

Guido – Dokumentacja

Filip Uliasz

17 sierpnia 2023

Spis treści

1	Partytura	2
1.1	Zależności	2
1.2	Pola	2
1.3	Metody	2
2	Tonacja	4
2.1	Zależności	4
2.2	Pola	4
2.3	Metody	4
3	Akord	5
3.1	Zależności	5
3.2	Pola	5
3.3	Metody	5
4	Dźwięk	7
4.1	Zależności	7
4.2	Pola	7
4.3	Metody	7
5	Typy wyliczeniowe wykorzystane w bibliotece	9
5.1	‘NazwyDzwiekow’	9
5.2	‘WartosciNut’	9
5.3	‘Metrum’	10
5.4	‘Funkcje’	10
5.5	‘Przewroty’	10
5.6	‘BezwzgledneKodyDzwiekow’	10
6	Obsługa plików wejściowych	11
6.1	Format plików wejściowych	11
6.2	Obsługa wczytywania plików	12
6.3	Możliwe wyjątki	12

1 Partytura

1.1 Zależności

Klasa 'Partytura' importuje:

- klasę **Tonacja**
- klasę **Akord**
- klasę **Metrum** (jest to typ wyliczeniowy)

1.2 Pola

- `_tonacja` - przechowuje obiekt klasy `Tonacja`
- `_metrum` - przechowuje metrum. Jest to pole typu `Enum`
- `_liczba_taktow` - przechowuje liczbę taktów w partyturze
- `_lista_akordow` - przechowuje listę akordów w partyturze

1.3 Metody

- Konstruktor parametryczny

```
def __init__(self, nowa_tonacja: Tonacja, nowe_metrum: str,
             nowa_liczba_taktow): int
```

gdzie: `nowa_tonacja`, `nowe_metrum` i `nowa_liczba_taktow` są parametrami tworzonej partytury.

- Akcesor metrum:

```
def podaj_metrum(self) -> metrum.Metrum:
```

- Akcesor tonacji:

```
def podaj_tonacje(self) -> tonacja.Tonacja:
```

- Akcesor liczby taktów:

```
def podaj_liczbe_taktow(self) -> int:
```

- Metoda umożliwiająca dodanie taktu do partytury:

```
def dodaj_akord(self, nowy_akord: akord.Akord) -> None:
```

gdzie `nowy_akord` jest obiektem klasy `Akord`, który zostanie dodany na koniec listy akordów partytury.

- Metoda umożliwiająca zakończenie taktu:

```
def zakoncz_takt(self) -> None:
```

- Metoda, która sprawdza, czy liczba taktów jest poprawna:

```
def czy_poprawna_liczba_taktow(self) -> bool:
```

Metoda ta zwraca `True`, gdy liczba taktów w tablicy akordów jest taka sama jak zadeklarowana w konstruktorze i `False` w przeciwnym wypadku.

- Akcesor listy akordów:

```
def podaj_liste_akordow(self) -> List[Union[Akord, str]]:
```

który zwraca listę obiektów klasy `Akord` oraz liter `T`, którymi oznaczono koniec taktu w liście akordów.

2 Tonacja

2.1 Zależności

Klasa 'Tonacja' nie importuje żadnych innych klas.

2.2 Pola

- `_nazwa: str` - przechowuje nazwę tonacji
- `_czy_dur: bool` - przechowuje prawdę, jeśli tonacja jest durowa, lub fałsz, jeśli jest molowa.
- `_nazwy_dzwiekow_tonacji` - lista, w której znajdują się wszystkie stopnie właściwe dla pewnej tonacji
- `_lista_akordow` - przechowuje listę akordów w partyturze

W klasie 'Tonacja' przechowywane są również stałe:

- `_WSZYSTKIE_DUROWE_TONACJE` - lista wszystkich tonacji durowych znajdujących się w kole kwintowym
- `_WSZYSTKIE_MOLOWE_TONACJE` - lista wszystkich tonacji molowych znajdujących się w kole kwintowym
- `_SLOWNIK_DZWIEKOW_DUROWE` - słownik, w którym każdej tonacji z powyższej listy tonacji durowych przypisano właściwe jej stopnie
- `_SLOWNIK_DZWIEKOW_MOLOWE` - słownik, w którym każdej tonacji z powyższej listy tonacji molowych przypisano właściwe jej stopnie

2.3 Metody

- Konstruktor parametryczny

```
def __init__(self, nazwa_tonacji: str):
```

Konstruktor wypełnia pola klasy w zależności od podanej nazwy tonacji. W przypadku błędnej nazwy, podnoszony jest `ValueError`.

