Dependencies Used (Imported VIA maven):

- GraphViz
- Slf4j-api
- Junit

To import the dependencies add them to the pom.xml file like so:

```xml
<dependencies>
    <dependency>
        <groupId>guru.nidi</groupId>
        <artifactId>graphviz-java</artifactId>
        <version>0.18.1</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>2.0.6</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-simple</artifactId>
        <version>2.0.6</version>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter</artifactId>
        <version>5.9.2</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>5.9.2</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>RELEASE</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Features:

GraphParse(String filePath): This takes in a file path and creates a directed graph object
MAKE SURE TO PASS IN THE ABSOLUTE FILE PATH, if you do not it will throw an error if the file is not found in the root project folder

Using GraphParse(String filePath) with file in root project folder example:

```java
import java.io.IOException;

no usages
public class Main {
    no usages
    public static void main(String[] args) throws IOException {
        String path = "input.dot";
        GraphParse graph;
        graph = new GraphParse( filePath: "input.dot");
    }
}
```

Using GraphParse with file NOT in root project folder example:

```java
import java.io.IOException;

no usages
public class Main {
    no usages
    public static void main(String[] args) throws IOException {
        GraphParse graph;
        graph = new GraphParse( filePath: "C:\\Users\\Dimitar\\IdeaProjects\\CSE464P1\\src\\main\\java\\color.dot");
    }
}
```

String toString(): Returns the graph statistics as a string

```java
System.out.println(graph.toString());
```

void outputGraph(String filePath): outputs the toString string to the specified file
MAKE SURE TO PASS IN THE ABSOLUTE FILE PATH, if you do not it will throw an error if the file is not found in the root project folder
Using outputGraph with file NOT in root project folder:

```java
graph.outputGraph( filePath: "C:\\Users\\Dimitar\\IdeaProjects\\CSE464P1\\example\\outputgraphtest.txt");
```

void addNode(String label): adds a node to the graph with the specified label used as input, will not add the new node if the node already exists

```java
graph.addNode( label: "x");
```

void removeNode(string label): removes the specified node if it exists, even if node does not exist it will still print that the node has been removed.

```java
graph.removeNode( label: "x");
```

void addNodes(String[] labels): Takes in a string array of node labels and adds them to the graph, will not add duplicates.

```
String[] nodesToAdd = {"x","y","z"};
graph.addNodes(nodesToAdd);
```

void removeNodes(String[] labels): Takes in a string array of node labels and removes the nodes with those labels.

```
String[] nodesToRemove = {"x","y","z"};
graph.removeNodes(nodesToRemove);
```

void addEge(String srcLabel, String dstLabel): Takes in two node labels and creates an edge between them, if one of the nodes does not exist the node is created and then the edge is added

```
graph.addEdge( srcLabel: "a", dstLabel: "d");
```

void removeEdge(String srcLabel, String dstLabel): Takes in two node labels and removes the edge between them.

```
graph.removeEdge( srcLabel: "a", dstLabel: "d");
```

void outputDOTgraph(String filePath): Takes in the file path and outputs the DOT file of the graph, file is created if it does not exist.

```
graph.outputDOTGraph( path: "output1.dot");
```

void outputGraphics(String filePath): Takes in the file path and outputs the visual created from the dot file.

```
graph.outputGraphics( path: "output3.png");
```

Example of calling all features from main:

```java
import java.io.IOException;

no usages
public class Main {
    no usages
    public static void main(String[] args) throws IOException {
        GraphParse graph;
        graph = new GraphParse( filePath: "input.dot");
        System.out.println(graph.toString());
        graph.outputGraph( filePath: "outputgraphtest.txt");
        graph.addNode( label: "x");
        graph.removeNode( label: "x");
        String[] nodesToAdd = {"x","y","z"};
        graph.addNodes(nodesToAdd);
        String[] nodesToRemove = {"x","y","z"};
        graph.removeNodes(nodesToRemove);
        graph.addEdge( srcLabel: "a", dstLabel: "d");
        graph.removeEdge( srcLabel: "a", dstLabel: "d");
        graph.outputDOTGraph( path: "output1.dot");
        graph.outputGraphics( path: "output3.png");

    }
}
```