```
Script started on 2021-02-24 21:16:47-0600
l_ladios@ares:~$ cat ScoreEntryLab.info
 /*************************************************************************
     *                                                                       *
     *  NAME:  Leia Ladios                         CLASS:  CSC121-W01        *
     *                                                                       *
     *  Assignment: Lab P-3.38                     Level:   4                *
     *                                                                       *
     *  Description:                                                         *
     *                                                                       *
        P-3.38: Maintains top ten scores of a game application. Implements
                add and remove methods of Section 3.1.1
                using a doubly linked list instead of a singly linked list.

     *                                                                       *
    *************************************************************************/

l_ladios@ares:~$ cat GameEntry.java
// modified implementation from Java book
public class GameEntry {

        private String name;
        private int score;

        public GameEntry(String name, int score) {
                this.name = name;
                this.score = score;
        }

        public String getName() {
                return name;
        }

        public int getScore() {
                return score;
        }

        public void setScore(int newScore) {
                score = newScore;
        }

        public int equals(GameEntry sec) {
                if(this.getName().equals(sec.getName()) && this.getScore()
                        == sec.getScore()) {
                        return 1;
                }
                else {
                        return 0;
                }
        }
        public String toString() {
                return "Name: " + name + "    Score: " + score;
        }
}
```

```
l_ladios@ares:~$ javac GameEntry.java
l_ladios@ares:~$ cat DoublyLinkedList.java
import java.lang.ArrayIndexOutOfBoundsException;


//Modified Java book's implementation of a Singly linked list.
public class DoublyLinkedList<T extends GameEntry> {

        //nested Node<T> class
        private static class Node<T> {
                //private instance variables
                private T data;
                private Node<T> next;
                private Node<T> prev;


                //constructor Node class
                public Node(T data, Node<T> next, Node<T> prev) {
                        this.data = data;
                        this.next = next;
                        this.prev = prev;
                }

                /*
                 * access method
                 * @returns data - data inside a Node.
                 */
                public T getData() {
                        return data;
                }

                /*
                 * access method
                 * @returns next - reference to the next Node
                 */
                public Node<T> getNext() {
                        return next;
                }

                public Node<T> getPrev(){
                   return prev;
                }

                /*
                 * setter method
                 * @param newNext - new next value being assigned
                 */
                public void setNext(Node<T> newNext) {
                        this.next = newNext;
                }

                public void setPrev(Node<T> newPrev) {
                   this.prev = newPrev;
```

```
        }
}

//private instance variables
private Node<T> header;
private Node<T> trailer;
private int size;

//constructor
public DoublyLinkedList() {
        header = new Node<T>(null, null, null);
        trailer = new Node<T>(null, null, header);
        size = 0;
}

public T get(int index) throws ArrayIndexOutOfBoundsException {
    if(index < 0 || size <= index)
            throw new ArrayIndexOutOfBoundsException(index);

    Node<T> node = header;
    while(node.next != null && index > 0) {
        node = node.next;
        index--;
    }

     return node.getData();
}


/*
 * access method
 * @return size - size of linked list
 */
public int size() {
        return size;
}

/*
 * @return T/F - true if empty, false if filed.
 */
public boolean isEmpty() {
        return size == 0;
}

/*
 * @return first Element of list
 */
public T first() {
   if(isEmpty())
        return null;
        return header.getNext().getData();
}

/*
         * @return last Element of list
         */
        public T last() {
            if(isEmpty())
               return null;
                return trailer.getPrev().getData();
        }

        /*
         * adds to the top of the list
         * @param data - data for Node to be inserted.
         */
        public void addFirst(T data) {
                addBetween(data, header.getNext(), header);
        }


   /*
    * adds to the end of the list
    * @param data - data for Node to be inserted.
    */
   public void addLast(T data) {
     addBetween(data, trailer, trailer.getPrev());
   }


   public void addBetween(T data, Node<T> nt, Node<T> pr) {

      Node<T> newest = new Node<T>(data, nt, pr);

      pr.setNext(newest);
      nt.setPrev(newest);
      newest.setNext(nt);
      size++;
   }


 /*
 * add score into list only if it is higher than a score already on
 * the board.
