# Detailed Explanation of CORBA Assignment – Reverse String Application

## 1. What is CORBA?

CORBA (Common Object Request Broker Architecture) is a middleware standard developed by the Object Management Group (OMG). It allows objects written in different programming languages and running on different machines to communicate and invoke methods on each other as if they were local.

## 2. CORBA Architecture

Key Components:  
- ORB (Object Request Broker): Manages communication between client and server.  
- IDL (Interface Definition Language): Language-independent interface definition.  
- Stub and Skeleton: Auto-generated code to bridge client/server and ORB.  
- Naming Service: Registers services so that clients can find them.  
- POA (Portable Object Adapter): Manages server-side CORBA objects.

## 3. What is IDL?

IDL (Interface Definition Language) defines the contract between client and server in a language-neutral way. It gets compiled to generate source code in the chosen programming language (Java in this case).

## 4. Code Explanation

### ReverseModule.idl

module ReverseModule {  
 interface Reverse {  
 string reverse\_string(in string str);  
 };  
};  
  
This defines a module "ReverseModule" and an interface "Reverse" with a method "reverse\_string" that takes a string input and returns a reversed string.

### ReverseImpl.java

import ReverseModule.ReversePOA;  
  
class Reverseimpl extends ReversePOA {  
 Reverseimpl() {  
 super();  
 System.out.println("Reverse Object Created");  
 }  
  
 public String reverse\_string(String name) {  
 StringBuffer str = new StringBuffer(name);  
 str.reverse();  
 return ("Server send: " + str);  
 }  
}  
  
Explanation:  
- This is the server-side class that implements the reverse\_string logic.  
- It extends ReversePOA, a skeleton class generated from IDL.  
- The reverse\_string() method reverses the input string using StringBuffer.

### ReverseServer.java

This file starts the server, activates the POA, registers the object with the naming service, and waits for client requests.  
  
Key lines:  
- Initialize ORB  
- Activate POA  
- Create object of Reverseimpl  
- Register it with Naming Service  
- Run ORB to wait for client

### ReverseClient.java

This file connects to the naming service, finds the Reverse object, sends a string, receives the reversed string, and prints it.  
  
Key lines:  
- Initialize ORB  
- Connect to Naming Service  
- Resolve object name "Reverse"  
- Read user input  
- Call reverse\_string() and display result

## 5. Libraries and Their Purpose

- org.omg.CORBA.\*: Core CORBA classes.  
- org.omg.CosNaming.\*: For naming service.  
- ReverseModule.\*: Generated from IDL.  
- java.io.\*: To read input from user.

## 6. Algorithm or Workflow

1. Server creates and registers a CORBA object.  
2. Client looks up the object using Naming Service.  
3. Client sends a string to the server.  
4. Server reverses the string and returns it.  
5. Client prints the result.

## 7. class Reverseimpl extends ReversePOA

- ReversePOA is an abstract base class generated from IDL.  
- It contains method definitions but no actual code.  
- Reverseimpl extends ReversePOA and provides the actual logic for reverse\_string().  
- When the server runs, it creates an object of Reverseimpl.  
- This object handles requests from clients and performs the logic.

## 8. Line-by-Line Code Explanation

### ReverseModule.idl – Line-by-Line

module ReverseModule {  
 // 'module' is like a package. It helps group related code.  
   
interface Reverse {  
 // 'interface' defines a set of methods that a CORBA object will support.  
  
string reverse\_string(in string str);  
 // This method takes one input (a string) and returns a string.  
 // 'in' means it is an input parameter.  
};  
};

### ReverseImpl.java – Line-by-Line

import ReverseModule.ReversePOA;  
// Import the generated abstract class from IDL to implement logic.  
  
class Reverseimpl extends ReversePOA {  
// Declare a class named Reverseimpl that extends ReversePOA (which is abstract).  
  
Reverseimpl() {  
 super();  
 System.out.println("Reverse Object Created");  
}  
// Constructor: prints a message when an object is created.  
  
public String reverse\_string(String name) {  
// Implementation of the abstract method from IDL.  
  
StringBuffer str = new StringBuffer(name);  
// Convert the string to StringBuffer to allow modification.  
  
str.reverse();  
// Reverse the characters in the string.  
  
return ("Server send: " + str);  
// Return the reversed string with a message.  
}

### ReverseServer.java – Line-by-Line

ORB orb = ORB.init(args, null);  
// Initialize the ORB (Object Request Broker).  
  
POA rootPOA = POAHelper.narrow(orb.resolve\_initial\_references("RootPOA"));  
// Get reference to Root POA (Portable Object Adapter).  
  
rootPOA.the\_POAManager().activate();  
// Activate the POA manager to manage CORBA object requests.  
  
Reverseimpl rvr = new Reverseimpl();  
// Create an instance of our server object.  
  
org.omg.CORBA.Object ref = rootPOA.servant\_to\_reference(rvr);  
// Convert the servant (rvr) to a CORBA object reference.  
  
Reverse h\_ref = ReverseHelper.narrow(ref);  
// Narrow it to the correct object type (Reverse).  
  
org.omg.CORBA.Object objRef = orb.resolve\_initial\_references("NameService");  
// Get reference to the naming service.  
  
NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);  
// Narrow it to NamingContextExt to allow name binding.  
  
String name = "Reverse";  
NameComponent path[] = ncRef.to\_name(name);  
// Create a name path for the object.  
  
ncRef.rebind(path, h\_ref);  
// Bind the object in the naming service so clients can find it.  
  
orb.run();  
// Keep server running and waiting for client requests.

### ReverseClient.java – Line-by-Line

ORB orb = ORB.init(args, null);  
// Initialize the ORB on the client side.  
  
org.omg.CORBA.Object objRef = orb.resolve\_initial\_references("NameService");  
// Get reference to the Naming Service.  
  
NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);  
// Narrow the reference to NamingContextExt to allow string lookups.  
  
String name = "Reverse";  
Reverse Reverseimpl = ReverseHelper.narrow(ncRef.resolve\_str(name));  
// Lookup the server object by name and narrow it to correct type.  
  
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  
// Set up reading input from the user.  
  
String str = br.readLine();  
// Read the input string from user.  
  
String tempStr = Reverseimpl.reverse\_string(str);  
// Call the server’s method to reverse the string.  
  
System.out.println(tempStr);  
// Print the result sent back by the server.