Name: Omkar Deshpande

Roll No.: 43212

Batch: Q-10

Divisions: BE 10

Assignment 6

Title: To develop any distributed application using Messaging System in Publisher-Subscriber paradigm

Tools: Java Programming Environment, JDK 8, Eclipse IDE, Apache ActiveMQ 4.1.1, JMS

Theory:

Enterprise Messaging System:

EMS, or the messaging system, defines system standards for organizations so they can define their enterprise application messaging process with a semantically precise messaging structure.

EMS encourages you to define a loosely coupled application architecture in order to define an industry-accepted message structure; this is to ensure that published messages would be persistently consumed by subscribers. Common formats, such as XML or JSON, are used to do this. EMS recommends these messaging protocols: DDS, MSMQ, AMQP, or SOAP web services. Systems designed with EMS are termed Message-Oriented Middleware (MOM). An asynchronous communication is used while messaging in EMS.

Java Messaging Service

Java's implementation of an EMS in the Application Programming Interface (API) format is known as JMS.

JMS allows distributed Java applications to communicate with applications developed in any other technology that understands messaging through asynchronous messages. JMS applications contain a provider, clients, messages, and administered objects.

JMS provides a standard, portable way for Java programs to send/receive messages through a MOM product. Any application written in JMS can be executed on any MOM that implements the JMS API standards. The JMS API is specified as a set of interfaces as part of the Java API. Hence, all the products that intend to provide JMS behavior will

have to deliver the provider to implement JMS-defined interfaces. With programming patterns that allow a program to interface, you should be able to construct a Java application in line with the JMS standards by defining the messaging programs with client applications to exchange information through JMS messaging

The publish/subscribe messaging paradigm:

The publish/subscribe messaging paradigm is built with the concept of a topic, which behaves like an announcement board. Consumers subscribe to receiving messages that belong to a topic, and publishers report messages to a topic. The JMS provider retains the responsibility for distributing the messages that it receives from multiple publishers to many other subscribers based on the topic they subscribe to. A subscriber receives messages that it subscribes to based on the rules it defines and the messages that are published after the subscription is registered; they do not receive any messages that are already published, as shown in the following diagram:

JMS interfaces

JMS defines a set of high-level interfaces that encapsulate several messaging concepts. These high-level interfaces are further extended for the Point-To-Point and publish/subscribe messaging domains:

ConnectionFactory: This is an administered object with the ability to create a connection. Connection: This is an active connection handle to the provider.

Destination: This is an administered object that encapsulates the identity of a message destination where messages are sent to/received from.

Session: This is a single-threaded context for sending/receiving messages. To ensure a simple session-based transaction, concurrent access to a message by multiple threads is restricted. We can use multiple sessions for a multithreaded application.

MessageProducer: This is used to send messages.

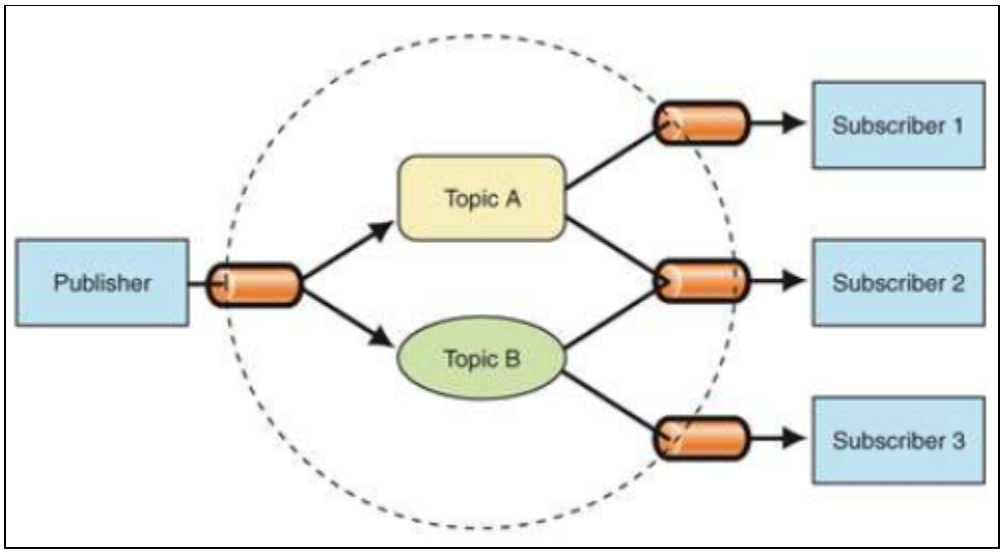
MessageConsumer: This is used to receive messages.

