**Assingment-4**

**Team No: 5**

**Problem Statement:**

**Develop a program for Raymond’s Algorithm.**

**Code with comment-lines:**

**Node.java:**

package Assignment4;

import java.util.ArrayList;

// This class represents a node in a network

public class Node {

// This is the name of the node

String node\_name;

// This is a list of nodes that have requested access to the critical section

ArrayList<Node> requestQueue = new ArrayList<Node>();

// This is the parent node of the current node

Node Parent;

// This boolean indicates whether the node currently holds the token

boolean hasToken = false;

// This boolean indicates whether the node has requested access to the critical section

boolean requestForCs = false;

// Getter method for requestForCs

public boolean isRequestForCs() {

return requestForCs;

}

// Setter method for requestForCs

public void setRequestForCs(boolean requestForCs) {

this.requestForCs = requestForCs;

}

// Getter method for node\_name

public String getNode\_name() {

return node\_name;

}

// Setter method for node\_name

public void setNode\_name(String node\_name) {

this.node\_name = node\_name;

}

// Getter method for the first node in the requestQueue

public Node getRequestQueue() {

return requestQueue.get(0);

}

// Method to add a node to the requestQueue

public void setRequestQueue(Node rq) {

this.requestQueue.add(rq);

}

// Getter method for the parent node

public Node getParent() {

return Parent;

}

// Getter method for hasToken

public boolean isHasToken() {

return hasToken;

}

// Setter method for hasToken

public void setHasToken(boolean hasToken) {

this.hasToken = hasToken;

}

// Setter method for the parent node

public void setParent(Node parent) {

Parent = parent;

}

// Method to check if a given node is in the requestQueue

public boolean hasRequest(Node child) {

boolean flag = false;

if (this.requestQueue != null) {

for (Node i : this.requestQueue) {

if (i == child) {

flag = true;

break;

}

}

}

return flag;

