Course Project – Hospital Information Management

Data Structures and Algorithms I

[CSIS-3103-002]

Team 3

Thomas Sadowski (Z00300233)

William Gray (Z00303491)

Keith Lopez (Z00285342)

Shaifur Rahmans (Z00249132)

Table of Contents

CONTENTS PAGE

Discussion Log

Discussion #1

Date: March 2, 2018

Time: Throughout the course

Method: Face-to-face / Blackboard / Google Hangout / Skype / Conference call / Online meeting / **Email** / Other (Please specify) _____

(Please circle the method you used)

Team member Signature:

1. _Thomas Sadowski_
2. _William Gray_____
3. _Keith Lopez_____
4. _Shaifur Rahmans__

Discussion #2

Date: April 12, 2018

Time: 2:30pm

Method: **Face-to-face** / Blackboard / Google Hangout / Skype / Conference call / Online meeting / Email / Other (Please specify) _____

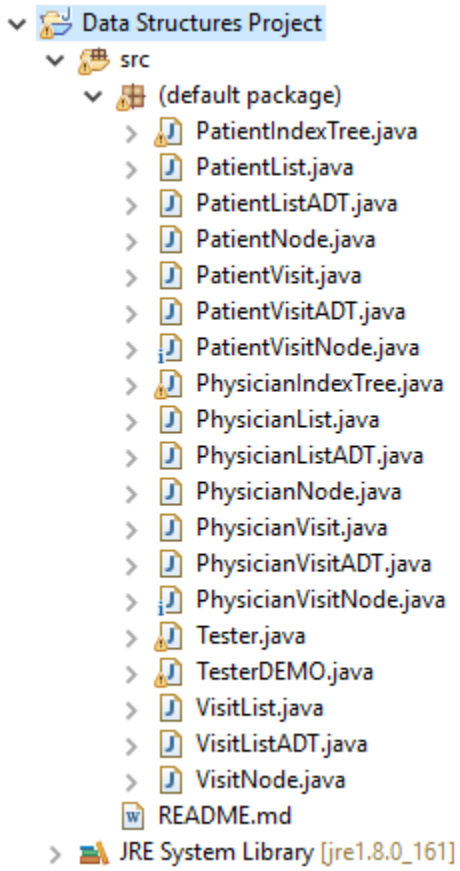
(Please circle the method you used)

Team member Signature:

1. _Thomas Sadowski_
2. _William Gray_____
3. _Keith Lopez_____
4. _Shaifur Rahmans__

Source code

Below is a screenshot of all the classes used in this project.



The next several pages consist of source code for each class.

PatientIndexTree.java

```
 /**
 * PatientIndexTree Class
 * Creates the Nodes for the Parent Tree and PatientIndexTree  and contains methods
related to node management that are self explanitory.
 * @author Thomas Sadowski, William Gray, Keith Lopez, Shaifur Rahmans
 * @version 1
 */

public class PatientIndexTree<E extends Comparable<E>>
{
    protected PatientTreeNode<E> root;

    public PatientIndexTree()
    {
    }

    public PatientIndexTree(String name, String info, String medicalCondition,
                            PatientIndexTree leftTree, PatientIndexTree rightTree)
    {
        root = new PatientTreeNode(name, info, medicalCondition);
        if (leftTree != null)
            root.left = leftTree.root;
        if (rightTree != null)
            root.right = rightTree.root;
    }

    protected PatientIndexTree(PatientTreeNode<E> root)
    {
        this.root = root;
    }

    public PatientIndexTree<E> getLeftSubtree()
    {
        if (root!=null && root.left!=null)
            return new PatientIndexTree<E>(root.left);
        else
            return null;
    }

    public PatientIndexTree<E> getRightSubtree()
    {
        if (root!=null && root.right!=null)
            return new PatientIndexTree<E>(root.right);
        else
            return null;
    }

    public String getData()
    {
        if (root!=null)
            return root.name;
        else
            return null;
```

```
}

public boolean isEmpty()
{
    return root == null;
}

public boolean isLeaf()
{
    return root != null && root.left == null && root.right == null;
}

protected void preOrderTraversal(ProcessData<E> visitOperation)
{
    preOrderTraversal(root, visitOperation);
}

private void preOrderTraversal(PatientTreeNode node, ProcessData visitOperation)
{
    if (node == null)
        return;
    visitOperation.process(node.name, node.info, node.medicalCondition);
    preOrderTraversal(node.left, visitOperation);
    preOrderTraversal(node.right, visitOperation);
}

protected void postOrderTraversal(ProcessData<E> visitOperation)
{
    postOrderTraversal(root, visitOperation);
}

private void postOrderTraversal(PatientTreeNode node, ProcessData visitOperation)
{
    if (node == null)
        return;
    postOrderTraversal(node.left, visitOperation);
    postOrderTraversal(node.right, visitOperation);
    visitOperation.process(node.name, node.info, node.medicalCondition);
}

protected void inOrderTraversal(ProcessData visitOperation)
{
    inOrderTraversal(root, visitOperation);
}

private void inOrderTraversal(PatientTreeNode node, ProcessData visitOperation)
{
    if (node == null)
        return;
    if (node.left != null && visitOperation instanceof PrePostProcess)
        ((PrePostProcess<E>)visitOperation).pre();
    inOrderTraversal(node.left, visitOperation);
    visitOperation.process(node.name, node.info, node.medicalCondition);
    inOrderTraversal(node.right, visitOperation);
```

```
        if (node.right != null && visitOperation instanceof PrePostProcess)
            ((PrePostProcess<E>)visitOperation).post();
    }

    public String find(String name)
    {
        return find(root, name);
    }

    public String find(PatientTreeNode subtreeRoot, String name)
    {
        if (subtreeRoot == null)
            return null;
        if (name.compareTo(subtreeRoot.name) == 0)
            return subtreeRoot.name;
        else if (name.compareTo(subtreeRoot.name)<0)
            return find(subtreeRoot.left, name);
        else
            return find(subtreeRoot.right, name);
    }

    public boolean contains(String name)
    {
        return contains(root, name);
    }

    public boolean add(String name, String info, String medicalCondition)
    {
        if (root == null)
        {
            root = new PatientTreeNode(name, info, medicalCondition);
            return true;
        }
        return add(root, name, info, medicalCondition);
    }

    public void printOrderedData()
    {
        inOrderTraversal(new ProcessData<String>()
        {
            @Override
            public void process(String name, String info, String medicalCondition)
            {
                System.out.print(name + " | " + info + " | " + medicalCondition + "
\n");
            }
        }
        );
    }

    private boolean contains(PatientTreeNode<E> subtreeRoot, String name)
    {
        if (subtreeRoot == null)
            return false;
        if (name.compareTo(subtreeRoot.name)==0)
```

```
            return true;
        else if (name.compareTo(subtreeRoot.name)<0)
            return contains(subtreeRoot.left, name);
        else
            return contains(subtreeRoot.right, name);
    }

    private boolean add(PatientTreeNode<E> subtreeRoot, String name, String info,
            String medicalCondition)
    {
        if (name.compareTo(subtreeRoot.name) == 0)
            return false;
        else if (name.compareTo(subtreeRoot.name)<0)
        {
            if (subtreeRoot.left == null)
            {
                subtreeRoot.left = new PatientTreeNode<E>(name, info,
medicalCondition);
                return true;
            }
            return add(subtreeRoot.left, name, info, medicalCondition);
        }
        else
        {
            if (subtreeRoot.right == null)
            {
                subtreeRoot.right = new PatientTreeNode<E>(name, info,
medicalCondition);
                return true;
            }
            return add(subtreeRoot.right, name, info, medicalCondition);
        }
    }

    protected interface ProcessData<String>
    {
        void process(String name, String info, String medicalCondition);
    }

    protected interface PrePostProcess<E> extends ProcessData<E>
    {
        void pre();
        void post();
    }

    /**
     * PatientTreeNode<E> Internnal Class of PatientIndexTree Class
     * Creates the Nodes for the PaitentTree
     * @author Thomas Sadowski, William Gray, Keith Lopez, Shaifur Rahmans
     * @version 1
     */

    protected static class PatientTreeNode<E>
    {
        protected String name;
```

```
        protected String info;
        protected String medicalCondition;
        protected PatientTreeNode<E> left;
        protected PatientTreeNode<E> right;

        public PatientTreeNode(String name, String info, String medicalCondition)
        {
            this.name = name;
            this.info = info;
            this.medicalCondition = medicalCondition;
        }

        @Override
        public String toString()
        {
            return name + " | " + info + " | " + medicalCondition;
        }
    }
}
```

