Programación de Sistemas Distribuidos Práctica 2: Aplicación usando CORBA

Aítuío Requejo Poítilla

UNIVERSIDAD NEBRIJA

Tabla de contenido

I. Enunciados

A. Ejercicio 1

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
```

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?
b) ¿Qué sucedería si lanzase varios servidores a la vez y un solo cliente?
c) ¿Puedes conectarte al servidor de un compañero? ¿Cómo lo harías?

C. Ejercicio 3

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 diferénciate.

II. Introducción

El objetivo de esta práctica es establecer una comunicación entre cliente y servidor mediante la utilización de CORBA. CORBA significa Common Object Request Broker Architecture siendo un estándar creado por OMG que permite el funcionamiento conjunto de componentes de software escritos en distintos lenguajes de programación ejecutándose en distintos sistemas. Permite distribuir objetos a través de redes de modo que las operaciones en dichos objetos puedan llamarse de forma remota.

CORBA incluye 4 componentes principales:

- ORB -> Maneja la comunicación, ordenación y desordenación de parámetros, siendo el manejo de parámetros transparente para aplicaciones de cliente y servidor CORBA.
- Servidor CORBA -> Crea los objetos CORBA y los inicializa con ORB. Los clientes pueden acceder a las referencias.
- Servicio de nombres -> Mantiene referencias a objetos CORBA.
- Nodo CORBARequest -> Actúa como cliente dentro de CORBA.

Por último, cabe desatacar lo que es un archivo IDL. IDL es un lenguaje declarativo para la definición de interfaces de CORBA. Permite a los programadores y los usuarios de los objetos estar seguros de que se invoca a la operación correcta del objeto correcto, aunque la única información adicional que se necesita es la referencia al objeto.

III. Resultados

Para poder realizar el correcto funcionamiento de la comunicación CORBA probé a ejecutar el comando idlj -fall Hello.idl, al ejecutarlo me saltó un error que decía que no tenía el open jdk instalado y para instalarlo bastaba con escribir en la terminal de LINUX el comando:

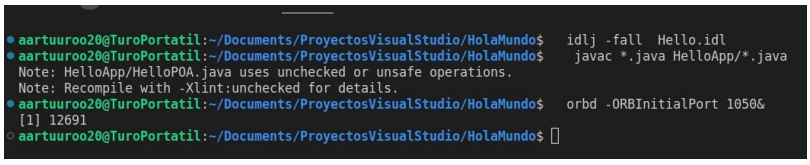
sudo apt install openjdk-8-jdk-headless

Este comando permite la instalación de OpenJDK en la versión 8, al estar investigando esta versión es más antigua y no se puede instalar una versión más reciente debido a que se dejó de incluir CORBA en versiones posteriores.
Una vez instalado me dejo ejecutar con total normalidad todo el proceso que se va a mostrar a continuación.

A. Ejercicio 1

Para este ejercicio debemos realizar una comunicación simple entre servidor y cliente mediante CORBA e imprimir Hello World. Para ello primero hay que entender los archivos que genera CORBA y en qué orden ejecutarlos. Hay una serie de comandos para poder ejecutar y compilar el código, abriendo 3 terminales (Comandos solo validos en sistema operativo LINUX) en visual studio empezamos poniendo lo siguiente:

1) Terminal
