Laboratorium 4

Celem pierwszej części ćwiczenia jest:

- doskonalenie umiejętności tworzenia klas w wykorzystaniem dziedziczenia i kompozycji,
- nabycie umiejętności rysowania prostych figur geometrycznych ,
- nabycie umiejętności implementacji interfejsów,
- nabycie umiejętności obsługi zdarzeń od klawiatury (`KeyListener`), myszy (`MouseListener`) oraz uaktywniania widoku (`ActionListener`) .

Celem drugiej części ćwiczenia jest nabycie umiejętności tworzenia komentarzy dokumentujących w programach źródłowych języka Java.

Zadanie 0 (analiza przykładowego programu)

W ramach tego ćwiczenia należy napisać ***graficzny edytor grafów,*** który umożliwia za pomocą myszki i klawiatury edycję ***grafu*** złożonego z ***węzłów*** oraz ***krawędzi*** rysowanych na panelu wypełniającym główne okno aplikacji. Przykład działania takiego edytora pokazują programy `SimpleGraphEditor.jar` oraz program `ColorGraphEditor.jar.`

Pierwszy program jest edytorem graficznym, który umożliwia rysowanie na płaszczyźnie prostych grafów złożonych z węzłów i krawędzi. W tym programie węzły są reprezentowane przez punkty na płaszczyźnie o współrzędnych `x`, `y`. Krawędzie są reprezentowane przez odcinki łączące parę punktów. Graf jest reprezentowany jako lista wierzchołków oraz lista krawędzi. Program umożliwia rysowanie grafu na panelu wypełniającym główne okno aplikacji. Dostępne operacje edycyjne (tworzenie, przesuwanie oraz usuwanie węzłów i krawędzi) mogą być wykonywane za pomocą myszki i klawiatury.
Ponieważ węzły i krawędzie poza położeniem na płaszczyźnie nie mają innych atrybutów, to w programie nie ma dodatkowych okien dialogowych. Program umożliwia wypisanie struktury grafu (lista węzłów oraz lista krawędzi) oraz zapis/odczyt grafu do/z pliku binarnego.

Program `ColorGraphEditor.jar` powstał przez rozszerzenie definicji węzłów i krawędzi o dodatkowe atrybuty. Dla węzłów dodano atrybut `nazwa` i `kolor`, a dla krawędzi tylko `kolor`. Program został rozbudowany o dodatkowe opcje menu kontekstowego oraz okna dialogowe umożliwiające zmianę tych atrybutów dla wskazanego węzła lub wskazanej krawędzi. Ten program stanowi więc uniwersalny edytor grafów o kolorowych węzłach i krawędziach.

Jeśli wybranym kolorom nadamy umownie jakieś znaczenie, to takim edytorem można narysować graf, który będzie graficznie obrazował określone informacje np. poglądowy schemat komunikacji miejskiej, gdzie węzły grafu oznaczają przystanki, a różne kolory krawędzi oznaczają różne środki komunikacji (autobus, tramwaj, kolej, itp.). Jednak taki program nie umożliwia prezentowania bardziej szczegółowych informacji o systemie komunikacji np. poprzez „podświetlanie" przebiegu trasy wybranej linii komunikacyjnej. Żeby taka szczególna funkcjonalność była możliwa to model danych musi być poszerzony o specyficzne dla konkretnego zastosowania informacje: np. zawierać definicję linii komunikacyjnej reprezentowanej jako lista kolejnych przystanków oraz dodatkowe atrybuty – nazwa i rodzaj środka transportu. Ponadto interfejs użytkownika musi zawierać specyficzne dla tego zastosowania polecenia edycyjne lub okna dialogowe (np. polecenie „edytuj linie", które otwiera okno dialogowe umożliwiające edycję listy przystanków dla tej linii.

W materiałach ćwiczenia został udostępniony kod źródłowy programu do edycji listy kolorowych kółek (***węzłów grafu***). Ten kod przedstawia sposób realizacji graficznego interfejsu użytkownika z obsługą poleceń edycyjnych za pomocą myszki i klawiatury. Proszę przeanalizować ten program zwracając szczególną uwagę na sposób rysowania elementów graficznych na panelu oraz sposób obsługi zdarzeń generowanych przez klawiaturę i myszkę.

