

# Specyfikacja implementacyjna projektu 2

Sebastian Pietrykowski, Paweł Borkowski  
Grupa projektowa nr 3

11 maja 2022

## 1 Opis ogólny

Program zostanie napisany w języku Java. Do utworzenia aplikacji okienkowej zostanie użyta JavaFx. Obsługa przez użytkownika będzie odbywała się poprzez graficzny interfejs użytkownika.

Cel projektu 2 jest zdefiniowany w specyfikacji funkcjonalnej.

## 2 Środowisko deweloperskie

Projekt 2 jest tworzony w środowisku programistycznym Visual Studio Enterprise Edition 2022 w wersji 17.1 w języku Java. Wykorzystywany jest kompilator Java SE Development Kit 18.0.1.1. Ponadto wykorzystane zostanie oprogramowanie JavaFX w wersji 18.0.1 oraz Scene Builder w wersji 18.0.0

W projekcie 2 wykorzystywana jest konwencja nazewnicza Camel Case.

Do pracy przy projekcie używany jest system kontroli wersji Git. Poszczególne gałęzie noszą nazwy funkcjonalności implementowanej w programie, są one scalane do gałęzi głównej po prawidłowym przejściu testów.

## 3 Opis pakietów

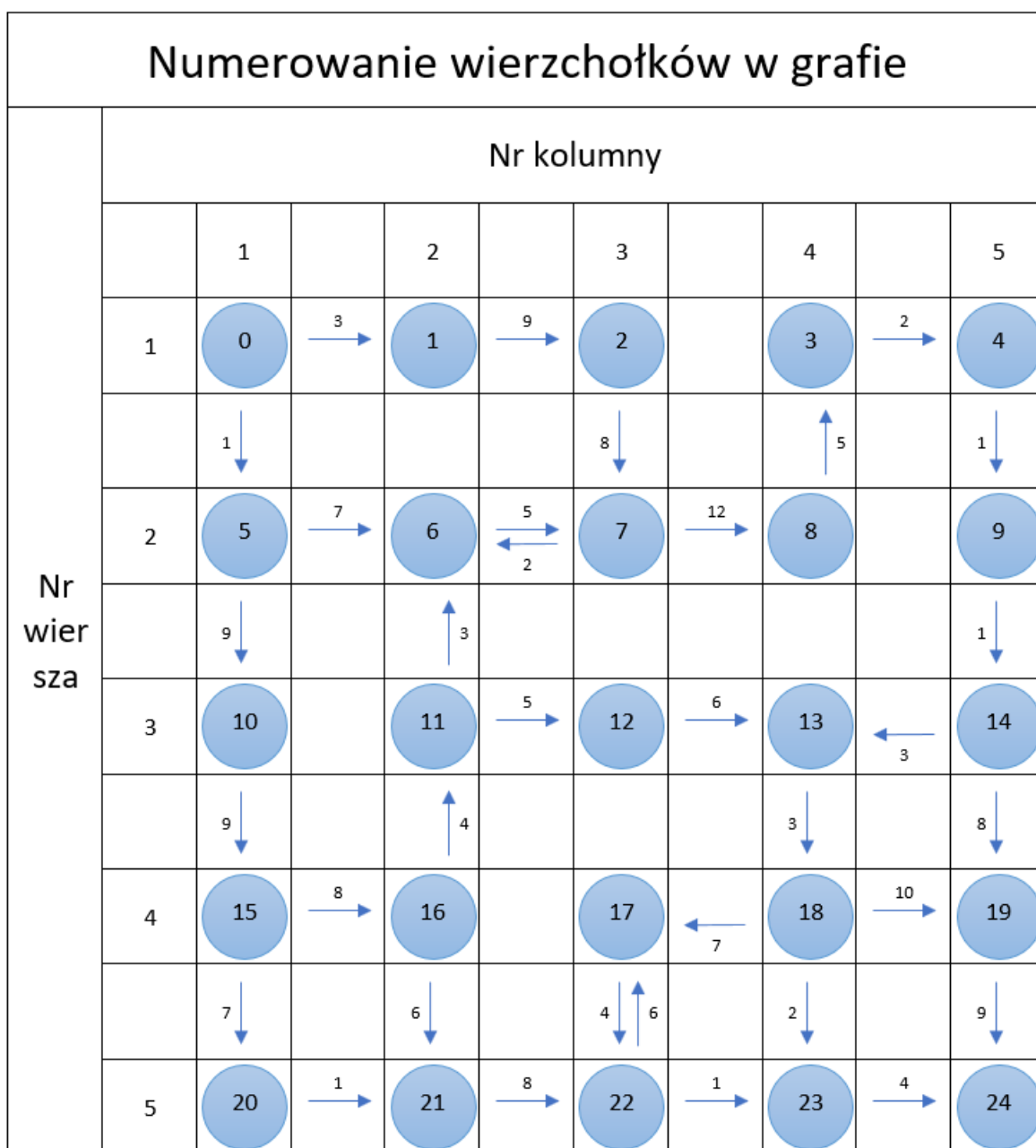
Projekt 2 zostanie podzielony na dwa pakiety. Pierwszy pakiet będzie składał się z kodu tworzącego aplikację – `ApplicationPackage`, natomiast drugi zawierać będzie testy jednostkowe – `TestPackage`.

## 4 Opis klas programu

Rozdział 4 poświęcony jest analizie klas tworzących program. Dokładnie opisane są tutaj pola oraz metody każdej z klas.

Uwaga: klasy odpowiedzialne za graficzny interfejs opisane są w rozdziale 5.

