

CORBA – Eclipse

CSCI 201L

Jeffrey Miller, Ph.D.
jeffrey.miller@usc.edu

[HTTP://WWW-SCF.USC.EDU/~CSCI201](http://www-scf.usc.edu/~csci201)

Outline

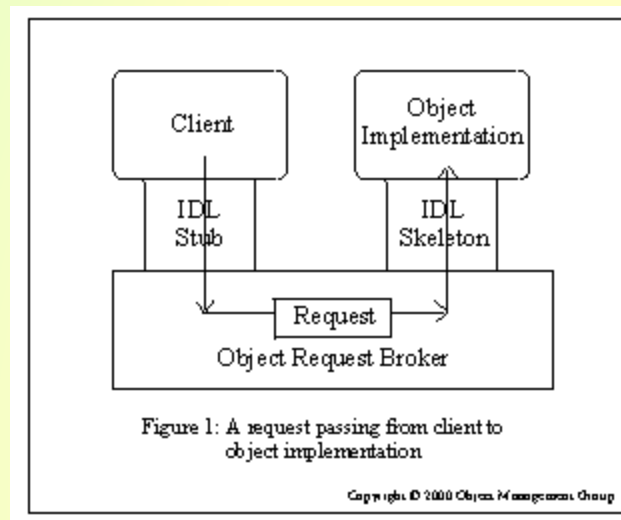
- CORBA
- Program

CORBA Overview

- The Common Object Request Broker Architecture (CORBA) is the Object Management Group's (OMG) open, vendor-independent architecture and infrastructure that computer applications use to work together over networks
 - › <http://www.omg.org/gettingstarted/corbafaq.htm>
- CORBA is operating system- and language-independent
- CORBA uses IDL to map programming language specifics to OMG standards, with mappings from IDL to C, C++, C++11, Java, Ruby, COBOL, Smalltalk, Ada, Lisp, Python, and IDLscript defined

CORBA Request

- CORBA clients communicate with an IDL stub, which is automatically generated from an IDL file
- CORBA servers communicate with an IDL skeleton, which is automatically generated from an IDL file
- The Object Request Broker (ORB) in the middle performs the communication and passes the data along in a standardized format from the stub to the skeleton



CORBA Application Example

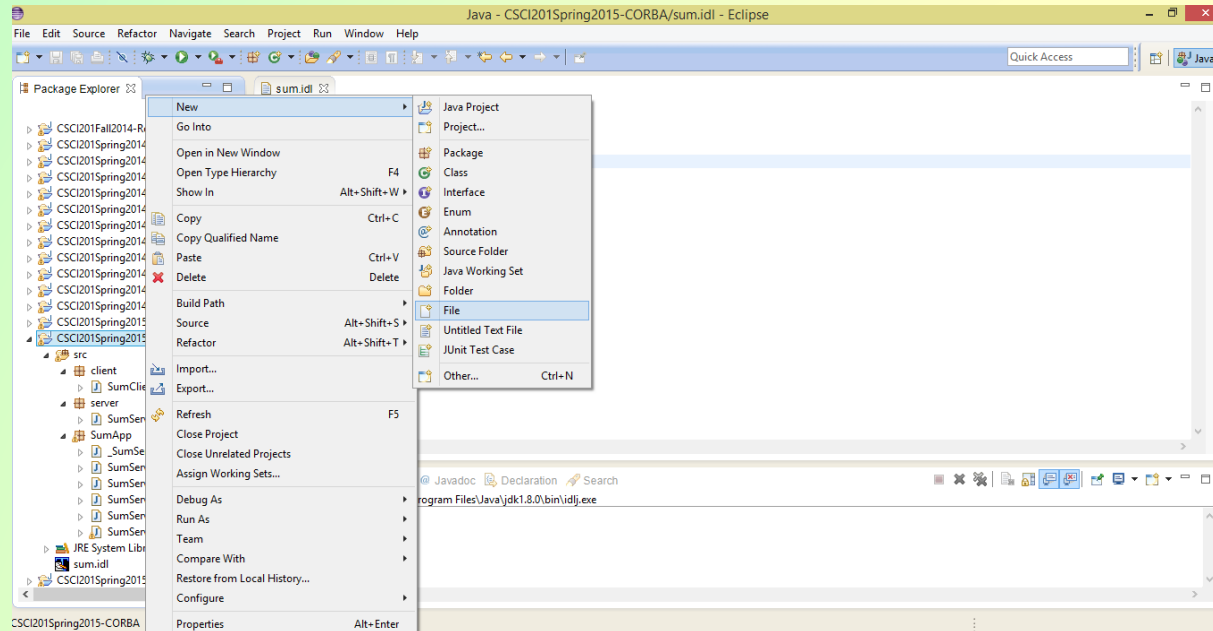
- Here are the steps for setting up a CORBA application
 1. Write the code
 - 1.1 Write the IDL file
 - 1.2 Generate the stub and skeleton code
 - 1.3 Write the server code
 - 1.4 Write the client code
 2. Compile the code
 3. Start the ORB server, server application, and client application

CORBA Application Example – Step 1.1

- Step 1.1 – Write the IDL file
 - › Create a project in Eclipse
 - › The IDL file defines the interface that will be used by the client and server for communicating and passing objects
 - › When the IDL file gets compiled, it will produce a number of files, known as the stub and skeleton
 - The stub is used by the client to communicate with the server
 - The skeleton is used by the server to communicate with the client
 - The stub and skeleton communicate with the ORB server to facilitate the remote procedure call
 - › The module in the IDL file will correspond to the package and directory in which the Java code will be generated

CORBA Application Example – Step 1.1

- Step 1.1 – Write the IDL file
 - › To create the IDL file in Eclipse, right-click on the project and select New->File
 - › Name the file sum.idl then type the code below



