormatter();
bf.Serialize(fs, object_name);
fs.Close();

…

FileStream fs = File.Open(@" filename ", FileMode.Open, FileAccess.Read);
BinaryFormatter bf = new BinaryFormatter();
object_name = (class_name)bf.Deserialize(fs);
fs.Close();
```

Serialisation

- **SOAP serialisation:**

```
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Soap;
…
FileStream fs = File.Open(@" filename ", FileMode.Create, FileAccess Write);
SoapFormatter sf = new SoapFormatter();
sf.Serialize(fs, object_name);
fs.Close();

…

FileStream fs = File.Open(@" filename ", FileMode.Open, FileAccess.Read);
SoapFormatter sf = new SoapFormatter();
object_name = (class_name)sf.Deserialize(fs);
fs.Close();
```

Serialisation

- **XML serialisation:**

```
using System.Runtime.Serialization;
using System.Xml.Serialization
…
FileStream fs = File.Open(@" filename ", FileMode.Create, FileAccess Write);
XmlSerializer xs = new XmlSerializer(typeof(class_name));
xs.Serialize(fs, object_name);
fs.Close();

…

FileStream fs = File.Open(@" filename ", FileMode.Open , FileAccess.Read);
XmlSerializer xs = new XmlSerializer(typeof(class_name));
object_name = (class_name)xs.Deserialize(fs);
fs.Close();
```

Generics

- The problem: how to write functionality that applies equally to different types?

- The bad solution:

```
object[] objArray = new object[3];
objArray[0] = 3;
objArray[1] = '3';
objArray[2] = "three";
```

- There is no checking of data types!

Generics

- The problem: how to write functionality that applies equally to different types?

- The good solution:

```
TYPE Minimum<TYPE>(TYPE first, TYPE second)
   where TYPE : IComparable<TYPE>
{
   TYPE result = first;
   if (second.CompareTo(first) < 0)
      result = second;
   return result;
}
```

Generics

- The problem: how to write functionality that applies equally to different types?

- Can also apply to classes:

```
class class_name <type_name[, ...]>
    [ where type_name : class_or_interface
    [...]]
{
    ...
}
```

Generics

- The problem: how to write functionality that applies equally to different types?

- We've seen it:

```
List <int>  listOfIntegers = new List < int > ();
```

- Can also embed generic declarations:

```
List<  List < int >  > listOfIntegerLists = new List <  List < int >  > ();
```

Summary

- Session 11. Files and Generics
  - Objectives
  - File Types
  - Files Access Modes
  - Sequential Access Text Files
  - Random Access Binary Files
  - Serialisation
  - Generics

Testing

Which of the following is NOT a type of file?

a) External file
b) Text file
c) Binary file
d) All of the above
e) None of the above

Testing

Which of the following Microsoft.Net classes is used to read from a text file?

a) FileReader
b) StreamReader
c) TextReader
d) All of the above
e) None of the above

Testing

Which of the following is NOT an approach to serialisation?

a) Binary
b) HTML
c) SOAP
d) XML
e) None of the above

Testing

Which of the following is NOT a method of accessing a file?

a) Sequential access
b) Remote access
c) Random access
d) All of the above
e) None of the above