DSA ASSIGNMENT -1 REPORT

on

CAR AUDIO BLUETOOTH SYSTEM

IN JAVA

DONE BY

SNEHALAKSHMI S

CH.EN.U4CCE21057

for

DATA STRUCTURES AND ALGORITHMS LAB – 19CCE202

AMRITA SCHOOL OF ENGINEERING – CHENNAI

5th December , 2022

TABLE OF CONTENTS

ABSTRACT

To Reverse engineer an application used in a car audio player. This application helps you connect your mobile phones / Bluetooth enabled device to play audio files / dial or receive calls etc. A maximum of 5 devices can be connected. If you want to include a new device, you have to first delete one of the existing devices and insert the new one.

Also to define a solution for the following condition,

 # 5 devices are trying to connect to the application in the same instance. Only one device can be connected at a time. Implement the above scenario using various test cases.

APPROACH AND ALGORITHM

We try using Stack and Queue data structure to built a Bluetooth system in car audio player to connect any device from multiple devices.

Here for pairing of various devices we implement queue using linked list in such a way that: the first paired device will be connected if there are no previously connected devices .

For connecting devices, all the previously connected devices are stored in a stack in such a way that: The lastly connected device is preferred during new connection.

Here linked list (only singly linked list for both pairing and connecting) is used for memory management and faster processing.

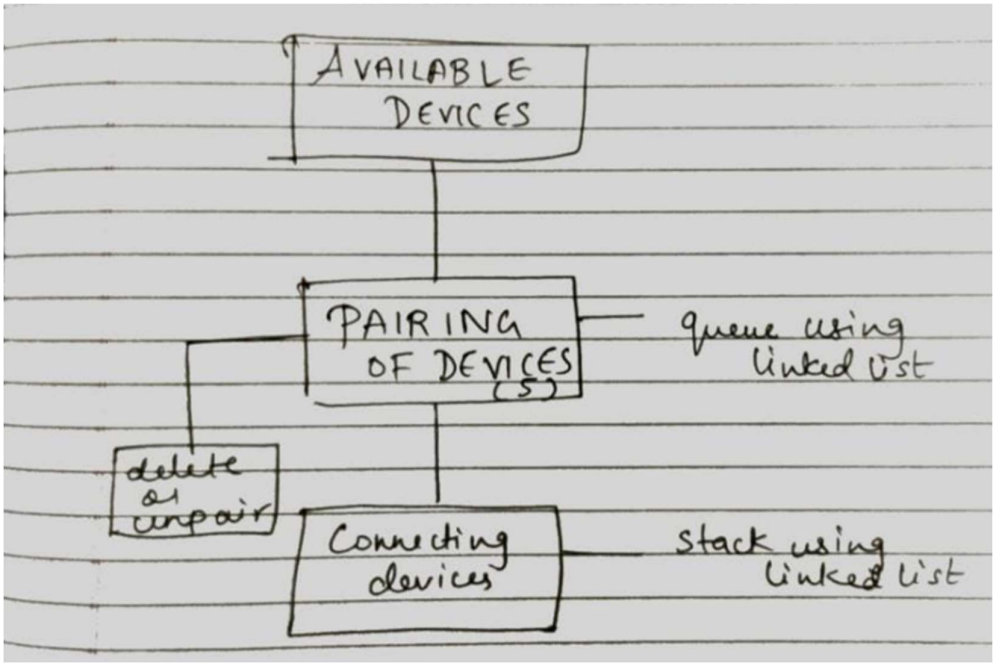
ALGORITHM:-

1. A menu driven program that runs continuously asking to perform any desired operations below:
    i. To view available devices and paired devices
    ii. To pair any new device (maximum is 5)
    iii. To connect any 1 device from paired devices
    iv. To remove any paired device to pair a new device

    Each of the operations has an separate algorithm or method that uses different data structures
2. Available devices are already given or assigned and any 1 device can be paired at a time, however multiple loops will facilitate to pair more devices consecutively (only 5 paired devices, i.e queue.length do not exceed 5). The paired devices which are stored in a queue as and when you insert a new device to pair is saved and displayed when required by the user.

