**Projekt Programowanie Obiektowe w Javie**

**Grupa:**

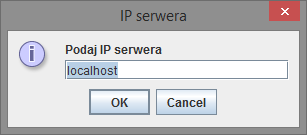
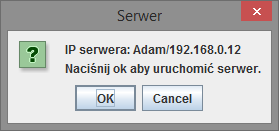
* Adam Furmanek
* Tomasz Gębski

**Temat:** Gra Szachy

**Sposób działania**

**Projekt składa się z pięciu klas:**  
- Serwer - łączy się z klientami, odbiera i przesyła informacje o wykonanym ruchu drugiemu graczowi;  
- Klient - łączy się z serwerem, odpowiedzialny za wysyłanie i odbieranie informacji o ruchu;  
- Okno - odpowiedzialna za utworzenie okna, na którym umieszczony zostanie panel;  
- Panel - nakładany na okno, służy do wyświetlania aktualnego stanu rozgrywki;  
- Pole - mała klasa, której obiekty przetrzymują informacje znajdujących się na nich pionkach.

**1. Połączenie**  
Aby uruchomić rozgrywkę, należy najpierw uruchomić Serwer, a następnie dwukrotnie Klient, z dowolnych komputerów w danej sieci lokalnej. Należy podać ip komputera, na którym znajduje się serwer (jeśli jest to ten samo co Klient, można zostawić sugerowane ip „localhost”). Pierwszy podłączony klient staje się graczem koloru czarnego, natomiast drugi koloru białego.



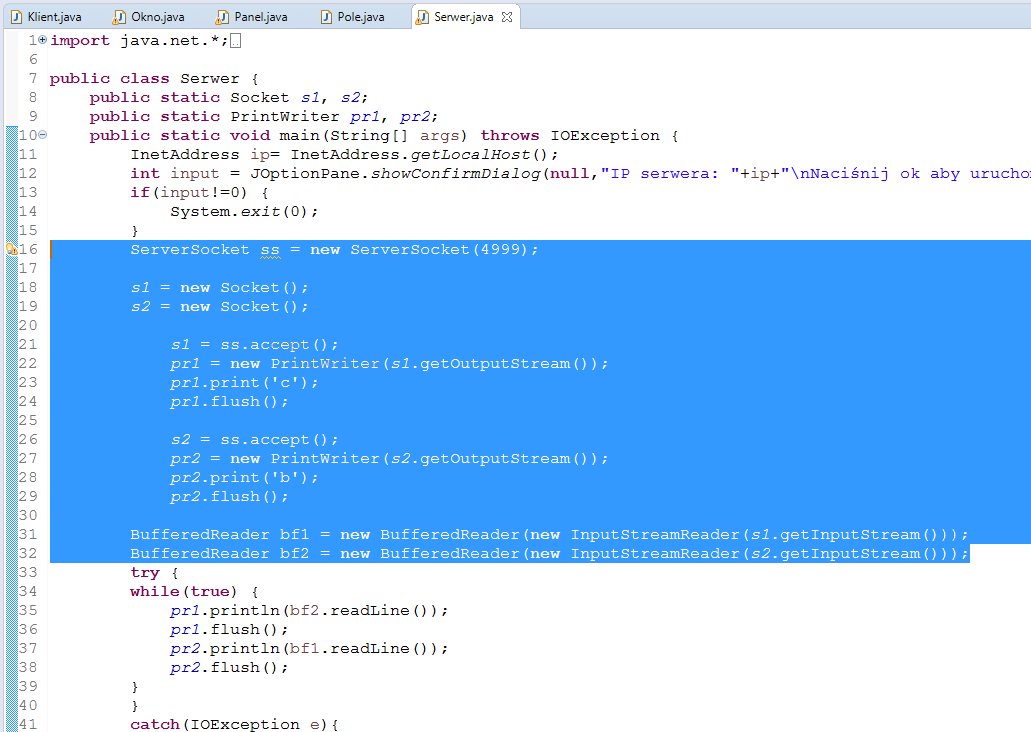
**2. Rozgrywka**  
Po połączeniu, swoją turę otrzymuje gracz biały. Gracz, który nie ma tury jest informowany o tym kursorem czekania (obracające się kółko ładowania). Po wykonaniu ruchu, informacja o nim zostaje przekazana do serwera i automatycznie do drugiego klienta. Teraz drugi gracz może wykonać ruch, a pierwszy zostaje zablokowany pod względem ruchu. Gracze mają do dyspozycji wszystkie możliwe ruchy, tak jak w oficjalnych zasadach szachów:  
- pionek może iść do przodu o 1 pole (lub o 2 w przypadku pierwszegu ruchu) i bić po skosie;  
- wieża porusza się w pionie lub w poziomie o nieograniczoną ilość pól;  
- skoczek wykonuje dla siebie charakterystyczny skok w kształcie litery L;  
- goniec porusza się po skosie o nieograniczoną ilość pól;  
- hetman porusza się tak jak wieża i goniec jednocześnie;  
- król porusza się w każdą stronę jednocześnie.  
- promocja pionka na hetmana po dojściu do końca planszy  
- roszada, czyli przeskok króla nad gońcem, gdy ten zostanie przysunięty  
Gra resetowana jest w momencie zbicia któregokolwiek z króli, a przegrany gracz, zaczyna kolejną rundę.



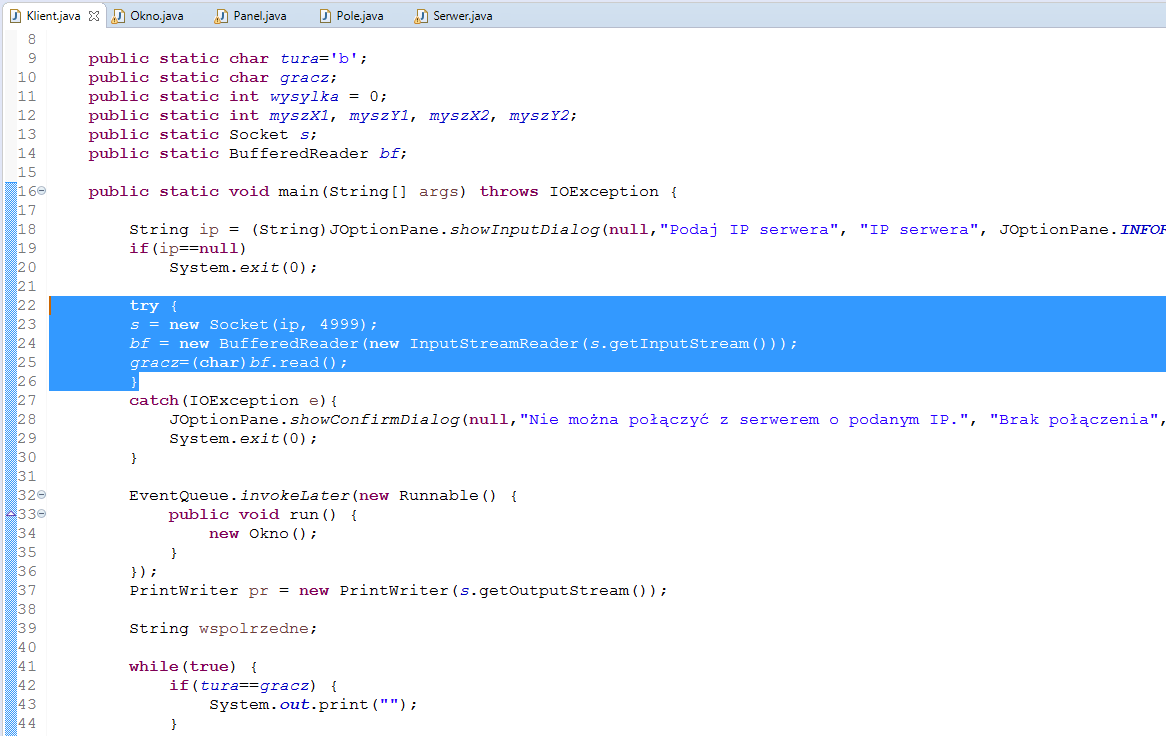
**3. Koniec gry**  
Kiedy jeden z graczy się rozłączy, serwer poinformuje o tym drugiego gracza i wyłączy jego okno, a następnie wyłączy samego siebie.