PatientList.java

```
/**
 * PatientList Class
 * Creates the PatientList Node , contains methods related to node management that
are self explanitory.
 * @author Thomas Sadowski, William Gray, Keith Lopez, Shaifur Rahmans
 * @version 1
 */

import java.util.NoSuchElementException;

public class PatientList implements PatientListADT
{
    private PatientNode head;
    private int size;

    public PatientList()
    {
        head = null;
        size = 0;
    }

    @Override
    public void addToFront(String name, String info, String medicalCondition)
    {
        PatientNode newNode = new PatientNode(name, info, medicalCondition);
        newNode.setNext(head);
        head = newNode;
        size++;
    }

    @Override
    public void addToEnd(String name, String info, String medicalCondition)
    {
        if (head == null)
        {
            addToFront(name, info, medicalCondition);
            return;
        }
        else if (head.getNext() == null)
        {
            PatientNode newNode = new PatientNode(name, info, medicalCondition);
            head.setNext(newNode);
        }
        else
            addToEnd(name, info, medicalCondition, head.getNext());
        size++;
    }

    private void addToEnd(String name, String info, String medicalCondition,
PatientNode node)
    {
        if (node.getNext() != null)
```

```
            addToEnd(name, info, medicalCondition, node.getNext());
        else
        {
            PatientNode newNode = new PatientNode(name, info, medicalCondition);
            node.setNext(newNode);
        }
    }

    @Override
    public String removeFirst()
    {
        if (head == null)
            throw new NoSuchElementException();
        String str = head.toString();
        head = head.getNext();
        size--;
        return str;
    }

    @Override
    public String removeLast()
    {
        if (head == null)
            throw new NoSuchElementException();
        String str = head.toString();
        if (head.getNext() == null)
        {
            head = null;
            size--;
            return str;
        }
        else return removeLast(head);
    }

    private String removeLast(PatientNode node)
    {
        PatientNode temp = node.getNext();
        if (temp.getNext() != null)
            removeLast(temp);
        else
        {
            node.setNext(null);
            size--;
            return temp.toString();
        }
        return "*";
    }

    @Override
    public boolean removeByQuery(String name, String info)
    {
        PatientNode query = new PatientNode(name, info, "filler");
        if (head == null)
            throw new NoSuchElementException();
        else if (head.equals(query))
```

```
        {
            head = head.getNext();
            size--;
            return true;
        }
        else return removeByQuery(query, head);
    }

    private boolean removeByQuery(PatientNode query, PatientNode traverse)
    {
        if (traverse.getNext() == null)
            return false;
        PatientNode temp = traverse.getNext();
        if (temp.equals(query))
        {
            traverse.setNext(temp.getNext());
            size--;
            return true;
        }
        else return removeByQuery(query, traverse.getNext());
    }

    @Override
    public void addVisit(String name, String info, String physician)
    {
        PatientNode query = new PatientNode(name, info, "filler");
        if (head == null)
            throw new NoSuchElementException();
        else if (head.equals(query))
            head.addVisit(name, info, physician);
        else addVisit(query, head, physician);
    }

    private void addVisit(PatientNode query, PatientNode traverse, String physician)
    {
        if (traverse.getNext() == null)
            throw new NoSuchElementException(); ;
        PatientNode temp = traverse.getNext();
        if (temp.equals(query))
            temp.addVisit(query.getName(), query.getInfo(), physician);
        else addVisit(query, traverse.getNext(), physician);
    }

    @Override
    public PatientVisit getVisits(String name, String info)
    {
        PatientNode query = new PatientNode(name, info, "filler");
        if (head == null)
            throw new NoSuchElementException();
        else if (head.equals(query))
            return head.getVisits();
        else return getVisits(query, head);
    }

    private PatientVisit getVisits(PatientNode query, PatientNode traverse)
```

```
{
    if (traverse.getNext() == null)
        throw new NoSuchElementException();
    PatientNode temp = traverse.getNext();
    if (temp.equals(query))
        return temp.getVisits();
    return getVisits(query, traverse.getNext());
}

@Override
public boolean contains(String name, String info)
{
    PatientNode query = new PatientNode(name, info, "filler");
    if (head == null)
        return false;
    else if (head.equals(query))
        return true;
    else return contains(query, head);
}

private boolean contains(PatientNode query, PatientNode traverse)
{
    if (traverse.getNext() == null)
        return false;
    traverse = traverse.getNext();
    if (traverse.equals(query))
        return true;
    return contains(query, traverse);
}

@Override
public String[] toArray()
{
    String[] patients = new String[size];
    if (head == null)
        throw new NoSuchElementException();
    toArray(patients, 0, head);
    return patients;
}

private void toArray(String[] patients, int index, PatientNode node)
{
    if (node.getNext() == null)
        patients[index] = node.toString();
    else
    {
        patients[index] = node.toString();
        index++;
        toArray(patients, index, node.getNext());
    }
}

public PatientNode[] toNodeArray()
{
    PatientNode[] patients = new PatientNode[size];
```

```
        if (head == null)
            throw new NoSuchElementException();
        toNodeArray(patients, 0, head);
        return patients;
    }

    private void toNodeArray(PatientNode[] patients, int index, PatientNode node)
    {
        if (node.getNext() == null)
            patients[index] = node;
        else
        {
            patients[index] = node;
            index++;
            toNodeArray(patients, index, node.getNext());
        }
    }
}
```

PatientListADT.java

```java
/**
 * PatientListADT
 * ADT for the PatientList Class
 * @author Thomas Sadowski, William Gray, Keith Lopez, Shaifur Rahmans
 * @version 1
 */

interface PatientListADT
{
    void addToFront(String name, String info, String medicalCondition);
    void addToEnd(String name, String info, String medicalCondition);
    String removeFirst();
    String removeLast();
    boolean removeByQuery(String name, String info);
    void addVisit(String name, String info, String physician);
    PatientVisit getVisits(String name, String info);
    boolean contains(String name, String info);
    String[] toArray();
}
```

PatientNode.java

```java
/**
 * PatientNode Class
 * Cretates the Patient Node for the Patients information and contains methods
related to node management that are self explanitory.
 * @author Thomas Sadowski, William Gray, Keith Lopez, Shaifur Rahmans
 * @version 1
 */

public class PatientNode
{
    private String name;
    private String info;
    private String medicalCondition;
    private PatientVisit visit;
    private PatientNode next;

    public PatientNode()
    {
        name = "J. Doe";
        info = "Unknown";
        medicalCondition = "Unknown";
        visit = new PatientVisit();
        next = null;
    }

    public PatientNode(String name, String info, String medicalCondition)
    {
        this.name = name;
        this.info = info;
        this.medicalCondition = medicalCondition;
        visit = new PatientVisit();
        next = null;
    }

    public void setName(String str)
    {
        name = str;
    }

    public void setInfo(String str)
    {
        info = str;
    }

    public void setNext(PatientNode node)
    {
        next = node;
    }

    public String getName()
    {
        return name;
    }
```

```
    public String getInfo()
    {
        return info;
    }

    public String getMedicalCondition()
    {
        return medicalCondition;
    }

    public PatientNode getNext()
    {
        return next;
    }

    public PatientVisit getVisits()
    {
        return visit;
    }

    public void addVisit(String patientName, String info, String physicianName)
    {
        visit.addToEnd(patientName, info, physicianName);
    }

    public boolean equals(PatientNode other)
    {
        return name.equals(other.getName()) && info.equals(other.getInfo());
    }

    public String toString()
    {
        return "Patient Name: " + name + " | Patient Info: " + info
                + " | Medical Condition: " + medicalCondition;
    }
}
```