Sposób numerowania wierzchołków w grafie został przedstawiony na poniższej grafice:



Rys.1 Sposób numerowania wierzchołków w grafie na przykładzie pewnego grafu.

**public class Edge** – przechowuje jedną krawędź w grafie.

- Pola

- private int fromVertex – numer wierzchołka, od którego zaczyna się krawędź
- private int toVertex – numer wierzchołka, do którego biegnie krawędź
- private double weight – waga krawędzi.

- Metody

- public Edge( int from, int to, double weight ) – konstruktor
- public int getFromVertex() – zwraca wartość pola fromVertex
- public int getToVertex() – zwraca wartość pola toVertex

- `public double getWeight()` – zwraca wartość pola `weight`.

**public class Graph** – klasa odpowiedzialna za przechowywanie grafu. Posiada metody czytające graf z pliku, zapisujące go do pliku oraz sprawdzające pewne warunki.

- Pola
  - `private int columns` – liczba kolumn w grafie
  - `private int rows` – liczba wierszy w grafie
  - `private LinkedList<Edge> adjList[]` – lista sąsiedztwa przechowująca krawędzie w grafie, tablica ma rozmiar `numberOfVertices`, każdy element zawiera listę
  - `private int numberOfVertices` – liczba wierzchołków w grafie.
- Metody
  - `public Graph( int columns, int rows )` – konstruktor; przypisuje wartości polom `columns`, `rows`, `numberOfVertices`, tworzy `adjList[]`
  - `public int getColumns()` – zwraca wartość pola `columns`
  - `public int getRows()` – zwraca wartość pola `rows`
  - `public int getNumberOfVertices()` – zwraca wartość pola `numberOfVertices`
  - `public void addEdge( Edge edge )` – dodaje do listy `edges` krawędź `edge`
  - `public Graph readGraph( File file )` – czyta graf z pliku, korzystając z klasy `FileChooser`
  - `public void writeGraph( File file )` – zapisuje graf do pliku `public`
  - `boolean doesHaveAllEdges()` – sprawdza, czy graf posiada wszystkie możliwe krawędzie `public`
  - `public Set<Integer> potencialNeighbors( int vertex )` – zwraca zbiór wierzchołków, które mogłyby sąsiadować z wierzchołkiem o numerze `vertex`
  - `public Set<Integer> neighbors( int vertex )` – zwraca zbiór wierzchołków, do których istnieje krawędź z wierzchołka `vertex`
  - `boolean containsEdge( int startVertexNumber, int endVertexNumber )` – zwraca `true` jeżeli w grafie znajduje się krawędź z wierzchołka o numerze `startVertexNumber` do wierzchołka o numerze `endVertexNumber`, w przeciwnym wypadku zwraca `false`
  - `public int getMaxPossibleNumberOfEdges()` – zwraca maksymalną liczbę krawędzi, jakie może zawierać graf
  - `public boolean isEdgeProper( Edge edge )` – zwraca wartość `true` jeżeli dana krawędź może istnieć na grafie i nie została jeszcze utworzona, zwraca `false` w przeciwnym wypadku.

**public class Generator** – klasa odpowiedzialna za generowanie grafu zgodnie z jednym z trzech dostępnych trybów.

- Pola
  - `private Graph graph` – graf utworzony w konstruktorze.
- Metody
  - `public Generator( int columns, int rows, double fromWeight, double toWeight )` – konstruktor, tworzy obiekt `graph`
  - `public Graph generateCompleteGraph( void )` – zwraca graf wygenerowany zgodnie z trybem 1: z wszystkimi możliwymi krawędziami (pomiędzy wierzchołkami sąsiadującymi poziomo lub pionowo) oraz z losowymi wagami krawędzi z zakresu (`fromWeight`,`toWeight`)

- `public Graph generateConnectedGraph( int startVertexNumber )` – zwraca graf wygenerowany zgodnie z trybem 2: spójny z losowymi wagami krawędzi z zakresu (`fromWeight,toWeight`), zaczynając generowanie od wierzchołka o numerze `startVertexNumber`; wywołuje metodę `RandomConnectedGraphGenerator( graph ).generate()`
- `public Graph generateRandomGraph( void )` – zwraca graf wygenerowany zgodnie z trybem 3: z losowo występującymi krawędziami (spójny lub niespójny) oraz z losowymi wagami krawędzi z zakresu (`fromWeight,toWeight`).

Sposób działania metody `generateRandomGraph`:

1. Wylosuj liczbę krawędzi do utworzenia większą od 0 i mniejszą lub równą `Graph.maxPossibleNumberOfEdges()`
2. Dodaj do zbioru wszystkie wierzchołki grafu
3. Wylosuj ze zbioru wierzchołek
4. Pobierz zbiór potencjalnych sąsiadów wierzchołka – `Graph.potentialNeighbors(int vertex)`
5. Wylosuj wierzchołek-sąsiad
6. (a) Jeżeli istnieje krawędź z wylosowanego wierzchołka do wierzchołka-sąsiada – `containsEdge( int startVertexNumber, int endVertexNumber )`, usuń wierzchołek-sąsiad ze zbioru sąsiadów, w przypadku wyczyszczenia zbioru sąsiadów, usuń wierzchołek ze zbioru wierzchołków i wykonaj czynności ponownie dla nowego wylosowanego wierzchołka
- (b) Jeżeli nie istnieje krawędź z wylosowanego wierzchołka do wierzchołka-sąsiada – `containsEdge( int startVertexNumber, int endVertexNumber )`, dodaj taką krawędź do grafu – jej waga jest losową liczbą z zakresu (`fromWeight, toWeight`)
7. Zwróć graf

**public class RandomConnectedGraphGenerator extends Generator** – klasa odpowiedzialna za generowanie losowego grafu spójnego zgodnie z algorytmem Prima.

