J2EE (CS596-019)
Project-2 (10 pts)

Name: HANYUAN ZHUANG

Student ID: 130301013

I. Introduction

Implement a Client/Server Java objects running on different machines. The client agent object reads input command/arithmetic operation from the user (U), invoke "Calculator" methods on the Server objects through Java IDL capability, gets the result back and pass it back to the User (U). The interactions between the User, Client Agent, and the "Calculator" Server are synchronous, i.e., the user has to wait to receive the response to the calculation operation request before responding by another request, etc.

II. Implementation

a. Define IDL interface(Calculator.idl) as below:

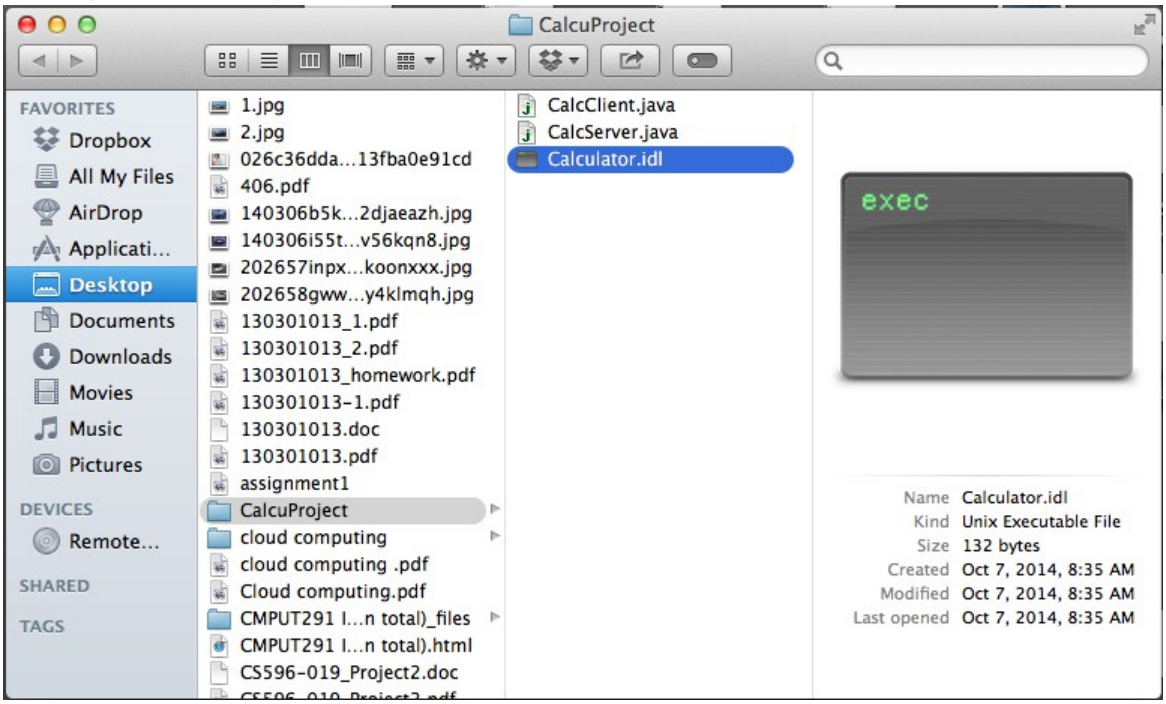
```
module final {
    interface Calc{
        long calculate (in long opcode, in long op1, in long op2);
        long exit();
    }
}
```

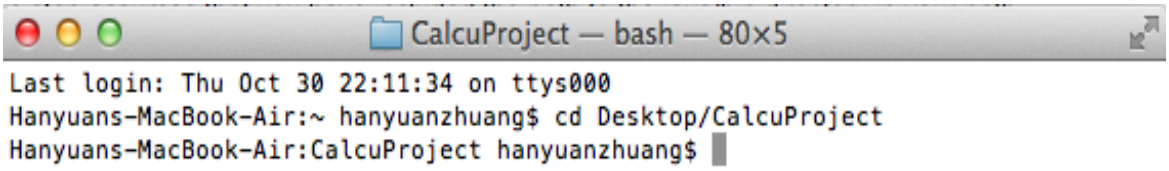
b. Define the Server class (CalculatorServer.java, see the source code below) which implements calculte(int, int, int) and exit() method defined in IDL interface.