**Najważniejsze fragmenty kodu  
1. Mechanizm łączenia serwera z klientami.**

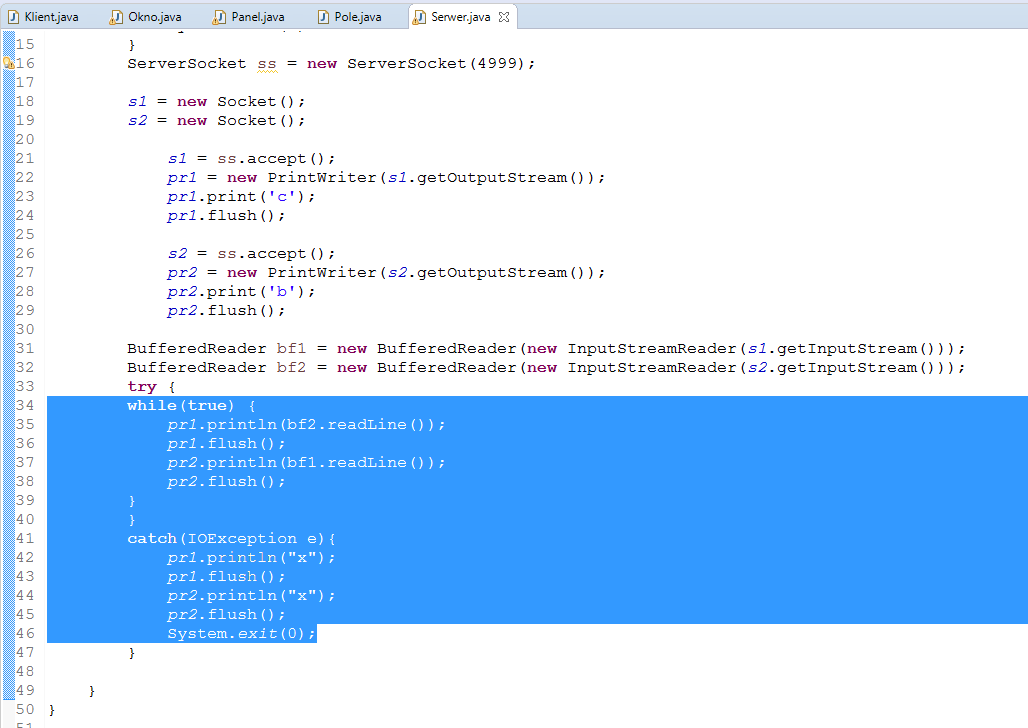


16: inicjacja obiektu ServerSocket  
21-24: połączenie z pierwszym klientem i wysłanie mu wiadomości ‘c’ (czarny)  
26-29: połączenie z drugim klientem i wysłanie mu wiadomości ‘b’ (biały)  
31-32: inicjacja obiektów odbierających wiadomości

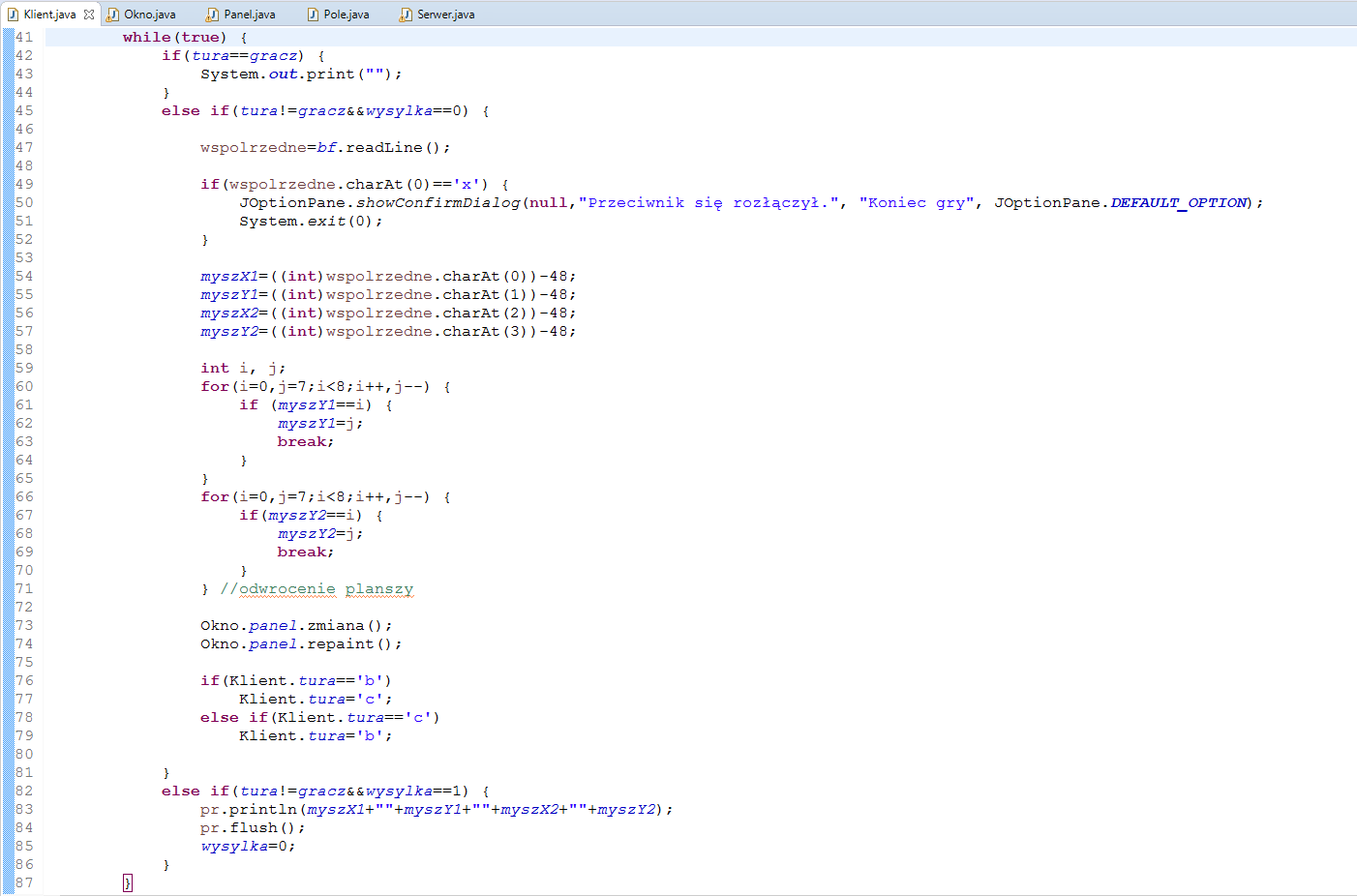


23-25: iniacjacja obiektu Socket, połączenie z serwerem odebranie i zapisanie pierwszej otrzymanej informacji (jakim jest się graczem: ‘b’ – biały, ‘c’ – czarny)  
37: inicjacja obiektu wysyłającego informacje

