



Escuela Politécnica Superior

**Diego González Sanz**

Asignatura: Programación de sistemas distribuidos  
Profesor: Jefferson Bravo Montes

## Práctica 2

### Aplicación usando CORBA

<u>Introducción</u>	<u>3</u>
<u>Objetivos</u>	<u>3</u>
<u>Ejercicio 1</u>	<u>4</u>
<u>Ejercicio 2</u>	<u>11</u>
<u>Ejercicio 3</u>	<u>13</u>
<u>Conclusión</u>	<u>34</u>
<u>Bibliografía</u>	<u>34</u>

## 1. Introducción

---

CORBA es un estándar desarrollado por el grupo OMG que permite la interacción entre componentes de software escritos en diferentes lenguajes y que se ejecutan en diferentes sistemas. Esto se logra mediante la distribución de objetos a través de redes, lo que permite llamar y ejecutar operaciones en objetos de forma remota. CORBA no está limitado a ningún lenguaje de programación específico y utiliza una sintaxis llamada IDL para describir objetos. En resumen, CORBA facilita la integración de sistemas heterogéneos y fomenta la reutilización de componentes de software en diferentes entornos.

CORBA consta de cuatro componentes principales:

- Object Request Broker (ORB)
- El servidor CORBA
- El servicio de nombres
- El nodo CORBARequest.

El ORB se encarga de la comunicación entre los componentes de CORBA y de la ordenación y desordenación de los parámetros para que el manejo de parámetros sea transparente para las aplicaciones cliente y servidor. El servidor CORBA crea objetos CORBA e inicializa con un ORB, y las referencias a estos objetos se colocan en el servicio de nombres para que los clientes puedan acceder a ellos. El nodo CORBARequest actúa como un cliente CORBA que solicita operaciones en los objetos. En conjunto, estos componentes permiten la interacción y distribución de objetos a través de redes heterogéneas.

Además, en esta práctica utilizamos el IDE de Visual Studio Conde. El lenguaje de programación es Java y esta práctica se ha realizado en Linux Ubuntu.

Para utilizar el JDK de Java en Linux (Ubuntu 22.04.1 LTS) debemos ejecutar este comando en la terminal “sudo apt install openjdk-8-jdk-headless”. CORBA solo se puede instalar en Java 8, por eso mismo utilizamos ese comando

## 2. Objetivos

---

El objetivo de esta práctica es la implementación de CORBA con el lenguaje CORBA y que el alumno busque en Internet el funcionamiento de la misma, cómo se debe programar y el conocimiento de por qué se hace uso.

### 3. Ejercicio 1

---

Enunciado:

Vamos a hacer un Hola Mundo en CORBA (1 punto):

La aplicación contendrá un archivo IDL, un archivo servidor y uno de cliente. Todas las instrucciones de la aplicación deben estar comentadas en castellano, con nuestras palabras para argumentar que se entiende.

Compilaremos primero el IDL, luego el servidor y luego el cliente usando los siguientes códigos respectivamente:

```
$ idlj -fall count.idl  
$ javac Server.java  
$ javac Client.java
```

Para ejecutar el programa necesitamos tener abiertas tres ventanas del Símbolo del sistema. La primera iniciará el puerto, la segunda ejecutará el servidor y la tercera el cliente. El código para ejecutarla es, respectivamente:

```
$ tnameserv -ORBInitialPort 2000  
$ java Server -ORBInitialHost localhost -ORBInitialPort 2000  
$ java Client -ORBInitialHost localhost -ORBInitialPort 2000
```

Resolución del ejercicio:

Primero creamos un archivo con formato IDL que lo llamamos “count” y lo ejecutamos en la terminal con el siguiente comando “\$ idlj -fall count.idl”. El motivo de hacer esto es porque el primer paso para crear una aplicación CORBA es especificar todos los objetos e interfaces utilizando IDL de OMG (Object Management Group).

```

EXPLORADOR ... dienendo count.idl
EJERCICIO-1-PRACTICA-2-SL... > HelloApp
E count.idl
1 module HelloApp
2 {
3     interface Hello
4     {
5         string sayHello();
6         oneway void shutdown();
7     };
8 }

PROBLEMAS SALIDA CONSOLA DE DÉPURAÇÓN TERMINAL
diego@diego-VivoBook-ASUSLaptop-X415DA:~/Sistemas Distribuidos/Ejercicio-1-Practica-2-Sistemas-Distribuidos$ idlj -fall count.idl
diego@diego-VivoBook-ASUSLaptop-X415DA:~/Sistemas Distribuidos/Ejercicio-1-Practica-2-Sistemas-Distribuidos$ 

ESQUEMA LÍNEA DE TIEMPO

```

*Ejecución idlj -fall count.idl*

En el fichero “count.idl” primero estamos creando un módulo “HelloApp” (un módulo en IDL es similar al package en Java), y definimos una interfaz “Hello”.

La interfaz define una operación “sayHello()” que lo que devuelve es una cadena y un método “shutdown()” que cierra el ORB. Esta llamada lo que consigue es que el ORB detenga el procesamiento de peticiones y se prepare para la “destrucción”, es decir, que no haya más peticiones.

### Implementamos el servidor (HelloServer.java):

En el servidor consta de dos clases, la primera es “HelloImpl” que es el sirviente y la clase servidor “HelloServer”.

El sirviente “HelloImpl” es una implementación de la interfaz “count.idl” y cada instancia de count se implementa mediante una instancia de “HelloImpl”. El sirviente contiene un método para cada operación IDL, en nuestro caso “sayHello()” y “shutdown()”.

La clase servidor “HelloServer” tiene el método main() que primero crea e inicializa la instancia de ORB, después se obtiene una referencia al POA raíz y después activa el POAManager. A continuación, creamos una instancia de sirviente con la clase “HelloImpl” e informamos al ORB cogiendo el objeto de “HelloImpl”, llamamos a la función “setORB” y le pasamos el objeto que nos habíamos creado al inicializar el ORB que se llama “orb”. Obtenemos una referencia de objeto CORBA para un contexto de nomenclatura en el que registrar el nuevo objeto CORBA. Obtenemos el

contexto de nomenclatura raíz y registramos el nuevo objeto en el contexto de nombre con el nombre de “Hello” y ahora espera las invocaciones del nuevo objeto cliente.

```
intellij J Server.java x J Server.class J Client.java
Server.java > t Server > main(String[])
import HelloImpl;
import org.omg.CORBA.*;
import org.omg.CORBA.NamingContextPackage.*;
import org.omg.CORBA.POA;
import org.omg.PortableServer.POAManager;
import java.util.Properties;
```

```
class HelloImpl extends HelloPOA {
    private ORB orb;
    public void setORB(ORB orb_val) {
        orb = orb_val;
    }
    // implement sayHello() method
    public String sayHello() {
        return "Hello world !\n";
    }
    // implement shutdown() method
    public void shutdown() {
        orb.shutdown(false);
    }
}
```

Dentro de HelloServer crear clase HelloImpl

```
# countid J Server.java x J Server.class J Client.java
J Server.java > t Server > main(String[])
28
29     public class Server {
30
31         public static void main(String args[]) {
32             Run(Debug
33             try{
34                 // Creamos e inicializamos los ORB
35                 ORB orb = ORB.init(args, null);
36
37                 // Cogemos la referencia de RootPOA y activamos el POAManager
38                 POA rootpoa = POAManagerHelper.narrow(orb.resolve_initial_references("RootPOA"));
39                 rootpoa.the_POAManager().activate();
40
41                 // Creamos una instancia de sirviente con la clase HelloImpl
42                 HelloImpl helloImpl = new HelloImpl();
43                 helloImpl.setORB(orb);
44
45                 // Obtenemos una referencia de objeto CORBA para un contexto de nomenclatura raíz
46                 org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
47                 Hello href = HelloHelper.narrow(ref);
48
49
50                 // Cogemos la raíz Naming context y NameService invoca del nombre NameService
51                 org.omg.CORBA.Object objRef =
52                     orb.resolve_initial_references("NameService");
53
54                 NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
55
56                 // Registraremos el nuevo objeto en el contexto name con el nombre de 'Hello'
57                 // Obtenemos la referencia de objeto en la denominación
58                 String name = "Hello";
59                 NameComponent path[] = ncRef.to_name(name);
60                 ncRef.rebind(path, href);
61
62                 System.out.println("HelloServer ready and waiting ...");
63
64                 // Esperamos a la invocación de los clientes
65                 orb.run();
66             }
67         }
68
69         catch (Exception e) {
70             System.err.println("ERROR: " + e);
71             e.printStackTrace(System.out);
72         }
73
74         System.out.println("HelloServer Exiting ...");
75     }
76
77 }
```

Main de HelloServer

### Implementamos el cliente (HelloClient.java):

Primero creamos e inicializamos un ORB, después obtenemos una referencia al contexto de nomenclatura de raíz, buscamos “Hello” en el contexto de nombres y recibe una referencia ese objeto CORBA. Y por último invoca las operaciones “sayHello()” y “shutdown()” que son las que nos hemos creado del objeto e imprime el resultado.

```

count.idl  J Server.java  J Serverclass  J Client.java  x
J Client.java > Client > main(String[])
1 import HelloApp.*;
2 import org.omg.CosNaming.*;
3 import org.omg.CosNaming.NamingContextPackage.*;
4 import org.omg.CORBA.*;
5
6 public class Client
7 {
8     static Hello helloImpl;
9
10    RunDebug
11    public static void main(String args[])
12    {
13        try{
14            // Creamos e inicializamos el ORB
15            ORB orb = ORB.init(args, null);
16
17            // Cogemos la raiz Naming context y NameService invoca del nombre NameService
18            org.omg.CORBA.Object objRef =
19            orb.resolve_initial_references("NameService");
20            NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
21
22            // Resolvemos la referencia de objeto en la denominacion
23            String name = "Hello";
24            helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));
25
26            // Mostramos por terminal la siguiente informacion
27            System.out.println("Obtained a handle on server object: " + helloImpl);
28            System.out.println(helloImpl.sayHello());
29
30            // Se cierra el ORB
31            helloImpl.shutdown();
32
33        } catch (Exception e) {
34            System.out.println("ERROR : " + e);
35            e.printStackTrace(System.out);
36        }
37    }
38 }