   a) idlj -fall Hello.idl -> Compila el archivo Hello.idl cargando las distintas funciones que añadamos ahí, también genera una serie de archivos que proporcionan funcionalidades de comunicación.
   b) javac *.java HelloApp/*.java -> Compila los archivos .java que se han generado al ejecutar el comando previo.
   c) orbd -ORBInitialPort 1050& -> Permite abrir un puerto para iniciar la comunicación (si no se abre o especifica el puerto la comunicación será imposible).



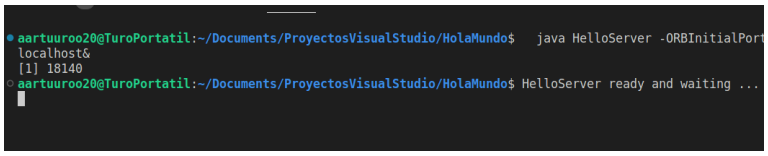
En cuanto a los ficheros generados por el compilador IDL son siempre los mismos, estos archivos son los siguientes:

- Hello.java -> Este archivo es una interfaz vacía que extiende a 2 clases bases para objetos CORBA y entidades IDL, así como la interfaz HelloOperations.
- HelloOperations.java -> Este archivo define 2 métodos Java sayhello(), que devuelve un string y shutdown(), este archivo contendrá las signaturas reales de las operaciones. También está muy ligado al archivo Hello.java donde se tiene que realizar una división de trabajo ya que en algunos casos no es necesario que algunas operaciones no extiendan de org.omg.CORBA.object.
- HelloPOA.java -> Contiene un código esqueleto usado por POA que sirve una base para la clase que tiene el código que implementa las operaciones de la interfaz. En este archivo realizaremos las invocaciones a los métodos creados.
- HelloHolder.java -> Proporciona soporte para manejar los parámetros de entrada/salida otra interfaz.
- HelloHelper.java -> Proporciona una serie de métodos útiles para los objetos Hello, por ejemplo narrow() el cual convierte referencias a objetos CORBA al tipo Hello.
- _HelloStub.java -> Permite crear un proxy en el lado del cliente que permite la implementación de la interfaz Hello en el lado del cliente.

Estos archivos se van a generar siempre que compilemos el archivo IDL donde siempre van a ser los mismos, si estos archivos ya están generados solo se actualizarán con los cambios ya realizados.

2) Terminal

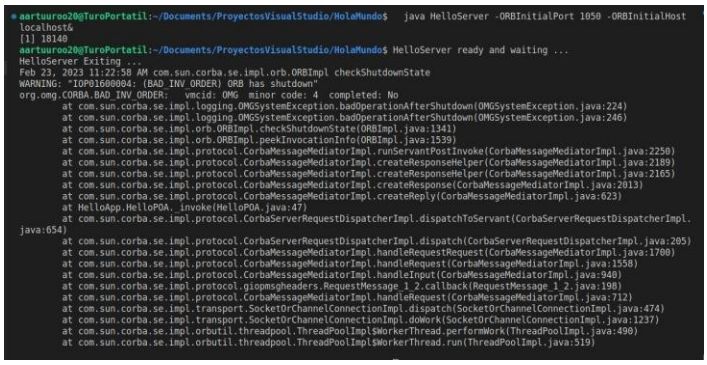
java HelloServer -ORBInitialPort 1050 -ORBInitialHost localhost& -> Inicia el servidor en el puerto abierto y lo mantiene a la espera.



3) Terminal

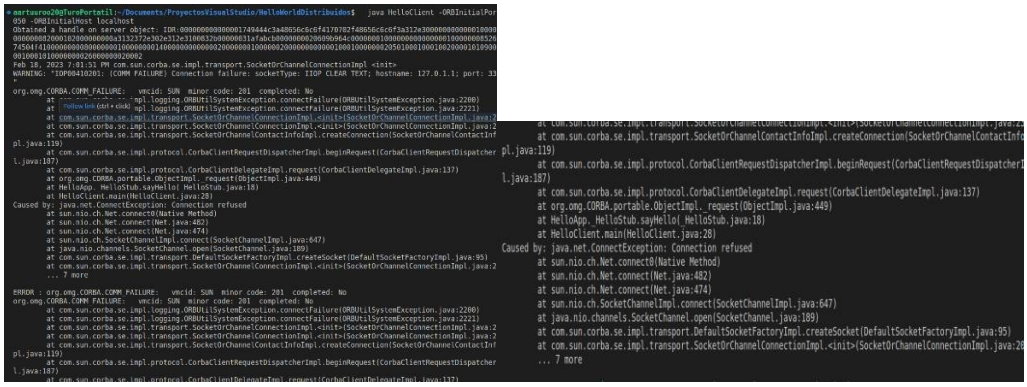
java HelloClient -ORBInitialPort 1050 -ORBInitialHost localhost -> Inicia el cliente y recibe la información por parte del servidor a través de mismo puerto (Si el puerto no coincide no es posible la comunicación).

Una vez ejecutado el cliente el servidor cierra la comunicación:

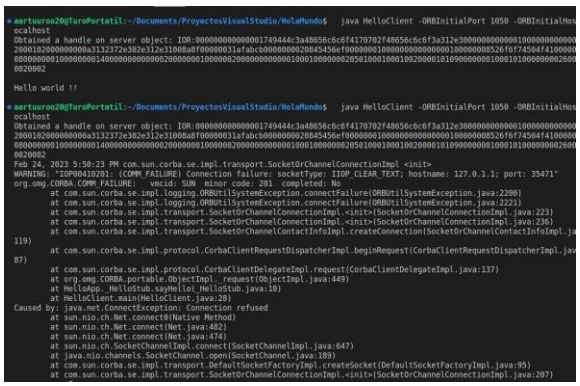


B. Ejercicio 2

a) Al ejecutar antes el cliente que el servidor se produce un error donde el cliente se detiene y lanza una excepción de conexión fallida con el servidor al no estar iniciado y no poder obtener sus métodos. Además, el cliente también necesita tener alguna referencia del objeto CORBA para poder realizar las operaciones deseadas. Si el servidor no se ha iniciado el cliente no será capaz de acceder a la referencia al no encontrarla.



