1. Poglavlje: **Rad s podacima**

Često imamo potrebu zapamtiti neke podatke, pa ih u sljedećim sessijama programa vratiti. Za to postoji više načina. Ovdje ćemo nabrojati osnovne principe.

1.1 Windows Registry

Za spremanje (uglavnom manje količine podataka) može se koristiti Windows Registry. WR je stablasta struktura. Imamo Nekoliko osnovnih grana, a svaka grana može imati proizvoljan broj podgrana i proizvoljan broj listova (par ključ,ime).
Klase u C# sa rad s registrijem su u namespaceu Microsoft.Win32. Osnovne klase su Registry (predstavlja korijen stabla) i RegistryKey (predstavlja pojedinu granu/list).
Registry možemo ručno gledati/editirati programom regedit (za koji nam trebaju administratorske ovlasti). U korijenu ne možemo dodavati grane, nego imamo nekoliko osnovnih grana (CurrentUser, CurrentConfig, LocalMachine, … za pristup nekima su nam potrebne administratorske ovlasti). Običaj je otvoriti nekog od njih, te u grani Software otvoriti granu s imenom trvtke, te u njoj otvoriti granu s imenom programa, te u nju spremati podatke potrebne za naš program.

```
using System;
using System.Security.Permissions;
using Microsoft.Win32;

[assembly: RegistryPermissionAttribute(SecurityAction.RequestMinimum,
    ViewAndModify = "HKEY_CURRENT_USER")]

class RegKey
{
    static void Main()
    {
        RegistryKey glavni =
            Registry.CurrentUser.CreateSubKey(@"Software\Vinko\Proba");

        using (RegistryKey
            prva = glavni.CreateSubKey("prvaGrana"),
            druga = glavni.CreateSubKey("drugaGrana"))
        {
            druga.SetValue("Language1", "moj");
            druga.SetValue("Level", "Intermediate");
            druga.SetValue("ID", 123);
        }

        Console.WriteLine("There are {0} subkeys under {1}.",
            glavni.SubKeyCount.ToString(), glavni.Name);
        foreach (string subKeyName in glavni.GetSubKeyNames())
        {
            using (RegistryKey
                tempKey = glavni.OpenSubKey(subKeyName))
            {
                Console.WriteLine("\nThere are {0} values for {1}.",
                    tempKey.ValueCount.ToString(), tempKey.Name);
                foreach (string valueName in tempKey.GetValueNames())
                {
                    Console.WriteLine("{0,-8}: {1}", valueName,
                        tempKey.GetValue(valueName).ToString());
                }
            }
        }

        using (RegistryKey
            testSettings = glavni.OpenSubKey("drugaGrana", true))
        {
            testSettings.DeleteValue("id");
```

```
            Console.WriteLine((string)testSettings.GetValue(
                "id", "ID not found."));
        }

        Console.Write("\nDelete newly created registry key? (Y/N) ");
        if (Char.ToUpper(Convert.ToChar(Console.Read())) == 'Y')
        {
            Registry.CurrentUser.DeleteSubKeyTree(@"Software\Vinko\Proba");
            Console.WriteLine("\nRegistry key {0} deleted.",
                glavni.Name);
        }
        else
        {
            Console.WriteLine("\nRegistry key {0} closed.",
                glavni.ToString());
            glavni.Close();
        }
    }
}
```

Zadatak 1

Napravite program u kojem imamo jednu tekst kontrolu u koju možemo unijeti ime. Neka program zapamti (u registri, prilikom gašenja programa) ime i poziciju prozora na ekranu, te neka se prilikom svakog sljedećeg starta pojavi na istoj poziciji i s istim imenom.

