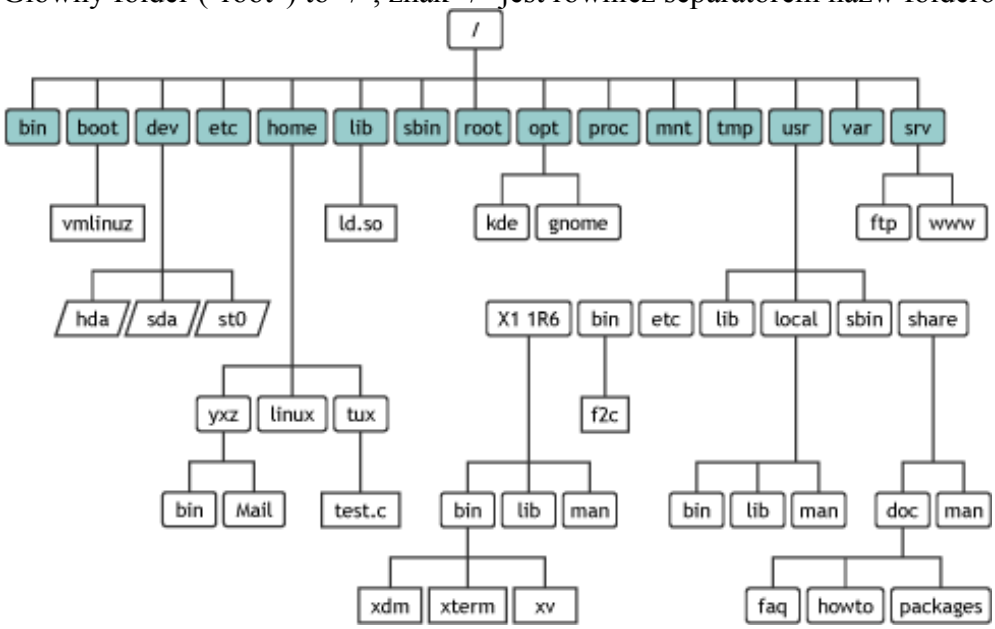
System plików

Sposób zapisywania i odczytywania danych na nośnikach danych (dyski FDD/HDD/SSD, CD/DVD, ale też dyski sieciowe w sieciowych systemach plików). Obsługa systemu plików jest jedną z podstawowych funkcji systemu operacyjnego. Różne systemy operacyjne mają różne systemy plików, np. DOS – FAT, Windows – NTFS, Unix – UFS, Linux – ext, …
Niektóre systemy obsługują kilka rodzajów systemów plików;
Podstawowe operacje systemu plików
– utworzenie/usunięcie katalogu
– utworzenie/usunięcie pliku
– otwarcie/zamknięcie pliku
– zapis/odczyt pliku
Aplikacje korzystają z systemu plików wyłącznie **pośrednio**, poprzez system operacyjny;

Hierarchia systemu plików

Pierwsze systemy plików miały strukturę "płaską", obecnie wszystkie systemy plików mają strukturę hierarchiczną, którą tworzą katalogi i podkatalogi – w formie drzewa
W systemach Unix/Linux/Mac OS jest tylko jedno takie drzewo
(niezależnie od liczby fizycznych dysków), a dodatkowe dyski są "doczepiane" do tego drzewa; Główny folder ("root") to "/", znak "/" jest również separatorem nazw folderów



W systemie Windows każdy wolumin (dysk lub część dysku, tzw. partycja) jest oznaczony literą (pierwszy wolumin to dysk C, drugi D itd.) i ma własne drzewo katalogów;
Separatorem nazwy woluminu jest ":", a separatorem nazw folderów "\".
System Windows pierwotnie był tylko nakładką na DOS, dlatego zachowuje niektóre reguły systemu DOS – np. małe i duże litery w nazwach nie są rozróżniane

Polecenia powłoki Windows

Dla lepszego zrozumienia reguł dotyczących dysków i ścieżek warto poćwiczyć polecenia powłoki systemu Windows:

- <litera>: - zmiana dysku (np. "d:")
- cd [dysk:]ścieżka- zmiana katalogu
- dir [dysk:][ścieżka][maska-plików] – lista katalogów i plików

- type [dysk:][ścieżka]plik – treść pliku
- copy [dysk:][ścieżka]plik [dysk:][ścieżka][plik] – kopia pliku
- rename [dysk:][ścieżka]plik plik – zmiana nazwy pliku
- <polecenie> /? – pomoc

