**PLANIFICACION DE TAREA Y TIEMPOS**

-> Hacer el diagrama gant

- Aprendizaje Investigacion

- Extraer las caracteristicas principales que se van a utilizar para los ejercicios

- Hacer los ejercicios

-Documentar los ejercicios

+ En palabras suyas hay que dedicarle un tercio del tiempo a documentar. Y lo demas ha hacer ejercicios

+ PONER planificacion por semanas y cuantas horas a cada tarea

- 20-21 : Planteamiento del problema y poner de acuerdo las cosas

- 24-28

- 1-5

**ENTRADA DE JAVA RMI (usar diapositivas de clase para completar)**

- Siglas

- Entorno en el que se creo

- Funcionamiento

- Estado hoy en dia ( sirve para hacer conexiones y acciones concretas de forma eficiente,.

Muy usado en entornos java, ejemplo (esta detras del funcionamiento de los @Bean en java y SpringBoot)):

**EJERCICIOS DE CONCEPTO**

*SETUP BASICO, STUB Y SKELETON*

Ejemplo en eclipse

* - Ejercicio base de funcionamiento basico [(tutorial de oracle)](https://docs.oracle.com/javase/tutorial/rmi/overview.html)  (como los de clase) (en local) (se puede usar el siguiente para hacer la explicacion, y el ejercicio en local -> [ver](https://www.javatpoint.com/RMI) ) [Una version con registry](https://docs.oracle.com/javase/8/docs/technotes/guides/rmi/hello/hello-world.html) ejecucion en eclipse.

Ejemplo de consola

* Usar Naming.rebind para unir los 2 programas a traves un nombre de red ( en diferente dispositivos) -> [ver](https://www.javatpoint.com/RMI)
* Compilacion de los programas del tutorial de oracle probar ha hacer ejecucion en diferentes ordenadores. Usar java 1.8 o inferior, porque sino no funcionara.

*NAMING REGISTRY Y SERIALIZACION EN RMI*

- Basico de mandar int , char ..... (las anteriores entrarian aqui)

- Aplicar Serializable, y mandar objetos enteros. (usar ejemplo de aita)

- Hacer un server al que accedan varios clientes a la vez (aplicar concurrencia).

*SEGURIDAD*

-> Una vez que consiga hacerlos funcionar en remoto o en diferente ordenador

* -Implementar la seguridad. Aplicacion de la clase RMISecurityManager, en elejemplo anterior

*CARGA DINAMICA*

* - Carga dinamica, usando la clase RMIClassLoader para implementar el Stub manualmente..
* Mirar Compute engine ejemplo de moodle ejemplo clase, modificarlo para aplicarlo en tema real. Es un server mas potente que los clientes, para poder hacer tareas mas importates.
  + La implementacion de la interfaz esta en el cliente y se ejecutaba en el server, se mandaba/cargaba

***EJERCICIO GRANDE***

* Plantearlo desde el punto de vista de la carga dinamica. Tiene que sea ejemplos que aprobechen y justifiquen el uso de Rmi.

BIBLIOGRAFIA

# PLANIFICACIÓN

# ENTRADA

RMI son las siglas de Remote Method Invocation (Invocación Remota de Métodos). Es un mecanismo que permite a un objeto que reside en un sistema (JVM) acceder/invocar a un objeto que se ejecuta en otra JVM. RMI se utiliza para construir aplicaciones distribuidas; proporciona comunicación remota entre programas Java. Se proporciona en el paquete **java.rmi.**

## Objetivos de RMI

Los objetivos de RMI son los siguientes

* Minimizar la complejidad de la aplicación.
* Preservar la seguridad de tipos.
* Recolección de basura distribuida.
* Minimizar la diferencia entre trabajar con objetos locales y remotos.

## Funcionamiento

RMI (Java Remote Method Invocation) es un mecanismo ofrecido por Java para invocar un método de manera remota. Forma parte del entorno estándar de ejecución de Java y proporciona un mecanismo simple para la comunicación de servidores en aplicaciones distribuidas basadas exclusivamente en Java. Si se requiere comunicación entre otras tecnologías debe utilizarse CORBA o SOAP en lugar de RMI.

RMI se caracteriza por la facilidad de su uso en la programación por estar específicamente diseñado para Java; proporciona paso de objetos por referencia y paso de tipos arbitrarios en los parametros.