**2. Mechanizm przesyłania informacji o ruchu**

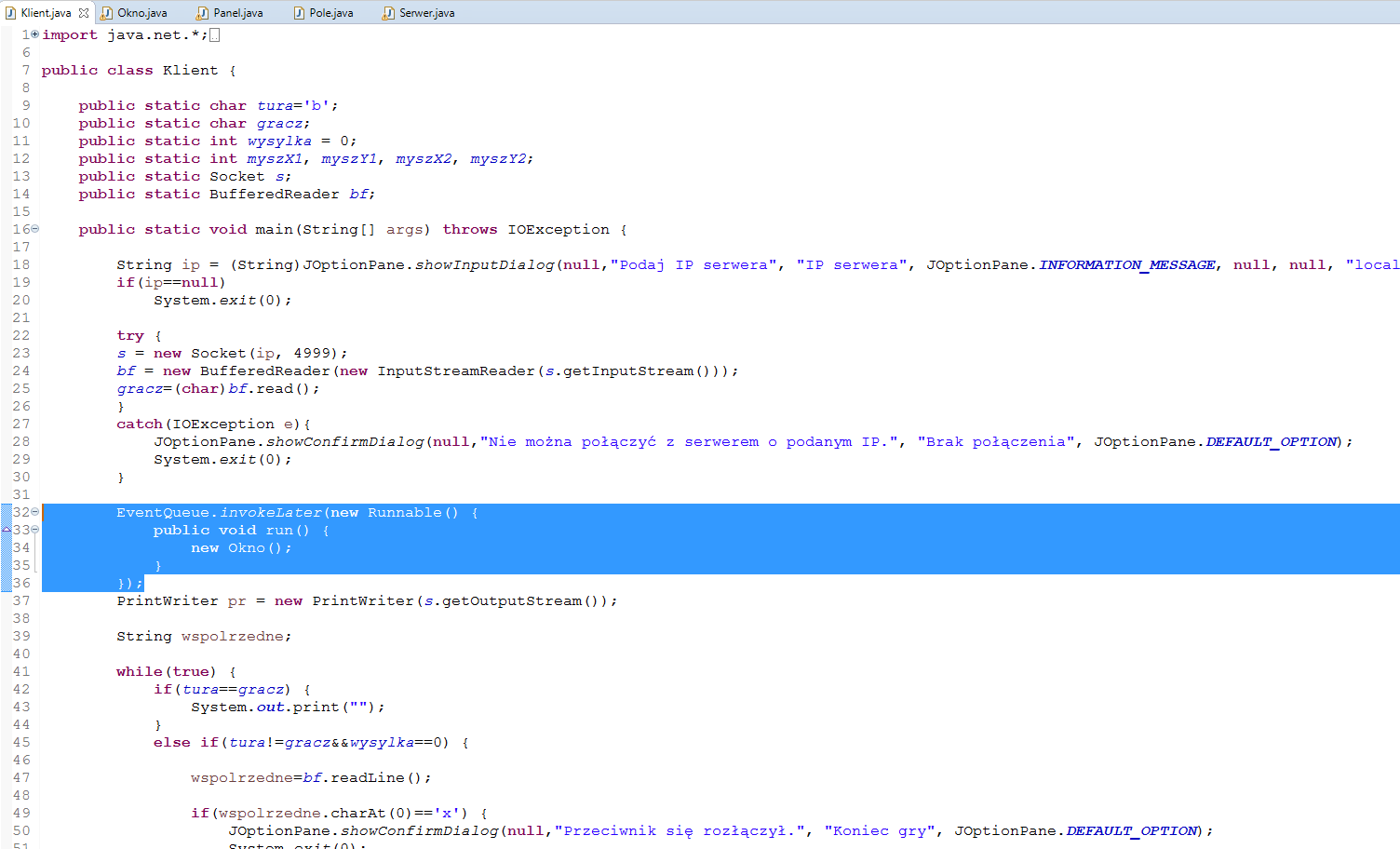


34-39: pętla główna serwera, ciągłe odbieranie od jednego i wysyłanie drugiemu klientowi  
41-47: wyjątek w razie utracenia połączenia z którymś z graczy, zostaje wysłany ‘x’ oznaczający koniec programu

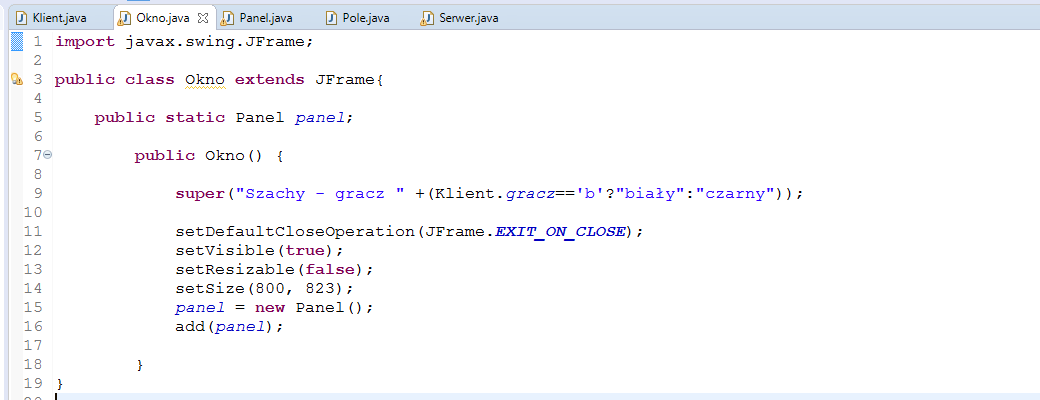


41-87: pętla główna klienta  
42-44: przypadek, kiedy tura należy do gracza, klient nie robi nic  
45-81: przypadek gdy gracz czeka na swoją turę, po udanym odebraniu w linijce 47, program zapisuje współrzędne, z których zostałą pobrana figura i na których została naniesiona  
59-71: odwrócenie wspołrzędnych, ponieważ obaj gracz mają swoje figury po tej samej stronie (na dole)  
73-74: wykonanie ruchu przeciwnika, ponowny render okna  
76-79: oddanie tury graczowi  
82-86: przypadek gdy gracz właśnie wykonał ruch i musi wysłać informację o nim na serwer