b) Realicé la prueba manteniendo 3 servidores abiertos, al iniciar el primer cliente no ocurría nada extraño, el ultimo servidor abierto le devolvía al cliente la información deseada, el cliente lo recibía, lo imprimía y el servidor se cerraba, manteniendo así los otros servidores restantes abiertos. Sin embargo, cuando trataba de iniciar un nuevo cliente me saltaba el siguiente error:



Esto indica que, debido al servicio de nombres, el cual proporciona una forma de encontrar objetos mediante nombres, no es capaz de acceder al objeto debido a que hay varios con el mismo identificador registrado.

c) Para poder realizar una conexión entre dos equipos independientes primero una máquina debe actuar como servidor y otra como cliente.

- En la máquina servidor debemos ejecutar el siguiente comando:
  java HelloServer -ORBInitialPort 1050 -ORBInitialHost localhost&
  Cambiando los parámetros de -ORBInitialPort por un puerto abierto en el propio equipo de la máquina servidor, también habría que cambiar el parámetro -ORBInitialHost por la propia IP del equipo.
- Posteriormente en la máquina que actuase como cliente se debe ejecutar el comando:
  java HelloClient -ORBInitialPort 1050 -ORBInitialHost localhost
  Debemos modificar los mismos parámetros que en el servidor, en el -ORBInitialPort debemos establecer el mismo puerto establecido en el servidor y en el parámetro -ORBInitialHost escribir la dirección del equipo.

C. Ejercicio 3

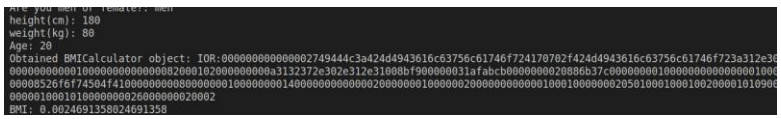
- Enlace al repositorio -> https://github.com/aartuuroo20/BMICalculator_CORBA_Practica2SSD
- Enlace al repositorio original -> https://github.com/hyuko21/BMICalculator_CORBA

Para realizar este ejercicio:

1. Realicé un fork sobre una calculadora de IMC (índice de masa corporal).
2. Una vez echo el fork probé a ejecutar la aplicación mediante los comandos escritos en el README.txt.
3. Después de ejecutar todos los comandos empecé a realizar las modificaciones a la aplicación.

Este proyecto funciona de tal forma que el cliente al introducir su peso y altura y le envía estos datos al servidor, donde este devuelve el IMC, todo esto se realiza mediante comunicación CORBA y la implementación de una función calculateBMI.

Resultados de la aplicación sin modificación:



Las modificaciones realizadas han sido 3:

1. La primera fue realizar un cálculo del TMB (metabolismo basal) tanto para hombres como para mujeres.
2. La segunda fue realizar una adaptación en el cliente y en el servidor para así una vez realizados los cálculos al igual que el cliente, el servidor se cierre.
3. La última fue unas adaptaciones en el cliente donde habiendo creado dos hashMaps, estos almacenen comidas con sus respectivas calorías para así identificar si poder subir o bajar de peso.

Para calcular el TMB empecé escribiendo en el archivo IDL la función con sus parámetros de entrada, el nombre y el tipo de dato devuelto.

