WELCOME TO TUTORIAL 3 OF COMP 6231

DISTRIBUTED SYSTEMS DESIGN

KNOW YOUR TA – Harsh Deep Kour
harsh.comp6231@gmail.com

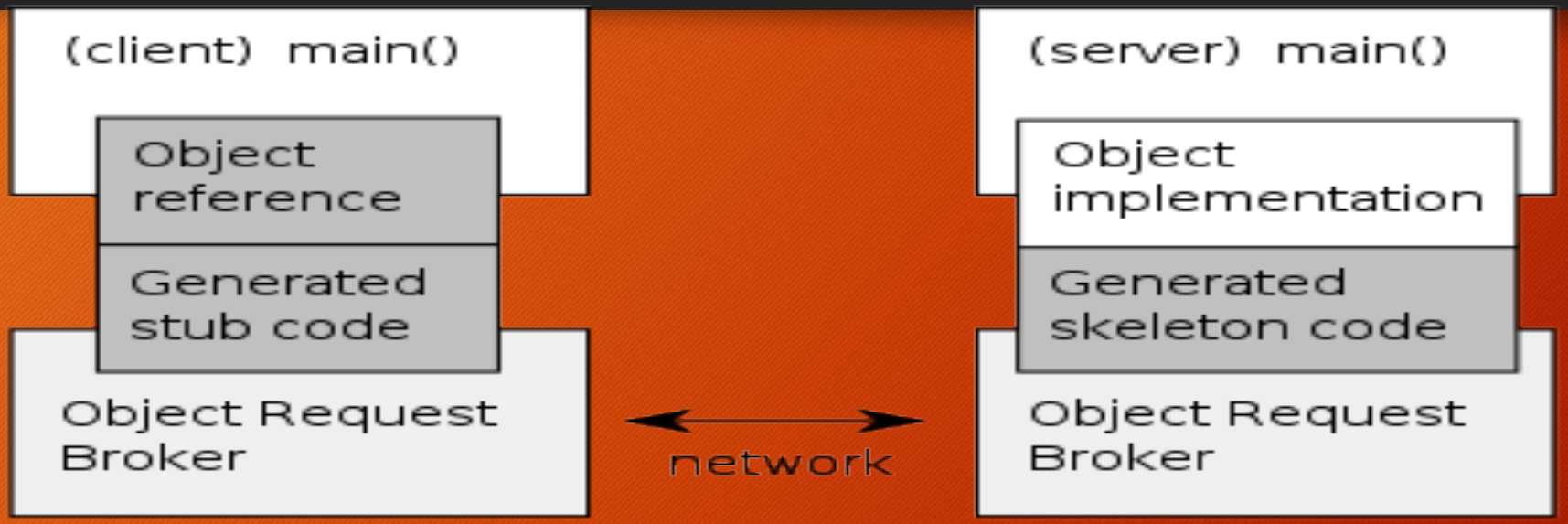
TODAY'S TOPIC

CORBA PROGRAMMING

COMMON OBJECT REQUEST BROKER ARCHITECTURE (CORBA)

- CORBA designed to facilitate the communication system between devices that are designed on diverse platform.
- CORBA is a standard defined by the Object Management Group(OMG) in the year of 1991.
- It enables the collaboration between systems irrespective of operating system , programming language and hardware.

CORBA STRUCTURE



OVERVIEW OF STRUCTURE

- Server: It has object implementation code and logic for generating skeleton.
- Client: It has reference of all the objects and logic for generating stub.
- Object Request Broker(ORB):

