



Search or jump to...

Pull requests Issues Marketplace Explore

Watch 1 Fork 63 Star 1

wsei-csharp201 / cs-lab04-Implementacje-interfejsow-implicit-explicit-kompozycja

Public

Code

Issues

Pull requests

Actions

Projects

Wiki

Security

Insights

main

1 branch

0 tags

Go to file

Add file

Code

kmolenda Update README.md

0400033 on 19 Nov 2020 5 commits

.gitignore

Initial commit

16 months ago

ClassDiagram-ver1.png

Initial commit

16 months ago

ClassDiagram-ver2.png

Initial commit

16 months ago

Devices.cs

Initial commit

16 months ago

Documents.cs

Initial commit

16 months ago

README.md

Update README.md

16 months ago

UnitTestCopier.cs

Initial commit

16 months ago

README.md

## Zadanie. Implementacje interfejsów (*explicit, implicit*), kompozycja - Copier

- Krzysztof Molenda, ver. 01/2020.11.10

Dla wszystkich zadań utwórz jedno *solution*. Projekty nazywaj: `Zadanie1`, `Zadanie2`, ... . Będą to projekty typu *Console Application*.

Podobnie dla projektów typu *unit tests*, przyjmij nazwy `Zadanie1UnitTests`, `Zadanie2UnitTests`, ... .

Zadania wykonuj w podanej kolejności - są zależne od siebie.

Realizując to ćwiczenie, nabędziesz umiejętności modelowania obiektowego, uwzględniającego hierarchie interfejsów i klas, klasy abstrakcyjne, dziedziczenie, implementacje interfejsów (*explicit* i *implicit*), ... .

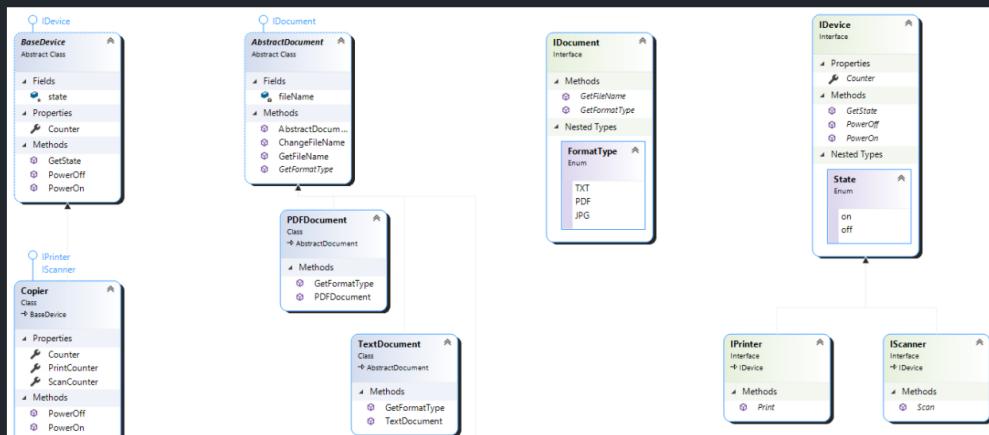
Zastosujesz relacje *is-a* oraz *has-a*, wzorzec delegacji, kompozycję zamiast dziedziczenia.

Orientacyjny czas realizacji ćwiczenia: od 60 do 90 minut.

⚠ UWAGA: jeśli jakieś założenie nie zostało precyzyjnie zdefiniowane, samodzielnie podejmij decyzje projektowe.

### Zadanie 1

Korzystając z załączonego kodu, diagramu klas oraz kodu testów jednostkowych zaprogramuj klasę `Copier` symulującą działanie kserokopiarki (kserowanie dokumentów), z możliwością niezależnego drukowania oraz skanowania dokumentów.



### About

Ćwiczenie C#: implementacje interfejsów (*implicit, explicit*), domyślne implementacje metod w interfejsach (C# 8), kompozycja zamiast dziedziczenia

Readme

1 star

1 watching

63 forks

### Releases

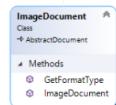
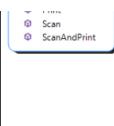
No releases published

### Packages

No packages published

### Languages

C# 100.0%



- Plik `Devices.cs`
- Plik `Documents.cs`
- Plik `UnitTestCopier.cs`

Przyjmij, że:

- `property PrintCounter` zwraca aktualną liczbę wydrukowanych dokumentów,
- `property ScanCounter` zwraca liczbę zeskanowanych dokumentów,
- `property Counter` zwraca liczbę uruchomień kserokopiarki
- Jeśli kserokopiarka jest wyłączona, wysyłanie do niej zadań nie ma żadnego skutku i żadnej reakcji z jej strony.
- Włączanie urządzenia, które jest już włączone, nie przynosi żadnego skutku. Analogicznie dla wyłączania urządzenia.
- Metoda `Print()` wypisuje na konsolę datę i godzinę wydruku, słowo `Print` oraz nazwę drukowanego dokumentu