 */
public void addEntry(T entry) {


   if(size <= 2 ) {
      if(isEmpty() || entry.getScore() > header.getNext().getData().getScore()) {
         addLast(entry);
      }
      else if(isEmpty()) {
         addFirst(entry);
      }
      else if(entry.getScore() > header.getNext().getData().getScore()
            && entry.getScore() < trailer.getPrev().getData().getScore()) {
         addBetween(entry, trailer, header);
```

```
      }
   }
   else {
     boolean added = false;
     Node<T> current = header.getNext();


     while(current.getNext() != trailer && !added) {
       if(current.getData().getScore() <= entry.getScore()
          && current.getNext().getData().getScore() > entry.getScore()) {
          //enter in between current and current.getNext()
          addBetween(entry, current.getNext(), current);
          added = true;
       }
       else if(entry.getScore() < first().getScore()) {
          addFirst(entry);
          added = true;
       }
       else {
          current = current.getNext();
       }
     }
     if(!added) {
        addLast(entry);
        added = true;
     }
     if(size > 10) {
        removeFirst();
     }


   }

}
       /*
        * removes the first Node on the list.
        * @return ret - data of Node being removed.
        */
   public T removeFirst() {
        if(isEmpty())
                  return null;
        Node<T> first = header.getNext();
        T ret = first.getData();
        header.setNext(first.getNext());
        size--;
        return ret;

   }

   public T removeLast() {
         if(isEmpty())
              return null;
         Node<T> last = trailer.getPrev();
         T ret = last.getData();
         trailer.setPrev(last.getPrev());
         size--;
         return ret;
   }


        public void remove(int index) {

                int i = 0;
                Node<T> current = header.getNext();
                boolean removed = false;

                if(isEmpty()) {
                        System.out.println("There is nothing to remove");
                }
                if(index == 1) {
                        removeFirst();
                }
                else {
                        while(current.getNext() != trailer && !removed) {
                                if(i == (index-1)) {
                                        current.getPrev().setNext(current.getNext(
                                        current.getNext().setPrev(current.getPrev(
                                        size--;

                                        removed = true;
                                }
                                else{
                                        current = current.getNext();
                                        i++;
                                }
                        }

                        if(index == size) {
                           current.getPrev().setNext(trailer);
                           trailer.setPrev(current.getPrev());
                                        size--;
                        }
                }
        }

        public String toString() {
                int i = 0;

                String build = "";

                Node<T> current = header.getNext();
                while(i < size && current.getNext() != null) {
                   if(size > 2) {
                        build += (i+1) + ") " + current.getData().toString()
                              + "\n\n";
                        current = current.getNext();
                        i++;
```

```
                }

                else{
                   build += (i+1) + ") " + current.getData().toString()
                         + "\t\t null \n\n";
            current = current.getNext();
            i++;
                   }

               }
               return build;
        }
}


l_ladios@ares:~$ javac DoublyLinkedList.java
cal_ladios@ares:~$ cat PlayerListMain.java
import java.util.Scanner;
public class PlayerListMain {

public static void main(String[] args) {
        // TODO Auto-generated method stub
        Scanner in = new Scanner(System.in);

        DoublyLinkedList<GameEntry> scoresList = new DoublyLinkedList<GameEntry>()


        scoresList.addEntry(new GameEntry("Carl", 78));
        scoresList.addEntry(new GameEntry("James", 80));
        scoresList.addEntry(new GameEntry("Marjorie", 92));
        scoresList.addEntry(new GameEntry("Mary", 90));
        scoresList.addEntry(new GameEntry("Kami", 121));
        scoresList.addEntry(new GameEntry("Clarie", 150));
        scoresList.addEntry(new GameEntry("Maybell", 139));
