**Pliki**

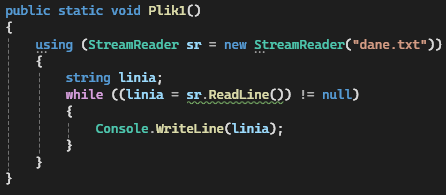
**Zwalnianie obiektów z pamięci.**

Za zwalnianie pamięci po obiektach będącymi instancjami klas odpowiedzialny jest odśmiecacz (ang. garbage collector). Cyklicznie przeszukuje pamięć sterty w poszukiwaniu obiektów, które nie są wskazywane przez żadną referencję. Jeżeli taki obiekt zostanie znaleziony jest usuwany. Użytkownik nie ma żadnego wpływu. Dzięki temu niebezpieczeństwo wycieku pamięci jest mniejsze.

Usuwając obiekty uruchamiana jest metoda Finalize. W której powinny być umieszczone polecenia zamykające zewnętrzne nie zarządzone zasoby.

Są sytuacje w których nie chcemy czekać aż odśmiecacz znajdzie zasoby pozbawione referencji i je usunie. Przykładem są klasy korzystające z plików. Chcemy, aby dostęp do nich został zamknięty jak najszybciej aby nie blokować prób ich ponownego otwarcia. W takiej sytuacji klasy definiują interfejs IDisposable, wymuszający obecność metody Dispose, w której powinny znaleźć się polecenia zwalniające wszystkie zewnętrzne zasoby używane przez obiekt. Tę metodę możemy wywołać, gdy mamy pewność, że obiekt nie jest już potrzebny. Możemy również wykorzystać ten mechanizm, kiedy nasza klasa zajmuje dużo pamięci i chcemy mieć pewność, że ta pamięć będzie zwolniona najszybciej jak to możliwe.

Słowo kluczowe using nie dotyczy tylko przestrzeni nazw ale również użyte jako modyfikator w deklaracji zmiennej referencyjnej, z inicjacją wartości, wówczas na rzecz tego obiektu zostanie automatycznie wywołana metoda Dispose wraz z zakończeniem metody w której ta zmienna lokalna jest zadeklarowana. Modyfikator using może być użyty tylko w przypadku typów implementujących interfejs IDisposable



Żeby powyższy kod zadziałał koniecznie musimy założyć plik dane.txt w miejscu gdzie jest plik wykonywalny. Dodajmy taki plik do projektu, wybierając z menu Projekt polecenie: Dodaj nowy element… - plik tekstowy  w kategorii ogólne czyli General. Aby dodany plik był kopiowany do katalogu ze skompilowanym programem należy go zaznaczyć w podoknie Eksplorator rozwiązań i w oknie właściwości zmienić jego ustawienie „Kopiuj do katalogu wyjściowego na Zawsze kopiuj . Do pliku należy wpisać w edycji kilka linijek tekstu.

Klasa StreamReader pozwala czytać z pliku tak jak ze strumienia. Żeby ten program zadziałał kursor musi być ustawiony na początku pliku. Wtedy metoda StreamReadr -ReadLine odczyta linijkę i ustawi się na początku następnej linijki, jeżeli kursor będzie na końcu pliku to warunek w pętli wskaże na null i pętla się nie wykona. Polecenie (linia=sr.ReadLine()) !=null operator przypisania „=” nie tylko zmienia wartość zmiennej linia, ale również zwraca jej nową wartość jako wynik. Możemy zatem sprawdzić, czy wynik ten jest różny od null.

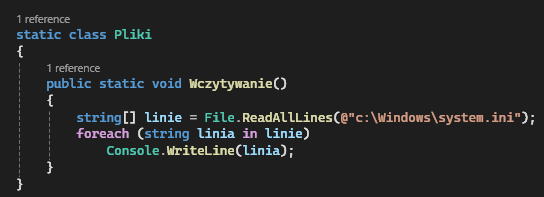
Pliki tekstowe.

Poznaliśmy klasę StreamReader – dzięki ,której odczytaliśmy plik tekstowy linijka po linijce. Do dyspozycji mamy analogiczną klasę StreamWrite umożliwiającą zapisywanie plików.