```
module BMICalculatorApp {
    interface BMICalculator {
        double calculateBMI(in double height, in double weight);
        double calculateTMB(in double height, in double weight, in double age);
        double calculateTMBf(in double height, in double weight, in double age);
        oneway void shutdown();
    };
};
```

Para compilar el archivo y poder realizar las funciones ejecuté el comando:

idlj -fall bmicalculator.idl

Una vez realizados estos cambios vamos a los archivos java en la carpeta BMICalculatorApp. En con- creto empezamos abriendo el archivo de operaciones y escribimos las distintas operaciones nuevas que queremos implementar, donde los tipos de datos y nombre tienen que coincidir con los del ar- chivo IDL.

```
public interface BMICalculatorOperations
{
  double calculateBMI (double height, double weight);
  double calculateTMB (double height, double weight, double age);
  double calculateTMBf (double height, double weight, double age);
  void shutdown ();
} // interface BMICalculatorOperations
```

Una vez hayamos escrito estos cambios aparecerán una serie de errores en otros archivos de la misma carpeta diciendo que faltan métodos por implementar, en concreto los archivos de POA y Stub. En el archivo Stub escribimos las funciones nuevas a implementar al igual que en el archivo POA donde inicializamos en el constructor las funciones nuevas y las propias opciones donde se eje- cutarán.

```
private static java.util.Hashtable _methods = new java.util.Hashtable ();
static
{
  _methods.put ("calculateBMI", new java.lang.Integer (0));
  _methods.put ("calculateTMB", new java.lang.Integer (1));
  _methods.put ("calculateTMBf", new java.lang.Integer (2));
  _methods.put ("shutdown", new java.lang.Integer (3));
```

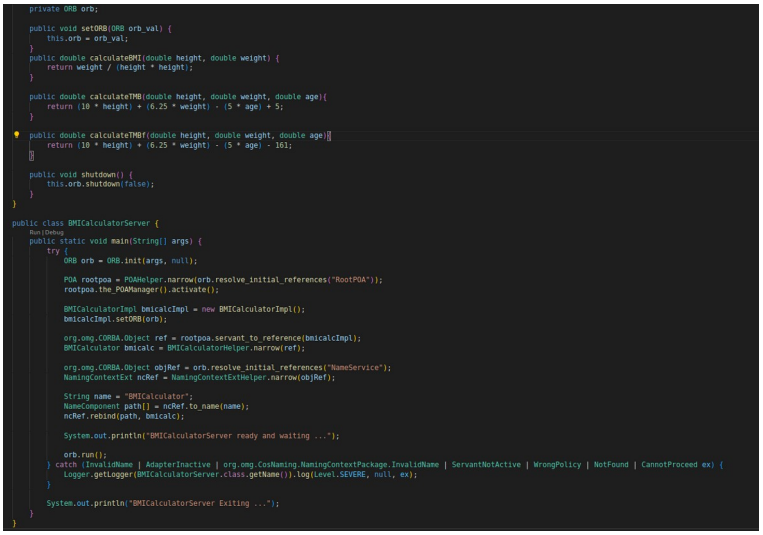
```
case 1:  // BMICalculatorApp/BMICalculator/calculateTMB
{
  double height = in.read_double ();
  double weight = in.read_double ();
  double age = in.read_double ();
  double $result = (double)0;
  $result = this.calculateTMB (height, weight, age);
  out = $rh.createReply();
  out.write_double ($result);
  break;
}

case 2:  // BMICalculatorApp/BMICalculator/calculateTMBf
{
  double height = in.read_double ();
  double weight = in.read_double ();
  double age = in.read_double ();
  double $result = (double)0;
  $result = this.calculateTMBf (height, weight, age);
  out = $rh.createReply();
  out.write_double ($result);
  break;
}

case 3:  // BMICalculatorApp/BMICalculator/shutdown
{
  this.shutdown ();
  out = $rh.createReply();
  break;
```

Una vez realizado estos cambios debemos compilar los archivos java para que así nos reconozcan las nuevas funciones implementadas, para ello lo compilamos mediante el comando:

javac bmicalculator/*.java BMICalculatorApp/*.java

Posteriormente en el propio servidor implementamos las funciones y ya el cliente las llama y hace uso de ellas.
El segundo cambio realizado sigue el mismo concepto y procedimiento de implementación.

Cambios realizados en el servidor:

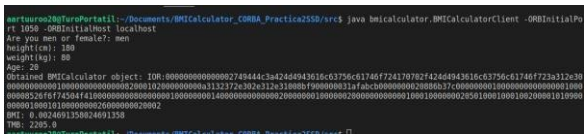