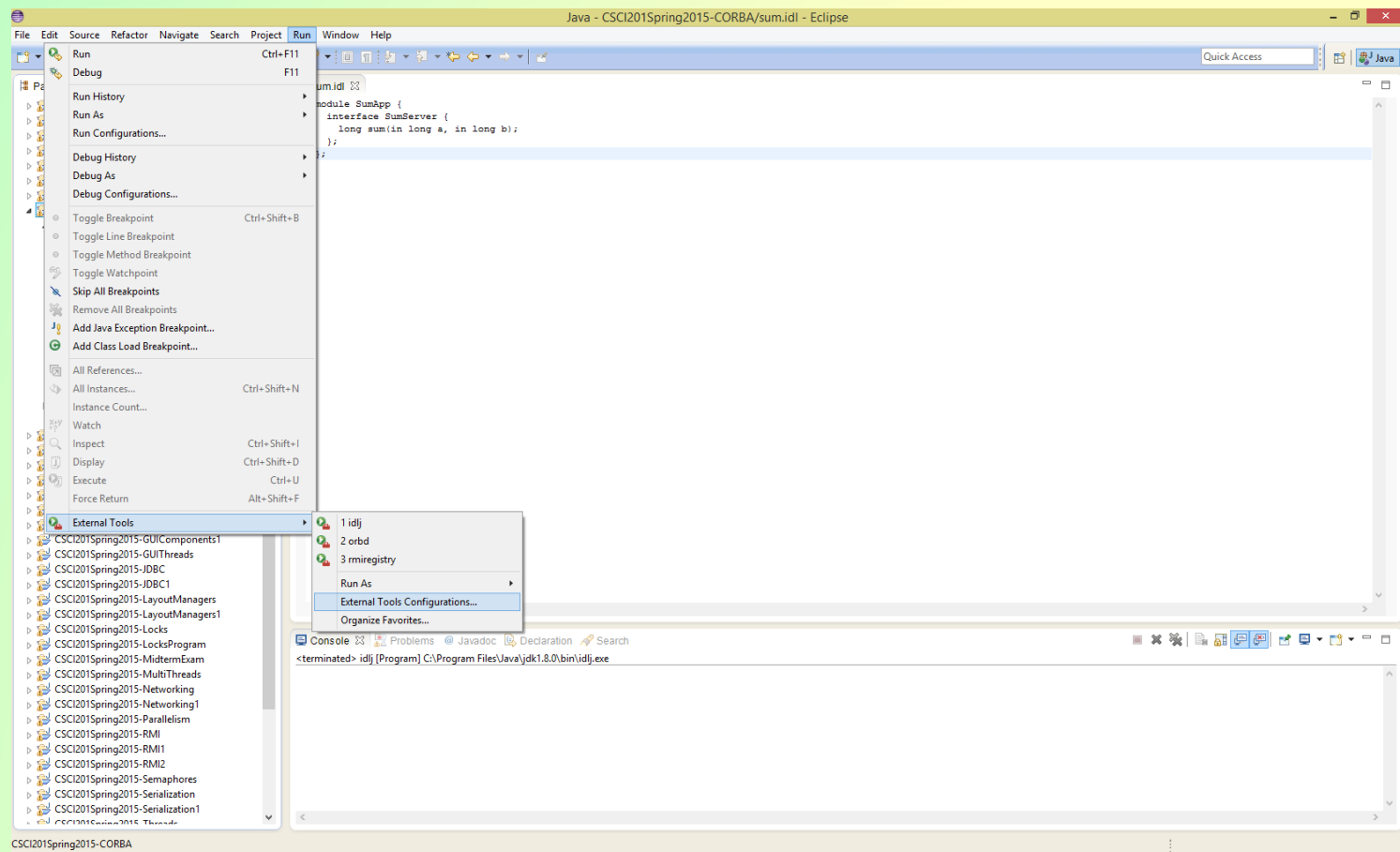
```
1  module SumApp {  
2      interface SumServer {  
3          long sum(in long a, in long b);  
4      };  
5  };
```

CORBA Application Example – Step 1.2

- Step 1.2 – Generate the stub and skeleton code
 - › There is an `idlj` program that comes with the JDK for generating the stub and skeleton code in Java
 - › The following files will be generated by the `idlj` program for the previous IDL file
 - `_SumServerStub.java`
 - `SumServer.java`
 - `SumServerHelper.java`
 - `SumServerHolder.java`
 - `SumServerOperations.java`
 - `SumServerPOA.java`

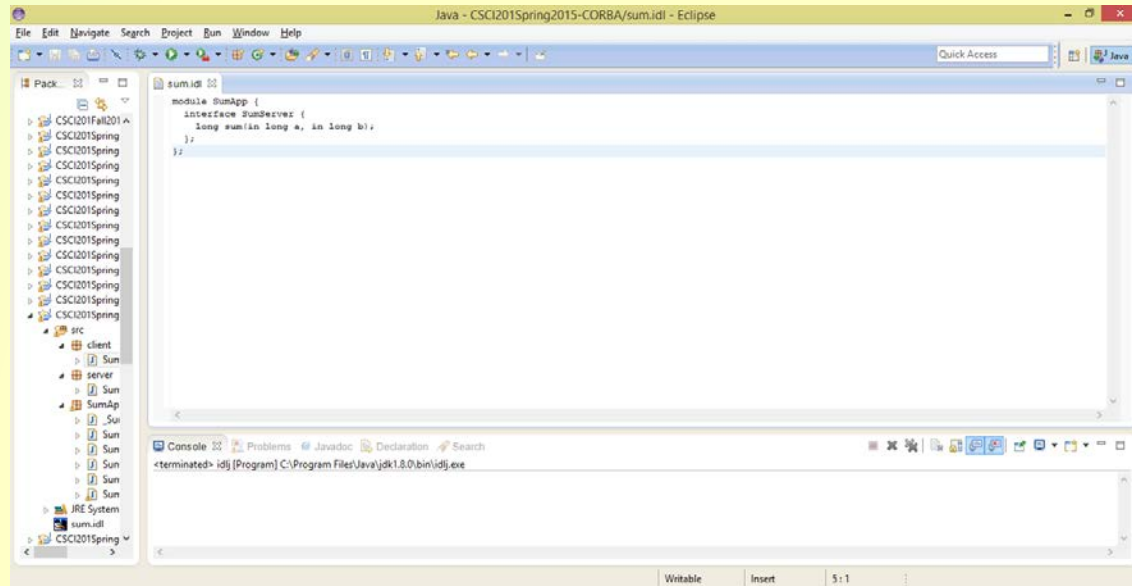
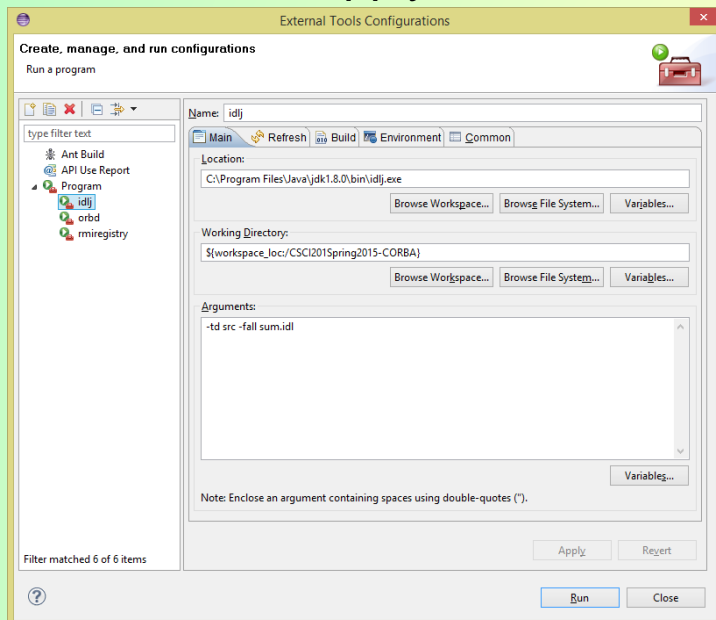
CORBA Application Example – Step 1.2

