Name: **Datta Tandale**, Roll no: **B1851002**, PRN: **72000298C**

ASSIGNMENT NO. 6

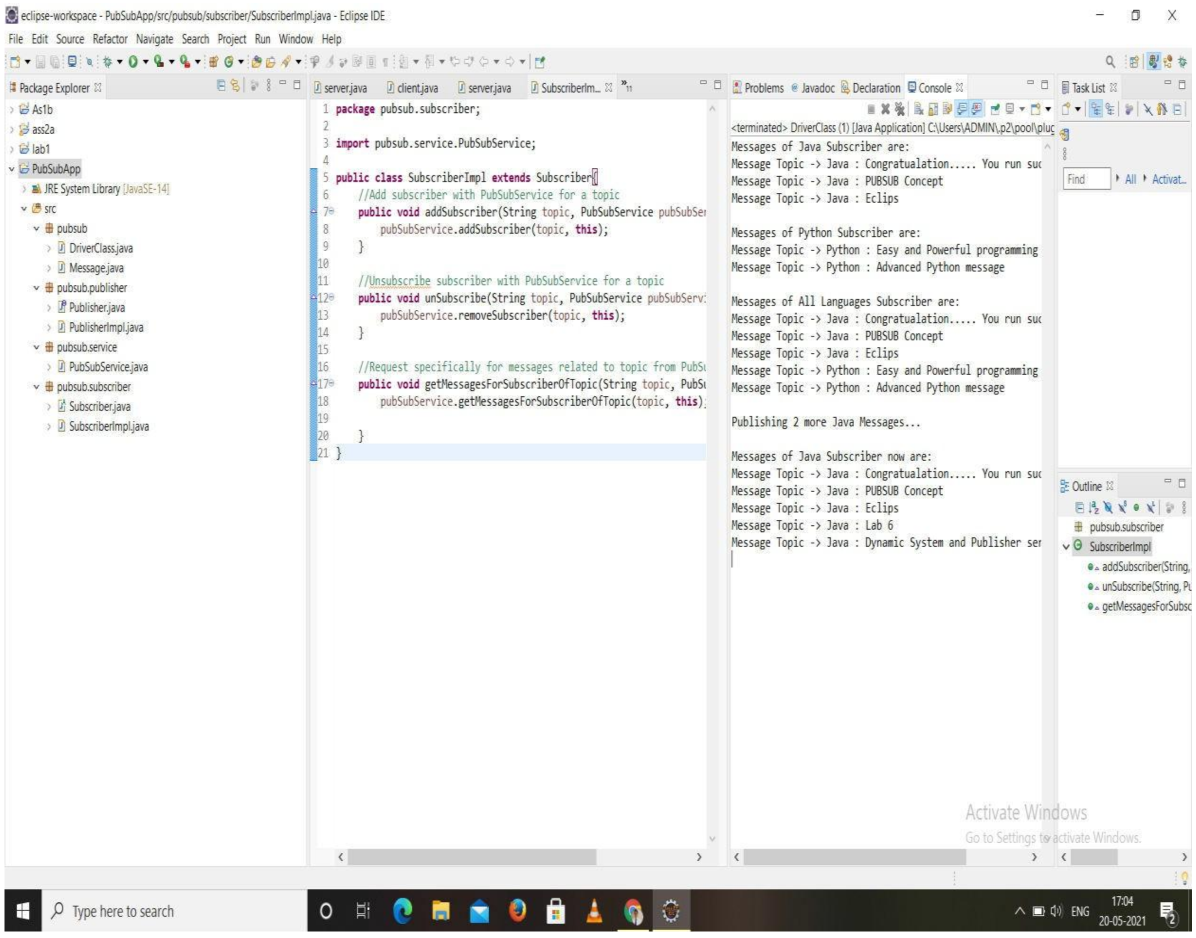
Aim/objective:

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

Tools / Environment:

