**Fundamentals of Java Messaging Service Training**

**Class Notes on 31-jan-2023**

<https://docs.google.com/document/d/1wVaUqh7pVp0VHpqgZc7QNYsin4_EzP5qOXAUjs9tA_g/edit?usp=sharing>

JMS - Just an API - Collection of interfaces (just like JDBC API)

* We need vendors to implement for this
* Such as glassfish, websphere, jboss etc.
* The implementors of JDBC API are: ORACLE (ojdbc8.jar), MYSQL, etc.
* Using this, we can send small amount of messages from one application to another

Example: customer’s details will be sent from one application to another for further processing

* The scope is very narrow

IBM MQ is a complete middleware message oriented platform that enables the exchange of data between applications, systems, and services. IBM MQ internally uses JMS api for achieving this

RabbitMQ can handle large volumes of messages and handle high concurrent requests through parallel processing, while JMS may not be as scalable

JMS is just the basic - first step towards messaging

Where, IBM MQ, RabbitMQ are the middleware solutions built on top of JMS

IBM MQ & Rabbit MQ are products that use JMS as its core API for messaging

They can also build other features around it such as scalability, performance, concurrency etc.

Kafka is again another messaging system

But, its scope is wider

While the former technologies provide the messaging solutions between the applications in a small quantity, kafka provides the solutions where you can send/receive HUMONGOUS amount of messages (petabytes)

We can use kafka when dealing with UNSTRUCTURED DATA in applications such as FACEBOOK, WHATSAPP, STOCK EXCHANGE, TWITTER….

This falls into the category of BIG DATA (hadoop, kafka, spark…)

Employee DB TABLE:

EMP\_ID EMP\_NAME EMP\_SALARY LOCATION PROJECT

1 Guru 383.2 BLR PROJ1

NOSQL DB

Unstructured data doesn’t follow any schema

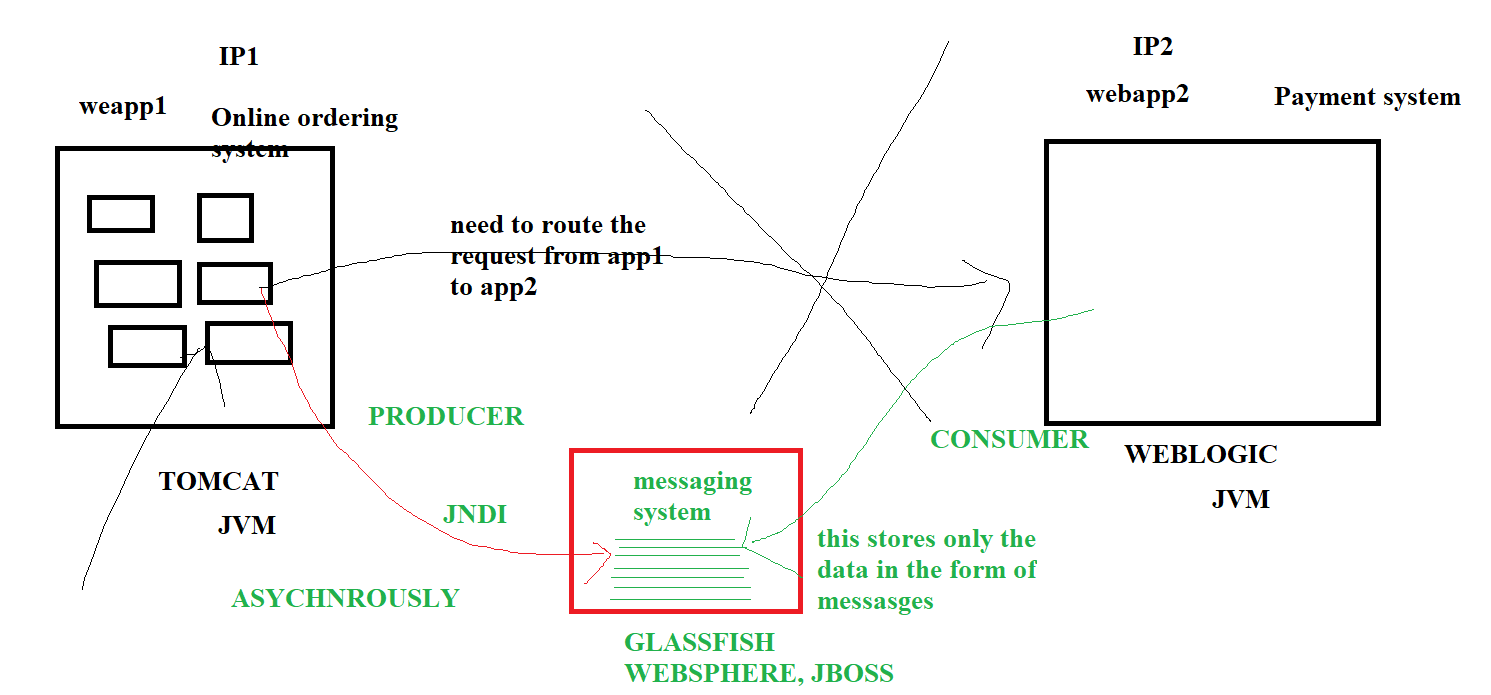
First row may contain 5 columns

2nd row may contain only 3 columns of data

We basically use NOSQL DB for storing voluminous data (JSON format, or some other format)

Application of JMS:

JMS in a nutshell:



JMS supports different types of messages:

String

Object (You want to store the Employee’s data, you have Employee java class - structured data)

You can instantiate Employee class and populate the object and you can store this object into JMS

Object type: it could be anything, internally all data is stored in SERIALIZED FORMAT

Messaging is a concept

JMS is a messaging concept implemented in Java

JMS is one of the ways to implement messaging in Java

Kafka is another messaging system

JMS deals with smaller amount of data/messages

You can send/receive messages from applications implemented only in Java

Kafka deals with huge amount of data/messages

You can send/receive messages from applications implemented in different languages

JMS is an API (Application Programming Interface) - a collection of classes & methods

Java API usually don’t contain the implementation (JDBC, JMS..)

JMS API provides us a set of unimplemented methods (they are called abstractions)

Any interested party can implement them

JSM is an API

Glassfish provides an implementation for JMS API

WebLogic/WebSphere is another JMS implementation provider

That means, they provide a set of JAR FILES that we need to add to our project

IBM MQ, RABBIT MQ are exclusive messaging platform/middle solutions that provides other features other than just messaging

Asynchronous type

Message will arrive automatically to the client

Client need not be present when the sender sends the message

JMS is a push-type in nature where the JMS providers (glassfish) upon receiving messages in queue/topic will push the messages to the consumers automatically. (this is an observer design pattern)

Kafka is a pull-type messaging system, consumers have to pull the required messages by specifying the offset or whatever

In JMS, messages are stored in RAM (main memory), once the messages are consumed, they get deleted permanently or won’t be persisted, unless we use some external mechanism (we need to write these data into DB tables by ourselves)

RAM is limited (1 GB, 4 GB, 16 GB, 32 GB)

In Kafka, messages are stored for a defined amount of time (By default, it stores for 7 days)

These messages will be there in the kafka system for 7 days, whether they are consumed or not (HARD DISK), if you want to store more messages, you can increase the HD capacity, it also supports clustering feature, whereas you can keep storing the messages into multiple HARD DISKs which are kept in different SERVERS (you can maintain 1000 physical servers with 1000 HARD DISK with 1 TB of data)

JMS supports both Queues & topics

Praveen:

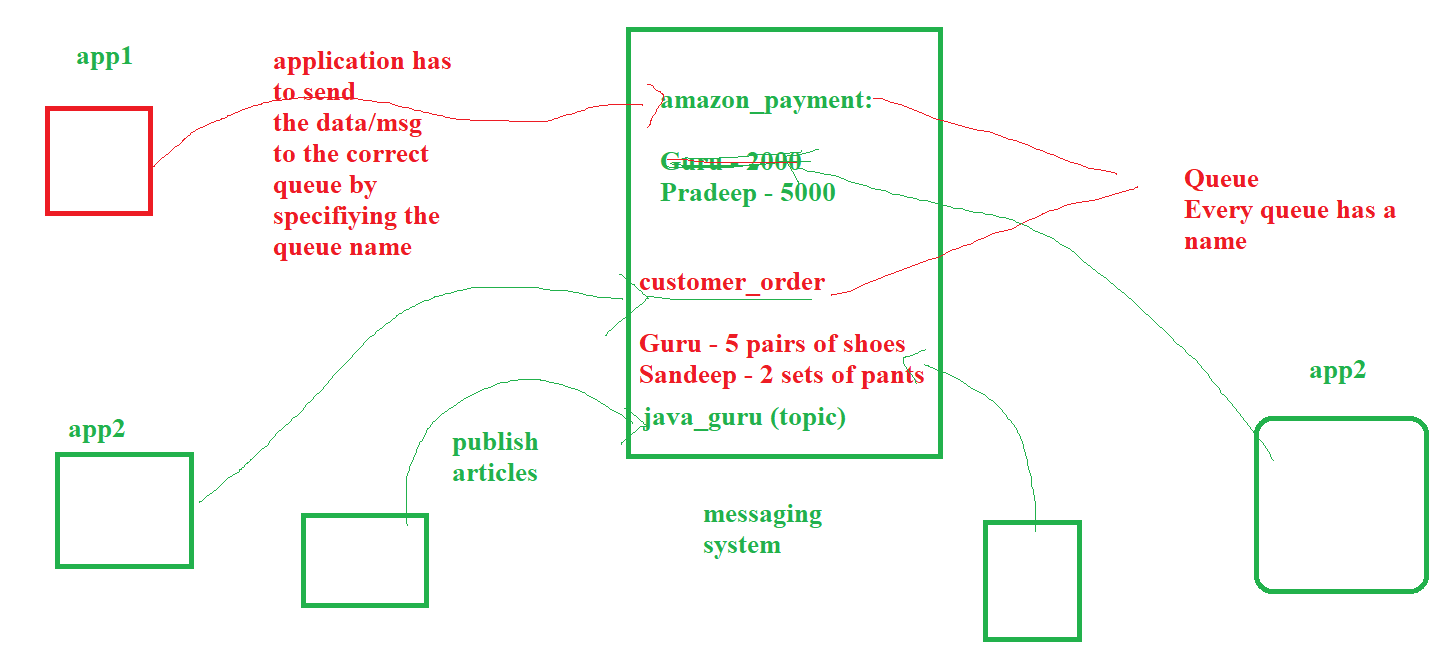
Can you pls share if JPA related to JMS by any means?

