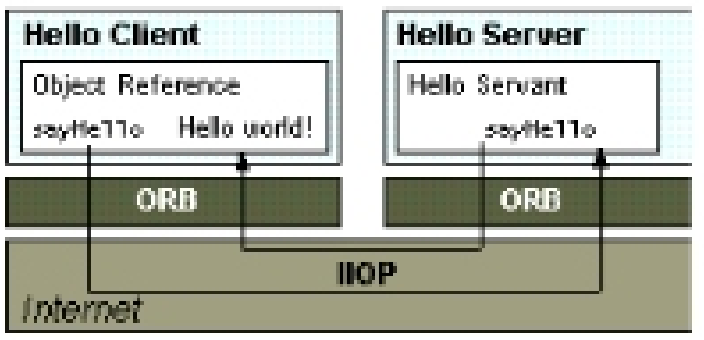
CORBA Example

Description of the example

This example illustrates the basic tasks in building a CORBA distributed application using Java IDL. You will build the classic Hello World program as a distributed application. The Hello World program has a single operation that returns a string to be printed.



1.The client invokes the sayHello method of the HelloServer.
2.The ORB transfers that invocation to the servant object registered for that IDL interface.
3.The servant's sayHello method runs, returning a Java String.
4.The ORB transfers that String back to the client.
5.The client prints the value of the String.

Getting started

You will need two things: *version 1.2 of the JDK* software and the *idltojava* compiler. The JDK provides the API and ORB needed to enable CORBA-based distributed object interaction. The idltojava compiler uses the IDL-to-Java mapping to convert IDL interface definitions to corresponding Java interfaces, classes, and methods, which you can then use to implement your client and server code.

Writing the IDL Interface

In this section, you will write a simple IDL interface for the Hello World program. The IDL interface defines the contract between the client and server parts of your application, specifying what operations and attributes are available. OMG IDL is programming-language independent. You must map it to Java before writing any of the implementation code. (Running idltojava on the IDL file does this for you automatically.) Here's the complete Hello.idl file:

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

OMG IDL is a purely declarative language designed for specifying programming-language-independent operational interfaces for distributed applications. OMG specifies a mapping from IDL to several different programming languages, including C, C++, Smalltalk, COBOL, Ada, and Java. When mapped, each statement in OMG IDL is translated to a corresponding statement in the programming language of choice. You can use the tool *idltojava* to map an IDL interface to Java and implement the client class. When you map the same IDL to C++ and implement the server in that language, the Java client and C++ server interoperate through the ORB as though they were written in the same language.

Steps needed to write the IDL for the Hello World application

Step 1: Declaring the CORBA IDL Module

A CORBA module is a namespace that acts as a container for related interfaces and declarations. It **corresponds closely to a Java package.** Each module statement in an IDL file is mapped to a Java package statement.

```
module HelloApp {
        // Add subsequent lines of code here.
};
```

Step 2: Declaring the Interface

Like Java interfaces, CORBA interfaces declare the API contract that an object has with other objects. **Each interface statement in the IDL maps to a Java interface statement when mapped.**

In your Hello.idl file, enter the interface statement:

```
module HelloApp {
   interface Hello  // Add
   {                // these
                    // four
   };               // lines.
};
```

When you compile the IDL, this statement will generate an interface statement in the Java code. Your client and server classes will implement the Hello interface in different ways.

Step 3: Declaring the Operations

CORBA operations are the behavior that servers promise to perform on behalf of clients that invoke them. **Each operation statement in the IDL generates a corresponding method statement in the generated Java interface.**

In your Hello.idl file, enter the operation statement:

```
module HelloApp {
   interface Hello
   {
      string sayHello();  // Add this line.
   };       // notice the semicolon
};          //notice the semicolon
```

Because our little Hello World application has only a single operation, Hello.idl is now complete.

Mapping Hello.idl from IDL to Java

The tool *idltojava* reads OMG IDL files and creates the required Java files. The *idltojava* defaults are set up so that if you need both client and server files (as you do for our Hello World program), you simply enter the tool name and the name of your IDL file:

1.Go to a command line prompt.
2.Change directory to the location of your Hello.idl file.
3.Enter the compiler command:

idltojava Hello.idl

If you list the contents of the directory, you will see that a directory called HelloApp has been created and that it contains five files. Open Hello.java in your text editor. It looks like this:

```
/* Hello.java as generated by idltojava */
package HelloApp;
public interface Hello
   extends org.omg.CORBA.Object {
   String sayHello();
}
```