FILS
Course: Software Architectures

Lab. #8. JMS Homework

We are developing an integrated system for emergency healthcare. It should perform various scenarios like the following:
Scenario: Suppose somebody has a serious health problem and is taken by an emergency unit (Salvare). While the emergency unit is on its way to the hospital with the patient, the hospital must be notified of the situation using the EmergencyUnit application that is installed in the emergency unit. Also FamilyDoctor, the application of the patient's GP, should be notified. When the patient enters the hospital the doctors may need additional data so they send a request for the medical history of the patient to PatientRegistry, a service-oriented (not web service-oriented!) application that accesses a repository (database) containing medical information about all the citizens.
Let consider the following interaction of components aimed to carry out the above scenario :

1. EmergencyUnit sends a message (Message1) to a topic (jms/NewPatientTopic). The message should contain the name of the patient and his/her condition.
2. HospitalGateway is a MessageDrivenBean that listens to jms/NewPatientTopic. When a message (Message1) is received on jms/NewPatientTopic, it sends another message (Message2) to a queue jms/InquireQueue containg the name of the patient. Since it`s a request-reply scenario the returnnaddress must also be set in the message. It listens for the reply to the inquiry on the return address set in the message, jms/ResponseQueue.
3. FamilyDoctor is a MessageDrivenBean that listens to jms/NewPatientTopic. Since it is not always online it must be a durable subscriber.
4. PatientRegistry is a MessageDrivenBean that listens to jms/InquireQueue. It receives a message (Message2), looks up in the database and responds with a message (Message3) to the return address specified in the received message. The message should contain the medical history of the patient and the correlation id.

Hints 1. You first have to create the topics and queues. Two possibilities of software IDEs (container provision):

a) in Eclipse GlassFish (Services->GlassFish->Start->View Admin Console->Resources->JMS) or https://docs.oracle.com/javaee/6/tutorial/doc/bncfa.html or https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/JMSProducer/JMSProducer.html or https://jstobigdata.com/jms/install-and-setup-glassfish-for-jms/
b) in Spring project using Spring Initializr as is described in the followings: (https://start.spring.io/, https://spring.io/guides/gs/messaging-jms/, tutorials: https://javacodehouse.com/blog/spring-initializr/ or https://www.javaguides.net/2019/04/create-spring-boot-project-with-spring-initializer.html).

Hints 2. For example if you have chosen to use the Spring project, please, consider the followings:

The only dependency we have to import is "Spring for Apache ActiveMQ 5". This will also configure some default settings (which you can then override):
- if an ActiveMQ broker is found in the classpath it will be used, otherwise, an embedded one is used. You can also configure to use a broker at another IP address.
- a default JMSTemplate bean is added in the container, that will send messages to queues (not topics). (JMSTemplate is a helper class that simplifies synchronous JMS access code. It implements JmsOperations, thus has methods of many other classes: Connection, MessageProducer, MessageConsumer, etc.).
- a default MessageConverter bean is added to the container. This will be an instance of SimpleMessageConverter that can convert Strings, Serializable objects, Maps<String,?> and byte[] arrays to JMS messages.
-a default JmsListenerContainerFactory that creates JMSListeners that listen to queues (not topics).

We send a message from the main method and we receive it in a listener. Note that we do not need to create the destinations (queues, topics) beforehand, if they do not exist they are created by the ActiveMQ broker when the message is sent.

```
@SpringBootApplication
public class Jmswad1Application {

  public static void main(String[] args) {
    ConfigurableApplicationContext context =
SpringApplication.run(Jmswad1Application.class, args);
    JmsTemplate jmsTemplate = context.getBean(JmsTemplate.class);
    jmsTemplate.convertAndSend("mailbox", "hello world");
  }
}

@Component
public class Receiver {
  @JmsListener(destination = "mailbox")
  public void receiveMessage(String email) {
    System.out.println("Received <" + email + ">");
  }
}
```

2. In the second example, we want to send messages that encode a custom object, in this case, an email. However, we avoid making Email Serializable as serialization poses security risks and some brokers forbid it. Instead, we write a custom MessageConverter that will convert the message to JSON format, in a TextMessage. The converter is used both by the sender and receiver.

```
@SpringBootApplication
public class Jmswad2Application {
    @Bean
    public MessageConverter jacksonJmsMessageConverter() {
        MappingJackson2MessageConverter converter = new
MappingJackson2MessageConverter();
        converter.setTargetType(MessageType.TEXT);
        converter.setTypeIdPropertyName("_type");
        return converter;
    }

    public static void main(String[] args) {
        ConfigurableApplicationContext context =
SpringApplication.run(Jmswad2Application.class, args);
        JmsTemplate jmsTemplate = context.getBean(JmsTemplate.class);
        jmsTemplate.convertAndSend("mailbox", new Email("Jack","hi jack"));
    }
}

public class Email {

    private String to;
    private String body;

    public Email(String to, String body) {
        this.to = to;
        this.body = body;
    }

    @Override
    public String toString() {
        return String.format("Email{to=%s, body=%s}", getTo(), getBody());
    }
```

```
}

@Component
public class Receiver {
  @JmsListener(destination = "mailbox")
  public void receiveMessage(Email email) {
    System.out.println("Received <" + email + ">");
  }
}
```

