

TRABAJO ALTERNATIVO PBL: JAVA RMI

S. CONCURRENTES Y DISTRIBUIDOS

INGENIERÍA INFORMÁTICA 3.CURSO



**Mondragon
Unibertsitatea**

**Goi Eskola
Politeknikoa**

AUTOR: Mikel Murua Errasti

TUTOR: Unai Muñoz Valenti

2023-06-08

ÍNDICE

1. INTRODUCCIÓN	4
2. PLANIFICACIÓN	4
3. OBJETIVOS DEL PROYECTO	4
4. TEMAS PRINCIPALES RMI	5
4.1. TEMA 1: ENTRADA/BASES DE RMI	5
4.1.1. ¿QUÉ ES RMI?	5
4.1.2. OBJETIVOS DE RMI	5
4.1.3. FUNCIONAMIENTO	5
4.1.4. SETUP BASICO, STUB Y SKELETON	7
4.1.5. STUB	7
4.1.6. SKELETON	7
4.1.7. ESTADO ACTUAL DE LA ARQUITECTURA SKELETON STUB	8
4.1.8. EJERCICIOS: STUB SKELETON	8
4.2. TEMA 2: NAMING REGISTRY	11
4.2.1. FUNCIONAMIENTO	11
4.2.2. SERIALIZACIÓN EN RMI	12
4.2.3. EJERCICIOS: SERVIDOR DE NOMBRES Y SERIALIZACIÓN	12
4.3. TEMA 3: SEGURIDAD	18
4.3.1. FICHEROS DE POLÍTICAS DE SEGURIDAD	18
4.3.2. GESTOR DE SEGURIDAD	19
4.3.3. RESTRICCIONES DE SEGURIDAD EN APPLETS Y APLICACIONES JAVA	19
4.3.4. DESCARGA DE CÓDIGO/CLASES REMOTO	20
4.3.5. PROCESO DE EJECUTAR EN DIFERENTES máquinaS	20
4.3.6. EJERCICIOS DE SEGURIDAD	21
4.4. TEMA 4: CARGA DINÁMICA	22
4.4.1. FUNCIONAMIENTO	22
4.4.2. EJERCICIOS CARGA DINÁMICA	23
5. EJERCICIOS COMPLETOS	27
5.1. SIMULACIÓN REPARTIDORES	27
5.2. SIMULACIÓN BANCO	29
6. RMI ASINCRONO	31
7. CONCLUSIONES	32
8. LÍNEAS FUTURAS	32
9. BIBLIOGRAFIA	33

TABLA DE IMAGENES

Imagen 1: Diagrama Gantt

Imagen 2: Funcionamiento Rmi

Imagen 3: Arquitectura Skeleton Stub

Imagen 4: Nota de deprecación de Skeleton

Imagen 5: Estructura servidor de nombres

Imagen 6: Funcionamiento servidor de nombres

Imagen 7: Diagrama del ejercicio 005-SimulaciónCoche

Imagen 8: Funcionamiento de Seguridad en Java

Imagen 9: Funcionamiento carga dinámica

Imagen 10: Diagrama AccesoEntreClientes

Imagen 11: Funcionamiento Simulación Repartidores

TABLA DE TABLAS

Tabla 1: Registro de nombres del ejercicio 005-SimulaciónCoche

Tabla 2: Registro de nombres simulación repartidores

TABLA DE ECUACIONES

Ecuación 1: Interés compuesto de cuenta

Ecuación 2: Préstamo total

1. INTRODUCCIÓN

Este documento es una guía completa sobre RMI que proporciona una referencia de los aspectos más importantes de esta tecnología. Comienza con una explicación breve de qué es RMI y cómo funciona de manera general. A continuación, se abordan diferentes temas relacionados con RMI, donde se presentan los fundamentos de forma general y se profundiza en los detalles y funcionamiento a través de ejercicios prácticos.

Además, se plantean dos ejercicios más complejos para reforzar los conocimientos adquiridos en los temas principales.

Antes de terminar se analizarán también las limitaciones de Java RMI en cuanto a su incapacidad para utilizar conexiones asíncronas, junto con las soluciones propuestas por la comunidad de Java para superar estas limitaciones.

Finalmente, se ofrece una recopilación de los puntos relevantes extraídos durante el desarrollo de este proyecto.

2. PLANIFICACIÓN

Antes de empezar con el trabajo, se vio necesario el hecho de pensar que clase de tareas se tenían que realizar, y su duración.

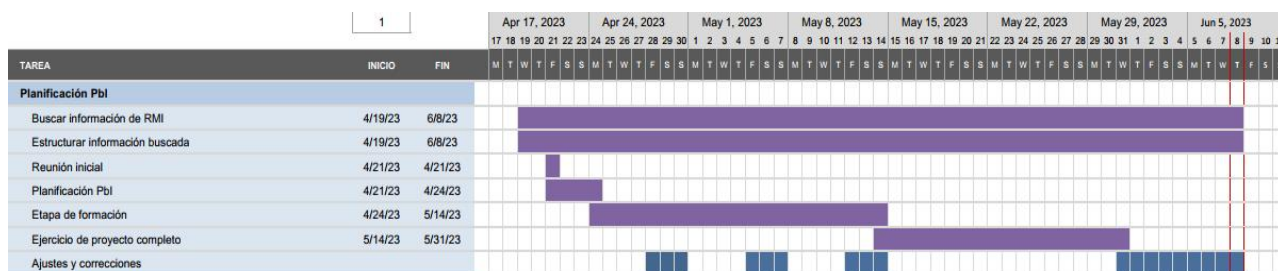


Imagen 1: Diagrama Gantt

Y el resto de la planificación podemos apreciarla a continuación. [LINK HACIA EL GANTT](#)

3. OBJETIVOS DEL PROYECTO

Los objetivos principales de este proyecto son los siguientes:

- Organizar la información recopilada en temas claros y estructurados, que permitan a los estudiantes seguir un progreso lógico y comprender la secuencia de conceptos.
- Diseñar y plantear ejercicios prácticos que permitan a los estudiantes aplicar los conocimientos teóricos adquiridos en situaciones reales y fortalecer su comprensión práctica de RMI.
- Crear una documentación útil y accesible para futuros usuarios.

4. TEMAS PRINCIPALES RMI

En este apartado, nos adentraremos en los temas fundamentales de Remote Method Invocation (RMI), una tecnología de comunicación en entornos distribuidos basada en Java. Exploraremos los conceptos esenciales de RMI y su aplicación en el desarrollo de sistemas distribuidos.

4.1. TEMA 1: ENTRADA/BASES DE RMI

En este tema, se explicarán los fundamentos de RMI, que permiten invocar métodos de objetos remotos de manera transparente. Además, se abordará la arquitectura utilizada para la comunicación entre sistemas remotos y las razones que llevaron al diseño del RMI moderno.

4.1.1. ¿QUÉ ES RMI?

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 JVM¹, a esto se le denomina comunicación remota.

RMI se utiliza para construir aplicaciones distribuidas; proporciona comunicación remota entre programas Java. Se proporciona en el paquete **java.rmi**.

4.1.2. 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.

4.1.3. 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, se debe emplear 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 parámetros.

¹ JVM: Abreviación de Java Virtual Machine, es una máquina virtual capaz de interpretar y ejecutar instrucciones expresadas en un código binario el cual es generado por el compilador del lenguaje Java.

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:

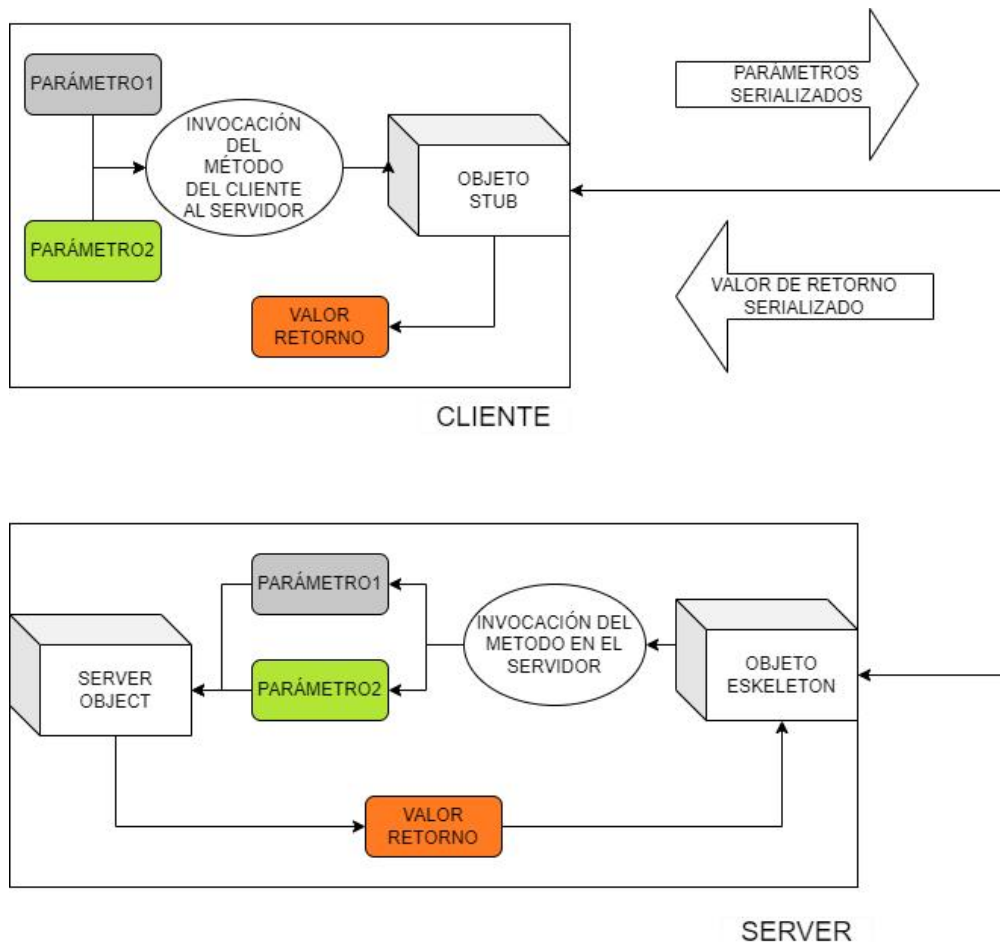


Imagen 2: Funcionamiento Rmi

- Encapsulado (marshalling) de los parámetros (utilizando la funcionalidad de serialización de Java).
- El cliente invoca el método del servidor para ejecutarlo. Para eso le pasa los parámetros serializados. Y se queda esperando su respuesta.
- El servidor recibe los parámetros y ejecuta el método 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.

