Motilal Nehru National Institute of TechnologyAllahabad Prayagraj-211004 [India]

Department of Computer Science & Engineering

Programme Name: B.Tech Course Code: CS17201

Semester: VII Branch: Computer Science & Engg. Course Name: Distributed Systems (Lab)

Lab Assignment 8

Name-Sugandh mishra
reg-20204211
sec-cse c

Lab #	Name of Experiment
8	Implement CORBA mechanism by using 'C++' program at one end and 'Java' program on the other.

server.cpp—-----

```
package server;

import org.omg.CORBA.ORB;
import org.omg.CosNaming.NameComponent;
import org.omg.CosNaming.NamingContextExt;
import org.omg.CosNaming.NamingContextExtHelper;
import org.omg.PortableServer.POA;
import org.omg.PortableServer.POAHelper;

import Quiz.QuizServerHelper;

public class QuizServer {

   public static final String SERVER_NAME = "Quiz";

   /**
    * @param args
    */
   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
            QuizServant helloImpl = new QuizServant();

            // get object reference from the servant
            org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
            Quiz.QuizServer href = QuizServerHelper.narrow(ref);

            // get the root naming context
            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);

            // bind the Object Reference in Naming
            NameComponent path[] = ncRef.to_name(SERVER_NAME);
            ncRef.rebind(path, href);

            System.out.println("QuizServer 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 ...");  }

}

package server;


import Quiz.CompleteQuestion;
import Quiz.QuestionHolder;
import Quiz.QuestionImpl;
import Quiz.QuizServerPOA;
import Quiz.QuizServerPackage.QuizException;
import Quiz.QuizServerPackage.answersIdsHolder;

import java.util.*;

public class QuizServant extends QuizServerPOA {

   /** Counter for the question IDs */
   private int numQuestions;
```

```
    /** Map to save questions */
    private Map<Integer, CompleteQuestion> compleQuestions;

    public QuizServant() {
        this.numQuestions = 0;
        this.compleQuestions = new HashMap<Integer, CompleteQuestion>();
    }

    @Override
    public int insertQuestion(CompleteQuestion question) throws QuizException {
        System.out.println("> receive new question " + question);
        question.id = this.numQuestions++;
        // add to map
        this.compleQuestions.put(question.id, question);
        System.out.println("send question id: " + question.id);
        return question.id;
    }

    @Override
    public boolean getQuestion(QuestionHolder randomQuestion)
            throws QuizException {
        System.out.println("> receive random question request");
        if (!this.compleQuestions.isEmpty()) {
            // Get random complete question
            CompleteQuestion question = this.compleQuestions.get(getRandomMapKey());
            randomQuestion.value = new QuestionImpl(question.id, question.sentence,
question.answers);
            System.out.println("send question: " + randomQuestion.value);
            return true;
        } else {
            return false;
        }
    }

    @Override
    public boolean answerQuestion(int questionId, char[] answer,
            answersIdsHolder correct) throws QuizException {

        if (this.compleQuestions.containsKey(questionId)) {
            char[] correctAnswers = compleQuestions.get(questionId).correctAnswers;
            correct.value = correctAnswers;

            // Answer is wrong in this case
            if (answer.length != correctAnswers.length) {
                // System.out.print("Different answer lenght");
                return false;
            }
```

```
            boolean flag = true;
            for (int i = 0; i < answer.length; i++) {
                flag = flag & (answer[i] == correctAnswers[i]);
            }
            return flag;

        } else {
            System.err.println("Question ID exits not.");
            return false;
        }

    }

    @Override
    public int deleteQuestion(int questionId) throws QuizException {
        if (this.compleQuestions.containsKey(questionId)) {
            compleQuestions.remove(questionId);
            return questionId;
        } else {
            System.err.println("Quesiton ID exits not.");
            return -1;
        }
    }

    /**
     * Get a random key from the map this.compleQuestions.
     *
     * @return random map key
     */
    private Integer getRandomMapKey() {
        List<Integer> mapKeys = new ArrayList<Integer>(this.compleQuestions.keySet());
        int randomIndex = (int) (Math.random() * mapKeys.size());
        return mapKeys.get(randomIndex);
    }

}
```

client.cpp

```
#include "Quiz.hh"
#include <iostream>

/** Name is defined in the server */
#define SERVER_NAME     "Quiz"

Quiz::QuizServer_ptr service_server;

using namespace std;

void insert_question(const char* sentence, int numAnswers, Quiz::Answer** answers, int
```

```
numCorrectAnswers, CORBA::Char* correctAnswers);
void create_questions();

int main(int argc, char ** argv)
{
   try {
       //-------------------------------------------------------------------------
       // Initialize ORB object.
       //-------------------------------------------------------------------------
       CORBA::ORB_ptr orb = CORBA::ORB_init(argc, argv);

       //-------------------------------------------------------------------------
       // Resolve service
       //-------------------------------------------------------------------------
       service_server = 0;

       try {

           //---------------------------------------------------------------------
           // Bind ORB object to name service object.
           // (Reference to Name service root context.)
           //---------------------------------------------------------------------
           CORBA::Object_var ns_obj = orb->resolve_initial_references("NameService");

           if (!CORBA::is_nil(ns_obj)) {
               //-----------------------------------------------------------------
               // Bind ORB object to name service object.
               // (Reference to Name service root context.)
               //-----------------------------------------------------------------
               CosNaming::NamingContext_ptr nc =
CosNaming::NamingContext::_narrow(ns_obj);

               //-----------------------------------------------------------------
               // The "name text" put forth by CORBA server in name service.
               // This same name ("MyServerName") is used by the CORBA server when
               // binding to the name server (CosNaming::Name).
               //-----------------------------------------------------------------
               CosNaming::Name name;
               name.length(1);
               name[0].id = CORBA::string_dup(SERVER_NAME);
               name[0].kind = CORBA::string_dup("");

               //-----------------------------------------------------------------
               // Resolve "name text" identifier to an object reference.
               //-----------------------------------------------------------------
               CORBA::Object_ptr obj = nc->resolve(name);

               if (!CORBA::is_nil(obj)) {
                   service_server = Quiz::QuizServer::_narrow(obj);
```

```
                }
            }
        } catch (CosNaming::NamingContext::NotFound &) {
            cerr << "Caught corba not found" << endl;
        } catch (CosNaming::NamingContext::InvalidName &) {
            cerr << "Caught corba invalid name" << endl;
        } catch (CosNaming::NamingContext::CannotProceed &) {
            cerr << "Caught corba cannot proceed" << endl;
        }

        //--------------------------------------------------------------------
        // Do stuff
        //--------------------------------------------------------------------
        if (!CORBA::is_nil(service_server)) {
                        cout << "QuizClient client is running ..." << endl;

                    orb->register_value_factory("IDL:Quiz/Answer:1.0", new
Quiz::Answer_init());

                        create_questions();

                        //
                        // get random question
                        //
                        orb->register_value_factory("IDL:Quiz/Question:1.0", new
Quiz::Question_init());

                        Quiz::Question* received_question = new OBV_Quiz::Question();
                        service_server->getQuestion(received_question);
                        const char* received_question_sentence =
received_question->sentence();
                        CORBA::Long received_question_id = received_question->id();
                        Quiz::Question::AnswerSeq received_question_answers =
received_question->answers();
                        int numAnswers = received_question_answers.length();

                        cout << "Received Question: id=" << received_question_id << ",
sentence=" << received_question_sentence << endl;

                        for(int i = 0; i < numAnswers; i++) {
                            if(received_question_answers[i]) {
                                cout << "\t" << received_question_answers[i]->id() << ": "
<< received_question_answers[i]->sentence() << endl;
                            }
                        }
        }

        //--------------------------------------------------------------------
        // Destroy OBR
```

```
        //-------------------------------------------------------------------------
        orb->destroy();

    } catch (CORBA::UNKNOWN) {
        cerr << "Caught CORBA exception: unknown exception" << endl;
    }
}


void insert_question(const char* sentence, int numAnswers, Quiz::Answer** answers, int
numCorrectAnswers, CORBA::Char* correctAnswers)
{
        Quiz::Question::AnswerSeq* answersSeq = new
OBV_Quiz::Question::AnswerSeq(numAnswers, numAnswers, answers, 1);
        Quiz::CompleteQuestion::CharSeq* correctAnswersSeq = new
OBV_Quiz::CompleteQuestion::CharSeq(numCorrectAnswers, numCorrectAnswers, correctAnswers,
1);
        Quiz::CompleteQuestion* new_question = new OBV_Quiz::CompleteQuestion(0, sentence,
*answersSeq, *correctAnswersSeq);

    CORBA::Long question_received_id = service_server->insertQuestion(new_question);
        cout << "send question and received id " << question_received_id << endl;
}

void create_questions()
{
    // create first question
        const char* question_sentence = "It applies to a software layer that provides a
programming abstraction as well as masking the heterogeneity of the underlying networks,
hardware, operating systems and programming languages. What is it?";
        Quiz::Answer** question0_answers = new Quiz::Answer*[3];
        question0_answers[0] = new OBV_Quiz::Answer('a', "Hetereogenity");
        question0_answers[1] = new OBV_Quiz::Answer('b', "Middleware");
        question0_answers[2] = new OBV_Quiz::Answer('c', "Opennes");
        CORBA::Char question0_correctAnswers[] = {'b'};
        insert_question(question_sentence, 3, question0_answers, 1,
question0_correctAnswers);

        // create second question
        question_sentence = "It refers to a running program (a process) on a networked
computer that accepts requests from programs running on other computers to perform a
service and responds appropriately.";
        Quiz::Answer** question1_answers = new Quiz::Answer*[3];
        question1_answers[0] = new OBV_Quiz::Answer('a', "Server");
        question1_answers[1] = new OBV_Quiz::Answer('b', "Middleware");
        question1_answers[2] = new OBV_Quiz::Answer('c', "Client");
        CORBA::Char question1_correctAnswers[] = {'a'};
        insert_question(question_sentence, 3, question1_answers, 1,
question1_correctAnswers);
```

```
}
```