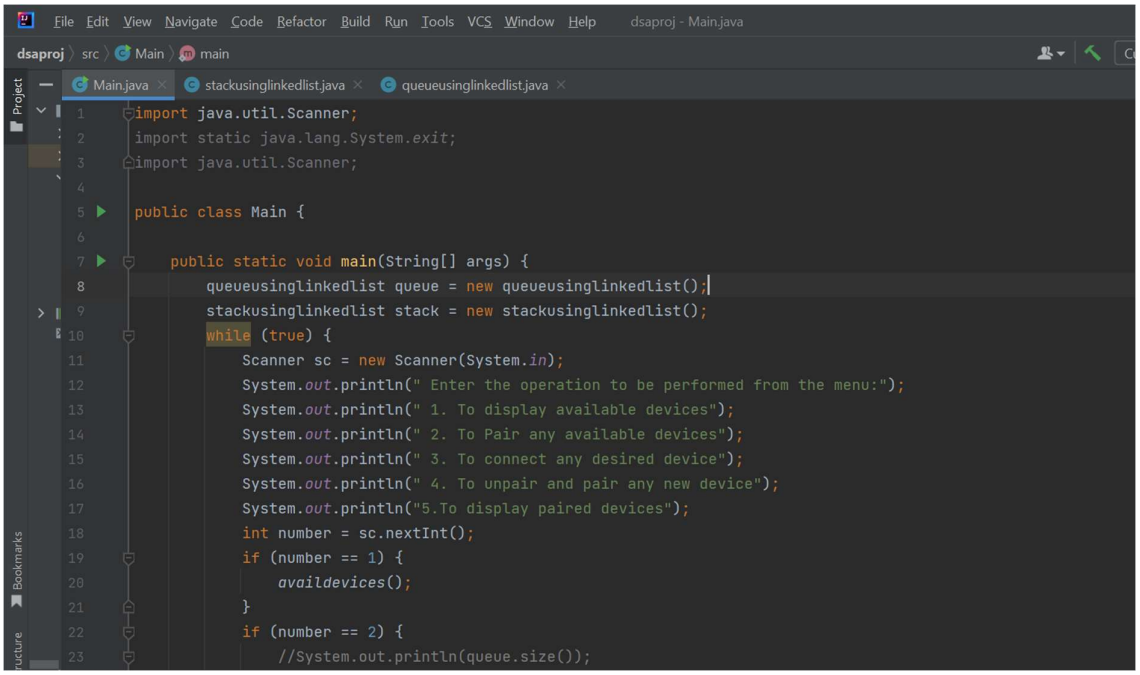
3. All the previously connected devices are stored in a stack and when user requires operation of connecting a device from paired;
   - ➢ it checks for stack elements and displays the recently connected device using peek() function in stack
   - ➢ if stack is empty then the first device or front element in queue is connected and the new stack is updated with that device.

4. In order to remove any device and pair a new device we again update the queue by removing the front element and adding a new device at the rear end
   Now this automatically happens while pairing of devices, if paired devices are more than 5 , then the system automatically removes the front device from queue and the new device is added at the rear end of the queue.



