MALMÖ HÖGSKOLA

Programmering med C# II

Filer och strömmar

Agenda:

- Filtyper
- Strömmar
- Läsa och skriva textfiler
- Läsa och skriva till XML filer

Farid Naisan, University Lecturer, Malmö University, farid.naisan@mah.se

Spara data för att återhämta igen

- När vi kör ett program, jobbar vi med data som är i datorns arbetsminne (RAM). Data försvinner så snart programmet stängs.
- För att kunna spara data mer permanent, behöva de sparas på sekundära minnen såsom hårddiskar, CD, DVD, etc.
- Data kan sparas i olika former, textfiler, xml-filer, databaser, binära filer, etc.
- Varje programmeringsspråk har stöd för att hantera lagring data.

Vad är det som sparas?

- Vi jobbar med objekt och ett objekt består huvudsakligen av data (instansvariabler) som sparar ”status” (state) av ett objekt, och operationer (metoder) som beskriver ett objekts beteende (behavior).
  - Vad är det då som skall sparas, instansvariabler, metoder, eller båda?
- Det är självklart ett objekts status, dvs. instans-variabler som behöver sparas.

Hur sparas data?

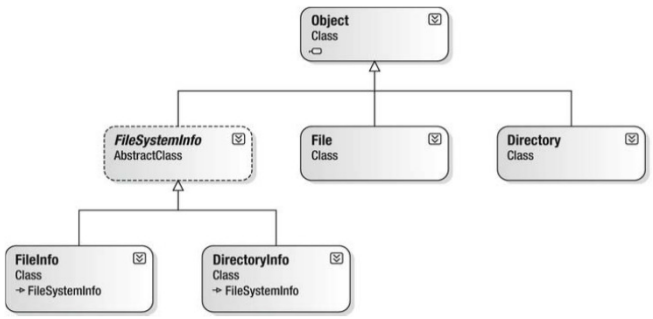
- Spara data: värden av instansvariabler → fil
  - **Öppna** en befintlig fil eller skapa en ny
  - **Spara** data
    - hämta ett objekts status sparade i dess instans variabler, spara till fil.
  - **Stäng** filen (så fort skrivningen är klar)
- Läsa data: värden i filen →instansvariabler
  - **Öppna** en befintlig fil
  - **Läs** data
    - Läs värden och spara till objektets instansvariabler, eller i temporära variabler.
  - **Stäng** filen (så fort inläsningen är klar)

System.IO

- Namnrymden IO innehåller många kärntyper för hantering av mappar och filer.
- IO innehåller en del abstrakta och en del icke-abstrakta klasser och typer.
- Exempel av icke-abstrakta klasser är
  - BinaryWriter, BinaryReader
  - StreamWriter, StreamReader
  - StringWriter, StringReader
  - Och många flera.

Mappar och filer

- För manipulering av filer och datorns mappstruktur, .NET tillhandhåller flera klasser.
- Dessa kan användas för att skapa, ta bort, kopiera eller flytta filer och mappar.



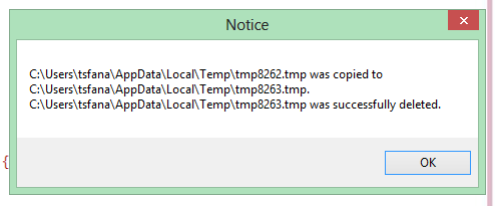
FileInfo och DirectoryInfo

- Klassen **FileInfo** kan du använda för att erhålla detaljer om filer på datorns hårddisk. Det kan vara information såsom tiden filen är skapade, filens storlek och attribut, etc.
- Klassen hjälper också dig med kopiering, flyttning och borttagning av filer.
- På samma sätt kan **DirectoryInfo** typen användas för att manipulera mappar. Denna typ hjälper dig att skapa, flytta, ta bort och enumerera över mappar och undermappar.