```
    private ORB orb;

    public void setORB(ORB orb_val) {
        this.orb = orb_val;
    }
    public double calculateBMI(double height, double weight) {
        return weight / (height * height);
    }

    public double calculateTMB(double height, double weight, double age){
        return (10 * height) + (6.25 * weight) - (5 * age) + 5;
    }

    public double calculateTMBf(double height, double weight, double age){
        return (10 * height) + (6.25 * weight) - (5 * age) - 161;
    }

    public void shutdown() {
        this.orb.shutdown(false);
    }
}

public class BMICalculatorServer {
    Run | Debug
    public static void main(String[] args) {
        try {
            ORB orb = ORB.init(args, null);

            POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
            rootpoa.the_POAManager().activate();

            BMICalculatorImpl bmicalcImpl = new BMICalculatorImpl();
            bmicalcImpl.setORB(orb);

            org.omg.CORBA.Object ref = rootpoa.servant_to_reference(bmicalcImpl);
            BMICalculator bmicalc = BMICalculatorHelper.narrow(ref);

            org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
            NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

            String name = "BMICalculator";
            NameComponent path[] = ncRef.to_name(name);
            ncRef.rebind(path, bmicalc);

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

            orb.run();
        } catch (InvalidName | AdapterInactive | org.omg.CosNaming.NamingContextPackage.InvalidName | ServantNotActive | WrongPolicy | NotFound | CannotProceed ex) {
            Logger.getLogger(BMICalculatorServer.class.getName()).log(Level.SEVERE, null, ex);
        }

        System.out.println("BMICalculatorServer Exiting ...");
    }
}
```

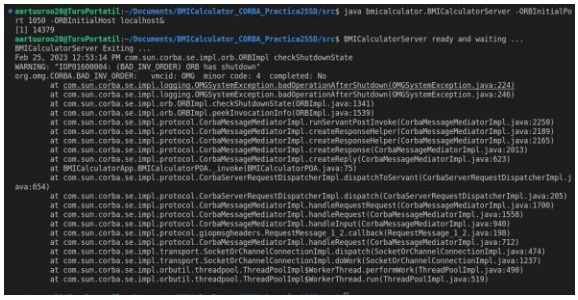
Cambios realizados en el cliente:

```
public class BMICalculatorClient {
    static BMICalculator bmicalcImpl;

    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 = NamingContextExtHelper.narrow(objRef);

            String name = "BMICalculator";
            bmicalcImpl = BMICalculatorHelper.narrow(ncRef.resolve_str(name));

            Scanner in = new Scanner(System.in);

            System.out.print("Are you man or female?: ");
            String gender = in.nextLine();


            System.out.print("height(cm): ");
            double height = in.nextDouble();

            System.out.print("weight(kg): ");
            double weight = in.nextDouble();

            System.out.print("Age: ");
            double age = in.nextInt();

            System.out.println("Obtained BMICalculator object: " + bmicalcImpl);
            System.out.println("BMI: " + bmicalcImpl.calculateBMI(height, weight));

            HashMap<String, Integer> listaComidasSubirPeso = new HashMap<>();
            listaComidasSubirPeso.put("Primera comida", 1000);
            listaComidasSubirPeso.put("Segunda comida", 1200);
            listaComidasSubirPeso.put("Tercera comida", 1000);
            listaComidasSubirPeso.put("Cuarta comida", 1200);
            int totalCalorias = 0;
            for (int calorias : listaComidasSubirPeso.values()) {
                totalCalorias += calorias;
            }

