Desarrollo de Aplicaciones Distribuidas

AUTORES:
Alejandro Calderón Mateos
Javier García Blas
David Expósito Singh
Laura Prada Camacho

Departamento de Informática
Universidad Carlos III de Madrid
Julio de 2012

JAVA RMI:
CALLBACK DE CLIENTE

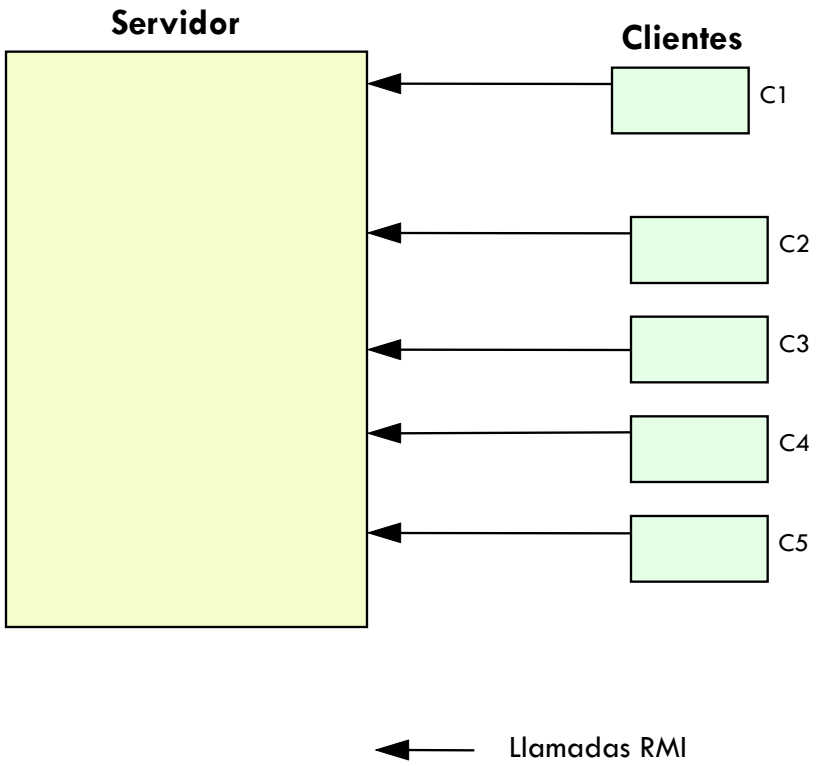
Contenidos

1. Introducción:
   1. *Callback* de cliente
2. *Callback* de cliente en Java RMI
   1. Arquitectura del sistema
   2. Elementos a desarrollar
3. Ajustes en desarrollo y despliegue
   1. Descarga del resguardo
   2. Políticas de seguridad

Contenidos

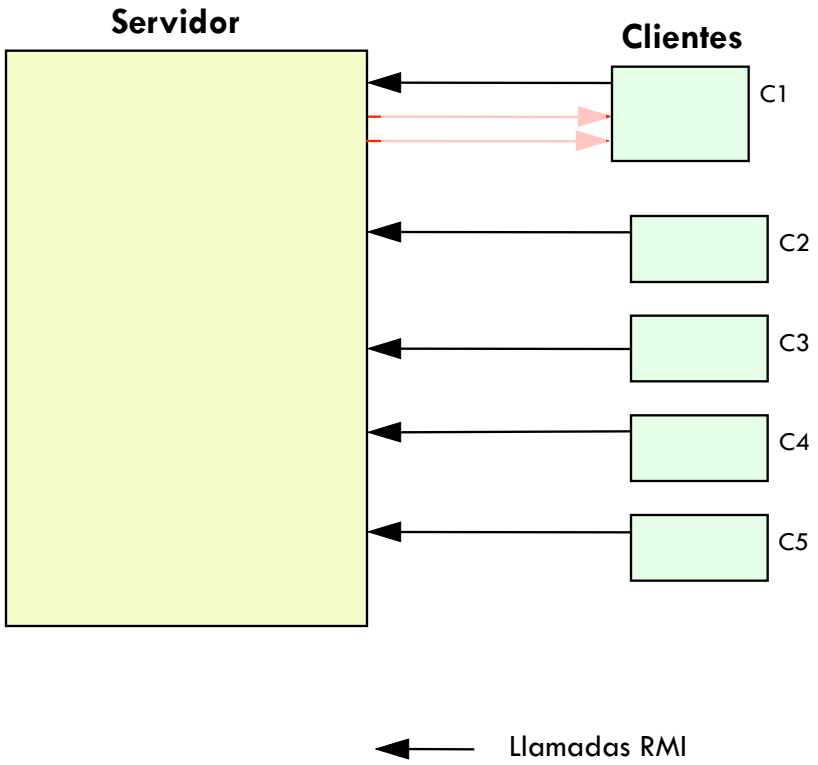
1. **Introducción:**
   1. ***Callback* de cliente**
2. *Callback* de cliente en Java RMI
   1. Arquitectura del sistema
   2. Elementos a desarrollar
3. Ajustes en desarrollo y despliegue
   1. Descarga del resguardo
   2. Políticas de seguridad