JAVA CODE

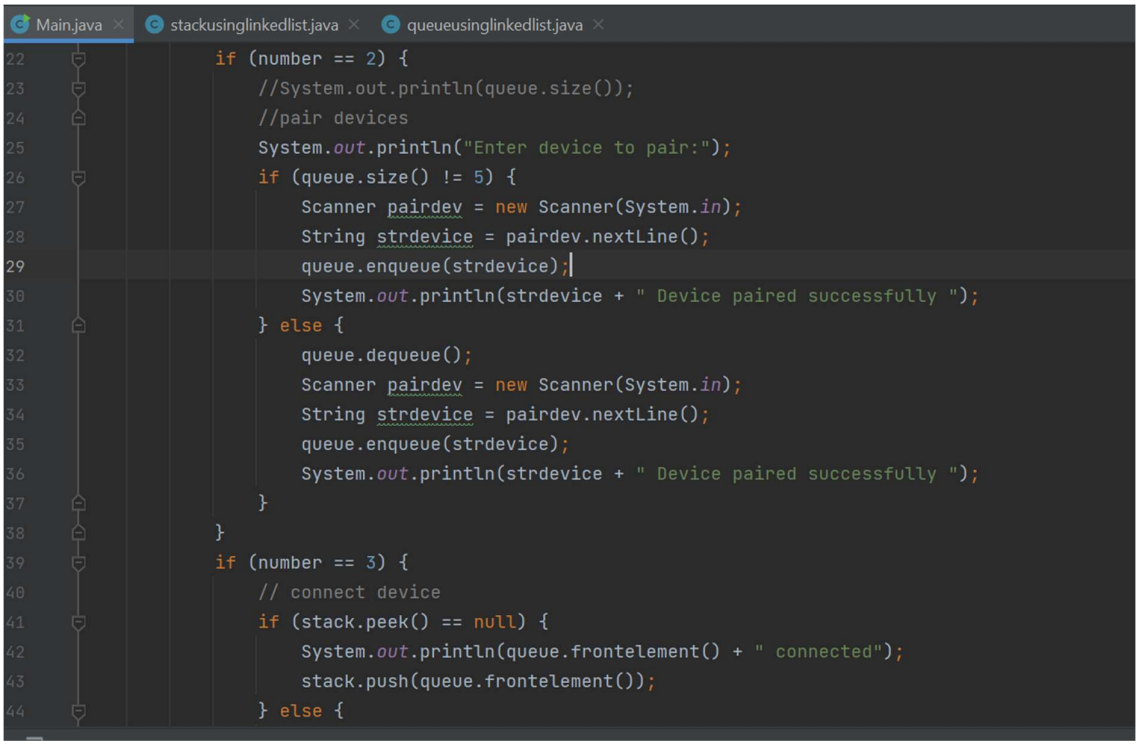
MAIN CLASS:-



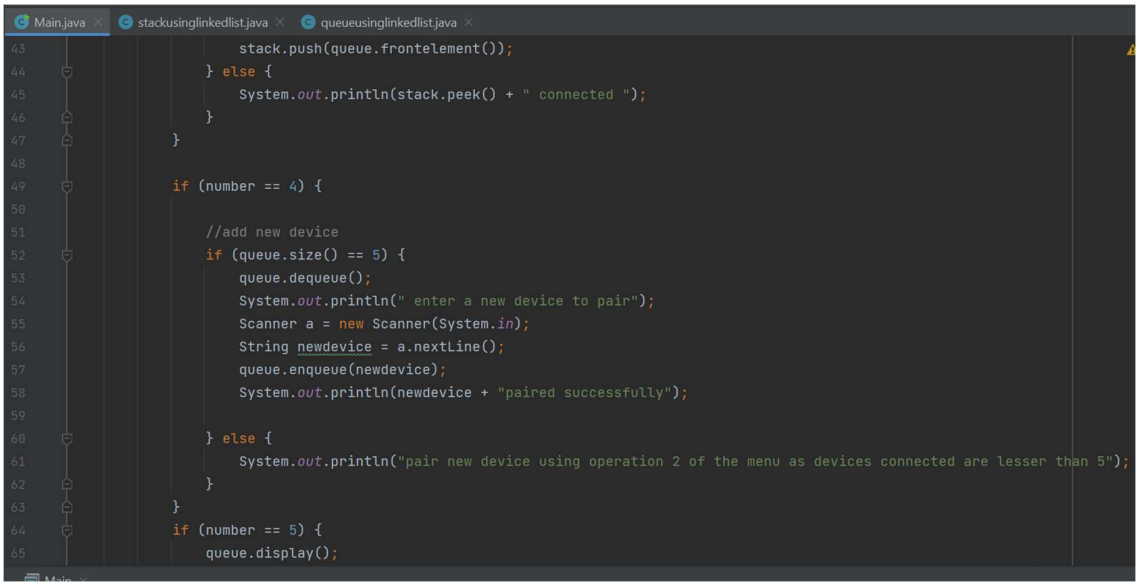
```java
import java.util.Scanner;
import static java.lang.System.exit;
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        queueusinglinkedlist queue = new queueusinglinkedlist();
        stackusinglinkedlist stack = new stackusinglinkedlist();
        while (true) {
            Scanner sc = new Scanner(System.in);
            System.out.println(" Enter the operation to be performed from the menu:");
            System.out.println(" 1. To display available devices");
            System.out.println(" 2. To Pair any available devices");
            System.out.println(" 3. To connect any desired device");
            System.out.println(" 4. To unpair and pair any new device");
            System.out.println("5.To display paired devices");
            int number = sc.nextInt();
            if (number == 1) {
                availdevices();
            }
            if (number == 2) {
                //System.out.println(queue.size());
```



```java
            if (number == 2) {
                //System.out.println(queue.size());
                //pair devices
                System.out.println("Enter device to pair:");
                if (queue.size() != 5) {
                    Scanner pairdev = new Scanner(System.in);
                    String strdevice = pairdev.nextLine();
                    queue.enqueue(strdevice);
                    System.out.println(strdevice + " Device paired successfully ");
                } else {
                    queue.dequeue();
                    Scanner pairdev = new Scanner(System.in);
                    String strdevice = pairdev.nextLine();
                    queue.enqueue(strdevice);
                    System.out.println(strdevice + " Device paired successfully ");
                }
            }
            if (number == 3) {
                // connect device
                if (stack.peek() == null) {
                    System.out.println(queue.frontelement() + " connected");
                    stack.push(queue.frontelement());
                } else {
```