Jeśli pliki nie są duże, wygodniej jest korzystać z zestawu statycznych metod zdefiniowanych w klasie System.IO,File. Pierwsza o nazwie **ReadAllLines**, służy do odczytania całego pliku. Zachowuje przy tym podział na linie/akapity, które otrzymujemy rozdzielone w tablicy łańcuchów string[].

**Exercise 2**

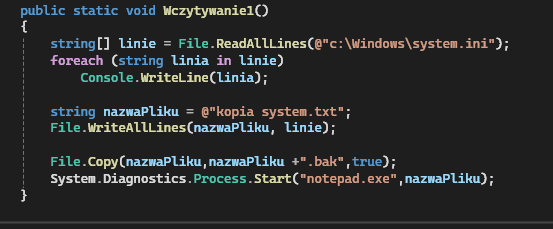
W zadaniu wczytujemy zawartość pliku i wyświetlamy go w konsoli.



**File.WriteAllLines** -to funkcja , która pozwala zapisać tablicę łańcuchów do pliku.

**Exercise 3**

Wczytujemy zawartość pliku wyświetlamy, kopiujemy do lokalnego folderu: (ze zmienioną nazwą). Kopiujemy plik bez zmiany jego zawartości File.Copy .Process.Start która pozwala na uruchomienie zewnętrznego programu tu uruchamiamy notatnik żeby pokazać skopiowany plik.



**Exercise 4 TRUDNE**

Użyć metodę ReadAllLines i WriteAllLines do zapisania i odczytania z plików tekstowych dwuwymiarowych tablic liczb rzeczywistych w formacie przypominającym CSV. Utworzyć dwie metody realizujące to zadanie: zapiszTablicę, i czytajTablicę Obydwie metody opierają się na użyciu podwójnych pętli for. W metodzie zapiszTablicę budowana jest w ten sposób jednowymiarowa tablica łańcuchów. Pętla zewnętrzna indeksowana zmienną j, odpowiedzialna jest za zbieranie kolejnych łańcuchów do tablicy, natomiast pętla wewnętrzna z indeksem i – za tworzenie linii ze skonwertowanych do łańcuchów wartości z jednego wiersza tablicy. Elementy w linii rozdzielane są znakiem podanym przez użytkownika. Domyślnie jest to średnik.

Kod główny (main):

*Console.WriteLine("Czytanie i zapisywanie do pliku");*

*double[,] tablica = new double[4, 8];*

*int licznik = 0;*

*for (int j = 0; j < tablica.GetLength(1); j++)*

*for (int i = 0; i < tablica.GetLength(0); i++)*

*tablica[i, j] = licznik++;*

*Tab7.zapiszTablice("tablica.csv", tablica);*

*Console.WriteLine();*

*double[,] tablica\_kopia = Tab7.czytajTablice("tablica.csv");*

*Console.WriteLine("Rozmiar tablicy: "+tablica\_kopia.GetLength(0)+"x"+tablica\_kopia.GetLength(1));*

*for(int j = 0;j< tablica\_kopia.GetLength(1); j++)*

*{*

*for (int i = 0; i < tablica\_kopia.GetLength(0); i++)*

*Console.Write(tablica[i, j].ToString() + "\t");*

*Console.WriteLine();*

*}*

Metody wywołane:

*static class Tab7*

*{*

*public static void zapiszTablice(string sciezkaPlik, double[,] tablica, char separator = ';')*

*{*

*int liczbaWierszy = tablica.GetLength(1);*

*string[] linie =new string[liczbaWierszy];*

*for(int j=0;j<liczbaWierszy;j++)*

*{*

*string linia = " ";*

*for(int i = 0; i < tablica.GetLength(0); i++)*

*{*

*linia += tablica[i, j].ToString() + separator;*

*}*

*linie[j] = linia.TrimEnd(separator);*

*}*

*File.WriteAllLines(sciezkaPlik, linie);*

*}*

*public static double[,] czytajTablice (string sciezkaPliku, char separator = ';')*