}

}

**Main.java:**

**package Assignment4;**

**import java.io.BufferedReader;**

**import java.io.FileReader;**

**import java.io.IOException;**

**import java.util.ArrayList;**

**import java.util.Arrays;**

**public class Main {**

**static int flag=0;**

**//recieve confirmation of request by Token holder**

**public static void receiveTokenRequest(Node child) {**

**if (child.getParent().isHasToken()) {**

**child.getParent().setRequestQueue(child);**

**System.out.println("Token request by "+child.getNode\_name()+" is accepted by "+child.getParent().getNode\_name());**

**} else {**

**forwardTokenRequest(child);**

**}**

**}**

**//forwarding request to the token holder**

**private static void forwardTokenRequest(Node child) {**

**if ((child.getParent() != null) && (!child.getParent().hasRequest(child))) {**

**child.getParent().setRequestQueue(child);**

**System.out.println("Token request by "+child.getNode\_name()+" is forwarded to "+child.getParent().getNode\_name());**

**receiveTokenRequest(child.getParent());**

**}**

**}**

**//upon completetion of critical section access release of token**

**public static void releaseToken(Node holder,char[] arr,ArrayList<Node> node\_list, String token\_holder,Node prev) {**

**if (!holder.isRequestForCs()) {//if the node is just forwarded request**

**while (!holder.requestQueue.isEmpty() ) {//will continue until all request of it's queue is not completed**

**if(prev!=null && !holder.hasRequest(prev) && !prev.requestQueue.isEmpty())**

**holder.setRequestQueue(prev);//setting current node as request to next node queue**

**Node nextHolder = holder.requestQueue.remove(0);//removal of first element to whom token will be sent**

**holder.setHasToken(false);**

**nextHolder.setHasToken(true);**

**System.out.println("Token passed to " + nextHolder.getNode\_name());**

**nextHolder.setParent(nextHolder); // Next holder should not have a parent in this context**

**holder.setParent(nextHolder);**

**if(!holder.requestQueue.isEmpty()) {//after passing token status of current holder**

**System.out.print("Current nodes in queue of " + holder.getNode\_name() + " is: ");**

**for (Node i : holder.requestQueue) {**

**if(i!=null)**

**System.out.print(i.getNode\_name() + " , ");**

**}**

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

**}**

**//again recursive call for next node**

**releaseToken(nextHolder,arr,node\_list,token\_holder,holder);**

**if(!holder.requestQueue.isEmpty()) {**

**System.out.println("Token passed to " + holder.getNode\_name());**

**holder.getRequestQueue().toString();**

**}**

**}**

**} else {//if node is requested for access**

**holder.setRequestForCs(false);**

**aquireToken(holder);//aquire method called**

**holder.setParent(holder);**

**if(Character.toString(arr[arr.length-1]).equalsIgnoreCase(holder.getNode\_name())) {**

**flag=0;**

**}else {**

**flag=1;**

**}**

**//status of structure after access of critical section**

**tree\_structure(node\_list,token\_holder);**

**holder.setHasToken(false);**

**System.out.println(holder.getNode\_name() + " is out of CS");**

**if(!holder.requestQueue.isEmpty()) {**

**//recall for the request of current node**

**releaseToken(holder,arr,node\_list,token\_holder,prev);**

**}else {**

**}**

**}**

**}**

**//confirmation for Access in criticak section**

**public static void aquireToken(Node holder) {**

**holder.setHasToken(true);**

**System.out.println("Token Aquired by "+holder.getNode\_name());**

**System.out.println("working in CS");**

**}**

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

**// TODO Auto-generated method stub**

**//variable declaration**

**int no\_of\_node=0,ini = 0,end = 0,o=0;**

**ArrayList<Node> node\_list=new ArrayList<Node>();**

**int[][] connectivity = null;**

**String cnnt;**

**String token\_holder = null;**

**ArrayList<Node> request\_order=new ArrayList<Node>();**

**char[] temp = null;**

**try {**

**// Create a FileReader instance**

**FileReader fileReader = new FileReader("test41.txt");**

**// Create a BufferedReader instance**

**BufferedReader bufferedReader = new BufferedReader(fileReader);**

**// Read lines from the file**

**//read no of nodes**

**no\_of\_node= Integer.parseInt(bufferedReader.readLine());**

**char[] node\_values= bufferedReader.readLine().toCharArray();**

**//first token holder**

**token\_holder=bufferedReader.readLine();**

**//request order read**

**temp = bufferedReader.readLine().toCharArray();**

**connectivity = new int[no\_of\_node][no\_of\_node];//adjacency matrix definition for node-parent relation**

**for (int[] row : connectivity) Arrays.fill(row, 0);//array initialization**

**cnnt = bufferedReader.readLine();**

**while (cnnt != null) {**

**String[] parts = cnnt.split(",");**

**if(parts!=null) {**

**for(int i=0;i<no\_of\_node;i++) {**

**if(node\_values[i]==parts[0].charAt(0)) {**

**ini=i;**

**break;**

**}**

**}**

**for(int i=0;i<no\_of\_node;i++) {**

**if(node\_values[i]==parts[1].charAt(0)) {**

**end=i;**

**break;**

**}**

**}**

**}**

**connectivity[ini][end]=1;**

**cnnt = bufferedReader.readLine();**

**}**

**//node creation**

**for(int i=0;i<node\_values.length;i++) {**

**Node n=new Node();//new node creation**

**n.setNode\_name(Character.toString(node\_values[i]));//set node name**

**if(n.getNode\_name().equalsIgnoreCase(token\_holder)) {**

**n.setHasToken(true);//initialise token holder**

**}**

**else {**

**n.setHasToken(false);**

**}**

**for(int j=0;j<node\_values.length;j++) {**

**if(connectivity[i][j]==1) {**

**for(Node nd : node\_list) {**

**if(nd.getNode\_name().equalsIgnoreCase(Character.toString(node\_values[j]))) {**

**n.setParent(nd);//parent to node link establishment**

**break;**

**}**

**}**

**break;**

**}**

**}**

**node\_list.add(n);**

**for(char c: temp) {**

**if(Character.toString(c).equalsIgnoreCase(n.getNode\_name())) {**

**request\_order.add(n);**

**break;**

**}**

**}**

**}**

**// Close the reader**

**bufferedReader.close();**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**System.out.println("current token holder is : "+token\_holder);**

