*2433115*

*CSCU9V5 Assignment*

Operating Systems Concurrency and Distribution

Table of Contents

[1.0 Basic Problem 2](#_Toc531018251)

[2.0 Advanced Features 3](#_Toc531018252)

[2.1 Advanced Feature 1 3](#_Toc531018253)

[2.2 Advanced Feature 2 4](#_Toc531018254)

[2.3 Advanced Feature 3 5](#_Toc531018255)

[2.4 Advanced Feature 4 6](#_Toc531018256)

[2.5 Advanced Feature 5 6](#_Toc531018257)

[2.6 Advanced Feature 6 7](#_Toc531018258)

[3.0 System flow diagram 9](#_Toc531018259)

[4.0 Code 10](#_Toc531018261)

[4.1 ringManager Class 10](#_Toc531018262)

[ringMemberImpl Class 13](#_Toc531018263)

[4.2 ringMember Interface 15](#_Toc531018264)

[4.3 criticalSection Class 15](#_Toc531018265)

[4.4 tok Class 20](#_Toc531018266)

[4.5 Java.policy File 23](#_Toc531018267)

# 1.0 Basic Problem

The problem I have been tasked with is to implement an RMI solution to a ring of interconnected nodes which pass a token. Within the basic solution, the token was simply the connection between the previous node and the next node on the ring. Once this connection was successful it gave the next node “permission” to access its critical section.

The critical section within this assignment was to access a shared resource and output the specified data. In this case the shared resource was a file and the output data was a timestamp of when the node was writing to the file. The benefits of using RMI over a socket-based solution (lab 3) is that RMI is blocking. This means that only one process can ever have access to the shared resource at a single point in time, whereas if using sockets, the critical section must be placed within a thread to lower the chances of deadlock entering the system.

To start the program, the user must start an RMIRegistry to hold the ID’s of the different nodes. Then the node must be started, outlining the required policy file with the command, *java -Djava.security.policy=java.policy* and with the arguments:

* Host
* ID

The manager class is in charge of initialising the shared resource (file) and injecting the first token to the starting node. The first node is specified by the user within the terminal arguments and the file is either defaulted to “record.txt” or, as will be discussed later, a user defined file.

# 2.0 Advanced Features

## 2.1 Advanced Feature 1

import java.io.Serializable;  
  
*/\*\*  
 \* Advanced Feature 1  
 \*  
 \* This is the token object  
 \*  
 \* It has been made serializable so it will not reset the token value every time it is instantiate  
 \* thus allowing for the token to increment with every exchange  
 \*  
 \*  
 \*/*public class tok implements Serializable {  
  
 private int token;  
 private String[] tokens\_list;  
 private int[] visits;  
  
 */\*\*  
 \* Constructor  
 \*  
 \** ***@param*** *t This is the value of the first token  
 \*/* public tok(int t) {  
  
 this.token = t;  
  
 }  
  
 */\*\*  
 \* Return the value of the current token  
 \*  
 \** ***@return*** *\*/* public int getToken() {  
 return token;  
 }  
  
 */\*\*  
 \* When called the token is incremented by 1  
 \*  
 \** ***@param*** *t This is the value of the token before it has been incremented  
 \*/* public void setToken(int t) {  
 token = t;  
 // Checking that the passed starting token is not negative  
 // if it is, inform user and exit program  
 if (token < 0) {  
 System.*out*.println("Token cannot be negative");  
 System.*exit*(1);  
 } else {  
 // If not negative, increment  
 token++;  
 }  
 }  
  
}

The aim of this feature is to implement a physical token object. This is my tok class. The tok class has been made serializable as we need to keep track of what number token it is. Making a class serializable allows it to be instantiated once, with all the variables within the class to be declared and assigned and any subsequent uses of the class doesn’t redo any of this. This allows for a counter to be implemented within the setToken and to be retrieved via the getToken. There is only one validation check for the starting token which is to check that it is not negative.

A reference to this token is then passed to the next node via the takeToken method in the ringMemberImpl and it is then incremented. This continues until the user ends the program. Further features will be added to this class later on in the report.