c. Define the Client class (CalculatorClient.java, see the source code below) which reads input command/arithmetic operation from the user, and print out the result get from the Server.

d. To run this Client-Server application on your development machine:

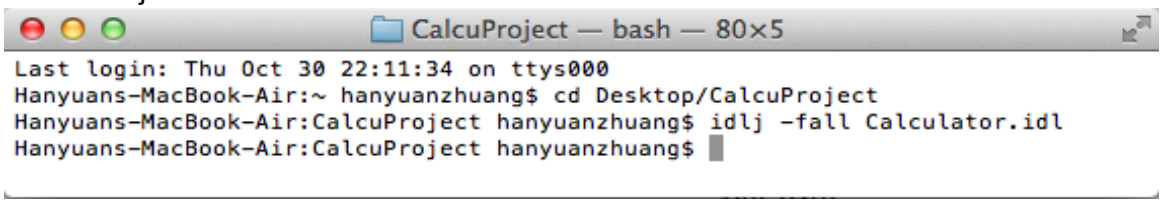
1. Change to the directory that contains the file Calculator.idl.

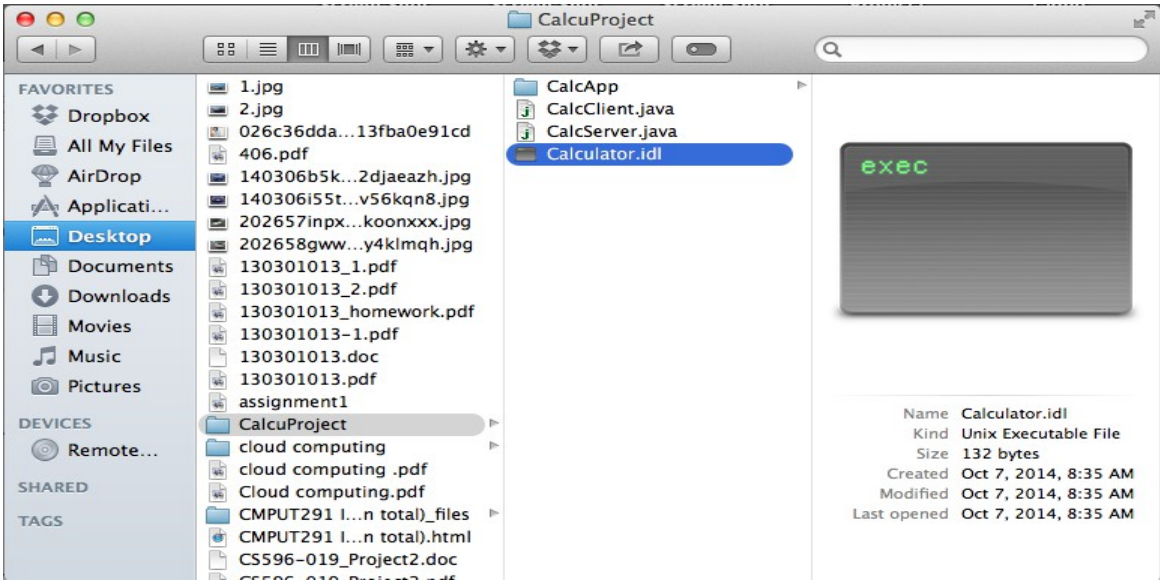




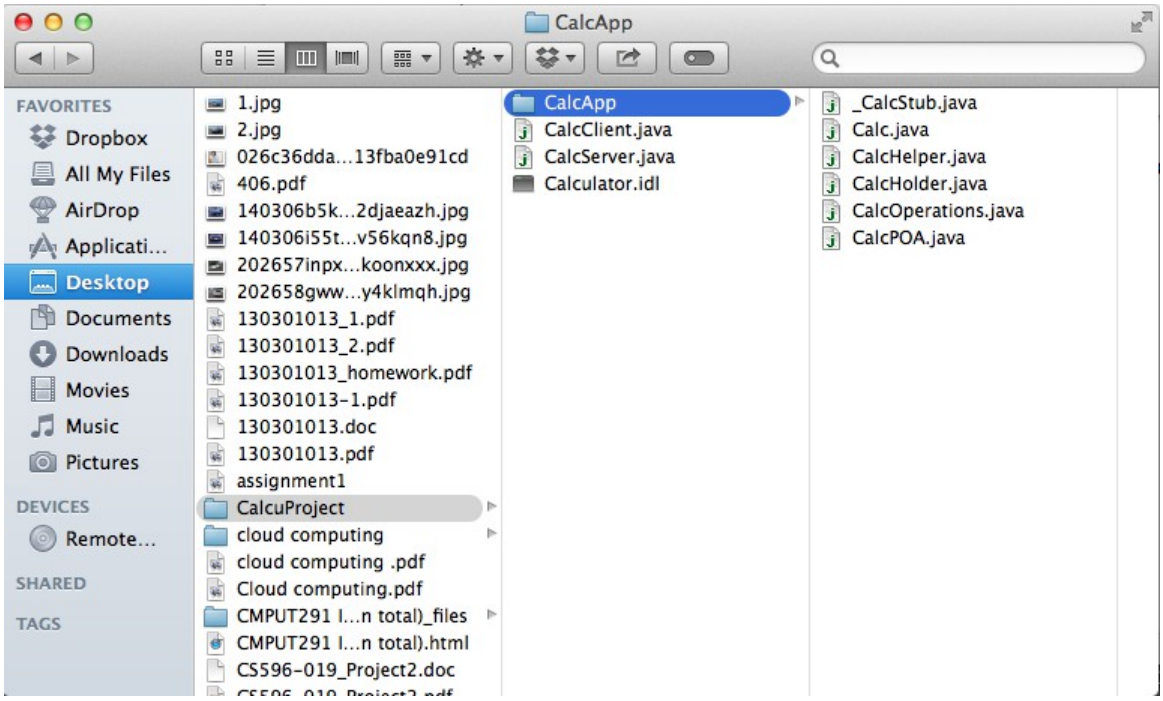
2. Run the IDL-to-Java compiler, idlj, on the IDL file to create stubs and skeletons. This step assumes that you have included the path to the java/bin directory in your path.

```
idlj -fall  Calculator.idl
```





The idlj compiler generates a number of files. The actual number of files generated depends on the options selected when the IDL file is compiled.



CalcPOA.java

This abstract class is the stream-based server skeleton, providing basic CORBA functionality for the server. It extends org.omg.PortableServer.Serva nt , and implements the InvokeHandler interface and the CalcOperations interface. The server class CalcImpl extends CalcPOA.

_CalcStub.java

This class is the client stub, providing CORBA functionality for the client. It extends org.omg.CORBA.portable.ObjectImpl and implements the Calc.java interface.

