mgr inż. Adam Kozak
2013

Projektowanie obiektowe i programowanie w Javie

Specyfikacja projektów zaliczeniowych

Wymagania ogólne:

1. Projekty powinny spełniać wymogi opisane w opisie wymagań
2. Kod programu powinien być:
   a) sformatowany,
   b) zgodny z konwencjami nazewnictwa w Javie,
   c) opatrzony w komentarze JavaDoc dla wszystkich klas, metod oraz publicznych pól statycznych finalnych,
   d) zaprojektowany w sposób obiektowy, logika aplikacji powinna być oddzielona od interfejsu użytkownika,
   e) interfejs użytkownika powinien być napisany w Swingu - ręcznie lub przy użyciu dowolnego edytora GUI
3. Dla klas implementujących logikę powinny być przygotowane przynajmniej dwa testy jednostkowe JUnit.
4. Projekt powinien mieć przygotowany skrypt Ant budujący wykonywalny plik JAR oraz dokumentację JavaDoc oraz target wykonujący testy jednostkowe.
5. (* dla chętnych) W dokumentacji powinien być diagram klas w UML.

Są dwa projekty: JCommander i Komunikator internetowy.

mgr inż. Adam Kozak
2013

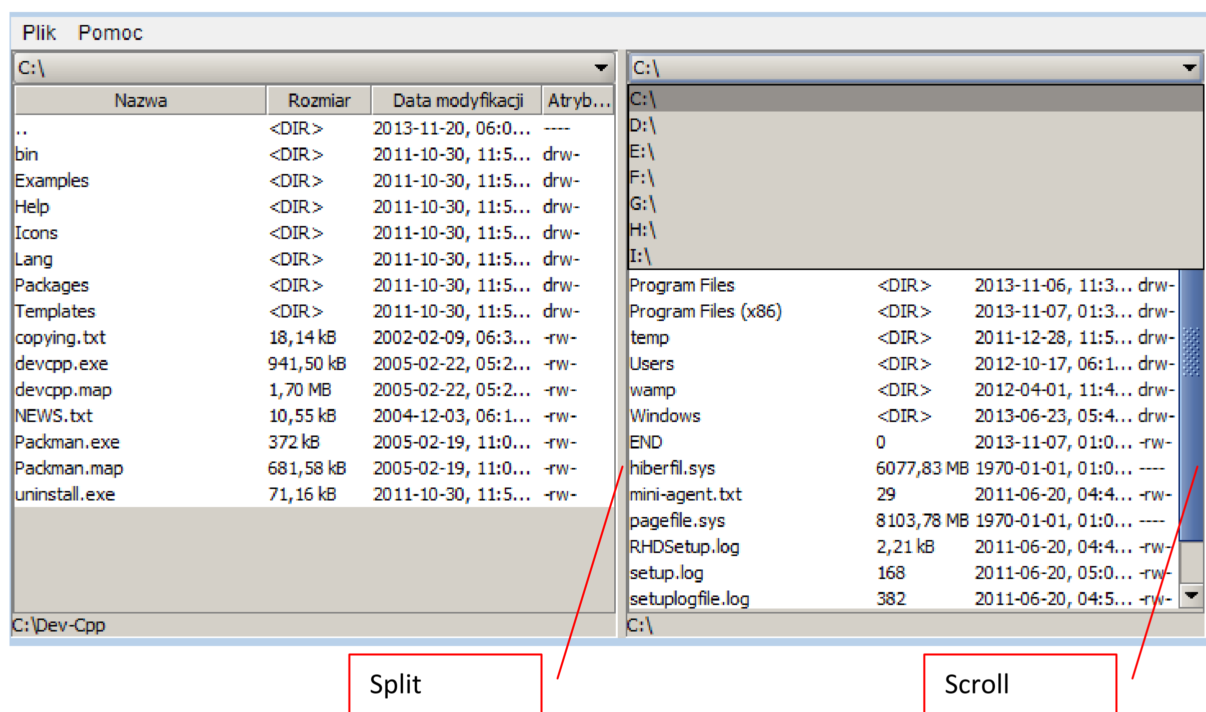
JCommander

Celem zadania jest zaprojektowanie i wykonanie aplikacji JCommander na wzór Total Commander, która zapewnia możliwość wykonywania podstawowych operacji na plikach działając na dwóch oknach.

