



PRÁCTICA 4:

MULTIENVÍO NO ORDENADO EN

JAVA RMI

- José Ramón Martínez
- Víctor Pérez

Índice de contenido

1. Introduccion	3
2. Implementacion	4
2.1 GroupMember	4
2.2 GroupMessage	5
2.3 GroupServerInterface	5
2.4 ObjectGroup	6
2.5 ClientInterface	11
2.6 Cliente	11
2.7 GroupServer	17
2.8 SendingMessage	22
3. Resultados	23

Índice de figuras

Figura 1. Implemetacion de rmi	3
Figura 2. Ejemplo de callback	4
Figura 3. Constructor y atributos de la clase GroupMember	4
Figura 4. Clase GroupMessage	5
Figura 5. Interfaz GroupServerInterface	6
Figura 6. Atributos y constructor de la clase ObjetGroup	7
Figura 7. Método Sending	7
Figura 8. Método EndSending	7
Figura 9. Método sendGroupMessage	8
Figura 10. Método isMember	8
Figura 11. Getters y setters de alias, miembro y propietario	9
Figura 12. Método addMember	10
Figura 13. Método removeMember	10
Figura 14. Método ListMembers	11
Figura 15. Clase ClienteInterface	11
Figura 16. Atributos y constructor de la clase Client	12
Figura 17. Método DepositMessage	12
Figura 18. Método receiveGroupMessage	13
Figura 19. Método crearGrupo	13
Figura 20. Método EliminarGrupo	14
Figura 21. Método AñadirMiembro	14
Figura 22. Método EliminarMiembro	15
Figura 23. Método Terminar	15
Figura 24. Método MostrarGrupos	15
Figura 25. Método MostrarMiembros	16
Figura 26. Método Main de la clase Client	16
Figura 27. Menu del metodo main de clase Client	17
Figura 28. Atributos de la clase GroupServer	17
Figura 29. Constructor de la clase GroupServer	17
Figura 30. Método sendGroupMessage	18
Figura 31. Método createGroup	18
Figura 32. Método findGroup alias	19
Figura 33. Método findGroup por id	19
Figura 34. Método removeGroup	20
Figura 35. Método addMember	20
Figura 36. Método removeMember	21
Figura 37. Método isMember	21
Figura 38. Métodos ListMembers y ListGroup	22
Figura 39. Main de la clase GroupServer	22
Figura 40. Clase SendingMessage	23

1. Introduccion

En la anterior práctica (practica 3) implementamos un servidor de grupos de objetos distribuidos, los cuales nos permitían crear y eliminar grupos, dar de alta y baja a sus miembros, y también bloquear y desbloquear altas y bajas de sus miembros.

Además de todo lo anterior mencionado, en esta práctica (practica 4), se desarrolla los métodos necesarios para poder realizar envíos entre distintos miembros de un grupo. Esto envíos no deben ser ordenados por lo q no es necesaria la implementación de números de secuencia.

Para la realización de esta practica debemos utilizar una versión centralizada, de tal manera que un nodo tendrá la función de servidor y los clientes serán los que se conecten a este para solicitar sus servicios. Para la implementación de los mensajes entre miembros del mismo grupo, hemos de tener en cuenta que los clientes implementaran una cola de mensajes de los envíos realizados por un thread(hilo), esto permitirá envíos simultáneos dentro del mismo grupo.

Como conceptos de teoría debemos tener muy presentes como debe ser la implementación RMI (figura 1), la cual posee como partes más importantes un **módulo de comunicación** el cual es responsable del envío de los mensajes de petición (cliente) y respuesta (servidor), un **módulo de referencia remota**, el cual es el responsable de traducir las referencias entre objetos locales y remotos, generando las referencias a objetos remotos y el **proxy**, el cual Permite que el cliente vea los objetos remotos igual que los locales. Se encarga de transformar una invocación “local” en remota.

