

# Assignment 2

## Aim

Develop any distributed application using CORBA to demonstrate object brokering.

## Objective

To develop a distributed application using CORBA that demonstrates object brokering by enabling clients to register and discover objects, facilitate message passing between clients and servers, and showcase proper exception handling and communication.

## Theory

When two applications/systems in a distributed environment interact with each other, there are quite a few unknowns between those applications/systems, including the technology they are developed in (such as Java/ PHP/ .NET), the base operating system they are running on (such as Windows/Linux), or system configuration (such as memory allocation). They communicate mostly with the help of each other's network address or through a naming service. Due to this, these applications end up with quite a few issues in integration, including content (message) mapping mismatches. An application developed based on CORBA standards with standard Internet Inter-ORB Protocol (IIOP), irrespective of the vendor that develops it, should be able to smoothly integrate and operate with another application developed based on CORBA standards through the same or different vendor.

### ***Common Object Request Broker Architecture (CORBA):***

CORBA is an acronym for Common Object Request Broker Architecture. It is an open source, vendor-independent architecture and infrastructure developed by the Object Management Group (OMG) to integrate enterprise applications across a distributed network. CORBA specifications provide guidelines for such integration applications, based on the way they want to interact, irrespective of the technology; hence, all kinds of technologies can implement these standards using their own technical implementations.

Except legacy applications, most of the applications follow common standards when it comes to object modeling, for example. All applications related to, say, "HR&Benefits" maintain an object model with details of the organization, employees with demographic information, benefits, payroll, and deductions. They are only different in the way they handle the details, based on the country and region they are operating for. For each object type, similar to the HR&Benefits systems, we can define an interface using the Interface Definition Language (OMG IDL).

### ***The IDL Programming Model:***

The IDL programming model, known as Java™ IDL, consists of both the Java CORBA ORB and the idlj compiler that maps the IDL to Java bindings that use the Java CORBA ORB, as well as a set of APIs, which can be explored by selecting the org.omg prefix from the Packages section of the API index. Java IDL adds CORBA (Common Object Request Broker Architecture) capability to the Java platform, providing standards-based

interoperability and connectivity. Runtime components include a Java ORB for distributed computing using IIOP communication.

To use the IDL programming model, define remote interfaces using OMG Interface Definition Language (IDL), then compile the interfaces using idlj compiler. When you run the idlj compiler over your interface definition file, it generates the Java version of the interface, as well as the class code files for the stubs and skeletons that enable applications to hook into the ORB.

### ***Portable Object Adapter (POA) :***

An object adapter is the mechanism that connects a request using an object reference with the proper code to service that request. The Portable Object Adapter, or POA, is a particular type of object adapter that is defined by the CORBA specification. The POA is designed to meet the following goals:

- Allow programmers to construct object implementations that are portable between different ORB products.
- Provide support for objects with persistent identities.

### ***Inter-ORB communication***

To invoke the remote object instance, the client can get its object reference using a naming service. Replacing the object reference with the remote object reference, the client can make the invocation of the remote method with the same syntax as the local object method invocation. ORB keeps the responsibility of recognizing the remote object reference based on the client object invocation through a naming service and routes it accordingly.

An Object Request Broker (ORB) is part of the Java Platform. The ORB is a runtime component that can be used for distributed computing using IIOP communication. Java IDL is a Java API for interoperability and integration with CORBA. Java IDL included both a Java-based ORB, which supported IIOP, and the IDL-to-Java compiler, for generating client-side stubs and server-side code skeletons. J2SE v.1.4 includes an Object Request Broker Daemon (ORBD), which is used to enable clients to transparently locate and invoke persistent objects on servers in the CORBA environment.

