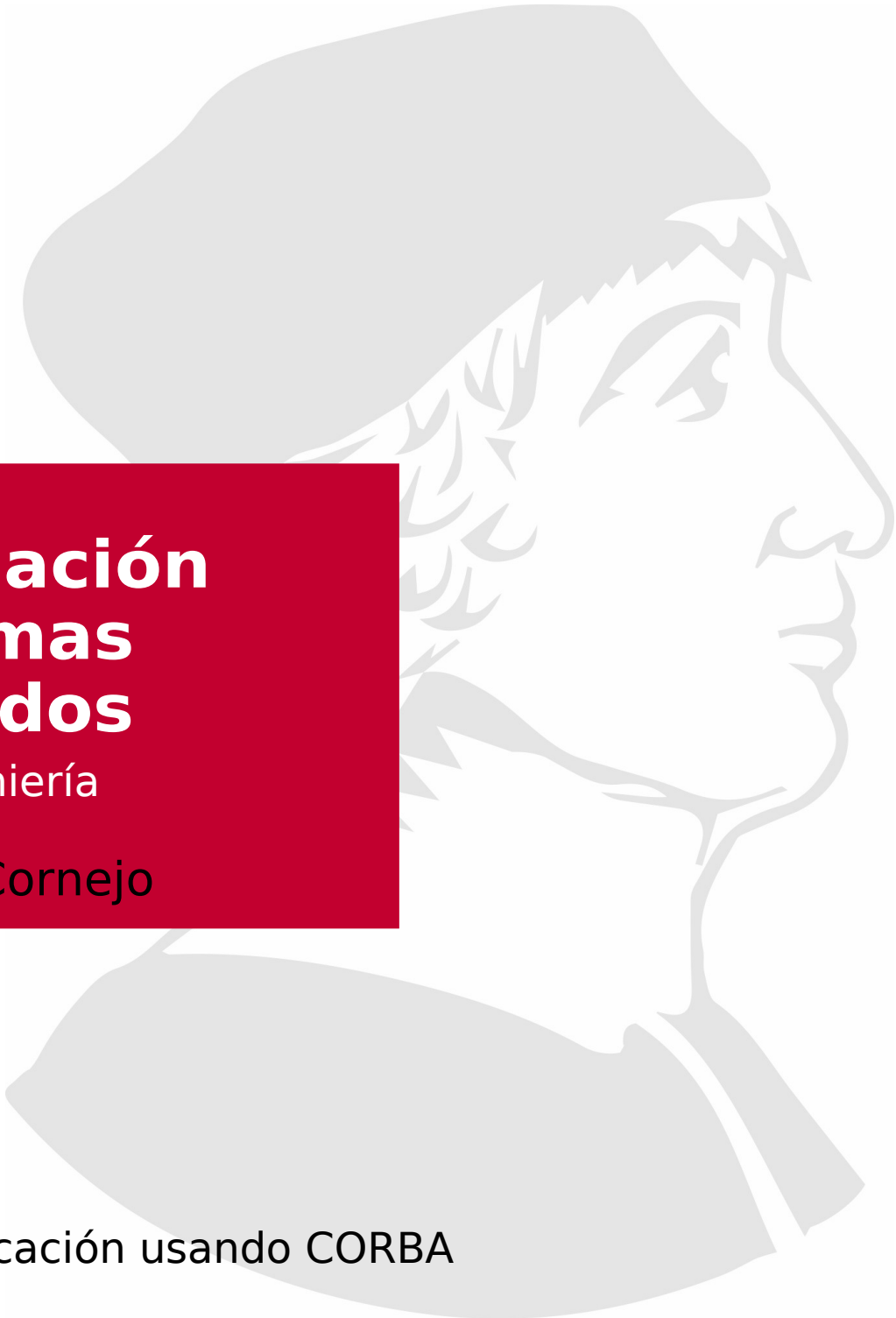


Programación de sistemas distribuidos

Grado en Ingeniería Informática

Sara Marcos Cornejo

Práctica 2: Aplicación usando CORBA

UNIVERSIDAD NEBRIJA

Tabla de contenido

I. Introducción

¿Qué es CORBA?

Common Object Request Broker Architecture (CORBA) es un estándar definido por Object Management Group (OMG) que permite que diversos componentes de software escritos en múltiples lenguajes de programación y que corren en diferentes computadoras, puedan trabajar juntos; es decir, facilita el desarrollo de aplicaciones distribuidas en entornos heterogéneos.

Su objetivo es ayudar a reducir la complejidad, disminuir los costes y acelerar la introducción de nuevas aplicaciones informáticas, promoviendo la teoría y la práctica de la tecnología de objetos en los sistemas distribuidos.

Es una tecnología que oculta la programación a bajo nivel de aplicaciones distribuidas. No obstante, también brinda al programador una tecnología orientada a objetos; las funciones objetos y estos objetos pueden estar en diferentes máquinas, pero el programador accederá a ellos a través de funciones normales dentro de su programa.

CORBA, además de ser una especificación multiplataforma, también define servicios habitualmente necesarios como seguridad y transacciones. Y así este no es un sistema operativo en sí, en realidad es un middleware.

II. Objetivos

- Primera toma de contacto con esta tecnología.
- Estudiar las funcionalidades y ventajas que ofrece.

