



MEKDELA AMBA UNIVERSITY

COLLEGE OF NATURAL AND COMPUTATIONAL SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

Course Title: Introduction to Distributed System

Course Code: CoSc4038

Title: **Write program using CORBA to demonstrate object broker.**

Members	ID
1. Haileyesus Gasha	1200653
2. Hamrawit Getachew	1202722
3. Tenaye Birhan	1201237
4. Selamawit Alemu	1201122

Submitted to: Mr. Melese Alemante (MSC)

Submission date: 18/08/2015 E.C

Tulu Awliya, Ethiopia

Contents

Introduction.....	1
Components.....	1
CORBA Implementation.....	1
The basic steps for CORBA development.....	1
Implementation	2

Introduction

The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together. CORBA is a standard for distributing objects across networks so that operations on those objects can be called remotely.

Components

CORBA includes four components:

Object Request Broker (ORB): The Object Request Broker (ORB) handles the communication, marshaling, and unmarshaling of parameters so that the parameter handling is transparent for a CORBA server and client applications.

CORBA server: The CORBA server creates CORBA objects and initializes them with an ORB. The server places references to the CORBA objects inside a naming service so that clients can access them.

Naming service: The naming service holds references to CORBA objects.

CORBA Request node: The CORBA Request node acts as a CORBA client.

CORBA Implementation

Data communication from client to server is accomplished through a well-defined object-oriented interface. The Object Request Broker (ORB) determines the location of the target object, sends a request to that object, and returns any response back to the caller. The illustration below identifies how a client sends a request to a server through the ORB:

The basic steps for CORBA development

1. Create the IDL to Define the Application Interfaces

The IDL provides the operating system and programming language independent interfaces to all services and components that are linked to the ORB. The IDL specifies a description of any services a server component exposes to the client. The term "IDL Compiler" is often used, but the IDL is actually translated into a programming language.

2. Translate the IDL

An IDL translator typically generates two cooperative parts for the client and server implementation, stub code and skeleton code. The stub code generated for the interface classes is associated with a client application and provides the user with a well-defined Application Programming Interface (API).

3. Compile the Interface Files

Once the IDL is translated into the appropriate language, these interface files are compiled and prepared for the object implementation.

4. Complete the Implementation

If the implementation classes are incomplete, the spec and header files and complete bodies and definitions need to be modified before passing through to be compiled. The output is a complete client/server implementation.

5. Compile the Implementation

Once the implementation class is complete, the client interfaces are ready to be used in the client application and can be immediately incorporated into the client process. This client process is responsible for obtaining an object reference to a specific object, allowing the client to make requests to that object in the form of a method call on its generated API.

6. Link the Application

Once all the object code from steps three and five have been compiled, the object implementation classes need to be linked to the programming language linker. Once linked to the ORB library, in this example, ORBexpress, two executable operations are created, one for the client and one for the server.

7. Run the Client and Server

The development process is now complete and the client will now communicate with the server. The server uses the object implementation classes allowing it to communicate with the objects created by the client requests.

In its simplest form, the server must perform the following:

- ✚ Create the required objects.
- ✚ Notify the CORBA environment that it is ready to receive client requests.
- ✚ Process client requests by dispatching the appropriate servant.

Implementation

Step 1: Create two separate projects or files for client and server.

Step 2: write an interface .idl file and save it in CORBA_Server project

Calculator.idl

```
module CORBACalculator
{
    interface Calculator
    {
        double add(in double a,in double b );
        double sub(in double a,in double b );
        double mul(in double a,in double b );
    }
}
```

```

double div(in double a,in double b );

double mod(in double a,in double b );

oneway void shutdown();

};

};

```

Step 3: we need to compile the idl file to compile this there is an idlj file in the jre folder it is:

ik (C:) > Program Files > Java > jdk1.8.0_181 > bin

	Name	Date modified	Type	Size
	appletviewer	3/30/2023 3:29 PM	Application	17 KB
	extcheck	3/30/2023 3:29 PM	Application	17 KB
	idlj	3/30/2023 3:29 PM	Application	17 KB
	jabswitch	3/30/2023 3:29 PM	Application	35 KB
	jar	3/30/2023 3:29 PM	Application	17 KB
	jarsigner	3/30/2023 3:29 PM	Application	17 KB
	java	3/30/2023 3:29 PM	Application	203 KB

