Name: Phuc Le

Course: CS146

Date: 11/19/17

Report Programming Assignment 2

1. Key concepts of this programming design.
2. Implement Binary Search tree data structure to manage patient in a hospital room.
3. Use Binary Search tree to illustrate the queue in the hospital.
4. Simulating normal activities in a hospital by using Binary Search tree’s method.
5. Implement Hash Table data structure to manage patient in a hospital.
6. Use Hash Table normal function to illustrate activities in a waiting room.
7. Classes:
8. Patient.

* Purpose: Object the contain patient’s information help manage patient more effective.
* Attribute:
  + Name: contain name of the patient.
  + Condition: contain current condition of the patient. (The greater the number, the higher priority).
* Method:
  + getName: Return name of patient.
  + getCondition: Return current condition of patient.
  + setName: set the name of patient.
  + setCondition: set the condition of patient.
  + getInfo: return all information of a patient including name and condition.

1. HashTable

- Purpose: This Hash Table class contain all the attribute as well as function needed in a Hash Function modified to fit the hospital waiting room.

- Attribute:

* 1. Chaining: Array of a Linked List Containing Patient Class.

- Method:

* 1. getPatient: Return Patient object base on the input condition.
  2. get: Return name of Patient base on the input condition.
  3. put: Adding a new patient into the Hashtable using their condition for Hash function.
  4. delete: delete a patient given their condition.

1. Hospitalhashtable

- Purpose: Main running class, which demonstrating the activities of a waiting room using basic Hash Function activities such as Insert, delete, find.

* Attribute:
  + name: array of random distinct names.
  + Patients: array of Object Patient.
  + Hash: instance of the HashTable class.
  + currentUsedIndex: variable used to keep track of the usage of name in the name array. (Which make sure that we don’t have two patients with the same name.)
  + currentOccupiedSeat: Keep track of number of patients in the queue at the moment.
* Method:
  + RandomAPatient: Random a distinct patient with distinct name and condition.
  + RandomPatients: Random an array of distinct patients.
  + PrintCurrentHashTable: Function use to print the current Patient in the hash table
  + CreateHashTable: Function to create a Hash Table base on the existing array of patient.
  + GetCondition(String a): Get input condition to delete or find a patient passing a parameter to get more information(a = “find” or “delete”).

1. Node<Patient>

- Purpose: This is the “Node” object used in the binary search tree.

- Attribute:

* 1. Data: contain the Patient object.
  2. Left: the child Node.
  3. Right: the right child Node

- Method:

* 1. toString: return the patient object information.
  2. setRight: set the right children.
  3. setLeft: set the left children.
  4. setP: set parent.

1. MyComparator

- Purpose: Comparator class used to compare the 2 node base on the condition of the patient object inside it.

1. BstHospital

- Purpose: This is the Binary Search tree Class modified to fit the waiting room activites.

- Attribute:

* + root: the root of the tree.
  + Comparator: comparator instance of comparator class.

- Method:

* + compare: compare 2 patient base on their condition.
  + TreeMinimum: return the minimum condition patient given the root of that tree.
  + TreeMaximum: return the maximum condition patient given the root of that tree.
  + Search: Search for a specific Node that contain a patient with a given condition. This method take in the root of that tree and condition.
  + InorderTreeWalk: this method print the patient in the tree out ascendingly base on their condition.
  + Insert: insert a new node into the current tree.
  + Transplant: Helping method for deleting.
  + Delete: delete a node from a tree and maintain the property of that Binary Tree.

1. hospitaltree

- Purpose: Main running class for hospital waiting room using binary search tree, which demonstrating the activities of a waiting room using basic Binary search tree Function activities such as Insert, delete, find and sort.