- Pola

- `public RandomConnectedGraphGenerator( int columns, int rows, double fromWeight, double toWeight, int startVertexNumber )` – konstruktor; używa konstruktora klasy `Generator`
- `private int startVertexNumber` – wierzchołek, od którego ma się zaczynać generowanie grafu
- `private Set<Integer> visited` – zbiór wierzchołków, do których utworzono już połączenie w grafie
- `private Set<Integer> frontier` – zbiór wierzchołków, do których można bezpośrednio poprowadzić krawędź z jednego z wierzchołków ze zbioru `visited`.

- Metody

- `public Graph generate()` – generuje graf
- `private int visitedNeighbor( int vertex )` – zwraca numer losowego wierzchołka sąsiadującego z wierzchołkiem `vertex`, który znajduje się w zbiorze `visited`; w przypadku braku takich wierzchołków, zwraca wartość -1; korzysta z metody `Graph.potentialNeighbors( int vertex )`
- `private ArrayList<Integer> unvisitedNeighbors( int vertex )` – zwraca listę wierzchołków sąsiadujących z wierzchołkiem `vertex`, nienależących do zbioru `visited`.

Sposób działania metody `generate()`:

1. Wybierz losowy wierzchołek z grafu `graph`, dodaj go do zbioru `visited`

2. Dodaj do zbioru **frontier** wierzchołki sąsiadujące z wylosowanym wierzchołkiem, nieznajdujące się w zbiorze **visited** – metoda **visitedNeighbor( int vertex )**
3. Wykonuj, dopóki zbiór **frontier** nie jest pusty:
  - (a) Wybierz losowy wierzchołek ze zbioru **frontier** (wierzchołek 2)
  - (b) Wybierz losowy wierzchołek sąsiadujący z wierzchołkiem 2, znajdujący się w zbiorze **visited** – metoda **visitedNeighbor( int vertex )** (wierzchołek 1)
  - (c) Utwórz krawędź z wierzchołka 1 do wierzchołka 2 na grafie **graph**
  - (d) Usuń wierzchołek 2 ze zbioru **frontier**
  - (e) Dodaj wierzchołek 2 do zbioru **visited**
  - (f) Dodaj do zbioru **frontier** wierzchołki sąsiadujące z wierzchołkiem 2, nienależące do zbioru **visited** – metoda **unvisitedNeighbors( int vertex )**
4. Zwróć **graph**

**public class VertexWithDistance** – klasa reprezentująca wierzchołek używana do kolejki priorytetowej w algorytmie Dijkstry.

- Pola
  - **private int vertex** – number wierzchołka w grafie
  - **private int distance** – łączna odległość wierzchołka od początku grafu.
- Metody
  - **public VertexWithDistance( int vertex, int distance )** – Konstruktor
  - **public int getVertex()** – zwraca wartość pola **vertex**
  - **public double getDistance()** – zwraca wartość pola **distance**
  - **public void setDistance( double distance )** – ustawia wartość **distance**.

**public class Dijkstra** – służy do wyznaczania najkrótszej ścieżki pomiędzy jednym wierzchołkiem a drugim.

- Pola
  - **private Graph graph** – graf, na którym wykonywany jest algorytm Dijkstry
  - **private int start** – number wierzchołka, od którego mierzy się odległości w grafie
  - **private int[] predecessors** – tablica, w której wartość dla indeksu równoważnego z numerem wierzchołka w grafie przechowuje numer wierzchołka będącego poprzednikiem tego wierzchołka
  - **private int[] distances** – tablica łącznych najkrótszych odległości do danych wierzchołków od wierzchołka **start**
  - **private Set<Integer> checkedVerticesSet** – zawiera wierzchołki sprawdzone algorytmem – wyznaczono krawędzie biegnących z niego do sąsiadów
  - **private PriorityQueue<VertexWithDistance> uncheckedVerticesPQ** – kolejka zawierająca wierzchołki niesprawdzone algorytmem – nie wyznaczono krawędzi biegnących z niego do sąsiadów; priorytet  $1/distance$ , zdefiniowany w konstruktorze obiektu jako **Comparator**.
- Metody
  - **public Dijkstra( Graph graph )** – Konstruktor
  - **private void initiateValues()** – inicjuje wartości obiektu zgodnie z algorytmem

- `private void relax( int fromVertex, int toVertex )` – potencjalnie dodaje wierzchołek `toVertex` do tablic `predecessors[]` i `distances[]`, zgodnie z algorytmem
- `private void dijkstra()` – funkcja odpowiedzialna za wykonanie algorytmu Dijkstry; ustawia wartości w tablicy `predecessors[]` tak, aby wartość tablicy o danym indeksie równoważnemu numerowi wierzchołka odpowiadała numerowi wierzchołka będącego jego poprzednikiem; ustawia wartości w tablicy `distances` tak, aby wartość tablicy o danym indeksie równoważnemu numerowi wierzchołka odpowiadała łącznej najkrótszej odległości od wierzchołka `start`; wywołuje metody `initiateValues` i `relax`
- `public int determineShortestPath[] ( int fromVertex, int toVertex )` – zwraca tablicę zawierającą kolejne wierzchołki składające się na najkrótszą możliwą ścieżkę z wierzchołka `fromVertex` do wierzchołka `toVertex`; w przypadku braku połączenia między wierzchołkami zwraca wartość `null`.

