Programmeren in C#

IO: Input en Output
Introductie tot data opslag

Werken met bestanden

File en Directory

- .Net voorziet enkele klassen om makkelijk met bestanden en mappen te werken.
- **File** en **Directory** zijn statische klassen gedeclareerd in de System.IO namespace.
  - **File** gebruikt men voor de manipulatie van bestanden.
  - **Directory** gebruikt men voor de manipulatie van mappen.
- **FileInfo** en **DirectoryInfo** vereisen een instantie en kunnen ook gebruikt worden om files te openen, te sluiten en te manipuleren.
- De statische klasse Path bevat geen functies om files te manipuleren maar wordt gebruikt om bestandsnamen te beheren.

File class

- Bevat statische functies om bestanden te manipuleren.
- Die functies vereisen een bestandsnaam.
- We kunnen de klas gebruiken om bestanden te:
  - Verwijderen: Delete(string path)
  - Kopiëren: Copy(string source, string dest)
  - Verplaatsen: Move(string source, string dest)
  - FileSecurity: GetAccessControl / SetAccessControl
  - Bestanden te openen: Open, OpenRead, OpenWrite
    - => Opent een stream of streamReader om met bestanden te werken
      (We bekijken streams in het volgende hoofdstuk.)
  - Metadata: CreationTime, AccessTime, …
  - Controleren of het bestand bestaat met Exist

Lezen en schrijven van hele bestanden

- File voorziet enkele statische functies om snel hele bestanden in te lezen of weg te schrijven.
- Enkel te gebruiken indien het om kleine bestanden gaat!
- We kunnen zowel tekst als binaire bestanden inlezen in het geheugen of wegschrijven naar een bestand.
  - Lezen van hele bestanden:
    - Binair : `public static byte[] ReadAllBytes (string path);`
    - Tekst: `public static string ReadAllText(string path, Encoding encoding);`

      => Deze methoden openen een bestand, lezen de inhoud, sluiten de file en geven de inhoud terug in een type.
  - Schrijven naar een bestand:
    - Binair: `public static void WriteAllBytes(string path, byte[] bytes);`
    - Test: `public static void WriteAllText(string path,string content, Encoding encoding);`

      => Opent het bestand, schrijft de inhoud (als het bestand reeds bestaat wordt de inhoud overschreven!) en sluit de file.

Directory

- De Directory klas werkt volgens hetzelfde principe als de file klasse.
- We kunnen de directory klas gebruiken om:
  - Te controleren of een map bestaat: Exist
  - Een map aan te maken: CreateDirectory
  - De huidige map op te vragen: GetCurrentDirectory (Set om te wijzigen)
  - De bovenliggende map te kennen GetParentDirectory
  - De namen van de harde schijven op te vragen: GetLogicalDrives
  - Alle mappen in een bepaalde map te kennen: GetDirectories
  - Alle bestanden in een map op te vragen: GetFiles
  - Of zowel bestanden als mappen GetFileSystemEntries
  - De laatste 3 kunnen we ook met een Enumerate… functie opvragen.

Path

- Path wordt gebruikt om makkelijker met bestands- en mapnamen om te gaan.
  - Map: c:\mijnMap
  - Bestand: mijnFile.txt
  - Path: c:\mijnMap\mijnFile.txt
- We kunnen Path o.a. gebruiken om:
  - de mapnaam te kennen van een path.
  - de bestandsnaam te kennen van een path.
  - een file extensie op te vragen.
  - te controleren of bestand een extensie heeft.
  - de werkmap op te vragen.
  - een willekeurige bestandsnaam te creëren met GetRandomFileName().