PatientVisit.java

```
/**
 * PatientVisit Class
 * Creates Nodes for the patient visit and contains methods related to node
management that are self explanatory.
 * @author Thomas Sadowski, William Gray, Keith Lopez, Shaifur Rahmans
 * @version 1
 */

import java.util.NoSuchElementException;

public class PatientVisit implements PatientVisitADT
{
    private PatientVisitNode head;
    private int size;
    private VisitList list;

    public PatientVisit()
    {
        head = null;
        size = 0;
        list = new VisitList();
    }

    @Override
    public void addToFront(String patientName, String info, String physicianName)
    {
        list.addToFront(patientName, info, physicianName);
        VisitNode visitNode = new VisitNode(patientName, info, physicianName);
        PatientVisitNode newNode = new PatientVisitNode(visitNode);
        newNode.setNext(head);
        head = newNode;
        size++;
    }

    @Override
    public void addToEnd(String patientName, String info, String physicianName)
    {
        list.addToEnd(patientName, info, physicianName);
        VisitNode visitNode = new VisitNode(patientName, info, physicianName);
        if (head == null)
        {
            addToFront(patientName, info, physicianName);
            return;
        }
        else if (head.getNext() == null)
        {
            PatientVisitNode newNode = new PatientVisitNode(visitNode);
            head.setNext(newNode);
        }
        else
            addToEnd(visitNode, head.getNext());
        size++;
    }
```

```
private void addToEnd(VisitNode visitNode, PatientVisitNode node)
{
    if (node.getNext() != null)
        addToEnd(visitNode, node.getNext());
    else
    {
        PatientVisitNode newNode = new PatientVisitNode(visitNode);
        node.setNext(newNode);
    }
}

@Override
public String removeFirst()
{
    list.removeFirst();
    if (head == null)
        throw new NoSuchElementException();
    String str = head.toString();
    head = head.getNext();
    size--;
    return str;
}

@Override
public String removeLast()
{
    list.removeLast();
    if (head == null)
        throw new NoSuchElementException();
    String str = head.toString();
    if (head.getNext() == null)
    {
        head = null;
        size--;
        return str;
    }
    else
       return removeLast(head);
}

private String removeLast(PatientVisitNode node)
{
    PatientVisitNode temp = node.getNext();
    if (temp.getNext() != null)
        removeLast(temp);
    else
    {
        node.setNext(null);
        size--;
        return temp.toString();
    }
    return "*";
}
```

```
@Override
public boolean removeByQuery(VisitNode visitNode)
{
    list.removeByQuery(visitNode);
    PatientVisitNode query = new PatientVisitNode(visitNode);
    if (head == null)
        throw new NoSuchElementException();
    else if (head.equals(query))
    {
        head = head.getNext();
        size--;
        return true;
    }
    else return removeByQuery(query, head);
}

private boolean removeByQuery(PatientVisitNode query, PatientVisitNode traverse)
{
    if (traverse.getNext() == null)
        return false;
    PatientVisitNode temp = traverse.getNext();
    if (temp.equals(query))
    {
        traverse.setNext(temp.getNext());
        size--;
        return true;
    }
    return removeByQuery(query, traverse.getNext());
}

@Override
public boolean contains(VisitNode visitNode)
{
    PatientVisitNode query = new PatientVisitNode(visitNode);
    if (head == null)
        return false;
    else if (head.equals(query))
        return true;
    return contains(query, head);
}

private boolean contains(PatientVisitNode query, PatientVisitNode traverse)
{
    if (traverse.getNext() == null)
        return false;
    PatientVisitNode temp = traverse.getNext();
    if (temp.equals(query))
        return true;
    return contains(query, traverse.getNext());
}

@Override
public String[] toArray()
{
    String[] patients = new String[size];
```

```
        if (head == null)
            throw new NoSuchElementException();
        toArray(patients, 0, head);
        return patients;
    }

    private void toArray(String[] patients, int index, PatientVisitNode node)
    {
        if (node.getNext() == null)
            patients[index] = node.toString();
        else
        {
            patients[index] = node.toString();
            index++;
            toArray(patients, index, node.getNext());
        }
    }

    public VisitList getList()
    {
        return list;
    }

    public int getSize()
    {
        return size;
    }
}
```

PatientVisitADT.java

```
/**
 * PatientVisitADT
 * ADT for the PatientVisit Class
 * @author Thomas Sadowski, William Gray, Keith Lopez, Shaifur Rahmans
 * @version 1
 */

public interface PatientVisitADT
{
    void addToFront(String patientName, String medicalCondition, String
physicianName);
    void addToEnd(String patientName, String medicalCondition, String physicianName);
    String removeFirst();
    String removeLast();
    boolean removeByQuery(VisitNode query);
    boolean contains(VisitNode query);
    String[] toArray();
}
```

PatientVisitNode.java

```
/**
 * PatientVisitNode Class
 * Creates the  VisitNode for the Patients and contains methods related to node
management that are self explanatory.
 * @author Thomas Sadowski, William Gray, Keith Lopez, Shaifur Rahmans
 * @version 1
 */

public class PatientVisitNode
{
    VisitNode visitNode;
    private PatientVisitNode next;

    public PatientVisitNode()
    {
        visitNode = new VisitNode();
        next = null;
    }

    public PatientVisitNode(VisitNode visitNode)
    {
        this.visitNode = visitNode;
        next = null;
    }

    public void setVisitNode(VisitNode visitNode)
    {
        this.visitNode = visitNode;
    }

    public void setNext(PatientVisitNode node)
    {
        next = node;
    }

    public VisitNode getVisitNode()
    {
        return visitNode;
    }

    public PatientVisitNode getNext()
    {
        return next;
    }

    public boolean equals(PatientVisitNode other)
    {
        if (visitNode.equals(other))
            return true;
        else return false;
    }

    public String toString()
```

```
    {
        return visitNode.toString();
    }
}
```