Program składa się z czterech klas: `Node`, `Graph`, `GraphPanel` oraz `GraphEditor`. Pierwsze dwie klasy stanową model danych i nie zawierają implementacji operacji edycyjnych. Klasa `GraphEditor` tworzy główne okno aplikacji oraz implementuje główne menu programu. Ta klasa dziedziczy po klasie `JFrame` i implementuje interfejs `ActionListener`. Najważniejszą częścią programu jest klasa `GraphPanel`, w której jest zaimplementowana obsługa operacji edycyjnych za pomocą myszki i klawiatury. Ta klasa dziedziczy po klasie `JPanel`, a ponadto implementuje trzy interfejsy `MouseListener`, `MouseMotionListener` oraz `KeyListener`.

W konstruktorze tej klasy zostały podłączone do panelu słuchacze zdarzeń od myszki i klawiatury. Ponadto dla panelu wywoływane jest metoda `setFocusable(true),` której zadaniem jest nadanie panelowi właściwości „*focusable*”, czyli umożliwienie przyjmowania zdarzeń od klawiatury.

Zadaniem metody `setMouseCursor` jest ustawienie stosownego do aktualnej sytuacji kształtu kursora myszki.

Żeby było możliwe rysowanie po panelu dowolnych figur geometrycznych konieczne jest **przedefiniowanie** !!! metody `paintComponent(Graphics g).` Ta metoda będzie wywoływana automatycznie przez system graficzny za każdym razem, gdy zachodzi potrzeba ponownego odrysowania treści okna.

Uwaga: Proszę nie pomylić tej metody z metodą o nazwie `paintComponents,` która również jest dostępna w klasie `JPanel`.

Do metody `paintComponent` przekazywany jest parametr `g` z klasy `Graphics`.
Klasa `Graphics` umożliwia wykonanie prostych operacji graficznych (np. zmianę koloru rysownia, rysowanie odcinków, łamanych oraz elips.

Uwaga: Jeśli jest potrzeba wykonania bardziej zaawansowanych operacji (np. zmiana grubości linii, którą są rysowane figury to można parametr `g` rzutować na klasę `Graphics2D`.

Pierwszą instrukcją w przedefiniowanej metodzie `paintComponent` musi być wywołanie oryginalnej metody `paintComponent` z klasy `JPanel.` Ta instrukcja wypełni całą powierzchnię panelu kolorem tła. Na tym tle należy wyrysować oczekiwany obraz.

Tak przygotowana metoda `paintComponent` będzie mogła być użyta przez system graficzny za każdym razem, gdy zajdzie potrzeba odrysowania zawartości fragmentu ekranu zajmowanego przez obiekt klasy `GraphPanel`. Potrzeba odrysowania może wynikać z przyczyn zewnętrznych (np. okno innej aplikacji zostanie zamknięte lub przesunięte i odsłoni ten fragment ekranu). Takie sytuacje wykrywa system graficzny i automatycznie wywoła metodę `paintComponent`. Potrzeba odrysowania obrazu może wynikać również z przyczyn wewnętrznych. Taka sytuacja ma miejsce gdy nastąpi zmiana w modelu danych spowodowana przez wykonanie operacji edycyjnych (np. utworzenie nowego węzła grafu). Innym powodem wewnętrznym może być zmiana trybu wyświetlania (np. zmiana skali rysunku lub ukrycie części rysowanych elementów dla poprawy czytelności). W takiej sytuacji nie należy wywoływać metody `paintComponent` w bezpośredni sposób. Zamiast tego należy poinformować system graficzny o potrzebie odrysowania panelu poprzez wywołanie dla obiektu panelu metody `repaint()`. To spowoduje, że system graficzny wywoła metodę `paintComponent` po obsłużeniu wszystkich innych zdarzeń o wyższym priorytecie.

Uwaga: Metody `repaint()` nie wolno wywołać wewnątrz metody `paintComponent` oraz wewnątrz wszystkich innych metod, które są wywoływane przez metodę `paintComponent` (np. metody które rysują poszczególne węzły lub krawędzie grafu).