NO

JPA stands for Java Persistence Architecture, it is related to storing the data into DB TABLE

You can store data using JDBC API, you need to write lot of boilerplate codes

JPA provides various classes, using which your persistence tasks becomes so easier, you don’t need to deal with SQL Queries, it supports the ORM concept (Hibernate )



JMS supports both Queue & topic

Kafka supports only topic

Time: 11.35

Let’s take a break for 15 minutes and return

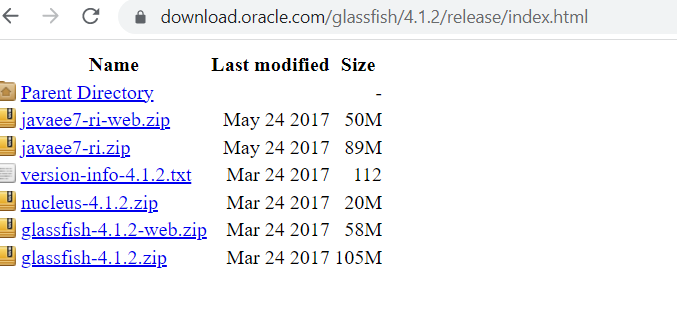
Summary:

1. Queue (1 to 1)
2. Topic (1 to many)

Using Glassfish server, we are going to create Queues & Topics

– End of Theory –

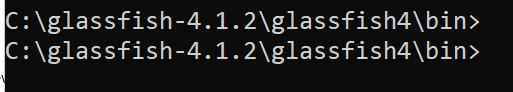
We are going to start the glassfish server



We need the following s/w:

1. JDK1.8
2. ECLIPSE IDE
3. GLASSFISH

Change the directory to glassfish/bin:



asadmin start-domain domain1

Here, asadmin is the command name

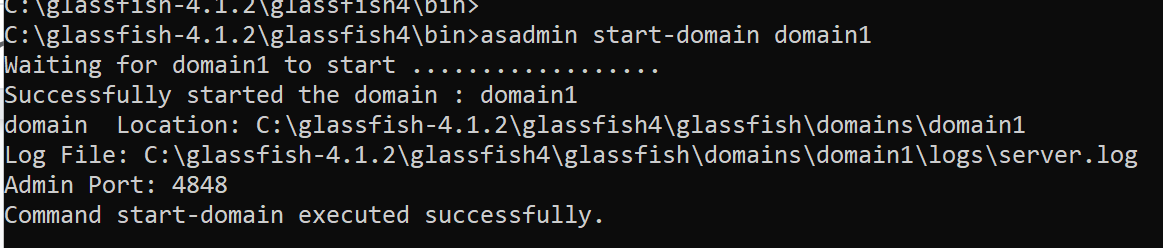
start-domain is the parameter

Domain1 is the pre-createed domain during glassfish server installation

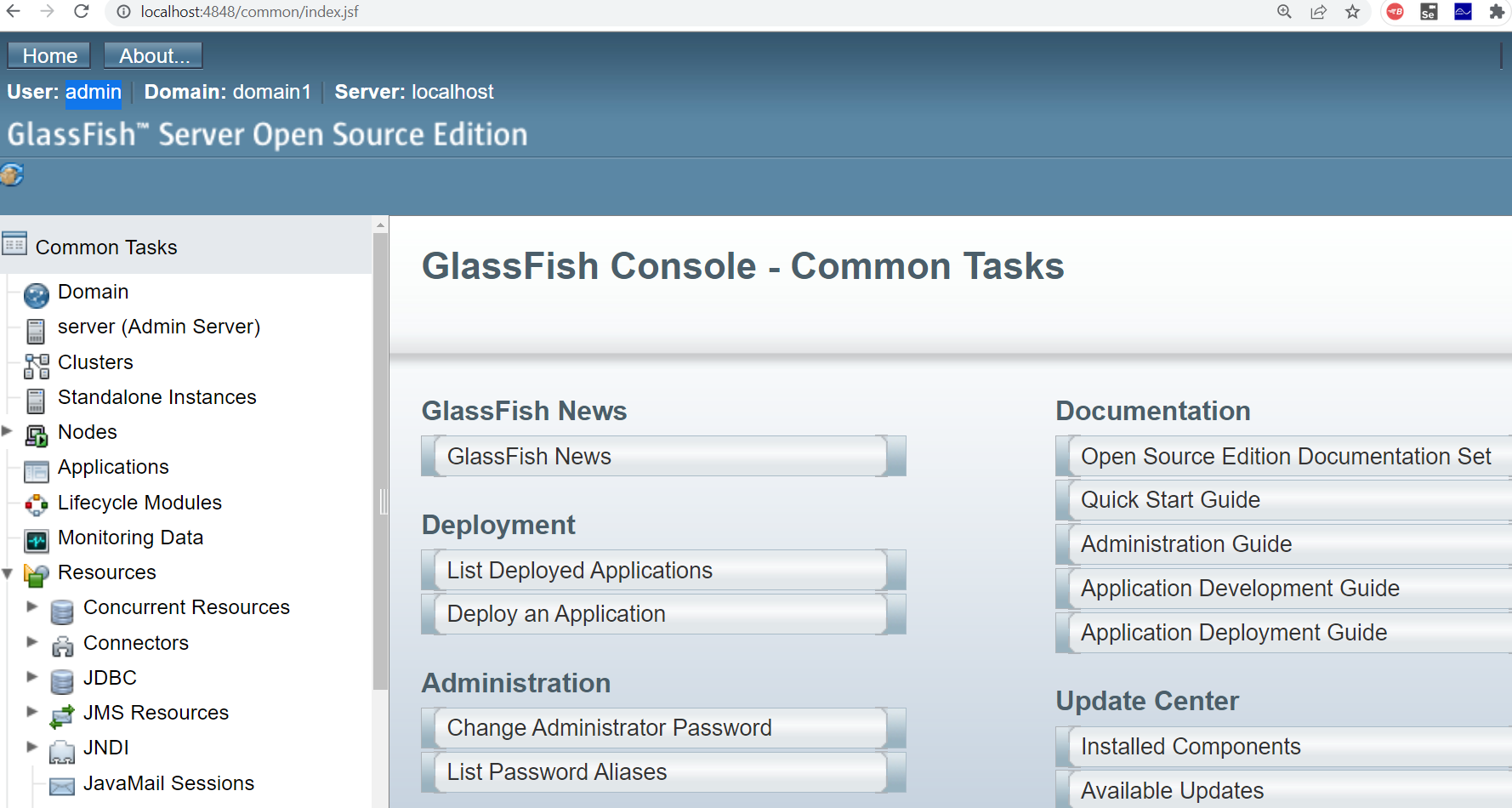
You need to check the compatibility between glassfish version and jdk version

In my lab setup doc, I have specified that glassfish 4.1.2 goes well with jdk1.8

If you have greater version of jdk, you have to check the compatibility and download the required version of glassfish server



Once this is successfully executed, open the browser and point to localhost:4848



5 minutes to setup this

To create our own domain:

asadmin create-domain –adminport <port-number> <domain-name>

Example:

asadmin create-domain --adminport 7800 guru-domain

Q: What is the significance of domain in glassfish?

A domain is a logically related group of glassfish server instances that are managed as a single unit.