- Akcesor nazwy:

```
def podaj_nazwe(self) -> str:
```

- Akcesor trybu:

```
def podaj_tonacje(self) -> tonacja.Tonacja:
```

- Akcesor listy dźwięków:

```
def podaj_liste_nazw_dzwiekow(self) -> list[str]:
```

3 Akord

3.1 Zależności

Klasa 'Tonacja' importuje:

- Klasę dzwiek
- Klasę tonacja

Oraz wykorzystuje typy wyliczeniowe:

- WartosciNut
- Funkcje
- Przewroty

3.2 Pola

- `_dlugosc`: `WartosciNut` - przechowuje wartości nut (długość akordu jest jednakowa dla wszystkich głosów)
- `_sopran`: `Dzwiek` - przechowują dźwięki odpowiednio dla każdego z głosów
- `_alt`: `Dzwiek` - j.w.
- `_tenor`: `Dzwiek` - j.w.
- `_bas`: `Dzwiek` - j.w.

3.3 Metody

- Konstruktor parametryczny

```
def __init__(self, nowy_sopran: dzwiek.Dzwiek, nowy_alt: dzwiek.Dzwiek,
              nowy_tenor: dzwiek.Dzwiek,
              nowy_bas: dzwiek.Dzwiek, dlugosc:
              float):
```

Konstruktor przyporządkowuje odpowiednie dźwięki do odpowiednich pól. W przypadku, gdy `dlugosc` przyjmuje wartość spoza `Enuma`, zwracana jest `Funkcja.BŁAD`.

- Akcesory dźwięków:

```
def podaj_sopran(self) -> dzwiek.Dzwiek:
def podaj_alt(self) -> dzwiek.Dzwiek:
def podaj_tenor(self) -> dzwiek.Dzwiek:
def podaj_bas(self) -> dzwiek.Dzwiek:
```

- Metoda zwracająca funkcję danego akordu w odniesieniu do pewnej konkretnej tonacji:

```
def ustal_funkcje(self, dana_tonacja: tonacja.Tonacja) -> funkcje.Funkcja:
```

Metoda zwraca jedną z dopuszczalnych wartości typu wyliczeniowego Funkcja. Jeżeli dźwięki nie tworzą żadnej sensownej w danej tonacji funkcji, podnoszony jest

- Metoda zwracająca przewrót akordu:

```
def ustal_przewrot(self, dana_tonacja: tonacja.Tonacja) -> przewroty.  
    Przewrot:
```

Metoda zwraca jedną z dopuszczalnych wartości typu wyliczeniowego WartosciNut, przy czym wywołuje ona w swoim ciele metodę `ustal_funkcje`, by poznać funkcję. Jeżeli funkcja jest niepoprawna, niemożliwe jest określenie przewrotu akordu, w związku z czym zwraca się `Przewrot.NIE_ZDEFINIOWANO`.

4 Dźwięk

4.1 Zależności

Klasa 'Dźwięk' importuje:

- Klasę 'Tonacja'

Klasa wykorzystuje typy wyliczeniowe:

- 'NazwyDzwiekow', który służy do czuwania nad poprawnym nazewnictwem dźwięków
- 'BezwzględneKodyDzwiekow', używany przez metodę `Dźwięk.podaj_swoj_kod_bezwzględny()`.

4.2 Pola

- `_nazwa_dzwieku`: `NazwyDzwiekow` - przechowuje wartość typu Enum "NazwyDzwiekow"
- `_oktawa_dzwieku`: `bool` - przechowuje oktavę, do której dźwięk przynależy

4.3 Metody

- Konstruktor parametryczny

```
def __init__(self, nowa_oktawa_dzwieku: int, nowa_nazwa_dzwieku: str):
```

gdzie `nowa_oktawa_dzwieku` jest przypisywana do pola `_oktawa_dzwieku`, a `nowa_nazwa_dzwieku` służy do poprawnego wyboru odpowiedniej wartości z enum `NazwyDzwiekow`. Jeśli w Enum brak odpowiedniej wartości, konstruktor zwraca `ValueError`.