## 2.2 Advanced Feature 2

The aim of this feature is to retrieve a user input file name from the arguments provided by the user. This consisted of assigning a global static variable within the ringManager class and using this variable as the file input for the FileWriter. I added a check before the FileWriter, in case the user inputs null for this argument, as they want to use the default, which set the file to “record.txt”. This filename is then passed through to the takeToken method in the ringMemberImpl and finally to the Critical section for the write to be carried out.

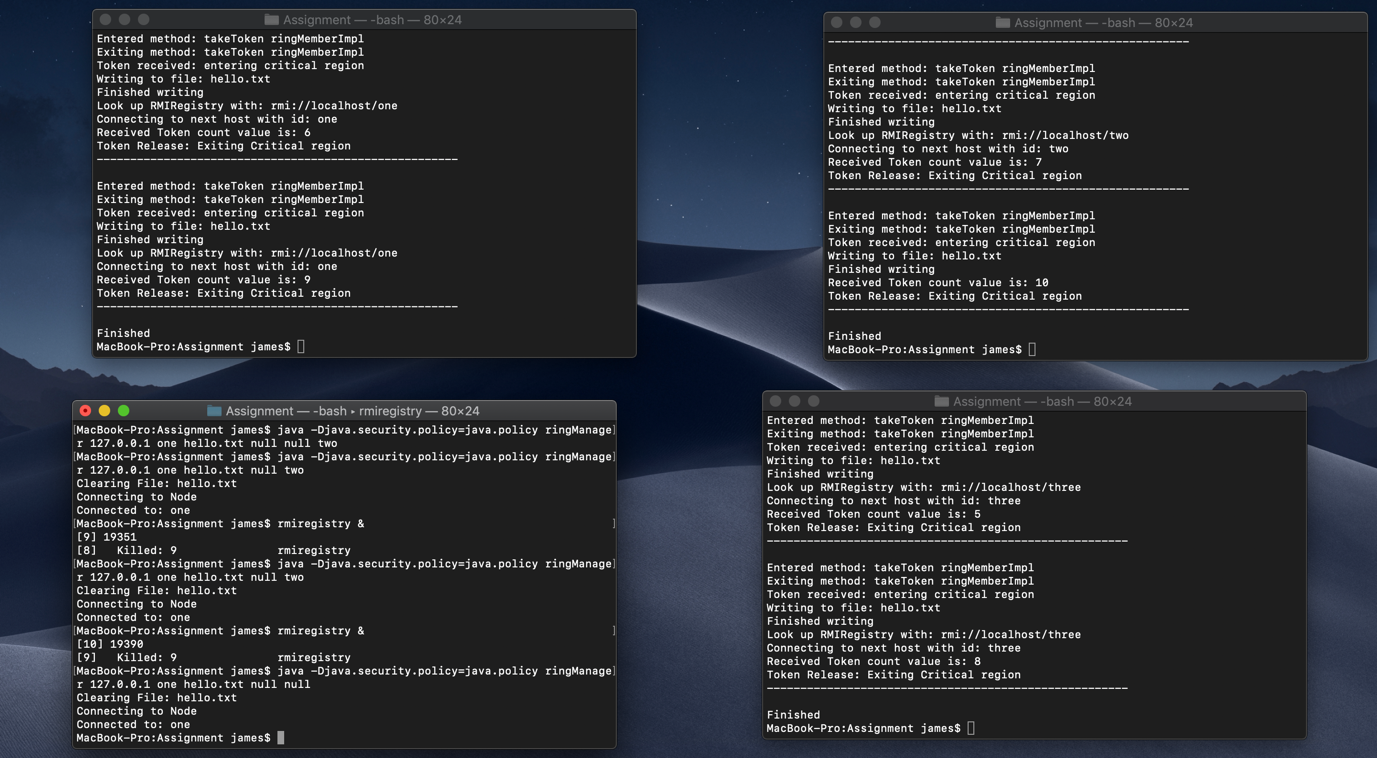
As for the permissions file, the provided file only allowed permission to a specific filename, “record.txt”. In order to allow for the user to choose any file name they wanted I changed this to “/-“. This means that the permission is for any file that is within the same directory.