        scoresList.addEntry(new GameEntry("Timmy", 40));


        boolean goAgain = true;
        while(goAgain) {
                boolean done = false;
                System.out.println("\nWould you like to\n 1)add \n 2)remove a score
                                "\t(Please enter 1 or 2)");
                int choice = in.nextInt();

                while(!done) {
                        if(choice == 1) {
                                System.out.println("\nPlease enter a score");

                                System.out.print("Enter Score: ");
                                int score = in.nextInt();
                                System.out.print("Enter Name: ");
                                String name = in.next();
                                scoresList.addEntry(new GameEntry(name, score));
                                System.out.println("\nWould you like to add another
                                String keepGoing= in.next();

                                if(keepGoing.equalsIgnoreCase("N")) {
                                        done = true;
                                }
                                else if(keepGoing.equalsIgnoreCase("Y")){
                                        done = false;
                                }

                        }
                        else if(choice == 2) {

                                System.out.println("\n" + scoresList);
                                System.out.println("\nWhich entry would you like to
                                                + "(Select a number)");

                                scoresList.remove(in.nextInt());

                                System.out.println("\nWould you like to remove anot
                                String keepGoing= in.next();

                                if(keepGoing.equalsIgnoreCase("N")) {
                                        done = true;
                                        System.out.println(scoresList);
                                }
                                else if(keepGoing.equalsIgnoreCase("Y")){
                                        done = false;
                                }

                        }

                }
                System.out.println("\nDo you want to \n1)remove/add another score \
                                "\t(Please enter 1 or 2)");

                int toGoAgain = in.nextInt();

                if(toGoAgain == 2) {
                        goAgain = false;
                }
        }

        System.out.println(scoresList);

        }

}

l_ladios@ares:~$ javac PlayerListMain.java
l_ladios@ares:~$ java PlayerListMain

Would you like to
 1)add
```

```
 2)remove a score?      (Please enter 1 or 2)
1

Please enter a score
Enter Score: 80
Enter Name: Leia

Would you like to add another score? (Y/N)
Y

Please enter a score
Enter Score: 150
Enter Name: Hellen

Would you like to add another score? (Y/N)
Y

Please enter a score
Enter Score: 300
Enter Name: Cole

Would you like to add another score? (Y/N)
N

Do you want to
1)remove/add another score
2)exit ?        (Please enter 1 or 2)
1

Would you like to
 1)add
 2)remove a score?      (Please enter 1 or 2)
2

1) Name: Carl    Score: 78

2) Name: James    Score: 80

3) Name: Leia    Score: 80

4) Name: Mary    Score: 90

5) Name: Marjorie    Score: 92

6) Name: Kami    Score: 121

7) Name: Maybell    Score: 139

8) Name: Clarie    Score: 150

9) Name: Hellen    Score: 150

10) Name: Cole    Score: 300
```

```
Which entry would you like to remove?(Select a number)
1

Would you like to remove another score? (Y/N)
Y

1) Name: James    Score: 80

2) Name: Leia    Score: 80

3) Name: Mary    Score: 90

4) Name: Marjorie    Score: 92

5) Name: Kami    Score: 121

6) Name: Maybell    Score: 139

7) Name: Clarie    Score: 150

8) Name: Hellen    Score: 150

9) Name: Cole    Score: 300


Which entry would you like to remove?(Select a number)
5

Would you like to remove another score? (Y/N)
Y

1) Name: James    Score: 80

2) Name: Leia    Score: 80

3) Name: Mary    Score: 90

4) Name: Marjorie    Score: 92

5) Name: Maybell    Score: 139

6) Name: Clarie    Score: 150

7) Name: Hellen    Score: 150

8) Name: Cole    Score: 300


Which entry would you like to remove?(Select a number)
8
```

```
Would you like to remove another score? (Y/N)
N
1) Name: James    Score: 80

2) Name: Leia    Score: 80

3) Name: Mary    Score: 90

4) Name: Marjorie    Score: 92

5) Name: Maybell    Score: 139

6) Name: Clarie    Score: 150

7) Name: Hellen    Score: 150