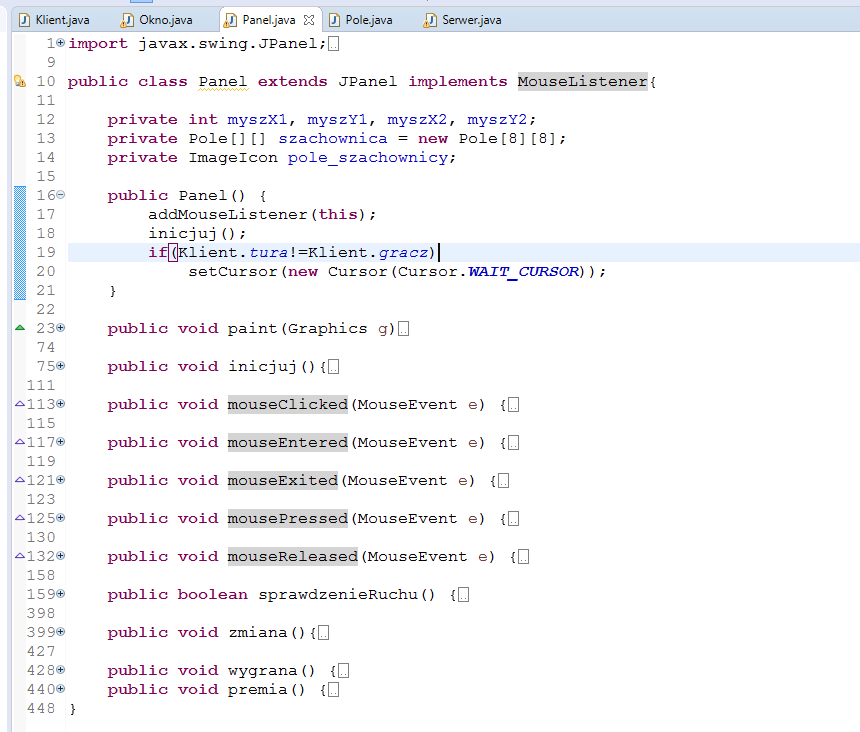
**3. Inicjacja okna i planszy**



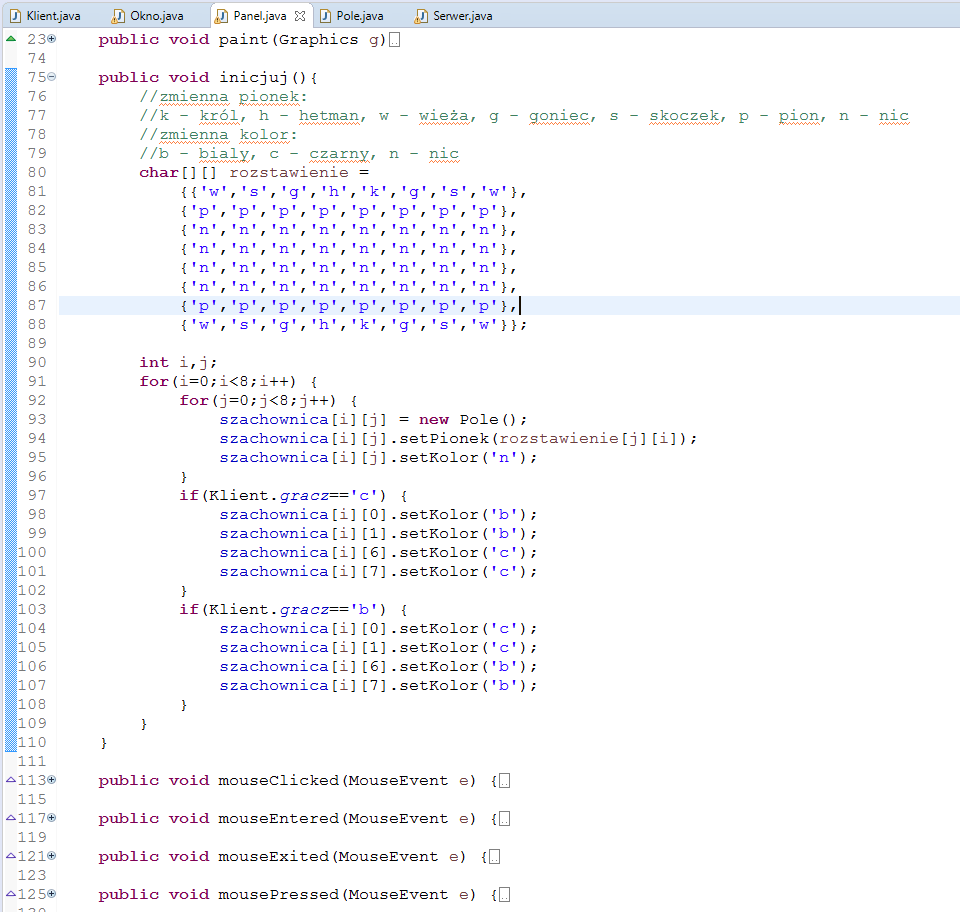
32-36: utworzenie okna w nowym wątku



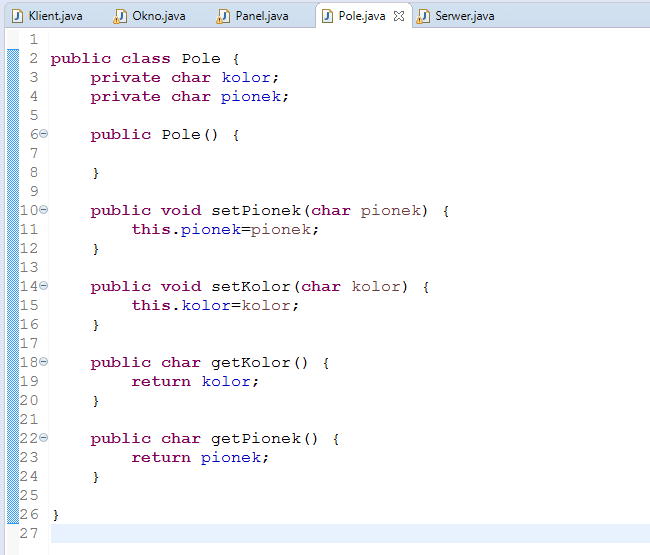
9-16: nadanie nazwy okna, sposobu wyłączenia, włączenie widoczności, zablokowanie zmieniania rozmiaru, określenie wielkości, nałożenie panelu



10: dziedziczenie po Jpanel, zaimplementowanie obłusgi myszki  
17, 113-132: zaimplementowanie obsługi myszki  
18: wywołanie funkcji inicjującej