4. 1. 4. SETUP BASICO, STUB Y SKELETON

Dado que RMI es un esquema de objetos distribuidos solo 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; solo 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 usando 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.

4. 1. 5. 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, este 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.

4. 1. 6. SKELETON

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 elimina la necesidad de esqueletos.

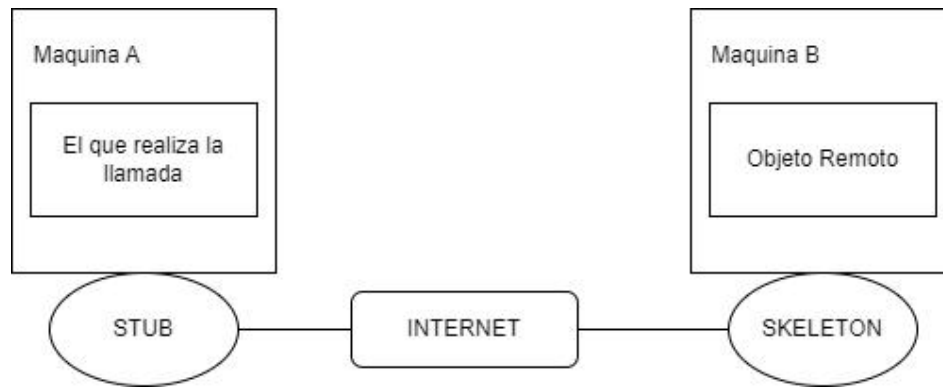


Imagen 3: Arquitectura Skeleton Stub

4. 1. 7. ESTADO ACTUAL DE LA ARQUITECTURA SKELETON STUB

Deprecation Note: Support for static generation of Java Remote Method Protocol (JRMP) stubs and skeletons has been deprecated. Oracle recommends that you use dynamically generated JRMP stubs instead, eliminating the need to use this tool for JRMP-based applications. See the `java.rmi.server.UnicastRemoteObject` specification at <http://docs.oracle.com/javase/8/docs/api/java/rmi/server/UnicastRemoteObject.html> for further information.

Imagen 4: Nota de deprecación de Skeleton

La arquitectura java RMI, se actualizó con la versión de [java 1.5](#), la cual introdujo los Stubs dinámicos eliminando la necesidad de utilizar Skeleton. Quitando totalmente la necesidad de compilar con la herramienta **rmic**.

4. 1. 8. EJERCICIOS: STUB SKELETON

001- Hello world con naming (módulo antiguo)

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

Explicación

Este ejercicio es para ilustrar, que RMI además del código, necesita levantar el servidor de forma separada, el proceso para levantar el servicio se explicara en el siguiente ejercicio.

En referencia a lo que pasa en el ejercicio, como el código se intenta ejecutar sin haber realizado la compilación del código ni el levantamiento del servicio Rmi, eclipse no será capaz de ejecutar el programa y la ejecución fallara, ya que no será capaz de establecer la conexión entre el cliente y el servidor.

Output

```

Problems @ Javadoc Declaration Console
MyServer [Java Application] C:\Users\Lenovo\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_15.0.1
java.rmi.ConnectException: Connection refused to host: 192.168.56.1; nested exception is:
    java.net.ConnectException: Connection refused: connect
  
```


002 - Compilación y ejecución (módulo antiguo)

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

1. Abre CMD , colócate 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.

Explicación

En este ejercicio se intenta mostrar el proceso de levantar el servicio de RMI, mediante la herramienta rmic.

La función de los comandos del enunciado es la siguiente:

- **javac <archivo java>** -> Compila el código 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 ejecución del servidor
- **java MyClient** -> Inicia la ejecución del cliente

Como curiosidad, si se levanta este servicio rmi en la línea de comandos, la ejecución del servidor y el cliente se pueden hacer de forma normal en eclipse. Debido a que el servicio permite la interconectividad entre servidor y cliente a través del puerto 5000.

Output

Terminal 1 el de la compilación y levantamiento del server

```
> javac .\Adder.java
> javac .\AdderRemote.java
> javac .\MyClient.java
> javac .\MyServer.java
> .\rmic 4.1.1_JAR\rmic-4.1.1-b001.jar AdderRemote
> rmiregistry 5000
```

-> El método rmiregistry que se está utilizando para levantar el servicio esta deprecated y el sistema lo notifica

```
WARNING: A terminally deprecated method in java.lang.System has been called
WARNING: System::setSecurityManager has been called by sun.rmi.registry.RegistryImpl
WARNING: Please consider reporting this to the maintainers of sun.rmi.registry.RegistryImpl
WARNING: System::setSecurityManager will be removed in a future release
```

Terminal 2: El servidor

```
\001- Hello world Naming\src> java MyServer
```

Terminal 3: El cliente

```
PS C:\Users\Lenovo\Documents\GitHub\PBL\zzz-Workplace\001- Hello world Naming\src> java MyClient
MENSAJE RECIBIDO-->38
```

-> Ejecución alterna en eclipse si el servicio rmi está levantado



```
MyServer [Java Application] C:\Users\Lenovo\p2\pool\pl
<terminated> MyClient [Java Application]
MENSAJE RECIBIDO-->38
```

4. 2. TEMA 2: NAMING REGISTRY

El servicio de nombres RMI, conocido como RMI Registry, es un componente que proporciona un mecanismo para buscar y acceder a objetos remotos dentro de una aplicación. Este actúa como un directorio centralizado donde los objetos remotos registran su ubicación para permitir que los clientes encuentren y obtengan referencias a los objetos remotos necesarios para realizar llamadas a métodos remotos.

4. 2. 1. FUNCIONAMIENTO

Al utilizar el RMI Registry, una aplicación puede registrar sus objetos remotos junto con un nombre único asociado a cada uno de ellos. Esto permite que otros componentes o aplicaciones accedan a esos objetos utilizando el nombre registrado, sin necesidad de conocer detalles específicos sobre su ubicación o implementación interna.

Una vez que un objeto remoto se registra en el RMI Registry, puede ser descubierto y utilizado por otras partes de la aplicación. Los clientes pueden buscar objetos remotos específicos en el registro utilizando su nombre único y obtener una referencia a ellos. Esta referencia actúa como un enlace entre el cliente y el objeto remoto, permitiendo al cliente invocar métodos en el objeto remoto de forma transparente, como si estuviera trabajando con un objeto local.

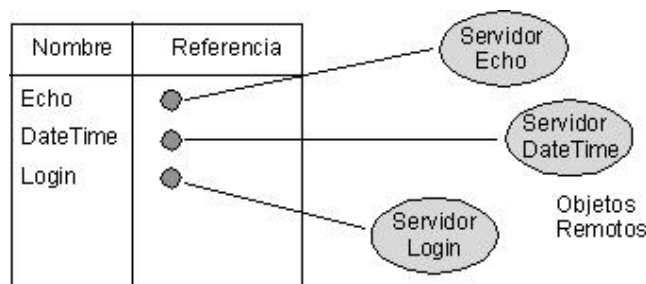


Imagen 5: Estructura servidor de nombres

El servicio de nombres RMI es fundamental en la comunicación entre objetos remotos. Cada objeto se identifica mediante un nombre único. A través de la interfaz `rmi.registry.Registry` y la clase `rmi.Naming`, podemos llamar a diversos métodos para administrar los objetos remotos en el registro, de la siguiente manera



Imagen 6: Funcionamiento servidor de nombres

La clase `rmi.Naming` ofrece funcionalidades para agregar, eliminar y acceder a objetos remotos en la tabla de registros. El servidor del servicio de nombres se encarga de registrar los objetos utilizando las llamadas **`bind()`** o **`rebind()`** en una instancia del registro correspondiente.

Cuando un cliente desea invocar un objeto remoto, utiliza el método **`lookup()`** para obtener la referencia del objeto en el registro mediante su nombre `bind`. Esto le permite interactuar con el objeto y realizar llamadas a métodos remotos.

El servicio de nombres se encuentra por defecto en el puerto 1099. Por otro lado, también es posible desarrollar nuestro propio servicio de nombres utilizando las clases e interfaces de **`java.rmi`** y seleccionar un puerto personalizado al lanzarlo.

4. 2. 2. SERIALIZACIÓN EN RMI

RMI es una tecnología que permite a los desarrolladores centrarse en el diseño de la lógica de la aplicación en lugar de los detalles de la transmisión de datos. RMI facilita el acceso remoto a objetos, permitiendo que se acceda a un objeto remoto como si fuera un objeto local.

Esta facilidad se debe a la serialización de objetos, que convierte un objeto en un flujo de bytes mediante la interfaz `Serializable` incluida en el paquete `java.io`. Este proceso puede ser revertido para crear una copia idéntica del objeto original.

Usando estas clases, un objeto 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 uso posterior.

4. 2. 3. EJERCICIOS: SERVIDOR DE NOMBRES Y SERIALIZACIÓN

003-Helloworld con Registry

Partiendo de los recursos iniciales de [“001 - Hello world con naming”](#), modifíquelo para utilizar el servidor de nombres de RMI, en vez de `Naming` y `NamingRegistry`.

Además, asegúrese de utilizar la misma clave para que los extremos de la conexión puedan comunicarse correctamente.

Explicación

La finalidad del ejercicio, es demostrar la diferencia entre la arquitectura usando `Skeletons`, y usando el servidor de nombres.