To compile idl file we write the following command on cmd

```

C:\> Command Prompt

Microsoft Windows [Version 10.0.19041.450]
(c) 2020 Microsoft Corporation. All rights reserved.

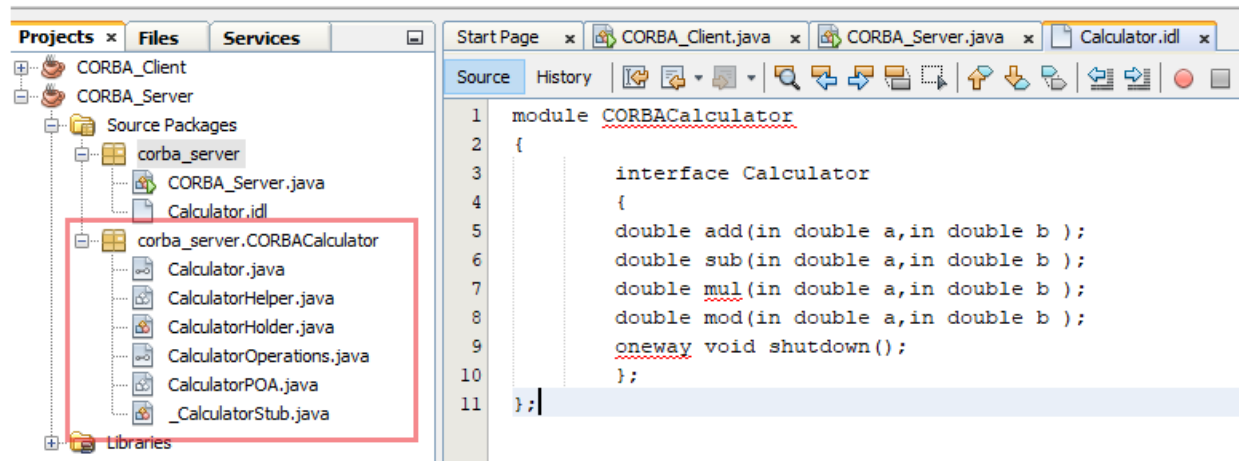
C:\Users\A>cd C:\Users\A\Documents\CORBA_Server\src\corba_server

C:\Users\A\Documents\CORBA_Server\src\corba_server>idlj -fall Calculator.idl

C:\Users\A\Documents\CORBA_Server\src\corba_server>

```

When we compile Calculator.idl file it has created 6 files as follows



Calculator.java – the “signature” file that implements the Calculator contract

CalculatorHelper.java – provides functions for casting references to their types

CalculatorHolder.java – the instance “holder”

CalculatorOperators.java – the operations specific to Calculator.

CalculatorPOA.java – the server skeleton

_ CalculatorStub.java – the client Stub

Step 4: we create in CORBA_Server project the Calculator object to implement the interface

CalculatorObject.java

```
import CORBACalculator.*;

import org.omg.CORBA.*;

public class CalculatorObject extends CalculatorPOA {

    private ORB orb;

    public void setORB(ORB orb_val)

    {

        orb=orb_val;

    }

    // implementing all the methods

    @Override
```

```
public double add(double a, double b)
{
    double result=a+b;
    return result;
}
```

@Override

```
public double sub(double a, double b)
{
    double result=a-b;
    return result;
}
```

@Override

```
public double mul(double a, double b)
{
    double result=a*b;
    return result;
}
```

@Override

```
public double div(double a, double b)
{
    try
    {
        double result=a/b;
        return result;
    }
```

```

        catch(Exception e)
        {
            System.out.println(e);
        }

        return 0;
    }

    @Override

    public double mod(double a, double b) {

        double result=a%b;

        return result;
    }

    @Override

    public void shutdown() {

        orb.shutdown(false);

    }

}

```

Step 5: write java class to create and initialize the ORB, to create servant and register it with the ORB and to get object reference from the servant.

StartServer.java

```

import CORBACalculator.*;

import org.omg.CosNaming.*;

import org.omg.CosNaming.NamingContextPackage.*;

import org.omg.CORBA.*;

import org.omg.PortableServer.*;

import java.util.*;

public class StartServer {

    public static void main(String args[])

```



```

{
    try
    {
        // create and initialize the ORB

        ORB orb=ORB.init(args,null);

        POA rootpoa=POAHelper.narrow(orb.resolve_initial_references("RootPOA"));

        rootpoa.the_POAManager().activate();


        //Create servant and register it with the ORB

        CalculatorObject calcobj=new CalculatorObject();

        calcobj.setORB(orb);


        // get object reference from the servant

        org.omg.CORBA.Object ref= rootpoa.servant_to_reference(calcobj);

        Calculator href= CalculatorHelper.narrow(ref);


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

        NamingContextExt ncRef= NamingContextExtHelper.narrow(objref);


        NameComponent path[]= ncRef.to_name("ABC");

        ncRef.rebind(path, href);

        System.out.println("Calculator server ready and waiting...");

        //Wait for invocation from client

        for(;;)
        {

            orb.run();
        }
    }
}