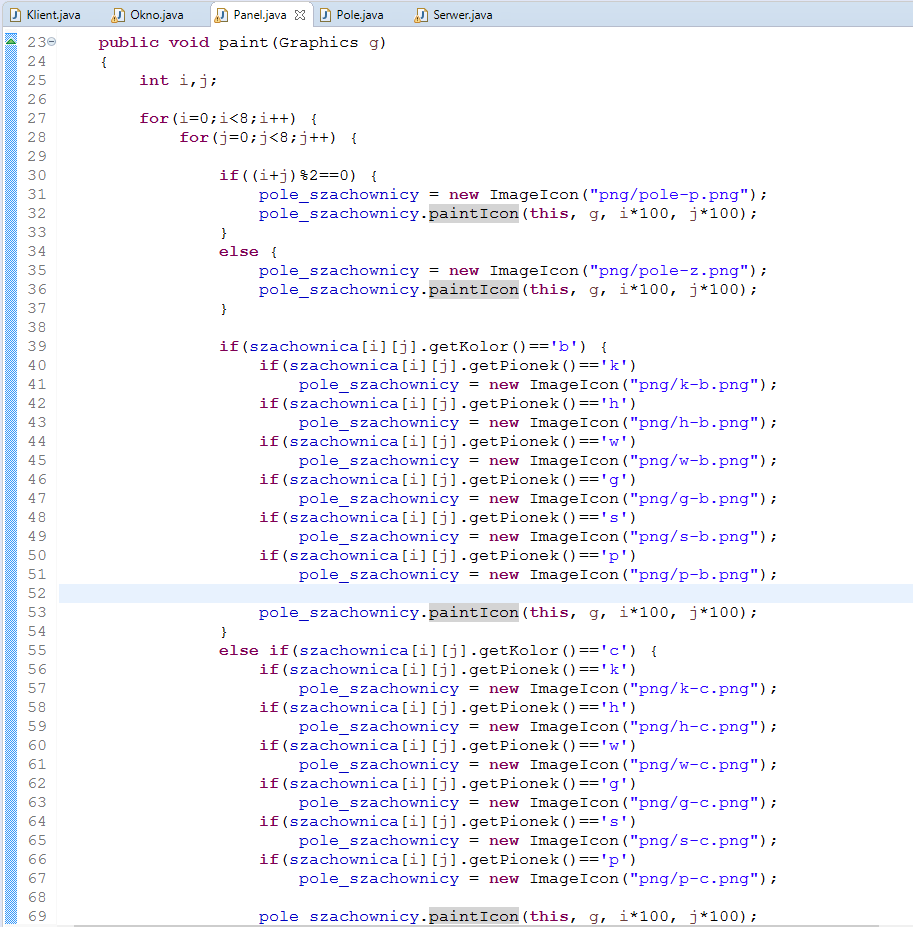


80-88: tymczasowa tablica rozstawienia  
91-109: stworzenie szachownicy z obiektów Pole, nałożenie kolorów i figur z tymczasowej tablicy rozstawienia



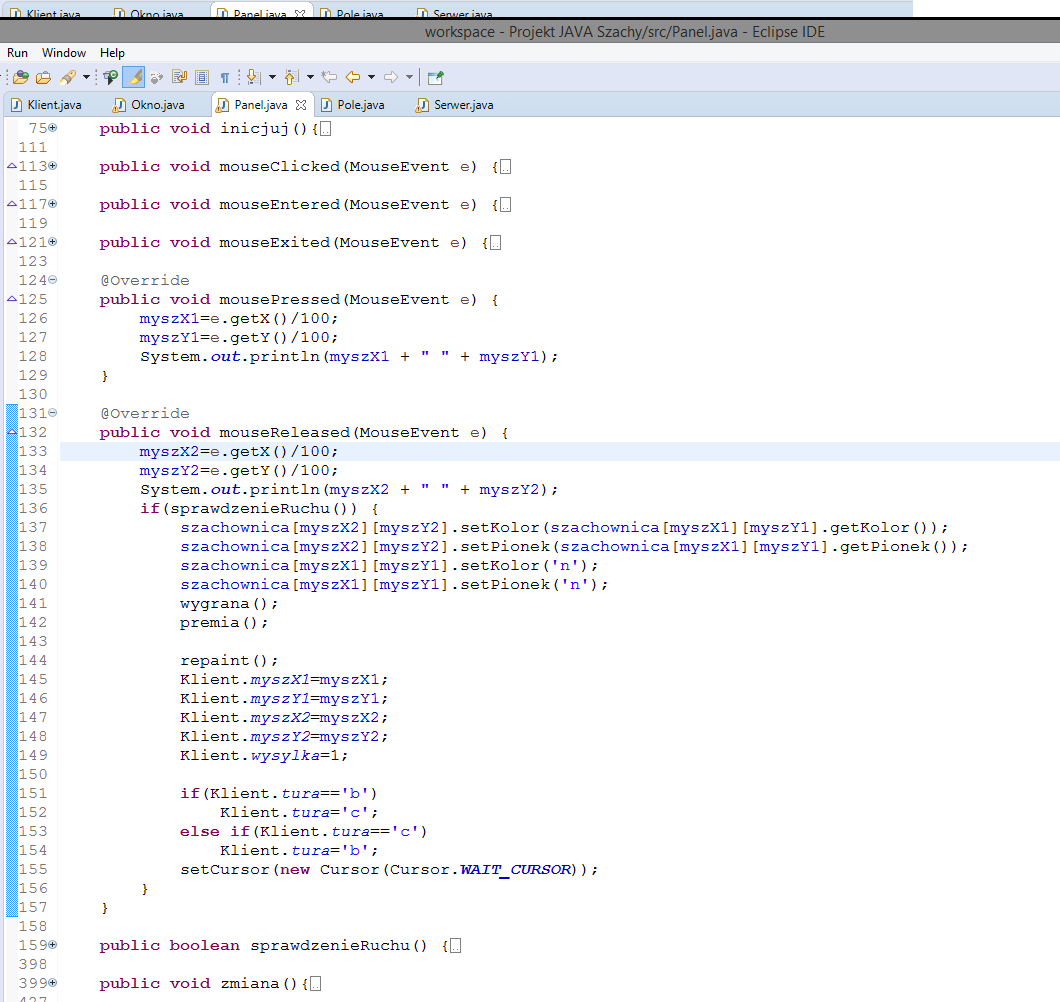
Ciało klasy Pole

**4. Renderowanie obrazu**



30-37: nałożenie żółto-białej szachownicy  
39-69: nakładanie odpowiednich figur w odpowiednich miejscach

**5. Obsługa własnego ruchu**



126-127: pobranie współrzędnych po naciśnięciu myszki  
133-134: pobranie współrzędnych po zwolnieniu myszki  
136: sprawdzenie poprawności ruch  
137-140: wykonanie ruchu  
141-144: sprawdzenie wygranej, sprawdzenie premii pionka, render obrazu  
145-148: przesłanie informacji o ruchu do Klienta  
149: przesłanie informacji dla Klienta, że nastąpił ruch i trzeba go wysłać  
151-154: oddanie tury



