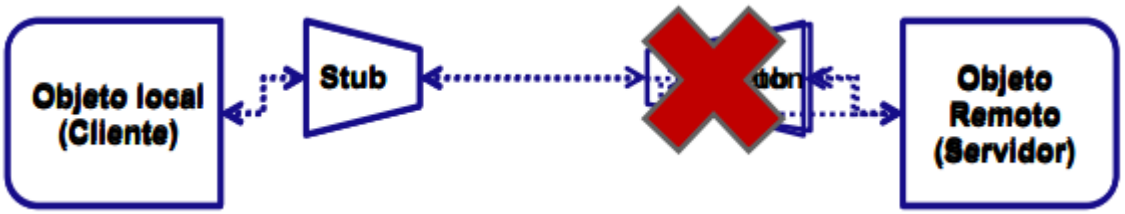
Práctica 3: Programación con RMI (API de Java)

En la p2 llamada a procedimiento remoto. Para una correcta adecuación a la semántica de invocación de objetos, los sistemas de objetos requieren remote method invocation o RMI.

En lugar de procedimientos, vamos a empezar a hablar de objetos remotos (ahora llamaremos a Métodos en lugar de Procedimientos).

Esquema de RMI (similar al esquema de SUN RCP)

Objeto Local <——-> Stub (Objeto Intermedio) ←——> Objeto Real



• Integra de forma natural el modelo de objetos distribuidos en el lenguaje de programación Java, preservando la mayor parte de la semántica de objetos en este lenguaje -> el uso de objetos remotos es lo más similar posible a un objeto local

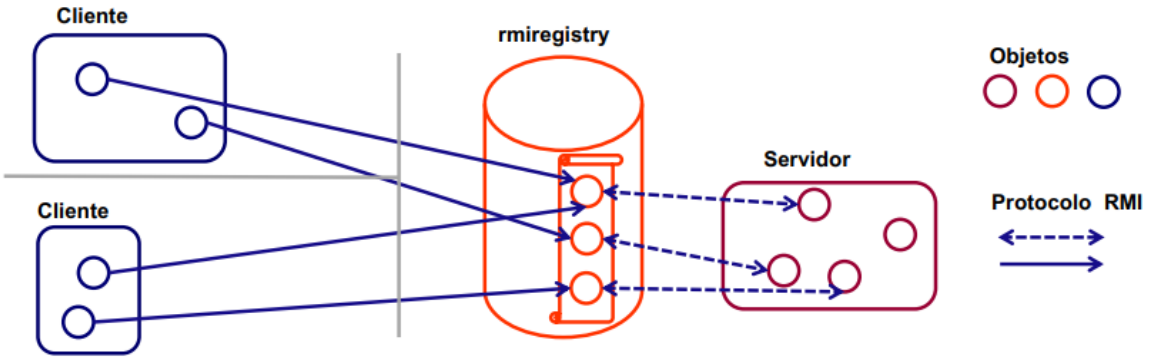
Un objeto remoto es aquel cuyos métodos se pueden invocar desde una máquina virtual de Java remota distinta. Ese objeto será definido mediante una o varias interfaces remotas. Esas interfaces se implementarán mediante clases.

- En el modelo de objetos distribuidos de la plataforma Java, un **objeto remoto** es aquel cuyos métodos pueden invocarse desde otra JVM, posiblemente en otra computadora

- Un objeto de este tipo **se describe** mediante una o más **interfaces remotas**, que son **interfaces** Java donde se declaran los métodos del objeto remoto. Las **clases** implementan los métodos declarados en las **interfaces** y eventualmente, métodos adicionales
- ***Remote method invocation*** (RMI) es la acción de invocar un método en una **interfaz remota** de un objeto remoto. La invocación de un método en una **interfaz remota** sigue la misma sintaxis que en un objeto local

Modelo de Objetos Distribuidos

- Normalmente, las aplicaciones RMI constan de dos aplicaciones separadas: **servidor** y **cliente**
- Servidor:
  1. Crea objetos remotos
  2. Hace accesibles las referencias a dichos objetos remotos
  3. Aguarda a la invocación de métodos sobre dichos objetos remotos por parte de los clientes
- Cliente:
  1. Obtiene una referencia remota a uno o más objetos remotos en el servidor
  2. Invoca métodos sobre los objetos remotos
- RMI proporciona los mecanismos para que servidor y cliente se comuniquen e intercambien información. Esta aplicación normalmente se denomina **aplicación de objetos distribuidos**



Entidad intermedia: registro rmi -> el servidor exportará/registrará ahí su objeto remoto.
Entonces estará disponible ese objeto por parte de los clientes.
Los clientes se conectan a registro rmi y obtienen la referencia al objeto remoto.