A domain typically includes one or more instances of Glassfish servers, and each instance can host its own applications

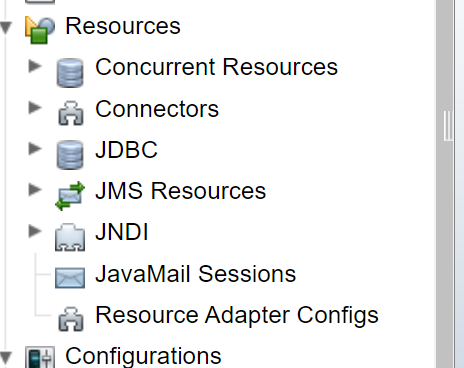
This also includes the various resources that are used in all the applications

JNDI

JMS

CONNECTOR

JDBC



To stop the running domain:

asadmin stop-domain domain1

This will stop the running domain

Summary:

1. asadomain start-domain domain1

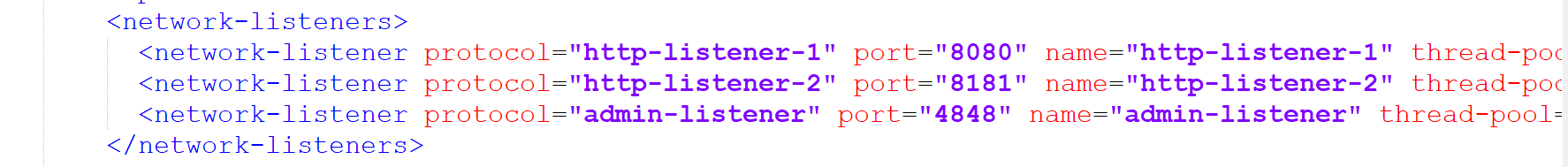
2. asadomain stop-domain domain1

3. asadmin create-domain --adminport 7800 guru-domain

**How to change the default port no?**

There is a configuration file called “domain.xml” under “domains\domain1\config” folder

Open this and look for 4848 and change this to whatever port no you want



You must restart the server

Q: Can we create jms queue in tomcat server?

No, Apache tomcat server is a servlet container, and doesn’t include the built-in support for JMS

Because, JMS is an enterprise feature

Only the application servers can support

Such as: glassfish, weblogic, websphere, jboss

**How to create a connection factory?**

We need to start from the connection factory always

First, after the domain is started, enter “asadmin”

asadmin>

asadmin> create-jms-resource --restype javax.jms.ConnectionFactory --description "testconnection" jms/cf1

Connector resource jms/cf1 created.

Command create-jms-resource executed successfully.

asadmin>

To verify this, open JMS resources and check for the connection factories in the admin console



Next stop is creating the destination type (queue or topic)

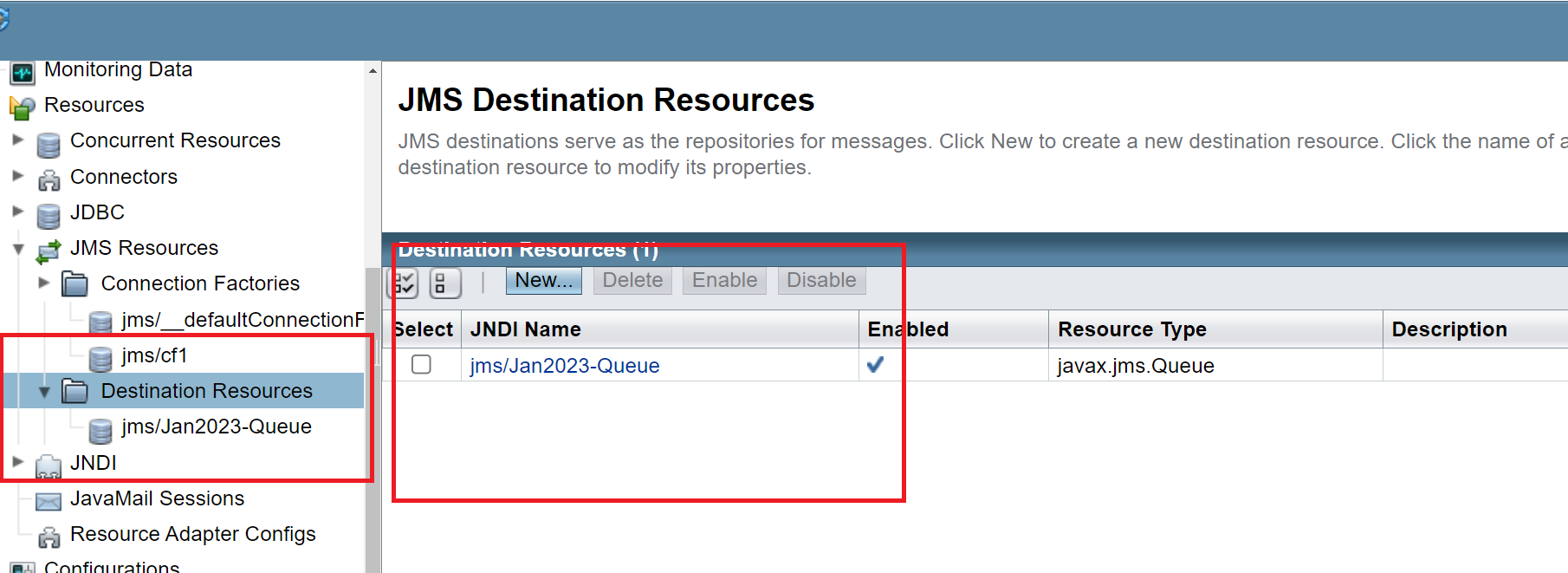
For this exercise, let’s create a Queue

asadmin> create-jms-resource --restype javax.jms.Queue --property Name=PhysicalQueue jms/Jan2023-Queue

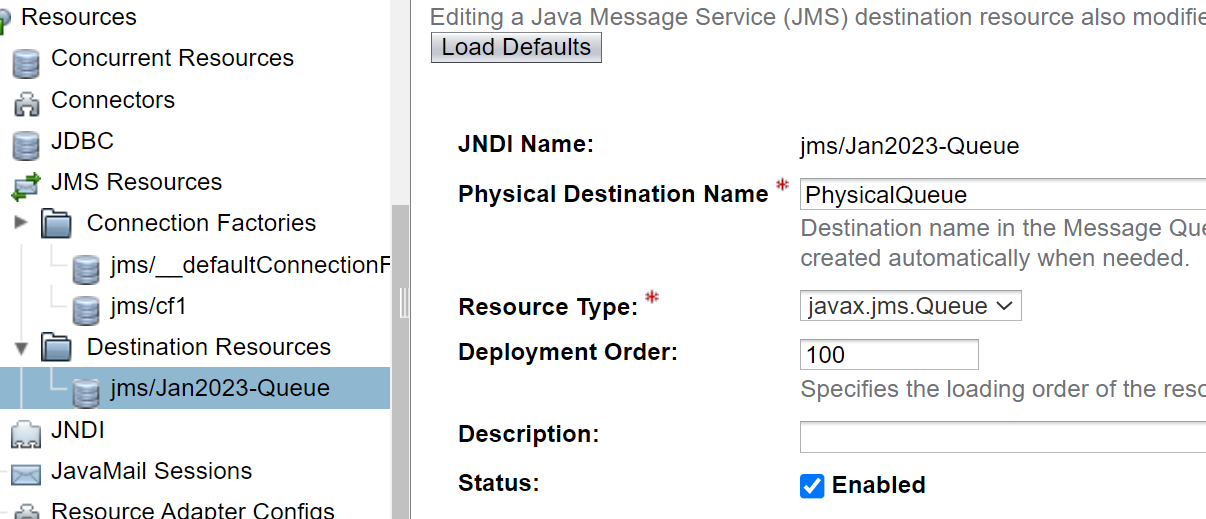
Administered object jms/Jan2023-Queue created.

Command create-jms-resource executed successfully.

asadmin>



Click on this queue name on your left:



From the above screenshot, we can see that queue name has been displayed along with the property called “JNDI”

JNDI stands for Java Naming Directory Interface

It is like yellow pages

As already told, app server contains many resources to support the applications

Resources such as: DB Resources, JMS resources, resources related to connection, security etc.

If you want to create DB connection pools, that can be configured in this admin console

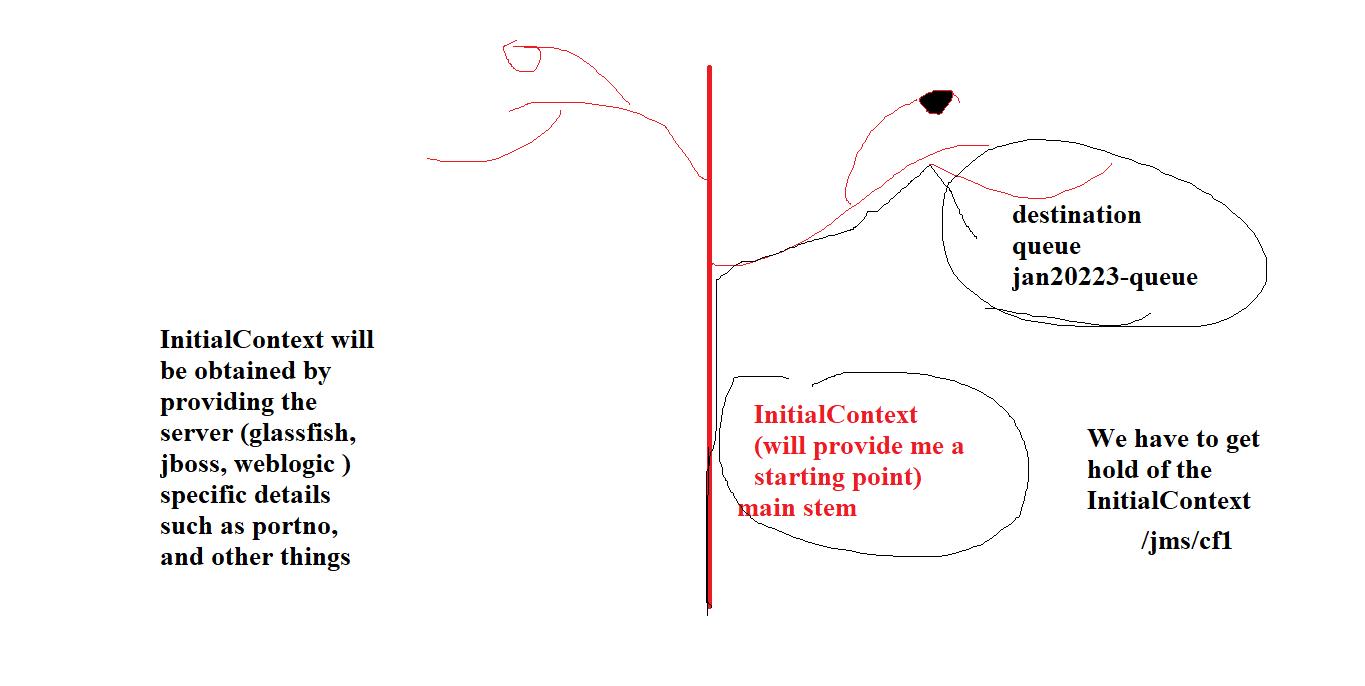
Technical definition of JNDI:

A JAVA API, that allows java programs to look up data and objects via name (just like yellow pages)

LDAP, DNS, NIS and all the resources can be defined using JNDI

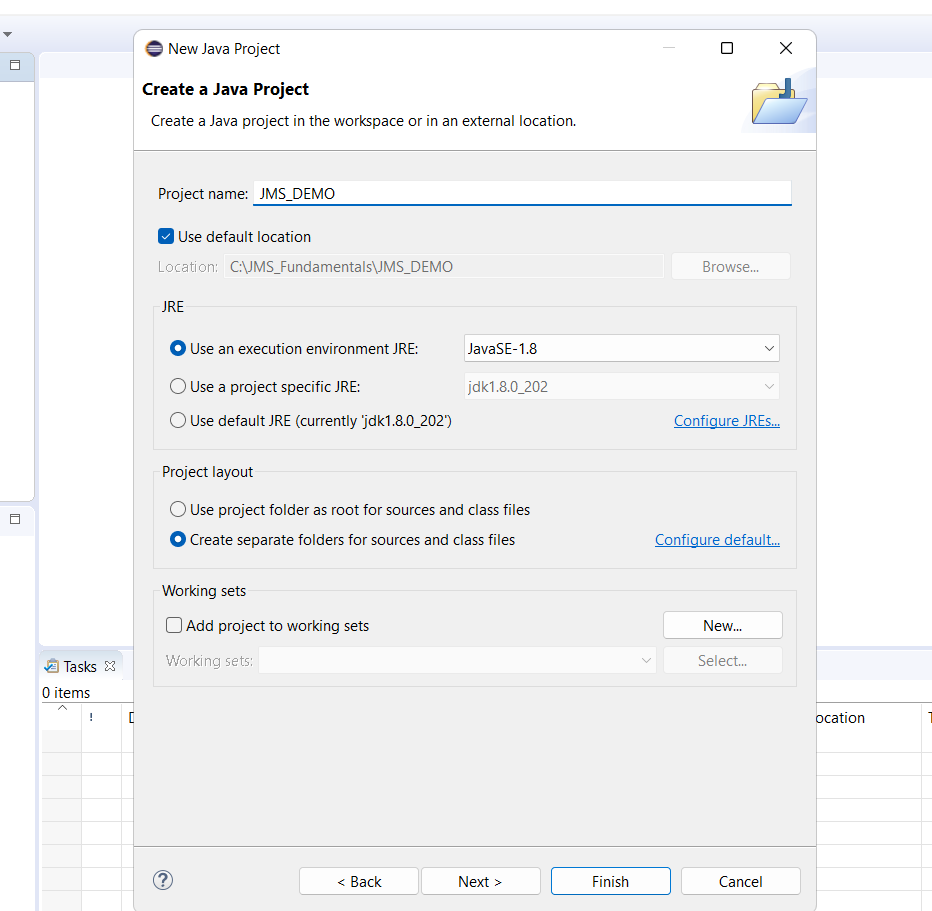
JNDI acts as an intermediary between a client and a naming or directory service

It provides a standard interface for naming and directory services



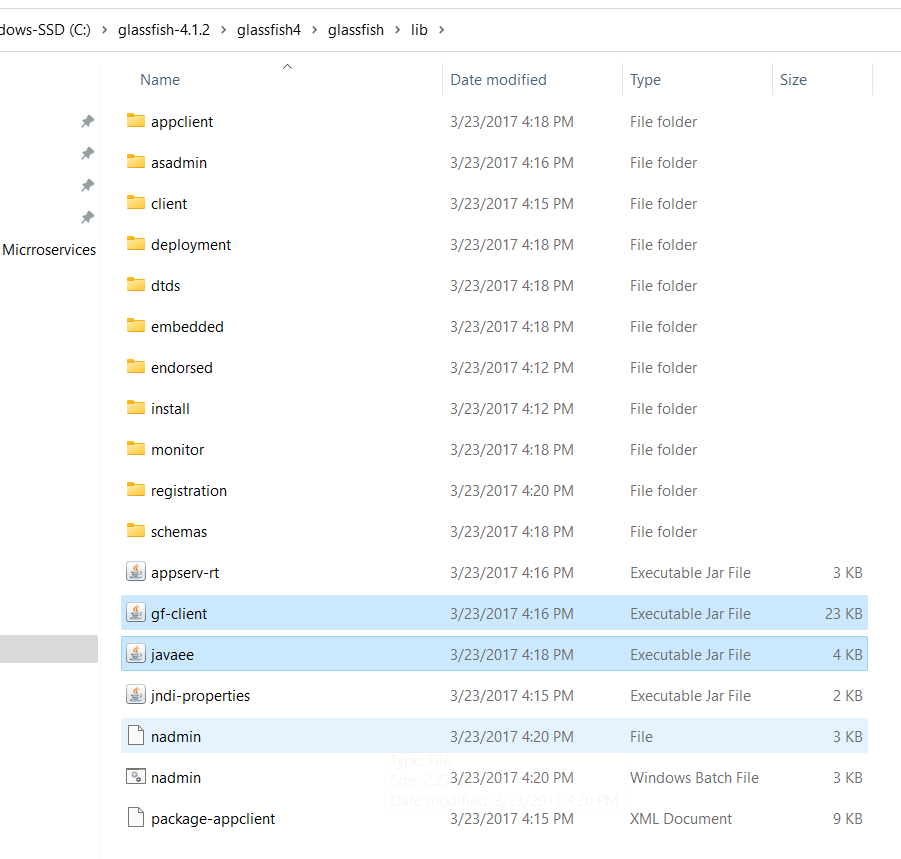
Let’s create Java project “JMS\_DEMO” in Eclipse

Select “jdk1.8”

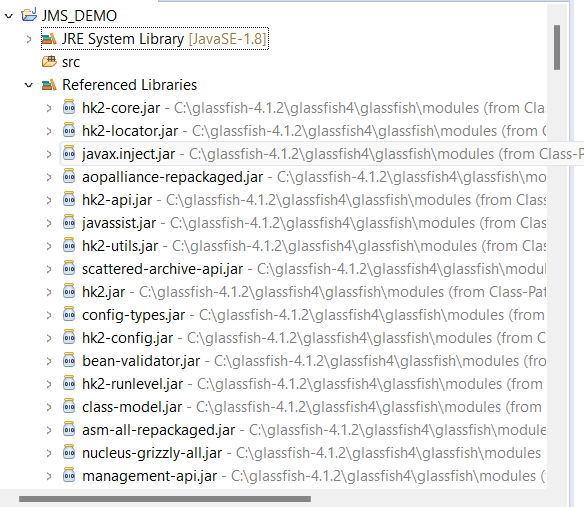


We must add the following two “jar” files from Glassfish’s installation directory

They are “JMS IMPLEMENTATION” class files



Now, your project looks like this:



Summary:

1. We have started the domain
2. We have created the following JMS resources:
   1. Connection Factory
   2. Queue
3. We have setup Java project in Eclipse (JMS\_DEMO)
4. We have added the required .jar files (JMS implementation ) into this project

At this point, we are ready to write JMS programs to send/receive messages

Time: 1 pm

Let’s take a lunch break and return at 2 pm

**Program 1: Just send a receive a simple text message**

Sender.java and copy the following code:

package com.ofss;

import java.util.Properties;

import javax.jms.Queue;

import javax.jms.QueueConnection;

import javax.jms.QueueConnectionFactory;

import javax.jms.QueueSender;

import javax.jms.QueueSession;

import javax.jms.TextMessage;

import javax.naming.InitialContext;

import com.sun.messaging.jmq.jmsserver.core.Session;

public class Sender {

public static void main(String[] args) {

try {

Properties props=new Properties();

// The property names will be the same always

// Step1: obtaining the Initial Context object

// The property values will be different depends on jms implementor

props.setProperty("java.naming.factory.initial", "com.sun.enterprise.naming.SerialInitContextFactory");

props.setProperty("java.naming.factory.url.pkgs", "com.sun.enterprise.naming");

props.setProperty("java.naming.provider.url", "iiop://localhost:3700");// Inter ORB protocol of glassfish server

InitialContext ctx=new InitialContext(props);

// Step 2:

// since this method returns Object, we need to typecast it accordingly

// If you deal with Queue, then you need to typecast this object into QueryConnectionFactory object

QueueConnectionFactory f=(QueueConnectionFactory)ctx.lookup("jms/cf1");

QueueConnection con=f.createQueueConnection();

// Step 3: Create a queue session from this connection

QueueSession ses=con.createQueueSession(false, Session.AUTO\_ACKNOWLEDGE);

// Step 4: Get the queue name using JNDI

Queue q=(Queue)ctx.lookup("jms/Jan2023-Queue");

// Step 5: Create a sender object (because we are going to send the msg)

QueueSender sender=ses.createSender(q);

// Step 6: Create TextMessage now

TextMessage msg=ses.createTextMessage();

msg.setText("Welcome to the first msg");

// Step 7: now, call the send method

sender.send(msg);

System.out.println("Message has been successfully produced and sent to the given queue");

}

catch(Exception e)

{

e.printStackTrace();

}

}

}

Run this as Java application

If everything goes fine, this msg must be stored into the queue you mentioned

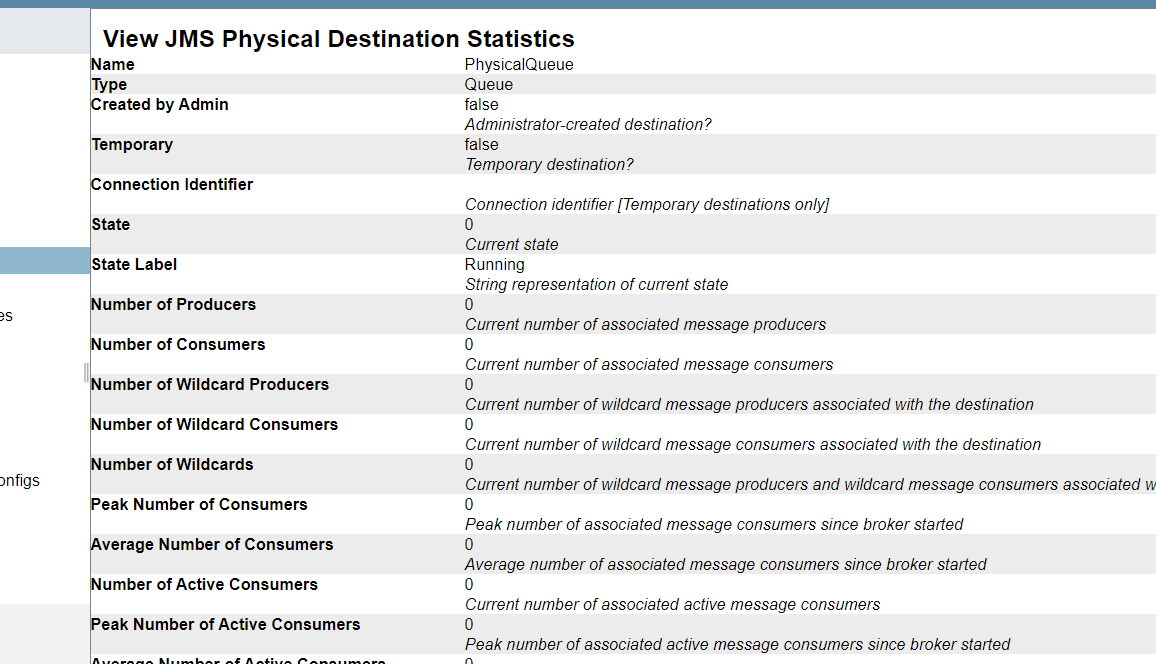
5 minutes

How to see the message information in the admin console?

Click on Monitoring data

Click on Resource

Click on JMS Physical Destination



Currently, glassfish doesn’t provide the feature to view the messages

You could store various types of messages (text, object..)

About the following properties you have set:

props.setProperty("java.naming.factory.initial", "com.sun.enterprise.naming.SerialInitContextFactory");

props.setProperty("java.naming.factory.url.pkgs", "com.sun.enterprise.naming");

props.setProperty("java.naming.provider.url", "iiop://localhost:3700");// Inter ORB protocol of glassfish server

InitialContext ctx=new InitialContext(props);

This sets the properties for JNDI Initial Context

java.naming.factory.initial -> specifies the initial context factory to be used for creating Initial Context object

Java.naming.factory.url.pkgs → Specifies the package prefixes to use when searching for a context factory

Java.naming.provider.url → specifies the URL of the provider to use. In this case, it is using the Inter ORB protocol (IIOP) and the provider URL is set to the glassfish server running on localhost on port no 3700 (This is not the admin port no)

5 minutes

**A word on IIOP:**

Stands for Internet Inter-ORB protocol

This is a network protocol used for distributed object communication

CORBA - Common Object Request Broker Architecture, this is precursor to our modern SOA (Service Oriented Architecture such as SOAP and RESTful)

We could communicate from Java program to C++ program and any other program running on different machines (different IP addresses)

We have dealt with two applications running on two JVMs

Those days, we were using RMI, CORBA, IDL….

These days, all these have been replaced by SOA (SOAP, REST)

First one is our Sender program that is executed on JVM1

Second one is our glassfish server that is running on JVM2

Program 2: Let’s send a series of numbers

For this, I am going to write a for loop and then will be sending the numbers to the queue

Create Sender1.java

package com.ofss;

import java.util.Properties;

import javax.jms.Queue;

import javax.jms.QueueConnection;

import javax.jms.QueueConnectionFactory;

import javax.jms.QueueSender;

import javax.jms.QueueSession;

import javax.jms.TextMessage;

import javax.naming.InitialContext;

import com.sun.messaging.jmq.jmsserver.core.Session;

// This program sends a series of numbers to the queue

public class Sender1 {

public static void main(String[] args) {

try {

Properties props=new Properties();

// The property names will be the same always

// Step1: obtaining the Initial Context object

// The property values will be different depends on jms implementor

props.setProperty("java.naming.factory.initial", "com.sun.enterprise.naming.SerialInitContextFactory");

props.setProperty("java.naming.factory.url.pkgs", "com.sun.enterprise.naming");

props.setProperty("java.naming.provider.url", "iiop://localhost:3700");// Inter ORB protocol of glassfish server

InitialContext ctx=new InitialContext(props);

// Step 2:

// since this method returns Object, we need to typecast it accordingly

// If you deal with Queue, then you need to typecast this object into QueryConnectionFactory object

QueueConnectionFactory f=(QueueConnectionFactory)ctx.lookup("jms/cf1");

QueueConnection con=f.createQueueConnection();

// Step 3: Create a queue session from this connection

QueueSession ses=con.createQueueSession(false, Session.AUTO\_ACKNOWLEDGE);

// Step 4: Get the queue name using JNDI

Queue q=(Queue)ctx.lookup("jms/Jan2023-Queue");

// Step 5: Create a sender object (because we are going to send the msg)

QueueSender sender=ses.createSender(q);

// Step 6: Create TextMessage now

TextMessage msg=ses.createTextMessage();

for (int i=1;i<=10;i++)

{

msg.setText(String.valueOf(i));

System.out.println("Sending the message "+String.valueOf(i));

sender.send(msg);

}

System.out.println("All the Messages have been successfully produced and sent to the given queue");

}

catch(Exception e)

{

e.printStackTrace();

