

CSCU9V5  
Distributed Systems Assignment Report

Student no: 2519302

November 25<sup>th</sup> 2018

# Contents

<b>1</b>	<b>Problem Description</b>	<b>iii</b>
<b>2</b>	<b>Assumptions</b>	<b>iv</b>
<b>3</b>	<b>Solution implementation</b>	<b>v</b>
3.1	ringManager . . . . .	v
3.2	ringMemberImpl . . . . .	vi
3.3	ringMember . . . . .	vii
3.4	criticalSection . . . . .	vii
3.5	TokenObject . . . . .	vii
3.6	Full run of program . . . . .	viii
<b>4</b>	<b>Advanced Implementation</b>	<b>ix</b>
4.1	Injecting an actual token Object which keeps a counter of how many ob- jects it has been passed to . . . . .	ix
4.2	Allow user to input file name for shared file . . . . .	ix
4.3	Specify number of passes a token can take in the network . . . . .	ix
4.4	Direct the n <sup>th</sup> node to get extra sleep . . . . .	ix
4.5	Directing the m <sup>th</sup> node to be skipped every second visit of the token . . . .	ix
<b>A</b>	<b>Code Listings</b>	<b>x</b>
A.1	ringManager.java . . . . .	x
A.2	ringMemberImpl.java . . . . .	xiii
A.3	ringMember.java . . . . .	xv
A.4	criticalSection.java . . . . .	xvi
A.5	TokenObject.java . . . . .	xix

# 1 Problem Description

This report will detail the implementation of a token passing ring node network in Java. This implementation uses RMI as the means of communicating with each node in the network and communication between each node is directed by a token passed through this network.

The general description of the task is as such, we have a ringManager class that acts as a client which initializes a connection with the first node in the network and passes a token to that node. From there the ringMemberImpl class that acts as a server node in the network with the task of receiving the token it is passed, recording that transaction onto a file and releasing the token on to the next node in the network. Each node in the network is to record their transactions onto the same file each time, this is allowed since RMI makes blocking calls to each node. This is done until some stopping condition is provided. Below is a diagram that outlines the network architecture for this task:

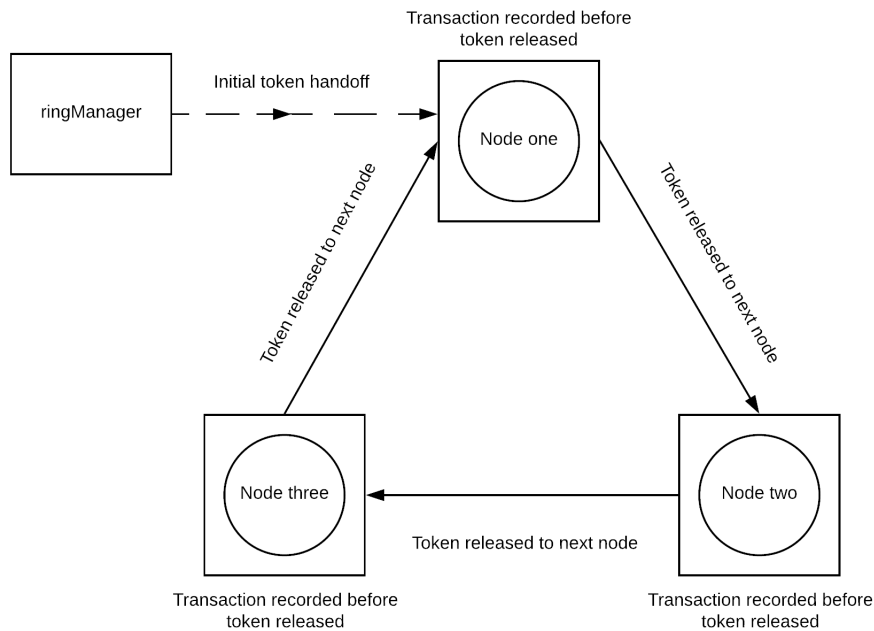


Figure 1: Architecture of this ring network

## 2 Assumptions

In the process of implementing the solution to this project, a few assumptions were made. The assumptions made were as follows:

- 1 Assume that there can be multiple nodes with the same ID
- 2 Assume that the user enters their inputs correctly

Assumption number one, is made since, if this program is run across different machines, it is possible for there to be more than one node with the same ID. This is due to the fact that each node is on a different host which as far as the program is concerned does not pose an issue upon instantiation. This raises the requirement to check both the node ID and the host of that node.