(Los clientes también pueden acceder a objetos remotos mediante funciones que devuelven referencias o mediantes parámetros).

Los clientes de objetos remotos interactúan con interfaces remotas (con el stub), nunca con las clases que implementan dichas interfaces (sus instancias).

Los argumentos y resultados de una invocación remota se pasan por copia (no por referencia). El stub transferirá la información a través de la red (realiza una compactación y una copia de la información).

Los objetos remotos se pasan por “referencia” -> identificador del objeto en cuestión, identificador del método de objeto y paso por copia parámetros.

Creación de Aplicaciones RMI en Java

- La utilización de Java RMI para aplicaciones distribuidas sigue las siguientes etapas:
  1. Diseñar e implementar los componentes de la aplicación distribuida**
  2. Compilar el código fuente
  3. Hacer las clases accesibles a través de la red
  4. Iniciar la aplicación

1. Diseñar e implementar los componentes de la aplicación distribuida: Esta etapa incluye:
   a) Definir las interfaces remotas: Los métodos que pueden ser invocados remotamente por los clientes (sólo los de las interfaces, no los de las clases que los implementan)
   b) Implementar los objetos remotos: Deben implementar una o más interfaces remotas (también pueden implementar otras que sólo sean locales, pero no estarán disponibles para los clientes)
   c) Implementar los clientes

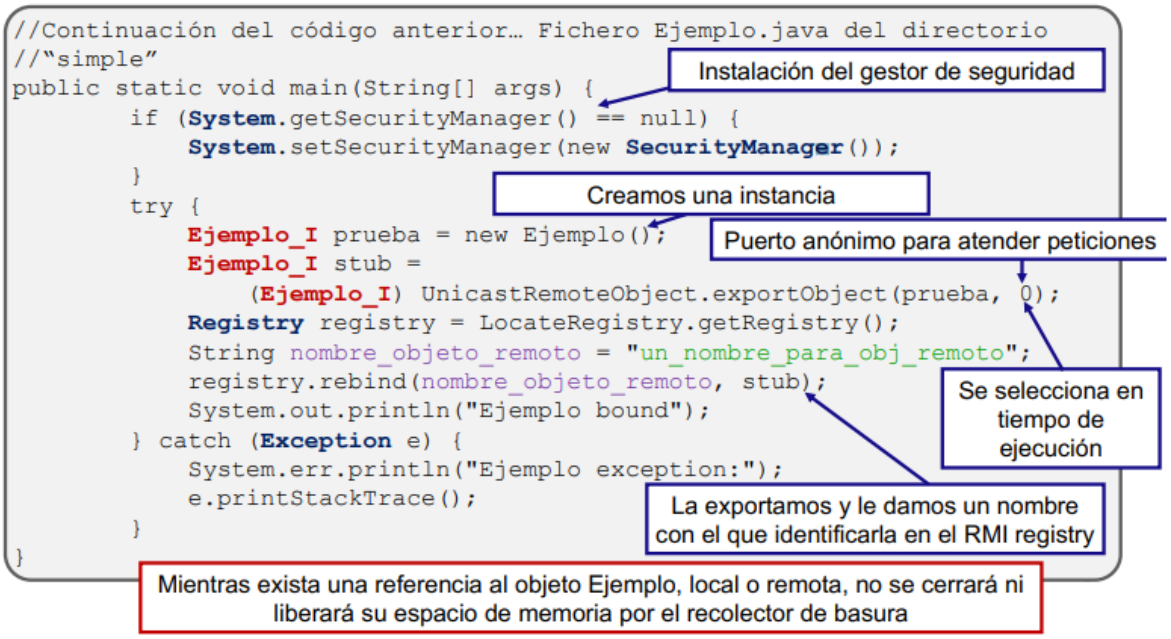
1. Definir interfaz:
- hereda de Remote
- cada uno de sus métodos devuelve una excepción

2. Implementar el objeto remoto mediante una clase que extienda la interfaz definida (o interfaces)

3. Hacer visible el objeto remoto para que los clientes puedan acceder a ellos. Pasos a seguir:
   - Crear e instalar un gestor de seguridad (“security manager”)
   - Crear y exportar uno o más objetos remotos
   - Registrar al menos un objeto remoto en el registro RMI

Este proceso puede realizarse dentro de algún método de inicialización de la propia clase o llevarse a cabo en cualquier otro método de otra clase. En el ejemplo, será la propia clase que implemente el objeto remoto quien lo realice.

