Filer og mapper med metode kald

Læsning og skrivning af filer er en grundlæggende færdighed for enhver C # udvikler. I denne opgave vil vi tackle nogle fælles scenarier med hensyn til læsning og skrivning af filer samt når der arbejdes med mapper. I opgaven 7 skal alt fil og mappe manipulation placeres i metoder. Opgave 7 er meget vigtig at få løst og den kan løses på mange måder

Der er fokus på:

* Sådan læser du en tekstfil
* Sådan skriver du til en tekstfil
* Sådan listes filer i en mappe
* Arbejder med streams

## Filer og mapper

.NET Framework indeholder et namespace System.IO, dette namespace indeholder alle funktioner, du har brug for når du skal læse og skrive til filer. Læsning og skrivning af/til filer kan opnås på flere forskellige måder, men den enkleste tilgang er at bruge File klassen.



Filklassen definerer et antal metoder, der gør læsning og skrivning af filer nemme.

### Skriv til tekstfil

For at skrive til en tekstfil kan du bruge File.WriteAllText metoden. Denne metode har to overloads, den første accepterer en sti og indholdet (som en streng), og den næste accepterer også en encoding.

Vi kommer ind på encoding senere, så nu skal vi holde fast ved den første metode:

Ved hjælp af metode kaldet skriver vi strengen "Hello World" til den lokal fil med navnet "hello.txt".

Brug af prefixet ". \" i stien indikerer, at vi bruger den aktuelle mappe - det er den mappe, hvorfra dit program afvikles.

Vi bruger en form for en streng konstant kendt som en verbatim streng. Enhver verbatim streng, der starter med "@" prefix før det første citationstegn.

Ved hjælp af en verbatim streng ændres semantikken af escape tegn:

Dette er nyttigt, når du udskriver sti variabler, da "\" tegn repræsenterer en sti-separator på Windows-maskiner. Du kan også bruge en regulær konstant streng, men du skal undgå sti-separator, hvilket resulterer i en dobbelt "\":

Hvis filen, der er angivet med sti-variablen, ikke findes, oprettes den. Hvis den eksisterer, over skrives den. Senere i denne opgave, opnår du kontrol over disse operationer.

### Læs en tekstfil

At læse en tekstfil er også en simpel operation. Metoden File.ReadAllText har også to overloads:

Resultatet af metoden kan tildeles en variabel.

Hvis filen ikke eksisterer, kastes en FileNotFoundException undtagelse for at informere dig om denne kendsgerning.

### Slet en tekst fil

Du kan slette en tekstfil ved hjælp af File.Delete metoden. Metoden tager et argument stien til den fil, du vil slette.

Metoden returnerer ikke noget (void), så du kan bare kalde den:



## Opgave 1 – skriv til fil

Skriv meddelelsen "Han skød først" i en lokal tekstfil med navnet "StarWars.txt"



## Opgave 2 – læs en tekstfil

Læs meddelelsen tilbage - ud af tekstfilen "StarWars.txt" og skriv beskeden til konsollen. Du skal først læse indholdet tilbage til en variabel, der hedder content. Skriv derefter content-variablen til konsollen ved hjælp af Console.WriteLine metoden.



## Opgave 3 – slet en fil

Slet filen "StarWars.txt"



# Arbejde med mapper

.NET Framework giver også et standard API til at arbejde med mapper (directories/folders)

Den statiske Directory klasse tilvejebringer metoder til at arbejde med mapper, herunder metoder til oprettelse, kopiering og sletning af mapper samt visning af filer i disse mapper.

## Opret en mappe

Du kan nemt oprette en mappe ved hjælp af metoden Directory.CreateDirectory



Dette metodeopkald returnerer en forekomst (instans) af DirectoryInfo, som er en kontekstuel klasse bundet til mappestien. DirectoryInfo type tillader os at indkapsle en bestemt mappe som konteksten for disse kald.

For at oprette en mappe skal du overføre stien til metoden:

Hvis mappen allerede eksisterer, sker der intet. Hvis mappen ikke eksisterer, vil den blive oprettet.