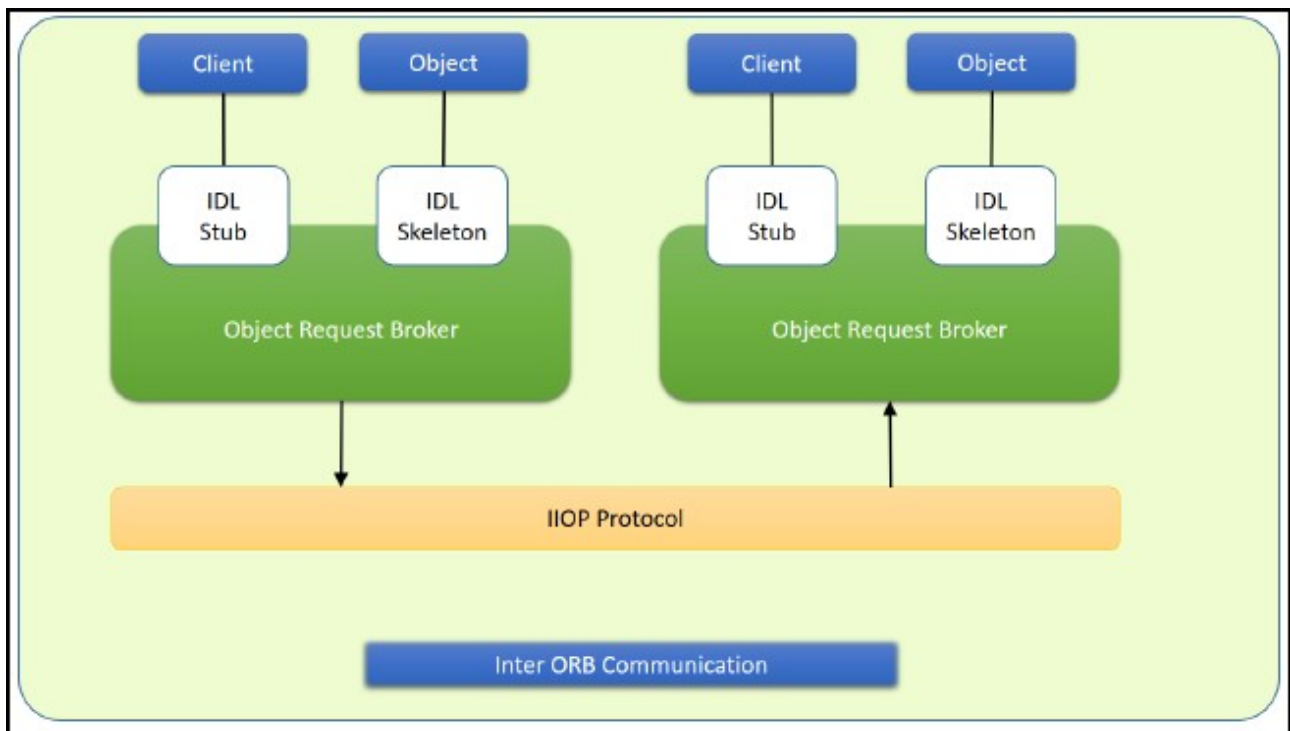


Figure 1: CORBA Architecture

## Code

### ReverseClient.java

```
import ReverseModule.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import java.io.*;

class ReverseClient {

    public static void main(String args[]){

        Reverse ReverseImpl=null;

        try{

            //initialize the ORB
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);

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

            String name = "Reverse";
            ReverseImpl = ReverseHelper.narrow(ncRef.resolve_str(name));

            System.out.println("Enter String=");
            BufferedReader br = new BufferedReader(new
            InputStreamReader(System.in));
            String str = br.readLine();
```

```

        String tempStr = ReverseImpl.reverse_string(str);
        System.out.println(tempStr);
    }catch(Exception e){
        e.printStackTrace();
    }
}
}

```

## ReverseImpl.java

```

import ReverseModule.ReversePOA;
import java.lang.String;

class ReverseImpl extends ReversePOA{

    ReverseImpl(){
        super();
        System.out.println("Reverse Object Created");
    }

    public String reverse_string(String name){

        StringBuffer str=new StringBuffer(name);
        str.reverse();

        return (("Server Send " + str));
    }

}

```

## ReverseModule.idl

```

module ReverseModule {

    interface Reverse{

        string reverse_string(in string str);

    };

};

```

## ReverseServer.java

```
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ReverseModule.Reverse;

class ReverseServer {

    public static void main(String args[]){

        try{

            //initialize the ORB
            org.omg.CORBA.ORB orb =
org.omg.CORBA.ORB.init(args,null);

            //initialise the BOA/POA
            POA rootPOA =
POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
            rootPOA.the_POAManager().activate();

            //creating the calculator object
            ReverseImpl rvr = new ReverseImpl();

            //
            org.omg.CORBA.Object ref =
rootPOA.servant_to_reference(rvr);

            System.out.println("Step 1");
            Reverse h_ref = ReverseModule.ReverseHelper.narrow(ref);
            System.out.println("Step 2");

            org.omg.CORBA.Object objRef =
orb.resolve_initial_references("NameService");

            System.out.println("Step 3");
            NamingContextExt ncRef =
NamingContextExtHelper.narrow(objRef);
            System.out.println("Step 4");

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

            System.out.println("Server reading and waiting...");
            orb.run();

        }catch(Exception e){

            e.printStackTrace();

        }

    }

}
```

