Ref. No.: (1)

Date : / /

Assignment Lab DS-2

Aim: Develop distributed application using CORBA.

Obj: Object brokering using CORBA in java.

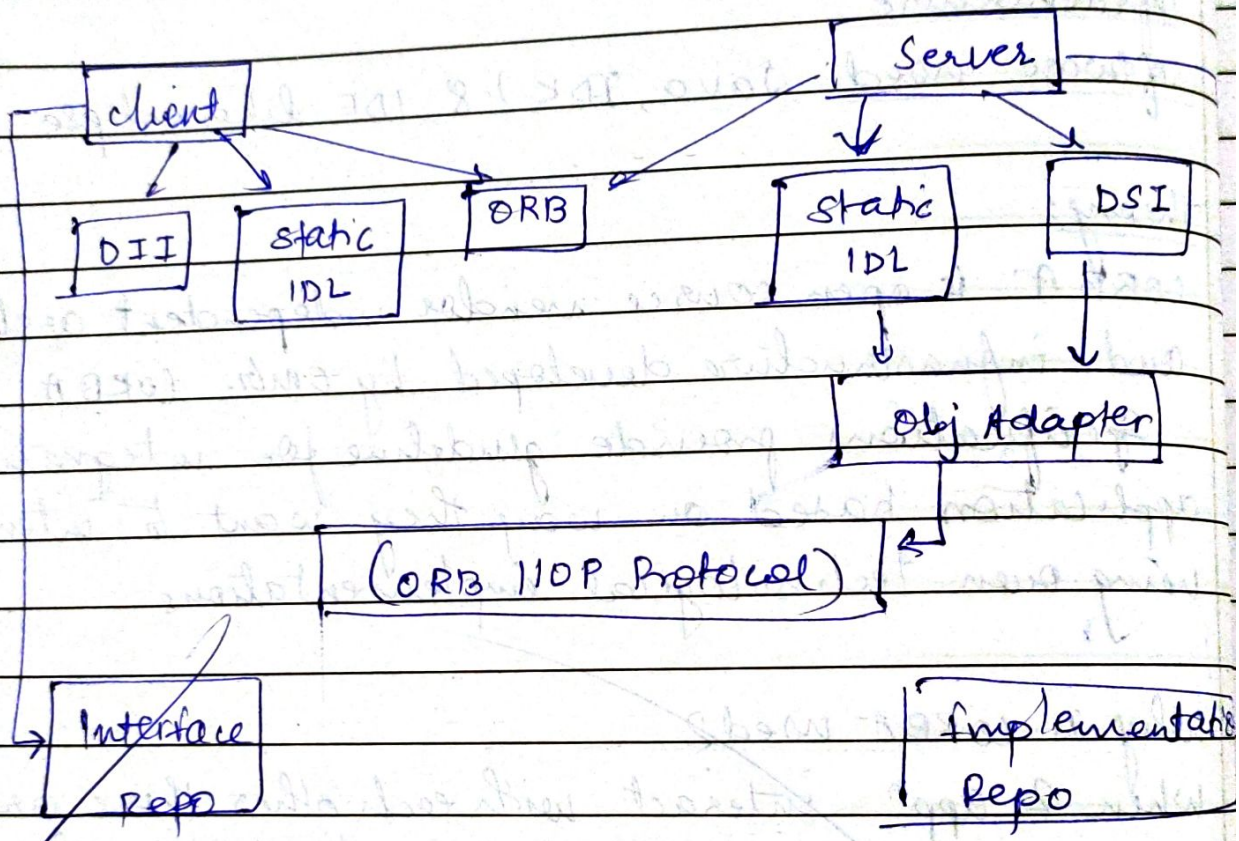
Infrastructure:
Software used: Java, JDK 1.8, IDE like eclipse

Theory:
CORBA: is open source vendor independent architecture and infrastructure developed by OMG. CORBA Specifications provide guideline for integration application based on way they want to interact using own technological implementations

Why is CORBA used?
When 2 app" interact with each other there are unknowns between these applications. They communicate with each other mostly through network address. Due to these they end up with mis-matches.
An app" dev based on CORBA standards with ORB protocol should smoothly integrate and operate with another CORBA vendor.