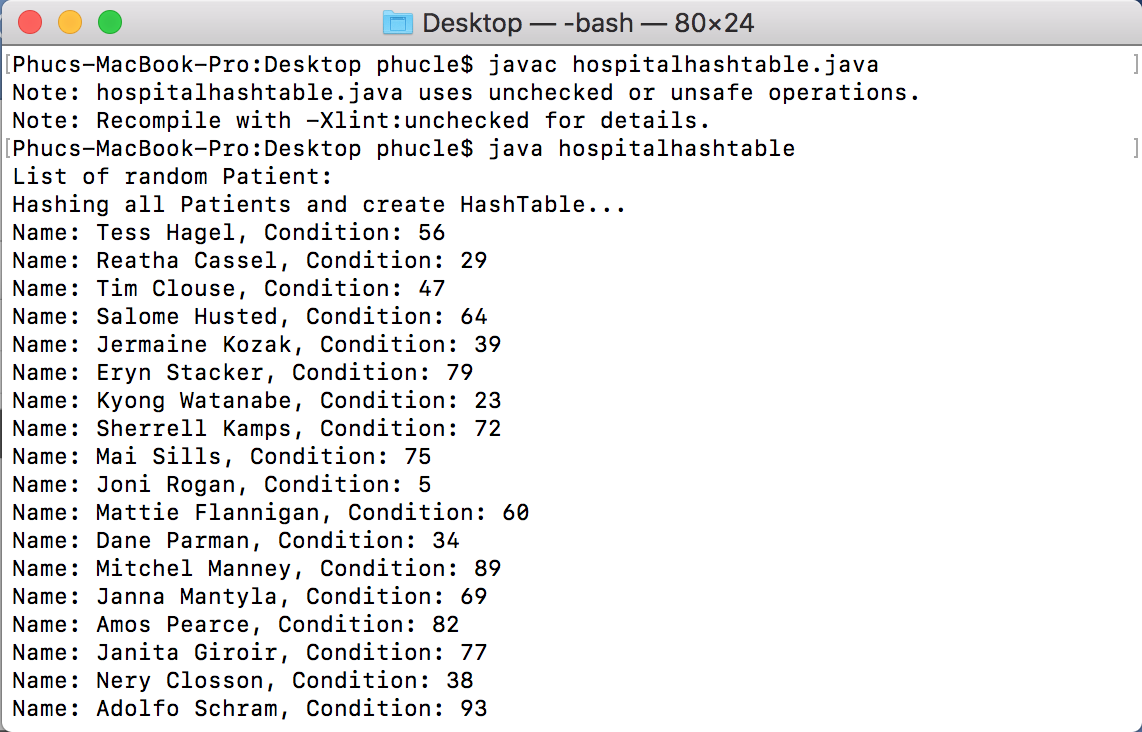
* Attribute:
  + name: array of random distinct names.
  + Patients: array of Object Patient.
  + BST: instance of the Bsthospital class.
  + currentUsedIndex: variable used to keep track of the usage of name in the name array. (Which make sure that we don’t have two patients with the same name.)
  + currentOccupiedSeat: Keep track of number of patients in the queue at the moment.
* Method:
  + RandomAPatient: Random a distinct patient with distinct name and condition.
  + RandomPatientsTree: Random an array of distinct patients and insert them into a binary search tree.
  + Print: Function use to print the current Patient in the Binary search tree in ascending order base on the patient condition.
  + GetCondition(String a): Get input condition to delete or find a patient passing a parameter to get more information(a = “find” or “delete”).

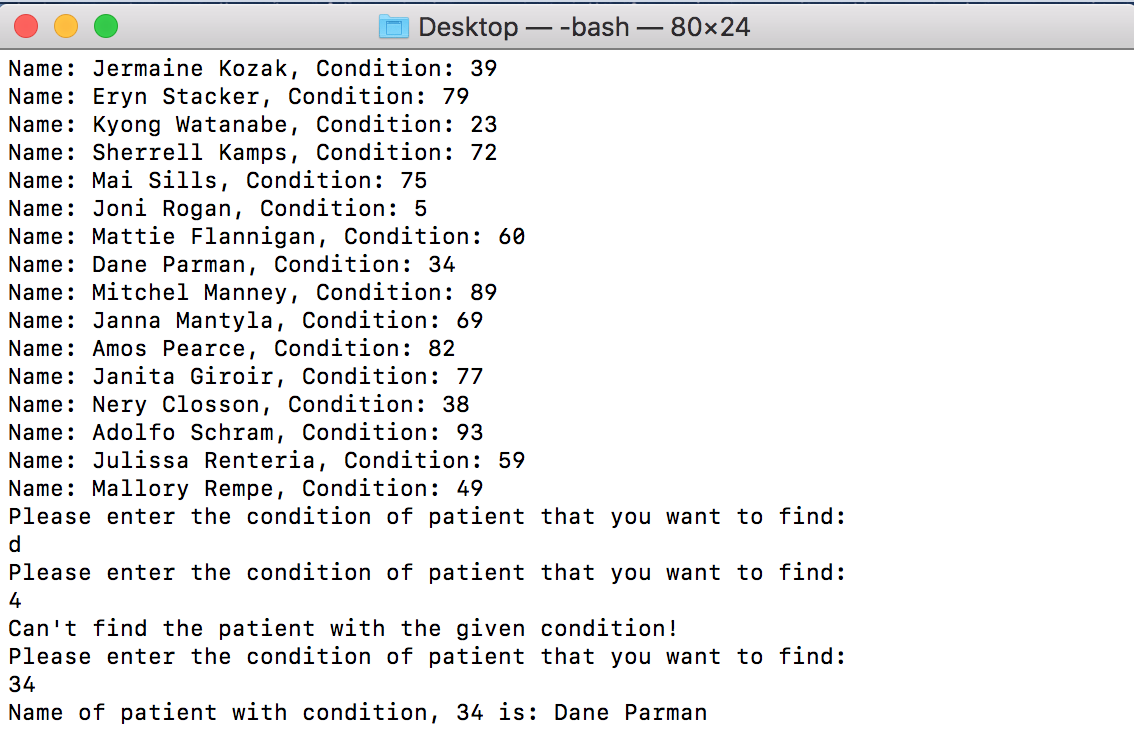
1. Screenshots and procedure to verify this code.

For the Hash Table version.