Algorytm Dijkstry (z "Algorytmy dla grafów"):

```
inicjujNS1Z( Graf G, Wierzchołek s ) {
    Integer p[ G.liczbaWierzchołków() ]; // poprzednik
    Double o[ G.liczbaWierzchołków() ]; // odległość
    foreach( w : G.wierzchołki() )
        o[w]= INFTY;
        p[w]= 0;
    }
    o[s]= 0;
    return <o,p>;
}

relax( Graf G, Wierzchołek u, Wierzchołek v, Integer o, Integer p ) {
    if( o[u] > o[v] + G.waga(u,v) ) {
        o[u]= o[v] + G.waga(u,v);
        p[u]= v;
    }
}

aDijkstry( Graf G, Wierzchołek s ) {
    <o,p>= InicjujNS1Z( G, s );
    SET<Wierzchołek> w();
    PRIORITY_QUEUE<Wierzchołek> q( G.wierzchołki() ); // priorytet to 1/o[]
    while( ! q.empty() ) {
        u= q.get(); // z minimalną o
        w.add(u);
        for( v : G.sąsiednie(u) )
            relax( u, v, G );
    }
}
```

Uwaga!

Wagi muszą być nieujemne.

**public class Bfs** – implementacja algorytmu przeszukiwania wszerz (Breadth-first search – BFS), służącego do sprawdzania spójności grafu.

- Pola
  - `private Graph graph` – graf, na którym wykonywany jest algorytm BFS
  - `private Queue<Integer> fifo` – kolejka fifo

- private boolean visited[] – tablica odwiedzonych wierzchołków.
- Metody
  - public Bfs( Graph graph, int start ) - Konstruktor
  - public boolean checkConnectivity( Graph graph, int startVertexNumber ) – funkcja sterująca algorytmem BFS.

Algorytm BFS (z "Algorytmy dla grafów")::

```
szukaj_wszerz( Graf G, Wierzchołek s ) {
    Color c[ G.liczba_wierzchołków() ];
    Integer poprzednik[ G.liczba_wierzchołków() ];
    Integer l[ G.liczba_wierzchołków() ];
    foreach( w : G.wierzchołki() ) {
        c[w]= BIAŁY;
        l[w]= INFTY;
        poprzednik[w]= 0;
    }
    c[s]= SZARY; l[s]= 0;
    FIFO.empty();
    FIFO.put(s);
    while( ! FIFO.empty() ) {
        w= FIFO.get();
        foreach( v : G.sąsiednie(w) ) {
            if( c[v] == BIAŁY ) {
                c[v]= SZARY
                l[v]= l[w]+1
                poprzednik[v]= w
                FIFO.put(v);
            }
        }
        c[w]= CZARNY;
    }
}
```

## 5 Opis Klas GUI

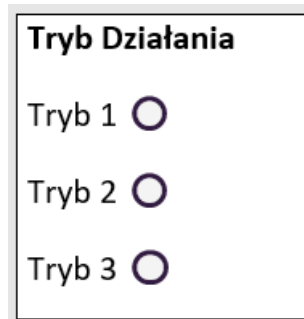
W poniższym rozdziale opisane są klasy odpowiedzialne za graficzny interfejs programu. Podobnie jak w rozdziale 4 zdefiniowane są pola oraz metody każdej z klas.

**public class Window extends Application** – klasa główna sterująca aplikacją.

- Pola
  - public static final int WINDOW\_WIDTH – szerokość okna
  - public static final int WINDOW\_HEIGHT – wysokość okna.
- Metody
  - public void start(Stage stage) – wczytuje scenę z pliku Window.fxml
  - public static void main(String[] args) – wywołuje metodę launch().

**public class WindowController** – klasa kontroler dla pliku Window.fxml.

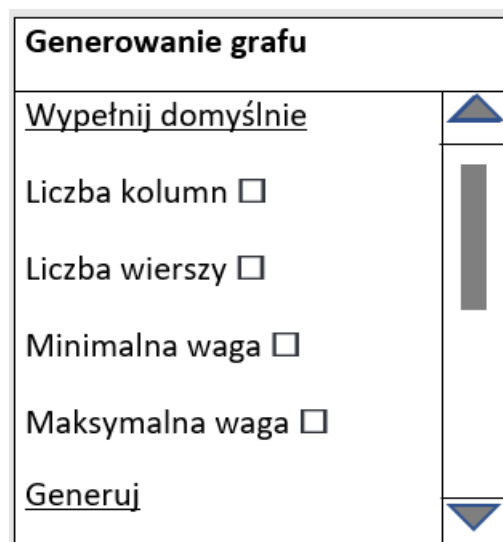
- Pola
  - private Graph graph – graf, który jest aktualnie rozpatrywany w programie
  - private GraphPane graphPane – kontener zawierający podgląd grafu
  - private ArrayList<PathInfo> pathsInfo – zbiór dróg na grafie wraz z informacjami o ich wi-  
doczności.