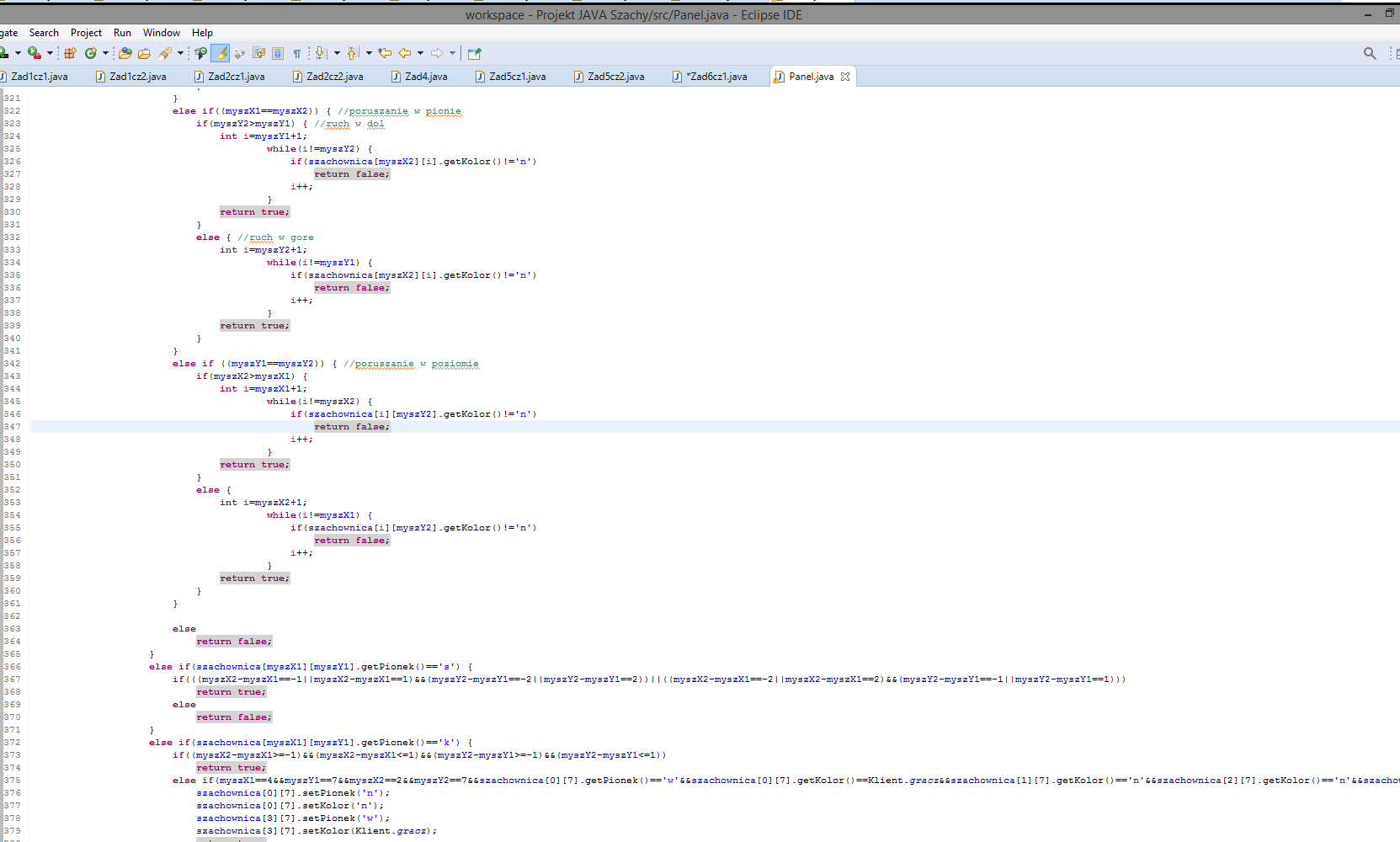
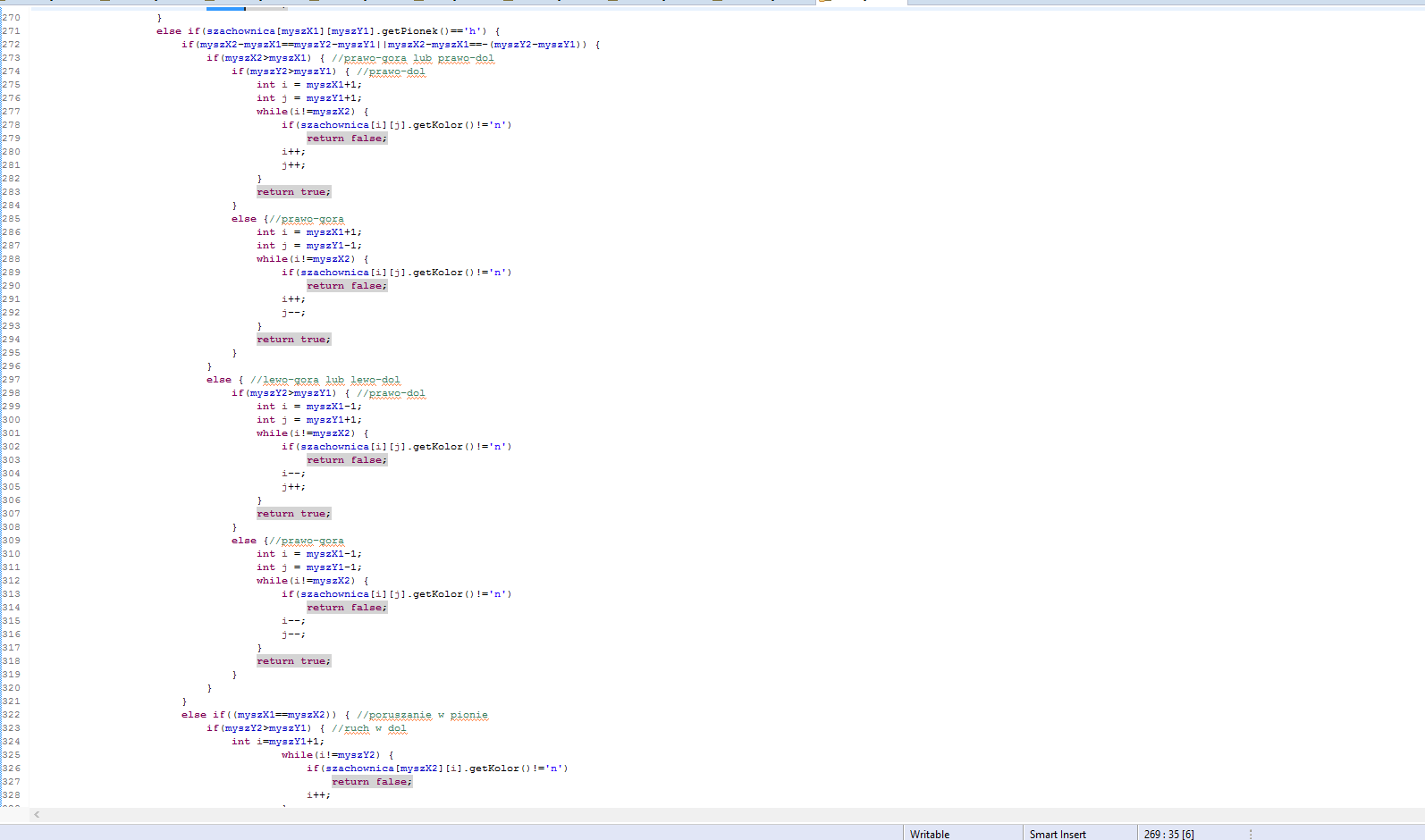
159-397: funkcja sprawdzająca ruch  
161: sprawdzane jest czy tura należy do gracza, czy drugie współrzędne nie wskazują tego samego miejsca, czy na docelowym miejscu nie ma pionka tego samego koloru, czy próbujemy ruszyć pionek dobrego koloru.