```

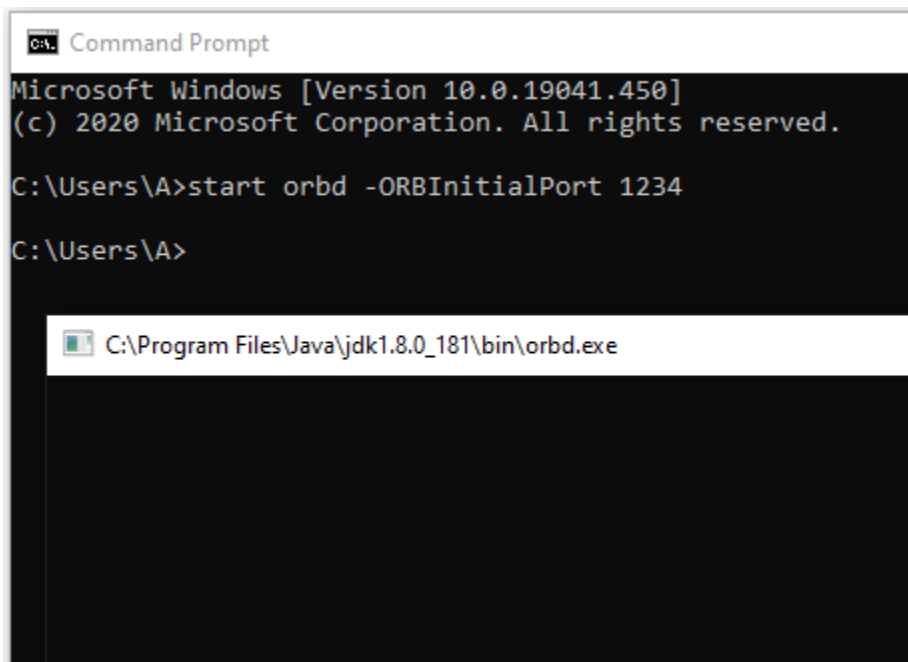
```

    }
}
catch(Exception e)
{
    System.err.println(e);
    e.printStackTrace(System.out);
}

System.out.println("Server Exiting...");
}
}

```

Step 6: Go to cmd and start orbd with port number because ORB should be run in the background



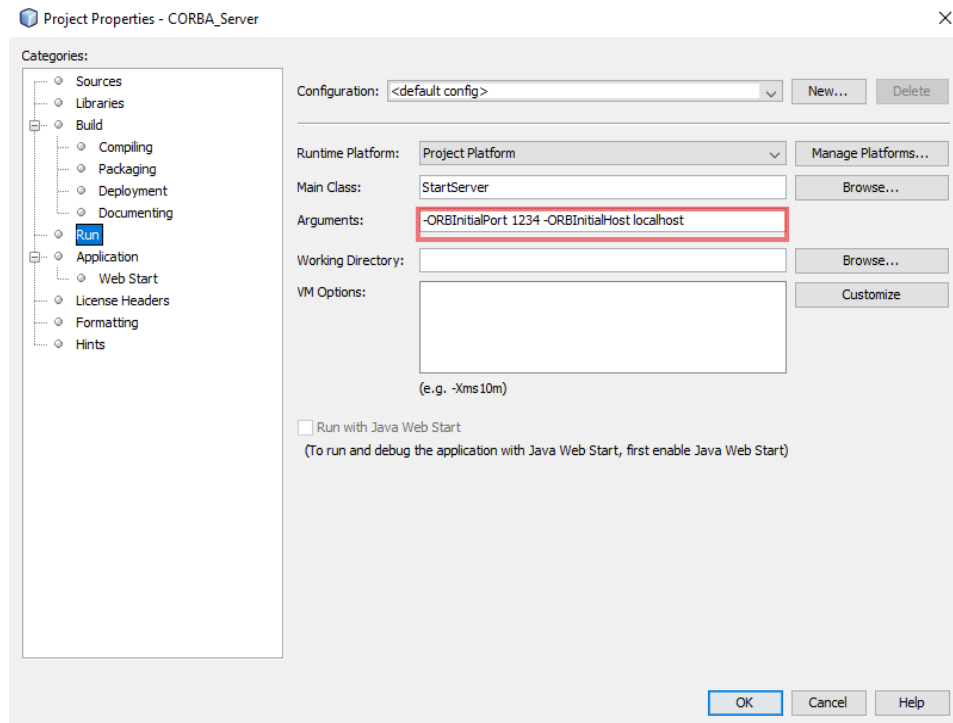
```

C:\Users\A>start orbd -ORBInitialPort 1234
C:\Users\A>