Callback de cliente



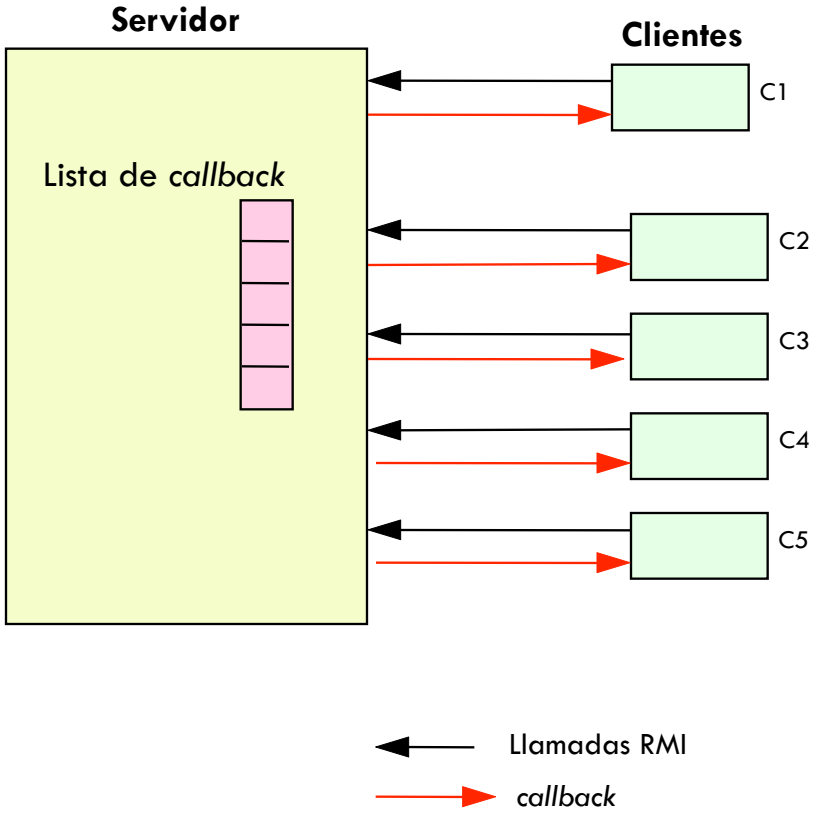
- En el modelo cliente-servidor el servidor es pasivo
- El cliente pide un servicio y el servidor devuelve unos resultados como respuesta en un plazo de tiempo

Callback de cliente



- ¿Qué pasa cuando el cliente quiere una o varias respuestas de forma asíncrona?
- Ejemplo: apuntarse para ser informados de los cambios en la bolsa

Callback de cliente

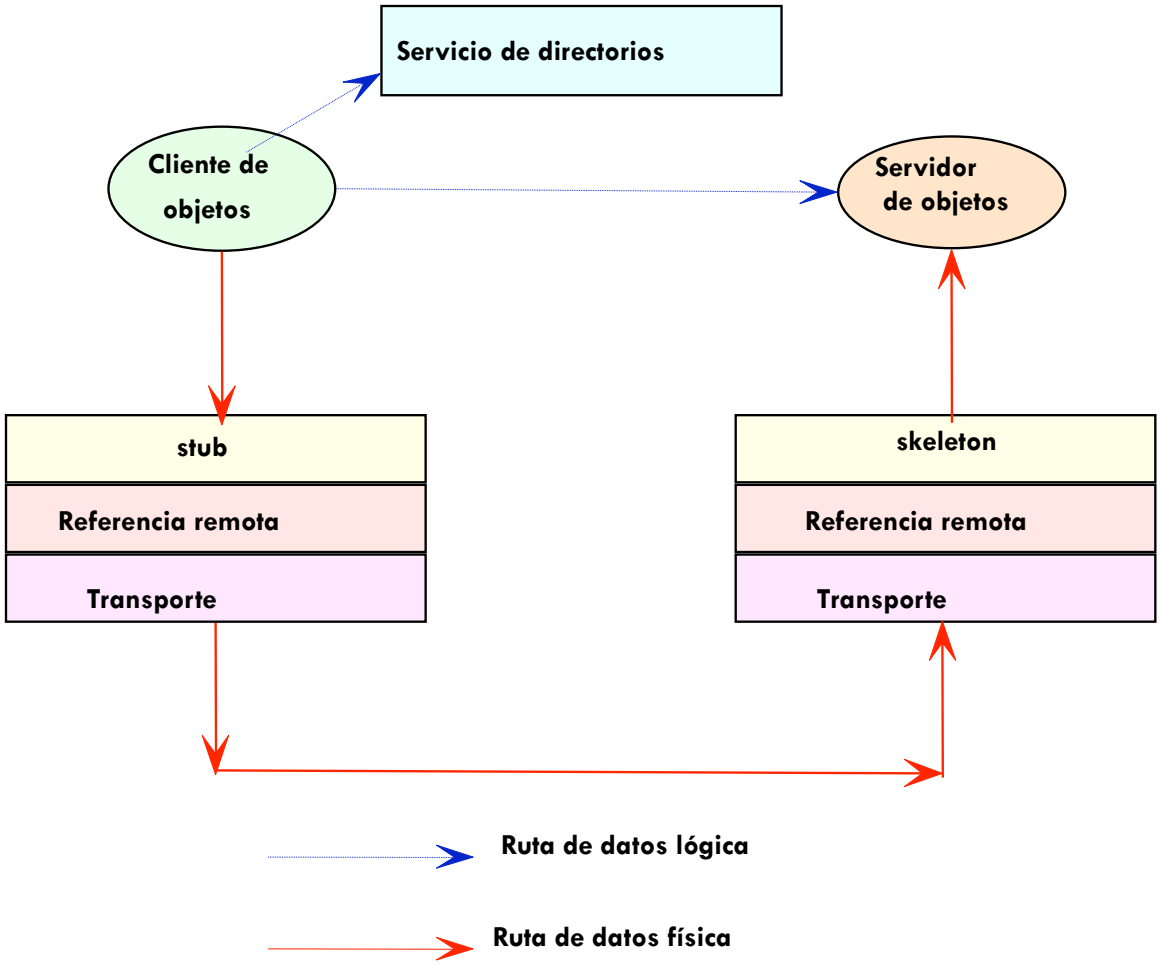


- Alternativas:
  - *Polling*
    - *Preguntar cada cierto tiempo*
    - *Peticiones sin cambios causan sobrecarga*
  - *Callback*
    - *Registrarse en el servidor para que nos avise ante los eventos*