**for(int x=0;x<temp.length;x++) {**

**for(Node n : node\_list) {**

**if(n.getNode\_name().equalsIgnoreCase(Character.toString(temp[x]))) {**

**if(!n.isHasToken()) {**

**n.setRequestForCs(true);**

**//request generate as per given order**

**receiveTokenRequest(n);**

**}**

**else {**

**//structure of tree after al l request recieved**

**tree\_structure(node\_list,token\_holder);**

**char[] arr = new char[temp.length-1];**

**for(char i: temp) {**

**if(i!=temp[x]) {**

**arr[o]=i;**

**o++;**

**}**

**}**

**//token relase operation to give access of critical section**

**releaseToken(n,arr,node\_list,token\_holder,null);**

**}**

**break;**

**}**

**}**

**}**

**}**

**//tree structure or status of whole structure**

**private static void tree\_structure(ArrayList<Node> node\_list, String token\_holder) {**

**// TODO Auto-generated method stub**

**System.out.println(" The tree Structure ");**

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

**for(Node n : node\_list) {**

**if(n.getNode\_name().equals(token\_holder)) {**

**System.out.print("Node : "+n.getNode\_name()+" Is in CS: "+n.isHasToken()+" Request queue :");**

**if(n.isRequestForCs()) System.out.print(" | "+n.getNode\_name()+" | ");**

**for(Node m :n.requestQueue) {**

**if(m!=null) System.out.print(" | "+m.getNode\_name()+" | ");**

**}**

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

**}else {**

**System.out.print("Node : "+n.getNode\_name()+" Node parent :"+n.getParent().getNode\_name()+" Is in CS: "+n.isHasToken()+" Request queue :");**

**if(n.isRequestForCs()) System.out.print(" | "+n.getNode\_name()+" | ");**

**for(Node m :n.requestQueue) {**

**if(m!=null)System.out.print(" | "+m.getNode\_name()+" | ");**

**}**

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

**}**

**}**

**}**

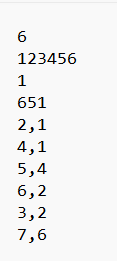
**}**

**Prerequisite:**

**Note:** In this algorithm the input file format will be:

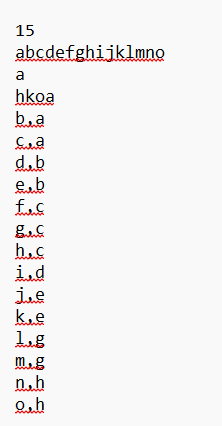
* Node name must be in one single letter.
* Total number of nodes.
* Actual order of nodes.(without space and comma ‘,’)
* Initiator node
* Requested nodes for CS and the last node is representing the initiator node.(without space comma ‘,’)
* Parent Child relation- First number indicates the child node while the second number indicates the parent node.(separated by a comma ‘,’)

