Wyższa Szkoła Informatyki Stosowanej i Zarządzania
Grupa IO2

ALGORYTMY PRZETWARZANIA OBRAZÓW

(Dokumentacja Oprogramowania)

Autor:
Kacper Kaleta

Prowadzący:
dr hab. Korzyńska Anna

Warszawa 2019/2020

Spis Treści:

1 Wprowadzenie

Program przetwarzania obrazów, nakładania zniekształceń radiometrycznych oraz ich korekcji. Przygotowany w semestrze letnim 2020 r. w ramach ukończenia uczelnianych zajęć: Algorytmy Przetwarzania Obrazów.

Pełna nazwa projektu IO2_08:
Program prezentacji zasad przebiegu procesu wprowadzania i korekcji zniekształceń radiometrycznych z wykorzystaniem obrazów szaro-odcieniowych oraz ich fragmentów w postaci obrazowej a także tablic liczb.

Niniejszy dokument skupia się na budowie i strukturze oprogramowania pomijając opis funkcjonalności oraz praktycznych zastosowań aplikacji.

2 Wymagania Systemowe

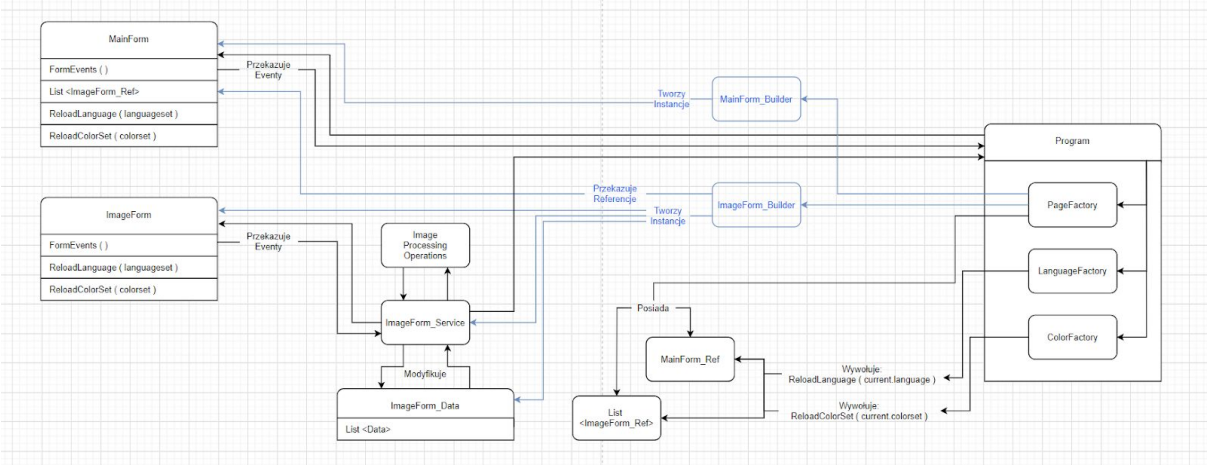
Uruchomienie programu wymaga dowolnego systemu operacyjnego Windows wspierającego framework .NET v4.7.2

3 Wykorzystane Narzędzia

- Visual Studio Community 2019 v16.5.4
- AutoMapper v9.0.0
- EmguCv v4.1.1.3497
- ZedGraph v5.1.7
- Kontrola wersji git w połączeniu z platformą github.com

4 Struktura Projektu

4.1 Spojrzenie Ogólne



Podstawowym celem powyższego projektu jest zapewnienie elastycznej struktury pozwalającej na bezproblemowe dodawanie kolejnych funkcjonalności do projektu.

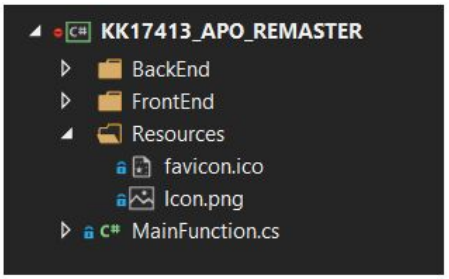
Struktura projektu wzoruje się architekturą *MVC*.
Można zaobserwować podobieństwa przypisując rolę *Modelu* do elementów klasy **Program**, odpowiedzialnym za operacje na obrazach oraz odpowiednią konfigurację aplikacji, rolę *Widoku* wszystkim klasą z katalogu FrontEnd - więcej informacji w podrozdziale [**Porządek wewnątrz projektu - elementy wizualne**],
oraz rolę *Kontrolera* klasie **ImageForm_Service**, odpowiedzialnej za obsługę wydarzeń (ang. Events) przesyłanych od użytkownika poprzez formularze.

W przeciwieństwie jednak do wzorca *MVC*, występujący wewnątrz tego projektu *Model* nie posiada dostępu do komunikowania się z częścią *Widoku*. Za to zadanie jest odpowiedzialna klasa **ImageForm_Service** której głównym, oraz jedynym, obowiązkiem jest obustronna komunikacja między implementacją FrontEnd'u a BackEnd'u.
ImageForm_Service jest klasyczną implementacją wzorca projektowego *Mediator*.