- Implementación del objeto remoto



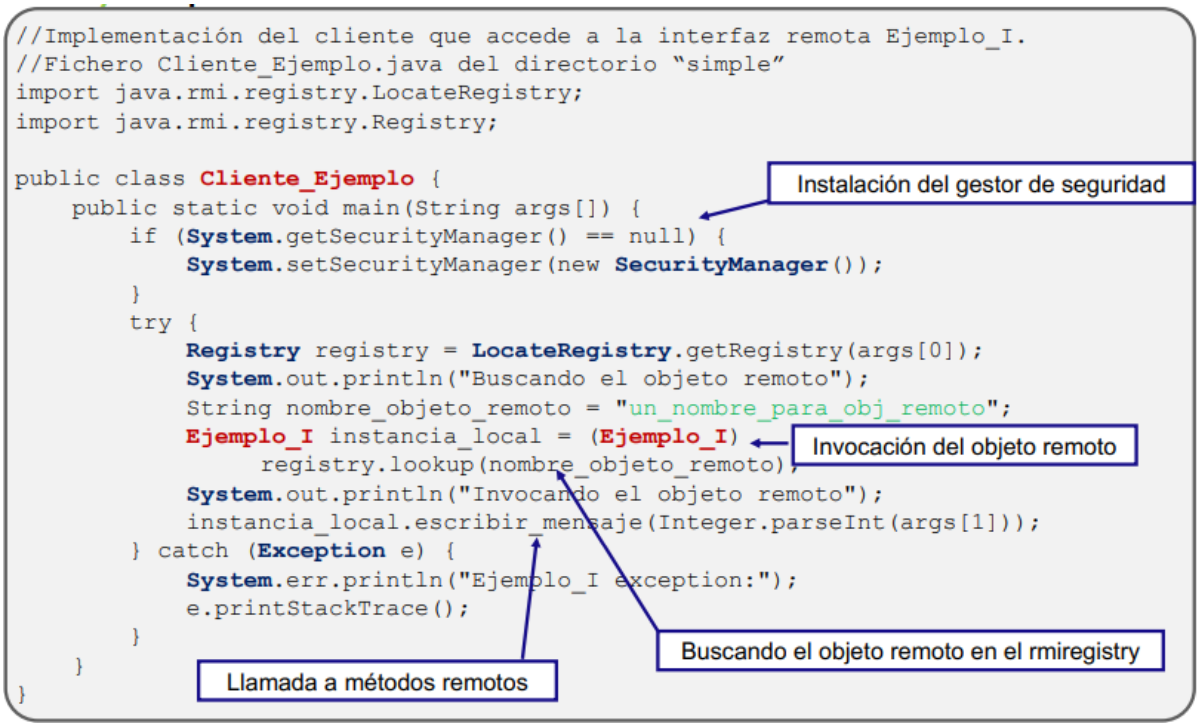
(Explicacion de chatGPT)
El código que has proporcionado es la implementación de un servidor de objetos remotos en Java que utiliza el API RMI (Remote Method Invocation). El servidor crea una instancia de la clase Ejemplo, la exporta como objeto remoto y la registra en el registro RMI para que los clientes puedan acceder a ella.

A continuación, se explica el significado de cada línea de código:

1. `public static void main(String[] args)` : Este es el método principal de la clase Ejemplo.java. Aquí es donde comienza la ejecución del programa.
2. `if (System.getSecurityManager() == null)` : Esta línea comprueba si hay un administrador de seguridad instalado. Si no hay ningún administrador de seguridad, se instala uno nuevo.
3. `System.setSecurityManager(new SecurityManager())` : Si no hay ningún administrador de seguridad, se instala uno nuevo. En este caso, se está utilizando el administrador de seguridad predeterminado de Java.
4. `Ejemplo_I prueba = new Ejemplo()` : Se crea una nueva instancia de la clase Ejemplo.
5. `Ejemplo_I stub = (Ejemplo_I) UnicastRemoteObject.exportObject(prueba, 0)` : La instancia de la clase Ejemplo se exporta como objeto remoto y se convierte en un stub (un objeto que actúa como intermediario entre el cliente y el objeto remoto).
6. `Registry registry = LocateRegistry.getRegistry()` : Se obtiene una referencia al registro RMI.
7. `String nombre_objeto_remoto = "un_nombre_para_obj_remoto"` : Se establece un nombre para el objeto remoto.