Toda aplicación RMI normalmente se descompone en 2 partes:

* -Un servidor, que crea algunos objetos remotos, crea referencias para hacerlos accesibles, y espera a que el cliente los invoque.
* -Un cliente, que obtiene una referencia a objetos remotos en el servidor, y los invoca.

A través de RMI, un programa Java puede exportar un objeto, con lo que dicho objeto estará accesible a través de la red y el programa permanece a la espera de peticiones en un puerto TCP. A partir de ese momento, un cliente puede conectarse e invocar los métodos proporcionados por el objeto.

La invocación se compone de los siguientes pasos:



* -Encapsulado (marshalling) de los parámetros (utilizando la funcionalidad de serialización de Java).
* - El cliente invoca el metodo del metodo del servidor para ejecutarlo hay. Para eso le pasa los parametros serializados. Y se queda esperando su respuesta.
* -El servidor recibe los parametros y ejecuta el metodo concreto que le ha dicho el cliente. Al terminar la ejecución, el servidor serializa el valor de retorno (si lo hay) y lo envía al cliente.
* -El código cliente recibe la respuesta y continúa como si la invocación hubiera sido local.

# EJERCICIOS DE CONCEPTO

En este apartado se iran trabajando los diferentes conceptos que componen Rmi de una forma ordenada para construir una comprension practica del tema.

## SETUP BASICO, STUB Y SKELETON

El RMI (Remote Method Invocation) es una API que proporciona un mecanismo para crear aplicaciones distribuidas en java. Esta permite a un objeto invocar métodos en un objeto que se ejecuta en otra JVM, a esto se le denomina comunicacion remota.

### Interfaces de objetos remotos

Dado que RMI es un esquema de objetos distribuidos sólo en Java, todas las interfaces de objetos están escritas en Java. Los stubs de cliente y los esqueletos de servidor se generan a partir de esta interfaz, pero utilizando un proceso ligeramente diferente al de CORBA. En primer lugar, la interfaz para el objeto remoto tiene que ser escrita como una extensión de la interfaz java.rmi.Remote. La interfaz Remote no introduce ningún método en la interfaz del objeto; sólo sirve para marcar objetos remotos para el sistema RMI. Además, todos los métodos de la interfaz deben ser declarados como lanzadores de la java.rmi.RemoteException . La RemoteException es la clase base para muchas de las excepciones que RMI define para las operaciones remotas, y los ingenieros de RMI decidieron exponer el modelo de excepción en las interfaces de todos los objetos remotos RMI. Este es uno de los inconvenientes de RMI: requiere alterar una interfaz existente para aplicarla a un entorno distribuido.

Esta comunicación remota entre las aplicaciones se hace utilizando dos objetos, el stub en la parte del cliente y el skeleton en la parte del servidor, haciendo posible la comunicación con el objeto remoto.

### Stub

El stub es un objeto que actúa como puerta de enlace para el lado del cliente. Todas las peticiones salientes se enrutan a través de él. Reside en el lado del cliente y representa el objeto remoto. Cuando el llamador invoca un método en el objeto stub, éste realiza las siguientes tareas:

* Inicia una conexión con la máquina virtual remota (JVM)
* Escribe y transmite (marshals) los parámetros a la Máquina Virtual remota (JVM),
* Espera el resultado
* Lee (unmarshals) el valor de retorno o excepción.
* Por último, devuelve el valor a la persona que llama.

### Esqueleton

El esqueleto es un objeto que actúa como puerta de enlace para el objeto del lado del servidor. Todas las peticiones entrantes se enrutan a través de él. Cuando el esqueleto recibe la petición entrante, realiza las siguientes tareas:

* Lee los parámetros del método remoto
* Invoca el método en el objeto remoto real, y
* Escribe y transmite (marshals) el resultado a la persona que llama.
* En el SDK de Java 2, se introdujo un protocolo stub que elimina la necesidad de esqueletos.



### Estado actual de la arquitectura Skeleton Stub



La arquitectura java RMI, se actualizo con la version de [java 1.5](https://nick-lab.gs.washington.edu/java/jdk1.5b/guide/rmi/relnotes.html) , la cual introdujo los Stubs dinamicos eliminando la necesidad de utilizar Skeleton. Quitando totalmente la necesidad de compilar con la herramienta rmic.