FUNCTION CLASS:-

1. **stackusinglinkeddlist**

Main.java | stackusinglinkedlist.java | queueusinglinkedlist.java

```java
import static java.lang.System.exit;

2 usages
public class stackusinglinkedlist {
    // A linked list node
    4 usages
    private class Node {

        2 usages
        String data; // integer data
        2 usages
        Node link; // reference variable Node type
    }

    // create global top reference variable global
    8 usages
    Node top;

    // Constructor
    1 usage
    stackusinglinkedlist() {
        this.top = null;
    }
```

```java

    // Utility function to add an element x in the stack
    1 usage
    public void push(String x) // insert at the beginning
    {
        // create new node temp and allocate memory
        Node temp = new Node();

        // check if stack (heap) is full. Then inserting an
        //  element would lead to stack overflow
        if (temp == null) {
            System.out.print("\nHeap Overflow");
            return;
        }

        // initialize data into temp data field
        temp.data = x;

        // put top reference into temp link
        temp.link = top;

```

Main.java | stackusinglinkedlist.java | queueusinglinkedlist.java

```
37
38          // update top reference
39          top = temp;
40      }
41
42      // Utility function to check if the stack is empty or
43      // not
        1 usage
44      public boolean isEmpty() {
45          return top == null;
46      }
47
48      // Utility function to return top element in a stack
        2 usages
49      public String peek() {
50          // check for empty stack
51          if (!isEmpty()) {
52              return top.data;
53          } else {
54              //System.out.println("Stack is empty");
55              return null;
56          }
57      }
```

Main.java | stackusinglinkedlist.java | queueusinglinkedlist.java

```
53          } else {
54              //System.out.println("Stack is empty");
55              return null;
56          }
57      }
58
59      // Utility function to pop top element from the stack
60      public void pop() // remove at the beginning
61      {
62          // check for stack underflow
63          if (top == null) {
64              //System.out.print("\nStack Underflow");
65              return;
66          }
67
68          // update the top pointer to point to the next node
69          top = (top).link;
70      }
71  }
72
73
```