Para marcar esa diferencia se pretende actualizar el primer ejercicio realizado con `Stub Skeleton`, poniendo un registro de nombres en su lugar. Esto, además de simplificar el proceso, quita la necesidad de compilación y de utilizar la línea de comandos.

Output



004-Calculadora

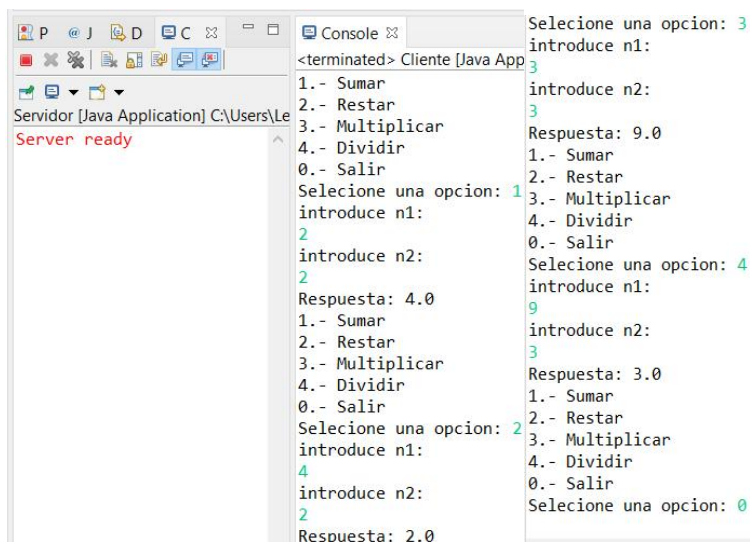
Partiendo del ejercicio anterior, se quiere desarrollar un programa al que sea posible realizar diferentes cálculos en función de lo necesario. Las opciones que puede realizar las siguientes:

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

Explicación

Este ejercicio está pensado para ilustrar que es posible hacer múltiples llamadas a un objeto registrado. Y que manteniendo una estructura igual, se pueden hacer diferentes acciones cambiando únicamente la interfaz.

Output



```

<terminated> Cliente [Java App
1.- Sumar
2.- Restar
3.- Multiplicar
4.- Dividir
0.- Salir
Seleccione una opcion: 1
introduce n1:
2
introduce n2:
2
Respuesta: 4.0
1.- Sumar
2.- Restar
3.- Multiplicar
4.- Dividir
0.- Salir
Seleccione una opcion: 2
introduce n1:
4
introduce n2:
2
Respuesta: 2.0
Seleccione una opcion: 3
introduce n1:
3
introduce n2:
3
Respuesta: 9.0
1.- Sumar
2.- Restar
3.- Multiplicar
4.- Dividir
0.- Salir
Seleccione una opcion: 4
introduce n1:
9
introduce n2:
3
Respuesta: 3.0
1.- Sumar
2.- Restar
3.- Multiplicar
4.- Dividir
0.- Salir
Seleccione una opcion: 0
  
```

005-SimulaciónCoche

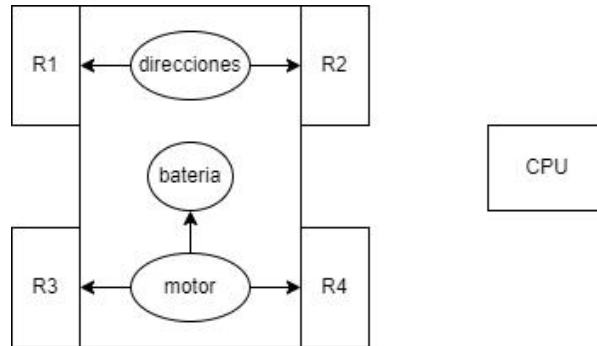


Imagen 7: Diagrama del ejercicio 005-SimulaciónCoche

Se quiere hacer un sistema que simule el funcionamiento de un coche eléctrico. Este está compuesto de diferentes módulos que funcionan siguiendo unas especificaciones muy concretas. Construye la arquitectura mostrada cumpliendo las siguientes especificaciones:

- La CPU está conectada a todos los elementos.
- Las ruedas delanteras R1 y R2, pueden girar derecha o izquierda.
- Las ruedas traseras R3 Y R4 pueden ir adelante o atrás.
- Todas las ruedas pueden frenar.
- La batería se consume un 5% cada vez que las ruedas van adelante o atrás.
- Las acciones también se tiene que reflejar en los componentes (servidores)

Las opciones de control en la CPU son los siguientes:

- 1.- Adelante
- 2.- Atrás
- 3.- Girar
- 4.- Parar
- 5.- Comprobar batería
- 0.- Salir

Explicación

Este ejercicio tiene como objetivo trabajar la conexión de un cliente con múltiples Stubs (servidores) que cuentan con diferentes implementaciones. Esto nos permitirá realizar acciones coordinadas y controlar el coche de manera efectiva.

Además, este ejercicio nos ayudará a comprender y trabajar correctamente las conexiones entre el cliente y el servidor. Como referencia las conexiones usadas son las siguientes:

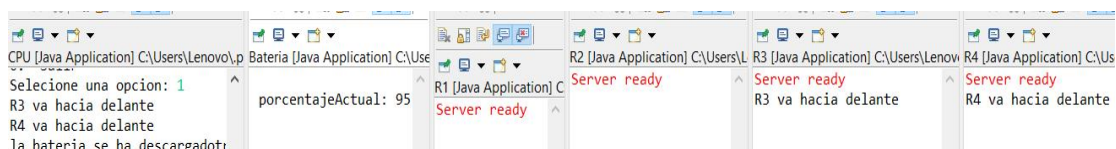
ELEMENTO	PUERTO	NOMBRE REGISTRO
R1	1091	direccion1
R2	1092	direccion2
R3	1093	motor3
R4	1094	motor4
Bateria	1095	bateria

Tabla 1: Registro de nombres del ejercicio 005-SimulaciónCoche

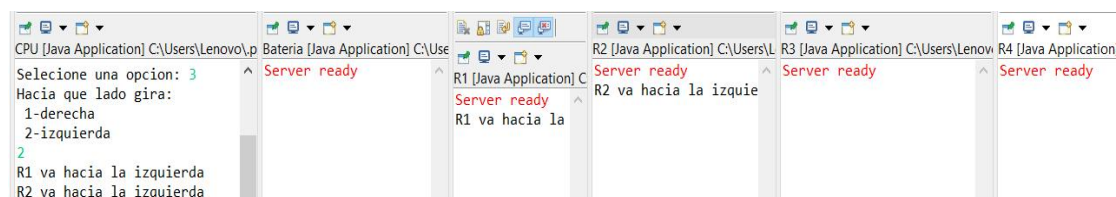
Nota adicional: cada puerto solo puede ser usado por 1 proceso a la vez.

Output

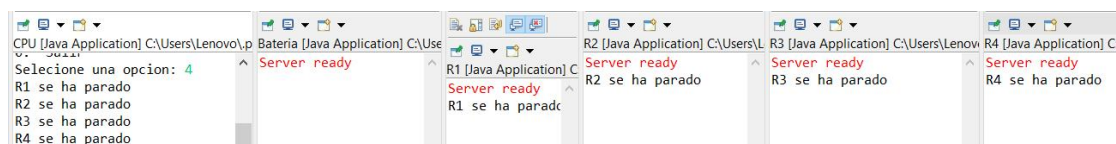
Opciones 1,2



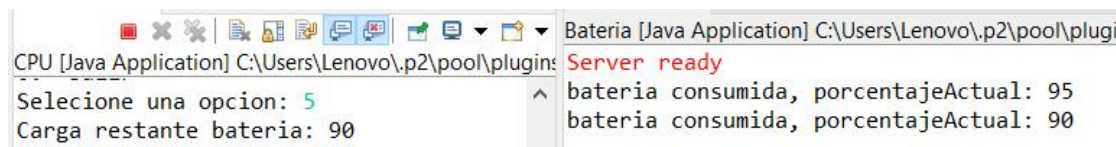
Opción 3



Opción 4



Opción 5



006- Saludador Serializado

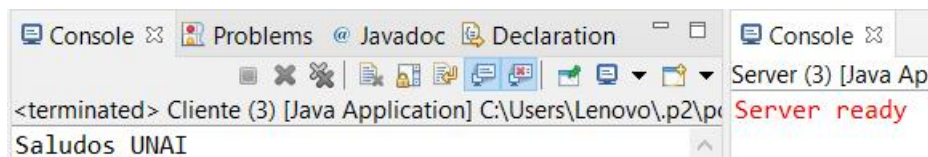
Partiendo del ejercicio “[003-Helloworld con Registry](#)” Crea un programa de hello world, que mande el objeto saludador del cliente al Servidor en vez de hacerlo con un simple String.

Explicación

En este ejercicio se ha hecho hincapié en trabajar todo el proceso de pasar objetos propios del cliente al servidor, para ello se ha creado el objeto Saludador y como es de nuestra creación, es necesario hacer que implemente la interfaz Serializable para que se pueda transmitir. Ya que de no aplicarlo saltara la excepción:

```
java.rmi.MarshalException: error marshalling arguments; nested exception is:
    java.io.NotSerializableException: client.Saludador
```

Output



007- Comprobador de Urls

Se requiere desarrollar un programa que permita validar URL. En el caso de que sea válida se debe mostrar el HTML de la página web referenciada por dicha URL. En caso contrario, el programa debe informar al usuario que la URL no es válida.

Explicación

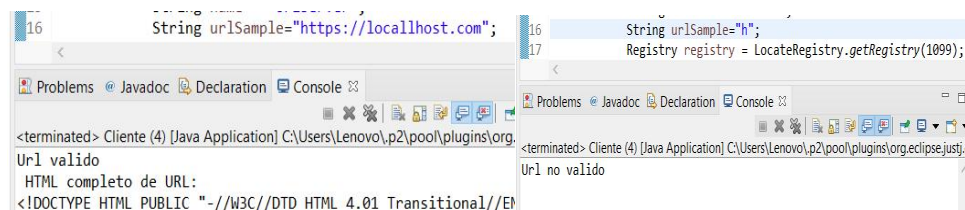
En este ejercicio se intenta trabajar a la transferencia de objetos nativos de Java desde el cliente al servidor. En este caso, se ha utilizado un objeto String que implementa de fábrica la interfaz Serializable y se ha pasado como URL.

