LABWORK
COURSE: DISTRIBUTED SYSTEMS
CHAPTER 1: INTRODUCTION AND ARCHITECTURE IN DISTRIBUTED SYSTEMS

1. Web server apache2

1.1. Content

In the class, we have studied the chapter1: Introduction of Distributed Systems. We have learned the definition and some features of DS. We knew that all the network services based on the theory of DS. In this labwork, let's try to construct a WWW system. Concretely, we will install a web server.

1.2. Requirements

1.2.1. Theory

- Handling Unix OS
- Base of Computer Network

1.2.2. Devices

- PC or a VM

1.2.3. Software

1.3. Practical Steps

1.3.1. Install web server apache2

First, we need to install a webserver software. Today, the most commonly used webserver is *apache*. Run the following command:

```
sudo apt install apache2
```

Try to access this webserver in typing the IP of this PC in another PC (they must be in the same LAN network). If it appears the default page of apache (something like "Apache2 Ubuntu default page"), that means you installed successfully the apache webserver.

Question 1: What is the path of the html file that contains the content of the default website apache?

Question 2: What is the default port on which webserver is listening?

1.3.2. Install virtual hosts for apache2

Web server apache2 is able to host several virtual machines with only one IP address. Now we try to make running 2 domains: example.com and test.com
First, create two folders containing content for these two domaims:

```
sudomkdir -p /var/www/example.com/public_html
sudomkdir -p /var/www/test.com/public_html
```

Change permission:

```
sudochmod -R 755 /var/www
```

Question 3: Explain what permission 755 means.

Write the content for these 2 website (edit the index.html file in 2 folders public_html you have just created)

```
<html>
<head>
<title>Welcome to Example.com!</title>
</head>
<body>
<h1>Success!  The  example.com  virtual  host  is
working!</h1>
</body>
</html>
```

(change to `test.com` with the correspondent file).

The default configuration file of virtual host of apache is:

```
/etc/apache2/sites-available/000-default.conf
```

Now, create the two following new files:
`/etc/apache2/sites-available/example.com.conf`
`/etc/apache2/sites-available/test.com.conf`

Here is the content of file example.com.conf

```
<VirtualHost *:80>
ServerAdmin admin@example.com
ServerName example.com
ServerAlias www.example.com
DocumentRoot /var/www/example.com/public_html
ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Now, for file test.com.conf :

```
<VirtualHost *:80>
ServerAdmin admin@test.com
ServerName test.com
ServerAlias www.test.com
```

```
DocumentRoot /var/www/test.com/public_html
ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Run these two following commands to activate these files above:

```
sudo a2ensite example.com.conf
sudo a2ensite test.com.conf
```

Restart the apache service:

```
sudo service apache2 restart
```

Open the file `/etc/hosts` and add these lines:

```
127.0.0.1 example.com
127.0.0.1 test.com
```

Now, open the web browser and test the 2 addresses: *example.com* and *test.com*

Question 4: What do you see after typing these 2 addresses? Explain it.

Question 5: Try to make other machines in the same LAN access to these 2 addresses.

2. Interface in Java

2.1. Content

In the class, we have studied the characteristic Openness of Distributes Systems. In order to guarantee the feature Openness, we have to construct *interface* between components of the system. In this section, we'll construct a simple client-server model, where the client sends a series of numbers to the server. The latter will sort these received numbers in using the method *sort* declared in an interface. There will be different way of implementing the method *sort* of this interface.

2.2. Requirements

2.2.1. Theory

- Java programming

2.2.2. Devices

- PC

2.2.3. Software

- Eclipse IDE
- Installed JDK/JRE

2.3. Practical Steps

2.3.1. Install requirements

- Download and install IDE Eclipse:
https://www.eclipse.org/downloads/packages/
- Download JDK/JRE:
https://www.oracle.com/technetwork/java/javase/downloads/index.html

2.3.2. Construct the program

First, open Eclipse and create a new java project in choosing *File* → *New* → *Java project*. You name it whatever you want.
After creating the project, you'll see a folder named *src*. This is the folder that contains the source code.
Create two packages in that folder in right-click in *src* and choose *New* → *Package*.
Name these two packages as follows:
*com.hust.soict.**your_name**.client_server*
*com.hust.soict.**your_name**.helper*
(Attention: replace ***your_name*** by your name.)

In the *client_server* package, create a two classes and name it: Client and Server.
Now, it's time to write code.

