**Creating SQL Script to execute while application loads**

By default, data.sql scripts are now run before Hibernate is initialized. This aligns the behavior of basic script-based initialization with that of Flyway and Liquibase.

spring.jpa.defer-datasource-initialization=true

**Calling properties from the properties file**

"@ConfigurationProperties(prefix = "sfg.brewery", ignoreUnknownFields = false)" , we can define properties starting with sfg.brewery in application.properties and it will map to field of the class.

Ex : sfg.brewery.beer-inventory-service-host=http://localhost:8082

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Caching**

1.dependency:

<**dependency**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-starter-cache</**artifactId**>

</**dependency**>

<**dependency**>

<**groupId**>javax.cache</**groupId**>

<**artifactId**>cache-api</**artifactId**>

</**dependency**>

<**dependency**>

<**groupId**>org.ehcache</**groupId**>

<**artifactId**>ehcache</**artifactId**>

</**dependency**>

2.xml file in resource (ehcache.xml)

<**config**

xmlns:jsr107=*'http://www.ehcache.org/v3/jsr107'*

xmlns=*'http://www.ehcache.org/v3'*>

<**service**>

<**jsr107:defaults** enable-management=*"true"* enable-statistics=*"true"*/>

</**service**>

<**cache** alias=*"beerCache"* uses-template=*"config-cache"*/>

<**cache** alias=*"beerListCache"* uses-template=*"config-cache"*/>

<**cache-template** name=*"config-cache"*>

<**expiry**>

<**ttl** unit=*"minutes"*>5</**ttl**>

</**expiry**>

<**resources**>

<**heap**>1</**heap**>

<**offheap** unit=*"MB"*>1</**offheap**>

</**resources**>

</**cache-template**>

</config>

3. application.properties

spring.cache.jcache.config=classpath:ehcache.xml

4. creat a class

*@Configuration*

*@EnableCaching*

public class CacheConfig {

}

5. using in the method

@Cacheable(cacheNames = "beerCache", key = "#beerId",

condition = "#showInventoryOnHand == false ")

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**JaxB dependencies**

Jakarta XML Binding [4.0](https://jakarta.ee/specifications/xml-binding/4.0/) (requires Java SE 11 or >

<**dependency**>

<**groupId**>jakarta.xml.bind</**groupId**>

<**artifactId**>jakarta.xml.bind-api</**artifactId**>

<**version**>4.0.0</**version**>

</**dependency**>

<**dependency**>

<**groupId**>com.sun.xml.bind</**groupId**>

<**artifactId**>jaxb-impl</**artifactId**>

<**version**>4.0.0</**version**>

<**scope**>runtime</**scope**>

</**dependency**>

-----------------------------------------------------------

<**properties**>

<**jaxb.version**>2.3.0</**jaxb.version**>

</**properties**>

<**dependency**>

<**groupId**>javax.xml.bind</**groupId**>

<**artifactId**>jaxb-api</**artifactId**>

<**version**>${jaxb.version}</**version**>

</**dependency**>

<**dependency**>

<**groupId**>com.sun.xml.bind</**groupId**>

<**artifactId**>jaxb-core</**artifactId**>

<**version**>${jaxb.version}</**version**>

</**dependency**>

<**dependency**>

<**groupId**>com.sun.xml.bind</**groupId**>

<**artifactId**>jaxb-impl</**artifactId**>

<**version**>${jaxb.version}</**version**>

</**dependency**>

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Plugin to auto clean**

<**plugin**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-maven-plugin</**artifactId**>

</**plugin**>

<**plugin**>

<**artifactId**>maven-clean-plugin</**artifactId**>

<**executions**>

<**execution**>

<**id**>auto-clean</**id**>

<**phase**>initialize</**phase**>

<**goals**>

<**goal**>clean</**goal**>

</**goals**>

</**execution**>

</**executions**>

</**plugin**>

**Applying Rules to BOM**

**(Sfg-beer-works dependency Link :** https://mvnrepository.com/artifact/com.github.sfg-beer-works/sfg-brewery-bom)

<**plugin**>

<**groupId**>org.apache.maven.plugins</**groupId**>

<**artifactId**>maven-enforcer-plugin</**artifactId**>

<**executions**>

<**execution**>

<**id**>enforce-versions</**id**>

<**goals**>

<**goal**>enforce</**goal**>

</**goals**>

<**configuration**>

<**rules**>

<**requireMavenVersion**>

<**version**>[3.6.0,)</**version**>

</**requireMavenVersion**>

<**requireJavaVersion**>

<**version**>11</**version**>

</**requireJavaVersion**>

<**requireReleaseDeps**>

<**onlyWhenRelease**>true</**onlyWhenRelease**>

<**message**>Release builds must not have on

snapshot dependencies

</**message**>

</**requireReleaseDeps**>

</**rules**>

</**configuration**>

</**execution**>

</**executions**>

</**plugin**>

**Excluding the specific class during springBoot class loading**

*@SpringBootApplication(****exclude = ClassName.class****)*

public class MsscBreweryApplication {

----

**H2 DB CONFIG**

Dependecy:

<dependency>  
 <groupId>com.h2database</groupId>  
 <artifactId>h2</artifactId>  
 <scope>runtime</scope>  
</dependency>

Application.properties:

spring.datasource.url=jdbc:h2:mem:testdb  
spring.datasource.driverClassName=org.h2.Driver  
spring.datasource.username=sa  
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

Run: http://localhost:8080/h2-console

**MySQL DB Connection in properties file**

spring.datasource.url=jdbc:mysql://127.0.0.1:3306/beerservice?useUnicode=true&char

acterEncoding=UTF-8&serverTimezone=UTC

spring.datasource.username=beer\_service

spring.datasource.password=password

spring.jpa.database=mysql

spring.jpa.hibernate.ddl-auto=update

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

**Hikari Connection Pooling**

spring.datasource.hikari.maximum-pool-size=5

best for MqSql

spring.datasource.hikari.data-source-properties.cachePrepStmts=true

spring.datasource.hikari.data-source-properties.prepStmtCacheSize=250

spring.datasource.hikari.data-source-properties.prepStmtCacheSqlLimit=2048

spring.datasource.hikari.data-source-properties.useServerPrepStmts=true

spring.datasource.hikari.data-source-properties.useLocalSessionState=true

spring.datasource.hikari.data-source-properties.rewriteBatchedStatements=true

spring.datasource.hikari.data-source-properties.cacheResultSetMetadata=true

spring.datasource.hikari.data-source-properties.cacheServerConfiguration=true

spring.datasource.hikari.data-source-properties.elideSetAutoCommits=true

spring.datasource.hikari.data-source-properties.maintainTimeStats=false

Enable logging for config troubleshooting

logging.level.org.hibernate.SQL=DEBUG

logging.level.com.zaxxer.hikari.HikariConfig=DEBUG

logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE

JMS

**1.0 Apache ActiveMQ Artemis dependencies and usage vn:172**

By spring initializer it will add only core and client, it only gives ability to talk to servers

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-artemis</artifactId>  
</dependency>

**To run embedded servers we have to add extra dependencies (typical case)**

<dependency>  
 <groupId>org.apache.activemq</groupId>  
 <artifactId>artemis-server</artifactId>  
</dependency>  
<dependency>  
 <groupId>org.apache.activemq</groupId>  
 <artifactId>artemis-jms-server</artifactId>  
</dependency>

Inside main application

ActiveMQServer server = ActiveMQServers.*newActiveMQServer*(new ConfigurationImpl()  
 .setPersistenceEnabled(false)  
 .setJournalDirectory("target/data/journal")  
 .setSecurityEnabled(false)  
 .addAcceptorConfiguration("invm", "vm://0"));  
  
server.start();

***1.1 Hello world Example***

**Create a Message Model**

@Data  
@Builder  
@AllArgsConstructor  
@NoArgsConstructor  
public class HelloWorldMessage implements Serializable {  
  
 static final long *serialVersionUID* = 6056263104526923315L;  
  
 private UUID id;  
 private String message;  
}

**Enabling Task Scheduling**

@EnableScheduling  
@EnableAsync  
@Configuration  
public class TaskConfig {  
  
 @Bean  
 TaskExecutor taskExecutor(){  
 return new SimpleAsyncTaskExecutor();  
 }  
}

Img:1.0

**Message convertor to convert json ↔ object**

import org.springframework.jms.support.converter.\*;

@Configuration  
public class JmsConfig {  
  
 public static final String *MY\_QUEUE* = "my-hello-world";  
  
 @Bean  
 public MessageConverter messageConverter() {  
 MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter();  
 converter.setTargetType(MessageType.*TEXT*);  
 converter.setTypeIdPropertyName("\_type"); //need tobe set in order to convert incoming massage to java object (default is "none")  
 return converter;  
 }  
}

Img:1.2

MessageConverter class:

Strategy interface that specifies a converter between Java objects and JMS messages.