W maskach plików (m.in. dla poleceń dir i copy) można uzywać symboli wieloznacznych: "?" (zastępuje jeden dowolny znak) oraz "*" (zastępuje dowolną liczbę znaków) – np. `dir *.txt`

Dysk i folder bieżący

W systemie Windows każdy dysk ma folder bieżący, a dodatkowo jeden z dysków jest dyskiem bieżącym; Przełączenie się na inny dysk nie powoduje zmiany folderu bieżącego, ani na dysku źródłowym, ani na docelowym. Jeżeli w poleceniach (np. dir) pominąć dysk – to polecenie odnosi się do dysku bieżącego; jeżeli pominąć folder – to polecenie dotyczy folderu bieżącego, np.:

- `dir` // bieżący folder bieżącego dysku
- `dir c:` // bieżący folder dysku c:
- `dir \arch` // folder "arch" głównego folderu bieżącego dysku
- `dir arch` // folder "arch" bieżącego dysku i folderu

Specjalne nazwy folderów i ścieżki względne

Funkcjonują też dwie specjalne nazwy folderów:

- `.` // folder bieżący bieżącego dysku
- `..` // folder nadrzędny do "."

Nazw "." i ".." można używać we wszystkich poleceniach (dir, copy, …):

- `dir ..` // folder nadrzędny do bieżącego, bieżący dysk
- `dir .\..` // jw.
- `dir ..\..` // folder dwa poziomy powyżej bieżącego

Ścieżka bezwzględna (jak w kwalifikowanej nazwie pliku) zawiera literę dysku i pełną ścieżkę, zaczynając od "\"; Każda inna postać ścieżki to ścieżka względna

Folder bieżący programu

Windows jest systemem wielozadaniowym – każdy uruchomiony program ma dysk i folder bieżący, niezależnie od innych uruchomionych programów. Folder bieżący, zależnie od sposobu uruchomienia:

- Menager plików: (2x kliknięcie) – folder, w którym znajduje się plik programu
- Skrót – określony we właściwościach skrótu – domyślnie jw., ale można zmienić
- Linia polecenia – folder, z którego uruchomiono program

```
C:\Temp> D:\Programy\Program.exe  // C:\Temp
```

Klasa Directory

Klasa Directory jest klasą statyczną, zawiera wyłącznie metody statyczne służące do programowej obsługi katalogów; Jako jeden z argumentów większości tych metod należy podać ścieżkę katalogu Istnieje też klasa DirectoryInfo – klasa "instancyjna", reprezentująca konkretny katalog (wskazany podczas tworzenia obiektu); wygodniejsza, jeżeli operacji ma być więcej

```
Directory.Delete(@"C:\Temp");

DirectoryInfo di = new DirectoryInfo(@"C:\Temp");
di.Delete();
```

Wybrane metody

- **GetCurrentDirectory**, **SetCurrentDirectory** – zwraca lub ustawia bieżący dysk i bieżący katalog
- CreateDirectory – tworzy katalog
- Delete – usuwa katalog, ewentualnie razem z zawartością
- **EnumerateDirectories** – zwraca listę podkatalogów; rezultat jest typu IEnumerable, który można zamienić np. na łańcuch:

  ```
  String list = String.Join("\r\n", Directory.EnumerateDirectories("."));
  ```

  Jako druki argument można podać maskę (np. "log-2018-??")
- **Exists** – sprawdza, czy katalog o wskazanej nazwie istnieje
- **EnumerateFiles** – zwraca listę plików we wskazanym katalogu
  (rezultat jest typu IEnumerable, jak przy EnumerateDirectories)
  Jako druki argument można podać maskę (np. "*.txt")
- **GetDirectiories**, **GetFiles** – jak Enumerate, ale zwraca tablicę `String[]`

  ```
  String list = String.Join("\r\n", Directory. GetFiles(".", "*.txt"));
  ```

- GetCreationTime, GetLastAccessTime, GetLastWriteTime

Klasa File

Klasa File jest klasą statyczną, zawiera wyłącznie metody statyczne służące do programowej obsługi plików; Jako jeden z argumentów większości tych metod należy podać ścieżkę do pliku (bezwzględną lub względną) Istnieje też klasa FileInfo – klasa "instancyjna", reprezentująca konkretny plik (wskazany podczas tworzenia obiektu);

```
File.Copy("plik.txt", "kopia.txt");

FileInfo fi = new FileInfo("plik.txt");
fi.CopyTo("kopia.txt");
```