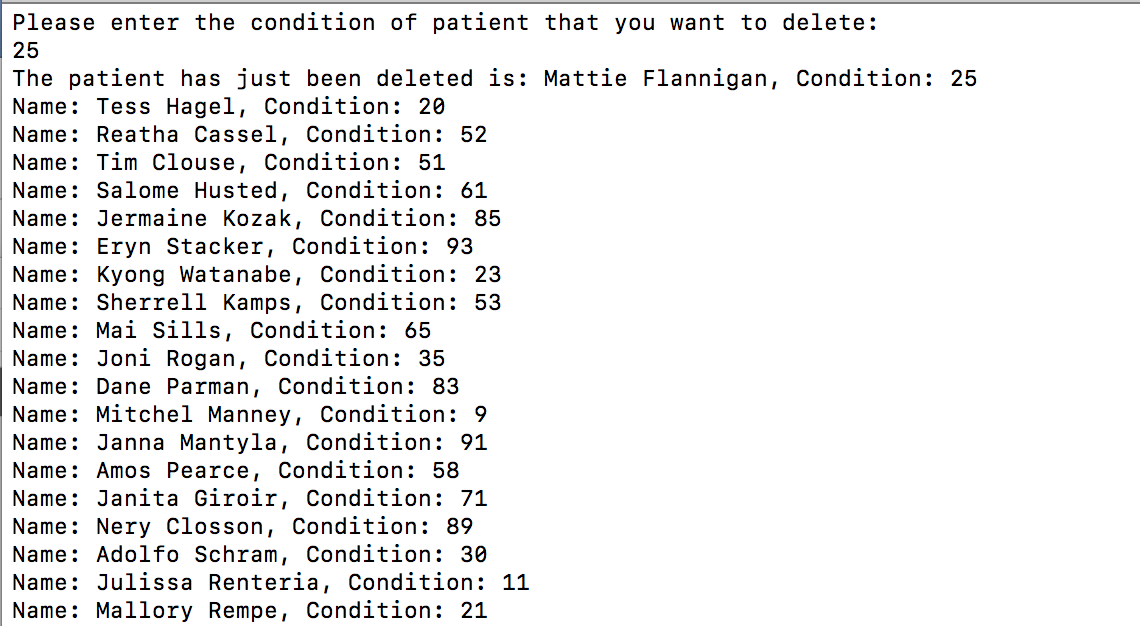
- After opening the file in terminal This window appears:

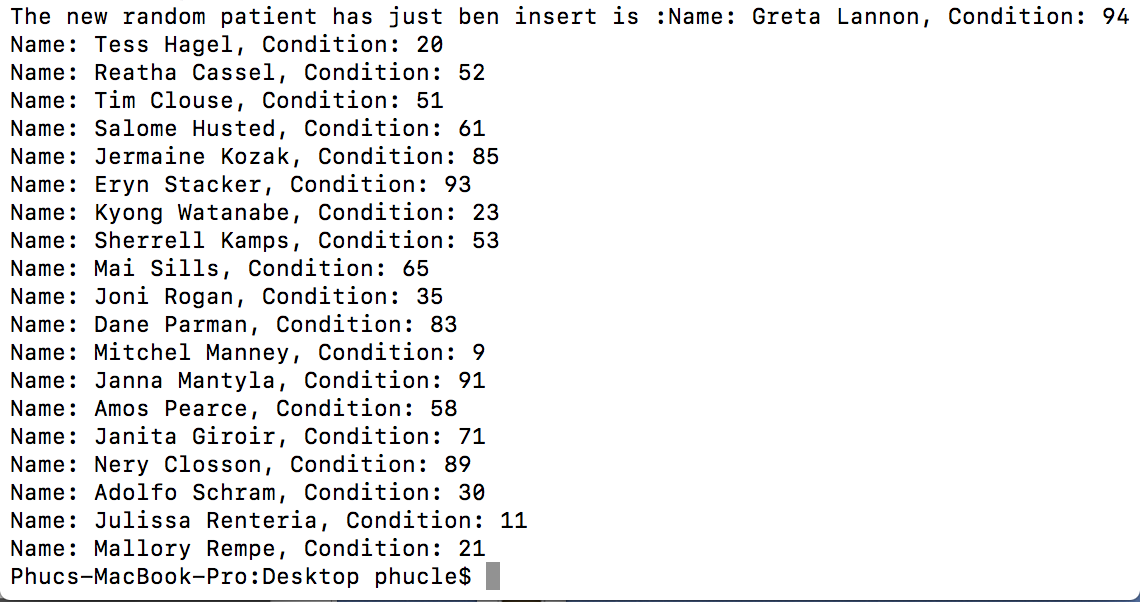
The program automatically creates a list of patient and insert hashing them into hash table. Then print out the current patient the hash table for user.

* The program then ask user to input a condition to search for a patient. The program will ask until the user enter a valid condition.