8. `registry.rebind(nombre_objeto_remoto, stub)` : El stub se registra en el registro RMI con el nombre especificado.
9. `System.out.println("Ejemplo bound")` : Se imprime un mensaje indicando que el objeto remoto ha sido registrado con éxito.
10. `catch (Exception e) { System.err.println("Ejemplo exception:"); e.printStackTrace(); }` : Si se produce alguna excepción durante la ejecución del programa, se captura y se imprime un mensaje de error. En este caso, se captura cualquier excepción genérica que pueda ocurrir.

4. Implementación del cliente



ver explicación apuntes

(explicación chat gpt):

El código que has proporcionado es la implementación de un cliente que accede a un objeto remoto en Java utilizando el API RMI (Remote Method Invocation).

A continuación, se explica el significado de cada línea de código:

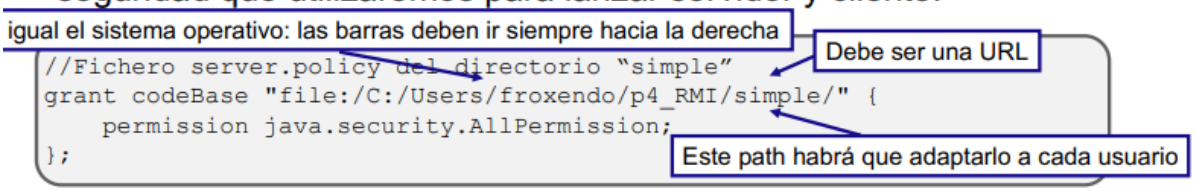
1. `public static void main(String args[])` : Este es el método principal de la clase Cliente_Ejemplo.java. Aquí es donde comienza la ejecución del programa.
2. `if (System.getSecurityManager() == null)` : Esta línea comprueba si hay un administrador de seguridad instalado. Si no hay ningún administrador de seguridad, se instala uno nuevo.

3. `System.setSecurityManager(new SecurityManager())` : Si no hay ningún administrador de seguridad, se instala uno nuevo. En este caso, se está utilizando el administrador de seguridad predeterminado de Java.
4. `Registry registry = LocateRegistry.getRegistry(args[0])` : Se obtiene una referencia al registro RMI en el host especificado por el primer argumento de la línea de comandos.
5. `System.out.println("Buscando el objeto remoto")` : Se imprime un mensaje indicando que se está buscando el objeto remoto.
6. `String nombre_objeto_remoto = "un_nombre_para_obj_remoto"` : Se establece el nombre del objeto remoto que se va a buscar.
7. `Ejemplo_I instancia_local = (Ejemplo_I) registry.lookup(nombre_objeto_remoto)` : Se busca el objeto remoto en el registro RMI y se devuelve una referencia a él. La referencia se convierte en una instancia local de la interfaz Ejemplo_I.
8. `System.out.println("Invocando el objeto remoto")` : Se imprime un mensaje indicando que se va a invocar el objeto remoto.
9. `instancia_local.escribir_mensaje(Integer.parseInt(args[1]))` : Se invoca el método escribir_mensaje del objeto remoto y se le pasa el segundo argumento de la línea de comandos como parámetro. El método puede hacer cualquier cosa que esté definida en la interfaz remota Ejemplo_I.
10. `catch (Exception e) { System.err.println("Ejemplo_I exception:"); e.printStackTrace(); }` : Si se produce alguna excepción durante la ejecución del programa, se captura y se imprime un mensaje de error. En este caso, se captura cualquier excepción genérica que pueda ocurrir.

5. Lanzar la aplicación para cliente y servidor; en el ejemplo se puede usar simplemente javac. En aplicaciones más complejas tendremos que tener en cuenta la estructura de paquetes, etc.