1.2 Rad s folderima/datotekama

Datoteke se koriste uglavnom za spremanje veće količine podataka. Osnovne klase za rad s datotekama/direktorijima se nalaze u namespaceu System.IO. Klase za rad s datotekema su:
File – sadrži statičke naredbe za rad s datotekama
Directory – sadrži statičke naredbe za rad s folderima
FileSystemInfo – bazna klasa za FileInfo i DirectoryInfo – one sadrže uglavnom nestatičke naredbe za rad s konkretnom datotekom/folderom i s njima možemo preciznije upravljati.
Npr. FileSystemInfo ima svojstva Attributes, CreationTime, Exists, Extension, FullName, LastAccessTime, LastWriteTime, Name.
DirectoryInfo ima metode Create, CreateSubdirectory, Delete, GetDirectories, GetFiles, MoveTo, Parent, Root.
Npr. (ovaj dio koda neće raditi ako se nema prava pristupa/pisanja u određeni folder)

```
            DirectoryInfo d1 = new DirectoryInfo(".");
            DirectoryInfo d2 = new DirectoryInfo(@"c:\vinko\proba");
            d2.Create();
            DirectoryInfo d3 = new DirectoryInfo(@"c:\windows");

            FileInfo[] images = d3.GetFiles("*.jpg", SearchOption.AllDirectories);
```

Za rad sa folderima možemo koristiti i klasu Directory, koja ima statičke funkcije (pa u svakoj naredbi trebamo pisati puno putanju) i funkcije uglavnom primaju/vraćaju stringove/nizove stringova, a ne specijalizirane klase/nizove klasa. Za pobrojati npr. sve drivere možemo koristiti Directory.GetDrives, koja vraća niz stringova. Ali za to možemo koristiti i klasu DriveInfo, koja nam opet može dati potpunije informacije.
FileInfo klasa osim što daje mogućnost pregleda podataka o datoteci, daje i mogućnost da datoteku obrišemo/premjestimo/kreiramo i otvorimo za različite načine čitenje/pisanja.

```
            // Make a new file on the C drive.
            FileInfo f = new FileInfo(@"C:\Test.dat");
            FileStream fs = f.Create();
            // Use the FileStream object...

            // Close down file stream.
            fs.Close();
```

ili

```
            using(FileStream fs2 = f2.Open(FileMode.OpenOrCreate,
FileAccess.ReadWrite, FileShare.None))
```

OpenText, CreateText ili AppendText funkcije vraćju StreamReader ili StreamWriter.

```
    static void Main(string[] args)
    {
        // Get a StreamReader object.
        FileInfo f5 = new FileInfo(@"C:\boot.ini");
        using (StreamReader sreader = f5.OpenText())
        {
            // Use the StreamReader object...
        }
    }
```

Ili direktno bez filea:

```
        string fileLoc = Path.Combine(Environment.GetFolderPath(
           Environment.SpecialFolder.MyDocuments), "MyDoc.txt");

        // Create the file if it does not exist.
        if (!File.Exists(fileLoc))
        {
            using (StreamWriter swNew = new StreamWriter(fileLoc))
            {
                swNew.WriteLine("Sample text");
            }
        }

        // Display the contents in a TextBox.
        using (StreamReader sr = new StreamReader(fileLoc))
        {
            Console.WriteLine(sr.ReadToEnd());
        }
```

File struktura nam daje statičke metode kojima možemo napraviti gotovo jednake stvari, a imamo i neke dodatne naredbe: (Read/Write)All(Bytes/Lines/Text).

```
Console.WriteLine("***** Simple IO with the File Type *****\n");
string[] myTasks = {
"Fix bathroom sink", "Call Dave",
"Call Mom and Dad", "Play Xbox 360"};
// Write out all data to file on C drive.
File.WriteAllLines(@"C:\tasks.txt", myTasks);
// Read it all back and print out.
foreach (string task in File.ReadAllLines(@"C:\tasks.txt"))
{
Console.WriteLine("TODO: {0}", task);
}
Console.ReadLine();
```