Convert a Java object to a JMS Message using the supplied session to create the message object.  
Params: object – the object to convert ,session – the Session to use for creating a JMS Message  
Returns the JMS Message.

MappingJackson2MessageConverter class : Message converter that uses Jackson 2.x to convert messages to and from JSON. Maps an object to a BytesMessage, or to a TextMessage if the targetType is set to MessageType.TEXT. Converts from a TextMessage or BytesMessage to an object.

**Sender**

@RequiredArgsConstructor  
@Component  
public class HelloSender {  
  
 private final JmsTemplate jmsTemplate; //JmsTemplate(for messaging) is pre configured like JdbcTemplate(for db)  
  
 @Scheduled(fixedRate = 2000)  
 public void sendMessage(){  
 System.*out*.println("I'm Sending a message");  
  
 HelloWorldMessage message = HelloWorldMessage  
 .*builder*()  
 .id(UUID.*randomUUID*())  
 .message("Hello World!")  
 .build();  
  
 jmsTemplate.convertAndSend(JmsConfig.*MY\_QUEUE*, message);  
  
 System.*out*.println("Message Sent!");  
 }  
}

**Note:** Jms maintains transactional. If client doesnot confirms receiving the message then that message gets requed and resent.

**Listener**

@Component  
public class HelloMessageListener {  
  
 @JmsListener(destination = JmsConfig.*MY\_QUEUE*)  
 public void listen(@Payload HelloWorldMessage helloWorldMessage,  
 @Headers MessageHeaders headers, Message message){  
 System.*out*.println("I Got a Message!!!!!");  
  
 System.*out*.println(helloWorldMessage);  
  
 // uncomment and view to see retry count in debugger  
 // throw new RuntimeException("foo");  
 }  
}

**O/P:**

I'm Sending a message

Message Sent!

I Got a Message!!!!!

HelloWorldMessage(id=68aa1f77-c379-49a1-acd4-20ccf1bbf229, message=Hello World!)

***1.11 Sending and Receiving Message Example*** vn:180

@RequiredArgsConstructor  
@Component  
public class HelloSender {

“”””””””””

private final ObjectMapper objectMapper;

“”””””””

@Scheduled(fixedRate = 2000)  
 public void sendAndRecieveMessage() throws JMSException {  
  
 HelloWorldMessage message = HelloWorldMessage  
 .*builder*()  
 .id(UUID.*randomUUID*())  
 .message("Hello ")  
 .build();  
  
 Message recievedMessage = jmsTemplate.sendAndReceive(JmsConfig.*MY\_SEND\_RCV\_QUEUE*, new MessageCreator() {  
 @Override  
 public Message createMessage(Session session) throws JMSException {  
 Message helloMessage = null;  
 try {  
 //we have to manually configure converter using objectMapper bcz  
 // our convertor(MessageConverter from JmsConfig.class) doesnot work here  
 helloMessage = session.createTextMessage(objectMapper.writeValueAsString(message));  
 //set property to convert to java Object  
 helloMessage.setStringProperty("\_type","guru.springframework.sfgjms.model.HelloWorldMessage");  
 System.*out*.println("Sending Hello");  
 return helloMessage;  
 }catch (JsonProcessingException e){  
 throw new JMSException(e.getMessage());  
 }  
 }  
 });  
  
 System.*out*.println(recievedMessage.getBody(String.class));  
  
 }  
}

import org.springframework.jms.support.converter.\*;

@Configuration  
public class JmsConfig {

“””””””

public static final String *MY\_SEND\_RCV\_QUEUE* = "replybacktome";

@Bean  
 public MessageConverter messageConverter() {

“””””””””””  
 }  
}

@RequiredArgsConstructor  
@Component  
public class HelloMessageListener {  
  
 private final JmsTemplate jmsTemplate;  
  
 “””””””””””””””  
  
 @JmsListener(destination = JmsConfig.*MY\_SEND\_RCV\_QUEUE*)  
 public void listenForHello(@Payload HelloWorldMessage helloWorldMessage,  
 @Headers MessageHeaders headers, Message message) throws JMSException {  
  
 HelloWorldMessage payloadMsg = HelloWorldMessage  
 .*builder*()  
 .id(UUID.*randomUUID*())  
 .message("World!")  
 .build();  
  
 jmsTemplate.convertAndSend(message.getJMSReplyTo(),payloadMsg);  
  
 }  
}

**O/P:**

Sending Hello

{"id":"9e77dd6f-c3a1-49f7-b46e-99c88b613d85","message":"World!"}

**1.2 ActiveMQ Artemis Docker** vn:181

GitHub Link: <https://github.com/vromero/activemq-artemis-docker>

Docker command to run the image: docker run -it --rm -p 8161:8161 -p 61616:61616 vromero/activemq-artemis

Error!!!

Ports are not available: listen tcp 0.0.0.0:61616: bind: An attempt was made to access a socket in a way forbidden by its access permissions.

Solution**:** In windows as administrator

> net stop winnat

> net start winnat

Port: <http://127.0.0.1:8161>

To use ActiveMQ Artemis from our spring boot app (client) (disable embedded server)

In application.properties

spring.artemis.user=artemis  
spring.artemis.password=simetraehcapa

Img:1.3

**Note**: ActiveMQ is the implementation of Message (I) from javax.jms package.

Example to use Spring **Message** Type

* Easy to swap b/w kafka, rabbit, …

jmsTemplate.convertAndSend((Destination) springMessage.getHeaders().get("jms\_replyTo"), "got it!");

Example to use Jms **Message** Type

jmsTemplate.convertAndSend(jmsMessage.getJMSReplyTo(), payloadMsg);

**1.3 JMS Messaging b/w MicroServices**

***Service: Mssc-beer-service2***

1.create jms config file: refer Img:1.2

@Configuration  
public class JmsConfig {  
  
 public static final String BREWING\_REQUEST\_QUEUE = "brewing-request";

//Convert Message to Object (or vice-versa)  
 “”””””””  
}

2.add ActiveMQ credentials to application.properties file: refer Img:1.3

3.add events

@Data  
@RequiredArgsConstructor  
@Builder  
public class BeerEvent implements Serializable {  
  
 static final long *serialVersionUID* = -6612649371973922530L;  
  
 private final BeerDto beerDto;  
}

public class BrewBeerEvent extends BeerEvent{  
 public BrewBeerEvent(BeerDto beerDto) {  
 super(beerDto);  
 }  
}

public class NewInventoryEvent extends BeerEvent{  
 public NewInventoryEvent(BeerDto beerDto) {  
 super(beerDto);  
 }  
}

4.enable task scheduling: refer Img:1.0

5.create brewing service

@Slf4j  
@RequiredArgsConstructor  
@Service  
public class BrewingService {  
 private final BeerRepository beerRepository;  
 private final BeerInventoryService beerInventoryService;  
 private final JmsTemplate jmsTemplate;  
 private final BeerMapper beerMapper;  
  
 @Scheduled(fixedRate = 5000) //every 5 seconds  
 public void checkForLowInventory(){  
 List<Beer> beers = beerRepository.findAll();  
  
 beers.forEach(beer -> {  
 Integer invQOH = beerInventoryService.getOnhandInventory(beer.getId());  
 *log*.debug("Checking Inventory for: " + beer.getBeerName() + " / " + beer.getId());  
 *log*.debug("Min Onhand is: " + beer.getMinOnHand());  
 *log*.debug("Inventory is: " + invQOH);  
  
 if(beer.getMinOnHand() >= invQOH){  
 jmsTemplate.convertAndSend(JmsConfig.*BREWING\_REQUEST\_QUEUE*, new BrewBeerEvent(beerMapper.beerToBeerDto(beer)));  
 }  
 });  
 }  
}

6.create the listeners

@Service  
@RequiredArgsConstructor  
@Slf4j  
public class BrewBeerListener {  
  
 private final BeerRepository beerRepository;  
 private final JmsTemplate jmsTemplate;  
  
 @JmsListener(destination = JmsConfig.*BREWING\_REQUEST\_QUEUE*)  
 public void listen(BrewBeerEvent brewBeerEvent){  
 BeerDto beerDto = brewBeerEvent.getBeerDto();  
  
 Beer beer = beerRepository.getOne(beerDto.getId());  
 beerDto.setQuantityOnHand(beer.getQuantityToBrew());  
 NewInventoryEvent newInventoryEvent = new NewInventoryEvent(beerDto);  
  
 *log*.debug("Brewed beer " + beer.getBeerName() + " : QOH: " + beerDto.getQuantityOnHand());  
  
 jmsTemplate.convertAndSend(JmsConfig.*NEW\_INVENTORY\_QUEUE*, newInventoryEvent);  
 }  
}

6.1 add new Constant to JmsConfig class

@Configuration  
public class JmsConfig {  
  
 “”””””””””

public static final String *NEW\_INVENTORY\_QUEUE* = "new-inventory";  
  
 //Convert Message to Object (or vice-versa)  
 “”””””””  
}

Error!!!

org.springframework.jms.listener.adapter.ListenerExecutionFailedException: Listener method 'public void com.vtech.msscbrewery.services.brewing.BrewBeerListener.listen(com.vtech.msscbrewery.events.BrewBeerEvent)' threw exception; nested exception is org.springframework.jms.support.converter.MessageConversionException: Failed to convert JSON message content; nested exception is com.fasterxml.jackson.databind.exc.InvalidDefinitionException: Cannot construct instance of `**java.time.OffsetDateTime**` (no Creators, like default construct, exist): cannot deserialize from Object value (no delegate- or property-based Creator)

reason: MessageConvertor doesnot use the springboot managed instance of jaxon ObjectMapper, so we should configure it.