Wybrane metody

- **Exists** – sprawdza, czy plik o wskazanej nazwie istnieje
- **ReadAllLines**, **ReadAllText** – odczytuje cały tekst; zwraca kolekcję (IEnumerable) albo pojedynczy String
- AppendAllLines, AppendAllText – dopisuje na końcu pliku kolekcję łańcuchów (typu IEnumerable, zatem m.in. tablicę String[])albo pojedynczy łańcuch

- Copy, Move – kopiuje/przenosi istniejący plik do wskazanej lokalizacji
- Create, Open, OpenRead, OpenWrite – tworzy i/lub otwiera plik; zwraca obiekt FileStream (strumień binarny)
- CreateText – tworzy plik tekstowy; zwraca StreamWriter
- OpenText – otwiera plik tekstowy; zwraca StreamReader
- GetCreationTime, GetLastAccessTime, GetLastWriteTime

Klasa Path

Wybrane metody

- **Combine** – łączy kilka łańcuchów, dodając "\" w razie potrzeby
- **GetFullPath** – można użyć do konwersji ścieżki względnej na bezwzględną:

```
String fullPath = Path.GetFullPath(".");
```

- GetDirectoryName, GetFileName, GetExtension
- ChangeExtension – zmienia rozszerzenie nazwy, reszta bez zmian

Wybrane właściwości

- **DirectorySeparatorChar** – "\" albo "/", zależnie od platformy, "\" dla Windows
- **AltDirectorySeparatorChar** – alternatywny separator, "/" dla Windows
  można używać "\" i "/" zamiennie, działają obie wersje

Klasa Environment

Klasa Environment dostarcza informacje o tzw. otoczeniu (dosł. środowisku) programu Zawiera niektóre ustawienia systemu, np. nazwę komputera, użytkownika, wersję systemu operacyjnego, folder systemowy, bieżący folder dla procesu, komendę uruchomienia programu. Udostępnia kilka metod, a w tym GetFolderPath, która dostarcza ścieżki do folderów systemowych, np. pulpitu

Wybrane metody

- GetFolderPath – zwraca ścieżkę do folderu systemowego, np.:

```
String Pulpit = Environment.GetFolderPath(
   Environment.SpecialFolder.Desktop
   );
```

Typ wyliczeniowy **SpecialFolder** zawiera m.in. foldery systemowe Windows, System, ProgramFiles, ProgramFilesX86, MyDocuments, MyMusic, MyPictures, Desktop

Wybrane właściwości

- MachineName – nazwa komputera
- UserName – nazwa użytkownika systemu
- OSVersion – obiekt OperatingSystem z informacją o systemie

Strumienie

System operacyjny stosuje model komunikacji ze światem zewnętrznym oparty na plikach i strumieniach – znajduje to odbicie w językach C/C++ oraz (w mniejszym stopniu) C#.
Plikiem jest nie tylko plik na dysku, ale też konsola, port szeregowy RS, port równoległy, drukarka, połączenie sieciowe, wirtualne połączenie między programami, itd.
Strumień jest programową reprezentacją pliku, definiuje dostępne operacje wejścia/wyjścia

Pliki można podzielić na 2 kategorie:

- **Pliki tekstowe** – zawierają wyłącznie kody znaków, w określonym rodzaju ich kodowania (np. Unicode, ISO, Windows, …), oraz kody sterujące (jak np. CR, LF, EOF, …) ;
- **Pliki binarne** – pliki o dowolnej zawartości, oprócz tekstowych;
  Zazwyczaj sposób reprezentacji danych w plikach binarnych jest taki sam, jak w pamięci operacyjnej komputera – do plików binarnych można zapisać wartości dowolnych zmiennych bez ich konwersji na tekst, w dokładnie takiej samej postaci, w jakiej istnieją w programie;

Zapis i odczyt plików binarnych jest szybszy (nie ma konwersji z/na tekst) i mają one mniejszą objętość, ale ich obsługa jest bardziej złożona i bardziej podatna na błędy

Klasa Encoding

Encoding - klasa statyczna, reprezentuje sposób kodowania znaków
Kodowanie dotyczy wyłącznie odczytu i zapisu tekstu poza program C# - wewnętrznie znaki i łańcuchy są zawsze zapisane w Unicode (UTF-16). Nie można utworzyć obiektu klasy Encoding przy pomocy operatora new, zamiast tego można wykorzystać kodowania "standardowe", dostępne jako właściwości klasy Encoding:

- Encoding.ASCII – kod ASCII 7-bitowy
- Encoding.Unicode – Unicode, kodowanie UTF-16
- Encoding.UTF8
- Encoding.UTF32

Można również utworzyć dowolne inne kodowanie, przy pomocy metody GetEncoding, np.:

```
Encoding.GetEncoding("windows-1250")
```

Najbardziej popularne metody kodowania polskich znaków:

- `ibm852` – system DOS
- `windows-1250` – system Windows
- `iso-8859-2` – standard międzynarodowy

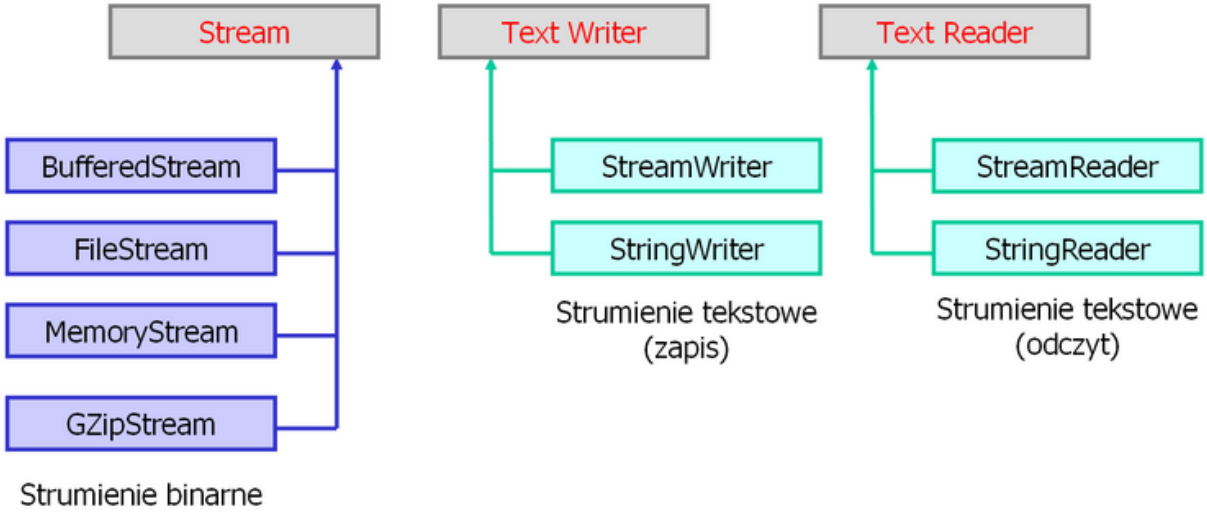
```
FF FE 5A 00  61 00 7C 01  F3 00 42 01  07 01 20 00  ˙ţZ.a.|.ó.B... .
67 00 19 01  5B 01 6C 00  05 01 20 00  6A 00 61 00  g...[.l... .j.a.
7A 01 44 01  0D 00 0A 00                           z.D.....

EF BB BF 5A  61 C5 BC C3  B3 C5 82 C4  87 20 67 C4  đ»żZaĹĽĂłĹ‚Ä† gÄ
99 C5 9B 6C  C4 85 20 6A  61 C5 BA C5  84 0D 0A     ™Ĺ›lÄ… jaĹşĹ„..

5A 61 BF F3  B3 E6 20 67  EA 9C 6C B9  20 6A 61 9F  Zażółć gęślą jaź
F1 0D 0A                                           ń..

5A 61 BE A2  88 86 20 67  A9 98 6C A5  20 6A 61 AB  Zaľ˘†  g©˜lĄ ja«
E4 0D 0A                                           ä..
```

Ten sam tekst, kodowanie UTF-16, UTF-8, windows-1250, ibm852;
FF FE oraz EF BB to sekwencje BOM, CR=0x0D, LF=0x0A, spacja=0x20