Przykład: `15.11.2020 22:31:13 Print: aaa.pdf`

- Metoda `Scan()` wypisuje na konsolę datę i godzinę skanu, słowo `Scan` oraz nadaną nazwę pliku zeskanowanego dokumentu w formacie:
  - dla dokumentów typu `PDF` : `PDFScanXXXX.pdf`, gdzie XXXX oznacza numer skanowanego dokumentu od pierwszego uruchomienia kserokopiarki,
  - dla dokumentów typu `JPG` : `ImageScanXXXX.jpg`, gdzie XXXX oznacza numer skanowanego dokumentu od pierwszego uruchomienia kserokopiarki,
  - dla dokumentów typu `TXT` : `TextScanXXXX.txt`, gdzie XXXX oznacza numer skanowanego dokumentu od pierwszego uruchomienia kserokopiarki.

Przykład: `15.11.2020 22:31:13 Scan: ImageScan1.jpg`

- Metoda `ScanAndPrint()` wykonuje skanowanie (do `JPG`) i natychmiastowy wydruk zeskanowanego dokumentu.

Weryfikuj napisany kod za pomocą dołączonych testów jednostkowych.

W klasie `Program` i metodzie `Main()` napisz kod symulujący działanie kserokopiarki. Zobacz, jakie komunikaty tekstowe będą wypisywane na konsoli. Przykładowy kod:

```
class Program
{
    static void Main()
    {
        var xerox = new Copier();
        xerox.PowerOn();
        IDocument doc1 = new PDFDocument("aaa.pdf");
        xerox.Print(in doc1);

        IDocument doc2;
        xerox.Scan(out doc2);

        xerox.ScanAndPrint();
        System.Console.WriteLine( xerox.Counter );
        System.Console.WriteLine( xerox.PrintCounter );
        System.Console.WriteLine( xerox.ScanCounter );
    }
}
```

## Zadanie 2

Po rozwiązaniu zadania 1, wzorując się na istniejącym kodzie, dopisz i dołącz do hierarchii interfejsów i klas:

- interfejs `IFax`,
- klasę `MultifunctionalDevice` (urządzenie wielofunkcyjne) modelującą urządzenie łączące w sobie funkcjonalność drukarki, skanera oraz faxu.

Opracuj brakujące testy jednostkowe dla zaprojektowanej klasy.

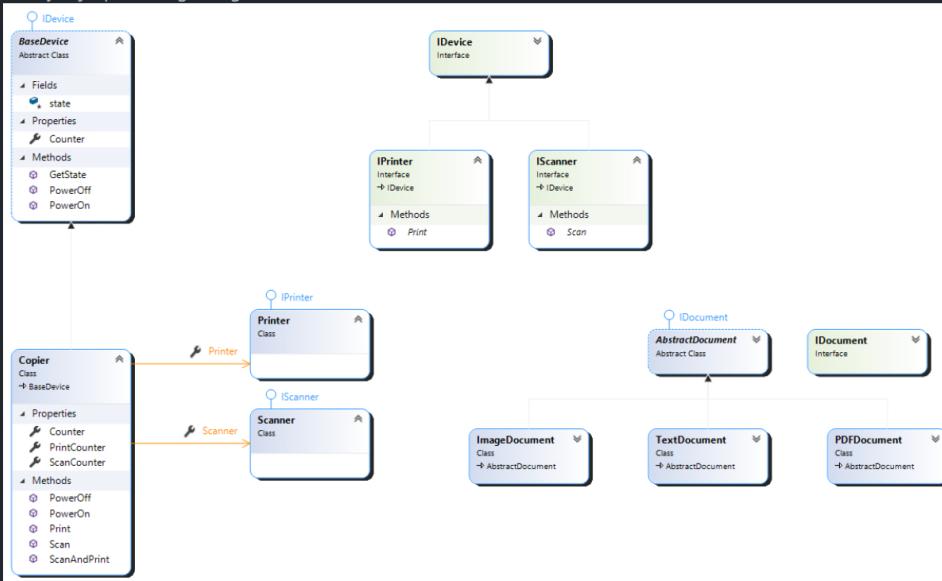
Zbadaj działanie swojego rozwiązania w programie konsolowym.

## Zadanie 3

Utwórz nowy projekt i przekopiuj kod z zadania 1 (bez klasy `Copier`).

