EJERCICIOS: STUB SKELETON

001- Hello world con naming (módulo antiguo)

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

002 - Compilación y ejecución (módulo antiguo)

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

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

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.

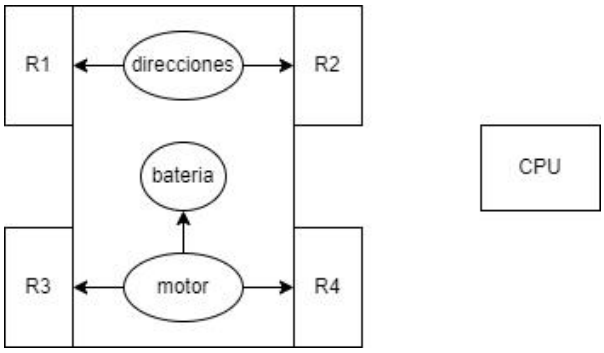
Además, asegúrese de utilizar la misma clave para que los extremos de la conexión puedan comunicarse correctamente.

004-Calculadora

Partiendo del ejercicio anterior, se quiere desarrollaron 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
5. Terminar el programa.

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

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.

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.

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..

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.

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.

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:

$$\text{Total} = \text{Inicial}(1 + (\text{Interés}/\text{NCapiAño}))^{\text{numCapiAño}*\text{años}}$$

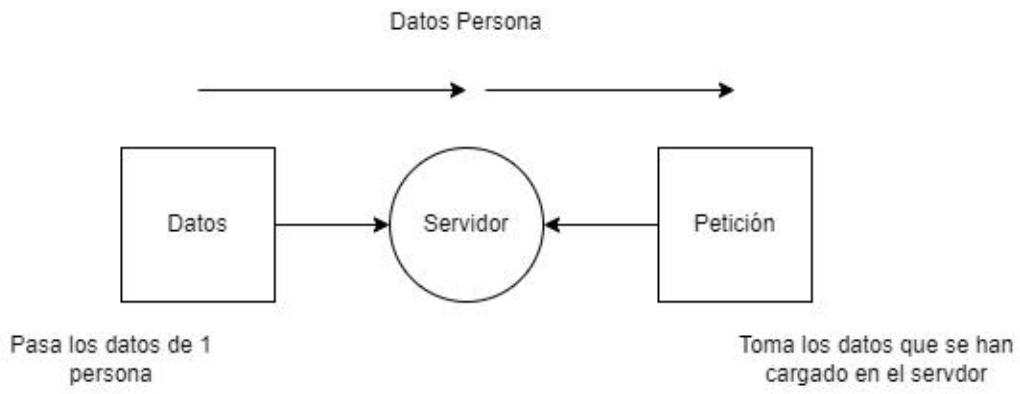
El total a devolver de un préstamo:

$$\text{Total} = \text{prestamo} + (\text{tasaInterés} * \text{plazoAños})$$

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).

013- 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.

EJERCICIOS COMPLETOS

014-SimulaciónRepartidores

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.

0015 SimulaciónBanco

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.