III. Elementos

Algunos de los componentes que se van a utilizar son:

➢ **Object Request Broker (ORB):** El Object Request Broker (ORB), es un componente fundamental de la arquitectura CORBA y su misión es facilitar la comunicación entre objetos. Este se encarga de enviar las peticiones a los objetos y retornar las respuestas a los clientes que las invocan por el proceso de serialización. Su principal característica la transparencia. Facilita la comunicación entre cliente y servidor ocultando:

- La localización de los objetos: El cliente no tiene por qué saber dónde se encuentra el objeto destino.
- La implementación de los objetos: El cliente ignora el lenguaje de programación con el que se ha escrito el objeto remoto, su implementación o el sistema operativo en el que se encuentra.
- Estado de ejecución del objeto: Al enviar una petición sobre un objeto remoto, el ORB se encarga de inicializar el objeto en caso de ser necesario, antes de enviarle la petición.
- Mecanismos de comunicación de los objetos: El cliente no sabe qué mecanismos de comunicación utiliza el ORB para enviar las peticiones y retornar el resultado.

➢ Interface Definition Language (IDL)

El lenguaje de definición de interfaz o IDL (Interface Definition Language), es un lenguaje que se utiliza para definir las interfaces entre los componentes de una aplicación y es el elemento que soporta la interoperabilidad de la tecnología. El IDL sólo puede definir interfaces, no implementaciones. Al especificar las interfaces entre objetos en CORBA, IDL es el encargado de asegurar la independencia del lenguaje de programación utilizado.

Para que esto sea posible, es necesario asegurar que los datos son correctamente intercambiados entre dos lenguajes distintos, para lo cual IDL define los tipos de datos de una forma independiente del lenguaje con las correspondencias necesarias. Por ejemplo, un tipo long en C++ de 32 bits puede corresponderse con un int de Java.

Tendremos principalmente dos tipos diferentes de IDL en CORBA:

- **Stub IDL**: es la interfaz estática a los servicios declarados en las interfaces IDL, para el cliente todas las llamadas parecen locales. Actuará como proxy del objeto remoto, realizando la invocación de métodos remotos, incluyendo la serialización, la recepción de respuestas y la deserialización. El cliente puede tener tantos stubs como interfaces IDL existan. Es generado a partir del IDL en el lenguaje de programación del cliente (C, C++ Java, Smalltalk, Ada,...) por un compilador IDL.

- **Skeleton IDL**: es el representante estático del cliente en el servidor. Para el servidor todas las llamadas parecen locales. Es generado a partir del IDL por un compilador IDL y realiza la deserialización de las invocaciones del cliente.

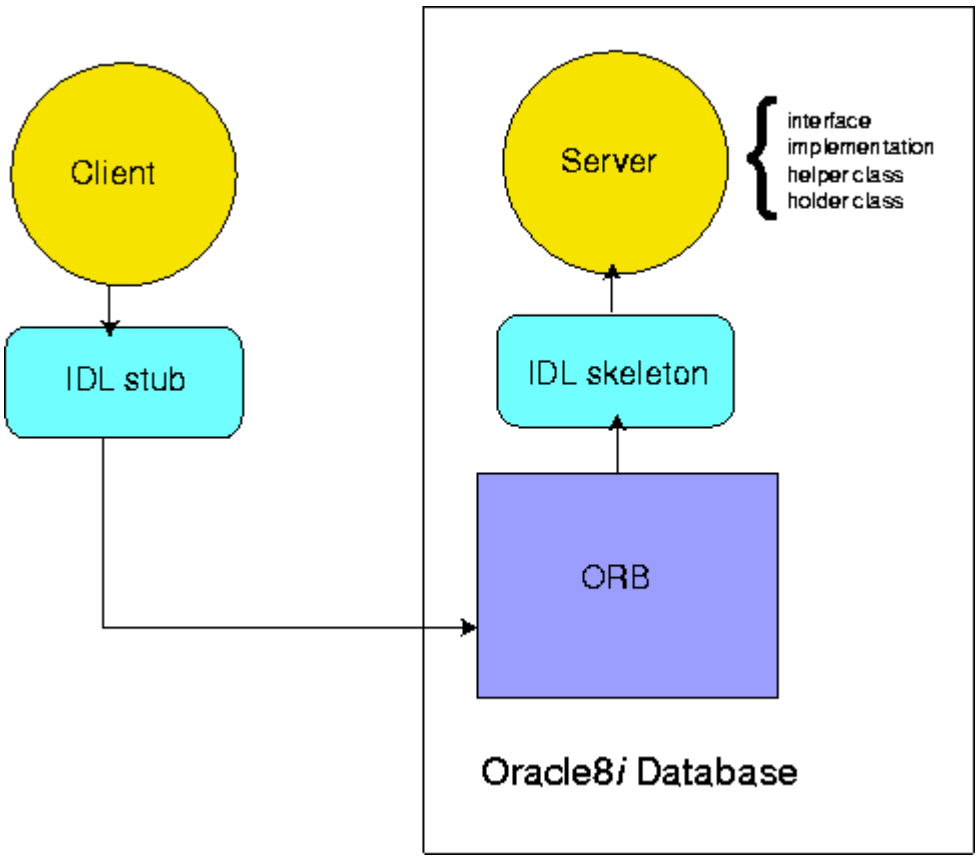


Figura 1 componentes aplicación CORBA

IV. Funcionamiento

Todos los componentes CORBA son objetos, cada cual tendrá una interfaz y una identidad única, cada uno de los objetos se podrá implementar en un lenguaje de programación distinto y ejecutarse sobre cualquier sistema operativo. El servidor crea objetos remotos, hace accesibles referencias a esos objetos remotos y espera a que los clientes invoquen a estos objetos o a sus métodos. Por otro lado el cliente obtiene una referencia de uno o más objetos remotos en el servidor e invoca a sus métodos.