Assumption number two assumes that the user is aware of what should be entered in the command line interface. Due to this assumption, no checks are done upon user input in the command line interface. The only checks that are present for command line input are related to the number of parameters entered.

This section of the report aims to explain the solution implemented in this project. Additionally, accompanying screenshots of how each class when instantiated in the command line will be shown to demonstrate what the program should look like before the token passing is initiated.

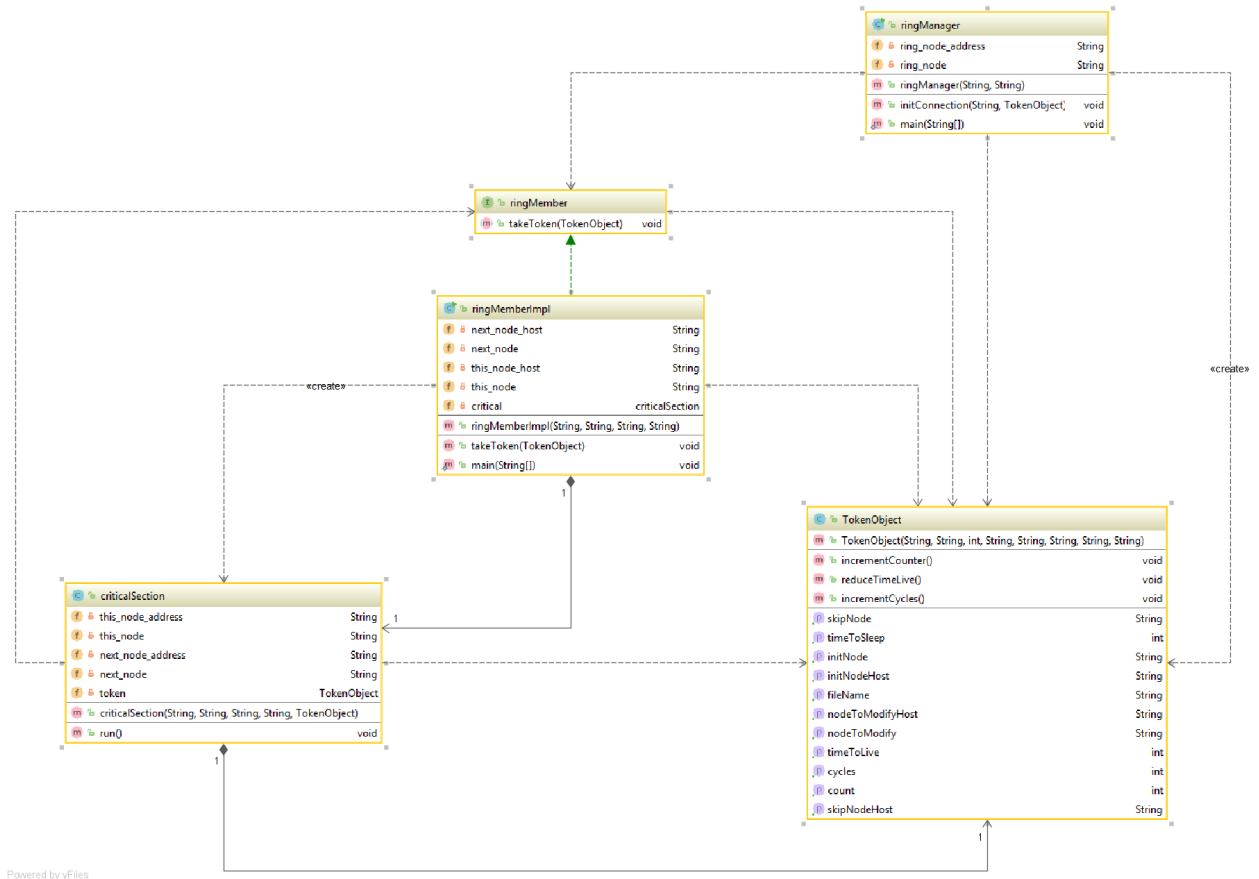


Figure 2: Class diagram

### 3.1 ringManager

The `ringManager` class, as mentioned before serves as the client to initialize connection to the network and passes the token to a node. In this class, we have two instance variables holding the initial node's ID and the host in which that node exists. This is so that the client can resolve the node to which it should send the token off to. The method `initConnection()` serves as the function that handles initializing the connection between `ringManager` and the first node it sends the token to. Inside this method, we handle the clearing of the shared file for writing whenever we restart the network as well as getting a remote reference to the node and injecting the token into the network.

In the main method for the ringManager, we accept 8 parameters in the command line interface. These variables are:

- the host on which the initial node exists

- the initial node ID
- the filename(with extension) of the shared file
- the time to live of the token (max number of passes token can take)