PhysicianIndexTree.java

```
/**
 * PhysicianIndexTree<E> Class
 * Creates the PhysicianIndexTree and its nodes contains methods related to node
management that are self explanatory.
 * @author Thomas Sadowski, William Gray, Keith Lopez, Shaifur Rahmans
 * @version 1
 */

public class PhysicianIndexTree<E extends Comparable<E>>
{
    protected PhysicianTreeNode<E> root;

    public PhysicianIndexTree()
    {
    }

    public PhysicianIndexTree(String name, String specialty, PhysicianIndexTree
leftTree,
                                          PhysicianIndexTree rightTree)
    {
        root = new PhysicianTreeNode(name, specialty);
        if (leftTree != null)
            root.left = leftTree.root;
        if (rightTree != null)
            root.right = rightTree.root;
    }

    protected PhysicianIndexTree(PhysicianTreeNode<E> root)
    {
        this.root = root;
    }

    public PhysicianIndexTree<E> getLeftSubtree()
    {
        if (root!=null && root.left!=null)
            return new PhysicianIndexTree<E>(root.left);
        else
            return null;
    }

    public PhysicianIndexTree<E> getRightSubtree()
    {
        if (root!=null && root.right!=null)
            return new PhysicianIndexTree<E>(root.right);
        else
            return null;
    }

    public String getData()
    {
        if (root!=null)
            return root.name;
        else
```

```
            return null;
    }

    public boolean isEmpty()
    {
        return root == null;
    }

    public boolean isLeaf()
    {
        return root != null && root.left == null && root.right == null;
    }

    protected void preOrderTraversal(ProcessData<E> visitOperation)
    {
        preOrderTraversal(root, visitOperation);
    }

    private void preOrderTraversal(PhysicianTreeNode node, ProcessData
visitOperation)
    {
        if (node == null)
            return;
        visitOperation.process(node.name, node.specialty);
        preOrderTraversal(node.left, visitOperation);
        preOrderTraversal(node.right, visitOperation);
    }

    protected void postOrderTraversal(ProcessData<E> visitOperation)
    {
        postOrderTraversal(root, visitOperation);
    }

    private void postOrderTraversal(PhysicianTreeNode node, ProcessData
visitOperation)
    {
        if (node == null)
            return;
        postOrderTraversal(node.left, visitOperation);
        postOrderTraversal(node.right, visitOperation);
        visitOperation.process(node.name, node.specialty);
    }

    protected void inOrderTraversal(ProcessData visitOperation)
    {
        inOrderTraversal(root, visitOperation);
    }

    private void inOrderTraversal(PhysicianTreeNode node, ProcessData visitOperation)
    {
        if (node == null)
            return;
        if (node.left != null && visitOperation instanceof PrePostProcess)
            ((PrePostProcess<E>)visitOperation).pre();
        inOrderTraversal(node.left, visitOperation);
```

```
        visitOperation.process(node.name, node.specialty);
        inOrderTraversal(node.right, visitOperation);

        if (node.right != null && visitOperation instanceof PrePostProcess)
            ((PrePostProcess<E>)visitOperation).post();
    }

    public String find(String name)
    {
        return find(root, name);
    }

    public String find(PhysicianTreeNode subtreeRoot, String name)
    {
        if (subtreeRoot == null)
            return null;
        if (name.compareTo(subtreeRoot.name) == 0)
            return subtreeRoot.name;
        else if (name.compareTo(subtreeRoot.name)<0)
            return find(subtreeRoot.left, name);
        else
            return find(subtreeRoot.right, name);
    }

    public boolean contains(String name)
    {
        return contains(root, name);
    }

    public boolean add(String name, String specialty)
    {
        if (root == null)
        {
            root = new PhysicianTreeNode(name, specialty);
            return true;
        }
        return add(root, name, specialty);
    }

    public void printOrderedData()
    {
        inOrderTraversal(new ProcessData<String>()
        {
            @Override
            public void process(String name, String specialty)
            {
                System.out.print(name + " | " + specialty + " \n");
            }
        }
        );
    }

    private boolean contains(PhysicianTreeNode<E> subtreeRoot, String name)
    {
        if (subtreeRoot == null)
```

```
            return false;
        if (name.compareTo(subtreeRoot.name)==0)
            return true;
        else if (name.compareTo(subtreeRoot.name)<0)
            return contains(subtreeRoot.left, name);
        else
            return contains(subtreeRoot.right, name);
    }

    private boolean add(PhysicianTreeNode<E> subtreeRoot, String name, String
specialty)
    {
        if (name.compareTo(subtreeRoot.name) == 0)
            return false;
        else if (name.compareTo(subtreeRoot.name)<0)
        {
            if (subtreeRoot.left==null)
            {
                subtreeRoot.left = new PhysicianTreeNode<E>(name, specialty);
                return true;
            }
            return add(subtreeRoot.left, name, specialty);
        }
        else
        {
            if (subtreeRoot.right == null)
            {
                subtreeRoot.right = new PhysicianTreeNode<E>(name, specialty);
                return true;
            }
            return add(subtreeRoot.right, name, specialty);
        }
    }

    /**
    * ProcessData<E> Internal Interface of PhysicianIndexTree Class
    * Creates the Nodes for the PhysicianTree
    * @author Thomas Sadowski, William Gray, Keith Lopez, Shaifur Rahmans
    * @version 1
    */
    protected interface ProcessData<String>
    {
        void process(String name, String specialty);
    }

    /**
    * PrePostProcess<E> Internal Interface of PhysicianIndexTree Class
    * Creates the Nodes for the PhysicianTree
    * @author Thomas Sadowski, William Gray, Keith Lopez, Shaifur Rahmans
    * @version 1
    */
    protected interface PrePostProcess<E> extends ProcessData<E>
    {
        void pre();
        void post();
```

```
    }

    /**
    * PhysicianTreeNode<E> Internal Class of PhysicianIndexTree Class
    * Creates the Nodes for the PhysicianTree
    * @author Thomas Sadowski, William Gray, Keith Lopez, Shaifur Rahmans
    * @version 1
    */
    protected static class PhysicianTreeNode<E>
    {
        protected String name;
        protected String specialty;
        protected PhysicianTreeNode<E> left;
        protected PhysicianTreeNode<E> right;

        public PhysicianTreeNode(String name, String specialty)
        {
            this.name = name;
            this.specialty = specialty;
        }

        @Override
        public String toString()
        {
            return name + " | " + specialty;
        }
    }
}
```

PhysicianList.java

```java
/**
 * PhysicianList Class
 * Creates the Physicians Node , contains methods related to node management that are
self explanatory.
 * @author Thomas Sadowski, William Gray, Keith Lopez, Shaifur Rahmans
 * @version 1
 */

import java.util.NoSuchElementException;

public class PhysicianList implements PhysicianListADT
{
    private PhysicianNode head;
    private int size;

    public PhysicianList()
    {
        head = null;
        size = 0;
    }

    @Override
    public void addToFront(String name, String specialty)
    {
        PhysicianNode newNode = new PhysicianNode(name, specialty);
        newNode.setNext(head);
        head = newNode;
        size++;
    }

    @Override
    public void addToEnd(String name, String specialty)
    {
        if (head == null)
        {
            addToFront(name, specialty);
            return;
        }
        else if (head.getNext() == null)
        {
            PhysicianNode newNode = new PhysicianNode(name, specialty);
            head.setNext(newNode);
        }
        else
            addToEnd(name, specialty, head.getNext());
        size++;
    }

    private void addToEnd(String name, String specialty, PhysicianNode node)
    {
        if (node.getNext() != null)
            addToEnd(name, specialty, node.getNext());
        else
```

```
        {
            PhysicianNode newNode = new PhysicianNode(name, specialty);
            node.setNext(newNode);
        }
    }

    @Override
    public String removeFirst()
    {
        if (head == null)
            throw new NoSuchElementException();
        String str = head.toString();
        head = head.getNext();
        size--;
        return str;
    }

    @Override
    public String removeLast()
    {
        if (head == null)
            throw new NoSuchElementException();
        String str = head.toString();
        if (head.getNext() == null)
        {
            head = null;
            size--;
            return str;
        }
        else return removeLast(head);
    }

    private String removeLast(PhysicianNode node)
    {
        PhysicianNode temp = node.getNext();
        if (temp.getNext() != null)
            removeLast(temp);
        else
        {
            node.setNext(null);
            size--;
            return temp.toString();
        }
        return "*";
    }

    @Override
    public boolean removeByQuery(String name, String specialty)
    {
        PhysicianNode query = new PhysicianNode(name, specialty);
        if (head == null)
            throw new NoSuchElementException();
        else if (head.equals(query))
        {
            head = head.getNext();
```

```
            size--;
            return true;
        }
        else return removeByQuery(query, head);
    }

    private boolean removeByQuery(PhysicianNode query, PhysicianNode traverse)
    {
        if (traverse.getNext() == null)
            return false;
        PhysicianNode temp = traverse.getNext();
        if (temp.equals(query))
        {
            traverse.setNext(temp.getNext());
            size--;
            return true;
        }
        else return removeByQuery(query, traverse.getNext());
    }

    @Override
    public void addVisit(String name, String specialty, String patient)
    {
        PhysicianNode query = new PhysicianNode(name, specialty);
        if (head == null)
            throw new NoSuchElementException();
        else if (head.equals(query))
            head.addVisit(name, specialty, patient);
        else addVisit(query, head, patient);
    }

    private void addVisit(PhysicianNode query, PhysicianNode traverse, String
patient)
    {
        if (traverse.getNext() == null)
            throw new NoSuchElementException();
        PhysicianNode temp = traverse.getNext();
        if (temp.equals(query))
            temp.addVisit(query.getName(), query.getSpecialty(), patient);
        else addVisit(query, traverse.getNext(), patient);
    }

    @Override
    public PhysicianVisit getVisits(String name, String specialty)
    {
        PhysicianNode query = new PhysicianNode(name, specialty);
        if (head == null)
            throw new NoSuchElementException();
        else if (head.equals(query))
            return head.getVisits();
        else return getVisits(query, head);
    }

    private PhysicianVisit getVisits(PhysicianNode query, PhysicianNode traverse)
    {
```

```
        if (traverse.getNext() == null)
            throw new NoSuchElementException();
        PhysicianNode temp = traverse.getNext();
        if (temp.equals(query))
            return temp.getVisits();
        else return getVisits(query, traverse.getNext());
    }

    @Override
    public boolean contains(String name, String specialty)
    {
        PhysicianNode query = new PhysicianNode(name, specialty);
        if (head == null)
            return false;
        else if (head.equals(query))
            return true;
        return contains(query, head);
    }

    private boolean contains(PhysicianNode query, PhysicianNode traverse)
    {
        if (traverse.getNext() == null)
            return false;
        PhysicianNode temp = traverse.getNext();
        if (temp.equals(query))
            return true;
        else return contains(query, traverse.getNext());
    }

    @Override
    public String[] toArray()
    {
        String[] physicians = new String[size];
        if (head == null)
            throw new NoSuchElementException();
        toArray(physicians, 0, head);
        return physicians;
    }

    private void toArray(String[] physicians, int index, PhysicianNode node)
    {
        if (node.getNext() == null)
            physicians[index] = node.toString();
        else
        {
            physicians[index] = node.toString();
            index++;
            toArray(physicians, index, node.getNext());
        }
    }

    public PhysicianNode[] toNodeArray()
    {
        PhysicianNode[] physicians = new PhysicianNode[size];
        if (head == null)
```

```
            throw new NoSuchElementException();
        toNodeArray(physicians, 0, head);
        return physicians;
    }

    private void toNodeArray(PhysicianNode[] physicians, int index, PhysicianNode
node)
    {
        if (node.getNext() == null)
            physicians[index] = node;
        else
        {
            physicians[index] = node;
            index++;
            toNodeArray(physicians, index, node.getNext());
        }
    }
}
```