Kolejnym ważnym elementem klasy `GraphPanel` jest zestaw metod, które implementują interfejsy `MouseListener`, `MouseMotionListener` oraz `KeyListener`. To w tych metodach jest zawarta implementacja wszystkich operacji edycyjnych wykonywanych przez użytkownika aplikacji za pomocą myszki i klawiatury.

Ostatnim elementem klasy `GraphPanel` są dwie przeciążone metody `createPopupMenu`. W tym programie metody te są wywoływane po „zwolnienu" prawego przycisku myszki (zob. metodę `mouseReleased`). Metody `createPopupMenu` ilustrują sposób utworzenia i obsługi menu kontekstowego, czyli takiego menu, którego opcje są wyświetlane bezpośrednio na panelu. Menu kontekstowe to obiekt klasy `JPopupMenu`. Do tego menu są dodawane kolejne opcje czyli obiekty klasy `JMenuItem`. Po dodaniu wszystkich opcji menu jest wyświetlane na pomocą metody `show`.

Do każdej opcji menu za pomocą metody `addActionListener` dołączany jest słuchacz zdarzeń akcji, który ma być wywołany po wybraniu tej opcji menu. W tym programie procedura obsługi zdarzenia jest utworzona bezpośrednio za pomocą wyrażenia Lambda co pozwoliło na znaczne skrócenie kodu programu. Ten sam efekt można osiągnąć przez użycie jako słuchacza zdarzeń obiektu klasy anonimowej, która implementuje interfejs `ActionListener`. Sposób użycia klasy anonimowej został w programie pokazany w komentarzu.

Zadanie 1 (wariant prosty)

Proszę napisać uniwersalny program do edycji „kolorowych grafów" o funkcjonalności zbliżonej do programu ColorGraphEditor.jar. Program powinien umożliwiać następujące funkcje:

- tworzenie, edycję lub usuwanie węzłów (kolorowe kółka)
- tworzenie, edycję lub usuwanie krawędzi (kolorowe odcinki łączące pary węzłów)
- zmianę położenia na płaszczyźnie wskazanych węzłów, krawędzi lub całego grafu,
- zapis/odczyt grafu do/z pliku binarnego.
- wyświetlanie struktury grafu:
  - listy węzłów i ich atrybutów (położenie, kolor, nazwa)
  - listy krawędzi i ich atrybutów (kolor)
- tworzenie co najmniej jednego przykładu grafu, który pokazuje możliwości edytora.
- wyświetlanie krótkiej instrukcji obsługi i danych autora.

Zadanie 2 (wariant złożony - dla osób ambitnych)

Osoby, które zdecydowały się zrobić wariant złożony nie muszą robić wariantu prostego (zadania nr 1)

Proszę wybrać jakieś zagadnienie, które można opisać za pomocą specyficznego grafu. Dla tego wybranego zagadnienia proszę krótko opisać słownie proponowany sposób reprezentacji informacji w formie grafu (zob. dwa przykłady poniżej).
Proszę napisać program, który umożliwi graficzną edycję grafu, który reprezentuje to wybrane zagadnienie. Definicja klas reprezentujących węzły i krawędzie grafu powinna zawierać atrybuty specyficzne dla wybranego zagadnienia. Ponadto sposób wyświetlania

może być dostosowany do zagadnienia. Węzły grafu mogą być rysowane za pomocą prostych ikonek lub innych symboli, przy których są wyświetlane dodatkowe napisy (np. węzeł reprezentujący przystanek komunikacji może być rysowany w postaci małej chorągiewki, przy której jest wypisana nazwa przystanku).
Program edytora powinien umożliwiać następujące funkcje:

- tworzenie, edycję lub usuwanie węzłów,
- tworzenie, edycję lub usuwanie krawędzi,
- zmianę położenia na płaszczyźnie wskazanych węzłów, krawędzi lub całego grafu,
- zapis/odczyt grafu do/z pliku binarnego,
- wyświetlanie struktury grafu,
  (sposób prezentacji struktury grafu zależy od rodzaju zagadnienia, np. dla grafu reprezentującego schemat komunikacji można wypisywać trasy poszczególnych linii, dla grafu reprezentującego drzewo genealogiczne można wypisywać wszystkich przodków oraz wszystkich potomków wybranej osoby)
- tworzenie co najmniej jednego przykładu grafu, który pokazuje możliwości edytora. (do programu można dołączyć plik binarny zawierający reprezentatywny przykład, który będzie wczytywany po wybraniu odpowiedniej opcji menu).