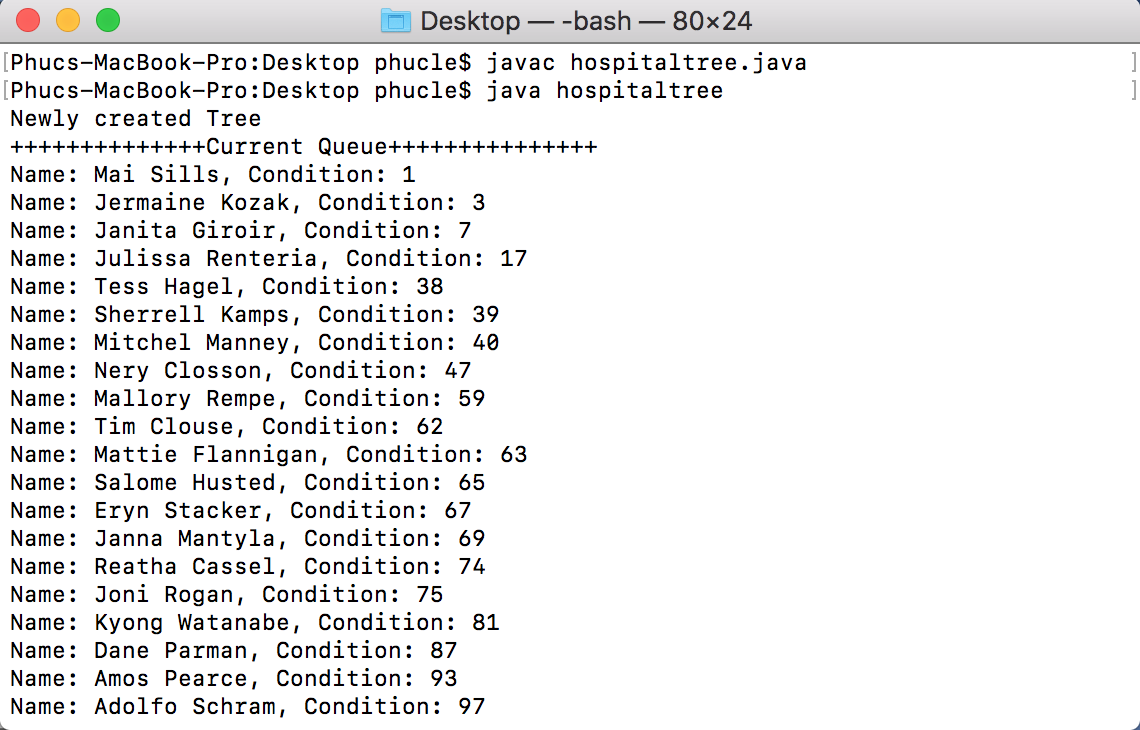
* Then, the program will ask user to input a condition to delete a patient. The program will ask until the user enter correct condition.

