|  |  |  |
| --- | --- | --- |
| Aleksandra Duda | Gr. 31 lab 02 | Techniki Obiektowe |
| Nr indeksu: 130510 | Laboratorium nr 1 | .10.20r |

**Sprawozdanie z labolatorium nr 1**

1. **Cel ćwiczenia**

Należało zaprojektować i zaimplementować narzędzie wspomagające

wymianę walut, korzystając z tabeli kursów średnich NBP.

1. **Przebieg ćwiczenia**

Dane do programu pobierano ze zdalnego repozytorium, a dokładniej z

<https://www.nbp.pl/kursy/xml/lasta.xml>.

Program miał realizować wybór waluty źródłowej, wybór waluty docelowej, wprowadzenie kwoty oraz obliczenie wartości w walucie docelowej.

W naszym pliku lastA.xml encją jest pozycja. Każda z pozycji zawiera nazwę waluty, przelicznik, kod waluty i kurs średni.

Aby zaimplementować narzędzie należało zaplanować wcześniej architekturę rozwiązania.

Podzieliłam program na trzy moduły, gdzie **Data.py** odpowiada za reprezentację i pobieranie danych, **BusinessLogic.py** odpowiada za logikę biznesową aplikacji, a **Interface.py** za interfejs użytkownika.

**Charakterystyka każdego z podziałów:**

* Reprezentacja danych i pobieranie ich ze zdalnego repozytorium realizują klasy takie jak: **Connect, Parsing, CurrencySet, Currency** w pliku **Data.py**

W klasie Connect łączę się ze zdalnym repozytorium lastA.xml, w klasie Parsing dokonuję parsowania tekstu, w klasie CurrencySet dokonuję odczytu ze słownika zbioru wszystkich pozycji, w klasie Currency pozyskuję dane każdej pozycji z osobna takie jak: nazwa, kurs średni, kod, czy przelicznik.



* W moim przypadku logika biznesowa aplikacji została zawarta w pliku **BusinessLogic.py** gdzie mamy klasę **Convert**, która jest odpowiedzialna za to co ma robić aplikacja – dokonać konwersji z waluty źródłowej na docelową.

Metoda get\_set() służy do pobrania sparsowanego zbioru pozycji, natomiast metoda get\_currency() służy do tworzenia obiektu klasy Currency.

Metody convert\_from\_pln I convert\_to\_pln służą do konwersji z i na polską walutę, żeby łatwiej było liczyć z waluty źródłowej (która jest inna niż polska) na walutę docelową (jeśli ta jest również inna niż polska). Zakładam, że podane wartości w pliku lastA.xml odnoszą się do polskiego złotego.

Klasa **Convert** korzysta z pliku gdzie jest reprezentacja danych, czyli z pliku **Data.py**



* Interfejs użytkownika jest realizowany za pomocą modułu **Interface.py**. za pomocą klasy **UserInterface.**

Interfejs korzysta z modułu logiki biznesowej. W konstruktorze jest tworzony obiekt klasy **Convert()** , która jest zaimplementowana w module **BusinessLogic.py.**

Metoda **questions()**  ma za zadanie zapytać użytkownika jaką ilość pieniędzy danej waluty chce przekonwertować na walutę docelową, pobrać odpowiedzi użytkownika i przekazać do metod modułu logiki biznesowej.



1. **Wnioski**

Do realizacji wyżej opisanego zadania zdecydowałam się użyć języka Python.

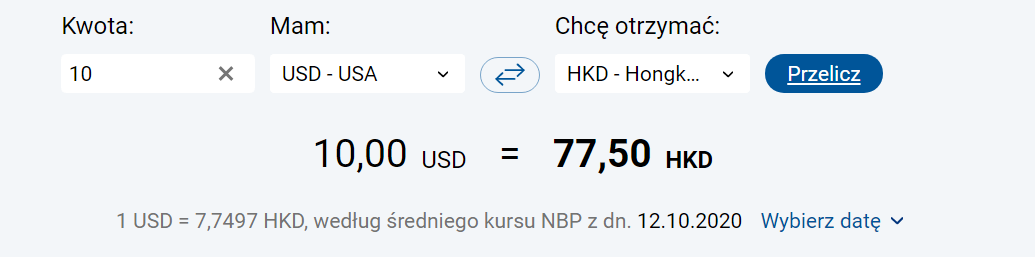
Uważam, że podział na reprezentację danych, logikę biznesową i interfejs użytkownika jest świetny i bardzo wygodny.

Dzięki temu jest większe prawdopodobieństwo, że ktoś kto będzie czytał nasz kod go zrozumie, dzięki temu, że każdy moduł realizuje określony typ zadań. Kod jest mniej chaotyczny.

Jest to też wygodne z tego względu, że jeśli aplikację pisałoby kilka osób to każda może się zając inną warstwą aplikacji, bo te są od siebie niezależne.

Myślę, że dawanie klasie jedną odpowiedzialność jest również dobrym rozwiązaniem. Dużo łatwiej w takim przypadku edytować kod, bo w przypadku konieczności doknania zmian nie wystąpią więcej niż jeden powód do modyfikacji. Implementacja dwóch różnych klas zamiast jednej , która realizuje dwie funkcje daje nam niezależność tych obu funkcji od siebie.

Porównując wartości z kallkulatora walut ze strony <https://www.money.pl/pieniadze/kalkulator/>, który korzysta z kursu NBP uzyskuję wynik:



Wpisując tą samą walutę źródłową i docelową oraz kwotę w swój program uzyskuję ten sam wynik:

