Date: 10th Dec 2021

Name: Rohan Khurana
Roll No: 1802910129

Distributed System Lab

LAB – 10

Aim: Write a program to implement CORBA mechanism by using C++ program at one end and Java program on the other.

Program:

Server.cpp

```
#include <iostream>
#include "OB/CORBA.h"
#include <OB/Cosnaming.h>
#include "crypt.h"
#include "cryptimpl.h"
using namespace std;
int main(int argc, char** argv)
{
 // Declare ORB and servant object
 CORBA::ORB_var orb;
 CryptographicImpl* CrypImpl = NULL;
 try {
 // Initialize the ORB.
 orb = CORBA::ORB_init(argc, argv);
 // Get a reference to the root POA
 CORBA::Object_var rootPOAObj =
 orb->resolve_initial_references("RootPOA");
 // Narrow it to the correct type
 PortableServer::POA_var rootPOA =
 PortableServer::POA::_narrow(rootPOAObj.in());
 // Create POA policies
 CORBA::PolicyList policies;
 policies.length(1);
 policies[0] =
 rootPOA->create_thread_policy
 (PortableServer::SINGLE_THREAD_MODEL);
 // Get the POA manager object
 PortableServer::POAManager_var manager = rootPOA->the_POAManager();
 // Create a new POA with specified policies
```

```
 PortableServer::POA_var myPOA = rootPOA->create_POA
 ("myPOA", manager, policies);
 // Free policies
CORBA::ULong len = policies.length();
 for (CORBA::ULong i = 0; i < len; i++)
 policies[i]->destroy();
 // Get a reference to the Naming Service root_context
 CORBA::Object_var rootContextObj =
 orb->resolve_initial_references("NameService");
 // Narrow to the correct type
 CosNaming::NamingContext_var nc =
 CosNaming::NamingContext::_narrow(rootContextObj.in());
 // Create a reference to the servant
 CrypImpl = new CryptographicImpl(orb);
 // Activate object
 PortableServer::ObjectId_var myObjID =
 myPOA->activate_object(CrypImpl);
 // Get a CORBA reference with the POA through the servant
 CORBA::Object_var o = myPOA->servant_to_reference(CrypImpl);
 // The reference is converted to a character string
 CORBA::String_var s = orb->object_to_string(o);
 cout << "The IOR of the object is: " << s.in() << endl;
 CosNaming::Name name;
 name.length(1);
 name[0].id = (const char *) "CryptographicService";
 name[0].kind = (const char *) "";
 // Bind the object into the name service
 nc->rebind(name,o);
 // Activate the POA
 manager->activate();
 cout << "The server is ready.
 Awaiting for incoming requests..." << endl;
 // Start the ORB
 orb->run();
 } catch(const CORBA::Exception& e) {
 // Handles CORBA exceptions
 cerr << e << endl;
 }
 // Decrement reference count
 if (CrypImpl)
```

```
     CrypImpl->_remove_ref();
 // End CORBA
 if (!CORBA::is_nil(orb)){
 try{
 orb->destroy();
 cout << "Ending CORBA..." << endl;
 } catch (const CORBA::Exception& e)
 {
 cout << "orb->destroy() failed:" << e << endl;
 return 1;
 }
 }
 return 0;
}
```

Client.cpp

```
#include <iostream>
#include <string>
#include "OB/CORBA.h"
#include "OB/Cosnaming.h"
#include "crypt.h"
using namespace std;
int main(int argc, char** argv)
{
 // Declare ORB
 CORBA::ORB_var orb;
 try {
 // Initialize the ORB
 orb = CORBA::ORB_init(argc, argv);
 // Get a reference to the Naming Service
 CORBA::Object_var rootContextObj =
 orb->resolve_initial_references("NameService");
 CosNaming::NamingContext_var nc =
 CosNaming::NamingContext::_narrow(rootContextObj.in());
 CosNaming::Name name;
name.length(1);
 name[0].id = (const char *) "CryptographicService";
 name[0].kind = (const char *) "";
 // Invoke the root context to retrieve the object reference
 CORBA::Object_var managerObj = nc->resolve(name);
```

```
 // Narrow the previous object to obtain the correct type
 ::CaesarAlgorithm_var manager =
 ::CaesarAlgorithm::_narrow(managerObj.in());
 string info_in,exit,dummy;
 CORBA::String_var info_out;
 ::CaesarAlgorithm::charsequence_var inseq;
 unsigned long key,shift;
 try{
 do{
 cout << "\nCryptographic service client" << endl;
 cout << "----------------------------" << endl;
 do{ // Get the cryptographic key
 if (cin.fail())
 {
 cin.clear();
 cin >> dummy;
 }
 cout << "Enter encryption key: ";
 cin >> key;
 } while (cin.fail());
 do{ // Get the shift
 if (cin.fail())
 {
 cin.clear();
 cin >> dummy;
 }
 cout << "Enter a shift: ";
 cin >> shift;
 } while (cin.fail());
 // Used for debug pourposes
 //key = 9876453;
//shift = 938372;
 getline(cin,dummy); // Get the text to encrypt
 cout << "Enter a plain text to encrypt: ";
 getline(cin,info_in);
 // Invoke first remote method
 inseq = manager->encrypt
 (info_in.c_str(),key,shift);
 cout << "----------------------------------------"
 << endl;
```

```
 cout << "Encrypted text is: "
 << inseq->get_buffer() << endl;
 // Invoke second remote method
 info_out = manager->decrypt(inseq.in(),key,shift);
 cout << "Decrypted text is: "
 << info_out.in() << endl;
 cout << "----------------------------------------"
 << endl;
 cout << "Exit? (y/n): ";
 cin >> exit;
 } while (exit!="y");
 // Shutdown server message
 manager->shutdown();
 } catch(const std::exception& std_e){
 cerr << std_e.what() << endl;
 }
 }catch(const CORBA::Exception& e) {
 // Handles CORBA exceptions
 cerr << e << endl;
 }
 // End CORBA
 if (!CORBA::is_nil(orb)){
 try{
 orb->destroy();
 cout << "Ending CORBA..." << endl;
 } catch(const CORBA::Exception& e)
 {
 cout << "orb->destroy failed:" << e << endl;
 return 1;
 }
 }
return 0;
}
```