Es importante destacar que el objeto URL utilizado en este ejercicio lanzará una excepción `UrlMalformedURLException` si la URL utilizada para la inicialización no es válida. Utilizaremos esta excepción para determinar si la URL es válida o no.

Si necesitas más información sobre el objeto URL, más información del objeto `Url`, en:

[22] Leer una URL con Java.

Output



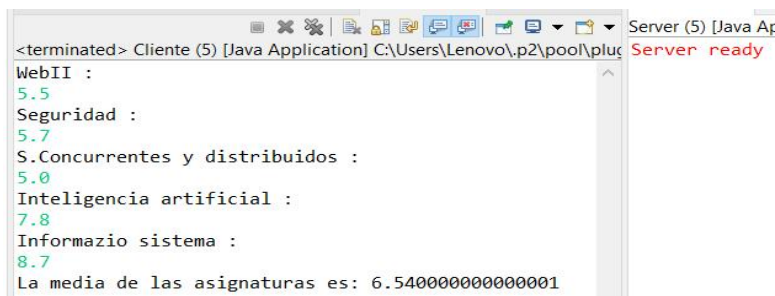
008- Meter notas del curso y calcula la media

Realiza un programa en Java RMI en el que solicites al usuario las notas de las 5 asignaturas del semestre y las guardes en un archivo. A continuación, el programa leerá el archivo y enviará los objetos de las asignaturas al servidor para calcular la nota media del curso..

Explicación

En este ejercicio, se sigue trabajando la serialización de los objetos en Java como en el anterior. En este caso, se utilizará la serialización para los objetos List y Double, permitiéndonos almacenar y transferir estos objetos de manera eficiente.

Output



```
<terminated> Cliente (5) [Java Application] C:\Users\Lenovo\p2\pool\plug
WebII :
5.5
Seguridad :
5.7
S.Concurrenates y distribuidos :
5.0
Inteligencia artificial :
7.8
Informazio sistema :
8.7
La media de las asignaturas es: 6.540000000000001
```

009- Transmisión de archivos entre cliente y server.

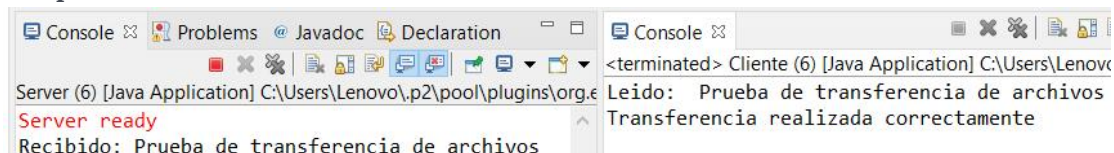
Desarrolla un programa que sea capaz de leer un archivo en el cliente y posteriormente transferirlo al servidor. El programa deberá validar si la transferencia se realizó exitosamente o si hubo algún error durante el proceso.

Explicación

En este ejercicio en particular, se pretende demostrar que no es imprescindible utilizar la interfaz Serializable cuando se emplea el tipo de datos byte para la transferencia de archivos. Esto se debe a que dicha transferencia se realiza directamente gracias a este tipo de datos. Es importante destacar que en el servidor solo se transfiere el contenido del archivo y no el objeto File en sí.

Esto es debido a que el objeto File solo es una representación del pathName del archivo de la máquina de origen, no de su contenido. Y en un servidor remoto no es posible manipular el archivo del origen desde el servidor, ya a que al cambiar de máquina se pierde el acceso al fichero original. De esta manera se evita cualquier tipo de pérdida de datos y se garantiza una transferencia confiable.

Output



```
Server (6) [Java Application] C:\Users\Lenovo\p2\pool\plugins\org.e
Server ready
Recibido: Prueba de transferencia de archivos
```

4. 3. TEMA 3: SEGURIDAD

Java realiza un importante control de seguridad en los programas, lo cual evita que los programas escritos en dicho lenguaje puedan dañar la información del sistema o acceder a información privada. Para lograr este objetivo, se implementan las siguientes medidas de seguridad:

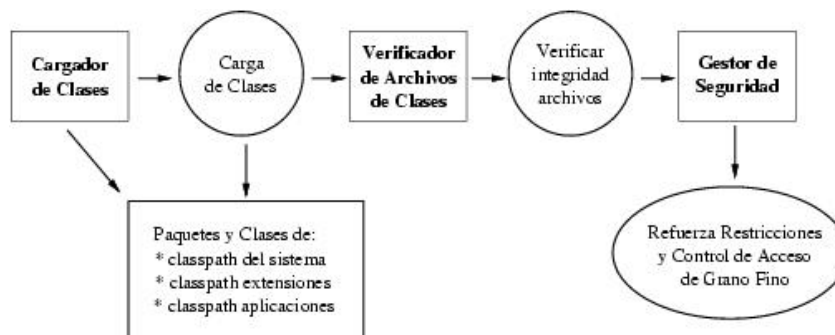


Imagen 8: Funcionamiento de Seguridad en Java

Cargador de clases: Este componente se encarga de cargar las clases de manera separada para evitar ataques. El orden de búsqueda de las clases es el siguiente: sistema, extensión y usuario. El cargador de clases es una clase Java que puede ser extendida para definir cargadores de clases especiales, pero solo por las aplicaciones. Ya que si un applet pudiera definir su propio cargador, podría modificar el cargador del sistema y potencialmente tomar el control de la máquina en la que se ejecuta el navegador.

Verificador de archivos de clases: La función de este componente es validar los bytecodes. El sistema distingue entre dos tipos de códigos:

- De confianza: Estas son generalmente las clases del sistema y las validadas por el usuario y no se validan.
- No confiables: Estos se validan.

Gestor de seguridad: Se encarga de verificar el acceso en tiempo de ejecución. Es una clase del sistema que puede ser extendida por las aplicaciones.

4. 3. 1. FICHEROS DE POLÍTICAS DE SEGURIDAD

Los archivos de política (policy) establecen las políticas de seguridad (permisos) que se aplican a los programas Java. En ausencia de especificaciones, estas políticas afectarán a todos los programas del sistema. Sin embargo, es posible restringir el alcance de la política al especificar las siguientes propiedades:

- Base de códigos (CodeBase): Hace referencia a la ubicación en el sistema (ruta o URL) de las implementaciones de las interfaces utilizadas por el código, indicando desde dónde se descargan dichas implementaciones.
- Firmado por (Signed by): Aquí se puede especificar quién debe haber firmado el programa para que se le aplique la política correspondiente.

Para utilizar el archivo de políticas definido, existen dos opciones: añadir la política en archivo de propiedades de seguridad o incorporarlo como propiedad del sistema al ejecutar el intérprete de Java. De esta manera, nos aseguramos de que se utilice la política establecida en el archivo correspondiente.

Por defecto, una aplicación Java no instala ningún gestor de seguridad. Si deseamos imponer restricciones de seguridad, debemos definir e implementar un gestor de seguridad adecuado.

4.3.2. GESTOR DE SEGURIDAD

El gestor de seguridad (SecurityManager) es el encargado de determinar si una determinada operación está permitida o no, impidiendo su ejecución en caso contrario. Cuando una aplicación carga un gestor de seguridad, todas las acciones sujetas a posibles restricciones de seguridad se verifican en dicho gestor antes de llevarse a cabo.

En las aplicaciones independientes, por defecto no se carga ningún gestor de seguridad. Sin embargo, podemos hacer que se cargue especificándolo en la línea de comando, como se mencionó anteriormente, o bien desde el propio código de nuestra aplicación.

El gestor de seguridad por defecto (clase SecurityManager) es el que se carga para los Applets. Este gestor sigue la política de seguridad indicada en los archivos de políticas que hemos especificado (tanto en el archivo de propiedades de seguridad como en la línea de comando).

Además de utilizar el gestor de seguridad por defecto, también tenemos la opción de definir nuestro propio gestor de seguridad creando una subclase de SecurityManager y sobrescribiendo los métodos relevantes para personalizar su comportamiento.

Para instalar un gestor de seguridad de manera que se utilice para establecer las operaciones permitidas, podemos utilizar el método setSecurityManager de la clase System.

GESTOR DE SEGURIDAD EN RMI

Uno de los problemas más comunes que se encuentra al trabajar con RMI son las restricciones de seguridad del estándar de Java.

Sin embargo, en Java RMI se utiliza la clase RMISecurityManager, que aplica la política de seguridad a las clases que se cargan como stubs para objetos remotos. Esta clase sobrescribe todos los métodos relevantes de comprobación de acceso del SecurityManager estándar de Java, proporcionando así un control más específico y adaptado a RMI.

4.3.3. RESTRICCIONES DE SEGURIDAD EN APPLETS Y APLICACIONES JAVA

Los applets son programas Java que se pueden descargar desde fuentes remotas y ejecutarse automáticamente. Debido al riesgo que representan, estos programas están sujetos a una serie de restricciones importantes:

- No pueden acceder a métodos nativos.
- No pueden leer ni escribir en los archivos de la máquina local donde se ejecutan, a menos que se proporcione la URL absoluta del archivo.
- No pueden establecer conexiones de red con hosts diferentes al host desde el cual se descargó el applet.
- No pueden ejecutar programas en la máquina donde se están ejecutando.
- No pueden leer las propiedades del sistema.

Es posible modificar la configuración para eliminar algunas de estas restricciones, estableciendo los permisos necesarios para ello. Del mismo modo, aunque las aplicaciones no tengan estas restricciones de seguridad de forma predeterminada, es posible aplicarles estas restricciones mediante la configuración correspondiente.