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

OUTPUT

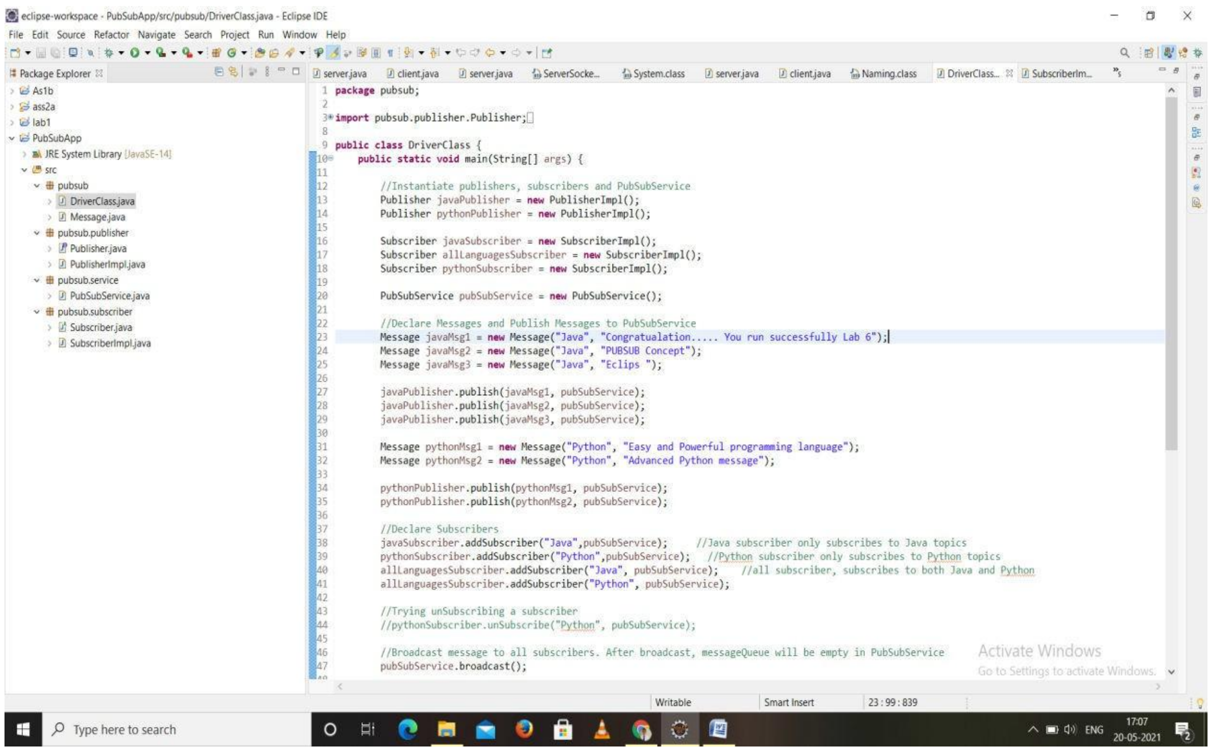


```java
package pubsub.subscriber;

import pubsub.service.PubSubService;

public class SubscriberImpl extends Subscriber{
    //Add subscriber with PubSubService for a topic
    public void addSubscriber(String topic, PubSubService pubSubSer
        pubSubService.addSubscriber(topic, this);
    }

    //Unsubscribe subscriber with PubSubService for a topic
    public void unSubscribe(String topic, PubSubService pubSubServi
        pubSubService.removeSubscriber(topic, this);
    }

    //Request specifically for messages related to topic from PubSu
    public void getMessagesForSubscriberOfTopic(String topic, PubSu
        pubSubService.getMessagesForSubscriberOfTopic(topic, this);

    }
}
```

```
<terminated> DriverClass (1) [Java Application] C:\Users\ADMIN\.p2\pool\plug
Messages of Java Subscriber are:
Message Topic -> Java : Congratualation..... You run suc
Message Topic -> Java : PUBSUB Concept
Message Topic -> Java : Eclips

Messages of Python Subscriber are:
Message Topic -> Python : Easy and Powerful programming
Message Topic -> Python : Advanced Python message

Messages of All Languages Subscriber are:
Message Topic -> Java : Congratualation..... You run suc
Message Topic -> Java : PUBSUB Concept
Message Topic -> Java : Eclips
Message Topic -> Python : Easy and Powerful programming
Message Topic -> Python : Advanced Python message

Publishing 2 more Java Messages...

Messages of Java Subscriber now are:
Message Topic -> Java : Congratualation..... You run suc
Message Topic -> Java : PUBSUB Concept
Message Topic -> Java : Eclips
Message Topic -> Java : Lab 6
Message Topic -> Java : Dynamic System and Publisher ser
```