Do you want to
1)remove/add another score
2)exit ?        (Please enter 1 or 2)
2
1) Name: James    Score: 80

2) Name: Leia    Score: 80

3) Name: Mary    Score: 90

4) Name: Marjorie    Score: 92

5) Name: Maybell    Score: 139

6) Name: Clarie    Score: 150

7) Name: Hellen    Score: 150


l_ladios@ares:~$ exit
exit

Script done on 2021-02-24 21:18:32-0600
nner.java:2076)
        at PlayerListMain.main(PlayerListMain.java:55)
l_ladios@ares:~$ script
Script started, file is typescript
l_ladios@ares:~$ cat ScoreEntryLab.info
 /**************************************************************************
     *                                                                        *
     *  NAME:  Leia Ladios                        CLASS:  CSC121-W01          *
     *                                                                        *
     *  Assignment: Lab P-3.38                    Level:   4                  *
     *                                                                        *
     *  Description:                                                          *
     *                                                                        *
     *     P-3.38: Maintains top ten scores of a game application. Implements *
     *             add and remove methods of Section 3.1.1                    *
     *             using a doubly linked list instead of a singly linked list.*
     *                                                                        *
     **************************************************************************/

l_ladios@ares:~$ cat GameEntry.java
// modified implementation from Java book
public class GameEntry {

        private String name;
        private int score;

        public GameEntry(String name, int score) {
                this.name = name;
                this.score = score;
        }

        public String getName() {
                return name;
        }

        public int getScore() {
                return score;
        }

        public void setScore(int newScore) {
                score = newScore;
        }

        public int equals(GameEntry sec) {
                if(this.getName().equals(sec.getName()) && this.getScore()
                        == sec.getScore()) {
                        return 1;
                }
                else {
                        return 0;
                }
        }
        public String toString() {
                return "Name: " + name + "    Score: " + score;
        }
}

l_ladios@ares:~$ javac GameEntry.java
l_ladios@ares:~$ cat DoublyLinkedList.java
import java.lang.ArrayIndexOutOfBoundsException;


//Modified Java book's implementation of a Singly linked list.
public class DoublyLinkedList<T extends GameEntry> {
```

```
//nested Node<T> class
private static class Node<T> {
        //private instance variables
        private T data;
        private Node<T> next;
        private Node<T> prev;


        //constructor Node class
        public Node(T data, Node<T> next, Node<T> prev) {
                this.data = data;
                this.next = next;
                this.prev = prev;
        }

        /*
         * access method
         * @returns data - data inside a Node.
         */
        public T getData() {
                return data;
        }

        /*
         * access method
         * @returns next - reference to the next Node
         */
        public Node<T> getNext() {
                return next;
        }

        public Node<T> getPrev(){
           return prev;
        }

        /*
         * setter method
         * @param newNext - new next value being assigned
         */
        public void setNext(Node<T> newNext) {
                this.next = newNext;
        }

        public void setPrev(Node<T> newPrev) {
           this.prev = newPrev;
        }
}

//private instance variables
private Node<T> header;
private Node<T> trailer;
private int size;

//constructor
```

```
public DoublyLinkedList() {
        header = new Node<T>(null, null, null);
        trailer = new Node<T>(null, null, header);
        size = 0;
}

public T get(int index) throws ArrayIndexOutOfBoundsException {
    if(index < 0 || size <= index)
            throw new ArrayIndexOutOfBoundsException(index);

    Node<T> node = header;
    while(node.next != null && index > 0) {
        node = node.next;
        index--;
    }

     return node.getData();
}


/*
 * access method
 * @return size - size of linked list
 */
public int size() {
        return size;
}

/*
 * @return T/F - true if empty, false if filed.