- Akcesor nazwy dźwięku:

```
def podaj_nazwe_dzwieku(self) -> str:
```

Zwraca wartość enuma, jest to typ `string`.

- Metoda zwracająca stopień dźwięku w pewnej tonacji:

```
def podaj_swoj_stopien(self, odpytywana_tonacja: tonacja.Tonacja) -> int:
```

gdzie `odpytywana_tonacja` jest typu 'Tonacja' i w zależności od niej podajemy stopień. Należy pamiętać, że dla prymy tonacji (np. dźwięk 'c' w tonacji 'C-dur') metoda zwróci 0, a dla septymy (np. dźwięk 'h' w tonacji 'C-dur') metoda zwróci 6.

- Metoda zwracająca względny kod dźwięku w tonacji:

```
def podaj_swoj_kod_względny(self, odpytywana_tonacja: tonacja.Tonacja) -> int:
```

Względny kod dźwięku jest wyliczany jako:

$$\text{względny kod dźwięku} = \text{numer oktawy} \cdot 7 + \text{stopień dźwięku w tonacji} \quad (1)$$

- Metoda zwracająca bezwzględny kod dźwięku w tonacji:

```
def podaj_swoj_kod_bezwzgledny(self) -> int:
```

Bezwzględny kod dźwięku jest niezależny od tonacji i reprezentuje pewną częstotliwość. Rekurencyjny wzór na bezwzględny kod dźwięku możemy wyrazić jako:

$$\begin{cases} x \sharp = x + 1 \\ x \flat = x - 1 \\ x = \text{oktawa}(x) \cdot 12 + \text{kod}(x) \end{cases}$$

gdzie:

$$\begin{cases} \text{kod}(x) = 0; & x = \text{dźwięk } c \\ \text{kod}(x) = 2; & x = \text{dźwięk } d \\ \text{kod}(x) = 4; & x = \text{dźwięk } e \\ \text{kod}(x) = 5; & x = \text{dźwięk } f \\ \text{kod}(x) = 7; & x = \text{dźwięk } g \\ \text{kod}(x) = 9; & x = \text{dźwięk } a \\ \text{kod}(x) = 11; & x = \text{dźwięk } h \end{cases}$$

oraz

$\text{oktawa}(x)$ jest numerem oktawy, w której dźwięk się znajduje (2)

5 Typy wyliczeniowe wykorzystane w bibliotece

5.1 ‘NazwyDzwiekow’

Elementami typu są wszystkie dźwięki gamowłaściwe występujące w tonacjach koła kwintowego. Nazwami elementów są nazwy dźwięków w języku polskim, a wartościami są oznaczenia przyjęte na potrzeby projektu (każde podwyższenie o pół tonu oznaczamy jako ‘#’, każde obniżenie to ‘b’). Są to:

Nazwa dźwięku w języku polskim	Kod na potrzeby projektu
ces	cb
c	c
cis	c#
cisis	c##
des	db
d	d
dis	d#
es	eb
e	e
eis	e#
fes	fb
f	f
fis	f#
fisis	f##
ges	gb
g	g
gis	g#
gisis	g##
as	ab
a	a
ais	a#
b	hb
h	h
his	h#

5.2 ‘WartosciNut’

Elementami tego typu są wartości (inaczej: długości trwania) nut. Wartości tego wyliczenia to liczby rzeczywiste. Na potrzeby niniejszego projektu ograniczono się do:

Nazwa wartości	Wartość liczbową
(cała) nuta	4.0
półnuta z kropką	3.0
półnuta	2.0
ćwierćnuta z kropką	1.5
ćwierćnuta	1.0
ósemka	0.5

5.3 'Metrum'

Elementami tego typu są metra (metrum), w których mogą występować partytury. Na potrzeby projektu ograniczono się do:

- 3/4
- 4/4

5.4 'Funkcje'

Elementami tego typu są funkcje harmoniczne (inaczej: rodzaje akordów). Na potrzeby tego typu ograniczono się do takich funkcji, jak:

- Tonika (moll Tonika)
- Subdominanta (moll Subdominanta)
- Dominanta
- Dominanta septymowa
- Błąd - element zwracany wówczas, gdy niemożliwe jest przyporządkowanie danych dźwięków do żadnego, przewidzianego w projekcie, typu.