Rys.2 Zakładka "Tryb działania"

Poniżej opisane są pola dotyczące wyboru trybu działania programu:

- private ToggleGroup modesGroup – grupa przycisków w zakładce Tryb, zawiera: mode1Button, mode1Button, mode3Button
- private RadioButton mode1Button – przycisk Tryb 1 w zakładce Tryb działania
- private RadioButton mode2Button – przycisk Tryb 2 w zakładce Tryb działania
- private RadioButton mode3Button – przycisk Tryb 3 w zakładce Tryb działania

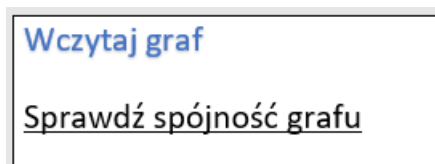


Rys.3 Zakładka "Generowanie grafu"

- private Hyperlink defaultFillHyperlink – hiperłącze Wypełnij domyślnie w zakładce Generowanie grafu, wypełnia domyślnymi wartościami: ...
- private TextField columnsTextField – pole tekstowe po etykiecie Liczba kolumn; użytkownik wpisuje w nim liczbę kolumn
- private TextField rowsTextField – pole tekstowe po etykiecie Liczba rzędów; użytkownik wpisuje w nim liczbę rzędów

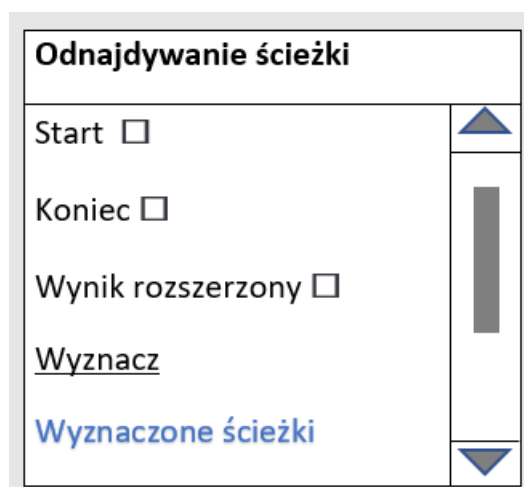


- `private TextField minWeightTextField` – pole tekstowe po etykiecie Minimalna waga; użytkownik wpisuje w nim minimalną wagę krawędzi
- `private TextField maxWeightTextField` – pole tekstowe po etykiecie Maksymalna waga; użytkownik wpisuje w nim maksymalną wagę krawędzi
- `private Hyperlink generateHyperlink` – hiperłącze Generuj, pobiera wartości z zakładki Tryb działania (`mode1Button`, `mode2Button`, `mode3Button` i Generowanie grafu (`columnsTextField`, `rowsTextField`, `minWeightTextField`, `maxWeightTextField`), na podstawie tych wartości wywołuje metody generujące graf (klasa `Generate`) i wyświetla go na podglądzie grafu (`graphPane`), wypisuje kolejne informacje w zakładce Komunikaty



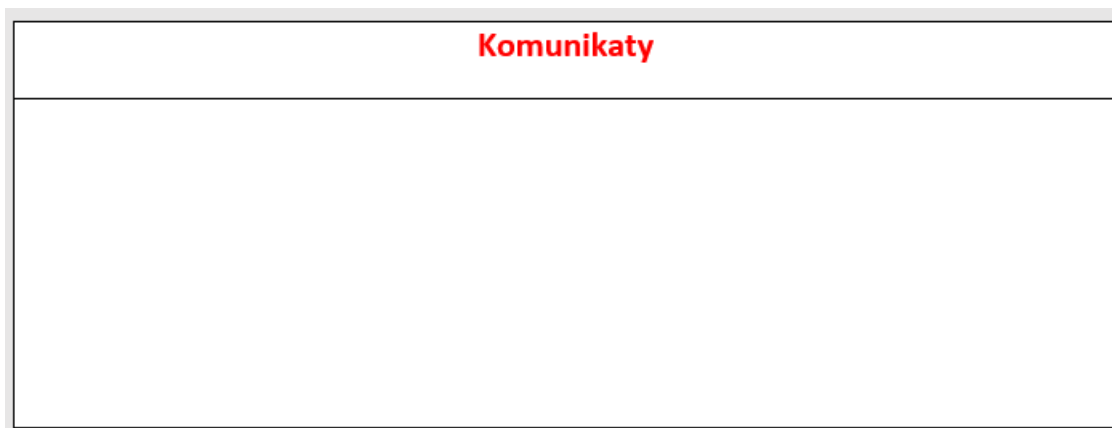
Rys.4 Zakładka "Wczytaj graf" / "Sprawdź spójność grafu"

- `private Hyperlink readGraphHyperlink` – hiperłącze Wczytaj graf; pole służące do wczytania grafu z struktury katalogów dostępnej na urządzeniu
- `private Hyperlink checkConnectivityHyperlink` – hiperłącze Sprawdź spójność grafu, wywołuje metodę sprawdzającą spójność grafu – `Bfs.checkConnectivity()`, wywołuje funkcję otwierającą okienko z informacją, wypisuje informację w zakładce Komunikaty



Rys.5 Zakładka "Odnajdywanie ścieżki"

- `private TextField startTextField` – pole tekstowe po etykiecie Start; użytkownik wpisuje w nim numer wierzchołka startowego od którego będzie szukana ścieżka
- `private TextField endTextField` – pole tekstowe po etykiecie Koniec; użytkownik wpisuje w nim numer wierzchołka końcowego do którego będzie szukana ścieżka
- `private CheckBox extendedResultCheckBox` – pole wyboru po etykiecie Wynik rozszerzony; w celu zobaczenia rozszerzonego wyniku użytkownik zaznacza pole wyboru Wynik rozszerzony.
- `private Hyperlink determineHyperlink` – hiperłącze Wyznacz w zakładce Odnajdywanie ścieżki, po kliknięciu następuje wyznaczenie ścieżki.
- `private Hyperlink determinedPathsHyperlink` – hiperłącze Wyznaczone ścieżki w zakładce Odnajdywanie ścieżki, po kliknięciu otwiera się nowe okno Wyznaczone ścieżki