FileInfo en DirectoryInfo

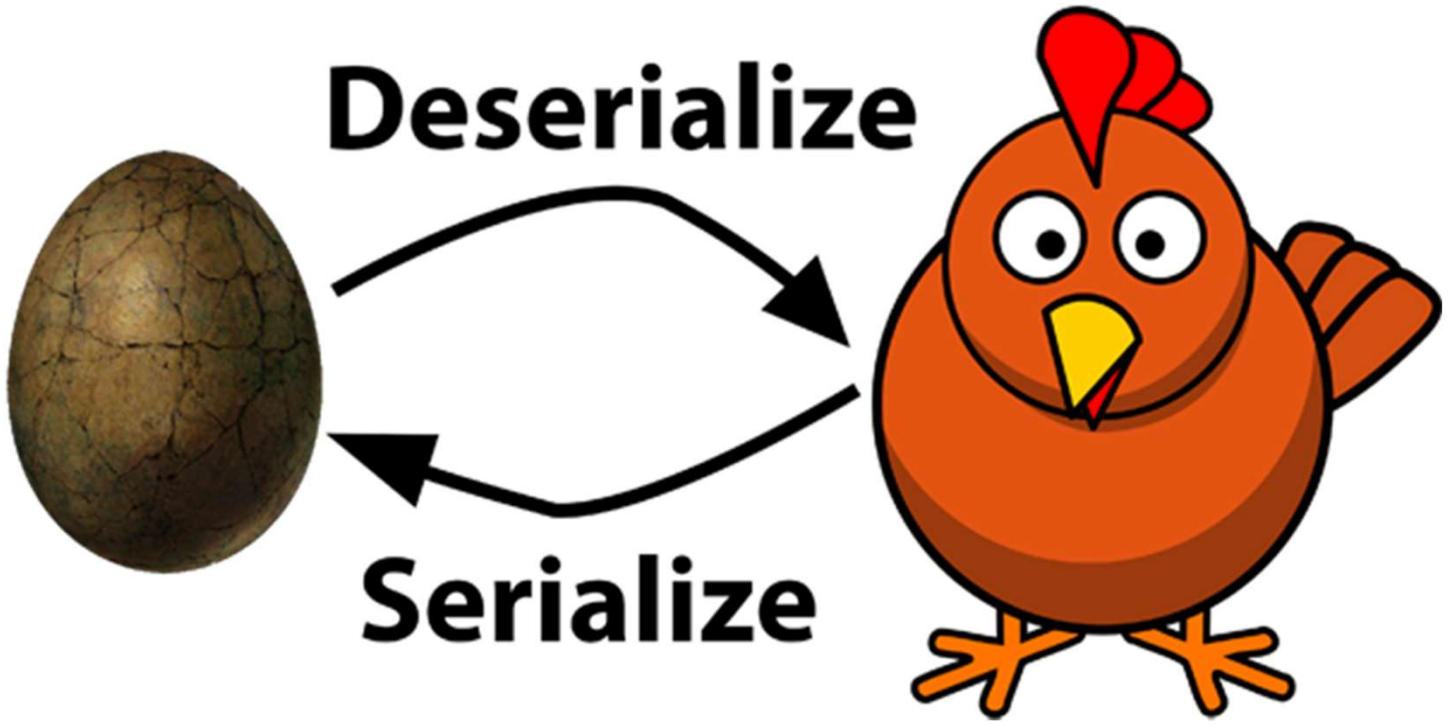
- Beide klassen bevatten de meeste functionaliteiten van de statische klassen File en Directory maar kunnen niet gebruikt worden om hele bestanden in te lezen of te schrijven.
- Moeten geïnstantieerd worden met de bestandsnaam. Indien men meer informatie moet hebben over een bestand of map is het makkelijker om een instantie te gebruiken in plaats van een statische functie waar je steeds naar hetzelfde bestand moet verwijzen.

Labo:
werken met File en Directory

- Maak functies in de CLI om:
  - In een map alle bestanden of mappen op te vragen.
  - Informatie van een bestand op te vragen:
    - Creatie datum en tijd
    - Wanneer het bestand het laatst gewijzigd is.
    - Attributen : ReadOnly, Hidden, Archive, Normal
  - De huidige ‘werkmap locatie’ van de applicatie te wijzigen.
  - De inhoud van een file te tonen op het scherm (tekst).