Interfejs użytkownika powinien się składać z:

1. Dwóch paneli służących do przeglądania folderów i plików, które powinny:
    a. zawierać listę dysków dla Windows lub / dla linuxa (metoda java.io.File.listRoots)
    b. wskazywać bieżącą ścieżkę
    c. być niezależne od reszty aplikacji i reużywalne tj. powinny być napisane w osobnej klasie, która jest gotowym komponentem możliwym do wykorzystania także w innych aplikacjach - powinna jednak wystawiać listener aktualnej selekcji i metodę ustawiającą początkowy katalog do wyświetlenia (opcjonalnie możliwość ustawienia innych parametrów komponentu w zależności od fantazji)
2. Przycisków lub menu w którym byłaby możliwość wykonania operacji Kopiuj i Przenieś między panelami:
    a. w przypadku istnienia pliku docelowego aplikacja powinna pytać o nadpisanie pliku,
    b. podczas kopiowania powinno się pokazać okienko wyświetlające postęp (progress bar lub same procenty)
    c. podczas kopiowania dużego lub dużej liczby plików interfejs powinien się odświeżać prawidłowo (tj. samo kopiowanie nie powinno następować w wątku Swinga - wątek kopiujący powinien uaktualniać okienko postępu przez EventQueue.invokerLater)

Przykładowy interfejs:



Logika aplikacji ogranicza się do mechanizmu kopiowania i przenoszenia, który powinien być wydzielony w taki sposób, żeby można było go przetestować testami JUnit. Może być to zrealizowane przez klasę która ma metody kopiowania i przenoszenia, przyjmujące parametry 1) plik źródłowy 2) plik docelowy 3) listener wywoływany przy aktualizacji postępu, który w samej aplikacji zaktualizuje pasek postępu, a podczas testów jednostkowych może być nullem.

Test takiej logiki może się ograniczać do skopiowania / przeniesienia pliku i sprawdzenia czy rzeczywiście został przeniesiony.