- the node ID to send extra sleep to
- the host where the extra sleep node exists
- the node ID to skip
- the host where the node to skip exists

All of these variables save the first two are entered here for the sole purpose of initializing them within the token Object. This eliminates the need for each node in the network to store these variables.

Below is an example of how ringManager should look like when run in the command line, along with the arguments in their respective order:

```

C:\WINDOWS\system32\cmd.exe
C:\Users\Rogue One\Desktop\CSCU9V5-Assignment\VS Code\bin>java -Djava.security.policy=java.policy ringManager localhost one record.txt 11 two localhost two localhost
Ring manager host is: DESKTOP-HW13P
Ring element host is: one
Clearing record.txt file...
Connecting to Node
C:\Users\Rogue One\Desktop\CSCU9V5-Assignment\VS Code\bin>

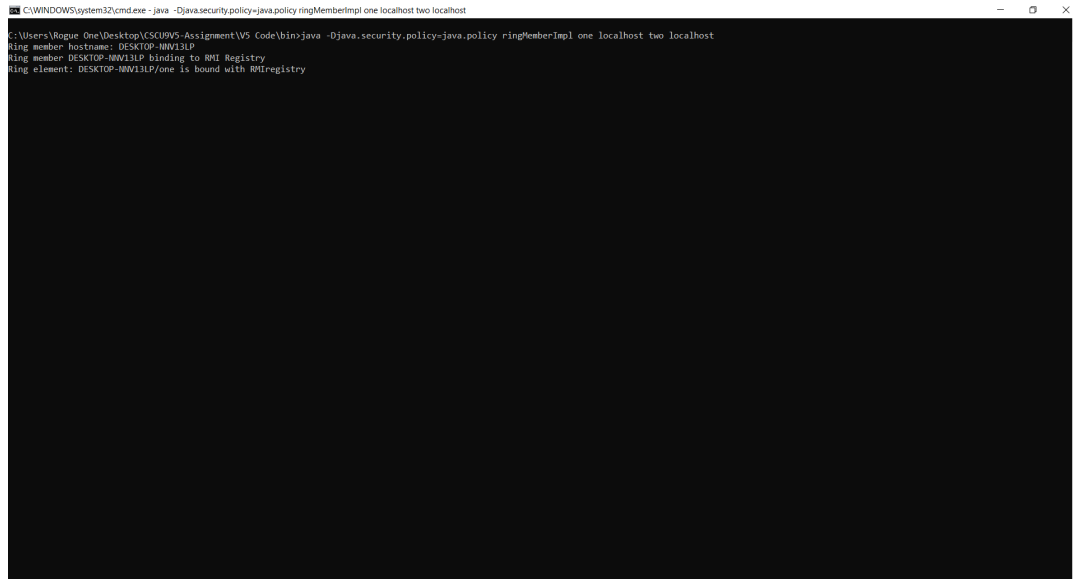
```

Figure 3: Example run of ringManager in Command Prompt

### 3.2 ringMemberImpl

The ringMemberImpl class, as mentioned in the first section, serves as each server node. This class implements the ringMember interface. This implementation is to allow for RMI method calls to the node. In this class resides the takeToken() method which deals with the receiving of the token and passing the token on to the next node in the network. This is representative of the critical section in this program, however due to the nature of RMI calls being blocking calls, the actual critical section is implemented as a separate Thread which is started within this method body. The main method of this class is fairly simple, with the 4 variables passed in as command line arguments:

- the node ID of the current node to initialize



```
C:\WINDOWS\system32\cmd.exe - java -Djava.security.policy=java.policy ringMemberImpl one localhost two localhost
C:\Users\Rogue One\Desktop\CSC4995-Assignment\VS Code\bin>java -Djava.security.policy=java.policy ringMemberImpl one localhost two localhost
Ring member hostname: DESKTOP-MW13LP
Ring member DESKTOP-MW13LP binding to RMI Registry
Ring element: DESKTOP-MW13LP/one is bound with RMIRegistry
```

Figure 4: Example of ringMemberImpl on Command Prompt

- the host on which to initialize this node
- the node ID of the immediate next node
- the host on which the immediate next node exists

Below is an example of how an instance of ringManagerImpl would look like when run using the command line:

### 3.3 ringMember

This is an interface. This interface implements the method takeToken() which is called in the class ringMemberImpl. This interface is implemented in order to allow for RMI method calls on the node.

### 3.4 criticalSection

The criticalSection class is the class where the actual critical section of execution is held. This class extends Thread so that it can be executed on a separate Thread in order to circumvent a deadlock situation given that RMI method calls are blocking calls. Given that this section is executed as a Thread, the run() method is implemented here. Upon running of the Thread, most of the logic regarding the activity of the node is handled here. These include, keeping track of the number of passes the token has taken through the node, keeping track of the number of cycles through the network the token has taken, logging the transaction of the token to a shared file and handling which node should be skipped and which node is getting extra time to sleep.

### 3.5 TokenObject

The TokenObject class serves as the class that instantiates the token Object. This class contains the instance variables as passed in from the command line for ringManager.

Having these variables in the token itself allows for greater ease in keeping track of where the token is and as a result, allows for easier manipulation of node activity.

### 3.6 Full run of program

```

C:\WINDOWS\system32\cmd.exe
C:\Users\Rogue One\Desktop\CSCU9V5-Assignment\V5 Code\bin>java -Djava.security.policy=java.policy ringManager localhost one record.txt 11 two localhost two localhost
Ring manager host is: DESKTOP-MWV13P
Ring element host is: one
Clearing record.txt file...
Connecting to Node
C:\Users\Rogue One\Desktop\CSCU9V5-Assignment\V5 Code\bin>

C:\WINDOWS\system32\cmd.exe - java -Djava.security.policy=java.policy ringMemberImpl one localhost two L...
-----
Entered method takeToken(): ringMemberImpl
Exiting method takeToken(): ringMemberImpl
Writing to file: record.txt...
Looking up RMIRegistry with rmi://localhost/two
Received token, count value is: 6
Taking a nap...

Token released, exiting critical region

-----
Entered method takeToken(): ringMemberImpl
Exiting method takeToken(): ringMemberImpl
Writing to file: record.txt...
Looking up RMIRegistry with rmi://localhost/two
Received token, count value is: 9
Taking a nap...

Token released, exiting critical region

-----

C:\WINDOWS\system32\cmd.exe - java -Djava.security.policy=java.policy ringMemberImpl three localhost one...
-----
Entered method takeToken(): ringMemberImpl
Exiting method takeToken(): ringMemberImpl
Writing to file: record.txt...
Looking up RMIRegistry with rmi://localhost/one
Received token, count value is: 5
Taking a nap...

Token released, exiting critical region

-----
Entered method takeToken(): ringMemberImpl
Exiting method takeToken(): ringMemberImpl
Writing to file: record.txt...
Looking up RMIRegistry with rmi://localhost/one
Received token, count value is: 8
Taking a nap...

Token released, exiting critical region

-----

C:\Windows\System32\cmd.exe - java -Djava.security.policy=java.policy ringMemberImpl two localhost three L...
-----
Entered method takeToken(): ringMemberImpl
Exiting method takeToken(): ringMemberImpl
Writing to file: record.txt...
Token on cycle: 2

Entered method takeToken(): ringMemberImpl
Exiting method takeToken(): ringMemberImpl
Writing to file: record.txt...
Looking up RMIRegistry with rmi://localhost/three
Received token, count value is: 7
Taking a nap...

Extra sleep of: 3 seconds
Token released, exiting critical region

-----
Entered method takeToken(): ringMemberImpl
Exiting method takeToken(): ringMemberImpl
Writing to file: record.txt...
Token on cycle: 4

Max passes reached, all remaining nodes waiting...