```

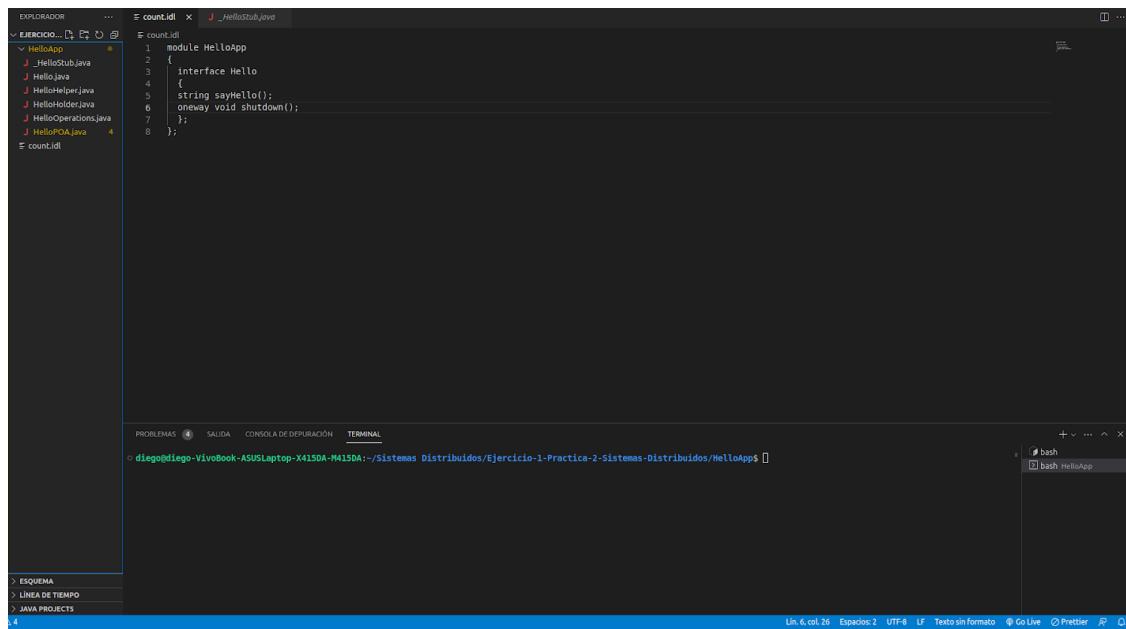
Main de HelloClient

### La explicación de cómo debemos ejecutar el programa e información adicional importante para tener en cuenta:

Para ejecutar este programa debe iniciarse desde un puerto mayor o igual a 1024 porque estamos usando el software de Solaris, que tiene reservados los primeros 1024 puertos.

Debemos utilizar la opción -ORBInitialPort para anular el puerto predeterminado.

Cuando hemos ejecutado el siguiente comando que lo hemos compilado antes "\$ idlj -fall count.idl" nos saldrá la siguiente carpeta "HelloApp" con varios ficheros. Eso nos crea stubs y skeletons que constituyen proxies de CORBA para conseguir una transparencia en la invocación de métodos remotos. El stub (o proxy del cliente) está desplegado en el proceso del cliente y es un objeto local del cliente. El skeleton (o proxy del servidor) es un objeto local implementado en el objeto servidor, y desplegado en el proceso servidor. Recibe las peticiones enviadas desde el stub e invoca el método correspondiente en el objeto servidor.



Fichero count.idl

Utilizamos la opción -fall con el compilador idl porque se genera enlaces del lado del cliente y del lado del servidor. Además, el compilador idl genera una serie de archivos que se crean mediante las opciones seleccionadas cuando compilamos el archivo IDL. Todos esos archivos que se generan de manera automática tienen una funcionalidad estándar. En nuestro caso los archivos que se han generado al ejecutar “count.idl” son los siguientes:

- HelloPOA.java

Es una clase abstracta y es el esqueleto del servidor basado en secuencias y proporciona la funcionalidad básica de CORBA para el servidor. Extiende org.omg.PortableServer.Servant e implementa la interfaz InvokeHandler y la interfaz HelloOperations. La clase de servidor HelloImpl amplía HelloPOA.

- HelloStub.java

Esta clase es el stub del cliente que proporciona la funcionalidad CORBA para el cliente. Extiende org.omg.CORBA.portable.ObjectImpl e implementa la interfaz Hello.java.

- Hello.java

Esta interfaz contiene la versión Java de nuestra interfaz IDL. La interfaz Hello.java amplía org.omg.CORBA.Object y proporciona la funcionalidad de objeto CORBA estándar. También amplía la interfaz HelloOperations y org.omg.CORBA.portable.IDLEntity.

- HelloHelper.java

Esta clase proporciona una funcionalidad auxiliar, que se llama “narrow()” y convierte las referencias de objetos CORBA a sus tipos adecuados. También la clase “HelloHelper” es responsable de leer y escribir el tipo de datos en flujos de CORBA y también insertar y extraer el tipo de dato de Any. También la clase “HelloHolder” delega en los métodos de la clase “HelloHelper” la escritura (“write”) y lectura (“read”).

- HelloHolder.java

Es una clase que la definimos como final y contiene una instancia pública de la clase “Hello”. Si el tipo de IDL es un parámetro de out o inout se utiliza siempre la clase “HelloHolder”. Es una clase que proporciona operaciones para los argumentos “org.omg.CORBA.portable.InputStream” y “org.omg.CORBA.portable.OutputStream” que no se asignan fácilmente a la semántica de Java pero CORBA lo permite. También la clase “HelloHolder” delega en los métodos de la clase “HelloHelper” la escritura (“write”) y lectura (“read”).

- HelloOperations.java

Esta interfaz contiene los métodos “sayHello()” y “shutdown()”. La asignación de IDL a Java coloca todas las operaciones definidas en la interfaz de IDL en este archivo, que comparten tanto los stubs como los skeletons.

Ahora tenemos que compilar todos los archivos .java, incluidos los stubs y skeletons (los que están en la carpeta de “HelloApp”). Para esto ejecutamos en la terminal el siguiente comando “\$ javac \*.java HelloApp/\*.java”.

```

EXPLORADOR ... E count.idl J HelloHolder.java J Server.java x J Client.java 1
EJERCICIO-1-PRACTICA-2-2...
HelloApp
HelloImpl
HelloOperations
HelloPOA
HelloStub
Hello
HelloHelper
HelloHolder
HelloOperations
HelloImpl
Hello
Client
Client.java
count.idl
HelloImpl
Server
Server.java

1 import HelloApp.*;
2 import org.omg.CosNaming.*;
3 import org.omg.CosCommon.NamingContextPackage.*;
4 import org.omg.CORBA.*;
5 import org.omg.PortableServer.*;
6 import org.omg.PortableServer.POA;
7
8 import java.util.Properties;
9
10 class HelloImpl extends HelloPOA {
11     private ORB orb;
12
13     public void setORB(ORB orb_val) {
14         orb = orb_val;
15     }
16
17     // implement sayHello() method
18     public String sayHello() {
19         return "Hello world !!!";
20     }
21
22     // implement shutdown() method
23     public void shutdown() {
24         orb.shutdown(false);
25     }
26
27
28     public class Server {
29
30         Run(Debug
31         public static void main(String args[]) {
32
33             // ...
34
35             System.out.println("HelloWorldServer started");
36
37             ORB orb = ORB.init(args, null);
38             HelloPOA poa = (HelloPOA) orb.resolve_initial_references("HelloImpl");
39             HelloImpl helloImpl = new HelloImpl();
40             poa.registerObject(helloImpl, "HelloImpl");
41             orb.bind("HelloImpl", helloImpl);
42             orb.run();
43
44         }
45     }
46
47     public static void main(String args[]) {
48
49         System.out.println("HelloWorldClient started");
50
51         ORB orb = ORB.init(args, null);
52         HelloPOA poa = (HelloPOA) orb.resolve_initial_references("HelloImpl");
53         HelloOperations helloOperations = (HelloOperations) poa
54             .findObject("HelloImpl");
55         helloOperations.sayHello();
56
57         System.out.println("HelloWorldClient finished");
58
59     }
60
61 }

```

Ejecución javac \*.java HelloApp/\*.java

**Cuando tengamos todos los ficheros compilados tendremos que ejecutar 3 terminales.**

La primera terminal iniciará el puerto y lo haremos con el siguiente comando “\$ tnameserv -ORBInitialPort 2000”

```

diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-1-Practica-2-Sistemas-Distribuidos$ tnameserv -ORBInitialPort 2000
Contexto de implementación para: HelloImpl
TransientNameServer: definiendo puerto para referencias a objeto iniciales en: 2000
Listo.

```

Terminal Iniciar Puerto



En el servidor sucede esto:

```

PROBLEMAS SALIDA CONSOLA DE DEPURACION TERMINAL
diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-1-Practica-2-Sistemas-Distribuidos$ java Server -ORBInitialHost localhost -ORBInitialPort 2000
HelloServer ready and waiting ...
HelloServer Exiting
feb 25, 2023 8:31:01 PM com.sun.corba.se.impl.orb.ORBImpl checkShutdownState
ADVERTENCIA: CORBA.BAD_INV_ORDER: BAD_INV_ORDER orb has shutdown!
org.omg.CORBA.BAD_INV_ORDER:
        cause: OMG minor code: 4 completed: No
        at com.sun.corba.se.impl.logging.OMGSystemException.badOperationAfterShutdown(OMGSystemException.java:244)
        at com.sun.corba.se.impl.logging.OMGSystemException.badOperationAfterShutdown(OMGSystemException.java:246)
        at com.sun.corba.se.impl.orb.ORBImpl.checkShutdownState(ORBImpl.java:1341)
        at com.sun.corba.se.impl.orb.ORBImpl.peekInvocationInfo(ORBImpl.java:1539)
        at com.sun.corba.se.impl.protocol.CorbaMessageMediatorImpl.runServantPostInvoke(CorbaMessageMediatorImpl.java:228)
        at com.sun.corba.se.impl.protocol.CorbaMessageMediatorImpl.servantPostInvoke(CorbaMessageMediatorImpl.java:2189)
        at com.sun.corba.se.impl.protocol.CorbaMessageMediatorImpl.createResponseHelper(CorbaMessageMediatorImpl.java:2165)
        at com.sun.corba.se.impl.protocol.CorbaMessageMediatorImpl.createResponse(CorbaMessageMediatorImpl.java:2013)
        at com.sun.corba.se.impl.protocol.CorbaMessageMediatorImpl.createReply(CorbaMessageMediatorImpl.java:623)
        at HelloApp.HelloPOA._invoke(HelloPOA.java:47)
        at com.sun.corba.se.impl.protocol.CorbaserverRequestDispatcherImpl.dispatchToServant(CorbaserverRequestDispatcherImpl.java:654)
        at com.sun.corba.se.impl.protocol.CorbaserverRequestDispatcherImpl.dispatch(CorbaserverRequestDispatcherImpl.java:205)
        at com.sun.corba.se.impl.protocol.CorbaMessageMediatorImpl.handleRequest(CorbaMessageMediatorImpl.java:1700)
        at com.sun.corba.se.impl.protocol.CorbaMessageMediatorImpl.handleRequest(CorbaMessageMediatorImpl.java:1558)
        at com.sun.corba.se.impl.protocol.CorbaMessageMediatorImpl.handleInput(CorbaMessageMediatorImpl.java:940)
        at com.sun.corba.se.impl.protocol.giopheaders.RequestMessage_1_2.callback(RequestMessage_1_2.java:198)
        at com.sun.corba.se.impl.protocol.CorbaMessageMediatorImpl.handleRequest(CorbaMessageMediatorImpl.java:712)
        at com.sun.corba.se.impl.transport.SocketOrChannelConnectionImpl.dispatch(SocketOrChannelConnectionImpl.java:474)
        at com.sun.corba.se.impl.transport.SocketOrChannelConnectionImpl.dowork(SocketOrChannelConnectionImpl.java:1237)
        at com.sun.corba.se.impl.orbutil.threadpool.ThreadPoolImpl$WorkerThread.performWork(ThreadPoolImpl.java:490)
        at com.sun.corba.se.impl.orbutil.threadpool.ThreadPoolImpl$WorkerThread.run(ThreadPoolImpl.java:519)

diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-1-Practica-2-Sistemas-Distribuidos$ 

```

Terminal Servidor llamado del cliente

Como estaba esperando en la linea de “orb.run()”, cuando un cliente le ha llamado desde el puerto se ejecuta para que el servidor pueda enviar y mostrar datos.