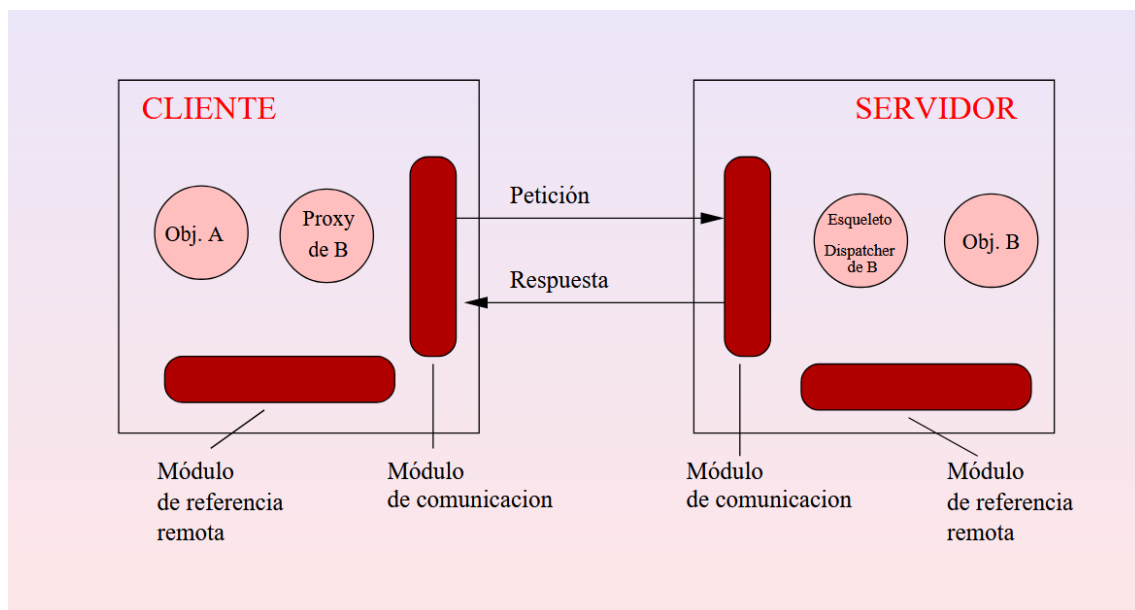


Figura 1. Implemetacion de rmi

Además de lo anterior explicado también se utilizarán callbacks(figura 2), el cual como su nombre indica es una retrollamada. Es una función “A” que se usa como argumento de otra función “B”. Cuando se llama a “B”, ésta ejecuta “A”. Para conseguirlo usualmente lo que se para a “B” es un puntero a “A”.

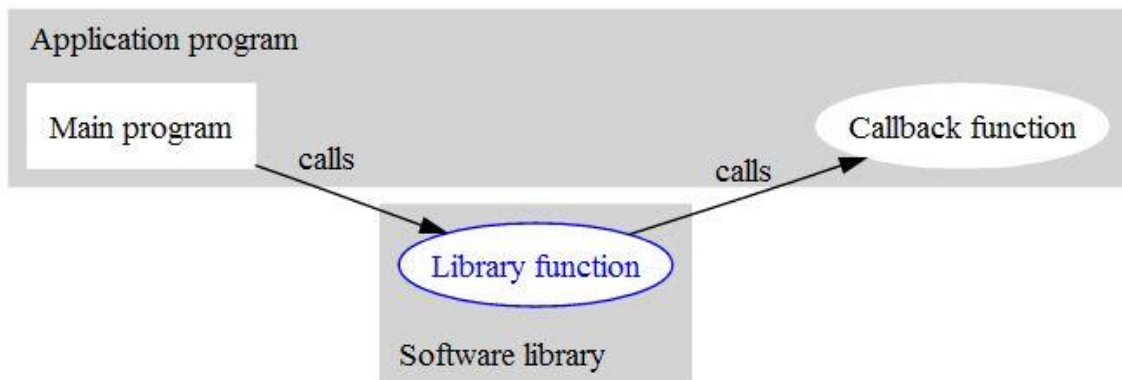


Figura 2. Ejemplo de callback

2. Implementacion

2.1 GroupMember

Esta clase serializable es la encargada de recoger la información perteneciente a cada miembro del grupo (alias del miembro, hostname, identificador numérico del usuario, identificador numérico del grupo y el puerto). Además, se crea el constructor correspondiente como podemos observar en la figura 3. Hemos implementado también los métodos set y get correspondientes.

```

public class GroupMember implements Serializable {
    String alias;
    String hostname;
    int idUser;
    int idGroup;
    int port;

    public GroupMember(String alias, String hostname, int idUser, int idGroup, int port) {
        this.alias = alias;
        this.hostname = hostname;
        this.idUser = idUser;
        this.idGroup = idGroup;
        this.port = port;
    }

    public boolean equals(Object o) {
        GroupMember m = (GroupMember) o;
        if (this.alias.equals(m.alias) && this.hostname.equals(m.hostname) &&
            this.idGroup == m.idGroup && this.idUser == m.idUser && this.port == m.port) return true;
        return false;
    }

    //SETTER:
    public void setAlias(String alias) {this.alias = alias;}

    public void setHostname(String hostname) {this.hostname = hostname;}

    public void setIdUser(int idUser) {this.idUser = idUser;}

    public void setIdGroup(int idGroup) {this.idGroup = idGroup;}

    public void setPort(int port) {this.port = port;}

    //GETTER:
    public String getAlias() {return alias;}

    public String getHostname() {return hostname;}

    public int getIdUser() {return idUser;}

    public int getIdGroup() {return idGroup;}

    public int getPort() {return port;}
}
  
```

Figura 3. Constructor y atributos de la clase GroupMember

2.2 GroupMessage

En esta clase serializable tendremos objetos que son mensajes de un grupo. Esta clase poseerá como atributos, un array de byter con el nombre mensaje y un GroupMember que será el emisor. Además de estos atributos tenemos el constructor correspondiente a esta clase además de los métodos set y get para mensaje y para emisor (figura 4).

```
package centralizedgroups;

import java.io.Serializable;

/**
 *
 * @author usuario
 */
public class GroupMessage implements Serializable{
    GroupMember emisor;
    byte[] mensaje;
    public GroupMessage(GroupMember emisor,byte[] mensaje){
        this.mensaje=mensaje;
        this.emisor=emisor;
    }
    public byte[] getMensaje() {
        return mensaje;
    }

    public void setMensaje(byte[] mensaje) {
        this.mensaje = mensaje;
    }

    public GroupMember getEmisor() {
        return emisor;
    }

    public void setEmisor(GroupMember emisor) {
        this.emisor = emisor;
    }
}
```

Figura 4. Clase GroupMessage

2.3 GroupServerInterface

Esta clase como se puede ver en la figura 5 es una interface remoto para el servidor de grupos. Los servicios que ofrezcan serán sus métodos. Lo métodos que posee serán los siguientes:

Un método para **crear un nuevo grupo**, para crear un nuevo grupo con identificador textual Galias, el propietario es su creador y se retorna el identificador numérico del nuevo grupo.

Un método para **eliminar un grupo** con identificador numérico.

