Classes

AdjacencyMapGraph<V,E>

implements Graph<V,E>

Constructors

- `public` **AdjacencyMapGraph**`(boolean directed)`
  Constructs an empty graph. Directed if parameter is true.

NamedAdjacencyMapGraph<V,E>

extends AdjacencyMapGraph<V,E>

Constructors

- `public` **NamedAdjacencyMapGraph**`(boolean directed)`
  Constructs an empty graph. Directed if parameter is true.

Methods

- `public Vertex<V>` **getVertex**`(V name)`
  returns the Vertex object corresponding to the name in the parameter, or null if there is no vertex with that name.
- `public boolean` **vertexInGraph**`(V name)`
  returns a boolean indicating whether the graph contains a vertex with the name in the parameter.
- `public Edge<E>` **insertEdge**`(V uName, V vName, E element) throws IllegalArgumentException`
  inserts an edge whose vertices have the names uName and vName into the graph. Like the insertEdge method of AdjacencyMapGraph, it throws an IllegalArgumentException if there is already an edge (u,v) in the graph.
- `public Edge<E>` **getEdge**`(V uName, V vName) throws IllegalArgumentException`
  returns the edge whose endpoints are named by uName and vName, or null if the graph contains no such edge.

ArrayList<E>

implements List<E>

Constructors

- `public` **ArrayList<E>**`()`
  Constructs an empty list.

BST<K extends Comparable<K>, V>

Type and class declarations and selected methods from the BST class.

```
public class BST<K extends Comparable<K>, V> {
  // The public inner class for individual nodes.
  public class Node {
    protected Node left, right; // this Node's children
    protected Node parent;      // this Node's parent
    private K key;              // this Node's key
    private V value;            // the associated value

    /**
     * Constructor for a Node.
     *
     * @param key this node's key
     * @param value this node's value
     */
    public Node(K key, V value) {
      this.key = key;
      this.value = value;
      parent = left = right = sentinel;
    }

    /**
     * @return the key of this Node
     */
    public K getKey() { return key;}

    /**
     * @return the value in this Node
     */
    public V getValue() { return value; }

    /**
     * Set the value in this Node.
     */
    public void setValue(V newValue) { value = newValue; }

    /**
     * @return the String representation of this Node
     */
    public String toString() { return "key = " + key + ", value = " + value; }
  }

  // Instance variables for the BST<K,V> class. They are protected so that
  // subclasses can access them.
  protected Node root;      // root of this BST
  protected Node sentinel;  // how to indicate an absent node

  /**
   * Constructor for a BST. Makes an empty BST.
   */
  public BST() {
    sentinel = new Node(null, null);
    root = sentinel;
  }

  /**
   * Return a boolean indicating whether this BST is empty.
   */
  public boolean isEmpty() { return root == sentinel; }

  /**
   * Return a reference to the root, or null if this BST is empty.
   */
  public Node getRoot() {
    if (root == sentinel)
      return null;
    else
      return root;
  }

  /**
   * Return a string of 2*s spaces, for indenting.
   */
  private String indent(int s) {
    String result = "";
    for (int i = 0; i < s; i++)
      result += "  ";
    return result;
  }

  /**
   * Return a String representing the subtree rooted at a node.
   *
   * @param x the root of the subtree
   * @param depth the depth of x in the BST
   * @return the String representation of the subtree rooted at x
   */
  private String print(Node x, int depth) {
    if (x == sentinel)
      return "";
    else
      return print(x.right, depth + 1) + indent(depth) + x.toString() + "\n"
          + print(x.left, depth + 1);
  }

  /**
   * Return a String representation of this BST, indenting each level by two
   * spaces. Right subtrees appear before subtree roots, which appear before
   * left subtrees, so that when viewed sideways, we see the BST structure.
   */
  public String toString() {
    if (root == sentinel)
      return "";
    else
      return print(root, 0);
  }
```

HashMap<K, V>

implements Map<K, V>

Constructors

- `public` **HashMap<K, V>**`()`
  Constructs an empty map.

HashSet<E>

implements Set<E>

Constructors

- `public` **HashSet<E>**`()`
  Constructs an empty set.

LinkedList<E>

implements List<E>

Constructors

- `public` **LinkedList**`()`
  Constructs an empty list.

Methods

- `public void` **addFirst**`(E o)`
  Inserts the given element at the beginning of this list.
- `public void` **addLast**`(E o)`
- `public E` **getFirst**`()`
  Returns the first element in this list.
- `public E` **getLast**`()`
  Returns the last element in this list.
- `public E` **removeFirst**`()`
  Removes and returns the first element from this list.
- `public E` **removeLast**`()`
  Removes and returns the last element from this list.

