Paradygmaty Programowania
Laboratorium 5

Programowanie zdarzeniowe

Sprawozdanie proszę wysłać przez platformę Teams lub na adres mailowy podany przez prowadzącego zajęcia (wyłącznie w przypadku problemów z oprogramowaniem Teams). Sprawozdanie powinno zawierać **imię i nazwisko**, **numer albumu** i **datę**.

Cel ćwiczenia

Celem ćwiczenia jest zaznajomienie studentów z przykładową implementacją wzorca architektonicznego Model-Widok-Kontroler (MVC) w języku Python z użyciem biblioteki Qt.

Przebieg ćwiczenia

Ćwiczenie przebiega według następującego harmonogramu:
— zapoznanie się z treścią punktowanych zadań i ewentualne wyjaśnienie kwestii niezrozumiałych,
— wspomagane przez prowadzących samodzielne wykonanie zadań.

Uruchamianie skryptów

Skrypty można uruchamiać:

1. na serwerze *lab09011.elka.pw.edu.pl*,
2. na własnym komputerze z interpreterem języka Python.

Skrypt wykorzystywany w tym ćwiczeniu korzysta z pakietu PySide2. Umożliwia on używanie z poziomu języka Python biblioteki Qt, pozwalającej m.in. na łatwe tworzenie aplikacji z graficznym interfejsem użytkownika (GUI). W przypadku braku pakietu PySide2 (lub innego), można go doinstalować, wydając polecenie:

```
pip3 install nazwa_pakietu
```

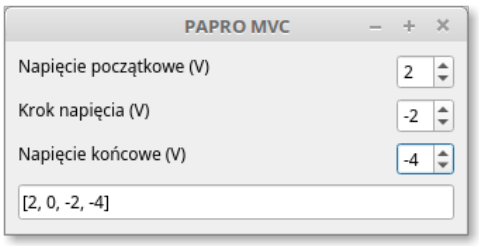
Na serwerze *lab09011* instalacja pakietów jest możliwa wyłącznie na naszym własnym koncie, co wymaga wydania polecenia:

```
pip3 install nazwa_pakietu --user
```

Zadania do wykonania

Wstęp

Proszę uruchomić skrypt *zad1.py*. Otworzy się okno jak na poniższym rysunku:



Skrypt emuluje działanie generatora sekwencji napięć, podobnego do tych wykorzystywanych w symulatorach układów elektronicznych czy w aplikacjach sterujących przyrządami pomiarowymi. Sekwencja taka jest ciągiem arytmetycznym, w którym użytkownik definiuje wartość początkową, krok oraz wartość końcową. Parametry te są zadawane za pomocą pól tekstowych z pokrętłami (ang. *spinboxes*). Zmiana dowolnego z tych parametrów powoduje natychmiastową generację nowej sekwencji, która jest – również natychmiast – wyświetlana w polu tekstowym (ang. *line edit*) w dolnej części okna.

W aplikacji można wyróżnić cztery klasy:

— `PaproModel` – model przechowujący stan obiektu, którym w tym przypadku jest sekwencja zaimplementowana jako lista liczb całkowitych. Lista ta jest atrybutem prywatnym, co sugeruje nazwa rozpoczynająca się od podkreślenia: `_sekwencja`. Z tego względu konieczne stało się zdefiniowanie metod dostępowych: `zwróć_sekwencję` ("gettera"), służącej do odczytu i `zmień_sekwencję` ("settera"), służącej do przypisania nowej zawartości.

— `PaproKontroler` – kontroler, pozwalający użytkownikowi na modyfikację modelu. Dokonuje też weryfikacji wzajemnej relacji wartości początkowej, końcowej i kroku. Jeśli relacje pomiędzy wartościami tych parametrów odpowiadają ciągowi o przynajmniej jednym elemencie, ciąg taki jest generowany. Jeśli nie, zwracana jest wartość pusta, w języku Python oznaczana jako `None`. W obu przypadkach nowy ciąg jest zapisywany w modelu jako `_sekwencja`.

