CS 548—Fall 2018
Enterprise Software Architecture and Design
Assignment Nine—Message-Oriented Middleware

In the previous assignments, you defined both service-oriented and RESTful interfaces to a clinical information system. In this assignment, you will augment this basic system with a simple message-oriented middleware to effect a form of event-driven architecture. When a patient treatment is entered into the system, clearly both the patient and the provider parts of the system are interested knowing about this. However there are other parts of the enterprise that may also be interested. In this assignment, we will add use message-oriented middleware to notify additional parts of the enterprise when new treatments are added, as a simple example of an event-driven architecture.

Add a JMS queue for new treatments, organized as a publish-subscribe queue. When a new treatment is added to the treatments for a patient, a notification of this treatment is published on the queue. Any departments that wish to be notified of a new treatment must subscribe to the channel, and will receive notifications when treatments are added. The treatment information in the notification message can simply be a treatment DTO.

For this assignment, we will assume two departments, Billing and Research, that subscribe to this channel. The Billing department will want to be notified of any new treatment in order to bill patients. The Research department will specifically be interested in learning about new drug treatments. You must use publish-subscribe rather than point-to-point queues to notify these departments of the addition of a treatment. Furthermore the Research department should filter notifications based on metadata, so that it only receives notifications of new drug treatments.

Entities

The implementation of the Billing department is very simple. It is represented in the domain model by a `BillingRecord` entity that has a one-to-one relationship with a `Treatment` entity. Add this entity (with getters and setters) to your domain model:

```
package edu.stevens.cs548.clinic.billing;

@Entity
public class BillingRecord implements Serializable {

	@Id @GeneratedValue(strategy=IDENTITY)
	private long id;

	private String description;
```

```
	@Temporal(TemporalType.DATE)
	private Date date;

	private float amount;

	@OneToOne @PrimaryKeyJoinColumn
	private Treatment treatment;
}
```

Upon receiving a notification of a treatment, the Billing department should add a billing record to its database. Just use random numbers, within reason, for treatment costs. In a realistic system, you would also record a summary patient balance due in another table, which would be updated when new treatments are received, but we will eschew this in this assignment.

The Research department has a notion of subjects:

```
package edu.stevens.cs548.clinic.research;

@Entity
public class Subject implements Serializable {

	public Subject() {
		super();
		treatments = new ArrayList<DrugTreatmentRecord>();
	}

	/*
	 * This will be same as patient id in Clinic database
	 */
	@Id @GeneratedValue(strategy = IDENTITY)
	private long id;

	/*
	 * Anonymize patient (randomly generated when assigned)
	 */
	private long subjectId;

	@OneToMany(mappedBy = "subject")
	private Collection<DrugTreatmentRecord> treatments;
}
```

where a drug treatment record saves information about the treatment:

```
@Entity
public class DrugTreatmentRecord implements Serializable {

	@Id @GeneratedValue
	private long id;
```

```
        @Temporal(TemporalType.DATE)
        private Date date;

        private String drugName;

        private float dosage;

        @ManyToOne
        private Subject subject;
}
```

The Research database might be kept separate from the clinic database in a real setting, but for simplicity we will keep them together in the domain model. When a notification of a new drug treatment is received, the Research department will save a record of that treatment, creating a record for the subject if this is the first time a treatment has been recorded for this patient. In a real application, this database could also keep track of e.g. patients who have provided informed consent, but we will ignore these possibilities.

Services

The Billing department subscribes to the channel for new treatment notifications by registering a message listener bean. This bean creates a new `BillingRecord` entity (record), related to the entity for the treatment. *Complete a message-driven bean, called `TreatmentListener`, in the `ClinicBillingService` project, in the `edu.stevens.cs548.clinic.billing.jms` package*. The metadata for this MDB will need to specify a destination (say "`jms/Treatment`"). You will need to inject a `BillingService` bean into this MDB, in order to add a new billing record to the database.

