AGH Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

PROGRAMOWANIE OBIEKTOWE W JAVIE I C#

Projekt aplikacji do wyszukiwania połączeń komunikacji miejskiej w Krakowie.

Autor: **Szydłowski Artur**
Nr albumu: 28 53 51
Kierunek: Automatyka i Robotyka
Wydział Inżynierii Mechanicznej i Robotyki

Prowadzący: mgr inż. Tymoteusz Turlej

Kraków, 2017r

1. Temat.

Projekt aplikacji do wyszukiwania połączeń komunikacji miejskiej w Krakowie, którą nazwałem krótko *"Do Celu"*.

2. Cel projektu.

Stworzenie aplikacji z rozkładem jazdy komunikacji miejskiej, która umożliwiałaby wyszukiwanie optymalnych połączeń pomiędzy dwoma dowolnie wybranymi punktami na mapie o wybranej godzinie danego dnia tygodnia.

3. Zakres projektu.

a.) Założenia wstępne

- stworzenie bazy danych zawierającej rozkład jazdy, z możliwością łatwej edycji i aktualizacji
- stworzenie bazy dynych zawierającej listę przystanków wraz z ich położeniem na mapie oraz przyporządkowaniem do odpowiedniego rozkładu jazdy
- niezależność od systemu operacyjnego, możliwość działania aplikacji zarówno przez stronę internetową jak i w trybie offline po pobraniu
- graficzny interfejs użytkownika zawierający pasek menu z wyborem miejsca startu oraz celu podróży, dnia tygodnia, godziny i innych dodatkowych opcji, a także interaktywna mapa
- powiadomienia o utrudnieniach w ruchu, korkach, remontach oraz pogodzie

b.) Modyfikacja założeń

- pomysł na stworzenie kilku algorytmów wyszukiwania połączeń – komfortowego (bez przesiadek), najszybszego (z uwzględnieniem wszelkich możliwych połączeń, również pieszych, bez ograniczenia liczby przesiadek) i optymalnego (z uwzględnieniem pewnej ograniczonej liczby przesiadek i dojść pieszych, ale tylko na krótkich dystansach)
- możliwość zaznaczenia dowolnych pozycji na mapie przez użytkownika (nie tylko przystanków)
- dynamiczne wczytywanie rozkładu jazdy tylko tych linii, które są potrzebne
- rezygnacja z pracy nad systemem powiadomień z powodu ograniczeń czasowych

c.) Zakres końcowy

Udało mi się zrealizować większość z założeń wstępnych w stopniu pozwalającym na używanie aplikacji, jednak program wciąż wymaga przeprowadzenia procesu debugowania, zabezpieczenia działania aplikacji systemem obsługi wyjątków i dopracowania wielu aspektów technicznych i graficznych.

4. Wymagania programu.

a.) System operacyjny – dowolny wyposażony w wirtualną maszynę Javy.