Un método para **eliminar un grupo** con identificador textual.

Un método para **localizar el grupo** por su identificador textual.

Un método para **localizar un grupo** por su id.

Un método para **añadir un miembro** a un grupo.

Un método **comprobar si un miembro pertenece** a un grupo.

Un método **LinkedList** para la **lista de miembros**.

Un método **LinkedList** para la **lista de grupos**.

Un método para **enviar un mensaje** por un grupo.

2.4 ObjectGroup

Esta clase se usará para implementar cada grupo de objetos. Hemos usado `idUserContador` para asignar número de identificación diferentes a los nuevos miembros. Hemos implementado un `mutex lock` para garantizar la exclusión mutua gracias a `ReentrantLock`. Para el control de bloqueos de altas y bajas hemos implementado la variable boolean `stop` y la variable `Condition` inicializada con la variable `mutex`. El constructor y los métodos implementados son los siguientes:

```
package centralizedgroups;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.LinkedList;

/**
 *
 * @author viktitors
 */
public interface GroupServerInterface extends Remote{

    int createGroup(String galias, String oalias, String ohostname,int port) throws RemoteException;
    int findGroup(String galias) throws RemoteException;
    String findGroup(int gid)throws RemoteException;
    boolean removeGroup(String galias, String oalias) throws RemoteException;
    GroupMember addMember(String galias, String alias, String hostname,int port) throws RemoteException;
    boolean removeMember(String galias, String alias) throws RemoteException;
    GroupMember isMember(String galias, String alias) throws RemoteException;
    public LinkedList<String> ListMembers(String galias)throws RemoteException;
    public LinkedList<String> ListGroup()throws RemoteException;
    boolean sendGroupMessage(GroupMember gm, byte[] msg)throws RemoteException;

}
```

Figura 5. Interfaz `GroupServerInterface`

Constructor `ObjectGroup`: crea un objeto de grupo cuyo alias e identificador numérico se indica en los argumentos, también se inicializa el propietario. Además, debido a la funcionalidad del envío de mensajes es necesario pasar como parámetro el número de puerto. Todo ello recogido en la figura 6.

```

public class ObjectGroup {

    int gid;
    String alias;
    LinkedList<GroupMember> miembros = new LinkedList<GroupMember>();
    GroupMember propietario;
    int idUserContador = 0;
    boolean stop = false;
    final ReentrantLock mutex = new ReentrantLock(true);
    final Condition control = mutex.newCondition();

    private int numMembers=0;

    public ObjectGroup(int gid, String alias, String ualias, String uhostname,int port) {
        this.gid = gid;
        this.alias = alias;
        this.propietario = new GroupMember(ualias, uhostname, 0, gid, port);
        this.miembros.add(propietario);
    }
}

```

Figura 6. Atributos y constructor de la clase ObjectGroup

Void Sending(): controla el número de envíos que hay en curso. Todo ello se debe dar bajo exclusión mutua lo que permitirá controlar el bloqueo de altas/bajas como podemos observar en la figura 7.

```

public void Sending() {
    mutex.lock();
    try{
        numEnvios=this.miembros.size()-1;
        if(numEnvios>0) stop=true;
    }finally{
        mutex.unlock();
    }
}

```

Figura 7. Método Sending

Void EndSending(): notificar el final de un envío realizado. Se desbloquearan las altas y bajas de miembros del cuando no haya otros envíos en curso. Nuestra condición podrá lanzar un signalAll() cuando el contador alcance 0 como se puede observar en la figura 8.

```

public void EndSending() {
    mutex.lock();
    try{
        numMembers--;

        if(numMembers==0){
            stop=false;
            control.signalAll();
        }
    }finally{
        mutex.unlock();
    }
}

```

Figura 8. Método EndSending

SendGroupMessage(GroupMember gm, byte msg[]): es un metodo multienvio perteneciente al mensaje pasado como parámetro por gm, también pasado como parámetro, que indicará al miembro del grupo al que se le enviara el mensaje. Cuando termine de lanzar un hilo que envia el mensaje a todos los miembros del grupo, este método booleano devolverá true. Si hay envíos en curso, no se podrán resolver altas y bajas de miembro de dicho grupo, como podemos observar en la figura 9.

```
public boolean sendGroupMessage(GroupMember gm, byte msg[]) throws RemoteException{
    mutex.lock();
    //try {

        SendingMessage thread;
        GroupMessage mensaje;
        this.Sending();
        for(GroupMember miembro : this.miembros){
            if(!miembro.alias.equals(gm.alias)){
                mensaje=new GroupMessage(gm,msg);

                thread=new SendingMessage(miembro,this,mensaje);

            }
        }
        mutex.unlock();
        return true;
    }
}
```

Figura 9. Método sendGroupMessage

GroupMember isMember(String ualias): recorre la lista de miembros del grupo y si este coincide, lo devuelve, sino, devuelve null. Todo esto bajo exclusión mutua. Como se observa en la figura 10.

```
GroupMember isMember(String ualias) {
    mutex.lock();
    GroupMember miembro = null;
    Iterator it = miembros.iterator();
    while (it.hasNext()) {
        miembro = (GroupMember) it.next();
        if (miembro.alias.equals(ualias)) {
            mutex.unlock();
            return miembro;
        }
    }
    mutex.unlock();
    return null;
}
```