## 2.3 Advanced Feature 3

The aim of this feature is to restrict the number of times the token can be passed around the loop. Within the critical section class, I declared a “rounds” variable, which I then hard coded in a specific number of exchanges.

I then used an “if statement” within the release method which checked if the current token number is less than the specified number of exchanges, if it is, the token is passed on to the next node. If not, the system exits.

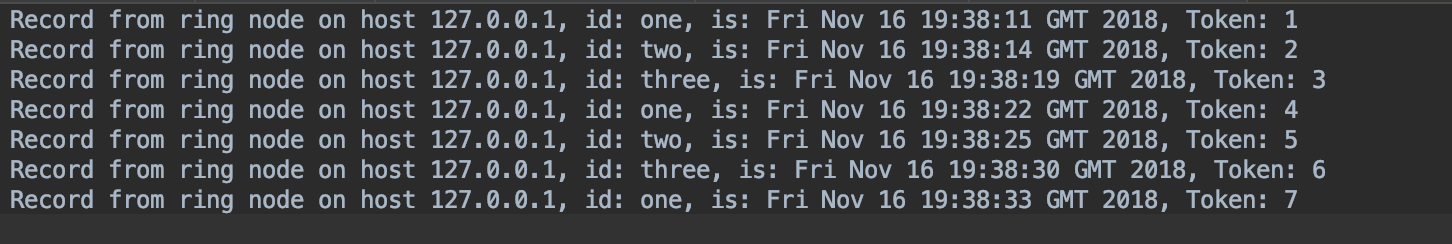
This image shows that the program ran for a specified 10 exchanges of the token:



## 2.4 Advanced Feature 4

The aim of this feature is to allow the user to specify a node that they wish to have longer time in the critical section. To start this, I declared a new global static variable in the ringManager class, “id\_longer\_CS”. This variable is then passed through to the critical section class, where it is then compared to the ID of the current node. If they are equal the sleep is extended to 5 seconds and if they are different, the sleep remains the default 3 seconds.

This image shows when node ID: three is specified it has a longer critical section. (5 seconds as opposed to 3):



## 2.5 Advanced Feature 5

The aim of this feature is to allow the user to specify a node which will be skipped every second visit. Firstly, I declared a new global static variable in the ringManager class, “skipped\_node\_id”.

Then I decided I would implement an array within the tok class to keep track of the number of visits for each individual nodes. This was split into two methods:

* Populate\_visits
* Visit\_counter

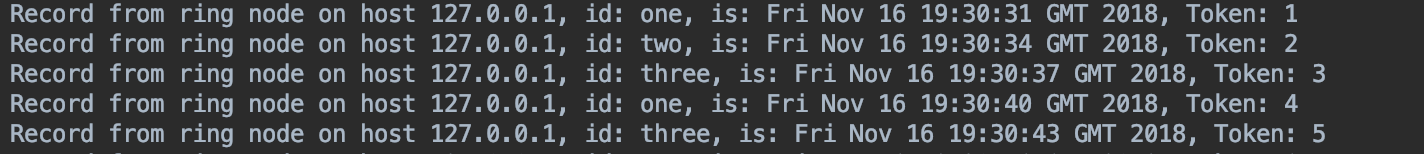
The populate visits method received the parameter nodes which was an array of the list of nodes that have been added to the RMIRegistry. It then assigns the global variable “token\_list” to this for future reference. The length of this token\_list is then used to initialise a new array, “visits”, to hold an individual counter for each node and each index is assigned to 0 at the beginning. This is all done with the “inject\_token” method in the ringManager class before the first token is injected:

*token.populate\_visits(listOfNodes); // Populate an array to keep track of the number of visits to each node*

The specified node ID is then passed to the takeToken method and then onto the run method in the critical section class. The run method then compares the current node to the specified node to be skipped every second visit. If true, the “visit\_counter” method is called in the tok class, providing the host and the ID. This method then finds the index of the ID in the token\_list array, it then uses this index to increment the value of the corresponding index in the values array.

To check if this is a second visit, I calculate the 2 moduli of the number of visits. If even, return true, false otherwise. This Boolean value is returned to the run method, if true the next node is connected to and the token passed on, if false the node continues to its critical section like normal.