## 4. Ejercicio 2

Preguntas sobre Hola Mundo en CORBA (puedes añadir capturas):

**¿Qué sucede si lanzo antes el cliente que el servidor?:**

Nos salen bastantes errores si no iniciamos el puerto y nos está diciendo que hay un fallo de conexión en donde está en la terminal el texto ADVERTENCIA, así que primero se debe obligatoriamente iniciar el puerto.

```

• diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-1-Practica-2-Sistemas-Distribuidos$ java Client -ORBInitialHost localhost -ORBInitialPort 2000
Feb 25, 2023 8:54:30 PM com.sun.corba.se.impl.transport.SocketOrChannelConnectionImpl <init>
ADVERTENCIA: "IOP00419201: (COMM_FAILURE) Connection failure: socketType: IIOP_CLEAR_TEXT; hostname: localhost; port: 2000"
org.omg.CORBA.COMM_FAILURE: vmcid: SUN minor code: 201 completed: No
    at com.sun.corba.se.impl.logging.ORBUtilSystemException.connectFailure(ORBUtilSystemException.java:2200)
    at com.sun.corba.se.impl.logging.ORBUtilSystemException.connectFailure(ORBUtilSystemException.java:2221)
    at com.sun.corba.se.impl.transport.SocketOrChannelConnectionImpl.<init>(SocketOrChannelConnectionImpl.java:223)
    at com.sun.corba.se.impl.transport.SocketOrChannelContactInfoImpl.createConnection(SocketOrChannelContactInfoImpl.java:119)
    at com.sun.corba.se.impl.protocol.CorbaClientRequestDispatcherImpl.beginRequest(CorbaClientRequestDispatcherImpl.java:187)
    at com.sun.corba.se.impl.protocol.CorbaClientDelegateImpl.request(CorbaClientDelegateImpl.java:137)
    at com.sun.corba.se.impl.logging.BootstrapResolverImpl.invoke(BootstrapResolverImpl.java:99)
    at com.sun.corba.se.impl.resolver.BootstrapResolverImpl.resolve(BootstrapResolverImpl.java:132)
    at com.sun.corba.se.impl.resolver.BootstrapResolverImpl.resolve(BootstrapResolverImpl.java:127)
    at com.sun.corba.se.impl.resolver.CompositeResolverImpl.resolve(CompositeResolverImpl.java:47)
    at com.sun.corba.se.impl.resolver.CompositeResolverImpl.resolve(CompositeResolverImpl.java:47)
    at Client.main(Client.java:18)

Caused by: java.net.ConnectException: Conexión rehusada
    at sun.nio.ch.Net.connect0(Net.java:482)
    at sun.nio.ch.Net.connect(Net.java:474)
    at sun.nio.ch.SocketChannelImpl.connect(SocketChannelImpl.java:647)
    at java.nio.channels.SocketChannel.open(SocketChannel.java:189)
    at com.sun.corba.se.impl.transport.DefaultSocketFactoryImpl.createSocket(DefaultSocketFactoryImpl.java:95)
    at com.sun.corba.se.impl.transport.SocketOrChannelConnectionImpl.<init>(SocketOrChannelConnectionImpl.java:267)
    ... 11 more

ERROR : org.omg.CORBA.COMM_FAILURE: vmcid: SUN minor code: 201 completed: No
org.omg.CORBA.COMM_FAILURE: vmcid: SUN minor code: 201 completed: No
    at com.sun.corba.se.impl.logging.ORBUtilSystemException.connectFailure(ORBUtilSystemException.java:2200)
    at com.sun.corba.se.impl.logging.ORBUtilSystemException.connectFailure(ORBUtilSystemException.java:2221)
    at com.sun.corba.se.impl.transport.SocketOrChannelConnectionImpl.<init>(SocketOrChannelConnectionImpl.java:223)
    at com.sun.corba.se.impl.transport.SocketOrChannelContactInfoImpl.createConnection(SocketOrChannelContactInfoImpl.java:119)
    at com.sun.corba.se.impl.protocol.CorbaClientRequestDispatcherImpl.beginRequest(CorbaClientRequestDispatcherImpl.java:187)

```

Terminal Servidor

Teniendo claro el anterior error, si lo ejecutamos teniendo el puerto abierto 2000, sucede que lanza un error indicando que no encuentra el Naming Context que es el id del servicio que estamos accediendo, en nuestro caso es “Hello” y como en el servidor lo tenemos puesto, pero no lo hemos inicializado, nos está indicando que no lo encuentra.

```

• diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-1-Practica-2-Sistemas-Distribuidos$ java Client -ORBInitialHost localhost -ORBInitialPort 2000
ERROR : org.omg.CosNaming.NamingContextPackage.NotFound: IDL:omg.org/CosNaming/NamingContext/NotFound:1.0
org.omg.CosNaming.NamingContextPackage.NotFound: IDL:omg.org/CosNaming/NamingContext/NotFound:1.0
    at org.omg.CosNaming.NamingContextPackage.NotFoundHelper.read(NotFoundHelper.java:72)
    at org.omg.CosNaming.NamingContextExtExtStub.resolve_NamingContextEx(SOD.java:105)
    at Client.main(Client.java:25)

```

Terminal Cliente

### ¿Qué sucedería si lanzase varios servidores a la vez y un solo cliente?:

Si lanzas varios servidores a la vez y sólo tienes un cliente, es probable que el cliente sólo pueda conectarse a uno de los servidores y los demás queden inactivos. La forma en que se establece la conexión entre el cliente y el servidor depende del protocolo utilizado y la configuración de red, pero normalmente el cliente se conecta al primer servidor disponible que responda a su solicitud.

Aunque es posible configurar el cliente para conectarse a varios servidores al mismo tiempo y distribuir la carga de trabajo entre ellos. Sin embargo, esto requeriría una configuración adicional y un protocolo que permita la comunicación simultánea con múltiples servidores.

### ¿Puedes conectarte al servidor de un compañero? ¿Cómo lo harías?:

Conectarse al servidor de un compañero dependerá de varios factores, como la configuración de red, la seguridad, los permisos y la accesibilidad del servidor. Si se tiene permiso y acceso al servidor de

un compañero, se podría intentar conectarse de una de las siguientes formas: a través de una conexión directa utilizando un protocolo como FTP (Protocolo de Transferencia de Ficheros), utilizando un servicio de almacenamiento en la nube como Dropbox o Google Drive y podría compartir la carpeta del servidor conmigo para que pueda acceder a ella y modificar los archivos (añadiendo, eliminando...). Es importante tener los permisos necesarios antes de intentar conectarte a su servidor.

## 5. Ejercicio 3

### Enunciado:

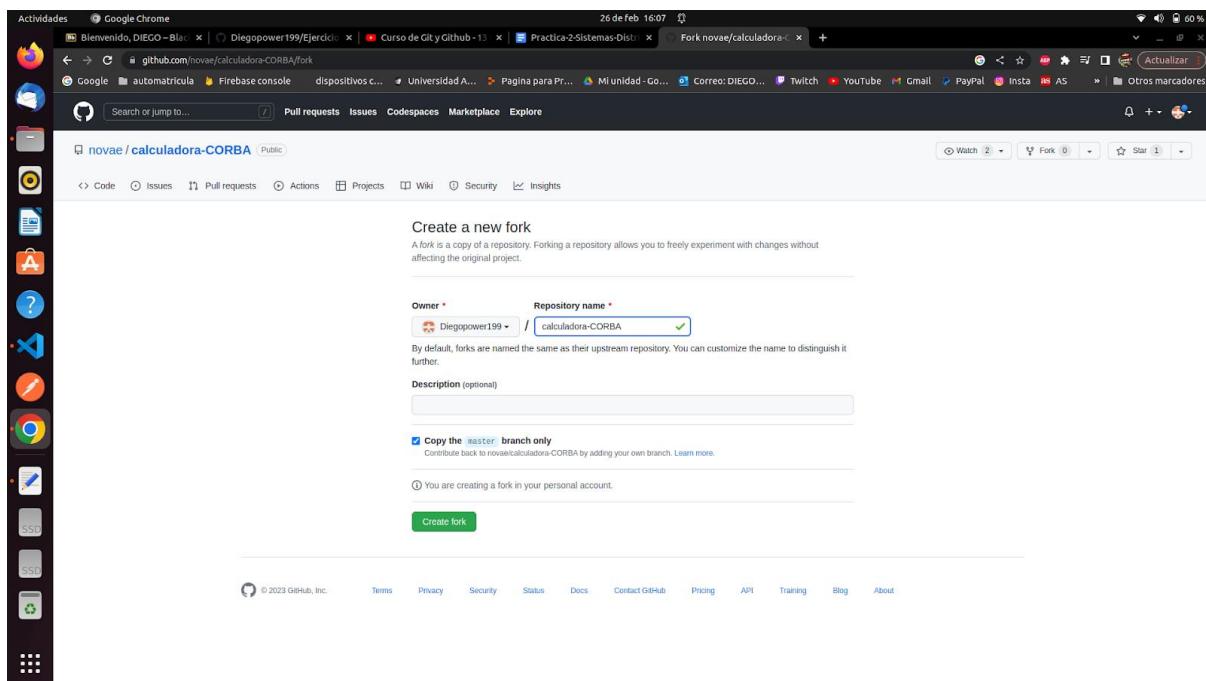
Actualiza un repositorio de GitHub con una aplicación Java CORBA (7 puntos)

Aquí debéis hacer un fork de una aplicación en GitHub y realizar modificaciones en ella. Por ejemplo, imaginemos que tenemos una calculadora que funciona con CORBA y únicamente tiene las funciones de suma, resta, multiplicar y dividir. Podéis añadir, por ejemplo: operar con raíces cuadradas o añadir que utilice decimales. Pautas:

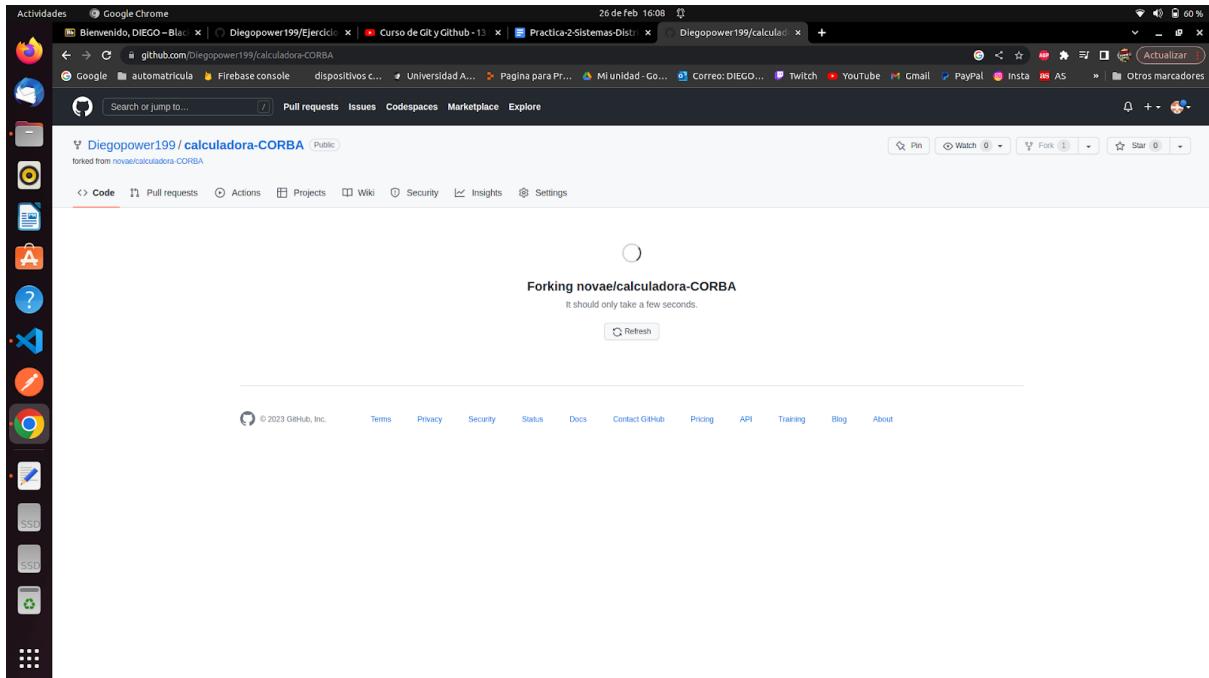
- El código que se añada debe ser por un lado pegado en este documento y, por otro lado, se deben realizar los commits en el repositorio.
- El código debe contener comentarios propios respecto a cómo funciona la aplicación.
- Toda la información que tenga el README.md nunca está demás.
- Intenta que sea una aplicación/funcionalidad diferente la que modificas (3 puntos) No todos los compañeros vamos a tener Calculadoras, busca otras aplicaciones y díferéntiate.

### Resolución del ejercicio:

Primero le damos al botón Fork del proyecto que queremos copiar y nos saldrá lo siguiente:

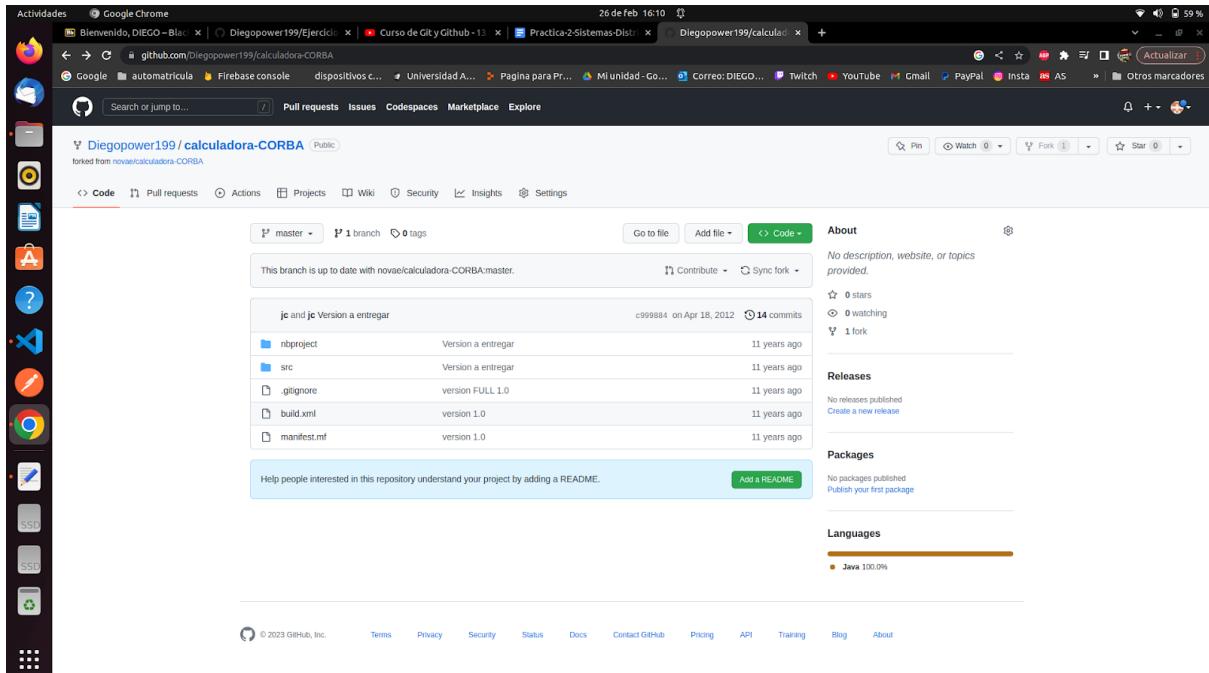


Nos saldrá la siguiente foto, indicando que se está cargando la copia y ya lo tendríamos



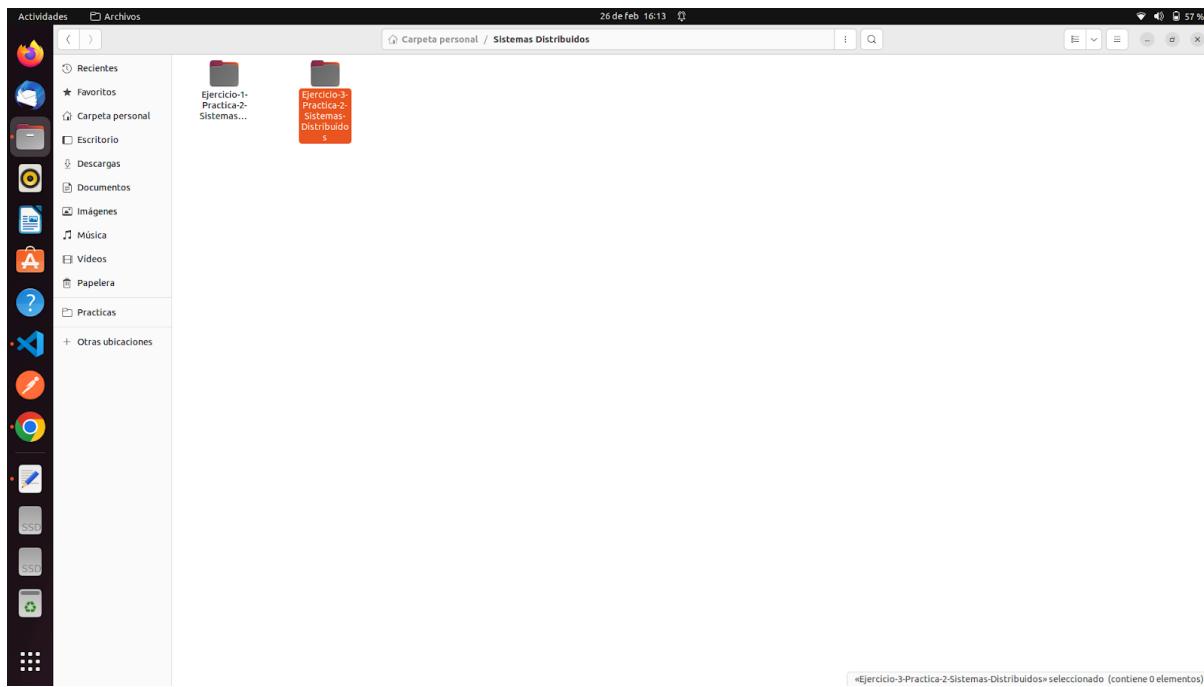
Pantalla Carga fork

Aquí tenemos toda la información y ahora tenemos un fork del repositorio elegido.



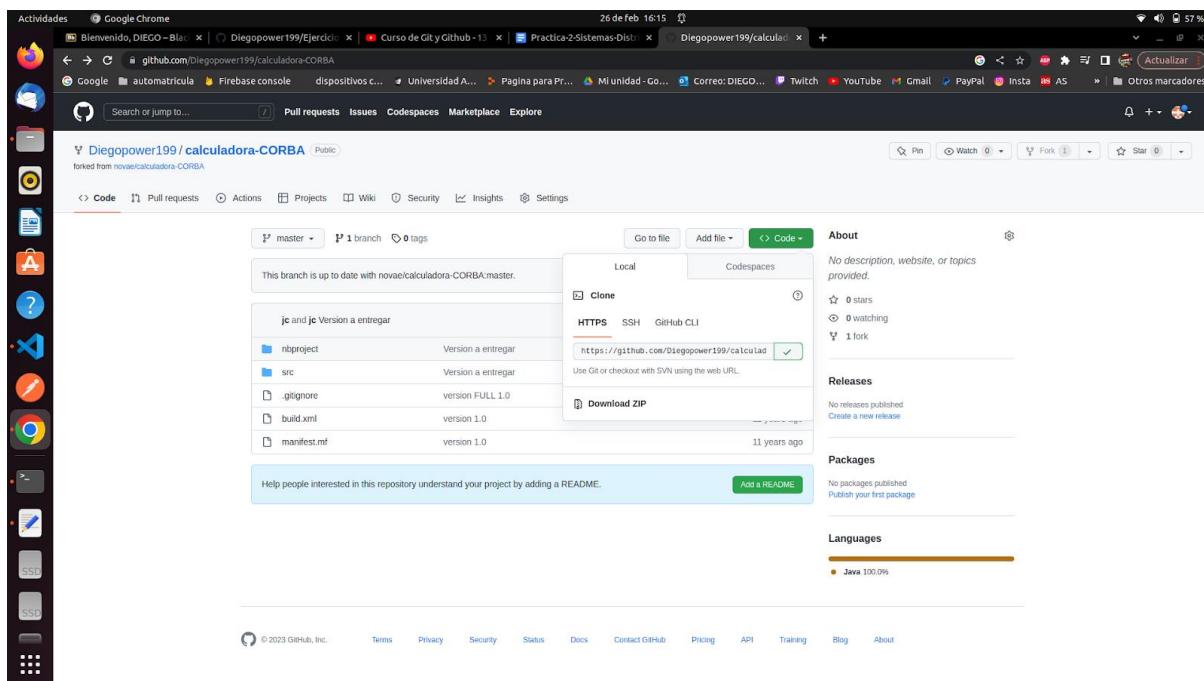
Repositorio clonado GitHub

Como nosotros queremos traer este repositorio que hemos generado con un fork, creamos una carpeta, en nuestro caso la he llamado “Ejercicio-3-Practica-2-Sistemas-Distribuidos”.



*Carpeta creada del proyecto*

Ahora abrimos una terminal dentro de la carpeta seleccionada, y ejecutamos el comando “`$ git clone urlRepositorio`” con una ruta que GitHub nos proporciona cuando damos al repositorio en el botón “<> Code”. Para ello debemos copiar la ruta que nos aparezca ahí.