client.cpp

```
package client;

import java.io.*;
import org.omg.CORBA.ORB;
import org.omg.CosNaming.NamingContextExt;
import org.omg.CosNaming.NamingContextExtHelper;

import Quiz.AnswerImpl;
import Quiz.Answer;
import Quiz.CompleteQuestion;
import Quiz.CompleteQuestionImpl;
import Quiz.Question;
import Quiz.QuestionHolder;
import Quiz.QuizServerHelper;
import Quiz.QuizServerOperations;
import Quiz.QuizServerPackage.QuizException;
import Quiz.QuizServerPackage.answersIdsHolder;

public class QuizClientInteractive {

   private static QuizServerOperations serverImpl;
   private static BufferedReader reader;

   public static final String SERVER_NAME = "Quiz";

   /**
    * @param args
    */
   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);

           // resolve the Object Reference in Naming
           serverImpl = QuizServerHelper.narrow(ncRef.resolve_str(SERVER_NAME));

           /*
            * Start
            */
           System.out.println("Welcome to the Quiz Client.");
```

```
            reader = new BufferedReader(new InputStreamReader(System.in));
            showMenuDialog();

        } catch (Exception e) {
            System.out.println("ERROR : " + e) ;
            e.printStackTrace(System.out);
        }
    }
    private static void showMenuDialog(){
        try {

            System.out.println("\n------ Please choose an option:");
            System.out.println("a: Insert a new question");
            System.out.println("b: Request a random question");
            System.out.println("c: Remove a question");

            String option = reader.readLine();

            switch(option) {
                case "a": insertQuestion(); break;
                case "b": newRandomQuestion(); break;
                case "c": removeQuestion(); break;
                default: returnToMenu(); break;
            }

        } catch(IOException e) {
            System.out.println("IOException " + e);
        } catch(QuizException e) {
            System.out.println("QuizException " + e);
        } catch(Exception e) {
            System.out.println("Exception " + e);
        }

    }

    /**
     * Remove a question.
     *
     * @throws IOException
     * @throws QuizException
     */
    private static void removeQuestion() throws IOException, QuizException {
        System.out.println("Write the ID of the question that you would like to remove");
        int questionId = readInteger();
        serverImpl.deleteQuestion(questionId);

        returnToMenu();
    }
```

```
    /**
     * Get a random question and answer it.
     *
     * @throws IOException
     * @throws QuizException
     */
    private static void newRandomQuestion() throws IOException, QuizException {
            System.out.println("------ Request Question");

            /*
             * Get random question from server
             */
            QuestionHolder myQuestionHolder = new QuestionHolder();
            serverImpl.getQuestion(myQuestionHolder);
            Question receivedQuestion = myQuestionHolder.value;

            /*
             * Print question and alternatives
             */
            System.out.println("The question is: " + receivedQuestion.sentence);
            Answer[] answers = receivedQuestion.answers;
            for(int i=0; i<answers.length; i++) {
                System.out.println(answers[i].id + ": " + answers[i].sentence);
            }

            /*
             * Answer question
             */
            System.out.println("Which is the correct answer? Write the letters followed by
a comma (,) in order if there are more than one correct answer");
            String correctAnswers = reader.readLine();
