Job Sheet 15 Graph

From:

AL AZHAR RIZQI RIFA’I FIRDAUS

Class:

1 I

Absence:

01

Student Number Identity:

2241720263

Department:

Information Technology

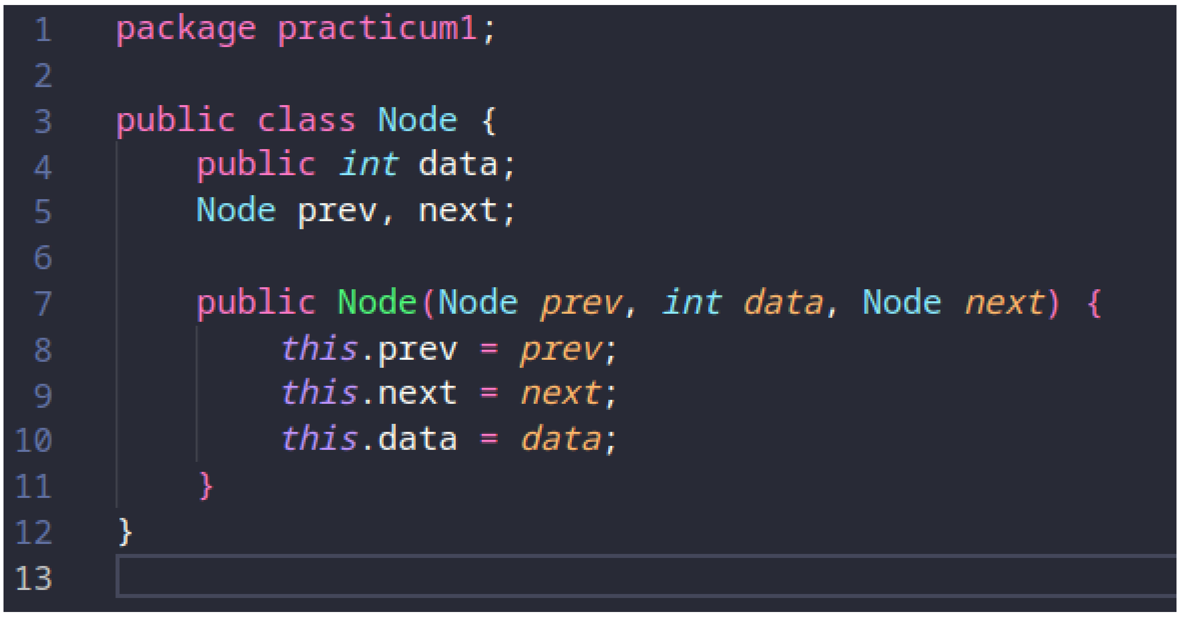
Study Program:

Informatics Engineering

Practicum 1 :

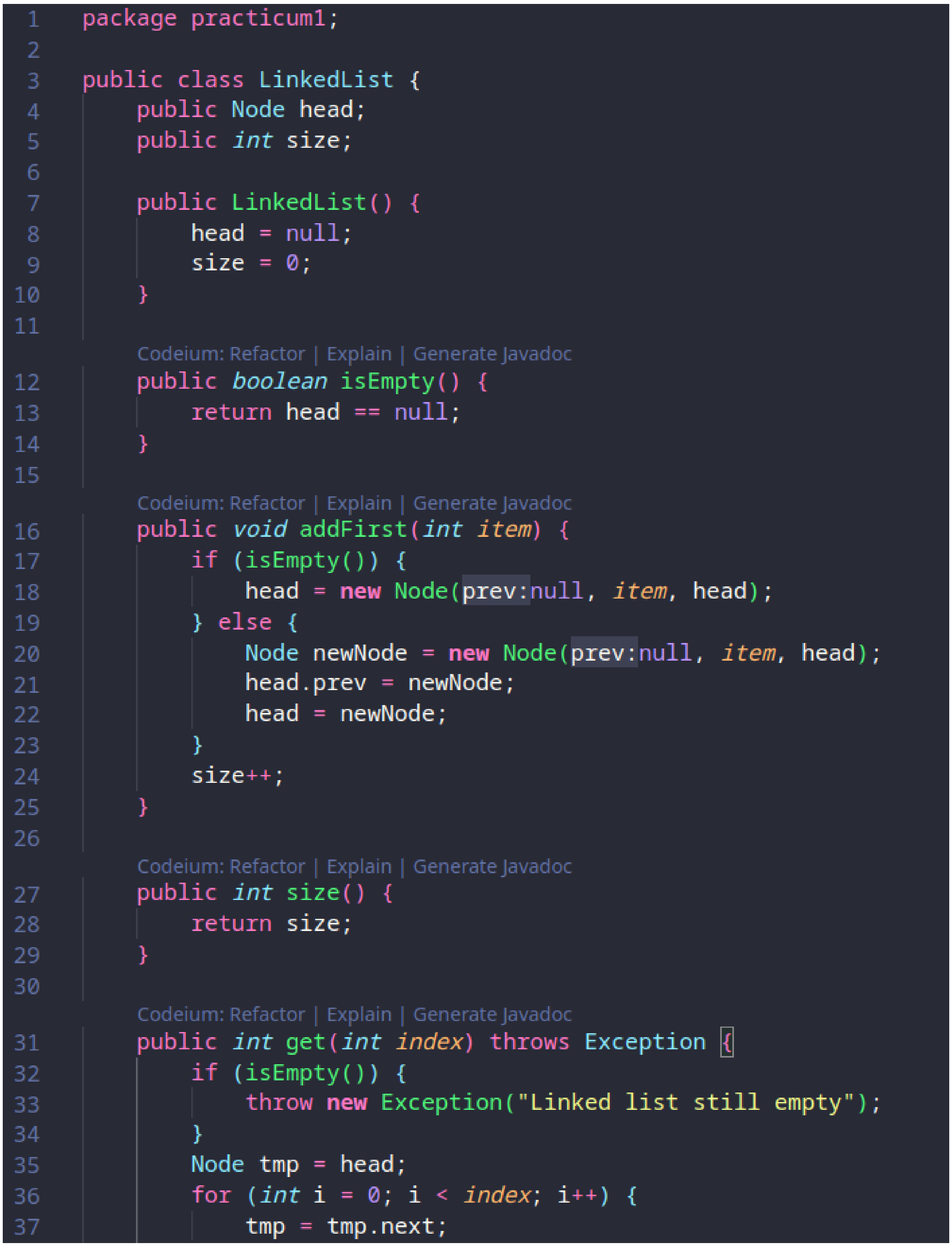
Code :

- Node

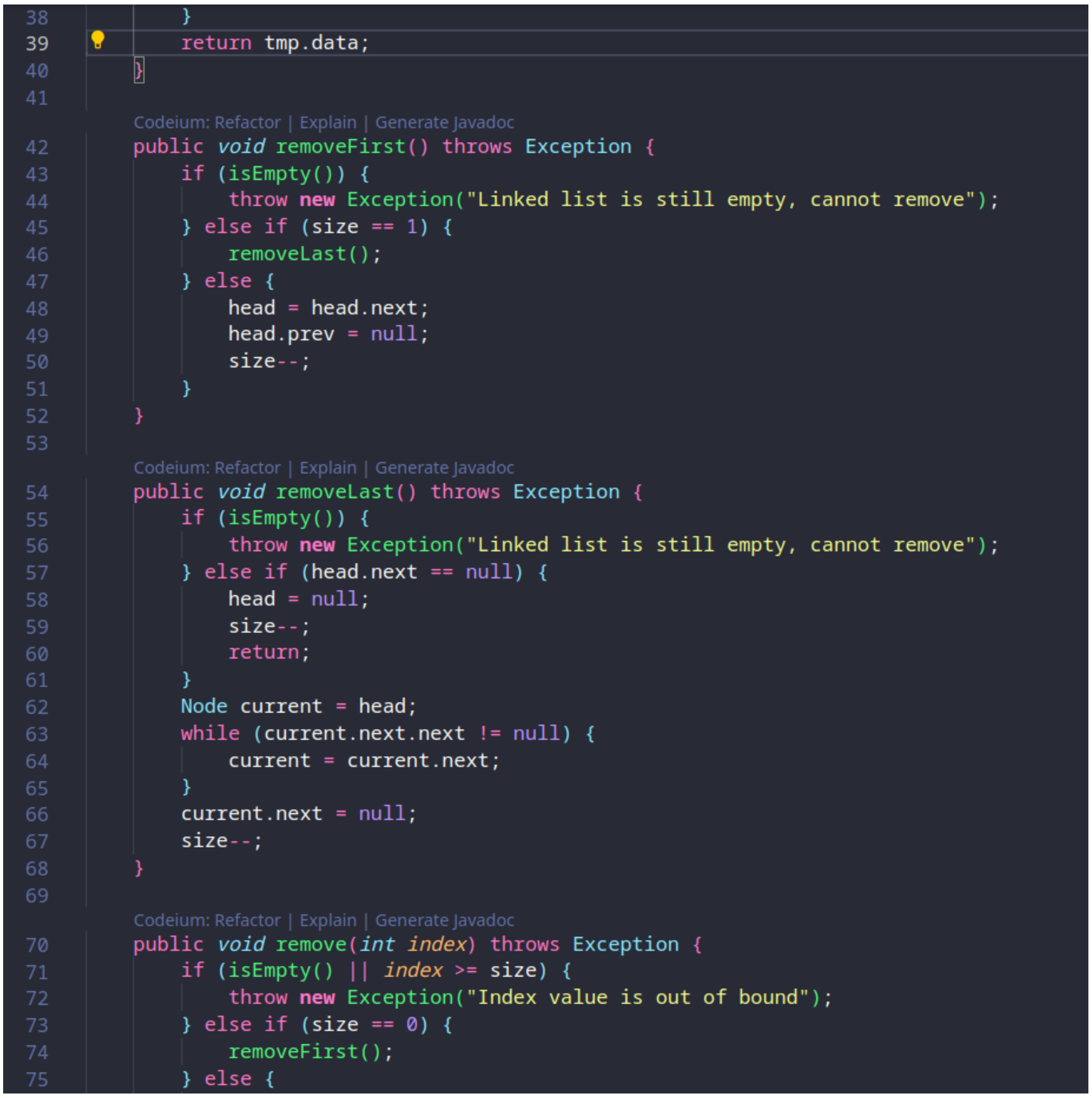


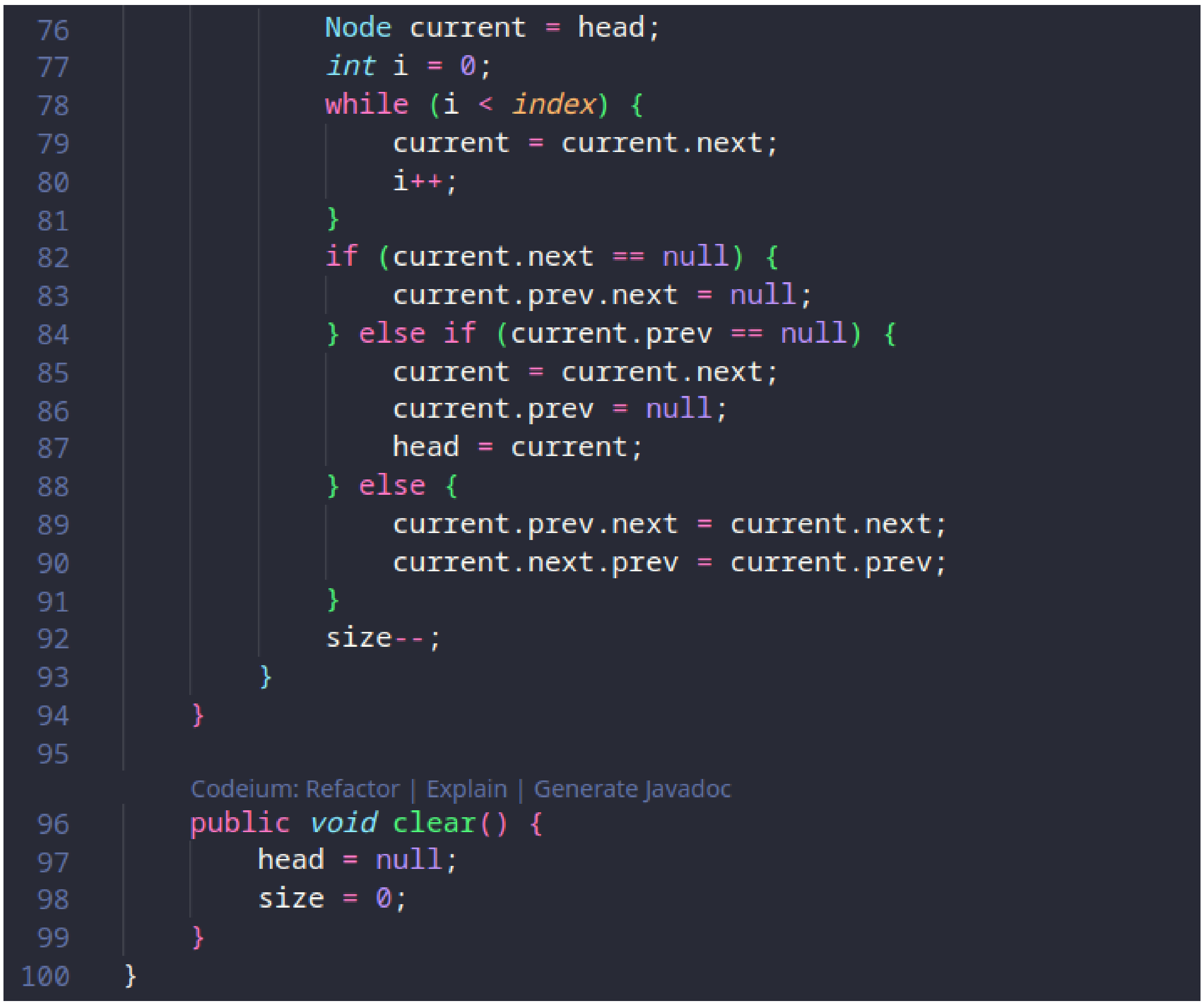
```java
package practicum1;

public class Node {
    public int data;
    Node prev, next;

    public Node(Node prev, int data, Node next) {
        this.prev = prev;
        this.next = next;
        this.data = data;
    }
}

```

- Linked List



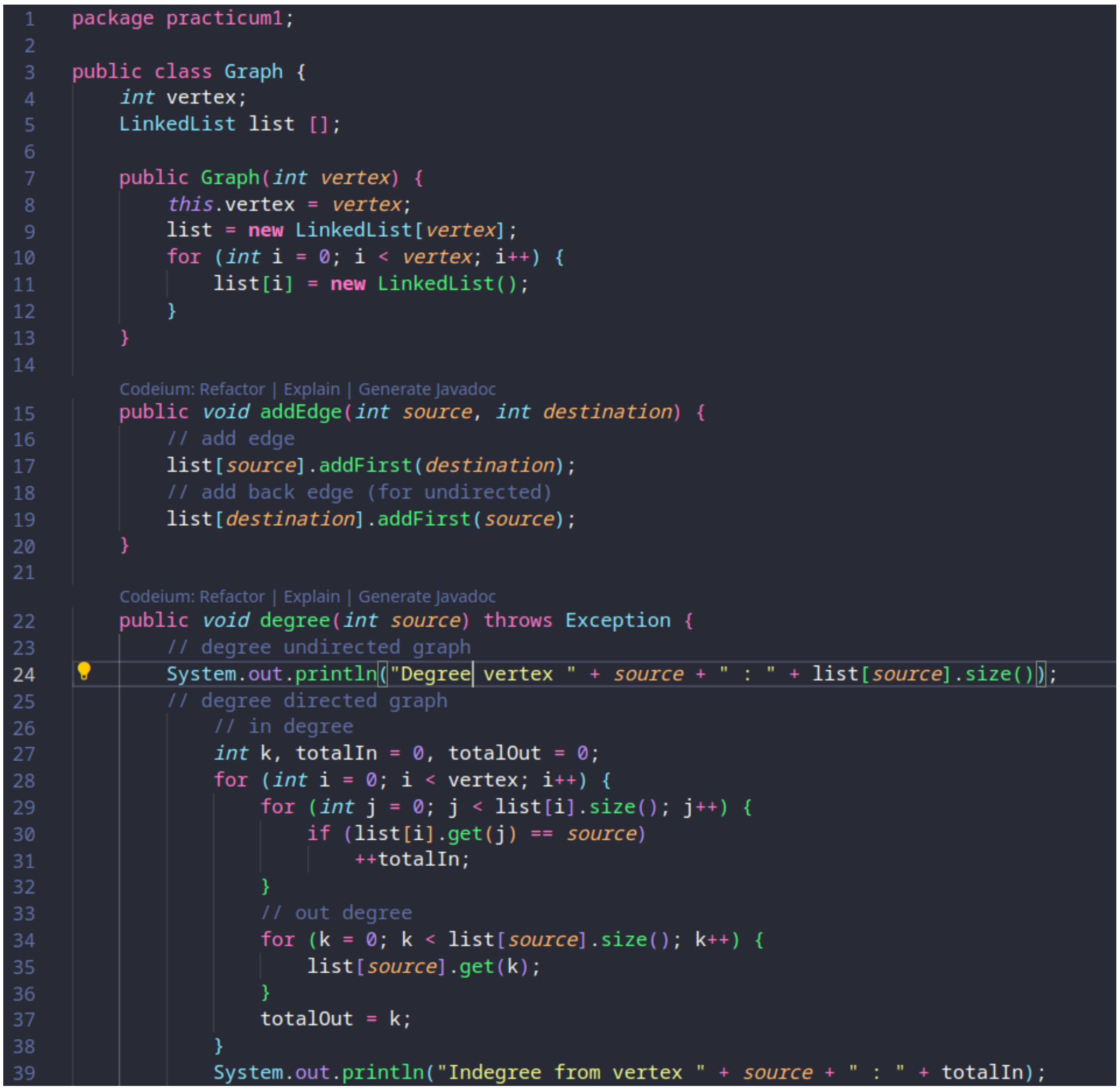
```java
package practicum1;

public class LinkedList {
    public Node head;
    public int size;

    public LinkedList() {
        head = null;
        size = 0;
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public boolean isEmpty() {
        return head == null;
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void addFirst(int item) {
        if (isEmpty()) {
            head = new Node(prev:null, item, head);
        } else {
            Node newNode = new Node(prev:null, item, head);
            head.prev = newNode;
            head = newNode;
        }
        size++;
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public int size() {
        return size;
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public int get(int index) throws Exception {
        if (isEmpty()) {
            throw new Exception("Linked list still empty");
        }
        Node tmp = head;
        for (int i = 0; i < index; i++) {
            tmp = tmp.next;
```


```
        }
        return tmp.data;
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void removeFirst() throws Exception {
        if (isEmpty()) {
            throw new Exception("Linked list is still empty, cannot remove");
        } else if (size == 1) {
            removeLast();
        } else {
            head = head.next;
            head.prev = null;
            size--;
        }
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void removeLast() throws Exception {
        if (isEmpty()) {
            throw new Exception("Linked list is still empty, cannot remove");
        } else if (head.next == null) {
            head = null;
            size--;
            return;
        }
        Node current = head;
        while (current.next.next != null) {
            current = current.next;
        }
        current.next = null;
        size--;
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void remove(int index) throws Exception {
        if (isEmpty() || index >= size) {
            throw new Exception("Index value is out of bound");
        } else if (size == 0) {
            removeFirst();
        } else {
```