- Step 1.2 – Generate the stub and skeleton code
 - › To run **idlj** within Eclipse, go to “Run->External Tools->External Tools Configuration...”



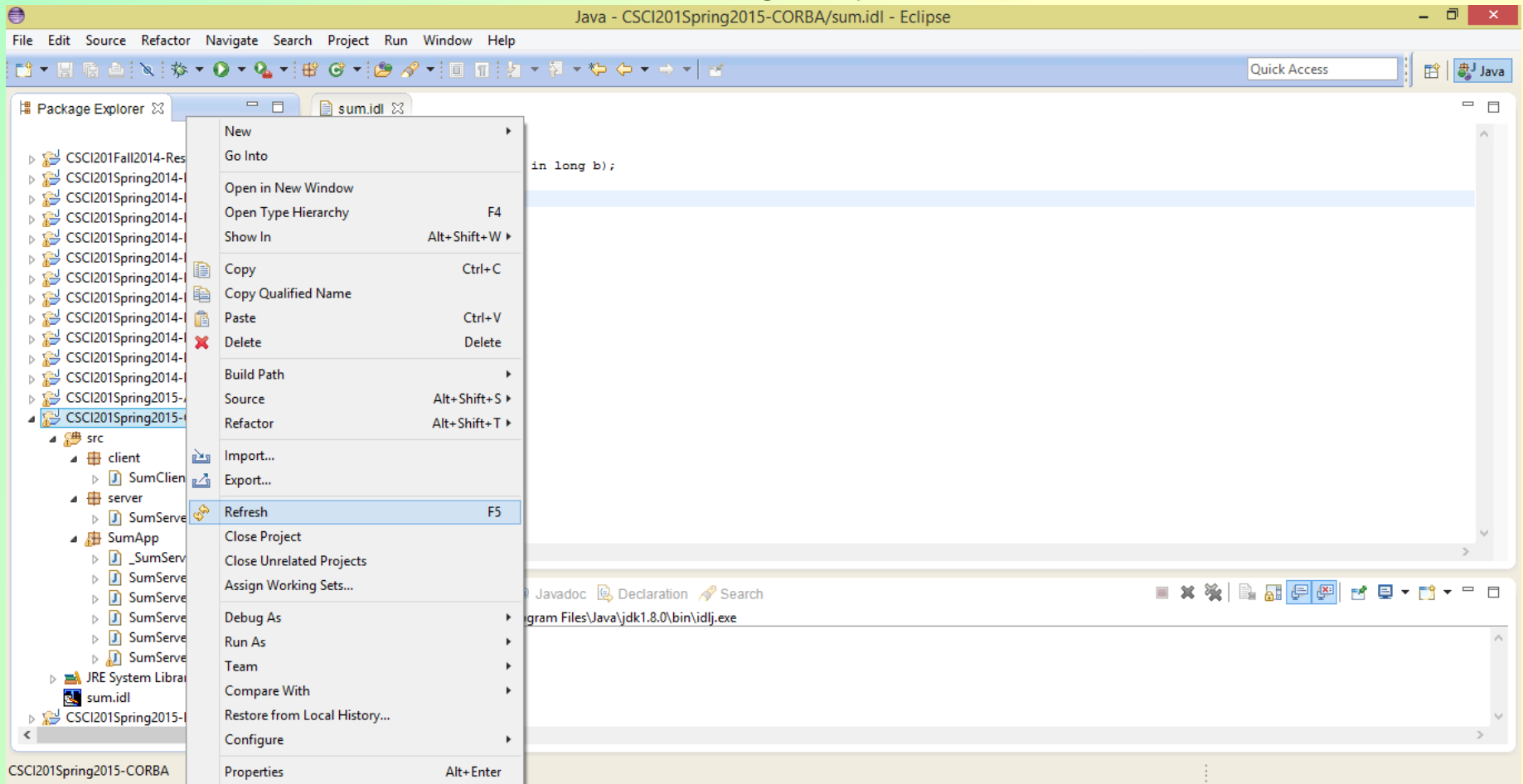
CORBA Application Example – Step 1.2

- Step 1.2 – Generate the stub and skeleton code
 - › Click the “New Launch Configuration” button and name it “idlj”
 - › For the “Location,” click “Browse File System” to find the **idlj** program in the bin directory of your JDK directory
 - › For the “Working Directory,” click “Browse Workspace” to find your project directory
 - › Type the following in the “Arguments” box
 - `-td src -fall sum.idl`
 - › Click “Apply” then “Run” – the stubs and skeletons should be generated



CORBA Application Example – Step 1.2

- Step 1.2 – Generate the stub and skeleton code
 - › The stubs and skeletons should have been generated in the src directory, so right-click on the project and click “Refresh” to see them
 - › You should see the SumApp package in your project now