**Example:**

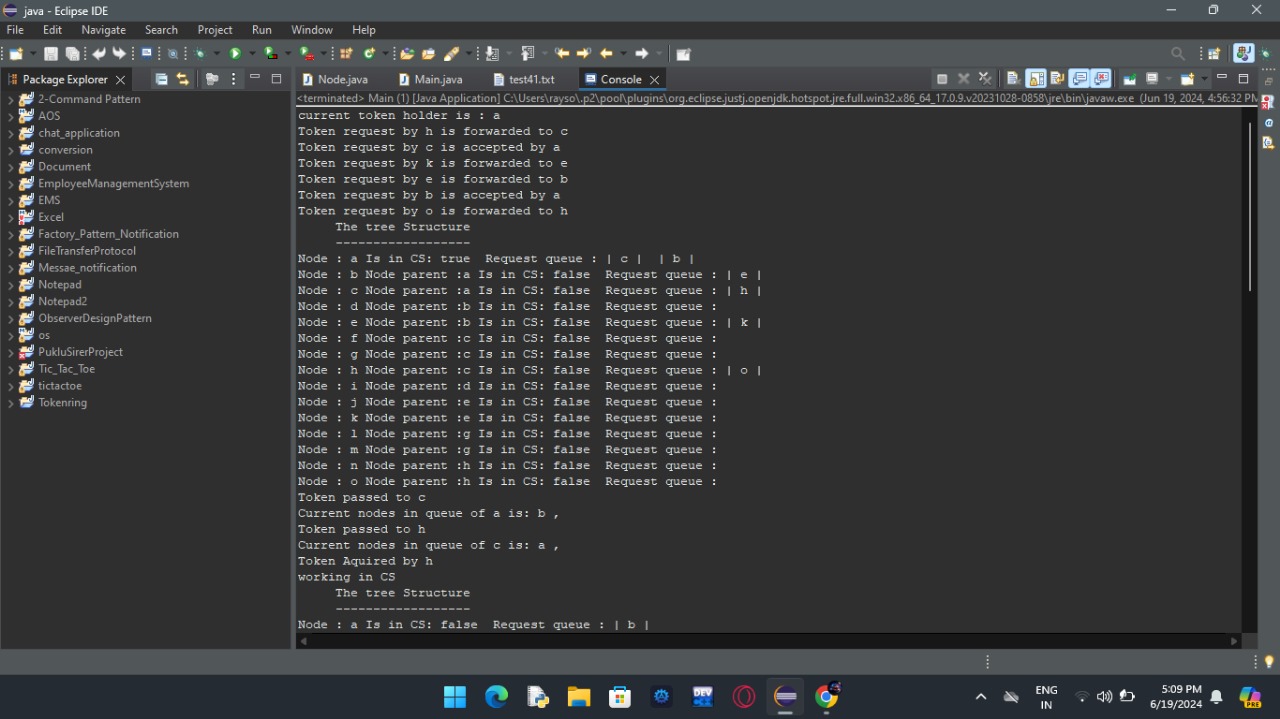