            HashMap<String, Integer> listaComidasBajarPeso = new HashMap<>();
            listaComidasBajarPeso.put("Primera comida", 500);
            listaComidasBajarPeso.put("Segunda comida", 500);
            listaComidasBajarPeso.put("Tercera comida", 500);
            int totalCaloriasBajarPeso = 0;
            for (int calorias : listaComidasBajarPeso.values()) {
                totalCaloriasBajarPeso += calorias;
            }

            if(gender == "female"){
                double tmb = bmicalcImpl.calculateTMBf(height, weight, age);
                System.out.println("Your TMB is: " + tmb);

                System.out.println("If you want to gain weight, you need to eat more,  an example of what you should eat is: ");
                System.out.println(listaComidasSubirPeso.keySet() + " " + listaComidasSubirPeso.values() + " calories, " + "total calories: " + totalCalorias + " calories");

                System.out.println("If you want to lose weight, you need to eat less,  an example of what you should eat is: ");
                System.out.println(listaComidasBajarPeso.keySet() + " " + listaComidasBajarPeso.values() + " calories, " + "total calories: " + totalCaloriasBajarPeso + " calories");

                System.out.println("If you want to mantain weight, you are eating the right amount of calories.");

            }else{
                double tmb = bmicalcImpl.calculateTMB(height, weight, age);
                System.out.println("Your TMB is: " + tmb);

                System.out.println("If you want to gain weight, you need to eat more,  an example of what you should eat is: ");
                System.out.println(listaComidasSubirPeso.keySet() + " " + listaComidasSubirPeso.values() + " calories, " + "total calories: " + totalCalorias + " calories");

                System.out.println("If you want to lose weight, you need to eat less,  an example of what you should eat is: ");
                System.out.println(listaComidasBajarPeso.keySet() + " " + listaComidasBajarPeso.values() + " calories, " + "total calories: " + totalCaloriasBajarPeso + " calories");

                System.out.println("If you want to mantain weight, you are eating the right amount of calories.");
            }

            bmicalcImpl.shutdown();
        } catch (org.omg.CORBA.ORBPackage.InvalidName | NotFound | CannotProceed | InvalidName ex) {
            Logger.getLogger(BMICalculatorClient.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

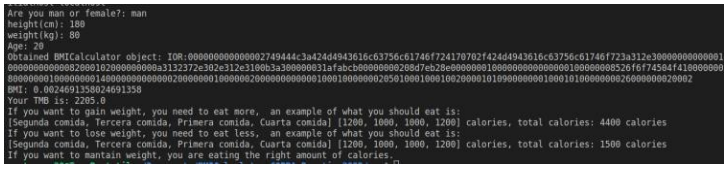
Resultados cliente:



Resultados Servidor:



Para la última modificación como bien mencione antes realice la creación de 2 HashMaps que almacenan información de comidas con sus respectivas calorías. Esta funcionalidad informa al cliente o consumidor de la comunicación tener la información necesaria para saber si quiere subir de peso, bajar de peso o mantenerse en el peso en el que se encuentra.



IV. Conclusiones

En esta segunda practica he aprendido, lo que es CORBA, como funciona, como poder realizar la instalación de la librería OpenJDK en un sistema operativo Linux, como poder implementarla y llevarlo a cabo en un proyecto, véase en la realización del ejercicio 1 y el ejercicio 3. Al haber realizado el ejercicio 3 he podido profundizar más en el conocimiento del metabolismo de las personas y su IMC correspondiente. También aprendí que CORBA está obsoleta porque para poder hacer uso de este tipo de comunicación requiere de volver a versiones más antiguas de Java quedando así desactualizada y utilizándose cada vez menos.

V. Bibliografía

- CORBA - https://www.ibm.com/docs/es/integration-bus/10.0?topic=corba-common- object-request-broker-architecture
- Arquitectura CORBA - http://www.jtech.ua.es/j2ee/2002-2003/modulos/idl-corba/apuntes/tema1.htm
- Java IDL: The "Hello World" Example - https://docs.oracle.com/javase/7/docs/technotes/guides/idl/jidlExample.html
- CORBA Object - https://www.geeksforgeeks.org/client-server-software-development- introduction-to-common-object-request-broker-architecture-corba/
- Servicio de nombres - http://www.jtech.ua.es/j2ee/2002-2003/modulos/idl-corba/apuntes/tema5.htm