## Slet en mappe

Sletning af en mappe leveres af Directory.Delete metoden. Du kan kun fjerne en mappe, der allerede eksisterer ellers kastes en undtagelse:



Hvis den mappe, du forsøger at fjerne, ikke er tom, vil der blive kastet en undtagelse. Hvis du vil slette alle filer og mapper i en mappe, skal du bruge den anden overload, der accepterer det boolske ”rekursive ” dette vil slette alle filer og mapper under hovedmappen.



## Enumerating filer i mappe

Du kan nemt finde filer i en mappe ved hjælp af Directory.GetFiles metoden:



Disse overloads returnerer hele stien til hver fil i mappen. Lad os se på parametrene:

* path - Den fulde sti til mappen, der indeholder filerne
* searchPattern - Et wildcard-udtryk, der bruges til at matche et filnavn, f.eks. \* .exe
* SearchOption - En mulighed, der giver dig mulighed for at begrænse din søgning til kun den øverste mappe eller alle undermapper.

Eksempel



Ovennævnte kald, finder alle filer (\*) i. \folder, mens:



Dette metode kald vil finder alle JPEG-billedfiler i alle mapper under mapper.

Du kan så nemt enumerate over filerne ved hjælp af et loop:



## Opgave 4 – opret en mappe

Opret en ny mappe under den aktuelle mappe, navngiv denne mappe ”Droids”.



## Opgave 5 – slet en mappe

Slet mappen ”Droids” og sørg for at alle filer fjernes ved hjælp af den rekursive egenskaben.



## Opgave 6 – Enumerate filer i en mappe

Opret en lokal variabel navngivet files, files er et streng array. Du skal bruge Directory.GetFiles for at få alle tekstfiler i Droids og alle undermapper. Loop gennem hvert element ved hjælp af et foreach loop, skriv hvert element til konsollen ved hjælp af WriteLine.



## Brug at streams

En stream er en ordnet sekvens af bytes, der kan bruges til at læse og skrive til en form for bagrundslager - dette kunne være en fil. Du kan tænke på en stream som en abstraktion over den en lagerenhed.

Ved at behandle vores input- eller outputkilder som steams, kan vi implementere en mere generisk tilgang til læsning og skrivning af data.



Streams giver mulighed for langt større kontrol over, hvordan du læser og skriver dine data.

## Stream klassen

Stream-klassen er en abstrakt klasse, der repræsenterer basissættet af tilgængelige metoder over alle Stream-typer.

Dette er en nøgleabstraktionen, der giver os mulighed for at definere operationer, der arbejder med strømme i stedet for at skulle implementere forskellige og unikke API'er til forskellige input og output.

Der er forskellige implementeringer af Stream-typen, herunder:

* FileStream - Dette giver dig mulighed for at læse / skrive til en fil
* MemoryStream - Dette giver dig mulighed for at læse / skrive til hukommelsen
* NetworkStream - Dette giver dig mulighed for at læse / skrive til en netværk socket.

### Readers og Writers

Strømme er typisk designet til at læse og skrive absolutte byte værdier, men .NET giver også et yderligere sæt implementeringer af readers/writers, der understøtter skriveværdier i andre formater - konverterer til bytes som en del af processen.

Fælles implementeringer er:

* BinaryReader/BinaryWriter - til læsning og skrivning af data primitiver i binært format.
* StreamReader/StreamWriter - til læsning og skrive karakterværdier i specifikke encodings.

### Læs fra en fil med FileStream

Typisk når du læser en fil ved hjælp af stream metoden, består operationen af to forskellige operationer:

* Indhentning af en stream fra en fil
* Oprettelse af en reader til at læse fra strømmen.

Lad os tage et kig:



For det første skaber vi en forekomst af vores FileStream klasse. Argumenterne til FileStream klassen i ovenstående kald er stien til den fil, vi vil læse, og en værdi for FileMode enumeration, som styrer, hvordan filen åbnes. Vi bruger FileMode.Open i dette eksempel, fordi vi vil åbne filen til læsning.