```

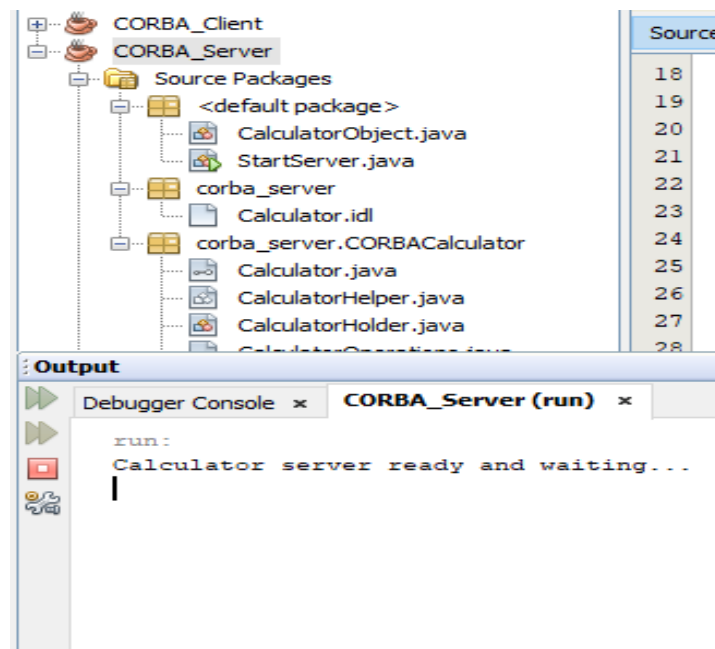
C:\Program Files\Java\jdk1.8.0_181\bin\orbd.exe