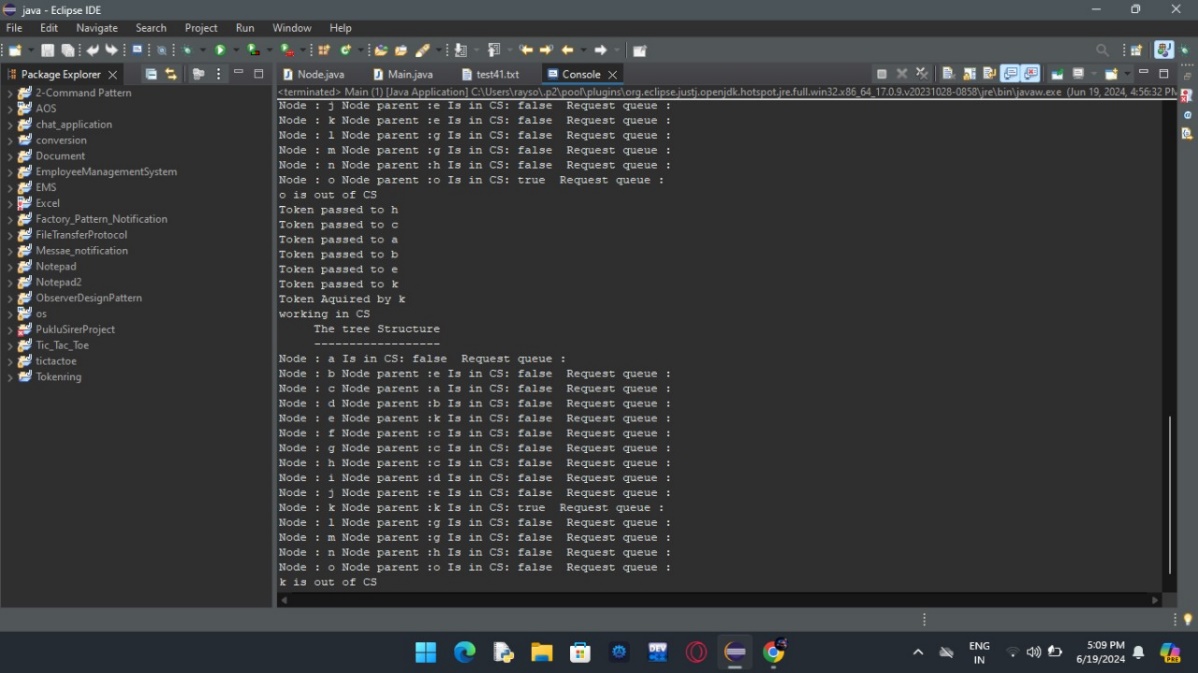
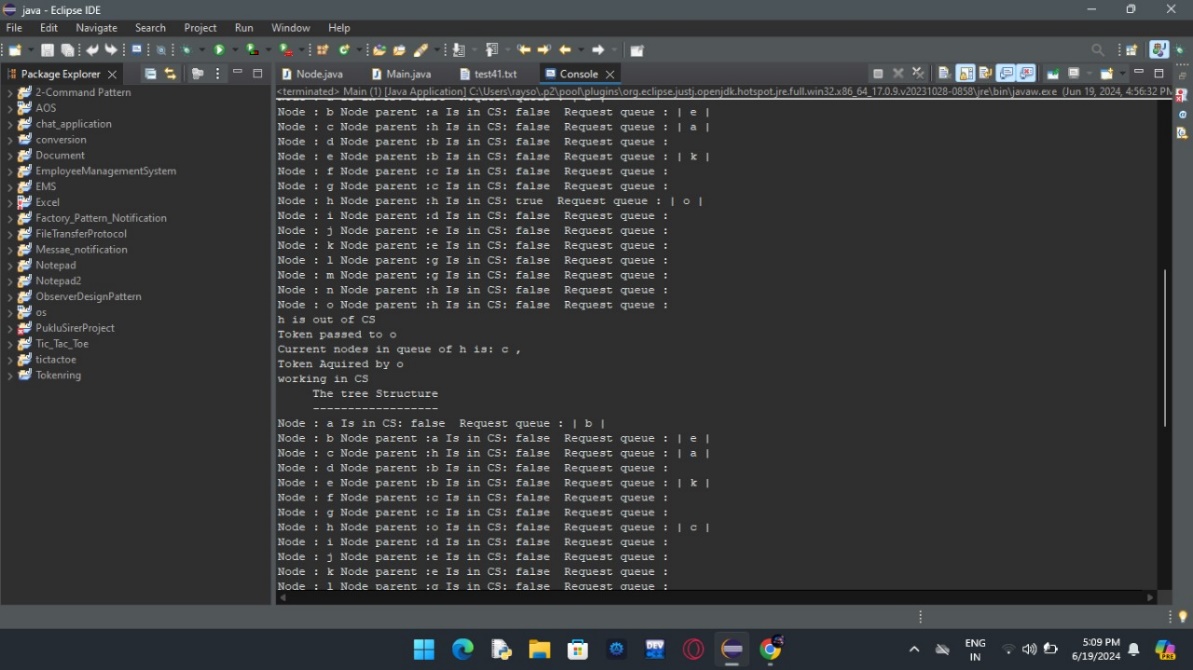
****