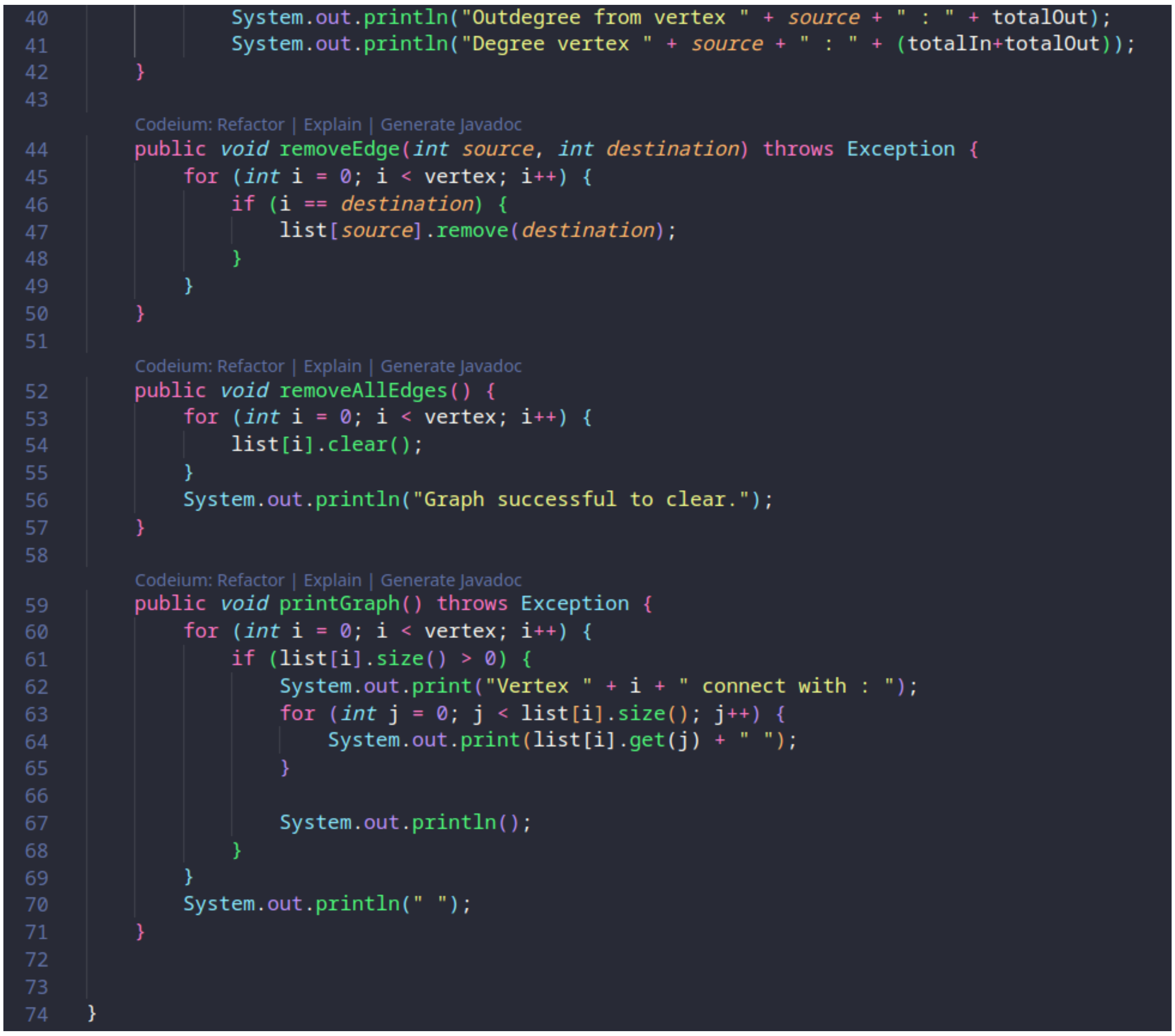


```
                Node current = head;
                int i = 0;
                while (i < index) {
                    current = current.next;
                    i++;
                }
                if (current.next == null) {
                    current.prev.next = null;
                } else if (current.prev == null) {
                    current = current.next;
                    current.prev = null;
                    head = current;
                } else {
                    current.prev.next = current.next;
                    current.next.prev = current.prev;
                }
                size--;
            }
        }

        Codeium: Refactor | Explain | Generate Javadoc
        public void clear() {
            head = null;
            size = 0;
        }
}
```

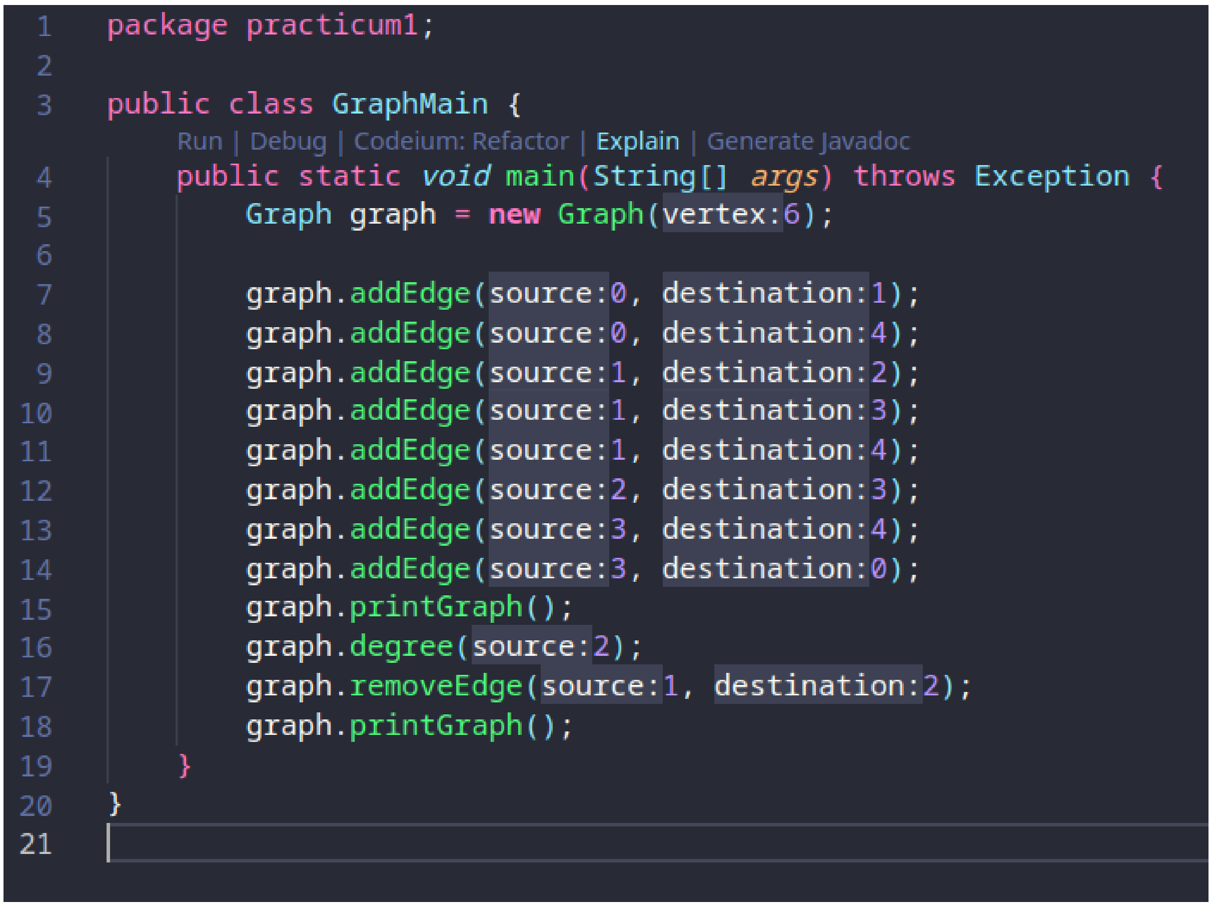
- Graph



```java
package practicum1;

public class Graph {
    int vertex;
    LinkedList list [];

    public Graph(int vertex) {
        this.vertex = vertex;
        list = new LinkedList[vertex];
        for (int i = 0; i < vertex; i++) {
            list[i] = new LinkedList();
        }
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void addEdge(int source, int destination) {
        // add edge
        list[source].addFirst(destination);
        // add back edge (for undirected)
        list[destination].addFirst(source);
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void degree(int source) throws Exception {
        // degree undirected graph
        System.out.println("Degree vertex " + source + " : " + list[source].size());
        // degree directed graph
            // in degree
            int k, totalIn = 0, totalOut = 0;
            for (int i = 0; i < vertex; i++) {
                for (int j = 0; j < list[i].size(); j++) {
                    if (list[i].get(j) == source)
                        ++totalIn;
                }
                // out degree
                for (k = 0; k < list[source].size(); k++) {
                    list[source].get(k);
                }
                totalOut = k;
            }
            System.out.println("Indegree from vertex " + source + " : " + totalIn);
```

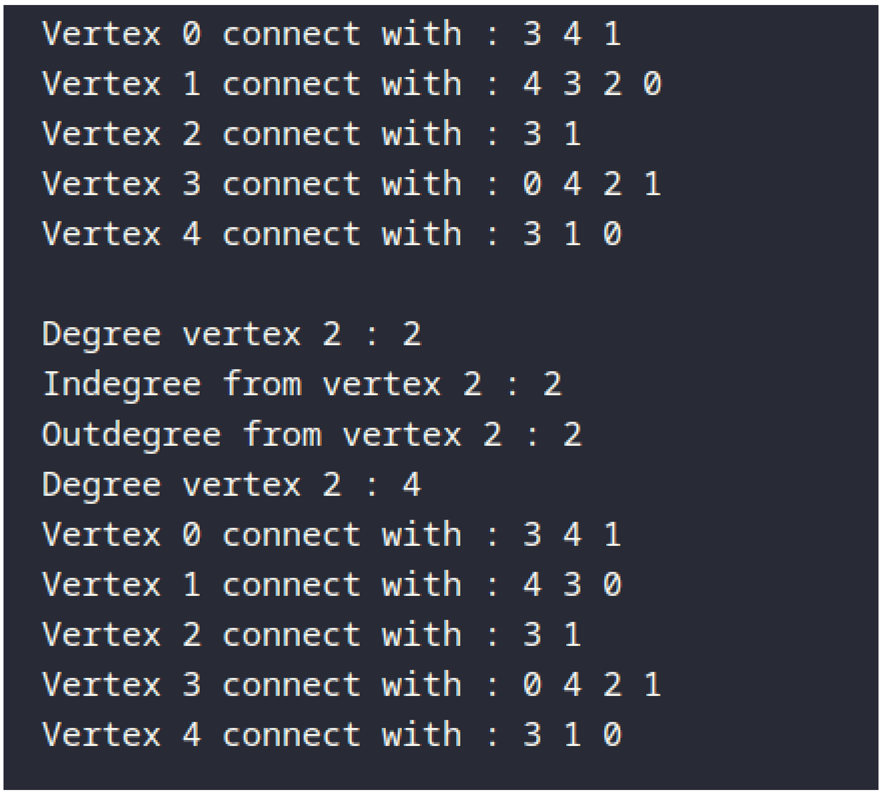

```
            System.out.println("Outdegree from vertex " + source + " : " + totalOut);
            System.out.println("Degree vertex " + source + " : " + (totalIn+totalOut));
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void removeEdge(int source, int destination) throws Exception {
        for (int i = 0; i < vertex; i++) {
            if (i == destination) {
                list[source].remove(destination);
            }
        }
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void removeAllEdges() {
        for (int i = 0; i < vertex; i++) {
            list[i].clear();
        }
        System.out.println("Graph successful to clear.");
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void printGraph() throws Exception {
        for (int i = 0; i < vertex; i++) {
            if (list[i].size() > 0) {
                System.out.print("Vertex " + i + " connect with : ");
                for (int j = 0; j < list[i].size(); j++) {
                    System.out.print(list[i].get(j) + " ");
                }

                System.out.println();
            }
        }
        System.out.println(" ");
    }


}
```

- Main



```java
package practicum1;

public class GraphMain {
    Run | Debug | Codeium: Refactor | Explain | Generate Javadoc
    public static void main(String[] args) throws Exception {
        Graph graph = new Graph(vertex:6);

        graph.addEdge(source:0, destination:1);
        graph.addEdge(source:0, destination:4);
        graph.addEdge(source:1, destination:2);
        graph.addEdge(source:1, destination:3);
        graph.addEdge(source:1, destination:4);
        graph.addEdge(source:2, destination:3);
        graph.addEdge(source:3, destination:4);
        graph.addEdge(source:3, destination:0);
        graph.printGraph();
        graph.degree(source:2);
        graph.removeEdge(source:1, destination:2);
        graph.printGraph();
    }
}
```

- Result



```
Vertex 0 connect with : 3 4 1
Vertex 1 connect with : 4 3 2 0
Vertex 2 connect with : 3 1
Vertex 3 connect with : 0 4 2 1
Vertex 4 connect with : 3 1 0

Degree vertex 2 : 2
Indegree from vertex 2 : 2
Outdegree from vertex 2 : 2
Degree vertex 2 : 4
Vertex 0 connect with : 3 4 1
Vertex 1 connect with : 4 3 0
Vertex 2 connect with : 3 1
Vertex 3 connect with : 0 4 2 1
Vertex 4 connect with : 3 1 0
```

Question :

1. Mention 3 kinds of algorithm that uses Graph fundamental,what's the use of those ?

- **Depth-First Search (DFS): DFS is a graph traversal algorithm that explores as far as possible along each branch before backtracking. It visits all vertices of a graph by going as deep as possible before backtracking. DFS is useful for solving problems such as finding connected components, detecting cycles, and exploring paths in a graph.**
- **Breadth-First Search (BFS): BFS is another graph traversal algorithm that visits all vertices of a graph by exploring all the neighbors of a vertex before moving to the next level of neighbors. BFS explores the vertices in a level-by-level manner, starting from the source vertex. It is commonly used to find the shortest path between two vertices, find the minimum spanning tree, or solve puzzles with multiple states.**
- **Dijkstra's Algorithm: Dijkstra's algorithm is used to find the shortest path between two vertices in a weighted graph with non-negative edge weights. It calculates the shortest path from a source vertex to all other vertices in the graph. Dijkstra's algorithm is widely applied in various applications such as network routing, GPS navigation, and resource allocation.**

2. In class Graph, there is an array with LinkedList data type, LinkedList list[]. What's the aim of this?

- **The aim of having an array of LinkedList data type, LinkedList list[], in the Graph class is to represent the adjacency list representation of the graph. Each element of the array represents a vertex, and the corresponding LinkedList stores the adjacent vertices connected to that vertex. This representation allows for efficient storage and manipulation of the connections between vertices in the graph.**

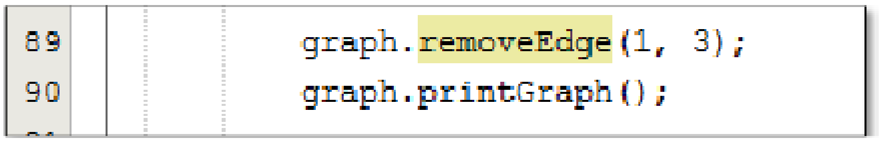
3. What is the reason of calling method addFirst() to add data, instead of calling other add methods in Linked list when using method addEdge in class Graph?