*Obtener la url del repositorio*

El comando completo es “`$ git clone https://github.com/Diegopower199/calculadora-CORBA.git`”. De esta forma podemos ver que estamos descargando a nuestra carpeta ese repositorio.

```
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL
• diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos$ git clone https://github.com/Diegopower199/calculadora-CORBA.
git
Clonando en 'calculadora-CORBA'...
remote: Enumerating objects: 255, done.
remote: Total 255 (delta 0), reused 0 (delta 0), pack-reused 255
Recibiendo objetos: 100% (255/255), 9.72 MiB | 2.66 MiB/s, listo.
Resolviendo deltas: 100% (111/111), listo.
diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos$
```

Comando clone

Y abrimos otra terminal desde dentro del proyecto clonado.

Cuando lo hagamos, hacemos un “\$ git add -A” para añadir todos los ficheros, después un commit para saber que estamos modificando con el comando “git commit -m ‘que fichero hemos cambiado’” y por último un “\$ git push origin master” que añade todos los cambios a GitHub.

```
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL
• diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA$ git add -A
• diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA$ git commit -m "añadir proyecto a Git Hub"
[master 7d78b40] añadir proyecto a Git Hub
 1 file changed, 1 insertion(+)
   create mode 100644 README.md
• diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA$ git push origin master
Enumerando objetos: 4, listo.
Comprimiendo objetos: 100% (4/4), listo.
Comprimiendo objetos: 100% (2/2), listo.
Escribiendo objetos: 100% (3/3), 301 bytes | 301.00 KiB/s, listo.
Total 3 (delta 1), reusados 0 (delta 0), pack-reusados 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Diegopower199/calculadora-CORBA.git
 c999884..7d78b40 master -> master
diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA$
```

Comandos para subir a Git Hub

Vamos a hacer tres cambios en el programa: el primero es cambiar los valores de los parámetros de int a double, es decir, que sean valores decimales. El segundo es añadir que se puedan hacer operaciones de raíz cuadrada y se va a usar un número como parámetro. Y el tercero es añadir la otra operación que es la potencia que va a tener dos números como parámetro.

En la siguiente foto hemos cambiado en el interface del suma.idl todos los valores “long” por “double” y también hemos añadido que se pueda hacer un cálculo de la raíz cuadrada y un cálculo de potencia.

```

    1 module sumApp;
    2     interface suma{
    3         double sumar(double primerNumero, double segundoNumero);
    4         double restar(double primerNumero, double segundoNumero);
    5         double multiplicar(double primerNumero, double segundoNumero);
    6         double dividir(double primerNumero, double segundoNumero);
    7         double raizcuadrada(double primerNumero);
    8         double potencia(double primerNumero, double segundoNumero);
    9     };
   10 }
   11

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$ idlj -f all suma.idl
diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$ 

```

Abrir Puerto Servidor Cliente Subir GitHub bash src

Compilar suma.idl

Nos saldrán varios errores en la clase “SumaServer.java” y en la “SumaClient.java” ya que son en los únicos ficheros donde no se han actualizado, porque los hemos creado después de compilar el IDL.

En la clase “SumaServer.java” nos está saliendo un error indicando que no existen esas funciones con “double”, por lo que nos vamos a corrección rápida.

```

    1 package calculadora;
    2
    3 import org.omg.CORBA.*;
    4 import org.omg.PortableServer.*;
    5 import org.omg.PortableServer.POA;
    6
    7 class SumaImpl extends sumaPOA{
    8     public void shutdown(){
    9         POA poa = this_POA();
   10         poa.shutdown(false);
   11     }
   12 }
   13
   14
   15
   16
   17
   18
   19
   20
   21
   22
   23
   24
   25
   26
   27
   28
   29
   30
   31
   32
   33
   34

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$ idlj -f all suma.idl
diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$ 

```

Abrir Puerto Servidor Cliente Subir GitHub bash src

Errores SumaServer

Cuando hacemos click ahí nos saldrá si queremos añadir los métodos: seleccionamos la opción Add.



Eliminamos las funciones que ya estaban puestas y nos saldrá esto:

```

Actividades > Visual Studio Code
Archivo Editar Selección Ver Ir Ejecutar Terminal Ayuda
27 feb 15:34
SumaServer.java - calculadora-CORBA - Visual Studio Code

EXPLORADOR ... README.md M sumIDL M SumaServer.java M J sumPOA.java 9, M J SumaClient.java 1, M
CALCULADORA-CORBA src > J SumaServer.java > sumIDL > J SumaImpl.java > dividir(double, double)
> orbdb
> orbdb
> orbdb
> sumIDL
J_sumaStub.c... M 7 import org.omg.PortableServer.POA;
J_sumaStubj... M 8 class SumaImpl extends sumaPOA{
J_suma.class M 9     private ORB orb;
J_suma.java M 10     import org.omgPortableServer.POAOA;
J_sumaHelper... M 11     public void setORB(ORB orb_val){
J_sumaHolder... U 12         orb=orb_val;
J_sumaOperat... M 13     }
J_sumaOperat... M 14 }
J_sumaPOA.c... M 15     public void shutdown(){
J_sumaPOA.... 9,M 16         orb.shutdown(false);
J_sumaPOA.... 9,M 17 }
J_sumaPOA.... 9,M 18
J_sumaPOA.... 9,M 19
J_sumaPOA.... 9,M 20
J_sumaPOA.... 9,M 21
J_sumaPOA.... 9,M 22
J_sumaPOA.... 9,M 23
J_sumaPOA.... 9,M 24
J_sumaPOA.... 9,M 25
J_sumaPOA.... 9,M 26
J_sumaPOA.... 9,M 27
J_sumaPOA.... 9,M 28
J_sumaPOA.... 9,M 29
E sumIDL M 30
J SumaClient.c... M 31
J SumaClient.... 1, M 32
J SumaImpl.class. M 33
J SumaServer.c... M 34
J SumaServer.java M 35
> .gitignore
E manifest.mf M 36
O README.md M 37
Paso1.bat M 38
Paso2.bat M 39
Paso3.bat M 40
sumIDL M 41
sumPOA.java M 42
sumPOA.... 9,M 43
sumPOA.... 9,M 44
}

```

Poner funciones correctas en SumaServer

Por último, debemos implementar la lógica de cada una de las funciones que hemos hecho y devolver el resultado calculado.

A continuación, enseño el código de cada una de las funciones. (Voy a poner una a una, pero lo hago en Word).

```

    package CALCULADORA.CORBA;
    import org.omgPortableServer.POA;
    import org.omgPortableServer.ORB;
    import org.omgCosNaming.NamingContextPackage.*;
    import org.omgCORBA.*;

    public class SumaImpl extends sumaPOA {
        private ORB orb;
        public void setORB(ORB orb_val) {
            orb=orb_val;
        }
        public void shutdown() {
            orb.shutdown(false);
        }
        public double sumar(double primerNumero, double segundoNumero) {
            return primerNumero + segundoNumero;
        }
        public double restar(double primerNumero, double segundoNumero) {
            return primerNumero - segundoNumero;
        }
        public double multiplicar(double primerNumero, double segundoNumero) {
            return primerNumero * segundoNumero;
        }
        public double dividir(double primerNumero, double segundoNumero) {
            return primerNumero / segundoNumero;
        }
        public double raizCuadrada(double primerNumero) {
            return Math.sqrt(primerNumero);
        }
        public double potencia(double primerNumero, double segundoNumero) {
            return Math.pow(primerNumero, segundoNumero);
        }
    }
  
```

Poner el resultado en el return a todas las funciones

Cuando ya tengamos todas las funciones implementadas en clase “SumaServer.java” nos tendremos que ir a la clase “SumaClient.java”.

En la clase “SumaClient.java” nos saldrá esto y el motivo es porque lo que devuelve la función es un double y no un Integer, por lo cual donde pone “Integer” ponemos el objeto “Double” y con este paso hemos terminado. Además, debemos hacer otros dos “System.out.println” de la raíz cuadrada y la potencia.

```

    package CALCULADORA.CORBA;
    import org.omgCosNaming.NamingContextPackage.*;
    import org.omgCORBA.*;

    public class SumaClient{
        static suma sumaImpl;
        public static void main(String args[]){
            try{
                ORB orb=ORB.init(args,null);
                org.omg.CORBA.Object objRef=orb.resolve_initial_references("NameContextExHelper.na");
                NamingContextExt ncRef=(NamingContextExt)objRef;
                String name="Suma";
                sumaImpl=NameHelper.narrow(ncRef.resolve_string(name));
                System.out.println("Resultado suma es: "+ Integer.toString(sumaImpl.sumar(primerNumero: 20, segundoNumero: 30)));
                System.out.println("Resultado resta es: "+ Integer.tostring(sumaImpl.restar(primerNumero: 20, segundoNumero: 30)));
                System.out.println("Resultado multiplicacion es: "+ Integer.toString(sumaImpl.multiplicar(primerNumero: 20, segundoNumero: 30)));
                System.out.println("Resultado division es: "+ Integer.Double(sumaImpl.dividir(primerNumero: 20, segundoNumero: 30)));
                sumaImpl.shutdown();
            }catch(Exception e){
                System.err.println("ERROR: "+e);
                e.printStackTrace(System.out);
            }
        }
    }
  
```

Errores en SumaClient por los tipos

Y ponemos lo siguiente y ya saldría bien

```

    package CALCULADORA_CORBA;
    import org.omg.Costaming.*;
    import org.omg.CosNaming.NamingContextPackage.*;
    import org.omg.CORBA.*;

    public class SumaClient{
        static suma sumimpl;
        Run[Debug] public static void main(String args[]){
            try{
                ORB orb=ORB.init(args,null);
                org.omg.CORBA.Object objRef=orb.resolve_initial_references("NameService");
                NamingContextExt ncRef=NamingContextExHelper.narrow(objRef);
                String name="Suma";
                sumimpl=(suma)ncRef.resolve(name);
                System.out.println("Resultado suma es: " + Double.toString(sumimpl.sumar(primerNumero;20.5, segundoNumero;30.5)));
                System.out.println("Resultado resta es: " + Double.toString(sumimpl.restar(primerNumero;20.5, segundoNumero;30.5)));
                System.out.println("Resultado multiplicacion es: " + Double.toString(sumimpl.multiplicar(primerNumero;20.5, segundoNumero;30.5)));
                System.out.println("Resultado division es: " + Double.toString(sumimpl.dividir(primerNumero;20.5, segundoNumero;30.5)));
                System.out.println("Resultado raiz cuadrada es: " + Double.toString(sumimpl.raizCuadrada(primerNumero;20.5)));
                System.out.println("Resultado potencia es: " + Double.toString(sumimpl.potencia(primerNumero;20.5, segundoNumero;30.5)));
                sumimpl.shutdown();
            }catch(Exception e){
                System.err.println("ERROR: "+e);
                e.printStackTrace(System.out);
            }
        }
    }

```

Cambiar los tipos correctamente

Cuando ya hemos acabado de implementar todo debemos ejecutar que se compilen todos los .java y se hace con el siguiente comando “\$ javac \*.java sumaApp/\*.java”.

```

diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$ idlj -f all suma.idl
diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$ javac *.java sumaApp/*.java
Note: Recompile with -Xlint:unchecked for details.
diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$ 

```

Comando para compilar todos los .java

Antes de ejecutar los programas pondré el código añadido en este documento.

