The NodeIndex Specification

Overview: The *NodeIndex* is designed to maintain bidirectional relations between node labels (strings), node IDs (integers), and *Node* objects. It ensures consistency between the mappings of labels to IDs and IDs to labels, allowing efficient addition, retrieval, update, and removal of nodes.

Mathematical Definitions of Sets and Functions: Let Labels be the set of all valid node labels (non-empty strings without whitespace):

$$L = \{\ell \in Strings | \ \ell \neq \text{null}, \ell \neq \ , \ell \ contaions \ non - whitespace \ characters\}$$

Let ***NodeIds*** be the set of all valid *nodeIds* (integers): $I = \{i \in Z\}$

Let ***Nodes*** be the set of all valid *Node* objects: $N = \{n \mid n.Id, n.Label \ \in L\}$

Define ***Label-to-ID Map*** as a function: $L2I : L \to I \times N$

This maps a node label (case-insensitive) to a tuple of node ID and *Node* reference.

Define ***ID-to-Label Map*** as a function: $I2L : I \to L$

This maps a *nodeId* to its label.

Invariants: The following invariants must always hold:

Bidirectional Consistency:

$$\forall \ell \in Dom(L2I), (\ell, (i, n)) \in L2I \implies (i, \ell) \in I2L$$

$$\forall i \in dom(I2L), (i, \ell) \in I2L \implies (\ell, (i, n)) \in L2I$$

Unique Mapping: Labels map to a single Id and Node. Ids map to a single label.

Case-Insensitive Labels: Label comparisons are case-insensitive in *L2I*.

Operations

1. AddOrUpdateNode(n)

Purpose: Add a new node or update an existing node in the index.

Input: $n \in N$

Preconditions: $n.Label \in L$ and $n.Id \in I$

Process: **Consistency Check and Cleanup**:

$If\ there\ exists\ \ell \in dom(L2I)\ such\ that\ lower(\ell) = lower(n.Label)$:

$Let\ (i\ell, n\ell) = L2I(\ell)$

$If\ i\ell \neq n.Id$

$Remove\ \ell\ from\ L2I$

$Remove\ i\ell\ from\ I2L$

$If\ there\ exists\ i \in dom(I2L)\ such\ that\ i = n.Id$

$Let\ \ell i = I2L(i)$

$If\ lower(\ell i) \neq lower(n.Label)$

$Remove\ \ell i\ from\ L2I$

$Remove\ i\ from\ I2L$

Insertion/Update:

Add or update the mapping:

$L2I(n.Label) = (n.Id, n)$

$I2L(n.Id) = n.Label$

Postconditions: Mappings are updated to reflect n. Invariants are maintained.

2. RemoveNodeByLabel(ℓ)

Purpose: Remove a node from the index using its label.

Input: $\ell \in L$

Preconditions: ℓ is not null, empty, or whitespace.

Process Lookup:

$If\ \ell \in dom(L2I)$

$Let\ (i, n) = L2I(\ell)$

$Remove\ \ell\ from\ L2I$

$Remove\ i\ from\ I2L$

$Return\ True$

$Else$: $Return\ False$

3. RemoveNodeById(i)

Purpose: Remove a node from the index using its ID.

Input: $i \in I$

Preconditions: None.

Process Lookup:

$If\ i \in dom(I2L)$

$Let\ \ell = I2L(i)$

$Remove\ i\ from\ I2L$

$Remove\ \ell\ from\ L2I$

$Return\ True$

$Else$: $Return\ False$

4. GetNodeIdByLabel(ℓ)

Purpose: Retrieve the Id of a node by its label.

Input: $\ell \in L$

Preconditions: ℓ is not null, empty, or whitespace.

Process Lookup:

$If\ \ell \in dom(L2I)$

$Let\ (i, n) = L2I(\ell)$

$Return\ i$

$Else$: $Return\ null$

5. GetNodeByLabel(ℓ)

Purpose: Retrieve the `Node` object by its label

Input: $\ell \in L$

Preconditions: ℓ is not null, empty, or whitespace.

Process Lookup:

If $\ell \in dom(L2I)$

Let $(i, n) = L2I(\ell)$

$Return\ n$

$Else$: $Return\ null$

6. GetLabelByNodeId(i)

Purpose: Retrieve the label of a node by its ID.

Input: $i \in I$

Preconditions: None.

Process Lookup:

If $i \in dom(I2L)$

Return $I2L(i)$

Else: Return null

7. DisplayIndex()

Purpose: Output the state of the *NodeIndex* mappings for debugging or visualization.

Process Iteration:

$$For\ each\ \ell \in dom(L2I)$$

$Retrieve\ (i, n) = L2I(\ell)$

Output ℓ, i, n

Ensuring Data Consistency: The class maintains data consistency through:

- **Synchronized Updates**: Any addition or removal operation updates both *L2I* and *I2L* mappings simultaneously.
- **Cleanup of Conflicting Entries**: Before adding or updating, the class checks for existing entries with matching labels or IDs and resolves conflicts.

- **Case-Insensitive Label Handling**: Labels are stored and compared in a case-insensitive manner to prevent duplicates due to case differences.

Error Handling: Invalid Inputs: If a null, empty, or whitespace label is provided, methods return *null* or *false* as appropriate.

If a null *Node* or a *Node* with invalid properties is provided to *AddOrUpdateNode*, an exception is thrown.

Assertions: The class may use assertions (in an implementation context) to ensure internal state validity after operations.

Dependencies: Node Class

$n.Id \in I$ and $n.Label \in L$

It is assumed that each *Node* object provides access to its Id and $label$.