Figura 10. Método isMember

Get y Set para Gid ,Alias,miembro, propietario, recogidos en la figura 11

```

public int getGid() {
    return gid;
}

public void setGid(int gid) {
    mutex.lock();
    this.gid = gid;
    mutex.unlock();
}

public String getAlias() {
    return alias;
}

public void setAlias(String alias) {
    mutex.lock();
    this.alias = alias;
    mutex.unlock();
}

public LinkedList<GroupMember> getMiembros() {
    return miembros;
}

public void setMiembros(LinkedList<GroupMember> miembros) {
    mutex.lock();
    this.miembros = miembros;
    mutex.unlock();
}

public GroupMember getPropietario() {
    return propietario;
}

public void setPropietario(GroupMember propietario) {
    mutex.lock();
    this.propietario = propietario;
    mutex.unlock();
}

```

Figura 11. Getters y setters de alias, miembro y propietario

GroupMember addMember(String alias, String hostname, int port): se añade un miembro al cual se le asignara un nuevo identificador. Si ya existe un miembro con el alias indicado se retornará null. Además, el invocador si las inserciones y borrados de sus miembros están bloqueadas mediante la variable stop como se puede ver en la figura 12.

```

public GroupMember addMember(String alias, String hostname, int port) {
    mutex.lock();
    try {

        if(stop) {
            control.await();
        }

        if (this.isMember(alias) == null) {
            GroupMember nuevo = new GroupMember(alias, hostname, this.miembros.size()/*idUserContador++*/, gid, port);
            this.miembros.add(nuevo);
            System.out.println("Miembro añadido.");

            return nuevo;
        }
    } catch (InterruptedException ex) {
        Logger.getLogger(ObjectGroup.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
        mutex.unlock();
    }
    return null;
}

```

Figura 12. Método addMember

boolean removeMember(String alias): elimina del grupo al objeto indicado por el alias que es pasado como parámetro. No se podrá eliminar al propietario ni a un objeto que no pertenece a ese grupo, devolviendo null como se puede ver en la figura 13. Todo esto bajo exclusión mutua.

```

boolean removeMember(String alias) {
    mutex.lock();
    try {

        if(stop) {
            control.await();
        }

        GroupMember miembro = this.isMember(alias);
        if( (miembro != null) && !(propietario.alias.equals(alias)) ) {

            return miembros.remove(miembro);
        }

    } catch (InterruptedException ex) {
        Logger.getLogger(ObjectGroup.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
        mutex.unlock();
    }
    return false;
}

```

Figura 13. Método removeMember

LinkedList<String> ListMembers(): permite recorrer la lista de miembros y añadir su correspondiente alias, como se puede ver en la figura 14.

```

public LinkedList<String> ListMembers() {
    mutex.lock();
    LinkedList<String> listamiembros=new LinkedList<String>();
    for(GroupMember miembro : this.miembros){
        listamiembros.add(miembro.alias);
    }
    mutex.unlock();
    return listamiembros;
}

```

Figura 14. Método ListMembers

2.5 ClientInterface

Es una interfaz que será usada en la clase Client. Por ello añadimos dos nuevos métodos como se puede observar en la figura 15.

```

package centralizedgroups;

import java.rmi.Remote;
import java.rmi.RemoteException;

/**
 *
 * @author viktitors
 */
public interface ClientInterface extends Remote {
    void DepositMessage(GroupMessage m)throws RemoteException;
    byte[] receiveGroupMessage(String galias)throws RemoteException;
}

```

Figura 15. Clase ClientInterface

2.6 Cliente

Esta clase es la utilizada para que el cliente pida su alias. Además, se abrirá un menú con las diferentes opciones sobre el servidor. Cuando creamos un nuevo grupo, el cliente invocador se añadirá como propietario de ese grupo. Además, es el main de esta clase será donde se indicará la política de seguridad del cliente. En la figura 16 podemos ver los atributos de esta clase y el constructor

```

public class Client extends UnicastRemoteObject implements ClientInterface {

    //auxiliar attributes:
    int id; // Identificador del grupo
    static GroupServerInterface srv;
    String alias = "", galias = "", ualias = "", hostname = "";

    //client attributes:
    String aliasusuario, aliasgrupo;
    int port;
    String serverip;

    //utilities:
    public Scanner in;
    LinkedList<GroupMessage> queue=new LinkedList<GroupMessage>();
    final ReentrantLock mutex = new ReentrantLock(true);
    final Condition control = mutex.newCondition();

    public Client() throws RemoteException {
        super();
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }
        System.out.println("Introduce el alias del cliente:");
        in = new Scanner(System.in);
        this.aliasusuario = in.next();

        System.out.println("Introduce el puerto del cliente:");
        in = new Scanner(System.in);
        this.port=in.nextInt();

        System.out.println("Introduce la ip del servidor:");
        in = new Scanner(System.in);
        this.serverip=in.next();
    }
}

```

Figura 16. Atributos y constructor de la clase Client

public void DepositMessage(GroupMessage m): un callback usado para depositar los mensajes m pasado como parámetro en la cola local del cliente. Debe garantizar exclusión mutua. Se muestra en la figura 17.

```

public void DepositMessage(GroupMessage m) throws RemoteException{
    mutex.lock();
    try{
        queue.add(m);

        control.signal();
    }finally{
        mutex.unlock();
    }
}