Klasa Console stosuje kodowanie wg ustawień systemu; można je odczytać (InputEncoding oraz OutputEncoding). Strumienie tekstowe mają kodowanie domyślne UTF-8, ale tworząc strumień można wskazać inne; Rozsądek nakazuje stosować wszędzie UTF-8, a inne jedynie do odczytu plików zapisywanych przez starsze programy. Jeżeli strumień tekstowy, np. plik, ma tzw. BOM (Byte Order Mark), który oznacza kodowanie inne, niż wskazane przy tworzeniu strumienia, to zostanie ono zmienione. W przypadku UTF-16 i UTF-32 dochodzi jeszcze problem kolejności zapisu bajtów (little/big endian); C# obsługuje oba, powinno wybierać właściwe dla platformy (Intel/AMD – little endian, IBM/SPARC – big endian)

Hierarchia klas strumieni

Wszystkie strumienie (a także inne klasy związane z plikami i katalogami, np. File, Directory, Path) są zdefiniowane w przestrzeni nazw System.IO, którą należy dodać do programu



StreamReader

StreamReader jest strumieniem tekstowym, służącym do odczytu plików. Konstruktor StreamReader ma kilka wersji, a w tym:

```
StreamReader(String path)
StreamReader(String path, Encoding encoding)
```

path – nazwa pliku, np. "Plik.txt", albo nazwa kwalifikowana, np. @"d:\pliki\plik.txt"; Rozszerzenie nie wpływa na odczyt; **encoding** – kodowanie znaków, np. Encoding.UTF8; jeżeli plik ma BOM, to kodowanie zostanie zmienione wg BOM. Plik musi istnieć, inaczej próba otwarcia skończy się błędem. Przykład:

```
StreamReader sr = new StreamReader("plik.txt");
```

Metody **Read** i **ReadLine** – działają identycznie, jak metody klasy Console, tyle że czytają z pliku. Właściwość **EndOfStream** – pozwala sprawdzić, czy została już odczytana cała zawartość pliku. Metoda **Close** – zamyka plik; Dopóki plik jest otwarty, jest zablokowany dla innych programów i zużywa zasoby; kiedy plik nie jest już potrzebny, należy go zamknąć:

```
StreamReader sr = new StreamReader("plik.txt");
while (!sr.EndOfStream)
{
  ln = sr.ReadLine();
}
sr.Close();
```

StreamReader

StreamReader jest strumieniem tekstowym, służącym do zapisu plików Konstruktor StreamWriter ma kilka wersji, a w tym:

```
StreamReader(String path, Boolean append,
  Encoding encoding)
```

path – nazwa pliku (jak w StreamReader); **append** – czy dane dopisać do pliku (domyślnie false); **encoding** – kodowanie znaków; Jeżeli plik nie istnieje, to zostanie utworzony, jeżeli istnieje **append** == false, to zostanie usunięty i utworzony na nowo. Przykład:

```
StreamWriter sw = new StreamWriter("plik.txt");
```

Metody **Write** i **WriteLine** – działają identycznie, jak metody klasy Console, tyle że zapisują do pliku. Metoda **Close** – zamyka plik; Dopóki plik jest otwarty, jest zablokowany dla innych programów i zużywa zasoby; kiedy plik nie jest już potrzebny, należy go zamknąć

```
StreamWriter sw = new StreamWriter("plik.txt");
sw.WriteLine("Dziś mamy {0}", DateTime.Now);
sw.Close();
```

FileStream

Strumień binarny do zapisu i odczytu plików. Konstruktor ma kilkanaście wersji, a w tym:

```
FileStream(String path, FileMode mode, FileAccess access)
```

path – nazwa pliku (jak w StreamReader);
mode – tryb, w jakim plik ma być otwarty:

- CreateNew – tworzy nowy plik (jeżeli istnieje – wyjątek)
- Create – tworzy nowy lub nadpisuje isniejący
- Open – otwiera istniejący (jeżeli nie ma – wyjątek),
- Truncate – nadpisuje istniejący (jeżeli nie ma – wyjątek),
- Append – tworzy lub otwiera istniejący do dopisywania

access – Read, Write, ReadWrite; ten argument można pominąć (domyślnie przyjmuje wartość ReadWrite, oprócz mode=Append, wtedy Write)

Przykład:

```
FileStream fs = new FileStream(
   @"C:\plik.dat",
   FileMode.Create
   FileAccess.Write
);
```