  - Tip: maak een aparte assembly waar je alle handige tools plaatst. Je kan die later in andere projecten herbruiken!

Werken met serializers



XmlSerializer

- Gebruikt om objecten om te zetten naar een Xml-structuur en omgekeerd.
- We kunnen de XML serializer gebruiken door middel van attributen of door de interface IXmlSerializable te implementeren.
- Er zijn echter enkele zeer belangrijke spelregels wanneer we XmlSerializer bebruiken!
  - Er moet steeds een lege constructor voorhanden zijn.
    - Een contructor met default parameters is GEEN lege contructor
  - Enkel publieke velden en properties worden behandeld.
  - Het is niet nodig om klassen, velden of properties te decoreren om ervoor te zorgen dat ze worden behandeld door de serializer.
  - De XmlSerializer gaat heel vlot om met toevoegingen die nadien gebeuren in het basisobject. Het is niet nodig om die als optioneel te decoreren.

Van en naar XML via de XmlSerializer

- Om de XmlSerializer te gebruiken moeten we eerst een instantie creëren. Die instantie verwacht het type van de klasse die gebruikt moet worden.
- Een Xml creëren gebeurt met de functie Serialize() die het object en een Stream,TextWriter of XmlWriter verwacht.

```
StringWriter sw = new StringWriter();
XmlSerializer xml = new XmlSerializer(myItem.GetType());
xml.Serialize(sw,myItem);
```

- Uit een Xml terug een object creëren doen we met de functie Deserialize() die ook een Stream, TextWriter of XmlWriter verwacht.

```
StringReader sr = new StringReader(xmlString);
XmlSerializer xml = new XmlSerializer(typeof(Item));
Item myItem = xml.Deserialize(sr) as Item;
```

Verfijnen van de XmlSerializer

- Het is mogelijk om bepaalde velden of properties te negeren door ze te decoreren met [**XmlIgnore**].
- Het is mogelijk om velden of properties op te slaan als xml-attributen door die velden te decoreren met [**XmlAttribute**].
- Het is mogelijk om de naamgeving aan te passen in de Xml van elementen en attributen:
  - [**XmlElement(“LastName”**)] public string Name;
  - [**xmlAttribute(“ClientID”)**] public int ExternalID;
- De volgorde van elementen in de Xml kunnen worden bepaald door een order parameter door te geven: [XmlElement(**Order =** 2)].
- Collecties kunnen we hernoemen door [**XmlArray**(“Naam”)] en [**XmlArrayItem**(“Item”)] te gebruiken.

Gebruik van overgeërfde klassen in XmlSerialize

- Als we werken met een parent class en child classes is het nodig om dit te configureren zodat de XmlSerializer de juiste conversies kan maken.
  - Als we vertrekken van de Persoon klasse en we hebben de overgeërfde klasses Werknemer en Klant, dan moet de Serializer en Deserializer weten dat de Persoon klasse ofwel een Klant ofwel een Werknemer is.
- Dit kunnen we op 2 manieren doen:
  - Door de basis klasse te decoreren met de mogelijke child classes. Hiervoor gebruiken we [**XmlInclude(**typeof(Werknemer))] en [**XmlInclude**(typeof(Klant))]

```
[XmlInclude(typeof(Werknemer))]
[XmlInclude(typeof(Klant))]
public class Persoon {
  …
```

  - Door in de Serializer een array van mogelijke child classes mee te geven in de contructor:

```
XmlSerializer s = new XmlSerializer(typeof(Persoon),new Type[] {typeof(Klant),typeof(Werknemer)});
```

Json Serializer

- Json Serializer is geïntegreerd in .Net Core 3+. In .Net standard of .Net Framework 4.6.1+ moeten we de System.Text.Json Nuget package installeren.
- De Json serializer bevindt zich in de namesapeces ‘System.Text.Json’ en ‘System.Text.Json.Serialization’.
- Gebruik de functie JsonSerializer.Serialize(<class>()) of JsonSerializer.Serialize(<uw klasse>, <opties>).
- We gebruiken JsonSerializer.Deserialize<T>(<json text>) om het vanuit de json data de class te herstellen. Type T is uiteraard de ontvangende class