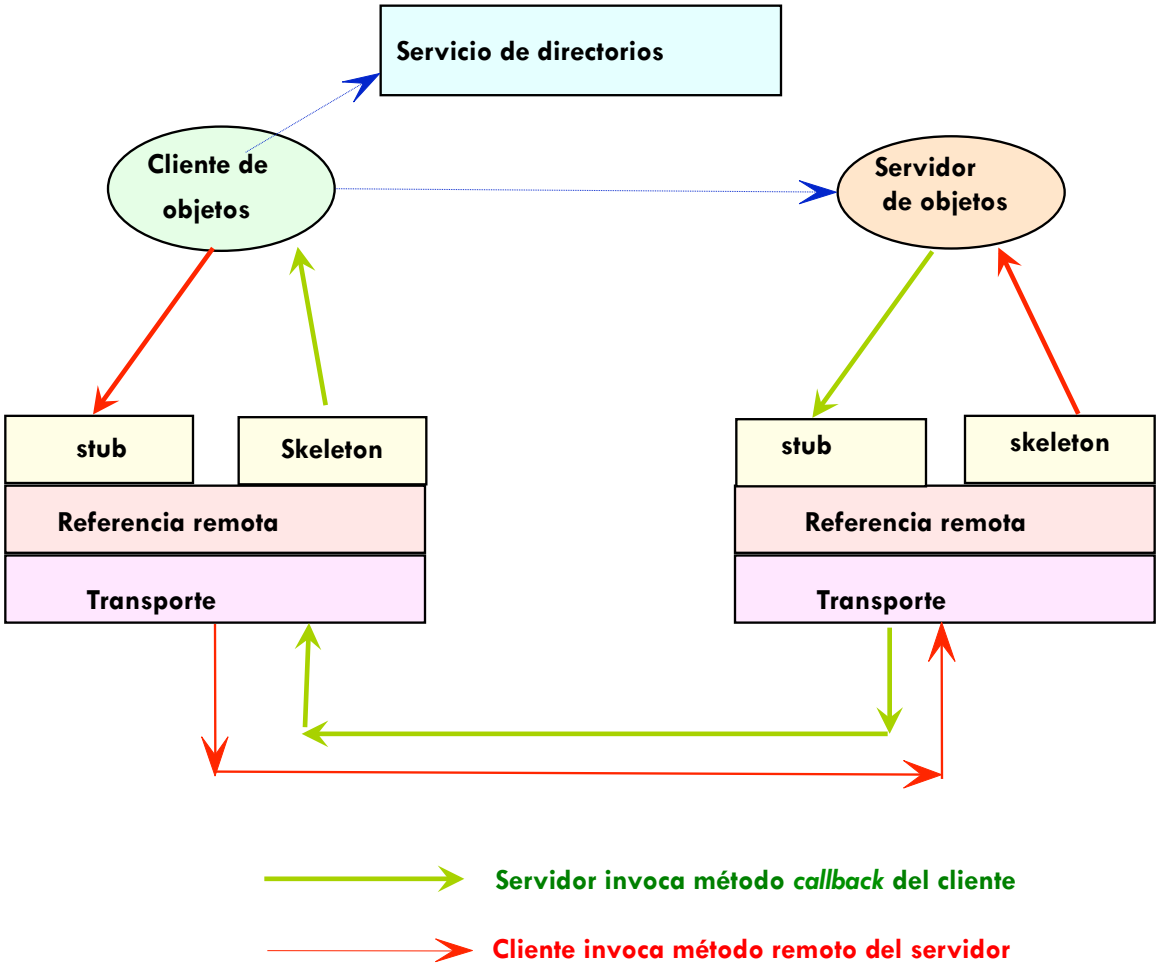
Contenidos

1. Introducción:
   1. *Callback* de cliente
2. *Callback* de cliente en Java RMI
   1. **Arquitectura del sistema**
   2. Elementos a desarrollar
3. Ajustes en desarrollo y despliegue
   1. Descarga del resguardo
   2. Políticas de seguridad

Arquitectura de RMI (no *callback*)



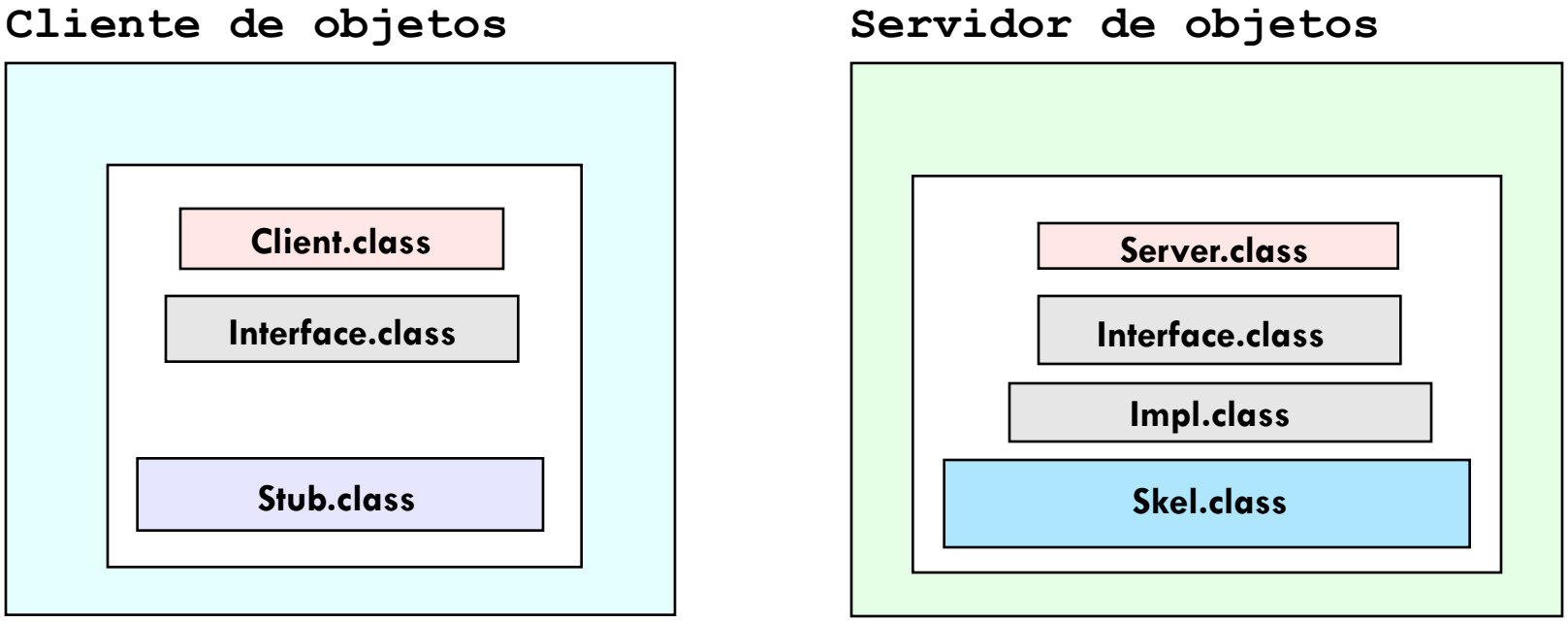
Arquitectura de RMI (*callback*)



