Komunikator

Cel zajęć: Napisanie prostego, okienkowego komunikatora z wykorzystaniem poznanych na wcześniejszych zajęciach mechanizmów komunikacji sieciowej.

Wstęp:
Do zaimplementowania GUI komunikatora wykorzystamy bibliotekę tkinter, która pojawiła się już na zajęciach w poprzednim roku. Do komunikacji wykorzystamy mechanizm socketów z poprzednich zajęć. Projekt będzie składał się z kilku klas/modułów odpowiedzialnych za wybrane funkcje programu.

TODO:

I Moduł GUI – moduł odpowiedzialny za warstwę graficzną aplikacji

(Poniższa instrukcja stanowi tylko wskazówkę jak można podejść do tematu. Własne pomysły na rozwiązanie problemu są mile widziane i zostaną odpowiednio docenione)

1. Zapoznać się z: http://python-textbok.readthedocs.io/en/1.0/Introduction_to_GUI_Programming.html
2. Stworzenie klasy Gui dziedziczącej po klasie *Frame* (klasa z biblioteki *tkinter*)
3. Dodanie widgetu *Text* odpowiedzialnego za wyświetlanie wiadomości przychodzących i wysyłanych.
4. Dodanie widgetu *Text* wyświetlającego wpisywaną przez użytkownika wiadomość.
5. Dodanie przycisku „Wyślij” i oprogramowanie go w odpowiedni sposób (patrz punkt 5).
6. Implementacja metody, która zostanie wywołana po naciśnięciu przycisku Wyślij. Metoda powinna:
    a. - pobrać wiadomość z odpowiedniego okienka a następnie wyczyścić jego zawartość
    b. - wyświetlić wiadomość w odpowiednim okienku
    c. - przekazać wiadomość do modułu komunikacyjnego (dodać dopiero po napisaniu wspomnianego modułu)

7. Implementacja metody odpowiedzialnej za cykliczne odpytywanie modułu komunikacyjnego o nowe wiadomości – wykorzystać metodę *after*(1, …) z biblioteki *tkinter*.
8. Testowe stworzenie instancji tak napisanej klasy, wywołanie odpowiednich metod i sprawdzenie czy działa.

Definition Of Done:

1. Okienko wyświetla się po uruchomieniu apliacji.
2. Wymagane formatki i przyciski dodane oraz oprogramowane

3. Okienko do wyświetlania wiadomości jest niemodyfikowalne dla użytkownika.
4. Po naciśnięciu przycisku „Wyślij” wiadomość jest wyświetlana w odpowiednim miejscu.

Wskazówki:

Zablokowanie możliwości edycji widgetu *Text*:

```
textForm.config(state=DISABLED)
```

(gdzie *textForm* to nazwa zmiennej)

Odblokowanie możliwości edycji widgetu *Text*:

```
textForm.config(state=NORMAL)
```

Podpięcie metody pod przycisk:

```
button = Button(self.parent, text="Nazwa przycisku",
command=lambda:jakas_metoda())
```

Dodanie element do okienka:

```
element.pack()
```

II Moduł komunikacyjny – moduł odpowiedzialny za odbieranie i wysyłanie wiadomości z wykorzystaniem mechanizmu socketów.

1. Zapoznać się z: https://pymotw.com/2/socket/tcp.html
2. Stworzyć klasę, która odpowiedzialna będzie za wysyłanie i odbieranie wiadomości.
3. W zależności czy program ma działać jako serwer/klient zainicjalizować odpowiednie mechanizmy komunikacyjne.
4. Zaimplementować metodę *listen()* która będzie cyklicznie wywoływana z poziomu *gui*. Metoda ma sprawdzać czy przyszła jakaś wiadomość i przekazywać ją do *gui*.
5. Zaimplementować metodę *send*(…), która będzie wysyłać wiadomość otrzymaną od *gui*.

Definition Of Done:

1. Komunikacja działa.
2. Aplikacja nie zawiesza się.

Wskazówki:

Nieblokujące odczytywanie wiadomości:

```
connection.setblocking(0)
lub
socket.setblocking(0)
```

(w bloku try/catch)

III Moduł startowy (Main) – moduł odpowiedzialny za stworzenie powyższych modułów i zainicjalizowanie ich.

W powyższym podejściu oba moduły „znają się” wzajemnie. W tym nieidealnym podejściu występuje taki problem, że w momencie tworzenia jednego modułu chcielibyśmy mieć już stworzony drugi, aby móc przekazać go w konstruktorze. Rozwiązaniem tego problemu jest:

Moduł 2 przyjmuje instancję Modułu 1 jako argument w konstruktorze.

Moduł 1 posiada metodę, która w argumencie ma instancję Modułu 2 i ustawią ją jako pole.

Kolejność działania:

- utworzenie instancji Modułu 1
- utworzenie instancji Modułu 2 i przekazanie instancji Modułu 1 jako argumentu
- wywołanie metody w Module 1 ustawiającej instancję Modułu 2 jako pole tego modułu