1. Explain 5 refactoring

Link commit: https://github.com/welldonepig/CSE-464-2023-tchen128/commit/f84d5a9274e2bece9ff33406679ea66cc5cd06ff

a. Add logger private static final Logger logger = Logger.getLogger(GraphParser.class.getName());

purpose to replace e.printStackTrace() in catching exception to logger.log(Level.SEVERE, "An exception occurred", e) so that we can use another log level to alert or in the future we want to export log to file easier.

b. Merge 2 catch clause with the same handle improve the readability

```
catch(NoSuchElementException e) {
   e.printStackTrace();
} catch (ImportException e) {
   e.printStackTrace();
}
```

--> catch(NoSuchElementException | ImportException e)

c. Remove duplicate function to improve the readability

remove public void outputDOTGraph(String filePath) because it have the same implement with public void outputGraph(String filePath)

d. Change this instance-reference to a static reference because it is static method
parser.outputGraphics("output.dot", "output.png") --> outputGraphics("output.dot", "output.png")

e. Remove redundant toString call and add final to immutable field prevent change in process
System.out.println(parser.toString());
->System.out.println(parser);

private List<String> nodes;
->private final List<String> nodes;

2. Apply Template method pattern

Link commit: https://github.com/welldonepig/CSE-464-2023-tchen128/commit/ff619f8d9ba4870c98e6f47df642f5dfd65f9dbf

I create abstract class GraphSearchTemplate to act as a template for BfsAlgo and DfsAlgo implementation

- Main flow for every implement search function will include 5 methods:

```
public Path search(String src, String dst) {
    init(src);

    while (!queueIsEmpty()) {
        String currentNode = getNextNode();
        if (currentNode.equals(dst)) {
            return constructPath(currentNode);
        }

        processNeighbors(currentNode);
    }
    return null;
}
```

init (abstract method)
queueIsEmpty (abstract method)
getNextNode (abstract method)
processNeighbors (abstract method)
constructPath (common)

- BfsAlgo and DfsAlgo will implement this flow to do the search

```
public class DfsAlgo extends GraphSearchTemplate {

    private final Deque<String> stack;

    public DfsAlgo(DirectedGraph<String, DefaultEdge> graph) {...}

    @Override
    public void init(String src) {
        stack.push(src);
        visited.add(src);
    }

    @Override
    public boolean queueIsEmpty() {
        return stack.isEmpty();
    }

    @Override
    public String getNextNode() {
        return stack.pop();
    }

    @Override
    public void processNeighbors(String currentNode) {
        Set<DefaultEdge> outgoingEdges = graph.outgoingEdgesOf(currentNode);
        for (DefaultEdge edge : outgoingEdges) {
            String edgeTarget = graph.getEdgeTarget(edge);
            if (!visited.contains(edgeTarget)) {
                stack.push(edgeTarget);
                parentMap.put(edgeTarget, currentNode);
                visited.add(edgeTarget);
            }
        }
    }
}
```

Benefit is about code reusability by encapsulate common behavior reduce duplicate code. Beside we can enforce design into child class make it more organized code.

Example input and output

```
GraphParser parser = new GraphParser();
parser.parseGraph( filePath: "input2.dot");
Path res = parser.graphSearch( src: "a",  dst: "c", Algorithm.BFS);
System.out.println("Path:" + res);
```

```
Path:a -> b -> c
```

```
GraphParser parser = new GraphParser();
parser.addNodes(new String[] {"a", "b", "c", "d", "e", "f"});
parser.addEdge( srcLabel: "a", dstLabel: "b");
parser.addEdge( srcLabel: "a", dstLabel: "e");
parser.addEdge( srcLabel: "e", dstLabel: "f");
parser.addEdge( srcLabel: "f", dstLabel: "c");
Path res = parser.graphSearch( src: "a",  dst: "c", Algorithm.DFS);
System.out.println("Path:" + res);
```

```
Path:a -> e -> f -> c
```

3. Apply Strategy pattern

Link commit: https://github.com/welldonepig/CSE-464-2023-tchen128/commit/698ddf2b683bbcd56192ce6586bfef6776d1cc44