String

Constructors

- `public` **String**`()`
- `public` **String**`(char[] arg)`
  Creates a new instance of the String class from the array arg.
- `public` **String**`(String str)`
  Creates an instance of the String class from the parameter str.

Methods

- `public char` **charAt**`(int index)`
  Returns the char value at the specified index. An index ranges from 0 to length() - 1. The first char value of the sequence is at index 0, the next at index 1, and so on, as for array indexing.
- `public int` **compareTo**`(String str)`
  Compares the current object to str. If both strings are equal, 0 (zero) is returned. If the current string is lexicographically less than the argument, an int less than zero is returned. If the current string is lexicographically greater than the argument, an int greater than zero is returned.
- `public boolean` **equals**`(Object arg)`
- `public boolean` **equalsIgnoreCase**`(String arg)`
  Returns true if the current object is equal to arg. arg must not be null, and must be of exact length and content as the current object. equalsIgnoreCase disregards the case of the characters.

- `public int` **indexOf**`(char c)`

- `public int` **indexOf**`(char c, int index)`
  Returns the index of the first occurrence of the character c in the current object not less than index (default of 0). Returns −1 if there is no such occurrence.
- `public int` **indexOf**`(String str)`

- `public int` **indexOf**`(String str, int index)`
  Returns the index of the first occurrence of the string str in the current object not less than index (default of 0). Returns −1 if there is no such occurrence.
- `public int` **length**`()`
  Returns the length of this string. The length is equal to the number of 16-bit Unicode characters in the string.
- `public String` **substring**`(int startindex) throws StringIndexOutOfBoundsException`

- `public String` **substring**`(int startindex, int lastindex) throws StringIndexOutOfBoundsException`
  Returns the substring of the current object starting with startindex and ending with lastindex-1 (or the last index of the string in the case of the first method).
- `public String` **toLowerCase**`()`
  Returns the current object with each character in lowercase, taking into account variations of the specified locale (loc).
- `public String` **toUpperCase**`()`
  Returns the current object with each character in uppercase, taking into account variations of the specified locale (loc).

TreeMap<K, V>

implements Map<K, V>

Constructors

- `public` **TreeMap<K, V>**`()`
  Constructs an empty map.

TreeSet<E>

implements Set<E>

Constructors

- `public` **TreeSet<E>**`()`
  Constructs an empty set.

Interfaces

Edge<E>

Methods

- `E` **getElement**`()`
  Returns the element associated with the edge.

Graph<V, E>

Methods

- `int` **numVertices**`()`
  Returns the number of vertices of the graph.
- `int` **numEdges**`()`
  Returns the number of edges of the graph.
- `Iterable<Vertex<V>>` **vertices**`()`
  Returns the vertices of the graph as an iterable collection.
- `Iterable<Edge<E>> edges()`
  Returns the edges of the graph as an iterable collection.
- `int` **outDegree**`(Vertex<V> v) throws IllegalArgumentException`
  Returns the number of edges leaving vertex v.
- `int` **inDegree**`(Vertex<V> v) throws IllegalArgumentException`
  Returns the number of edges for which vertex v is the destination.
- `Iterable<Edge<E>>` **outgoingEdges**`(Vertex<V> v) throws IllegalArgumentException`
  Returns an iterable collection of edges for which vertex v is the origin.
- `Iterable<Edge<E>>` **incomingEdges**`(Vertex<V> v) throws IllegalArgumentException`
  Returns an iterable collection of edges for which vertex v is the destination.
- `Edge<E>` **getEdge**`(Vertex<V> u, Vertex<V> v) throws IllegalArgumentException`
  Returns the edge from u to v, or null if they are not adjacent.
- `Vertex<V>[]` **endVertices**`(Edge<E> e) throws IllegalArgumentException`
  Returns the vertices of edge e as an array of length two.
  If the graph is directed, the first vertex is the origin, and the second is the destination.
- `Vertex<V>` **opposite**`(Vertex<V> v, Edge<E> e) throws IllegalArgumentException`
  Returns the vertex that is opposite vertex v on edge e.
- `Vertex<V>` **insertVertex**`(V element)`
  Inserts and returns a new vertex with the given element.
- `Edge<E>` **insertEdge**`(Vertex<V> u, Vertex<V> v, E element) throws IllegalArgumentException`
  Inserts and returns a new edge between vertices u and v, storing given element.
- `void` **removeVertex**`(Vertex<V> v) throws IllegalArgumentException`
  Removes a vertex and all its incident edges from the graph.
- `void` **removeEdge**`(Edge<E> e) throws IllegalArgumentException`
  Removes an edge from the graph.