- **The reason for calling the addFirst() method instead of other add methods in the LinkedList class when using the addEdge() method in the Graph class is to implement an adjacency list representation for an undirected graph. Calling addFirst() adds the destination vertex to the adjacency list of the source vertex and vice versa, effectively creating an edge between the two vertices in an undirected manner.**

4. How do we detect prev pointer when we are about to remove an edge of a graph?

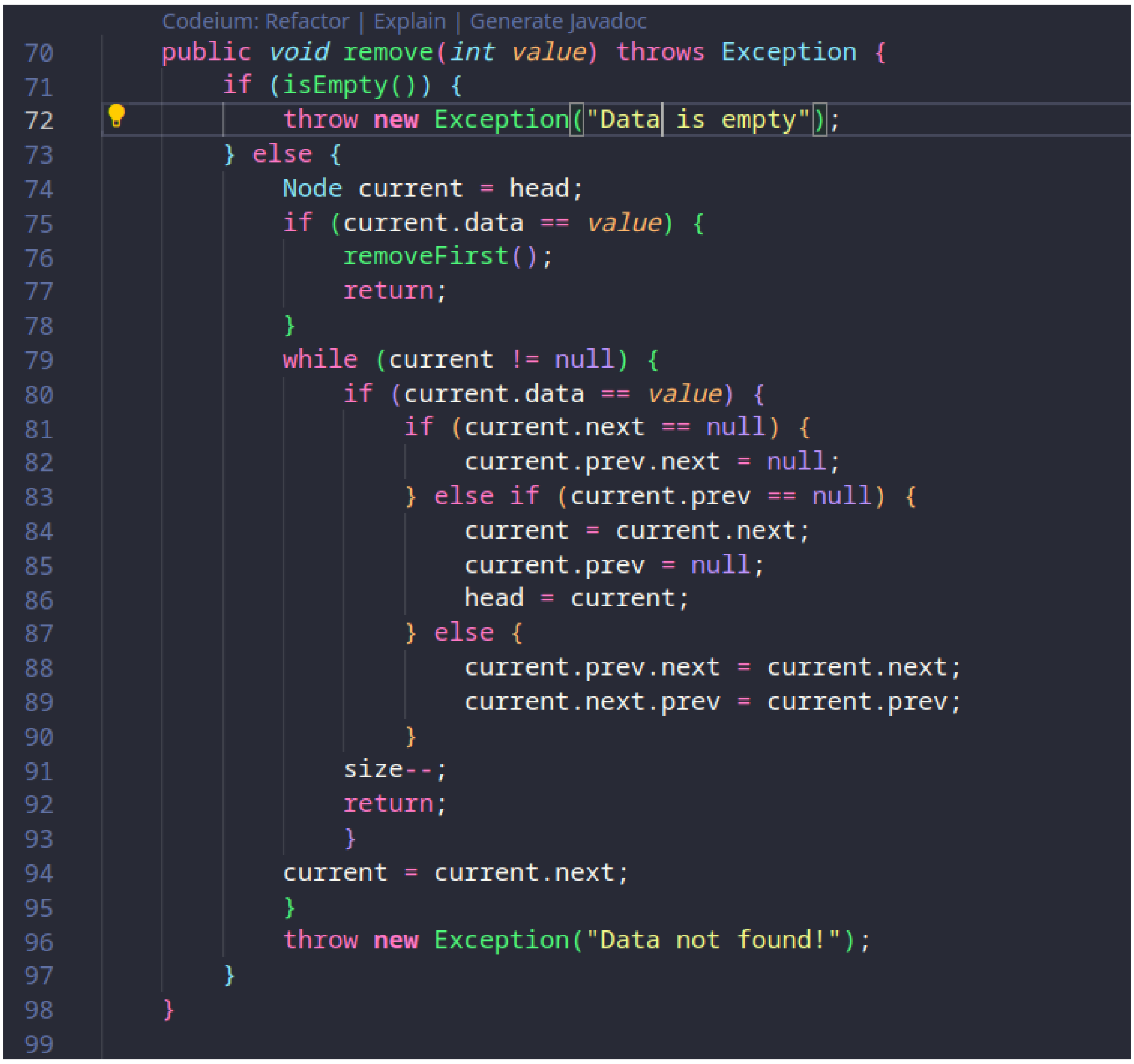
- **In the provided code, the prev pointer is used in the linked list structure to maintain the doubly linked list connection between nodes. However, when removing an edge in the Graph class, the code does not specifically use the prev pointer. The removal of an edge in this code is done by iterating over the linked list of the source vertex and removing the destination vertex from it. The prev pointer is not directly involved in this process.**

5. Why in practicum 1.2, the 12th step is to remove path that is not the first path to produce the wrong output? What's the solution?

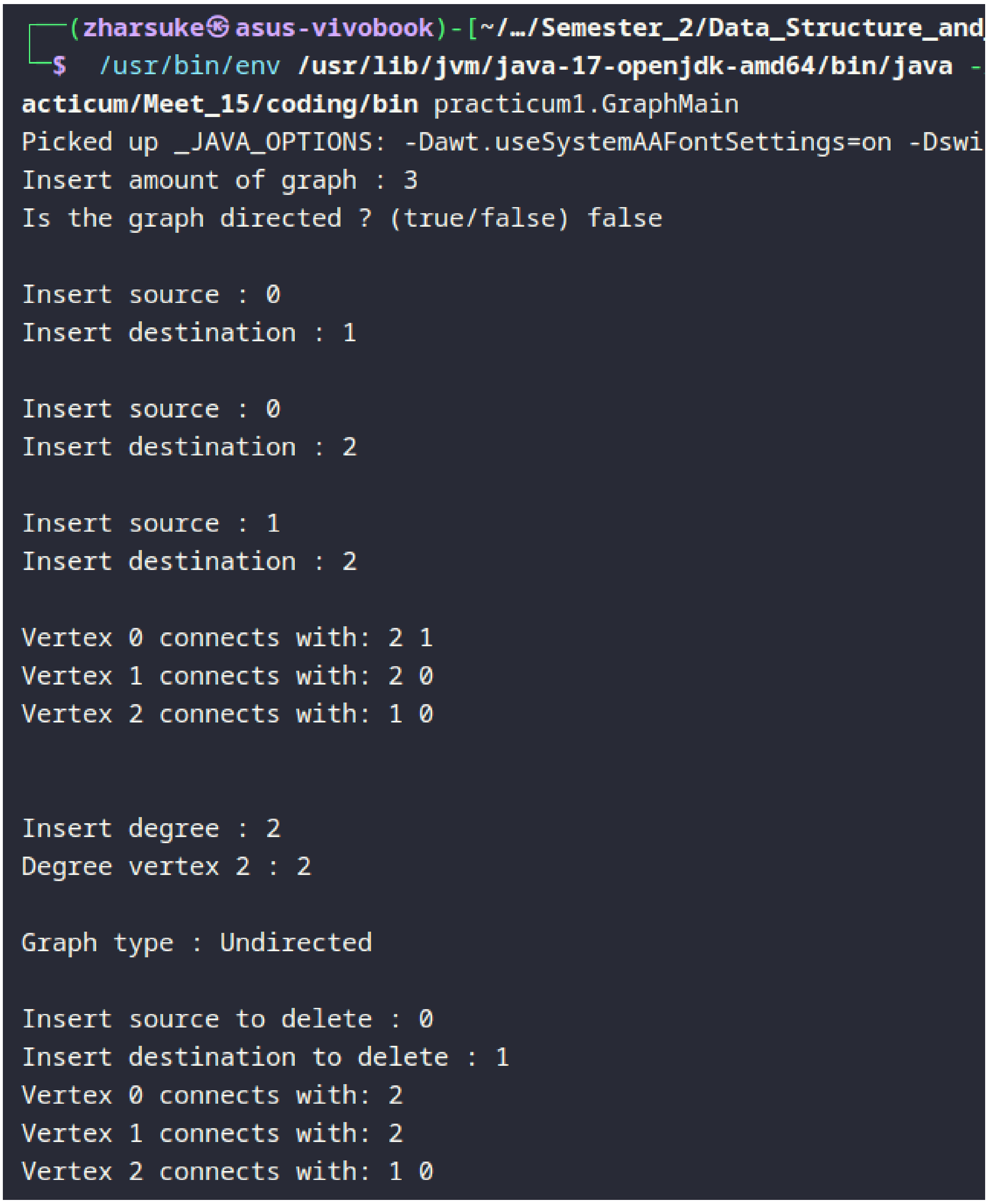


```java
graph.removeEdge(1, 3);
graph.printGraph();
```

- **actually the edge removal still produces the correct output, because in the removeEdge() method, the edge removal process is based on the index value, not the data value. The solution is to change the remove() method in LinkedList.java to remove based on value**



```java
    public void remove(int value) throws Exception {
        if (isEmpty()) {
            throw new Exception("Data is empty");
        } else {
            Node current = head;
            if (current.data == value) {
                removeFirst();
                return;
            }
            while (current != null) {
                if (current.data == value) {
                    if (current.next == null) {
                        current.prev.next = null;
                    } else if (current.prev == null) {
                        current = current.next;
                        current.prev = null;
                        head = current;
                    } else {
                        current.prev.next = current.next;
                        current.next.prev = current.prev;
                    }
                size--;
                return;
                }
            current = current.next;
            }
            throw new Exception("Data not found!");
        }
    }
```

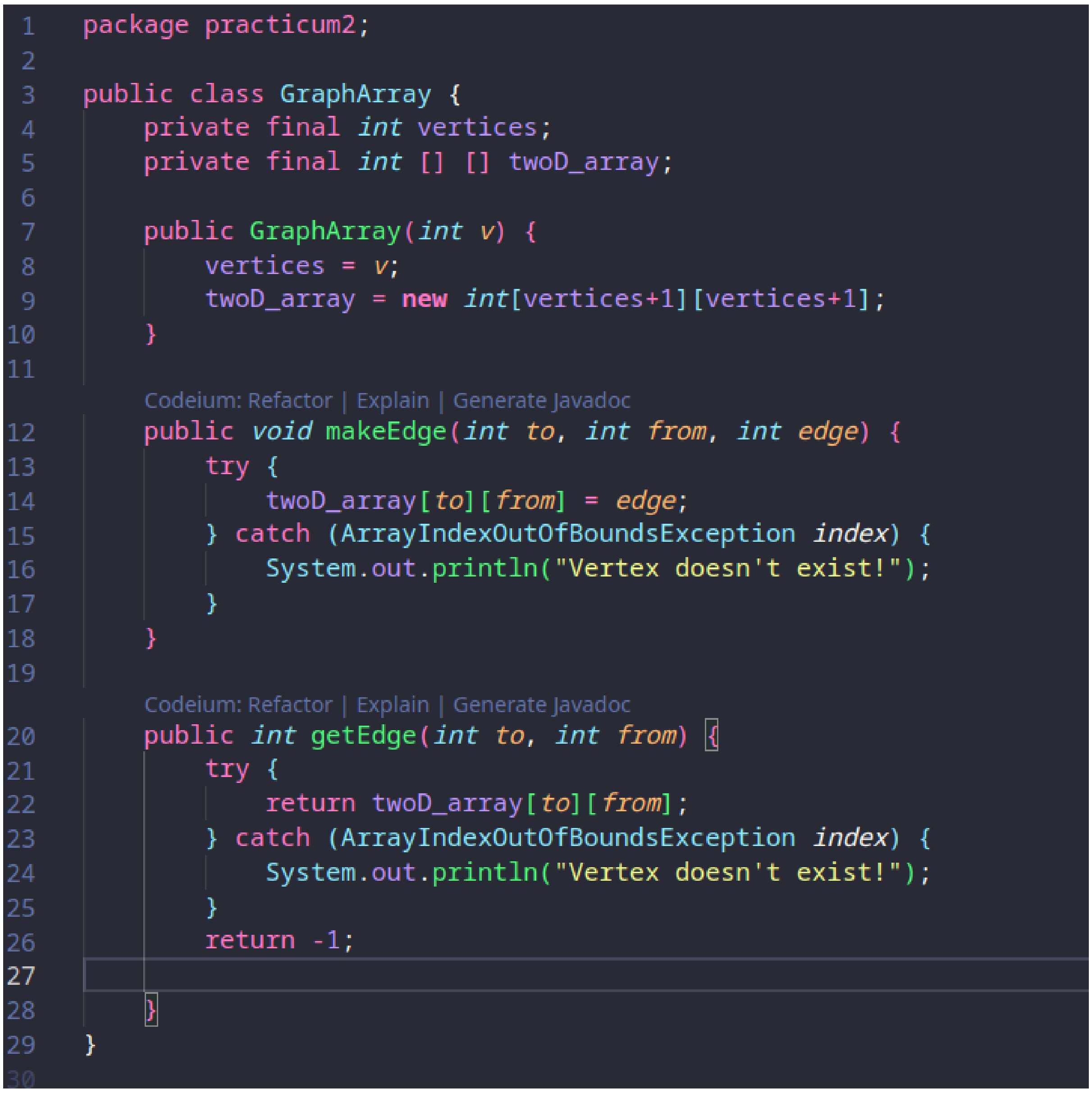


```
┌──(zharsuke㉿asus-vivobook)-[~/…/Semester_2/Data_Structure_and
└─$  /usr/bin/env /usr/lib/jvm/java-17-openjdk-amd64/bin/java -
acticum/Meet_15/coding/bin practicum1.GraphMain
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswi
Insert amount of graph : 3
Is the graph directed ? (true/false) false

Insert source : 0
Insert destination : 1

Insert source : 0
Insert destination : 2

Insert source : 1
Insert destination : 2

Vertex 0 connects with: 2 1
Vertex 1 connects with: 2 0
Vertex 2 connects with: 1 0


Insert degree : 2
Degree vertex 2 : 2

Graph type : Undirected

Insert source to delete : 0
Insert destination to delete : 1
Vertex 0 connects with: 2
Vertex 1 connects with: 2
Vertex 2 connects with: 1 0
```

Practicum 2 :

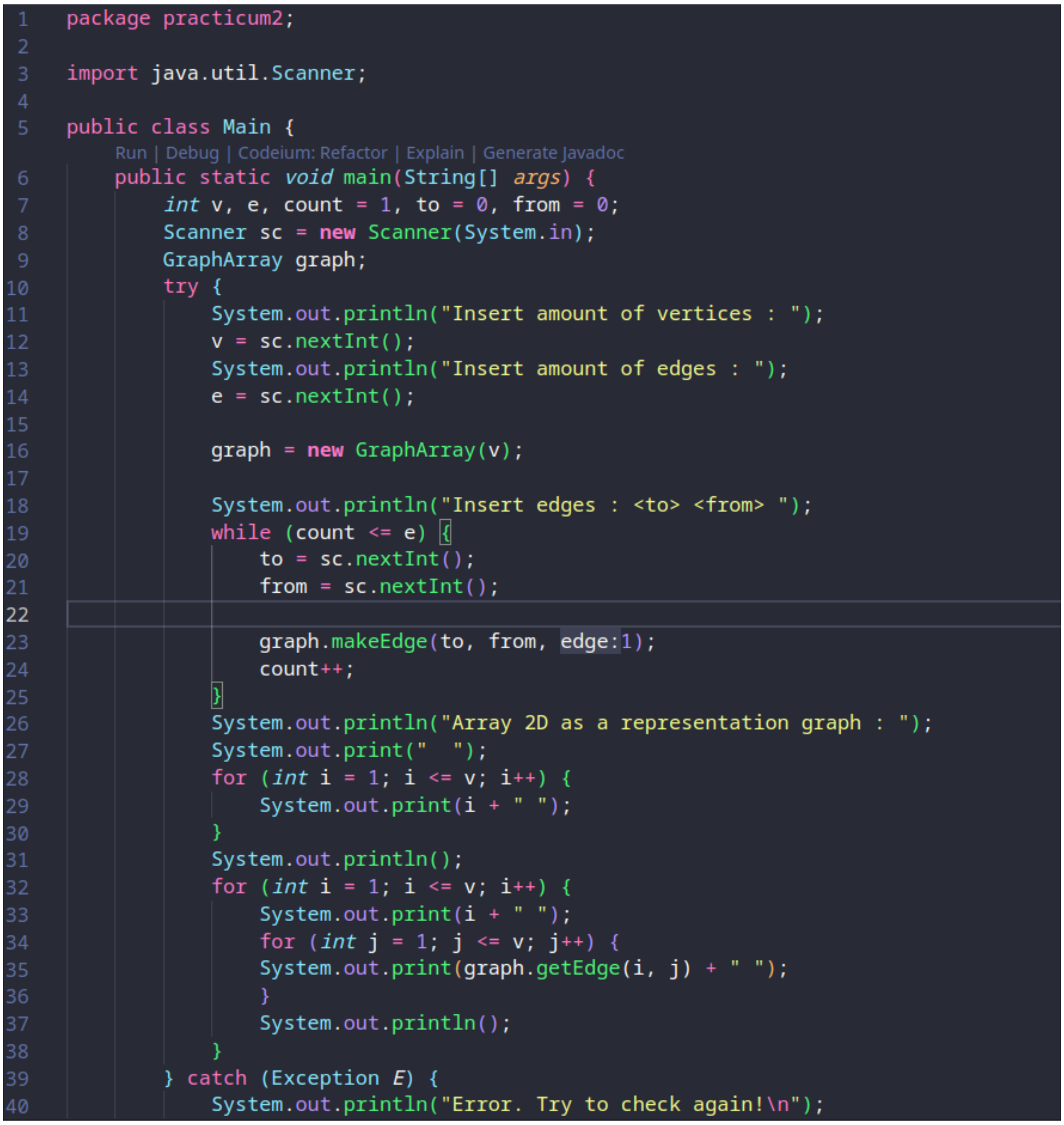
Code :