```

Figura 17. Método DepositMessage

byte[] receiveGroupMessage(String galias): recoge de la cola local el siguiente mensaje correspondiente al grupo cuyo alias hemos indicado como parámetro. Si no hay ninguno, este se bloqueara hasta la llegada de alguno, y si no existe el grupo devolverá null. Se muestra en la figura 18.

```

public byte[] receiveGroupMessage(String alias) throws RemoteException{
    mutex.lock();
    try {

        int existegrupo=srv.findGroup(alias);
        byte[]msg=null;
        if(existegrupo==-1) return null;

        while (true) {

            for(GroupMessage m : this.queue){
                if(m.emisor.idGroup==(existegrupo)){

                    msg=m.mensaje;
                    this.queue.remove(m);
                    return msg;
                }
            }

            control.await();

        }
    } catch (RemoteException ex) {
        Logger.getLogger(Client.class.getName()).log(Level.SEVERE, null, ex);
        return null;
    } catch (InterruptedException ex) {
        Logger.getLogger(Client.class.getName()).log(Level.SEVERE, null, ex);
        return null;
    }
    finally{
        mutex.unlock();
    }
}

```

Figura 18. Método receiveGroupMessage

void crearGrupo(): Permite crear un grupo mediante la llamada al método createGroup(), como se muestra en la figura 19

```

public void crearGrupo() throws UnknownHostException {
    try {
        System.out.println("Pon nombre de grupo");

        aliasgrupo = in.next();
        id = srv.createGroup(this.aliasgrupo, this.aliasusuario, InetAddress.getLocalHost().getHostAddress(),this.port);
        if (id != -1) {
            System.out.println("Hemos creado tu grupo, de id: " + id);
        }
    } catch (RemoteException ex) {
        Logger.getLogger(Client.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```

Figura 19. Método crearGrupo

void EliminarGrupo(): Permite borrar un grupo mediante la llamada al método removeGroup(), como se muestra en la figura 20

```

public void EliminarGrupo() {
    try {
        System.out.println("Pon nombre del dueño");
        ualias = in.next();
        System.out.println("Pon nombre del grupo");
        galias = in.next();
        if (srv.removeGroup(galias, ualias)) {
            System.out.println("Borrado con éxito");
        }
    } catch (RemoteException ex) {
        Logger.getLogger(Client.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```

Figura 20. Método EliminarGrupo

void AñadirMiembro(): Permite añadir miembros al grupo mediante la llamada al método `addMember()` si es que este no pertenecía ya al grupo, como se muestra en la figura 21

```

public void AñadirMiembro() {
    try {
        System.out.println("Pon alias de grupo");
        galias = in.next();
        System.out.println("Pon nombre de usuario");
        ualias = in.next();
        System.out.println("Pon nombre de host");
        hostname = in.next();
        System.out.println("Pon puerto de usuario");
        port = in.nextInt();

        if (srv.addMember(galias, ualias, hostname, port) != null) {
            System.out.println("Añadido con éxito");
        } else {
            System.out.println("Ya existía");
        }
    } catch (RemoteException ex) {
        Logger.getLogger(Client.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```

Figura 21. Método AñadirMiembro

void EliminarMiembro(): Permite eliminar miembros al grupo mediante la llamada al método `removeMember()` si es que este pertenece al grupo, como se muestra en la figura 22

```

public void EliminarMiembro() {
    try {
        System.out.println("Pon nombre de usuario");
        ualias = in.next();
        System.out.println("Pon nombre de grupo");
        galias = in.next();
        if (!srv.removeMember(galias, ualias)) {
            System.out.println("Ya existía");
        } else {
            System.out.println("Todo bien");
        }
    } catch (RemoteException ex) {
        Logger.getLogger(Client.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```

Figura 22. Método EliminarMiembro

void Terminar(): Permite darse de baja como servidor para ello deberá utilizar el método `unexportObject`, , como se muestra en la figura 23

```

public void Terminar() {
    try {
        UnicastRemoteObject.unexportObject(this, true);
    } catch (NoSuchObjectException ex) {
        Logger.getLogger(Client.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```

Figura 23. Método Terminar

void MostrarGrupos(): lista todos los grupos utilizando un bucle for y llamando al método `ListGroup()`, como se muestra en la figura 24

```

public void MostrarGrupos() {
    try {
        System.out.println("Lista de Grupos:");
        for (String grupo : srv.ListGroup()) {
            System.out.println(grupo);
        }
    } catch (RemoteException ex) {
        Logger.getLogger(Client.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```

Figura 24. Método MostrarGrupos

void MostrarMiembros(): lista todos los miembros de un grupo utilizando un bucle for y llamando al método `ListMembers()`, como se muestra en la figura 25.


```

public void MostrarMiembros() {
    try {
        System.out.println("Introduce el nombre del grupo a listar: ");
        galias=in.next();
        System.out.println("Lista de miembros del grupo: "+galias);
        for(String miembro : srv.ListMembers(galias)){
            System.out.println(miembro);
        }
    } catch (RemoteException ex) {
        Logger.getLogger(Client.class.getName()).log(Level.SEVERE, null, ex);
    }
}
}

```

Figura 25. Método MostrarMiembros

void main(): este método consiste de varias partes:

- Se establece la policy a usar indicando el archivo en el proyecto.
 - Se crea un objeto cliente.
 - Se crea un registro del objeto remoto, este permite llamadas por el puerto remoto.
 - Utilizamos un binder el cual conecta con rmiregistry en este ordenador e informa que objeto remoto debe publicarse al exterior.
 - Creamos y obtenemos el registro de la ip del servidor y el puerto.
 - Buscamos el proxy del objeto remoto, buscándolo por nombre.
 - Devolvemos la referencia para el objeto remoto (servidor) con su nombre específico. Todo lo mencionado hasta ahora se puede observar en la **figura 26**.
-
- Creamos el menú que se mostrara con las opciones correspondientes como se puede observar en la **figura 27**.

```

public static void main(String[] args) throws UnknownHostException, MalformedURLException {
    System.setProperty("java.security.policy", "src\\centralizedgroups\\cliente-policy");

    try {
        Client c = new Client();

        int serverPort=1099;

        Registry reg = LocateRegistry.getRegistry(c.serverip, serverPort);
        srv=(GroupServerInterface) reg.lookup("GroupServer");

        LocateRegistry.createRegistry(c.port);
        Naming.rebind("rmi://" + InetAddress.getLocalHost().getHostAddress() + ":" + c.port + "/" + c.aliasusuario, c);

        System.out.println("Cliente conectado desde: " + InetAddress.getLocalHost().getHostAddress() + ":" + c.port);
    }
}

```

Figura 26. Método Main de la clase Client

```

while (i < 10) {
    System.out.println("1. Crear grupo");
    System.out.println("2. Eliminar grupo");
    System.out.println("3. Añadir miembro");
    System.out.println("4. Eliminar miembro");
    System.out.println("5. Recibir mensajes");
    System.out.println("6. Enviar mensaje");
    System.out.println("7. Terminar");
    System.out.println("8. Mostrar lista de miembros del grupo");
    System.out.println("9. Mostrar lista de grupos");
    int s=c.in.nextInt();
    String nombregrupo;
    byte[] msg;
    switch (s) {
        case 1://1. Crear grupo
            c.crearGrupo(); break;
        case 2://2. Eliminar grupo
            c.eliminarGrupo();break;
        case 3://3. Añadir miembro
            c.añadirMiembro(); break;
        case 4://4. Eliminar miembro
            c.eliminarMiembro();break;
        case 5://5. recibir mensajes
            System.out.println("Introduzca el nombre del grupo:");
            nombregrupo=c.in.next();
            msg=c.receiveGroupMessage(nombregrupo);
            if(msg==null){System.out.println("No hay mensajes en el grupo: "+nombregrupo);}else{System.out.println("Mensaje: "+new String(msg));}break;
        case 6: //6. enviar mensaje
            System.out.println("Introduzca el nombre del grupo: ");
            nombregrupo=c.in.next();System.out.println("Introduce el mensaje: ");
            msg=c.in.next().getBytes();
            GroupMember gm= srv.isMember(nombregrupo, c.aliasusuario);System.out.println("Es: "+ c.aliasusuario+" miembro del grupo: "+nombregrupo+" ?");
            if(gm!=null){ if(c.srv.sendGroupMessage(gm,msg))System.out.println("Mensaje enviado");
            else System.out.println("Error al enviar");}
            else System.out.println("Estas intentando enviar un mensaje a un grupo al que no perteneces");break;
        case 7:c.terminar();
            i = 10;break;
        case 8:
            c.mostrarMiembros(); break;
        case 9:
            c.mostrarGrupos();break;}
    i++;
}

```

Figura 27. Menu del metodo main de clase Client

2.7 GroupServer

Mediante esta clase se llevará la implementación del servidor extendido de `UnicastRemoteObject` y se implementará la interfaz `GroupServerInterface`. Para ello como variables hemos utilizado una lista de grupos mediante la utilización de la clase `LinkedList`, también se ha utilizado una variable `ReentrantLock` para garantizar la exclusión mutua. Además, hemos utilizado también un id del grupo inicializado en 0, como se muestra en la figura 28.

```

public class GroupServer extends UnicastRemoteObject implements GroupServerInterface {

    LinkedList<ObjectGroup> objetos = new LinkedList<ObjectGroup>();
    ReentrantLock lock = new ReentrantLock(true);
    int idGroup = 0;
}

```

Figura 28. Atributos de la clase GroupServer

public GroupServer(): constructor que hereda todas las características de su clase padre, como se puede observar en la figura 29.

```

public GroupServer() throws RemoteException {
    super();
    lock.lock();
    if (System.getSecurityManager() == null) {
        System.setSecurityManager(new SecurityManager());
    }
    lock.unlock();
}
}

```

Figura 29. Constructor de la clase GroupServer

boolean sendGroupMessage(GroupMember gm, byte[] msg): si el id del grupo coincide con el id del grupo del miembro, este llama al método SenGroupMessage(gm, msg) para enviar el mensaje al grupo determinado(pasado como parámetro). Si no encuentra el grupo devuelve false. Todo ello se puede observar en la figura 30.

```
public boolean sendGroupMessage(GroupMember gm, byte[] msg) throws RemoteException{
    lock.lock();
    System.out.println(gm.alias+" IS SENDING A MESSAGE");
    if ((this.findGroup(gm.idGroup))==null){
        lock.unlock();
        return false;
    }

    for(ObjectGroup grupo : this.objetos){
        if(grupo.gid==gm.idGroup){
            lock.unlock();
            return grupo.sendGroupMessage(gm, msg);
        }
    }
    lock.unlock();
    return false;
}
```

Figura 30. Método sendGroupMessage

int createGroup(String galias, String oalias, String ohostname,int port): crea el grupo con los datos pasados como parámetro. Devuelve el id de dicho grupo creado, como se puede observar en la figura 31.

```
@Override
public int createGroup(String galias, String oalias, String ohostname,int port) {
    lock.lock();
    System.out.println("CREATE GROUP");
    System.out.println("Creando grupo "+galias+"...");
    if (this.findGroup(galias) == -1) {
        ObjectGroup grupo = new ObjectGroup(++idGroup, galias, oalias, ohostname,port);
        objetos.add(grupo);
    }
    lock.unlock();
    System.out.print("Creado.");
    return idGroup;
}
```