Rys.6 Zakładka "Komunikaty"

- `private Label messagesLabel` – etykieta w zakładce **Komunikaty**, na której będą wypisywane komunikaty.
- Metody
  - `public static WindowController getWindowController()` – zwraca obiekt klasy kontrolujący widok FXML
  - `private void defaultFill( ActionEvent event )` – wypełnia pola domyślnymi wartościami: `columnsTextField` – 5, `rowsTextField` – 5, `minWeightTextField` – 0, `maxWeightTextField` – 1, w zakładce Tryb działania spośród grupy przycisków ustawia zaznaczony `mode3Button`, `startTextField` – 0, `endTextField` – jeżeli graf jest wczytany – numer wierzchołka w grafie o największym numerze, jeżeli nie – 0
  - `public void onGenerateClicked( ActionEvent event )` – wywołuje funkcje generujące graf po kliknięciu na przycisk Generuj
  - `private void readGraph( ActionEvent event )` – korzystając z klasy `FileChooser`, pozwala wybrać użytkownikowi plik zawierający graf ze struktury katalogów na urządzeniu, wywołuje metodę wczytującą graf – `Graph.readGraph( File file )`, wyświetla graf na podglądzie grafu – `graphPane`
  - `private void writeGraph( ActionEvent event )` – zapisuje graf do pliku korzystając z klasy `FileChooser` i metody `Graph.writeGraph( File file )`; domyślnie grafy zapisywane są do folderu `Graphs`, nazwa pliku jest wyrażona przez czas utworzenia pliku, np. `Graph-2022.05.10-09:15:34`, pobierana przez klasę `LocalDateTime`
  - `private void determinePath( ActionEvent event )` – wyznacza najkrótszą możliwą ścieżkę w grafie pomiędzy wierzchołkami o numerach pobranych z pól `startTextField` i `endTextField`; wywołuje metodę `Dijkstra.determineShortestPath( int fromVertex, int toVertex )`, zapisuje ścieżkę do ...
  - `private void showDeterminedPathsWindow( ActionEvent event )` – wyświetla nowe okno z informacjami o wyznaczonych ścieżkach, posiada możliwość zmiany na podglądzie grafu; tworzy nowy obiekt klasy `DeterminedPathsWindow` i wyświetla to okno
  - `int showConnectivityDialog()` – wyświetla nowe okno z informacją o spójności grafu; korzysta z klasy `Alert` i `Bfs`, wypisuje komunikat również do etykiety `messagesLabel`



Rys.7 Okno "Sprawdź spójność"

- `private void repaintScene()` – ładuje od nowa widok dla aktualnej sceny w celu odświeżenia zmian
- `private void addPath( PathInfo pathInfo )` – dodaje drogę do listy tablicowej `pathsInfo[]`
- `private void removePath(int number)` – usuwa drogę z listy tablicowej `pathsInfo[]` o numerze `number` w `pathsInfo[]`.

**public class GraphPane extends Pane** – kontener zawierający podgląd grafu.

- Pola
  - \* `private Graph graph`
  - \* `private ArrayList<Circle> vertices` – zawiera reprezentację wierzchołków w postaci okręgów, aby móc odczytywać ich lokalizację.
- Metody
  - \* `public GraphPane( Graph graph )` – konstruktor
  - \* `public void setGraph( Graph graph )` – ustawia wartość pola `graph`
  - \* `public void drawAll()` - wywołuje mniejsze metody rysujące w celu narysowania całości
  - \* `private void drawColumnsAndRowsNumbers()` – rysuje numery kolumn i wierszy
  - \* `private void drawVertices()` – rysuje wierzchołki w postaci okręgów
  - \* `private void drawEdges( boolean weightsVisibility )` – rysuje krawędzie między wierzchołkami; wywołuje metodę `drawEdgesArrow`
  - \* `private void draw EdgeArrow( Edge edge, int x1, int y1, int x2, int y2, boolean weightsVisibility )` – rysuje krawędzie w postaci strzałek od punktu `(x1,y1)` do punktu `(x2,y2)`
  - \* `private void getEdgeColor( Edge edge )` - przypisuje krawędzi kolor w zależności od jej wagi wg skali określonej na widoku.

**public class Path** – klasa przechowująca drogę na grafie, zawiera podstawowe informacje o niej.

- Pola
  - \* `private int fromVertex` – wierzchołek, od którego zaczyna się ścieżka
  - \* `private int toVertex` – wierzchołek, na którym kończy się ścieżka
  - \* `private int path[]` – numery kolejnych wierzchołków w ścieżce.
- Metody
  - \* `public Path( int fromVertex, int toVertex, int path[] )` – konstruktor, ustawia wartości pól `fromVertex`, `toVertex`, `path[]`
  - \* `public int getFromVertex()` – zwraca wartość pola `fromVertex`

- \* `public int getToVertex()` – zwraca wartość pola `toVertex`
- \* `public int getPath()` – zwraca nową tablicę będącą kopią pola `path[]`.


**public class PathInfo** – klasa przechowująca informacje o drodze na podglądzie grafu, wykorzystywana w klasie `DeterminedPathsWindow`

– Pola

- \* `private Path path` – zawiera podstawowe informacje o ścieżce
- \* `private boolean visibility` – widoczność na podglądzie grafu
- \* `private boolean weightVisibility` – widoczność wag na podglądzie grafu.