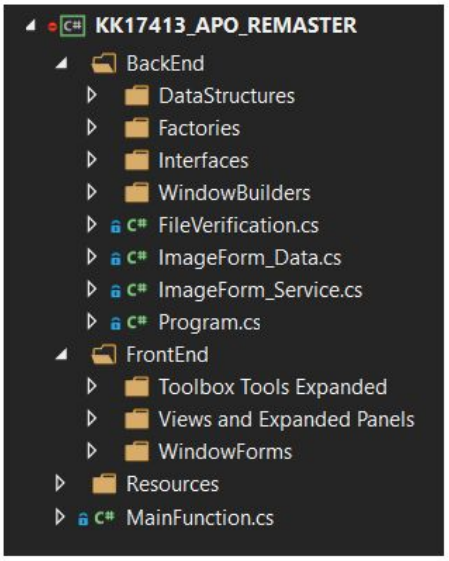
4.2 Porządek wewnątrz projektu - elementy wizualne

Porządek został zapewniony poprzez odseparowanie logiki graficznych elementów od reszty programu.

Formularze programu umożliwiają interakcję użytkownika z programem. Ich implementacje znajdują się wewnątrz katalogu **FrontEnd > WindowForms**.



Najbardziej skomplikowane formularze składają się z widoków (ang. Views), a te z kolei są zbudowane z grupy o najprostszej budowie, paneli.
Zarówno widoki jak i panele znajdują się w katalogu **FrontEnd > Views and Expanded Panels**.



Katalog **FrontEnd > Toolbox Tools Expanded** zawiera w sobie implementacje nowych, autorskich narzędzi oraz implementacje poszerzonych wersji już istniejących, wbudowanych wewnątrz standardu VisualStudioWindowsForms.
Znajdują się tam autonomiczne komponenty takie jak klasa **Histogram** lub rozszerzona wersja wbudowanej klasy formularza (klasa **Form**).

4.3 Porządek wewnątrz projektu - odseparowanie konstruktorów

Najczęściej wykorzystywanym wewnątrz tego projektu wzorcem projektowym jest Builder. Niemalże każda klasa niebędąca odpowiedzialna za trzymanie danych lub niebędąca fabryką posiada przynajmniej jedną implementację buildera.

Odseparowanie procesu konstruowania obiektów od samych obiektów pozwoliło na sprawniejszą nawigację między plikami projektu.
Jednocześnie wykorzystanie builderów ułatwiło ponowne wykorzystanie kodu. Najlepszy przykład takiej sytuacji zaobserwowałem podczas pisania nowego rodzaju okienka popup który miał jednocześnie pokazywać w pełni funkcjonalny wykres histogramu. Klasa Histogramu jest zbudowana z ponad 250 współdziałających ze sobą elementów. Posiadając wcześniej przygotowany builder histogramu nawet nie zajrzałem do jego implementacji. Nie przejmowałem się argumentami konstruktorów ani konfiguracją jego komponentów.
Tym wszystkim zajął się za mnie builder.

Co większe, bardziej złożone obiekty powstawały za pomocą wzorca ClassicBuilder.
W przypadku mniejszych, mniej wymagających wystarczało zastosowanie FluentBuildera.

4.4 Elastyczność projektu

Najważniejszym wymaganiem nałożonym na strukturę programu było umożliwienie dodawania dużej ilości nowych funkcjonalności przy minimalnym nakładzie pracy oraz minimalnej ilości modyfikacji do wprowadzenia.

Wymaganie to zostało spełnione za pomocą kilkukrotnie
wykorzystanego wzorca projektowego *Fabryki*.
Wewnątrz projektu znajduje się fabryka odpowiedzialna za operacje na obrazach, fabryka konstruująca system okienek popup, fabryka trzymająca mapy tłumaczeń językowych, fabryka trzymająca mapy motywów kolorystycznych programu.
Każda z fabryk działa niezależnie od reszty. Nie było potrzeby wykorzystywania wzorca Abstrakcyjnej Fabryki.
Największą zaletą takiego podejścia jest automatyzacja pisania nowego kodu.
Tworząc nową operację przetwarzania obrazu jedyne co musiałem zrobić to było napisać jej implementacje. Cała reszta odbywała się automatycznie bez potrzeby mojej ingerencji. Przykładowo bez potrzeby podpinania danej operacji do innych funkcjonalności lub informowania dowolnego innego miejsca w projekcie o jej istnieniu.

Wadą podejścia wykorzystującego wzorce Kreacyjne jakimi są Fabryki jest wzrost ilości kodu wewnątrz programu. W tym projekcie również jest to dostrzegalne, jest to jednak zbyt mały projekt aby mogło to stać się problemem.

5 Zebrane Statystyki O Projekcie

Ilość plików:	41
Ilość katalogów:	12
Ilość linijek napisanego kodu:	6 418
Ilość linijek kodu po kompilacji:	1 967
Ilość przeprowadzanych refaktoryzacji:	1
Ilość commitów na git:	248
Wykorzystane wzorce projektowe: - Fabryka - ClassicBuilder - FluentBuilder - Mediator - Wrapper	5
Czas powstawania projektu: od **22.03.2020** do **13.06.2020** 2 miesiące i 22 dni	83dni