Dernæst opretter vi en StreamReader der som argument tager FileStream instansen (file). Readeren vil tillade os at læse indholdet af en fil som strenge med en bestemt encoding.

For at læse en linje fra filen ved hjælp af en StreamReader kan vi bruge ReadLine-metoden:



ReadLine metoden gør det muligt for os at læse en hel linie (læser til carriage return/line feed (CRLF) eller kombination af disse (afhængigt af operativsystemet).

Herefter fremføres stream til starten af den næste linje , så det næste kald vil fortsætter fra hvor vi slap af.

Hvis vi ønskede at læse hele filen på én gang, kan vi i stedet bruge ReadToEnd metoden:



ReadToEnd læser fra den aktuelle position til slutningen, hvilket betyder, at hvis du allerede har gjort tidligere læsninger, fortsætter den fra den sidste læsningsposition til slutningen.

### Luk reader og fil

Det er vigtigt, at du lukker din stream, når du er færdig med det. Dette gør det muligt for systemet at frigive ressourcer.



Hvis du kalder Close på StreamReader, lukker denne automatisk den underliggende strøm:



Dette gør det lettere at kontrollere lukningen af alt, hvad du benytter.

### Læs fil linje for linje

For at læse hver enkelt linje i filen kan vi bruge en løkke og tjekke readeren på hver gentagelse:



Readeren giver en EndOfStream egenskab, der er sat til sand, hvis vi når slutningen af filen. Dette kan benyttes som betingelse for en løkke.

### Skriv til fil med FileStream

Ligesom vi kan læse en fil ved hjælp af FileStream, kan vi også skrive til en fil.

Der er et par muligheder, når det drejer sig om at skrive til en fil - afhængigt af om du vil oprette en ny fil, overskrive en eksisterende fil eller endda tilføje til filen.



Ligesom som før åbner vi vores fil ved hjælp af en FileStream. FileMode enumeration tilbyder nogle muligheder, når du skriver en fil:

* Create - Opretter en ny fil - hvis filen findes, overskrives den.
* CreateNew - Opretter en ny fil, kaster en undtagelse hvis fil allerede findes
* Append - Tillader tilføjelse i eksisterende fil.

Vi vil holde tingene simple og oprette en ny fil ved hjælp af FileMode.Create. Dernæst opretter vi vores StreamWriter ved hjælp af vores FileStream som konstruktørparamete

For at skrive en linje til filen ved hjælp af StreamWriter, kan vi bruge WriteLine metoden.



WriteLine metodens overload accepterer et streng argument og konverterer det til de data, der er skrevet til den underliggende Stream - i dette tilfælde vores fil.

Husk, at det er vigtigt at lukke filen, når du er færdig med det:



Hvis vi ønsker at skrive en collection af elementer til vores fil, kan vi benytte



## Opgave 5 – læs fra fil

Læs fra den lokale fil Movies.txt

* Opret en fil stream kaldet file ved hjælp af den lokale fil "Movies.txt"
* Opret en stream reader kaldet reader ved hjælp af file stream
* Hvis vi ikke er i slutningen af streamen, skal du oprette en lokal string med navnet movie og tildele den værdien af den næste linje fra filen
* Skriv hver film til konsollen ved hjælp af WriteLine metoden.



## Opgave 6 – skriv til fil

Skriv til den lokale fil Actors.txt

* Opret en fil stream kaldet file ved hjælp af den lokale fil "Movies.txt" for at overskrive filen
* Opret en stream writer kaldet writer ved hjælp af file stream
* Skriv hvert element i listen actors til filen på en ny linje. Brug en foreach loop til dette



## Opgave 7 – Fil og mappe manipulation

Lav et program der implementere logik til fil og mappe manipulation

==================================================

                                      H1 Queue Operations Menu

==================================================

1. Add file   
  
2. Delete file   
  
3. Display files  
  
4. Add folder  
  
5. Search for file in folder  
  
6. Exit

Enter your choice:1