Svi Streamovi su izvedeni iz abstraktne klase Stream. Dolje imamo primjer rada s FileStreamovima, ali Streamovi su općenitiji od Datoteka, općenito predstavljaju skupinu podataka (npr. možemo koristiti i MemoryStream i NetworkStream).

```
Console.WriteLine("***** Fun with FileStreams *****\n");
// Obtain a FileStream object.
using(FileStream fStream = File.Open(@"C:\myMessage.dat",
FileMode.Create))
{
// Encode a string as an array of bytes.
string msg = "Hello!";
byte[] msgAsByteArray = Encoding.Default.GetBytes(msg);
// Write byte[] to file.
fStream.Write(msgAsByteArray, 0, msgAsByteArray.Length);
// Reset internal position of stream.
fStream.Position = 0;
// Read the types from file and display to console.
Console.Write("Your message as an array of bytes: ");
byte[] bytesFromFile = new byte[msgAsByteArray.Length];
for (int i = 0; i < msgAsByteArray.Length; i++)
{
bytesFromFile[i] = (byte)fStream.ReadByte();
Console.Write(bytesFromFile[i]);
}
// Display decoded messages.
Console.Write("\nDecoded Message: ");
Console.WriteLine(Encoding.Default.GetString(bytesFromFile));
}
Console.ReadLine();
```

Za rad s tekstualnim datotekama pogledajte

```
using (StreamWriter writer = File.CreateText("reminders.txt"))
{
    writer.WriteLine("Don't forget Mother's Day this year...");
    writer.WriteLine("Don't forget Father's Day this year...");
    writer.WriteLine("Don't forget these numbers:");
    for (int i = 0; i < 10; i++)
        writer.Write(i + " ");
    // Insert a new line.
    writer.Write(writer.NewLine);
}
```

Ili

```
using (StreamReader sr = File.OpenText("reminders.txt"))
{
    string input = null;
    while ((input = sr.ReadLine()) != null)
    {
        Console.WriteLine(input);
    }
}
```

Još neke zgodne klasu su String(Reader/Writer) – za pisanje u string, Binary(Reader/Writer) – za binarno pisanje u datoteke.

1.3 Obavijesti o promjenama datoteka

.net nam pruža jednostavan način provjeravanja promjena datoteka, slično kao i ostalih događaja (klikova, …) u sistemu. Za to nam služi FileSystemWatcher klasa:

```
static void Main(string[] args)
{
FileSystemWatcher watcher = new FileSystemWatcher();
try
{
watcher.Path = @"C:\MyFolder";
}
catch(ArgumentException ex)
{
Console.WriteLine(ex.Message);
return;
}
// Set up the things to be on the lookout for.
watcher.NotifyFilter = NotifyFilters.LastAccess
| NotifyFilters.LastWrite
| NotifyFilters.FileName
| NotifyFilters.DirectoryName;
// Only watch text files.
watcher.Filter = "*.txt";
// Add event handlers.
watcher.Changed += new FileSystemEventHandler(OnChanged);
watcher.Created += new FileSystemEventHandler(OnChanged);
watcher.Deleted += new FileSystemEventHandler(OnChanged);
watcher.Renamed += new RenamedEventHandler(OnRenamed);
// Begin watching the directory.
watcher.EnableRaisingEvents = true;
// Wait for the user to quit the program.
Console.WriteLine(@"Press 'q' to quit app.");
while(Console.Read()!='q');
}
static void OnChanged(object source, FileSystemEventArgs e)
{
Console.WriteLine("File: {0} {1}!", e.FullPath, e.ChangeType);
}
static void OnRenamed(object source, RenamedEventArgs e)
{
Console.WriteLine("File: {0} renamed to\n{1}", e.OldFullPath, e.FullPath);
}}
```

