

**Object Oriented Programming with  
Java - Advanced Course  
Project Documentation**

Group 9

February 10<sup>th</sup>, 2020

*(this page intentionally left blank)*

## Table of contents

0. Declaration of Authorship .....	4
1. Introduction .....	5
2. Group members .....	5
3. Project Documentation: .....	5
3.1. Project Introduction: .....	5
3.2. Requirements: .....	5
3.3. Technical description: .....	6
3.3.1. Input: .....	7
3.3.2. Process: .....	7
3.3.3. Output: .....	8
3.4. Implementations: .....	8
3.4.1. Input: .....	8
3.4.1.1. Retrieve user command – “Arguments” class: .....	8
3.4.1.2. Reading process – “GraphParser” class: .....	9
3.4.1.3. Graph implementation: .....	9
3.4.2. Process: .....	11
3.4.2.1. Main class: .....	11
3.4.2.2. Total nodes, edges, IDs: .....	11
3.4.2.3. “Connectivity” class: .....	12
3.4.2.4. “Diameter” class: .....	12
3.4.2.5. “ShortestPath” class: .....	13
3.4.2.6. “Betweenness Centrality” class: .....	13
3.4.3. Output: .....	13
3.4.3.1. “GraphWriter” class: .....	13
3.5. Data flow: .....	14
3.6. User Handbook: .....	14
3.6.1. General .....	15
3.6.2. Specific cases .....	15
3.7. Thread: .....	20
3.7.1. “ReadFileThread” class: .....	20
3.7.2. “ExecutingThread” class: .....	20
3.8. Exception: .....	20
3.9. Java Logging – “MyLogger” class: .....	21
4. Milestones: .....	21
4.1. Milestone 1: .....	21
4.2. Milestone 2: .....	23
4.3. Milestone 3: .....	24

## 0. Declaration of Authorship

I hereby declare that the submitted project is my own unaided work or the unaided work of our team. All direct or indirect sources used are acknowledged as references.

I am aware that the project in digital form can be examined for the use of unauthorized aid and in order to determine whether the project as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future projects submitted. Further rights of reproduction and usage, however, are not granted here.

This work was not previously presented to another examination board and has not been published.

First and Last Name

Frankfurt am Main, 10.02.2020

Nguyen Quynh Huong



Luu Nguyen Phat



Ngo Minh Thong



Tran Huu Le Huy



## 1. Introduction:

This document is about “Communication Network Analysis” project in the purpose of the “Object-oriented programming with Java – Advanced Course”. This project documentation refers to project introduction, technical description, designs, implementation, user manual and three milestone reports.

## 2. Group members:

1. Mr. Ngo Minh Thong
2. Ms. Nguyen Quynh Huong
3. Mr. Tran Huu Le Huy Huy
4. Mr. Luu Nguyen Phat

## 3. Project Documentation:

### 3.1. Project Introduction:

Nowadays, communication networks grow in importance and maintenance becomes an increasingly challenging task. In order to provide an overview over the network infrastructure, this project is created to aggregate and display information appropriately. By the use of advanced object-oriented concepts in Java programming language, the aim of this project is to create a program which analyses the graph-based communication network models. The program has fully functional, meets the given requirements and provides a good documentation for users can install, run and see the expected results.

The intended audience of this document is the course instructor, who will use it as the basis for a determination of a portion of our grade. The communication network model is based on basic graph definitions, people who are interested in graph also can use this as a tool to refer some specific properties of graph.

### 3.2. Requirements:

The project has to be satisfied those basic following requirements:

- The program is able to start from the command line interface (CLI) or graphical user interface (GUI) and read graph information (for examples: nodes, edges, weights) from a provided network model file (.graphml) with different sizes into corresponding Java data structures using an adequate clean Object-Orientation Programming.

- Based on the stored input data, the program will process and acquire the necessary information in order to implement required tasks to find such as:
  - Graph properties which are number of nodes, number of edges, the vertex names or IDs, the edge names or IDs, connectivity ( whether the graph is connected), diameter (the maximum distance between any two nodes considering the shortest path between them).
  - Shortest path between two vertices using Dijkstra algorithm.
  - Betweenness centrality for a selected node by the formula:

$$g(v_x) = \sum_{v_i \neq v_x \neq v_j} \frac{\sigma_{v_i v_j}(v_x)}{\sigma_{v_i v_j}}$$

with  $\sigma_{v_i v_j}$  being the total number of shortest paths from node  $v_i$  to  $v_j$  and  $\sigma_{v_i v_j}(v_x)$  being the number of shortest paths that pass through node  $v_x$ . This measure indicates how central a specific node is for all existing shortest path between all nodes in a given graph.

- All of the output results may be written into specified file(s) and / or displayed on the user interface (CLI or GUI)

And some additional requirements:

- The program was developed using an adequate object orientation.
- Information will be stored in suitable Java Collections or appropriate alternative data structures.
- The program contains adequate error handling.
- The program works with streams and files. At least one reading and writing file access has to be made.
- The program contains at least two threads.
- The program is developed using the clean code standard(s) as presented in the lecture.
- Appropriate logging is provided using `java.util.Logging` or comparable.

### 3.3. Technical description:

In this project, a developed program is possible to basically analyze a graph-based communication network model, from the **input** of a model over the appropriate **processing** of the model data to the **output** of the information. In details:

### 3.3.1. Input:

Via command line interface (CLI): users run the program, specify the input file and put arguments corresponding to properties which they want. The input file is a XML-based format file (*.graphml*).

The commands follow this format:

Run program	Input file	Properties	Output file
<code>java -jar ComNetAnalyze.jar</code>	<code>input.graphml</code>	<code>-a</code>	<code>output.graphml</code>

- Run program: required
- Input file name: required, specifies the input file.
- Properties: variety of properties as follows:

	total nodes, total edges, all node IDs and all edge IDs (leave properties field blank)
<code>-s x y</code>	shortest path between two nodes: x and y
<code>-b x</code>	betweenness centrality measurement of node x
<code>-a</code>	all properties

- Output file: specifies the output file when choose -a property.