2. queueusinglinkedlist

Main.java | stackusinglinkedlist.java | queueusinglinkedlist.java

```java
import java.util.*;
2 usages
public class queueusinglinkedlist {
    3 usages
    int count = 0;

    // A linked list (LL) node to store a queue entry
    6 usages
    class QNode {
        3 usages
        String key;
        4 usages
        QNode next;

        // constructor to create a new linked list node
        1 usage
        public QNode(String key) {
            this.key = key;
            this.next = null;
        }
    }
```

Main.java | stackusinglinkedlist.java | queueusinglinkedlist.java

```java
    10 usages
    QNode front, rear;

    public void Queue() { this.front = this.rear = null; }

    // Method to add an key to the queue.
    3 usages
    void enqueue(String key) {
        count++;

        // Create a new LL node
        QNode temp = new QNode(key);

        // If queue is empty, then new node is front and
        // rear both
        if (this.rear == null) {
            this.front = this.rear = temp;
            return;
        }

```

Main.java | stackusinglinkedlist.java | queueusinglinkedlist.java

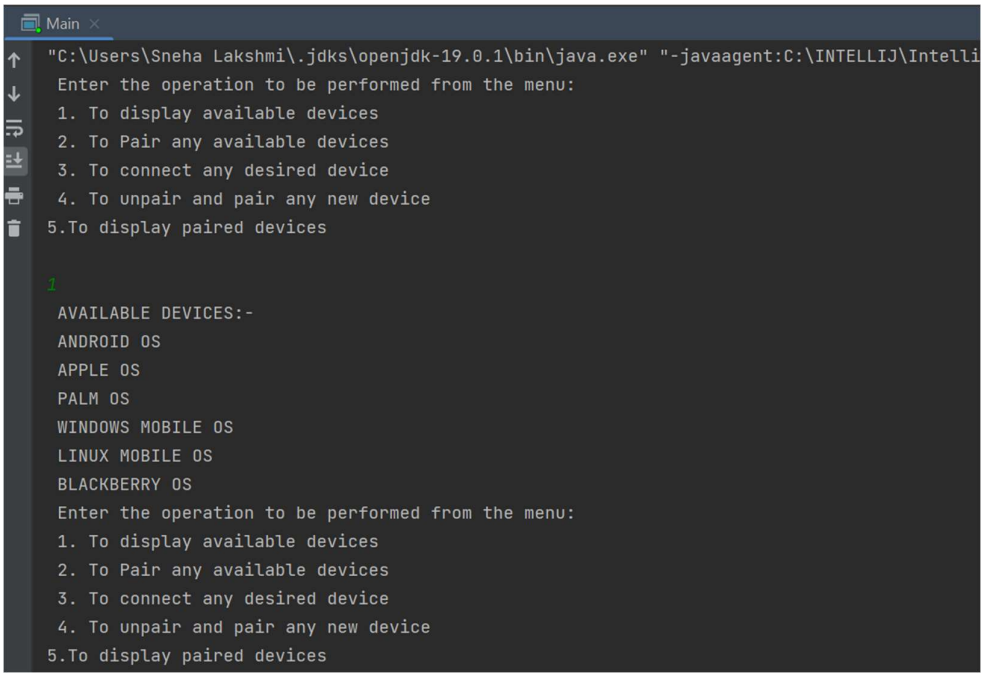
```
37
38          // Add the new node at the end of queue and change
39          // rear
40          this.rear.next = temp;
41          this.rear = temp;
42      }
43
44      // Method to remove an key from queue.
        2 usages
45      void dequeue() {
46          // If queue is empty, return NULL.
47          if (this.front == null)
48              return;
49
50          // Store previous front and move front one node
51          // ahead
52          count--;
53          QNode temp = this.front;
54          this.front = this.front.next;
55
```

```
54          this.front = this.front.next;
55
56          // If front becomes NULL, then change rear also as
57          // NULL
58          if (this.front == null)
59              this.rear = null;
60      }
61
        2 usages
62      public int size() {
63          return count;
64      }
65
        2 usages
66      public String frontelement() {
67          return front.key;
68      }
69
```

