

Fragmenty kodu biblioteki - interfejsy

Częściowy kod do zadania zlokalizowany jest w folderze `src/`.

Założenia ogólne

Nie dokonujemy modyfikacji kodu dostarczonych interfejsów.

W miarę możliwości odtwarzamy układ klas i ich składowanych przedstawiony na diagramie.

Jeśli pewne fragmenty nie zostały jednoznacznie wyjaśnione, samodzielnie podejmujemy decyzje projektowe, starając się osiągnąć główny cel - bezpieczne i poprawne zarządzanie plikami i folderami tymczasowymi.

Uwagi dla implementowanych klas:

- Konstruktor inicjuje (tworzy) wymagany zasób zewnętrzny (plik/katalog).
- Konstruktor bezargumentowy w klasach implementujących `ITempFile` wykorzystuje mechanizmy obsługi plików tymczasowych dostarczonych w bibliotekach C# (statyczna klasa `System.IO.Path` i metody `Path.GetTempFileName`, `Path.GetTempPath`, `Path.GetRandomFileName` i inne zlokalizowane w tej klasie).
- Jeśli nie podano ścieżki, plik tymczasowy tworzony jest folderze domyślnym dla plików tymczasowych dla profilu użytkownika (np. w Windows w `C:\Users\USER\AppData\Local\Temp\tmp35C7.tmp`).
- Konstruktor z argumentem będącym nazwą pliku (wraz z pełną ścieżką) tworzy we wskazanej lokalizacji plik tymczasowy.
- Konstruktor bezargumentowy w klasach implementujących `ITempDir` (a więc katalog tymczasowy) tworzy katalog w folderze domyślnym dla plików tymczasowych dla profilu użytkownika (zwyczajowo w Windows w `C:\Users\USER\AppData\Local\Temp\`); przy generowaniu losowej nazwy można posłużyć się np. strukturą GUID.
- Konstruktor z argumentem będącym pełną ścieżką dostępu do folderu tworzy we wskazanej lokalizacji folder tymczasowy.
- W przypadku podania - w konstruktorach wymagających parametru - niepoprawnej ścieżki, zgłoszany jest stosowny wyjątek.

Należy zadbać, aby:

- Przy tworzeniu obiektu reprezentującego plik/katalog tymczasowy **nie został zgłoszony wyjątek po jego fizycznym utworzeniu** - zasób zewnętrzny (niezarządzalny) został utworzony, a obiekt nie będzie istniał - może spowodować to wycieki pamięci lub destabilizację systemu operacyjnego (np. nie zamknięty plik).
- Przy używaniu obiektu reprezentującego plik/katalog tymczasowy **nie likwidować dostępu do zasobu** - spowoduje to zgłoszenie wyjątku `ObjectDisposedException`.
- Po zakończeniu korzystania z pliku/katalogu tymczasowego został on skutecznie i bezwarunkowo usunięty.

Należy obsłużyć ewentualnie pojawiające się wyjątki z `System.IO` (<https://docs.microsoft.com/en-us/dotnet/api/system.io.ioexception> (np. w sytuacji tworzenia katalogu podając jego nazwę, a katalog taki we wskazanej lokalizacji już istnieje)

⚠️ `ITempElement` można rozszerzyć o innego rodzaju elementy tymczasowe, np. tymczasowy obszar w pamięci RAM (`ITempMemory`) - jest to poza zakresem ćwiczenia.

Etapy realizacji zadania

Zadanie 1 - rozruch, prosta implementacja `IDisposable` i przesłonięcie `Object.Finalize`

- Utwórz `solution` z projektem `TempElementsLib` typu `class library` .Net Standard 2.1
- Dodaj do `solution` projekt `TempElementsConsoleApp` typu `console application` .Net Core 3.1
- Korzystając z notatek z wykładu oraz rozdziału z podręcznika *Essential C#* (rozdział Well formed type → Garbage collection, Resource cleanup) zaprogramuj klasę `TempFile`, implementującą `ITempFile` (w podręczniku oraz notatkach jest praktycznie pełen kod tej klasy).
- Obiekt typu `TempFile` przechowuje informacje o utworzonym (w konstruktorze) pliku tymczasowym oraz zarządza tym zasobem zewnętrznym:
 - plik tymczasowy jest traktowany jako strumień bajtów zmateriaлизowany na dysku