Name: **Datta Tandale**, Roll no: **B1851002**, PRN: **72000298C**

//Driver



```
package pubsub;

import pubsub.publisher.Publisher;

public class DriverClass {
    public static void main(String[] args) {

        //Instantiate publishers, subscribers and PubSubService
        Publisher javaPublisher = new PublisherImpl();
        Publisher pythonPublisher = new PublisherImpl();

        Subscriber javaSubscriber = new SubscriberImpl();
        Subscriber allLanguagesSubscriber = new SubscriberImpl();
        Subscriber pythonSubscriber = new SubscriberImpl();

        PubSubService pubSubService = new PubSubService();

        //Declare Messages and Publish Messages to PubSubService
        Message javaMsg1 = new Message("Java", "Congratualation..... You run successfully Lab 6");
        Message javaMsg2 = new Message("Java", "PUBSUB Concept");
        Message javaMsg3 = new Message("Java", "Eclips ");

        javaPublisher.publish(javaMsg1, pubSubService);
        javaPublisher.publish(javaMsg2, pubSubService);
        javaPublisher.publish(javaMsg3, pubSubService);

        Message pythonMsg1 = new Message("Python", "Easy and Powerful programming language");
        Message pythonMsg2 = new Message("Python", "Advanced Python message");

        pythonPublisher.publish(pythonMsg1, pubSubService);
        pythonPublisher.publish(pythonMsg2, pubSubService);

        //Declare Subscribers
        javaSubscriber.addSubscriber("Java",pubSubService);     //Java subscriber only subscribes to Java topics
        pythonSubscriber.addSubscriber("Python",pubSubService);   //Python subscriber only subscribes to Python topics
        allLanguagesSubscriber.addSubscriber("Java", pubSubService);    //all subscriber, subscribes to both Java and Python
        allLanguagesSubscriber.addSubscriber("Python", pubSubService);

        //Trying unSubscribing a subscriber
        //pythonSubscriber.unSubscribe("Python", pubSubService);

        //Broadcast message to all subscribers. After broadcast, messageQueue will be empty in PubSubService
        pubSubService.broadcast();
```