2.3.2.1. Client

Now you open and edit the Client.java file.
First, you have to import some classes of Java libraries:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;
```

Then, in the *main* method, you initialize a socket instant in using *Socket* class:

```
Socket socket = new Socket("127.0.0.1", 9898);
```

You can replace the IP address and the port number as you like, but make attention that this is the IP of the server and the port the server program is listening on.

Now, we have to initialize two instances of two classes *BufferedReader* and *PrintWriter* for sending and receiving data.

```
        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
```

```
        PrintWriter out = new
PrintWriter(socket.getOutputStream(), true);
```

Initialize an instance of Scanner class

```
System.out.println(in.readLine());
Scanner scanner = new Scanner(System.in);
```

Now, you have to write yourself a *while* loop to get numbers from user and send it to server, until user types empty.
Suggestion: you can use this line to get message string:

```
String message = scanner.nextLine();
```

Question 6: What is the code of the while loop?

In the end, don't forget to close the *socket* and the *scanner*:

```
socket.close();
scanner.close();
```

2.3.2.2. Server

Now you open and edit the Server.java file. The goal is to construct a multi-threaded server that receives numbers from client and sort them and send back the result to the client.
First, you have to import some classes of Java libraries:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import com.hust.soict.haianh.helper.*;
import java.util.Arrays;
```

In the *main* method, write the code as follows:

```
System.out.println("The Sorter Server is running!");
int clientNumber = 0;
try (ServerSocket listener = new ServerSocket(9898)) {
     while (true) {
        new Sorter(listener.accept(), clientNumber++).start();
     }
}
```

Make attention that the port number must be the same as the one in the client code.
Outside of the main method, create a class Sorter for a thread. In fact, this class extends the class Thread:

```
    private static class Sorter extends Thread {
        private Socket socket;
        private int clientNumber;
```

```
        public Sorter(Socket socket, int clientNumber) {
            this.socket = socket;
            this.clientNumber = clientNumber;
            System.out.println("New client #" + clientNumber + " connected
at " + socket);
        }

        public void run() {
            try {
                BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
                PrintWriter out = new PrintWriter(socket.getOutputStream(),
true);

                // Send a welcome message to the client.
                out.println("Hello, you are client #" + clientNumber);

                // Get messages from the client, line by line; Each line has
several numbers separated by a space character
                while (true) {
                    String input = in.readLine();
                    if (input == null || input.isEmpty()) {
                        break;
                    }
                    //Put it in a string array
                    String[] nums = input.split(" ");

                    //Convert this string array to an int array
                    int[] intarr = new int[ nums.length ];

                    int i = 0;

                    for ( String textValue : nums ) {
                           intarr[i] = Integer.parseInt( textValue );
                      i++;
                    }

                    //Sort the numbers in this int array
                    new SelectionSort().sort(intarr);
                    //Convert the int array to String
                    String strArray[] = Arrays.stream(intarr)
                                              .mapToObj(String::valueOf)
                                              .toArray(String[]::new);
                    //Send the result to Client
                    out.println(Arrays.toString(strArray));
                }
            } catch (IOException e) {
                System.out.println("Error handling client #" +
clientNumber);
            } finally {
                try { socket.close(); } catch (IOException e) {}
                System.out.println("Connection with client # " +
clientNumber + " closed");
            }
        }
    }
```

Question 7: What is the role of the method *run*? When is it called?

In the code above, you can see the call of the method *sort* of the class *SelectionSort*. Now, we can construct an interface and declare the method *sort* inside. The class *SelectionSort* is one of the classes that implement this interface.

2.3.2.3. Interface and different implementation

Now, right-click on the package *com.hust.soict.your_name.helper*, choose *New → Interface*
Create a new interface and name it *NumberSorter*.
In this file, write the code below:

```
public interface NumberSorter {
	void sort(int arr[]);
}
```

In the same package, create a new class and name it *SelectionSort*.
In fact, the class *SelectionSort* will implement the interface *NumberSorter* and define concretely the method *sort* based on the algorithm *selection sort*.
Open the file SelectionSort.java and write the code below:

```
public class SelectionSort implements NumberSorter{
	public void sort(int arr[]) {
		int n = arr.length;

	  // One by one move boundary of unsorted subarray
	  for (int i = 0; i < n-1; i++)
	  {
		// Find the minimum element in unsorted array
		int min_idx = i;
		for (int j = i+1; j < n; j++)
		    if (arr[j] < arr[min_idx])
			min_idx = j;

		// Swap the found minimum element with the first
		// element
		int temp = arr[min_idx];
		arr[min_idx] = arr[i];
		arr[i] = temp;
	  }
	}
}
```

2.3.2.4. Run the program

Now, try to run the whole program. Don't forget to run the server before the client.