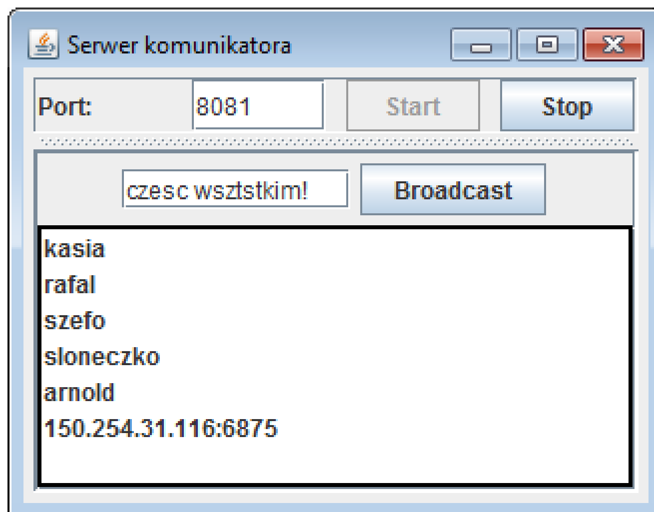
mgr inż. Adam Kozak
2013

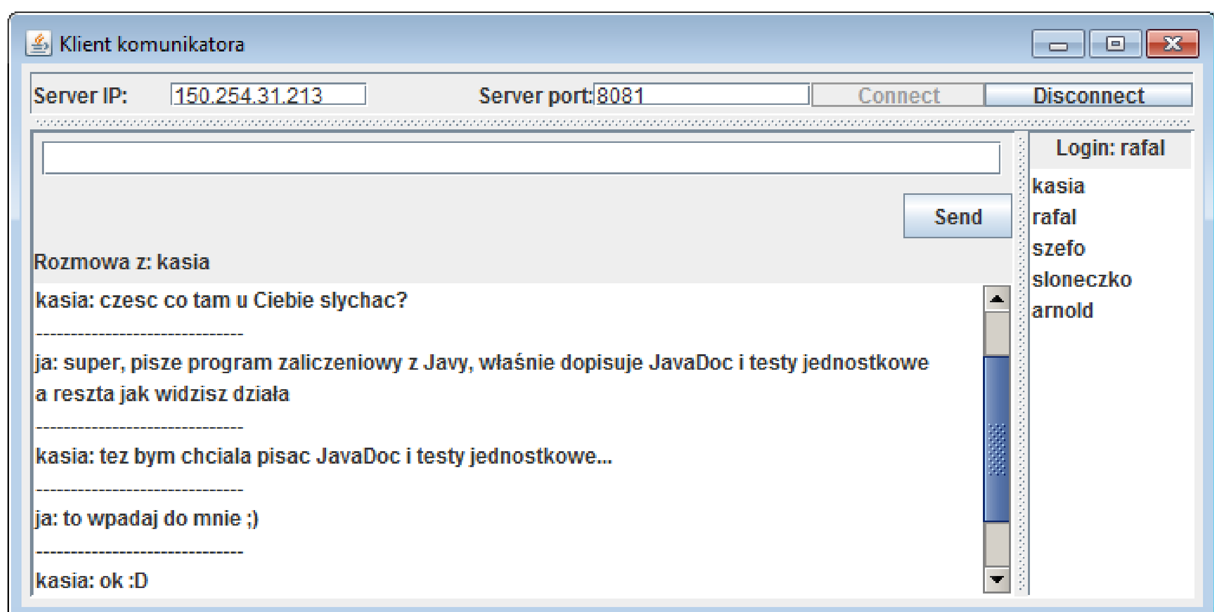
Komunikator internetowy

Komunikator internetowy powinien składać się z części serwerowej oraz klienta.

Na zaliczenie na ocenę maksymalnie 4,5 wystarczy serwer z którym można rozmawiać przez klienta telnet przy czym serwer komunikuje się z klientem Javowym lub przez telnet za pomocą następującego protokołu (powinna być więc możliwość komunikacji między programami napisanymi przez wszystkie osoby):

1. Przy połączeniu serwer odpowiada klientowi: @SERVER:HELLO
2. LOGIN nick - klient loguje się do serwera z danym nickiem, serwer odpowiada:
    a. @SERVER:OK - gdy dany login jest dostępny
    b. @SERVER:INUSE - gdy dany login jest juz zajety
    c. @SERVER:WRONG - gdy login sklada sie z innych znakow niz litery, cyfry i _
3. USERS - serwer zwraca listę zalogowanych użytkowników po przecinku: @USERS:kasia,rafal,szefo,sloneczko,arnold,150.254.31.116:6875
   użytkownicy którzy nie zdażyli się jeszcze zalogować będą wyświetlani w formacie IP:port socketa z którego się łączą
4. SEND login text - wysyła "text" do uzytkownika o podanym loginie np. SEND rafal czesc co tam u Ciebie slychac?
5. Serwer w dowolnym momencie może nadesłać wiadomość od innego użytkownika: @MESSAGE user text
   np. @MESSAGE kasia czesc co tam u Ciebie slychac?
6. BROADCAST text – serwer wysyła text do wszystkich zalogowanych użytkowników łącznie z nadawcą np.
   BROADCAST Czesc wszystkim!
   Serwer wysyła wszystkim zalogowanym użytkownikom komunikat:
   @BROADCAST: Czesc wszystkim!
7. DISCONNECT - użytkownik się rozłącza z serwerem
8. Przy podaniu przez klienta innej komendy niż LOGIN, USERS, SEND, BROADCAST i DISCONNECT serwer zwraca @SERVER:UNKNOWN
9. Wielkość liter komendy od klienta powinna być bez znaczenia dla serwera (tj. dopuszczalne są komendy LOGIN, Login, login etc.)





Logika aplikacji powinna zawierać rozszerzalny mechanizm obsługi komend po stronie serwera. Rozszerzalność komend oznacza, że aby dodać kolejną komendę do powyższego protokołu wystarczy napisać nową implementację komendy i dodać klasę w jakimś mechanizmie konfiguracyjnym (np. plik tekstowy lub plik .properties, zawierający listę klas z pełnymi ścieżkami pakietowymi). Każda komenda powinna być obsłużona jako osobna implementacja interfejsu komendy, np.

```
/**
 * Interfejs komendy serwerowej
 */
public interface ServerCommand {

                /**
                 * Metoda implementująca wykonanie komendy
                 * @param message - wiadomość od klienta (już bez samej komendy np. dla komendy "LOGIN
rysiek", tutaj będzie tylko "rysiek")
                 * @param client - klasa obsługująca wymianę danych z klientem, który wywołał komendę
                 */
                void execute(String message, ClientWorker client);

                /**
                 * Komenda na której przesłanie ma być odpalana metoda execute, lub null
```

```
	 * @return - komenda
	 */
	String getCommand();
}

/**
 * Implementacja interfejsu {@link ServerCommand} dla komendy LOGIN
 */
class LoginCommand implements ServerCommand {
	@Override
	public String getCommand() {
		return "LOGIN";
	}

	@Override
	public void execute(String message, ClientWorker client) {
		//Zalogowanie do aplikacji
	}
}
```

Testy jednostkowe powinny testować właściwe zachowanie serwera czyli np.

- Test czy serwer po połączeniu zwraca @SERVER:HELLO
- Test czy po poprawnym wykonaniu LOGIN maciek, komenda USERS zwróci w odpowiedzi @USERS:maciek
- Test czy po komendzie BROADCAST text, serwer odpowie @BROADCAST: text