

Sprawozdanie - projekt z programowania w języku Java

Grzegorz Bujak Arkadiusz Markowski grupa 2ID11B
Politechnika świętokrzyska w Kielcach

Spis treści

Założenia projektu	1
Moduł sieciowy	1
Edytor tekstu	2
Rzysyłane wiadomości	2
Produkowanie wiadomości przez edytor	2
Praca na wielu dokumentach	3

Założenia projektu

Nasz projekt polegał na napisaniu edytora tekstu w języku Java z wykorzystaniem biblioteki JavaFX. Edytor ma funkcję pisanie tego samego dokumentu przez wiele osób w tym samym czasie. Podczas pisanie, synchronizuje dokument pomiędzy użytkownikami.

Moduł sieciowy edytora używa topologii klient-serwer, ale serwer nie jest aplikacją terminalową. Interfejs graficzny zapewnia możliwość dołączenia do serwera oraz hostowania własnego dokumentu.

Moduł sieciowy

Moduł sieciowy naszego projektu składa się z klas:

- **Networker<T extends Serializable>**

Jedna instancja dla serwera i klienta. Jeśli program działa w trybie serwera, tworzy instancję **ServerSocket**, która ciągle akceptuje nowych klientów. W trybie klienta, tworzy **Dispatcher<T>**, który będzie wysyłał wiadomości tylko do serwera oraz **SocketHandler<T>**, który będzie odbierał wiadomości od serwera.

- **Dispatcher<T extends Serializable>**

Klasa odpowiada za wysyłanie wiadomości.

Przechowuje listę wszystkich wiadomości (obiekty klasy **T**) oraz **HashMap<Socket, Integer>**, który przechowuje sockety, do których ma wysyłać wiadomości oraz **Integer** będący ID ostatniej wiadomości wysłanej do socketu.

Zapewnia metodę **void addAndDispatch(T)**, która dodaje wiadomość typu **T** do listy wiadomości i wysyła tą wiadomość do wszystkich socketów.

Ma też metodę `void addSocket(Socket)`, która dodaje `Socket` do mapy i wywołuje metodę `dispatch()`, która wyśle temu socketowi wszystkie wiadomości dodane do `Dispatchera` od początku działania programu.

- `SocketHandler<T extends Serializable>`

Klasa jest odpowiedzialna za odbieranie wiadomości. Do jej konstruktora należy podać domknięcie (obiekt interfejsu funkcjonalnego `Consumer<T>`). Obiekt w nieskończoność odbiera nadchodzące wiadomości i wywołuje domknięcie z każdą wiadomością.

Wszystkie klasy powinny działać na własnych wątkach. Nie zapewniają możliwości zamknięcia, więc wątki powinny być ustawione jako `Daemony`.

Edytor tekstu

Edytor tekstu to lista `ListView` bloków tekstowych.

Bloki tekstowe to obiekty naszej własnej klasy `ExpandingTextArea`. Jest to `TextArea`, która zmienia swoją wysokość podczas pisania.

Wciśnięcie klawisza `enter` nie powoduje wstawienia nowej linii w polu tekstowym, lecz jest przechwytywane przez klasę `MainGuiController`, która wstawia do listy nowe pole tekstowe po tym, w którym znajduje się kursor oraz kopiuje do niego wszystko co znajdowało się za kursorem w polu tekstowym.

Wciśnięcie `backspace`, gdy kursor znajduje się na początku pola tekstowego powoduje skasowanie tego pola tekstowego i skopiowanie jego zawartości za kursorem do poprzedniego pola tekstowego.

Rozsyłane wiadomości

- pochodne klasy `DocumentAction`.
 - `AddBlockAction` - dodanie nowego bloku w edytorze
 - `ChangeBlockAction` - zmiana tekstu w bloku
 - `RemoveBlockAction` - usunięcie bloku

Pochodne tej klasy mają metody:

- `void commit(Document)` - do wprowadzania zmian w edytowanym w tej chwili dokumencie
- `void commit(List<SerializableBlock>)` - do wprowadzania zmian w dokumencie znajdującym się w postaci serializowanej w `Datastore`

Produkowanie wiadomości przez edytor

Podczas pisania, edytor generuje wiadomości. Są to obiekty klasy `DocumentAction`. Tutaj opisaliśmy warunki generowania wiadomości:

- zmiana zawartości pola tekstowego
 - > wygenerowanie wiadomości `ChangeBlockAction`
- usunięcie bloku
 - > wygenerowanie wiadomości `RemoveBlockAction`
 - > wygenerowanie wiadomości `ChangeBlockAction` dla bloku przed usuniętym na wypadek przeniesienia do niego tekstu z usuniętego
- dodanie bloku
 - > wygenerowanie wiadomości `AddBlockAction`

-> wygenerowanie wiadomości `ChangeBlockAction` dla bloku, w którym znajdował się kursor na wypadek przeniesienia z niego tekstu do nowego bloku

Nie jest możliwe przeniesienie tekstu do następnego bloku, dlatego żadna akcja nie generuje wiadomości `ChangeBlockAction` dla bloku znajdującego się pod aktywnym.

Praca na wielu dokumentach

TODO!