After the input file is opened, the program continues to read then store the graph data in an object Graph – which is designed and implemented by basic data structures in Java Collections. The arguments passed by users will be transfer to the next step – Process – to "calculate" properties.

### 3.3.2. Process:

This "Process" receives arguments from "Input" step, then performs following tasks:

- Calculates number of nodes, edges and their identities (IDs)
- Determines whether the model is connected or not and calculate its diameter
- Finds shortest path between two vertices according to the Dijkstra algorithm (if specified)
- Calculates the betweenness centrality measure for a selected node (if specified).

Finally, it passes result of performed tasks to "Output".

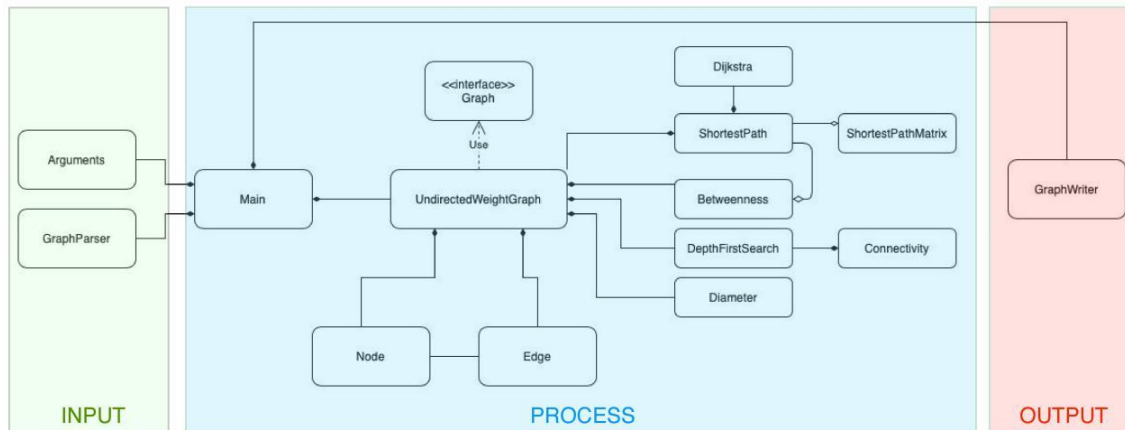
### 3.3.3. Output:

This step receives the result of previous step then output to the CLI and/or a file—specified from "Input".

### 3.4. Implementations:

The program has adequate object-oriented. The figure below shows relationship between all implemented classes. Based on their task, they are divided into 3 groups: Input, Process and Output.

Figure 1 - Relationship between classes



### 3.4.1. Input:

#### 3.4.1.1. Retrieve user command – “Arguments” class:

The program uses command line user interface (CLI). It is necessary to retrieve and parses arguments from user through CLI. A class called “Arguments” has been created for that purpose. In sequence: it receives arguments from main() entry and check those arguments whether they are in the correct format or not, then it returns back tasks for “Main” class.

Figure 2 - The “Arguments” class

Arguments
- taskAnalysed: ArrayList<ArrayList<String>> - outputFile: String - isInputvalid : boolean = true - fileName : String
+getTaskAnalysed(): ArrayList<ArrayList<String>> +getOutputFile(): String +getFileName(): String -checkEmptyArg(String[] taskArray): void -checkFileExistence(String fileName) -analyseS(String[] taskArray, int index): boolean -analyseB(String[] taskArray, int index): boolean -analyseA(String[] taskArray, int index): boolean +analyse(String[] taskArray): void



### 3.4.1.2. Reading process – “GraphParser” class:

The input file has extension “.graphml”, a XML-based file type. In this project, we have created a class called “GraphParser” to read XML elements from the input file (by using “Simple API for XML” – SAX). It parses XML elements to corresponding objects (which are defined in the graph implementation (3.4.1.3)). Then, a new graph is created and all objects are transferred to the graph.

Figure 3 – The structure of graphml file

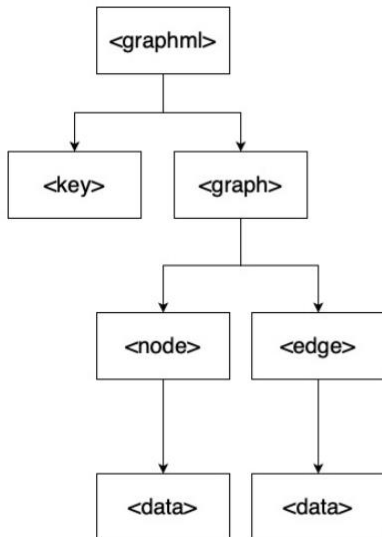
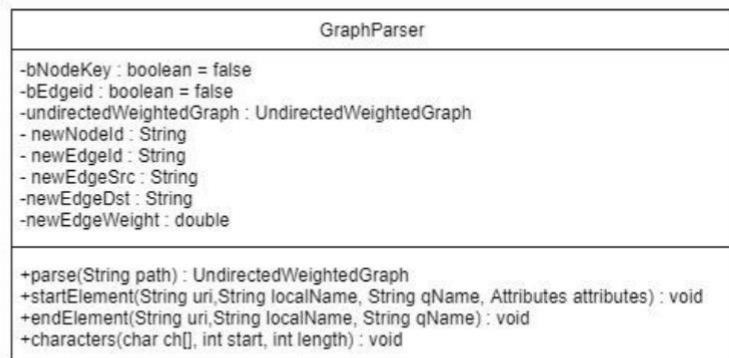


Figure 4 – The “GraphParser” class



- The `<graphml>` tag tells SAX should read it as a graphml element. Inside this element, there are nodes and edges – the essential of a graph. It also describes graph properties
- The `<key id="e_weight" for="edge" attr.name="weight" attr.type="double"/>` points out whether the edges of the graph is weighted or not.
- The `<graph>` tag contains graph information. Its attributes describe graph properties such as: undirected, directed, weighted ...
- The `<node>` tag contains node information:
  - `<data>`: the identification of itself.
- The `<edge>` tag contains edge information:
  - `<data>`: the identification of itself
  - `<data>`: weight of this edge.