**Outputs:**

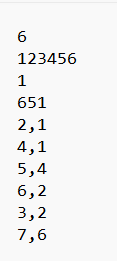
**Input-1:**



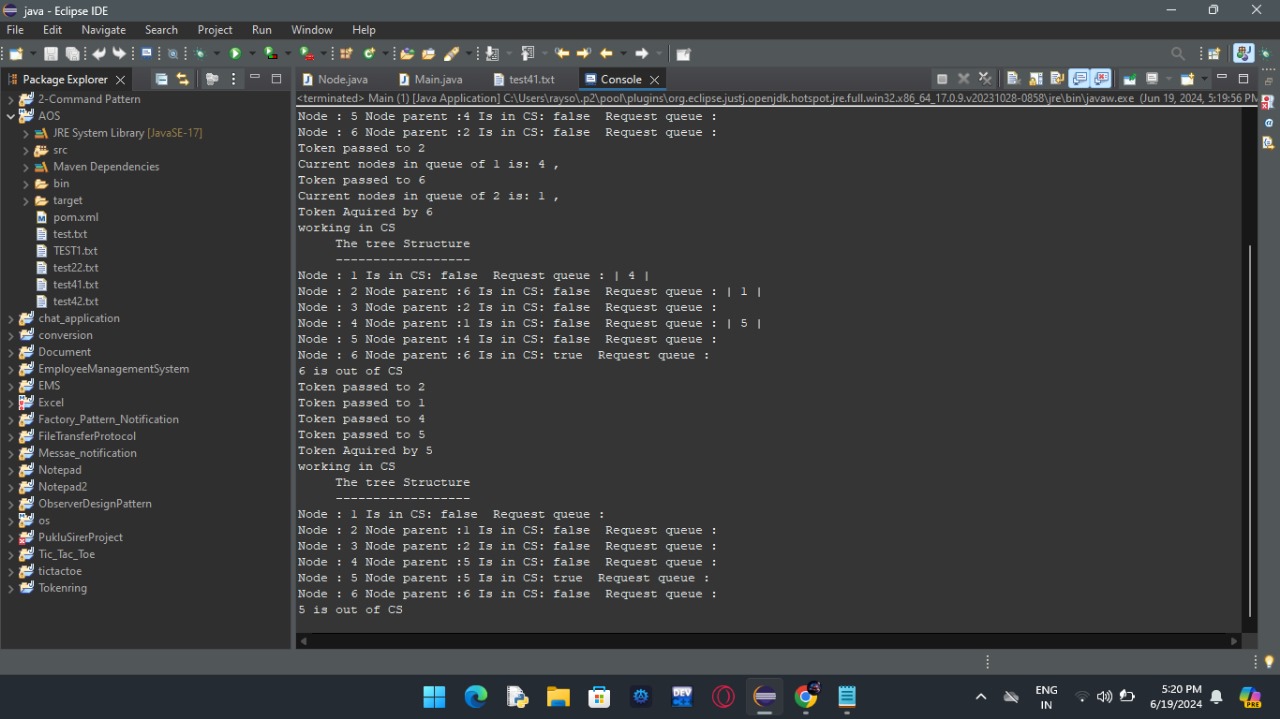
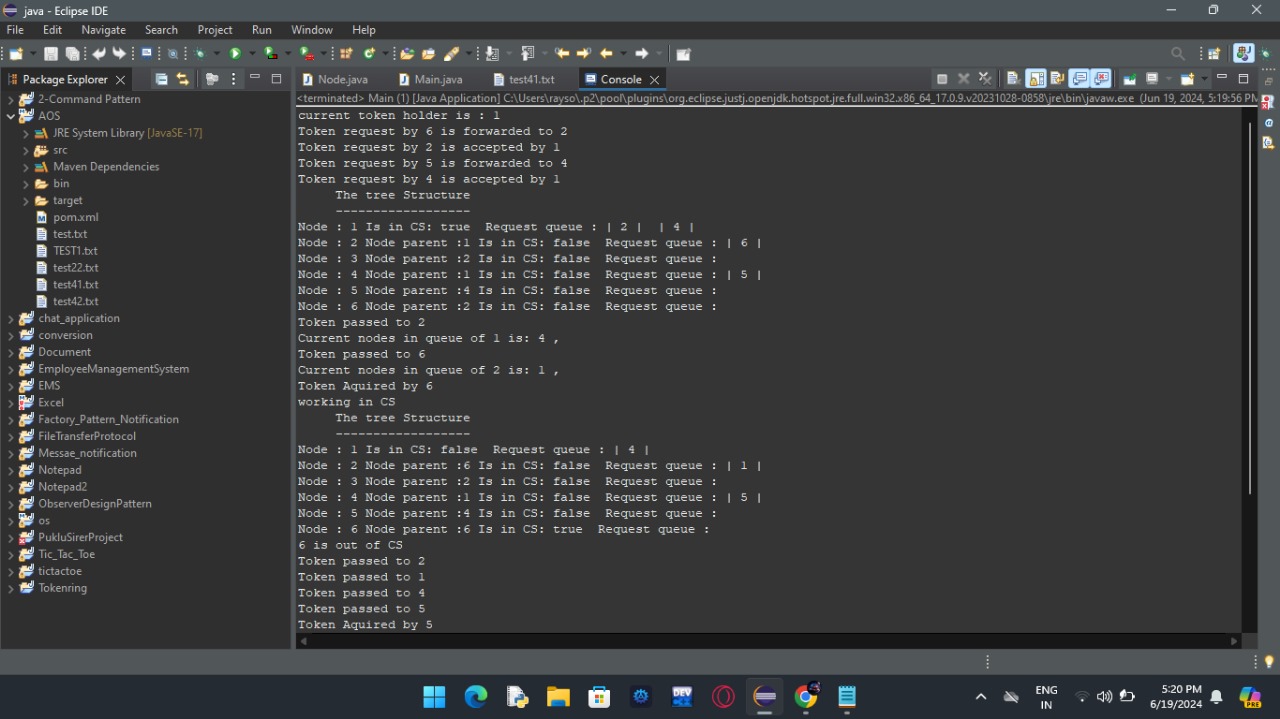
**Output-1:**