### Archivo suma.idl:

```

module sumaApp{
    interface suma{
        double sumar(in double primerNumero, in double segundoNumero);
        double restar(in double primerNumero, in double segundoNumero);
        double multiplicar(in double primerNumero, in double segundoNumero);
        double dividir(in double primerNumero, in double segundoNumero);
        double raizCuadrada(in double primerNumero);
        double potencia (in double primerNumero, in double segundoNumero);
        oneway void shutdown();
    };
}

```

### Archivo SumaServer:

Dentro de la clase Sumalmpl las siguientes funciones:

```
class Sumalmpl extends sumaPOA{

    public double sumar(double primerNumero, double segundoNumero) {
        return primerNumero + segundoNumero;
    }

    public double restar(double primerNumero, double segundoNumero) {
        return primerNumero - segundoNumero;
    }

    public double multiplicar(double primerNumero, double segundoNumero) {
        return primerNumero * segundoNumero;
    }

    public double dividir(double primerNumero, double segundoNumero) {
        return primerNumero / segundoNumero;
    }

    public double raizCuadrada(double primerNumero) {
        return Math.sqrt(primerNumero);
    }

    public double potencia(double primerNumero, double segundoNumero) {
        return Math.pow(primerNumero, segundoNumero);
    }
}
```

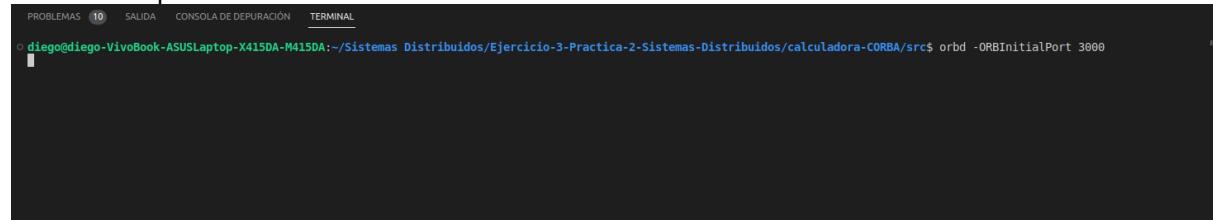
Archivo SumaClient:

Dentro del main y en el try cuando estamos mostrando los resultados (Esto está dentro del main en el try):

```
System.out.println("Resultado suma es: " + Double.toString(sumalmpl.sumar(20.5, 30.5)));
System.out.println("Resultado resta es: " + Double.toString(sumalmpl.restar(20.5, 30.5)));
System.out.println("Resultado multiplicacion es: " + Double.toString(sumalmpl.multiplicar(20.5, 30.5)));
System.out.println("Resultado division es: " + Double.toString(sumalmpl.dividir(20.5, 30.5)));
System.out.println("Resultado raiz cuadrada es: " + Double.toString(sumalmpl.raizCuadrada(20.5)));
System.out.println("Resultado potencia es: " + Double.toString(sumalmpl.potencia(20.5, 30.5)));
```

**El resultado que nos saldrá es el siguiente:**

Iniciamos el puerto 3000



Terminal iniciar puerto

Ejecutamos el servidor:

```
PROBLEMAS 10 SALIDA CONSOLA DE DEPURACIÓN TERMINAL
diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$ java SumaServer -ORBInitialPort 3000 -ORBInitialHost localhost
Servidor de suma listo y en espera
[]
```

Terminal Servidor

Ahora ejecutamos el cliente:

```
PROBLEMAS 10 SALIDA CONSOLA DE DEPURACIÓN TERMINAL
diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$ java SumaClient -ORBInitialPort 3000 -ORBInitialHost localhost
Resultado suma es: 51.0
Resultado resta es: -10.0
Resultado multiplicacion es: 51.0
Resultado division es: 0.6721311475409836
Resultado raiz cuadrada es: 4.527692569068799
Resultado potencia es: 1.0197477651445544640
diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$
```

Terminal Cliente

Y la terminal también se cerrará cuando el cliente haya terminado:

```
diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$ java SumaServer -ORBInitialPort 3000 -ORBInitialHost localhost
Servidor de suma listo y en espera
Adios cerrando servidor
feb 27, 2023 10:16:59 PM com.sun.corba.se.impl.orb.ORBImpl checkShutdownState
ADVERTENCIA: "IP081600004 : [BAD_INV_ORDER] ORB has shutdown"
org.omg.CORBA.BAD_INV_ORDER: vnicid: OMNI minor code: -1 completed: No
    at com.sun.corba.se.impl.protocol.giopInvocation.dispatchOperationAfterShutdown(OMGSystemException.java:224)
    at com.sun.corba.se.impl.logging.OMGSystemException.bddOperationAfterShutdown(OMGSystemException.java:246)
    at com.sun.corba.se.impl.orb.ORBImpl.checkShutdownState(ORBImpl.java:1341)
    at com.sun.corba.se.impl.orb.ORBImpl.peekInvocationInfo(ORBImpl.java:1539)
    at com.sun.corba.se.impl.protocol.CorbaMessageMediatorImpl.runServerPostInvoke(CorbaMessageMediatorImpl.java:2250)
    at com.sun.corba.se.impl.protocol.CorbaMessageMediatorImpl.handleRequest(CorbaMessageMediatorImpl.java:2289)
    at com.sun.corba.se.impl.protocol.CorbaMessageMediatorImpl.createResponseHelper(CorbaMessageMediatorImpl.java:2165)
    at com.sun.corba.se.impl.protocol.CorbaMessageMediatorImpl.createResponse(CorbaMessageMediatorImpl.java:2013)
    at com.sun.corba.se.impl.protocol.CorbaMessageMediatorImpl.createReply(CorbaMessageMediatorImpl.java:623)
    at sunapp.sumaPOA.invoke(summaPOA.java:108)
    at com.sun.corba.se.impl.protocol.giopInvocation.dispatchRequest(CorbaServerRequestDispatcherImpl.java:654)
    at com.sun.corba.se.impl.protocol.CorbaServerRequestDispatcherImpl.dispatch(CorbaServerRequestDispatcherImpl.java:205)
    at com.sun.corba.se.impl.protocol.CorbaMessageMediatorImpl.handleRequest(CorbaMessageMediatorImpl.java:1768)
    at com.sun.corba.se.impl.protocol.CorbaMessageMediatorImpl.handleRequest(CorbaMessageMediatorImpl.java:1558)
    at com.sun.corba.se.impl.protocol.CorbaMessageMediatorImpl.handleInput(CorbaMessageMediatorImpl.java:940)
    at com.sun.corba.se.impl.protocol.giopmsgheaders.RequestMessage_1_2.callback(RequestMessage_1_2.java:98)
    at com.sun.corba.se.impl.protocol.giopmsgheaders.RequestMessage_1_2.handleRequest(CorbaMessageMediatorImpl.java:712)
    at com.sun.corba.se.impl.transport.SocketOrChannelConnectionImpl.performWork(SocketOrChannelConnectionImpl.java:474)
    at com.sun.corba.se.impl.transport.SocketOrChannelConnectionImpl.dowork(SocketOrChannelConnectionImpl.java:1237)
    at com.sun.corba.se.impl.orbutil.threadpool.ThreadPoolImpl$WorkerThread.performWork(ThreadPoolImpl.java:496)
    at com.sun.corba.se.impl.orbutil.threadpool.ThreadPoolImpl$WorkerThread.run(ThreadPoolImpl.java:519)
diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$
```

Terminal Servidor

Ahora para que sea diferente vamos a añadir varias funciones. En el IDL hemos puesto que calcule el seno, coseno, tangente, logaritmo neperiano, logaritmo en base 10, de un número entero a binario y de un binario a un número entero. Para finalizar como es una calculadora debemos implementar que opción quiere el cliente y que números va a introducir por consola para que se haga la operación con ellos.

```

1 module sumAPI {
2     double sumar(in double primerNumero, in double segundoNumero);
3     double restar(in double primerNumero, in double segundoNumero);
4     double multiplicar(in double primerNumero, in double segundoNumero);
5     double dividir(in double primerNumero, in double segundoNumero);
6     double raizCuadrada(in double primerNumero);
7     double potencia(in double primerNumero, in double segundoNumero);
8     double coseno (in double primerNumero);
9     double tangente (in double primerNumero);
10    double logaritmoNaperiano (in double primerNumero);
11    double logaritmoBaseDiez (in double primerNumero);
12    string decimalToBinario (in long primerNumero);
13    long binarioDecimal (in string numeroEnBinario);
14    oneway void shutdown();
15 }
16 
```

Fichero suma.idl

Acabamos de compilar el fichero suma.idl y ahora tendremos que añadir en el “SumaServer.java” las funciones nuevas, y en el “SumaClient.java” mostrar los resultados de esas operaciones.

```

idl -fall suma.idl

```

Comando para compilar suma.idl

Nos saldrán varios errores en la clase “SumaServer.java” y en la “SumaClient.java” ya que son en los únicos ficheros donde no se han actualizado, porque los hemos creado después de compilar el IDL.

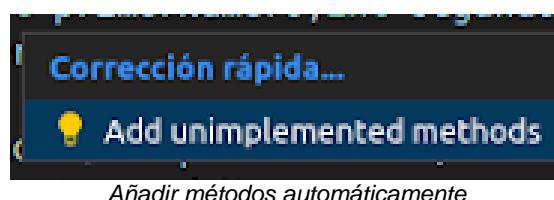
En la clase “SumaServer.java” nos está saliendo un error indicando que no existen esas funciones con “double”, por lo que nos vamos a corrección rápida.

```

    28 feb 10:34 28 feb 10:34
    Actividades > Visual Studio Code
    Archivo Editar Selección Ver Ir Ejecutar Terminal Ayuda
    README.md sumайд M J SumaServer.java X J SumaClient.java 1 J SumoClient.class
    EXPLORADOR ... README.md sumайд M J SumaServer.java X J SumaClient.java 1 J SumoClient.class
    CALCULADORA-CORBA src > J SumaServer.java > SumaImpl
    > orb.db
    > orb.db
    > sumaApp
    > Paso1.bat
    > Paso2.bat
    > Paso3.bat
    & sumaidl M 18
    J SumaClient.class J SumaServer.java 1
    J SumaImpl.class 20
    J SumaServer.class 22
    J SumoClient.class 24
    J SumoServer.java 26
    & .gitignore 28
    & manifest.mf 30
    README.md 32
    PROBLEMAS 28 SALIDA CONSOLA DE DEPURACIÓN TERMINAL
    diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$ idlj -fall sumaidl
    diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$ 
    * Histórico restaurado
    o diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$ 
  
```

Clase SumaServer fallos

Cuando hacemos click ahí nos saldrá si queremos añadir los métodos, seleccionamos la opción Add:



Eliminamos las funciones que ya estaban puestas y nos saldrá esto

```

Actividades > Visual Studio Code
Archivo Editar Selección Ver Ir Ejecutar Terminal Ayuda
... README.md sumайд M Sumaimpl> binarioToDecimal(String)
CALCULADORA-CORBA src > J SumaServer.java > Sumaimpl> SumaClient.java M X J SumaClient.java J SumoClient.class
    > orb.db
    > orb
    > orb.db
    > sumaApp
        > Paso1.bat
        > Paso2.bat
        > Paso3.bat
        & suma.idl M
        J SumaClient.class
        J Sumaimpl.class
        J SumaServer.class M
        & ignore
        & manifest.mf
        & README.mfd
EXPLORADOR ... README.md sumайд M Sumaimpl> binarioToDecimal(String)
36     public double raizCuadrada(double primerNumero) {
37         return Math.sqrt(primerNumero);
38     }
39
40     public double potencia(double primerNumero, double segundoNumero) {
41         return Math.pow(primerNumero, segundoNumero);
42     }
43
44     public double seno(double primerNumero) {
45         return Math.sin(primerNumero);
46     }
47
48     public double coseno(double primerNumero) {
49         return Math.cos(primerNumero);
50     }
51
52     public double tangente(double primerNumero) {
53         return Math.tan(primerNumero);
54     }
55
56     public double logaritmoNaperiano(double primerNumero) {
57         return Math.log(primerNumero);
58     }
59
60     public double logaritmoBaseDiez(double primerNumero) {
61         return Math.log10(primerNumero);
62     }
63
64     public String decimalToBinario(int primerNumero) {
65         return Long.toBinaryString(primerNumero);
66     }
67
68     public int binarioToDecimal(String numeroEnBinario) {
69         return Integer.parseInt(numeroEnBinario, 2);
70     }
71
72
73
74     public class SumaServer{
PROBLEMAS 17 SALIDA CONSOLA DE DEPURACIÓN TERMINAL
diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$ idl2 -fall suma.idl
diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$ 
+ Historial restaurante
diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$ 
+ Almir-Puent...
+ Servidor.svc
+ Cliente.svc
+ Subir-Github
+ bash inc
+ 
Lin: 68, col: 58 Espacios: 4 UTF-8 CRLF ⓘ Java ⓘ Go Live ⓘ Prettier ⓘ 
+ ... x
ESQUEMA
LÍNEA DE TIEMPO
JAVA PROJECTS
master < 0 17

```

Poner el resultado en el return a todas las funciones

Ahora tendremos que mostrar en la clase cliente “SumaClient.java” todas las funciones que hemos implementado.

Aquí estamos creando una variable “scan” para que el usuario puede escribir en la terminal con el objeto Scanner, después nos creamos dos variables double donde vamos a poner los números, las funciones que tengan parámetros con long se coge el entero y ya estaría, y después otra variable, pero String que es para la última opción de si quiero pasar de un binario a número decimal (número entero). Estamos haciendo un do while porque el usuario hasta que no introduzca el numero 0 no podrá salir del bucle. Para seleccionar una opción debes introducir el valor numérico, por ejemplo, en la suma “1”, resta “2” etc. además, todo esto se pone entre un try catch para evitar las excepciones, por ejemplo, dividir entre 0...

Introducimos la opción en la variable operación seleccionada y entramos en el switch case. Los casos disponibles son del 0 al 13, si el usuario introduce otro número, la opción seleccionada es por defecto (default) y nos mostrará por pantalla que esa opción no existe. También tenemos que tener en cuenta que cada case debe tener un break para que no se ejecute los demás casos.

```

    package CALCULADORA.CORBA;
    import J_SumaClient;
    import sumImpl;
    import java.util.Scanner;
    import sun.misc.CashManager;
    import org.omg.CORBA.Object;
    import org.omg.CORBA.NamingContextPackage.*;
    import org.omg.CORBA.*;

    public class SumaClient {
        static suma sumaImpl;
        Run(Debug)
        public static void main(String args[]){
            try{
                ORB orb = ORB.init(args, null);
                Object objRef = orb.resolve_initial_references("NameService");
                NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
                String name = ncRef.list("sum");
                sumaImpl = sumImplHelper.narrow(ncRef.resolve(name));
                Scanner scan = new Scanner(System.in);
                int operacionSeleccionada = 0;
                double numero1 = 0.0;
                double numero2 = 0.0;
                String numeroBinario = "";
                do{
                    System.out.println("0 Salir");
                    System.out.println("1 Resta");
                    System.out.println("2 Multiplicacion");
                    System.out.println("3 Division");
                    System.out.println("4 Potencia");
                    System.out.println("5 Seno");
                    System.out.println("6 Coseno");
                    System.out.println("7 Tangente");
                    System.out.println("8 Logaritmo neperiano");
                    System.out.println("9 Logaritmo en base 10");
                    System.out.println("10 Base de logaritmo natural");
                    System.out.println("11 Binario a decimal");
                } catch (Exception e) {
                    e.printStackTrace();
                }
                operacionSeleccionada = scan.nextInt();
                switch(operacionSeleccionada){
                    case 0:
                        System.out.println("Adios");
                        break;
                    case 1:
                        System.out.println("Introduce el primer numero");
                        numero1 = scan.nextDouble();
                        System.out.println("Introduce el segundo numero");
                        numero2 = scan.nextDouble();
                        System.out.println("Resultado resta es: " + Double.toString(sumaImpl.sumar(numero1, numero2)));
                        break;
                    case 2:
                        System.out.println("Introduce el primer numero");
                        numero1 = scan.nextDouble();
                        System.out.println("Introduce el segundo numero");
                        numero2 = scan.nextDouble();
                        System.out.println("Resultado multiplicacion es: " + Double.toString(sumaImpl.multiplicar(numero1, numero2)));
                        break;
                    case 3:
                        System.out.println("Introduce el primer numero");
                        numero1 = scan.nextDouble();
                        System.out.println("Introduce el segundo numero");
                        numero2 = scan.nextDouble();
                        System.out.println("Resultado division es: " + Double.toString(sumaImpl.dividir(numero1, numero2)));
                        break;
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
  
```

Primera parte clase SumaClient

A continuación, voy a mostrar todos los case del switch:

```

switch (operacionSeleccionada) {
    case 0:
        System.out.println("Adios");
        break;
    case 1:
        System.out.println("Introduce el primer numero");
        numero1 = scan.nextDouble();
        System.out.println("Introduce el segundo numero");
        numero2 = scan.nextDouble();
        System.out.println("Resultado resta es: " + Double.toString(sumaImpl.sumar(numero1, numero2)));
        break;
    case 2:
        System.out.println("Introduce el primer numero");
        numero1 = scan.nextDouble();
        System.out.println("Introduce el segundo numero");
        numero2 = scan.nextDouble();
        System.out.println("Resultado multiplicacion es: " + Double.toString(sumaImpl.multiplicar(numero1, numero2)));
        break;
    case 3:
        System.out.println("Introduce el primer numero");
        numero1 = scan.nextDouble();
        System.out.println("Introduce el segundo numero");
        numero2 = scan.nextDouble();
        System.out.println("Resultado division es: " + Double.toString(sumaImpl.dividir(numero1, numero2)));
        break;
}
  
```

Case con valores del 0 - 4

```

case 5: {
    System.out.println("Introduce el primer numero");
    numero1 = scan.nextDouble();

    System.out.println("Resultado raiz cuadrada es: " + Double.toString(sumImpl.raizCuadrada(numero1)));
    break;
}
case 6: {
    System.out.println("Introduce el primer numero");
    numero1 = scan.nextDouble();

    System.out.println("Introduce el segundo numero");
    numero2 = scan.nextDouble();

    System.out.println("Resultado potencia es: " + Double.toString(sumImpl.potencia(numero1, numero2)));
    break;
}
case 7: {
    System.out.println("Introduce el primer numero");
    numero1 = scan.nextDouble();

    System.out.println("Resultado seno es: " + Double.toString(sumImpl.seno(numero1)));
    break;
}
case 8: {
    System.out.println("Introduce el primer numero");
    numero1 = scan.nextDouble();

    System.out.println("Resultado coseno es: " + Double.toString(sumImpl.coseno(numero1)));
    break;
}
case 9: {
    System.out.println("Introduce el primer numero");
    numero1 = scan.nextDouble();

    System.out.println("Resultado tangente es: " + Double.toString(sumImpl.tangente(numero1)));
    break;
}
case 10: {
    System.out.println("Introduce el primer numero");
    numero1 = scan.nextDouble();

    System.out.println("Resultado logaritmo neperiano es: " + Double.toString(sumImpl.logaritmoNeperiano(numero1)));
    break;
}
}

```

Case con valores del 5 - 10

Para finalizar con esta clase cuando salga del do while se ejecutará la función “sumimpl.shutdown();” que cierra el ORB.

```

case 11: {
    System.out.println("Introduce el primer numero");
    numero1 = scan.nextDouble();

    System.out.println("Resultado logaritmo en base 10 es: " + Double.toString(sumImpl.logaritmoBaseDiez(numero1)));
    break;
}
case 12: {
    System.out.println("Introduce el primer numero (entero): ");
    numero1 = scan.nextInt();

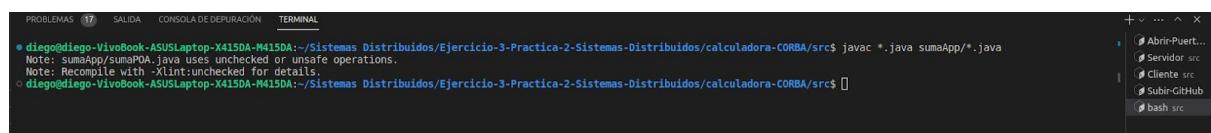
    System.out.println("Resultado decimal a binario es: " + sumImpl.decimalToBinario((int) numero1));
    break;
}
case 13: {
    System.out.println("Introduce el numero binario");
    numeroBinario = scan.nextInt();
    System.out.println("Resultado binario a decimal es: " + Integer.toString(sumImpl.binarioDecimal(numeroBinario)));
    break;
}
default: {
    System.out.println("Esta opcion no existe");
}
}
}
catch (Exception e) {
    throw new Error(e);
}
} while (operacionSeleccionada != 0);

sumImpl.shutdown();

```

Case con valores del 11 - 13, el default y la condición del bucle

Ahora comprobamos que esta todo correcto para darlo por finalizarlo. Pero antes de este paso debemos compilar los programas “.java” ya que hemos hecho modificaciones sobre ellos.



```

PROBLEMAS (17) SALIDA CONSOLA DE DEPURACIÓN TERMINAL
• diego@diego-VivoBook-ASUSLaptop-X415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$ javac *.java sumaApp/*.java
Note: sumaApp/sumaPOA.java uses unchecked or unsafe operations.
      Note: Recompile with -Xlint:unchecked for details.
○ diego@diego-VivoBook-ASUSLaptop-X415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$ 

```

Comando para compilar todos los .java

## Resultados:

Terminal para iniciar puerto:

```
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL
diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$ orb -ORBInitialPort 3001
```

Terminal iniciar puerto

### Terminal para ejecutar el servidor:

```
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL
diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$ java SumaServer -ORBInitialPort 3001 -ORBInitialHost localhost
Servidor de suma listo y en espera
```

Terminal Servidor

### Terminal para ejecutar el cliente:

Primero se muestra las 14 opciones que puede elegir el cliente

```
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL
diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$ java SumaClient -ORBInitialPort 3001 -ORBInitialHost localhost
0. Salir
1. Suma
2. Resta
3. Multiplicacion
4. Division
5. Raiz cuadrada
6. Potencia
7. Seno
8. Coseno
9. Tangente
10. Logaritmo neperiano
11. Logaritmo en base 10
12. Decimal a binario
13. Binario a decimal
?Que opcion eliges?:
```

Terminal Cliente menu

Cuando elegimos la opción deberemos introducir los siguientes datos que nos indica, en el caso de la suma será dos números.

```
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL
10. Logaritmo neperiano
11. logaritmo en base 10
12. Decimal a binario
13. Binario a decimal
?Que opcion eliges:
Introduce el primer numero
22.3
Introduce el segundo numero
33.2
Resultado suma es: 55.5

0. Salir
1. Suma
2. Resta
3. Multiplicacion
4. Division
5. Raiz cuadrada
6. Potencia
7. Seno
8. Coseno
9. Tangente
10. Logaritmo neperiano
11. logaritmo en base 10
12. Decimal a binario
13. Binario a decimal
?Que opcion eliges?:
```

Opción elegida 1, con respuesta y después muestra el menu

Nos mostrará el resultado final y nos volverá a pedir que introduzcamos otra operación, ahora hemos implementado la 7, que es el seno y solo nos pedirá un número y después mostrará el resultado, como a continuación.