Solution:

@Configuration  
public class JmsConfig {  
  
 public static final String *BREWING\_REQUEST\_QUEUE* = "brewing-request";  
 public static final String *NEW\_INVENTORY\_QUEUE* = "new-inventory";  
  
 //Strategy interface that specifies a converter between Java objects and JMS messages.  
 //Serialize message content to json using TextMessage  
 @Bean  
 public MessageConverter messageConverter(ObjectMapper objectMapper) {  
 MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter();  
 converter.setTargetType(MessageType.*TEXT*);  
 converter.setTypeIdPropertyName("\_type"); //need tobe set in order to convert incoming messge to java object (default is "none")  
 converter.setObjectMapper(objectMapper);  
 return converter;  
 }  
}

Error!!!

org.springframework.jms.listener.adapter.ListenerExecutionFailedException: Listener method 'public void com.vtech.msscbrewery.services.brewing.BrewBeerListener.listen(com.vtech.msscbrewery.events.BrewBeerEvent)' threw exception; nested exception is org.hibernate.LazyInitializationException: could not initialize proxy [com.vtech.msscbrewery.domain.Beer#7a08bdbb-e944-4762-966e-5822d4e61e3d] - no Session

reason: Hibernate was doing lazy initialization of property and we were running outside of transactional scope, so there was no hibernate session to work with.

Solution: add @Transactional for listener

import org.springframework.transaction.annotation.Transactional;

@Service  
@RequiredArgsConstructor  
@Slf4j  
public class BrewBeerListener {  
 private final BeerRepository beerRepository;  
 private final JmsTemplate jmsTemplate;  
  
 @Transactional  
 @JmsListener(destination = JmsConfig.*BREWING\_REQUEST\_QUEUE*)  
 public void listen(BrewBeerEvent brewBeerEvent){  
 BeerDto beerDto = brewBeerEvent.getBeerDto();  
   
 Beer beer = beerRepository.getOne(beerDto.getId());  
 beerDto.setQuantityOnHand(beer.getQuantityToBrew());  
 NewInventoryEvent newInventoryEvent = new NewInventoryEvent(beerDto);  
  
 *log*.info("Brewed beer " + beer.getBeerName() + " : QOH: " + beerDto.getQuantityOnHand());  
  
 jmsTemplate.convertAndSend(JmsConfig.*NEW\_INVENTORY\_QUEUE*, newInventoryEvent);  
 }  
}

Img:1.4

**Note:reffer** [**https://stackoverflow.com/questions/1099025/spring-transactional-what-happens-in-background**](https://stackoverflow.com/questions/1099025/spring-transactional-what-happens-in-background) **to what happnes in the background when we use @Transactional**

**Use: rollback feature,…**

**Setting log levels**

logging.level.guru.springframework=debug or info or …

**Talking to Different MicroService**

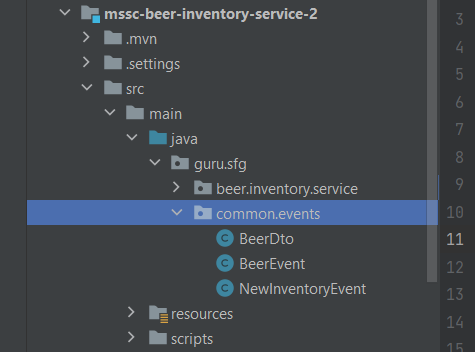
Talking from mssc-beer-service2 to mssc-beer-inventory-service(listener)

***mssc-beer-inventory-service*** changes

1.create services package and create NewInventoryListener.java class and also add constant to JmsConfig(NEW\_INVENTORY\_QUEUE)

@RequiredArgsConstructor  
@Component  
public class NewInventoryListener {  
 private final BeerInventoryRepository beerInventoryRepository;  
  
 @JmsListener(destination = JmsConfig.*NEW\_INVENTORY\_QUEUE*) // *NEW\_INVENTORY\_QUEUE= “*new-inventory”  
 public void listen(NewInventoryEvent event){  
 *log*.debug("Got Inventory: " + event.toString());  
 beerInventoryRepository.save(BeerInventory.*builder*()  
 .beerId(event.getBeerDto().getId())  
 .upc(event.getBeerDto().getUpc())  
 .quantityOnHand(event.getBeerDto().getQuantityOnHand()).build());   
 }}

2.create a new package(common.events) outside normal package and copy the required event classes from mssc-beer-service2



3.do the same for mssc-beer-service2 also

4.add ObjectMapper config in JmsConfig. refer Img:1.4

**Note:**We are maintaining same package name for events because in the mssc-beer-service2 we are trying to deserialize these event classes in different service(mssc-beer-inventory-service).

**2.0 Spring State Machine (SSM)**

Intorduction: reffer “[IntroductionToSpringStateMachine.pdf](pdf-notes/IntroductionToSpringStateMachine.pdf)” in NOTES > pdf-notes

Exmaple: refer “[paymentStateMachine](pdf-notes/PaymentsStateMachine.pdf)” in NOTES > pdf-notes (credit card example)

Credit Card Example Project SetUp

**Spring State Machine Dependecy**

<dependency>  
 <groupId>org.springframework.statemachine</groupId>  
 <artifactId>spring-statemachine-core</artifactId>  
 <version>3.1.0</version>  
</dependency>

**Add enums and class in domain packge**

package guru.springframework.msscssm.domain;

public enum PaymentState {  
 *NEW*, *PRE\_AUTH*, *PRE\_AUTH\_ERROR*, *AUTH*, *AUTH\_ERROR*}

public enum PaymentEvent {  
 *PRE\_AUTHORIZE*, *PRE\_AUTH\_APPROVED*, *PRE\_AUTH\_DECLINED*, *AUTHORIZE*, *AUTH\_APPROVED*, *AUTH\_DECLINED*}

@Data  
@Builder  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
public class Payment {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 @Column(name = "id", nullable = false)  
 private Long id;  
  
 @Enumerated(EnumType.*STRING*) //in db it set datatype as varchar and stores real value, otherwise it stores numbers  
 private PaymentState state;  
  
 private BigDecimal amount;  
  
}

**Create Payment Repository**

package guru.springframework.msscssm.repository;   
  
public interface PaymentRepository extends JpaRepository<Payment,Long> {  
}

**Configure Spring State Machine**

@Slf4j  
@EnableStateMachineFactory  
@Configuration  
public class StateMachineConfig extends StateMachineConfigurerAdapter<PaymentState, PaymentEvent> {  
  
 //States  
 @Override  
 public void configure(StateMachineStateConfigurer<PaymentState, PaymentEvent> states) throws Exception {  
 states.withStates()  
 .initial(PaymentState.*NEW*)  
 .states(EnumSet.*allOf*(PaymentState.class))  
 .end(PaymentState.*AUTH*)  
 .end(PaymentState.*PRE\_AUTH\_ERROR*)  
 .end(PaymentState.*AUTH\_ERROR*);  
 }  
 //transitions  
 @Override  
 public void configure(StateMachineTransitionConfigurer<PaymentState, PaymentEvent> transitions) throws Exception {  
 transitions.withExternal().source(PaymentState.*NEW*).target(PaymentState.*NEW*).event(PaymentEvent.*PRE\_AUTHORIZE*)  
 .and()  
 .withExternal().source(PaymentState.*NEW*).target(PaymentState.*PRE\_AUTH*).event(PaymentEvent.*PRE\_AUTH\_APPROVED*)  
 .and()  
 .withExternal().source(PaymentState.*NEW*).target(PaymentState.*PRE\_AUTH\_ERROR*).event(PaymentEvent.*PRE\_AUTH\_DECLINED*);  
 // .and()  
 // .withExternal().source(PaymentState.*NEW*).target(PaymentState.*PRE\_AUTH\_ERROR*).event(PaymentEvent.*AUTHORIZE*)  
 // .and()  
 // .withExternal().source(PaymentState.*NEW*).target(PaymentState.*AUTH*).event(PaymentEvent.*AUTH\_APPROVED*)  
 // .and()  
 // .withExternal().source(PaymentState.*NEW*).target(PaymentState.*AUTH\_ERROR*).event(PaymentEvent.*AUTH\_DECLINED*);  
 }  
}

