Node (H1)

Based on the codes from the book:
Artificial Intelligence : A Modern Approach
The copyrights of the codes belong to
Ravi Mohan, Peter Norvig, Stuart Russell, Ciaran O'Reilly

```
package aima.search.framework;

import java.util.List;

import aima.util.AbstractQueue;

/**
 * @author Ravi Mohan
 *
 */

/**
 * Artificial Intelligence A Modern Approach (2nd Edition): page 69.
 *
 * There are many ways to represent nodes, but we will assume that a node is a
 * data structure with five components:
 *
 * STATE: the state in the state space to which the node corresponds;
 * PARENT-NODE: the node in the search tree that generated this node; ACTION:
 * the action that was applied to the parent to generate the node; PATH-COST:
 * the cost, traditionally denoted by g(n), of the path from the initial state
 * to the node, as indicated by the parent pointers; and DEPTH: the number of
 * steps along the path from the initial state.
 */

public class Node {

	// STATE: the state in the state space to which the node corresponds;
	private Object state;

	// PARENT-NODE: the node in the search tree that generated this node;
	private Node parent;

	// ACTION: the action that was applied to the parent to generate the node;
	private String action;

	// PATH-COST: the cost, traditionally denoted by g(n), of the path from the
	// initial state to
	// the node, as indicated by the parent pointers;
	Double pathCost;

	// DEPTH: the number of steps along the path from the initial state.
	private int depth;

	private Double stepCost;

	public Node(Object state) {
		this.state = state;
		this.depth = 0;
		this.stepCost = new Double(0);
		this.pathCost = new Double(0);
	}

	public Node(Node parent, Object state) {
		this(state);
		this.parent = parent;
		this.depth = parent.getDepth() + 1;
	}

	public int getDepth() {
		return depth;
	}

	public boolean isRootNode() {
		return parent == null;
	}

	public Node getParent() {
		return parent;
	}

	public List<Node> getPathFromRoot() {
```

state
parent
action
pathCost;
depth;
stepCost;

getDepth();
isRootNode();
getParent();
getPathFromRoot();
getState();
setAction()
getAction();
setStepCost();
addToPathCost();
getPathCost();
getStepCost();
toString();

```
                Node current = this;
                AbstractQueue queue = new AbstractQueue();
                while (!(current.isRootNode())) {
                        queue.addToFront(current);
                        current = current.getParent();
                }
                queue.addToFront(current); // take care of root node
                return queue.asList();
        }

        public Object getState() {
                return state;
        }

        public void setAction(String action) {
                this.action = action;
        }

        public String getAction() {
                return action;
        }

        public void setStepCost(Double stepCost) {
                this.stepCost = stepCost;
        }

        public void addToPathCost(Double stepCost) {
                this.pathCost = new Double(parent.pathCost.doubleValue()
                                + stepCost.doubleValue());
        }

        /**
         * @return Returns the pathCost.
         */
        public double getPathCost() {
                return pathCost.doubleValue();
        }

        /**
         * @return Returns the stepCost.
         */
        public double getStepCost() {
                return stepCost.doubleValue();
        }

        @Override
        public String toString() {
                return getState().toString();
        }
}
```