CORBA

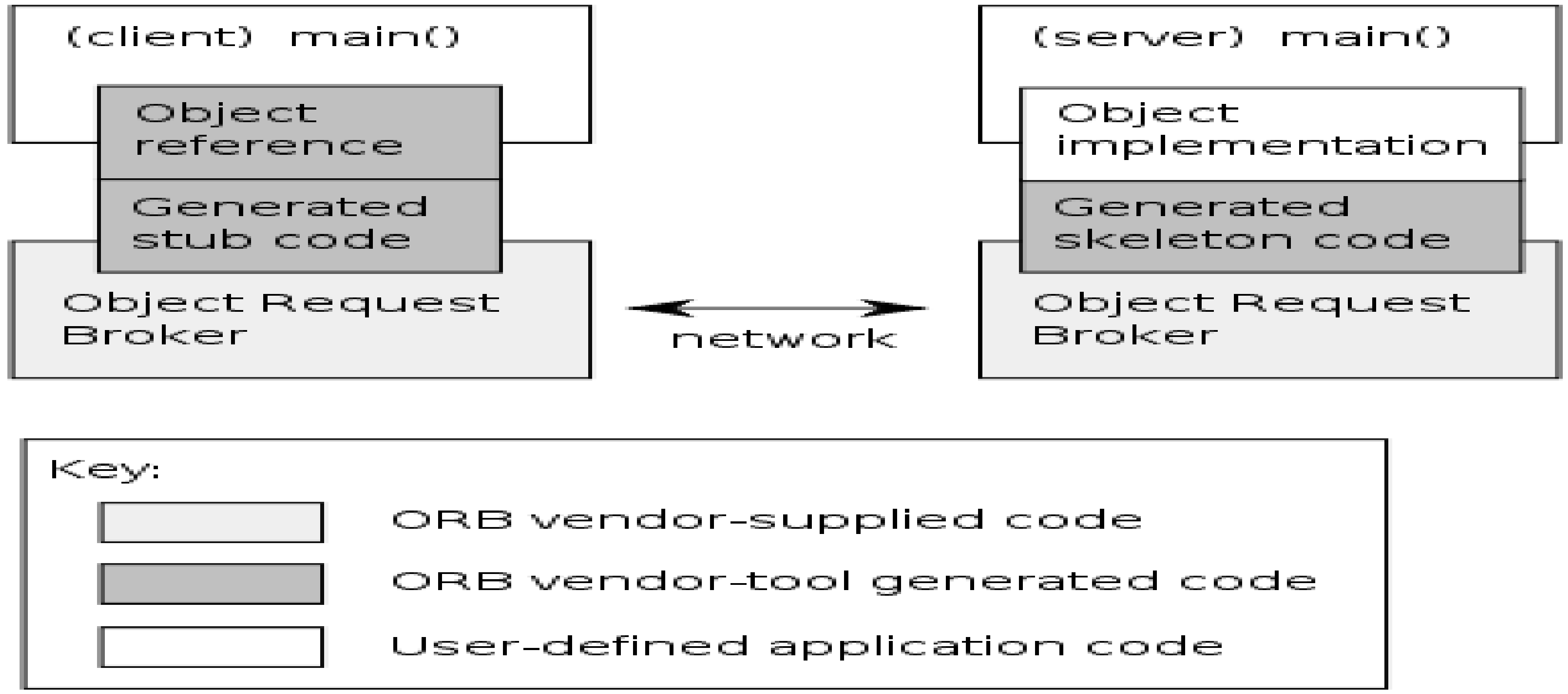
COMMON OBJECT REQUEST BROKER ARCHITECTURE (CORBA)

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:

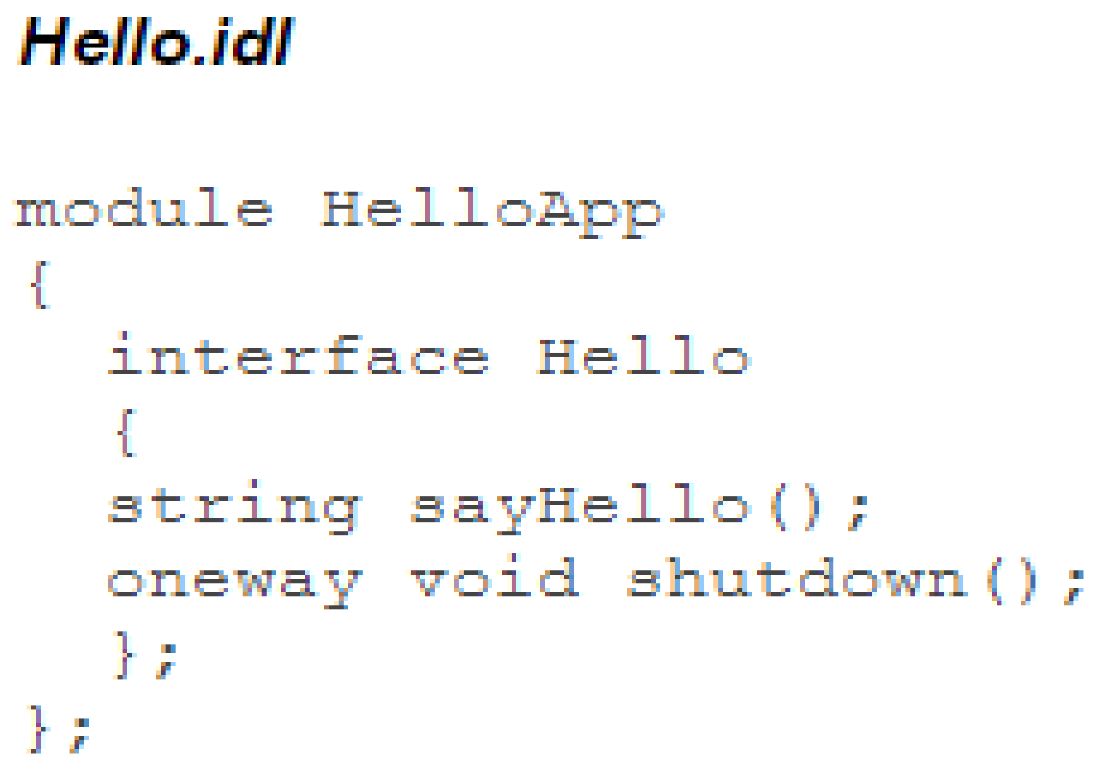


OVERVIEW OF STRUCTURE:

- 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.



Hello.idl

```
module HelloApp
{
   interface Hello
   {
   string sayHello();
   oneway void shutdown();
   };
};
```

STEP-1: CONTINUE

Some common variable type for IDL:

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

STEP-2

2. Compile that IDL file using following command on cmd.

   Idlj –fall Hello.idl

3. This will generate both client side and server side bindings.
4. Once you successfully run this command, it will create a folder and 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: SERVER-SIDE IMPLEMENTATION (1)

```
// HelloServer.java
// Copyright and License
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
```

STEP-3: SERVER-SIDE IMPLEMENTATION (2)

```
import java.util.Properties;
class HelloImpl extends HelloPOA {
private ORB orb;
public void setORB(ORB orb_val) {
orb = orb_val;
 }
// implement sayHello() method
 public String sayHello() {
   return "\nHello world !!\n";
 }
   // implement shutdown() method
 public void shutdown() {
   orb.shutdown(false);
 } }
```

STEP-3: SERVER-SIDE IMPLEMENTATION (3)

```
public class HelloServer {
 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
    HelloImpl helloImpl = new HelloImpl();
    helloImpl.setORB(orb);
```

STEP-3: SERVER-SIDE IMPLEMENTATION (4)

```
 // get object reference from the servant
org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
Hello href = HelloHelper.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);
```

STEP-3: SERVER-SIDE IMPLEMENTATION (5)

```
// bind the Object Reference in Naming
 String name = "Hello";
 NameComponent path[] = ncRef.to_name( name );
 ncRef.rebind(path, href);
  System.out.println("HelloServer ready and waiting ...");
 // wait for invocations from clients
 orb.run();
}
   catch (Exception e) {
   System.err.println("ERROR: " + e);
   e.printStackTrace(System.out);
 }
     System.out.println("HelloServer Exiting ...");
  }}
```

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: CLIENT-SIDE IMPLEMENTATION (1)

```
//HelloClient.java
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;

public class HelloClient
{
  static Hello helloImpl;
```

STEP-4: CLIENT-SIDE IMPLEMENTATION (2)

```
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);
```

STEP-4: CLIENT-SIDE IMPLEMENTATION (3)

```
// resolve the Object Reference in Naming
    String name = "Hello";
    helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));
    System.out.println("Obtained a handle on server object: " + helloImpl);
    System.out.println(helloImpl.sayHello());
    helloImpl.shutdown();
    } catch (Exception e) {
      System.out.println("ERROR : " + e) ;
      e.printStackTrace(System.out);
      }
  }}
```

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() and shutdown() operations and prints the result