The IDL interface is a design that works with multiple prog languages like C, C++, Java, Python, Ruby, IDL script. The systems encapsulate actual

implementation along with respective data handl
Hence clients are focussed to develop logic for
IDL interface exposed by application they want t
connect with method parameters.



The client and server stubs and skeletons are as proxies. The stub and skeleton represent client and server, respectively to counterparts. They establish this communication through ORB and pass arguments during invocations.

Java support for CORBA:

CORBA implements Java™ platform by provid distributed object framework, interoperability

with other languages.
CORBA standards provide the proven, interoperable infrastructure to Java platform. IIOP manages communication between Object component powering system.
When using IDL programming model, the interface is everything! It defines points of entry that can be called from remote process, such as types of arguments, parameter of information returned, using IDL, the programmer can make communicating standard language.
With RMI, the interface and implementation language are described in same language, so you don't have to worry about mapping. language level objects can be passed from one process to next. values can be returned by actual type, not declared type. Or you can allow CORBA compliant languages.

- The IDL programming model:
Consists of both Java CORBA ORB and idlj compiles that maps IDL to java bindings that use Java CORBA ORB, as well as set of API's which can be explored by selecting org.omg as prefix from package section of API index.

When you run idlj over idl file, it generates java version of interface as well as class code files.

Ref. No.:

Date : /

* Portable Object Adapter (POA):
  - Allow programmers to construct obj implem portable in diff ORB products
  - Provide support for obj with persistent ident

1. Creating CORBA using Java IDL:

1.1 In order to distribute a Java obj over network using CORBA, one has to owen interface
  - write an interface in CORBA idl
  - Generate base interface, java stub & skeleton using IDL-java compiler
  - writing server side implementation.

1.2 Modules:
Modules are declared in IDL using module key followed by name for module scope.

IDL modules using syntax modulename: x.eg.

```
//IDL
module jen {
module corba {
  interface NeatExample...
};
};
```

1.3 Interfaces
The declaration of interface includes interface

and interface body.

interface PrintServer : Server { ...

1.4 Data members and methods

- The interface body declares all data members and methods of interface.

readonly attribute string myString;
string parseString (in string buffer);

1.5 A complete IDL eg:

module OS {
module Services {
interface Server {
readonly attribute string serverName;
boolean init (in string sName);
};
interface Printable {
boolean print (in string header);
};
interface PrintServer : Server {
boolean printThis (in Printable p);
};
};
};

Ref. No.:

Date : / /

Turning IDL into Java

Once the remote interfaces in IDL are described, you need to generate Java classes that acts as starting point for implementing remote interfaces using IDL to java compiler.

IDL interface:

- A java interface with same name as IDL interface. This acts as basis for java implementation
- A helper class whose name is name of IDL interface with "Helper" appended to it.
- A holder class whose name is name of IDL interface with "Holder" appended to it.

The idltoj generate 2 other classes:

- A client stub class, called _interface-namestub, act as client-side implementation of interface knows how to convert method request into CO requests. The stub class for interface named Ser is called _SeaverStub.
- A Seaver skeleton class, called _interface-na Impl Base. That is Baseclass for service side implementation of interface. The base class con requests for object from ORB. and channel values back.

This creates the five java classes: a Java versi of interface, helper class, holder class, client and Seaver skeleton.