}

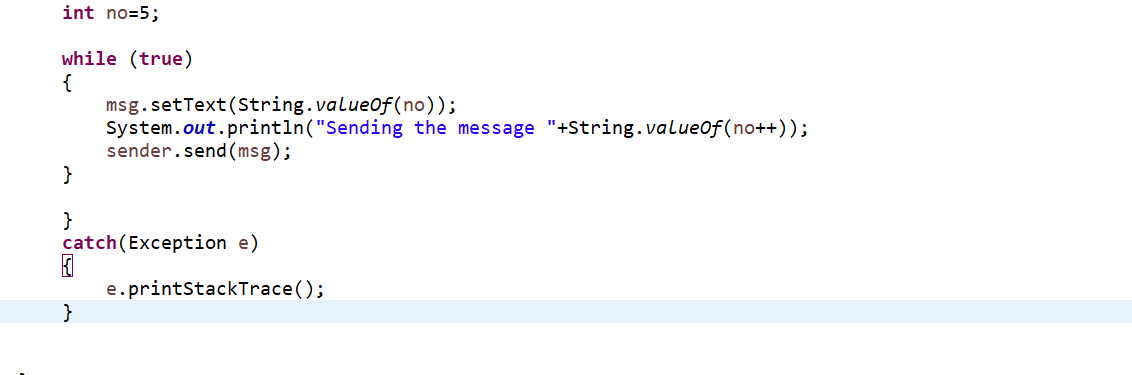
}

}

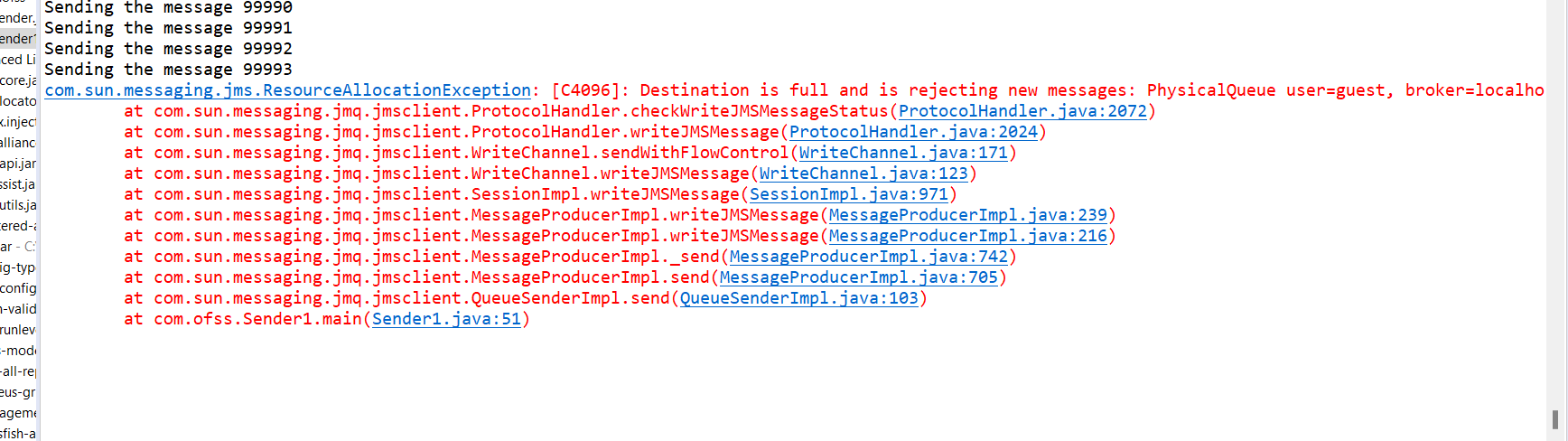
5 minutes

Program 3:

You convert the for loop into an endless for loop to simulate continuous streaming of data



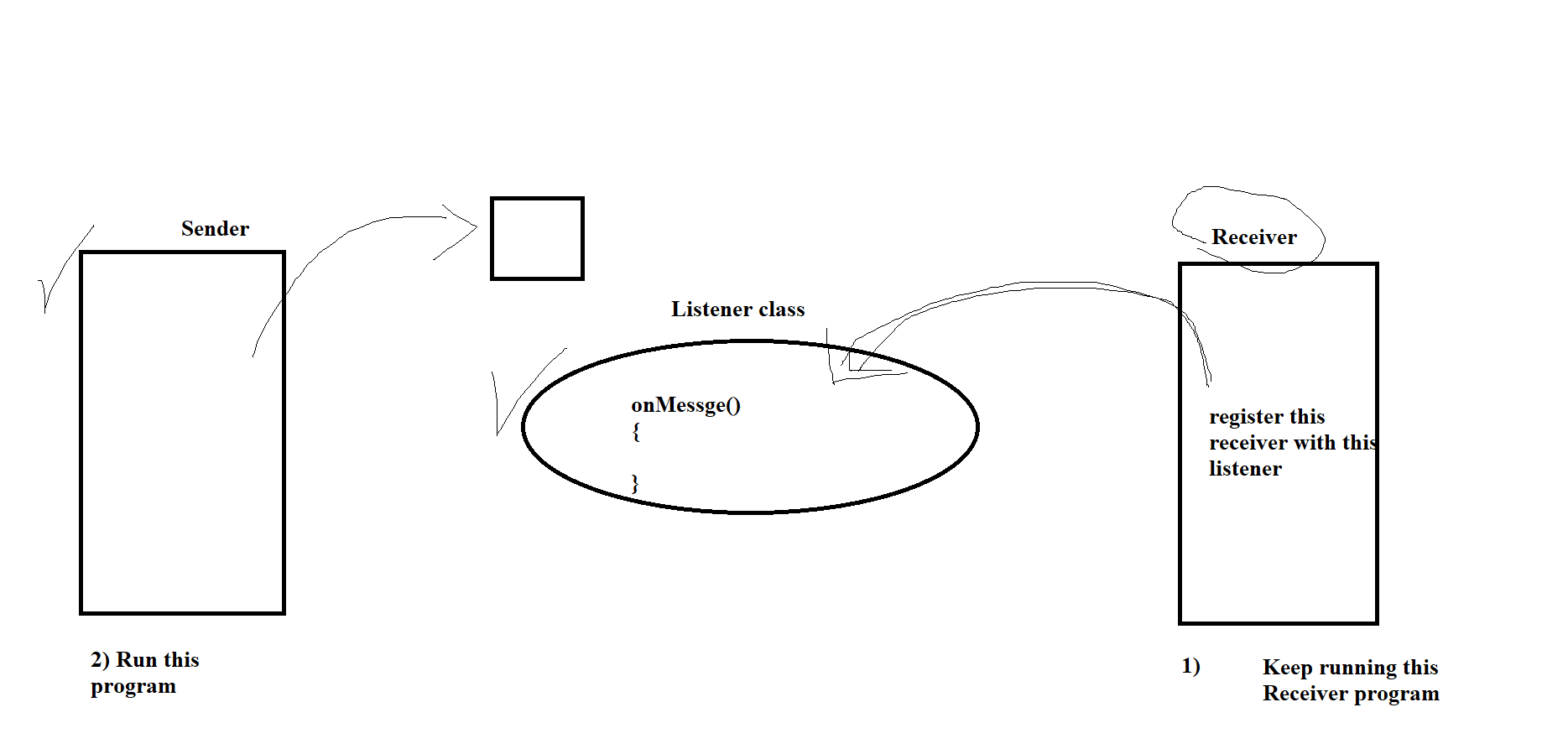
After inserting some amount of data, the program will throw run time exception



Currently, we have seen this program has successfully stored / sent the messages

Next, how to read the messages?

Let’s create another program “Listener.java”



Receiver.java

package com.ofss;

import java.util.Properties;

import javax.jms.Queue;

import javax.jms.QueueConnection;

import javax.jms.QueueConnectionFactory;

import javax.jms.QueueReceiver;

import javax.jms.QueueSender;

import javax.jms.QueueSession;

import javax.jms.TextMessage;

import javax.naming.InitialContext;

import com.sun.messaging.jmq.jmsserver.core.Session;

public class Receiver {

public static void main(String[] args) {

try {

Properties props=new Properties();

// The property names will be the same always

// Step1: obtaining the Initial Context object

// The property values will be different depends on jms implementor

props.setProperty("java.naming.factory.initial", "com.sun.enterprise.naming.SerialInitContextFactory");

props.setProperty("java.naming.factory.url.pkgs", "com.sun.enterprise.naming");

props.setProperty("java.naming.provider.url", "iiop://localhost:3700");// Inter ORB protocol of glassfish server

InitialContext ctx=new InitialContext(props);

// Step 2:

// since this method returns Object, we need to typecast it accordingly

// If you deal with Queue, then you need to typecast this object into QueryConnectionFactory object

QueueConnectionFactory f=(QueueConnectionFactory)ctx.lookup("jms/cf1");

QueueConnection con=f.createQueueConnection();

con.start();

// Step 3: Create a queue session from this connection

QueueSession ses=con.createQueueSession(false, Session.AUTO\_ACKNOWLEDGE);

// Step 4: Get the queue name using JNDI

Queue q=(Queue)ctx.lookup("jms/Jan2023-Queue");

// Step 5: Create a receiver object (because we are going to receive the msg)

QueueReceiver receiver=ses.createReceiver(q);

// Step 6: Create Listener object

Listener listener=new Listener(); // This is user defined class

// Step 7: Register this listener with receier

receiver.setMessageListener(listener);

System.out.println("Receiver is ready, waiting for messages.....");

while (true)

{

Thread.sleep(100000);

}

}

catch(Exception e)

{

e.printStackTrace();

}

}

}

Listener.java

package com.ofss;

import javax.jms.JMSException;

import javax.jms.Message;

import javax.jms.MessageListener;

import javax.jms.TextMessage;

// This is a listener program, you need to keep this program running

// so that as soon the msg arrives at the queue, this onMessage() method

// will be triggerd immeidately, so that we can consume the msg from the queue

public class Listener implements MessageListener{

@Override

public void onMessage(Message m) {

System.out.println("onMessage is trigged, that means a new msg. has arrived");

if (m instanceof TextMessage)

{

// If so, let's typecast this into TextMessage type

TextMessage msg=(TextMessage)m;

try {

System.out.println("Message received is "+msg.getText());

// you can do the data processsing here

} catch (JMSException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} // we need to call getText() method here

}

}

}

If you want to work on the messages you receive from the queue, then you write the steps/program inside onMessage() method or you create another method and pass the data from onMessage() to that new method()

Time: 3.55

Let’s meet at 4.10 pm

You can take 5 minutes to try this program

In real projects, you don’t need to create multiple connection factories and the queue connection

Just one connection factory and one queue connection is sufficient

For every new request, you need to create a separate session

In Queue based systems, one producer and there exists exactly one consumer

Like a OTP

In our last program, we have registered one receiver with one listener

Is it possible or okay to register one listener with multiple receivers?