- Metody służące do zapisu/odczytu mają bardzo ograniczone możliwości – potrafią jedynie zapisać/odczytać bajt (`Byte`) lub tablicę bajtów (`Byte[]`) i z tego względu FileStream jest mało przydatny. W praktyce korzysta się z innych strumieni (`BinaryReader` lub `BinaryWriter`), które wewnętrznie używają FileStream
- Każdy odczyt/zapis przesuwa pozycję strumienia o liczbę odczytanych/zapisanych bajtów; Pozycję strumienia można dowolnie zmieniać metodą **Seek**
- OS optymalizuje użycie dysku – dane są buforowane w pamięci i fizycznie zapisywane na dysku dopiero po zamknięciu pliku;Można wymusić zapis przed zamknięciem – met. Flush

BinaryReader

Strumień binarny do odczytu plików. Konstruktor BinaryReader ma trzy wersje, a w tym:

```
BinaryReader(Stream stream)
BinaryReader(Stream stream, Encoding encoding)
```

stream – strumień (np. FileStream), otwarty do odczytu
encoding – kodowanie znaków, np. Encoding.UTF8

Przykład:

```
FileStream fs
fs = new FileStream(@"C:\plik.dat", FileMode.Open);
BinaryReader br = new BinaryReader(fs);
```

BinaryReader Może odczytywać wszystkie typy proste, pojedyncze wartości lub tablice bajtów, np.:

```
Int32 ival = br.ReadInt32();
Byte bval = br.ReadByte();
Byte[] tab = br.ReadBytes(100);
```

Każdy odczyt przesuwa wskaźnik pliku (punkt odczytu);
Metoda Close – zamyka strumień, który został użyty do utworzenia BinaryReader (np. FileStream); kiedy plik nie jest już potrzebny, należy go zamknąć

```
sr.Close();
```

BinaryWriter

Strumień binarny do zapisu plików. Konstruktor BinaryWriter ma trzy wersje, a w tym:

```
BinaryWriter(Stream stream)
BinaryWriter(Stream stream, Encoding encoding)
```

stream – strumień (np. FileStream), otwarty do zapisu
encoding – kodowanie znaków, np. Encoding.UTF8

Przykład:

```
FileStream fs
fs = new FileStream("plik.dat",FileMode.Create);
BinaryWriter bw = new BinaryWriter(fs);
```

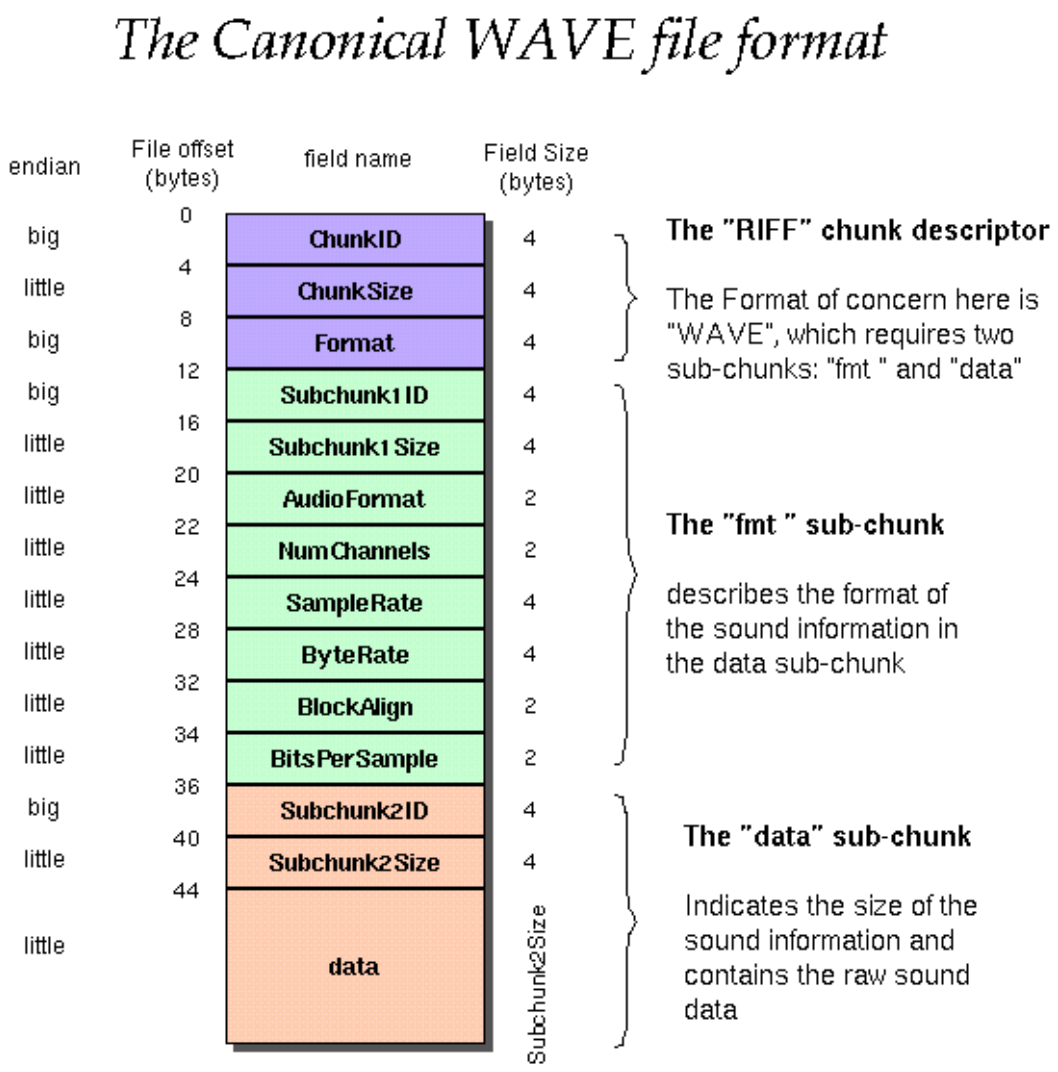
Zamiast tworzyć obiekt FileStream można też użyć File.Open();
BinaryWriter może zapisywać wszystkie typy proste – pojedyncze wartości lub tablice bajtów, służy do tego jedna metoda, **Write**, która ma kilkanaście wersji, np.:

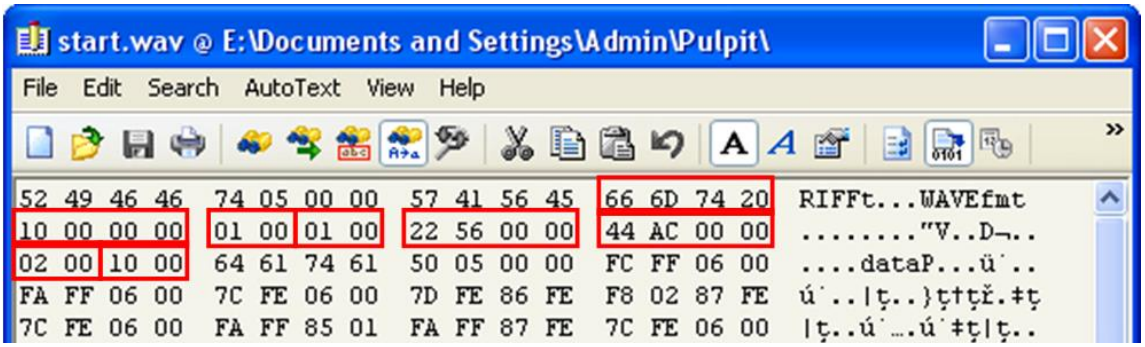
```
Double val = 1.2345;
bw.Write(val);
```

Każdy zapis przesuwa wskaźnik pliku (punkt zapisu);
Metoda Close zamyka strumień, który został użyty do utworzenia BinaryWriter:

```
sr.Close();
```

Standardowe formaty plików





Wyjątki programowe

Wyjątek programowy to rodzaj sytuacji nadzwyczajnej, np. błędu, z którym program nie potrafi sobie poradzić, którego obsłużenie jest oddane "wyższym warstwom" programu

- Przykład: jeżeli funkcja Int32.Parse otrzyma do przetworzenia ciąg znaków, który nie jest liczbą całkowitą, to wygeneruje wyjątek;
  Funkcja Parse nie może naprawić błędu, nie powinna też decydować jak zareagować na błąd (to należy do "wyższej warstwy" programu)
- Jeżeli wyjątek nie zostanie obsłużony, to program zostanie zatrzymany

Do obsługi wyjątków służą instrukcje:

- throw – rzucanie wyjątku
- try-catch – obsługa wyjątku
- try-finally – czyszczenie po wystąpieniu wyjątku

Instrukcja try-catch:

```
try
{
 StreamReader sr = new StreamReader (fileName);
 String ln = sr.ReadLine();
}
catch (FileNotFoundException e)
{
  Console.WriteLine("Plik nie istnieje\r\n");
}
catch
{
  Console.WriteLine("Błąd odczytu pliku \r\n");
}
```

System pomocy VS podaje listę typów wyjątków, które może rzucić konstruktor lub metoda. Czasami jest jeden-dwa typy, ale może być ich dużo więcej.