*{*

*string[] linie = File.ReadAllLines(sciezkaPliku);*

*int liczbaWierszy=linie.Length;*

*int liczbaKolumn = linie[0].Split(separator).Length;*

*double[,] tablica = new double[liczbaKolumn, liczbaWierszy];*

*for (int j = 0; j < liczbaWierszy; j++)*

*{*

*string[] elementy = linie[j].Split(separator);*

*for (int i= 0; i < liczbaKolumn; i++)*

*{*

*tablica[i, j] = double.Parse(elementy[i]);*

*}*

*}*

*return tablica;*

*}*

*}*

*}*

Dokumentacja

StreamReader odczytywanie pliku

StreamWriter zapisywanie tekstu do pliku

Poniższy kod używa klasy StreamReader do otwierania, odczytywania i zamykania pliku tekstowego. Aby automatycznie otworzyć plik, do konstruktora klasy StreamReader można przekazać ścieżkę dostępu do pliku tekstowego. Metoda ReadLine odczytuje każdy wiersz tekstu i w trakcie odczytu przesuwa wskaźnik pliku do następnego wiersza. Gdy metoda ReadLine dotrze do końca pliku, zwraca null.

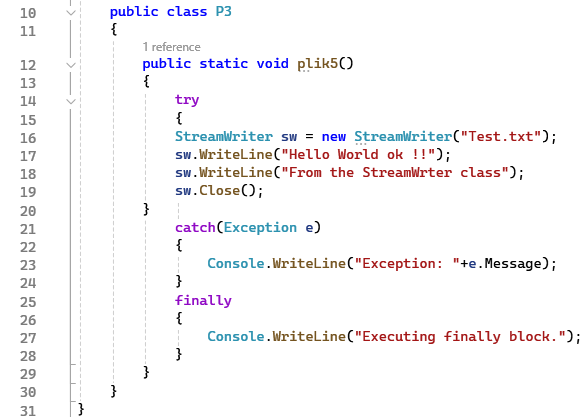
Exercise 5

Utwórz plik nazwij i napisz parę zdań. Potem przeczytaj linijkami zawartość używając metody ReadLine Wyświetl zawartość poprzez odpowiednią metodę (WriteLine())



**Exercise 6**

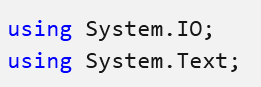
Zadeklaruj zmienną sw, która będzie wskazywała ścieżkę do pliku Tekst.txt. Wpisz dwie linijki i zamknij plik. Plik utworzy się w miejscu zapisu kodu źródłowego Linijki tekstu można odczytać po przez notatnik. W programie skorzystaj z try, catch, finally.

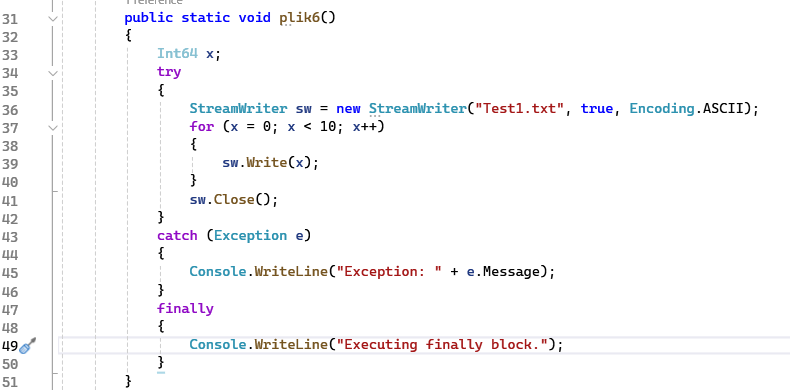


Metoda Write jest podobna do metody WriteLine, tylko nie przenosi kursora do następnej linii (powrót karetki). Jest ona przydatna , gdy zachodzi potrzeba pisania jednego znaku na raz.

**Exercise 7**

Poniższy program używa klasy StreamWriter do otwierania zapisywania i zamykania pliku tekstowego. W przeciwieństwie do poprzedniego przykładu ten kod przekazuje dwa dodatkowe parametry do konstruktora. Pierwszy parametr to ścieżka do pliku i nazwa. Drugi parametr, true, określa, że plik jest otwarty w trybie dołączenia. Jeśli jako drugi parametr zostanie określona wartość pliku – będzie nadpisywana przy każdym uruchomieniu kodu. Trzeci parametr określa Unicode, aby StreamWriter kodował pliki w formacie Unicode. Dla trzeciego parametru można również określić następujące metody kodowania: (ASC!!, Unicode, UTF7, UTF8). Plik powinien zawierać jedną linijkę tekstu: 0123456789.