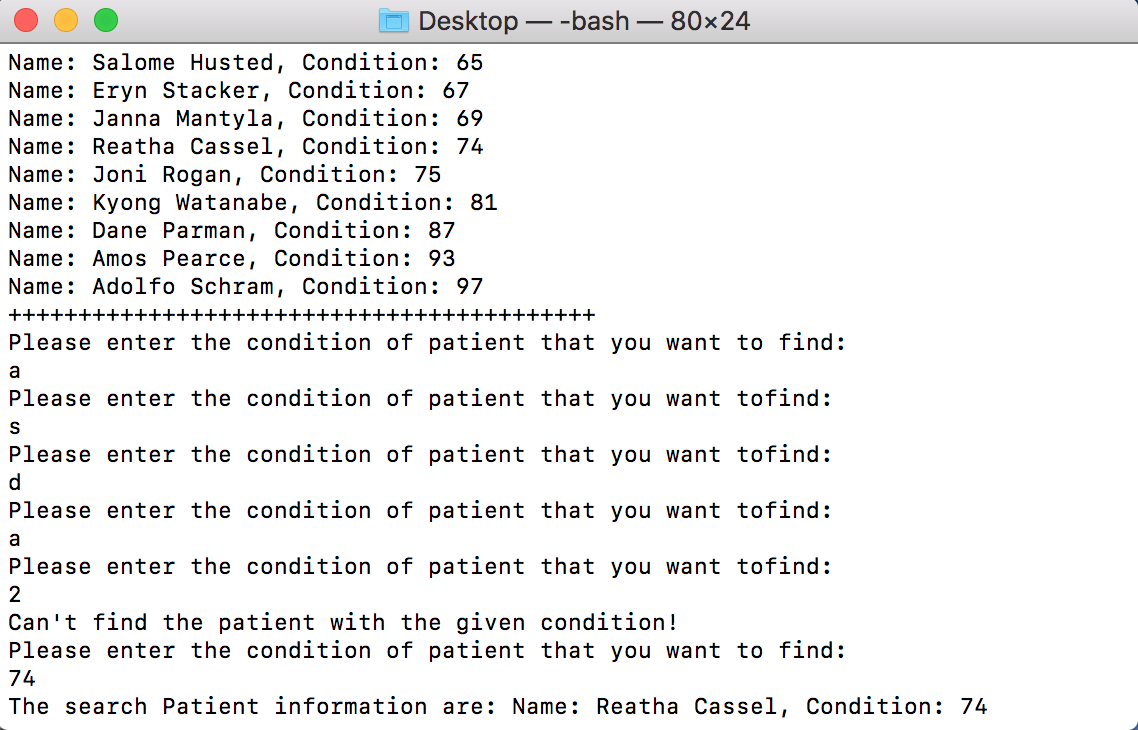
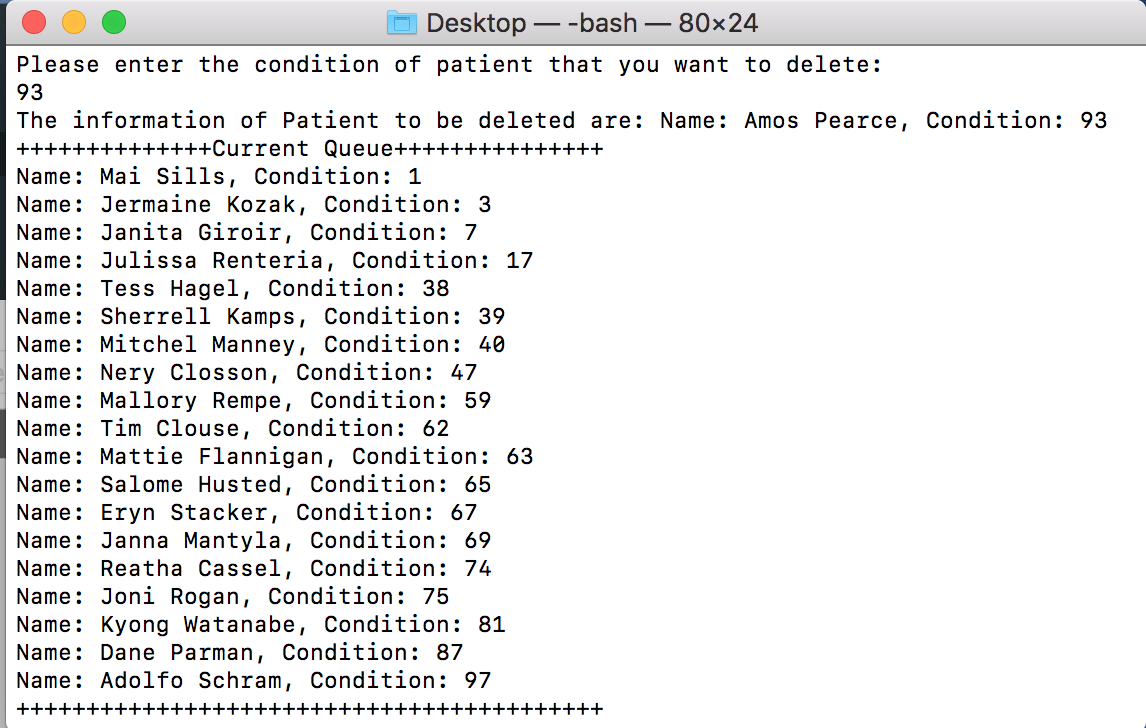
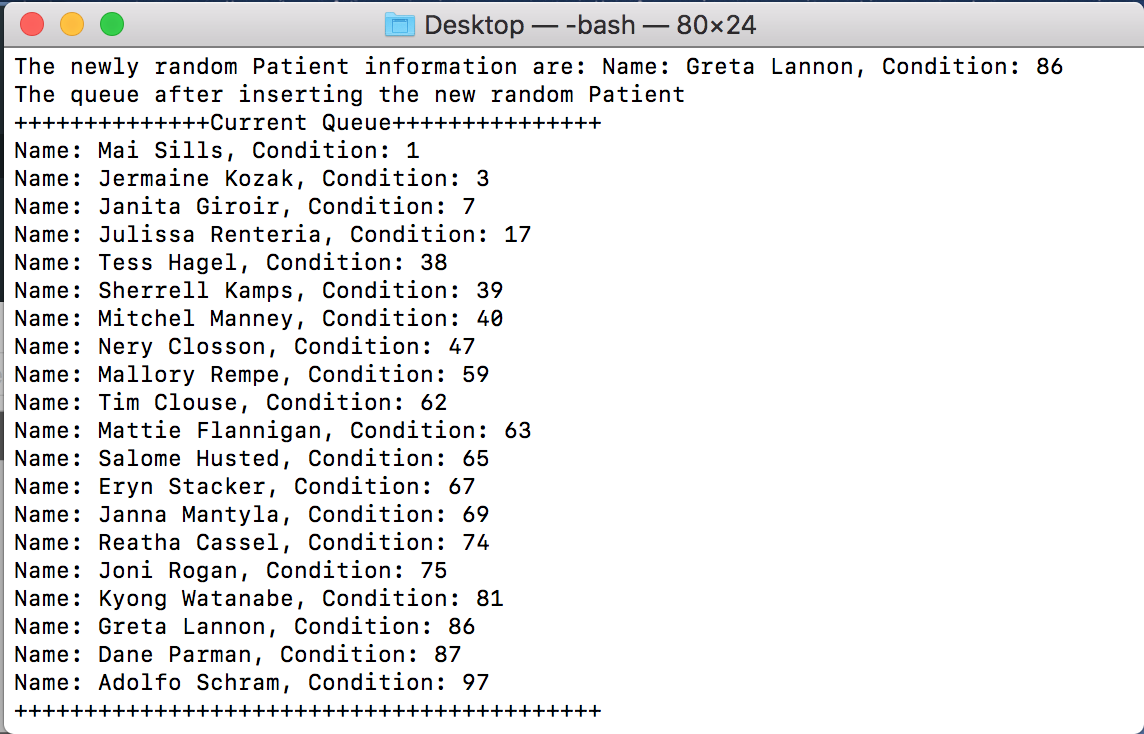


* The program then will automatically create a new distinct random patient and insert into the hash table. User now can look for the newly random and inserted patient.

For the Binary search tree version.