PhysicianListADT

```
/**
 * PhysicianListADT
 * ADT for thePhysicianList Class
 * @author Thomas Sadowski, William Gray, Keith Lopez, Shaifur Rahmans
 * @version 1
 */

interface PhysicianListADT
{
    void addToFront(String name, String specialty);
    void addToEnd(String name, String specialty);
    String removeFirst();
    String removeLast();
    boolean removeByQuery(String name, String specialty);
    void addVisit(String name, String specialty, String patient);
    PhysicianVisit getVisits(String name, String specialty);
    boolean contains(String name, String specialty);
    String[] toArray();
}
```

PhysicianNode.java

```
/**
 * PhysicianNode Class
 * Creates the PhysicianNode for the Physicians information and contains methods
related to node management that are self explanatory.
 * @author Thomas Sadowski, William Gray, Keith Lopez, Shaifur Rahmans
 * @version 1
 */

public class PhysicianNode
{
    private String name;
    private String specialty;
    private PhysicianVisit visit;
    private PhysicianNode next;

    public PhysicianNode()
    {
        name = "J. Doe";
        specialty = "Unknown";
        visit = new PhysicianVisit();
        next = null;
    }

    public PhysicianNode(String name, String specialty)
    {
        this.name = name;
        this.specialty = specialty;
        visit = new PhysicianVisit();
        next = null;
    }

    public void setName(String str)
    {
        name = str;
    }

    public void setSpecialty(String str)
    {
        specialty = str;
    }

    public void addVisit(String patientName, String medicalCondition, String
physicianName)
    {
        visit.addToEnd(patientName, medicalCondition, physicianName);
    }

    public void setNext(PhysicianNode node)
    {
        next = node;
    }

    public String getName()
```

```
    {
        return name;
    }

    public String getSpecialty()
    {
        return specialty;
    }

    public PhysicianNode getNext()
    {
        return next;
    }

    public PhysicianVisit getVisits()
    {
        return visit;
    }

    public boolean equals(PhysicianNode other)
    {
        return name.equals(other.getName()) &&
specialty.equals(other.getSpecialty());
    }

    public String toString()
    {
        return "Physician name: " + name + " | Specialty: " + specialty;
    }
}
```

PhysicianVisit.java

```
/**
 * PhysicianVisit Class
 * Creates Nodes for the physician visit and contains methods related to node
management that are self explanatory.
 * @author Thomas Sadowski, William Gray, Keith Lopez, Shaifur Rahmans
 * @version 1
 */

import java.util.NoSuchElementException;

public class PhysicianVisit implements PhysicianVisitADT
{
    private PhysicianVisitNode head;
    private int size;
    VisitList list;

    public PhysicianVisit()
    {
        head = null;
        size = 0;
        list = new VisitList();
    }

    @Override
    public void addToFront(String patientName, String info, String physicianName)
    {
        list.addToFront(patientName, info, physicianName);
        VisitNode visitNode = new VisitNode(patientName, info, physicianName);
        PhysicianVisitNode newNode = new PhysicianVisitNode(visitNode);
        newNode.setNext(head);
        head = newNode;
        size++;
    }

    @Override
    public void addToEnd(String patientName, String info, String physicianName)
    {
        VisitNode visitNode = new VisitNode(patientName, info, physicianName);
        list.addToEnd(patientName, info, physicianName);
        if (head == null)
        {
            addToFront(patientName, info, physicianName);
            return;
        }
        else if (head.getNext() == null)
        {
            PhysicianVisitNode newNode = new PhysicianVisitNode(visitNode);
            head.setNext(newNode);
        }
        else
            addToEnd(visitNode, head.getNext());
        size++;
    }
```

```
private void addToEnd(VisitNode visitNode, PhysicianVisitNode node)
{
    if (node.getNext() != null)
        addToEnd(visitNode, node.getNext());
    else
    {
        PhysicianVisitNode newNode = new PhysicianVisitNode(visitNode);
        node.setNext(newNode);
    }
}

@Override
public String removeFirst()
{
    list.removeFirst();
    if (head == null)
        throw new NoSuchElementException();
    String str = head.toString();
    head = head.getNext();
    size--;
    return str;
}

@Override
public String removeLast()
{
    list.removeLast();
    if (head == null)
        throw new NoSuchElementException();
    String str = head.toString();
    if (head.getNext() == null)
    {
        head = null;
        size--;
        return str;
    }
    else return removeLast(head);
}

private String removeLast(PhysicianVisitNode node)
{
    PhysicianVisitNode temp = node.getNext();
    if (temp.getNext() != null)
        removeLast(temp);
    else
    {
        node.setNext(null);
        size--;
        return temp.toString();
    }
    return "*";
}

@Override
```

```
    public boolean removeByQuery(VisitNode visitNode)
    {
        list.removeByQuery(visitNode);
        PhysicianVisitNode query = new PhysicianVisitNode(visitNode);
        if (head == null)
            throw new NoSuchElementException();
        else if (head.equals(query))
        {
            head = head.getNext();
            size--;
            return true;
        }
        else return removeByQuery(query, head);
    }

    private boolean removeByQuery(PhysicianVisitNode query, PhysicianVisitNode
traverse)
    {
        if (traverse.getNext() == null)
            return false;
        PhysicianVisitNode temp = traverse.getNext();
        if (temp.equals(query))
        {
            traverse.setNext(temp.getNext());
            size--;
            return true;
        }
        else return removeByQuery(query, traverse.getNext());
    }

    @Override
    public boolean contains(VisitNode visitNode)
    {
        PhysicianVisitNode query = new PhysicianVisitNode(visitNode);
        if (head == null)
            return false;
        else if (head.equals(query))
            return true;
        else return contains(query, head);
    }

    private boolean contains(PhysicianVisitNode query, PhysicianVisitNode traverse)
    {
        if (traverse.getNext() == null)
            return false;
        PhysicianVisitNode temp = traverse.getNext();
        if (temp.equals(query))
            return true;
        else return contains(query, traverse.getNext());
    }

    @Override
    public String[] toArray()
    {
        String[] physicians = new String[size];
```

```
        if (head == null)
            throw new NoSuchElementException();
        toArray(physicians, 0, head);
        return physicians;
    }

    private void toArray(String[] physicians, int index, PhysicianVisitNode node)
    {
        if (node.getNext() == null)
            physicians[index] = node.toString();
        else
        {
            physicians[index] = node.toString();
            index++;
            toArray(physicians, index, node.getNext());
        }
    }
}
```

