Dinesh Pamu

B2- 30

ASSIGNMENT NO. 2

Title:

Distributed application using CORBA to demonstrate object brokering.

Problem Statement:

Develop any distributed application using CORBA to demonstrate object brokering. (Calculator or String operations).

Tools / Environment:

CORBA Object Request Broker (ORB), Programming language, Integrated Development Environment (IDE).

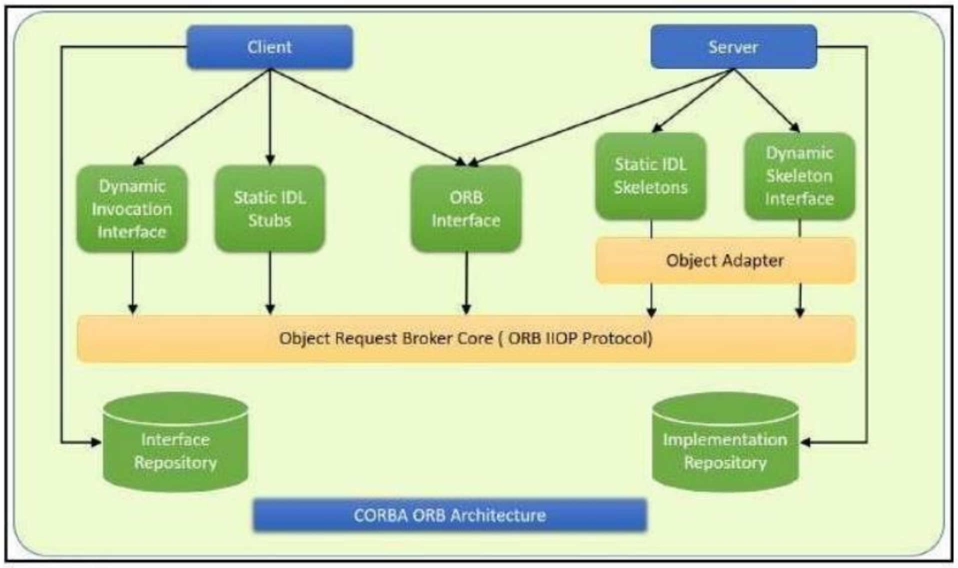
Theory:

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.

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.

The following diagram shows a single-process ORB CORBA architecture with the IDL configured as client stubs with object skeletons, The objects are written (on the right) and a client for it (on the left), as represented in the diagram. The client and server use stubs and skeletons as proxies, respectively. The IDL interface follows a strict definition, and even though the client and server are implemented in different technologies, they should integrate smoothly with the interface definition strictly implemented.


Client
Server
Dynamic Invocation Interface
Static IDL Stubs
ORB Interface
Static IDL Skeletons
Dynamic Skeleton Interface
Object Adapter
Object Request Broker Core (ORB IIOP Protocol)
Interface Repository
Implementation Repository
CORBA ORB Architecture

Designing Solution:

1. Define the IDL interface:

   Create an IDL file that defines the interface for the distributed application. For example, we can create an IDL interface for a Calculator object that has methods for addition.

   Once the remote interfaces in IDL are described, you need to generate Java classes that act as a starting point for implementing those remote interfaces in Java using an IDL-to-Java compiler.

2. Implement the server:

   Create a server application that implements the IDL interface. The server application will receive requests from clients and respond to them by invoking the appropriate method on the Calculator object.

3. Generate the stub and skeleton code:

   Use an IDL compiler to generate the stub and skeleton code for the server and client applications. The stub and skeleton code are used to translate the method calls made by the client application into remote procedure calls that can be executed on the server.

4. Implement the client:

   Create a client application that can connect to the server and make requests using the IDL interface. The client application will use the generated stub code to make remote method calls to the server.

5. Test the application:

   Run the server and client applications and test the remote method calls to make sure that the application is functioning as expected.

Conclusion:

CORBA provides network transparency; Java provides implementation transparency. CORBA complements the Java™ platform by providing a distributed object framework, services to support that framework, and interoperability with other languages. The Java platform complements CORBA by providing a portable, highly productive implementation environment. The combination of Java and CORBA allows you to build more scalable and more capable applications than can be built using the JDK alone.

Code:

1. **Defining the Interface** (Addition.idl)

```
module AdditionApp{

        interface Addition{

                long add(in long a,in long b);
                oneway void shutdown();

        };

};
```

2. **Implementing the Client** (Client.java)

```
package calclient;

import java.rmi.Naming;


public class client {

public static void main(String[] args) {
        // TODO Auto-generated method stub
        try {

        org.omg.CORBA.ORB orb = ORB.init(args, null);
        Object objRef = orb.resolve_initial_references("NameService");
        NamingContextExt scRef = NamingContextExtHelper.narrow(objRef);
Addition addobj = AdditionHelper.narrow(scRef.resolve_str("ABC"));

                Scanner sc = new Scanner(System.in);
                for (; ; ) {
                        System.out.println("Enter a:");
                        String aa = sc.nextLine();
                        System.out.println("Enetr b:");
                        String bb =sc.nextLine();
                        int a = Integer.parseInt(aa);
                        int b = Integer.parseInt(bb);
                        int r = addobj.add(a, b);
        System.out.println("result of Addition------> "+ r);
        System.out.println("----------------------------------------");

                }

        } catch (Exception e) {
                // TODO: handle exception
        }
        }

}
```

