By Olena Chernilevska

1) Instructions to run code

To run the program you need to type mvn clean javafx:run in the console. After that you can regenerate javadocs, if necessary. To do it, please, type mvn javadoc:javadoc in the console.

2) Short summary

The functionality is split between 4 subpackages:

cz.cuni.mff.java.project.grapheditor.algorithms;

cz.cuni.mff.java.project.grapheditor.graphs;

cz.cuni.mff.java.project.grapheditor.editor;

cz.cuni.mff.java.project.grapheditor.listeners.

2.1 ***cz.cuni.mff.java.project.grapheditor.algorithms***

The package contains 7 public classes that include:

a) a public abstract class AbstractAlgorithm that implements basic interface for all algorithms that may inherit from it;

b) a public class BFS that extends AbstractAlgorithm and implements the BFS algorithm. The program allows to run algorithms manually and automatically; consequently, there are two key methods that handle this functionality. First, a public method search() runs the whole algorithm at once and is used to search for paths automatically. Second, a public method executeSearchStep() allows to execute only one logical step at a time and is used to search for paths manually. Other significant methods include: a public method result() that returns the result of search and a public method running() that checks if the algorithm is running;

c) a public class DFS that extends AbstractAlgorithm and implements the DFS algorithm. The class has the same methods as BFS; however, the output may differ from a BFS one because the path, found via DFS, is not necessary the shortest one;

d) a public class MinimumSpanningTree that extends AbstractAlgorithm and implements the minimum spanning tree algorithm. The class has the same methods as BFS; however, its output is the list of edges that form the minimum spanning tree;

e) a public class ShortestCycle that extends AbstractAlgorithm and implements the graph shortest cycle algorithm. The class has the same methods as BFS/DFS/spanning tree classes; at the same time, because of algorithm complexity it is hard to display the meaningful differences between steps and, consequently, manual search is disabled. This results in the absence of implementation for such helper methods as executeSearchStep(), visitedVertices(), frontierVertices(), frontierEdges();

f) a public class Node that exists only to store information about vertex parents during DFS and BFS searches;

g) a public class Path that gets a string with information about the found path and returns a list of edges that can be displayed on the screen.

2.2 *cz.cuni.mff.java.project.grapheditor.graphs*

The package contains 2 public classes that include:

a) a public class AbstractGraph that implements the basic representation of a graph. It 1) stores information about vertices and edges, 2) allows to add and delete vertices, 3) allows to add and delete edges, 4) checks if a vertex with a certain id exists, 5) retrieves the neighbors of a vertex with a certain id;

b) a public class Graph that extends AbstractGraph and implements additional functionality that allows to read/write graphs from/to .xml files.

2.3 *cz.cuni.mff.java.project.grapheditor.editor*

The package contains 2 public classes that include:

a) a public class Editor that is the key class of the program and extends javafx.application.Application. It configures the program window (creates and displays a menu bar and a canvas) and binds different listeners to different javafx events.

These events include:

KeyEvent.KEY_PRESSED (is handled by KeyBoardPressHandler);

KeyEvent.KEY_RELEASED (is handled by KeyBoardReleaseHandler);

MouseEvent.MOUSE_PRESSED (is handled by MouseClickHandler);

MouseEvent.MOUSE_DRAGGED (is handled by MouseDragHandler);

MouseEvent.MOUSE_RELEASED (is handled by MouseDragHandler).

Additionally, Editor binds menu bar items to the other set of listeners that contains:

NewGraphHandler (handles File > New);

OpenFileHandler (handles File > Open);

SaveFileHandler (handles File > Save);

AlgorithmHandler (handles all algorithms);

StopAlgorithmHandler (handles Algorithms > Stop/Clear status).

Furthermore, the class implements the set of drawing methods that allow redrawing the graph every time it is changed (vertex is added/deleted/dragged/selected or edge is added/deleted). Finally, the class stores references to the relevant instances of Graph and AbstractAlgorithm classes. In fact, public AbstractAlgorithm algorithm may refer only to one instance of any algorithm class (due to the shared interface they can be assigned to AbstractAlgorithm and called without any problem);

b) a public class PointD that stores information about point coordinates which are used to draw vertices.

2.4 *cz.cuni.mff.java.project.grapheditor.listeners*

The package contains 10 public classes that share a similar structure: they accept the reference of the instance of the Editor class to be able to handle certain events associated with that instance. These classes include:

a) a public abstract class AlgorithmHandler that builds a special form to retrieve information necessary to run algorithm (the start and end nodes of path the user is searching for, the way how algorithm should run - manually or automatically) and binds it with another listener - RunButtonHandler;

b) a public class RunButtonHandler starts the execution of an algorithm. If it is executed automatically, the RunButtonHandler waits for the final result and then sends it to the Editor instance to be displayed to the user. Otherwise, it simply starts algorithm execution that is later processed by KeyBoardReleaseHandler which listens to SLASH key being released;

c) a public class KeyBoardPressHandler handles keys DELETE, SHIFT and CONTROL. If DELETE is pressed and some vertex is selected, then that vertex and all edges connected to it will be deleted. After that, the graph will be redrawn. In case SHIFT

or CONTROL is pressed, the relevant variables in the Editor instance are set to true. As a result, mouse clicks are not ignored and certain actions may be performed (i. e. the addition of a vertex, the addition/deletion of an edge);

d) a public class KeyBoardReleaseHandler handles keys SLASH, SHIFT and CONTROL. If SLASH is released one step of the algorithm is executed and the result is displayed. In case SHIFT or CONTROL is released, the relevant variables in the Editor instance are set to false. As a result, mouse clicks will be ignored;

e) a public class MouseClickHandler that handles cases when a mouse button is pressed. First, it checks if CONTROL or SHIFT keys are pressed too, and if it's true the MouseClickHandler performs actions either with vertices, or with edges. If these keys are not pressed the code also checks if any vertex is selected and, if true, activates drag mode;

f) a public class MouseDragHandler handles the dragging of selected vertex;

g) a public class NewGraphHandler clears all graph-related information and shows an empty canvas on the screen;

h) a public class OpenFileHandler starts the file dialog, handles reading the graph-related information from the selected file, and draws the graph;

i) a public class SaveFileHandler starts the file dialog and handles saving the graph-related information to the selected file;

j) a public class StopAlgorithmHandler clears all algorithm-related information and redraws the graph.