- plik tymczasowy otwarty jest w trybie read-write
- dostęp do fizycznego pliku poprzez pole `public readonly FileStream fileStream`
- dostęp do informacji o pliku poprzez pole `public readonly FileInfo fileInfo`
- do pliku można zapisywać tylko ciągi bajtów (np. `byte[]`) Aby do pliku binarnego wpisać tekst, możesz skorzystać z metody

```
public void AddText(string value)
{
    byte[] info = new UTF8Encoding(true).GetBytes(value);
    fileStream.Write(info, 0, info.Length);
    fileStream.Flush();
}
```

- wykorzystując obiekt `TempFile` można zarządzać w dowolny sposób utworzonym plikiem, dysponującą referencją `fileStream` (plik jest binarny, może więc reprezentować obraz, tekst, lub cokolwiek innego - po prostu strumień bajtów)
 - w programie konsolowym przetestuj działanie Twojej klasy, zbadaj zachowanie po utworzeniu obiektu (powinien pojawić się plik), po wykonaniu aktualizacji pliku, po opuszczeniu bloku `using` w którym wykorzystywany był plik
 - spróbuj jawnie zwolnić zasób wywołując metodę `Dispose`, a następnie ponownie odwołać się do tego pliku
 - spróbuj napisać kod wykorzystujący zasób, ale bez użycia instrukcji `using` - wykorzystaj `try-catch` i obsłuż stosowne wyjątki
 - sprawdź, jak działa instrukcja `using` w wersji C#8

Zadanie 2 - implementacja udogodnień, klasa `TempTxtFile`

Rozszerzasz klasę `TempFile` (w sensie formalnym - dziedziczenie), a raczej zwiększasz jej funkcjonalność do operowania na plikach tylko tekstowych. Celem jest ułatwienie obsługi tymczasowych plików tekstowych, poprzez udostępnienie metod takich jak `Write()` czy `ReadAllText()` bezpośrednio z poziomu klasy `TempFile`.

Skorzystasz ze strumieni "opakowujących": `StreamReader` oraz `StreamWriter`.

Pamiętaj o właściwym zamknięciu wszystkich strumieni przy wywołaniu metody `Dispose`.

Po wykonaniu zapisu do tekstuowego strumienia plikowego pamiętaj o opróżnieniu buforów (`Flush`).

Dodaj metody operujące na strumieniach: `ReadLine()`, `ReadAllText()`, `Write()`, `WriteLine()`,

Przetestuj działanie w aplikacji konsolowej.

Zadanie 3 - klasa `TempDir`

Postępując w podobny sposób jak w zadaniu 1, utwórz klasę `TempDir` implementującą interfejs `ITempDir` i przetestuj poprawność jej działania.

Zadanie 4 - kolekcja elementów tymczasowych

Bazując na interfejsie `ITempElements` utwórz klasę `TempElementsList`. Obiekt tej klasy przechowuje kolekcję elementów tymczasowych - np. różnego rodzaju plików tymczasowych, również folderów (formalnie zarządza elementami tymczasowymi, tworzy je, rejestruje ich lokalizacje i nazwy). Zrealizuj kolekcję jako listę tych elementów (`List<ITempElement>`).

Zgodnie z założeniami zapisanymi w interfejsie `ITempElements`, property `Elements` zwraca listę elementów typu `IReadOnlyCollection<ITempElement>`. Lista ta co prawda jest *read-only*, zatem nie można jej edytować, jednakże elementy listy mogą być edytowalne (lista przechowuje referencje).