5.5 'Przewroty'

Elementami tego typu są wszystkie możliwe przewroty akordów, w uwzględnionych w projekcie funkcjach, a zatem:

- Postać zasadnicza
- Przewrót pierwszy - tercja w basie
- Przewrót drugi - kwinta w basie
- Przewrót trzeci - septyma w basie. występuje tylko w przypadku dominanty septymowej
- Nie zdefiniowano - gdy podano błędną funkcję, nie można określić przewrotu.

5.6 'BezwzględneKodyDźwięków'

Elementami tego typu są nazwy dźwięków występujących w tonacji C-dur (bardziej obrazowo - białe klawisze fortepianu). Działanie typu opisano szczegółowo tutaj.

6 Obsługa plików wejściowych

6.1 Format plików wejściowych

Na potrzeby projektu, ustalono format danych wejściowych. Dane wejściowe są plikami tekstowymi (.txt). Pliki wejściowe dzielą się na nagłówek i ciało.

Nagłówek

- metrum - `metrum`: [3/4, 4/4]
- liczbę taktów - `takty`: [liczba naturalna]
- tonację - `tonacja`: [tonacje durowe i molowe z koła kwintowego]

Ciało W ciele pliku podaje się akordy. Akord jest uporządkowaną czwórką danych:

- dźwięk sopranu - para danych: dźwięk i cyfra symbolizująca oktawę, przy czym zamiast -is i -es korzystamy odpowiednio ze znaku # i b.
- dźwięk altu
- dźwięk tenoru
- dźwięk basu
- wartość akordu - podana jako liczba zmiennoprzecinkowa

Po podaniu wszystkich akordów mających znaleźć się w takcie występuje wielka litera "T" w nowej linii. W pliku wejściowym niedopuszczalna jest pusta linia, w której nie znajduje się żadna informacja. Będzie skutkowało to błędem i paniką programu.

Code Listing 1: Przykładowy poprawny plik wejściowy

```
metrum: 4/4
takty: 4
tonacja: C
c4, e3, g2, c1, 2.0
f3, c3, a2, f1, 1.0
g3, d3, h2, g1, 1.0
T
c4, e3, g2, c1, 2.0
f3, c3, a2, f1, 1.0
g3, d3, h2, g1, 1.0
T
c4, e3, g2, c1, 2.0
f3, c3, a2, f1, 1.0
g3, d3, h2, g1, 1.0
T
c4, e3, g2, c1, 2.0
f3, c3, a2, f1, 1.0
g3, d3, h2, g1, 1.0
T
```

6.2 Obsługa wczytywania plików

Do wczytania pliku wejściowego w powyższym formacie służy plik 'obsługa_pliku.py', w którym znajdują się trzy funkcje:

- Funkcja wywoływana w celu odczytania pliku. Wywołuje dwie poniższe metody.

```
def odczytuj_plik(sciezka_do_pliku: str) -> partytura.Partytura:
```

- Funkcja, która odczytuje nagłówek pliku

```
def utworz_partyture(plik: TextIO) -> partytura.Partytura:
```

- Funkcja, która wypełnia nowoutworzony obiekt akordami

```
def wypelnij_partyture_akordami(plik: TextIO, nowa_partytura: partytura.  
                                Partytura) -> partytura.Partytura:
```

6.3 Możliwe wyjątki

- 'BładWNaglowku' - podnoszony przez funkcję `utworz_partyture()`, gdy błąd zwraca funkcja `utworz_partyture()` lub kiedy nie utworzono partytury
- 'BładWCiele' - podnoszony przez funkcję `wypelnij_partyture_akordami()`, gdy błąd jest zwracany przez funkcję
- 'ValueError("Niepoprawne wczytanie danych")' - zwracany przez funkcję `odczytuj_plik`, kiedy wystąpił któryś z powyższych wyjątków.