- Graph Array

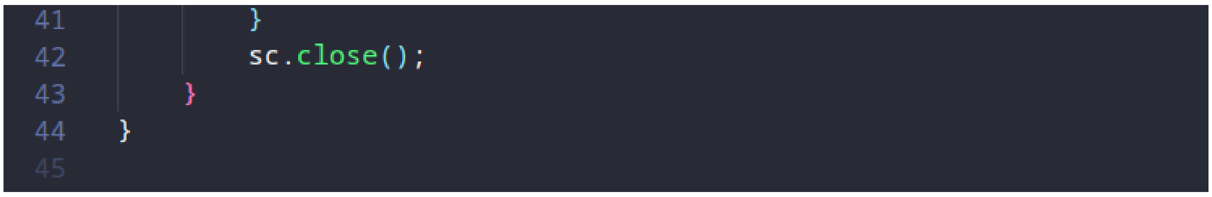


```java
package practicum2;

public class GraphArray {
    private final int vertices;
    private final int [] [] twoD_array;

    public GraphArray(int v) {
        vertices = v;
        twoD_array = new int[vertices+1][vertices+1];
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void makeEdge(int to, int from, int edge) {
        try {
            twoD_array[to][from] = edge;
        } catch (ArrayIndexOutOfBoundsException index) {
            System.out.println("Vertex doesn't exist!");
        }
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public int getEdge(int to, int from) {
        try {
            return twoD_array[to][from];
        } catch (ArrayIndexOutOfBoundsException index) {
            System.out.println("Vertex doesn't exist!");
        }
        return -1;

    }
}
```

- Main

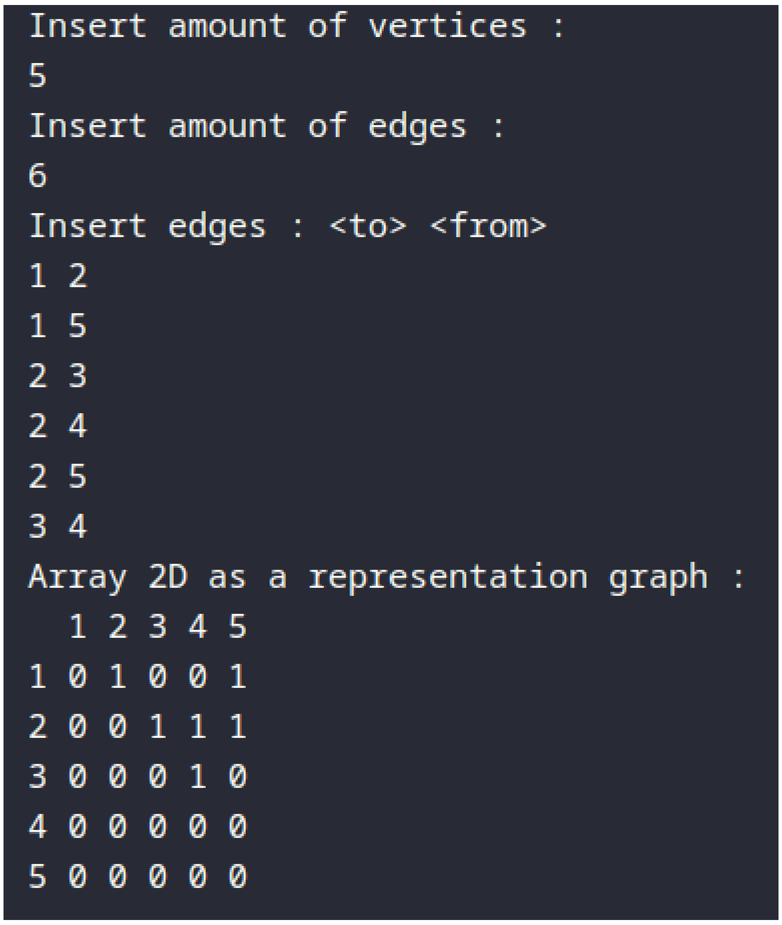

Run | Debug | Codeium: Refactor | Explain | Generate Javadoc

```java
package practicum2;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        int v, e, count = 1, to = 0, from = 0;
        Scanner sc = new Scanner(System.in);
        GraphArray graph;
        try {
            System.out.println("Insert amount of vertices : ");
            v = sc.nextInt();
            System.out.println("Insert amount of edges : ");
            e = sc.nextInt();

            graph = new GraphArray(v);

            System.out.println("Insert edges : <to> <from> ");
            while (count <= e) {
                to = sc.nextInt();
                from = sc.nextInt();

                graph.makeEdge(to, from, edge:1);
                count++;
            }
            System.out.println("Array 2D as a representation graph : ");
            System.out.print("  ");
            for (int i = 1; i <= v; i++) {
                System.out.print(i + " ");
            }
            System.out.println();
            for (int i = 1; i <= v; i++) {
                System.out.print(i + " ");
                for (int j = 1; j <= v; j++) {
                System.out.print(graph.getEdge(i, j) + " ");
                }
                System.out.println();
            }
        } catch (Exception E) {
            System.out.println("Error. Try to check again!\n");
```



```java
            }
            sc.close();
        }
    }
```

Result :



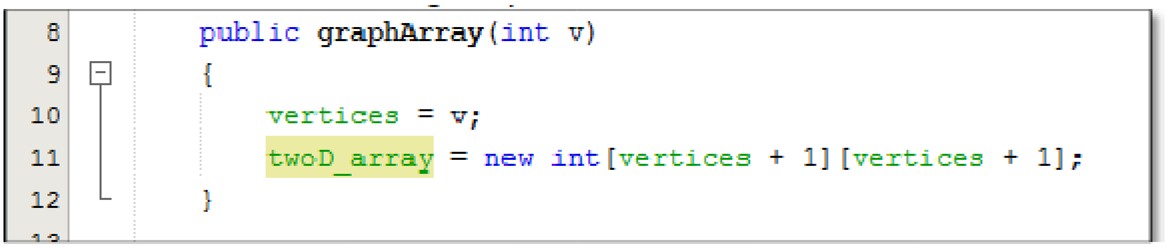
```
Insert amount of vertices :
5
Insert amount of edges :
6
Insert edges : <to> <from>
1 2
1 5
2 3
2 4
2 5
3 4
Array 2D as a representation graph :
  1 2 3 4 5
1 0 1 0 0 1
2 0 0 1 1 1
3 0 0 0 1 0
4 0 0 0 0 0
5 0 0 0 0 0
```

Question :

1. What is the degree difference between directed and undirected graphs?

- **The degree difference between directed and undirected graphs is that in an undirected graph, the degree of a vertex represents the total number of edges connected to that vertex, while in a directed graph, the degree is split into the in-degree (edges entering the vertex) and out-degree (edges leaving the vertex).**

2. In the graph implementation using adjacency matrix. Why does the number of vertices have to be added to 1 in the following array index?



```
    public graphArray(int v)
    {
        vertices = v;
        twoD_array = new int[vertices + 1][vertices + 1];
    }
```

- **In the graph implementation using an adjacency matrix, the number of vertices is added by 1 in the array index because the vertices in the graph are usually**

labeled starting from 1, not 0. By adding 1 to the number of vertices, it allows for the array indices to directly correspond to the vertex labels, making the implementation more intuitive and easier to understand.

3. What is the use of the getEdge() method?

   - **The getEdge() method in the code above is used to retrieve the value of the edge between two vertices in a graph represented by an adjacency matrix. It takes two parameters, to and from, which specify the vertices of interest, and returns the value (weight or presence) of the edge between those vertices.**

4. What kind of graph were implemented on practicum 1.3 ?

   - **The code above implements a graph using an adjacency matrix, which can be used to represent both directed and undirected graphs.**

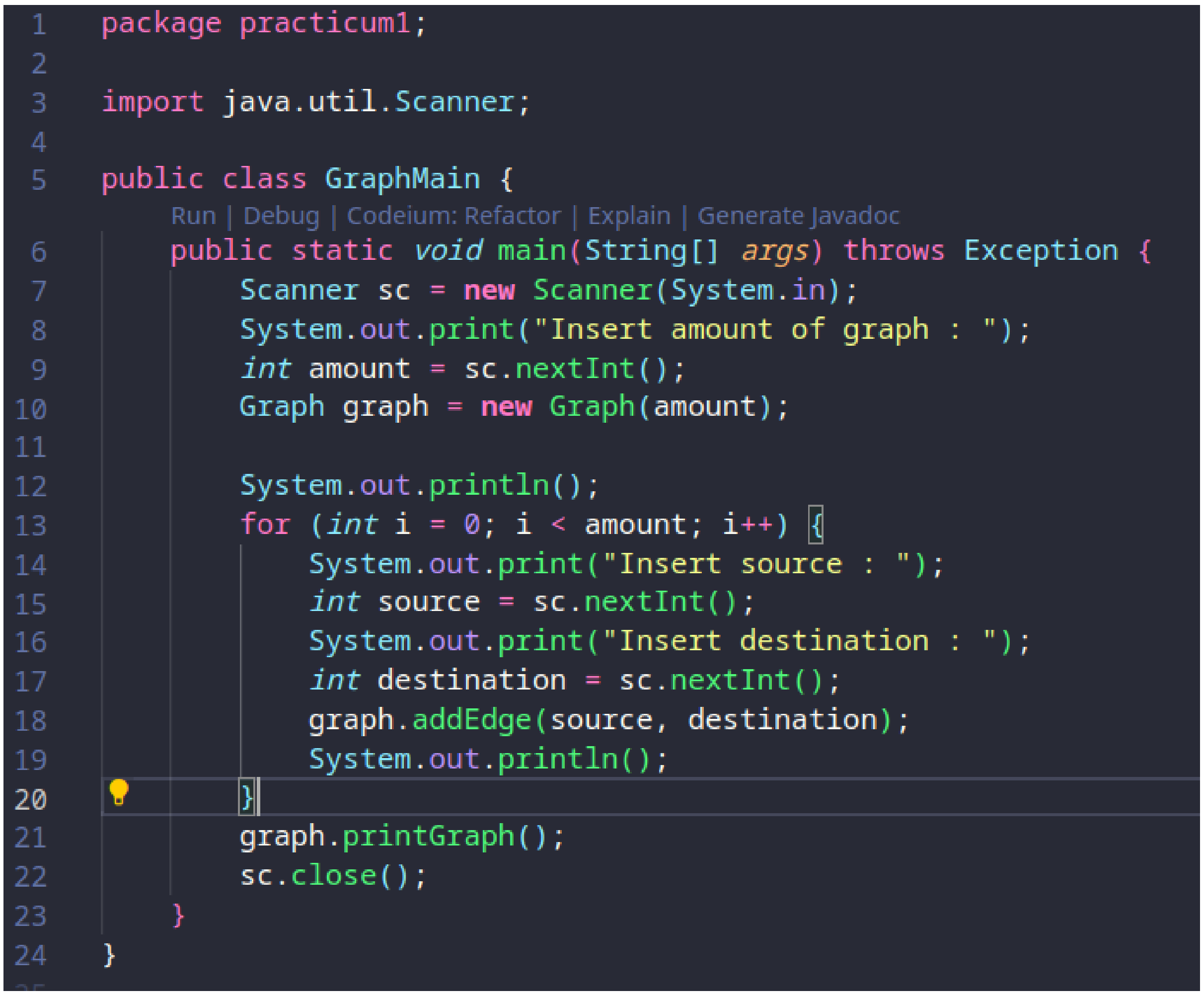
5. Why does the main method use try-catch Exception?

   - **The try-catch block in the main method is used to catch any exceptions that might occur during the execution of the code. By using the try-catch construct, the program can handle any potential errors or exceptions gracefully, preventing the program from crashing and providing an error message to the user.**

Assignment :

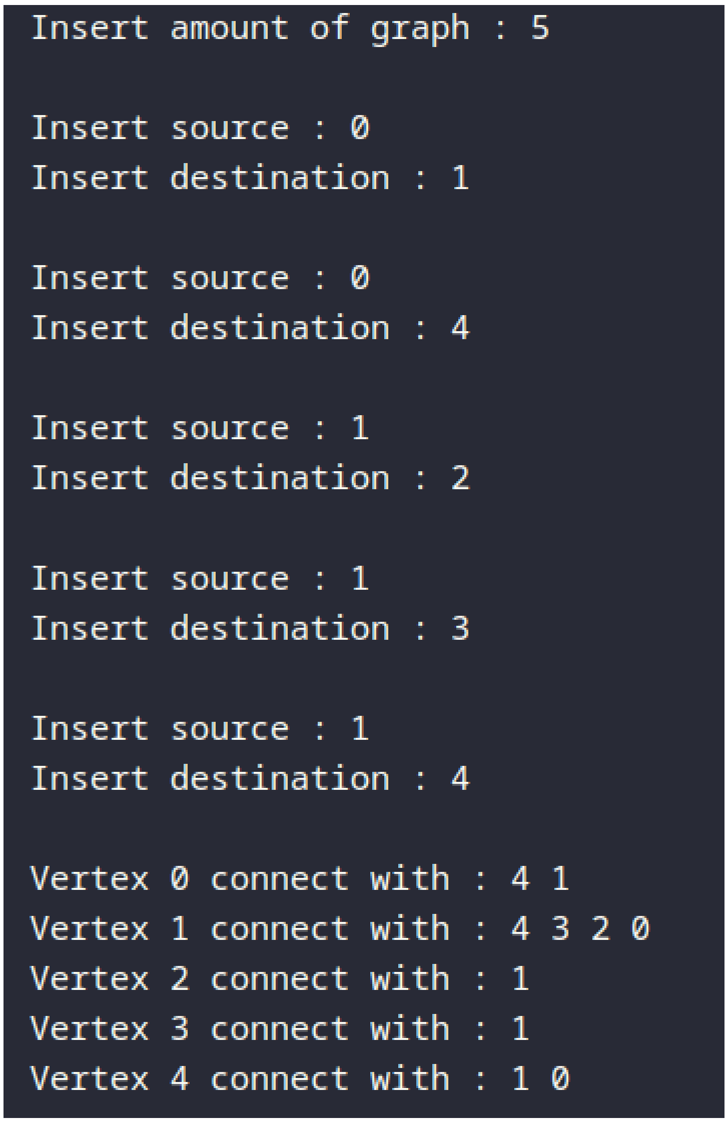
1. Convert the path in 1.2 as an input !