CORBA Application Example – Step 1.3

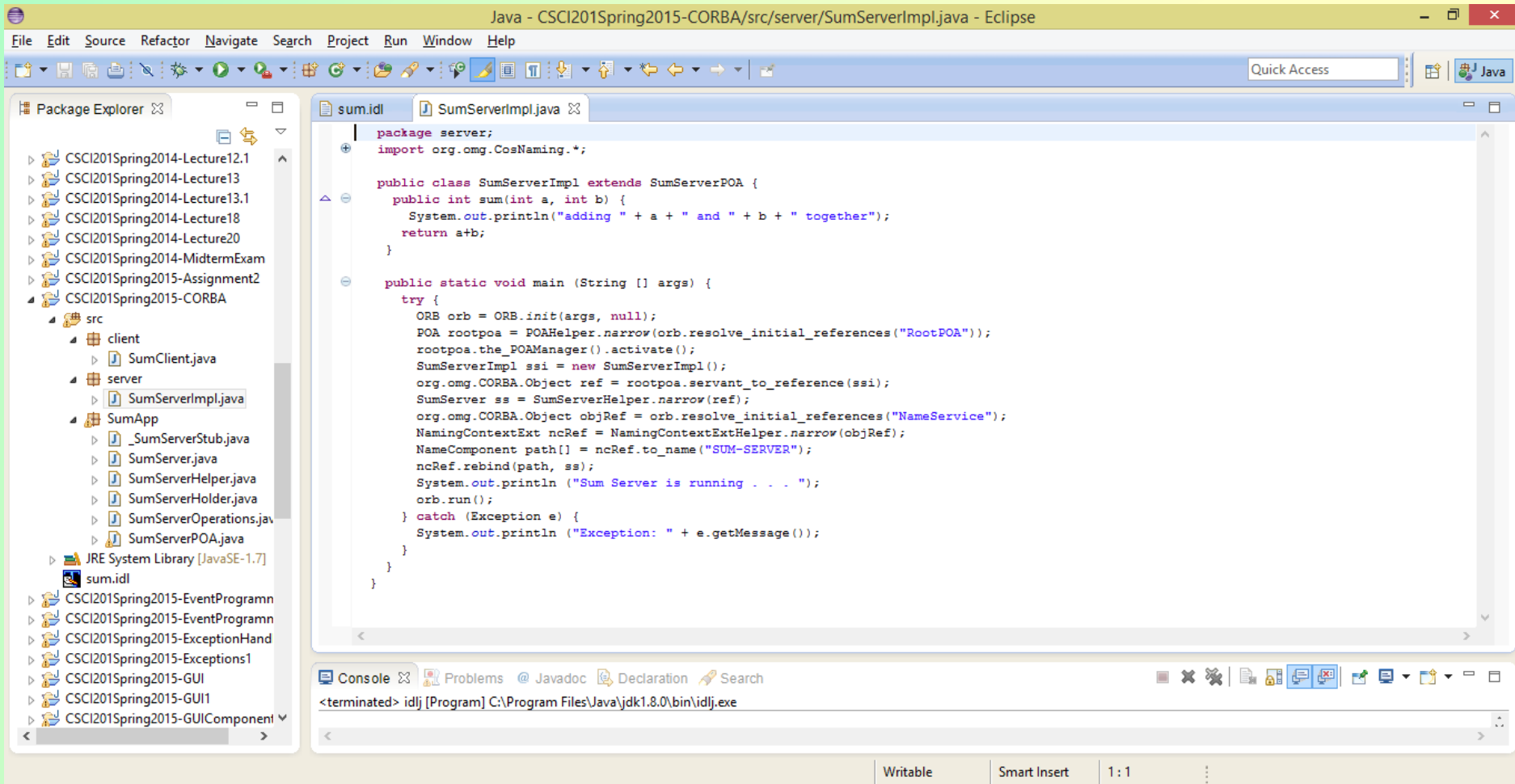
- Step 1.3 – Write the server code
 - › The server program will inherit from the `SumServerPOA` class that was generated as part of the `idlj` program
 - › The `SumServerPOA` class implements the `SumServerOperations` interface
 - This interface contains the methods we defined in the `sum.idl` file, but standardized to Java
 - › We create a main method in here also to communicate with the object request broker (ORB), registering the server with the ORB so clients are able to find it

CORBA Application Example – Step 1.3

```
1  package server;
2  import org.omg.CosNaming.*;
3  import org.omg.CORBA.*;
4  import org.omg.PortableServer.*;
5  import SumApp.*;
6
7  public class SumServerImpl extends SumServerPOA {
8      public int sum(int a, int b) {
9          System.out.println("adding " + a + " and " + b + " together");
10         return a+b;
11     }
12
13     public static void main (String [] args) {
14         try {
15             ORB orb = ORB.init(args, null);
16             POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
17             rootpoa.the_POAManager().activate();
18             SumServerImpl ssi = new SumServerImpl();
19             org.omg.CORBA.Object ref = rootpoa.servant_to_reference(ssi);
20             SumServer ss = SumServerHelper.narrow(ref);
21             org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
22             NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
23             NameComponent path[] = ncRef.to_name("SUM-SERVER");
24             ncRef.rebind(path, ss);
25             System.out.println ("Sum Server is running . . . ");
26             orb.run();
27         } catch (Exception e) {
28             System.out.println ("Exception: " + e.getMessage());
29         }
30     }
31 }
```

