```
public class Graph {

        /**

        Graph is a modifiable structure that consists of Nodes (which are String objects) and Edges
        (which are implemented as a distinct class).

        Representation Invariant: Nodes within the graph must be distinct, and Edges should only
        connect Nodes that exist within the graph.

        Abstraction Function: Graph can be seen as a container of Nodes and Edges. Specifically, the
        Graph is represented by the collection of Nodes denoted by "ex.nodes," and the collection of
        Edges denoted by "ex.edges."

        */




        /**
                @effects: Constructs a new empty Graph
         */
        public Graph(){
                throw new RuntimeException();
        }


        /**
                @param: new edge new_edge is added
                @modifies: edges
                @effects: new_edge added to edges
                @throws: If the graph does not contain either of the two nodes stored in the new_edge.
        */
        public void addEdge(Edge new_edge) {
            throw new RuntimeException();
}




/**
            @param: adds new node new_node
            @modifies: nodes
            @effects: if new_node is not already present, adds new_node to nodes
            @throws: if new_node is already present in the graph or is null
 */
public void addNode(String new_node) {
            throw new RuntimeException();
}

/**
            @return: this Graph object's Edges
 */
public TreeSet<Edge> getEdges() {
            throw new RuntimeException();
}

/**
            @return: this Graph object's Nodes
 */
public TreeSet<String> getNodes() {
            throw new RuntimeException();
}
```

```
    /**
        @throw: if representation invariant is violated
     */
    private void checkRep() throws RuntimeException {
        throw new RuntimeException();
    }

}

public class Edge {

    /**
        Graph is a fixed object that holds a labeled directed edge connecting two nodes.
        Representation Invariant: The beginning node, ending node, and label cannot be null.
        Abstraction Function: There is a node called "ex" that has two nodes represented by
        "ex.start" and "ex.end," and a label for the edge represented by "ex.label.
    */

    /**
```

```
            @param: node1 -> start-node of the new Edge
            @param: node2 -> end-node of the new Edge
            @param: lab -> length of the new Edge
            @effects: establishes a label lab for a newly created edge that connects node1 to node2
 */
public Edge(String node1, String node2, String lab) {
            throw new RuntimeException();
}


/**
            @return: this Edge object's parent Node
 */
public String getParent() {
            throw new RuntimeException();
}


/**
            @return: this Edge object's child Node
 */
public String getChild() {
            throw new RuntimeException();
}



/**
            @return: this Edge object's label
```

```
         */
        public String getLabel() {
                throw new RuntimeException();
        }


        /**
                @param: an Edge other_edge for this edge to be compared to
                @requires: other_edge is not null
                @return: if this = other_edge returns 0, if this > other_edge returns positive number, or
                if this < other_edge returns negative number.
         */
        @Override
        public int compareTo(Edge other_edge) {
             throw new RuntimeException();
        }


        /**
                @throw: if the object violates the representation invariant
         */
        private void checkRep() throws RuntimeException {
                throw new RuntimeException();
        }
}
```