### 3.4.1.3. Graph implementation:

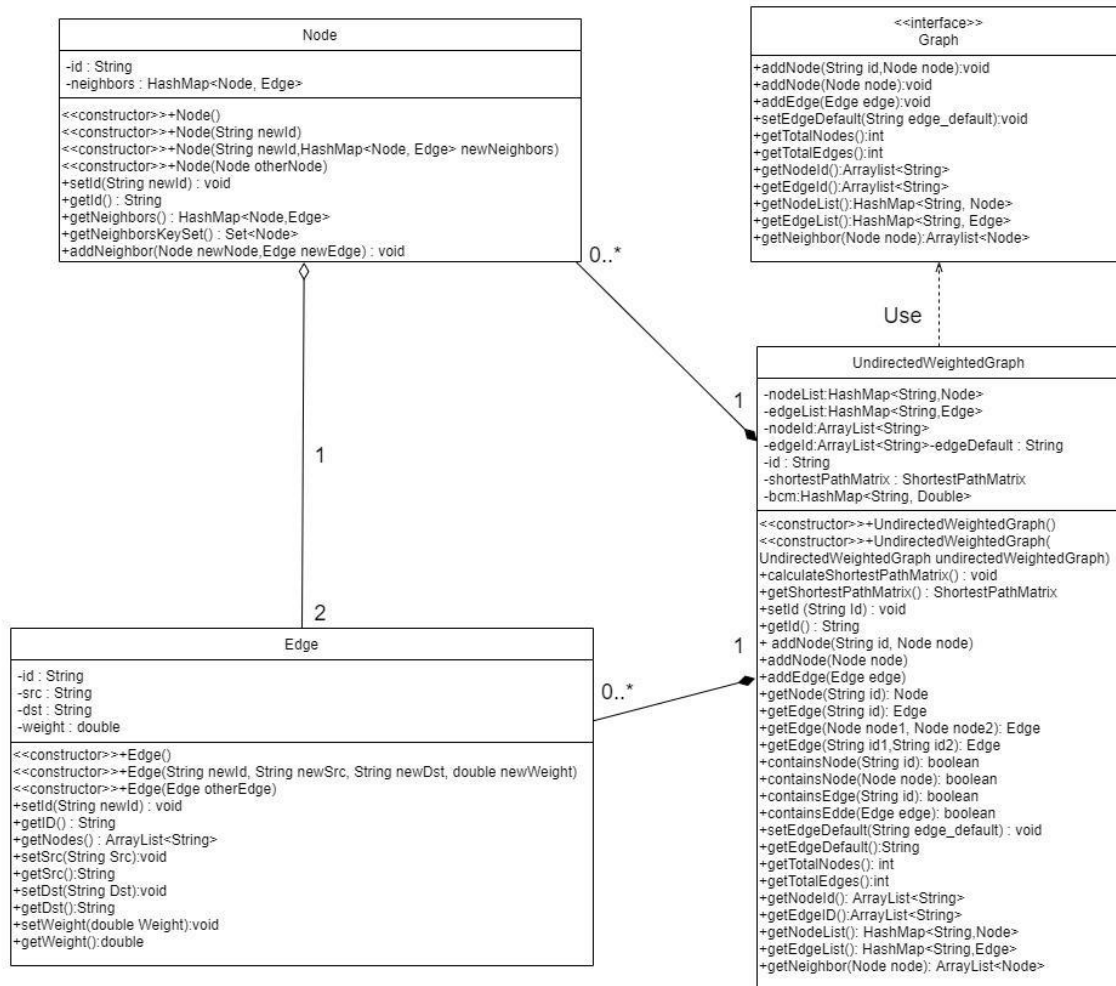


Figure 5 - Relationship between Graph, Node and Edge

Three objects have been created:

- Node: this object has 2 attributes: id and neighbours. id is its identity, make it unique to other nodes. neighbours is a list of pairs of its neighbour node and the edge connects it with that neighbour. It has 3 constructors: Node(), Node(String) and Node(String, Map<Node, Edge>) which does: creates a "null" node, a node with only its identity and a node has its identity, a list of pairs of its neighbour and a corresponding edge. Method getId returns its identity and method getNeighbours returns value of neighbours attribute.

- Edge: this object has 4 attributes: id, src, dst and weight. They are identity, the source node, the destination node and weight of the edge respectively. It has 2 constructors: Edge(), Edge(String, String, String, int) which does: create a "null" edge, create an edge with its identity, source node, destination node and weight respectively. Method getId() returns its identity, getNodes() returns a pairs of node of that edge, getSrc() returns its source node, getDst() returns its destination node.

- Graph interface: it is an interface of the graph instance (there are several types of graph such as directed, undirected, weighted, unweighted ... graph). These type of graphs will be inherited from it for future development of the program.

- `UndirectedWeightedGraph` inherited from `Graph` interface, it contains nodes and edges, two list of nodes and edges which are `nodeList` and `edgeList` respectively. `getTotalNodes()` returns total number of its nodes; `getTotalEdges()` returns total number of its edges; `addNode()` adds a new node, `addEdge()` adds a new `Edge` to graph; `getNodeID()` returns a list of IDs of all nodes, `getEdgeID()` returns a list of IDs of all edges; `getShortestPathMatrix()` returns `ShortestPathMatrix` object which contains all shortest paths of the graph. It is created for further calculation related to graph's shortest paths such as betweenness centrality measure.

There are many more methods of these class, all of them is documented in Javadoc of this project.