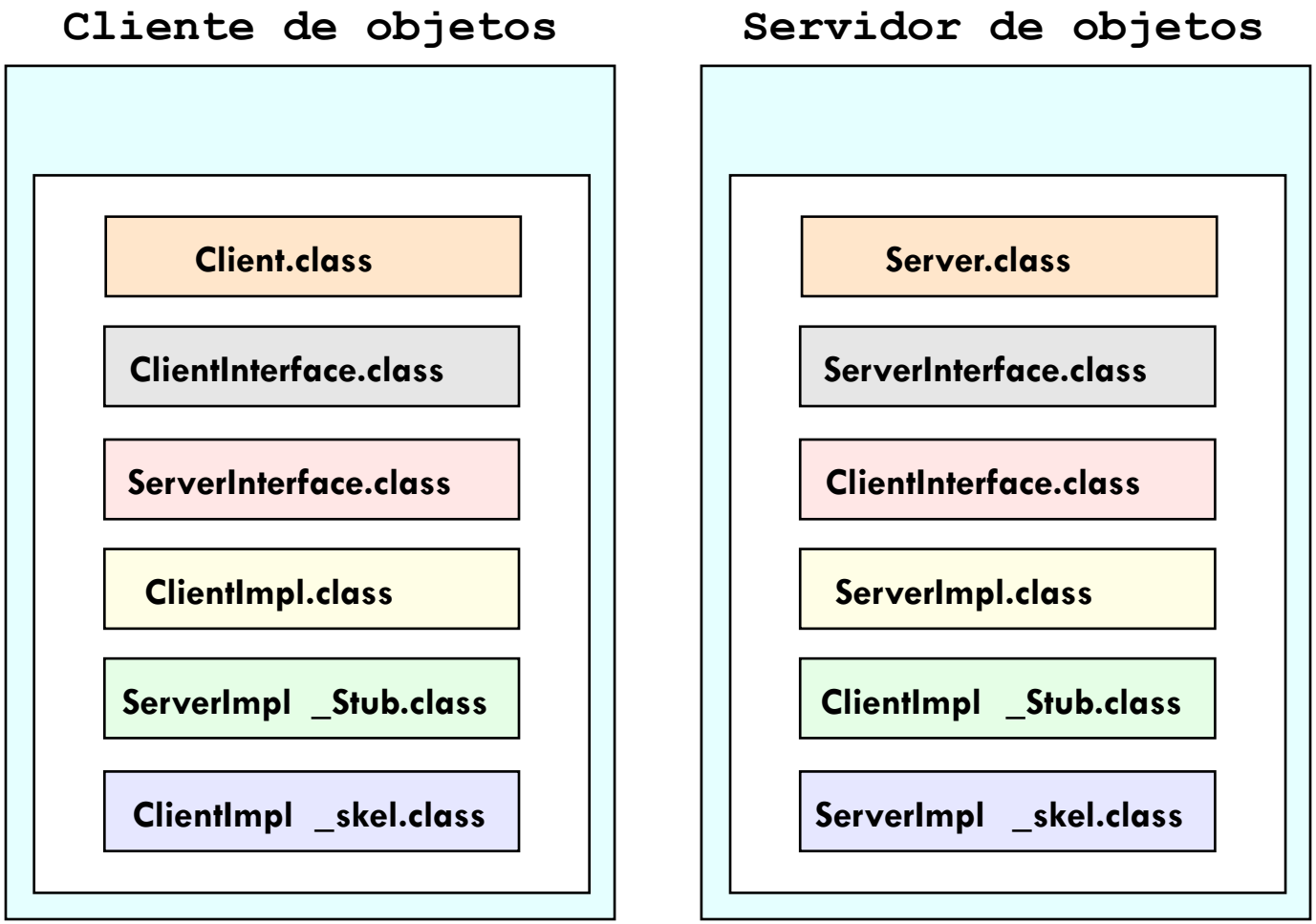
Contenidos

1. Introducción:
   1. *Callback* de cliente
2. *Callback* de cliente en Java RMI
   1. Arquitectura del sistema
   2. **Elementos a desarrollar**
3. Ajustes en desarrollo y despliegue
   1. Descarga del resguardo
   2. Políticas de seguridad

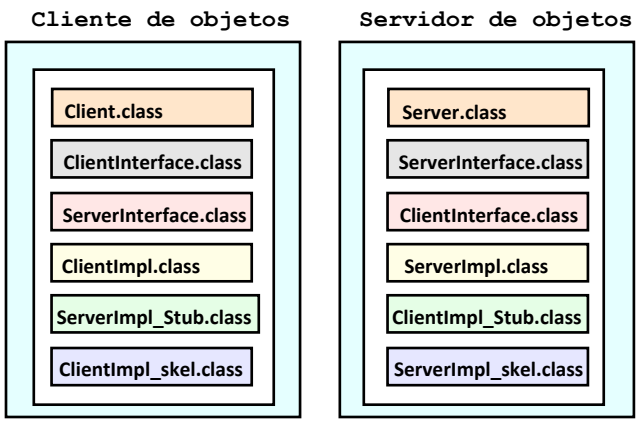
Invocación remota (no *callback*)



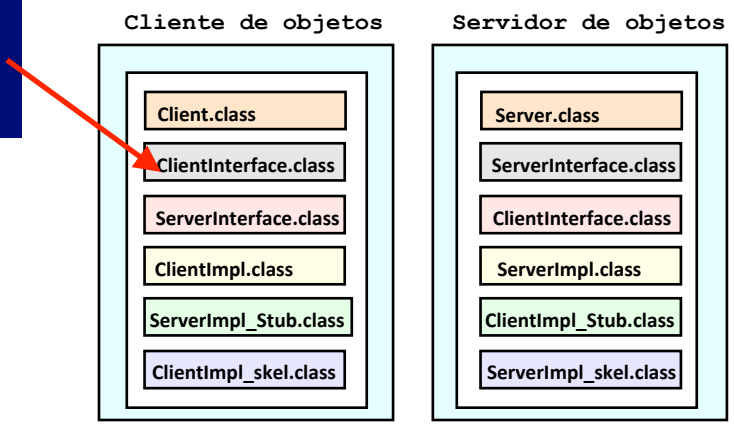
Invocación remota (*callback*)