Ref. No.:

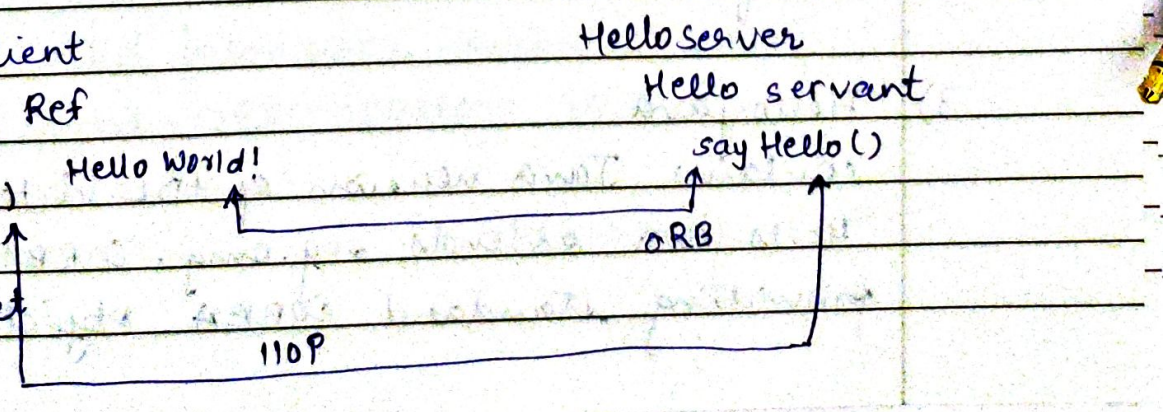
Date : / /

3. Implementing the solution:

1. Defining the interface (Hello.idl)
- The first step to create CORBA app° is to specify objects & their interfaces using OMG's IDL.
To complete the application, you provide HelloServer.java & HelloClient.java.

2. Implement the server (HelloServer.java)
- The eg server consists of two classes, servant & server. The servant, HelloImpl, is implementation of Hello IDL interface; each Hello instance is implemented by HelloImpl instance.

The HelloServer class has the server's main() method which:
- creates and initializes an ORB instance.
- gets a reference to root POA and activates POA manager.
- creates server instance
- Get CORBA obj ref to register new CORBA object.
- Gets root naming context.
- Waits for invocation of new obj from client.



Ref. No.:

Date : /

3. Implementing client Application (HelloClient.)
- The eg app" that follows:
- creates and initializes an ORB
- obtains a reference to root naming context
- looks up "Hello" in naming context & recieves a ref to CORBA obj
- invokes object's sayHello() and shutdown operation and prints results.

→ The files generated by idlj compiler for Hello.idl with -fall command line option are:

1) HelloPOA.java:
This abstract class is stream-based server skeleton, providing basic CORBA functionality for server. It extends org.omg.PortableServer.Servant, implements interface.

2) _HelloStub.java
This class is client stub, providing CORBA functionality for client. It extends org.omg.CORBA.portable.ObjectImpl and implements Hello interface.

3) Hello.java
contains Java version of IDL interface Hello.java extends org.omg.CORBA.Object providing standard CORBA object fun

Ref. No.:

Date : / /

4) Hello Helper.java

- Provides auxillary functionality, notably the narrow() method required to cast CORBA obj ref to proper types. The Helper class is responsible for reading & writing data type to CORBA streams.

5) HelloHolder.java

- Holds public instance member of type Hello whenever IDL type is an out or in parameter, Holder class is used. The holder class delegates to methods in Helper class for reading & writing. It implements org.omg.CORBA.portable.Streamable.

6) HelloOperations.java

- This interface contains the methods sayHello() and shutdown(). The IDL to java puts operation defined on IDL interface which is shared by both stub & skeleton.

The steps are:

1. javac *.java HelloApp/*.java

start orbd

To start orbd from UNIX command shell enter: orbd -ORBInitialPort 1050 &

start Hello Server:

java HelloServer -ORBInitialPort 1050-

Ref. No.:

Date : / /

Run client application:

java HelloClient -ORBInitialPort 1050 -ORBInitialHost localhost

When client is running, you will see response following your terminal.

Hello World! Hello server exiting...

NOTE: After completion kill the name server

→ Conclusion:

CORBA provides transparency; Java provides implementation transparency. CORBA complements Java™ platform by providing distributed object framework. The combination of Java and CORBA allows you to build more scalable and capable applications that can't built using JDK alone.