Technical feasibility

Scenario

– Possible with Publisher

It is possible with Queue system

In JMS, it is possible for multiple receivers to register with the same listener

This allows multiple consumers to receive the process messages sent to the same destination (Queue) concurrently

There are 1000 messages

We want to process all 1000 messages

If we have only a single thread, it will take “1000 seconds”

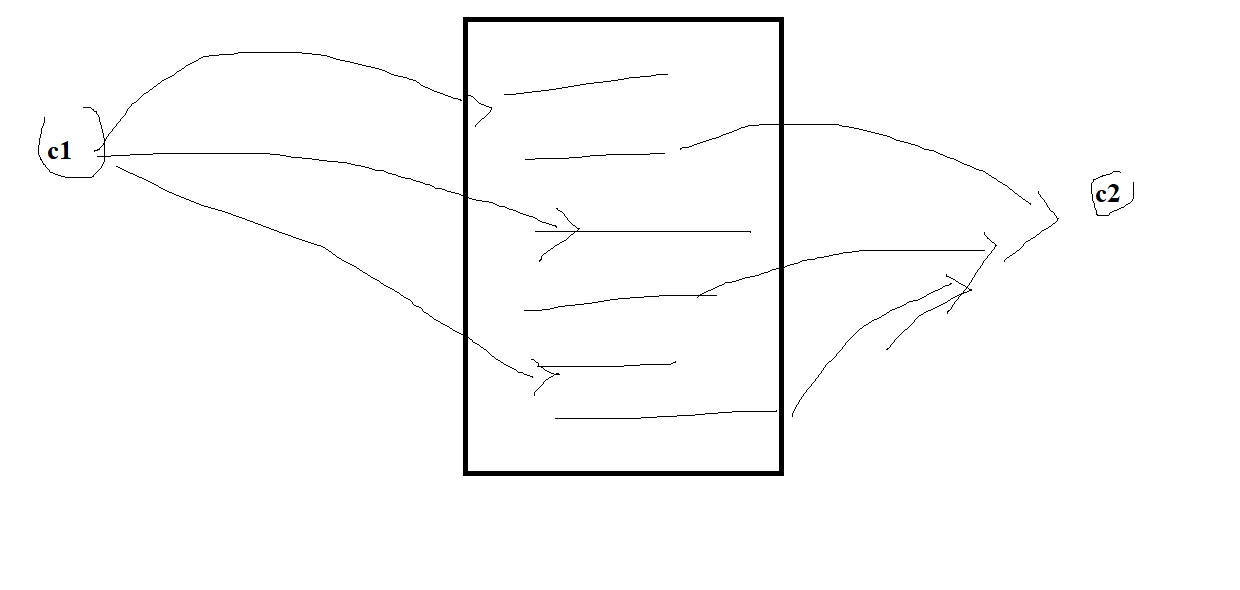
But, if we apply more workers/ threads, it will take lesser number of seconds

1 thread → 1000 ms

2 thread → 500 ms

4 threads → 250 ms

When multiple receivers are registered with the same listener, the JMS provider is responsible for distributing the messages equally to all the consumers and also ensures that the same message is not delivered to the multiple consumers



On the other hand, in case of a topic destination, multiple consumers can consume the same messages

**Program 2:**

Let’s see how to send Java objects as messages

Let’s define an Employee class with some properties

SenderObject.class:

package com.ofss;

import java.util.Properties;

import javax.jms.ObjectMessage;

import javax.jms.Queue;

import javax.jms.QueueConnection;

import javax.jms.QueueConnectionFactory;

import javax.jms.QueueSender;

import javax.jms.QueueSession;

import javax.jms.TextMessage;

import javax.naming.InitialContext;

import com.sun.messaging.jmq.jmsserver.core.Session;

public class SenderObject {

public static void main(String[] args) {

try {

Properties props=new Properties();

// The property names will be the same always

// Step1: obtaining the Initial Context object

// The property values will be different depends on jms implementor

props.setProperty("java.naming.factory.initial", "com.sun.enterprise.naming.SerialInitContextFactory");

props.setProperty("java.naming.factory.url.pkgs", "com.sun.enterprise.naming");

props.setProperty("java.naming.provider.url", "iiop://localhost:3700");// Inter ORB protocol of glassfish server

InitialContext ctx=new InitialContext(props);

// Step 2:

// since this method returns Object, we need to typecast it accordingly

// If you deal with Queue, then you need to typecast this object into QueryConnectionFactory object

QueueConnectionFactory f=(QueueConnectionFactory)ctx.lookup("jms/cf1");

QueueConnection con=f.createQueueConnection();

// Step 3: Create a queue session from this connection

QueueSession ses=con.createQueueSession(false, Session.AUTO\_ACKNOWLEDGE);

// Step 4: Get the queue name using JNDI

Queue q=(Queue)ctx.lookup("jms/Jan2023-Queue");

// Step 5: Create a sender object (because we are going to send the msg)

QueueSender sender=ses.createSender(q);

// Employee is user defined data type

// We must implement Serializable interface

Employee emp=new Employee("Guru", 38382.2);

// Step 6: Create ObjectMessage type

ObjectMessage objectMessage=ses.createObjectMessage();

objectMessage.setObject(emp);

objectMessage.setJMSTimestamp(System.currentTimeMillis());

sender.send(objectMessage);

System.out.println("Java object has been successfully sent....to the queue");

System.out.println("Message has been successfully produced and sent to the given queue");

}

catch(Exception e)

{

e.printStackTrace();

}

}

}

But, in Listener, you need to typecast accordingly

Modified Listener class

package com.ofss;

import javax.jms.JMSException;

import javax.jms.Message;

import javax.jms.MessageListener;

import javax.jms.ObjectMessage;

import javax.jms.TextMessage;

// This is a listener program, you need to keep this program running

// so that as soon the msg arrives at the queue, this onMessage() method

// will be triggerd immeidately, so that we can consume the msg from the queue

public class Listener implements MessageListener{

@Override

public void onMessage(Message m) {

System.out.println("onMessage is trigged, that means a new msg. has arrived");

if (m instanceof TextMessage)

{

System.out.println("This is a text msg");

// If so, let's typecast this into TextMessage type

TextMessage msg=(TextMessage)m;

try {

System.out.println("Message received is "+msg.getText());

// logic1

// logic2

// you can do the data processsing here

} catch (JMSException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} // we need to call getText() method here

}

else

{

System.out.println("it is not a text message ");

try {

Object obj=((ObjectMessage)m).getObject(); // still it is not Employee object

String className=obj.getClass().getName();

if (className.contains("Employee"))

{

Employee emp=(Employee)obj; // here, we are narrowing down the data type to Employee

System.out.println(emp); // toString will be called

}

} catch (JMSException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

}

Employee.java

package com.ofss;

import java.io.Serializable;

public class Employee implements Serializable {

String firstName;

double salary;

public Employee() {

super();

// TODO Auto-generated constructor stub

}

public Employee(String firstName, double salary) {

super();

this.firstName = firstName;

this.salary = salary;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public double getSalary() {

return salary;

}

public void setSalary(double salary) {

this.salary = salary;

}

@Override

public String toString() {

return "Employee [firstName=" + firstName + ", salary=" + salary + "]";

}

}

5 minutes

**Program 3:**

Let’s create a topic in glassfish

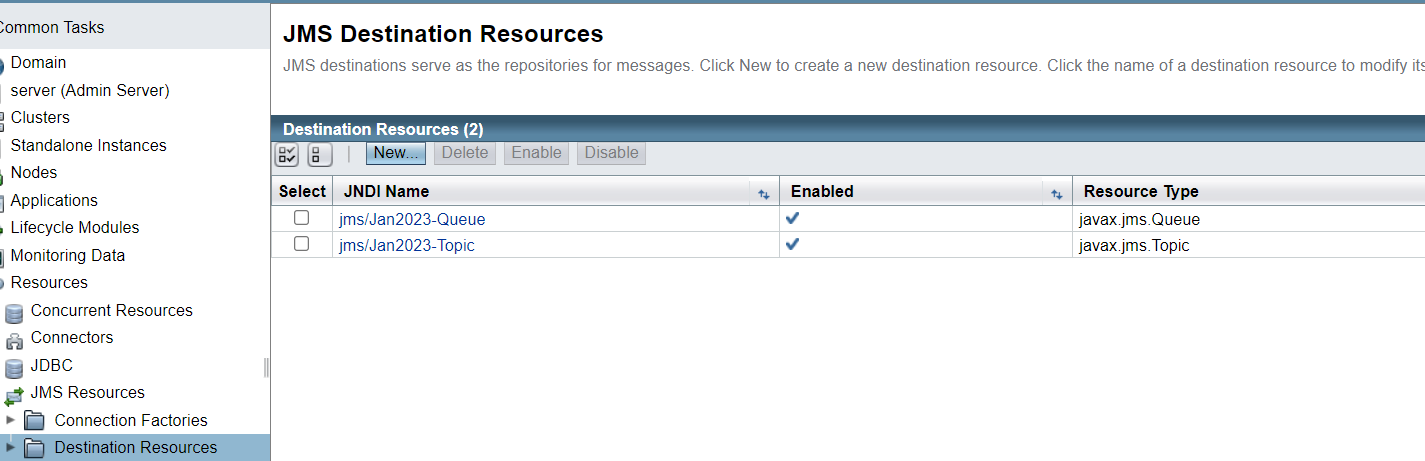
asadmin> create-jms-resource --restype javax.jms.Topic --property Name=PhysicalTopic jms/Jan2023-Topic

Administered object jms/Jan2023-Topic created.