CORBA Application Example – Step 1.3

▪ Step 1.3 – Write the server code



CORBA Application Example – Step 1.4

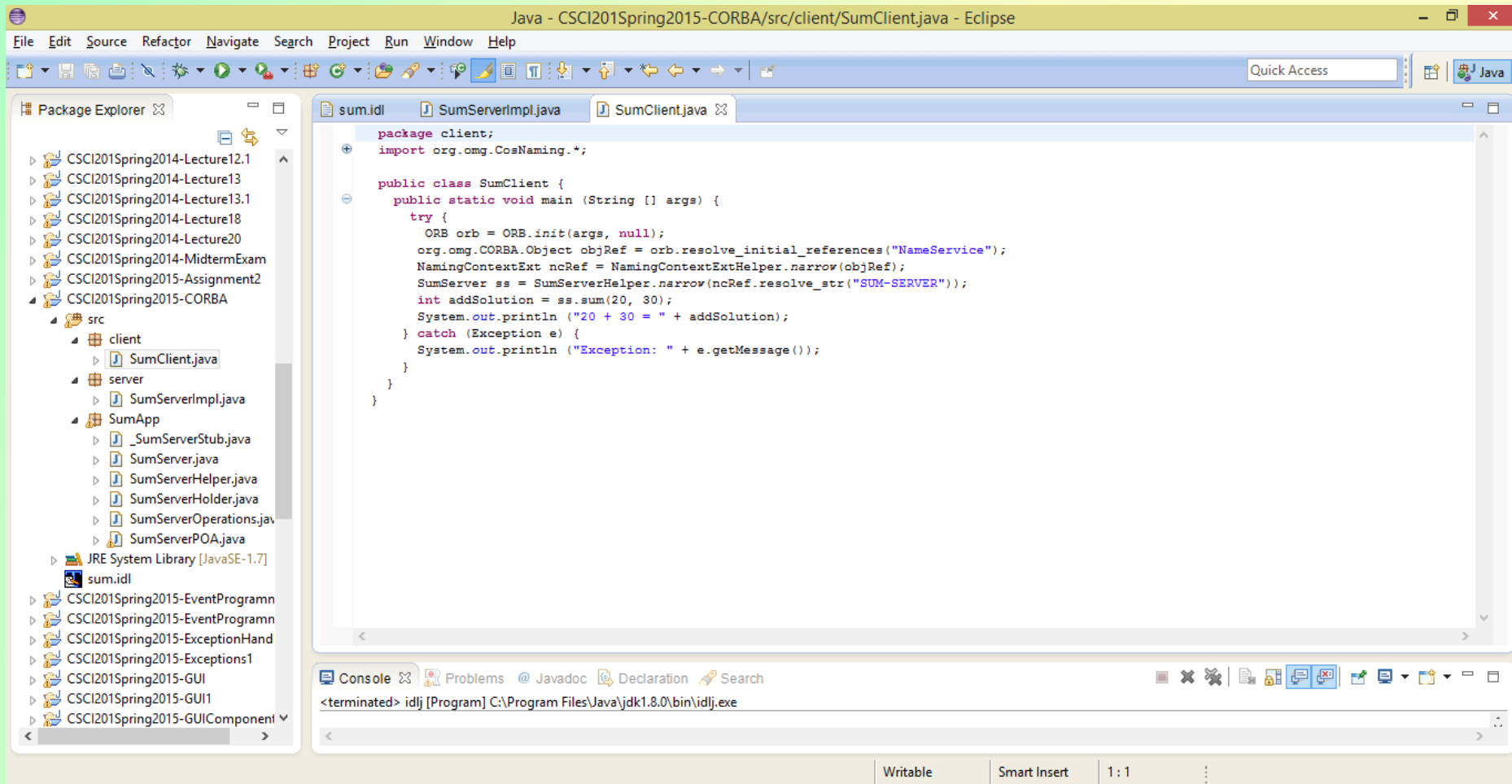
- Step 1.4 – Write the client code
 - › The client program needs to get a reference to the ORB then resolve the name of the server object it would like to invoke
 - This is **SUM-SERVER** in this case
 - › After getting an instance of a **SumServer** object from the server, it can invoke methods on it just like a method within its own JVM

CORBA Application Example – Step 1.4

```
1  package client;
2  import org.omg.CosNaming.*;
3  import org.omg.CORBA.*;
4  import SumApp.*;
5
6  public class SumClient {
7      public static void main (String [] args) {
8          try {
9              ORB orb = ORB.init(args, null);
10             org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
11             NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
12             SumServer ss = SumServerHelper.narrow(ncRef.resolve_str("SUM-SERVER"));
13             int addSolution = ss.sum(20, 30);
14             System.out.println ("20 + 30 = " + addSolution);
15         } catch (Exception e) {
16             System.out.println ("Exception: " + e.getMessage());
17         }
18     }
19 }
```


CORBA Application Example – Step 1.4

- Step 1.4 – Write the client code



CORBA Application Example – Step 2

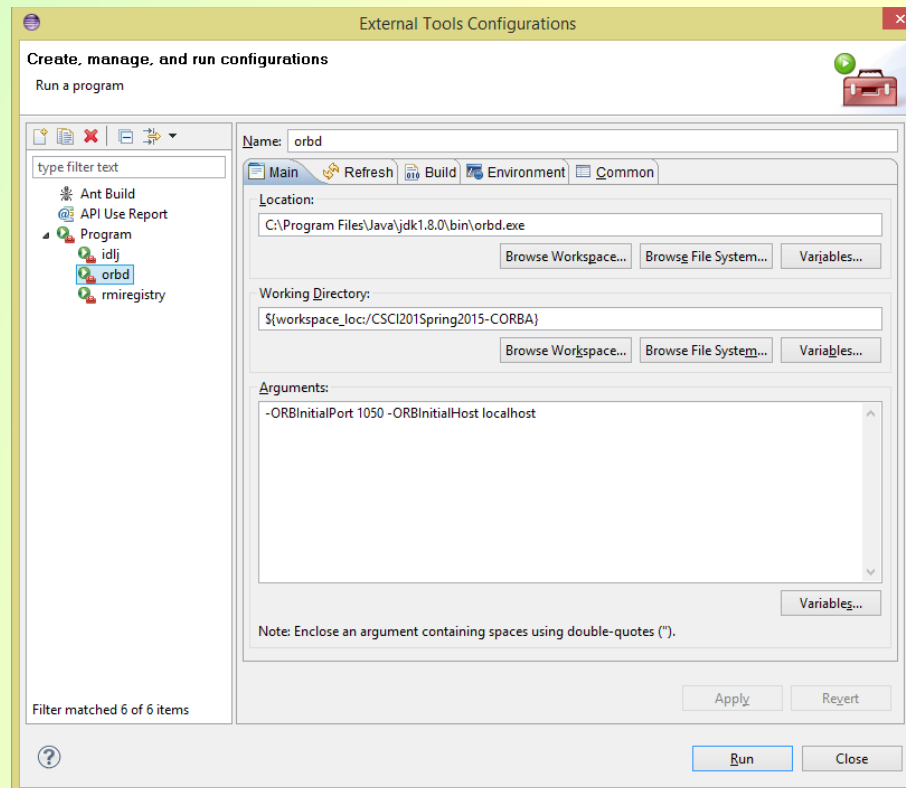
- Step 2 – Compile the code
 - › Eclipse compiles dynamically, so there is nothing else we need to do here