2.3.2.5. Implement other sort algorithms

It's time to implement yourself other sort algorithms. You do the same thing as above (create new class in the package helper and implement the *NumberSorter* interface). Try to implement the 3 algorithms below:

- Bubble sort
- Insertion sort
- Shell sort

3. Microservices

3.1. Contents

We now try to practice constructing a basic microservices architecture in using Kubernetes, a portable, extensible open-source platform for managing containerized workloads and services. We use also Docker to create containers.

3.2. Requirements

3.2.1. Theory

- Microservices fundamental
- Kube fundamentals: Pods, Services, Deployments et al.
- Docker

3.2.2. Hardwares

- Laptop/PC on Windows

3.2.3. Softwares

- VirtualBox
- Docker
- The kubernetes command line tool *kubectl*
- The minikube binary
- Git bash

3.3. PRACTICAL STEPS

Installation

- Install VirtualBox: https://www.virtualbox.org/wiki/Downloads
- In order to install tool kubectl on Windows, we have to install *Chocolatey* before: https://chocolatey.org/docs/installation#installing-chocolatey
- Now, install kubernetes tool with this command:

```
>choco install kubernetes-cli
```

Make sure the installation is successfully with this command:

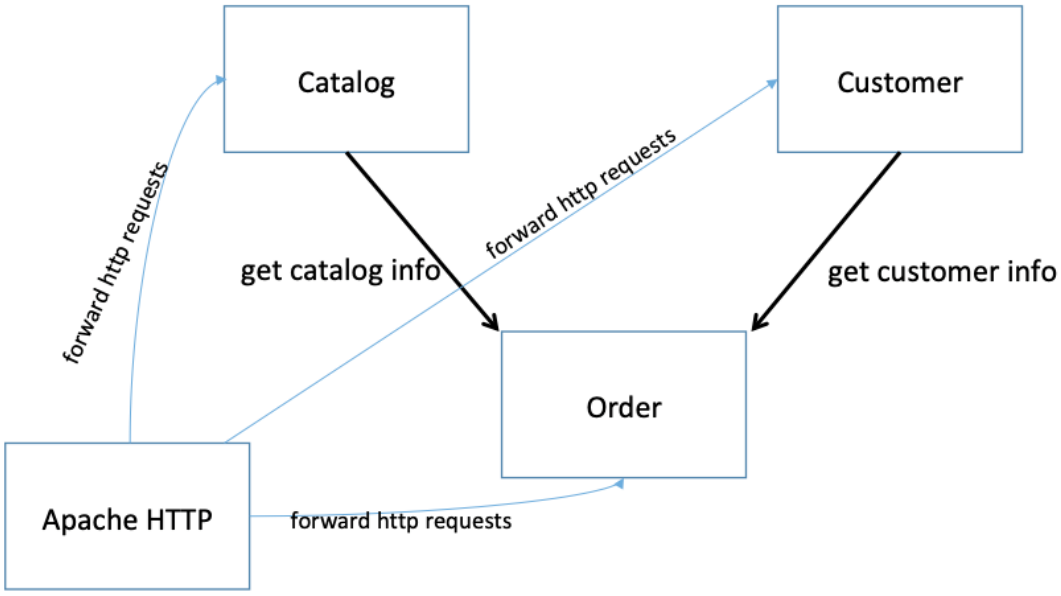
```
>kubectl version
```

- Install the file *minikube-windows-amd64.exe*:
https://github.com/kubernetes/minikube/releases
You have to rename it to *minikube.exe* and add it to your path. (Here is how to edit your path variable: https://www.java.com/en/download/help/path.xml)
- To install Docker, there is a problem on Windows here. We know that Docker for Windows requires Hyper-V to work, however VirtualBox does not work with Hyper-V enabled (?!?). So, there is a solution here: Use the Docker Toolbox (instead of official Docker for Windows): https://docs.docker.com/toolbox/overview/

Building web application in using microservices architecture

Now we will develop a simple web application in using microservices architecture. Concretely, we construct 4 services as follows:

1. Service *Apache HTTP*: Apache HTTP is used to provide the web page of the demo at port 8080. It also forwards HTTP requests to the 3 other microservices. This is not really necessary as each service has its own port on the Minikube host but it provides a single point of entry for the whole system. Apache HTTP is configured as a reverse proxy for this. Load balancing is left to Kubernetes.
2. Service *Order*: to process orders. This service is connected to customer service and catalog service to get information.
3. Service *Customer*: to handle customer data.
4. Service *Catalog*: to handle the items in the catalog.