### EJERCICIOS

#### 001- Hello world con naming (modulo antiguo)

**Enunciado**

Ejecuta el programa 001- Hello world con naming, y entiende su funcionamiento.

**Explicacion**

Como el codigo se intenta ejecutar sin haber realizado la compilacion del codigo ni el levantamiento del servicio Rmi, eclipse no sera capaz de ejecutar el codigo, y fallara, ya que no sera capaz de establecer la conexion entre el cliente y el servidor.

**Output**



#### 002 - Compilacion y ejecucion (modulo antiguo)

**Enunciado**

Haz funcionar el ejemplo anterior haciendo todo el proceso de compilacion. Las instrucciones son las siguientes:

1. Habre CMD ,colocate en la carpeta src del proyecto
2. Ejecuta el comando javac <nombre del archivo java> , y compila todos los archivo .java del src
3. Crea los objetos Stub y skeleton con rmic AdderRemote
4. Ejecuta el comando rmiregistry 5000 , para ejecutar el servidor rmi en el puerto 5000
5. Sin cerrar el terminal anterior abre un nuevo terminal, en el src y ejecuta java MyServer  , para empezar el servidor.
6. En otra terminal nueva con java MyClient  empieza el cliente.

**Explicacion**

La funcion de los comandos es la siguiente:

* Javac <archivo .java> -> Compila el codigo fuente.
* rmic AdderRemote   -> Crea los objetos stub y skeleton utilizando la herramienta rmic .
* rmiregistry 5000  -> Inicia el servicio rmi en el puerto indicado (por defecto es 5000),
* java MyServer  -> Inicia la ejecucion del servidor
* java MyClient  -> Inicia la ejecucion del cliente

Como curiosidad, si se levanta este servicio rmi en la linea de comandos, la ejecucion del servidor y el cliente se pueden hacer por ejecucion normal en eclipse. Debido a que el servicio permite la interconectividad entre servidor y cliente a traves del puerto 5000.

**Output**

Terminal 1 el de la compilacion y levantamiento del servicio



-> El metodo rmiregistry que se estas utilizando para levantar el servicio esta deprecated y el sistema lo notifica.



Terminal 2: el server



Terminal 3: el cliente



-> Ejecucion alterna en eclipse si el servicio rmi esta levantado



## NAMING REGISTRY

El servicio de nombres RMI (RMI registry) es un servidor que permite a una aplicación buscar objetos que están siendo exportados para su uso mediante llamadas a métodos remotos.

### Funcionamiento

Una vez que el objeto ha sido localizado, ya se puede utilizar utilizando la misma sintaxis que una llamada a un método local. Para encontrar los objetos, RMI utiliza un servicio que mantiene una tabla de direcciones de objetos remotos que están siendo exportados por sus aplicaciones de la siguiente forma.



A todos los objetos se les asigna nombres únicos que se utilizan para identificarlos. Algunos métodos pueden llamarse desde la interfaz rmi.registry.Registry, o desde la clase rmi.Naming, que permite añadir, eliminar y acceder a objetos remotos en la tabla de registro de objetos. El servidor del servicio de nombres registra los objetos mediante llamadas a bind() o rebind() sobre una instancia de un registro del objeto que está siendo exportado. De forma alternativa, rebind() reemplaza un objeto antiguo con un nombre dado, con un nuevo objeto.

Despues Para invocar un objeto remoto, el cliente necesita una referencia de ese objeto. En ese momento, el cliente obtiene el objeto del registro utilizando su nombre bind (mediante el método lookup()).



Hay dos formas de iniciar el servicio de nombres: una es usando la aplicación que proporciona java (rmiregistry), y otra es escribir nuestro propio servicio de nombres usando las clases e interfaces de java.rmi.\*

Normalmente la aplicación de servicio de nombres se inicia como una aplicación en background. Por defecto, se ejecutará sobre el puerto 1099, pero se puede seleccionar otro puerto cuando se lanza la aplicación.