– Metody

- \* `public PathInfo( Path path )` – konstruktor
- \* `public boolean getVisibility()` – zwraca wartość pola `visibility`
- \* `public boolean setVisibility( boolean flag )` – ustawia wartość pola `visibility`.
- \* `public boolean getWeightVisibility()` – zwraca wartość pola `weightVisibility`
- \* `public boolean setWeightVisibility( boolean flag )` – ustawia wartość pola `weightVisibility`.

| Wyznaczone ścieżki |       |        |                                     |                                     |  |
|--------------------|-------|--------|-------------------------------------|-------------------------------------|---|
| Etykieta           | Start | Koniec | Widoczność                          | Widoczność wag                      | Usuń  |
| S1                 | 2     | 4      | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/>   |
| S2                 | 6     | 9      | <input type="checkbox"/>            | <input type="checkbox"/>            | <input checked="" type="checkbox"/>   |
| S3                 | 7     | 10     | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/>  |

Rys.8 Okno "Wyznaczone ścieżki"

**public class DeterminedPathsWindow extends Group**

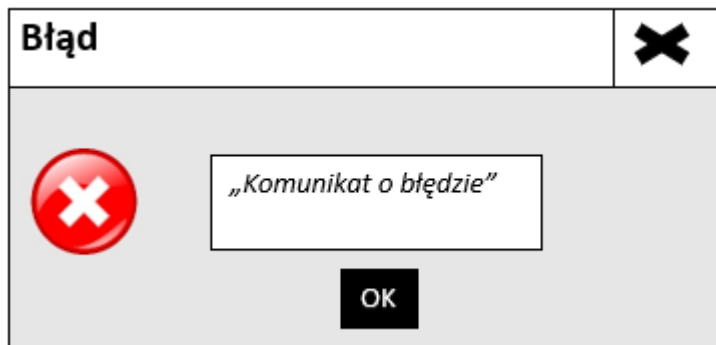
– Pola

- \* `private WindowController widnowController` – kontroler obiektu kontrolujący widok głównego okna z pliku `Window.fxml`
- \* `private GraphPane graphPane` – kontener zawierający podgląd grafu
- \* `private ArrayList<PathInfo> pathsInfo` – zbiór dróg na grafie wraz z informacjami o ich widoczności
- \* `private Hyperlink deleteHyperlink` – hiperłącze służące do usuwania zaznaczonych dróg
- \* `private ArrayList<Checkbox> visibilityCheckboxes` – pola wyboru w kolumnie `Widoczność` w oknie `Wyznaczone ścieżki`; użytkownik zaznacza pole wyboru aby zobaczyć ścieżki na diagramie
- \* `private ArrayList<Checkbox> weightVisibilityCheckboxes` – pola wyboru w kolumnie `Widoczność wag` w oknie `Wyznaczone ścieżki`; użytkownik zaznacza pole wyboru w celu zobaczenia wag krawędzi ścieżki na diagramie
- \* `private ArrayList<Checkbox> deleteCheckboxes` – pola wyboru w kolumnie `Usuń` w oknie `Wyznaczone ścieżki`; użytkownik zaznacza pole wyboru w celu zaznaczenia ścieżki do usunięcia.

– Metody

- \* `public DeterminedPathsWindow( WindowController )` – konstruktor, przez `windowController` pobiera `graphPane`, i `pathsInfo`

- \* `public void paint()` – dodaje do bieżącej klasy elementy graficzne, aby uzyskać efekt, jak na odpowiednim widoku, wykorzystuje do tego listę tablicową `pathsInfo`



Rys.9 Okno z komunikatem o błędzie

- \* `private int showWarningDialogRemovingPaths()` – wyświetla nowe okno z informacją o zamiarze usunięcia ścieżki, zwraca numer klikniętego przycisku: 1 – Tak, 2 – Nie; w przypadku zamknięcia okna zwracana jest wartość 2; korzysta z klasy `Alert`
- \* `private void deletePaths()` – usuwa z listy zaznaczone ścieżki zaznaczone `CheckBox`'ami.

## 6 Budowa wewnętrzna GUI

Menu programu zostanie zbudowane przy użyciu narzędzia `SceneBuilder` - będzie ono odczytywane przez program z pliku `.fxml`.

Podgląd na graf oraz okno Wyznaczone ścieżki zostaną zaprojektowane całkowicie w Javie, wykorzystując pobranie bieżącego obiektu klasy `Window` i modyfikowanie go w razie potrzeby.

## 7 Testowanie

Do realizacji testów jednostkowych zostanie wykorzystane narzędzie `JUnit`. Testy te będą znajdować się w osobnym pakiecie `TestPackage`. Przewiduje się następujące testy jednostkowe:

### 1. Klasa `Edge`

- (a) `public void EdgeTest()` – testuje konstruktor `public Edge( int from, int to )`.

### 2. Klasa `Graph`

- (a) `public void GraphTest()` – testuje konstruktor `public Graph( int columns, int rows )`
- (b) `public void addEdgeTest()` – testuje metodę `public void addEdge( Edge edge )`
- (c) `public void readGraph()` – testuje metodę `public void readGraph( File file )`
- (d) `public void writeGraph()` – testuje metodę `public void writeGraph()`
- (e) `public void doesHaveAllEdges()` – testuje metodę `boolean doesHaveAllEdges()`
- (f) `public void potentialNeighborsTest()` – testuje metodę `public Set<Integer> int[] potentialNeighbors( int vertex )`
- (g) `public void neighborsTest()` – testuje metodę `public Set<Integer> int[] neighbors( int vertex )`
- (h) `public void containsEdgeTest()` – testuje metodę `boolean containsEdge( int startVertexNumber, int endVertexNumber )`
- (i) `public void maxPossibleNumberOfEdgesTest()` – testuje metodę `public int maxPossibleNumberOfEdges()`
- (j) `public void isEdgeProperTest()` – testuje metodę `public boolean isEdgeProper( Edge edge )`.