### **3.4.2. Process:**

This "Process" receives arguments from "Arguments" class and graph information from "GraphParser". Then it performs following tasks:

- Queries from graph object the number of nodes, edges and their identities (IDs)
- Determines whether the model is connected or not and calculate its diameter
- Finds shortest path between two vertices according to the Dijkstra algorithm (if specified)
- Calculates the betweenness centrality measure for a selected node (if specified)
- Send data for "GraphWriter" for output data (if specified).

#### **3.4.2.1. Main class:**

Main class calls the method `analyse()` from "Arguments" class to analyse the object arguments that typed in by user. If the arguments are satisfied the conditions and input file exists, from "GraphParser" class, the method `parse()` is called to start reading the input file and converting and saving the graph information into Java data structure as a graph object in "UndirectedWeightedGraph" class. Then the program processes and executes the requested tasks using stored graph and prints the output data on the CLI or GUI. In the case an output file is detected in the arguments, this class calls `exportToXml()` or `exportToText()` from "GraphWriter" class to write all the graph information which are nodes and their IDs, edges and their IDs and their weights, number of nodes and edges, graph connectivity, graph diameter, all shortest paths and betweenness centrality of all nodes into the file.

#### **3.4.2.2. Total nodes, edges, IDs:**

Graph contains `nodeList` and `edgeList`, both use `Map` data structure, by calling `nodeList.size()`, `edgeList.size()` we will get the total nodes, edges respectively. By looping through

nodeList, for each item we return its ID and finally we will got all IDs of nodes. So, we can return IDs of all edges in the same way

#### 3.4.2.3. “Connectivity” class:

This class takes a graph as parameter, it checks this graph whether it is connected or not. To check the connectivity of a graph, a common way is trying to traverse the graph. In this program we use Depth- First Search Algorithm to implement that. When then method getConnectivity() is called, a new graph name DFSTree will be created through the algorithm from a random node of the undirected graph. If the tree has all the nodes of the given graph or has the same total nodes then the graph is connected. Then, we can use method isConnected() to get the connectivity. Its return type is boolean, true for connected, otherwise it is not connected.

Figure 6 - The “Connectivity” class

Connectivity
-connectivity:boolean = 0
+getConnectivity():boolean +isConnected(UndirectedWeightedGraph graph):boolean

#### 3.4.2.4. “Diameter” class:

This class find diameter of the graph. By calling method calculate() it does... The method return type is double.

Figure 7 - The “Diameter” class

Diameter
-diameter:double
+calculate(UndirectedWeightedGraph graph):double

### 3.4.2.5. “ShortestPath” class:

This class find shortest path from a node to another node. The algorithm is used to find shortest path is Dijkstra.

Figure 8 - The “ShortestPath” class

ShortestPath
<pre>-src : String -dst : String -pathList : ArrayList&lt;ArrayList&lt;String&gt;&gt; -numOfPath : int =0 -length : double = 0</pre>
<pre>+setSrc(String startId) : void +setDst(String endId) : void +getSrc() : String +getDst() : String +getPathList() : ArrayList&lt;ArrayList&lt;String&gt;&gt; +getLength() : double +getNumOfPath() : int &lt;&lt;constructor&gt;&gt;+ShortestPath(ShortestPath sp) &lt;&lt;constructor&gt;&gt;+ShortestPath(UndirectedWeightedGraph graph, String startNodeId, String endNodeId) -findPathList(HashMap&lt;String , HashSet&lt;String&gt;&gt; precedence, String startId, String endId) : ArrayList&lt;ArrayList&lt;String&gt;&gt;</pre>

### 3.4.2.6. “Betweenness Centrality” class:

This class finds shortest paths for every pair of nodes of the graph. Then calculates the betweenness centrality of the node. It takes a graph and a node ID as parameters. It returns the betweenness centrality measure.

Figure 9 - The “BetweennessCentrality” class

BetweennessCentrality
<pre>-nodeId:String -bcm:double</pre>
<pre>+getNodeId():String +getBCM():double +BetweennessCentrality(UndirectedWeightedGraph graph, String nodeId)</pre>

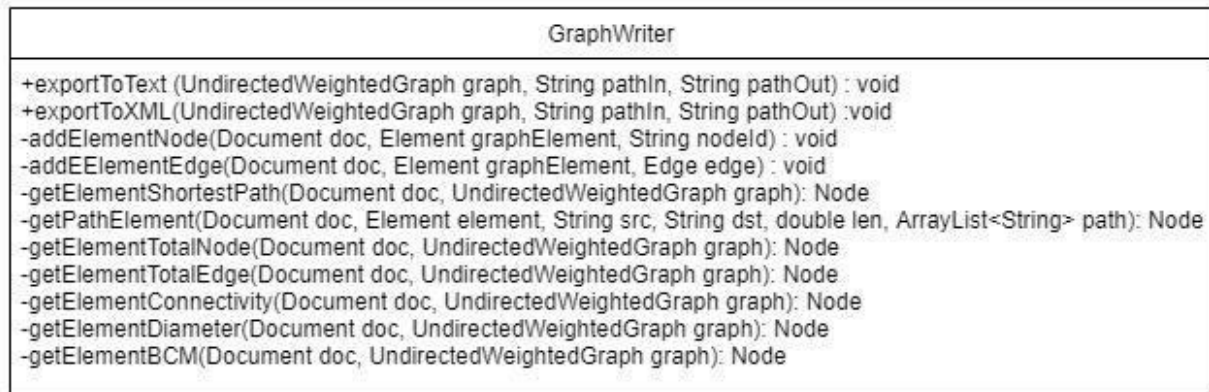
## 3.4.3. Output:

### 3.4.3.1. “GraphWriter” class:

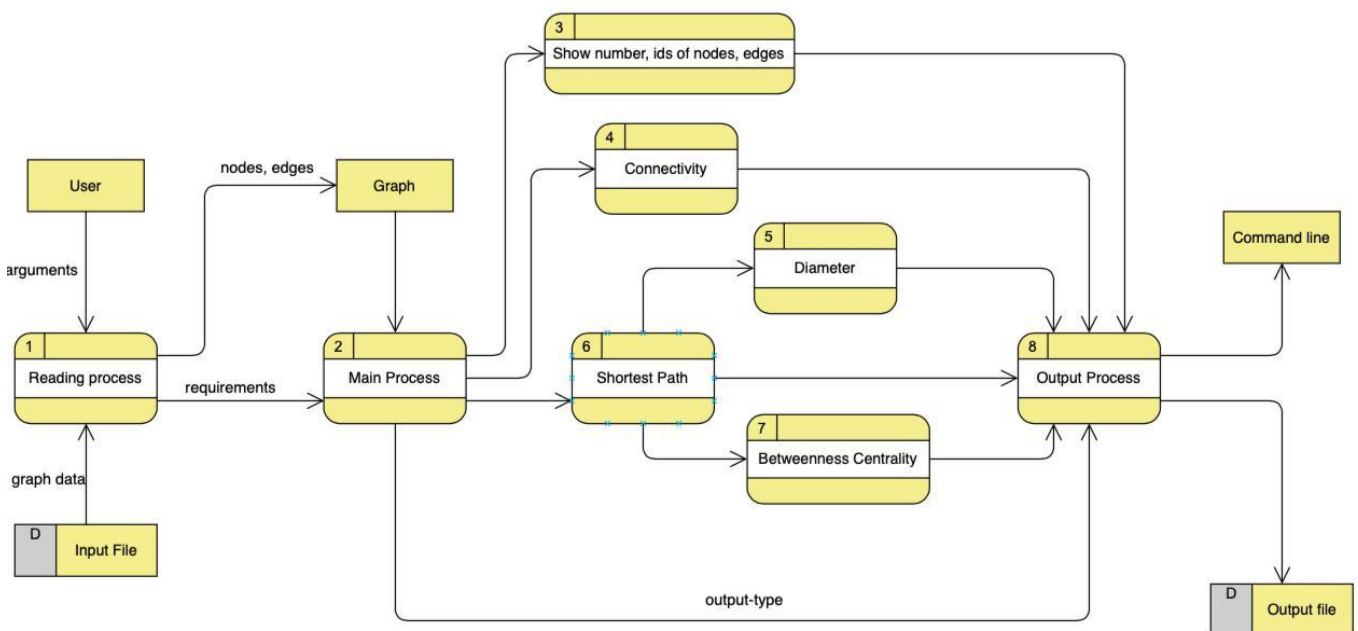
This class receives processed graph data, it writes data either to text file or to XML-based file. By calling exportToText(), we expect it write to a text file with normal format. Otherwise,

calling exportToXML() is to write XML format. They takes 3 parameters: graph, pathIn and pathOut which are the processed graph that need to be written, the input file and the output file respectively.

Figure 10 - The “GraphWriter” class



### 3.5. Data flow:

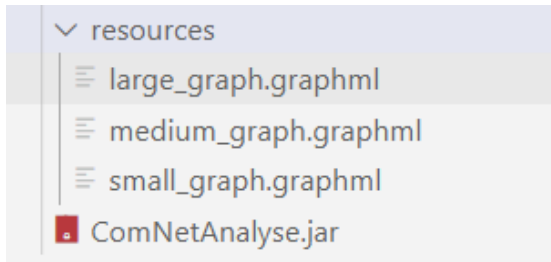


### 3.6. User Handbook:

#### System requirements:

- This program is developed in Java programming language, Java JDK is required to run the program.
- The program is packed in “ComNetAnalyse.jar” file, it requires a folder called “resources” contains input file .graphml, the folder should be placed in the same directory with the package.

For example:



### 3.6.1. General

- The program can only run in CLI. Make sure user's system can run jar file in CLI perfectly.

Run program	Input file	Properties
java -jar CommunicationNetworkAnalyse.jar	input.graphml	-a output.xml

The program's commands follow this format:

- Run program: required
- Input file name: required, specifies the input file.
- Properties: variety of properties as follows:

	total nodes, total edges, all node IDs and all edge IDs (leave properties field blank)
-s x y	shortest path between two nodes: "x" and "y"
-b z	betweenness centrality measurement of node "z"
-a filename	output all properties to file "filename"

- Output file: specifies the output file when choose -a property.

### 3.6.2. Specific cases

#### 🔧 CONDITION:

- Correct argument format
- Have existed input file

- ❖ **Case 1:** Print all the properties of the graph (number of nodes, number of edges, vertex IDs, edge IDs. Connectivity, Diameter):

- ✓ **The argument format:** java -jar project\_name.jar input\_filename.graphml

For example: java -jar ComNetAnalyse.jar large\_graph.graphml

```
C:\Users\Dong\Desktop>java -jar ComNetAnalyse.jar large_graph.graphml
Reading file large_graph.graphml
Processing content
### Graph attributes ###
Number of nodes: 250
Number of edges: 250
Vertex IDs: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249]
Edge IDs: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248]
Graph is connected
Graph diameter: 85.0
```

❖ **Case 2:** Print the betweenness centrality of a node of the graph:

- ✓ **The argument format:** `java -jar project_name.jar input_filename.graphml -b node_id`

For example: `java -jar ComNetAnalyse.jar large_graph.graphml -b 5`

```
C:\Users\Dong\Desktop>java -jar ComNetAnalyse.jar large_graph.graphml -b 5
Reading file large_graph.graphml
Processing content.....
### Betweenness centrality ###
Node 5: 17701.0
```

❖ **Case 3:** Print the shortest path between two nodes of the graph:

- ✓ **The argument format:** `java -jar project_name.jar input_filename.graphml -s node_id1 node_id2`