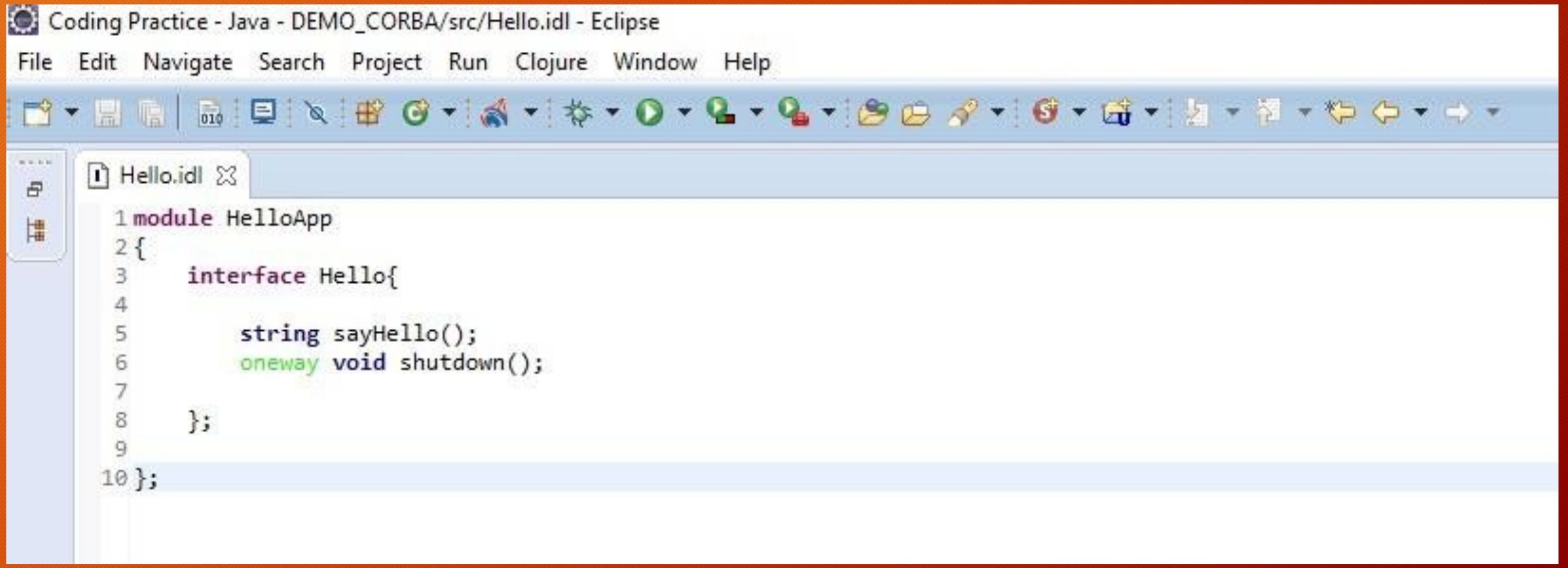
1) It’s implemented both the side , it takes care of routing all the request from client to server and response from server to client.

2) Client-side, it contains interface definition.

3) Server-side, it handles activation/deactivation of objects.

STEP 1: RUN CORBA ON SYSTEM

1. Create file with .idl extension inside java project directory.



STEP 1: CONTINUED

Some common variable type for IDL:

- boolean
- string
- any
- char
- float
- TRUE
- FALSE
- in
- inout

STEP 2: GENERATE THE BINDINGS

1)Open your CMD console, Change directory(cd) to **src** folderwithin the project location.

The location of your project can be known through clicking: Select the project DEMO_CORBA(in this case), click : **File -> Properties**