PhysicianVisitADT.java

```
/**
 * PhysicianVisitADT
 * ADT for the PhysicianVisit Class
 * @author Thomas Sadowski, William Gray, Keith Lopez, Shaifur Rahmans
 * @version 1
 */

public interface PhysicianVisitADT
{
    void addToFront(String patientName, String medicalCondition, String
physicianName);
    void addToEnd(String patientName, String medicalCondition, String physicianName);
    String removeFirst();
    String removeLast();
    boolean removeByQuery(VisitNode visitNode);
    boolean contains(VisitNode visitNode);
    String[] toArray();
}
```

PhysicianVisitNode.java

```
/**
 * PhysicianVisitNode Class
 * Creates the  VisitNode for the Physicians and contains methods related to node
management that are self explanatory.
 * @author Thomas Sadowski, William Gray, Keith Lopez, Shaifur Rahmans
 * @version 1
 */

public class PhysicianVisitNode
{
    VisitNode visitNode;
    private PhysicianVisitNode next;

    public PhysicianVisitNode()
    {
        visitNode = new VisitNode();
        next = null;
    }

    public PhysicianVisitNode(VisitNode visitNode)
    {
        this.visitNode = visitNode;
        next = null;
    }

    public void setVisitNode(VisitNode visitNode)
    {
        this.visitNode = visitNode;
    }

    public void setNext(PhysicianVisitNode node)
    {
        next = node;
    }

    public VisitNode getVisitNode()
    {
        return visitNode;
    }

    public PhysicianVisitNode getNext()
    {
        return next;
    }

    public boolean equals(PhysicianVisitNode other)
    {
        return visitNode.equals(other);
    }

    public String toString()
    {
        return visitNode.toString();
```

```
    }
}
```

VisitList.java

```
/**
 * VisitList Class
 * Creates the Visit Node , contains methods related to node management that are self
explanatory.
 * @author Thomas Sadowski, William Gray, Keith Lopez, Shaifur Rahmans
 * @version 1
 */

import java.util.NoSuchElementException;

public class VisitList implements VisitListADT
{
    private VisitNode head;
    private int size;

    public VisitList()
    {
        head = null;
        size = 0;
    }

    @Override
    public void addToFront(String patientName, String info, String physicianName)
    {
        VisitNode newNode = new VisitNode(patientName, info, physicianName);
        newNode.setNext(head);
        head = newNode;
        size++;
    }

    @Override
    public void addToEnd(String patientName, String info, String physicianName)
    {
        if (head == null)
        {
            addToFront(patientName, info, physicianName);
            return;
        }
        else if (head.getNext() == null)
        {
            VisitNode newNode = new VisitNode(patientName, info, physicianName);
            head.setNext(newNode);
        }
        else
            addToEnd(patientName, info, physicianName, head.getNext());
        size++;
    }

    private void addToEnd(String patientName, String info, String physicianName,
VisitNode node)
    {
        if (node.getNext() != null)
            addToEnd(patientName, info, physicianName, node.getNext());
```

```
        else
        {
            VisitNode newNode = new VisitNode(patientName, info, physicianName);
            node.setNext(newNode);
        }
    }

    @Override
    public String removeFirst()
    {
        if (head == null)
            throw new NoSuchElementException();
        String str = head.toString();
        head = head.getNext();
        size--;
        return str;
    }

    @Override
    public String removeLast()
    {
        if (head == null)
            throw new NoSuchElementException();
        String str = head.toString();
        if (head.getNext() == null)
        {
            head = null;
            size--;
            return str;
        }
        else return removeLast(head);
    }

    private String removeLast(VisitNode node)
    {
        VisitNode temp = node.getNext();
        if (temp.getNext() != null)
            removeLast(temp);
        else
        {
            node.setNext(null);
            size--;
            return temp.toString();
        }
        return "*";
    }

    @Override
    public boolean removeByQuery(VisitNode query)
    {
        if (head == null)
            throw new NoSuchElementException();
        else if (head.equals(query))
        {
            head = head.getNext();
```

```
            size--;
            return true;
        }
        else return removeByQuery(query, head);
    }

    private boolean removeByQuery(VisitNode query, VisitNode traverse)
    {
        if (traverse.getNext() == null)
            return false;
        VisitNode temp = traverse.getNext();
        if (temp.equals(query))
        {
            traverse.setNext(temp.getNext());
            size--;
            return true;
        }
        else return removeByQuery(query, traverse.getNext());
    }

    @Override
    public boolean contains(String patientName, String info, String physicianName)
    {
        VisitNode query = new VisitNode(patientName, info, physicianName);
        if (head == null)
            return false;
        else if (head.equals(query))
            return true;
        else return contains(query, head);
    }

    private boolean contains(VisitNode query, VisitNode traverse)
    {
        if (traverse.getNext() == null)
            return false;
        VisitNode temp = traverse.getNext();
        if (temp.equals(query))
            return true;
        else return contains(query, traverse.getNext());
    }

    @Override
    public String[] toArray()
    {
        String[] visits = new String[size];
        if (head == null)
            throw new NoSuchElementException();
        toArray(visits, 0, head);
        return visits;
    }

    private void toArray(String[] visits, int index, VisitNode node)
    {
        if (node.getNext() == null)
            visits[index] = node.toString();
```

```
        else
        {
            visits[index] = node.toString();
            index++;
            toArray(visits, index, node.getNext());
        }
    }
}
```

VisitListADT.java

```
/**
 * PatientVisitADT
 * ADT for the VisitList Class
 * @author Thomas Sadowski, William Gray, Keith Lopez, Shaifur Rahmans
 * @version 1
 */

interface VisitListADT
{
    void addToFront(String patientName, String specialty, String physicianName);
    void addToEnd(String patientName, String specialty, String physicianName);
    String removeFirst();
    String removeLast();
    boolean removeByQuery(VisitNode query);
    boolean contains(String patientName, String medicalCondition, String
physicianName);
    String[] toArray();
}
```

