EXPERIMENT NO. 6

Title:
To develop any distributed application with CORBA program using Java IDL
Objective:
By the end of this experiment, the student will be able to:

- Define CORBA interfaces using **IDL (Interface Definition Language)**
- Compile and generate stubs/skeletons using **idlj**
- Develop **Server** and **Client** programs using **Java IDL**
- Establish communication between distributed components via **ORB**

⚙️ Tools Required

- JDK (Java SE) 8 or higher
- Built-in **idlj compiler** (included in JDK)
- Command-line terminal or NetBeans IDE

🚀 Implementation Steps

Step 1 – Write IDL File
Create a file named Hello.idl in a folder (e.g., CORBAExample).
📄 **Hello.idl**

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

Step 2 – Compile the IDL File
Run the following command in the terminal (from the directory containing Hello.idl):
idlj -fall Hello.idl
This generates several files under the folder HelloApp/:

```
HelloApp/
├── Hello.java
├── HelloHelper.java
├── HelloHolder.java
├── HelloOperations.java
├── HelloPOA.java
└── _HelloStub.java
```

These files are **auto-generated** by the IDL compiler and contain stubs/skeletons for communication.

Step 3 – Create the Server Program
📄 **HelloServer.java**

```
import HelloApp.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
import java.util.Properties;
```

```
// Implementation class (Servant)
class HelloServant extends HelloPOA {
   private ORB orb;

   public void setORB(ORB orb_val) {
      orb = orb_val;
   }

   // Implementation of sayHello() method
   public String sayHello() {
      return "\nHello world!! (Response from CORBA Server)\n";
   }

   // Method to shutdown the server
   public void shutdown() {
      orb.shutdown(false);
   }
}

public class HelloServer {

   public static void main(String args[]) {
      try {
         // Create and initialize the ORB
         ORB orb = ORB.init(args, null);

         // Get reference to RootPOA and activate the POAManager
         POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
         rootpoa.the_POAManager().activate();

         // Create servant and register it with the ORB
         HelloServant helloRef = new HelloServant();
         helloRef.setORB(orb);

         // Get object reference from the servant
         org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloRef);
         Hello href = HelloHelper.narrow(ref);

         // Get the root naming context
         org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
         NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

         // Bind the Object Reference in Naming
         String name = "Hello";
         NameComponent path[] = ncRef.to_name(name);
         ncRef.rebind(path, href);

         System.out.println("HelloServer ready and waiting...");
```

```
            // Wait for client invocations
            orb.run();

        } catch (Exception e) {
            System.err.println("ERROR: " + e);
            e.printStackTrace(System.out);
        }

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

Step 4 – Create the Client Program

HelloClient.java

```
import HelloApp.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
import java.util.Properties;

public class HelloClient {

    public static void main(String args[]) {
        try {
            // Create and initialize the ORB
            ORB orb = ORB.init(args, null);

            // Get the root naming context
            org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
            NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

            // Resolve the Object Reference in Naming
            String name = "Hello";
            Hello helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));

            System.out.println("Obtained a handle on server object: " + helloImpl);
            System.out.println(helloImpl.sayHello());

        } catch (Exception e) {
            System.out.println("ERROR : " + e);
            e.printStackTrace(System.out);
        }
    }
}
```

🧱 Step 5 – Compilation Steps

Open a terminal in your project folder (where Hello.idl is located):

Compile all files:

javac HelloServer.java HelloClient.java HelloApp/*.java

🧭 Step 6 – Run the CORBA Application

1️⃣ Start the CORBA Naming Service

tnameserv -ORBInitialPort 1050 &

(keep this running)

2️⃣ Run the Server

java HelloServer -ORBInitialPort 1050 -ORBInitialHost localhost

Expected Output:

HelloServer ready and waiting...

3️⃣ Run the Client

In a new terminal:

java HelloClient -ORBInitialPort 1050 -ORBInitialHost localhost

Expected Output:

Obtained a handle on server object: IOR:...

Hello world!! (Response from CORBA Server)

✅ Expected Output

Server Console:

HelloServer ready and waiting...

Client Console:

Obtained a handle on server object: IOR:0000000000...

Hello world!! (Response from CORBA Server)

📜 Summary Table

Component	Description
IDL File	Defines remote interface (sayHello)
idlj Compiler	Generates stub and skeleton code
Server	Registers CORBA object in naming service
Client	Looks up remote object and calls sayHello()
Protocol	IIOP (Internet Inter-ORB Protocol)
ORB Tool	tnameserv for naming service

🏁 Conclusion

Thus, a **CORBA-based distributed Java application** was successfully developed and executed using **Java IDL**, demonstrating communication between distributed client and server applications through the **ORB**.