            String[] correct = correctAnswers.split(",");
            char[] correctA = new char[correct.length];

            for (int i=0; i < correct.length; i++) {
                correctA[i] = correct[i].charAt(0);
            }
            answersIdsHolder correctAnswersHolder = new answersIdsHolder();
            boolean answerIsCorrect = serverImpl.answerQuestion(receivedQuestion.id,
correctA, correctAnswersHolder);
            if(answerIsCorrect) {
                System.out.println("Correct answer!");
            } else {
                System.out.println("You're wrong, I'm sorry! These are the correct
answers:");
                System.out.println(correctAnswersHolder.value);
            }

            returnToMenu();
```

```
    }


    private static void insertQuestion() throws IOException, QuizException {
            System.out.println("------ Insert your new question:");
            String question = reader.readLine();

            int howManyAlt = 0;
            String alternative = null;
            int howManyCorr = 0;
            //int count = 0;
            char[] myCorrectAlternatives = new char[0];
            int currcount = 0;
            Answer[] alternatives = new Answer[0];



            System.out.println("How many alternative would you like to insert? ");
            howManyAlt = readInteger();

            alternatives= new Answer[howManyAlt];

            System.out.println("How many alternative are correct? ");
            howManyCorr = readInteger();
            myCorrectAlternatives = new char[howManyCorr];

            for(int count = 0; count < howManyAlt; count++) {
                System.out.println("Write your alternative! ");
                alternative = reader.readLine();
                alternatives[count] = new AnswerImpl(((char)(count+65)), alternative);

                if (currcount < howManyCorr) {
                    System.out.println("Is it correct? Y or N ");
                    if(readYes()){
                        myCorrectAlternatives[currcount] = ((char)(count+65));
                        currcount ++;
                    }
                }
            }

            /*
             * Create question and send it to the server
             */
            CompleteQuestion myQuestion = new CompleteQuestionImpl(question, alternatives,
myCorrectAlternatives);
            int receivedQuestionId = serverImpl.insertQuestion(myQuestion);
            System.out.println("> reseived question id: " + receivedQuestionId);

            returnToMenu();
    }
```

```java
    /**
     * Shows dialog to return to the menu or quit.
     *
     * @throws IOException
     */
    private static void returnToMenu() throws IOException {
        System.out.println("Would you like to do somenthing else? Y or N");
        if(readYes()){
            showMenuDialog();
        } else {
            System.out.println("Goodbye. Quit.");
        }
    }

    /**
     * Reads a line and pares it to an integer.
     *
     * @return Read integer
     * @throws IOException
     */
    private static int readInteger() throws IOException {
        try {
            return Integer.parseInt(reader.readLine());
        } catch (NumberFormatException e) {
            System.err.println("NumberFormatException: Could not parse integer");
            return -1;
        }
    }

    /**
     * Returns true, when a "Y" or "y" is read.
     *
     * @return True, when read "Y" or "y"
     * @throws IOException
     */
    private static boolean readYes() throws IOException {
        String line = reader.readLine().toLowerCase();
        return line.equals("y");
    }

}
```