### EJERCICIOS

#### 003-Helloworld con Registry

**Enunciado**

Basandote en el ejercicio 001- Hello world con naming , actualizalo sustituyendo Naming.lookUp, por:

Cliente:

Registry registry = LocateRegistry.*getRegistry*(1099);

Hello stub = (Hello) registry.lookup("Hello");

Server:

registry.bind("Hello", obj);

Nota: Recuerda que se tiene que poner la misma clave para que se comuniquen.

**Explicacion**

Con el codigo mostrado, se pretende actualizar el primer ejercicio realizado, poniendo un servidor de nombres que quita la necesidad de compilacion y de utilizar la linea de comandos, ademas de permitir idenfificar cada servicio facilmente con una clave unica, dando la opcion de poder utilizar varios servidores en caso de que sea necesario.

**Output**



#### 004-Calculadora

**Enunciado**

Se quiere desarrollar la capacidad de calculo del programa anterior, haciendo que sea posible realizar diferentes calculos en funcion de lo necesario. Modifica el programa 003-HelloWorld con Registry, para que pueda realizar las siguientes acciones:

1. Sumar
2. Restar
3. Multiplicar
4. Dividir
5. Terminar el programa.

**Explicación**

Es una extensión del ejercicio anterior que implementa la misma estructura. Esta pensada para ver que es posible llamar múltiples veces a un objeto registrado, para hacer las diferentes acciones mencionadas.

**Output**



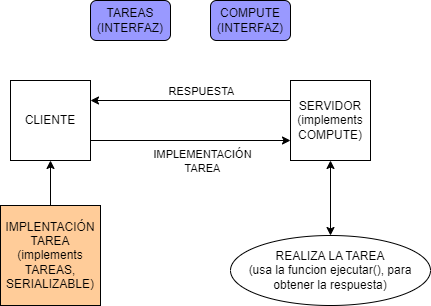
## SERIALIZACIÓN EN RMI

RMI facilita el olvidarse de los detalles de la transmisión de datos y centrarse en el diseño de la lógica de la aplicación. Puesto que permite acceder a un objeto remoto como si de un objeto local se tratase.

Esta facilidad de RMI es dada por la serialización de objetos que se hace usando la interfaz Serializable que el paquete java.io incluye. Que pueden convertir un objeto en un flujo de bytes y volver a ensamblar los bytes en una copia idéntica del objeto original.

Utilizando estas clases, un objeto en un proceso puede ser serializado y transmitido a través de una conexión de red a otro proceso en un host remoto. El objeto (o al menos una copia del mismo) puede entonces ser reensamblado en el host remoto para su posterior uso.

### FUNCIONAMIENTO



Con esta configuracion, solo es posible realizar una tarea concreta, o una serie de tareas pero con un resultado final comun, no permite ejecutar varias tareas implementadas de forma independiente cada una con su propio resultado.

### EJERCICIOS

#### 006- Saludador serializado

**Enunciado**

Crea un programa con el objeto Saludador, para que el cliente se lo mande al servidor y reciba la respuesta.

**Explicación**

String al ser una clase serializable, es posible transferirla directamente a traves de la red.

**Output**

#### 

#### 007- Comprobador de Urls

**Enunciado**

Crea otro programa, que cuando le pase una Url, si es valida, diga que es valida y enseñe el html de la pagina web a la que hace referencia en pantalla.

Y si no es valida, que diga que no es valida.

**Explicación**

Para este ejercicio se ha usado el objeto URL de la biblioteca de java. Este objeto al inicializarse, en caso de que el url usado para la inicializacion no sea valido saltara con un UrlMalformedException, que es lo que se usara para decir si es valido o no.

Para ver el Html de la pagina web, leemos el contenido del objeto con stream.

Mas informacion del objeto Url, en el [link.](https://lineadecodigo.com/java/leer-una-url-con-java/)

**Output**





#### 008- Meter notas del curso y calcula la media

**Enunciado**

Realiza un programa en JAVA en el que le pidas al usuario las notas de las 5 asignaturas del semestre y las guarde en un fichero. Posteriormente leerá el fichero y despues de mandar los objetos asignaturas al servidor te calculará la nota media del curso.

**Explicación**

En este caso es un ejemplo donde se mandan y se reciben objetos. Pero al ser estos objetos ( el List como el Integer) de java, ya son elementos Serializables y no es necesario implementar la intefaz para mandar el objeto de vuelta en el resultado.