 */
public boolean isEmpty() {
        return size == 0;
}

/*
 * @return first Element of list
 */
public T first() {
   if(isEmpty())
        return null;
        return header.getNext().getData();
}

/*
 * @return last Element of list
 */
public T last() {
    if(isEmpty())
       return null;
        return trailer.getPrev().getData();
}

/*
```

```
         * adds to the top of the list
         * @param data - data for Node to be inserted.
         */
        public void addFirst(T data) {
                addBetween(data, header.getNext(), header);
        }


   /*
    * adds to the end of the list
    * @param data - data for Node to be inserted.
    */
   public void addLast(T data) {
     addBetween(data, trailer, trailer.getPrev());
   }


   public void addBetween(T data, Node<T> nt, Node<T> pr) {

      Node<T> newest = new Node<T>(data, nt, pr);

      pr.setNext(newest);
      nt.setPrev(newest);
      newest.setNext(nt);
      size++;
   }


 /*
 * add score into list only if it is higher than a score already on
 * the board.
 */
public void addEntry(T entry) {


   if(size <= 2 ) {
      if(isEmpty() || entry.getScore() > header.getNext().getData().getScore()) {
         addLast(entry);
      }
      else if(isEmpty()) {
         addFirst(entry);
      }
      else if(entry.getScore() > header.getNext().getData().getScore()
            && entry.getScore() < trailer.getPrev().getData().getScore()) {
         addBetween(entry, trailer, header);
      }
   }
   else {
     boolean added = false;
     Node<T> current = header.getNext();


     while(current.getNext() != trailer && !added) {
       if(current.getData().getScore() <= entry.getScore()
            && current.getNext().getData().getScore() > entry.getScore()) {
            //enter in between current and current.getNext()
            addBetween(entry, current.getNext(), current);
            added = true;
       }
       else if(entry.getScore() < first().getScore()) {
          addFirst(entry);
          added = true;
       }
       else {
          current = current.getNext();
       }
     }
     if(!added) {
        addLast(entry);
        added = true;
     }
     if(size > 10) {
        removeFirst();
     }


   }

}
        /*
         * removes the first Node on the list.
         * @return ret - data of Node being removed.
         */
   public T removeFirst() {
        if(isEmpty())
                  return null;
        Node<T> first = header.getNext();
        T ret = first.getData();
        header.setNext(first.getNext());
        size--;
        return ret;

   }

   public T removeLast() {
         if(isEmpty())
              return null;
         Node<T> last = trailer.getPrev();
         T ret = last.getData();
         trailer.setPrev(last.getPrev());
         size--;
         return ret;
   }


        public void remove(int index) {

                int i = 0;
```

```
        Node<T> current = header.getNext();
        boolean removed = false;

        if(isEmpty()) {
                System.out.println("There is nothing to remove");
        }
        if(index == 1) {
                removeFirst();
        }
        else {
                while(current.getNext() != trailer && !removed) {
                        if(i == (index-1)) {
                                current.getPrev().setNext(current.getNext()
                                current.getNext().setPrev(current.getPrev()
                                size--;

                                removed = true;
                        }
                        else{
                                current = current.getNext();
                                i++;
                        }
                }

                if(index == size) {
                   current.getPrev().setNext(trailer);
                   trailer.setPrev(current.getPrev());
                                size--;
                }
        }
}

public String toString() {
        int i = 0;

        String build = "";

        Node<T> current = header.getNext();
        while(i < size && current.getNext() != null) {
           if(size > 2) {
                build += (i+1) + ") " + current.getData().toString()
                      + "\n\n";
                current = current.getNext();
                i++;

           }

           else{
              build += (i+1) + ") " + current.getData().toString()
                    + "\t\t null \n\n";
    current = current.getNext();
    i++;
           }
                }
                return build;
        }
}
```

```
l_ladios@ares:~$ javac DoublyLinkedList.java
cal_ladios@ares:~$ cat PlayerListMain.java