4.3.4. DESCARGA DE CÓDIGO/CLASES REMOTO

Debido a las restricciones mencionadas en el apartado de Restricciones de seguridad en applets y aplicaciones, la descarga dinámica de clases desde destinos remotos está completamente restringida en las políticas por defecto de Java.

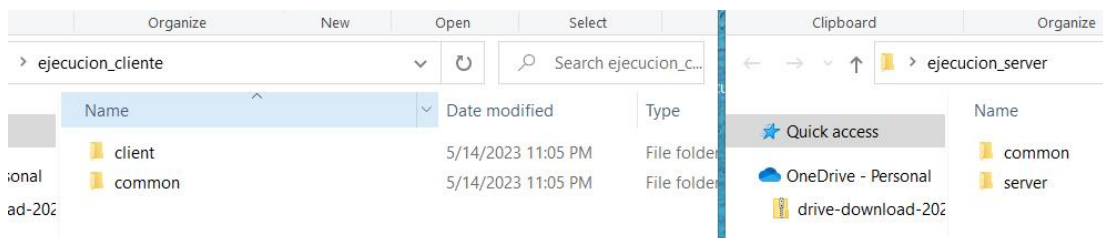
Para llevar a cabo este tipo de operaciones, se requiere un gestor de seguridad que sobrescriba los permisos actuales. Por ejemplo, en RMI, se descargará una clase Serializable desde otra máquina solo si existe un gestor de seguridad y dicho gestor permite la descarga de la clase desde esa máquina.

Esta acción se controla mediante el valor de la propiedad `java.rmi.server.useCodebaseOnly`. Establecer esta propiedad en falso habilita la carga remota de código, lo cual incrementa el nivel de riesgo de seguridad del sistema. Cabe destacar que su valor por defecto es true.

4.3.5. PROCESO DE EJECUTAR EN DIFERENTES MÁQUINAS

Los pasos que se deben seguir para ejecutar el código son los siguientes:

1. Compilar el código: Esto se puede hacer manualmente o se puede encontrar el código compilado en la carpeta "bin" del proyecto.
2. Copiar los archivos ".class" y la estructura de carpetas del proyecto a una ubicación separada.
3. Para la ejecución, se requiere levantar la máquina virtual utilizando el comando "**java <classpath>**" de la siguiente manera:



```
#
| server
|
| Server.class
#
| client
|
| Client.class
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Lenovo> cd "C:\Users\Lenovo\Desktop\ejecucion_server"
PS C:\Users\Lenovo\Desktop\ejecucion_server> java server.Server
Server ready
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Lenovo> cd "C:\Users\Lenovo\Desktop\ejecucion_cliente"
PS C:\Users\Lenovo\Desktop\ejecucion_cliente> java client.Client
He encontrado el registry
he encontrado el servidor
response: Hello World!
PS C:\Users\Lenovo\Desktop\ejecucion_cliente>
```

4.3.6. EJERCICIOS DE SEGURIDAD

010- Security hello world example

Partiendo del ejercicio “ [003-Helloworld con Registry](#)” aplica los siguientes cambios para configurar la seguridad de RMI:

- Cambia las propiedades del programa para habilitar la descarga dinámica del código en la aplicación.
- Aplica RMISecurityManager y sobrescribe el SecurityManager estándar de Java.
- Establece una política de seguridad que habilite todos los permisos.

Finalmente, sigue los pasos para realizar una ejecución remota en otra máquina.

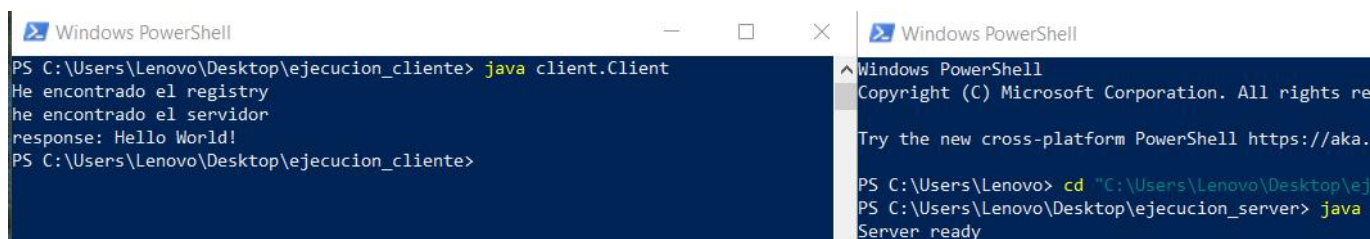
Explicación

La finalidad de este ejercicio es crear una base con las configuraciones mínimas de seguridad para habilitar la descarga dinámica y entender mediante la práctica como se configura la seguridad en RMI. La propiedad que permite la descarga dinámica de código debe configurarse únicamente en el servidor. Algunos de los puntos más importantes son los siguientes:

- Para facilitar la búsqueda, se ha configurado el cliente con la dirección IP del ordenador en el registro de búsqueda (lookup).
- Es importante tener en cuenta que cada CMD se trata como una JVM independiente dentro del sistema.

NOTA: Durante la ejecución del programa, hay que tener en cuenta los ajustes del firewall tanto en el ordenador como en la red en la que te encuentres, ya que suelen ser las razones principales de los errores.

Output



```
PS C:\Users\Lenovo\Desktop\ejecucion_cliente> java client.Client
He encontrado el registry
he encontrado el servidor
response: Hello World!
PS C:\Users\Lenovo\Desktop\ejecucion_cliente>

PS C:\Users\Lenovo> cd "C:\Users\Lenovo\Desktop\ejecucion_server"
PS C:\Users\Lenovo\Desktop\ejecucion_server> java -Djava.rmi.server.hostname=127.0.0.1 -jar RMIHelloWorldExample.jar
Server ready
```

4. 4. TEMA 4: CARGA DINÁMICA

Una de las características más notables de la plataforma Java es su capacidad para descargar dinámicamente software Java desde cualquier URL (Uniform Resource Locator) a una máquina virtual Java (JVM) que se ejecuta en un proceso diferente, normalmente en un sistema distinto.

Java RMI utiliza esta capacidad para descargar y ejecutar clases en sistemas donde dichas clases nunca han sido instaladas. Con la API de RMI, cualquier JVM puede descargar cualquier archivo de clases de Java, incluyendo las clases stub, y de esta manera, puede ejecutar llamadas a métodos remotos en un servidor remoto usando los recursos del sistema servidor.

La propiedad codebase se emplea para definir la ubicación de las clases que una aplicación necesita cargar en una JVM. Antes de enviar un objeto, el programa debe definir la propiedad codebase para que el receptor pueda conocer la ubicación de descarga de dicho código, en caso de no tenerlo disponible de forma local. Una vez el objeto ha sido de serializado, el receptor obtendrá el codebase y cargará las clases desde la ubicación correspondiente.

4. 4. 1. FUNCIONAMIENTO

A través de RMI, las aplicaciones pueden crear objetos remotos que reciben llamadas a métodos desde clientes en otras JVMs. Para establecer esta comunicación, RMI utiliza clases especiales llamadas stubs, que se descargan al cliente para facilitar la interacción con el objeto remoto.

Al igual que los applets, las clases necesarias para las llamadas a métodos remotos se pueden descargar desde URLs de tipo "file:///", pero esto requiere que el cliente y el servidor estén en el mismo host físico, a menos que se utilice otro protocolo como NFS para acceder a un sistema de archivos remoto.

En la mayoría de los casos, es preferible tener las clases necesarias para las llamadas a métodos remotos disponibles en un recurso de red, como un servidor HTTP o FTP.

La descarga de los stubs RMI se realiza de la siguiente manera:

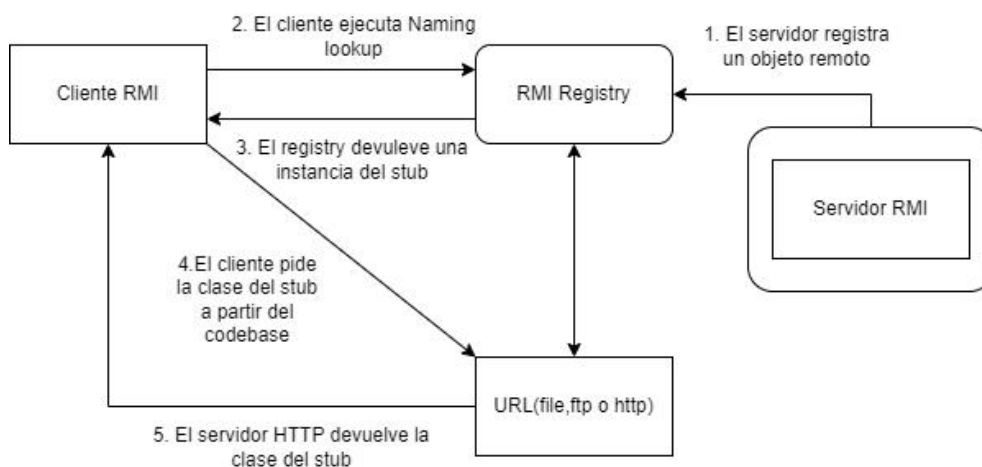


Imagen 9: Funcionamiento carga dinámica

1. El codebase de los objetos remotos se especifica en el servidor del objeto remoto mediante la propiedad `java.rmi.server.codebase`. El servidor RMI registra un objeto remoto, asignándole un nombre, en el registro RMI. El codebase establecido en la JVM del servidor se "anota" en la referencia al objeto remoto en el registro RMI.
2. Cuando el cliente RMI solicita una referencia a un objeto remoto, se le devuelve una instancia del stub del objeto remoto. Esta referencia será usada por el cliente para realizar llamadas a métodos del objeto remoto.
3. El registro RMI proporciona una referencia (la instancia del stub) a la clase solicitada. Si la definición de la clase para el stub se encuentra localmente en el CLASSPATH del cliente, que se busca antes que el codebase, el cliente cargará la clase localmente. Sin embargo, si la definición del stub no se encuentra en el CLASSPATH del cliente, este intentará recuperar la definición de la clase a partir del codebase del objeto remoto.
4. Luego, el cliente solicita la definición de la clase desde el codebase. El codebase empleado por el cliente es el URL asociado con la instancia del stub cuando la clase del stub fue cargada por el registro en el paso 1.
5. La definición de la clase del stub y cualquier otra clase necesaria se descargan en el cliente. Los pasos 4 y 5 son los mismos que el registro tuvo que realizar para cargar las clases del objeto remoto durante el registro inicial. Cuando el registro intentó cargar la clase del stub del objeto remoto, solicitó la definición de dicha clase consultando el codebase asociado con el objeto remoto.
6. Ahora el cliente tiene toda la información necesaria para invocar métodos en el objeto remoto. La instancia del stub actúa como un proxy del objeto remoto que existe en el servidor.