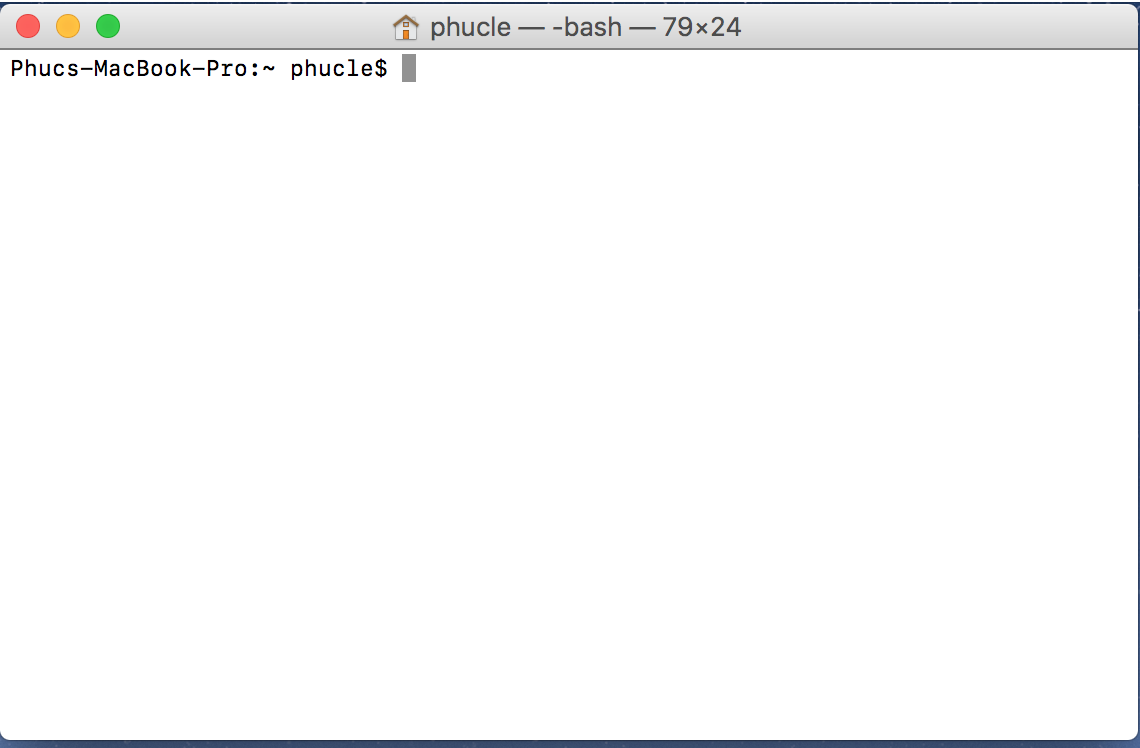
* After opening the file in terminal This window appears:

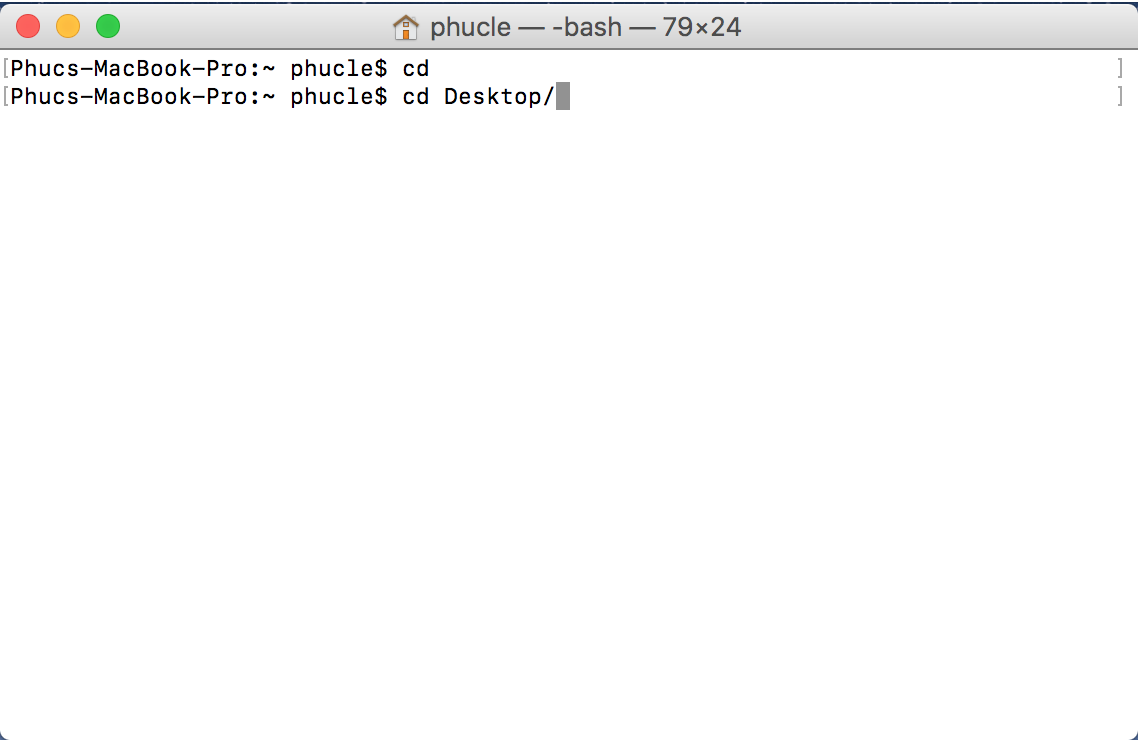
The program will automatically create a tree with random patient with distinct condition and name.