1.4 Serijalizacija

Pod serijalizacijom podrazumijevamo da objekt svoje trenutno stanje pretvori u niz byteova iz kojeg se onda deserijalizacijom omogućava iz tog niza kreiranje objekta sa istim stanje. To je pogodno bilo za spremanje objekta na neki medij, bilo za kreiranje identičnih objekata na udaljenom računalu. C# ima ugrađenu podršku za serijalizaciju. Svi ugrađeni tipovi se mogu serializirati, a kod vlastitih tipova je dovoljno klasi navesti atribut [Serializable] (i sva polja klase moraju imati taj atribut). Ukoliko neko polje klase nije potrebno serializirati, kod njega navedemo atribut [NonSerialized].

```
using System.Runtime.Serialization.Formatters.Binary;

[Serializable]
public class UserPrefs
{
    public string WindowColor;
    public int FontSize;
}

class Program
{
    static void Main(string[] args)
    {
        UserPrefs userData = new UserPrefs();
        userData.WindowColor = "Yellow";
        userData.FontSize = "50";
        // The BinaryFormatter persists state data in a binary format.
        // You would need to import System.Runtime.Serialization.Formatters.Binary
        // to gain access to BinaryFormatter.
        BinaryFormatter binFormat = new BinaryFormatter();
        using (Stream fStream = new FileStream("user.dat",
    FileMode.Create, FileAccess.Write, FileShare.None))
        {
            binFormat.Serialize(fStream, userData);
        }
    }
}
```

Sljedeći primjer varijablu (a može i više njih) sprema u datoteku auto.txt u .xhtml obliku (za pokretanje je u reference potrebno dodati System.Runtime.Serialization.Formatters.Soap).

```
using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Soap;

class Program
{
    [Serializable]
    class automobil
    {
        int presaoKilometara;
        string registracija;

        [NonSerialized]
        int trenutnaBrzina;

        public automobil(string reg, int km, int brz)
        {
            presaoKilometara = km;
            registracija = reg;
            trenutnaBrzina = brz;
        }
        public void ispis() {
            Console.WriteLine("{0}, {1}, {2}", registracija, presaoKilometara,
trenutnaBrzina);
        }
    }
    static void Main(string[] args)
    {
        automobil a = new automobil("ZG-000-ZG", 123456, 50);
        FileStream fs = File.Create("auto.txt");

        new SoapFormatter().Serialize(fs, a);

        fs.Close();
```

```
            Citaj();
        }
        static void Citaj()
        {
            FileStream fs = File.Open("auto.txt", FileMode.Open);

            automobil a = (automobil)new SoapFormatter().Deserialize(fs);

            fs.Close();
            a.ispis();
        }

}
```

Ukoliko bismo željeli podatke spremiti binarno, a ne u xml obliku, možemo koristiti `BinaryFormatter` klasu (koja se nalazi u `...Formatters.Binary` i njega ne moramo dodati u reference).
Ponekad ipak defaultno ponašanje Serializacije nije dovoljno. Ako želimo bolju kontrolu, trebamo dodati dvije metode i ISerizable interface:

```
using System.Runtime.Serialization;
    [Serializable]
    class automobil : ISerializable
    {
...
        public void GetObjectData(SerializationInfo i, StreamingContext c)
        {
            i.AddValue("presaoKilometara", presaoKilometara);
            i.AddValue("registracija", registracija);
        }
        automobil(SerializationInfo i, StreamingContext c)
        {
            presaoKilometara = i.GetInt32("presaoKilometara");
            if (c.State != StreamingContextStates.CrossProcess)
                registracija = i.GetString("registracija");
            //  x = (Tip)i.GetValue("x", typeof(Tip));
        }
    }
```

Na mjestu točkica je isti dio klase kao i u prethodnom primjeru, dok bi zakomentirani oblik trebali koristiti ako neki vlastiti tip imamo u klasi.
Na ovaj način možemo promijeniti što se serializira. Ako bismo željeli mijenjati kako se stvari serializiraju, trebali bismo napraviti klasu iz Formatter klase.
Postoje i drugi mehanizmi serijalizacije, a npr. XmlSerializer ne serializira privatne elemente.

Zadatak 2

Napravite program u kojem možemo unositi/mijenjati osobe (neka imamo ime/prezime/tel. broj). Neka se podaci spreme u datoteku i neka imamo Load/Save, te neka možemo odabrati koju datoteku koristiti. Neka se posljednje korištena datoteka prilikom sljedećeg starta automatski učita.

Zadatak 3

Korištenjem baza podataka napravite evidenciju studenata/ispita i ocjena koje je student dobio iz pojedinog kolegija.