For example: `java -jar ComNetAnalyse.jar large graph.graphml -s 1 16`

```
C:\Users\Dong\Desktop>java -jar ComNetAnalyse.jar large_graph.graphml -s 1 16
Reading file large_graph.graphml
Processing content
### Shortest path ###
Shortest path 1 to 16: path -> [1, 8, 16]; length -> 11.0
```

❖ **Case 4:** Create an .graphml or .xml output file to store the graph and all graph properties, all shortest path of between all nodes, betweenness centrality of all nodes:

- ✓ **The argument format:** `java -jar project_name.jar input_filename.graphml -a output_filename.graphml`

For example: `java -jar ComNetAnalyse.jar small_graph.graphml -a output.graphml`

```
C:\Users\Dong\Desktop>java -jar ComNetAnalyse.jar small_graph.graphml -a output.graphml
Reading file small_graph.graphml
Processing content
### Output file ###
Written file(s): output.graphml
```

The output file will look like:

Total of nodes, Total of edges, Node IDs :



```
output - Notepad
File Edit Format View Help
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
  <key attr.name="id" attr.type="double" for="node" id="v_id"/>
  <key attr.name="id" attr.type="double" for="edge" id="e_id"/>
  <key attr.name="weight" attr.type="double" for="edge" id="e_id"/>
  <graph>
    <totalNode value="15"/>
    <totalEdge value="28"/>
    <node id="no">
      <data key="v_id">0</data>
    </node>
    <node id="n1">
      <data key="v_id">1</data>
    </node>
  </graph>
</graphml>
```

## Edge IDs, Connectivity, Diameter, Shortest Path, Betweenness Centrality:

```
output - Notepad
File Edit Format View Help
<edge source="n13" target="n14">
  <data key="e_id">27</data>
  <data key="e_weight">8.0</data>
</edge>
<connectivity value="true"/>
<diameter value="25.0"/>
<shortestPath>
  <path source="0" target="0" weight="0.0">[0]</path>
  <path source="0" target="1" weight="10.0">[0, 9, 3, 1]</path>
  <path source="0" target="2" weight="8.0">[0, 2]</path>
  <path source="0" target="3" weight="7.0">[0, 9, 3]</path>
</shortestPath>
<betweennessCentrality>
  <node id="0">15.0</node>
  <node id="1">0.0</node>
  <node id="2">0.0</node>
  <node id="3">36.5</node>
  <node id="4">0.0</node>
  <node id="5">15.0</node>
  <node id="6">19.0</node>
  <node id="7">12.0</node>
  <node id="8">16.0</node>
  <node id="9">13.5</node>
  <node id="10">27.5</node>
  <node id="11">0.0</node>
  <node id="12">0.0</node>
  <node id="13">0.0</node>
  <node id="14">15.5</node>
</betweennessCentrality>
</graph>
</graphml>
```

- ❖ **Case 5:** Create an .txt or .doc output file to store the graph and all graph properties, all shortest path of between all nodes, betweenness centrality of all nodes:

- ✓ **The argument format:** `java -jar project_name.jar input_filename.graphml -a output_filename.txt`

For example: `java -jar ComNetAnalyse.jar small_graph.graphml -a output.txt`

```
C:\Users\Dong\Desktop>java -jar ComNetAnalyse.jar small_graph.graphml -a output.txt
Reading file small_graph.graphml
Processing content
### Output file ###
Written file(s): output.txt
```

The output file will look like:

Total of nodes, Total of edges, Node IDs, Edge IDs, Connectivity, Diameter, Shortest Path:

```

Read in file: 'resources/small_graph.graphml'
### Graph information ###
    Number of nodes: 15
    Number of edges: 28
    Vertex IDs: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14]
    Edge IDs: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27]
    Graph is connected
    Graph diameter: 25.0
### Shortest paths ###
    Source node '0':
        To node '0': path -> [0]; length -> 0.0
        To node '1': path -> [0, 9, 3, 1]; length -> 10.0
        To node '2': path -> [0, 2]; length -> 8.0
        To node '3': path -> [0, 9, 3]; length -> 7.0
        To node '4': path -> [0, 9, 3, 8, 5, 4]; length ->

```

Betweenness centrality:

```

### Betweenness centrality ###
    Node '0': 15.0
    Node '1': 0.0
    Node '2': 0.0
    Node '3': 36.5
    Node '4': 0.0
    Node '5': 15.0
    Node '6': 19.0
    Node '7': 12.0
    Node '8': 16.0
    Node '9': 13.5

```

### 🚧 PROBLEMS HAPPEN IF:

#### ❖ Case 1: No arguments or input file:

For example: `java -jar ComNetAnalyse.jar`

```

C:\Users\Dong\Desktop>java -jar ComNetAnalyse.jar
There were no commandline arguments passed!

```

#### ❖ Case 2: Wrong input file name:

For example: `java -jar ComNetAnalyse.jar large_graph.graph`

```

C:\Users\Dong\Desktop>java -jar ComNetAnalyse.jar large_graph.graph
The input file does not exist.
Please make sure it is located at: "C:\Users\Dong\Desktop/resources/" folder.

