GraphEdge

```
/**
 * <b>GraphEdge</b> represents an <b>immutable</b> directed edge to be used in a directed labeled
multi-graph.
 * Only the edge's label and destination node name are stored. The edge's
 * source node name is found within the graph's list of nodes and is used
 * to reference that node's outgoing edges; typical of adjacency list representation.
 * <p>
 */

      /**
       * Abstraction Function:
       * GraphEdge e represents a directed edge where:
       * e.label = label of the edge that may represent its weight
       * e.dest = name of destination node
       *
       *
       * Representation Invariant for every GraphEdge e:
       * e.dest != null && e.dest != empty string
       * e.label != null
       *
       */

    /**
     * @param d The GraphEdge's destination node name
     * @effects Constructs a new GraphEdge with
     *                    label = "" (empty string),
     *                    dest = d
     */
      public GraphEdge String d)


    /**
     * @param d The GraphEdge's destination node name
     * @param l The GraphEdge's label
     * @effects Constructs a new GraphEdge with
     *                    label = l,
     *                    dest = d
     */
      public GraphEdge String d, String l)

    /**
     * @return the name of this edge's destination node
     */
      public String getDest()

    /**
     * @return the label of this edge
     */
      public String getLabel()

    /** Standard hashCode function.
      @return an int that all objects equal to this will also
      return.
      */
      @Override
      public int hashCode()

    /** Compares two GraphEdges by lexicographically by
     * destination node name and secondarily by label
```

```
        @param e the GraphEdge to be compared
        @requires e != null
        @return a negative number if this < e,
        0 if this = e,
        a positive number if this > e.
     */
     @Override
     public int compareTo GraphEdge e

  /** Standard equality operation.
        @param obj The object to be compared for equality.
        @return true if and only if 'obj' is an instance of a GraphEdge
        and 'this' and 'obj' represent identical edges,
        where "identical" means having same label and destination node
     */
     @Override
     public boolean equals /*@Nullable*/ Object obj
```

GraphNode

```
/**
 * <b>GraphNode</b> represents a <b>mutable</b> node in a directed labeled multi-graph,
 * with an <b>immutable</b> name and a <b>mutable</b> collection of <b>immutable</b> GraphEdge
 * objects representing its outgoing edges.
 * <p>
 */

      /**
       * Abstraction Function:
       * GraphNode n represents a node of a graph where
       * n.name = name of the node
       * n.edges = collection of node's outgoing edges
       *
       *
       * Representation Invariant for every GraphNode n:
       * n.name != null && n.name != empty string
       * n.edges != null
       * n.edges does not contain duplicates and nulls (handled by GraphEdge comparable)
       *
       */

  /**
   * @param n The GraphNode's name
   * @effects Constructs a new GraphNode with name = n,
   *                    and empty set of edges
   */
    public GraphNode String n

  /**
   * @param e The GraphEdge to add to this node
   * @modifies this.edges
   * @effects Adds edge e to this.edges if it doesn't already exist
   * @return true if operation successful, false otherwise
   */
    public boolean addEdge GraphEdge e

  /**
   * @return the name of this node's name
   */
    public String getName
```

```
    /**
     * @return an int representing number of outgoing edges
     */
      public int outDegree()

    /** Determines whether two different nodes have the identical set of edges
      @param n The GraphNode to be compared with
      @return true if and only if this.edges is the same set of GraphEdges as n.edges
      */
      public boolean hasSameEdges (GraphNode n)

    /**
     * @return a list of Strings representing this.edges,
     *                   formated as [e.dest]([e.label]) where e is an arbitrary edge,
     *                   and maintains the same order as in this.edges
     */
      public LinkedList<String> edgeStrings ()

    /** Standard hashCode function.
            @return an int that all equal GraphNodes will return
      */
      @Override
      public int hashCode()

    /** Compares two GraphNodes lexicographically by name
          @param e the GraphNode to be compared
          @requires e != null
          @return a negative number if this < e,
                      0 if this = e,
                      a positive number if this > e.
      */
      @Override
      public int compareTo(GraphNode n)

    /** Standard equality operation.
          @param obj The object to be compared for equality.
          @return true if and only if 'obj' is an instance of a GraphNode
                      and 'this' and 'obj' represent identical nodes
                      where "identical" means having same name
      */
      @Override
      public boolean equals(/*@Nullable*/ Object obj)
```

Graph

```
/**
 * <b>Graph</b> represents a <b>mutable</b> directed labeled multi-graph
 * containing a set of GraphNode objects. GraphNodes store the represented node's
 * name and its set of edges, which are out-edges. Each edge in a GraphNode is a GraphEdge object
that
 * stores its label and its destination node name (a child node of the current GraphNode).
 * <p>
 */

      /**
       * Abstraction Function:
       * Graph g represents a graph where
       * g.nodes = its collection of nodes represented by GraphNode objects,
       * which each store its out-edges as a collection
       *
       *
       * Representation Invariant for every Graph g:
```

```
     * g.nodes != null
     * g.nodes does not contain duplicates and nulls (handled by GraphNode comparable)
     *
     */

    /**
     * @effects Constructs a new Graph with empty set of nodes
     */
      public Graph()

    /**Adds a new node to the graph if it doesn't already exist
     *
     * @param nodeName The name of the new node to add
     * @modifies this.nodes
     * @effects adds a new node to this.nodes with name = nodeName if it doesn't already exist.
     * @returns true if successful in adding new node, false otherwise
     */
      public boolean addNode String nodeName)

    /**Adds a new edge to the graph if its starting and ending nodes both exist
     * and it doesn't already exist
     *
     * @param parentName The starting node's name
     * @param childName The ending node's name
     * @modifies node with name = parentName in this.nodes
     * @effects adds a new edge to a node in this.nodes with name = nodeName if its starting and
ending nodes both exist
     *                    and it doesn't already exist.
     * @returns true if successful in adding new node, false otherwise
     */
      public boolean addEdge(String parentName, String childName, String label)

    /**
     * @return a list of Strings representing this.nodes,
     *                    formated as n.name where n is an arbitrary node,
     *                    and maintains the same order as in this.nodes
     */
      public LinkedList<String> nodeNames()

    /**
     * @param nodeName The name of the node to get
     * @return a GraphNode in this.nodes
     */
      public GraphNode getNode (String nodeName)
```