CORBA Application Example – Step 3

▪ Step 3.1 - Start the ORB server

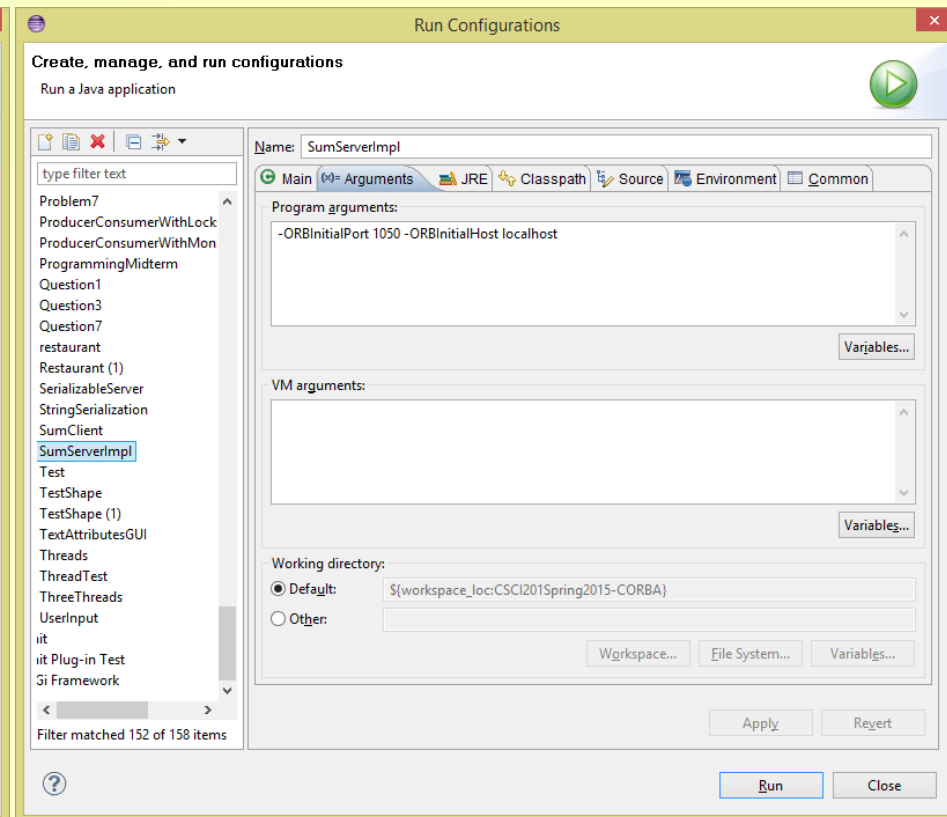
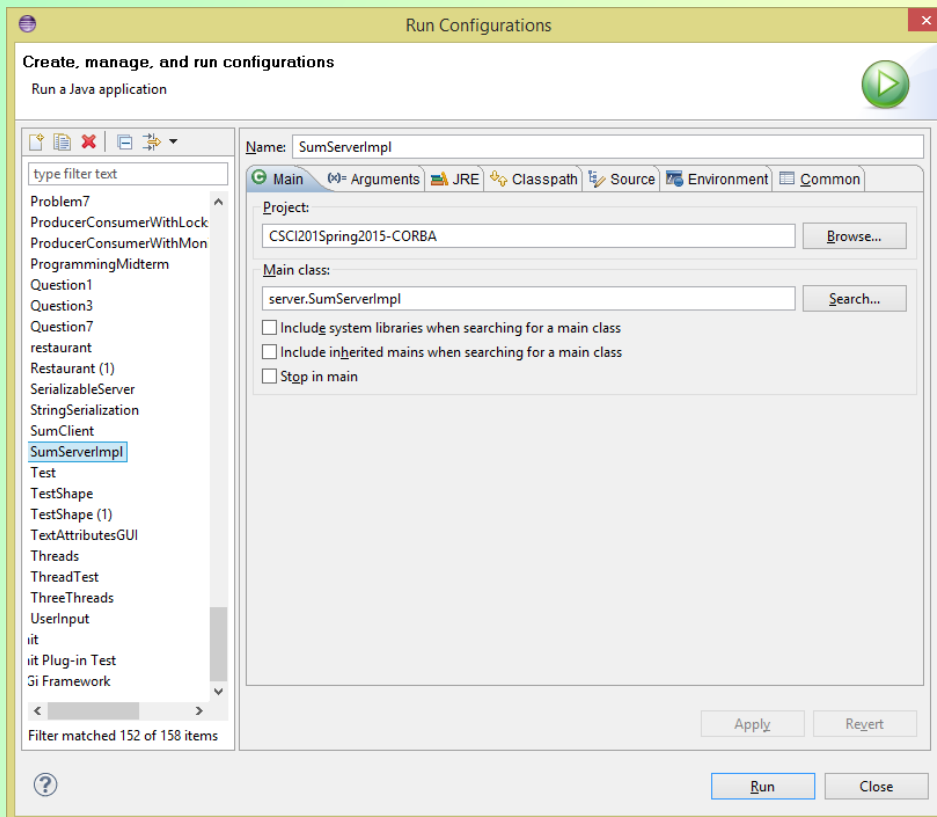
- › orbd is another program that comes with the JDK, so we need to configure it as an External Tool in Eclipse with the following arguments

```
-ORBInitialPort 1050 -ORBInitialHost localhost
```