Resulting file when ID two is selected to be missed:

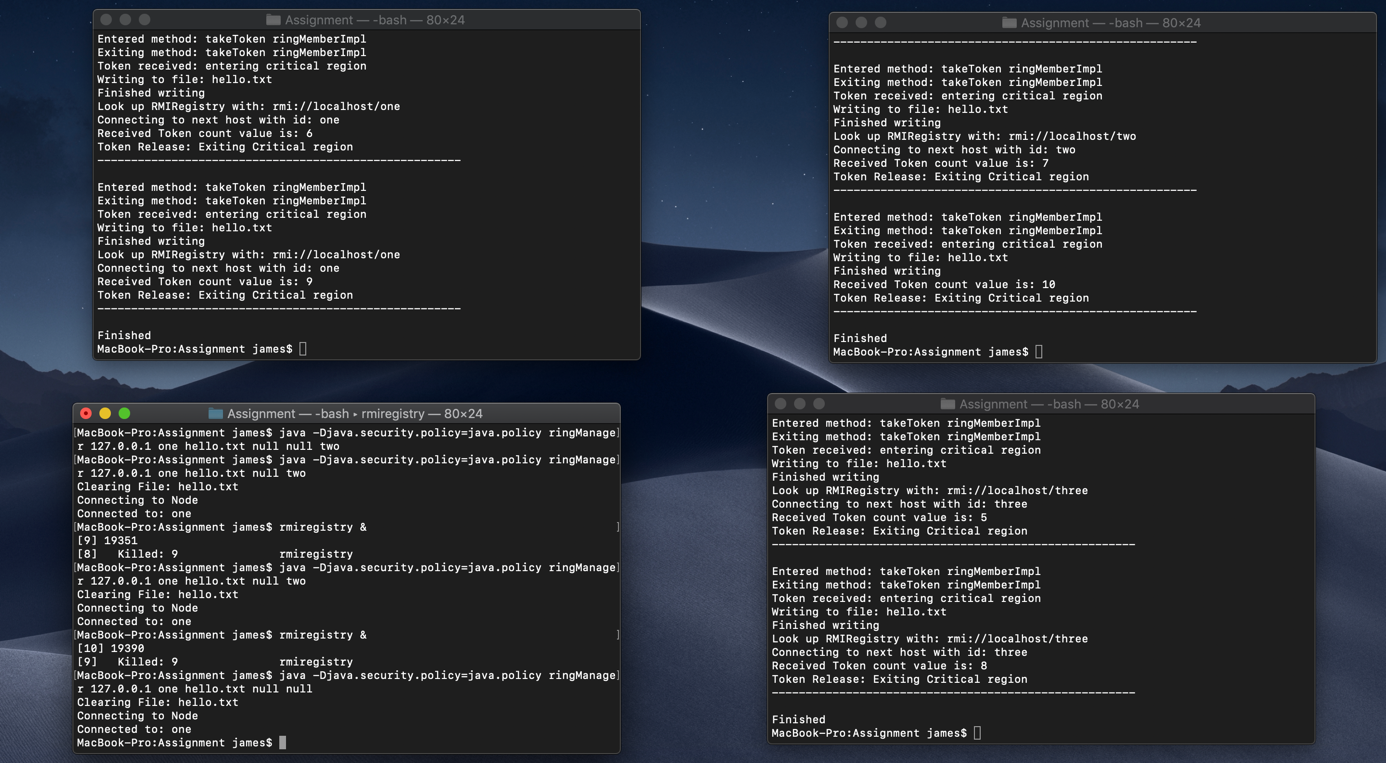


Notice that during the second visits the node ID two is missing.

## 2.6 Advanced Feature 6

The aim of this feature is to smooth out the completion of advanced feature 3. To achieve this, within the release method of the critical section class an “if statement” is used to check that the current token number is less than the hard-coded number of exchanges. If it isn’t, the command *System.exit(1)* is used to terminate this process. I realised that this would only terminate the process that was the last one to run so I added another “if statement” which checked if the token number was also less than the specified number exchanges minus the total number of nodes. This allowed all the processes to terminate in an orderly fashion.

