CORBA

What is it?

- An open standard to implement Distributed Objects based RMI (RMI doesn't mean Java RMI) across multiple languages.
- As CORBA is designed to support multiple languages including procedural, it is not necessary that a client is an object and servant is a class. There is no concept of class in CORBA as every language has its own notion of class
- Remote invocation semantics:
    - At-most-once by default
    - May-be for oneway methods

Major components:

CORBA IDL

- Interface Definition Language is a common language to expose remote methods that can be compiled in client/server specific language platform
- IDL interface specifies a name and set of methods that client can request
- It provides facilities to define
    - Module
    - Interface
    - Types
    - Attributes
    - Method signatures

ORB

- Operations to start and stop CORBA service
- Helps client to invoke a method on an Object. That means
    - Locating Object
    - Activating Object if required
    - Communication between client and object
- Receives request from client stub, delivers it to Servant with the help of Object Adapter on server side and returns back result to the client
- Provides other services as well, Naming service for example
- Uses resolve_initial_references method (for Java) to get the CORBA platform object of any service offered by ORB. RootPOA, Naming Service to name a few.

Object Adapter

- Bridges the gap between CORBA object implemented on IDL specific interface and Programming language specific interface implemented by Servant
- Mainly does:
    - Creates remote object reference for CORBA objects
    - Dispatches RMI request via skeleton to an appropriate Servant
    - Activates and Deactivates servant
- While registering a servant to POA, it creates a unique object name and the same will be used to map a CORBA request object to its servant. It maintains a remote object table that maps the name of CORBA objects to their servants
- Portable Object Adapter (POA)
    - All CORBA Object Adapters are POA as it allows Application and Servant to run on different ORB platforms
    - It is achieved by standardizing skeleton classes and interaction between POA and servant
    - Assists ORB to deliver the client request to specific servant

Skeleton

- Generated by IDL in server language to marshal and unmarshal argument/result

Client Stub/Proxies

- Generated by IDL in client language to provide proxy for remote method to the client program as well as marshal and unmarshal argument/result

Implementation Repository

- Activates/locates registered servers by mean of Object Adapter
- Format of each record as:
  {Object Adapter Name} | {Path to Object Implementation file} | {Host and Port of Server}
  Adapter name and path to implementation file are global to repository, hence made available when Server program was installed. Later when the Object implementation is activated on server, host and port would be added to implementation repository
- It allows extra information to be stored about each server as in Security, Authentication and access control. These control who can access and activate a servant.
- We may replicate the information of Implementation repository to achieve fault tolerance

Interface Repository

- Helps to achieve dynamic invocation.
- Resembles Java Reflection in CORBA
- It provides information about IDL interface to client and server who inquires it
- Information includes method name, parameters it accepts and return value details for any remote object reference

- IDL generates type ID for each type in IDL file. Hence each interface would be assigned an IDL type identifier. Example: `"IDL:HelloApp/Hello:1.0"`
- Interface repository uses that as ID to achieve a mapping of type identifier to interface
- As Static invocations use proxies they don't make use of Interface repository
- Not all ORB provide interface repository

Dynamic Invocation Interface (DII)

- Helps client to make dynamic invocations on remote CORBA objects without proxy
- For example, to implement object browser functionality, it is not known well in advance about all remote objects hence can't install proxy classes for all.
- Interface repository helps to get the details of methods available for each CORBA object
- Further information: https://docs.oracle.com/cd/E13203_01/tuxedo/tux80/creclien/dii.htm

Dynamic Skeleton Interface (DSI)

- Helps server to invoke methods on interface for which it has no Skeleton implementation while compiling server program
- In that case a Dynamic skeleton receives the request and identifies the target servant by looking at the content of request (arguments and return type for example)

CORBA Remote Object reference

- CORBA provides a reference to each remote object in following format, which is known as Interoperable Object Reference (**IOR**):
  {Interface repository type id} | {IIOP} | {Host} | {Port} | {Adapter Name} | {Object Name}
  Interface repository type id => IDL type id generated by IDL compiler
  IIOP (Internet inter-ORB protocol) => Protocol Version , its same as GIOP over TCP/IP

  IIOP + Host + Port = Server address
  Adapter Name + Object Name = Object Key

- Two types of IOR

Transient IOR	Persistent IOR
Lasts only as long as the process that hosts it	Lasts between activations of CORBA objects
Contains address details of server hosting the CORBA object	Contains address details of Implementation Repository it is registered with
Request received directly by Server and follows path Object Adapter Name -> Object Name to locate servant	1. Implementation repository receives the request 2. Fetch the Adapter Name and lookup its own table to find the host for it 3. Activates the CORBA object at host if needed 4. Returns that address details back to Client 5. Finally client sends the request to that address with Object key 6. Recipient host does use Object Key to locate servant

Invocation Types

- Static: When programmers have knowledge of which method to invoke while compiling the client code. More in use as it resembles more natural programming model
- Dynamic: When programmers don't have any prior knowledge of which method to invoke while compiling the client code

How to compile the IDL file

Generate only Client code:

Idlj –fclient <idl file>

Generate only Server code:

Idlj –fserver <idl file>

Generate both Client and Server code:

Idlj –fall <idl file>

Client *idlj -fclient HelloApp.idl*	**Server** *idlj -fserver HelloApp.idl*
Hello.java	Hello.java
HelloOperations.java	HelloOperations.java
HelloHelper.java	HelloPOA.java
HelloHolder.java	
_HelloStub.java	

Programming classes

Operations Interface:

- Contains remote method defined in IDL Interface

Hello Interface:

- Extends: Operations, org.omg.CORBA.Object and org.omg.CORBA.protbale.IDLEntity interfaces
- As we would be reading and writing Hello objects over the input output streams, it is necessary to implement last two interfaces mentioned above.
- org.omg.CORBA.Object represents CORBA Data Representation (CDR), an external data representation technique for CORBA
- It doesn't contain and body

Stub Class:

- Extends: org.omg.CORBA.portable.ObjectImpl class
- Implements: Hello interface

Holder Class:

- Used for OUT and INOUT parameters
- Keeps an instance of base interface
- Provides read and write methods to stream base interface instance.
- Implements org.omg.CORBA.portable.Streamable interface
- Example
  IDL method: *void getPerson(in string name, out Person p)*
  Same Java compilation: *void getPerson(String name, PersonHolder p)*

Helper Class:

- Assists to map CORBA object into language specific object
- narrow method will return language specific object reference for a given CORBA reference
- Helps Holder class to convert to and from CORBA object to platform specific object while writing and reading it

Skeleton/POA Class:

- Works a skeleton class for server.
- Extends: org.omg.PortableServer.Servant class
- Implements: Operations interface generated by IDL compiler and org.omg.CORBA.portable.InvokeHandler interfaces
- InvokeHandler interface works as dispatcher module and call _invoke method while dispatching each request
- _invoke method accepts method name, marshalled input stream of input parameters and OutputStream as parameters.
- It invokes the method specified and returns the marshaled result into OutputStream object originally received as parameter. The same OutputStream will be accessed by client stub to return the server response to client