STEP-5: RUN THE PROGRAM THROUGH CMD

1. Compile all the files: `javac *.java HelloApp/*.java`

2. Run ORBD through cmd: `start orbd -ORBInitialPort 1050`

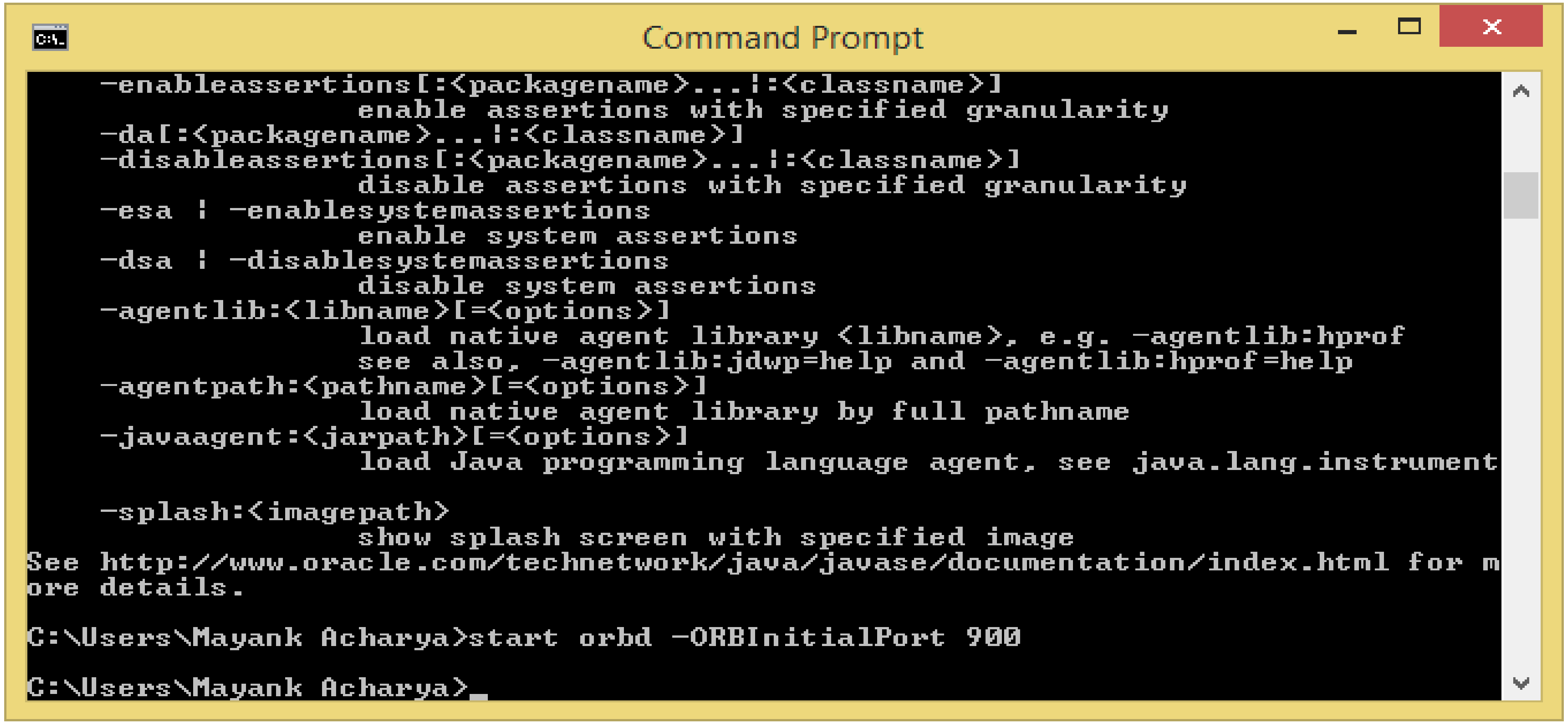
3. Start Server: `start java HelloServer -ORBInitialPort 1050 -ORBInitialHost localhost`