La manera de realizar la invocación por parte del cliente es usando **stub**, una interfaz de comunicación con el servidor generada a partir del IDL, usando la invocación dinámica para acceder a los objetos del servidor. Necesita para ello una referencia del objeto remoto o IOR (Interoperable Object References), el tipo del objeto y el nombre de la propia operación que desea invocar. Describiendo las interfaces IDL, un ORB genera automáticamente código en el lenguaje seleccionado para realizar la integración de las aplicaciones distribuidas.

El ORB, a partir de la petición del cliente encuentra el código de la implementación apropiada y transmite los parámetros, el control a la implementación de la interfaz a través del **skeleton** IDL e informa de excepciones en el proceso.

Para recibir la petición, recibe la invocación de uno de sus métodos como llamadas desde el ORB hacia la implementación de la interfaz, la llamada puede venir de un cliente que haya utilizado los stubs IDL; los esqueletos de la interfaz son específicos de cada interfaz y del adaptador de objetos que exista en la implementación de CORBA. Una vez completada la invocación, el control y los valores de retorno son devueltos al cliente.

Ejemplo funcionamiento: uso del lenguaje IDL y del compilador correspondiente cuando se construye la aplicación.

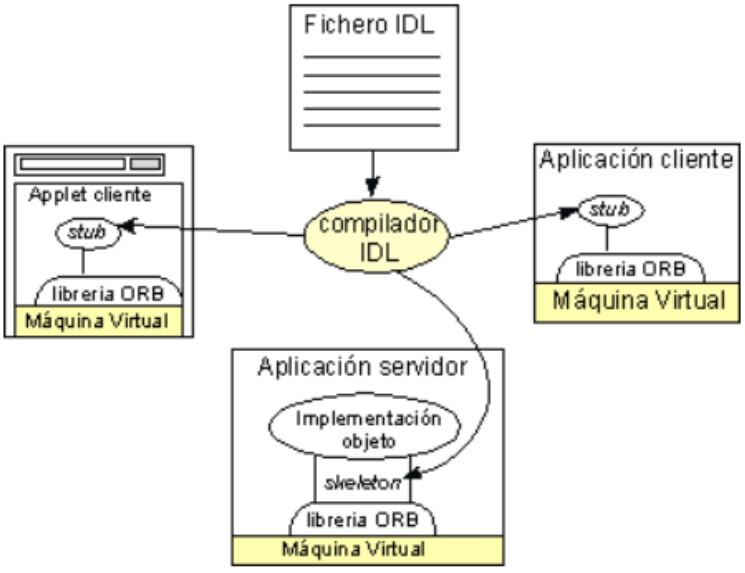


Figura 2 construcción aplicación

V. Entregables

Para poder realizar y ejecutar este programa se necesitará un **editor** como Atom, Eclipse, VisualStudio y el **JDK** de Java, que se puedes descargar de http://www.java.com/es/

A. Ejercicio 1

Realizar 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.

hello.idl

```
pract2sd > src > hello.idl
1  module HelloApp
2  {
3    interface Hello
4    {
5    string sayHello();
6    oneway void shutdown();
7    };
8  };
```

El primer paso para crear una aplicación CORBA es especificar todos sus objetos y sus interfaces utilizando el lenguaje de definición de interfaz (IDL) de OMG. El siguiente código está escrito en OMG IDL y describe un objeto CORBA cuya operación sayHello() devuelve una cadena y cuyo método shutdown() cierra el ORB

HelloClient.java:

```
pract2sd > src > HelloClient.java > HelloClient
1  import HelloApp.*;
2  import org.omg.CosNaming.*;
3  import org.omg.CosNaming.NamingContextPackage.*;
4  import org.omg.CORBA.*;
5
6  public class HelloClient
7  {
8    static Hello helloImpl;
     Run | Debug
9    public static void main(String args[])
10     {
11       try{
12         // crea e inicializa el ORB
13         ORB orb = ORB.init(args, null);
14
15         // obtiene la referencia del servicio de nombres
16         org.omg.CORBA.Object objRef =
17             orb.resolve_initial_references("NameService");
18         // Usa NamingContextExt, que es parte de la especificacion Interoperable
19         NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
20
21         // obtiene la referencia del objeto en el servicio de nombres
22         String name = "Hello";
23         helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));
24
25         System.out.println("Obtained a handle on server object: " + helloImpl);
26         System.out.println(helloImpl.sayHello());
27         helloImpl.shutdown();
28
29         } catch (Exception e) {
30           System.out.println("ERROR : " + e) ;
31           e.printStackTrace(System.out);
32           }
33   }
34 }
```

El cliente:

- Crea e inicializa un ORB
- Obtiene una referencia al contexto de nomenclatura raíz.
- Busca "Hola" en el contexto de nombres y recibe una referencia a ese objeto CORBA
- Invoca las operaciones sayHello() y shutdown() del objeto e imprime el resultado

HelloServer.java:

pract2sd > src > HelloServer.java > ...

```java
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
import java.util.Properties;

class HelloImpl extends HelloPOA {
  private ORB orb;

  public void setORB(ORB orb_val) {
    orb = orb_val;
  }

  // metodo sayHello() para que el cliente pueda invocarlo
  public String sayHello() {
    return "\nHello world !!\n";
  }

  // metodo shutdown() para que el cliente pueda terminar la ejecucion del servidor
  public void shutdown() {
    orb.shutdown(false);
  }
}

public class HelloServer {
  Run | Debug
  public static void main(String args[]) {
    try{
      // crea e inicializa el ORB
      ORB orb = ORB.init(args, null);

      // obtiene la referencia del rootpoa & activa el POAManager
      POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
      rootpoa.the_POAManager().activate();

      // crea el servant
      HelloImpl helloImpl = new HelloImpl();
      helloImpl.setORB(orb);

      // obtiene la referencia del objeto desde el servant
      org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
      Hello href = HelloHelper.narrow(ref);

      // obtiene la referencia del servicio de nombres
      org.omg.CORBA.Object objRef =
          orb.resolve_initial_references("NameService");
      // Usa NamingContextExt, que es parte de la especificacion Interoperable
      NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

      // vincula la referencia del objeto en el servicio de nombres
      String name = "Hello";
      NameComponent path[] = ncRef.to_name( name );
      ncRef.rebind(path, href);

      System.out.println("HelloServer ready and waiting ...");

      // espera por las invocaciones de los clientes
      orb.run();
    }
      catch (Exception e) {
        System.err.println("ERROR: " + e);
        e.printStackTrace(System.out);
      }
      System.out.println("HelloServer Exiting ...");

  }
}
```