Code :


Run | Debug | Codeium: Refactor | Explain | Generate Javadoc

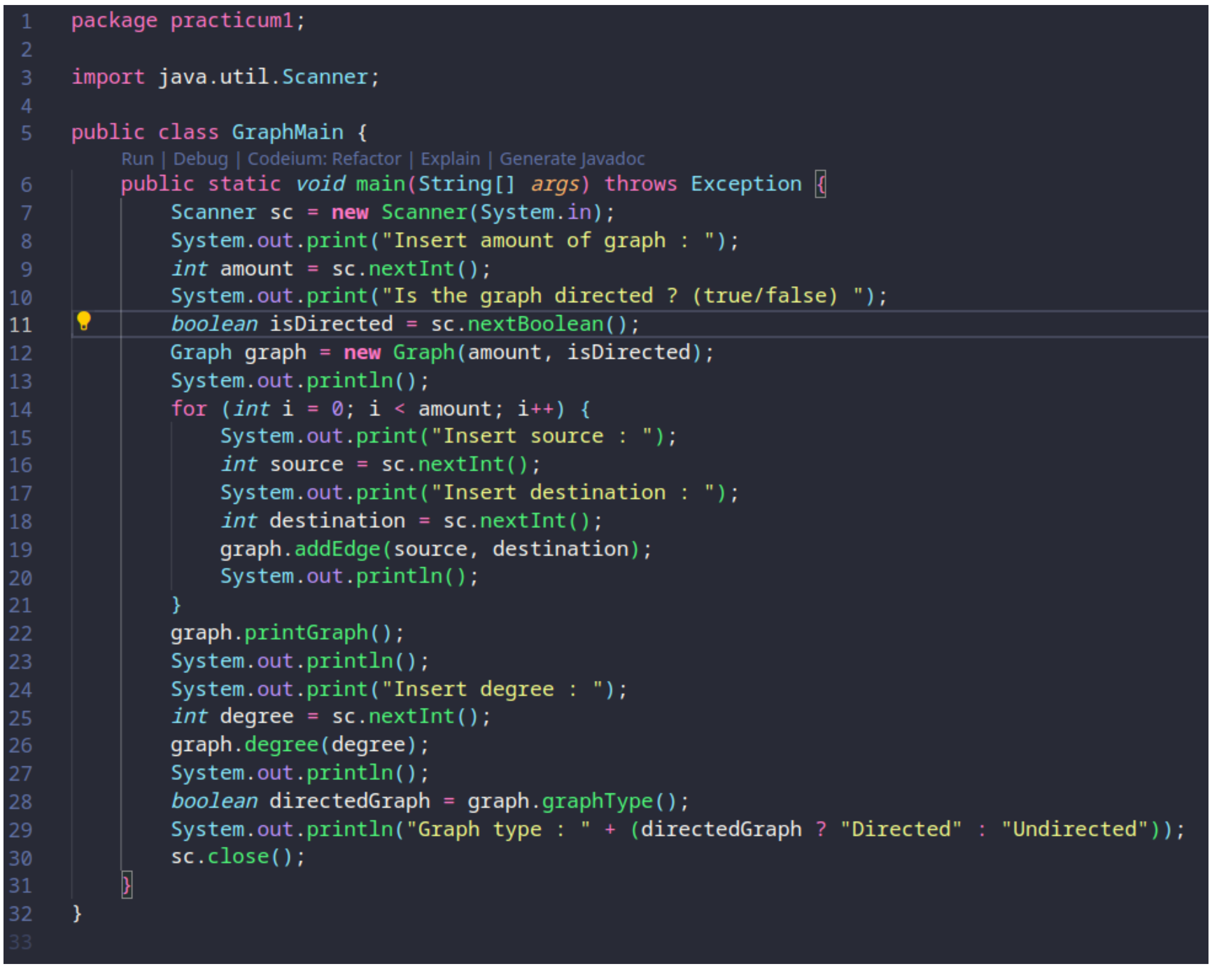
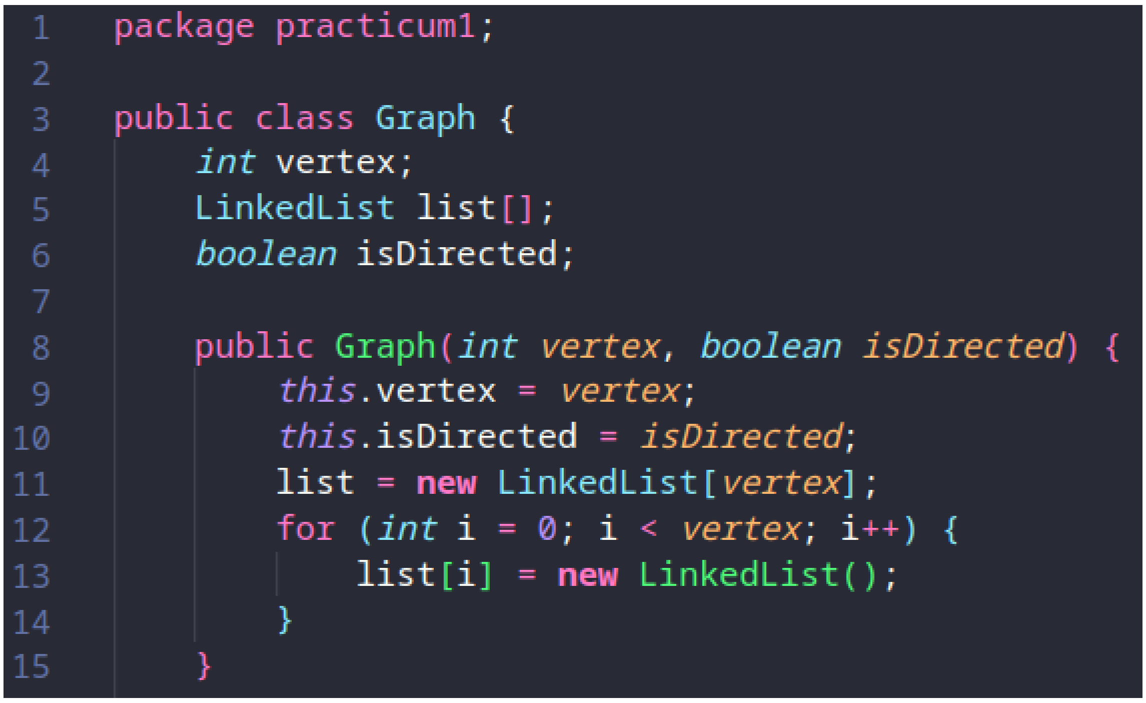
```java
package practicum1;

import java.util.Scanner;

public class GraphMain {
    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        System.out.print("Insert amount of graph : ");
        int amount = sc.nextInt();
        Graph graph = new Graph(amount);

        System.out.println();
        for (int i = 0; i < amount; i++) {
            System.out.print("Insert source : ");
            int source = sc.nextInt();
            System.out.print("Insert destination : ");
            int destination = sc.nextInt();
            graph.addEdge(source, destination);
            System.out.println();
        }
        graph.printGraph();
        sc.close();
    }
}
```

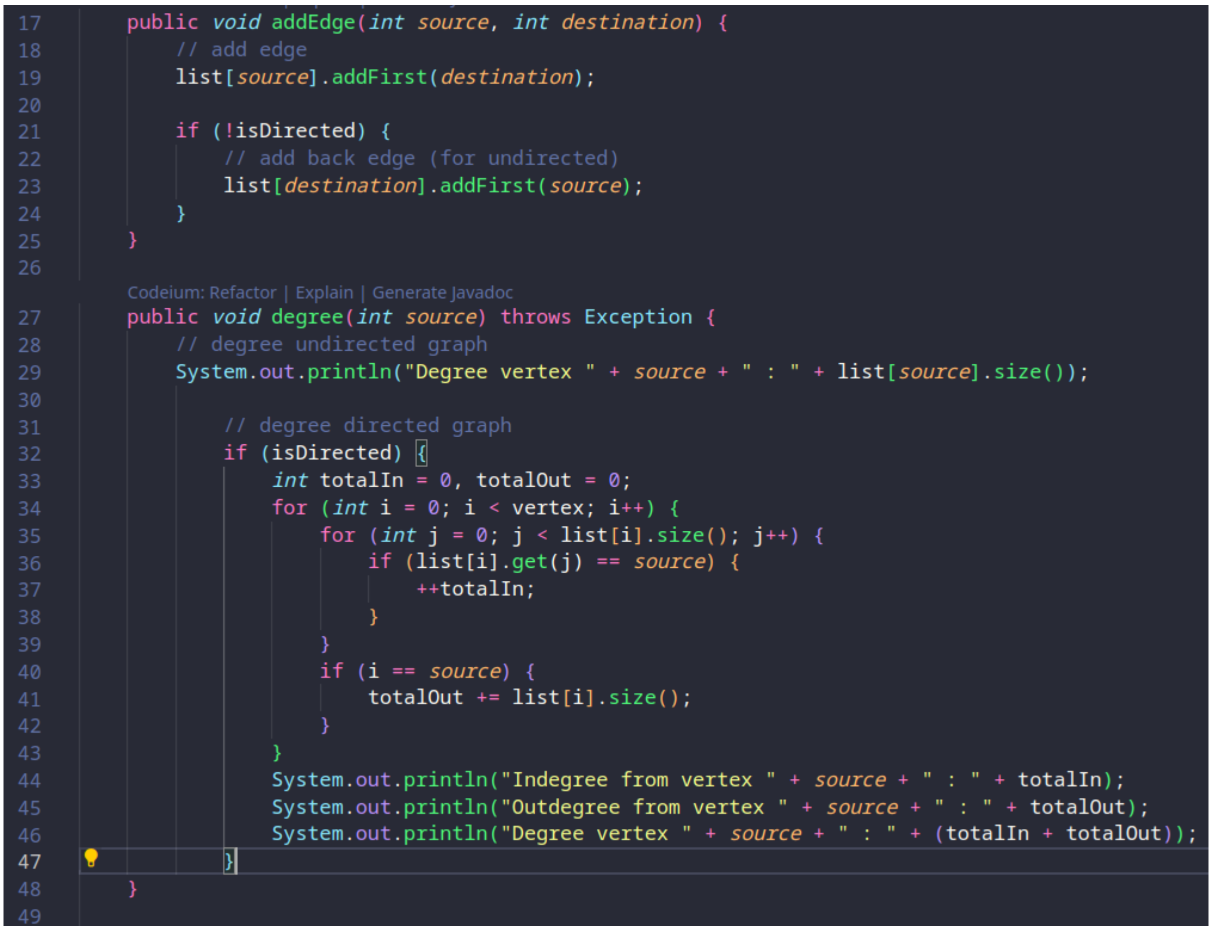
Result :


```
Insert amount of graph : 5

Insert source : 0
Insert destination : 1

Insert source : 0
Insert destination : 4

Insert source : 1
Insert destination : 2

Insert source : 1
Insert destination : 3

Insert source : 1
Insert destination : 4

Vertex 0 connect with : 4 1
Vertex 1 connect with : 4 3 2 0
Vertex 2 connect with : 1
Vertex 3 connect with : 1
Vertex 4 connect with : 1 0
```

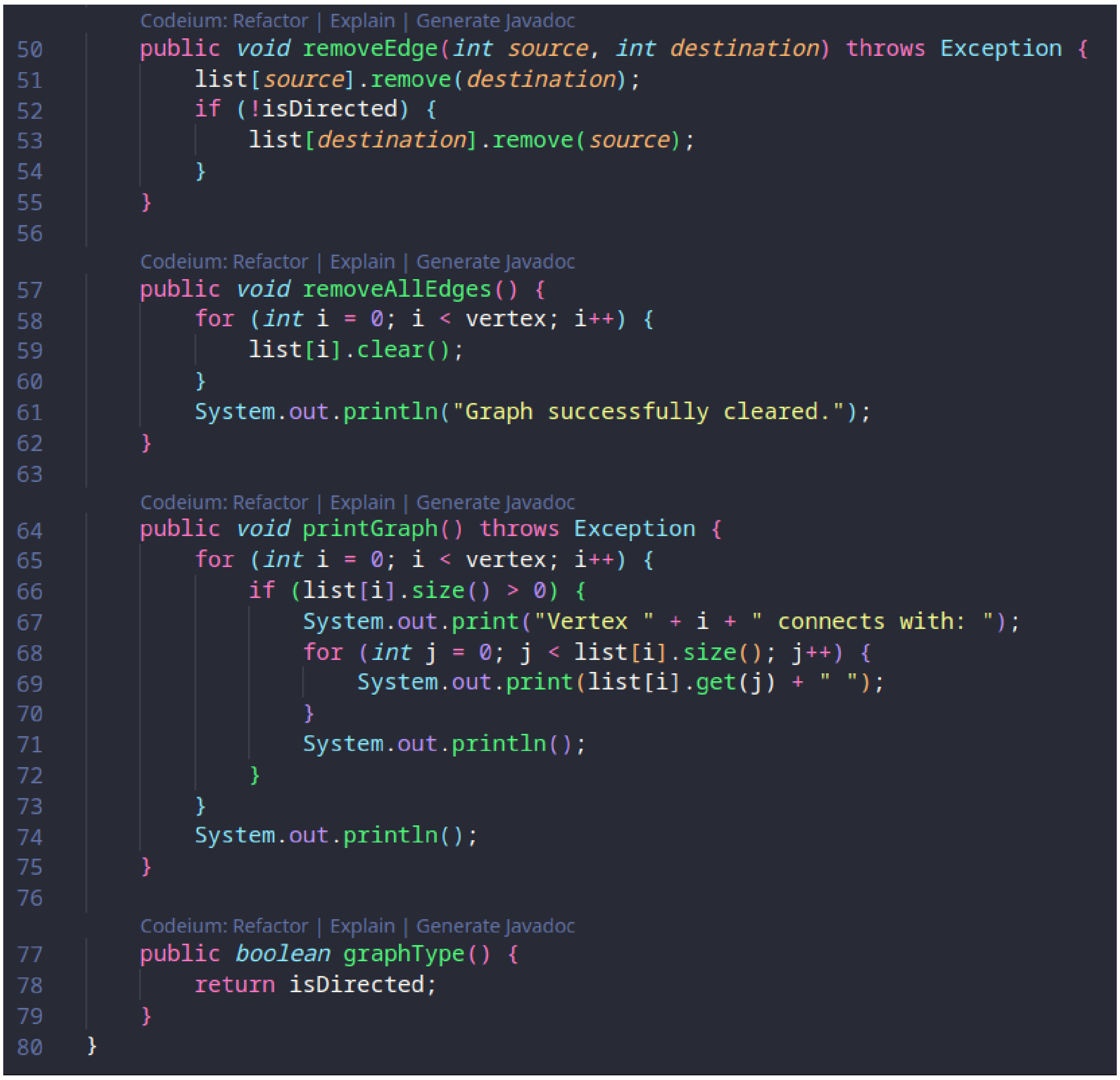
2. Add method graphType with boolean as its return type to differentiate which graph is directed or undirected graph. Then update all the method that relates to graphType() (only runs the statement based on the graph type) in practicum 1.2

- Code :


```java
package practicum1;

import java.util.Scanner;

public class GraphMain {
    Run | Debug | Codeium: Refactor | Explain | Generate Javadoc
    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        System.out.print("Insert amount of graph : ");
        int amount = sc.nextInt();
        System.out.print("Is the graph directed ? (true/false) ");
        boolean isDirected = sc.nextBoolean();
        Graph graph = new Graph(amount, isDirected);
        System.out.println();
        for (int i = 0; i < amount; i++) {
            System.out.print("Insert source : ");
            int source = sc.nextInt();
            System.out.print("Insert destination : ");
            int destination = sc.nextInt();
            graph.addEdge(source, destination);
            System.out.println();
        }
        graph.printGraph();
        System.out.println();
        System.out.print("Insert degree : ");
        int degree = sc.nextInt();
        graph.degree(degree);
        System.out.println();
        boolean directedGraph = graph.graphType();
        System.out.println("Graph type : " + (directedGraph ? "Directed" : "Undirected"));
        sc.close();
    }
}
```

```java
package practicum1;

public class Graph {
    int vertex;
    LinkedList list[];
    boolean isDirected;

    public Graph(int vertex, boolean isDirected) {
        this.vertex = vertex;
        this.isDirected = isDirected;
        list = new LinkedList[vertex];
        for (int i = 0; i < vertex; i++) {
            list[i] = new LinkedList();
        }
    }
```