Calc.java

This interface contains the Java version of our IDL interface. The Calc.java interface extends org.omg.CORBA.Object, providing standard CORBA object functionality. It also extends the CalcOperations interface and org.omg.CORBA.portable.IDLEntity.

CalcHelper.java

This class provides auxiliary functionality, notably the narrow() method required to cast CORBA object references to their proper types.The Helper class is responsible for reading and writing the data type to CORBA streams, and inserting and extracting the data type from Anys. The Holder class delegates to the methods in the Helper class for reading and writing.

CalcHolder.java

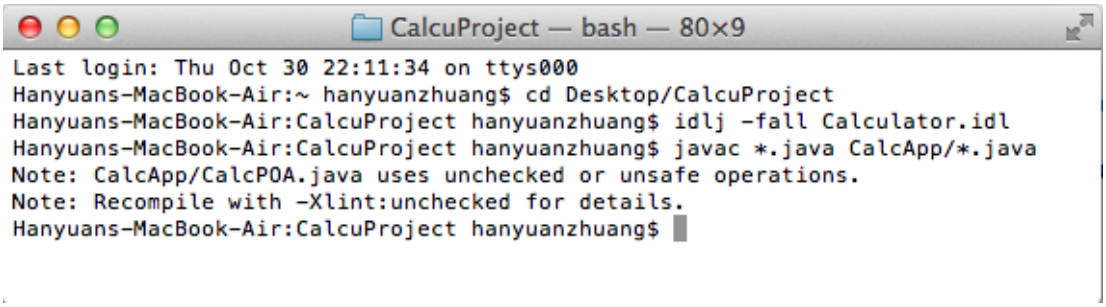
This final class holds a public instance member of type Calc. Whenever the IDL type is an out or an inout parameter, the Holder class is used. It provides operationsfor org.omg.CORBA.portable.OutputStream and org.omg.CORBA.portable.InputStream arguments, which CORBA allows, but which do not map easily to Java's semantics. The Holder class delegates to the methods in the Helper class for reading and writing. It implements org.omg.CORBA.portable.Streamable.

