/\*REFERENCE:

https://www.youtube.com/watch?v=GURClZeR96E&t=662s

https://www.javatpoint.com/Scanner-class

\*/

package com.sdsu.edu;

import java.io.\*;

import java.rmi.\*;

import java.rmi.registry.\*;

import java.rmi.server.UnicastRemoteObject;

import java.util.\*;

import java.util.concurrent.ConcurrentHashMap;

public class Server extends UnicastRemoteObject implements MessageInterface{

private static final long serialVersionUID = 1L;

//this variable has course name as key and advisor response as value for the particular key

ConcurrentHashMap<String,String> hm=new ConcurrentHashMap<String,String>();

protected Server() throws RemoteException {

super();

}

public static void main(String args[]){

try{

//Creates and exports a Registry instance on the local host that accepts requests on the specified port 1099

Registry reg= LocateRegistry.createRegistry(1099);

Server s=new Server();

//Replaces the binding for the specified name in this registry with vkbind

reg.rebind("vkbind", s);

System.out.println("server is ready...");

//this is used to fetch the data from the file which has the students course list

s.FetchFromFile();

//this is used to fetch the data from the file which has the students course as well as response list

s.FetchFromAdvisorFile();

}

catch(Exception e)

{

e.printStackTrace();

}

}

//this method is used to get the list of courses from file which are yet to be checked for the response from the advisor

public void FetchFromFile()

{

try{

//we open the file if it exists or create a new file

File inputFile = new File("/Users/varun/Downloads/Student.txt");

inputFile.createNewFile();

BufferedReader reader = new BufferedReader(new FileReader(inputFile));

//this variable is used to store the value of current line in the text file

String currentLine;

System.out.println("courses yet to be decided by advisor are ");

while((currentLine = reader.readLine()) != null) {

// trim newline when comparing with lineToRemove

String trimmedLine = currentLine.trim();

System.out.println(trimmedLine);

//we fetch the data that is already present in the textfile and save it in the hash map with value as null as they are yet to be reviewd by the advisor

hm.put(trimmedLine, "null");

}

reader.close();

}

catch(Exception e)

{

e.printStackTrace();

}

}

public void FetchFromAdvisorFile()

{

try{

File inputFile = new File("/Users/varun/Downloads/Advisor.txt");

inputFile.createNewFile();

BufferedReader reader = new BufferedReader(new FileReader(inputFile));

//this variable is used to store the value of current line in the text file

String currentLine;

System.out.println("responses yet to be notified are ");

while((currentLine = reader.readLine()) != null) {

// trim newline when comparing with lineToRemove

String trimmedLine = currentLine.trim();

String[] str=new String[2];

str=trimmedLine.split("%");

System.out.println("course name "+ str[0]+" decision is "+ str[1]);

//we fetch the data that is already present in the textfile and save it in the hash map with value

hm.put(str[0], str[1]);

}

reader.close();

}

catch(Exception e)

{

e.printStackTrace();

}

}

//this method is invoked by the student process to save the course name in queue

public void studentRequest(String name, String course) throws RemoteException

{

//WE SAVE THE COURSE IN HASH MAP WE STORE THE KEY AS COURSE NAME AND NULL AS VALUE

hm.put(course, "null");

//WE NEED TO STORE THE VALUE IN THE FILE FOR PERSISTENCY

BufferedWriter bw=null;

try

{

File yourFile = new File("/Users/varun/Downloads/Student.txt");

//IF THE FILE ALREADY EXISTS IT WONT DO ANYTHING ELSE IT WILL CREATE A NEW ONE

yourFile.createNewFile();

bw=new BufferedWriter(new FileWriter(yourFile,true));

bw.write(course);

bw.newLine();

} catch (IOException e) {

e.printStackTrace();

}

finally

{

try {

bw.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

//this method returns the queue which has the list of courses that needs to be waived

public Queue<String> advisorRequest() throws RemoteException

{

Queue<String> q1 = new LinkedList<String>();