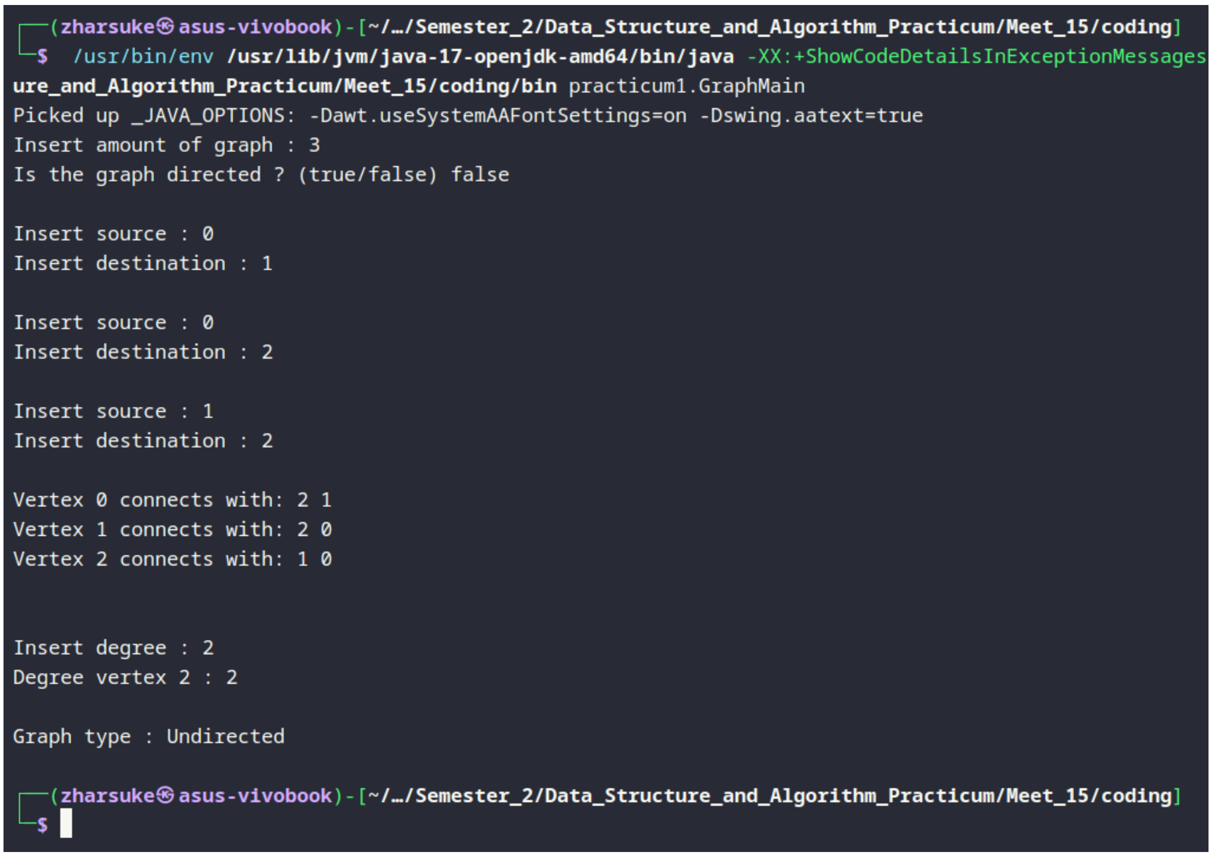


```
    public void addEdge(int source, int destination) {
        // add edge
        list[source].addFirst(destination);

        if (!isDirected) {
            // add back edge (for undirected)
            list[destination].addFirst(source);
        }
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void degree(int source) throws Exception {
        // degree undirected graph
        System.out.println("Degree vertex " + source + " : " + list[source].size());

            // degree directed graph
            if (isDirected) {
                int totalIn = 0, totalOut = 0;
                for (int i = 0; i < vertex; i++) {
                    for (int j = 0; j < list[i].size(); j++) {
                        if (list[i].get(j) == source) {
                            ++totalIn;
                        }
                    }
                    if (i == source) {
                        totalOut += list[i].size();
                    }
                }
                System.out.println("Indegree from vertex " + source + " : " + totalIn);
                System.out.println("Outdegree from vertex " + source + " : " + totalOut);
                System.out.println("Degree vertex " + source + " : " + (totalIn + totalOut));
            }
    }

```



```java
    Codeium: Refactor | Explain | Generate Javadoc
    public void removeEdge(int source, int destination) throws Exception {
        list[source].remove(destination);
        if (!isDirected) {
            list[destination].remove(source);
        }
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void removeAllEdges() {
        for (int i = 0; i < vertex; i++) {
            list[i].clear();
        }
        System.out.println("Graph successfully cleared.");
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void printGraph() throws Exception {
        for (int i = 0; i < vertex; i++) {
            if (list[i].size() > 0) {
                System.out.print("Vertex " + i + " connects with: ");
                for (int j = 0; j < list[i].size(); j++) {
                    System.out.print(list[i].get(j) + " ");
                }
                System.out.println();
            }
        }
        System.out.println();
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public boolean graphType() {
        return isDirected;
    }
}
```

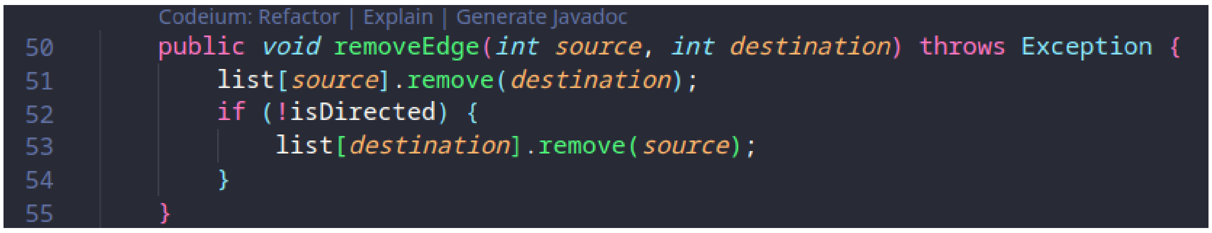
- Result :



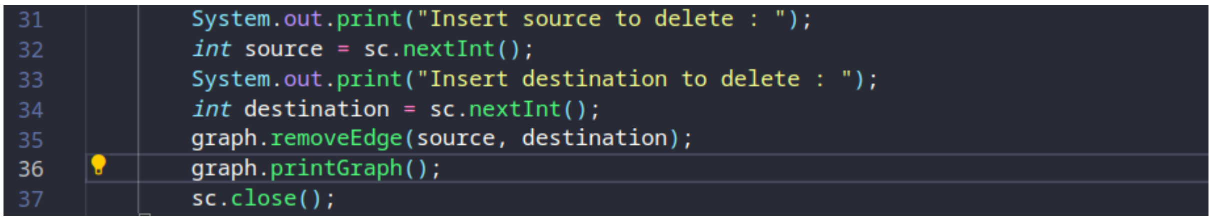
```
┌──(zharsuke㉿asus-vivobook)-[~/…/Semester_2/Data_Structure_and_Algorithm_Practicum/Meet_15/coding]
└─$  /usr/bin/env /usr/lib/jvm/java-17-openjdk-amd64/bin/java -XX:+ShowCodeDetailsInExceptionMessages
ure_and_Algorithm_Practicum/Meet_15/coding/bin practicum1.GraphMain
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Insert amount of graph : 3
Is the graph directed ? (true/false) false

Insert source : 0
Insert destination : 1

Insert source : 0
Insert destination : 2

Insert source : 1
Insert destination : 2

Vertex 0 connects with: 2 1
Vertex 1 connects with: 2 0
Vertex 2 connects with: 1 0


Insert degree : 2
Degree vertex 2 : 2

Graph type : Undirected

┌──(zharsuke㉿asus-vivobook)-[~/…/Semester_2/Data_Structure_and_Algorithm_Practicum/Meet_15/coding]
└─$ 
```

3. Modify method removeEdge() in practicum 1.2 so that it won't give the wrong path other than the initial path as an output !

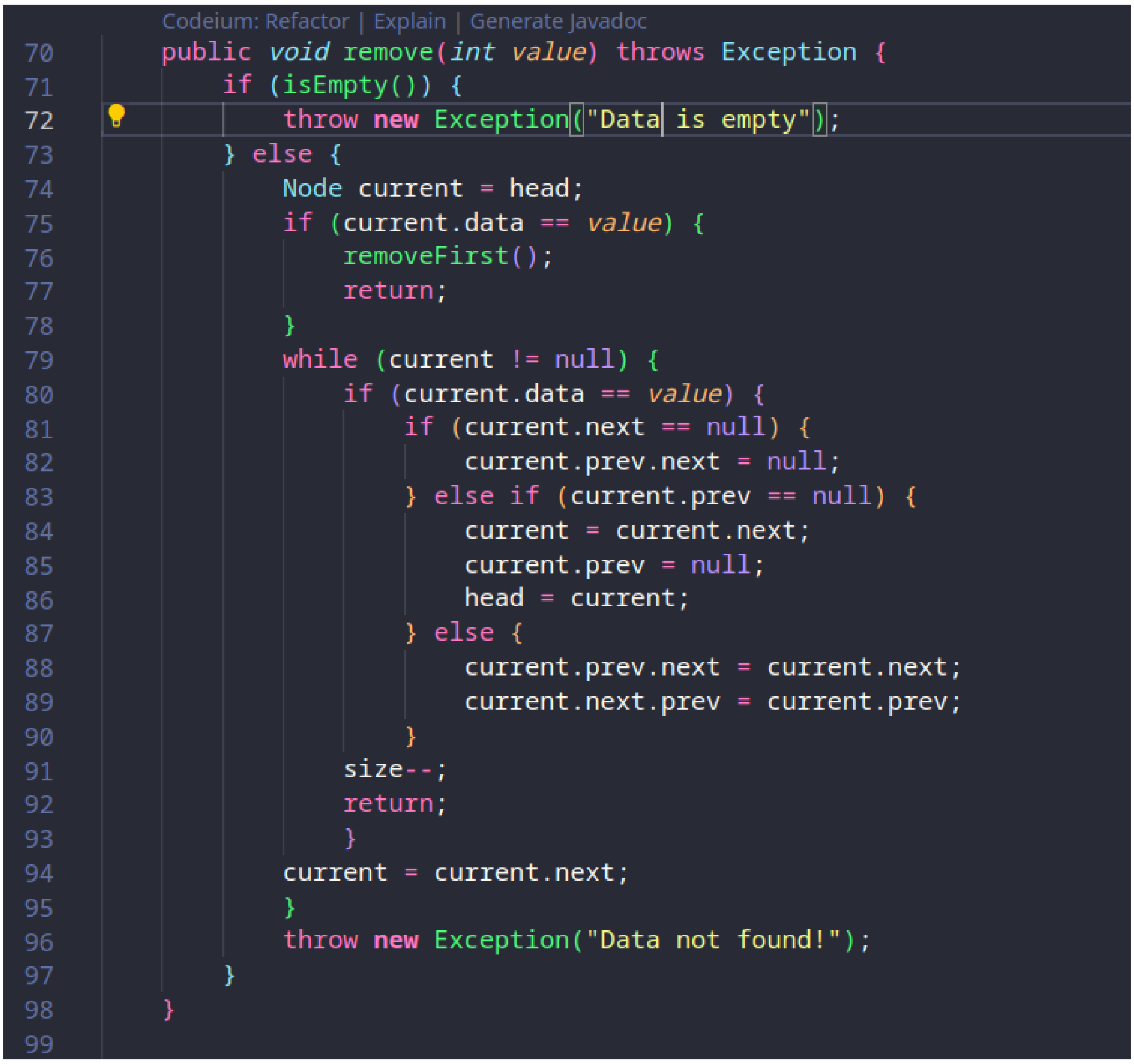
- Code :



```java
public void removeEdge(int source, int destination) throws Exception {
    list[source].remove(destination);
    if (!isDirected) {
        list[destination].remove(source);
    }
}
```

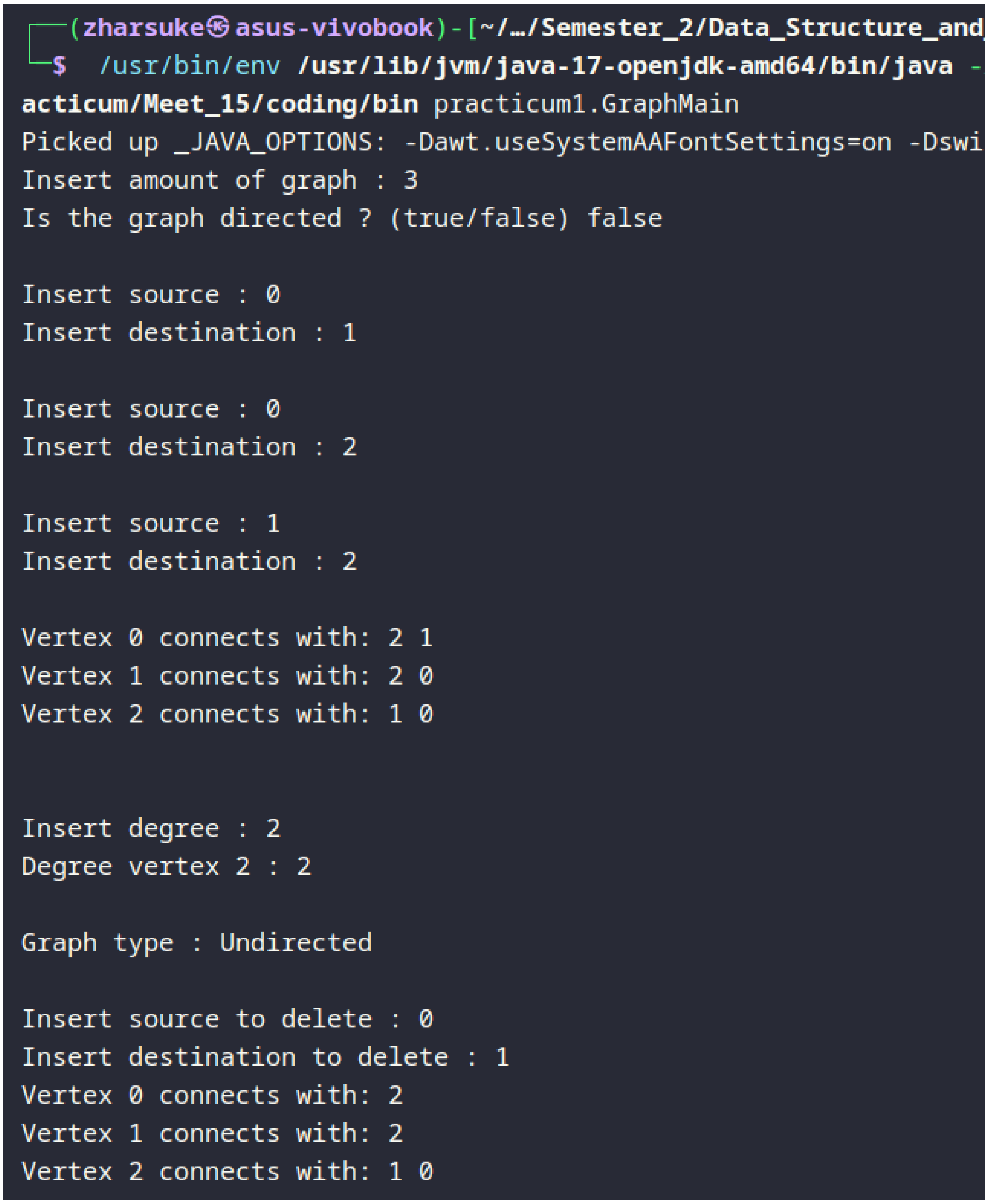


```java
        System.out.print("Insert source to delete : ");
        int source = sc.nextInt();
        System.out.print("Insert destination to delete : ");
        int destination = sc.nextInt();
        graph.removeEdge(source, destination);
        graph.printGraph();
        sc.close();
```



```java
    public void remove(int value) throws Exception {
        if (isEmpty()) {
            throw new Exception("Data is empty");
        } else {
            Node current = head;
            if (current.data == value) {
                removeFirst();
                return;
            }
            while (current != null) {
                if (current.data == value) {
                    if (current.next == null) {
                        current.prev.next = null;
                    } else if (current.prev == null) {
                        current = current.next;
                        current.prev = null;
                        head = current;
                    } else {
                        current.prev.next = current.next;
                        current.next.prev = current.prev;
                    }
                    size--;
                    return;
                }
                current = current.next;
            }
            throw new Exception("Data not found!");
        }
    }
```

- Result :

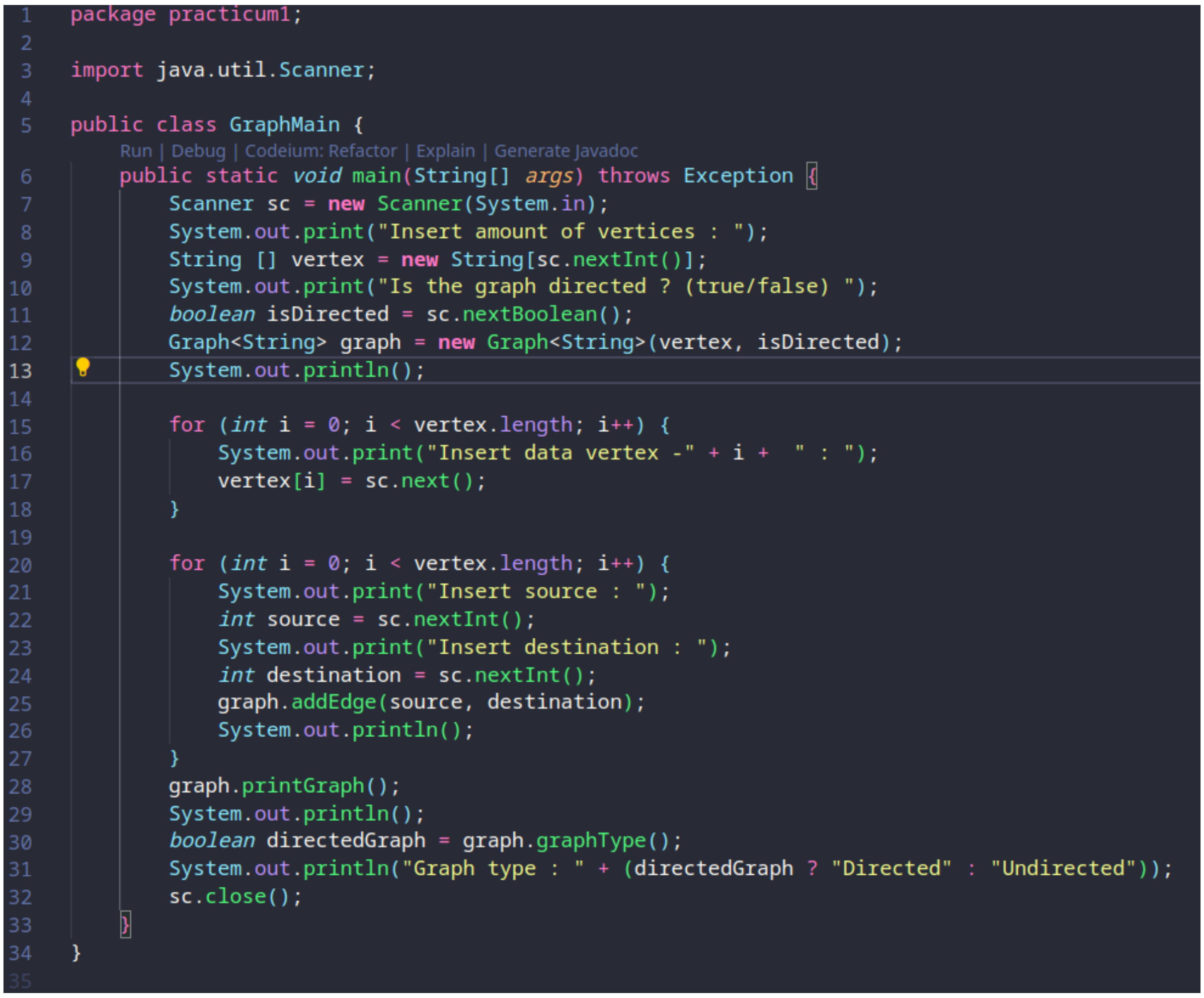


```
┌──(zharsuke㉿asus-vivobook)-[~/…/Semester_2/Data_Structure_and
└─$  /usr/bin/env /usr/lib/jvm/java-17-openjdk-amd64/bin/java -
acticum/Meet_15/coding/bin practicum1.GraphMain
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswi
Insert amount of graph : 3
Is the graph directed ? (true/false) false

Insert source : 0
Insert destination : 1

Insert source : 0
Insert destination : 2

Insert source : 1
Insert destination : 2

Vertex 0 connects with: 2 1
Vertex 1 connects with: 2 0
Vertex 2 connects with: 1 0


Insert degree : 2
Degree vertex 2 : 2

Graph type : Undirected

Insert source to delete : 0
Insert destination to delete : 1
Vertex 0 connects with: 2
Vertex 1 connects with: 2
Vertex 2 connects with: 1 0
```