```

#### ❖ Case 3: Wrong shortest path format arguments

For example:

- `java -jar ComNetAnalyse.jar large_graph.graphml -s 1`
- `java -jar ComNetAnalyse.jar large_graph.graphml -s`

```
C:\Users\Dong\Desktop>java -jar ComNetAnalyse.jar large_graph.graphml -s 1
The task -s has only 1 argument. 2 are needed.
```

```
C:\Users\Dong\Desktop>
```

```
C:\Users\Dong\Desktop>java -jar ComNetAnalyse.jar large_graph.graphml -s
The task -s has no argument. 2 are needed.
```

- java -jar ComNetAnalyse.jar large\_graph.graphml -s n1 n2
- java -jar ComNetAnalyse.jar large\_graph.graphml -s n1 2

```
C:\Users\Dong\Desktop>java -jar ComNetAnalyse.jar small_graph.graphml -s n1 n2
Reading file small_graph.graphml
Processing content
Node 'n1' does not exist in the graph.
```

```
C:\Users\Dong\Desktop>java -jar ComNetAnalyse.jar small_graph.graphml -s n1 2
Reading file small_graph.graphml
Processing content
Node 'n1' does not exist in the graph.
```

#### ❖ Case 4: Wrong betweenness centrality format arguments

For example:

- java -jar ComNetAnalyse.jar small\_graph.graphml -b 3 4
- java -jar ComNetAnalyse.jar small\_graph.graphml -b

```
C:\Users\Dong\Desktop>java -jar ComNetAnalyse.jar small_graph.graphml -b 3 4
The task -b has more than 1 argument.
```

```
C:\Users\Dong\Desktop>java -jar ComNetAnalyse.jar small_graph.graphml -b
The task -b has no argument. 1 are needed
```

- java -jar ComNetAnalyse.jar small\_graph.graphml -b n1

```
C:\Users\Dong\Desktop>java -jar ComNetAnalyse.jar small_graph.graphml -b n1
Reading file small_graph.graphml
Processing content
Node 'n1' does not exist in the graph.
```

#### ❖ Case 5: No output file name:

For example: java -jar ComNetAnalyse.jar small\_graph.graphml -a

```
C:\Users\Dong\Desktop>java -jar ComNetAnalyse.jar small_graph.graphml -a
The task -a has no argument. 1 are needed.
```

#### ❖ Case 6: Wrong argument format

For example:

- java -jar ComNetAnalyse.jar small\_graph.graphml -d
- java -jar ComNetAnalyse.jar -d small\_graph.graphml
- java -jar ComNetAnalyse.jar -s small\_graph.graphml
- java -jar ComNetAnalyse.jar 1 small\_graph.graphml

```
C:\Users\Dong\Desktop>java -jar ComNetAnalyse.jar small_graph.graphml -d
Unknown task -d. Try -s, -a or -b.

C:\Users\Dong\Desktop>java -jar ComNetAnalyse.jar -d small_graph.graphml
The input file does not exist.
Please make sure it is located at: "C:\Users\Dong\Desktop/resources/" folder.

C:\Users\Dong\Desktop>java -jar ComNetAnalyse.jar -s small_graph.graphml
The input file does not exist.
Please make sure it is located at: "C:\Users\Dong\Desktop/resources/" folder.

C:\Users\Dong\Desktop>java -jar ComNetAnalyse.jar 1 small_graph.graphml
The input file does not exist.
Please make sure it is located at: "C:\Users\Dong\Desktop/resources/" folder.
```

```
- java -jar ComNetAnalyse.jar 1 small_graph.graphml -
```

```
C:\Users\Dong\Desktop>java -jar ComNetAnalyse.jar small_graph.graphml -
Unknown task -. Try -s, -a or -b.
```

### 3.7. Thread:

#### 3.7.1. “ReadFileThread” class:

This class implements the Runnable interface. The program will use it when it begins to read the input file, run another thread and print into the command line:

```
Reading file small_graph.graphml
```

to tell the user that the program is still in the reading file process. After finishing the process, the thread is interrupted by interrupt() method.

#### 3.7.2. “ExecutingThread” class:

This class extends the Thread class. In the program all output will be execute before it prints the results into the command line. In this process it call another thread and print into the command line:

```
Processing content.....
```

to let the user know that they have to wait more for the output. After finishing the process, the thread is interrupted by interrupt() method.

### 3.8. Exception:

#### “NotFoundException” class:

Since the user works directly with the node id of some nodes of the graph in the input argument, it would be necessary to check if the graph contains a node with that id or not to avoid a NulPointerException error. This class was created to throws an exception and stop the program before more methods is running.

### 3.9. Java Logging – “MyLogger” class:

The program supports Java Logging for debug purpose:

A ‘MyLogger’ class is created. It writes program logs to the file “logfile.log”, which is in the same parent directory of ComNetAnalyse.jar file, instead of output system stream. However, it can be extended for future purpose with other different logging handler (ConsoleHandler, StreamHandler, SocketHandler and MemoryHandler).

- To create new logger instance, put this line inside a class we want to log:

```
MyLogger LOGGER = new MyLogger()
```

- To write logger information:

```
LOGGER.info(“your_message”)
```

- To write logger warning:

```
LOGGER.warn(“warning_message”)
```

## 4. Milestones:

### 4.1. Milestone 1:

#### Previous Background:

As this is our first progress report, we have little to report as previous background. Prior to this reporting period we:

- Discussed and analysed project's requirements.
- Designed the user interface and objects of the project.
- Created tasks, milestone and assigned tasks to team members.
- Used GitHub to control versions of source code.

#### Work Completed:

During this reporting period we have accomplished the following:

Task ID	Task description	Accomplished by
1	Reading input from users and parse arguments.	<u>Huong</u>
2	Load network model from user-specified file using regular expression	<u>Phat</u>
3	Store the given graph model by using HashMap in Java Collections	<u>Phat and Thong</u>
4	Implemented the algorithm determine connectivity of the graph by using Depth First Search.	<u>Thong</u>
5	Output number of nodes, edges and their identities (IDs).	<u>Huy</u>
6	Started writing project documents and designing related diagrams.	Team

#### **Work Scheduled:**

During the next reporting period, we plan to:

- Summarise reports from members and combine their works to project.
- Present the progress of the project have done by team

During the subsequent weeks, we plan to:

- Implement the Dijkstra algorithm to find shortest path between two given nodes
- Implement the function to find diameter of the graph
- Code the function to measure "betweenness centrality" of the graph
- Continue writing document about completed works.
- Improve some implementations of algorithm.
- Organize team meeting to discuss about the progress of working project.