**  **

**Input-2:**

****

**Output-2:**

** **

**Remarks:**

**Complexity:**

To analyze the time complexity of the provided code, let's consider the different methods and relevant data structures used:

1. \*\*Node class\*\*:

- The methods in the `Node` class have a constant time complexity, O(1), except for the `hasRequest(Node)` method, which has a time complexity of O(n), where n is the size of the `requestQueue` ArrayList.

2. \*\*Main class\*\*:

- `receiveTokenRequest(Node)` and `forwardTokenRequest(Node)` methods:

These methods have a constant time complexity, O(1), as they perform a constant number of operations and method calls.

- `releaseToken(Node, char[], ArrayList<Node>, String, Node)` method:

This method has a time complexity of O(n^2), where n is the total number of nodes. It iterates through the `requestQueue` ArrayList, which has a time complexity of O(n) in the worst case (due to the `remove(0)` operation), and it potentially calls itself recursively for each node in the queue, leading to a nested loop behavior.

- `aquireToken(Node)` method:

This method has a constant time complexity, O(1), as it performs a constant number of operations.

- `main(String[])` method:

- The file reading and node creation part has a time complexity of O(n^2), where n is the number of nodes. It involves nested loops to establish the parent-child relationships based on the connectivity matrix.

- The loop that generates requests has a time complexity of O(n), where n is the number of nodes.

- The call to `receiveTokenRequest(Node)` or `releaseToken(Node, char[], ArrayList<Node>, String, Node)` has a time complexity of O(n^2) in the worst case due to the complexity of the `releaseToken` method.

- `tree\_structure(ArrayList<Node>, String)` method:

This method has a time complexity of O(n), where n is the number of nodes, as it iterates through the list of nodes and prints their information.

Overall, the most time-consuming part of the code is the `releaseToken` method, which has a time complexity of O(n^2) in the worst case, where n is the total number of nodes. This is due to the nested loop behavior when iterating through the `requestQueue` and potentially calling itself recursively for each node in the queue.

The other parts of the code have time complexities ranging from O(1) to O(n^2), depending on the specific operations and loops involved.

It's important to note that the actual runtime performance of the code will also depend on other factors, such as the number of nodes, the structure of the network (connectivity), and the order of requests.