Wyjątki, jakie może rzucić jeden z konstruktorów StreamReader:

Exceptions

Exception	Condition
ArgumentException	*path* is an empty string ("").
ArgumentNullException	*path* is **null**.
FileNotFoundException	The file cannot be found.
DirectoryNotFoundException	The specified path is invalid, such as being on an unmapped drive.
IOException	*path* includes an incorrect or invalid syntax for file name, directory name, or volume label.

Zadania

Proszę napisać program, który…

1. Zapisuje do pliku kilka linijek tekstu. Plik powinien zostać utworzony na pulpicie i mieć rozszerzenie txt, aby można było łatwo sprawdzić jego zawartość w notatniku systemowym; Do utworzenia kwalifikowanej nazwy pliku należy wykorzystać metody klas `Environment` i `Path`, natomiast do zapisu pliku strumień `SteamWriter`
2. * Wykonuje zadania z punktu 1, ale stosując różne metody kodowania znaków (np. `ASCII`, `UTF16`, `UTF32`, `ibm852`, `windows-1250`, `iso-8859-2`); Należy sprawdzić czy tekst z pliku jest poprawnie wyświetlany w różnych programach (np. Notepad, MS Wrod)
3. Odczytuje zawartość pliku tekstowego (można wykorzystać plik z zadania 1) i kopiuje jego treść dna ekran. Cały plik należy odczytać pojedynczą instrukcją – wykorzystując odpowiednią metodę klasy `File`
4. Wykonuje zadania z punktu 3, ale wykorzystując strumień `StreamReader` (plik należy odczytywać linia po linii)
5. Wykonuje zadania z punktu 4, a dodatkowo przetwarza odczytane dane (treść pliku należy odpowiednio zmodyfikować w notatniku systemowym: każda linia powinna zawierać liczbę). Przetwarzanie danych może polegać np. na zamianie łańcucha znaków na liczbę oraz obliczeniu jej kwadratu
6. Wykonuje zadania z punktu 5, ale nazwę pliku powinien podawać użytkownik – program powinien dodać do nazwy pliku ścieżkę do folderu systemowego pulpitu; do utworzenia kwalifikowanej nazwy pliku należy wykorzystać metody klas `Environment` i `Path`
7. * Wykonuje zadania z punktu 6, a dodatkowo automatycznie dodaje rozszerzenie nazwy .txt, jeżeli użytkownik poda nazwę bez rozszerzenia.
8. * Wykonuje zadania z punktu 6 lub 7, a dodatkowo ma obsługę wyjątków. Wyjątek można sprowokować np. wpisując do pliku nie-liczby albo wpisując błędnie nazwę pliku
9. Odczytuje z pliku .wav informacje o jego formacie (liczba kanałów, częstotliwość próbkowania i rozdzielczość próbkowania – liczba bitów na próbkę).
   Plik (np. tada.wav) należy wcześniej skopiować z folderu Window\Media na pulpit.

   (Jako wprowadzenie do ćwiczenia 10 należy przećwiczyć polecenia powłoki systemu Windows: **używając wyłącznie konsoli** należy – przejść do folderu Pulpit, – utworzyć tam folder i 2-3 jego podfoldery, – skopiować tekst z konsoli do pliku tekstowego, – skopiować plik do innego folderu, – zmienić nazwę pliku, – wyświetlić jego treść; **Należy używać różnych notacji ścieżek względnych**, np. wyświetlać plik z innego folderu niż bieżący; należy też wypróbować znaczenie nazw specjalnych "." oraz "..")
10. Wykonuje interaktywnie wybrane polecenia powłoki systemu Windows (`dir`, `cd`, `type`). Program powinien pracować w „nieskończonej” pętli while i przy każdej iteracji wczytywać i wykonywać jedno polecenie użytkownika (np. `type plik.txt` – wyświetlenie pliku). Do realizacji poleceń należy wykorzystać metody klas `Direktory` oraz `File`
11. * Wykonuje zadania z punktu 10, a dodatkowo ma obsługę wyjątków – w przypadku błędu (np. podania nieprawidłowej ścieżki w poleceniu `cd` albo pliku w poleceniu `type`) nie może „wysypać się” – powinien wyświetlić komunikat o błędzie i działać dalej