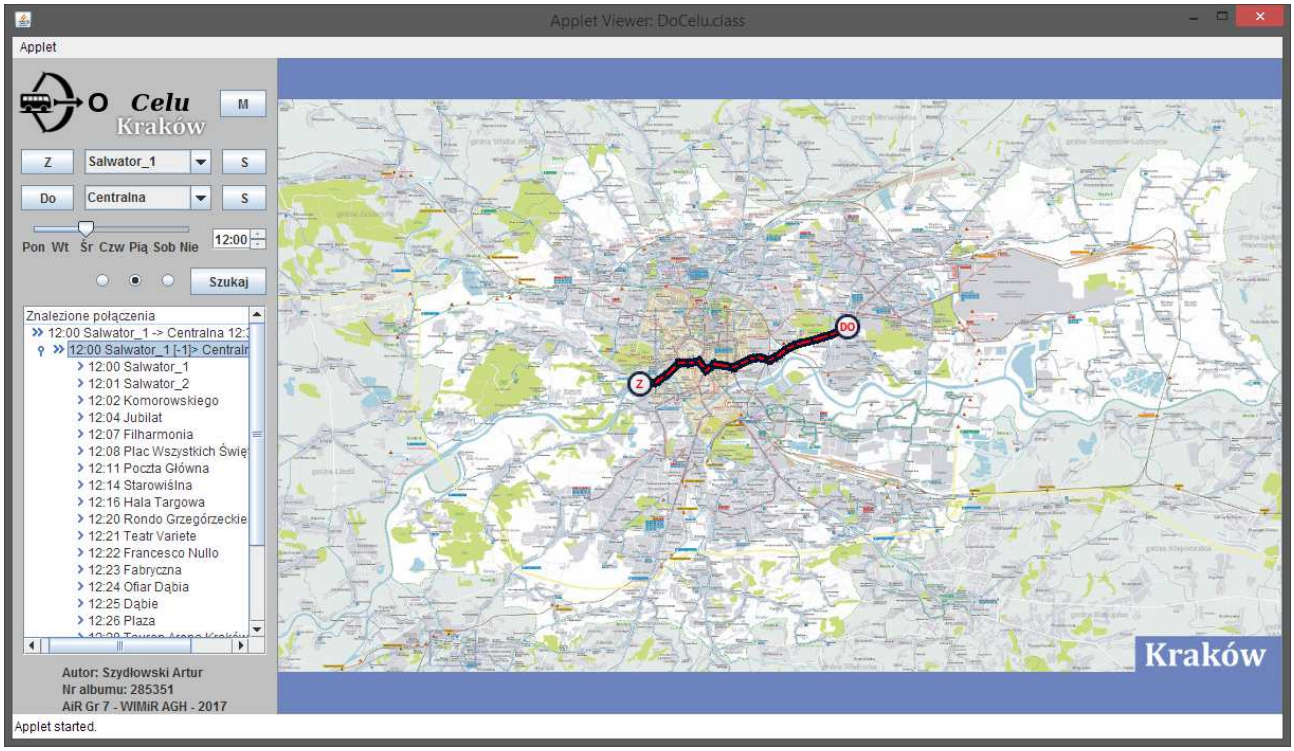
b.) Biblioteki – zestaw standardowych bibliotek Javy.

c.) Wersja środowiska – Java 7.0

d.) Wtyczki, kodeki, programy zewnętrzne – brak (ewentualnie mój program konsolowy do pobierania rozkładu jazdy i zapisywania w formie txt)

5. Opis działania aplikacji.

Po uruchomieniu aplikacji i wyszukaniu pewnej trasy, oczom użytkownika ukaże się następujący widok:



Okno Appletu

Przystanek startowy i docelowy można wybrać z rozwijanej listy lub zaznaczyć je bezpośrednio na mapie. W tym celu należy użyć przycisku "Z" lub "Do" z menu, a następnie kliknąć w odpowiednie miejsce na mapie. W trakcie wyszukiwania połączenia, zostaną z wybranych punktów doprowadzone ścieżki piesze do najbliższych przystanków.

Po prawej stronie znajduje się interaktywna mapa Krakowa, którą można przybliżyć w dowolnym miejscu przy pomocy myszki, a następnie poruszać nią poprzez kliknięcie w odpowiednią część obramowania. Jeśli został już wybrany jakiś punkt startowy, docelowy lub została wyszukana trasa, będzie to wyświetlane na mapie. Aby przybliżyć automatycznie do punktu startowego lub końcowego, można przycisnąć przycisk z literą S znajdujący się w menu po lewej stronie. Aby oddalić do pełnego widoku, należy nacisnąć przycisk z literą M.

Widok po zbliżeniu mapy

Po wybraniu interesującego nas dnia tygodnia i godziny odjazdu oraz sposobu wyszukiwania można nacisnąć przycisk "Szukaj". Jeśli program znajdzie jakieś połączenia, wyświetli je poniżej w postaci rozwijanego drzewa połączeń i przystanków pośrednich. Na mapie standardowo wyświetla się pierwszy wynik wyszukiwania, można to jednak zmienić poprzez kliknięcie w interesującą nas trasę z drzewa wyszukiwania. W trybie oddalonym połączenia między przystankami oznaczane są w postaci linii - tramwaje w kolorze czerwonym, autobusy na niebiesko, a trasy piesze na biało. Po zbliżeniu linie zmieniają się na strzałki kierunkowe (pojedyncze – przystanki pośrednie, podwójne – punkty przesiadki).