**Create a Test for StateMachineConfig**

@SpringBootTest  
class StateMachineConfigTest {  
  
 @Autowired  
 StateMachineFactory<PaymentState, PaymentEvent> factory;  
  
 @Test  
 void testNewStateMachine() {  
 StateMachine<PaymentState, PaymentEvent> sm = factory.getStateMachine(UUID.*randomUUID*());  
  
 sm.start();  
  
 System.*out*.println(sm.getState().toString());  
  
 sm.sendEvent(PaymentEvent.*PRE\_AUTHORIZE*);  
  
 System.*out*.println(sm.getState().toString());  
  
 sm.sendEvent(PaymentEvent.*PRE\_AUTH\_APPROVED*);  
  
 System.*out*.println(sm.getState().toString());  
  
 sm.sendEvent(PaymentEvent.*PRE\_AUTH\_DECLINED*);  
  
 System.*out*.println(sm.getState().toString());  
  
 }  
}

**O/P:**

**ObjectState [getIds()=[NEW], getClass()=class org.springframework.statemachine.state.ObjectState, hashCode()=1807430272, toString()=AbstractState [id=NEW, pseudoState=org.springframework.statemachine.state.DefaultPseudoState@7dda5b25, deferred=[], entryActions=[], exitActions=[], stateActions=[], regions=[], submachine=null]]**

**ObjectState [getIds()=[NEW], getClass()=class org.springframework.statemachine.state.ObjectState, hashCode()=1807430272, toString()=AbstractState [id=NEW, pseudoState=org.springframework.statemachine.state.DefaultPseudoState@7dda5b25, deferred=[], entryActions=[], exitActions=[], stateActions=[], regions=[], submachine=null]]**

**ObjectState [getIds()=[PRE\_AUTH], getClass()=class org.springframework.statemachine.state.ObjectState, hashCode()=1372771126, toString()=AbstractState [id=PRE\_AUTH, pseudoState=null, deferred=[], entryActions=[], exitActions=[], stateActions=[], regions=[], submachine=null]]**

**ObjectState [getIds()=[PRE\_AUTH], getClass()=class org.springframework.statemachine.state.ObjectState, hashCode()=1372771126, toString()=AbstractState [id=PRE\_AUTH, pseudoState=null, deferred=[], entryActions=[], exitActions=[], stateActions=[], regions=[], submachine=null]]**

From the O/P, In the last state, even we set the event to *PRE\_AUTH\_DECLINED* it was in **PRE\_AUTH** only because We don’t hava transition set up from **PRE\_AUTH** to **PRE\_AUTH\_ERROR.**

So if we implement like this, then the state will set to **PRE\_AUTH\_ERROR**

transitions.withExternal().source(PaymentState.*NEW*).target(PaymentState.*NEW*).event(PaymentEvent.*PRE\_AUTHORIZE*)  
 .and()  
 .withExternal().source(PaymentState.*NEW*).target(PaymentState.*PRE\_AUTH*).event(PaymentEvent.*PRE\_AUTH\_APPROVED*)  
 .and()  
 .withExternal().source(PaymentState.*PRE\_AUTH*).target(PaymentState.*PRE\_AUTH\_ERROR*).event(PaymentEvent.*PRE\_AUTH\_DECLINED*)  
 .and()  
 .withExternal().source(PaymentState.*NEW*).target(PaymentState.*PRE\_AUTH\_ERROR*).event(PaymentEvent.*PRE\_AUTH\_DECLINED*);

**Logging Configuration using State Change Listeners**

@Slf4j  
@EnableStateMachineFactory  
@Configuration  
public class StateMachineConfig extends StateMachineConfigurerAdapter<PaymentState, PaymentEvent> {  
  
 //States  
 @Override  
 public void configure(StateMachineStateConfigurer<PaymentState, PaymentEvent> states) throws Exception {  
 states.withStates()  
 .initial(PaymentState.*NEW*)  
 .states(EnumSet.*allOf*(PaymentState.class))  
 .end(PaymentState.*AUTH*)  
 .end(PaymentState.*PRE\_AUTH\_ERROR*)  
 .end(PaymentState.*AUTH\_ERROR*);  
 }  
 //transitions  
 @Override  
 public void configure(StateMachineTransitionConfigurer<PaymentState, PaymentEvent> transitions) throws Exception {  
 transitions.withExternal().source(PaymentState.*NEW*).target(PaymentState.*NEW*).event(PaymentEvent.*PRE\_AUTHORIZE*)  
 .and()  
 .withExternal().source(PaymentState.*NEW*).target(PaymentState.*PRE\_AUTH*).event(PaymentEvent.*PRE\_AUTH\_APPROVED*)  
 .and()  
 .withExternal().source(PaymentState.*PRE\_AUTH*).target(PaymentState.*PRE\_AUTH\_ERROR*).event(PaymentEvent.*PRE\_AUTH\_DECLINED*)  
 .and()  
 .withExternal().source(PaymentState.*NEW*).target(PaymentState.*PRE\_AUTH\_ERROR*).event(PaymentEvent.*PRE\_AUTH\_DECLINED*);  
 }  
  **--------------------------------------------------------------**  
 @Override  
 public void configure(StateMachineConfigurationConfigurer<PaymentState, PaymentEvent> config) throws Exception {  
 StateMachineListenerAdapter<PaymentState, PaymentEvent> adapter = new StateMachineListenerAdapter<>(){  
 @Override  
 public void stateChanged(State<PaymentState, PaymentEvent> from, State<PaymentState, PaymentEvent> to) {  
 *log*.info(String.*format*("state changed(from: %s, to:%s)",from,to));  
 }  
 };  
 config.withConfiguration().listener(adapter);  
 }

**---------------------------------------------------------**  
}

Img:stateMachineConfig

Test it with running test cases

**O/P:** **2022-09-10 11:47:09.670 INFO 1484 --- [ main] g.s.msscssm.config.StateMachineConfig : state changed(from: null, to:ObjectState [getIds()=[NEW], getClass()=class org.springframework.statemachine.state.ObjectState, hashCode()=653292129, toString()=AbstractState [id=NEW, pseudoState=org.springframework.statemachine.state.DefaultPseudoState@2b55ea4d, deferred=[], entryActions=[], exitActions=[], stateActions=[], regions=[], submachine=null]])**

**ObjectState [getIds()=[NEW], getClass()=class org.springframework.statemachine.state.ObjectState, hashCode()=653292129, toString()=AbstractState [id=NEW, pseudoState=org.springframework.statemachine.state.DefaultPseudoState@2b55ea4d, deferred=[], entryActions=[], exitActions=[], stateActions=[], regions=[], submachine=null]]**

**2022-09-10 11:47:09.713 INFO 1484 --- [ main] g.s.msscssm.config.StateMachineConfig : state changed(from: ObjectState [getIds()=[NEW], getClass()=class org.springframework.statemachine.state.ObjectState, hashCode()=653292129, toString()=AbstractState [id=NEW, pseudoState=org.springframework.statemachine.state.DefaultPseudoState@2b55ea4d, deferred=[], entryActions=[], exitActions=[], stateActions=[], regions=[], submachine=null]], to:ObjectState [getIds()=[NEW], getClass()=class org.springframework.statemachine.state.ObjectState, hashCode()=653292129, toString()=AbstractState [id=NEW, pseudoState=org.springframework.statemachine.state.DefaultPseudoState@2b55ea4d, deferred=[], entryActions=[], exitActions=[], stateActions=[], regions=[], submachine=null]])**

**ObjectState [getIds()=[NEW], getClass()=class org.springframework.statemachine.state.ObjectState, hashCode()=653292129, toString()=AbstractState [id=NEW, pseudoState=org.springframework.statemachine.state.DefaultPseudoState@2b55ea4d, deferred=[], entryActions=[], exitActions=[], stateActions=[], regions=[], submachine=null]]**