Figura 31. Método createGroup

int findGroup(String galias): busca el grupo por su alias, devolviendo el id de este si lo encuentra, -1 cuando no, como se puede observar en la figura 32.

```

@Override
public int findGroup(String galias) {
    lock.lock();
    System.out.println("FIND GROUP: "+galias);
    try{
        System.out.println("Buscando grupo "+galias+"...");
        for (ObjectGroup grupo : objetos) {
            if (grupo.alias.equals(galias)) {
                System.out.print("Encontrado grupo "+galias+".");
                return grupo.getGid();
            }
        }
        System.out.println("Grupo "+galias+" no encontrado");
        return -1;
    }finally{
        lock.unlock();
    }
}

```

Figura 32. Método findGroup alias

String findGroup(int gid): busca el grupo por su id, devolviendo el alias de este si lo encuentra, null cuando no, como se puede observar en la figura 33.

```

@Override
public String findGroup(int gid) {
    System.out.println("FIND GROUP: "+gid);
    try{
        lock.lock();
        System.out.println("Buscando grupo "+gid+"...");
        for (ObjectGroup grupo : objetos) {
            if (grupo.gid==gid) {
                System.out.print("Encontrado grupo "+gid+".");
                return grupo.getAlias();
            }
        }
        System.out.println("Grupo "+gid+" no encontrado");
        return null;
    }finally{
        lock.unlock();
    }
}

```

Figura 33. Método findGroup por id

boolean removeGroup(String galias, String oalias): busca un grupo, y si soy propietario lo elimina usando el método remove. Si lo borra devuelve true, sino false, como se puede observar en la figura 34.

```

@Override
public boolean removeGroup(String galias, String oalias) {
    lock.lock();
    System.out.println("REMOVE GROUP: "+galias);
    System.out.println("Eliminado grupo "+galias+"...");
    int gdelete = this.findGroup(galias);
    if (gdelete != -1) {
        for (ObjectGroup grupo : objetos) {
            if (grupo.getGid() == gdelete && grupo.getPropietario().alias.equals(oalias)) {
                objetos.remove(grupo);
                System.out.print(" Grupo "+galias+" eliminado.");
                lock.unlock();
                return true;
            }
        }
        System.out.println("Existe pero el dueño no es ese.");
    }
    lock.unlock();
    return false;
}

```

Figura 34. Método removeGroup

GroupMember addMember(String galias, String alias, String hostname,int port): comprueba el grupo y si el miembro no esta ya en el grupo, en ese caso devuelve un GroupMember añadiéndolo mediante la llamada al método addMember(), sino devuelve null, como se pude observar en la figura 35.

```

@Override
public GroupMember addMember(String galias, String alias, String hostname,int port) {
    lock.lock();
    System.out.println("ADD MEMBER: "+alias+ " TO GROUP: "+galias);
    System.out.print("Añadiendo miembro "+alias+"...");
    for (ObjectGroup grupo : objetos) {
        if (grupo.alias.equals(galias) && grupo.isMember(alias) == null) {
            lock.unlock();

            return grupo.addMember(alias, hostname,port);
        }
    }
    lock.unlock();
    return null;
}

```

Figura 35. Método addMember

boolean removeMember(String galias, String alias): comprueba el grupo y si el alias corresponde con un miembro, en ese caso devuelve un true y borra al miembro mediante la llamada al método removeMember(), sino devuelve false, como se pude observar en la figura 36.

```

@Override
public boolean removeMember(String galias, String alias) {
    lock.lock();
    System.out.println("REMOVE MEMBER: "+alias+ " FROM GROUP: "+galias);
    System.out.println("Eliminado miembro "+alias+"...");
    for (ObjectGroup grupo : objetos) {
        if (grupo.alias.equals(galias) && grupo.isMember(alias) != null) {
            lock.unlock();
            return grupo.removeMember(alias);
        }
    }
    lock.unlock();
    return false;
}

```

Figura 36. Método removeMember

GroupMember isMember(String galias, String alias): comprueba si el miembro pertenece al grupo, buscando el grupo y si este es miembro devuelve un objeto GroupMember usando el metodo isMember(alias), como se pude observar en la figura 37.

```

@Override
public GroupMember isMember(String galias, String alias) {
    lock.lock();
    System.out.println("IS MEMBER: "+alias+ " FROM GROUP: "+galias+" ?");
    System.out.println("Comprobando pertenencia miembro "+alias+" a grupo "+galias+"...");
    int groupid=findGroup(galias);
    if(groupid!=-1){
        for(ObjectGroup grupo : objetos){
            if (grupo.gid == groupid) {
                if(grupo.isMember(alias)==null)System.out.println("No es miembro");
                else System.out.println("Es miembro.");
                lock.unlock();
                return grupo.isMember(alias);
            }
        }
    }
    lock.unlock();
    return null;
}

```

Figura 37. Método isMember

LinkedList<String> ListMembers(String galias) y LinkedList<String>ListGroup(): permite listar los miembros de un grupo y los grupos que existen respectivamente, como se puede observar en la figura 38.

```
//MUESTRA TODOS LOS MIEMBROS DE UN GRUPO
public LinkedList<String> ListMembers(String galias){
    System.out.println("LIST MEMBERS FROM GROUP: "+galias);
    System.out.println("Miembros del grupo "+galias+": \n");
    for (ObjectGroup grupo : objetos) {
        if (grupo.getAlias().equals(galias)) {
            return grupo.ListMembers();
        }
    }
    return new LinkedList<String>();
}

public LinkedList<String>ListGroup() {
    System.out.println("LIST GROUPS: ");
    LinkedList<String> grupos= new LinkedList<String>();
    System.out.println("Lista de grupos:");
    for (ObjectGroup grupo : objetos) {
        grupos.add(grupo.getAlias());
    }
    return grupos;
}
}
```

Figura 38. Métodos ListMembers y ListGroup

void main(String[] args) void main(String[] args): es el método de la clase y consta de varias partes descritas a continuación, además podemos observar todo el método main en la **figura 39**.