El servidor consta de dos clases, el sirviente y el servidor. El servidor, HelloImpl , es la implementación de la interfaz Hello IDL; cada instancia de Hello se implementa mediante una instancia de HelloImpl . El sirviente es una subclase de HelloPOA , que genera el compilador idlj a partir del IDL de ejemplo. El sirviente contiene un método para cada operación IDL, en este ejemplo, los métodos sayHello() y shutdown() . Los métodos de servidor son como los métodos ordinarios de Java; el esqueleto proporciona el código adicional para tratar con el ORB, con argumentos y resultados de cálculo de referencias, etc.

La clase HelloServer tiene el método main() del servidor, que:

- Crea e inicializa una instancia ORB
- Obtiene una referencia al POA raíz y activa el POAManager
- Crea una instancia de sirviente (la implementación de un objeto CORBA Hello y se lo informa al ORB
- Obtiene una referencia de objeto CORBA para un contexto de nomenclatura en el que registrar el nuevo objeto CORBA
- Obtiene el contexto de nomenclatura raíz
- Registra el nuevo objeto en el contexto de nombres bajo el nombre "Hola"
- Espera las invocaciones del nuevo objeto del cliente

Compilación y ejecución:

- Se compilará primero el IDL, luego el servidor y luego el cliente usando los siguientes códigos:

$ idlj -fall hello.idl

$ javac HelloServer.java

$ javac HelloClient.java

Es tras esta compilación que se generan todos los archivos necesarios.

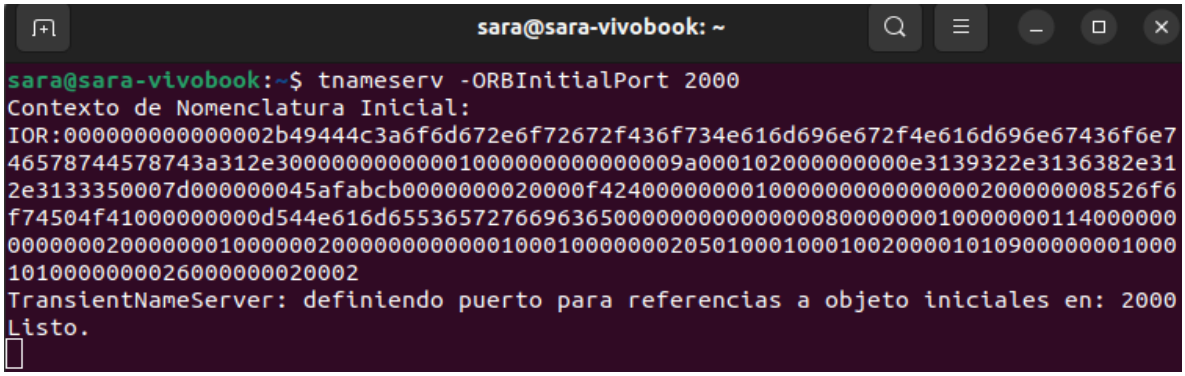
- Para la ejecución se necesita tener abiertas tres ventanas del Símbolo del sistema. La primera iniciará el puerto, la segunda ejecutará el servidor y la tercera el cliente. El código para jecutarla es, respectivamente:

$ tnameserv -ORBInitialPort 2000

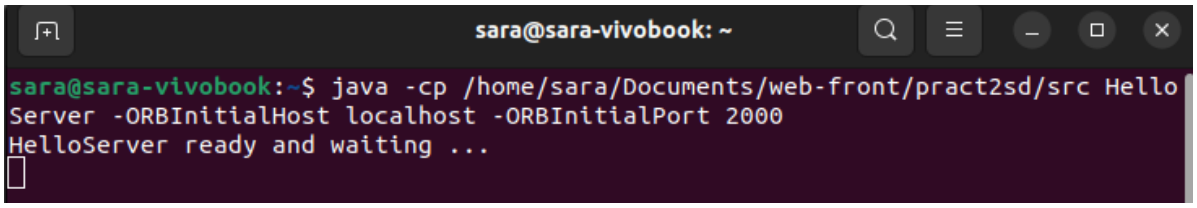
$ java HelloServer -ORBInitialHost localhost -ORBInitialPort 2000