I create SearchStrategy interface with 1 abstract method(executeSearch) to apply this strategy pattern and then I create 2 concrete class are DfsStrategy and BfsStrategy for implementaton.
Both strategy class represents a strategy for performing a search operation, and it uses the Strategy Pattern to allow different search algorithms to be used interchangeably. The actual search algorithm is encapsulated in the GraphSearchTemplate and by creating different implementations of this interface I can swap different search algorithms without modifying the client code.

```
public interface SearchStrategy {
    Path executeSearch(String src, String dst);
}
```

```
public class BfsStrategy implements SearchStrategy {
    private GraphSearchTemplate template;

    public BfsStrategy(GraphSearchTemplate template) { this.template = template; }

    @Override
    public Path executeSearch(String src, String dst) { return template.search(src, dst); }
}
```

Example to run the code is the same with part 2 template pattern

```
GraphParser parser = new GraphParser();
parser.addNodes(new String[] {"a", "b","c","d","e","f", "g","h"});
parser.addEdge( srcLabel: "a", dstLabel: "b");
parser.addEdge( srcLabel: "a", dstLabel: "c");
parser.addEdge( srcLabel: "c", dstLabel: "d");
parser.addEdge( srcLabel: "c", dstLabel: "e");
parser.addEdge( srcLabel: "b", dstLabel: "g");
parser.addEdge( srcLabel: "c", dstLabel: "h");
Path res = parser.graphSearch( src: "a",  dst: "h", Algorithm.BFS);
System.out.println(res);
Path res2 = parser.graphSearch( src: "a",  dst: "g", Algorithm.DFS);
System.out.println(res2);
```

```
a -> c -> h
a -> b -> g
```

4. Random walk search

Link commit: https://github.com/welldonepig/CSE-464-2023-tchen128/commit/7fd5374b66889550db5aa90dfa35ff7d2cd72c81

Implementation idea:

- Starting from a specified source node we find all child node of that and add to **currentNeighbors** list then we keep traversing the graph by randomly selecting unvisited neighbors in list
- In each step we will keep track of visited node and parent map
- The process go on in each step we will add new neighbor node into list and just pick 1 random to find the destination node.
- Till we find the destination node the process stop and log the result from parentMap.

Example to run the code

```
GraphParser parser = new GraphParser();
parser.parseGraph( filePath: "input2.dot");
Path res = parser.graphSearch( src: "a",  dst: "c", Algorithm.RANDOM);
System.out.println("Path:" + res);
System.out.println("--------------------------------");
Path res2 = parser.graphSearch( src: "a",  dst: "c", Algorithm.RANDOM);
System.out.println("Path:" + res2);
System.out.println("--------------------------------");
Path res3 = parser.graphSearch( src: "a",  dst: "c", Algorithm.RANDOM);
System.out.println("Path:" + res3);
```

Result random walk

```
Visit:e
Visit:b
Visit:g
Visit:f
Visit:h
Visit:c
Path:a -> b -> c
--------------------------------
Visit:e
Visit:g
Visit:h
Visit:f
Visit:b
Visit:c
Path:a -> b -> c
--------------------------------
Visit:b
Visit:c
Path:a -> b -> c
```

Input

```
GraphParser parser = new GraphParser();
parser.addNodes(new String[] {"a", "b","c","d","e","f", "g","h"});
parser.addEdge( srcLabel: "a", dstLabel: "b");
parser.addEdge( srcLabel: "a", dstLabel: "c");
parser.addEdge( srcLabel: "c", dstLabel: "d");
parser.addEdge( srcLabel: "c", dstLabel: "e");
parser.addEdge( srcLabel: "b", dstLabel: "g");
parser.addEdge( srcLabel: "c", dstLabel: "h");
Path res = parser.graphSearch( src: "a",  dst: "h", Algorithm.RANDOM);
System.out.println("Path:" + res);
System.out.println("--------------------------------");
Path res2 = parser.graphSearch( src: "a",  dst: "h", Algorithm.RANDOM);
System.out.println("Path:" + res2);
System.out.println("--------------------------------");
Path res3 = parser.graphSearch( src: "a",  dst: "h", Algorithm.RANDOM);
System.out.println("Path:" + res3);
```

Output

```
Visit:c
Visit:d
Visit:b
Visit:g
Visit:h
Path:a -> c -> h
--------------------------------
Visit:c
Visit:h
Path:a -> c -> h
--------------------------------
Visit:c
Visit:b
Visit:g
Visit:e
Visit:d
Visit:h
Path:a -> c -> h
```

5. Pull request link

https://github.com/welldonepig/CSE-464-2023-tchen128/pull/3