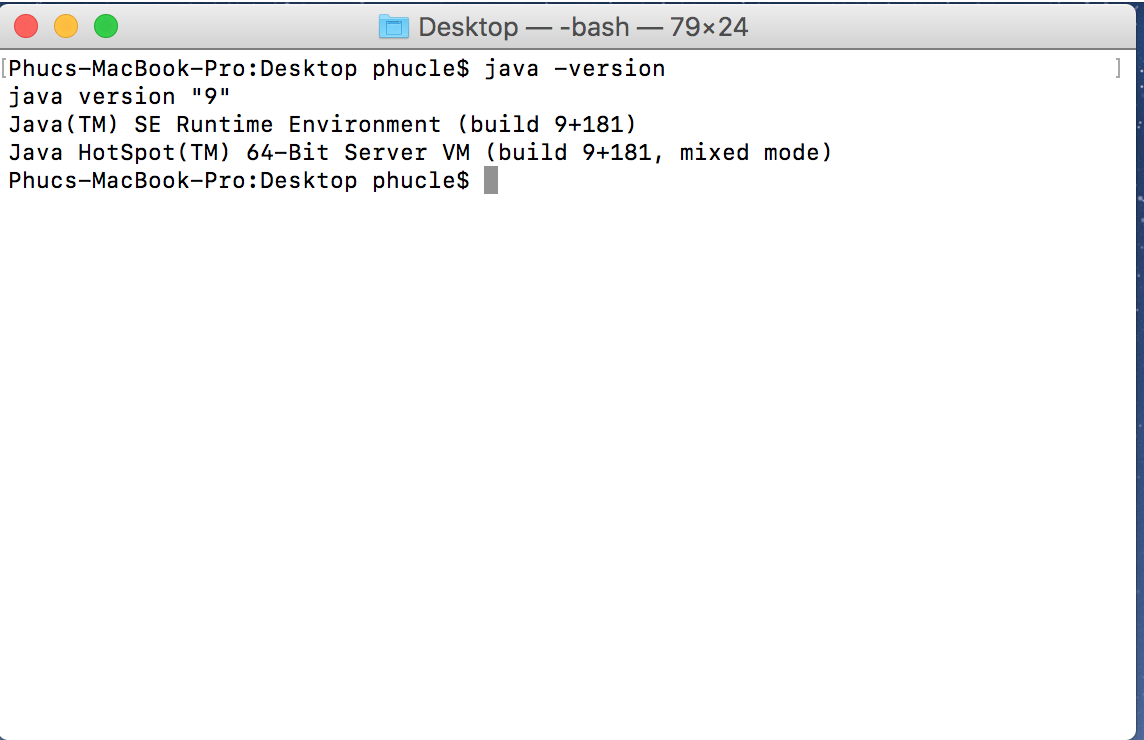
* Then user will be asked to input a condition to search for that patient. The program will ask until it get the valid input condition.
*  Then, the user may input a condition to delete the patient with that condition.
* After deleting the given condition patient, the program will automatically random a new patient with distinct name and condition then insert it into the tree. User may find the information of the newly insert patient.

1. How to unzip and run the program:

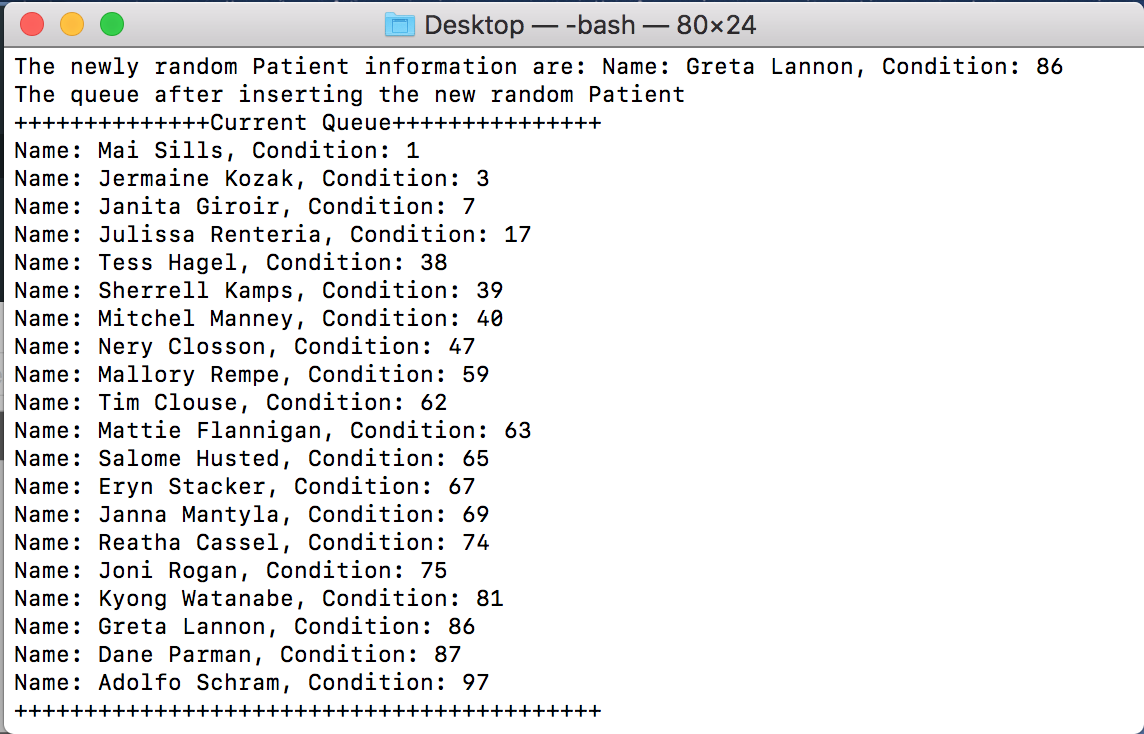
* Download the zip file named Le-PA1.zip and place in on your desktop.
* Unzip the file will create a folder named Le-PA1.
* Copy and Paste the .java file named ProgrammingHW1.java on your desktop.
* Open your terminal (Or command line if you are using window).
* This window will appear:



* Type “cd” then enter.
* Type “cd Desktop/” then enter.
* Type “java –version” to check if your computer has installed the Java SE runtime environment. If you see the below screen you are Ok. If you are not seeing this screen please download the Java RE and JDK, which can be found here:

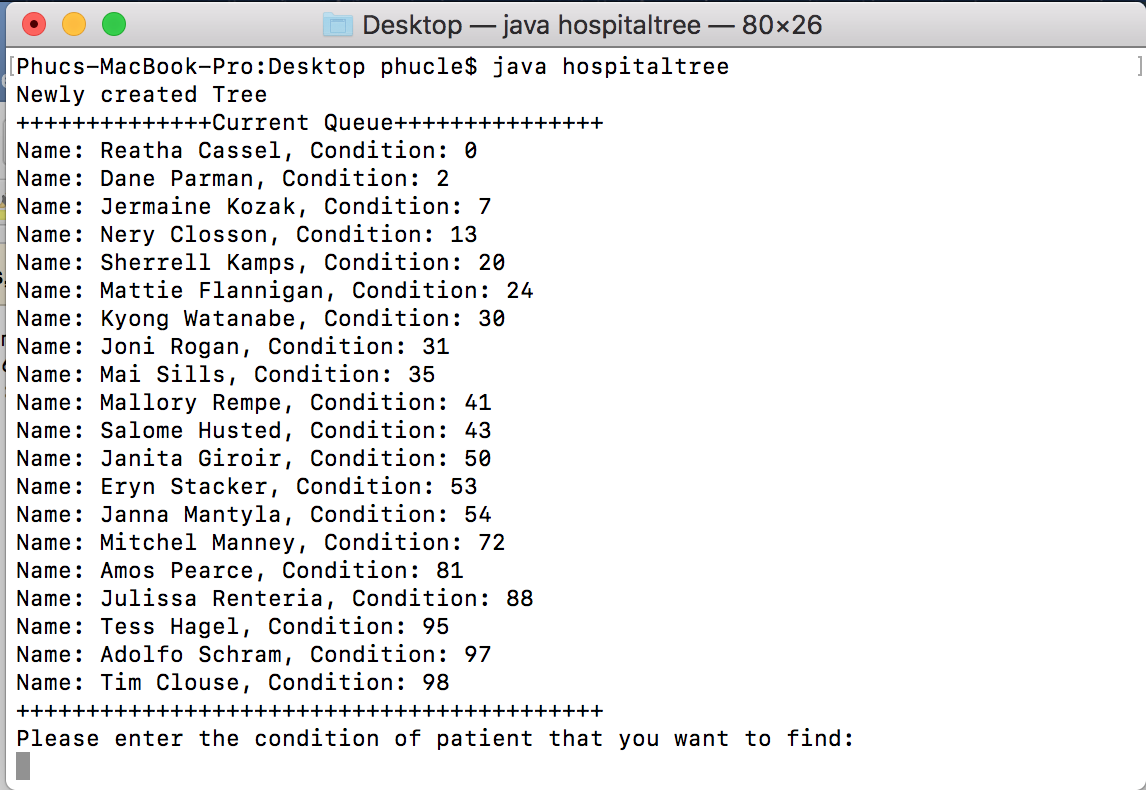
<http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html>

* For Hash Table version:

Type “javac hospitalhashtable.java” and enter, then type “java hospitalhashtable” and enter. The program should be running now and the following window will appear:

* For the Binay Search tree version:

Type “javac hospitaltree.java” and enter, then type “java hospitaltree” and enter. The program should be running now and the following window will appear:



1. Problems encountered during the implementation.

* The Binary search tree require a method to compare the 2 nodes and that is the reason why we have the comparator class.
* The condition that user input has to be valid and the program will continues asking for valid condition.

1. Lesson Learned.

* Understand how Binary Search Tree and Hash method work and how to code them in java environment.
* How to get an array with different values.
* Understand chaining solution for collision in hash function.
* Review on using comparator class.