$ java HelloClient -ORBInitialHost localhost -ORBInitialPort 2000

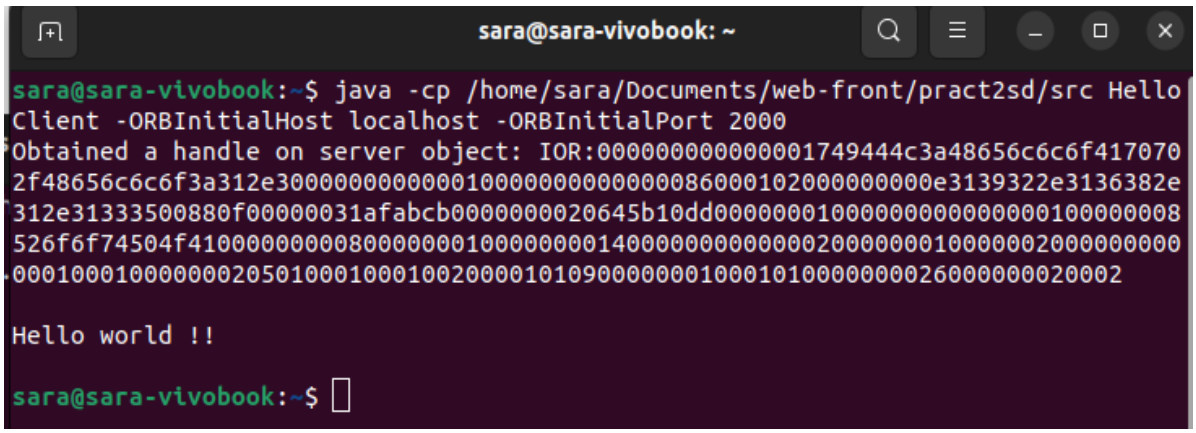
Puerto:



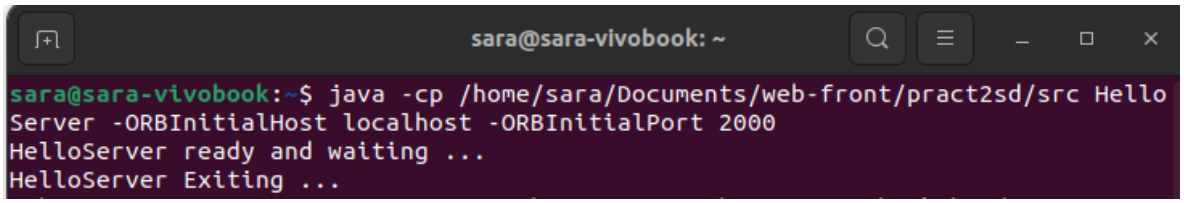
Servidor:



Cliente:



Servidor:



Como se puede observar, el programa funciona como se espera. Se crea en primer lugar el puerto a través del cual se va a realizar la comunicación. A continuación, se inicia el servidor que se queda a la escucha de la petición del cliente. Por último se inicia el cliente que le realiza la solicitud al servidor; este le devuelve el mensaje de Hello world !! y se cierra.

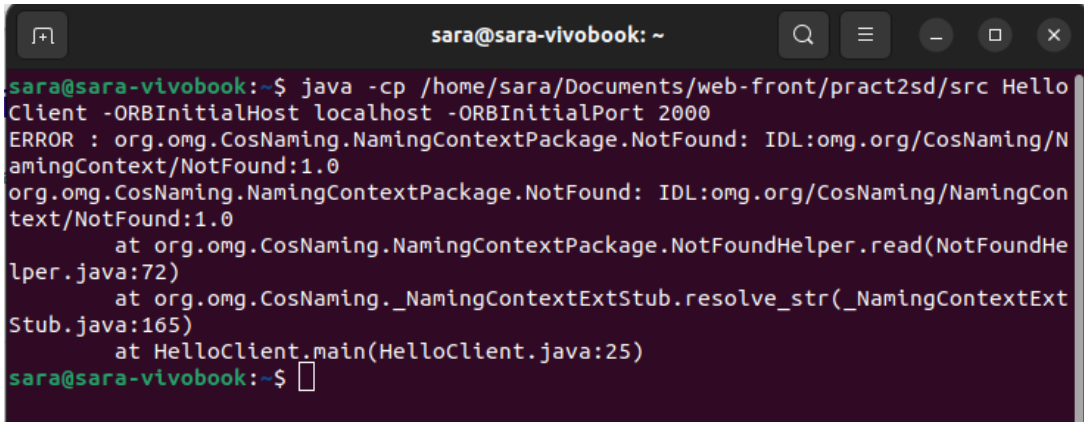
B. Ejercicio 2

Preguntas sobre Hola Mundo en CORBA (puedes añadir capturas) (2 puntos):

a. ¿Qué sucede si lanzo antes el cliente que el servidor?

Si lanzas el cliente CORBA antes que el servidor CORBA, el cliente no podrá establecer una conexión con el servidor. Cuando un cliente CORBA intenta conectarse a un servidor CORBA, necesita conocer la ubicación del servidor y su identificador de objeto. Si el servidor aún no se ha iniciado, el cliente no podrá obtener esta información y no podrá establecer una conexión.

El resultado obtenido al realizar esta acción sería el siguiente:

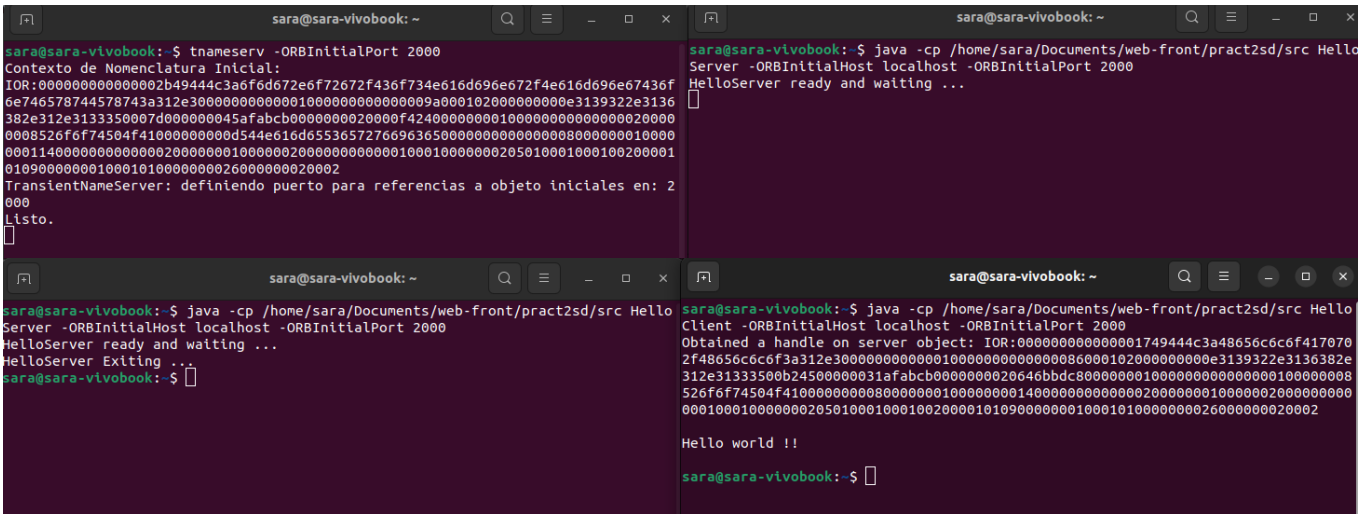


b. ¿Qué sucedería si lanzase varios servidores a la vez y un solo cliente?

Si se lanzasen varios servidores a la vez y un solo cliente, este se conecta solamente a un servidor. Esto se debe a que el cliente necesita conocer la ubicación del servidor y su identificador de objeto para conectarse a él. Si hay varios servidores en ejecución, cada uno tendrá una ubicación y un identificador de objeto diferentes, y el cliente no podrá conectarse a todos ellos simultáneamente. Si no se especifica la referencia la servidor se conecta a cualquiera a través del servicio de nombres.

El resultado obtenido al realizar esta acción sería el siguiente:

Server 1 (No conexión)



Server 2 (Conexión)

c. ¿Puedes conectarte al servidor de un compañero? ¿Cómo lo harías?

Sí, se puede. Para conectarse al servidor de un compañero, se necesita conocer la ubicación y los detalles de conexión del servidor. Si se proporciona esta información, se puede utilizar para conectarse al servidor CORBA utilizando nuestro propio cliente CORBA.

Se necesitará:

- La ubicación del servidor: esto incluirá la dirección IP o el nombre de dominio del servidor y el puerto en el que está escuchando.
- El identificador de objeto del servidor: esto es un identificador único que se utiliza para identificar el objeto remoto proporcionado por el servidor.

Con esta información, se puede crear un objeto de conexión de cliente CORBA en el propio código y utilizarlo para conectarse al servidor de un compañero. En la mayoría de los casos, también deberían asegurarse los protocolos de seguridad; que sean los adecuados antes de realizar la conexión. (Tomando que no hay restricciones de red que impidan la conexión al servidor, como firewalls o configuraciones de red restrictivas.

C. Ejercicio 3:

Actualizar un repositorio de Github con una aplicación Java CORBA (7 puntos).

Se debe 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. Se puede añadir por ejemplo: operar con raíces cuadradas o añadir que utilice decimales.

- **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 como funciona la aplicación.**

El repositorio sobre el cual se han realizado las modificaciones es:
https://github.com/aniket1897/Medical-Store

Consiste en un servidor centralizado de medicamentos, al que varios clientes pueden acceder reservándolos y pagándolos, muy similar al sistema de una farmacia.

Se proporcionan las funciones de:
- check_medicine() → Comprueba la disponibilidad de las medicinas.
- process_medicine ()→ Se realiza el procesamiento y se añade al carrito del número seleccionado del producto
- total_price() → Devuelve el precio total de los productos
- pay_bill() → Se realiza el pago

store.idl:

practica2sdmed > store.idl

```
module MedicalStore{
  interface medintf{
      string check_medicine(in string med_id);
      string process_medicine(in string med_id, in long quantity);
      long total_price();
      string pay_bill();
      };
};
```

Server.java:

practica2sdmed > Server.java > Server > Server()