Tworzenie obiektów tymczasowych realizowane jest za pomocą ich konstruktorów bezargumentowych (wymusza to bezargumentowa metoda generyczna `AddElement<T>()` oraz ograniczenie na typ generyczny `where T : new()`) - a więc w domyślnej lokalizacji plików tymczasowych systemu operacyjnego/profilu użytkownika.

Utworzony element można przesunąć do innej fizycznej lokalizacji metodą `MoveElementTo<T>(T element, string newPath)` (uwaga: można to zrobić również spoza kolekcji).

Element można usunąć (metoda `DeleteElement<T>(T element)`) - usuwamy fizycznie powiązany zasób oraz kasujemy sam element i jego rejestrację w liście.

Ponieważ elementy listy mogą być edytowalne, więc można również zwalniać zasoby z nimi związane działaniami spoza kolekcji. W takiej sytuacji metoda `RemoveDestroyed()` naprawia kolekcję, usuwając z niej nieaktualne wpisy i kompaktując kolekcję (skracając listę).

Kompaktując kolejkę (skracając listę):

Klasa implementuje `IDisposable` - wszystkie zarejestrowane w niej pliki i katalogi tymczasowe są bezwarunkowo usuwane w chwili zaprzestania korzystania z kolekcji lub zakończenia aplikacji.

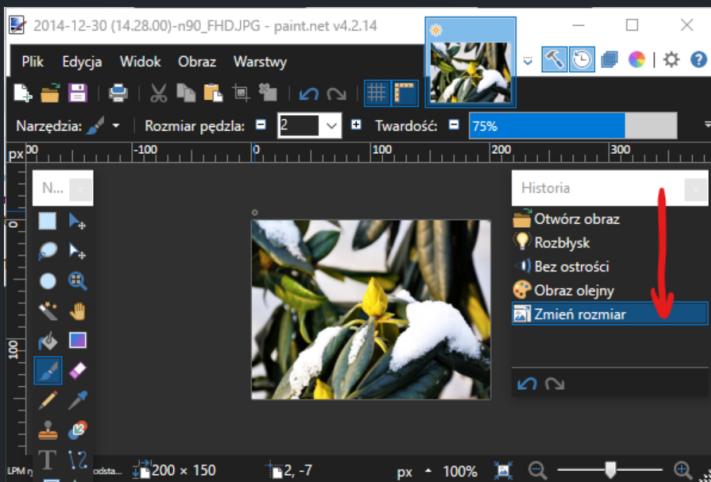
Przetestuj poprawność implementacji w aplikacji konsolowej.

Zadanie 5* - przykładowa aplikacja

Jeśli w implementacji w zadaniu 4 wykorzystasz `stos` jako kolekcję `Elements` (utworzysz np. `TempElementsStack`), w kolekcji tej możesz zapamiętywać historię zmian, zaś w plikach tymczasowych kolejne modyfikacje danych w Twojej aplikacji.

Zdejmując elementy ze stosu (np. metoda `T Pop<T>()` dodatkowo do zaimplementowania) będzie można wycofywać się z ostatnio wykonanych czynności.

Dobrym przykładem jest aplikacja do edycji plików graficznych, pamiętająca historię zmian i dającą możliwość wycofania się z nich nawet do punktu początkowego. W przypadku operowania na plikach o wysokiej rozdzielczości konieczne może być zapamiętywanie tych zmian właśnie w plikach tymczasowych (o dużych rozmiarach).



Zaproponuj aplikację demonstrującą działanie według powyżej podanej zasady (może być konsolowa, może być desktopowa, nie musi to być edytor graficzny - chodzi tylko o demonstrację działania historii zmian i przechowywania kolejnych modyfikacji danych w plikach tymczasowych).

Referencje

- [Working with Temporary Files in .Net](#)
- [How to use Temporary Files in C#](#)
- [C# Create temp file, write to it, print it, then delete it](#)
- [C# FileStream tutorial](#)



© 2022 GitHub, Inc.

[Terms](#)

[Privacy](#)

[Security](#)

[Status](#)

[Docs](#)

[Contact GitHub](#)

[Pricing](#)

[API](#)

[Training](#)

[Blog](#)

[About](#)