Napisz ponownie klasę `Copier`, ale przyjmując następujące założenia:

- kserokopiarka jest *luźnym połączeniem* skanera i drukarki - jest urządzeniem, które składa się z dwóch innych urządzeń: skanera i drukarki, współpracującymi ze sobą (w poprzednim kroku kserokopiarka była urządzeniem o własności skanowania i drukowania),
- kserokopiarka "steruje" swoimi składnikami (np. włącza je oraz wyłącza - w zależności od sytuacji),
- funkcjonalności takie same jak w zadaniu 1,
- skorzystaj z poniższego diagramu:



- będziesz musiał utworzyć klasy konkretne `Printer` oraz `Scanner` (implementacje `IPrinter` oraz `IScanner`),
- dodaj klasę `MultidimensionalDevice` (uwzględniając faksowanie) - według tych samych zasad,
- przekopiuj, zmodyfikuj i dodaj brakujące testy jednostkowe.

Zbadaj działanie swojego rozwiązania w programie konsolowym.

## Zadanie 4(\*)

Zadanie zrealizuj w C# 8, uwzględniając możliwość domyślnej implementacji metod interfejsu.

Utwórz nowy projekt. Przekopiuj kod z zadania 1 (ten, po jego zrealizowaniu).

Planujesz zmodyfikować definicję urządzenia (`IDevice`) - urządzenie może być dodatkowo w stanie `standby` (oszczędzania energii).

Musisz:

- Dodać do `enum State` nowy stan o nazwie `standby` - ta korekta nie spowoduje pojawienia się błędów w kodzie.
- Dodać do `IDevice` dwie metody: `StandbyOn()` oraz `StandbyOff()` - ta modyfikacja spowoduje błędy.
- Dodać do `IDevice` metodę `abstract protected void SetState(State state);`.
- Zmodyfikować metody interfejsu `PowerOn`, `PowerOff`, `StandbyOn`, `StandbyOff` domyślnie je implementując w interfejsie, z wykorzystaniem `SetState`
  - narysuj diagram stanów dla kserokopiarki
    - mamy trzy stany,
    - przejścia między stanami definiują operacje `PowerOn`, `PowerOff`, `StandbyOn`, `StandbyOff`.
- Usuń klasę `BaseDevice`, kserokopiarka tylko implementuje `IPrinter` oraz `IScanner`.
- Wirtualnie kserokopiarka składa się z dwóch modułów - drukującego i skanującego. Ponieważ urządzenie ma być energooszczędne, moduł nieużywany jest w stanie `standby`

- przykładowo, jeśli drukujemy, to skaner jest w *standby* do momentu uruchomienia skanowania, a wtedy moduł drukujący wchodzi automatycznie w tryb *standby*; analogicznie w drugą stronę,
- możemy "ręcznie" (metody `StandbyOn()`, `StandbyOff()`) wprowadzić całe urządzenie (oba moduły) w odpowiedni stan,
- ponieważ drukarka musi długo się inicjować przed wydrukiem (np. rozgrzewać) przyjmijmy, że automatycznie przechodzi w tryb *standby* po wydrukowaniu 3 dokumentów (formalnie to przejście powinno nastąpić po upływie określonego czasu, ale teraz nie tworzymy aplikacji wielowątkowej); jeśli np. drukujemy pod rząd 5 dokumentów, to po pierwszych 3 drukarka wchodzi w *standby* po to, by za chwilę wzbudzić się i wydrukować dwa pozostałe dokumenty,
- analogicznie skaner - dla kolejnych 2 skanowań,
- narysuj diagram stanów dla tej logiki.

Zaimplementuj tę funkcjonalność bez tworzenia klas konkretnych `Printen` oraz `Scanner`. Możesz część implementacji przenieść do `IPrinter` oraz `IScanner`. W klasie `Copier` albo skorzystasz z domyślnych implementacji metod w interfejsach, albo je przesłoniiesz.

**UWAGA:** stan kserokopiarki (jako jednego urządzenia) będzie definiowany przez stany modułu drukującego i modułu skanującego:

- kserokopiarka jest w stanie `off`, jeśli oba jej składniki są równocześnie w stanie `off`,
- kserokopiarka jest w stanie `standby`, jeśli oba jej moduły są w równocześnie stanie `standby`,
- kserokopiarka jest w stanie `on` w pozostałych przypadkach.

Zbadaj działanie swojego rozwiązania w programie konsolowym.

## Zadanie 5(\*)

Uwzględniając korekty zrealizowane w zadaniu 4, w nowym projekcie zaimplementuj układ z Zadania 3 (kserokopiarka składa się z *luźno połączonych* drukarki i skanera).

Zbadaj działanie swojego rozwiązania w programie konsolowym.



© 2022 GitHub, Inc.

[Terms](#)

[Privacy](#)

[Security](#)

[Status](#)

[Docs](#)

[Contact GitHub](#)

[Pricing](#)

[API](#)

[Training](#)

[Blog](#)

[About](#)