The source code is available at the link:
https://github.com/anhth318/microservices-demo

Use Git Bash, go to the folder where you want to place the folder of source code, type:

```
>git clone https://github.com/anhth318/microservices-demo.git
```

Go into the folder *microservices-demo*, then run:

```
./mvnw clean package -Dmaven.test.skip=true
```

or on the Windows:

```
mvnw.cmd clean package -Dmaven.test.skip=true
```

The command above is to build/re-build the 3 mentioned services.
Next, you create an account at the public Docker Hub: https://hub.docker.com/

Run the docker toolbox in your machine in double clicking on the icon *Docker Quickstart Terminal*.

Now, you will put our 4 services into docker images. After, you will upload it to the public Docker Hub server:
First, you have to log in to DockerHub. Open a terminal window:

```
>docker login
```

After, you provide the username and password of your DockerHub account.

Now, go to the working folder:

For the service *apache*:

```
>docker build --tag=microservice-kubernetes-demo-apache apache
```

```
>docker tag microservice-kubernetes-demo-apache your_docker_account/microservice-kubernetes-demo-apache:latest
```

```
>docker push your_docker_account/microservice-kubernetes-demo-apache
```

You do the same thing with 3 other services.

Question 8: What are the commands did you use?

Question 9: Open the website Docker Hub and login with your account. What's new in your docker hub repository?

FYI, Minikube is a lightweight Kubernetes implementation that creates a VM on your local machine and deploys a simple cluster containing only one node. Minikube is available for Linux, macOS, and Windows systems.
Now, it's time to use Minikube tool to create a cluster with only one *Kubernetes node* (a virtual machine) that will hosts all your running applications/services (under type of pods) handled by a *Kubernetes master.*

```
>minikube start
```

You deploy the services in using uploaded docker images in your Docker Hub.
You open the file *microservices.yaml* and replace all my docker hub account (anhth) with your own account of every line that begins with the word *image*, like the one below:

```
- image: docker.io/anhth/microservice-kubernetes-demo-apache:latest
```

Save the file and close it. Run the command below to deploy the images in your Docker Hub:

```
>kubectl apply -f microservices.yaml
```

The command above creates Pods. Pods might contain one or many Docker containers. In this case each Pod contains just one Docker container. Also services are created. Services have a cluster wide unique IP address and a DNS entry. Service can use many Pods to do load balancing.

Use this command to show all the information of your kubernetes cluster:

```
>kubectl get all
```

Question 10: What is the status of these created pods? Now, wait few minutes and re-type this command, what is the new status of these pods?

For more details, run `kubectl describe services`. This also works for pods (`kubectl describe pods`) and deployments (`kubectl describe deployments`).

You can see the logs of a pod (replace your pod ID):

```
>kubectl logs catalog-269679894-60dr0
```

You can even open a shell in a pod:

```
>kubectl exec catalog-269679894-60dr0  -it /bin/sh
```

You wait until all the status of the pods change to "Running", that means it's ready to run your application. Type the command below:

```
>minikube service apache
```

It will open the web page of the Apache httpd server in the web browser.
Now, enjoy your running application.
Click to the link "Customer", you will see all the customers' names. After, click "Add Customer" to add new customer.
Return to the Home page, do the same thing with Catalog and add new items.
Return to the Home page, click on the link "Order", and try to add some orders.

In the end, don't forget to remove all the services and deployments:

```
>kubectl delete service apache catalog customer order
>kubectl delete deployments apache catalog customer order
```

and stop the cluster:

```
>minikube stop
```

4. JMS and DDS architectures

4.1. Contents

In this labwork's section, we'll construct 2 architectures based on JMS and DDS. The goal is to help you understand the theory of event-based architecture.