162-171: jeśli ruszany jest pionek  
163-164: return true, jeśli pionek jest w swoim początkowym rzędzie (czyli nie wykonał jeszcze ruchu) i został przemieszczony o dwa pola, przy czym po drodze, ani na docelowym polu nie było innej figury  
165-166: return true, jeśli pionek został przemieszczony o jedno pole do przodu, które było wolne  
166-167: return true, jeśli został przesunięty na skos, a na polu docelowym jest figura przeciwnika  
169-170: return false, jeśli innych ruch

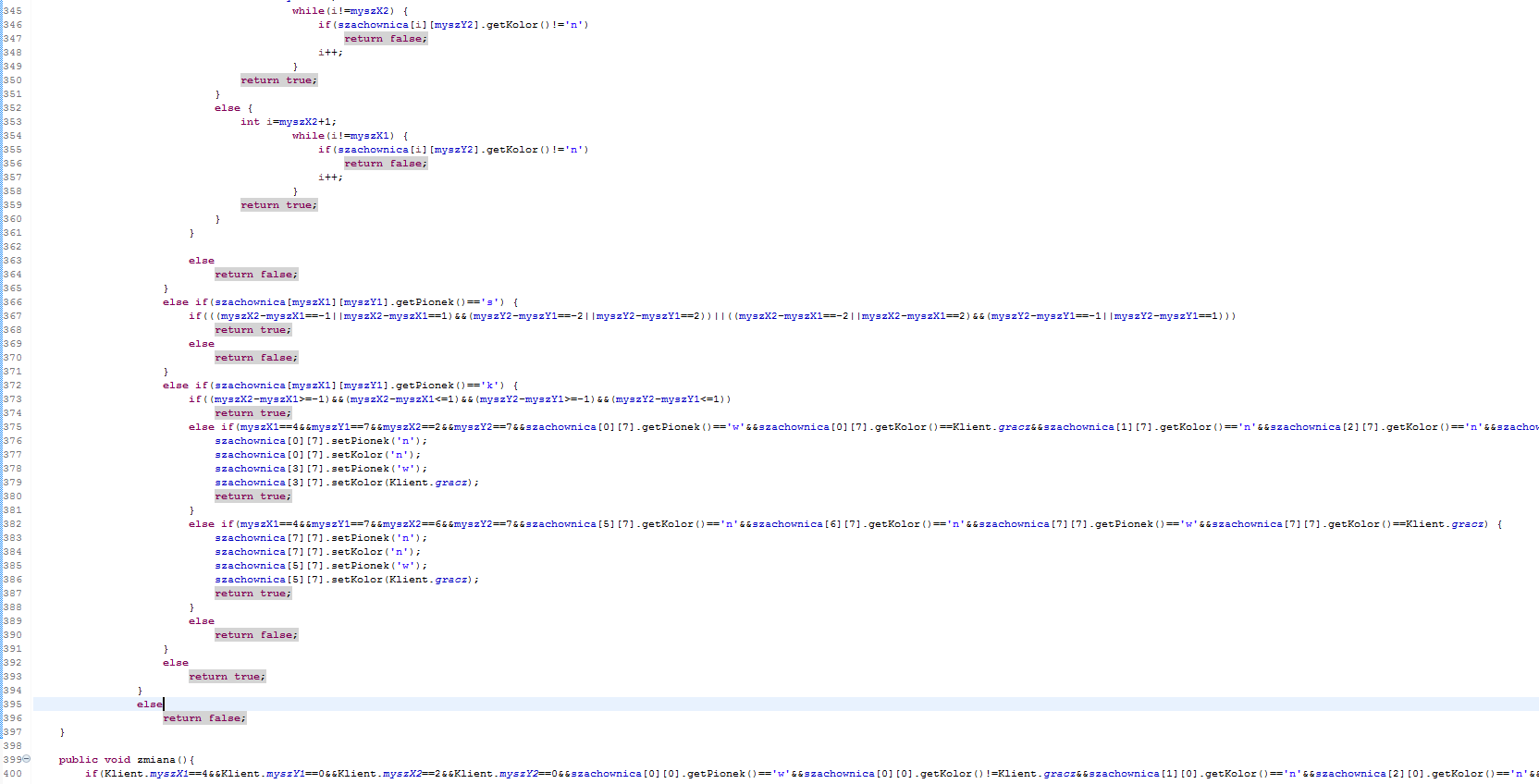
172-215 jeśli ruszana jest wieża  
173-192: ruch w pionie  
174-182: ruch w dół  
176-180: return false, jeśli między polem startowym, a docelowym nie jest pusto  
181: return true, jeśli pusto  
183-191: ruch w górę  
185-189: return false, jeśli między polem startowym, a docelowym nie jest pusto  
190: return true, jeśli pust  
193-212 ruch w poziomie(analogiczny do ruchu w pionie)  
213-214 return false, jeśli inny ruch



216-270: jeśli ruszany jest goniec  
217-266: jeśli ruch wykonany jest po skosie  
218-241: jeśli ruch w prawo  
219-229: jeśli ruch w dół  
220-228: return true, jeśli po drodze nie ma żadnych figur  
230-240: jeśli ruch w dół  
231-239: return true, jeśli po drodze nie ma żadnych figur  
242-265: jeśli ruch w lewo (analogicznie do ruchu w prawo)  
268-269: return false, jeśli inny ruch

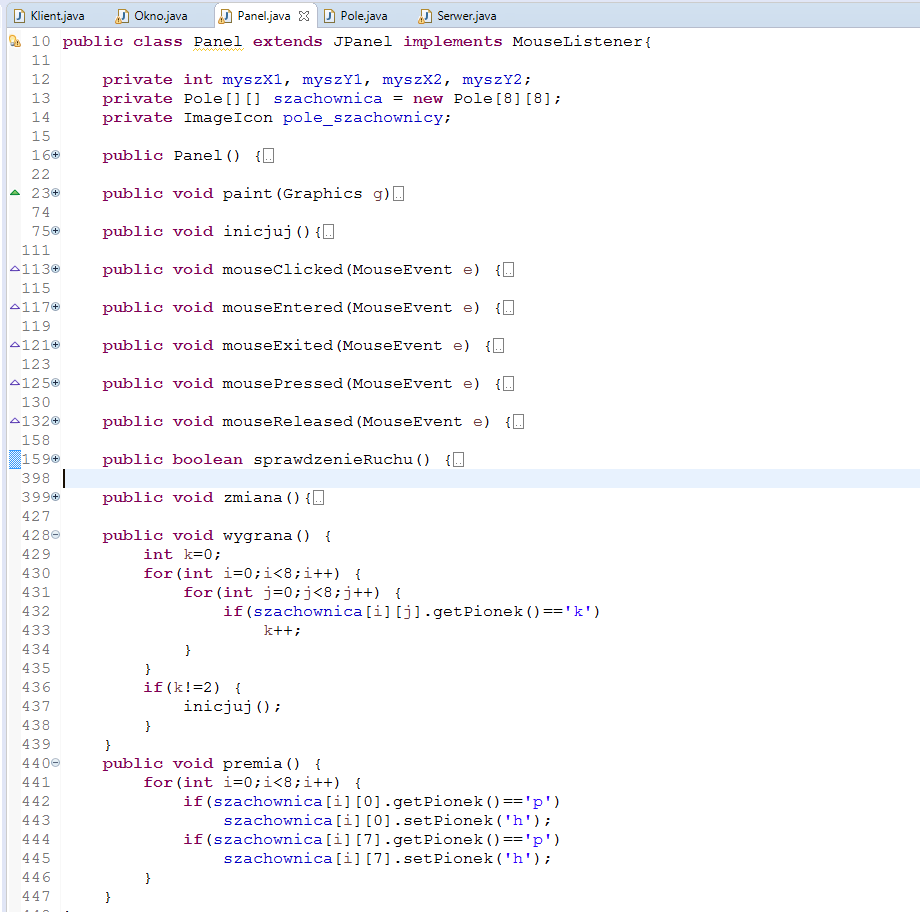


271-365: ruch jeśli hetman  
272-321: ruch po skosie (identyczny jak dla gońca)  
322-361: ruch w pionie/poziomie (identyczny jak dla wieży)  
363-364: return false, jeśli inny ruch