This image shows all the processed have finished in an orderly manner:



# 3.0 System flow diagram

# 4.0 Code

## 4.1 ringManager Class

import java.rmi.\*;  
import java.net.\*;  
import java.io.\*;  
  
public class ringManager {  
  
 private static String *id\_longer\_CS*; // id of node user has specified to have longer in Critical section  
 private static String *filename*; // File user wants to use as shared resource  
 private static String *skipped\_node\_id*; // ID of the node the user wants to skip every second visit  
  
 public ringManager(String ring\_node\_host, String ring\_node\_id) {  
 System.*setSecurityManager*(new SecurityManager());  
  
 initialise\_resource(); // Create or clear file  
 inject\_token(ring\_node\_host, ring\_node\_id); // Inject token into the ring to start  
  
 }  
  
 */\*\*  
 \* Method which creates the first the shared resource: file  
 \*  
 \*  
 \*/* public void initialise\_resource(){  
 // create fileWriter and clear file  
 try{  
 // To set filename to default if user doesn't specify one  
 if(*filename* == null){  
 *filename* = "record.txt";  
 }  
 System.*out*.println("Clearing File: " + *filename*);  
 // Creating the shared resource - a File  
 // create fileWriter - true means don't append to end of file  
 FileWriter fw\_id = new FileWriter(*filename*, false);  
 fw\_id.close();  
 }catch(IOException e){  
 System.*out*.println("Exception in clearing the file: " + e);  
 }  
 }  
  
 */\*\*  
 \* Method to start the ring by "injecting" the token into the ring  
 \*  
 \* This means connect to the first node which the user has specified  
 \*  
 \** ***@param*** *node\_host This is the host of the first node  
 \** ***@param*** *node\_id This is the ID of the first node  
 \*/* public void inject\_token(String node\_host, String node\_id){  
 // get remote reference to ring element/node and inject token by calling takeToken()  
 System.*out*.println("Connecting to Node");  
 try{  
 String[] listOfNodes = Naming.*list*("rmi://" + node\_host);  
 //Creating the token instance  
 int t = 0;  
 tok token = new tok(t);  
 token.populate\_visits(listOfNodes); // Populate an array to keep track of the number of visits to each node  
 //Setting the first token for the first exchange  
 token.setToken(t);  
 ringMember member = (ringMember) Naming.*lookup*("rmi://" + node\_host + "/" + node\_id);  
 member.takeToken(*filename*, token, *id\_longer\_CS*, *skipped\_node\_id*);  
 System.*out*.println("Connected to: " + node\_id);  
  
 }catch (Exception e){  
 System.*out*.println(e);  
 }  
  
 }  
  
 */\*\*  
 \* Each of the if statements adds future functionality  
 \*  
 \*  
 \* If not looking to use a feature but want to use on the is more advanced, put null as the argument  
 \*  
 \** ***@param*** *argv User entered arguments from the terminal  
 \*/* public static void main(String argv[])  
 {  
   
 // instantiate ringManager with parameters  
  
 //Checking that the required number of arguments have been input  
 if (argv.length < 2){  
 System.*out*.println("Usage: [host] <id> filename");  
 System.*exit*(1);  
  
 // Basic functionality  
 }else if(argv.length == 2){  
  
 //Extracting the arguments  
 String host = argv[0];  
 String id = argv[1];  
  
 // Creating instance of ringManager  
 ringManager manager = new ringManager(host, id);  
  
 // Advanced feature 2: user specified filename  
 }else if(argv.length == 3){  
  
 //Extracting the arguments  
 String host = argv[0];  
 String id = argv[1];  
 *filename* = argv[2];  
  
 // Creating instance of ringManager  
 ringManager manager = new ringManager(host, id);  
  
 // Advanced feature 4: user specify a node to have a longer critical section  
 }else if(argv.length == 4){  
  
 //Extracting the arguments  
 String host = argv[0];  
 String id = argv[1];  
 *filename* = argv[2];  
 *id\_longer\_CS* = argv[3];  
  
 // Creating instance of ringManager  
 ringManager manager = new ringManager(host, id);  
  
