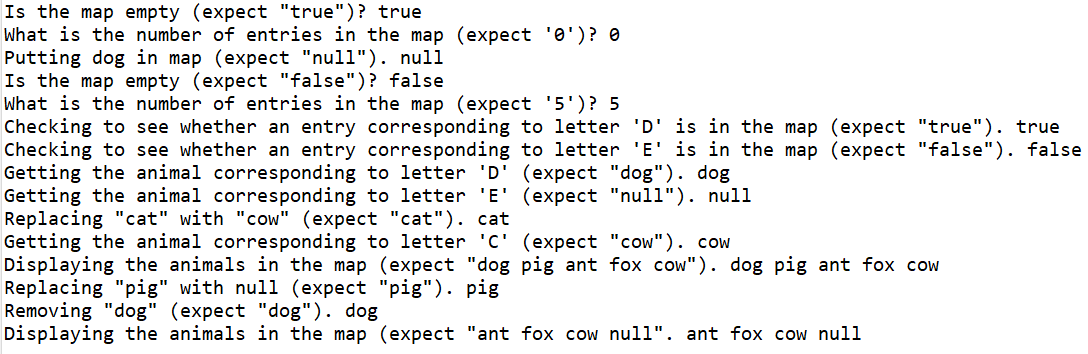
**1**



**package** Com.TSL.UtilitiesForUsingTheMapAdt;

/\*\*

\* **@author** EMILIA BUTU

\* version 1.0

\* since 2020-06

\*

\* Student name: Tom Lever

\* Completion date: 07/10/21

\*

\* Demonstrates the use of methods offered by MapInterface, with an implementation of the map using

\* array lists.

\*

\* The file contains basic operations related to Map ADT.

\*/

**public** **class** MapExample

{

/\*\*

\* main is the entry point of this program, which demonstrates the use of methods offered by MapInterface.

\* **@param** args

\*/

**public** **static** **void** main(String[] args)

{

//\*\*\* Task #1: declare a variable of MapInterface type, with type of elements of your choice

MapInterface<Character, String> the\_map\_of\_letters\_and\_animals;

//\*\*\* Task #2: instantiate the variable declared above using the ArrayListMap constructor

the\_map\_of\_letters\_and\_animals = **new** ArrayListMap<Character, String>();

//\*\*\* Task #3: check if the map is empty, and print out the answer you get

System.***out***.println("Is the map empty (expect \"true\")? " + the\_map\_of\_letters\_and\_animals.isEmpty());

//\*\*\* Task #4: use the appropriate method, and display the size of the map

System.***out***.println("What is the number of entries in the map (expect '0')? " + the\_map\_of\_letters\_and\_animals.size());

//\*\*\* Task #5: use the appropriate method to populate the map with 4-5 entries

System.***out***.println("Putting dog in map (expect \"null\"). " + the\_map\_of\_letters\_and\_animals.put('C', "cat"));

the\_map\_of\_letters\_and\_animals.put('D', "dog");

the\_map\_of\_letters\_and\_animals.put('P', "pig");

the\_map\_of\_letters\_and\_animals.put('A', "ant");

the\_map\_of\_letters\_and\_animals.put('F', "fox");

//\*\*\* Task #6: check if the map is empty, and print out the answer you get

System.***out***.println("Is the map empty (expect \"false\")? " + the\_map\_of\_letters\_and\_animals.isEmpty());

//\*\*\* Task #7: use the appropriate method, and display the size of the map

System.***out***.println("What is the number of entries in the map (expect '5')? " + the\_map\_of\_letters\_and\_animals.size());

//\*\*\* Task #8: check if certain values belong to the map, and replace some of them

System.***out***.println("Checking to see whether an entry corresponding to letter 'D' is in the map (expect \"true\"). " + the\_map\_of\_letters\_and\_animals.contains('D'));

System.***out***.println("Checking to see whether an entry corresponding to letter 'E' is in the map (expect \"false\"). " + the\_map\_of\_letters\_and\_animals.contains('E'));

System.***out***.println("Getting the animal corresponding to letter 'D' (expect \"dog\"). " + the\_map\_of\_letters\_and\_animals.get('D'));

System.***out***.println("Getting the animal corresponding to letter 'E' (expect \"null\"). " + the\_map\_of\_letters\_and\_animals.get('E'));

System.***out***.println("Replacing \"cat\" with \"cow\" (expect \"cat\"). " + the\_map\_of\_letters\_and\_animals.put('C', "cow"));

System.***out***.println("Getting the animal corresponding to letter 'C' (expect \"cow\"). " + the\_map\_of\_letters\_and\_animals.get('C'));