Tenemos que hacer accesibles las interfaces -> eso se hace mediante el gestor de seguridad (mediante ficheros de política de seguridad).

- Como ya hemos visto, servidor y cliente (en este caso, clases `Ejemplo` y `Cliente_Ejemplo`, respectivamente) se ejecutan instalando un gestor de seguridad (*security manager*).
- Por esta razón, para ejecutar cualquiera de ellos es necesario especificar un *fichero de políticas de seguridad* tal que conceda al código los permisos de seguridad que necesita para ejecutarse.
- A continuación mostramos un ejemplo de fichero de políticas de seguridad que utilizaremos para lanzar servidor y cliente.



- En este ejemplo, se conceden todos los permisos a las clases en el `classpath` del programa local, de manera que no se conceden permisos para el código descargado desde otras ubicaciones.
- El fichero se denomina "server.policy" y lo utilizaremos al lanzar, tanto el cliente, como el servidor.

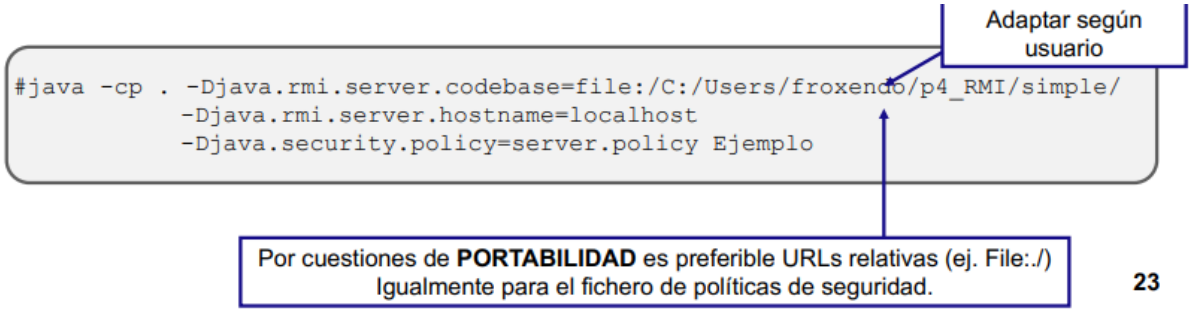
22

Escribir rutas relativas; no como en el ejemplo!!!
Los archivos de política de seguridad dan permisos a rutas (permiten todo el acceso en esa ruta, nada fuera de ella).

Antes de lanzar la aplicación tenemos que lanzar el registro rmi para que nuestra aplicación pueda hacer uso de él.

> rmiregistry & // Lanza/Activa el registro RMI en el puerto por defecto (1099)
> rmiregistry 2011 & // Lanza/Activa el registro RMI en el puerto 2011

Para lanzar el servidor:



23

- La orden anterior especifica distintas propiedades para el entorno Java:
  - `java.rmi.server.codebase` especifica la ubicación desde la que poder descargar definiciones de clases desde el servidor.
  - `java.rmi.server.hostname` especifica el nombre de la máquina donde colocar los stubs para los objetos remotos.
  - `java.security.policy` especifica el fichero de políticas de seguridad que se pretenden seguir o conceder.

Para lanzar en el cliente (más simple):

- Ejemplos de órdenes para lanzar clientes

```
#java -cp . -Djava.security.policy=server.policy Cliente_Ejemplo localhost 0
```

Cliente 0

```
#java -cp . -Djava.rmi.server.codebase=file:///C:/Users/froxendo/p4_RMI/
          -Djava.security.policy=server.policy Cliente_Ejemplo localhost 1
```

Cliente 1 especificando la propiedad `java.rmi.server.codebase`

Si el cliente implementara alguna clase desconocida por el servidor, entonces al lanzarlo deberíamos añadir una propiedad -codebase con la ruta de las clases desconocidas por el servidor.

Primera parte de la práctica: ejecutar los ejemplos del guión.
Ver script del guión -> (a tener en cuenta)

- Lanzar el servidor en segundo plano
- Si ocurre algún error, y queremos lanzar de nuevo el script, tenemos que tener en cuenta que el registro rmi sigue en proceso en segundo plano, por lo que deberemos matarlo o no volverlo a lanzar!!

(*) Otro ejemplo -> multihebrado (el cliente crea un conjunto de hebras, y estas llaman de forma concurrente al método remoto).