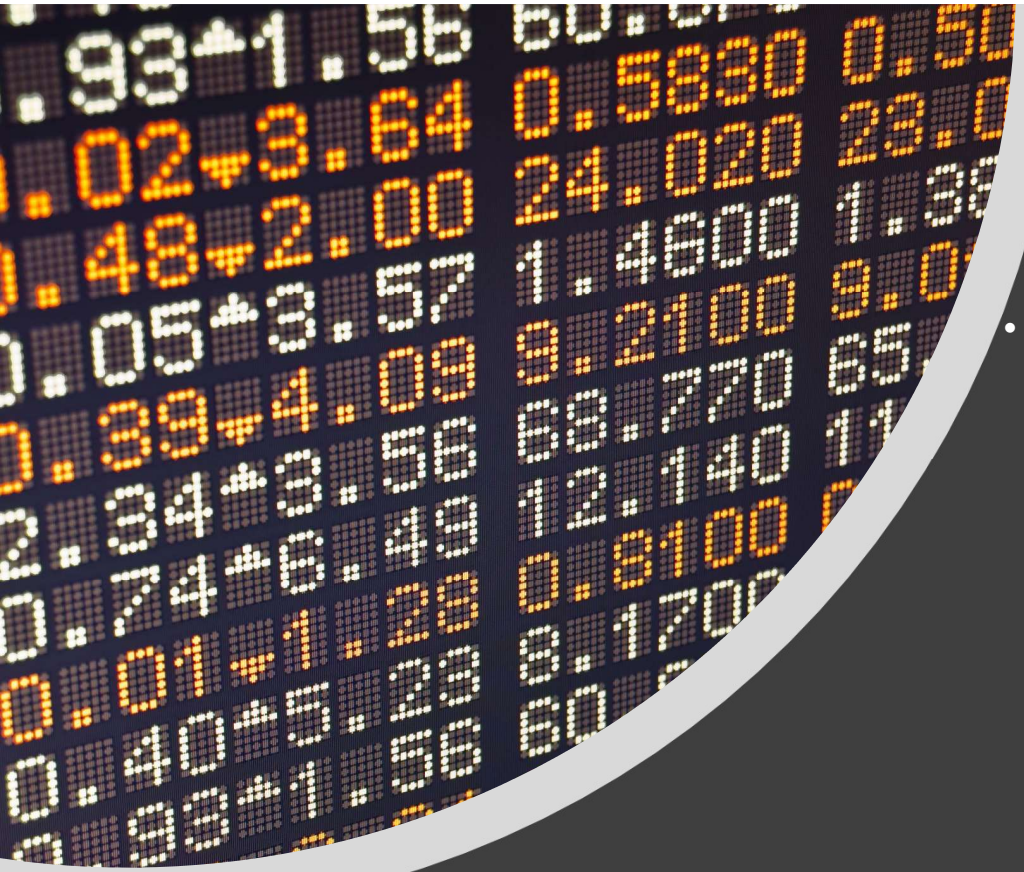
```
public class MyModel {
     public int ID { get; set; }
     public string Name { get; set; }
     public override string ToString() => $"Data model = [ID: {ID} Name: {Name}]";
}
static void Main(string[] args) {
     MyModel model = new MyModel { ID = 1, Name = "Jane Doe" };
     string json = JsonSerializer.Serialize(model);
     MyModel myRestoredModel = JsonSerializer.Deserialize<MyModel>(json);
     Console.WriteLine(myRestoredModel);
}
```