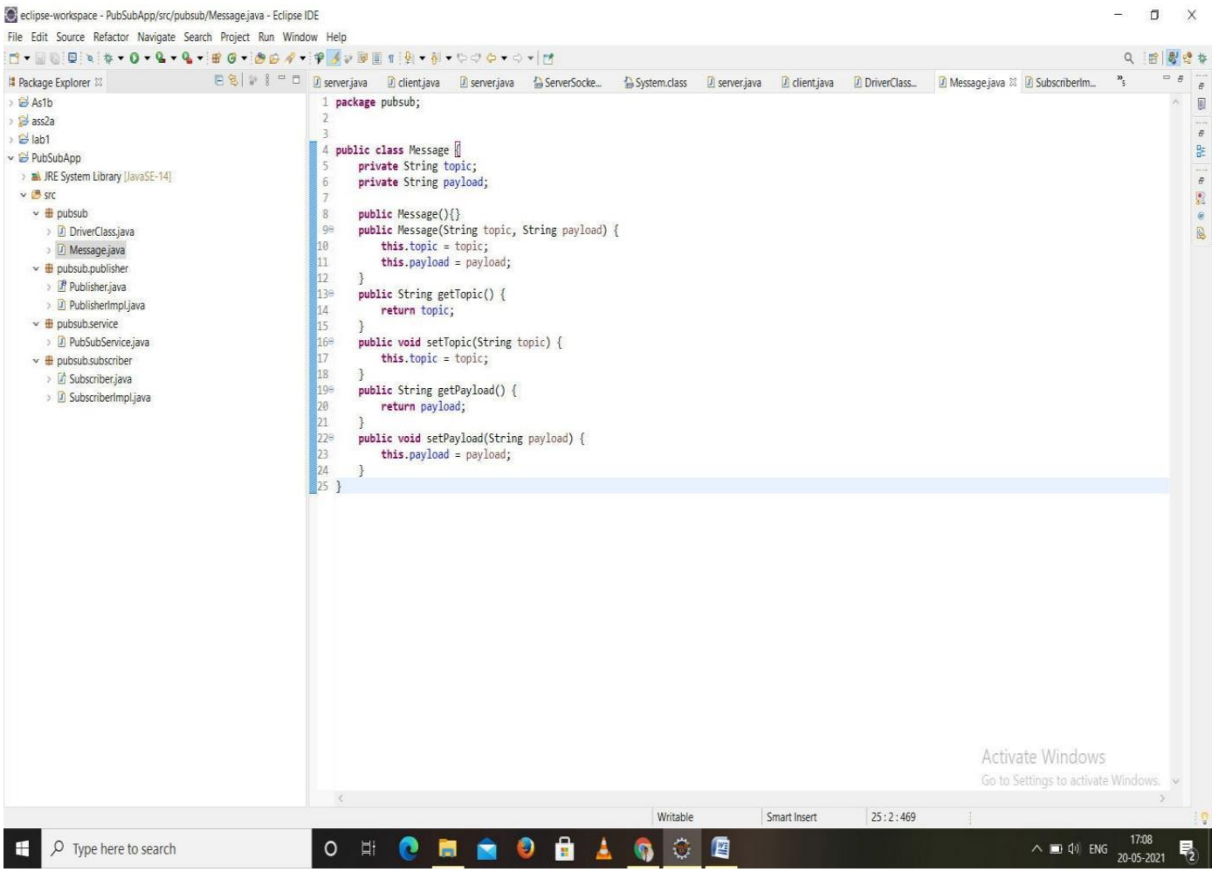
```
package pubsub;

import pubsub.publisher.Publisher;
import pubsub.publisher.PublisherImpl;
import pubsub.service.PubSubService;
import pubsub.subscriber.Subscriber;
import pubsub.subscriber.SubscriberImpl;

public class DriverClass {
        public static void main(String[] args) {

                //Instantiate publishers, subscribers and PubSubService
                Publisher javaPublisher = new PublisherImpl();
                Publisher pythonPublisher = new PublisherImpl();

                Subscriber javaSubscriber = new SubscriberImpl();
                Subscriber allLanguagesSubscriber = new SubscriberImpl();
                Subscriber pythonSubscriber = new SubscriberImpl();

                PubSubService pubSubService = new PubSubService();

                //Declare Messages and Publish Messages to PubSubService
```

```
		Message javaMsg1 = new Message("Java", "Congratualation..... You run successfully
Lab 6");
		Message javaMsg2 = new Message("Java", "PUBSUB Concept");
		Message javaMsg3 = new Message("Java", "Eclips ");

		javaPublisher.publish(javaMsg1,  pubSubService);
		javaPublisher.publish(javaMsg2,  pubSubService);
		javaPublisher.publish(javaMsg3, pubSubService);

		Message pythonMsg1 = new Message("Python", "Easy and Powerful programming
language");
		Message pythonMsg2 = new Message("Python", "Advanced Python message");

		pythonPublisher.publish(pythonMsg1, pubSubService);
		pythonPublisher.publish(pythonMsg2, pubSubService);

		//Declare Subscribers
		javaSubscriber.addSubscriber("Java",pubSubService);			//Java subscriber
only subscribes to Java topics
		pythonSubscriber.addSubscriber("Python",pubSubService);   //Python subscriber only
subscribes to Python topics
		allLanguagesSubscriber.addSubscriber("Java", pubSubService);  //all subscriber,
subscribes to both Java and Python
		allLanguagesSubscriber.addSubscriber("Python", pubSubService);

		//Trying unSubscribing a subscriber
		//pythonSubscriber.unSubscribe("Python", pubSubService);

		//Broadcast message to all subscribers. After broadcast, messageQueue will be empty
in PubSubService
		pubSubService.broadcast();

		//Print messages of each subscriber to see which messages they got
		System.out.println("Messages of Java Subscriber are: ");
		javaSubscriber.printMessages();

		System.out.println("\nMessages of Python Subscriber are: ");
		pythonSubscriber.printMessages();

		System.out.println("\nMessages of All Languages Subscriber are: ");
		allLanguagesSubscriber.printMessages();

		//After broadcast the messagesQueue will be empty, so publishing new messages to
server
		System.out.println("\nPublishing 2 more Java Messages...");
		Message javaMsg4 = new Message("Java", "Lab 6");
		Message javaMsg5 = new Message("Java", "Dynamic System and Publisher
services");

		javaPublisher.publish(javaMsg4, pubSubService);
		javaPublisher.publish(javaMsg5, pubSubService);

		javaSubscriber.getMessagesForSubscriberOfTopic("Java", pubSubService);
		System.out.println("\nMessages of Java Subscriber now are: ");
		javaSubscriber.printMessages();
```

Name: **Datta Tandale**, Roll no: **B1851002**, PRN: **72000298C**

```
        }
}
```