Responder en la memoria esto las siguientes preguntas

Ejercicio1: Ejecución de Servidor y Cliente Multihebrado

- ¿Qué ocurre con las hebras cuyo nombre acaba en 0?¿Qué hacen las demás hebras?¿Se entrelazan los mensajes?

- RMI es **multihebrado**, lo que habilita la gestión concurrente de las peticiones de los clientes.
- Esto también implica que las implementaciones de los objetos remotos han de ser seguras (*thread-safe*).

- Prueba a introducir el modificador `synchronized` en el método de la implementación remota: `public synchronized void escribir_mensaje (String mensaje) {…` y trata de entender las diferencias en la ejecución de los programas.

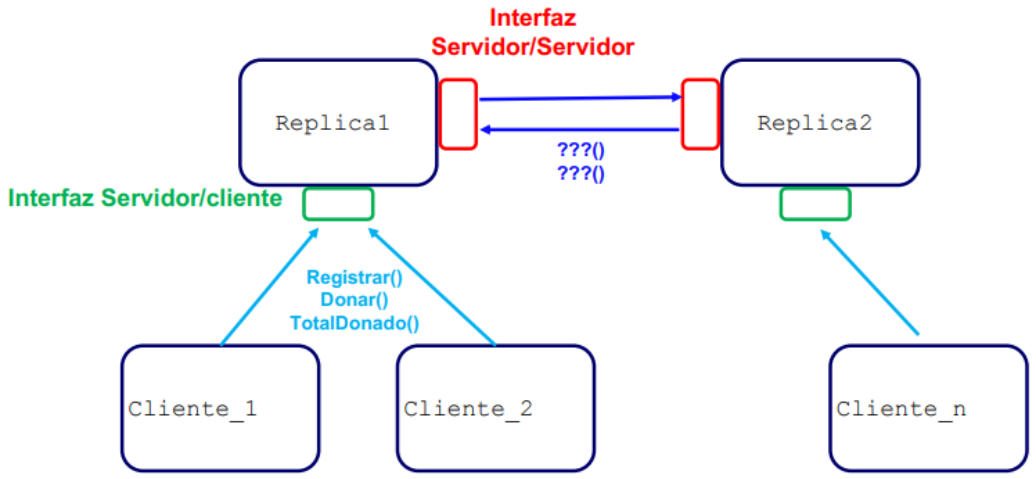
32

- Cuando terminan en 0 la hebra manda a dormir al servidor (?)

Tercer ejemplo: Contador (VER IMPLEMENTACIÓN, IMPORTANTE cómo se crean los objetos, cómo se hace uso de registro rmi…)

Tras ejecutar los ejercicios y completar la memoria, realizar el ejercicio propuesto: Servidor replicado de donaciones

Ejercicio propuesto 1:
Servidor replicado de donaciones



El registro de clientes ocurrirá en la réplica del servidor que tenga el menor número de clientes registrados. ¿Cómo podemos hacer funcionar eso? Con dos tipos de interfaces->

- Interfaz cliente servidor -> operaciones cliente servidor. Las donaciones quedarán registradas en la réplica donde se registró el cliente.
- Interfaz servidor servidor -> esta interfaz determinará qué operación hará un servidor u otro (para ver qué servidor tiene el menor número de clientes).
- Los servidores también ofrecerán una operación de consulta del total donado en un momento dado. Dicha operación sólo podrá llevarse a cabo si el cliente previamente se ha registrado y ha realizado al menos un depósito. Cuando un cliente consulte la cantidad total donada hasta el momento, sólo hará la petición a la réplica donde se encuentra registrado, y ésta será la encargada, realizando la operación oportuna con la otra réplica, de devolver el total donado hasta el momento.
- Total donado es la suma del subtotal de ambos servidores replicados (otro método de conexión).
- Hacer un menú de opciones DINÄMICO

En la memoria, explicar:

- Cómo se ha realizado la aplicación
- Cómo se puede compilar, lanzar la aplicación (paso a paso para el profesor)
- (Un ejemplo puede ser lanzar el script)

En el guión muestra cómo seguir la implementación del contador como estructura de proyecto en NetBeans.

Parte opcional: mayor número de servidores replicados, más funcionalidades, utilizar algoritmos de exclusión mutua … etc (ver normas de entrega de p3).