import java.util.Scanner;
public class PlayerListMain {

public static void main(String[] args) {
        // TODO Auto-generated method stub
        Scanner in = new Scanner(System.in);

        DoublyLinkedList<GameEntry> scoresList = new DoublyLinkedList<GameEntry>()


        scoresList.addEntry(new GameEntry("Carl", 78));
        scoresList.addEntry(new GameEntry("James", 80));
        scoresList.addEntry(new GameEntry("Marjorie", 92));
        scoresList.addEntry(new GameEntry("Mary", 90));
        scoresList.addEntry(new GameEntry("Kami", 121));
        scoresList.addEntry(new GameEntry("Clarie", 150));
        scoresList.addEntry(new GameEntry("Maybell", 139));
        scoresList.addEntry(new GameEntry("Timmy", 40));


        boolean goAgain = true;
        while(goAgain) {
                boolean done = false;
                System.out.println("\nWould you like to\n 1)add \n 2)remove a score
                                "\t(Please enter 1 or 2)");
                int choice = in.nextInt();

                while(!done) {
                        if(choice == 1) {
                                System.out.println("\nPlease enter a score");

                                System.out.print("Enter Score: ");
                                int score = in.nextInt();
                                System.out.print("Enter Name: ");
                                String name = in.next();
                                scoresList.addEntry(new GameEntry(name, score));

                                System.out.println("\nWould you like to add another
                                String keepGoing= in.next();

                                if(keepGoing.equalsIgnoreCase("N")) {
                                        done = true;
                                }
                                else if(keepGoing.equalsIgnoreCase("Y")){
                                        done = false;
                                }
```

```
                        }
                        else if(choice == 2) {

                                System.out.println("\n" + scoresList);
                                System.out.println("\nWhich entry would you like to
                                                + "(Select a number)");

                                scoresList.remove(in.nextInt());

                                System.out.println("\nWould you like to remove ano
                                String keepGoing= in.next();

                                if(keepGoing.equalsIgnoreCase("N")) {
                                        done = true;
                                        System.out.println(scoresList);
                                }
                                else if(keepGoing.equalsIgnoreCase("Y")){
                                        done = false;
                                }

                        }

                }
                System.out.println("\nDo you want to \n1)remove/add another score
                                "\t(Please enter 1 or 2)");

                int toGoAgain = in.nextInt();

                if(toGoAgain == 2) {
                        goAgain = false;
                }
        }

        System.out.println(scoresList);

        }

}

l_ladios@ares:~$ javac PlayerListMain.java
l_ladios@ares:~$ java PlayerListMain

Would you like to
 1)add
 2)remove a score?      (Please enter 1 or 2)
1

Please enter a score
Enter Score: 80
Enter Name: Leia

Would you like to add another score? (Y/N)
Y

Please enter a score
Enter Score: 150
Enter Name: Hellen

Would you like to add another score? (Y/N)
Y

Please enter a score
Enter Score: 300
Enter Name: Cole

Would you like to add another score? (Y/N)
N

Do you want to
1)remove/add another score
2)exit ?        (Please enter 1 or 2)
1

Would you like to
 1)add
 2)remove a score?      (Please enter 1 or 2)
2

1) Name: Carl    Score: 78

2) Name: James    Score: 80

3) Name: Leia    Score: 80

4) Name: Mary    Score: 90

5) Name: Marjorie    Score: 92

6) Name: Kami    Score: 121

7) Name: Maybell    Score: 139

8) Name: Clarie    Score: 150

9) Name: Hellen    Score: 150

10) Name: Cole    Score: 300



Which entry would you like to remove?(Select a number)
1

Would you like to remove another score? (Y/N)
Y

1) Name: James    Score: 80
```

```
2) Name: Leia    Score: 80

3) Name: Mary    Score: 90

4) Name: Marjorie    Score: 92

5) Name: Kami    Score: 121

6) Name: Maybell    Score: 139

7) Name: Clarie    Score: 150

8) Name: Hellen    Score: 150

9) Name: Cole    Score: 300


Which entry would you like to remove?(Select a number)
5

Woul
```