6. Najciekawsze rozwiązania.

Program jest zbudowany z 10 klas:

1.) **DoCelu** – klasa główna Appletu, zarządzająca działaniem całej aplikacji. Zbudowana jest z paneli klasy JPanel i JScrollPane. Zawiera komponenty klasy Swing takie jak: przyciski (JButton), przyciski typu radio (JradioBuuton), tekstowo-obrazkowe (JLabel, ImageIcon), listy rozwijane (JComboBox), suwak (JSlider), pole godziny (JSpinner) i drzewa (JTree). Implementuje ActionListener do obsługi menu użytkownika i TreeSelectionListener do wyboru wyświetlanej trasy.

2.) **CityMap** – panel interaktywnej mapy dziedziczący po klasie Jpanel, implementujący MouseListener. Składa się z tła, na którym wyświetlane są grafiki mapy w klasie BufferedImage oraz przezroczystych przycisków sterujących. Odpowiednie grafiki mapy są wczytywane w miarę potrzeby, przy zbliżeniu program pracuje tylko na małych wycinkach, a nie na dużym pliku pełnowymiarowej mapy. Przy odświeżaniu sprawdza dane zawarte w innych klasach i rysuje odpowiednie elementy, jeśli są zawarte w aktualnym widoku. Mapa umożliwia administratorowi również łatwe zaznaczanie pozycji przystanków.

3.) **Spot** – klasa opisująca stałe miejsce na mapie, takie jak przystanek autobusowy, tramwajowy, czy dworzec PKP. Zawiera jego położenie, nazwę oraz numery kursujących linii. W sposób statyczny zawiera również listę wszystkich przystanków, co umożliwia łatwą komunikację z innymi klasami. Lista ta jest wczytywana z pliku przy uruchomieniu aplikacji i może być aktualizowana.

4.) **MySpot** – klasa dziedzicząca po klasie Spot. Opisuje dowolny punkt na mapie (ustawiony przez użytkownika) i dojścia piesze do najbliższych przystanków.

5.) **Vector2D** – klasa opisująca dwuwymiarowy wektor położenia. Przy jej pomocy m.in. obliczane są odległości na mapie oraz sprawdzane jest zawarcie przystanku w widoku mapy.

6.) **Travel** – klasa opisująca środki komunikacji miejskiej. Zawiera w sposób statyczny listę wczytanych obiektów, zaś każdy obiekt ma przyporządkowany swój unikalny numer (ujemny lub dodatni w zależności od kierunku jazdy, 0 przyjmuje się dla dojść pieszych) oraz rozkład jazdy. Dane są wczytywane dynamicznie w razie potrzeby. Klasa pośredniczy w wymianie danych z rozkładu jazdy.

7.) **Timetable** – klasa opisująca rozkład jazdy. Zwraca m.in. informacje na temat następnego przystanku, godziny następnego kursu, najbliższych kursów, odległości między przystankami, itp.

8.) **Time** – klasa opisująca godzinę. Dane zapisywane są w formacie minutowym w tygodniu. Zwraca informacje na temat dnia, godziny, rodzaju dnia (roboczy, sobota, niedziela), itp.

9.) **Link** – klasa opisująca połączenie dwóch obiektów typu Spot (lub MySpot). Poza łączonymi obiektami zawiera również informację nt. numeru połączenia, godziny odjazdu (wyjścia), godziny dojścia oraz kosztu dotarcia do ostatecznego celu podróży (dotychczasowy koszt plus czas dojścia pieszego).

10.) **Path** – klasa zawierająca algorytmy wyszukujące trasy dojazdu, implementująca klasę Comparable. Obiekty tej klasy zawierają pojedynczą ścieżkę dojścia – listę obiektów Link, a także metody analogiczne do klasy Link, czyli zwracające czas dojścia, godzinę, itp. Klasa składa się również ze statycznej metody makeTree(), która tworzy drzewo wynikowe w oparciu o wyznaczone ścieżki dojścia, a także algorytmy quickPath(), directPath() i planowany optimalPath().