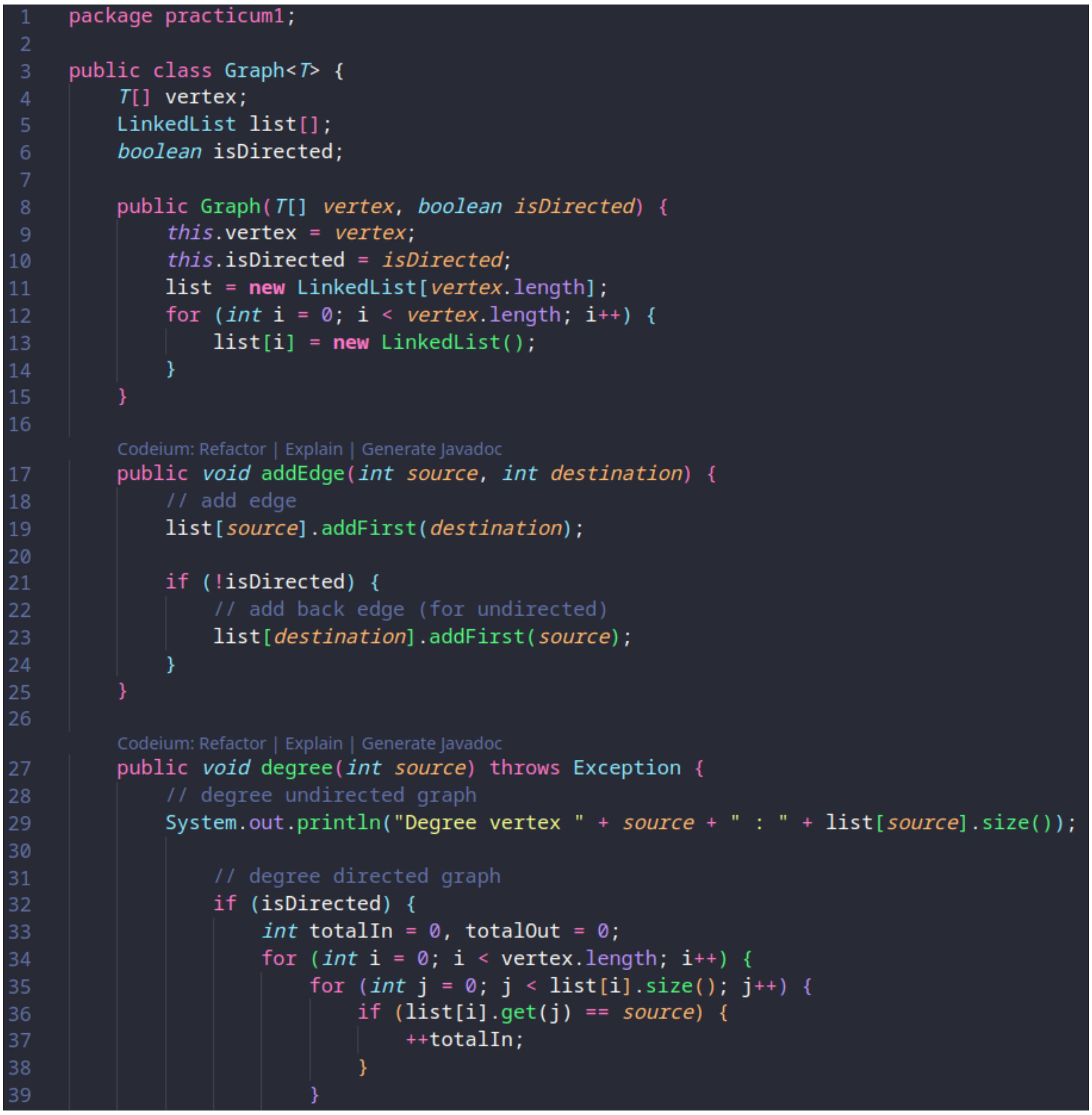
4. Convert vertex's data type in the graph of practicum 1.2. and 1.3 from integer to generic data type so that it can accepts all basic data type in Java programming language! For example, if the initial vertex are 0,1,2,3, dst. Then the next will be in form of region name, like Malang, Surabaya, Gresik, Bandung, dst.

- 1.2

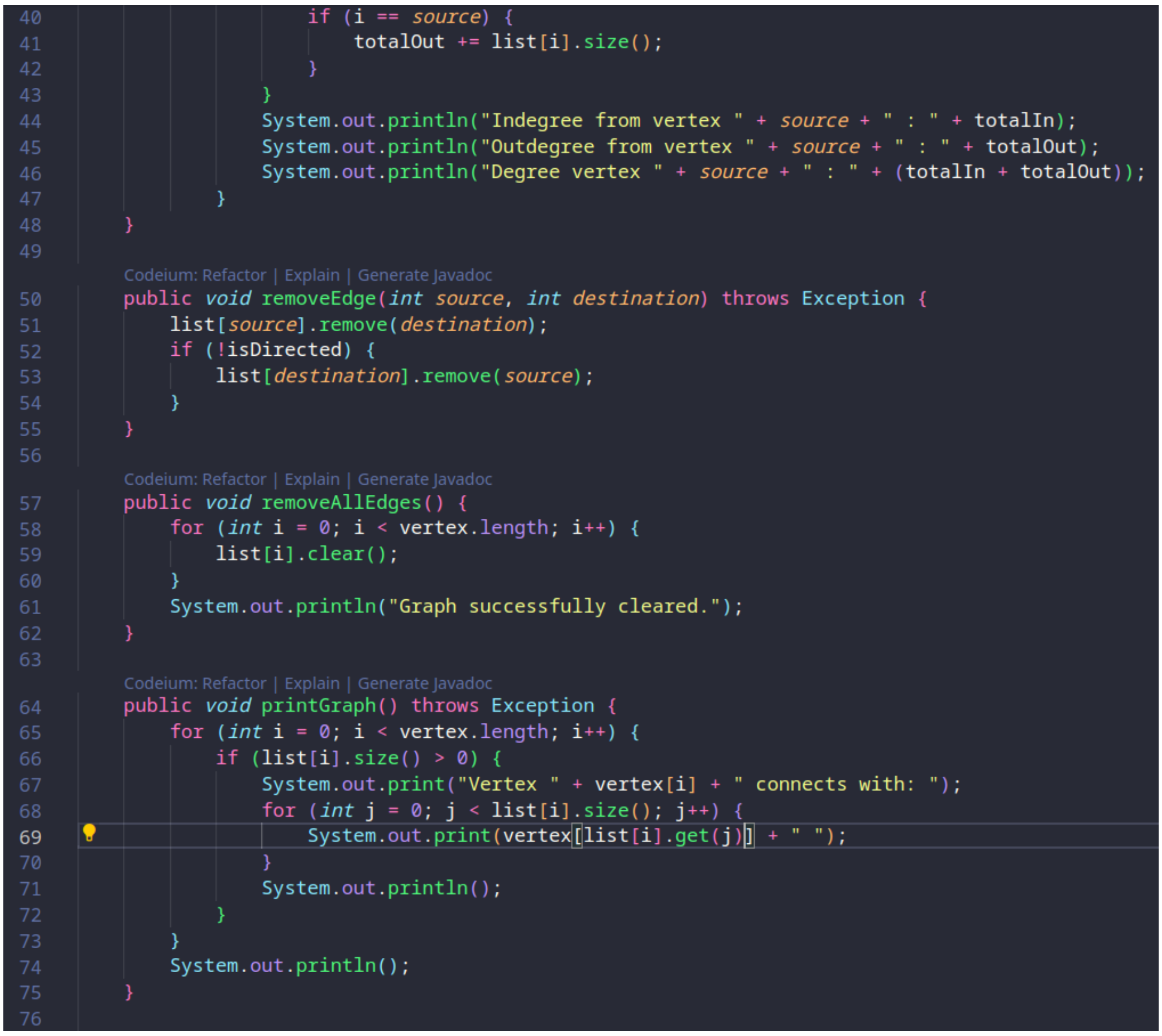
  Code :



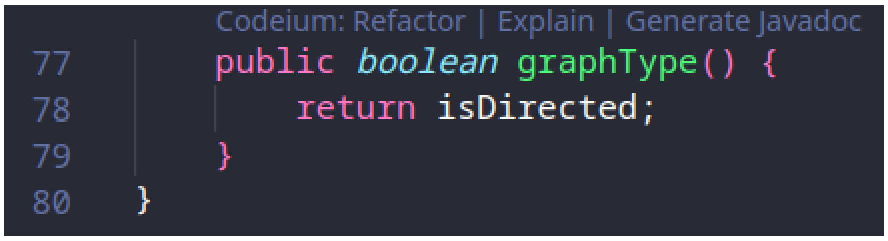
```
package practicum1;

import java.util.Scanner;

public class GraphMain {
    Run | Debug | Codeium: Refactor | Explain | Generate Javadoc
    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        System.out.print("Insert amount of vertices : ");
        String [] vertex = new String[sc.nextInt()];
        System.out.print("Is the graph directed ? (true/false) ");
        boolean isDirected = sc.nextBoolean();
        Graph<String> graph = new Graph<String>(vertex, isDirected);
        System.out.println();

        for (int i = 0; i < vertex.length; i++) {
            System.out.print("Insert data vertex -" + i +  " : ");
            vertex[i] = sc.next();
        }

        for (int i = 0; i < vertex.length; i++) {
            System.out.print("Insert source : ");
            int source = sc.nextInt();
            System.out.print("Insert destination : ");
            int destination = sc.nextInt();
            graph.addEdge(source, destination);
            System.out.println();
        }
        graph.printGraph();
        System.out.println();
        boolean directedGraph = graph.graphType();
        System.out.println("Graph type : " + (directedGraph ? "Directed" : "Undirected"));
        sc.close();
    }
}
```



```java
package practicum1;

public class Graph<T> {
    T[] vertex;
    LinkedList list[];
    boolean isDirected;

    public Graph(T[] vertex, boolean isDirected) {
        this.vertex = vertex;
        this.isDirected = isDirected;
        list = new LinkedList[vertex.length];
        for (int i = 0; i < vertex.length; i++) {
            list[i] = new LinkedList();
        }
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void addEdge(int source, int destination) {
        // add edge
        list[source].addFirst(destination);

        if (!isDirected) {
            // add back edge (for undirected)
            list[destination].addFirst(source);
        }
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void degree(int source) throws Exception {
        // degree undirected graph
        System.out.println("Degree vertex " + source + " : " + list[source].size());

            // degree directed graph
            if (isDirected) {
                int totalIn = 0, totalOut = 0;
                for (int i = 0; i < vertex.length; i++) {
                    for (int j = 0; j < list[i].size(); j++) {
                        if (list[i].get(j) == source) {
                            ++totalIn;
                        }
                    }
```

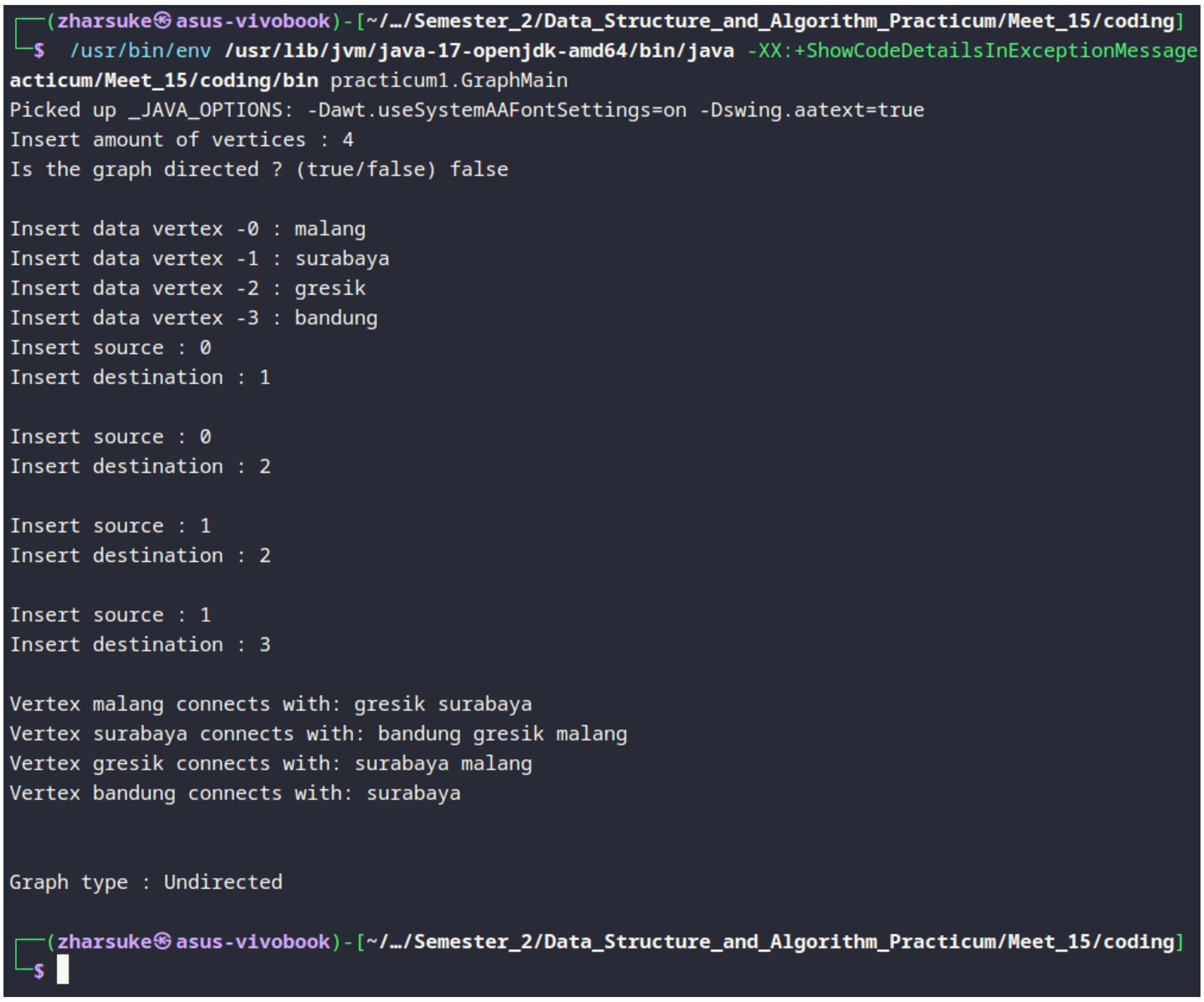


```java
                        if (i == source) {
                            totalOut += list[i].size();
                        }
                    }
                    System.out.println("Indegree from vertex " + source + " : " + totalIn);
                    System.out.println("Outdegree from vertex " + source + " : " + totalOut);
                    System.out.println("Degree vertex " + source + " : " + (totalIn + totalOut));
                }
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void removeEdge(int source, int destination) throws Exception {
        list[source].remove(destination);
        if (!isDirected) {
            list[destination].remove(source);
        }
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void removeAllEdges() {
        for (int i = 0; i < vertex.length; i++) {
            list[i].clear();
        }
        System.out.println("Graph successfully cleared.");
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void printGraph() throws Exception {
        for (int i = 0; i < vertex.length; i++) {
            if (list[i].size() > 0) {
                System.out.print("Vertex " + vertex[i] + " connects with: ");
                for (int j = 0; j < list[i].size(); j++) {
                    System.out.print(vertex[list[i].get(j)] + " ");
                }
                System.out.println();
            }
        }
        System.out.println();
    }

