Overview
The Graph ADT represents a directed labeled multi-graph where:

- Nodes are uniquely identified by string names
- Edges are directed and labeled with string names
- Multiple edges can exist between the same pair of nodes as long as they have unique labels
- Duplicate nodes are not allowed (each node name must be unique)

boolean addNode(string nodeAdded)
@Param: string name from nodeAdded
@Requires: a non-null string that isn't on the graph
@Modifies: this graph structure
@Effects: adds a new node with the given string name
If there's already a node with that name the graph structure remains unchanged
@Throws: NullPointerException if the string is null
@Return: true if a new node was added successfully or false if the node already exists

Boolean removeNode(string nodeRemove)
@Param: string name from nodeRemove
@Requires: a non-null string that is on the graph
@Modifies: this graph structure
@Effects: removes the node and all of its edges if the string exists
If the string doesn't exist, then the graph structure remains unchanged
@Throws: NullPointerException if the string is null
@Return: true if the node and its edges were successfully removed
false if the node doesn't exist

Boolean addEdge(String from, String to, String label)
@Param: string name for from, to, and label
@Requires: a non–null string that is on the graph for from and to, and the label of the edge doesn't already exist
@Modifies: this graph structure
@Effects: adds a new edge with the given string label between the nodes from and to
If the node from or to doesn't exist, then the graph is unchanged
If both nodes exist and the label doesn't already exist between the nodes from and to, then the edge is added
@Throws: NullPointerException if any of the strings are null
IllegalArgumentException if either the node to or the node from doesn't exist
@Return: true if the edge was successfully added
false if the edge label already exists between from and to

Boolean removeEdge(String from, String to, String label)
@Param: string name for from, to, and label
@Requires: a non-null string that is on the graph for from and to, and the label of the edge

exists between them
@Modifies: this graph structure
@Effects: removes the edge with the given string label between the nodes from and to
If the node from or to doesn't exist, then the graph is unchanged
If both nodes exist and the label exists between the nodes from and to, then the edge is removed
@Throws: NullPointerException if any of the strings are null
IllegalArgumentException if either the node to or the node from doesn't exist
@Return: true if the edge was successfully removed
false if the edge doesn't exist between from and to

Boolean hasNode(String nodeCheck)
@Param: string name for nodeCheck
@Requires: a non-null string
@Modifies: none
@Effects: none
@Throws: NullPointerException if the string is null
@Return: true if the nodeCheck exists in the graph, false otherwise

Boolean hasEdge(String from, String to, String label)
@Param: string name for from, to, and label
@Requires: a non-null string for from, to, and label
@Modifies: none
@Effects: none
@Throws: NullPointerException any of the strings are null
@Return: true if an edge with the given label exists between from and to, false otherwise

List<String> getChildren(String node)
@Param: string name for node
@Requires: a non-null string that exists in the graph
@Modifies: none
@Effects: none
@Throws: NullPointerException if the string is null
IllegalArgumentException if the node doesn't exist in the graph
@Return: a list of string names representing the nodes that have an edge from the given node
If there are no outgoing edges, returns an empty list

List<String> getEdges(String from, String to)
@Param: string names for from and to
@Requires: non-null strings for from and to that exist in the graph
@Modifies: none
@Effects: none
@Throws: NullPointerException if from or to is null
IllegalArgumentException if either from or to doesn't exist in the graph

@Return: a list of string labels representing the edges between from and to
If there are no edges between them, returns an empty list

@Representation Invariant:

- Nodes must be unique and represented as Strings
- Edges that connect existing nodes and have unique labels per (from, to) pair
- No null nodes or edges allowed

@Abstraction Function:

- The `adjacencyList` map represents the graph where:
    - Each key is a node
    - Each value is another map which represents edges from this node
    - The inner map stores destination nodes as keys and their edge labels as values