#### **Problems Encountered:**

The items below have been resolved:

- Error appears when checking connectivity of big graph
- Some members have trouble with cloning, committing and pushing repository from GitHub to Eclipse IDE.

#### **Changes in Requirements:**

There have been no changes in the initial requirements.

#### **Overall Assessment of the Project:**

The project is going well and assigned tasks have been done before the first milestone deadline. Project document is on progress of writing. All problems have been solved and some improvements on the code were made.

## 4.2. Milestone 2:

### Previous Background:

From the first milestone submission, we have done:

- Read input from users and parse arguments
- Load network model from user-specified file using regular expression
- Store the given graph model by using HashMap in Java Collections
- Implemented the algorithm determine connectivity of the graph by using Depth First Search
- Output number of nodes, edges and their identities (IDs)
- Started writing project documents and designing related diagrams

### Work Completed:

During this reporting period we have accomplished the following:

Task ID	Task description	Accomplished by
1	Implement the Dijkstra algorithm to find shortest path between two given nodes	<u>Phat</u>
2	Implement the function to find diameter of the graph	<u>Huy</u>
3	Code the function to measure "betweenness centrality" of the graph	<u>Thong</u>
4	Continue writing document about completed works.	<u>Huong</u>
5	Reorganize the code structure	<u>Thong</u>
6	Write some test for testing and debugging	<u>Phat</u>
7	Discuss about threads and integrate it for the Shortest path and Diameter algorithm for better performance	<u>Huy</u>
8	Start to link codes written by team members	Team
9	To execute the project, we exported the project to .jar file and test it in CLI	<u>Huong</u>

### Work Scheduled:

During the subsequent weeks, we plan to:

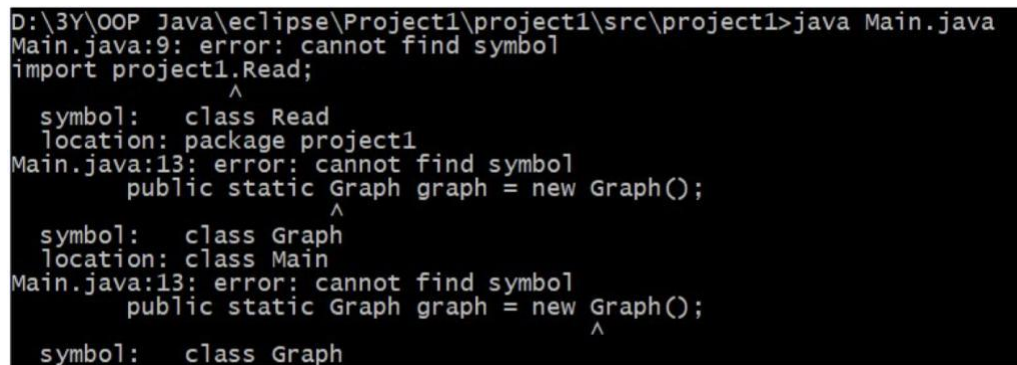
- Write Unit test and debug for the entire project
- Go over and double-check the source code
- Apply clean code techniques for future usage understandably
- Create Javadoc for other developers get the general idea from the methods

- Write friendly user manual
- Complete the final project documentation
- Export the project to .jar file for the submission

### Problems Encountered:

These problem below have been found:

- Unable to execute the project from CLI but in IDE (Eclipse) it works well



```
D:\3Y\OOP Java\eclipse\Project1\project1\src\project1>java Main.java
Main.java:9: error: cannot find symbol
import project1.Read;
            ^
    symbol:   class Read
    location: package project1
Main.java:13: error: cannot find symbol
    public static Graph graph = new Graph();
                        ^
    symbol:   class Graph
    location: class Main
Main.java:13: error: cannot find symbol
    public static Graph graph = new Graph();
                        ^
    symbol:   class Graph
```

- Unable to export .java file to .jar file

### Changes in Requirements:

There have been no changes in the initial requirements.

### Overall Assessment of the Project:

The project is going on due and assigned tasks had been done before the second milestone deadline. Project document is on progress of writing. New methods were inserted into the project and we are working on solving all the problems.

## 4.3. Milestone 3:

### Previous Background:

From the second milestone submission, we have done:

- Implement the Dijkstra algorithm to find shortest path between two given nodes
- Implement the function to find diameter of the graph
- Code the function to measure "betweenness centrality" of the graph
- Continue writing document about completed works.
- Reorganize the code structure
- Write some test for testing and debugging
- Discuss about threads and integrate it for the Shortest path and Diameter algorithm for better performance



- Start to link codes written by team members
- To execute the project, we exported the project to .jar file and test it in CLI

### Work Completed:

During this reporting period we have accomplished the following:

Task ID	Task description	Accomplished by
1	Graph implement: Create new interface called graph which implemented by new class called UndirectedWeightedGraph	<u>Thong</u>
2	Create new classes to calculate : Connectivity, Dijkstra, BetweennessCentrality, ShortestPath, Diameter instead of creating methods to calculate them in the last Milestone	<u>Phat</u>
3	Write Unit test and debug for the entire project	<u>Huong</u>
4	Go over and double-check the source code	<u>Team</u>
5	Apply clean code techniques for future usage understandably	<u>Phat</u>
6	Create Javadoc for other developers get the general idea from the methods	<u>Huy</u>
7	Create User handbook for user to know how to user the program	<u>Huy</u>
8	Complete the final project documentation	Team
9	Export the project to .jar file for the submission	<u>Huong</u>

### Problems Encountered:

These problem below have been found:

Failed to implement finding all shortest paths with Thread: Occasionally, there were some shortest paths are not calculated (skipped) while running with Thread.

### Changes in Requirements:

There have been no changes in the initial requirements.

### Overall Assessment of the Project:

The project is done on due and the rest of assigned tasks had been done until the final milestone deadline. We have not been able to solve the problem yet. Our group had a meeting to complete the final project documentation and final check the entire source code for final submission.