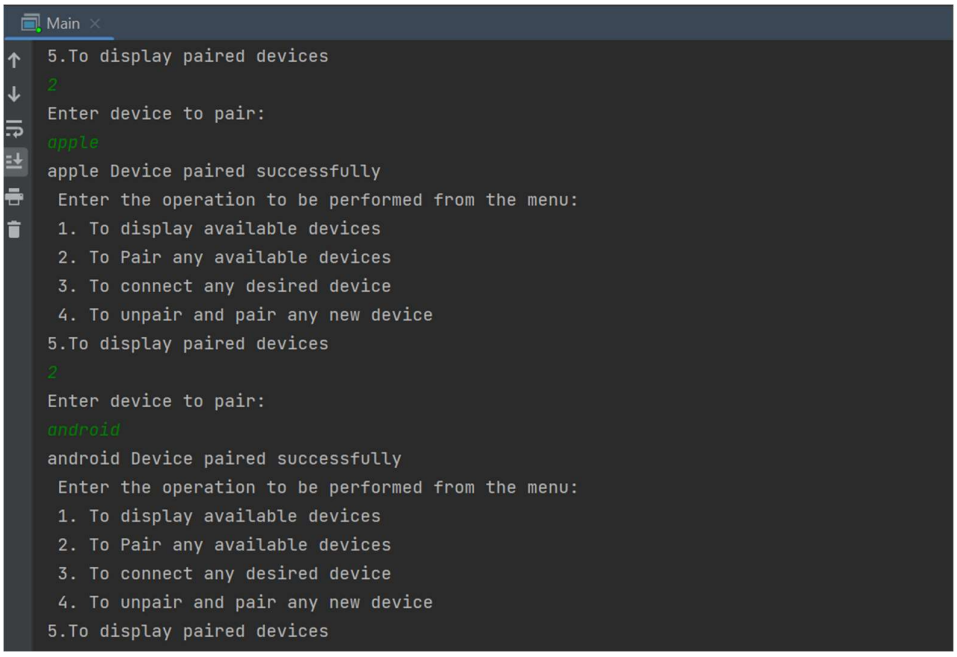

Main.java
stackusinglinkedlist.java
queueusinglinkedlist.java
1 usage
public void display() {
if (this.front == null) {
System.out.println(" No paired devices");
} else {
System.out.println(" PAIRED DEVICES ARE :");
QNode data = front;
while (data != null) {
System.out.println(data.key);
data = data.next;
}
}
}
}