2) Compile that IDLfile using following command on cmd->
Idlj -fall Hello.idl

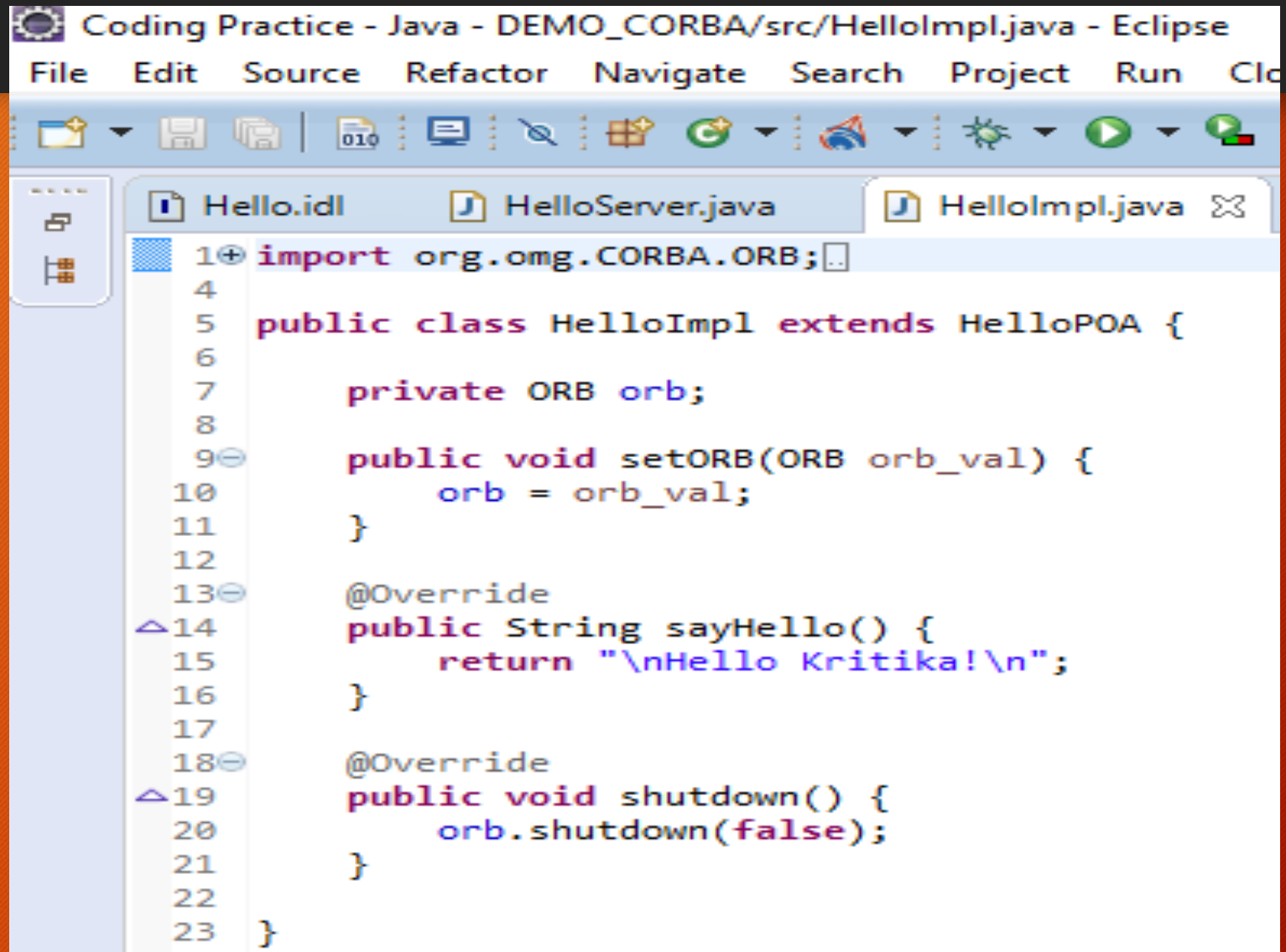
3) This will generate both client side and server side bindings.

STEP 2: CONTINUED

2.4) Once you successfully run this command, it will create a folder, which has 6 following files:

HelloPOA.java -> Server Skeleton
_HelloStub.java -> Client Stub
Hello.java -> java version of our idl interface file
HelloHelper.java -> this cast object reference to their proper types
HelloHolder.java -> it holds public instance of type Hello.
HelloOperations.java -> it contains methods that declared in interface.

STEP 3.1: SERVER-SIDE IMPLEMENTATION(IMPLEMENT THE SERVER IMPLEMENTATION CLASS)



STEP 3.2: SERVER-SIDE IMPLEMENTATION(IMPLEMENT THE SERVER)

```
1  import org.omg.CORBA.ORB;
16
17 public class HelloServer {
18
19     public static void main(String[] args) {
20         //create and initialize Orb
21         ORB orb = ORB.init(args, null);
22         //get reference to root POA and activate the POA Manager
23         try {
24             POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
25             rootpoa.the_POAManager().activate();
26
27             //create servant and register it with ORB
28             HelloImpl helloImpl = new HelloImpl();
29             helloImpl.setORB(orb);
30
31             //get object reference from servant
32             org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
33             Hello href = HelloHelper.narrow(ref);
34
35             //get root naming context
36             //NameService invokes the name service
37
38             org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
39
40             //Use NamingContextExt which is part of the Interoperable Naming Service(INS) specification.
41             NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
42
43             //bind the Object Reference in Naming
44             String name = "Hello";
45
46             NameComponent path[] = ncRef.to_name(name);
47             ncRef.rebind(path, href);
48             System.out.println("Hello Server Ready and Waiting.....");
49
50             //wait for invocation from clients
51             orb.run();
52
```