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL
8. Coseno
9. Tangente
10. Logaritmo neperiano
11. logaritmo en base 10
12. Decimal a binario
13. Binario a decimal
?Que opcion eliges?
Introduce el primer numero
38
Resultado seno es: -0.9880316240928618

-----
0. Salir
1. Suma
2. Resta
3. Multiplicacion
4. Division
5. Raiz cuadrada
6. Potencia
7. Seno
8. Coseno
9. Tangente
10. Logaritmo neperiano
11. logaritmo en base 10
12. Decimal a binario
13. Binario a decimal
?Que opcion eliges?:

```

Opción elegida 7, con respuesta y después muestra el menu

Por último, hemos elegido la opción 12 que es pasar de un numero decimal (entero) a binario y debemos pasar un numero int, long. Si le pasamos bien el numero nos devolverá el resultado como la siguiente foto:

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL
8. Coseno
9. Tangente
10. Logaritmo neperiano
11. logaritmo en base 10
12. Decimal a binario
13. Binario a decimal
?Que opcion eliges?
12
Introduce el primer numero (entero):
40
Resultado decimal a binario es: 101000

-----
0. Salir
1. Suma
2. Resta
3. Multiplicacion
4. Division
5. Raiz cuadrada
6. Potencia
7. Seno
8. Coseno
9. Tangente
10. Logaritmo neperiano
11. logaritmo en base 10
12. Decimal a binario
13. Binario a decimal
?Que opcion eliges?:

```

Opción elegida 12, con respuesta y después muestra el menu

Si ponemos un numero decimal nos mostrará un error y el cliente terminará la conexión y se cierra el ORB.

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL
8. Coseno
9. Tangente
10. Logaritmo neperiano
11. logaritmo en base 10
12. Decimal a binario
13. Binario a decimal
?Que opcion eliges?
12
Introduce el primer numero (entero):
40.1
java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:864)
    at java.util.Scanner.next(Scanner.java:1495)
    at java.util.Scanner.nextInt(Scanner.java:2117)
    at java.util.Scanner.nextInt(Scanner.java:2076)
    at sumaClient.main(SumaClient.java:46)

-----
diego@diego-VivoBook-ASUSLaptop-X415DA-M415DA:~/Sistemas Distribuidos/Ejercicio-3-Practica-2-Sistemas-Distribuidos/calculadora-CORBA/src$ 

```

Introducir un dato incorrecto

El código nuevo en este caso es:

En el fichero suma.idl:

```
double seno (in double primerNumero);  
  
double coseno (in double primerNumero);  
  
double tangente (in double primerNumero);  
  
double logaritmoNeperiano (in double primerNumero);  
  
double logaritmoBaseDiez (in double primerNumero);  
  
string decimalToBinario (in long primerNumero);  
  
long binarioToDecimal (in string numeroEnBinario);
```

En el fichero SumaServer.java:

```
class SumaImpl extends sumaPOA{  
  
    public double seno(double primerNumero) {  
  
        return Math.sin(primerNumero);  
  
    }  
  
    public double coseno(double primerNumero) {  
  
        return Math.cos(primerNumero);  
  
    }  
  
    public double tangente(double primerNumero) {  
  
        return Math.tan(primerNumero);  
  
    }  
  
    public double logaritmoNeperiano(double primerNumero) {  
  
        return Math.log(primerNumero);  
  
    }  
  
    public double logaritmoBaseDiez(double primerNumero) {  
  
        return Math.log10(primerNumero);  
  
    }  
}
```

```

public String decimalToBinario(int primerNumero) {
    return Long.toBinaryString(primerNumero);
}

public int binarioToDecimal(String numeroEnBinario) {
    return Integer.parseInt(numeroEnBinario, 2);
}
}

```

Para SumaClient.java:

```

Scanner scan = new Scanner (System.in);
int operacionSeleccionada = 0;
double numero1 = 0.0;
double numero2 = 0.0;
String numeroBinario = "";
do {
    System.out.println("0. Salir");
    System.out.println("1. Suma");
    System.out.println("2. Resta");
    System.out.println("3. Multiplicacion");
    System.out.println("4. Division");
    System.out.println("5. Raiz cuadrada");
    System.out.println("6. Potencia");
    System.out.println("7. Seno");
    System.out.println("8. Coseno");
    System.out.println("9. Tangente");
    System.out.println("10. Logaritmo neperiano");
    System.out.println("11. logaritmo en base 10");
    System.out.println("12. Decimal a binario");
    System.out.println("13. Binario a decimal");
    try {
        System.out.println("¿Que opcion eliges?: ");
        operacionSeleccionada = scan.nextInt();
        scan.nextLine();

        switch (operacionSeleccionada) {
            case 0: {
                System.out.println("Adios");
                break;
            }
            case 1:{{
                System.out.println("Introduce el primer numero");
                numero1 = scan.nextDouble();

                System.out.println("Introduce el segundo numero");
                numero2 = scan.nextDouble();

                System.out.println("Resultado suma es: " + Double.toString(sumimpl.sumar(numero1,
numero2)));
                break;
            }
            case 2: {
                System.out.println("Introduce el primer numero");
                numero1 = scan.nextDouble();

                System.out.println("Introduce el segundo numero");
            }
        }
    }
}

```

```

        numero2 = scan.nextDouble();

        System.out.println("Resultado resta es: " + Double.toString(sumimpl.restar(numero1,
numero2)));
        break;
    }
    case 3: {
        System.out.println("Introduce el primer numero");
        numero1 = scan.nextDouble();

        System.out.println("Introduce el segundo numero");
        numero2 = scan.nextDouble();

        System.out.println("Resultado multiplicacion es: " +
Double.toString(sumimpl.multiplicar(numero1, numero2)));
        break;
    }
    case 4: {
        System.out.println("Introduce el primer numero");
        numero1 = scan.nextDouble();

        System.out.println("Introduce el segundo numero");
        numero2 = scan.nextDouble();

        System.out.println("Resultado division es: " + Double.toString(sumimpl.dividir(numero1,
numero2)));
        break;
    }
    case 5: {
        System.out.println("Introduce el primer numero");
        numero1 = scan.nextDouble();

        System.out.println("Resultado raiz cuadrada es: " +
Double.toString(sumimpl.raizCuadrada(numero1)));
        break;
    }
    case 6: {
        System.out.println("Introduce el primer numero: ");
        numero1 = scan.nextDouble();

        System.out.println("Introduce el segundo numero");
        numero2 = scan.nextDouble();

        System.out.println("Resultado potencia es: " + Double.toString(sumimpl.potencia(numero1,
numero2)));
        break;
    }
    case 7: {
        System.out.println("Introduce el primer numero");
        numero1 = scan.nextDouble();

        System.out.println("Resultado seno es: " + Double.toString(sumimpl.seno(numero1)));
        break;
    }
    case 8: {
        System.out.println("Introduce el primer numero");
        numero1 = scan.nextDouble();

        System.out.println("Resultado coseno es: " + Double.toString(sumimpl.coseno(numero1)));
        break;
    }
    case 9: {
        System.out.println("Introduce el primer numero");
        numero1 = scan.nextDouble();
    }
}

```

```

        System.out.println("Resultado tangente es: " +
Double.toString(sumimpl.tangente(numero1)));
        break;
    }
    case 10: {
        System.out.println("Introduce el primer numero");
        numero1 = scan.nextDouble();

        System.out.println("Resultado logaritmo neperiano es: " +
Double.toString(sumimpl.logaritmoNeperiano(numero1)));
        break;
    }
    case 11: {
        System.out.println("Introduce el primer numero");
        numero1 = scan.nextDouble();

        System.out.println("Resultado logaritmo en base 10 es: " +
Double.toString(sumimpl.logaritmoBaseDiez(numero1)));
        break;
    }
    case 12: {
        System.out.println("Introduce el primer numero (entero): ");
        numero1 = scan.nextInt();

        System.out.println("Resultado decimal a binario es: " + sumimpl.decimalToBinario((int)
numero1));
        break;
    }
    case 13: {
        System.out.println("Introduce el numero binario");
        numeroBinario = scan.next();
        System.out.println("Resultado binario a decimal es: " +
Integer.toString(sumimpl.binarioToDecimal(numeroBinario)));
        break;
    }
    default: {
        System.out.println("Esa opcion no existe");
    }
}

}

catch (Exception e) {
    operacionSeleccionada = 0;
    e.printStackTrace();
}

System.out.println("\n\n-----\n-----\n");
}

} while (operacionSeleccionada != 0);

```

Por último, todas las pruebas que he hecho de modificación de datos se han realizado en un repositorio que ha sido eliminado, ya que había bastantes commit, y para que en el momento de explicar las modificaciones de los commit sea más sencilla y legible, he creado un nuevo repositorio.

El repositorio donde se ha realizado el ejercicio 3: <https://github.com/Diegopower199/calculadora-CORBA>

## 6. Conclusión

---

En resumen, en esta práctica hemos comprendido el funcionamiento que nos ofrece CORBA, además que hemos podido observar que podemos hacer cualquier cosa que se nos ocurra, por ejemplo, una calculadora.

## 7. Bibliografía

---

- <https://docs.oracle.com/javase/7/docs/technotes/guides/idl/jidlExample.html>
- <http://grasia.fdi.ucm.es/jpavon/docencia/dso/corba02ejemplojava.pdf>
- <http://www.jtech.ua.es/j2ee/2002-2003/modulos/idl-corba/apuntes/tema4.htm>
- <http://www.jtech.ua.es/j2ee/2004-2005/modulos/eai/sesion03-apuntes.htm>
- <https://www.youtube.com/watch?v=9YUaf-uxuRM&t=187s>