Command create-jms-resource executed successfully.

asadmin>

Verify this in the console:



We can use the previously created connection factory for this program

The topic name:

jms/Jan2023-Topic

Take the existing sender & receiver programs

Make copies as Publisher & Subscriber programs

Replace all queue related classes by topic related class

Publisher.java

package com.ofss;

import java.util.Properties;

import javax.jms.Queue;

import javax.jms.QueueConnection;

import javax.jms.QueueConnectionFactory;

import javax.jms.QueueSender;

import javax.jms.QueueSession;

import javax.jms.TextMessage;

import javax.jms.Topic;

import javax.jms.TopicConnection;

import javax.jms.TopicConnectionFactory;

import javax.jms.TopicPublisher;

import javax.jms.TopicSession;

import javax.naming.InitialContext;

import com.sun.messaging.jmq.jmsserver.core.Session;

public class Publisher {

public static void main(String[] args) {

try {

Properties props=new Properties();

// The property names will be the same always

// Step1: obtaining the Initial Context object

// The property values will be different depends on jms implementor

props.setProperty("java.naming.factory.initial", "com.sun.enterprise.naming.SerialInitContextFactory");

props.setProperty("java.naming.factory.url.pkgs", "com.sun.enterprise.naming");

props.setProperty("java.naming.provider.url", "iiop://localhost:3700");// Inter ORB protocol of glassfish server

InitialContext ctx=new InitialContext(props);

// upto here, the same code

// Step 2:

// since this method returns Object, we need to typecast it accordingly

// If you deal with Topic, then you need to typecast this object into TopicConnectionFactory object

TopicConnectionFactory f=(TopicConnectionFactory)ctx.lookup("jms/cf1");

TopicConnection con=f.createTopicConnection();

con.start();

// Step 3: Create a queue session from this connection

TopicSession ses=con.createTopicSession(false, Session.AUTO\_ACKNOWLEDGE);

// Step 4: Get the queue name using JNDI

Topic t=(Topic)ctx.lookup("jms/Jan2023-Topic");

// Step 5: Create a pubilshere object

TopicPublisher publisher= ses.createPublisher(t);

publisher.setTimeToLive(10000); // 10 seconds

System.out.println("Publisher destination is "+publisher.getDestination());

System.out.println("TTL ...."+publisher.getTimeToLive());

System.out.println("Topic name "+publisher.getTopic());

// Create the TextMessage now

TextMessage msg=ses.createTextMessage();

msg.setText("This is the first msg to the topic");

publisher.publish(msg);

System.out.println("The msg has been successfully published to the topic");

publisher.close();

ses.close();

con.close();

}

catch(Exception e)

{

e.printStackTrace();

}

}

}

Subscriber.java

package com.ofss;

import java.util.Properties;

import javax.jms.Queue;

import javax.jms.QueueConnection;

import javax.jms.QueueConnectionFactory;

import javax.jms.QueueReceiver;

import javax.jms.QueueSender;

import javax.jms.QueueSession;

import javax.jms.TextMessage;

import javax.jms.Topic;

import javax.jms.TopicConnection;

import javax.jms.TopicConnectionFactory;

import javax.jms.TopicSession;

import javax.jms.TopicSubscriber;

import javax.naming.InitialContext;

import com.sun.messaging.jmq.jmsserver.core.Session;

public class Subscriber {

public static void main(String[] args) {

try {

Properties props=new Properties();

// The property names will be the same always

// Step1: obtaining the Initial Context object

// The property values will be different depends on jms implementor

props.setProperty("java.naming.factory.initial", "com.sun.enterprise.naming.SerialInitContextFactory");

props.setProperty("java.naming.factory.url.pkgs", "com.sun.enterprise.naming");

props.setProperty("java.naming.provider.url", "iiop://localhost:3700");// Inter ORB protocol of glassfish server

InitialContext ctx=new InitialContext(props);

TopicConnectionFactory f=(TopicConnectionFactory)ctx.lookup("jms/cf1");

TopicConnection con=f.createTopicConnection();

con.start();

// Step 3: Create a topic session from this connection

TopicSession ses=con.createTopicSession(false, Session.AUTO\_ACKNOWLEDGE);

// Step 4: Get the topic name using JNDI

Topic t=(Topic)ctx.lookup("jms/Jan2023-Topic");

// Create TopicSubscriper object

TopicSubscriber receiver= ses.createSubscriber(t);

// Step 6: Create Listener object

Listener listener=new Listener(); // This is user defined class

// Step 7: Register this listener with receier

receiver.setMessageListener(listener);

System.out.println("Subscriber is ready, waiting for messages.....");

while (true)

{

}

}

catch(Exception e)

{

e.printStackTrace();

}

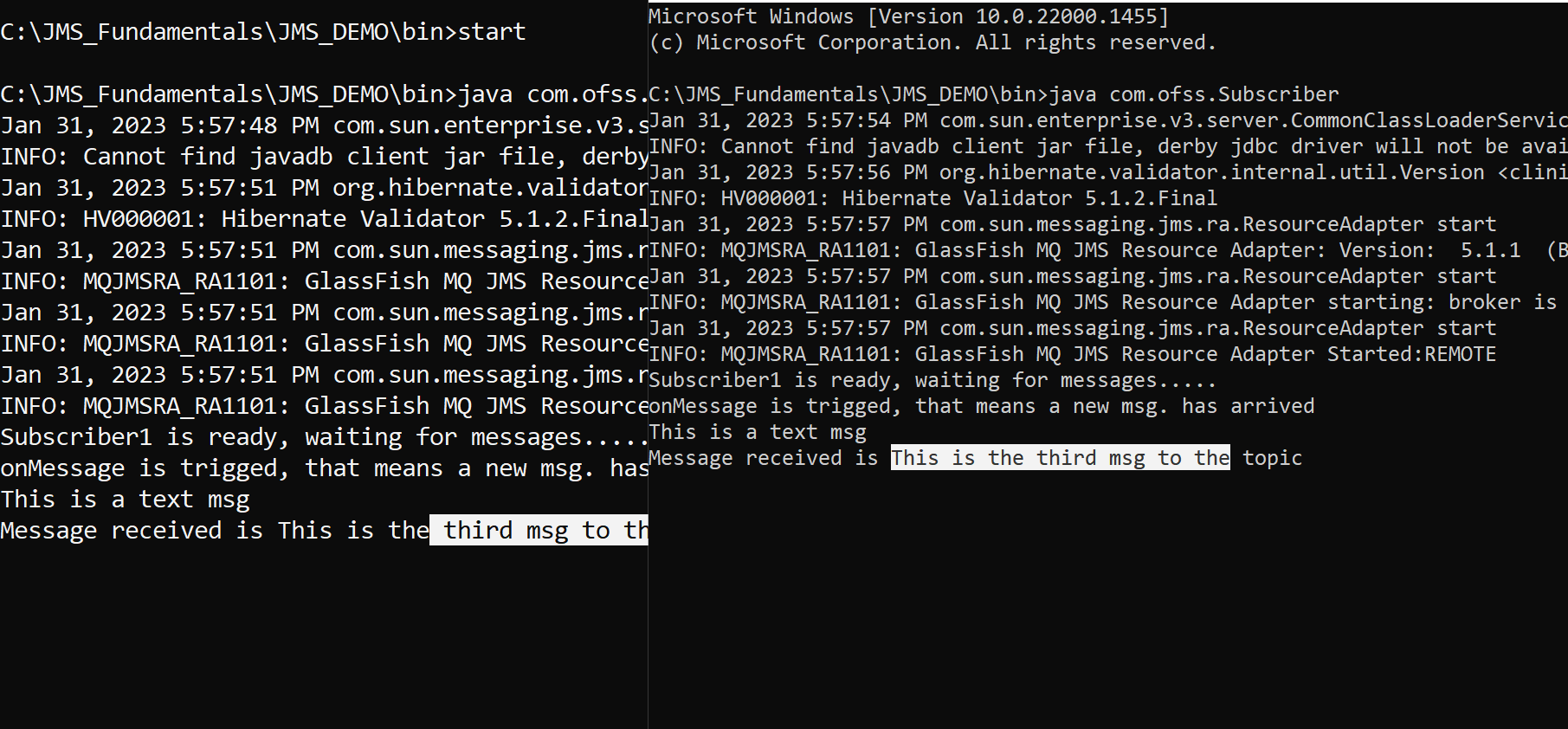
}

}

–

Run Publisher program from Eclipse

Run the Subscriber program multiple times in multiple windows simulating multiple consumers for the same topic



A real world example is a web application that is used to place an order for a particular customer

As part of placing this order (and storing it in a database), you may wish to do number of additional tasks:

1. Store the order in some sort of 3rd party back-end systems (SAP)
2. Send an email to the customer to inform them that their order has been successfully placed

Customers don’t like to wait for more than 2 seconds

If you make the customers to wait till the above 2 tasks are completed, that would become the synchronous model

**As soon as the customer placed the order, you can send the msg to JMS server and return control immediately to the customer with an URL (containing the orderid), asking them to check the status later**

Now, on the back system, onMessage() will pickup the order information from this queue/topic, then start proceeding for further processing, till time time, the customer don’t need to wait

Total Asynchronous model

The messages in the queue are meant for one customer

P2P

1 sender and 1 receiver

You can also set the retnetion period of the message in case of TOPIC

This is a configurable element that you can specify in the admin console, and also depends on the JMS provider

JBos and WebLogic & Glassfish can deal with it in a different way

[gurumurthy.ramamurthy@gmail.com](mailto:gurumurthy.ramamurthy@gmail.com)