**2022-09-10 11:47:09.717 INFO 1484 --- [ main] g.s.msscssm.config.StateMachineConfig : state changed(from: ObjectState [getIds()=[NEW], getClass()=class org.springframework.statemachine.state.ObjectState, hashCode()=653292129, toString()=AbstractState [id=NEW, pseudoState=org.springframework.statemachine.state.DefaultPseudoState@2b55ea4d, deferred=[], entryActions=[], exitActions=[], stateActions=[], regions=[], submachine=null]], to:ObjectState [getIds()=[PRE\_AUTH], getClass()=class org.springframework.statemachine.state.ObjectState, hashCode()=187723824, toString()=AbstractState [id=PRE\_AUTH, pseudoState=null, deferred=[], entryActions=[], exitActions=[], stateActions=[], regions=[], submachine=null]])**

**ObjectState [getIds()=[PRE\_AUTH], getClass()=class org.springframework.statemachine.state.ObjectState, hashCode()=187723824, toString()=AbstractState [id=PRE\_AUTH, pseudoState=null, deferred=[], entryActions=[], exitActions=[], stateActions=[], regions=[], submachine=null]]**

**2022-09-10 11:47:09.720 INFO 1484 --- [ main] g.s.msscssm.config.StateMachineConfig : state changed(from: ObjectState [getIds()=[PRE\_AUTH], getClass()=class org.springframework.statemachine.state.ObjectState, hashCode()=187723824, toString()=AbstractState [id=PRE\_AUTH, pseudoState=null, deferred=[], entryActions=[], exitActions=[], stateActions=[], regions=[], submachine=null]], to:ObjectState [getIds()=[PRE\_AUTH\_ERROR], getClass()=class org.springframework.statemachine.state.ObjectState, hashCode()=641691286, toString()=AbstractState [id=PRE\_AUTH\_ERROR, pseudoState=org.springframework.statemachine.state.DefaultPseudoState@49741e80, deferred=[], entryActions=[], exitActions=[], stateActions=[], regions=[], submachine=null]])**

**ObjectState [getIds()=[PRE\_AUTH\_ERROR], getClass()=class org.springframework.statemachine.state.ObjectState, hashCode()=641691286, toString()=AbstractState [id=PRE\_AUTH\_ERROR, pseudoState=org.springframework.statemachine.state.DefaultPseudoState@49741e80, deferred=[], entryActions=[], exitActions=[], stateActions=[], regions=[], submachine=null]]**

**Payment Service creation and initializing StateMachine from DB**

**Note**: saving states in db is costly operation (below technique)

public interface PaymentService {  
 Payment newPayment(Payment payment);  
 StateMachine<PaymentState, PaymentEvent> preAuth(Long paymentId);  
 StateMachine<PaymentState, PaymentEvent> authorizePayment(Long paymentId);  
 StateMachine<PaymentState, PaymentEvent> declineAuth(Long paymentId);  
}

@RequiredArgsConstructor  
@Service  
public class PaymentServiceImpl implements PaymentService {  
 private final PaymentRepository paymentRepository;  
 private final StateMachineFactory<PaymentState, PaymentEvent> stateMachineFactory;  
  
 @Override  
 public Payment newPayment(Payment payment) {  
 payment.setState(PaymentState.*NEW*);  
 return paymentRepository.save(payment);  
 }  
  
 @Override  
 public StateMachine<PaymentState, PaymentEvent> preAuth(Long paymentId) {  
 StateMachine<PaymentState, PaymentEvent> sm = build(paymentId);  
  
 return null;  
 }  
  
 @Override  
 public StateMachine<PaymentState, PaymentEvent> authorizePayment(Long paymentId) {  
 StateMachine<PaymentState, PaymentEvent> sm = build(paymentId);  
  
 return null;  
 }  
  
 @Override  
 public StateMachine<PaymentState, PaymentEvent> declineAuth(Long paymentId) {  
 StateMachine<PaymentState, PaymentEvent> sm = build(paymentId);  
  
 return null;  
 }  
  
 //resetting state from db  
 private StateMachine<PaymentState, PaymentEvent> build(Long paymentId){  
 Payment payment = paymentRepository.getOne(paymentId);  
 StateMachine<PaymentState, PaymentEvent> sm = stateMachineFactory.getStateMachine(Long.*toString*(payment.getId()));  
 sm.stop(); //depricated  
 sm.getStateMachineAccessor()  
 .doWithAllRegions(sma -> {

//depricated  
 sma.resetStateMachine(new DefaultStateMachineContext<>(payment.getState(), null, null, null));  
 });  
 sm.start(); //depricated  
 return sm;  
 }  
}

**Sending Event to the StateMachine And Intercepting the StateMachine**

@RequiredArgsConstructor  
@Service  
public class PaymentServiceImpl implements PaymentService {

public static final String *PAYMENT\_ID\_HEADER* = "payment\_id";  
  
 private final PaymentRepository paymentRepository;  
 private final StateMachineFactory<PaymentState, PaymentEvent> stateMachineFactory;

private final PaymentStateChangeInterceptor paymentStateChangeInterceptor;

@Override  
 public Payment newPayment(Payment payment) {  
 payment.setState(PaymentState.*NEW*);  
 return paymentRepository.save(payment);  
 }  
 @Transactional  
 @Override  
 public StateMachine<PaymentState, PaymentEvent> preAuth(Long paymentId) {  
 StateMachine<PaymentState, PaymentEvent> sm = build(paymentId);  
  
 sendEvent(paymentId, sm, PaymentEvent.*PRE\_AUTH\_APPROVED)*;  
  
 return sm;  
 }  
 @Transactional  
 @Override  
 public StateMachine<PaymentState, PaymentEvent> authorizePayment(Long paymentId) {  
 StateMachine<PaymentState, PaymentEvent> sm = build(paymentId);  
  
 sendEvent(paymentId, sm, PaymentEvent.*AUTHORIZE*);  
  
 return sm;  
 }

@Transactional  
 @Override  
 public StateMachine<PaymentState, PaymentEvent> declineAuth(Long paymentId) {  
 StateMachine<PaymentState, PaymentEvent> sm = build(paymentId);  
  
 sendEvent(paymentId, sm, PaymentEvent.*AUTH\_DECLINED*);  
  
 return sm;  
 }  
  
 private void sendEvent(Long paymentId, StateMachine<PaymentState, PaymentEvent> sm, PaymentEvent event){  
 Message msg = MessageBuilder.*withPayload*(event)  
 .setHeader(*PAYMENT\_ID\_HEADER*, paymentId)  
 .build();  
  
 sm.sendEvent(msg);  
 }  
  
 //resetting state from db and intercepting  
 private StateMachine<PaymentState, PaymentEvent> build(Long paymentId){  
 Payment payment = paymentRepository.getOne(paymentId);  
 StateMachine<PaymentState, PaymentEvent> sm = stateMachineFactory.getStateMachine(Long.*toString*(payment.getId()));  
 sm.stop();  
 sm.getStateMachineAccessor()  
 .doWithAllRegions(sma -> {

sma.addStateMachineInterceptor(paymentStateChangeInterceptor);  
 sma.resetStateMachine(new DefaultStateMachineContext<>(payment.getState(), null, null, null));  
 });  
 sm.start();  
 return sm;  
 }  
}

Img:paymentServiceImpl

**Intercepting**

package guru.springframework.msscssm.services;

@RequiredArgsConstructor  
@Component  
public class PaymentStateChangeInterceptor extends StateMachineInterceptorAdapter<PaymentState, PaymentEvent> {  
  
 private final PaymentRepository paymentRepository;  
  
 @Override  
 public void preStateChange(State<PaymentState, PaymentEvent> state, Message<PaymentEvent> message, Transition<PaymentState, PaymentEvent> transition, StateMachine<PaymentState, PaymentEvent> stateMachine, StateMachine<PaymentState, PaymentEvent> rootStateMachine) {  
 Optional.*ofNullable*(message).ifPresent(msg -> {  
 Optional.*ofNullable*(Long.class.cast(msg.getHeaders().getOrDefault(PaymentServiceImpl.*PAYMENT\_ID\_HEADER*, -1L)))  
 .ifPresent(paymentId -> {  
 Payment payment = paymentRepository.getOne(paymentId);  
 payment.setState(state.getId());  
 paymentRepository.save(payment);  
 });  
 });  
 }  
}

**2.1 Changing State from NEW to PRE\_AUTH**

Test cases

@SpringBootTest  
class PaymentServiceImplTest {  
  
 @Autowired  
 PaymentService paymentService;  
  
 @Autowired  
 PaymentRepository paymentRepository;  
  
 Payment payment;  
  
 @BeforeEach  
 void setUp() {  
 payment = Payment.*builder*().amount(new BigDecimal("12.99")).build(); //state NEW will be setting in service layer  
 }  
  
 @Transactional //otherwise we get lazyloading errors  
 @Test  
 void preAuth() {  
 Payment savedPayment = paymentService.newPayment(payment);  
  
 System.*out*.println("Should be NEW");  
 System.*out*.println(savedPayment.getState());  
  