FileInfo Exempel

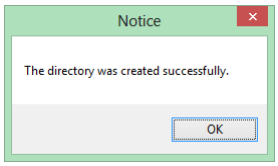
```
public  string TestFileInfo()
{
    string message = null;
    //Create a tempory file in the current user's temp
    string path = Path.GetTempFileName();
    FileInfo fi1 = new FileInfo(path);

    //Create a file to write to.
    using (StreamWriter sw = fi1.CreateText())
    {
        sw.WriteLine("Hello and welcome");
    }

    try
    {
        string path2 = Path.GetTempFileName();
        FileInfo fi2 = new FileInfo(path2);

        //Ensure that the target does not exist.
        fi2.Delete();

        //Copy the file.
        fi1.CopyTo(path2);
         message = String.Format("{0} was copied to {
        //Delete the newly created file.
        fi2.Delete();
        message += Environment.NewLine;
        message += String.Format("{0} was successfully deleted.", path2);
    }
    catch (Exception e)
    {
        message =  string.Format("The process failed: {0}", e.ToString());
    }
    return message;
}
```

```
private void TestFileInfo()
{
    FileIOTest obj = new FileIOTest();
    MessageBox.Show(obj.TestFileInfo(), "Notice");
}
```



DirectoryInfo exempel

```
public string TestDirectoryInfo()
{
    string message = null;

    // Specify the directories you want to manipulate.
    DirectoryInfo di = new DirectoryInfo(@"c:\MyDir");
    try
    {
        // Determine whether the directory exists.
        if (di.Exists)
        {
            // Indicate that the directory already exists.
            return"That path exists already.";
        }

        // Try to create the directory.
        di.Create();
        message = "The directory was created successfully.";

    }
    catch (Exception e)
    {
        return String.Format("The process failed: {0}", e.ToString());
    }
    finally { }
    return message;
}
```



Skriv kod för att spara/läsa objekt

- Som beskrivs tidigare, är det är tre huvudsteg när man skriver till/läser från en fil:
  - öppna filen
  - skriv/läs
  - stäng filen
- Filer används på olika sätt och för olika ändamål, för att skriva, läsa eller söka information
- Filer används för att nå data ofta sekventiellt men också godtyckligt.

Textfiler/Binärfiler

- För att utföra sådana operationer finns det olika Framework klasser att utnyttja.
  - För textfiler (filer som består av endast alpha-numeriska tecken)
    - StreamWriter, StreamReader
    - XMLWriter, XMLReader
  - För Binärafiler (binärt data) - serialisering
    - Stream
    - FileStream

Filer och strömmar

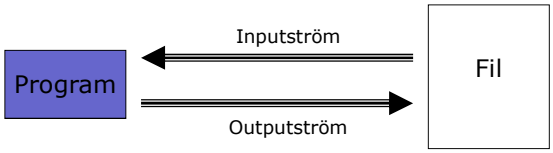
- En fil och en ström är båda en ordnad samling av bytes i en ihållande lagring.
- Med fil-och ström IO (input/output) menas att data förflyttas från eller till en lagringsmedia.
- Att spara information till en fil eller läsa information från en fil är en specialiserad användning av strömmar.
- En ström är ett sätt att kommunicera med andra enheter. Du kan skicka information från ditt program till en annan enhet eller hämta information från en enhet till ditt program.
- Den andra enheten kan t.ex. vara en hårddisk, en skrivare eller en annan dator via ett nätverk.
- En ström är alltid enkelriktad.

Skillnaderna

- Ström är representation av dataflöde from en sida till annan sida, t ex från en disk to minnet och från minnet till en disk.
- Användning av ström är ett sätt att skicka data till eller hämta data ifrån det, utan att behöva veta vilken typ av media finns på den andra sidan.
- Buffert används för att lagra ström data temporärt.
- En fil är en samling av data sparade på disk med ett specifikt namn och specifik sökväg.
  - Fil använder ström för att spara eller läsa data.

Typer av ström

- Det är huvudsakligen två typer av strömmar:
  - Inputströmmar som man läser information ifrån,
  - Outputströmmar som man skriver information till.



- I .NET har strömmar också en annan gruppering :
  - FileSteam
  - Memory Stream

En ström liknar en rör

- En ström kan liknas vid ett rör som öppnas mellan t.ex. ett datorprogram och en fil.
- Genom röret kan man skicka information men bara med ett tecken (Byte eller ibland Bit) i taget.
- För att skicka något måste man först skapa röret (strömmen) så att det finns en koppling mellan programmet och t.ex. filen.
- Därefter stoppas informationen in i röret en Byte (eller Bit) i taget.