Designing the solution:

In the ‘Publisher-Subscriber’ pattern, senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers.

For example, consider that a publisher publishes news (topics) related to politics and sports; they publish to the Messaging Broker, as shown in the following diagram. While Subscriber 1 receives news related to politics and Subscriber 3 receives news related to sports, Subscriber 2 will receive both political and sports news as it subscribed to the common topics.

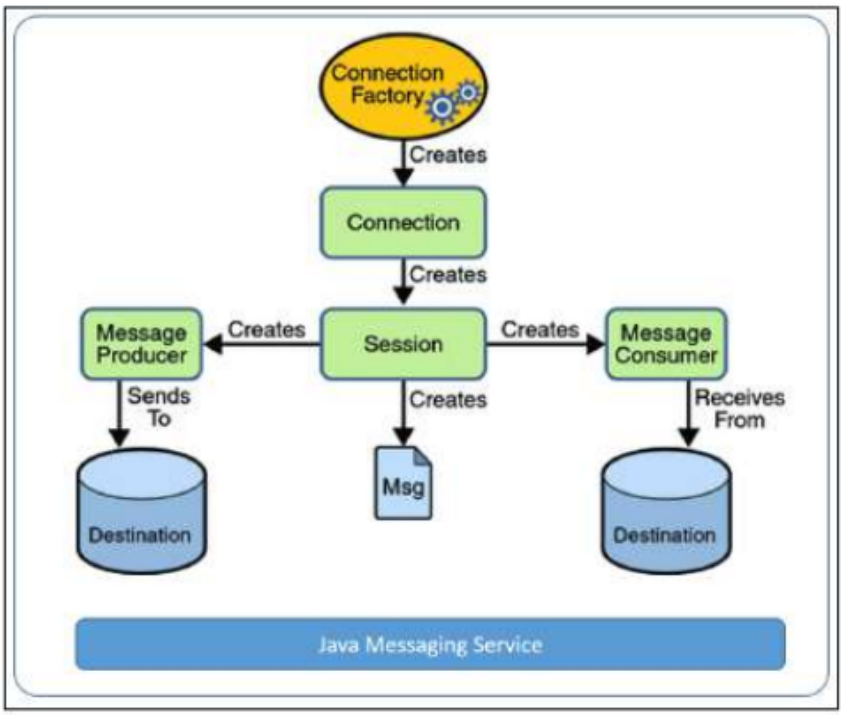
In designing our solution, we have to create one publisher and subscriber wherein the publisher creates the topic.



The Publisher/Subscriber pattern is mostly implemented in an asynchronous way (using message queue).

Publishers and subscribers have a timing dependency. A client that subscribes to a topic can consume only messages published after the client has created a subscription, and the subscriber must continue to be active in order for it to consume messages.
JMS is a Java API that allows applications to create, send, receive, and read messages. The JMS API enables communication that is loosely coupled, asynchronous and reliable.
To use JMS, we need to have a JMS provider that can manage the sessions, queues, and topics. Some examples of known JMS providers are Apache ActiveMQ, WebSphere MQ from IBM or SonicMQ from Aurea Software. Starting from Java EE version 1.4, a JMS provider has to be contained in all Java EE application servers.

A JMS provider is a messaging server that supports the creation of connections (multithreaded virtual links to the provider) and sessions (single-threaded contexts for producing and consuming messages). A JMS client is a Java program that either produces or consumes messages.
JMS messages are objects that communicate information between JMS clients and are composed of a header, some optional properties, and an optional body.



Administered objects are preconfigured JMS objects, such as a connection factory (the object a client uses to create a connection to a provider) and a destination (the object a client uses to specify a target for its messages).
JMS applications are usually developed in either the publish/subscribe or Point-To-Point paradigm.

Conclusion :
In this assignment we studied the publisher subscriber model which was implemented using JMS.