 StateMachine<PaymentState, PaymentEvent> sm = paymentService.preAuth(savedPayment.getId());  
  
 Payment preAuthedPayment = paymentRepository.getOne(savedPayment.getId());  
  
 System.*out*.println("Should be PRE\_AUTH");  
 System.*out*.println(sm.getState().getId());  
  
 System.*out*.println(preAuthedPayment);  
 }  
}

Img: PaymentServiceImplTest

**O/P:**

Should be NEW

NEW

Should be PRE\_AUTH

PRE\_AUTH

Payment(id=1, state=PRE\_AUTH, amount=12.99)

**State Machine Actions**

Create a preAuth action method which updates the states based on dummy if condition Img:stateMachineConfig

@Slf4j  
@EnableStateMachineFactory  
@Configuration  
public class StateMachineConfig extends StateMachineConfigurerAdapter<PaymentState, PaymentEvent> {  
  
 //States  
 @Override  
 public void configure(StateMachineStateConfigurer<PaymentState, PaymentEvent> states) throws Exception {  
 states.withStates()  
 .initial(PaymentState.*NEW*)  
 .states(EnumSet.*allOf*(PaymentState.class))  
 .end(PaymentState.*AUTH*)  
 .end(PaymentState.*PRE\_AUTH\_ERROR*)  
 .end(PaymentState.*AUTH\_ERROR*);  
 }  
 //transitions  
 @Override  
 public void configure(StateMachineTransitionConfigurer<PaymentState, PaymentEvent> transitions) throws Exception {  
 transitions.withExternal().source(PaymentState.*NEW*).target(PaymentState.*NEW*).event(PaymentEvent.*PRE\_AUTHORIZE*)

.action(preAuthAction())  
 .and()  
 .withExternal().source(PaymentState.*NEW*).target(PaymentState.*PRE\_AUTH*).event(PaymentEvent.*PRE\_AUTH\_APPROVED*)  
 .and()  
 .withExternal().source(PaymentState.*PRE\_AUTH*).target(PaymentState.*PRE\_AUTH\_ERROR*).event(PaymentEvent.*PRE\_AUTH\_DECLINED*)  
 .and()  
 .withExternal().source(PaymentState.*NEW*).target(PaymentState.*PRE\_AUTH\_ERROR*).event(PaymentEvent.*PRE\_AUTH\_DECLINED*);  
 }  
  
 //logging state change  
 @Override  
 public void configure(StateMachineConfigurationConfigurer<PaymentState, PaymentEvent> config) throws Exception {  
 StateMachineListenerAdapter<PaymentState, PaymentEvent> adapter = new StateMachineListenerAdapter<>(){  
 @Override  
 public void stateChanged(State<PaymentState, PaymentEvent> from, State<PaymentState, PaymentEvent> to) {  
 *log*.info(String.*format*("state changed(from: %s, to:%s)",from,to));  
 }  
 };  
 config.withConfiguration().listener(adapter);  
 }

**-----------------------------------------------------------------------**  
 // preauth actions  
 public Action<PaymentState, PaymentEvent> preAuthAction(){  
 return context -> {  
 System.*out*.println("PreAuth was called!!!");  
  
 if (new Random().nextInt(10) < 8) {  
 System.*out*.println("Approved");  
 context.getStateMachine().sendEvent(MessageBuilder.*withPayload*(PaymentEvent.*PRE\_AUTH\_APPROVED*)  
 .setHeader(PaymentServiceImpl.*PAYMENT\_ID\_HEADER*, context.getMessageHeader(PaymentServiceImpl.*PAYMENT\_ID\_HEADER*))  
 .build());  
 } else {  
 System.*out*.println("Declined! No Credit!!!!!!");  
 context.getStateMachine().sendEvent(MessageBuilder.*withPayload*(PaymentEvent.*PRE\_AUTH\_DECLINED*)  
 .setHeader(PaymentServiceImpl.*PAYMENT\_ID\_HEADER*, context.getMessageHeader(PaymentServiceImpl.*PAYMENT\_ID\_HEADER*))  
 .build());  
 }  
 };  
 }

**------------------------------------------------------------------------  
}**

Run the test cases Img: PaymentServiceImplTest //change println statement Should be System.*out*.println("Should be PRE\_AUTH or PRE\_AUTH\_ERROR ");

**O/P:**

Should be NEW

NEW

PreAuth was called!!!

Approved

Should be PRE\_AUTH or PRE\_AUTH\_ERROR

PRE\_AUTH

Payment(id=1, state=PRE\_AUTH, amount=12.99)

**2.2 Changing State from PRE\_AUTH to AUTH**

@Slf4j  
@EnableStateMachineFactory  
@Configuration  
public class StateMachineConfig extends StateMachineConfigurerAdapter<PaymentState, PaymentEvent> {  
  
 //States  
 @Override  
 public void configure(StateMachineStateConfigurer<PaymentState, PaymentEvent> states) throws Exception {  
 states.withStates()  
 .initial(PaymentState.*NEW*)  
 .states(EnumSet.*allOf*(PaymentState.class))  
 .end(PaymentState.*AUTH*)  
 .end(PaymentState.*PRE\_AUTH\_ERROR*)  
 .end(PaymentState.*AUTH\_ERROR*);  
 }  
 //transitions  
 @Override  
 public void configure(StateMachineTransitionConfigurer<PaymentState, PaymentEvent> transitions) throws Exception {  
 transitions  
 //NEW to PRE\_AUTH  
.withExternal().source(PaymentState.*NEW*).target(PaymentState.*NEW*).event(PaymentEvent.*PRE\_AUTHORIZE*).action(preAuthAction())  
 .and()  
 .withExternal().source(PaymentState.*NEW*).target(PaymentState.*PRE\_AUTH*).event(PaymentEvent.*PRE\_AUTH\_APPROVED*)  
 .and()  
 .withExternal().source(PaymentState.*PRE\_AUTH*).target(PaymentState.*PRE\_AUTH\_ERROR*).event(PaymentEvent.*PRE\_AUTH\_DECLINED*)  
 .and()  
 //PRE\_AUTH to AUTH ----  
.withExternal().source(PaymentState.*PRE\_AUTH*).target(PaymentState.*PRE\_AUTH*).event(PaymentEvent.*AUTHORIZE*).action(authAction())  
 .and()  
 .withExternal().source(PaymentState.*PRE\_AUTH*).target(PaymentState.*AUTH*).event(PaymentEvent.*AUTH\_APPROVED*)  
 .and()  
 .withExternal().source(PaymentState.*PRE\_AUTH*).target(PaymentState.*AUTH\_ERROR*).event(PaymentEvent.*AUTH\_DECLINED*);  
 }

-----  
  
 //logging state change  
 @Override  
 public void configure(StateMachineConfigurationConfigurer<PaymentState, PaymentEvent> config) throws Exception {  
 StateMachineListenerAdapter<PaymentState, PaymentEvent> adapter = new StateMachineListenerAdapter<>(){  
 @Override  
 public void stateChanged(State<PaymentState, PaymentEvent> from, State<PaymentState, PaymentEvent> to) {  
 *log*.info(String.*format*("state changed(from: %s, to:%s)",from,to));  
 }  
 };  
 config.withConfiguration().listener(adapter);  
 }  
 // preauth actions  
 public Action<PaymentState, PaymentEvent> preAuthAction(){  
 return context -> {  
 System.*out*.println("PreAuth was called!!!");  
  
 if (new Random().nextInt(10) < 8) {  
 System.*out*.println("Approved");  
 context.getStateMachine().sendEvent(MessageBuilder.*withPayload*(PaymentEvent.*PRE\_AUTH\_APPROVED*)  
 .setHeader(PaymentServiceImpl.*PAYMENT\_ID\_HEADER*, context.getMessageHeader(PaymentServiceImpl.*PAYMENT\_ID\_HEADER*))  
 .build());  
 } else {  
 System.*out*.println("Declined! No Credit!!!!!!");  
 context.getStateMachine().sendEvent(MessageBuilder.*withPayload*(PaymentEvent.*PRE\_AUTH\_DECLINED*)  
 .setHeader(PaymentServiceImpl.*PAYMENT\_ID\_HEADER*, context.getMessageHeader(PaymentServiceImpl.*PAYMENT\_ID\_HEADER*))  
 .build());  
 }  
 };  
 }  
**-----------------------------------------------------------------------**  
 //Auth action  
 public Action<PaymentState, PaymentEvent> authAction(){  
 return context -> {  
 System.*out*.println("Auth was called!!!");  
  
 if (new Random().nextInt(10) < 8) {  
 System.*out*.println("Auth Approved");  
 context.getStateMachine().sendEvent(MessageBuilder.*withPayload*(PaymentEvent.*AUTH\_APPROVED*)  
 .setHeader(PaymentServiceImpl.*PAYMENT\_ID\_HEADER*, context.getMessageHeader(PaymentServiceImpl.*PAYMENT\_ID\_HEADER*))  
 .build());  
  