Ponadto program powinien wyświetlać krótką instrukcję obsługi i dane autora.

Poniżej opisano dwa przykładowe zagadnienia, które mogą być reprezentowane za pomocą grafu. Do realizacji zadania można również wybrać inne przykłady.

Schemat komunikacji miejskiej:

- Zbiór węzłów grafu reprezentuje wszystkie przystanki. Dla każdego przystanku pamiętana jest nazwa i położenie przystanku.
- Uporządkowana lista wybranych węzłów reprezentuję trasę linię komunikacji miejskiej. Dla każdej linii oprócz trasy określona jest nazwa (numer) oraz rodzaj środka komunikacji.
- Pary sąsiednich węzłów na trasie linii komunikacyjnej tworzą krawędzie grafu. Kolor krawędzi może zależeć od rodzaju środka komunikacji.
- Edytor umożliwia dodawanie, edycję i usuwanie przystanków oraz dodawanie i usuwanie linii komunikacji.
- Dla wybranej linii edytor umożliwia edycję trasy (tworzenie uporządkowanej listy przystanków).
- Ponadto edytor umożliwia wyrysowanie w formie graficznej mapy zawierającej wszystkie przystanki (węzły grafu), wszystkie bezpośrednie połączenia między przystankami (krawędzie grafu) oraz trasę wskazanej linii wyróżnionym kolorem.

Drzewo genealogiczne:

- Węzły grafu reprezentują poszczególne osoby. Dla każdej osoby pamiętane jest nazwisko i imię, płeć oraz lata życia.
- Krawędzie grafu reprezentują relacje pokrewieństwa (małżeństwo, relacja rodzic-dziecko) .
- Edytor umożliwia dodawanie oraz usuwanie osób oraz określanie relacji pokrewieństwa między parami osób (małżeństwo, rodzic-potomek),
- Edytor sprawdza typowe ograniczenia:
  - małżeństwo dopuszczalne między osobami różnych płci,
  - każda osoba może mieć tylko jednego ojca i tylko jedną matkę,
  - rok urodzenia potomka musi mieścić się w przedziale wyznaczonym przez lata życia rodziców.
- Ponadto edytor umożliwia wyrysowanie dla wskazanej osoby drzewa przodków oraz drzewa potomków.

Zadanie 3 (dla ambitnych)

Program przykładowy `QuadraticEquation.java` zawiera definicję klasy reprezentującej trójmian kwadratowy. Metody zawarte w tej klasie umożliwiają utworzenie i wczytanie współczynników trójmianu, a następnie obliczenie i wyświetlenie na ekranie pierwiastków tego trójmianu. Wszystkie składniki tej klasy zostały opatrzone komentarzami dokumentującymi. Proszę przeanalizować ten program oraz zwrócić szczególną uwagę na sposób opisu składowych tej klasy w komentarzach dokumentujących.

Zalecany sposób tworzenia komentarzy dokumentujących jest omówiony w książce: Joshua Bloch, "Java - Efektywne programowanie" Wydanie II, Temat 44: „Tworzenie komentarzy dokumentujących dla wszystkich udostępnianych elementów API” (str. 217-224).

Wzorując się na programie `QuadraticEquation.java` proszę uzupełnić swój program z tego ćwiczenia o komentarze dokumentujące. Komentarze proszę dopisać we plikach modelu danych czyli w klasie reprezentującej węzły grafu, w klasie reprezentującej krawędzie grafu oraz w reprezentującej cały graf. Nie trzeba robić komentarzy w klasach reprezentujących interfejs użytkownika (okno aplikacji oraz inne okna dialogowe).