# CSCU9V5 Assignment Report

Student no: 2519302

November 25th 2018

# Contents

## 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:

todo

## Assumptions

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

1

## Solution implementation

## Code Listings

### 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 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) {
```

```
    // TODO Auto-generated catch block
    System.out.println("Something went wrong, unknown host: " + e
        );
  }

  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) {
    // TODO Auto-generated catch block
    System.out.println("Error Message: " + e);
  }

}

/**
 * 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);

}
```

```
}
```

### 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_id;
  private String next_host;
  private String this_id;
  private String this_host;
  private criticalSection critical;



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

    this_host = t_node;
    this_id = t_add;
    next_host = n_node;
    next_id = n_add;
  }

  /*
   * 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_host, this_id, next_host,
       next_id, 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 host][this id][next host][
          next id]");
      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_add = argv[1];
    String next_node = argv[2];
    String next_add = 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_add, next_node, next_add);
        // register object with RMI registry
        Naming.rebind("rmi://" + current_add + "/" + current_node,
            server);

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

    } 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();
    }

  }

}
```

### 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;

}
```

### 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))) {
  // get remote reference to next ring element, and pass
     token on
  // ...
  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");
      }

      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();
    }

  }
}
```

### 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
 *  eliminating the need to store each of these variables locally
    in the ringMemberImpl class
 *
 */
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 time_to_live    max number of passes for token in
      network
   * @param file_name     filename.extension of shared file
   * @param token_exit    boolean dictating if the node is to be
      shutdown
   */
  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;
  }

}
```