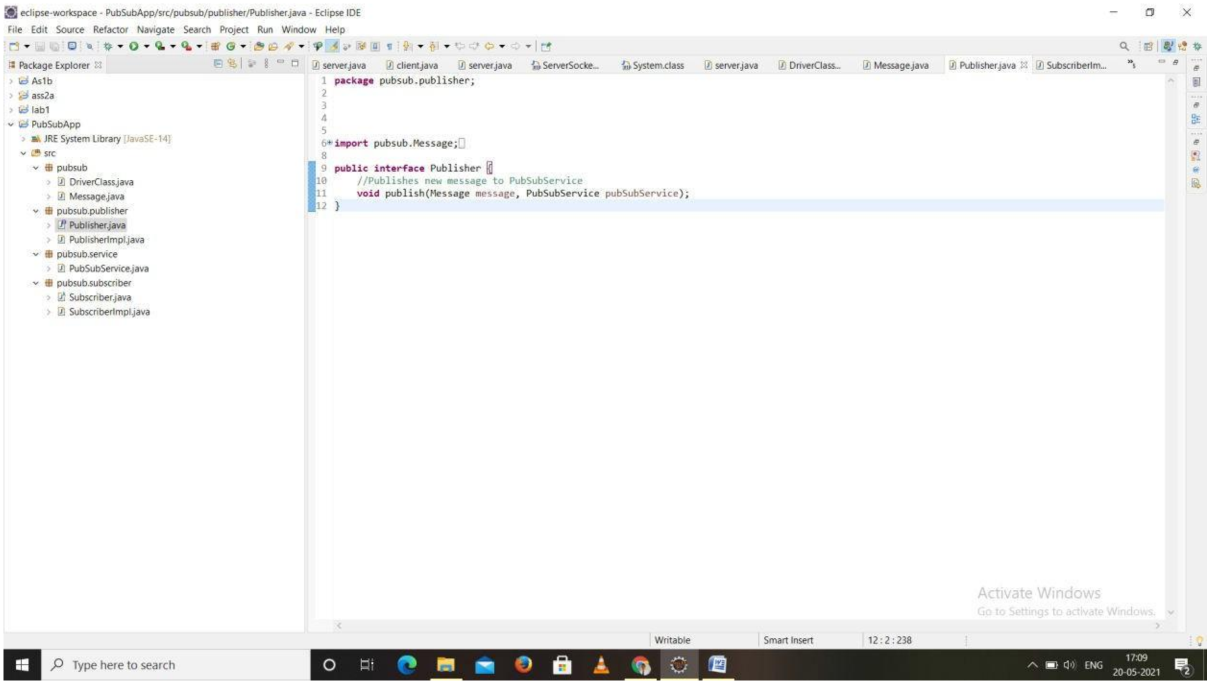
//class



```java
package pubsub;


public class Message {
    private String topic;
    private String payload;

    public Message(){}
    public Message(String topic, String payload) {
        this.topic = topic;
        this.payload = payload;
    }
    public String getTopic() {
        return topic;
    }
    public void setTopic(String topic) {
        this.topic = topic;
    }
    public String getPayload() {
        return payload;
    }
    public void setPayload(String payload) {
        this.payload = payload;
    }
}
```

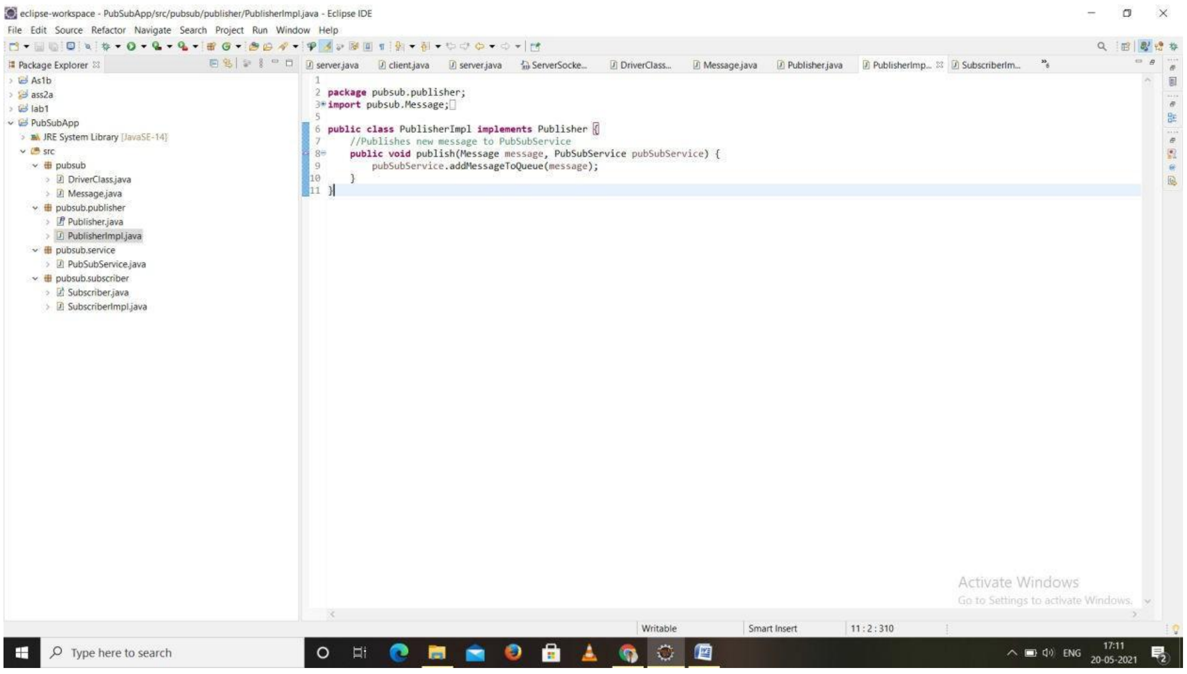
Name: **Datta Tandale**, Roll no: **B1851002**, PRN: **72000298C**