3. In the third example, we want to modify and inspect the headers of the message. Also the receiver, after receiving a message can send a message to another queue. Note that the receiver is injected with a reference to the JMSTemplate.

```
@SpringBootApplication
public class Jmswad3Application {

  public static void main(String[] args) {

    ConfigurableApplicationContext context =
SpringApplication.run(Jmswad3Application.class, args);
    JmsTemplate jmsTemplate = context.getBean(JmsTemplate.class);
    jmsTemplate.convertAndSend("mailbox", new Email("Jack", "hi jack"), new
MessagePostProcessor() {
      @Override
      public Message postProcessMessage(Message message) throws JMSException {
        message.setJMSCorrelationID("15");
        return message;
      }
    });
  }

  @Bean
  public MessageConverter jacksonJmsMessageConverter() {
    MappingJackson2MessageConverter converter = new
MappingJackson2MessageConverter();
    converter.setTargetType(MessageType.TEXT);
    converter.setTypeIdPropertyName("_type");
    return converter;
  }
}

public class Email {

    private String to;
    private String body;

    public Email(String to, String body) {
        this.to = to;
        this.body = body;
    }

    @Override
    public String toString() {
        return String.format("Email{to=%s, body=%s}", getTo(), getBody());
    }
}

@Component
public class Receiver {

  @Autowired
```

```
  private JmsTemplate jmsTemplate;

  @JmsListener(destination = "mailbox")
  public void receiveMessage(Email email, @Header(JmsHeaders.MESSAGE_ID) String
messageId) {
    System.out.println("Received <" + email + ">, message id " + messageId);
    jmsTemplate.convertAndSend("mailbox2", new Email("to", "body"));
  }

  @JmsListener(destination = "mailbox2")
  public void receiveMessage2(Email email, @Header(JmsHeaders.MESSAGE_ID) String
messageId) {
    System.out.println("Received 2 <" + email + ">, message id " + messageId);
  }
}
```

4. In the fourth example, we switch to publisher-subscriber model. To do that:
- we implement a custom JMSTemplate that sends to queues and register it.
-we implement a custom JmsListenerContainerFactory that creates listeners that listen to queues and register it

```
@SpringBootApplication
public class Jmswad4Application {

  public static void main(String[] args) {
    ConfigurableApplicationContext context =
SpringApplication.run(Jmswad4Application.class, args);
    JmsTemplate jmsTemplate = context.getBean(JmsTemplate.class);
    jmsTemplate.convertAndSend("mailbox", new Email("Jack", "hi jack"), new
MessagePostProcessor() {
      @Override
      public Message postProcessMessage(Message message) throws JMSException {
        message.setJMSCorrelationID("15");
        return message;
      }
    });
  }

  @Bean
  public MessageConverter jacksonJmsMessageConverter() {
    MappingJackson2MessageConverter converter = new
MappingJackson2MessageConverter();
    converter.setTargetType(MessageType.TEXT);
    converter.setTypeIdPropertyName("_type");
    return converter;
  }

  @Bean
  public JmsTemplate jmsTemplate(ConnectionFactory connectionFactory,
MessageConverter converter) {
    JmsTemplate template = new JmsTemplate();
    template.setMessageConverter(converter);
    template.setConnectionFactory(connectionFactory);
    template.setPubSubDomain(true);
    return template;
  }

  @Bean
  public JmsListenerContainerFactory<?> myFactory(ConnectionFactory
connectionFactory,

DefaultJmsListenerContainerFactoryConfigurer configurer) {
```

```
    DefaultJmsListenerContainerFactory factory = new
DefaultJmsListenerContainerFactory();
    configurer.configure(factory, connectionFactory);
    factory.setPubSubDomain(true);
    return factory;
  }
}

public class Email {

    private String to;
    private String body;

    public Email(String to, String body) {
        this.to = to;
        this.body = body;
    }

    @Override
    public String toString() {
        return String.format("Email{to=%s, body=%s}", getTo(), getBody());
    }
}

@Component
public class Receiver {

  @Autowired
  private JmsTemplate jmsTemplate;

  @JmsListener(destination = "mailbox", containerFactory = "myFactory")
  public void receiveMessage(Email email, @Header(JmsHeaders.MESSAGE_ID) String
messageId) {
    System.out.println("Received <" + email + ">, message id " + messageId);
  }

  @JmsListener(destination = "mailbox", containerFactory = "myFactory")
  public void receiveMessage2(Email email, @Header(JmsHeaders.MESSAGE_ID) String
messageId) {
    System.out.println("Received <" + email + ">, message id " + messageId);
  }
}
```