```

Figure 5: Example of a full run of the program on Command Prompt



## 4 Advanced Implementation

This section will detail the advanced implementations as implemented in this project.

### 4.1 Injecting an actual token Object which keeps a counter of how many objects it has been passed to

Implementing an actual token Object was done by making the TokenObject class as mentioned above in Subsection 3.5. This implementation also required a change to be made to the takeToken() method in both the ringMember interface and the ringMemberImpl class where it takes in a parameter of type TokenObject. This TokenObject then stores a counter variable and a method to increment this counter. From there in the critical section, whenever the token enters a node and the critical section Thread executes, the token has its counter increased by one. This implementation is critical to the further implementations.

### 4.2 Allow user to input file name for shared file

Implementing the file name from user input was done by accepting it in the command line run for ringManager and storing it in the TokenObject. From then it is merely passing that String variable from the token to the FileWriter in the critical section.

### 4.3 Specify number of passes a token can take in the network

This feature was implemented by accepting a value for the number of passes a token can take in the network from the command line run for ringMember and storing it in the TokenObject. From there it is merely keeping track of the number of passes the token has taken, I personally do it by counting down the number of passes taken and when the max number of passes has reduced to zero, the token is not passed anymore and the remaining nodes are left to wait.

### 4.4 Direct the $n^{\text{th}}$ node to get extra sleep

The  $n^{\text{th}}$  node is directed to get extra sleep by accepting the values for the node id and the host where that node exists into the TokenObject. From there a check is made in the critical section if the node the token is in is the node that is to get extra sleep. If that check resolves to true, the node is allocated a sleep time already initialized in the TokenObject and the node will print out a String stating the amount of sleep it is getting extra.

### 4.5 Directing the $m^{\text{th}}$ node to be skipped every second visit of the token

The  $m^{\text{th}}$  node is skipped by having the token keep track of which cycle it is on and which token it is to skip. The token keeps track of the cycle it is on by keeping track of the first node it passes and increments a counter when it passes that node again. Then in the critical section, a check can be made if the token is on an even number cycle **and** the node is the node to be skipped. If that condition resolves as true, the node does not increment the counter in the token and just passes the token on to the next node in the network.

## A Code Listings

### A.1 ringManager.java

```
import java.rmi.*;
import java.util.Scanner;
import java.net.*;
import java.net.UnknownHostException;
import java.io.*;

public class ringManager {

    private String ring_node_address;
    private String ring_node;

    /**
     * Constructor for ringManager
     * @param ring_node_address    start node host address
     * @param ring_node_id        start node id
     */
    public ringManager(String ring_node_address, String ring_node_id
    ) {
        System.setSecurityManager(new SecurityManager());

        this.ring_node_address = ring_node_address;
        this.ring_node = ring_node_id;
    }

    /**
     * Function to initialize connection between nodes in the
     * network
     * @param file_name    filename.extension for shared file
     * @param token        TokenObject object
     */
    public void initConnection(String file_name, TokenObject token)
    {
        try {
            // create fileWriter and clear file
            FileWriter file_writer = new FileWriter(file_name, false);
            // close fileWriter
            file_writer.close();
        } catch (Exception e) {
            // Error message for file writing process
            System.out.println("Error in printing file: " + e );
        }

        //get host name
        try {
            InetAddress inet_address = InetAddress.getLocalHost();
            String ring_manager_hostname = inet_address.getHostName();
            System.out.println("Ring manager host is: " +
                ring_manager_hostname);
            System.out.println("Ring element host is: " + ring_node);
        }
    }
}
```

```

    } catch (UnknownHostException e) {
        System.out.println("Cannot resolve host: ");
        e.printStackTrace();
    }

    System.out.println("Clearing record.txt file...");

    // get remote reference to ring element/node and inject token
    // by calling
    // takeToken()
    try {
        ringMember ring_member = (ringMember) Naming.lookup("rmi
            ://" + ring_node_address + "/" + ring_node);
        System.out.println("Connecting to Node");
        ring_member.takeToken(token);
    } catch (MalformedURLException | RemoteException |
        NotBoundException e) {
        System.out.println("An RMI related error has been thrown:
            ");
        e.printStackTrace();
    }
}

/**
 * Main method for ringManager
 * @param argv
 */
public static void main(String argv[]) {
    if((argv.length < 8 || argv.length > 8)) {
        System.out.println("Usage: [this node host][this node id][
            filename.extension][time to live][node to sleep][node
            to sleep host][node to skip][node to skip host]");
        System.out.println("Only " + argv.length + " parameters
            entered");
        System.exit(1);
    }

    String init_node_host = argv[0];
    String init_node_id = argv[1];
    String shared_filename = argv[2];
    int ttl = Integer.parseInt(argv[3]);
    String node_to_sleep = argv[4];
    String node_to_sleep_host = argv[5];
    String node_to_skip = argv[6];
    String node_to_skip_host = argv[7];

    //init TokenObject
    TokenObject token = new TokenObject(node_to_sleep,
        node_to_sleep_host, ttl, shared_filename, init_node_id,
        init_node_host, node_to_skip, node_to_skip_host);
    // instantiate ringManager with parameters
    ringManager client = new ringManager(init_node_host,
        init_node_id);
    client.initConnection(shared_filename, token);