VisitNode.java

```java
/**
 * VisitNode Class
 * Creates the Visit Node for dr and patient information and contains methods related
to node management that are self explanatory.
 * @author Thomas Sadowski, William Gray, Keith Lopez, Shaifur Rahmans
 * @version 1
 */

public class VisitNode
{
    private String patientName;
    private String info;
    private String physicianName;
    private VisitNode next;

    public VisitNode()
    {
        patientName = "J. Doe";
        info = "Unknown";
        physicianName = "Unassigned";
        next = null;
    }

    public VisitNode(String patientName, String info, String physicianName)
    {
        this.patientName = patientName;
        this.info = info;
        this.physicianName = physicianName;
        next = null;
    }

    public void setPatientName(String str)
    {
        patientName = str;
    }

    public void setInfo(String str)
    {
        info = str;
    }

    public void setNext(VisitNode node)
    {
        next = node;
    }

    public String getPatientName()
    {
        return patientName;
    }

    public String getinfo()
    {
```

```
        return info;
    }

    public String getPhysicianName()
    {
        return physicianName;
    }

    public VisitNode getNext()
    {
        return next;
    }

    public boolean equals(VisitNode other)
    {
        return patientName.equals(other.getPatientName()) &&
            info.equals(other.getinfo()) &&
physicianName.equals(other.getPhysicianName());
    }

    public String toString()
    {
        return "Patient Name: " + patientName + " | Patient Info: " + info +
                " | Physician Name: " + physicianName;
    }
}
```

Testing code

Tester.java

```
/**
 * Tester
 * Simulates a Hospital Information Management system.
 * @author Thomas Sadowski, William Gray, Keith Lopez, Shaifur Rahmans
 * @version 1
 */

import java.util.Scanner;
import java.util.*;

public class Tester
{
    static PatientList patients = new PatientList();
    static PatientIndexTree patientsTree = new PatientIndexTree();
    static PhysicianList physicians = new PhysicianList();
    static PhysicianIndexTree physiciansTree = new PhysicianIndexTree();
    static Scanner console = new Scanner(System.in);

    public static void main(String[] args)
    {
        int selection = -1;
        while (selection != 0)
        {
            mainMenu();
            selection = console.nextInt();
            switch (selection)
            {
                case 0:
                    System.out.println("Bye! ");
                    break;
                case 1:
                    addPatientDialogue();
                    break;
                case 2:
                    addPhysicianDialogue();
                    break;
                case 3:
                    addVisitDialogue();
                    break;
                case 4:
                    printPatientVisitDialogue();
                    break;
                case 5:
                    printPhysicianVisitDialogue();
                    break;
                case 6:
                    displayPatientsAlpha();
                    break;
                case 7:
                    displayPhysiciansAlpha();
```

```
                    break;
                default: System.out.println("You have entered an invalid value. ");
            }
        }
    }

    public static void mainMenu()
    {
        System.out.println("**********  M A I N  M E N U  **********");
        System.out.println("(1) Add a new patient \n"
                         + "(2) Add a new physician \n"
                         + "(3) Add a new visit for a patient \n"
                         + "(4) Display all visits of a patient based on visit order
\n"
                         + "(5) Display all visits of a physician based on visit
order \n"
                         + "(6) Display all patients in alphabetical order \n"
                         + "(7) Display all physicians in alphabetical order \n"
                         + "(0) Exit program \n");
    }

    public static void addPatientDialogue()
    {
        String choice = "a";
        while (choice.equalsIgnoreCase("a"))
        {
            System.out.println("********** A D D  P A T I E N T  D I A L O G U E
**********");
            String name, info, medicalCondition;
            System.out.println("Enter the patient's name: ");
            console.nextLine();
            name = console.nextLine();
            name = name.trim();
            System.out.println("Enter the patient's info: ");
            info = console.nextLine();
            info = info.trim();
            System.out.println("Enter the patient's medical condition: ");
            medicalCondition = console.nextLine();
            medicalCondition = medicalCondition.trim();
            patients.addToFront(name, info, medicalCondition);
            patientsTree.add(name, info, medicalCondition);
            do
            {
                System.out.println("Enter 'A' to add another or 'M' to return to the
main menu: ");
                choice = console.next();
                choice = choice.trim();
            }
            while (!choice.equalsIgnoreCase("a") && !choice.equalsIgnoreCase("m"));
            System.out.println();
        }
    }

    public static void addPhysicianDialogue()
    {
```

```
        String choice = "a";
        while (choice.equalsIgnoreCase("a"))
        {
            System.out.println("********** A D D  P H Y S I C I A N  D I A L O G U E
**********");
            String name, specialty;
            System.out.println("Enter the physician's name: ");
            console.nextLine();
            name = console.nextLine();
            name = name.trim();
            System.out.println("Enter the physician's specialty: ");
            specialty = console.nextLine();
            specialty = specialty.trim();
            physicians.addToFront(name, specialty);
            physiciansTree.add(name, specialty);
            do
            {
                System.out.println("Enter 'A' to add another or 'M' to return to the
main menu: ");
                choice = console.next();
                choice = choice.trim();
            }
            while (!choice.equalsIgnoreCase("a") && !choice.equalsIgnoreCase("m"));
            System.out.println();
        }
    }

    public static void addVisitDialogue()
    {
        String choice = "a";
        if (!patientsTree.isEmpty() && !physiciansTree.isEmpty())
            while (choice.equalsIgnoreCase("a"))
            {
                System.out.println("********** A D D  V I S I T  D I A L O G U E
**********");
                String patientName, patientInfo, physician, specialty;
                displayPatientsAlpha();
                System.out.println("Enter the patient's name: ");
                console.nextLine();
                patientName = console.nextLine();
                patientName = patientName.trim();
                System.out.println("Enter the patient's info: ");
                patientInfo = console.nextLine();
                patientInfo = patientInfo.trim();
                displayPhysiciansAlpha();
                System.out.println("Enter the physician's name: ");
                physician = console.nextLine();
                physician = physician.trim();
                System.out.println("Enter the physician's specialty: ");
                specialty = console.nextLine();
                specialty = specialty.trim();
                try
                {
                    patients.addVisit(patientName, patientInfo, physician);
                    physicians.addVisit(physician, specialty, patientName);
```

```
                }
                catch (NoSuchElementException e)
                {
                    System.out.println("The patient or doctor entered have not been
found in the "
                            + "database. ");
                }
                do
                {
                    System.out.println("Enter 'A' to add another or 'M' to return to
the main menu: ");
                    choice = console.next();
                    choice = choice.trim();
                }
                while (!choice.equalsIgnoreCase("a") &&
!choice.equalsIgnoreCase("m"));
                System.out.println();
            }
        else
            System.out.println("The database must have at least 1 patient and "
                    + "at least 1 physician. ");
    }

    public static void printPatientVisitDialogue()
    {
        String choice = "d";
        while (choice.equalsIgnoreCase("d"))
        {
            System.out.println("********** D I S P L A Y  P A T I E N T  V I S I T S"
                    + "  D I A L O G U E **********");
            String name, info;
            System.out.println("Enter the patient's name: ");
            console.nextLine();
            name = console.nextLine();
            name = name.trim();
            System.out.println("Enter the patient's info: ");
            info = console.nextLine();
            info = info.trim();
            if (patients.contains(name, info))
            {
                try
                {
                    String[] arr = patients.getVisits(name, info).toArray();
                    for (int i = 0; i < arr.length; i++)
                        System.out.println(arr[i]);
                }
                catch (NoSuchElementException e)
                {
                    System.out.println("No visits found for this patient. ");
                }
            }
            else
                System.out.println("Patient does not exist. ");
            do
            {
```

```
                System.out.println("Enter 'D' to display another or 'M' to "
                                   + "return to the main menu: ");
                choice = console.next();
                choice = choice.trim();
            }
            while (!choice.equalsIgnoreCase("d") && !choice.equalsIgnoreCase("m"));
            System.out.println();
        }
    }

    public static void printPhysicianVisitDialogue()
    {
        String choice = "d";
        while (choice.equalsIgnoreCase("d"))
        {
            System.out.println("********** D I S P L A Y  P H Y S I C I A N  V I S I
T S"
                               + "  D I A L O G U E **********");
            String name, specialty;
            System.out.println("Enter the physician's name: ");
            console.nextLine();
            name = console.nextLine();
            name = name.trim();
            System.out.println("Enter the physician's info: ");
            specialty = console.nextLine();
            specialty = specialty.trim();
            if (physicians.contains(name, specialty))
            {
                try
                {
                    String[] arr = physicians.getVisits(name, specialty).toArray();
                    for (int i = 0; i < arr.length; i++)
                        System.out.println(arr[i]);
                }
                catch (NoSuchElementException e)
                {
                    System.out.println("No visits found for this physician. ");
                }
            }
            else
                System.out.println("Physician does not exist. ");
            do
            {
                System.out.println("Enter 'D' to display another or 'M' to "
                                   + "return to the main menu: ");
                choice = console.next();
                choice = choice.trim();
            }
            while (!choice.equalsIgnoreCase("d") && !choice.equalsIgnoreCase("m"));
            System.out.println();
        }
    }

    public static void displayPatientsAlpha()
    {
```

```
        if (!patientsTree.isEmpty())
        {
            System.out.println("Patients: ");
            patientsTree.printOrderedData();
        }
        else
            System.out.println("No patients in database. ");
    }

    public static void displayPhysiciansAlpha()
    {
        if (!physiciansTree.isEmpty())
        {
            System.out.println("Physicians: ");
            physiciansTree.printOrderedData();
        }
        else
            System.out.println("No physicians in database. ");
    }
}
```