4. 4. 2. EJERCICIOS CARGA DINÁMICA

011- Cálculo dinámicos finanzas

Partiendo del ejercicio "011-Compute Engine Carga Dinámica (base)", implementa la funcionalidad para que los clientes puedan realizar los siguientes cálculos: sin realizar cambios en el servidor ni en las interfaces.

Interés compuesto:

$$Total = Inicial(1 + (Interés/NCapíAño))^{numCapíAño*años}$$

Ecuación 1: Interés compuesto de cuenta

El total a devolver de un préstamo:

$$Total = prestamo + (tasaInterés * plazoAños)$$

Ecuación 2: Préstamo total

Explicación

En este ejercicio, se pretende trabajar el uso de múltiples clientes que implementan la interfaz Compute, cada uno con su propia implementación. El objetivo es demostrar que, sin realizar modificaciones, es posible utilizar el servidor para ejecutar diferentes funciones a través de los distintos clientes.

Output

```

ComputeEngine (1) [Java Application] C:\Users\Lenovo\p2\pool\plugins\orc
ComputeEngine bound
Compound Interest after 5 years: 489.8457083016051
Amount after 5 years: 1489.845708301605
Interés: 240.0000000000000049960036108132044319063425064
Total: 1240.0000000000000049960036108132044319063425064

```

```

<terminated> ComputePrestamo [Java Application] C:\Users\Leno<terminated> ComputeInteresCompuesto
Introduce prestamo inicial:           Introduce monton inicial:
1000                                  1000
A cuanta tasa de interes:             Tasa anual de intereses :
8                                      8
El plazo de devolucion en años:       años en deposito:
3                                      5
Cantidad total de prestamo a devolver: 1240.00 € Cantidad final: 1489.85 €

```

012- Conversor de monedas

Desarrolla una aplicación de conversión de monedas que permita convertir una cantidad de EUROS a 9 tipos de monedas diferentes: Dólar Estadounidense (USD), Yen japonés (JPY), Libra esterlina (GBP), Dólar australiano (AUD), Dólar canadiense (CAD), Franco suizo (CHF), Renminbi chino (CNY), Dólar hongkonés (HKD) y Dólar neozelandés (NZD).

Explicación

En este ejercicio, se busca trabajar el uso de los enums como un elemento común entre el cliente y el servidor, además de proporcionar un ejemplo sencillo de transacciones monetarias

Para ver los valores que tienen las monedas mencionadas respecto al euro, es necesario hacer una búsqueda en el momento, debido a que el valor de las divisas del ejercicio cambia frecuentemente, un buen ejemplo para buscarlo sería el siguiente:

cambio de divisas euro a <moneda>

.Output

```

Introduce monton inicial(en euros):
1000
Elige moneda:
1-Dolar Estadounidense(USD)
2-Yen japones(JPY)
3-libra esterlina (GBP)
4-El dólar australiano (AUD)
5-El dólar canadiense (CAD)
6-franco suizo (CHF)
7-renminbi chino (CNY)
8-dólar hongkonés (HKD)
9-dólar neozelandés (NZD)
Elegido:
2
Cantidad en moneda seleccionada: 148126.68

```


013- AccesoEntreClientes

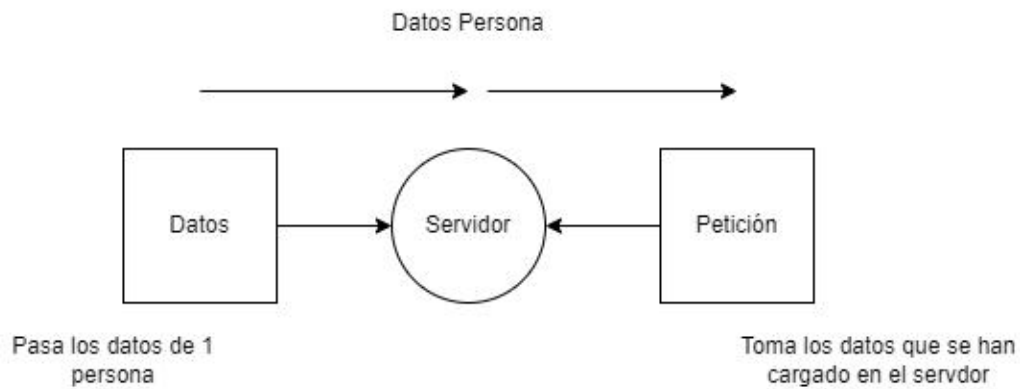


Imagen 10: Diagrama AccesoEntreClientes

Desarrolla una aplicación cliente-servidor con carga dinámica de métodos, donde el cliente 1 lanza un método que necesita datos del cliente 2. El cliente 2 ejecuta un método para proporcionar los datos al servidor, y el servidor ejecuta el método del cliente 1 con los datos del cliente 2 recibidos, enviando el resultado al cliente 1.

Explicación

En este ejercicio se intenta trabajar, el tener, la implementación de los métodos de cálculo y gestión, separada de la implementación de los datos. Haciendo una distribución de ambos en diferentes clientes, que se comunican a través de un servidor, que solo ejecuta lo que le mandan sin saber nada de nada.

Output

```
<terminated> ComputePi (2) [Java App]
1 Foo 125
```

-> recibimos los datos del Baul en ComputePi

5. EJERCICIOS COMPLETOS

En este apartado, se plantean 2 ejercicios completos, para poner a prueba los conocimientos y capacidades adquiridas en el apartado anterior. Además de poner a prueba, la capacidad de diseño y comprensión de arquitecturas del alumno.

5. 1. SIMULACIÓN REPARTIDORES

Desarrolla una aplicación que simule un sistema de gestión de pedidos de hamburguesas. El sistema consta de un cliente que genera un pedido y lo envía a una empresa de gestión de pedidos. La empresa de gestión de pedidos cuenta con tres empresas de reparto subcontratadas: Subcontrata 1, Subcontrata 2 y Subcontrata 3.

El objetivo del sistema es garantizar que el pedido siempre sea entregado por la subcontrata con el número más bajo. Y que el gestor de pedidos enseñe la cantidad total facturada hasta el momento.

Además el sistema debe ser capaz de manejar situaciones en las que el servidor de una subcontrata esté caído. En ese caso, se debe delegar el pedido a la siguiente subcontrata con el número más bajo que esté operativa. El objetivo es garantizar que el pedido se entregue lo más rápido posible. En caso de que no haya ninguna subcontrata disponible, el pedido se rechazara.

La aplicación debe ser capaz de mostrar el estado de las subcontratas (encendidas o apagadas) y notificar al cliente de la situación de su pedido. Finalmente, se deben implementar manejo de errores para garantizar la integridad y confiabilidad del sistema.

Explicación

La llamada de funciones de un servidor a otro es totalmente procedural, y funciona de la siguiente manera.

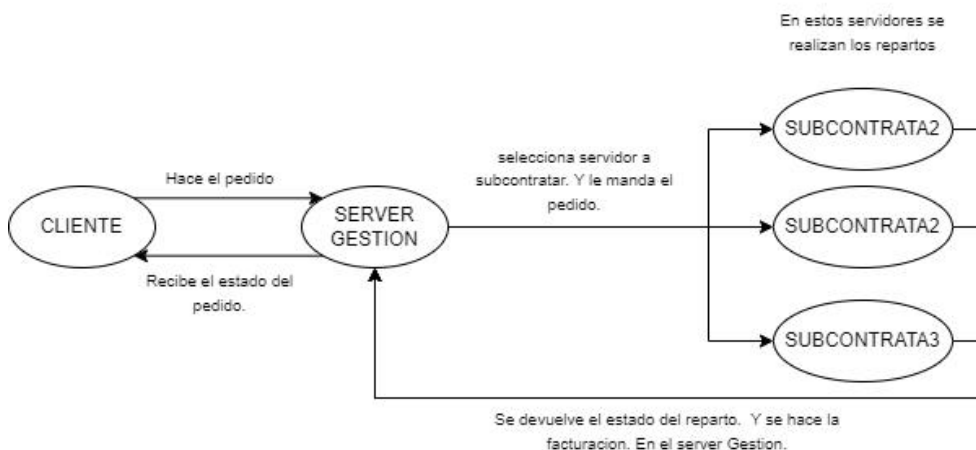


Imagen 11: Funcionamiento Simulación Repartidores

Los datos de conexion de los diferentes servidores es la siguiente.