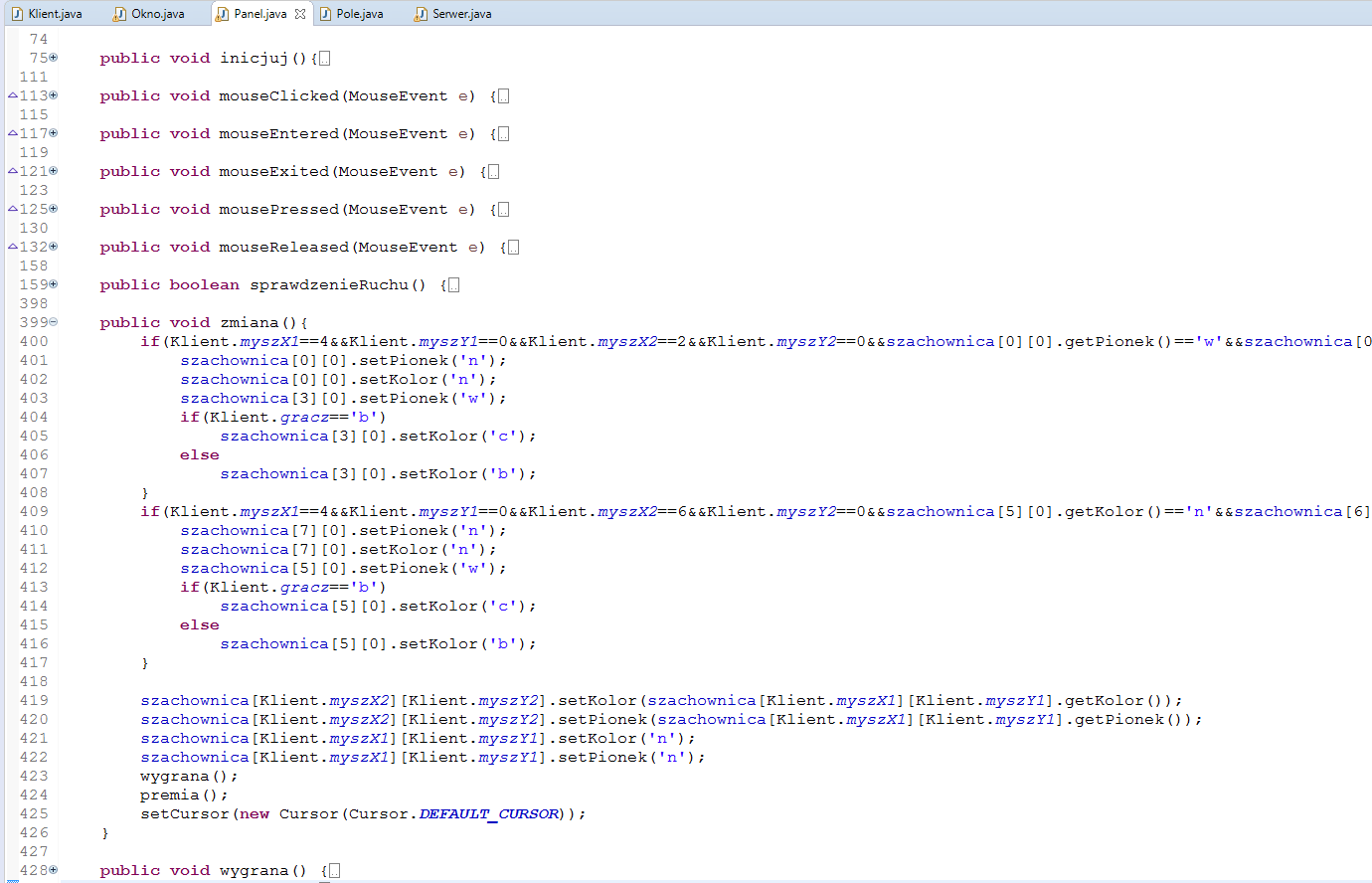
366-371: ruch jeśli skoczek  
367-368: return true, jeśli różnica w jednej współrzędnej = 1 i różnica w drugiej współrzędnej = 2 (charakterystyczny ruch w kształcie litery L w dowolnym kierunku)  
369-370 return false, jeśli inny ruch

372-391: ruch jeśli król  
373-374: return true, jeśli król porusza się o jedno pole w dowolnym kierunku  
375-381: return true, jeśli roszada z lewej strony <dłuższa roszada> (wieża i król, stoją na swoich początkowych miejscach)  
376-379: przeniesienie króla i wieży (wieża pojawia się obok króla, a król ją przeskakuje; jest to jedyna sytuacja, w której funkcja sprawdzająca wykonuje ruch na planszy)  
382-288: return true, jeśli roszada z prawej strony <krótsza roszada>  
383-386: przeniesienie króla i wieży  
389-390: return false, jeśli inny ruch



430-435: policzenie ile króli znajduje się na planszy  
436-438: reset gry, jeśli na planszy nie ma dwóch króli  
441-446: jeśli w pierwszym lub w ostatnim rzędzie znajduje się pionek, oznacza to, że doszedł on do końca i zostaje wymieniony na hetmana

**6. Obsługa ruchu przeciwnika**



400-417: obsługa roszady gracza, czyli dwóch charakterystycznych ruchów dotyczących króla i gońca (czterech jeśli liczyć obu graczy)  
419-422: obsługa zwykłego ruchu, czyli przeniesienie pionka na nowe miejsce  
423-424: sprawdzenie wygranej, sprawdzenie premii pionka

**Uwagi i wnioski:**  
W trakcie tworzenia programu, a także z perspektywy czasu po jego ukończeniu, programista chce wielokrotnie zmieniać założenia, architekturę i pomysły całego projektu. Podobnie jest również w tym przypadku, np. chęć całkowitego pozbycia się klasy Serwer, tak by Klient łączył się bezpośrednio z drugim Klientem, co kosztowałoby przebudowę architektury całej pracy. Na szczęście pomimo naszych perfekcyjnych natręctw, udało się skończyć i złożyć grę w całość, bez zbędnego dopieszczania najmniejszych detali. Ważna lekcja, poza zdobytą wiedzą, jest taka, by nie naprawiać czegoś co działa. Kolejnym wnioskiem jest potencjał jaki wywodzi się z języka Java. Wszystkie konieczne biblioteki potrzebne do tworzenia prostych aplikacji znajdują się już na swoim miejscu, składnia języka jest bardzo wygodna, a korzystanie z Javy sprawia, że programowanie obiektowe zaczyna robić się coraz bardziej zrozumiałe.