 // Advanced feature 5: user specify a node to be skipped every second visit  
 }else if(argv.length == 5){  
  
 //Extracting the arguments  
 String host = argv[0];  
 String id = argv[1];  
 *filename* = argv[2];  
 *id\_longer\_CS* = argv[3];  
 *skipped\_node\_id* = argv[4];  
  
 // Creating instance of ringManager  
 ringManager manager = new ringManager(host, id);  
 }  
 }  
}

## ringMemberImpl Class

import java.rmi.\*;  
import java.net.\* ;  
   
public class ringMemberImpl extends java.rmi.server.UnicastRemoteObject implements ringMember   
{  
 public ringMemberImpl(String t\_node, String t\_id, String n\_node, String n\_id) throws RemoteException {  
 this\_host = t\_node ;  
 this\_id = t\_id ;  
 next\_host = n\_node ;  
 next\_id = n\_id ;  
 }  
   
 public synchronized void takeToken(String filename, tok token, String id\_longer\_CS, String skipped\_node\_id) throws RemoteException {  
 // start critical section by instantiating and starting critical section thread  
 System.*out*.println("Entered method: takeToken ringMemberImpl");  
 c = new criticalSection(this\_host, this\_id, next\_host, next\_id, filename, token, id\_longer\_CS, skipped\_node\_id);  
 Thread cThread = new Thread(c);  
 cThread.start();  
 System.*out*.println("Exiting method: takeToken ringMemberImpl");  
 System.*out*.println("Token received: entering critical region");  
 }  
  
  
 public static void main(String argv[]) {  
 System.*setSecurityManager*(new SecurityManager());  
  
 // Checking that the required number of arguments have been input  
 // Exit program if not  
 if(argv.length != 4){  
 System.*out*.println("usage: [host] <id> [next host] <next id>");  
 System.*exit*(1);  
 }  
  
 // Extracting the arguments  
 String host = argv[0];  
 String id = argv[1];  
 String nHost = argv[2];  
 String nID = argv[3];  
  
 // instantiate ringMemberImpl class with appropriate parameters  
 // register object with RMI registry  
 try{  
 ringMember member = new ringMemberImpl(host, id, nHost, nID);  
 Naming.*bind*("//127.0.0.1/" + id, member);  
 System.*out*.println("------------------------------------------------------");  
 System.*out*.println("Ring node member " + id +" is bound with RMIRegistry");  
 System.*out*.println("------------------------------------------------------");  
 System.*out*.println(" ");  
 }catch(Exception e){  
 System.*err*.println(e);  
 }  
 }  
   
 private String next\_id;  
 private String next\_host;  
 private String this\_id;  
 private String this\_host;  
 private criticalSection c;  
 }

## 4.2 ringMember Interface

public interface ringMember extends java.rmi.Remote   
{  
   
 void takeToken(String file, tok token, String id\_longer\_CS, String skipped\_node\_id) throws java.rmi.RemoteException;  
  
}

## 4.3 criticalSection Class

import java.io.\*;  
import java.rmi.\*;  
import java.util.\*;  
  
public class criticalSection extends Thread {  
  
 public criticalSection(String t\_host, String t\_id, String n\_host, String n\_id, String file, tok token, String id\_longer\_CS, String skipped) {  
   
 this\_host =t\_host;  
 this\_id = t\_id;  
 next\_host = n\_host;  
 next\_id = n\_id;  
 this.file = file; // User specified file name  
 this.token = token; // The token object  
 this.rounds = 10; // Specified number of exchanges  
 this.id\_longer\_CS = id\_longer\_CS; // Node to have longer in Critical section  
 this.skipped\_node\_id = skipped; // node to be skipped  
  
 if(skipped\_node\_id != null){  
 this.skipping = true;  
 }else{  
 this.skipping = false;  
 }  
  
 try {  
 //Finding how many nodes are in the ring  
 this.listOfNodes = Naming.*list*("rmi://" + this\_host); //Array holding a list of the nodes  
 this.numOfNodes = listOfNodes.length; // variable which gets the length of this array  
 }catch (Exception e){  
 System.*err*.println(e);  
 }  
 }  
  
 public void run() {  
  
 int numOfToken = token.getToken();  
 if(skipping){  
 if (this\_id.equals(skipped\_node\_id)) {  
 boolean second\_visit = token.visit\_counter(this\_host, this\_id);  
 if (second\_visit) {  
 System.*out*.println("Second visit, moving to next node. Skipping node: " + this\_id);   
 release(next\_host, next\_id, numOfToken);  
 } else {  
 action(numOfToken);  
 }  
 } else {  
 action(numOfToken);  
 }  
 }else{  
 action(numOfToken);  
 }  
  