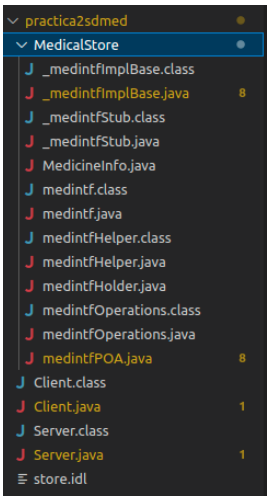
```
16      public Server(){
17          total=0;
18          medicine_list=new HashMap<String,Integer>();
19          prices=new int[10];
20          medicine_list.put("Paracetamol",15);
21          medicine_list.put("Lyrica",20);
22          medicine_list.put("Ibuprofen",20);
23          medicine_list.put("Codiene",20);
24          medicine_list.put("Cymbalta",20);
25          medicine_list.put("Ativan",20);
26          medicine_list.put("Losartan",20);
27          medicine_list.put("Actidone",20);
28          medicine_list.put("Lexapro",20);
29          medicine_list.put("Lyrica",20);
30          prices[0]=90;   prices[1]=150;  prices[2]=70;
31          prices[3]=120;  prices[4]=24;   prices[5]=98;
32          prices[6]=140;  prices[7]=274;  prices[8]=210;
33          prices[9]=50;
34          form[0]="Capsules";     form[1]="Syrup";    form[2]="Tablet";
35          form[3]="Capsules";     form[4]="Tablet";   form[5]="Tablet";
36          form[6]="Tablet";       form[7]="Syrup";    form[8]="Capsules";
37          form[9]="Tablet";
38      }
39
40     // checking for availabilty for medicines
41      public String check_medicine(String med_id){
42          store=-1;
43          int i=0;
44          if(!medicine_list.containsKey(med_id)   )
45              {
46                  return "Medicine not present";
47              }
48        String s="";
49          for(String ik: medicine_list.keySet() ){
50              if( ik.equals(med_id) )
51              {
52                  s="Medicine present with total available quantity: "+ medicine_list.get(med_id) +"\nPrice:"+prices[i];
53                  store=i;
54                  break;
55              }
56              i++;
57          }
58          return s;
59      }
61     // add to cart
62      public String process_medicine(String med_id,int quantity){
63          int q=  medicine_list.get(med_id);
64          String s="";
65          if(q>=quantity){
66              total+= prices[store] * quantity;
67              medicine_list.put( med_id, medicine_list.get(med_id) - quantity );
68              cesta.put(med_id,quantity);
69              System.out.println("Medicine "+med_id+" ordered with quantities "+quantity);
70              s= "Medicine "+med_id+" with " + quantity+ " added to cart";
71          }
72        return s;
73      }
74
75      //return total prices
76      public int total_price(){
77              return total;
78      }
79      // pay bill
80      public String pay_bill(){
81          System.out.println("Payment successful with amount "+total);
82          total=0;
83          return "Payment successful!!";
84      }
85
```

Client.java:

practica2sdmed > Client.java > Client > main(String[])

```
5   import java.util.*;
6   class Client{
        Run | Debug
7       public static void main(String[] args){
8           try{
9               int i;
10              String j;
11              try (Scanner sc = new Scanner(System.in)) {
12                  String s,b,a,t;
13                  ORB orb=ORB.init(args,null);
14                  org.omg.CORBA.Object objRef =
15                  orb.resolve_initial_references("NameService");
16                  NamingContext ncRef = NamingContextHelper.narrow(objRef);
17                  NameComponent nc=new NameComponent( "Medicines" , "" );
18                  NameComponent path[] = {nc} ;
19                  medintf lbref=medintfHelper.narrow( ncRef.resolve(path) );
20                  do{
21                      int p=0;
22                      System.out.println( ++p +".Check medicines");
23                      if(lbref.total_price() > 0){
24                                  System.out.println(++p +".View cart");
25                                  System.out.println( ++p +".Proceed to Payment");
26                      }
27                      System.out.println(++p +".Exit");
28                      System.out.print("Enter choice: ");
29                      i=sc.nextInt();
30                      switch(i){
31                          case 1:System.out.print("Enter medicine name: ");
32                          j=sc.next();
33                          s=lbref.check_medicine(j);
34                          System.out.println(s);
35                          System.out.println("If you want to  confirm type 'yes' else 'no'");
36                          b=sc.next();        a=b.toLowerCase();
37                          if(a.equals("yes")){
38                              //Informacion de la medicina
39                              System.out.print("Enter number of quantities: ");
40                              int y=sc.nextInt();
41                              s=lbref.process_medicine(j,y);
42                              if(s.equals("")){
43                                  s="Sorry medicines not in stock!!!";
44                              }
45                              System.out.println(s);
46                          }
47                          break;
51                          case 2:
52                              s=lbref.ver_cesta();
53                              System.out.println(s);
54                              System.out.println("Total amount to be paid: "+lbref.total_price());
55                              System. out.println("If you want to pay type 'yes' else 'no' ");
56                              b=sc.next();        a=b.toLowerCase();
57                              if(a.equals("yes")){
58                                  s=lbref.pay_bill();
59                                  System.out.println(s);
60                              }
61                              break;
62
63
64                          case 3:int pay=lbref.total_price();
65                          System.out.println("Total amount to be paid: "+pay);
66                          System.out.println("If you want to pay type 'yes' else 'no' ");
67                              b=sc.next();        a=b.toLowerCase();
68                          if(a.equals("yes")){
69                              s=lbref.pay_bill();
70                              System.out.println(s);
71                          }
72                          break;
73
74                          case 4:
75                              System.out.println("Thank you for shopping with us");
76                              break;
77
78                          default:
79
80                          break;
81                      }
82                  }while(i!=3);
83              }
84          }catch(Exception e){
85              e.printStackTrace();
86          }
87      }
```

En primer lugar se realiza la compilación al igual que se hizo en el ejercicio 1: (Compilando primero el IDL, luego el servidor y por último el cliente)



$ idlj -fall store.idl
$ javac Server.java
$ javac Client.java

Al realizar esta acción se generarán varios archivos (.java y .class) dentro de una carpeta llamada MedicalStore:

Una vez obtenido esto, se realiza la ejecución, teniendo abiertas tres ventanas del Símbolo del sistema.

$ tnameserv -ORBInitialPort 2000

$ java Server -ORBInitialHost localhost -ORBInitialPort 2000

$ java Client -ORBInitialHost localhost -ORBInitialPort 2000

La funcionalidad añadida es la siguiente. A los medicamentos se les ha añadido la propiedad de la forma farmacéutica (las diferentes maneras o formas en que encontramos los medicamentos, si es en cápsulas, comprimidos, jarabes, inyectables, supositorios, etc.), así como un método a través del cual se puede acceder a los medicamentos que se han añadido al carrito con sus respectivas cantidades.