```

}  
}

## A.2 ringMemberImpl.java

```
import java.rmi.*;
import java.net.*;
import java.net.UnknownHostException;

public class ringMemberImpl extends java.rmi.server.
    UnicastRemoteObject implements ringMember {

    private String next_node_host;
    private String next_node;
    private String this_node_host;
    private String this_node;
    private criticalSection critical;

    /**
     * Constructor for ringMemberImpl
     * @param t_node      current node id
     * @param t_host      current node host address
     * @param n_node      next node id
     * @param n_host      next node host address
     * @throws RemoteException
     */
    public ringMemberImpl(String t_node, String t_host, String
        n_node, String n_host) throws RemoteException {

        this_node = t_node;
        this_node_host = t_host;
        next_node = n_node;
        next_node_host = n_host;
    }

    /**
     * Function that receives token from previous node
     * @param token TokenObject
     */
    public synchronized void takeToken(TokenObject token) {

        // start critical section by instantiating and starting
        // criticalSection thread
        critical = new criticalSection(this_node, this_node_host,
            next_node, next_node_host, token);

        System.out.println("Entered method takeToken():
            ringMemberImpl");
        critical.start();
        System.out.println("Exiting method takeToken():
            ringMemberImpl");
    }

    /**
     * Main method for ringMemberImpl
     * @param argv      Command line arguments
     */
}
```

```

*/
public static void main(String argv[]) {
    System.setSecurityManager(new SecurityManager());
    if ((argv.length < 4) || (argv.length > 4)) {
        System.out.println("Usage: [this node][this node host][
            next node][next node host]");
        System.out.println("Only " + argv.length + " parameters
            entered");
        System.exit(1);
    }

    // get current node hostname, id and next node hostname, id
    // and filename from command line args
    String current_node = argv[0];
    String current_host = argv[1];
    String next_node = argv[2];
    String next_host = argv[3];

    // get host name
    try {
        InetAddress inet_address = InetAddress.getLocalHost();
        String member_hostname = inet_address.getHostName();

        System.out.println("Ring member hostname: " +
            member_hostname);
        System.out.println("Ring member " + member_hostname + "
            binding to RMI Registry");

        // instantiate ringMemberImpl class with appropriate
        // parameters
        ringMemberImpl server = new ringMemberImpl(current_node
            , current_host, next_node, next_host);
        // register object with RMI registry
        Naming.rebind("rmi://" + current_host + "/" +
            current_node, server);

        System.out.println("Ring element: " + member_hostname +
            "/" + current_node + " is bound with RMRegistry");

    } catch (UnknownHostException e) {
        System.out.println("Cannot resolve host: ");
        e.printStackTrace();
    } catch (RemoteException e) {
        System.out.println("RMI related exception thrown: ");
        e.printStackTrace();
    } catch (MalformedURLException e) {
        System.out.println("Error in input URL: ");
        e.printStackTrace();
    }
}
}
}