//\*\*\* Task #9: display the content of the map

System.***out***.print("Displaying the animals in the map (expect \"dog pig ant fox cow\").");

**for** (MapEntry<Character, String> the\_map\_entry : the\_map\_of\_letters\_and\_animals) {

System.***out***.print(" " + the\_map\_entry.getValue());

}

System.***out***.println();

//\*\*\* Task #10: remove a number of elements from the map

System.***out***.println("Replacing \"pig\" with null (expect \"pig\"). " + the\_map\_of\_letters\_and\_animals.put('P', **null**));

System.***out***.println("Removing \"dog\" (expect \"dog\"). " + the\_map\_of\_letters\_and\_animals.remove('D'));

System.***out***.print("Displaying the animals in the map (expect \"ant fox cow null\".");

**for** (MapEntry<Character, String> the\_map\_entry : the\_map\_of\_letters\_and\_animals) {

System.***out***.print(" " + the\_map\_entry.getValue());

}

System.***out***.println();

}

}

package Com.TSL.UtilitiesForUsingTheMapAdt;

//---------------------------------------------------------------------------

//MapInterface.java by Dale/Joyce/Weems Chapter 8

//

//A map provides (K = key, V = value) pairs, mapping the key onto the value.

//Keys are unique. Keys cannot be null.

//

//Methods throw IllegalArgumentException if passed a null key argument.

//

//Values can be null, so a null value returned by put, get, or remove does

//not necessarily mean that an entry did not exist.

//---------------------------------------------------------------------------

import java.util.Iterator;

public interface MapInterface<K, V> extends Iterable<MapEntry<K,V>>

{

V put(K k, V v);

// If an entry in this map with key k already exists then the value

// associated with that entry is replaced by value v and the original

// value is returned; otherwise, adds the (k, v) pair to the map and

// returns null.

V get(K k);

// If an entry in this map with a key k exists then the value associated

// with that entry is returned; otherwise null is returned.

V remove(K k);

// If an entry in this map with key k exists then the entry is removed

// from the map and the value associated with that entry is returned;

// otherwise null is returned.

//

// Optional. Throws UnsupportedOperationException if not supported.

boolean contains(K k);

// Returns true if an entry in this map with key k exists;

// Returns false otherwise.

boolean isEmpty();

// Returns true if this map is empty; otherwise, returns false.

boolean isFull();

// Returns true if this map is full; otherwise, returns false.

int size();

// Returns the number of entries in this map.

}

**package** Com.TSL.UtilitiesForUsingTheMapAdt;

//---------------------------------------------------------------------------

//MapEntry.java by Dale/Joyce/Weems Chapter 8

//

//Provides key, value pairs for use with a Map.

//Keys are immutable.

//---------------------------------------------------------------------------

**public** **class** MapEntry<K, V>

{

**protected** K key;

**protected** V value;

/\*\*

\* MapEntry(K k, V v) is the two-parameter constructor for MapEntry, which sets the key and value of this map entry

\* to provided keys and values.

\* **@param** k

\* **@param** v

\*/

MapEntry(K k, V v)

{

key = k; value = v;

}

/\*\*

\* getKey provides this map entry's key.

\* **@return**

\*/

**public** K getKey() {**return** key;}

/\*\*

\* getValue provides this map entry's value.

\*/

**public** V getValue(){**return** value;}

/\*\*

\* setValue sets this map entry's value.

\* **@param** v

\*/

**public** **void** setValue(V v){value = v;}

@Override

**public** String toString()

// Returns a string representing this MapEntry.

{

**return** "Key : " + key + "\nValue: " + value;

}

}

package Com.TSL.UtilitiesForUsingTheMapAdt;

//---------------------------------------------------------------------------

//ArrayListMap.java by Dale/Joyce/Weems Chapter 8

//

//Implements a map using an ArrayList.

//

//A map provides (K = key, V = value) pairs, mapping the key onto the value.

//Keys are unique. Keys cannot be null.

//

//Methods throw IllegalArgumentException if passed a null key argument.

//

//Values can be null, so a null value returned by put, get, or remove does

//not necessarily mean that an entry did not exist.

//---------------------------------------------------------------------------

import java.util.\*; // Iterator, ArrayList

public class ArrayListMap<K, V> implements MapInterface<K,V>

{

protected ArrayList<MapEntry<K, V>> map;

/\*\*

\* ArrayListMap() is a zero-parameter constructor for ArrayListMap, which sets the ArrayList of this map to a new

\* ArrayList with default capacity.