Algorytm "wygodny" (bez przesiadek) – sprawdza czy dane dwa przystanki są połączone i jeśli tak, to sprawdza najbliższe godziny połączeń po wskazanej przez użytkownika godzinie dnaego dnia tygodnia. Jeśli użytkownik ręcznie wybrał punkt startu, to połączenia zostaną wyszukane z najbliższych przystanków w okolicy. Przystanek docelowy musi być określony z listy przystanków.

Algorytm szybki – działą na zasadzie algorytmu wyszukiwania ścieżki A*. Polega on na tym, że zaczynając od punktu startowego tworzy listę możliwych połączeń z kolejnymi przystankami i oblicza dla każdego koszt dotarcia do celu. Algorytm bierze również pod uwagę trasy piesze. W kolejnych krokach zawsze analizuje przystanek o najmniejszym koszcie dojścia. Jeśli okaże się, że jakiś przystanek był już analizowany, cała ścieżka zostaje zapamiętana, jako jedna z optymalnych. Na koniec każdej iteracji lista przystanków jest sortowana według rosnącego kosztu. W ten sposób algorytm zawsze znajduje najszybszą trasę dotarcia do celu, aczkolwiek czasami wiąże się to z dużą ilością przesiadek. Dlatego do celów praktycznych należałoby ten algorytm udoskonalić i odrzucić nieracjonalne wyniki. Inną wadą mojego algorytmu jest to, że bierze pod uwagę tylko najwcześniejsze połączenia, a późniejsze odrzuca, co może skutkować tym, że może odrzucić również szybsze i bardziej racjonalne połączenia.

Algorytm optymalny – miał być algorytmem łączącym zalety obu powyższych. Zapewniać szybki dojazd do celu, ale z uwzględnieniem możliwości maksymalnie 1-2 przesiadek oraz krótkich tras pieszych. Ze względów ograniczeń czasowych, algorytm nie został opracowany w praktyce – powstał jedynie wstępny zarys jego struktury.

7. Napotkane problemy.

Niemożliwe jest, aby tak duży projekt nie przyspożył wielu problemów różnego rodzaju i tak też było w moim przypadku. Można by je podzielić m.in.na problemy wynikające z mojej nieznajomości języka JAVA (z którymi radziłem sobie dzięki dokumentacji, poradnikom i tematom na forach internetowych) czy błędy w kodzie (zwłaszcza przy współpracy różnych klas) oraz błędy w działaniu algorytmów.

Jednym z problemów było np. podzielenie dużej mapy na mniejsze pliki graficzne, co dokonałem przez napisanie prostego programu w C++ z użyciem biblioteki SFML. Innym pobranie aktualnego rozkładu jazdy z internetu, co rozwiązałem przez napisanie odpowiedniego programu w C#, pobierającego ze strony kod HTML i przetwarzającego go do odpowiedniej postaci do pliku txt. To jednak przysporzyło problemu z kodowaniem polskich znaków, ich prawidłowym zapisem i późniejszym wczytywaniem przez aplikację.

8. Dalsze kierunki rozwoju.

- Wprowadzenie do systemu pozostałych połączeń i przystanków autobusowych i tramwajowych oraz dodanie połączeń transportu kolejowego.
- Zamieścić aplikację na serwerze internetowym i uruchomić jej działanie w trybie online.
- Optymalizacja systemu zapisu i odczytu danych – m.in. kodyfikacja nazw przystanków i binarny format zapisu, co pozwoli nawet kilkukrotnie zmniejszyć ilość pamięci i nieco zwiększyć szybkość ich przetwarzania.
- Optymalizacja algorytmów wyszukiania połączeń oraz dopracowanie i udoskonalenie otrzymywanych wyników wyszukiwania szybkiego (nie wszystkie połączenia są sensowne). Opracowanie algorytmu optymalnego.
- Dopracowanie graficznego interfejsu użytkownika, m.in. sposobu wyświetlania wyników wyszukiwania.
- Opracowanie mechanizmu automatycznej aktualizacji rozkładu.
- Stworzeniu systemu powiadomień o utrudnieniach na trasie – korki, remonty, pogoda, itp.