//Publisher



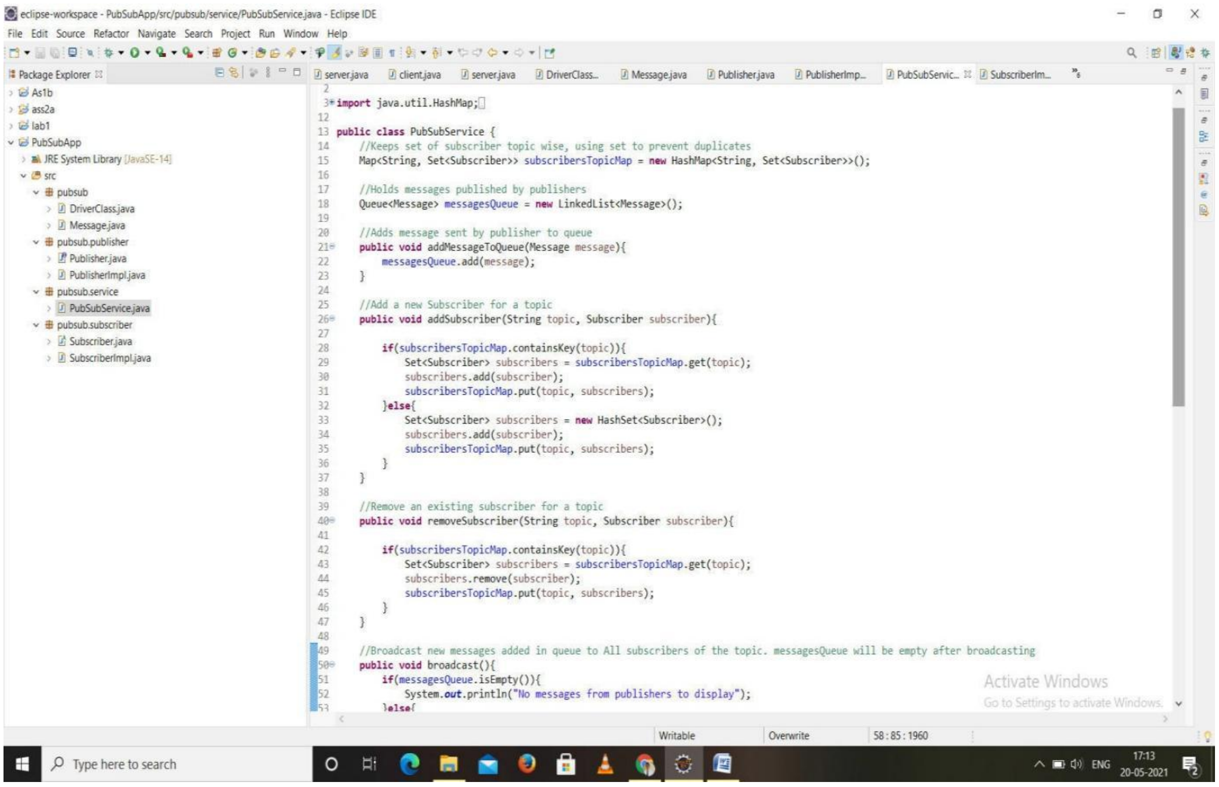
```java
package pubsub.publisher;




import pubsub.Message;

public interface Publisher {
    //Publishes new message to PubSubService
    void publish(Message message, PubSubService pubSubService);
}
```

