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

**BASICO**

**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.

**Como hacer progresion de dificultad**

- 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

EJERCICIOS COMPLETOS

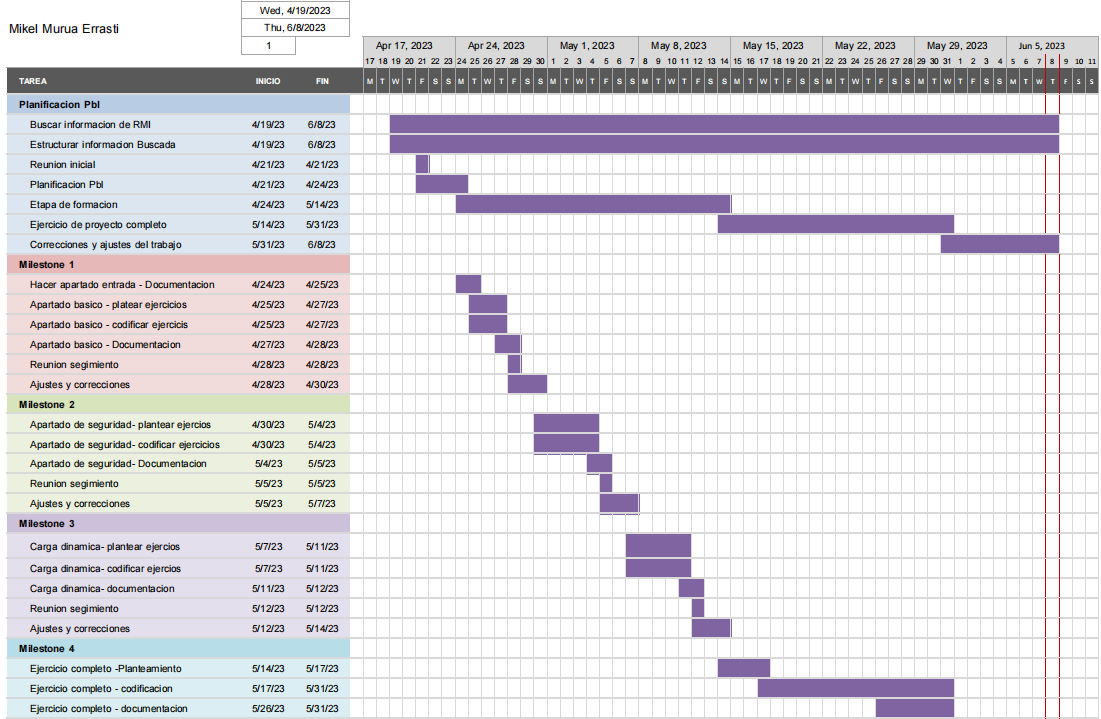
* File transfer

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

# PLANIFICACION



# 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

## 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.

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.



#### Registro RMI

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

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.

### Proyecto 001-Ejercicio Inicial

**Enunciado del programa (Como los enunciados de clase (toma de los ejemplos))**

**La arquitectura del programa**

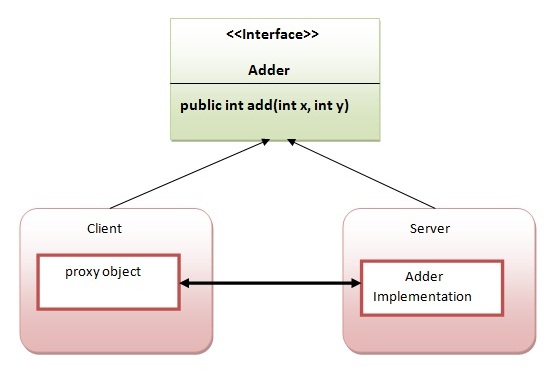
**- Explicar detalles que no aparecen a simple vista en el codigo**

**- Explicar la arquitectura, que parte va en el servidor, y en el cliente , y cual es la interfaz en comun si es que la tienen. Y como se comunican entre ellos (que informacion se mandan)**

NO PONER EL PANTALLAZO DEL WORKSPACE ES PREFERIBLE EL ESQUE MA DE LAS CLASES

Enunciado

#### Explicacion

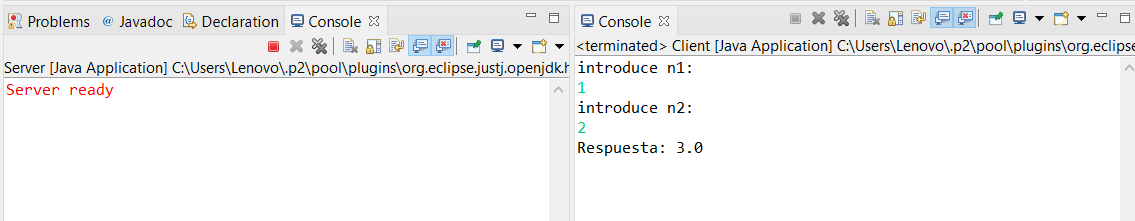
En este ejemplo, la aplicación cliente sólo necesita dos archivos, la interfaz remota y la aplicación cliente. En la aplicación rmi, tanto el cliente como el servidor interactúan con la interfaz remota. La aplicación cliente invoca métodos en el objeto proxy, RMI envía la petición a la JVM remota. El valor de retorno es enviado de vuelta al objeto proxy y luego a la aplicación cliente. 

### Para crear la interfaz remota, extienda la interfaz Remote y declare la RemoteException con todos los métodos de la interfaz remota. Aquí, estamos creando una interfaz remota que extiende la interfaz Remote. Sólo hay un método llamado add() y declara RemoteException.

Para crear la interfaz remota, extienda la interfaz Remote y declare la RemoteException con todo Ahora proporcione la implementación de la interfaz remota. Para proporcionar la implementación de la interfaz Remote, necesitamos extender la clase UnicastRemoteObject,o utilizar el método exportObject() de la clase UnicastRemoteObject. En caso de que extiendas la clase UnicastRemoteObject, debes definir un constructor que declare RemoteException.

En el cliente estamos obteniendo el objeto stub mediante el método lookup() del servidor de Naming e invocando el método sobre este objeto.

En este ejemplo, estamos ejecutando las aplicaciones servidor y cliente, en la misma máquina utilizando el puerto 1099 de LocateRegistry.

Output (explicacion de como es la solucion)

EJERCICIOS

000- Hello world

**Enunciado**

Ejecuta el programa, y entiende su funcionamiento. Una vez hecho esto compilalo.

001 - Compilacion de modulo (antiguo)

**Enunciado**

**Explicacion**

**Output**

002- Hello world con Registry.

**Enunciado**

RMI despues de mucho tiempo se ha actualizado y elementos que antaño habian sido un pilar fundamental de la plataforma han sido sustituidos por otros mas faciles de usar. Actualiza el programa de hello world con la interfaz Registry.

**Explicacion**

003- Calculadora

**Enunciado**

Se quiere explandir la capacidad de calculo del programa anterior, expandiendo los tipos de calculo que es capaz de hacer. Modifica el programa, para que pueda realizar las siguientes acciones:

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

**Explicacion**

**Output**