**Output**



#### 009- Transmisión de archivos entre cliente y server.

**Enunciado**

Crea un programa que lea un archivo en el cliente y despues tranfiera el archivo al servidor. Tiene que confirmar si la transferencia se ha realizado correctamente o no.

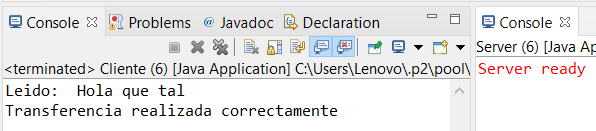
**Explicación**

En este ejercio, lo unico que hacemos es tomar el contenido del archivo de texto, pasarlo a un objeto serializable y pasarlo al servidor para que este lo guarde nuevamente en otro archivo en destino.

- Se le pasa unicamente el contenido debido a que el objeto File de java pese a ser serializable, no es conveniente hacer uso de el durante la transferencia, debido a que File solo es una representacion del pathName del archivo en cuestion en la maquina de origen, no de su contenido. Y en un servidor remoto no es posible manipular el archivo del origen desde el servidor, debido a que al cambiar de maquina se pierde el acceso a el fichero original. Por eso es mejor pasar su contenido.

- En lugar del objeto File se ha usado un String para guardar el contenido del archivo de origen. Y al ser serializable se puede transferir sin problemas por la red.

**Output**



## SEGURIDAD EN RMI

El protocolo RMI se penso para conectar dispositivos conocidos en un entorno controlado sin salidas al exterior.

Por lo cual realmente es un protocolo sin ninguna seguridad por si misma.

Aunque ha habido intentos de Hardening de la comunicacion usando la clase de seguridad , que implementa Ssl.

---------------------------------------------------------------------------------------------------

Uno de los problemas más comunes que uno encuentra con RMI es un fallo debido a restricciones de seguridad de java estandar. Esto es en parte debido a que pese un programa Java tiene la posibilidad de especificar un gestor de seguridad que determine su política de seguridad. Un programa no tendrá ningún gestor de seguridad a menos que se especifique uno.

#### Security Manager

En java la política de seguridad se establece construyendo un objeto SecurityManager y llamando al método setSecurityManager de la clase System.

[www.geeksforgeeks.org/java-rmi-rmisecuritymanager-class-in-java/](http://www.geeksforgeeks.org/java-rmi-rmisecuritymanager-class-in-java/)

En cambio para java Rmi se utiliza la clase RMISecurityManager que aplica la política de seguridad a las clases que se cargan como stubs para objetos remotos, sobrescribiendo todos los métodos de comprobación de acceso relevantes del SecurityManager estandar de java. Ya que con la seguridad por defecto de java, los objetos stub sólo pueden realizar operaciones de definición y acceso a clases en local.

#### Security policy

#### Case is only used in JVM on the localhost

#### SSL/TLS en Rmi

#### Descarga de codigo/clases remoto

Ciertas operaciones requieren que haya un gestor de seguridad. Por ejemplo, RMI descargará una clase Serializable desde otra máquina sólo si hay un gestor de seguridad y el gestor de seguridad permite la descarga de la clase desde esa máquina. La clase RMISecurityManager define un ejemplo de gestor de seguridad que normalmente permite tales descargas.

Esta accion se controla mediante el valor de la propiedad java.rmi.server.useCodebaseOnly. Establecer esta propiedad a false habilita la carga remota de código, lo que incrementa el nivel de riesgo de seguridad del sistema.

Aunque su valor por defecto es true.

Esta propiedad sera importante en el apartado siguiente.

## CARGA DINAMICA

--- Nota: En este apartado los ejercicios se tienen que estructurar, como los ejercicios de trocear calculos como Matrices del temario anterior. Calculos distribuidos con una carga balanceada.

La unica diferencia es que despues de dividir el trabajo, en vez de 1 thread por tarea, necesitamos un server con 1 tarea por server. Y se le comunicaria la tarea por Rmi.

para poder hacerse bien con 10 servidores (10 programas server que hacen el calculo) y despues le devuelven el resultado al cliente que lo junta el. O en su defecto un server principal que le devuelve el resultado.

--El ejercicio tochisimo de carga dinamica puede ser hacer lo comentado anteriormente con el juego de la vida, del temario anterior o similar.