Testing results

Console

```
<terminated> Tester (10) [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Apr 16, 2018, 10:26:26 PM)
**********  M A I N  M E N U  **********
(1) Add a new patient
(2) Add a new physician
(3) Add a new visit for a patient
(4) Display all visits of a patient based on visit order
(5) Display all visits of a physician based on visit order
(6) Display all patients in alphabetical order
(7) Display all physicians in alphabetical order
(0) Exit program

1
********** A D D  P A T I E N T  D I A L O G U E **********
Enter the patient's name:
Bob Dingle
Enter the patient's info:
123 Shakedown St
Enter the patient's medical condition:
Chiropractic
Enter 'A' to add another or 'M' to return to the main menu:
A

********** A D D  P A T I E N T  D I A L O G U E **********
Enter the patient's name:
Marco Polo
Enter the patient's info:
567 Shakeup St
Enter the patient's medical condition:
Cardiology
Enter 'A' to add another or 'M' to return to the main menu:
M

**********  M A I N  M E N U  **********
(1) Add a new patient
(2) Add a new physician
(3) Add a new visit for a patient
(4) Display all visits of a patient based on visit order
(5) Display all visits of a physician based on visit order
(6) Display all patients in alphabetical order
(7) Display all physicians in alphabetical order
(0) Exit program

2
********** A D D  P H Y S I C I A N  D I A L O G U E **********
Enter the physician's name:
Harry Killjoy
Enter the physician's specialty:
Chiropractic
Enter 'A' to add another or 'M' to return to the main menu:
A

********** A D D  P H Y S I C I A N  D I A L O G U E **********
Enter the physician's name:
Dweezle Zingo
Enter the physician's specialty:
Cardiology
Enter 'A' to add another or 'M' to return to the main menu:
M
```

```
**********  M A I N  M E N U  **********
(1) Add a new patient
(2) Add a new physician
(3) Add a new visit for a patient
(4) Display all visits of a patient based on visit order
(5) Display all visits of a physician based on visit order
(6) Display all patients in alphabetical order
(7) Display all physicians in alphabetical order
(0) Exit program

2
********** A D D  P H Y S I C I A N  D I A L O G U E **********
Enter the physician's name:
Harry Killjoy
Enter the physician's specialty:
Chiropractic
Enter 'A' to add another or 'M' to return to the main menu:
A

********** A D D  P H Y S I C I A N  D I A L O G U E **********
Enter the physician's name:
Dweezle Zingo
Enter the physician's specialty:
Cardiology
Enter 'A' to add another or 'M' to return to the main menu:
M

**********  M A I N  M E N U  **********
(1) Add a new patient
(2) Add a new physician
(3) Add a new visit for a patient
(4) Display all visits of a patient based on visit order
(5) Display all visits of a physician based on visit order
(6) Display all patients in alphabetical order
(7) Display all physicians in alphabetical order
(0) Exit program

3
********** A D D  V I S I T  D I A L O G U E **********
Patients:
Bob Dingle | 123 Shakedown St | Chiropractic
Marco Polo | 567 Shakeup St | Cardiology
Enter the patient's name:
Bob Dingle
Enter the patient's info:
123 Shakedown St
Physicians:
Dweezle Zingo | Cardiology
Harry Killjoy | Chiropractic
Enter the physician's name:
Harry Killjoy
Enter the physician's specialty:
Chiropractic
Enter 'A' to add another or 'M' to return to the main menu:
A
```

```
********** A D D  V I S I T  D I A L O G U E **********
Patients:
Bob Dingle | 123 Shakedown St | Chiropractic
Marco Polo | 567 Shakeup St | Cardiology
Enter the patient's name:
Marco Polo
Enter the patient's info:
567 Shakeup St
Physicians:
Dweezle Zingo | Cardiology
Harry Killjoy | Chiropractic
Enter the physician's name:
Dweezle Zingo
Enter the physician's specialty:
Cardiology
Enter 'A' to add another or 'M' to return to the main menu:
M

**********  M A I N  M E N U  **********
(1) Add a new patient
(2) Add a new physician
(3) Add a new visit for a patient
(4) Display all visits of a patient based on visit order
(5) Display all visits of a physician based on visit order
(6) Display all patients in alphabetical order
(7) Display all physicians in alphabetical order
(0) Exit program

4
********** D I S P L A Y  P A T I E N T  V I S I T S  D I A L O G U E **********
Enter the patient's name:
Bob Dingle
Enter the patient's info:
123 Shakedown St
Patient Name: Bob Dingle | Patient Info: 123 Shakedown St | Physician Name: Harry Killjoy
Enter 'D' to display another or 'M' to return to the main menu:
M

**********  M A I N  M E N U  **********
(1) Add a new patient
(2) Add a new physician
(3) Add a new visit for a patient
(4) Display all visits of a patient based on visit order
(5) Display all visits of a physician based on visit order
(6) Display all patients in alphabetical order
(7) Display all physicians in alphabetical order
(0) Exit program

5
********** D I S P L A Y  P H Y S I C I A N  V I S I T S  D I A L O G U E **********
Enter the physician's name:
Harry Killjoy
Enter the physician's info:
Chiropractic
Patient Name: Harry Killjoy | Patient Info: Chiropractic | Physician Name: Bob Dingle
Enter 'D' to display another or 'M' to return to the main menu:
M
```

```
**********  M A I N  M E N U  **********
(1) Add a new patient
(2) Add a new physician
(3) Add a new visit for a patient
(4) Display all visits of a patient based on visit order
(5) Display all visits of a physician based on visit order
(6) Display all patients in alphabetical order
(7) Display all physicians in alphabetical order
(0) Exit program

6
Patients:
Bob Dingle | 123 Shakedown St | Chiropractic
Marco Polo | 567 Shakeup St | Cardiology
**********  M A I N  M E N U  **********
(1) Add a new patient
(2) Add a new physician
(3) Add a new visit for a patient
(4) Display all visits of a patient based on visit order
(5) Display all visits of a physician based on visit order
(6) Display all patients in alphabetical order
(7) Display all physicians in alphabetical order
(0) Exit program

7
Physicians:
Dweezle Zingo | Cardiology
Harry Killjoy | Chiropractic
**********  M A I N  M E N U  **********
(1) Add a new patient
(2) Add a new physician
(3) Add a new visit for a patient
(4) Display all visits of a patient based on visit order
(5) Display all visits of a physician based on visit order
(6) Display all patients in alphabetical order
(7) Display all physicians in alphabetical order
(0) Exit program

0
Bye!
```