- Se establece la policy a usar indicando el archivo en el proyecto.
- Creamos el servidor de tipo GroupServer.
- Creamos el registro con el puerto deseado y obtenemos el registro creado.
- Bindeamos o rebindeamos si ya existe el objeto remoto puesto en el registro.

```
public static void main(String[] args) throws UnknownHostException, MalformedURLException {
    System.setProperty("java.security.policy", "src\\centralizedgroups\\server-policy");
    try {
        GroupServer srv = new GroupServer();
        String ip=InetAddress.getLocalHost().getHostAddress();//IP
        int port = 1099;
        LocateRegistry.createRegistry(port);
        Naming.rebind("GroupServer",srv);

        System.out.println("Servidor iniciado en: "+ip+" : "+port);

    } catch (RemoteException ex) {
        Logger.getLogger(GroupServer.class.getName()).log(Level.SEVERE, null, ex);
    }
}
}
```

Figura 39. Main de la clase GroupServer

2.8 SendingMessage

Es una extensión de la clase thread para enviar mensajes a los miembros de un grupo, a excepción del emisor.

Los argumentos requeridos se encuentran en el constructor y su método run es el encargado de los envíos, invocando el método de *DepositMessage* de los destinatarios. Cuando se terminen todos los envíos se invoca el método *EndSending* del grupo. Podemos observar también que no se contempla un envío fiable/ordenado, si falla algún envío, este no se tiene en cuenta. Todo esto se puede observar en la **figura 40**

```
public class SendingMessage extends Thread {
    GroupMessage mensaje;
    ObjectGroup grupo;
    GroupMember miembro;

    ClientInterface destinatario;

    public SendingMessage(GroupMember miembro, ObjectGroup grupo, GroupMessage mensaje) {
        this.mensaje=mensaje;
        this.miembro=miembro;
        this.grupo=grupo;
        this.start();
    }

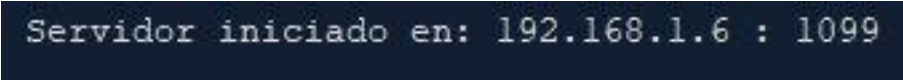
    @Override
    public void run(){
        System.setProperty("java.security.policy", "src\\centralizedgroups\\server-policy");
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager()); //gestor de seguridad
        }

        try {
            destinatario=(ClientInterface)Naming.lookup("rmi://" +miembro.hostname+" "+miembro.port+"/"+miembro.alias);
            if(destinatario!=null){
                Thread.sleep((int) (Math.random() * (60000-30000)) + 30000);
                destinatario.DepositMessage(mensaje);
            }
        } catch (RemoteException | NotBoundException ex) {
            Logger.getLogger(SendingMessage.class.getName()).log(Level.SEVERE, null, ex);
            System.out.println("Excepción en el thread.run()");
        } catch (InterruptedException ex) {
            Logger.getLogger(SendingMessage.class.getName()).log(Level.SEVERE, null, ex);
        }
        grupo.EndSending();
    } catch (MalformedURLException ex) {
        Logger.getLogger(SendingMessage.class.getName()).log(Level.SEVERE, null, ex);
    }
}
}
```

Figura 40. Clase SendingMessage

3. Resultados

1. Iniciamos el servidor



```
Servidor iniciado en: 192.168.1.6 : 1099
```

2. Iniciamos el cliente1


```
Introduce el alias del cliente:  
clientel  
Introduce el puerto del cliente:  
3000  
Introduce la ip del servidor:  
192.168.1.6  
Cliente conectado desde: 192.168.1.6:3000
```

3. Iniciamos el cliente 2

```
Introduce el alias del cliente:  
cliente2  
Introduce el puerto del cliente:  
4000  
Introduce la ip del servidor:  
192.168.1.6  
Cliente conectado desde: 192.168.1.6:4000
```

4. Cliente 1 crea un grupo

```
Elija una accion:  
1. Crear grupo  
2. Eliminar grupo  
3. Añadir miembro  
4. Eliminar miembro  
5. Recibir mensajes  
6. Enviar mensaje  
7. Terminar  
8. Mostrar lista de miembros del grupo  
9. Mostrar lista de grupos  
1  
Pon nombre de grupo  
gr  
Hemos creado tu grupo, de id: 1
```

5. Cliente 1 añade a cliente 2 al grupo creado

```
3
Pon alias de grupo
gr
Pon nombre de usuario
cliente2
Pon nombre de host
192.168.1.6
Pon puerto de usuario
4000
Añadido con éxito
```

6. Cliente 2 envía mensaje

```
6
Introduzca el nombre del grupo:
gr
Introduce el mensaje:
hola
Es: cliente2 miembro del grupo: gr ?
Mensaje enviado
```

7. Cliente 1 recibe el mensaje

```
5
Introduzca el nombre del grupo:
gr
Mensaje: hola
```