```

### A.3 ringMember.java

```
public interface ringMember extends java.rmi.Remote {

    /**
     * @param token TokenObject object
     * @throws java.rmi.RemoteException
     */
    public void takeToken(TokenObject token) throws java.rmi.
        RemoteException;

}
```

## A.4 criticalSection.java

```
import java.io.*;
import java.net.*;
import java.rmi.*;
import java.rmi.ConnectException;
import java.rmi.UnknownHostException;
import java.util.*;

/**
 * @author ast
 *
 * Critical section within executing Thread
 *
 */
public class criticalSection extends Thread {

    private String this_node_address;
    private String this_node;
    private String next_node_address;
    private String next_node;

    private TokenObject token;

    /**
     * Constructor for critical section
     * @param t_node      current node id
     * @param t_node_add  current node address
     * @param n_node      next node id
     * @param n_node_add  next node address
     * @param t            TokenObject object
     */
    public criticalSection(String t_node, String t_node_add, String
        n_node, String n_node_add, TokenObject t) {

        this_node = t_node;
        this_node_address = t_node_add;
        next_node = n_node;
        next_node_address = n_node_add;
        token = t;
    }

    /**
     * Initializes Thread for running of critical section
     */
    public void run() {
        // entering critical section
        try {

            System.out.println("Writing to file: record.txt...");

            // init timestamp
            Date time = new Date();
            String timestamp = time.toString();

            token.reduceTimeLive();
        }
    }
}
```



```

// increment cycles in TokenObject
if (token.getInitNode().equals(this_node) && token.
    getInitNodeHost().equals(this_node_address))
    token.incrementCycles();

// if getSkipNode() returns true, pass the token without
// incrementing,
// else continue as normal
if(token.getCycles() % 2 == 0 && (token.getSkipNode().
    equals(this_node) && token.getSkipNodeHost().equals(
    this_node_address))) {
    System.out.println("Token on cycle: " + token.getCycles
        () + "\n");
} else {
    // increment counter in TokenObject
    token.incrementCounter();

    // write timestamp (date) to file
    FileWriter file_writer = new FileWriter(token.
        getFileName(), true);

    // init PrintWriter to write to the file
    PrintWriter print_writer = new PrintWriter(file_writer,
        true);
    print_writer.println("Record from ring node on host: "
        + this_node_address + ", node: " + this_node
        + ", is: " + timestamp + ", token count: " +
        token.getCount());

    print_writer.close();
    file_writer.close();

    System.out.println("Looking up RMIRegistry with rmi://"
        + next_node_address + "/" + next_node);

    System.out.println("Received token, count value is: " +
        token.getCount());

    // sleep to symbolise critical section duration
    System.out.println("Taking a nap...\n");

    if (token.getNodeToModify().equals(this_node) && token.
        getNodeToModifyHost().equals(this_node_address)) {
        System.out.println("Extra sleep of: " + (token.
            getTimeToSleep() / 1000) + " seconds");
        sleep(token.getTimeToSleep());
    }
    sleep(2000);

    System.out.println("Token released, exiting critical
        region\n");

    // Print dashed lines for ease of reading
    System.out.print("
        -----\n

```

```

        n");
    }

    if (token.getTimeToLive() == 0) {
        System.out.println("Max passes reached, all remaining
            nodes waiting...");
        return;
    }

    // get remote reference to next ring element, and pass
    // token on
    // ...
    ringMember next_ring_element = (ringMember) Naming.lookup(
        "rmi://" + next_node_address + "/" + next_node);
    next_ring_element.takeToken(token);

} catch (MalformedURLException e) {
    System.out.println("Error in input URL: ");
    e.printStackTrace();
} catch (RemoteException e) {
    System.out.println("RMI related exception thrown: ");
    e.printStackTrace();
} catch (InterruptedException e) {
    System.out.println("Sleep error: ");
    e.printStackTrace();
} catch (IOException e) {
    System.out.println("Error in file writing: ");
    e.printStackTrace();
} catch (NotBoundException e) {
    System.out.println("Server not bound error: ");
    e.printStackTrace();
}

}
}