The Research department also subscribes to the channel for new treatment notifications by registering a message listener bean. This bean creates a new `DrugTreatmentRecord` entity, as well as a `Subject` entity if necessary. *Complete a message-driven bean, called `DrugTreatmentListener`, in the `ClinicResearchService` project, in the `edu.stevens.cs548.clinic.research.jms` package*. The metadata for this MDB will need to specify the same destination name as before. You will need to inject a `ResearchService` bean into this MDB, in order to add a new drug treatment record to the database.

In your Provider service, when a treatment is added, you will need to also add a treatment notification to the event queue[1]. Note that the treatment notification must include the primary key for the treatment, which the billing record will use as the foreign key reference to the corresponding treatment record. Therefore it is important that the treatment record be added to the database in the Provider

---

[1] See the Clinic-EDA-Demo-3 demo video and on, for a demonstration of how to do this, although you should be careful to follow this assignment specification and not the video where it diverges.

[2] The binding file for the treatment DTO schema should include `<serializable uid="1"/>` as a child

service, before the notification is queued. Furthermore, you will need to make sure that the insertion of the treatment record is synchronized with the database (using em.flush()), so that the primary key is generated for the treatment record before the treatment DTO is constructed for the notification. Finally the Provider service will need to set the metadata so subscribers (e.g. Research) can filter based on this metadata. Here is an example of the code for broadcasting the addition of a new drug treatment, for example:

```
@Resource(mappedName="jms/TreatmentPool")
private ConnectionFactory treatmentConnFactory;

@Resource(mappedName="jms/Treatment")
private Topic treatmentTopic;

Connection treatmentConn = null;
try {

	treatmentConn = treatmentConnFactory.createConnection();
	Session session = treatmentConn.createSession(true,
                                                    Session.AUTO_ACKNOWLEDGE);
	MessageProducer producer = session.createProducer(treatmentTopic);

	TreatmentDtoFactory f = new TreatmentDtoFactory();
	TreatmentDto treatment = f.createDrugTreatmentDto();
	DrugTreatmentType drugTreatment = new DrugTreatmentType();
	drugTreatment.setDrug(...);
	drugTreatment.setDosage(...);
	treatment.setDrugTreatment(drugTreatment);
	treatment.setDiagnosis(...);
	treatment.setId(...);
	treatment.setPatient(...);
	treatment.setProvider(...);

	ObjectMessage message = session.createObjectMessage();
	message.setObject(treatment);
	producer.send(message);

} catch (JMSException e) {
	logger.log(Level.SEVERE, "JMS Error: ", e);
} finally {
	try {
		if (treatmentConn != null) treatmentConn.close();
	} catch (JMSException e) {
		logger.severe("Error closing JMS connection: "+e);
	}
}
```

You should set up a JMS topic and connection factory in Glassfish using the administrative console. Give the topic a JNDI name of `jms/Treatment` (with physical destination name `Treatment`), and give the connection pool a JNDI name of `jms/TreatmentPool`. These are the names that you will use when injecting a topic and connection factory resource, respectively, when you are publishing a treatment

message[2] on the JMS publish/subscribe topic that you have set up. The metadata for the message-driven beans that receive messages specify `jms/Treatment` as their `mappedName`.

Testing

You are provided with simple JSF Web apps that shows the contents of the Billing and Research databases. You should record one or more videos that show the contents of these databases before adding treatments, then use the REST API to add treatments, then show the contents of these databases after the addition of those treatments. Make sure that your name appears in the video(s).

Submission

Your solution should be uploaded via the Canvas classroom, as a zip file. This zip file should have the same name as your Canvas userid. It should unzip to a folder with this same name, which should contain the files and subfolders with your submission. You should also include a completed rubric for the assignment.

---

[2] The binding file for the treatment DTO schema should include `<serializable uid="1"/>` as a child element, to ensure that the JAXB class generated from the schema implements the `java.io.Serializable` interface. You will need this in order to be able to send a DTO on a JMS queue or topic.