OUTPUT

// operation 1 is selected and available devices are displayed



//operation 2 is selected to pair a device

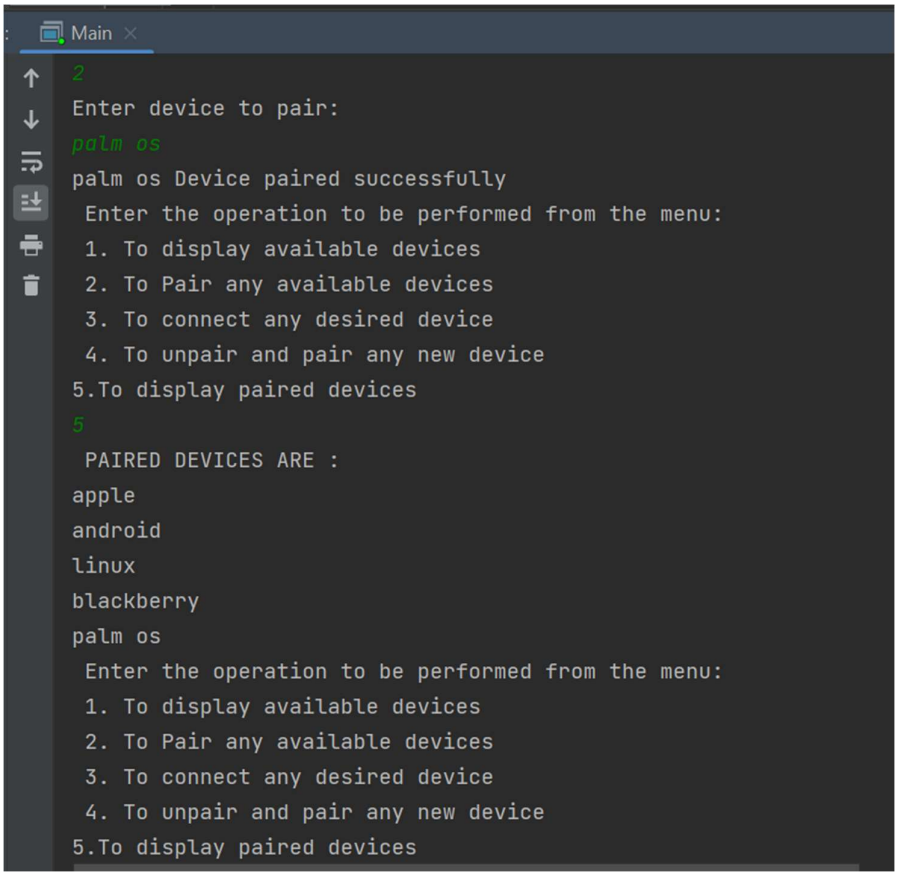


// as there are no earlier devices connected the first device from paired device is connected

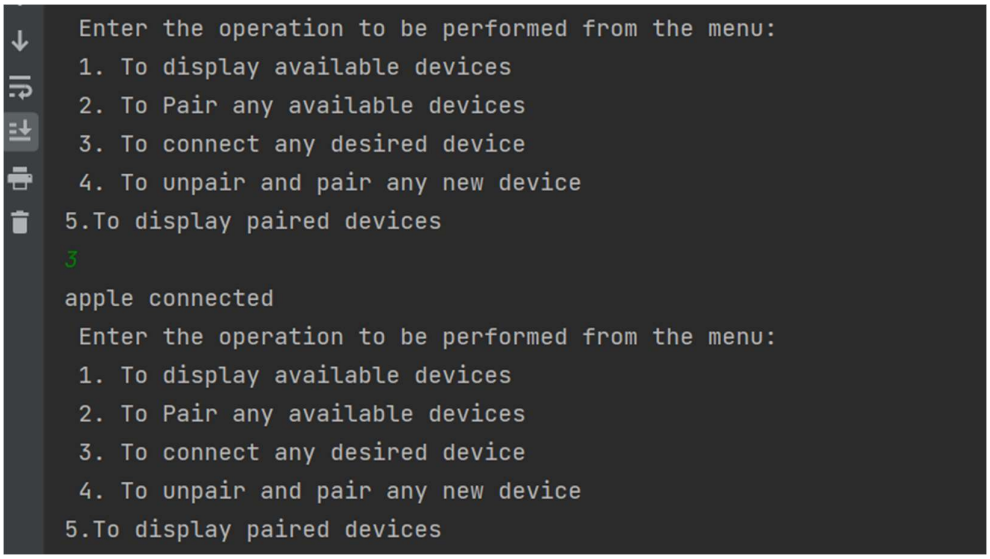
```
 4. To unpair and pair any new device
5.To display paired devices
3
apple connected
 Enter the operation to be performed from the menu:
 1. To display available devices
 2. To Pair any available devices
 3. To connect any desired device
 4. To unpair and pair any new device
5.To display paired devices
5
 PAIRED DEVICES ARE :
apple
android
 Enter the operation to be performed from the menu:
 1. To display available devices
 2. To Pair any available devices
 3. To connect any desired device
 4. To unpair and pair any new device
5.To display paired devices
```

```
5.To display paired devices
2
Enter device to pair:
linux
linux Device paired successfully
 Enter the operation to be performed from the menu:
 1. To display available devices
 2. To Pair any available devices
 3. To connect any desired device
 4. To unpair and pair any new device
5.To display paired devices
2
Enter device to pair:
blackberry
blackberry Device paired successfully
 Enter the operation to be performed from the menu:
 1. To display available devices
 2. To Pair any available devices
 3. To connect any desired device
 4. To unpair and pair any new device
5.To display paired devices
```