Iterator<E>

Methods

- `public boolean` **hasNext**`()`
  Returns true if the iteration has more elements.
- `public E` **next**`()`
  Returns the next element in the iteration.
- `public void` **remove**`()`
  Removes from the underlying collection the last element returned by the iterator (optional operation).

List<E>

Methods

- `public void` **add**`(int index, E element)`
  Inserts the specified element at the specified position in this list (optional operation).
- `public boolean` **add**`(E o)`
  Appends the specified element to the end of this list (optional operation).
- `public void` **clear**`()`
  Removes all of the elements from this list (optional operation).
- `public boolean` **contains**`(Object o)`
  Returns true if this list contains the specified element.
- `public E` **get**`(int index)`
  Returns the element at the specified position in this list.
- `public int` **indexOf**`(Object o)`
  Returns the index in this list of the first occurrence of the specified element, or -1 if this list does not contain this element.
- `public boolean` **isEmpty**`()`
  Returns true if this list contains no elements.
- `public Iterator<E>` **iterator**`()`
  Returns an iterator over the elements in this list in proper sequence.
- `public int` **lastIndexOf**`(Object o)`
  Returns the index in this list of the last occurrence of the specified element, or -1 if this list does not contain this element.
- `public ListIterator<E>` **listIterator**`()`
  Returns a list iterator of the elements in this list (in proper sequence).
- `public E` **remove**`(int index)`
  Removes the element at the specified position in this list (optional operation).
- `public boolean` **remove**`(Object o)`
  Removes the first occurrence in this list of the specified element (optional operation).
- `public E` **set**`(int index, E element)`
  Replaces the element at the specified position in this list with the specified element (optional operation).
- `public int` **size**`()`
  Returns the number of elements in this list.

ListIterator<E>

Methods

- `public void` **add**`(Object o)`
  Inserts the specified element into the list (optional operation).
- `public boolean` **hasNext**`()`
  Returns true if this list iterator has more elements when traversing the list in the forward direction.
- `public boolean` **hasPrevious**`()`
  Returns true if this list iterator has more elements when traversing the list in the reverse direction.
- `public E` **next**`()`
  Returns the next element in the list.
- `public int` **nextIndex**`()`
  Returns the index of the element that would be returned by a subsequent call to next.
- `public E` **previous**`()`
  Returns the previous element in the list.
- `public int` **previousIndex**`()`
  Returns the index of the element that would be returned by a subsequent call to previous.
- `public void` **remove**`()`
  Removes from the list the last element that was returned by next or previous (optional operation).
- `public void` **set**`(E o)`
  Replaces the last element returned by next or previous with the specified element (optional operation).

Map<K, V>

Methods

- `public void` **clear**`()`
  Removes all mappings from this map (optional operation).
- `public boolean` **containsKey**`(Object key)`
  Returns true if this map contains a mapping for the specified key.
- `public boolean` **containsValue**`(Object value)`
  Returns true if this map maps one or more keys to the specified value.
- `public Set` **entrySet**`()`
  Returns a set view of the mappings contained in this map.
- `public boolean` **equals**`(Object o)`
  Compares the specified object with this map for equality.
- `public V` **get**`(Object key)`
  Returns the value to which this map maps the specified key.
- `public int` **hashCode**`()`
  Returns the hash code value for this map.
- `public boolean` **isEmpty**`()`
  Returns true if this map contains no key-value mappings.
- `public Set<K>` **keySet**`()`
  Returns a set view of the keys contained in this map.

- `public V` **put**`(K key, V value)`
  Associates the specified value with the specified key in this map (optional operation).
- `public V` **remove**`(Object key)`
  Removes the mapping for this key from this map if it is present (optional operation).
- `public int` **size**`()`
  Returns the number of key-value mappings in this map.

Set<E>

Methods

- `public boolean` **add**`(Object o)`
  Adds the specified element to this set if it is not already present (optional operation).
- `public void` **clear**`()`
  Removes all of the elements from this set (optional operation).
- `public boolean` **contains**`(Object o)`
  Returns true if this set contains the specified element.
- `public boolean` **equals**`(Object o)`
  Compares the specified object with this set for equality.
- `public boolean` **isEmpty**`()`
  Returns true if this set contains no elements.
- `public Iterator<E>` **iterator**`()`
  Returns an iterator over the elements in this set.
- `public boolean` **remove**`(Object o)`
  Removes the specified element from this set if it is present (optional operation).
- `public int` **size**`()`
  Returns the number of elements in this set (its cardinality).

Vertex<V>

Methods

- `V` **getElement**`()`
  Returns the element associated with the vertex.