//FROM THE HASH MAP WE GET THE KEY AND CHECK THOSE KEY WHOSE VALUE IS NULL AND SAVE THAT KEY TO A QUEUE AND SEND IT TO ADVISOR PROCESS

for ( String key : hm.keySet() ) {

if(hm.get(key).equals("null"))

q1.add(key);

}

return q1;

}

//this method saves the name as well as the advisors response in the hashmap

public void advisorResponse(String course, String response) throws RemoteException {

//we store the value of course name against the response in this variable

hm.put(course, response);

//in order to achieve data persistent we store the advisor response in a text file

BufferedWriter bw=null;

try {

File yourFile = new File("/Users/varun/Downloads/Advisor.txt");

yourFile.createNewFile();

bw=new BufferedWriter(new FileWriter(yourFile,true));

//we save the course name followed by "%" sign and then response

bw.write(course+"%"+response);

bw.newLine();

//since we already have the response of the course we use this method to delete the course from the text file that has list of courses

deleteCourse(course);

} catch (IOException e) {

e.printStackTrace();

}

finally

{

try {

bw.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

private void deleteCourse(String course) {

try{

File inputFile = new File("/Users/varun/Downloads/Student.txt");

File tempFile = new File("/Users/varun/Downloads/myTempFile.txt");

//we read the data from the student file and save it into temporary text file

BufferedReader reader = new BufferedReader(new FileReader(inputFile));

BufferedWriter writer = new BufferedWriter(new FileWriter(tempFile));

String lineToRemove = course;

String currentLine;

while((currentLine = reader.readLine()) != null) {

// trim newline when comparing with lineToRemove

String trimmedLine = currentLine.trim();

if(trimmedLine.equals(lineToRemove)) continue;

//apart from the course which needs to be deleted all the remaining courses are copied into the temporary file

writer.write(currentLine + System.getProperty("line.separator"));

}

writer.close();

reader.close();

boolean successful = tempFile.renameTo(inputFile);

System.err.println(successful);

}

catch(Exception e)

{

e.printStackTrace();

}

}

private void deleteResponse(String course) {

try{

File inputFile = new File("/Users/varun/Downloads/Advisor.txt");

File tempFile = new File("/Users/varun/Downloads/myTempFile2.txt");

//we read the data from the student file and save it into temporary text file

BufferedReader reader = new BufferedReader(new FileReader(inputFile));

BufferedWriter writer = new BufferedWriter(new FileWriter(tempFile));

String lineToRemove = course;

String currentLine;

while((currentLine = reader.readLine()) != null) {

// trim newline when comparing with lineToRemove

String trimmedLine = currentLine.trim();

if(trimmedLine.contains(lineToRemove)) continue;

//apart from the course which needs to be deleted all the remaining courses are copied into the temporary file

writer.write(currentLine + System.getProperty("line.separator"));

}

writer.close();

reader.close();

boolean successful = tempFile.renameTo(inputFile);

System.err.println(successful);

}

catch(Exception e)

{

e.printStackTrace();

}

}

//this method returns the hashmap to the notification process

public ConcurrentHashMap<String, String> getAdvisorResponse() throws RemoteException {

//this method is created so that the key value from current hashmap is copied to this variable and it will be deleted

ConcurrentHashMap<String, String> h2=new ConcurrentHashMap<String, String>();

//we use iterator to iterate on the global variable to access each key and delete the respective value

Iterator<String> it1 = hm.keySet().iterator();

while(it1.hasNext()){

//we get the key and store it in a variable

String key = it1.next();

//we remove the key value pair

if(!hm.get(key).equals("null"))

{

//we are copying the values from global variable to local variable

h2.put(key,hm.get(key));

hm.remove(key);

deleteResponse(key);

}

}

// we return the hashmap to the notification process

return h2;

}

}