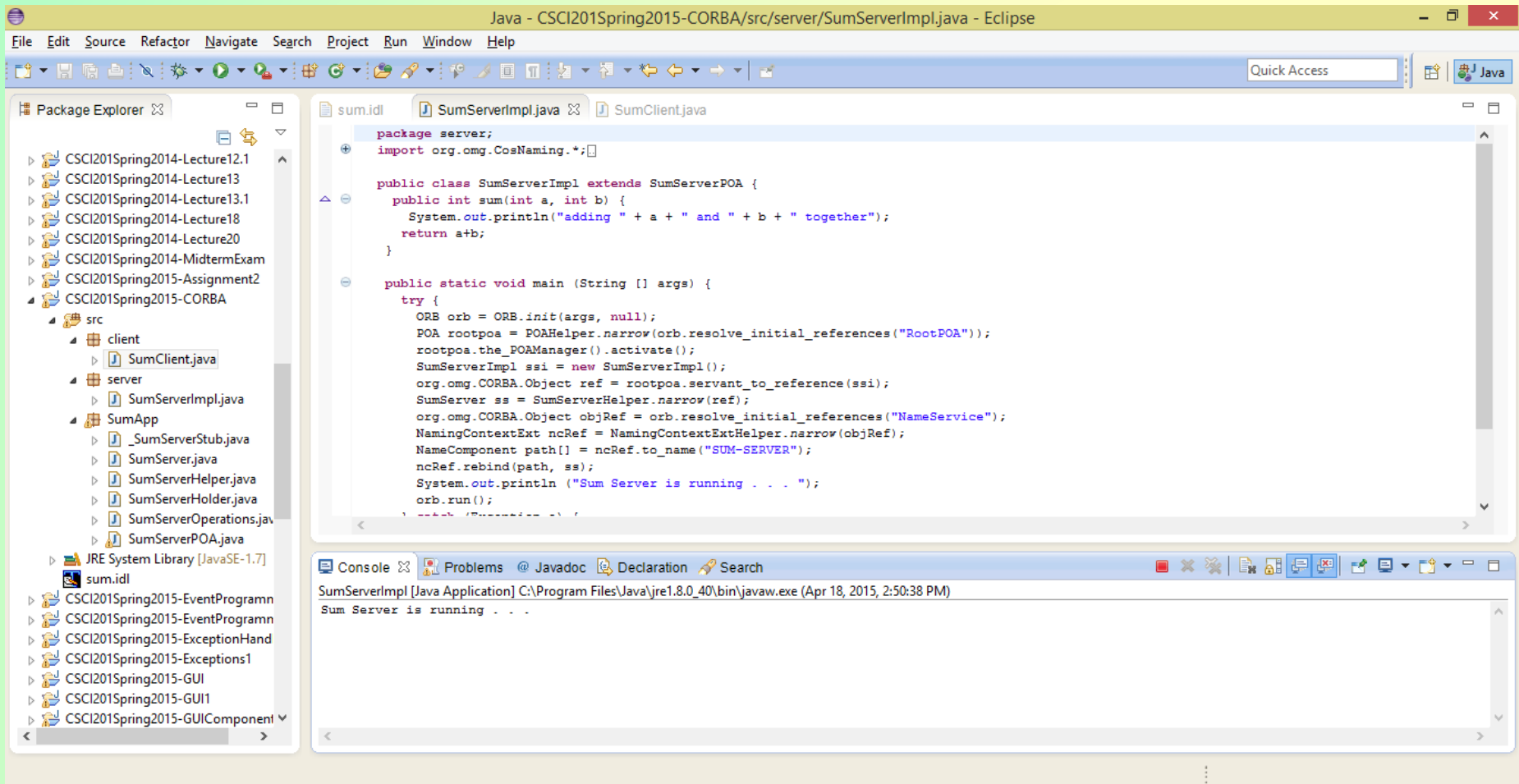
CORBA Application Example – Step 3

- Step 3.2 - Start the server application
 - Start the server application with the following command line arguments
`-ORBInitialPort 1050 -ORBInitialHost localhost`



CORBA Application Example – Step 3

- Step 3.2 - Start the server application
 - › If it starts successfully, you should see the following output