Bytes måste kunna går igenom röret

- För att informationen ska komma fram måste man ofta spola i röret (flush) eller stänga av uppkopplingen och då utförs en automatisk spolning.
- För att detta ska fungera måste det givetvis vara möjligt att dela upp informationen i många småbitar som kan skickas en i taget genom det smala röret.
- Detta uppnås bl. a. genom serialisering (en introduktion ges i slutet av denna presentation).

Serialisering

- Serialisering är en process där ett objektet och dess tillstånd (information) sparas (eller överföras) till en ström (file stream eller memory stream).
- Objektserialisering (Object Serialization) är med andra ord att skriva till eller läsa från en *ström*.
- Objektserialisering är att dela upp ett objekt och dess tillstånd i mindre delar och sedan överföra över nätverket eller spara på disk i ***serier***.
- Objektserialisering i .NET gör det möjligt att spara ett objekt med ett få antal kodrader och därmed spara både tid och plats (diskutrymme).

FileStream

- Namnrymdden som behöver importeras:

```
using System.IO;                    //FileStream, Reader/Writer classes
using System.Runtime.Serialization;                       //IFormatter
using System.Runtime.Serialization.Formatters.Binary; //BinaryFormatter
```

- IFormatter är en interface som tillhandahåller funktionalitet för formatering av serialiserade objekt.
- Klassen **`BinaryFormatter`** implementerar **`IFormatter`** och serialiserar/deserialiserar ett objekt, eller en graf av kopplade objekt, med ett binärt format.

Läsa och skriva en textfil

- Textfiler är användbara i manga sammanhang. Man kan t ex exportera och importera data mellan två olika applikationer med hjälp av textfiler.
- Textfiler tar mer diskutrymme, kan vara långsammare men har fördelen att de kan läsas med hjälp av vanliga
- Flera typer finns i .NET Ramverkets klassbibliotek som kan användas för textfiler. Några av dessa:
  - TextWriter, TextReader
  - StreamWriter, StreamReader
  - StringWriter, StringReader

Writer/Reader klasser

- **TextWriter/TextReader** är en abstract klass. Klassen tillhandahåller en abstraktion för läsning och skrivning av datakällor för sina subklasser (konkretklasser).
- **StreamWriter/StreamReader** är en konkreta klass av ovan.
  - Dessa två använder en läsbar/skrivbar ström som datakälla. De behöver en instans av klassen **Encoding** för översättning av tecken och Bytes, men de använder en default encoding om ingen instans specificeras.
- **StringWriter/Reader** är också en konkret implementation av TextWriter/TextReader .
  - De använder **StringBuilder/string** som datakälla..

StreamWriter och StreamReader

- Klasserna **StreamWriter** och **StreamReader** är behändiga verktyg för skrivning och läsning av tecken-baserade data (t ex. Strängar).
- Bägge jobbar som default med Unicode tecken som kan ändras till annan Encoding genom korrekt konfigurering av den System.Text.Encoding objektreferensen.
- Skrivning och läsning av textfiler involverar inte serialisering..

StreamWriter/Reader medlemmar

- Klassen StreamWriter har manga användbara medlemmar
  - Write()
  - WriteLine()
  - NewLine
  - Close()
  - Flush()
- Några medlemmar av StreamReader-klassen:
  - Read()
  - ReadLine()
  - ReadBlock()
  - Peek()
  - ReadToEnd()

StreamWriter/Reader - användning

- Att skriva till en textfil:

```
public void TestWrite(string fileName)
{
    StreamWriter writer = new StreamWriter(fileName);
    writer.WriteLine("Some text");
    writer.Close();
}
```

  - Metoderna Write och WriteLine fungerar på samma sätt som System.Console.

- Att läsa från en textfil:

```
public void TestRead(string fileName)
{
    StreamReader reader = new StreamReader(fileName);
    string strLine = reader.ReadLine();
    reader.Close();
}
```

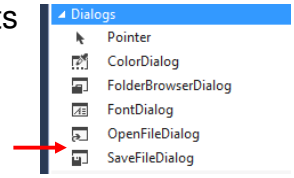
  - Metoderna Read och ReadLine fungerar på samma sätt som System.Console. .