//Publisher impl



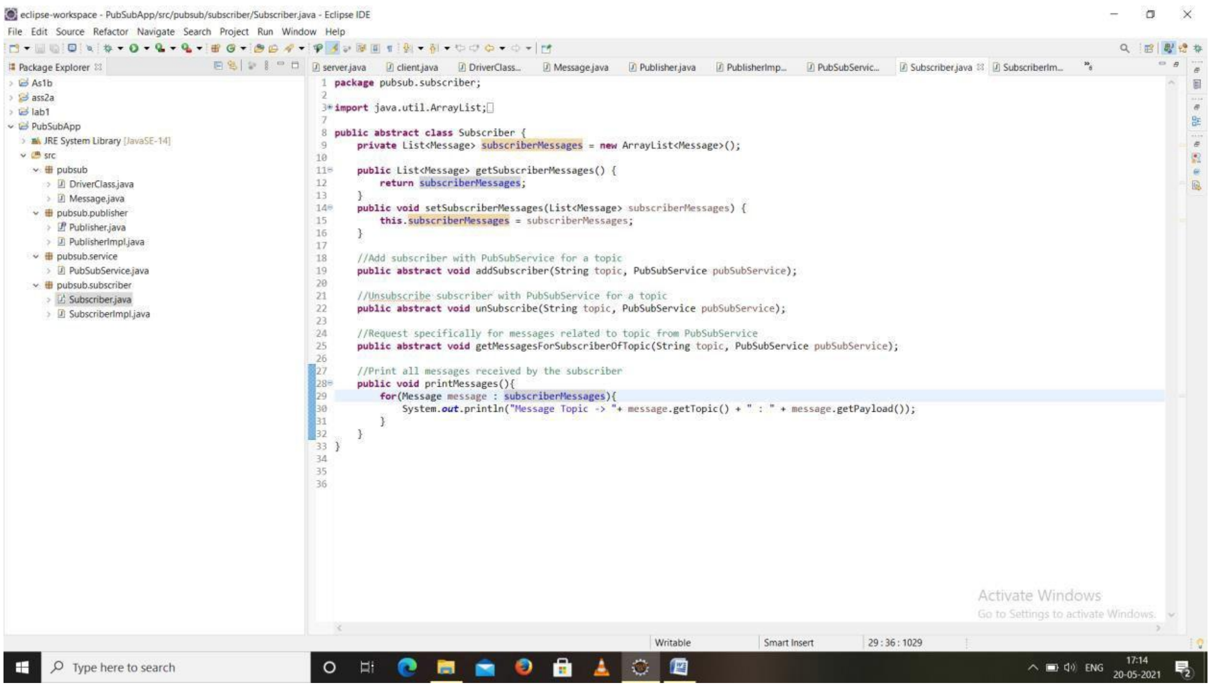
```java

package pubsub.publisher;
import pubsub.Message;

public class PublisherImpl implements Publisher {
    //Publishes new message to PubSubService
    public void publish(Message message, PubSubService pubSubService) {
        pubSubService.addMessageToQueue(message);
    }
}
```

Name: **Datta Tandale**, Roll no: **B1851002**, PRN: **72000298C**



```java

import java.util.HashMap;

public class PubSubService {
    //Keeps set of subscriber topic wise, using set to prevent duplicates
    Map<String, Set<Subscriber>> subscribersTopicMap = new HashMap<String, Set<Subscriber>>();

    //Holds messages published by publishers
    Queue<Message> messagesQueue = new LinkedList<Message>();

    //Adds message sent by publisher to queue
    public void addMessageToQueue(Message message){
        messagesQueue.add(message);
    }

    //Add a new Subscriber for a topic
    public void addSubscriber(String topic, Subscriber subscriber){

        if(subscribersTopicMap.containsKey(topic)){
            Set<Subscriber> subscribers = subscribersTopicMap.get(topic);
            subscribers.add(subscriber);
            subscribersTopicMap.put(topic, subscribers);
        }else{
            Set<Subscriber> subscribers = new HashSet<Subscriber>();
            subscribers.add(subscriber);
            subscribersTopicMap.put(topic, subscribers);
        }
    }

    //Remove an existing subscriber for a topic
    public void removeSubscriber(String topic, Subscriber subscriber){

        if(subscribersTopicMap.containsKey(topic)){
            Set<Subscriber> subscribers = subscribersTopicMap.get(topic);
            subscribers.remove(subscriber);
            subscribersTopicMap.put(topic, subscribers);
        }
    }

    //Broadcast new messages added in queue to All subscribers of the topic. messagesQueue will be empty after broadcasting
    public void broadcast(){
        if(messagesQueue.isEmpty()){
            System.out.println("No messages from publishers to display");
        }else{
```