 }  
  
  
 */\*\*  
 \* Method to carry out the desired output to the file  
 \*  
 \** ***@param*** *token\_num This is the value of the current token being passed  
 \*/* public void action(int token\_num){  
 // sleep to symbolise critical section duration  
 try{  
 if(this\_id.equals(id\_longer\_CS)){  
 System.*out*.println("Node selected to have a longer Critical section: " + this\_id);  
 Thread.*sleep*(5000); // Longer sleep for node that gets longer in Critical section  
 }  
 else {  
 Thread.*sleep*(3000); // Default sleep  
 }  
 }catch (InterruptedException e){  
 System.*out*.println("Sleep Failed: " + e);  
 }  
 // write timestamp (date) to file  
 try{  
  
 System.*out*.println("Writing to file: " + file);  
 // Create a new date instance  
 Date timeStmp = new Date();  
 // Convert this date object into a string for printing to the shared file  
 String timeStamp = timeStmp.toString();  
  
 // create fileWriter - true means append to end of file  
 FileWriter fw\_id = new FileWriter(file, true);  
 // create printWriter - true means flush buffer on each println  
 PrintWriter pw\_id = new PrintWriter(fw\_id, true);  
 pw\_id.println("Record from ring node on host " + this\_host + ", id: " + this\_id +  
 ", is: " + timeStamp +", Token: " + token\_num);  
  
 pw\_id.close();  
 fw\_id.close();  
 token.setToken(token\_num);  
 System.*out*.println("Finished writing");  
  
 }catch (IOException e){  
 System.*out*.println(e);  
 }  
 // get remote reference to next ring element, and pass token on ...  
 release(next\_host, next\_id, token\_num);  
 }  
  
 */\*\*  
 \* Method which handles the exchange of the token between processes  
 \*  
 \** ***@param*** *host Next host in the ring  
 \** ***@param*** *id Next id in the ring  
 \** ***@param*** *tokenNum Current token number  
 \*/* public void release(String host, String id, int tokenNum){  
 ringMember member;  
  
 if(skipping){  
 numOfNodes = numOfNodes - 1;  
 }  
  
 // Checking that the token is under the specified number of exchanges  
 if(tokenNum < rounds){  
 try{  
 // checking that there are enough exchanges left for the node to be able have another critical Section  
 // if not, it will start the next process but this process will be terminated  
 if(tokenNum > (rounds - numOfNodes)){  
 System.*out*.println("Look up RMIRegistry with: rmi://localhost/"+id);  
 System.*out*.println("Connecting to next host with id: " + id);  
 // Finding the next node on the host by using its id  
 member = (ringMember)Naming.*lookup*("rmi://"+host+"/"+id);  
 System.*out*.println("Received Token count value is: " + tokenNum);  
 System.*out*.println("Token Release: Exiting Critical region");  
 System.*out*.println("------------------------------------------------------");  
 System.*out*.println(" ");  
 member.takeToken(file, token, id\_longer\_CS, skipped\_node\_id);  
 System.*out*.println("Finished");  
 System.*exit*(1); // Advanced feature 6: Clean up in an orderly fashion  
 }  
  
 System.*out*.println("Look up RMIRegistry with: rmi://localhost/"+id);  
 System.*out*.println("Connecting to next host with id: " + id);  
 // Finding the next node on the host by using its id  
 member = (ringMember)Naming.*lookup*("rmi://"+host+"/"+id);  
 System.*out*.println("Received Token count value is: " + tokenNum);  
 System.*out*.println("Token Release: Exiting Critical region");  
 System.*out*.println("------------------------------------------------------");  
 System.*out*.println(" ");  
 member.takeToken(file, token, id\_longer\_CS, skipped\_node\_id);  
  