\*/

public ArrayListMap()

{

map = new ArrayList<MapEntry<K, V>>();

}

/\*\*

\* ArrayListMap(int initCapacity) is a one-parameter constructor for ArrayListMap, which sets the ArrayList of this

\* map to a new ArrayList with capacity initCapacity.

\* @param initCapacity

\*/

public ArrayListMap(int initCapacity)

{

map = new ArrayList<MapEntry<K, V>>(initCapacity);

}

public V put(K k, V v)

// If an entry in this map with key k already exists then the value

// associated with that entry is replaced by value v and the original

// value is returned; otherwise, adds the (k, v) pair to the map and

// returns null.

{

if (k == null)

throw new IllegalArgumentException("Maps do not allow null keys.");

MapEntry<K, V> entry = new MapEntry<K, V>(k, v);

MapEntry<K,V> temp;

Iterator<MapEntry<K,V>> search = map.iterator(); // Arraylist iterator

while (search.hasNext())

{

temp = search.next();

if (temp.getKey().equals(k))

{

search.remove();

map.add(entry);

return temp.getValue(); // k found, exits method

}

}

// No entry is associated with k.

map.add(entry);

return null;

}

public V get(K k)

// If an entry in this map with a key k exists then the value associated

// with that entry is returned; otherwise null is returned.

{

if (k == null)

throw new IllegalArgumentException("Maps do not allow null keys.");

for (MapEntry<K,V> temp: map) // uses ArrayList iterator

if (temp.getKey().equals(k))

return temp.getValue(); // k found, exits method

// No entry is associated with k.

return null;

}

public V remove(K k)

// If an entry in this map with key k exists then the entry is removed

// from the map and the value associated with that entry is returned;

// otherwise null is returned.

{

if (k == null)

throw new IllegalArgumentException("Maps do not allow null keys.");

MapEntry<K,V> temp;

Iterator<MapEntry<K,V>> search = map.iterator(); // Arraylist iterator

while (search.hasNext())

{

temp = search.next();

if (temp.getKey().equals(k))

{

search.remove();

return temp.getValue(); // k found, exits method

}

}

// No entry is associated with k.

return null;

}

public boolean contains(K k)

// Returns true if an entry in this map with key k exists;

// Returns false otherwise.

{

if (k == null)

throw new IllegalArgumentException("Maps do not allow null keys.");

for (MapEntry<K,V> temp: map)

if (temp.getKey().equals(k))

return true; // k found, exits method

// No entry is associated with k.

return false;

}

public boolean isEmpty()

// Returns true if this map is empty; otherwise, returns false.

{

return (map.size() == 0); // uses ArrayList size

}

public boolean isFull()

// Returns true if this map is full; otherwise, returns false.

{

return false; // An ArrayListMap is never full

}

public int size()

// Returns the number of entries in this map.

{

return map.size(); // uses ArrayList size

}

public Iterator<MapEntry<K,V>> iterator()

// Returns the Iterator provided by ArrayList.

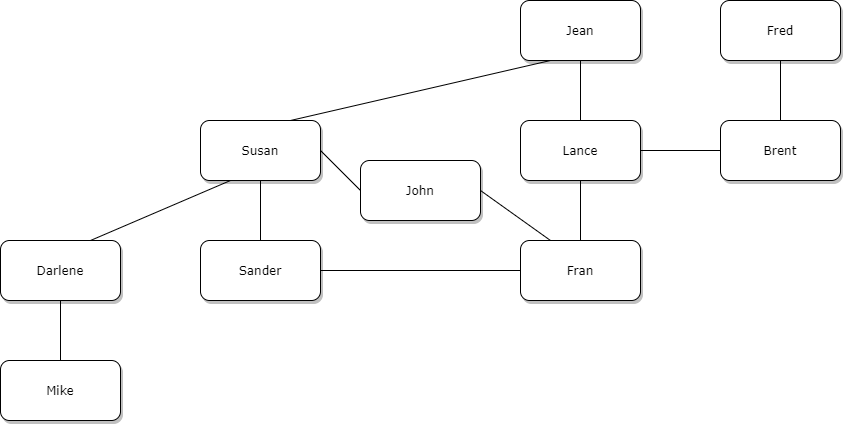
{

return map.iterator(); // returns ArrayList iterator

}

}

**2**



It seems like Mike could work for Darlene, and Darlene would be the supervisor of, senior to, Mike. Same with Fred and Brent, Darlene and Susan, and Brent and Lance. It seems like John could be a President, Susan and Fran Vice Presidents, Sander an Assistant VP, Darlene a Project Manager, Lance a Senior Project Managers with Jean as a COO.