Dodaj kod na początku pliku Class1.cs 



**Wybór pliku do otwarcia (openFileDialog)**

Aby pracować na dokumencie trzeba wybrać plik. Możliwość wybrania pliku będzie potrzebna zarówno w przypadku zwykłych plików tekstowych, jak i utworzonych w edytorze tekstu, arkuszy kalkulacyjnych i wreszcie plików graficznych. Utwórzmy nowy projekt o nazwie OpenFileDialogForm.





1. Formatka tak jak się uczyliśmy.

2. Definiujemy nowe pole tekstowe przez przypisanie do niego nowej instancji obiektu pola tekstowego. TextBox textFromFile = new TextBox();

3. Po utworzeniu pola tekstowego konfigurujemy jego właściwości: nazwę, położenie, szerokość, możliwość wpisania kilku linijek, ustawiamy paski przewijania w pionie i w poziomie wartość włączona. textFromFile.ScrollBars = ScrollBars.Both

Zawijanie znaków ustawiamy jako włączone: text.FormFile.WordWrap = false.

4. Dodajemy pole tekstowe do programu Dontrols.Add(textFormFile).

5. Tworzymy przycisk, przypisujemy zmienną do obiektu, który go reprezentuje. Ustawiamy szereg własności. np.: openFileDialog1Button.TabIndex = 1;czyli definiujemy pozycję numer dwa przycisku użycia w formularzu klawisza tabulatora.

6. Definiujemy metodę która ma się wywołać w momencie kliknięcia przycisku:

7. Dzięki try musimy sprawdzić czy w kodzie nie wystąpią błędy.

8. Definiujemy dwie zmienne będące ciągami znaków i przechowujące odpowiednio treści pliku:string selectedFileContent, oraz ścieżkę dostępu do niego: string selectedFilePath.

9. Używamy nowego obiektu otwierania pliku: using (OpenFileDialog openFileDialog = new OpenFileDialog())

10. Ustawiamy ścieżkę od której obiekt ma zacząć poszukiwania:

*openFileDialog.InitialDirectory = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);*

11.Ustawiamy filtr plików, które mają być brane pod uwagę. Ten filtr składa się z ciągu znaków, które mają zostać wyświetlone w oknie użytkownikowi, potem po znaku pionowej kreski znajduje się rozszerzenie pliku , następnie można kontynuować tę formę zapisu, czyli powtórzyć najpierw opis dla użytkownika, a następnie rozszerzenie pliku: openFileDialog.Filter = "Text files (\*.txt)| \*.txt|Allfiles (\*.\*)|\*.\*";

12. Ustawiamy filtr numer 1 jako domyślny: openFileDialog.FilterIndex = 1;

13. W przypadku gdy użytkownik otworzył nie ten plik o który mu chodziło, po ponownym kliknięciu ma zostać przywrócona ostatnia wybrana ścieżka dostępu, co znacznie ułatwi korzystanie z programu: openFileDialog.RestoreDirectory = true;

14. Dajemy możliwość otwarcia tylko jednego pliku: openFileDialog.Multiselect = false;

15. Następnie sprawdzamy, czy w oknie został kliknięty przycisk zatwierdzający: if (openFileDialog.ShowDialog() = DialogResult.OK).

16. W takim przypadku wpisujemy do zmiennej przechowującej ścieżkę do pliku i jego nazwę: selectedFilePatch = openFileDialog.FileName;

17. Następnie z użyciem obiektu do odczytu zawartości pliku czytamy wszystkie jego linie: using (StreamReader reader = new StreamReader(selectedFilePatch))

18. Teraz do zmiennej. W której ma się znaleźć zawartość pliku, czytamy go aż do końca: selectedFileContent = reader.ReadToEnd();

19.Wpisujemy do pola tekstowego zdefiniowanego na początku programu zawartość ze zmiennej zawierającej całą treść pliku: textFromFile,Text = selectedFileContent,

W przypadku błędów : *catch (Exception ex)*.

Wybór lokalizacji do zapisania pliku (SaveFileDialog)

Kiedy mamy już plik z wprowadzonymi przez nas zmianami, należałoby jeszcze zapisać te zmiany na dysku. Istnieją specjalne metody, które pozwolą wybrać miejsce przechowywania informacji w formie pliku. Utwórzmy nowy projekt o nazwie SaveFileDialogForm