Invocación remota (*callback*)



- Los clientes se registran como servidor de objeto remoto para *callbacks*.
- Dos conjuntos de *proxies*.
- Piezas clave:
  - Interface remota de cliente.
  - Servicio de registro de interfaces de cliente en servidor



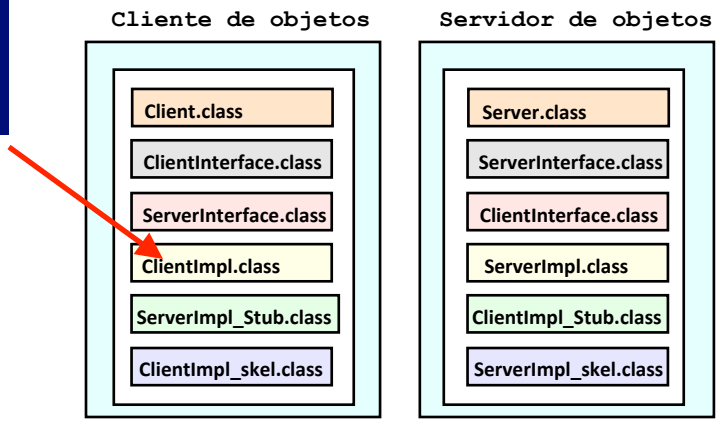
ClientInterface.java

- Interfaz remota del cliente
- Debe contener, al menos,
  un método para ser invocado por el servidor

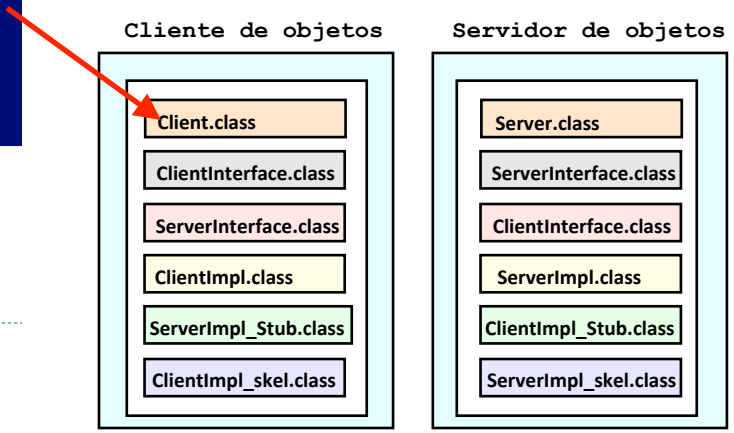
```
import java.rmi.*;

public interface ClientInterface extends java.rmi.Remote
{
    public String notifyMe ( String message )
            throws java.rmi.RemoteException ;
}
```

ClientImpl.java

- Implementación de la Interfaz remota del cliente
- Generación the proxies con *rmic ClientImpl*



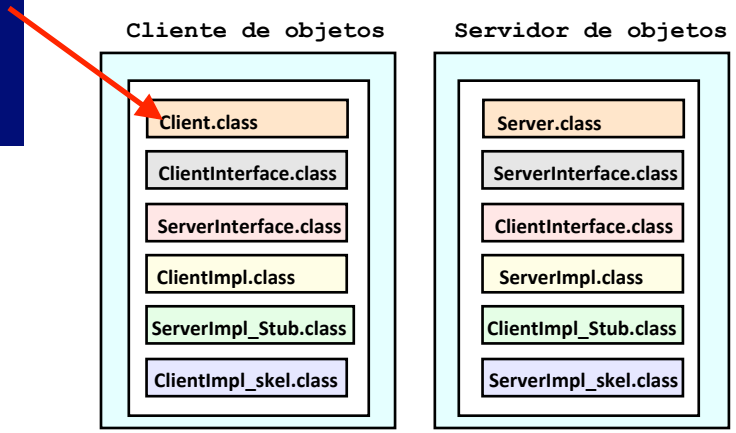

```
import java.rmi.*;
import java.rmi.server.*;

public class      ClientImpl
       extends    UnicastRemoteObject
       implements ClientInterface
{
   public ClientImpl() throws RemoteException
   {  super( );  }

   public String notifyMe ( String message )  {
      String returnMessage = "Call back received: " + message;
      System.out.println(returnMessage);
      return returnMessage;
   }
}
```



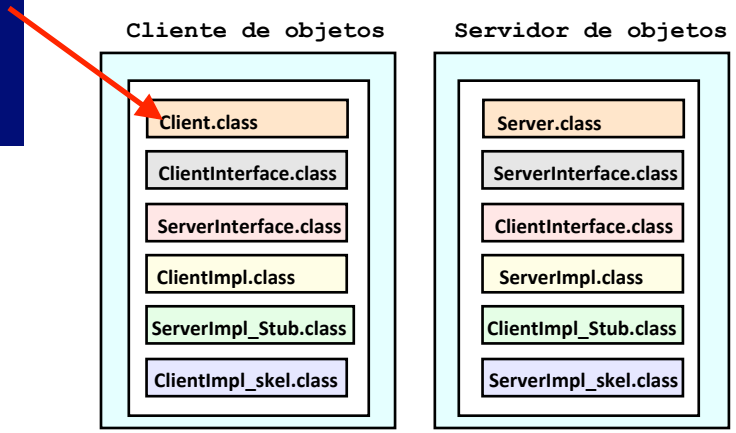
Client.java (1/3)

```
import java.io.*;
import java.rmi.*;

public class Client
{
  public static void main(String args[])
  {
    try
    {
      InputStreamReader is = new InputStreamReader(System.in);
      BufferedReader br = new BufferedReader(is);
      System.out.println("Enter the RMIRegistry host namer:");
      String hostName = br.readLine();
      System.out.println("Enter the RMIregistry port number:");
      String portNum = br.readLine();
      System.out.println("Enter how many seconds to stay registered:");
      String timeDuration = br.readLine();
      int time = Integer.parseInt(timeDuration);
```

Client.java (2/3)




```
String regURL = "rmi://" + hostName + ":" + portNum  + "/callback";
ServerInterface h = (ServerInterface)Naming.lookup(regURL);
System.out.println("Lookup completed ");
System.out.println("Server said " + h.sayHello());

ClientInterface callbackObj = new ClientImpl();
h.registerForCallback(callbackObj);
System.out.println("Registered for callback.");
```