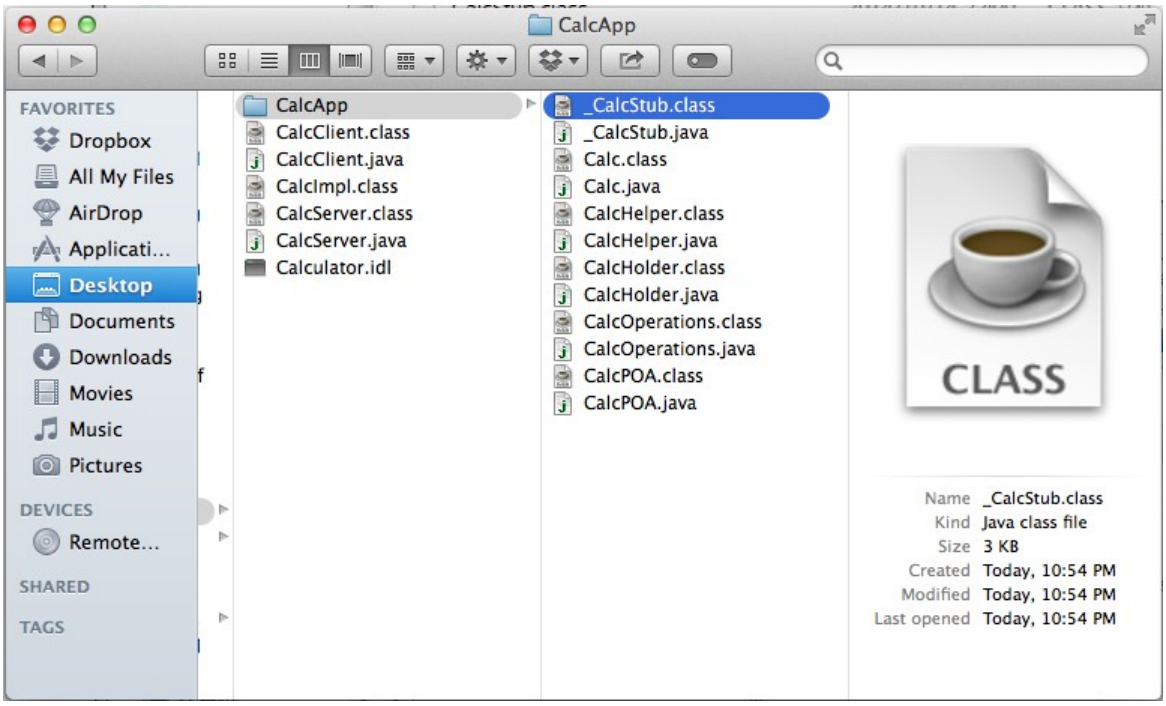
CalcOperations.java

This interface contains the methods calculate (in long opcode, in long op1, in long op2) and exit(). The IDL-to-Java mapping puts all of the operations defined on the IDL interface into this file, which is shared by both the stubs and skeletons.

3. Compile the .java files, including the stubs and skeletons (which are in the directory CalcApp). This step assumes the java/bin directory is included in your path.