Spara data i xml format

- Ett objekts data kan sparas till XML filer.
- C# har bra stöd för att spara till och läsa ifrån XML filer.
- De objekt som kan användas är:
  - XMLWriter, XMLReader
  - XMLSerializer
- Namnrymden:
  - System.XML
  - System.XML.Serialization

Fildialoger

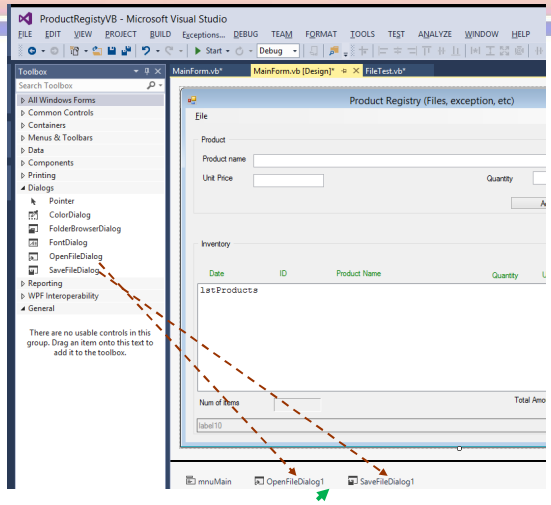
- För att ge användaren möjlighet att på ett grafiskt sätt öppna eller spara filer kan man använda sig av .NET ramverkets OpenFileDialog resp SaveFileDialog klasserna.
- Man kan skapa objektet med programmeringskod (precis som allt annat i språket) eller ritar kontrollerna från VSs verktygslåda.
- Dessa är behändiga, ger massvis med val och alternativ och är dessutom lätta att använda.



Fildialoger

- Men tänk på att dialogernas huvuduppgift är att leverera användarens val av ett filnamn, dvs en textsträng gjord av enhet, sökväg och filnamn.
- Dialogerna kan naturligtvis inte hämta/lagra informationen i den angivna filen. Detta är programmerarens uppgift. Här följer exempel som är en del av en det kodexempel som finns på arbetsmodulen.

Fildialoger



Obs· Vissa kontroller har ingen visuell funktion och hamnar här under formuläret!

SaveFileDialog

- **SaveToFile** är en separat funktion som sparar data till fil. Se kodexemplet.

```
private void mnuFileSaveAs_Click(object sender, System.EventArgs e)
{
    //Show save-dialogbox
    if (saveFileDialog1.ShowDialog() == DialogResult.OK)
    {
        fileName = saveFileDialog1.FileName;  //important
        SaveToFile();
    }
}

private void mnuFileSave_Click(object sender, System.EventArgs e)
{
    if (fileName == String.Empty)  //no file name ==> Save As
        mnuFileSaveAs_Click(sender, e);
    else
        SaveToFile();

}
```

OpenFileDialog

- ReadFile är en separat funktion som läser data från fil. Se kodexemplet.

```
private void mnuFileOpen_Click(object sender, System.EventArgs e)
{
    AskUserIfSaveDataToFile(sender, e);  //Save current data?
    //Show open-dialogbox
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        fileName = openFileDialog1.FileName;
        string msg = ReadFile();

        if (msg != String.Empty)
            MessageBox.Show(msg);
        else
            UpdateListBox();
    }
}
```


Sammanfattning

- Importera System.IO (`using System.IO;`) i filer där du använder strömmar och filfunktioner.
- Använd klassen Stream eller FileStream för att skapa strömmen. Med FileStream blir det lättare.
- Använd StreamWriter och StreamReader för att läsa/skriva textfiler.
- Använd FileInfo och DirectoryInfo klasser när du vill jobba med filer och mappar.
- **OpenFileDialog** och **SaveFileDialog** är mycket bra verktyg för att låta användaren välja fil och sökväg.

Och kom alltid ihåg:

- Läs en fil på samma sätt och i exakt samma data ordning du sparat den. Använda relaterad typpar, t ex StreamWriter/StreamReader.
  - Det går inte att blanda olika sätt på samma fil.

Nyttiga länkar

System.IO: http://msdn.microsoft.com/en-us/library/system.io.aspx

XMLTextReader:
http://msdn.microsoft.com/en-us/library/system.xml.xmltextreader.aspx

XMLTextWriter:
http://msdn.microsoft.com/en-us/library/system.xml.xmltextwriter.aspx

System.Runtime.Serialization:

http://msdn.microsoft.com/en-us/library/system.runtime.serialization.aspx

System.XML:
http://msdn.microsoft.com/en-us/library/y3y47afh.aspx