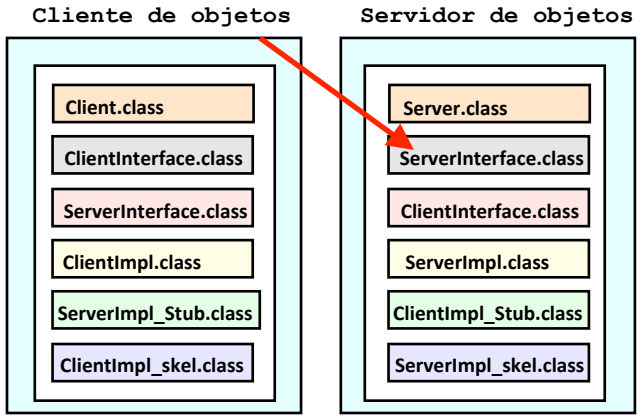


Client.java (3/3)

```
      try {
         Thread.sleep(time * 1000);
         h.unregisterForCallback(callbackObj);
         System.out.println("Unregistered for callback.");
      } catch (InterruptedException ex){ /* sleep over */ }

    } // try
    catch (Exception e)
    {
       System.out.println("Exception in CallbackClient: " + e);
    }

  } // main
} // class
```

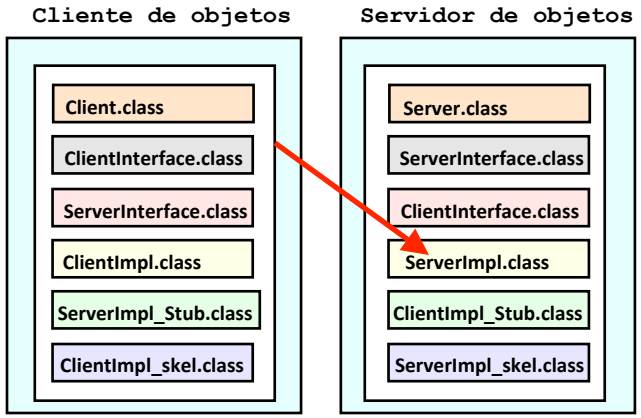
ServerInterface.java

- Extensión en la parte servidora
- Método remoto de registro del cliente para *callback*



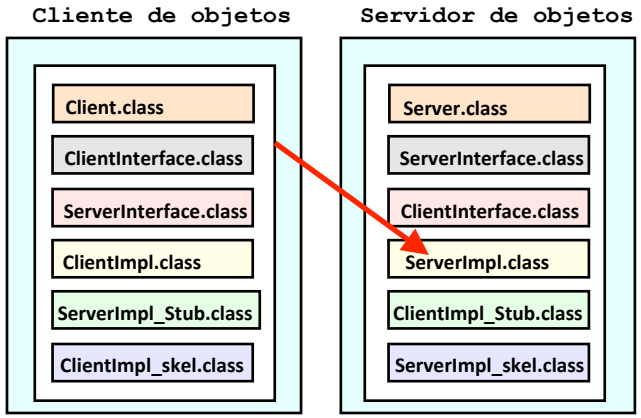

```
import java.rmi.*;

public interface ServerInterface extends Remote
{
   public String sayHello( )
          throws java.rmi.RemoteException;

   public void registerForCallback(
       ClientInterface callbackClientObject
   ) throws java.rmi.RemoteException;

   public void unregisterForCallback(
       ClientInterface callbackClientObject
   ) throws java.rmi.RemoteException;
}
```

ServerImpl.java (1/4)



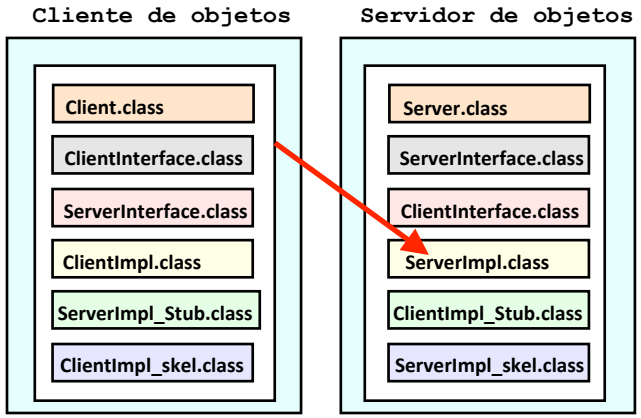

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.Vector;

public class      ServerImpl
       extends    UnicastRemoteObject
       implements ServerInterface {

   private Vector clientList;
   public ServerImpl() throws RemoteException {
      super( );
      clientList = new Vector();
   }

   public String sayHello( ) throws RemoteException {
      return("hello");
   }
```

ServerImpl.java (2/4)



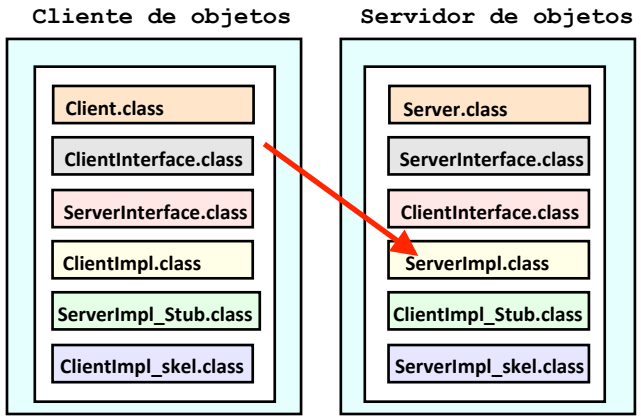