	Puerto	Nombre
Servidor Gestion	1090	ServerGestion
Subcontrata1	1091	Subcontrata1
Subcontrata2	1092	Subcontrata2
Subcontrata3	1093	Subcontrata3

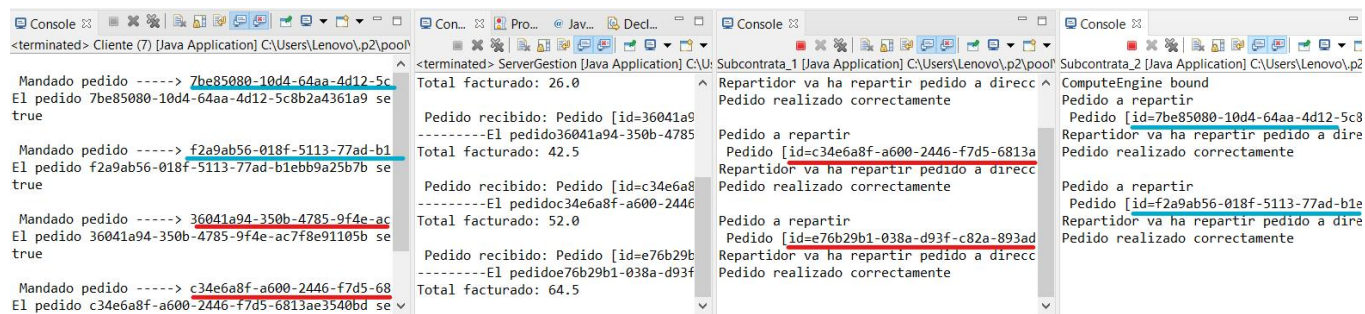
Tabla 2: Registro de nombres simulación repartidores

El cliente únicamente se conecta al servidor Gestión.

Para hacer la comprobación de los servidores, se llama a todos los servidores Subcontratas, y si falla se recoge en una excepción, y si conecta se le manda el pedido y se corta el bucle.

Output

En los primeros 2 pedidos ha estado encendido la subcontrata 2, después se ha encendido la subcontrata 1, como se ve el tráfico se redirige a la subcontrata 1



```

<terminated> Cliente (7) [Java Application] C:\Users\Lenovo\p2\pool\
Mandado pedido -----> 7be85080-10d4-64aa-4d12-5c
El pedido 7be85080-10d4-64aa-4d12-5c8b2a4361a9 se
true

Mandado pedido -----> f2a9ab56-018f-5113-77ad-b1
El pedido f2a9ab56-018f-5113-77ad-b1ebb9a25b7b se
true

Mandado pedido -----> 36041a94-350b-4785-9f4e-ac
El pedido 36041a94-350b-4785-9f4e-ac7f8e91105b se
true

Mandado pedido -----> c34e6a8f-a600-2446-f7d5-68
El pedido c34e6a8f-a600-2446-f7d5-6813ae3540bd se
true

<terminated> ServerGestion [Java Application] C:\U
Total facturado: 26.0

Pedido recibido: Pedido [id=36041a9
-----El pedidoc36041a94-350b-4785
Total facturado: 42.5

Pedido recibido: Pedido [id=c34e6a8
-----El pedidoc34e6a8f-a600-2446
Total facturado: 52.0

Pedido recibido: Pedido [id=e76b29b
-----El pedidoe76b29b1-038a-d93f
Total facturado: 64.5

Subcontrata_1 [Java Application] C:\Users\Lenovo\p2\pool
Repartidor va ha repartir pedido a direcc
Pedido realizado correctamente

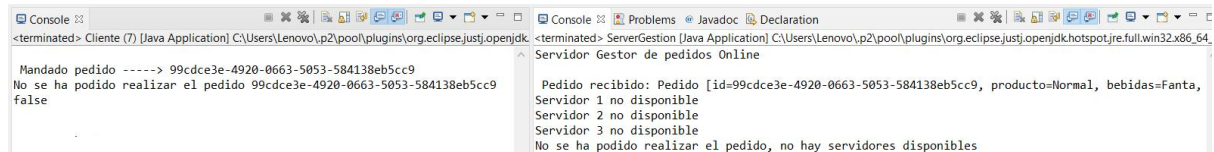
Pedido a repartir
Pedido [id=c34e6a8f-a600-2446-f7d5-6813a
Repartidor va ha repartir pedido a direcc
Pedido realizado correctamente

Pedido a repartir
Pedido [id=e76b29b1-038a-d93f-c82a-893ad
Repartidor va ha repartir pedido a direcc
Pedido realizado correctamente

Subcontrata_2 [Java Application] C:\Users\Lenovo\p2
ComputeEngine bound
Pedido a repartir
Pedido [id=7be85080-10d4-64aa-4d12-5c8
Repartidor va ha repartir pedido a dire
Pedido realizado correctamente

Pedido a repartir
Pedido [id=f2a9ab56-018f-5113-77ad-b1e
Repartidor va ha repartir pedido a dire
Pedido realizado correctamente
  
```

Caso en la que no hay ninguna subcontrata funcionando



```

<terminated> Cliente (7) [Java Application] C:\Users\Lenovo\p2\pool\plugins\org.eclipse.justi.openjdk
Mandado pedido -----> 99cdce3e-4920-0663-5053-584138eb5cc9
No se ha podido realizar el pedido 99cdce3e-4920-0663-5053-584138eb5cc9
false

<terminated> ServerGestion [Java Application] C:\Users\Lenovo\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64
Servidor Gestor de pedidos Online

Pedido recibido: Pedido [id=99cdce3e-4920-0663-5053-584138eb5cc9, producto=Normal, bebidas=Fanta,
Servidor 1 no disponible
Servidor 2 no disponible
Servidor 3 no disponible
No se ha podido realizar el pedido, no hay servidores disponibles
  
```

5. 2. SIMULACIÓN BANCO

Desarrolla una aplicación utilizando RMI que simule un sistema bancario. El sistema consta de un cliente y dos servidores: un servidor de banco y un servidor de bóveda.

El cliente debe ser capaz de realizar las siguientes transacciones:

- 1- Consultar saldo.
- 2- Realizar depósito.
- 3- Realizar retiro.
- 4- Transferir fondos.
- 5- Solicitar un préstamo a 6 meses.

El servidor de banco recibe las instrucciones de transacciones bancarias por parte de los clientes y las envía al servidor de bóveda para que este manipule los datos de todas las cuentas. Además, registra un registro de todas las transacciones que pasan por él.

El servidor de bóveda se encarga de gestionar la bóveda y realizar todas las operaciones necesarias en las cuentas bancarias. Debe tener en cuenta todos los posibles casos que pueden surgir durante las transacciones.

Toda la comunicación entre el cliente, el servidor de banco y el servidor de bóveda debe ejecutarse mediante carga dinámica, utilizando RMI para la transmisión de objetos y llamadas a métodos remotos. Además, se deben implementar mecanismos de seguridad para proteger la comunicación y manejar errores y excepciones de manera adecuada, a fin de proporcionar una experiencia de usuario robusta y confiable.

Explicación

Se ha utilizado la comunicación mediante cadenas de texto (String) para permitir el envío de diferentes comandos utilizando el mismo tipo de variable. Esta elección se debe a su facilidad de manejo y flexibilidad.

En los mensajes, se ha empleado el carácter "_" como divisor en lugar del "." para evitar problemas en comandos que contengan espacios. Esto se debe a que el uso del "." presentaba inconvenientes al procesar cantidades en formato decimal, ya que al dividir la cadena se perdían los decimales.

Al desarrollar los programas, se han considerado los siguientes casos de uso:

- No se puede retirar más dinero del disponible en la cuenta en ninguna de las transacciones.
- Es necesario verificar la existencia de los IDs de los clientes involucrados en la transacción, para realizar la operación.
- En el caso de los préstamos, se requiere realizar una llamada al banco para verificar el estado y otra llamada para realizar o devolver el préstamo

- Si no existe un préstamo, se debe ofrecer la opción de solicitarlo. Si el cliente acepta, se procede a efectuar el préstamo.
- Al devolver el préstamo en su totalidad, el estado del préstamo debe cambiarse a "false". En caso de devolver una cantidad mayor que el monto restante del préstamo, solo se cobrará el importe del préstamo de la cuenta.

Output

```

ComputePi (2) [Java Application] C:\Users\Lenovo\p2\pool\plugin Banco (1) [Java Application] C:\Users\Lenovo\p2\pool\plugins\org.ec Baul de datos operativo
Opciones:
1- ConsultarSaldo
2- Ingresar saldo
3- Extraer saldo
4- Transferencia
5- Pedir prestamo a 6 meses
0- Salir
Opcion: 1
Saldo total disponible para cliente id[1]:
5311.37
1_CONSULTAR

```

```

Console Console Problems Javadoc Declaration Console
ComputePi (2) [Java Application] C:\Users\Lenovo\p2\pool\plugin Banco (1) [Java Application] C:\Users\Lenovo\p2\pool\plugins\org.ec Baul de datos operativo
Opciones:
1- ConsultarSaldo
2- Ingresar saldo
3- Extraer saldo
4- Transferencia
5- Pedir prestamo a 6 meses
0- Salir
Opcion: 2
Introduce la cantidad a INGRESAR:3000
Operacion hecha de forma correcta
Dinero aportado: 3000.0
Saldo tras operacion: 8622.8
1_INGRESAR_3000.0

```

```

ComputePi (2) [Java Application] C:\Users\Lenovo\p2\pool\plugin Banco (1) [Java Application] C:\Users\Lenovo\p2\pool\plugins\org.ec Baul de datos operativo
Opciones:
1- ConsultarSaldo
2- Ingresar saldo
3- Extraer saldo
4- Transferencia
5- Pedir prestamo a 6 meses
0- Salir
Opcion: 3
Introduce la cantidad a RETIRAR:2000
Operacion hecha de forma correcta
Dinero retirado:2000.0
Saldo tras operacion: 4406.64
1_RETIRAR_2000.0

```

Quitar más del saldo disponible

```

ComputePi (2) [Java Application] C:\Users\Lenovo\p2\pool\plugins\org.eclipse.just Banco (1) [Java Application] C:\Users\Lenovo\p2\pool\plugins\ Baul de datos operativo
Opciones:
1- ConsultarSaldo
2- Ingresar saldo
3- Extraer saldo
4- Transferencia
5- Pedir prestamo a 6 meses
0- Salir
Opcion: 3
Introduce la cantidad a RETIRAR:10000
Dinero insuficiente en cuenta intentelo con otra cantidad
1_RETIRAR_2000.0
1_RETIRAR_10000.0