# Screenshots

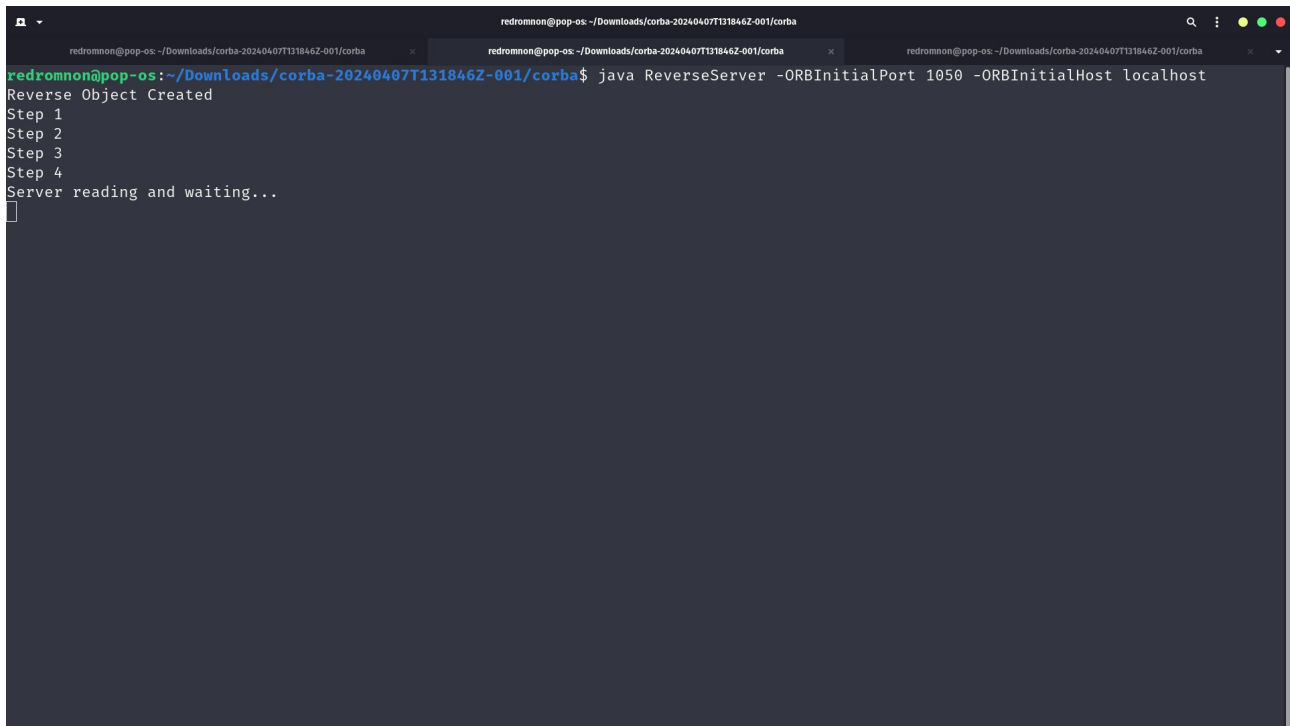
## Orbd service

```
redromnon@pop-os: ~/Downloads/corba-20240407T131846Z-001/corba
redromnon@pop-os: ~/Downloads/corba-20240407T131846Z-001/corba$ idlj -fall ReverseModule.idl
redromnon@pop-os: ~/Downloads/corba-20240407T131846Z-001/corba$ javac *.java ReverseModule/*.java
ReverseModule/_ReverseStub.java:46: warning: IORCheckImpl is internal proprietary API and may be removed in a future release
com.sun.corba.se.impl.orbutil.IORCheckImpl.check(str, "ReverseModule._ReverseStub");
                                ^
Note: ReverseModule/ReversePOA.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
1 warning
redromnon@pop-os: ~/Downloads/corba-20240407T131846Z-001/corba$ orbd -ORBInitialPort 1050 -ORBInitialHost localhost
```

## Client

```
redromnon@pop-os: ~/Downloads/corba-20240407T131846Z-001/corba
redromnon@pop-os: ~/Downloads/corba-20240407T131846Z-001/corba$ java ReverseClient -ORBInitialPort 1050 -ORBInitialHost localhost
Enter String=
cat
Server Send tac
redromnon@pop-os: ~/Downloads/corba-20240407T131846Z-001/corba$
```

## Server



```
redromnon@pop-os: ~/Downloads/corba-20240407T131846Z-001/corba
redromnon@pop-os: ~/Downloads/corba-20240407T131846Z-001/corba$ java ReverseServer -ORBInitialPort 1050 -ORBInitialHost localhost
Reverse Object Created
Step 1
Step 2
Step 3
Step 4
Server reading and waiting...
█
```

## Conclusion

Thus, we have successfully implemented CORBA architecture and demonstrating object brokering.