//Suciber



```java
package pubsub.subscriber;

import java.util.ArrayList;

public abstract class Subscriber {
    private List<Message> subscriberMessages = new ArrayList<Message>();

    public List<Message> getSubscriberMessages() {
        return subscriberMessages;
    }
    public void setSubscriberMessages(List<Message> subscriberMessages) {
        this.subscriberMessages = subscriberMessages;
    }

    //Add subscriber with PubSubService for a topic
    public abstract void addSubscriber(String topic, PubSubService pubSubService);

    //Unsubscribe subscriber with PubSubService for a topic
    public abstract void unSubscribe(String topic, PubSubService pubSubService);

    //Request specifically for messages related to topic from PubSubService
    public abstract void getMessagesForSubscriberOfTopic(String topic, PubSubService pubSubService);

    //Print all messages received by the subscriber
    public void printMessages(){
        for(Message message : subscriberMessages){
            System.out.println("Message Topic -> "+ message.getTopic() + " : " + message.getPayload());
        }
    }
}



```

Related Theory:

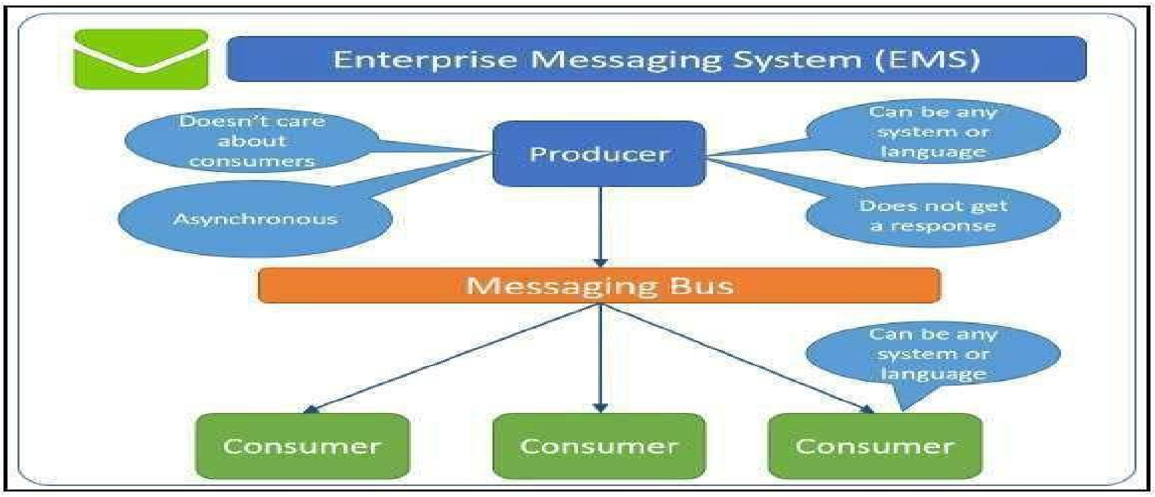
Large distributed systems are often overwhelmed with complications caused by heterogeneity and interoperability. Heterogeneity issues may arise due to the use of different programming languages, hardware platforms, operating systems, and data representations. Interoperability denotes the ability of heterogeneous systems to communicate meaningfully and exchange data or services. With the introduction of middleware, heterogeneity can be alleviated and interoperability can be achieved.

Middleware is a layer of software between the distributed application and the operating system and consists of a set of standard interfaces that help the application use networked resources and services.

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

.


Enterprise Messaging System (EMS)
Doesn't care about consumers
Asynchronous
Producer
Can be any system or language
Does not get a response
Messaging Bus
Can be any system or language
Consumer
Consumer
Consumer

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 administrated objects.

JMS providing 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

Conclusion:

This assignment includes study of Publish-Subscribe model of Communication which is implemented using JMS and Apache ActiveMQ. The topic based filtering requires the messages to be broadcasted into logical channels, the subscribers only receives messages from logic channels they are subscribed.