Step 7: Set configuration (IP and Port number) to server project



Step 8: Run server project

Output



In client project

Step 9: Copy the six files (Generated by .idl file) in the client project

Step 10: write java class to

StartClient.java

```
import CORBACalculator.*;

import org.omg.CosNaming.*;

import org.omg.CORBA.*;

import org.omg.PortableServer.*;

import java.util.*;

public class StartClient {

    public static void main(String [] args)

    {

        try

        {

            ORB orb=ORB.init(args,null);

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

            NamingContextExt ncRef=NamingContextExtHelper.narrow(objref);

            Calculator

            calcobj=(Calculator)CalculatorHelper.narrow(ncRef.resolve_str("ABC"));

            Scanner input=new Scanner(System.in);

            System.out.println("=====

            =====");

            System.out.println("=====Calculator=====

            =====");

            System.out.println("=====

            =====");

            for(;;)

            {
```

```

System.out.println("Enter first number: ");

String inp1=input.next();

double a=Double.parseDouble(inp1);

System.out.println("Enter second number: ");

String inp2=input.next();

double b=Double.parseDouble(inp2);

System.out.println("Enter operator    1- For addition+,-,*,/,%): ");

System.out.println("\t\t 2- For Subtraction");

System.out.println("\t\t 3- For Multiplication");

System.out.println("\t\t 4- For Division");

System.out.println("\t\t 5- For Modulus");


double result;

int opp=input.nextInt();

switch(opp)
{
    case 1:

        result=calcobj.add(a,b);

        System.out.println("Sum= "+result);

        break;

    case 2:

        result=calcobj.sub(a,b);

        System.out.println("Difference= "+result);

        break;

    case 3:

        result=calcobj.mul(a,b);

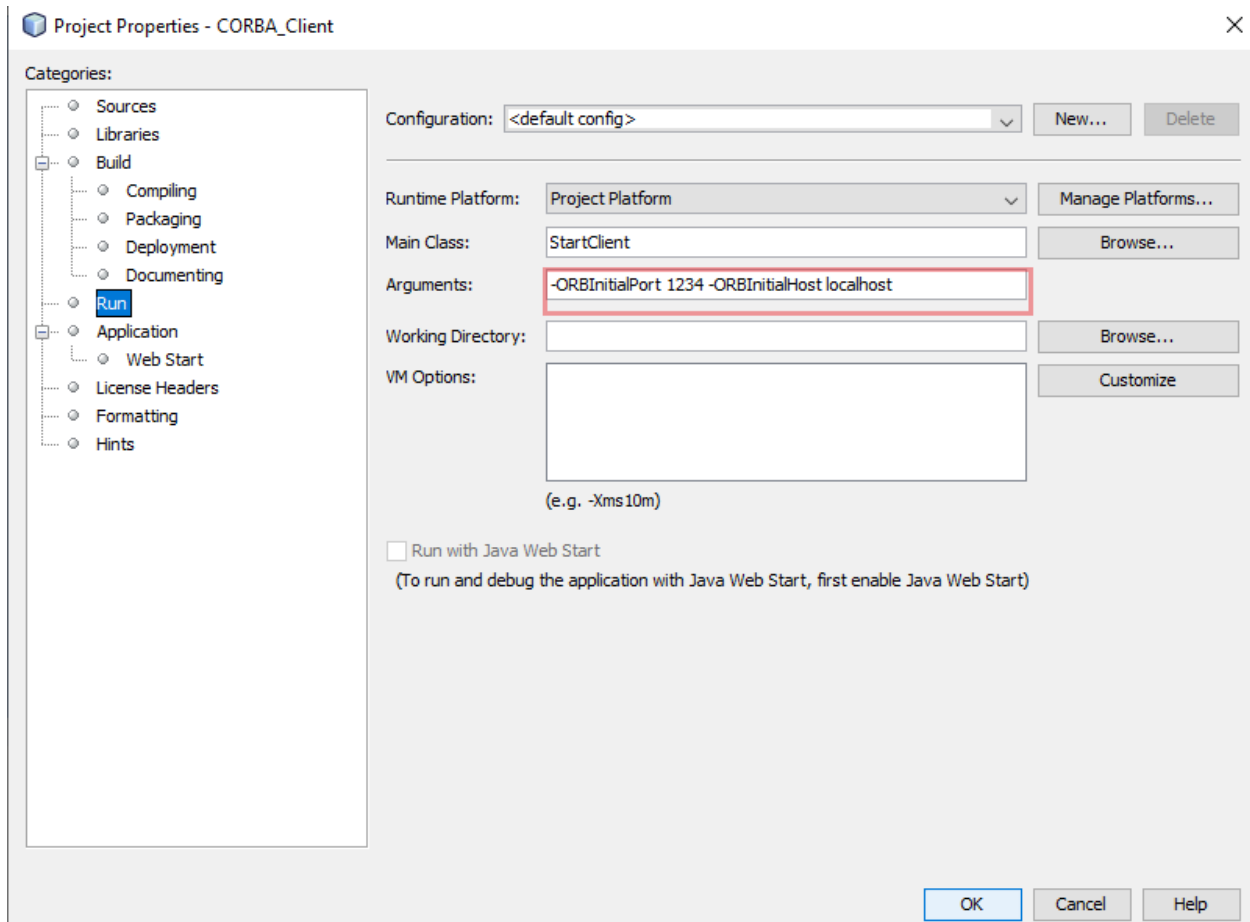
```

```

        System.out.println("Product= "+result);
        break;
        case 4:
            result=calcobj.div(a,b);
            System.out.println("Quotient= "+result);
            break;
        case 5:
            result=calcobj.add(a,b);
            System.out.println("Remainder= "+result);
            break;
        default:
            System.out.println("Invalid operator");
            break;
    }
}
}
catch(Exception e)
{
    System.out.println("Invalid operator");
}
}
}

```

Step 11: Configure client set the same IP and port number.



Step 12: Run client project

The screenshot displays an IDE window with the following components:

- Projects:** CORBA_Client, CORBA_Server
- Files:** StartPage, StartServer.java, StartClient.java, CalculatorObject.java
- Source:** Line 26: `String inp2=input.next();`
- Output:**
 - run:
 - =====Calculator=====
 - Enter first number:
12
 - Enter second number:
32
 - Enter operator 1- For addition
2- For Subtraction
3- For Multiplication
4- For Division
5- For Modulus
1
 - Sum= 44.0
 - Enter first number:
36
 - Enter second number:
76
 - Enter operator 1- For addition
2- For Subtraction
3- For Multiplication
4- For Division
5- For Modulus
2
 - Difference= -40.0
 - Enter first number:
45
 - Enter second number:
2
 - Enter operator 1- For addition
2- For Subtraction
3- For Multiplication
4- For Division
5- For Modulus
3
 - Product= 90.0