4.2. Requirements

4.2.1. Theory

- Handling Unix OS
- Base of event-based architecture, the publish/subscribe model.
- Java and C++ language

4.2.2. Hardware

- PC with Ubuntu OS

4.2.3. Software

4.3. PRACTICAL STEPS

4.3.1. JMS

JMS (Java Message Service) is an API that provides the facility to create, send and read messages. It provides loosely coupled, reliable and asynchronous communication. JMS is also known as a messaging service.

Install the server glassfish:

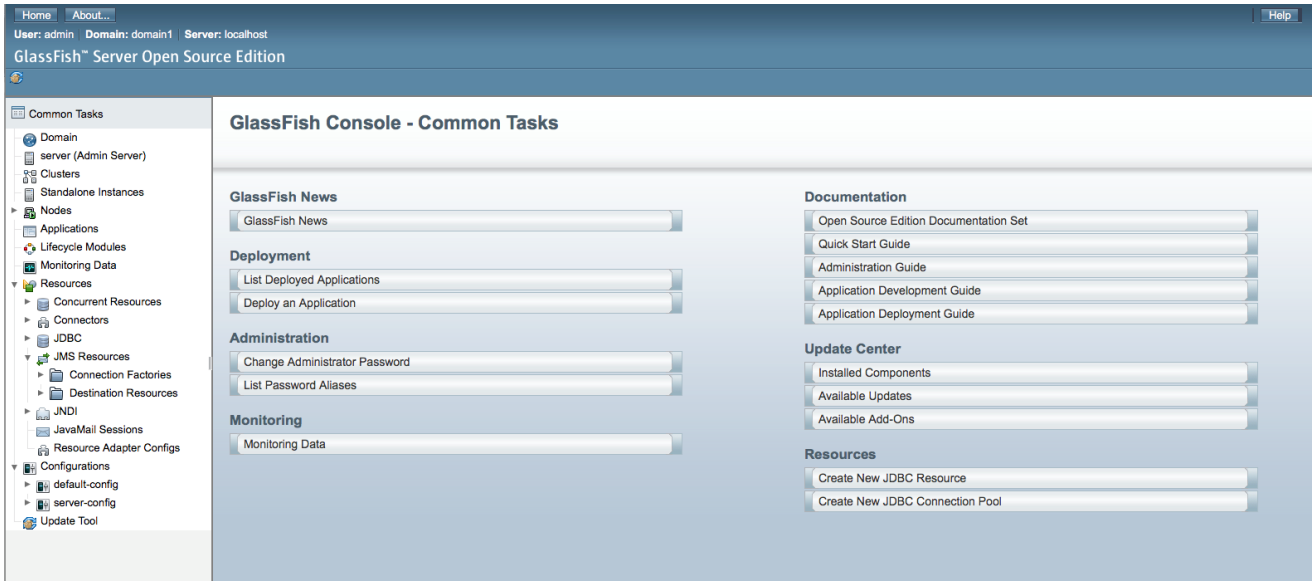
- Download it at this link http://download.java.net/glassfish/4.1.1/release/glassfish-4.1.1.zip
- Unzip to the folder glassfish4.
- Start the glassfish:

```
glassfish4/bin/asadmin start-domain
```

Now, server glassfish has made running a domain called *domain1.* In addition, glassfish supports the web interface in the port 4848. Use your web browser and go to address:

http://localhost:4848

You'll see the web interface as below. Make attention to the *JMS Resources* part where we have to create *Connection Factories* and *Destination resources*.



Question 11: What is the role of application server glassfish?

Creating 2 JNDI

Now, we have to create 2 JNDI: *myTopicConnectionFactory* and *myTopic*.
Normally, you can use the web interface, however there will be some errors. So, you are recommended to create 2 JNDI in using command line.
Go to the folder *glassfish4/bin/* and type the command:

```
./asadmin
```

Create resource Connection Factory

```
asadmin>create-jms-resource --restype
javax.jms.TopicConnectionFactory
```

Then, you will be asked the name of JNDI, type *myTopicConnectionFactory*

```
Enter the value for the jndi_name
operand>myTopicConnectionFactory
```

Create resource Destination:

```
asadmin> create-jms-resource --restype javax.jms.Topic
```

In the same way, you type the jndi name: *myTopic*

Go to the web interface to check if these two JNDI have been created.

Question 12: Why do we need to create the 2 JNDI above?

Construct the Sender and Receiver.

In this step, you will use the Java language, you are recommended to use IDE Eclipse.