Gebruik van de Json Serializer

- De class moet een publieke contructor bevatten zonder parameters
- Alle **publieke properties** in een klasse worden serialized. Maar als de property read only is, wordt de waarde wel opgenomen in de Json maar niet terug gezet tijdens de deserialize.
- Velden worden genegeerd, ook de publieke. (momenteel)
- Alle property namen zijn standaard hoofdlettergevoelig. Dit kan in de opties worden aangepast.
- Enum types worden standaard opgeslagen als int. (er bestaat een JsonStringEnumConverter die een enum kan omzetten).
- Circulaire referenties worden herkend en zijn niet toegestaan!
- De types die met JSonSerializer kunnen worden serialized/deserialized:
  - Alle .Net primitieve types zijn toegestaan.
  - **POCO's** (User defined **P**lain **O**ld **C**lr **O**bjects).
  - Arrays van toegestane types.
  - Lijsten uit de namespace System.Collections en System.Collections.Generic
  - Dictionary als de Tvalue een POCO of primitieve is.

Verfijnen van de Json Serializer

- Gebruik de **[JsonIgnore]** property om bepaalde Velden te negeren
- Veldnamen kunnen anders zijn in de Json data bestand als in de class. Om dit te specifiëren gebruiken we de property **[JsonPropertyName(<aangepaste naam>)]**
- Om de serialize en desirialize te sturen kunnen we opties specifieren door middel van de **JsonSerializerOptions**.
  - Hierin kunnen we :
    - Json bestanden worden standaard zo compact mogelijk geschreven. We gebruiken de optie **WriteIndented** op true te zetten.
    - De property namen zijn standaard hoofdlettergevoelig. Gebruik **PropertyNameCaseInsensitive** = true om dit te wijzigen.
    - We kunnen de **PropertyNamingPolicy** zetten om de Json velden CamelCase te maken.
    - We kunnen de **encoding** aanpassen.
    - **IgnoreReadOnlyProperties** om read only properties te vermijden
    - **IgnoreNullValues** : Properties die een null waarde bevatten worden genegeerd
    - …



Labo

- Vertrek van volgende klasse structuur:
  - De class GebruikersGroep bevat volgende velden: Naam, Omschrijving en Rechten (een enum met de volgende mogelijkheden die kunnen gecombineerd worden: Gast, Basis, Uitgebreid, Administrator).
  - De class Gebruiker en die bevat volgende velden : Naam, Login, Omschrijving, Paswoord, LaatstIngelogd (datum en tijd) en een Lijst met de groepen waar die bepaalde gebruiker lid van is.
  - Maak instanties aan van beide klassen en voeg er gegevens aan toe.
  - Het paswoord mag **niet** worden opgeslagen.
- Export en Import naar XML via de XmlSerializer.
- Export en Import naar Json via de JsonSerializer.

Labo III

- Een dierenarts vraagt ons om een eenvoudig programma te schrijven om zijn administratie te doen.
  - Hij heeft graag een lijst van al zijn klanten met hun contact gegevens.
    - Naam, voornaam, geboortedatum, …
    - Het is handig dat hij de klanten op meerdere telefoonnummers kan bereiken en via mail of social media.
    - Een klant kan ook externe adressen hebben, werkadres, buitenverblijf,…
  - Een klant heeft natuurlijk één of meerdere dieren.
    - Van deze dieren willen we volgende zaken kennen:
      - Naam en eventueel ID van de chip
      - Soort dier
      - Beschrijving
      - Lijst van Medische ingrepen of onderzoeken + datum ingreep + prijs
      - Lijst met Labo data

Labo III (vervolg)

- Creëer een aparte bibliotheek waarin de data modellen komen
- Het is natuurlijk nodig dat we een leuke interface nodig hebben om de data te beheren. Creëer de nodige schermen in WPF om de data in te vullen en op te zoeken.
- Het is natuurlijk nodig dat de data bewaard wordt op een efficiënte manier. Gebruik hiervoor Xml Serializing of als alternatief JSON serializing.
- Voorzie een mogelijkheid om de labo data te importeren uit een file die de dierenarts zelf kan selecteren.