3. **Server-side Implementation** (StartServer.java)

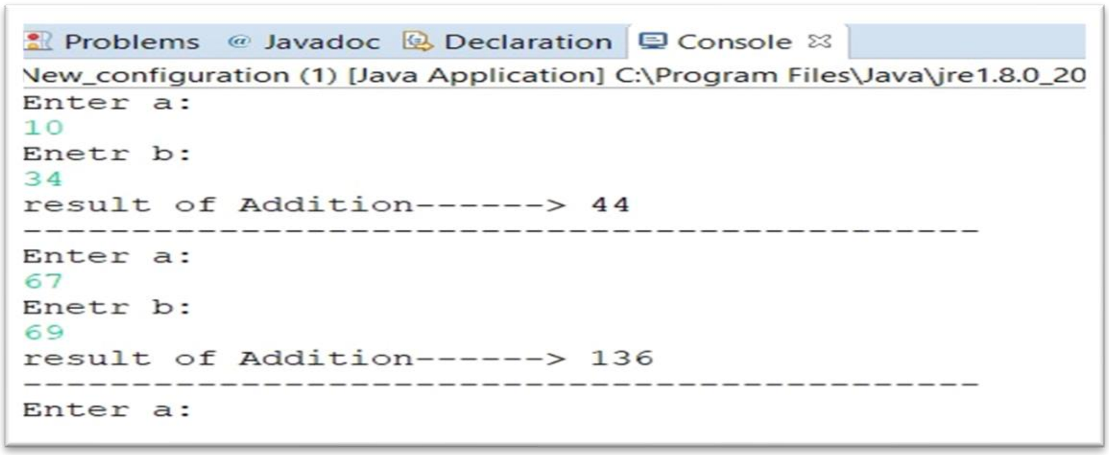
```
package Server;

import java.rmi.Naming;
import org.omg.CORBA.ORB;
import org.omg.CORBA.ORBPackage.InvalidName;
import org.omg.CosNaming.NameComponent;
import org.omg.CosNaming.NamingContextExt;
import org.omg.CosNaming.NamingContextExtHelper;
import org.omg.CosNaming.NamingContextPackage.CannotProceed;
import org.omg.CosNaming.NamingContextPackage.NotFound;
import org.omg.PortableServer.POA;
import org.omg.PortableServer.POAHelper;
import org.omg.PortableServer.POAManagerPackage.AdapterInactive;
import org.omg.PortableServer.POAPackage.ServantNotActive;
import org.omg.PortableServer.POAPackage.WrongPolicy;

import AdditionApp.Addition;
import AdditionApp.AdditionHelper;

public class StartServer {

    public static void main(String[] args) throws AdapterInactive {
        // TODO Auto-generated method stub

        try {
            ORB = ORB.init(args, null);
            POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
            rootpoa.the_POAManager().activate();
            AdditionObj addobj = new AdditionObj();
            addobj.setOrb(orb);

            org.omg.CORBA.Object ref = rootpoa.servant_to_reference(addobj);
            Addition href = AdditionHelper.narrow(ref);

            org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
            NamingContextExt scref = NamingContextExtHelper.narrow(objRef);
            NameComponent[] path = scref.to_name("ABC");
            scref.rebind(path, href);
            System.out.println("Addition server ready and waiting......");

            for (;;) {
                orb.run();
            }
```

```
			} catch (InvalidName e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			} catch (ServantNotActive e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			} catch (WrongPolicy e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			} catch (org.omg.CosNaming.NamingContextPackage.InvalidName e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			} catch (NotFound e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			} catch (CannotProceed e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
	}

}
```

4. **Server-side Interface (**AdditionObj.java**)**

```
package Server;

import com.sun.corba.se.internal.POA.POAORB;
import com.sun.corba.se.internal.iiop.ORB;

import AdditionApp.AdditionPOA;

public class AdditionObj extends AdditionPOA {
	private org.omg.CORBA.ORB orb;
	public void setOrb(org.omg.CORBA.ORB orb2) {
		this.orb = orb2;
	}

	@Override
	public int add(int a, int b) {
		// TODO Auto-generated method stub
		int r = a+b;
		return r;
	}

	@Override
	public void shutdown() {
		// TODO Auto-generated method stub
		orb.shutdown(false);
	}
```

Output:



```
New_configuration (1) [Java Application] C:\Program Files\Java\jre1.8.0_20
Enter a:
10
Enetr b:
34
result of Addition------> 44
--------------------------------------------
Enter a:
67
Enetr b:
69
result of Addition------> 136
--------------------------------------------
Enter a:
```