The screenshot displays the Eclipse IDE interface. The title bar indicates the active file is `SumServerImpl.java` in the `src/server` package. The Package Explorer on the left shows a project structure with a `src` folder containing `client`, `server`, and `SumApp` subfolders. The `server` folder contains `SumServerImpl.java`. The main editor window shows the source code of `SumServerImpl.java`, which implements the `SumServerPOA` interface. The code includes package declarations, imports, and a `main` method that initializes the ORB, resolves initial references, and starts the server. The console at the bottom shows the output of the `main` method, which prints `Sum Server is running . . .`.

```
package server;
import org.omg.CosNaming.*;

public class SumServerImpl extends SumServerPOA {
    public int sum(int a, int b) {
        System.out.println("adding " + a + " and " + b + " together");
        return a+b;
    }

    public static void main (String [] args) {
        try {
            ORB orb = ORB.init(args, null);
            POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
            rootpoa.the_POAManager().activate();
            SumServerImpl ssi = new SumServerImpl();
            org.omg.CORBA.Object ref = rootpoa.servant_to_reference(ssi);
            SumServer ss = SumServerHelper.narrow(ref);
            org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
            NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
            NameComponent path[] = ncRef.to_name("SUM-SERVER");
            ncRef.rebind(path, ss);
            System.out.println ("Sum Server is running . . . ");
            orb.run();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

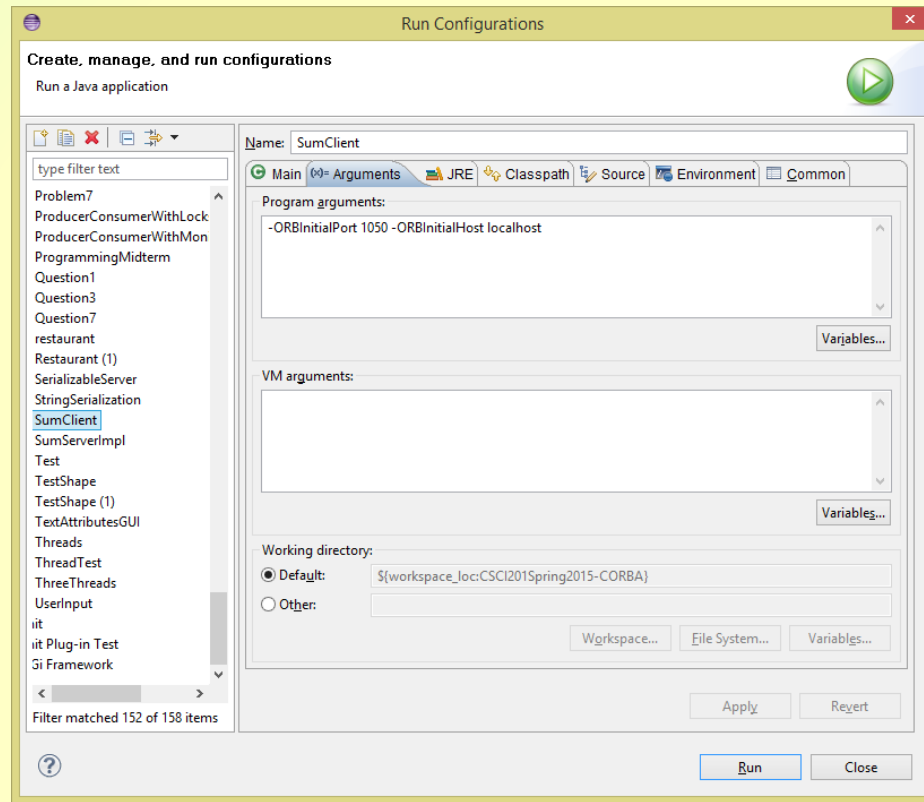
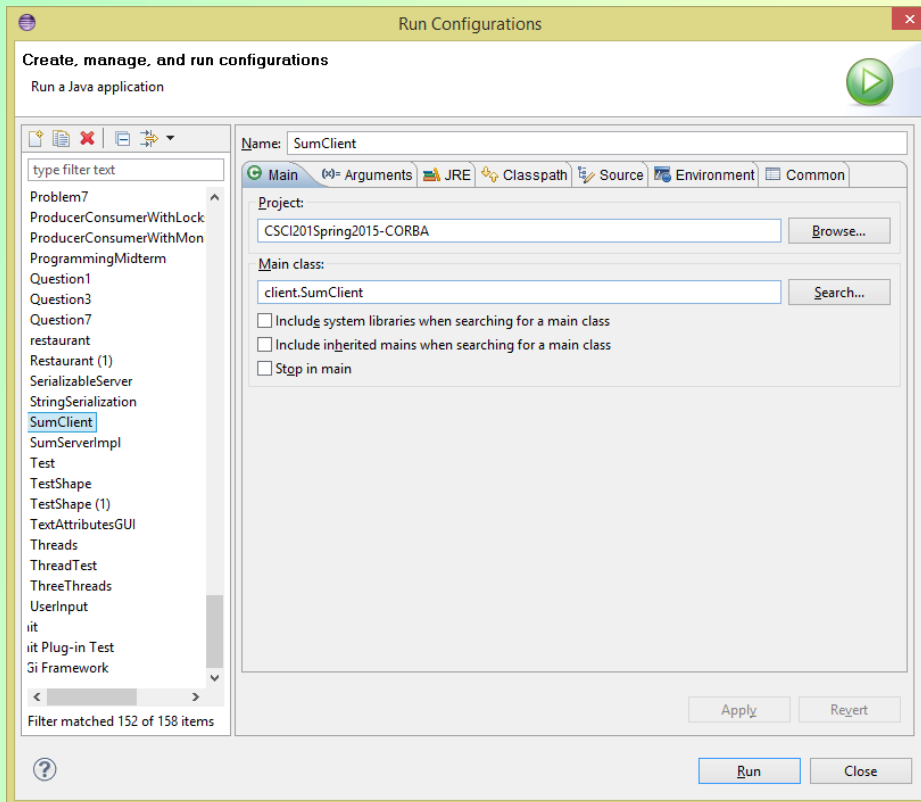
Console Output:

```
SumServerImpl [Java Application] C:\Program Files\Java\jre1.8.0_40\bin\javaw.exe (Apr 18, 2015, 2:50:38 PM)
Sum Server is running . . .
```

CORBA Application Example – Step 3

- Step 3.3 - Start the client application
 - Start the client application with the following command line arguments

`-ORBInitialPort 1050 -ORBInitialHost localhost`



CORBA Application Example – Step 3

- Step 3.3 - Start the client application
 - › If it runs successfully, you should see the following output

```
Java - CSCI201Spring2015-CORBA/src/client/SumClient.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Package Explorer
CSCI201Spring2014-Lecture12.1
CSCI201Spring2014-Lecture13
CSCI201Spring2014-Lecture13.1
CSCI201Spring2014-Lecture18
CSCI201Spring2014-Lecture20
CSCI201Spring2014-MidtermExam
CSCI201Spring2015-Assignment2
CSCI201Spring2015-CORBA
  src
    client
      SumClient.java
    server
      SumServerImpl.java
    SumApp
      _SumServerStub.java
      SumServer.java
      SumServerHelper.java
      SumServerHolder.java
      SumServerOperations.java
      SumServerPOA.java
  JRE System Library [JavaSE-1.7]
  sum.idl
  CSCI201Spring2015-EventProgramn
  CSCI201Spring2015-EventProgramn
  CSCI201Spring2015-ExceptionHand
  CSCI201Spring2015-Exceptions1
  CSCI201Spring2015-GUI
  CSCI201Spring2015-GUI1
  CSCI201Spring2015-GUIComponent
sum.idl SumServerImpl.java SumClient.java
package client;
import org.omg.CosNaming.*;

public class SumClient {
    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);
            SumServer ss = SumServerHelper.narrow(ncRef.resolve_str("SUM-SERVER"));
            int addSolution = ss.sum(20, 30);
            System.out.println ("20 + 30 = " + addSolution);
        } catch (Exception e) {
            System.out.println ("Exception: " + e.getMessage());
        }
    }
}
Console Problems @ Javadoc Declaration Search
<terminated> SumClient [Java Application] C:\Program Files\Java\jre1.8.0_40\bin\javaw.exe (Apr 18, 2015, 2:52:21 PM)
20 + 30 = 50
Writable Smart Insert 7:26
```

Outline

- Interface Definition Language
- CORBA
- Program

Program

- Create a CORBA program where the CORBA server object communicates over RMI to find the value of π to a certain number of digits
- Utilize the RMI code from the previous lecture