STEP 3.2: CONTINUED

```
52
53          } catch (InvalidName e) {
54              // TODO Auto-generated catch block
55              e.printStackTrace();
56          } catch (AdapterInactive e) {
57              // TODO Auto-generated catch block
58              e.printStackTrace();
59          } catch (ServantNotActive e) {
60              // TODO Auto-generated catch block
61              e.printStackTrace();
62          } catch (WrongPolicy e) {
63              // TODO Auto-generated catch block
64              e.printStackTrace();
65          } catch (org.omg.CosNaming.NamingContextPackage.InvalidName e) {
66              // TODO Auto-generated catch block
67              e.printStackTrace();
68          } catch (NotFound e) {
69              // TODO Auto-generated catch block
70              e.printStackTrace();
71          } catch (CannotProceed e) {
72              // TODO Auto-generated catch block
73              e.printStackTrace();
74          }
75      }
76  }
77
```

SERVER-SIDE CODE: SUMMARY

The HelloServer class has the server's main() method, which:

- Creates and initializes an ORB instance
- Gets a reference to the root POA and activates the POAManager
- Creates a servant instance (the implementation of one CORBA Hello object) and tells the ORB about it
- Gets a CORBA object reference for a naming context in which to register the new CORBA object
- Gets the root naming context
- Registers the new object in the naming context under the name "Hello"
- Waits for invocations of the new object from the client

STEP 4: IMPLEMENT THE CLIENT

Hello.idl | HelloImpl.java | HelloServer.java | HelloClient.java

```
1  package HelloApp;
2  import org.omg.CORBA.ORB;
8
9  public class HelloClient {
10     static Hello helloImpl;
11     public static void main(String[] args){
12
13         //create and Initialize the ORB
14         ORB orb = ORB.init(args,null);
15         try {
16             //get the root naming context
17             org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
18
19             //Use NamingContextExt instead of NamingContext. This is part of Interoperable Naming Service
20             NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
21
22             //resolve the object reference in naming
23             String name = "Hello";
24
25             helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));
26             System.out.println("Obtain a handle on Server Object: "+ helloImpl);
27             System.out.println(helloImpl.sayHello());
28             //helloImpl.shutdown();
29         } catch (InvalidName e) {
30             e.printStackTrace();
31         } catch (NotFound e) {
32             e.printStackTrace();
33         } catch (CannotProceed e) {
34             e.printStackTrace();
35         } catch (org.omg.CosNaming.NamingContextPackage.InvalidName e) {
36             e.printStackTrace();
37         }
38     }
39 }
```

CLIENT-SIDE CODE: SUMMARY

- Creates and initializes an ORB
- Obtains a reference to the root naming context
- Looks up "Hello" in the naming context and receives a reference to that CORBA object
- Invokes the object's sayHello() operation and prints the result

STEP 5(A): RUN THE PROGRAM THROUGH CMD

1. Compile all the files: `javac *.java HelloApp/*.java`
2. Run ORBD through cmd: `start orbd -ORBInitialPort 1050`
3. Start Server: `start java HelloServer -ORBInitialPort 1050 -ORBInitialHost localhost`
4. Start Client: `java HelloClient -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: IOR: (binary code) Hello Kritika!`

STEP 5(B): RUN THE PROGRAM THROUGH ECLIPSE

CORBA ‘Hello World’ using Java:

http://www.ejbtutorial.com/corba/tutorial-for-corba-hello-world-using-java

FOLLOW THESE STEPS PROVIDED WITH PICTORIAL REPRESENATTION FOR RUNNING YOUR APPLICATION ON ECLIPSE.

THANK YOU
THAT’S IT FOR TODAY☺️