 } else {  
 System.*out*.println("Auth Declined! No Credit!!!!!!");  
 context.getStateMachine().sendEvent(MessageBuilder.*withPayload*(PaymentEvent.*AUTH\_DECLINED*)  
 .setHeader(PaymentServiceImpl.*PAYMENT\_ID\_HEADER*, context.getMessageHeader(PaymentServiceImpl.*PAYMENT\_ID\_HEADER*))  
 .build());  
 }  
 };  
 }

**-----------------------------------------------------------------------**  
}

Write testcases:

@SpringBootTest  
class PaymentServiceImplTest {  
 @Autowired  
 PaymentService paymentService;  
 @Autowired  
 PaymentRepository paymentRepository;  
  
 Payment payment;  
  
 @BeforeEach  
 void setUp() {  
 payment = Payment.*builder*().amount(new BigDecimal("12.99")).build();  
 }

@Transactional  
 @Test  
 void preAuth() {  
 Payment savedPayment = paymentService.newPayment(payment);

System.*out*.println("Should be NEW");  
 System.*out*.println(savedPayment.getState());  
 StateMachine<PaymentState, PaymentEvent> sm = paymentService.preAuth(savedPayment.getId());  
 Payment preAuthedPayment = paymentRepository.getOne(savedPayment.getId());  
  
 System.*out*.println("Should be PRE\_AUTH or PRE\_AUTH\_ERROR");  
 System.*out*.println(sm.getState().getId());  
  
 System.*out*.println(preAuthedPayment);  
 }  
**-----------------------------------------------------------------------RUN THIS TESTCASE**  
 @Transactional  
 @RepeatedTest(10) //10 test case will run  
 void testAuth() {  
 Payment savedPayment = paymentService.newPayment(payment);  
 // calling preauth service  
 StateMachine<PaymentState, PaymentEvent> preAuthSM = paymentService.preAuth(savedPayment.getId());  
  
 if (preAuthSM.getState().getId() == PaymentState.*PRE\_AUTH*) {  
 System.*out*.println("Payment is Pre Authorized");  
  
 StateMachine<PaymentState, PaymentEvent> authSM = paymentService.authorizePayment(savedPayment.getId());  
  
 System.*out*.println("Result of Auth: " + authSM.getState().getId());  
 } else {  
 System.*out*.println("Payment failed pre-auth...");  
 }  
 }

**-----------------------------------------------------------------------**  
}

**O/P:**

**Testcase1:**

PreAuth was called!!!

Approved

Payment is Pre Authorized

Auth was called!!!

Auth Declined! No Credit!!!!!!

Result of Auth: AUTH\_ERROR

**Testcase2**:

PreAuth was called!!!

Approved

Payment is Pre Authorized

Auth was called!!!

Auth Approved

Result of Auth: AUTH

**Testcase3…10**

**2.3 State Machine Guards (guard the State based on condition)**

@Slf4j  
@EnableStateMachineFactory  
@Configuration  
public class StateMachineConfig extends StateMachineConfigurerAdapter<PaymentState, PaymentEvent> {  
  
 //States  
 @Override  
 public void configure(StateMachineStateConfigurer<PaymentState, PaymentEvent> states) throws Exception {  
 states.withStates()  
 .initial(PaymentState.*NEW*)  
 .states(EnumSet.*allOf*(PaymentState.class))  
 .end(PaymentState.*AUTH*)  
 .end(PaymentState.*PRE\_AUTH\_ERROR*)  
 .end(PaymentState.*AUTH\_ERROR*);  
 }  
 //transitions  
 @Override  
 public void configure(StateMachineTransitionConfigurer<PaymentState, PaymentEvent> transitions) throws Exception {  
 transitions  
 //NEW to PRE\_AUTH  
 .withExternal().source(PaymentState.*NEW*).target(PaymentState.*NEW*).event(PaymentEvent.*PRE\_AUTHORIZE*).action(preAuthAction())

.guard(paymentIdGuard())  
 .and()  
 .withExternal().source(PaymentState.*NEW*).target(PaymentState.*PRE\_AUTH*).event(PaymentEvent.*PRE\_AUTH\_APPROVED*)  
 .and()  
 .withExternal().source(PaymentState.*PRE\_AUTH*).target(PaymentState.*PRE\_AUTH\_ERROR*).event(PaymentEvent.*PRE\_AUTH\_DECLINED*)  
 .and()  
 //PRE\_AUTH to AUTH  
 .withExternal().source(PaymentState.*PRE\_AUTH*).target(PaymentState.*PRE\_AUTH*).event(PaymentEvent.*AUTHORIZE*).action(authAction())  
 .and()  
 .withExternal().source(PaymentState.*PRE\_AUTH*).target(PaymentState.*AUTH*).event(PaymentEvent.*AUTH\_APPROVED*)  
 .and()  
 .withExternal().source(PaymentState.*PRE\_AUTH*).target(PaymentState.*AUTH\_ERROR*).event(PaymentEvent.*AUTH\_DECLINED*);  
 }  
  
 //logging state change  
 @Override  
 public void configure(StateMachineConfigurationConfigurer<PaymentState, PaymentEvent> config) throws Exception {  
 StateMachineListenerAdapter<PaymentState, PaymentEvent> adapter = new StateMachineListenerAdapter<>(){  
 @Override  
 public void stateChanged(State<PaymentState, PaymentEvent> from, State<PaymentState, PaymentEvent> to) {  
 *log*.info(String.*format*("state changed(from: %s, to:%s)",from,to));  
 }  
 };  
 config.withConfiguration().listener(adapter);  
 }  
**----------------------------------------------------------------------- only if paymentHeader is present then state will change**  
 public Guard<PaymentState, PaymentEvent> paymentIdGuard(){  
 return context -> {  
 return context.getMessageHeader(PaymentServiceImpl.*PAYMENT\_ID\_HEADER*) != null; //will return true or false  
 };  
 }

**-----------------------------------------------------------------------**  
 // preauth actions  
 public Action<PaymentState, PaymentEvent> preAuthAction(){ ‘’’’’’’

//Auth action  
 public Action<PaymentState, PaymentEvent> authAction(){ ‘’’’’’’

“””””””””””””””refer above images for full code””””””””””””””””””

**2.4 Creating more actions and refactoring code**

Create preAuthAction & authAction in separate component. Create 4 more actions, AuthApprovedAction, AuthDeclinedAction, preAuthApprovedAction, preAuthDeclinedAction. (all are under same packge “actions” inside “config”) and inject it to StateMachineConfig.java class.

preAuthAction

package guru.springframework.msscssm.config.actions;

@Component  
public class PreAuthAction implements Action<PaymentState, PaymentEvent> {  
 @Override  
 public void execute(StateContext<PaymentState, PaymentEvent> stateContext) {  
 System.*out*.println("PreAuth was called!!!");  
  
 if (new Random().nextInt(10) < 8) {  
 System.*out*.println("Approved");  
 stateContext.getStateMachine().sendEvent(MessageBuilder.*withPayload*(PaymentEvent.*PRE\_AUTH\_APPROVED*)  
 .setHeader(PaymentServiceImpl.*PAYMENT\_ID\_HEADER*, stateContext.getMessageHeader(PaymentServiceImpl.*PAYMENT\_ID\_HEADER*))  
 .build());  
 } else {  
 System.*out*.println("Declined! No Credit!!!!!!");  
 stateContext.getStateMachine().sendEvent(MessageBuilder.*withPayload*(PaymentEvent.*PRE\_AUTH\_DECLINED*)  
 .setHeader(PaymentServiceImpl.*PAYMENT\_ID\_HEADER*, stateContext.getMessageHeader(PaymentServiceImpl.*PAYMENT\_ID\_HEADER*))  
 .build());  
 }  
 }  
}

authAction.

package guru.springframework.msscssm.config.actions;

@Component  
public class AuthAction implements Action<PaymentState, PaymentEvent> {  
 @Override  
 public void execute(StateContext<PaymentState, PaymentEvent> stateContext) {  
 System.*out*.println("Auth was called!!!");  
  
 if (new Random().nextInt(10) < 8) {  
 System.*out*.println("Auth Approved");  
 stateContext.getStateMachine().sendEvent(MessageBuilder.*withPayload*(PaymentEvent.*AUTH\_APPROVED*)  
 .setHeader(PaymentServiceImpl.*PAYMENT\_ID\_HEADER*, stateContext.getMessageHeader(PaymentServiceImpl.*PAYMENT\_ID\_HEADER*))  
 .build());  
  