Se han añadido las siguientes funciones en el Server.java, llamándolas en el store.idl

```
//return form of medicine
public String form_medicine(String med_id){
    String s="";
    int i=0;
    for(String ik: medicine_list.keySet() ){
        if( ik.equals(med_id) )
        {
            s="Form of medicine: "+ form[i];
            break;
        }
        i++;
    }
    return s;
}
//return medicines in cart
public String ver_cesta(){
    String s="";
    for(String ik: cesta.keySet() ){
        s+="Medicine: "+ ik + " Quantity: "+cesta.get(ik)+"\n";
    }
    return s;

}
```

```
module MedicalStore{
  interface medintf{
      string check_medicine(in string med_id);
      string process_medicine(in string med_id, in long quantity);
      string form_medicine(in string med_id);
      string ver_cesta();
      long total_price();
      string pay_bill();
      };
};
```

En el Client.java se ha implementado también alguna modificación para gestionar las opciones que puede elegir el usuario.
Por último, en los archivos de la carpeta MedicalStore también se han modificado algunos archivos tales como:

medintfPOA.java, _medintfImplBase.java

```
case 3:  // MedicalStore/medintf/ver_cesta
{
  String $result = null;
  $result = this.ver_cesta ();
  out = $rh.createReply();
  out.write_string ($result);
  break;
}

case 4:  // MedicalStore/medintf/total_price
{
  int $result = (int)0;
  $result = this.total_price ();
  out = $rh.createReply();
  out.write_long ($result);
  break;
}
```

medintfstub.java

```
public String form_medicine (String med_id)
{
        org.omg.CORBA.portable.InputStream $in = null;
        try {
            org.omg.CORBA.portable.OutputStream $out = _request ("form_medicine", true
            $out.write_string (med_id);
            $in = _invoke ($out);
            String $result = $in.read_string ();
            return $result;
        } catch (org.omg.CORBA.portable.ApplicationException $ex) {
            $in = $ex.getInputStream ();
            String _id = $ex.getId ();
            throw new org.omg.CORBA.MARSHAL (_id);
        } catch (org.omg.CORBA.portable.RemarshalException $rm) {
            return form_medicine (med_id        );
        } finally {
            _releaseReply ($in);
        }
} // form_medicine

public String ver_cesta ()
{
        org.omg.CORBA.portable.InputStream $in = null;
        try {
            org.omg.CORBA.portable.OutputStream $out = _request ("ver_cesta", true);
            $in = _invoke ($out);
            String $result = $in.read_string ();
            return $result;
        } catch (org.omg.CORBA.portable.ApplicationException $ex) {
            $in = $ex.getInputStream ();
            String _id = $ex.getId ();
            throw new org.omg.CORBA.MARSHAL (_id);
        } catch (org.omg.CORBA.portable.RemarshalException $rm) {
            return ver_cesta (        );
        } finally {
            _releaseReply ($in);
        }
} // ver_cesta
```

medintfOperations.java

```
/**
* MedicalStore/medintfOperations.java .
* Generated by the IDL-to-Java compiler (portable), versi
* from store.idl
* martes 28 de febrero de 2023 23H43' CET
*/

public interface medintfOperations
{
  String check_medicine (String med_id);
  String process_medicine (String med_id, int quantity);
  String form_medicine (String med_id);
  String ver_cesta ();
  int total_price ();
  String pay_bill ();
} // interface medintfOperations
```

Al ejecutar de nuevo, se obtiene lo siguiente:

```
sara@sara-vivobook: ~
sara@sara-vivobook:~$ java -cp /home/sara/Documents/web-front/practica2sdmed Cli
ent -ORBInitialHost localhost -ORBInitialPort 2000
1.Check medicines
2.Exit
Enter choice: 1
Enter medicine name: Paracetamol
Medicine present with total available quantity: 15
Price:70
Form of medicine: Tablet
If you want to  confirm type 'yes' else 'no'
yes
Enter number of quantities: 3
Medicine Paracetamol with 3added to cart
1.Check medicines
2.View cart
3.Proceed to Payment
4.Exit
Enter choice: 2
Medicine: Paracetamol Quantity: 3

Total amount to be paid: 210
If you want to pay type 'yes' else 'no'
yes
Payment successful!!
```

```
sara@sara-vivobook: ~
sara@sara-vivobook:~$ java -cp /home/sara/Documents/web-front/practica2sdmed Ser
ver -ORBInitialHost localhost -ORBInitialPort 2000
Server started!!
Medicine Paracetamol ordered with quantities 3
Payment successful with amount 210
```

Link al repositorio: https://github.com/saramarcoss/Medical-Store

V. Conclusiones:

En esta práctica se ha podido profundizar acerca del uso de CORBA, demostrando cómo se puede definir una interfaz común para el objeto de archivo y cómo se puede utilizar el ORB (Object Request Broker) para facilitar la comunicación entre objetos de diferentes sistemas.

Además, se han abordado temas como la implementación de un servidor CORBA, la creación de clientes CORBA y el manejo de excepciones. También se ha discutido el uso de herramientas de desarrollo para CORBA, como el compilador de IDL (Interface Definition Language) y el generador de stubs.

En resumen, esta práctica ha demostrado cómo CORBA puede ser utilizado para desarrollar sistemas distribuidos complejos. CORBA proporciona una solución de middleware efectiva para la comunicación entre objetos y se ha demostrado ser útil para la creación de sistemas independientes del lenguaje y de la plataforma.

VI. Bibliografía:

https://docs.oracle.com/cd/A97335_02/apps.102/a83722/corba1.htm
https://es.wikipedia.org/wiki/CORBA
http://www.jtech.ua.es/j2ee/2002-2003/modulos/idl-corba/apuntes/tema2.htm
https://docs.oracle.com/javase/7/docs/technotes/guides/idl/jidlExample.html