```
public synchronized void unregisterForCallback (
    ClientInterface callbackClientObject
) throws RemoteException
{
    if (clientList.removeElement(callbackClientObject)) {
        System.out.println("Unregistered client. ");
    } else {
        System.out.println("unregister: client wasn't registered.");
    }
}
```

ServerImpl.java (3/4)

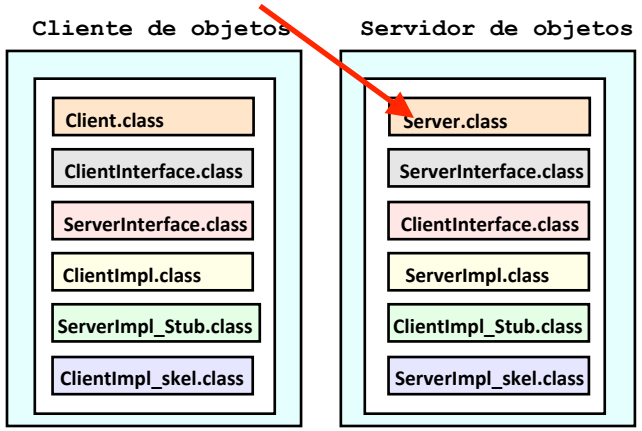



```
public synchronized void registerForCallback(
    ClientInterface callbackClientObject
) throws RemoteException
{
   if (!(clientList.contains(callbackClientObject)))
   {
      clientList.addElement(callbackClientObject);
      doCallbacks();
   }
}
```

ServerImpl.java (4/4)



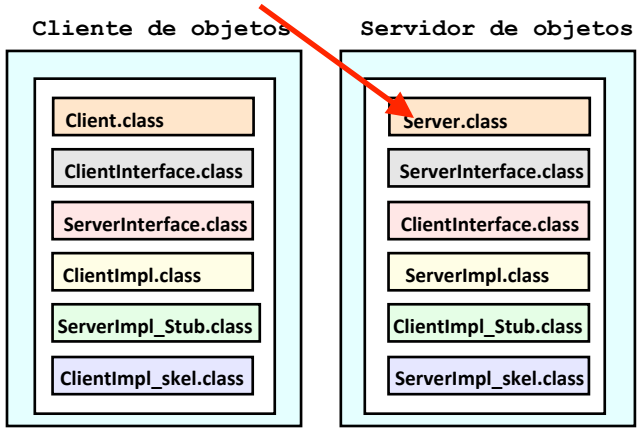

```
  private synchronized void doCallbacks( ) throws RemoteException
  {
    for (int i = 0; i < clientList.size(); i++)
    {
      System.out.println("doing "+ i +"-th callback\n");
      ClientInterface nextClient =
              (ClientInterface) clientList.elementAt(i);
      nextClient.notifyMe("Number of registered clients="
                          + clientList.size());
    } // for
  } // function

} // class
```



Server.java (1/3)

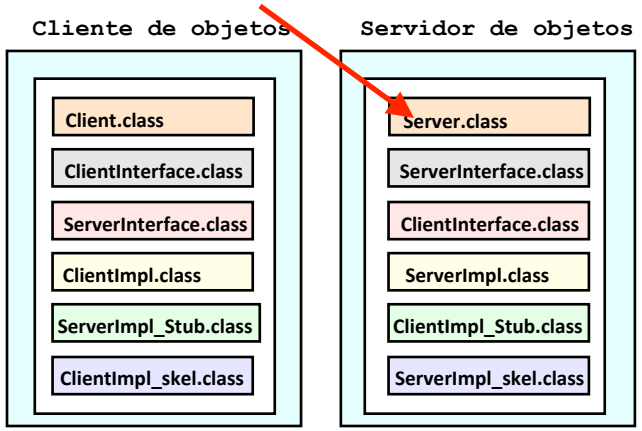
```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.net.*;
import java.io.*;


public class Server
{
   public static void main(String args[])
   {
     InputStreamReader is = new InputStreamReader(System.in);
     BufferedReader    br = new BufferedReader(is);
```

Server.java (2/3)




```
    String portNum, registryURL;
    try
    {
      System.out.println("Enter the RMIregistry port number:");
      portNum = (br.readLine()).trim();
      int RMIPortNum = Integer.parseInt(portNum);
      startRegistry(RMIPortNum);
      ServerImpl exportedObj = new ServerImpl();
      registryURL = "rmi://localhost:" + portNum + "/callback";
      Naming.rebind(registryURL, exportedObj);
      System.out.println("Callback Server ready.");
    } catch (Exception re) {
      System.out.println("Exception in HelloServer.main: " + re);
    }
} // end main
```

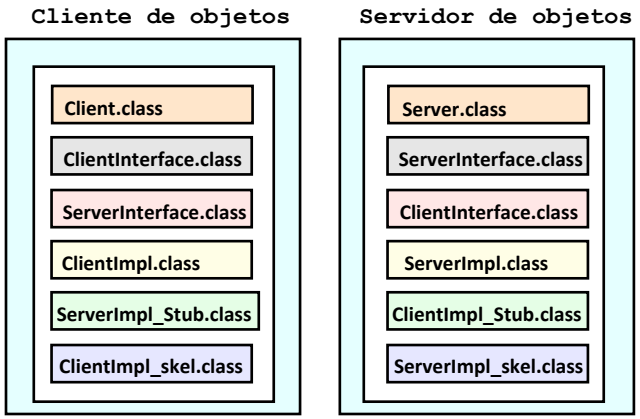
Server.java (3/3)




```
  private static void startRegistry (int RMIPortNum)
  throws RemoteException
  {
      try {
        Registry registry = LocateRegistry.getRegistry(RMIPortNum);
        registry.list( );
      } catch (RemoteException e) {
        // No valid registry at that port.
        Registry registry = LocateRegistry.createRegistry(RMIPortNum);
      }
  } // end startRegistry

} // end class
```

Compilación del ejemplo

guernika.lab.inf.uc3m.es




```
# javac  -cp /usr/lib/jvm/java-1.4.2-gcj-4.1-1.4.2.0/jre/lib/rt.jar  \
          -g  ClientInterface.java ClientImpl.java

# rmic ClientImpl

# javac -cp /usr/lib/jvm/java-1.4.2-gcj-4.1-1.4.2.0/jre/lib/rt.jar -g  *.java
```

Ejecución del ejemplo

guernika.lab.inf.uc3m.es

