Tutorial 2 - CORBA Hello World

IDL

An interface definition language (IDL), is a specification language used to describe a software component's application programming interface (API). IDLs describe an interface in a language-independent way, enabling communication between software components that do not share one language. For example, between those written in C++ and those written in Java.

IDLs provide a contract detailing the operations, parameters and type information that passes between a client and server objects. The IDL provides the language constructs for all concepts of the underlying object model and can incorporate exceptions, definitions of constructed types, and parameters of operations.

The Object Management Group (OMG) is a computer industry consortium that have created middleware standards based on the Common Object Request Broker Architecture (CORBA). OMG are located at `www.omg.org`. One of the standards that they have created is the IDL to Java Mapping Specification. The standard can be retrieved at: `http://www.omg.org/spec/I2JAV/1.3/`, this describes how the CORBA IDL is mapped to the language features of Java. Section 4 contains the primary material regarding the mapping.

Take a look at the standard and verify the mapping from IDL to Java of the following constructs as shown in Table 1

Table 1: CORBA IDL to Java Mapping Highlights

IDL	**Java**
module	package
boolean	boolean
char	char
string	java.lang.String
short	short
long	int

1 My first IDL

When you create your own workspace, you will need to install javac (Java Compiler), instructions to assist with this are to found at `https://www.digitalocean.com/community/tutorials/how-to-install-java-on-ubuntu-with-apt-get`

A basic IDL file should consist of a module, interface and an operation as shown in Listing 1

Listing 1: Hello World in Corba IDL

```
module HelloApp {
        interface Hello {
                string sayHello();
        };
};
```

Type the above code to a file called hello.idl. Provided that the JDK bin folder is in your PATH environment variable you will be able to then execute Listing 2 from the same folder as where you have placed hello.idl. Please note that the $ symbol means that what follows is a command to execute in the terminal / command line.

Listing 2: idlj command

```
$ idlj -fall -oldImplBase hello.idl
```

This will generate the Client and Server stubs as proxies for the server and client in a folder called HelloApp. The generated code implements the presentation and session layers.

Implementing the server

Create a server called HelloServer.java and place the code from Listing 3 therein.

Listing 3: Server code

```
// The package containing our stubs.
import HelloApp.*;
// All CORBA applications need these classes.
import org.omg.CORBA.*;
// needed for output to the file system.
import java.io.*;

public class HelloServer {

  public static void main(String args[]) {
    try {
```

```
      //Initialize the ORB
      ORB orb = ORB.init(args, null);
      //Instantiate the HelloServant on the server
      HelloServant helloRef = new HelloServant();
      //Connect the HelloServant to the orb
      orb.connect(helloRef);
      //Store an object Reference to the HelloServant in a
      //String format
      String ior = orb.object_to_string(helloRef);
      //Write the object reference to the helloServant to a
      //file called HelloIOR
      FileOutputStream fos = new FileOutputStream("HelloIOR");
      PrintStream ps = new PrintStream(fos);
      ps.print(ior);
      ps.close();
      //Run the orb so that it waits for requests from the
      //client
      orb.run();
    }
    catch(Exception e) {
      System.err.println("ERROR: " + e);
      e.printStackTrace(System.out);
    }
  }
}
```

Implementing the servant

Create a servant called HelloServant.java and insert the code from Listing 4

Listing 4: Servant code

```
import HelloApp.*;

//Servant must inherit the generated code
class HelloServant extends _HelloImplBase {
  // Add the sayHello method here in the next step.

  public String sayHello() {
  // Add the method implementation here in the next step.
  return "\nHello World!!\n";
  }
}
```

Implementing the client

Create the client called HelloClient.java and input the code from Listing 5

Listing 5: Client code

```
import HelloApp.*; // The package containing generated stubs.
import org.omg.CORBA.*; // All CORBA
// needed for output to the file system.
import java.io.*;

public class HelloClient {

  // Add the main method here in the next step.

  public static void main(String args[]) {
    // Put the try-catch block here in the next step.
    try { // Add the rest of the HelloStringifiedClient code
       ↪ here.

      //Initialize the ORB
      ORB orb = ORB.init(args, null);
      //Read the object Reference for the HelloServant
      BufferedReader br = new BufferedReader(new FileReader("
         ↪ HelloIOR"));
      String ior = br.readLine();
      //Convert the string object reference to an object
      org.omg.CORBA.Object obj = orb.string_to_object(ior);
      //Convert the object to the correct type i.e. Hello
      Hello helloRef = HelloHelper.narrow(obj);
      //Call theoperation on the servant
      String hello = helloRef.sayHello();
      System.out.println(hello);
    }
    catch(Exception e) {
      System.out.println("ERROR : " + e);
      e.printStackTrace(System.out);
    }
  }
}
```

Finally...

Compile it all, and run the server, then run the client. Modify the server side so that it issues a different reply to the request from the client.

2 Note

The Server is providing the client with access to a language-independent representation of an object reference, called an Interoperable Object Reference (IOR) which is being saved to a file. When we pass an object reference between two dierent CORBA systems, it is passed as an IOR. The IOR contains all the information necessary for a client to access a remote object. You can utilize the IOR parser located at `http://www2.parc.com/istl/projects/ILU/parseIOR/` to see the information contained in the generated IOR.