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 World! HelloServer exiting...
```

STEP-5: RUN THE PROGRAM THROUGH ECLIPSE



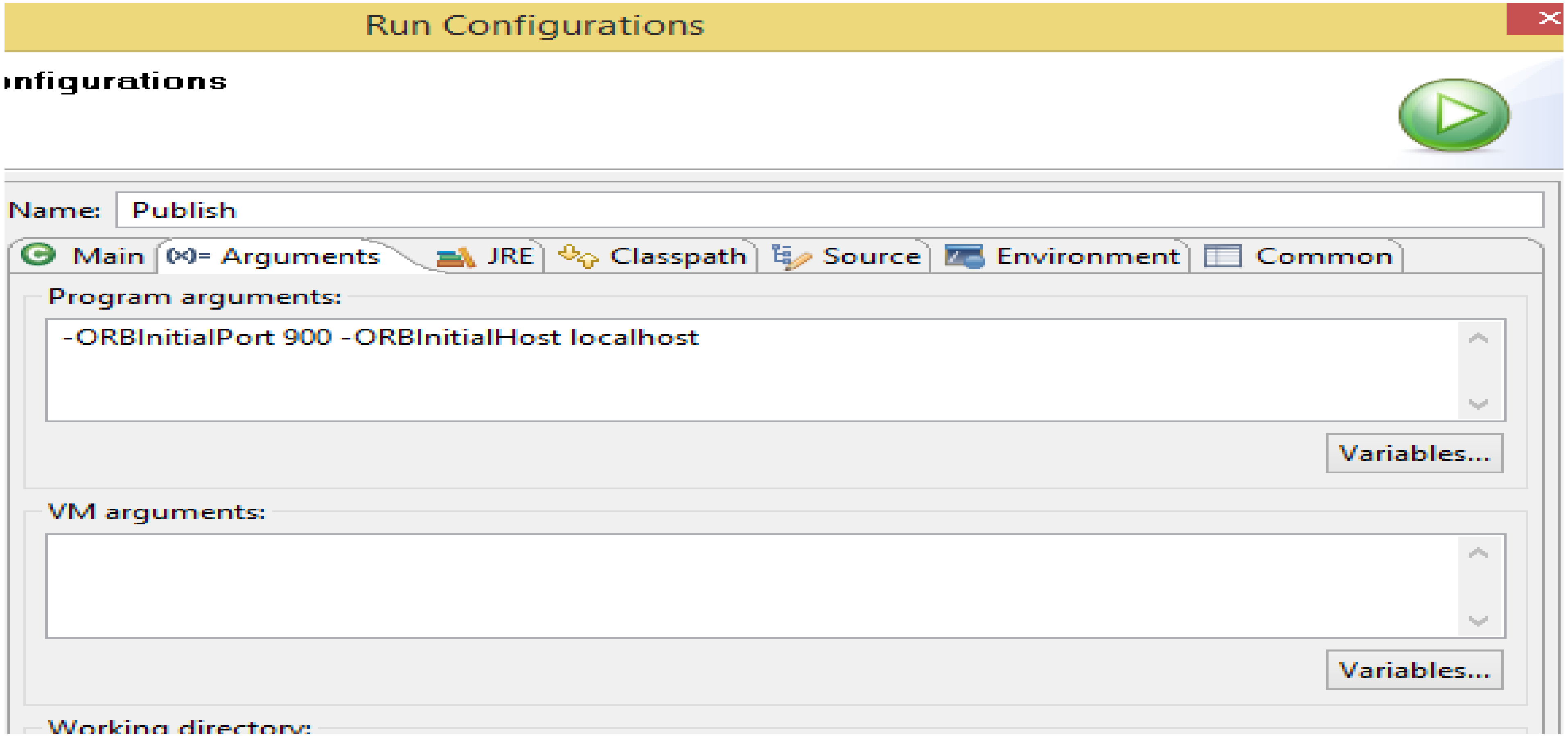
```
    -enableassertions[:<packagename>...|:<classname>]
                  enable assertions with specified granularity
    -da[:<packagename>...|:<classname>]
    -disableassertions[:<packagename>...|:<classname>]
                  disable assertions with specified granularity
    -esa | -enablesystemassertions
                  enable system assertions
    -dsa | -disablesystemassertions
                  disable system assertions
    -agentlib:<libname>[=<options>]
                  load native agent library <libname>, e.g. -agentlib:hprof
                  see also, -agentlib:jdwp=help and -agentlib:hprof=help
    -agentpath:<pathname>[=<options>]
                  load native agent library by full pathname
    -javaagent:<jarpath>[=<options>]
                  load Java programming language agent, see java.lang.instrument

    -splash:<imagepath>
                  show splash screen with specified image
See http://www.oracle.com/technetwork/java/javase/documentation/index.html for m
ore details.

C:\Users\Mayank Acharya>start orbd -ORBInitialPort 900

C:\Users\Mayank Acharya>_
```

STEP-5: RUN THE PROGRAM THROUGH ECLIPSE



Name: Publish

Program arguments:

```
-ORBInitialPort 900 -ORBInitialHost localhost
```

VM arguments:

REFERENCE LINK:

Oracle CORBA Tutorial:

- http://docs.oracle.com/javase/7/docs/technotes/guides/idl/jidlExample.html

CORBA ‘Hello World’ using Java:

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