```
# rmiregistry 9090  &

# java -Djava.security.policy=policy.all  Server
Enter the RMIregistry port number:
9090
Callback Server ready.
^Z
# bg

# java -Djava.security.policy=policy.all  Client
Enter the RMIRegistry host namer:
localhost
Enter the RMIregistry port number:
9090
Enter how many seconds to stay registered:
60
Lookup completed
Server said hello
doing 0-th callback
Call back received: Number of registered clients=1
Registered for callback.
…
```

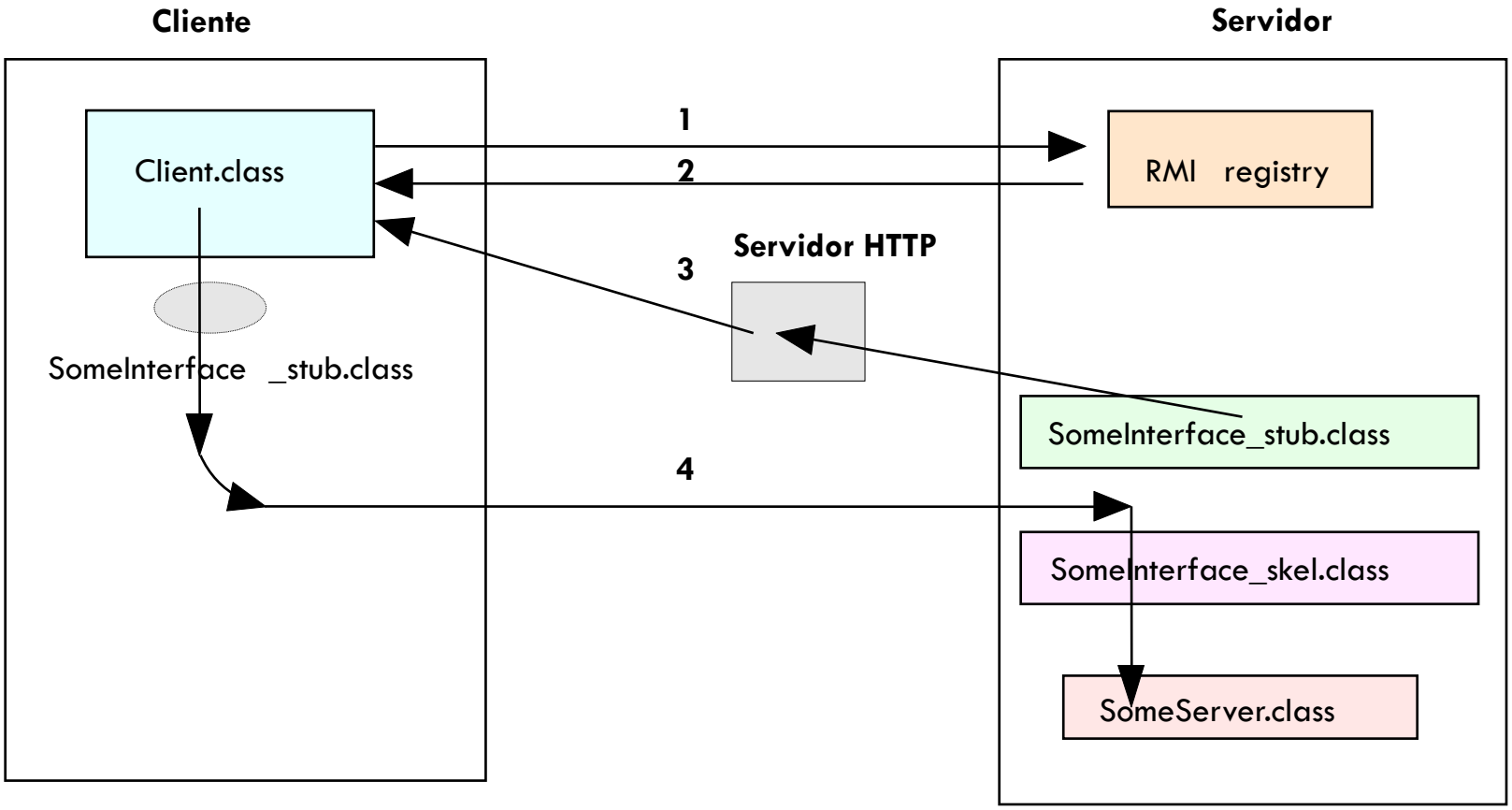
Contenidos

1. Introducción:
   1. *Callback* de cliente
2. *Callback* de cliente en Java RMI
   1. Arquitectura del sistema
   2. Elementos a desarrollar
3. **Ajustes en desarrollo y despliegue**
   1. **Descarga del resguardo**
   2. **Políticas de seguridad**

Descarga del resguardo

- Objetivo: eliminar la necesidad de que el cliente deba tener el *stub*.
- El *stub* es descargado dinámicamente desde un servidor web.
- Es necesario conocer la URL del servidor.
- La clase resguardo no es persistente.

Descarga del resguardo



Gestor de seguridad (por defecto)

- Security Manager

- Supervisión del programa:
  - Acceso a ficheros
  - Conexiones de red

- Nivel de seguridad inicial:
  - Únicamente conexiones locales.
  - No es posible la transferencia del resguardo.

Gestor de seguridad RMI

- Es posible usar una gestor de seguridad que configurado adecuadamente permita la descarga del resguardo.
  - Clase `RMISecurityManager`
  - Fichero de política de seguridad

Gestor de seguridad RMI

- Inicio del servicio de seguridad:

```
try {
   System.setSecurityManager(
        new RMISecurityManager());
} catch {
  //…
}
```

  - Añadir el anterior fragmento de código tanto al programa principal (*main*) del cliente como del servidor.
  - Usar antes de utilizar los servicios del registro RMI.

Gestor de seguridad RMI

- Ejemplo de fichero de políticas de seguridad:

```
grant
{
    permission java.net.SocketPermission
               "*:1024-65535", "connect,accept,resolve";

    permission java.net.SocketPermission
               "*:80", "connect";
} ;
```

Gestor de seguridad RMI

- Ejemplo de despliegue de políticas de seguridad:

```
Java –Djava.rmi.server.codebase=<http://...URL>
     -Djava.rmi.server.hostname=<nombre servidor>
     -Djava.security.policy=<ruta completa al fichero
                                de política de seguridad>
```

Resguardos y gestor de seguridad RMI

1. Crear directorio
2. Definir interfaz remoto.
3. Realizar su implementación.
4. Generar ficheros resguardo y esqueleto.
5. Diseñar programa servidor.
6. Copiar fichero resguardo al servidor HTTP.
7. Activar registro RMI.
8. Construir fichero de políticas de seguridad.
9. Activar servidor.

Resguardos y gestor de seguridad RMI