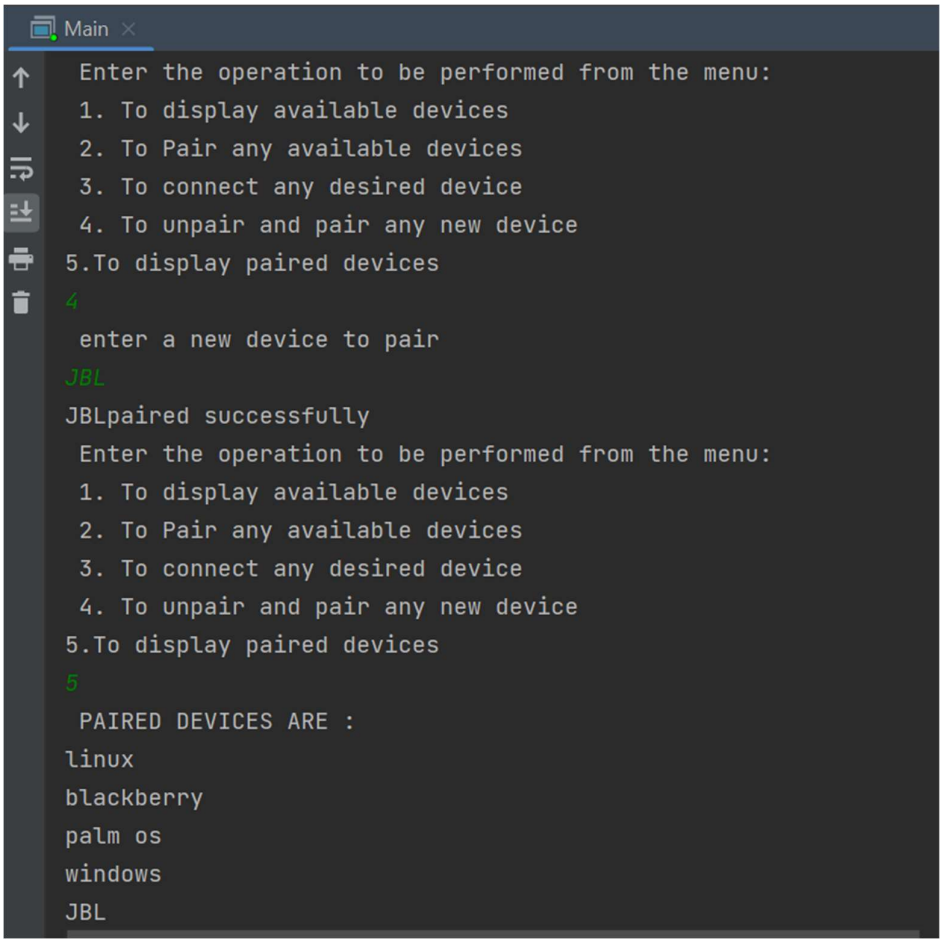
// 5 devices are paired



// since apple is previously connected , top element of stack , it is connected while trying to connect a device



// When a new device is trying to pair , the front element from queue or the firstly paired device is removed and new device is added at the rear end of the queue



INFERENCE

I have developed a car audio Bluetooth system that uses mainly 2 data structures: Stack and queue with singly linked list implementation.

Implementing stack using linked list have reduced the Time Complexity as O(1), for all push(), pop(), and peek(), as we are not performing any kind of traversal over the list. We perform all the operations through the current pointer only.

Also implementing queue using linked list have reduced the Time Complexity as O(1), The time complexity of both operations enqueue() and dequeue() is O(1) as it only changes a few pointers in both operations.

As per the algorithm, devices are connected to the Bluetooth system of the car audio player and also allows multiple operations to be performed at the same time , to restart the system program is to be terminated and all the connected or paired devices information will be lost and the system is implemented from first.