```



```java
    Codeium: Refactor | Explain | Generate Javadoc
    public boolean graphType() {
        return isDirected;
    }
}
```

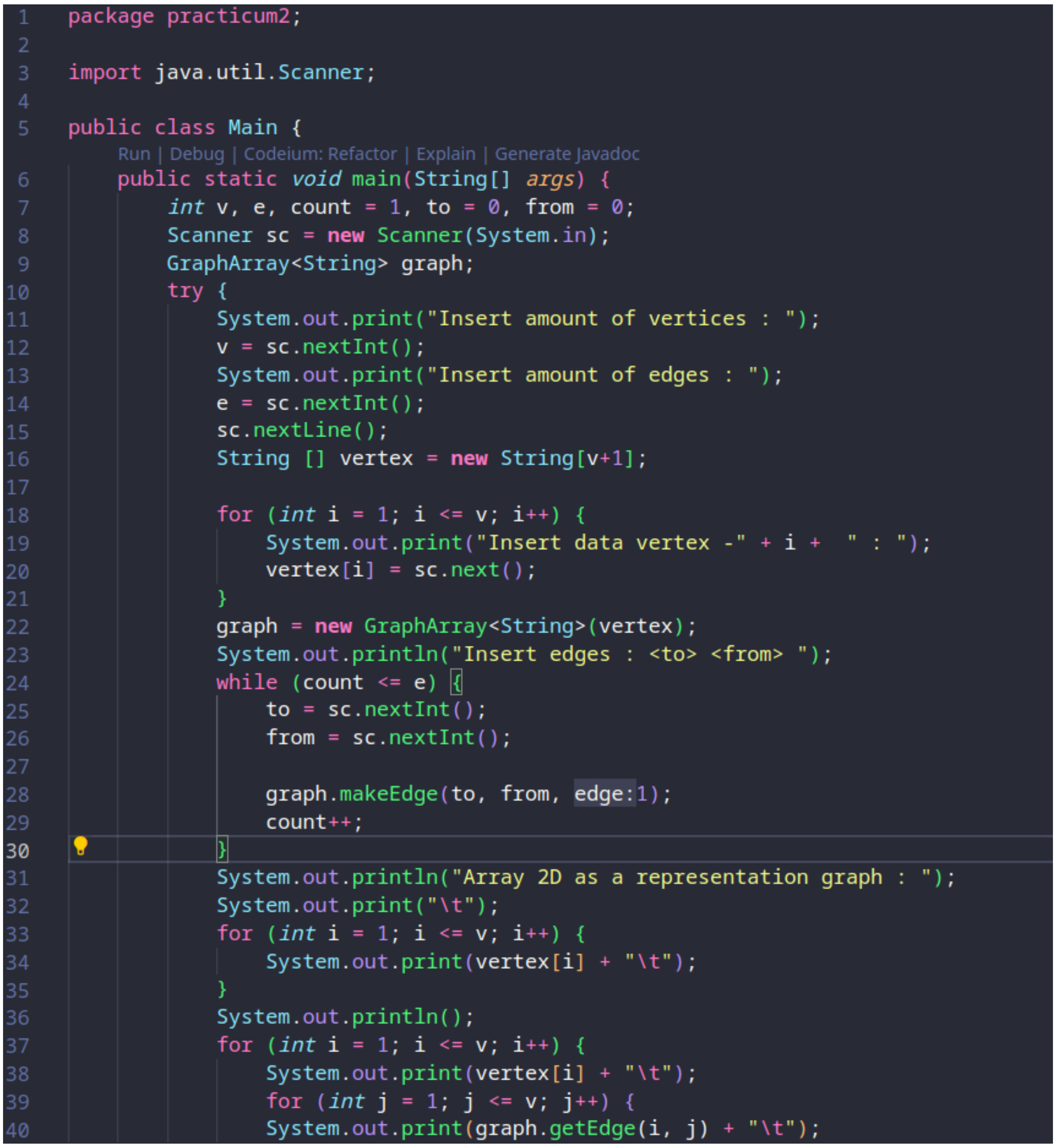
Result :



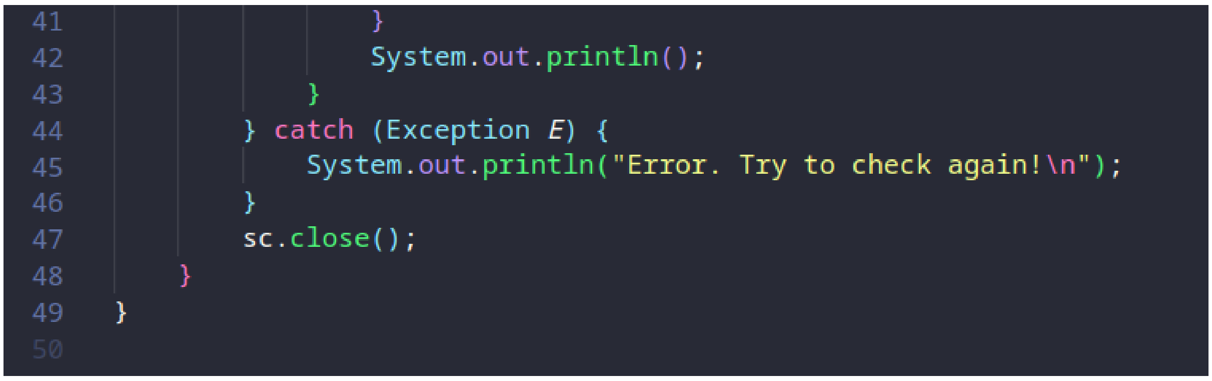
```
┌──(zharsuke㉿asus-vivobook)-[~/…/Semester_2/Data_Structure_and_Algorithm_Practicum/Meet_15/coding]
└─$ /usr/bin/env /usr/lib/jvm/java-17-openjdk-amd64/bin/java -XX:+ShowCodeDetailsInExceptionMessage
acticum/Meet_15/coding/bin practicum1.GraphMain
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Insert amount of vertices : 4
Is the graph directed ? (true/false) false

Insert data vertex -0 : malang
Insert data vertex -1 : surabaya
Insert data vertex -2 : gresik
Insert data vertex -3 : bandung
Insert source : 0
Insert destination : 1

Insert source : 0
Insert destination : 2

Insert source : 1
Insert destination : 2

Insert source : 1
Insert destination : 3

Vertex malang connects with: gresik surabaya
Vertex surabaya connects with: bandung gresik malang
Vertex gresik connects with: surabaya malang
Vertex bandung connects with: surabaya


Graph type : Undirected

┌──(zharsuke㉿asus-vivobook)-[~/…/Semester_2/Data_Structure_and_Algorithm_Practicum/Meet_15/coding]
└─$ █
```

- 1.3

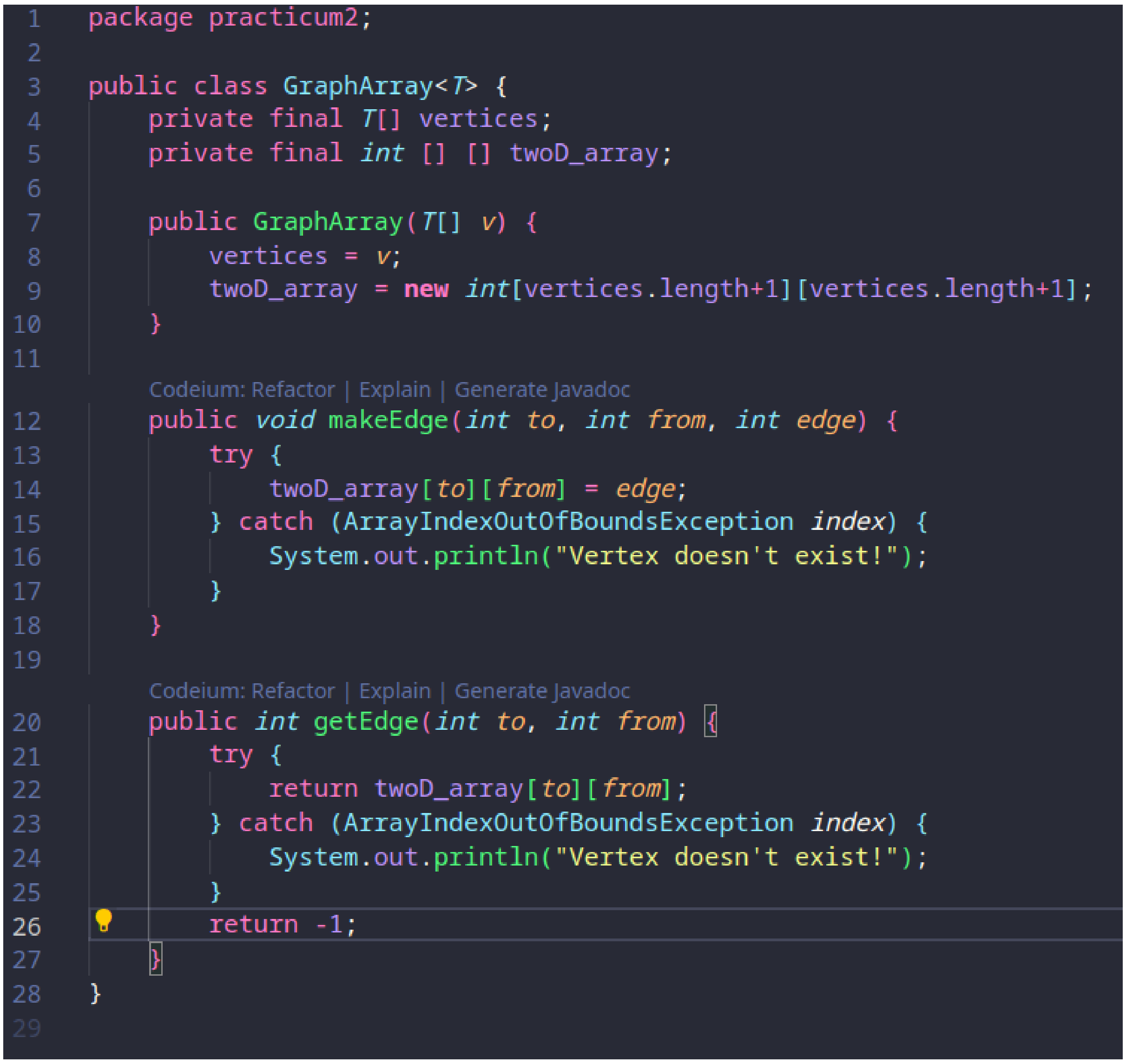
  Code :


Run | Debug | Codeium: Refactor | Explain | Generate Javadoc

```java
package practicum2;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        int v, e, count = 1, to = 0, from = 0;
        Scanner sc = new Scanner(System.in);
        GraphArray<String> graph;
        try {
            System.out.print("Insert amount of vertices : ");
            v = sc.nextInt();
            System.out.print("Insert amount of edges : ");
            e = sc.nextInt();
            sc.nextLine();
            String [] vertex = new String[v+1];

            for (int i = 1; i <= v; i++) {
                System.out.print("Insert data vertex -" + i +  " : ");
                vertex[i] = sc.next();
            }
            graph = new GraphArray<String>(vertex);
            System.out.println("Insert edges : <to> <from> ");
            while (count <= e) {
                to = sc.nextInt();
                from = sc.nextInt();

                graph.makeEdge(to, from, edge:1);
                count++;
            }
            System.out.println("Array 2D as a representation graph : ");
            System.out.print("\t");
            for (int i = 1; i <= v; i++) {
                System.out.print(vertex[i] + "\t");
            }
            System.out.println();
            for (int i = 1; i <= v; i++) {
                System.out.print(vertex[i] + "\t");
                for (int j = 1; j <= v; j++) {
                System.out.print(graph.getEdge(i, j) + "\t");
```

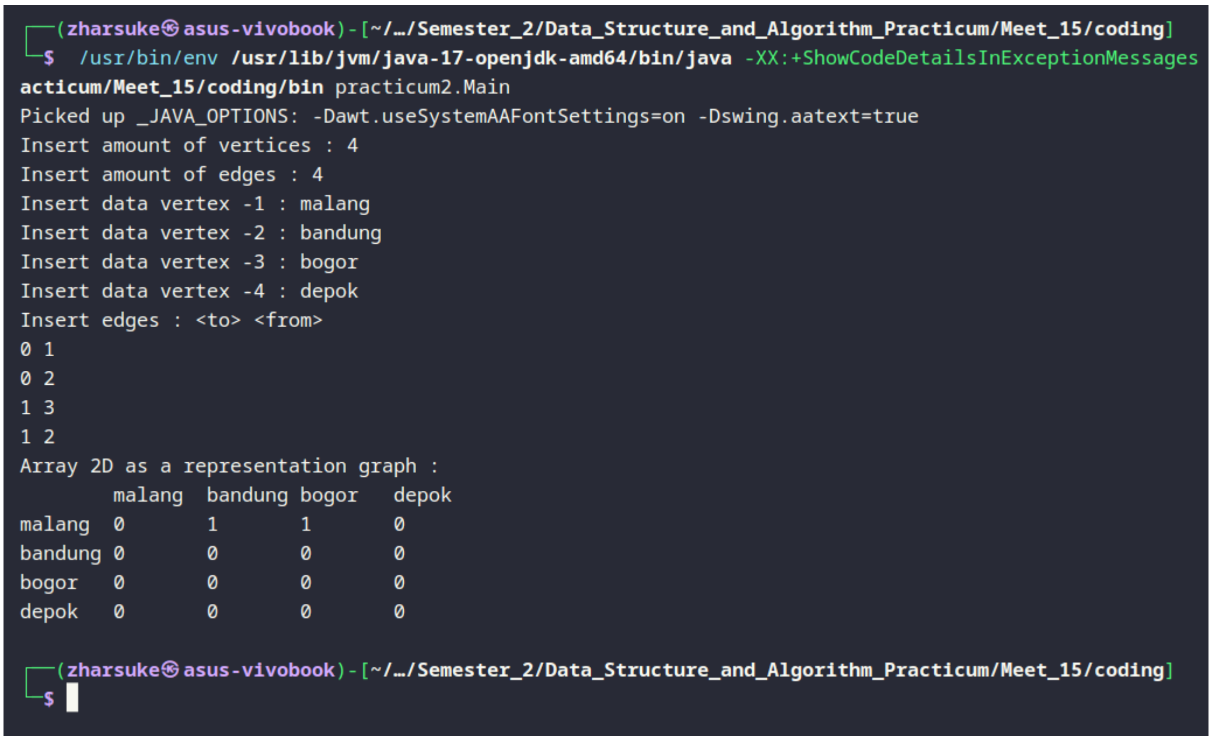


```java
                }
                System.out.println();
            }
        } catch (Exception E) {
            System.out.println("Error. Try to check again!\n");
        }
        sc.close();
    }
}
```



```java
package practicum2;

public class GraphArray<T> {
    private final T[] vertices;
    private final int [] [] twoD_array;

    public GraphArray(T[] v) {
        vertices = v;
        twoD_array = new int[vertices.length+1][vertices.length+1];
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void makeEdge(int to, int from, int edge) {
        try {
            twoD_array[to][from] = edge;
        } catch (ArrayIndexOutOfBoundsException index) {
            System.out.println("Vertex doesn't exist!");
        }
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public int getEdge(int to, int from) {
        try {
            return twoD_array[to][from];
        } catch (ArrayIndexOutOfBoundsException index) {
            System.out.println("Vertex doesn't exist!");
        }
        return -1;
    }
}
```

- Result :



```
┌──(zharsuke㉿asus-vivobook)-[~/…/Semester_2/Data_Structure_and_Algorithm_Practicum/Meet_15/coding]
└─$  /usr/bin/env /usr/lib/jvm/java-17-openjdk-amd64/bin/java -XX:+ShowCodeDetailsInExceptionMessages
acticum/Meet_15/coding/bin practicum2.Main
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Insert amount of vertices : 4
Insert amount of edges : 4
Insert data vertex -1 : malang
Insert data vertex -2 : bandung
Insert data vertex -3 : bogor
Insert data vertex -4 : depok
Insert edges : <to> <from>
0 1
0 2
1 3
1 2
Array 2D as a representation graph :
        malang  bandung bogor   depok
malang  0       1       1       0
bandung 0       0       0       0
bogor   0       0       0       0
depok   0       0       0       0

┌──(zharsuke㉿asus-vivobook)-[~/…/Semester_2/Data_Structure_and_Algorithm_Practicum/Meet_15/coding]
└─$ 
```