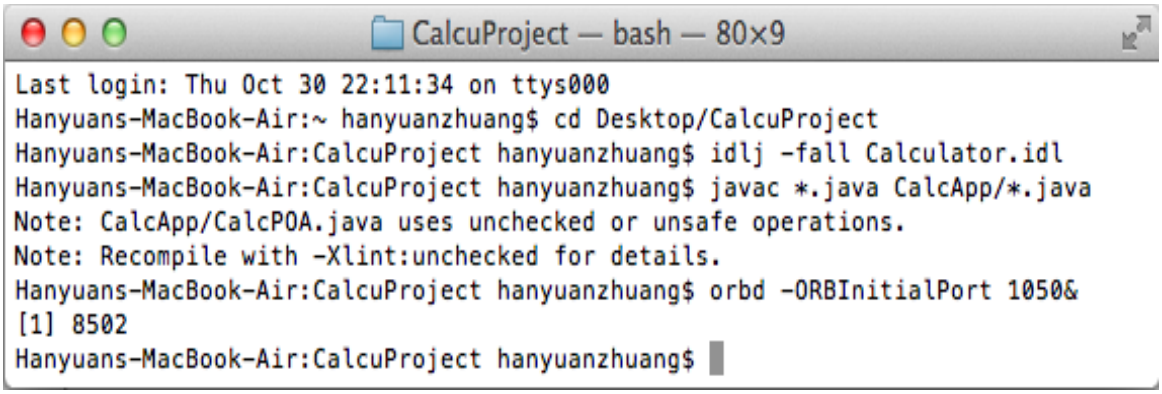
```
javac *.java CalcApp/*.java
```

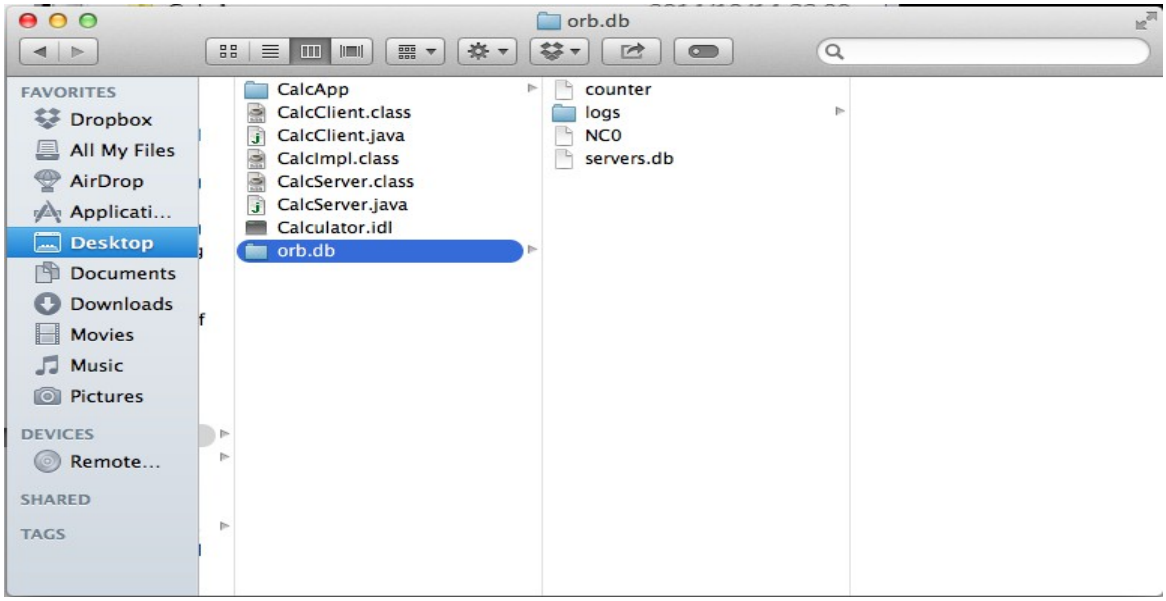




4. Start orbd.

orbd -ORBInitialPort 1050&





Note that 1050 is the port on which you want the name server to run. The -ORBInitialPort argument is a required command-line argument.

5. Start the Calculator Server

```
java CalcServer -ORBInitialPort 1050 -ORBInitialHost localhost&
```

You will see Calculator Server ready and waiting... when the server is started.

```
CalcuProject — bash — 80×16
Last login: Thu Oct 30 22:11:34 on ttys000
Hanyuans-MacBook-Air:~ hanyuanzhuang$ cd Desktop/CalcuProject
Hanyuans-MacBook-Air:CalcuProject hanyuanzhuang$ idlj -fall Calculator.idl
Hanyuans-MacBook-Air:CalcuProject hanyuanzhuang$ javac *.java CalcApp/*.java
Note: CalcApp/CalcPOA.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
Hanyuans-MacBook-Air:CalcuProject hanyuanzhuang$ orbd -ORBInitialPort 1050&
Hanyuans-MacBook-Air:CalcuProject hanyuanzhuang$ java CalcServer -ORBInitialPort
 1050 -ORBInitialHost localhost&
[2] 8514
Hanyuans-MacBook-Air:CalcuProject hanyuanzhuang$ CalculatorServer ready and wait
ing ...
Hanyuans-MacBook-Air:CalcuProject hanyuanzhuang$
```