### 3. Klasa `Generator`

- (a) `public void GeneratorTest()` – testuje konstruktor `public Graph generateConnectedGraph( int startVertexNumber )`
- (b) `public void public generateCompleteGraphTest()` – testuje metodę `public Graph generateCompleteGraph( void )`
- (c) `public void generateConnectedGraph()` – testuje metodę `public Graph generateConnectedGraph( int startVertexNumber )`
- (d) `public void generateRandomGraph()` – testuje metodę `public Graph generateConnectedGraph( int startVertexNumber )`.

#### 4. Klasa `RandomConnectedGraphGeneratorTest`

- (a) `public void public Graph generateTest()` – testuje metodę `public Graph generate()`
- (b) `public void visitedNeighborTest()` – testuje metodę `private int visitedNeighbor( int vertex )`
- (c) `public void unvisitedNeighborsTest()` – testuje metodę `private ArrayList<Integer> unvisitedNeighbors( int vertex )`.

#### 5. Klasa `VertexWithDistance`

- (a) `public void VertexWithDistanceTest()` – testuje konstruktor `public VertexWithDistance( int vertex, int distance )`.

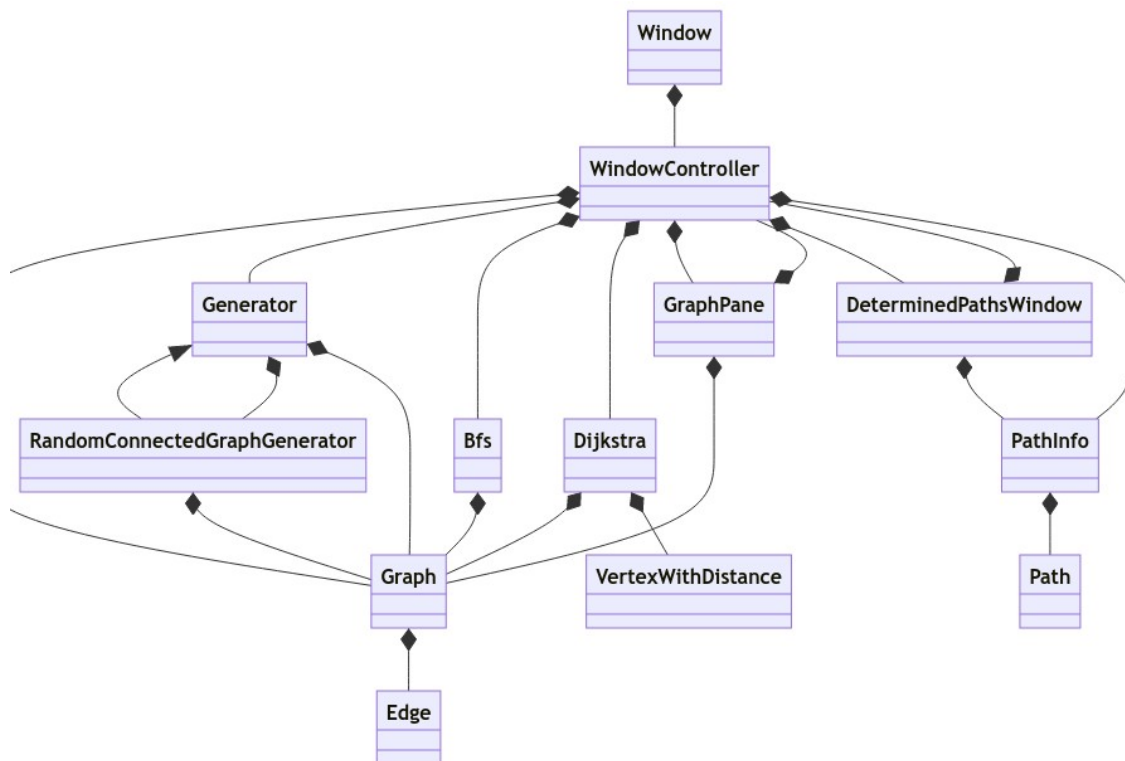
#### 6. Klasa `BFS`

- (a) `public void BfsTest()` – testuje konstruktor `public BFS( Graph graph, int start )`
- (b) `public void bfsCheckConnectivityTest()` – testuje metodę `public boolean checkConnectivity( Graph graph, int startVertexNumber )`.

#### 7. Klasa `Dijkstra`

- (a) `public void DijkstraTest()` – testuje konstruktor `(public Dijkstra( Graph graph )`
- (b) `public void initiateValuesTest()` – testuje metodę `private void initiateValues()`
- (c) `public void relaxTest()` – testuje metodę
- (d) `public void` – testuje metodę
- (e) `public void dijkstraTest()` – testuje metodę `private void dijkstra()`
- (f) `public void determineShortestPathTest()` – testuje metodę `public int determineShortestPath[]( int fromVertex, int toVertex )`.

## 8 Diagram klas



Rys.10 Diagram klas

## Literatura

- [1] Jacek Starzyński. *Prezentacja "Algorytmy dla grafów" na podstawie: Cormen, Leiserson, Rivest, Stein: "Wprowadzenie do algorytmów", WNT 2004*

Wszystkie rysunki są własne, rys.10 (diagram klas) został stworzony w aplikacji internetowej [mermaid.live](https://mermaid.live).