Create a new project and name it: *JMSTopicProject*.
Attention: you have to add the following libraries into the project:

- *gf-client.jar*: in the folder `glassfish4/glassfish/lib`
- *javax.jms.jar*: download it from the Internet.

Create three files representing 3 classes: *MySender.java, MyReceiver.java,* and *MyListener.java*
Here are the content of these files:

File: MySender.java

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import javax.naming.*;
import javax.jms.*;

public class MySender {
   public static void main(String[] args) {
      try
      {   //Create and start connection
         InitialContext ctx=new InitialContext();
         TopicConnectionFactory f=(TopicConnectionFactory)ctx.lookup("myTopic
ConnectionFactory");
         TopicConnection con=f.createTopicConnection();
         con.start();
         //2) create queue session
         TopicSession ses=con.createTopicSession(false, Session.AUTO_ACKNOW
LEDGE);
         //3) get the Topic object
         Topic t=(Topic)ctx.lookup("myTopic");
         //4)create TopicPublisher object
         TopicPublisher publisher=ses.createPublisher(t);
         //5) create TextMessage object
         TextMessage msg=ses.createTextMessage();

         //6) write message
         BufferedReader b=new BufferedReader(new InputStreamReader(Syste
m.in));
         while(true)
         {
            System.out.println("Enter Msg, end to terminate:");
            String s=b.readLine();
```

```
            if (s.equals("end"))
               break;
            msg.setText(s);
            //7) send message
            publisher.publish(msg);
            System.out.println("Message successfully sent.");
         }
         //8) connection close
         con.close();
      }catch(Exception e){System.out.println(e);}
   }
}
```

File: MyReceiver.java

```
import javax.jms.*;
import javax.naming.InitialContext;

public class MyReceiver {
   public static void main(String[] args) {
      try {
         //1) Create and start connection
         InitialContext ctx=new InitialContext();
         TopicConnectionFactory f=(TopicConnectionFactory)ctx.lookup("myTopic
ConnectionFactory");
         TopicConnection con=f.createTopicConnection();
         con.start();
         //2) create topic session
         TopicSession ses=con.createTopicSession(false, Session.AUTO_ACKNOW
LEDGE);
         //3) get the Topic object
         Topic t=(Topic)ctx.lookup("myTopic");
         //4)create TopicSubscriber
         TopicSubscriber receiver=ses.createSubscriber(t);

         //5) create listener object
         MyListener listener=new MyListener();

         //6) register the listener object with subscriber
         receiver.setMessageListener(listener);

         System.out.println("Subscriber1 is ready, waiting for messages...");
         System.out.println("press Ctrl+c to shutdown...");
         while(true){
            Thread.sleep(1000);
         }
      }catch(Exception e){System.out.println(e);}
   }
```

```
}
```

File: MyListener.java

```
import javax.jms.*;
public class MyListener implements MessageListener {

   public void onMessage(Message m) {
      try{
      TextMessage msg=(TextMessage)m;

      System.out.println("following message is received:"+msg.getText());
      }catch(JMSException e){System.out.println(e);}
   }
}
```

Congratulations! You have been done. Now it's time to test the Sender and the Receiver.

Question 13: Explain the message passing method of Sender and Receiver in basing on the theory of event-based architecture.

4.3.2. DDS

In this section, we will take a look at the DDS model. Concretely, we will install the open source software OpenDDS.

Install OpenDDS

You have to install the 3 following programs:

- C++ compiler
- GNU Make
- Perl

Download the file .tar.gz (the newest version) at this link:
http://download.ociweb.com/OpenDDS/

Decompress the file:

```
tar -xvzf OpenDDS-3.8.tar.gz
```

Go the decompressed folder and type the 2 following commands:

```
./configure
make
```

Type this command to configure the environment parameters:

```
source setenv.sh
```

Go to the folder:

```
cd OpenDDS-3.8/tests/DCPS/Messenger/
```

Create and edit the content of the file *rtps.ini* as follows:

[common]
DCPSGlobalTransportConfig=$file
DCPSDefaultDiscovery=DEFAULT_RTPS
[transport/the_rtps_transport]
transport_type=rtps_udp

Start the subscriber:

```
./subscriber -DCPSConfigFile rtps.ini
```

Open a new tab to run publisher:
(attention: you have also to run the *source setenv.sh* command in the new tab)
Go to the same folder and run:

```
./publisher -DCPSConfigFile rtps.ini
```

Question 14: Compare the JMS and DDS.