6. Run the client application:

```
java CalcClient -ORBInitialPort 1050 -ORBInitialHost localhost
```

When the client is running, you will see a response such as the following on your terminal: Obtained a handle on server object

```
CalcuProject — java — 80×8
Last login: Thu Oct 30 22:26:32 on ttys000
Hanyuans-MacBook-Air:~ hanyuanzhuang$ cd Desktop/CalcuProject
Hanyuans-MacBook-Air:CalcuProject hanyuanzhuang$ java CalcClient -ORBInitialPort
 1050 -ORBInitialHost localhost
Obtained a handle on server object
Please type in the order of "opcode, op1 and op2"
```

Now you can do the Calculation here:

```
Please type in the order of "opcode, op1 and op2"
+ 5 10
= 15

Continue Calculator? (Y/N)Y
Please type in the order of "opcode, op1 and op2"
- 8 6
= 2

Continue Calculator? (Y/N)Y
Please type in the order of "opcode, op1 and op2"
* 5 6
= 30

Continue Calculator? (Y/N)Y
Please type in the order of "opcode, op1 and op2"
/ 10 2
= 5

Continue Calculator? (Y/N)N
Hanyuans-MacBook-Air:CalcuProject hanyuanzhuang$
```

III. Source Code

CalculatorServer.java

```
import CalcApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextExtPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;

import java.util.Properties;

class CalcImpl extends CalcPOA {
	private ORB orb;

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

	// implement calculate() method
	// (+)43 (-)45 (*)42 (/)47 (%)37
	public int calculate(int opcode, int op1,int op2) {

		switch (opcode) {
			case 43:
				return op1 + op2;
			case 45:
				return op1 - op2;
			case 42:
				return op1 * op2;
			case 47:
				return op1 / op2;
		}
		return opcode;
	}

	// implement exit() method
	public void exit() {
		try {
			orb.shutdown(false);
			System.out.println("I am out");
			// return 0;
		}
		catch (org.omg.CORBA.BAD_INV_ORDER ex) {
			// return -1;
		}
```

```
        }
}

public class CalcServer {

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

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

                        // create servant and register it with the ORB
                        CalcImpl calcImpl = new CalcImpl();
                        calcImpl.setORB(orb);

                        // get object reference from the servant
                        org.omg.CORBA.Object ref = rootpoa.servant_to_reference(calcImpl);
                        Calc href = CalcHelper.narrow(ref);

                        // get the root naming context
                        // NameService invokes the name service
                        org.omg.CORBA.Object objRef =
orb.resolve_initial_references("NameService");

                        // Use NamingContextExt which is part of the Interoperable
                        // Naming Service (INS) specification.
                        NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

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

                        System.out.println("CalculatorServer ready and waiting ...");

                        // wait for invocations from clients
                        orb.run();
                }
                catch (Exception ex) {
                        System.out.println("ERROR: " + ex);
                }
```

```
                System.out.println("CalculatorServer Exiting ...");

        }

}
```

CalculatorClient.java

```
import java.util.Scanner;

import CalcApp.*;

import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextExtPackage.*;
import org.omg.CORBA.*;

class CalcClient {
        static Calc calcImpl;

        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");
                // Use NamingContextExt instead of NamingContext. This is
                // part of the Interoperable naming Service.
                NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

                // resolve the Object Reference in Naming
                String name = "Calc";
                calcImpl = CalcHelper.narrow(ncRef.resolve_str(name));

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

                // Prompt the user to type
                boolean choose = true;
                while (choose) {
                        System.out.println("Please type in the order of \"opcode, op1 and op2\"");
                        String opcd;
                        int opcode;
                        int op1;
```

```
                    int op2;
                    Scanner in = new Scanner(System.in);
                    opcd = in.next();
                    op1 = in.nextInt();
                    op2 = in.nextInt();
                    opcode = opcd.charAt(0);
                    System.out.println("=" +  " " + calcImpl.calculate(opcode, op1, op2));
                    System.out.print("\nContinue Calculator? (Y/N)");
                    String chooseStr = in.next();
                    if (chooseStr.equals("Y") || chooseStr.equals("y"))
                    choose = true;
                    else
                            choose = false;
                }
                calcImpl.exit();


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

                }
        }
}
```

Calculator.idl

```
module CalcApp
{
  interface Calc
  {
        long calculate (in long opcode, in long op1, in long op2);
        oneway void exit();
  };
};
```