— `PaproWidok` – widok. Elementy graficznego interfejsu użytkownika są wczytywane z pliku *papro5.ui*. Ma on postać dokumentu XML, którego zawartość można obejrzeć (a nawet zmodyfikować) albo w dowolnym edytorze tekstu, albo w programie *QtCreator*, dostępnym w laboratorium a także w repozytoriach większości dystrybucji Linuxa. Użycie programu *QtCreator* zapewnia podgląd wpływu wprowadzanych zmian na wygląd aplikacji. Oprócz tego w klasie `PaproWidok` zdefiniowany jest przepływ komunikatów między poszczególnymi elementami modelu MVC – więcej na ten temat poniżej.

— `PaproMVC` – klasa główna, w skład której wchodzą obiekty pozostałych trzech klas.

W przedstawionym skrypcie wykorzystano dostępny w bibliotece Qt mechanizm **sygnałów i slotów**, będący jedną z możliwych implementacji programowania zdarzeniowego. Ko-

rzystanie z tego mechanizmu jest szczególnie proste i przejrzyste w języku Python. Elementy GUI zdefiniowane w bibliotece Qt **emitują** charakterystyczne dla siebie **sygnały** w odpowiedzi na działania użytkownika. Przykładem mogą być użyte w naszym skrypcie obiekty klasy `QSpinBox` (pola z pokrętłami), które po każdej zmianie zawartości emitują sygnał `valueChanged`. Fakt emisji tego sygnału nie widoczny w kodzie, bo decyduje o niej dopiero użytkownik podczas wykonania skryptu. Jeśli skrypt ma reagować na konkretny sygnał emitowany przez dany element GUI, należy powiązać z tym sygnałem funkcję zwaną **slotem**, która będzie *automatycznie* wywoływana po każdorazowej jego emisji. Powiązanie takie ma postać:

```
obiekt_emitujący.sygnał.connect(slot)
```

Użytkownik może także definiować własne sygnały, niezwiązane z żadnym elementem GUI. Dzięki temu można zapewnić np. automatyczną reakcję widoku na zmianę wartości pewnych atrybutów modelu. Tak zdefiniowany sygnał jest obiektem klasy `pyqtSignal`, a do jego emisji służy metoda `emit`. Obsługa takiego sygnału przebiega w sposób standardowy, tj. poprzez powiązanie z nim odpowiedniego slotu. Mechanizm definiowania własnych sygnałów jest również wykorzystany w naszym skrypcie.

Zadanie 1 – 4 pkt

Proszę zapoznać się z kodem źródłowym skryptu *zad1.py*. Następnie narysować diagram klas ilustrujący zależności pomiędzy poszczególnymi klasami i ich składowymi z uwzględnieniem sygnałów i slotów.

Zadanie 2 – 2 pkt

Korzystając z programu QtCreator (lub modyfikując plik *papro5.ui* w edytorze tekstowym i posiłkując się dokumentacją do pakietu PySide), zmodyfikować interfejs użytkownika, dodając do niego przynajmniej jeden element. Może to być np. przycisk (obiekt klasy `QPushButton`), którego wciśnięcie będzie wymagane do zapisania w modelu nowej sekwencji – na razie jest ona generowana na nowo natychmiast po zmianie dowolnego z jej parametrów. W skrypcie proszę wprowadzić zmiany pozwalające na działanie tak zmodyfikowanego GUI zgodnie z intencjami autora. W sprawozdaniu krótko opisać zmiany i spodziewane działanie nowej wersji programu, warto także zamieścić zrzut ekranu pokazujący wygląd zmodyfikowanej aplikacji. **Kod zmodyfikowanego skryptu oraz plik z GUI dołączyć do sprawozdania, tak by można było przetestować ich działanie.**

Dokumentację "widgetów" Qt, czyli elementów GUI, można znaleźć np. tutaj:
`https://doc.qt.io/qtforpython/PySide6/QtWidgets/index.html#module-PySide6.QtWidgets`