  
 }catch(Exception e){  
 System.*out*.println(e);  
 }  
 }else{  
 System.*out*.println("Received Token count value is: " + tokenNum);  
 System.*out*.println("Token Release: Exiting Critical region");  
 System.*out*.println("------------------------------------------------------");  
 System.*out*.println(" ");  
 // Once the number of exchanges is reached  
 // Finish and exit program  
 System.*out*.println("Finished");  
 System.*exit*(1); // Advanced feature 6: Clean up in an orderly fashion  
 }  
  
 }  
  
  
 private String this\_id;  
 private String this\_host;  
 private String next\_id;  
 private String next\_host;  
 private String file;  
 private tok token;  
 private int rounds;  
 private String id\_longer\_CS;  
 private int numOfNodes;  
 private String[] listOfNodes;  
 private String skipped\_node\_id;  
 private boolean skipping;  
}

## 4.4 tok Class

import java.io.Serializable;  
  
/\*\*  
 \* Advanced Feature 1  
 \*  
 \* This is the token object  
 \*  
 \* It has been made serializable so it will not reset the token value every time it is instantiate  
 \* thus allowing for the token to increment with every exchange  
 \*  
 \*  
 \*/  
public class tok implements Serializable {  
  
 private int token;  
 private String[] tokens\_list;  
 private int[] visits;  
  
 /\*\*  
 \* Constructor  
 \*  
 \* **@param** t This is the value of the first token  
 \*/  
 public tok(int t) {  
  
 this.token = t;  
  
 }  
  
 /\*\*  
 \* Return the value of the current token  
 \*  
 \* **@return** \*/  
 public int getToken() {  
 return token;  
 }  
  
 /\*\*  
 \* When called the token is incremented by 1  
 \*  
 \* **@param** t This is the value of the token before it has been incremented  
 \*/  
 public void setToken(int t) {  
 token = t;  
 // Checking that the passed starting token is not negative  
 // if it is, inform user and exit program  
 if (token < 0) {  
 System.out.println("Token cannot be negative");  
 System.exit(1);  
 } else {  
 // If not negative, increment  
 token++;  
 }  
 }  
  
  
 /\*\*  
 \* Advanced Feature 5  
 \*  
 \* Method to increment the corresponding visit index to the correct node to  
 \* keep track of the number of visits each node has had  
 \*  
 \* **@param** host This is the host computer  
 \* **@param** id This is the id that is being visited  
 \* **@return** If it is the second visit, return true, otherwise return false  
 \*/  
 public boolean visit\_counter(String host, String id) {  
  
 int index;  
 boolean second\_visit = false;  
 String node = "//" + host + ":1099/" + id; // This is the structure of how the nodes are represented in the RMIRegistry for comparison  
 for(index = 0; index < tokens\_list.length; index++){  
 if(node.equals(tokens\_list[index])){  
 visits[index] += 1; // Increment the value at the index of the ID up by one  
 second\_visit = visits[index] % 2 == 0; // Checking that the value is even, if even it means its the second visit  
 }  
  
 }  
  
 return second\_visit;  
 }  
  
 /\*\*  
 \* Advanced Feature 5  
 \*  
 \* Method which initialises an array to track the number of visits to each of the nodes  
 \* in the ring  
 \*  
 \* **@param** nodes This is an array containing the ID's of all the nodes, this is used to set the length of the visits array  
 \*/  
 public void populate\_visits(String[] nodes){  
  
 int index;  
 this.tokens\_list = nodes;  
 this.visits = new int[tokens\_list.length];  
  
 for (index = 0; index < visits.length; index++){  
 visits[index] = 0; // Initialising all the visits to 0 at the start of the ringManager  
 }  
  
  
 }  
  
}

## 4.5 Java.policy File

grant {  
 permission java.net.SocketPermission "\*:1024-65535", "connect,accept,resolve,listen";  
 permission java.io.FilePermission "/-", "read,write,delete";  
};