```

```

ComputePi (2) [Java Application] C:\Users\Lenovo\p2\pool\plugins\org.eclipse.just Banco (1) [Java Application] C:\Users\Lenovo\p2\pool\plugins\ Baul de datos operativo
Opciones:
1- ConsultarSaldo
2- Ingresar saldo
3- Extraer saldo
4- Transferencia
5- Pedir prestamo a 6 meses
0- Salir
Opcion: 4
Introduce id al que le quieras hacer la transferencia: 2
Introduce la cantidad a MANDAR:2000
Transferencia realizada correctamente
1_TRANSFERENCIA_2000.0_2

```

Devolver préstamo

```

ComputePi (2) [Java Application] C:\Users\Lenovo\p2\pool\plugins\org.eclipse.just.openjdk.hotspot.jre.full Banco (1) [Java Application] C:\Users\Lenovo\p2\pool\plugins\ Baul de datos operativo
Opcion: 5
Préstamo activo
Total del préstamo a devolver 3416.2299999999996
Quieres devolver parte del Préstamo:
1-SI
2-NO
1
cantidad a devolver: 50000
Operacion hecha de forma correcta
Dinero retirado:3416.2299999999996
Total del préstamo restante: 3416.2299999999996
Saldo tras operacion: 99054.53
Préstamo íntegramente pagado, muchas gracias por su esfuerzo
1_CONSULTARPRESTAMO
1_DEVOLVERPRESTAMO_2000.0
1_INGRESAR_100000.0
1_CONSULTARPRESTAMO
1_DEVOLVERPRESTAMO_50000.0

```

6. RMI ASINCRONO

RMI es la implementación de Remote Procedure Call en Java. Por definición, no es posible utilizar la API para llamadas asíncronas, ya que no está diseñada con ese propósito.

Incluso si se intentara implementar de forma asíncrona, surgirían los siguientes problemas:

- Problema del puente heredado: cuando hay un gran número de métodos en varias clases y se necesita construir un puente hacia estos métodos para reutilizar las implementaciones en el código existente, nos encontramos con la restricción de no poder cambiar el código existente. Además, es posible que ni siquiera tengamos acceso al código fuente existente.
- Problema del sintetizador de proxy virtual: el objetivo de este es reenviar el método a una implementación existente. Se utilizan entradas del problema del puente heredado y genera código que puede invocarse en un espacio de direcciones remoto.
- Problema de la invocación asíncrona: cuando se tienen un conjunto de invocaciones síncronas con retornos síncronos, encontrar un patrón de diseño que permita invocaciones asíncronas con retornos asíncronos. Siendo la alternativa bloquear el hilo de ejecución de los invocadores.

Teniendo en cuenta estos problemas, está claro que RMI (Remote Procedure Call) por sí solo no es adecuado para realizar llamadas asíncronas, dado que está diseñado para realizar llamadas de forma procedural, como hemos visto.

Sin embargo, esto no significa que RMI no pueda ser utilizado de manera asíncrona. De hecho, existen proyectos que demuestran que es posible lograrlo mediante la combinación con otras APIs y estructuras. A continuación, se presentan algunas soluciones creadas por la comunidad de Java:

Enlaces relevantes

[14] [Journal of Universal Computer Science: Asynchronous Calls in RMI.](#)

[15] [Journal of Object Technology: Asynchronous RMI in Java.](#)

[16] [Barak Bar-Orion : Asynchronous Calls.](#)

[17] [GitHub: barakb/asynrmi.](#)

[18] [Stack Overflow: Java Asynchronous RMI.](#)

7. CONCLUSIONES

En general, la mayoría de los objetivos establecidos al inicio del documento se han cumplido satisfactoriamente.

- La recopilación y estructuración de la información se ha realizado de manera clara y ordenada, siguiendo una secuencia lógica de conceptos.
- Los ejercicios diseñados son prácticos y útiles, ya que refuerzan la comprensión de RMI por parte de los estudiantes al aplicarlos en casos de uso relevantes.
- La documentación elaborada puede resultar útil para usuarios futuros que deseen aprender y utilizar RMI.
- Aunque se ha logrado proporcionar una base sólida en RMI, no se ha conseguido fomentar la participación activa, ni fomentar la investigación y análisis de los estudiantes.

8. LÍNEAS FUTURAS

Aunque se hayan logrado muchos de los objetivos iniciales con este proyecto, algunos de los puntos a mejorar que se han podido observar han sido los siguientes:

- Temas que les falta profundidad:
 - Ajustes y opciones de RMI: Explorar en mayor detalle las configuraciones y opciones disponibles en RMI para optimizar su uso.
 - Seguridad: En esta parte se podía haber metido el temario de Java Security y los permisos de archivo, enriqueciendo en gran medida el trabajo. Por desgracia no se ha podido implementar por falta de tiempo, y debido a que se alejaba del tema de RMI, pese ser un complemento muy adecuado al tema.
 - Carga dinámica: Había un apartado entero de descarga dinámica de stubs qué combinado con la parte de codebase, se podría haber utilizado para descargar archivos de todo tipo entre ordenadores, además de poder hacer ejecución remota de código. Estos no se han implementado por su alta dificultad.
- Hubiera estado bien, hacer una pequeña practica con herramientas de enumeración para ver los métodos de los programas desde fuera. Algunos enlaces serían estos:
 - [19] [GitHub - NickstaDB/BaRMiE BaRMiE](#).
 - [20] [EHC Group Blog: Herramienta de enumeración y ataque de Java RMI - Delitos informáticos..](#)

9. BIBLIOGRAFIA

- 1 Oracle Documentation (sin fecha de publicación): Java RMI Overview. Última comprobación: 30 de mayo de 2023. URL: <https://docs.oracle.com/javase/tutorial/rmi/overview.html>
- 2 Stanford University - CHAIMS Documentation (sin fecha de publicación): RMI Description. Última comprobación: 30 de mayo de 2023. URL: http://infolab.stanford.edu/CHAIMS/Doc/Details/Protocols/rmi/rmi_description.html
- 3 Oracle Documentation (sin fecha de publicación): rmic - The RMI Compiler. Última comprobación: 30 de mayo de 2023. URL: <https://docs.oracle.com/javase/10/tools/rmic.htm#JSWOR705>
- 4 JavaTpoint (sin fecha de publicación): RMI. Última comprobación: 30 de mayo de 2023. URL: <https://www.javatpoint.com/RMI>
- 5 Universidad de Alicante - Departamento de Lenguajes y Sistemas Informáticos (sin fecha de publicación): Tema 2: Introducción a RMI. Última comprobación: 30 de mayo de 2023. URL: <http://www.jtech.ua.es/j2ee/2002-2003/modulos/rmi/apuntes/tema2.htm>
- 6 Java Distributed computing (sin fecha de publicación)- Tema 3.6: Java Rmi. Última comprobación: 30 de mayo de 2023 . URL: https://docstore.mik.ua/orelly/java-ent/dist/ch03_06.htm
- 7 Universitat Politècnica de Catalunya - FIB (sin fecha de publicación): Laboratori de Llenguatges. Última comprobación: 30 de mayo de 2023. URL: https://studies.ac.upc.edu/FIB/PXC/manel/LAB/tut_3.html#serial
- 8 Oracle Documentation (sin fecha de publicación): RMI Security Recommendations. Última comprobación: 30 de mayo de 2023. URL: https://docs.oracle.com/javase/8/docs/technotes/guides/rmi/rmi_security_recommendations.html
- 9 JTech (sin fecha de publicación): Apuntes de la sesión 11. Última comprobación: 30 de mayo de 2023. URL: <http://www.jtech.ua.es/historico/paj/restringido/apuntes/sesion11-apuntes.htm>

10 Studies UPC (sin fecha de publicación): Laboratorio - Tutorial 3: Seguridad. Última comprobación: 30 de mayo de 2023. URL:

https://studies.ac.upc.edu/FIB/PXC/manel/LAB/tut_3.html#secure

11 JTech (2005-2006): Apuntes de la sesión 01. Última comprobación: 30 de mayo de 2023.

URL: <http://www.jtech.ua.es/j2ee/2005-2006/modulos/rmi/sesion01-apuntes.html>

12 Oracle Docs (sin fecha de publicación): Codebase. Última comprobación: 30 de mayo de 2023. URL: <https://docs.oracle.com/javase/8/docs/technotes/guides/rmi/codebase.html>

13 Oracle Docs (sin fecha de publicación): Overview. Última comprobación: 30 de mayo de 2023. URL: <https://docs.oracle.com/javase/tutorial/rmi/overview.html>

14 Journal of Universal Computer Science (2008): Asynchronous Calls in RMI. Última comprobación: 30 de mayo de 2023. URL: <https://lib.jucs.org/article/28080/>

15 Journal of Object Technology (2004): Asynchronous RMI in Java. Última comprobación: 30 de mayo de 2023. URL: https://www.jot.fm/issues/issue_2004_03/column5.pdf

16 Barak Bar-Orion (sin fecha de publicación): Asynchronous Calls. Última comprobación: 30 de mayo de 2023. URL: <http://barakb.github.io/asynchrmi/asynchronous-calls.html>

17 GitHub: barakb/asynchrmi. Última comprobación: 30 de mayo de 2023. URL: <https://github.com/barakb/asynchrmi>

18 Stack Overflow (2013): Java Asynchronous RMI. Última comprobación: 30 de mayo de 2023. URL: <https://stackoverflow.com/questions/15942654/java-asynchronous-rmi>

19 GitHub - NickstaDB/BaRMiE. (Sin fecha de publicación). BaRMiE. Última comprobación: 30 de mayo de 2023. URL: <https://github.com/NickstaDB/BaRMiE>.

20 EHC Group Blog. (11 de abril de 2018). Herramienta de enumeración y ataque de Java RMI - Delitos informáticos. Última comprobación: 30 de mayo de 2023. URL: <https://blog.ehcgroup.io/2018/04/11/21/59/33/2916/herramienta-de-enumeracion-y-ataque-de-java-rmi-barmie/delitos-informaticos/dpab>

21 Linea de Código (s.f.). Leer una URL con Java. Recuperado el 4 de junio de 2023, de <https://lineadecodigo.com/java/leer-una-url-con-java/>