```

## A.5 TokenObject.java

```
import java.io.Serializable;

/**
 * @author ast
 *
 * Token Object to serve as token passed through ring network,
 * also serves as an object that stores and passes on variables
 * such as,
 * filename, time to live, node for extra sleep and its host,
 * initial node and its host
 * and node to skip and its host to each node
 */
public class TokenObject implements Serializable {

    private int count; // variable holding number of times it has
        entered a critical section
    private String node_to_modify; // variable holding node id of
        node to give extra sleep time
    private String node_to_modify_host; // variable holding host of
        node to give extra sleep
    private int time_to_live; // variable holding max number of
        passes set for token
    private String file_name; // variable holding file name from
        user input
    private int sleep_time = 3000; // variable holding preset extra
        sleep time
    private int num_cycles; // variable holding number of cycles
        taken by token through the network
    private String init_node; // variable holding the initial node
        id
    private String init_node_host; // variable holding the initial
        node host
    private String skip_node; // variable holding the skip node id
    private String skip_node_host; // variable holding the skip node
        host

    /**
     * Constructor for TokenObject
     * @param node          node id for sleep modification
     * @param node_host      node host for sleep modification
     * @param time_to_live   max number of passes for token in
        network
     * @param file_name      filename.extension of shared file
     * @param init_node      initial node id
     * @param init_node_host initial node host
     * @param skip_node       node id for node to skip
     * @param skip_node_host node host for node to skip
     */
    public TokenObject(String node, String node_host, int
        time_to_live, String file_name, String init_node, String
        init_node_host, String skip_node, String skip_node_host) {
        this.node_to_modify = node;
    }
}
```

```

        this.node_to_modify_host = node_host;
        this.time_to_live = time_to_live;
        this.file_name = file_name;
        this.count = 0;
        this.num_cycles = 0;
        this.init_node = init_node;
        this.init_node_host = init_node_host;
        this.skip_node = skip_node;
        this.skip_node_host = skip_node_host;
    }

    /**
     * Function that increments the counter by one each time it
     * enters a critical section
     */
    public void incrementCounter() {
        this.count++;
    }

    /**
     * Function that returns count variable
     * @return count
     */
    public int getCount() {
        return this.count;
    }

    /**
     * Function that returns the token's time to live variable
     * @return time_to_live
     */
    public int getTimeToLive() {
        return this.time_to_live;
    }

    /**
     * Reduces time to live variable by one each time it enters a
     * critical section
     */
    public void reduceTimeLive() {
        time_to_live--;
    }

    /**
     * Function that returns node id for extra sleep
     * @return node_to_modify
     */
    public String getNodeToModify() {
        return this.node_to_modify;
    }

    /**
     * Function that returns host of node for extra sleep
     * @return node_to_modify_host
     */

```

```

public String getNodeToModifyHost() {
    return this.node_to_modify_host;
}

/**
 * Function that returns sleep_time
 * @return sleep_time
 */
public int getTimeToSleep() {
    return this.sleep_time;
}

/**
 * Function that returns the node to skip's id
 * @return skip_node
 */
public String getSkipNode() {
    return this.skip_node;
}

/**
 * Function that returns the node to skip's host
 * @return skip_node_host
 */
public String getSkipNodeHost() {
    return this.skip_node_host;
}

/**
 * Function that returns the initial node's id
 * @return init_node
 */
public String getInitNode() {
    return this.init_node;
}

/**
 * Function that returns the initial node's host
 * @return init_node_host
 */
public String getInitNodeHost() {
    return this.init_node_host;
}

/**
 * Function that increments the number of cycles token has
 * taken through the network
 */
public void incrementCycles() {
    num_cycles++;
}

/**
 * Function that returns the number of cycles the token has
 * taken through the network
 * @return num_cycles

```

```
    */
    public int getCycles() {
        return num_cycles;
    }

    /**
     * Function that returns filename for nodes to write to
     * @return file_name
     */
    public String getFileName() {
        return this.file_name;
    }
}
```