 } else {  
 System.*out*.println("Auth Declined! No Credit!!!!!!");  
 stateContext.getStateMachine().sendEvent(MessageBuilder.*withPayload*(PaymentEvent.*AUTH\_DECLINED*)  
 .setHeader(PaymentServiceImpl.*PAYMENT\_ID\_HEADER*, stateContext.getMessageHeader(PaymentServiceImpl.*PAYMENT\_ID\_HEADER*))  
 .build());  
 }  
 }  
}

AuthApprovedAction

package guru.springframework.msscssm.config.actions;  
  
@Component  
public class AuthApprovedAction implements Action<PaymentState, PaymentEvent> {  
 @Override  
 public void execute(StateContext<PaymentState, PaymentEvent> stateContext) {  
 System.*out*.println("Sending Notification of Auth approved");  
 }  
}

AuthDeclinedAction

@Component  
public class AuthDeclinedAction implements Action<PaymentState, PaymentEvent> {  
 @Override  
 public void execute(StateContext<PaymentState, PaymentEvent> stateContext) {  
 System.*out*.println("Sending Notification of Auth Declined");  
 }  
}

preAuthApprovedAction

@Component  
public class PreAuthApprovedAction implements Action<PaymentState, PaymentEvent> {  
 @Override  
 public void execute(StateContext<PaymentState, PaymentEvent> stateContext) {  
 System.*out*.println("Sending Notifaction of PreAuth Approved");  
 }  
}

preAuthDeclinedAction

@Component  
public class PreAuthDeclinedAction implements Action<PaymentState, PaymentEvent> {  
 @Override  
 public void execute(StateContext<PaymentState, PaymentEvent> stateContext) {  
 System.*out*.println("Sending Notifaction of PreAuth Declined");  
 }  
}

StateMachineConfig

@Slf4j  
@EnableStateMachineFactory  
@RequiredArgsConstructor  
@Configuration  
public class StateMachineConfig extends StateMachineConfigurerAdapter<PaymentState, PaymentEvent> {

**-----------------------------------------------------------------------**  
 private final Action<PaymentState, PaymentEvent> preAuthAction;  
 private final Action<PaymentState, PaymentEvent> authAction;  
 private final Action<PaymentState, PaymentEvent> preAuthApprovedAction;  
 private final Action<PaymentState, PaymentEvent> preAuthDeclinedAction;  
 private final Action<PaymentState, PaymentEvent> authApprovedAction;  
 private final Action<PaymentState, PaymentEvent> authDeclinedAction;  
 private final Guard<PaymentState, PaymentEvent> paymentIdGuard;  
**-----------------------------------------------------------------------**

//States  
 @Override  
 public void configure(StateMachineStateConfigurer<PaymentState, PaymentEvent> states) throws Exception {  
 states.withStates()  
 .initial(PaymentState.*NEW*)  
 .states(EnumSet.*allOf*(PaymentState.class))  
 .end(PaymentState.*AUTH*)  
 .end(PaymentState.*PRE\_AUTH\_ERROR*)  
 .end(PaymentState.*AUTH\_ERROR*);  
 }  
 //transitions  
 @Override  
 public void configure(StateMachineTransitionConfigurer<PaymentState, PaymentEvent> transitions) throws Exception {  
 transitions  
 //NEW to PRE\_AUTH  
 .withExternal().source(PaymentState.*NEW*).target(PaymentState.*NEW*).event(PaymentEvent.*PRE\_AUTHORIZE*).action(preAuthAction)

.guard(paymentIdGuard)  
 .and()  
 .withExternal().source(PaymentState.*NEW*).target(PaymentState.*PRE\_AUTH*).event(PaymentEvent.*PRE\_AUTH\_APPROVED*)

.action(preAuthApprovedAction)  
 .and()  
 .withExternal().source(PaymentState.*PRE\_AUTH*).target(PaymentState.*PRE\_AUTH\_ERROR*).event(PaymentEvent.*PRE\_AUTH\_DECLINED*)

.action(preAuthDeclinedAction)  
 .and()  
 //PRE\_AUTH to AUTH  
 .withExternal().source(PaymentState.*PRE\_AUTH*).target(PaymentState.*PRE\_AUTH*).event(PaymentEvent.*AUTHORIZE*).action(authAction)  
 .and()  
 .withExternal().source(PaymentState.*PRE\_AUTH*).target(PaymentState.*AUTH*).event(PaymentEvent.*AUTH\_APPROVED*).

action(authApprovedAction)  
 .and()  
 .withExternal().source(PaymentState.*PRE\_AUTH*).target(PaymentState.*AUTH\_ERROR*).event(PaymentEvent.*AUTH\_DECLINED*)

.action(authDeclinedAction);  
 }  
  
 //logging state change  
 @Override  
 public void configure(StateMachineConfigurationConfigurer<PaymentState, PaymentEvent> config) throws Exception {  
 StateMachineListenerAdapter<PaymentState, PaymentEvent> adapter = new StateMachineListenerAdapter<>(){  
 @Override  
 public void stateChanged(State<PaymentState, PaymentEvent> from, State<PaymentState, PaymentEvent> to) {  
 *log*.info(String.*format*("state changed(from: %s, to:%s)",from,to));  
 }  
 };  
 config.withConfiguration().listener(adapter);  
 }

}

**O/P:**

PreAuth was called!!!

Approved

Sending Notifaction of PreAuth Approved

Payment is Pre Authorized

Auth was called!!!

Auth Approved

Sending Notification of Auth approved

Result of Auth: AUTH

-Design Pattern - Event Sourcing and block chain talk vid:215

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**3.0 Sagas** vid: 217

Series of steps with componsating transactions.

**Transaction**: What, Where, Uses, Problem with Transaction In Micro Services, JTA (Java Transaction Api) and its problems.. refer [theProbWithTransactions](pdf-notes/TheProbWithTransactions.pdf) .

**\*Need for SAGAS**: [needForSagas](pdf-notes/TheNeedForSagas.pdf) .

\***SAGA Coordination**: [sagaCoordination](pdf-notes/SagaCoordination.pdf) (types of sagas).

\***OrderAllocation SAGA** : [orderAllocationSaga](pdf-notes/OrderAllocationSaga.pdf) .

-Updating Project to Maven Central vid:222

**Implementation in Project**

***Service: mssc-beer-order-service-2***

1.Create enum classes for maintaining States and Events.

package guru.sfg.beer.order.service.domain;

public enum BeerOrderStatusEnum {  
 *NEW*, *VALIDATED*, *VALIDATION\_PENDING*, *VALIDATION\_EXCEPTION*,  
 *ALLOCATION\_PENDING*, *ALLOCATED*, *ALLOCATION\_EXCEPTION*, *CANCELLED*,  
 *PENDING\_INVENTORY*, *PICKED\_UP*, *DELIVERED*, *DELIVERY\_EXCEPTION*}

public enum BeerOrderEventEnum {  
 *VALIDATE\_ORDER*, *CANCEL\_ORDER*, *VALIDATION\_PASSED*, *VALIDATION\_FAILED*,  
 *ALLOCATE\_ORDER*, *ALLOCATION\_SUCCESS*, *ALLOCATION\_NO\_INVENTORY*, *ALLOCATION\_FAILED*,  
 *BEERORDER\_PICKED\_UP*}

2. Create SpringStateMachine Configure class

@Configuration  
@EnableStateMachineFactory  
public class BeerOrderStateMachineConfig extends StateMachineConfigurerAdapter<BeerOrderStatusEnum, BeerOrderEventEnum> {  
  
 @Override  
 public void configure(StateMachineStateConfigurer<BeerOrderStatusEnum, BeerOrderEventEnum> states) throws Exception {  
 states.withStates()  
 .initial(BeerOrderStatusEnum.*NEW*)  
 .states(EnumSet.*allOf*(BeerOrderStatusEnum.class))  
 .end(BeerOrderStatusEnum.*PICKED\_UP*)  
 .end(BeerOrderStatusEnum.*DELIVERED*)  
 .end(BeerOrderStatusEnum.*CANCELLED*)  
 .end(BeerOrderStatusEnum.*DELIVERY\_EXCEPTION*)  
 .end(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)  
 .end(BeerOrderStatusEnum.*ALLOCATION\_EXCEPTION*);  
 }  
}

**Note: when you clean install this project we will get an error**

Error!!!: **The following constants from the property "guru.sfg.beer.order.service.domain.BeerOrderStatusEnum orderStatus" enum have no corresponding constant in the "guru.sfg.beer.order.service.web.model.OrderStatusEnum orderStatus" enum and must be be mapped via adding additional mappings: VALIDATED, VALIDATION\_PENDING, VALIDATION\_EXCEPTION, ALLOCATION\_PENDING, ALLOCATED, ALLOCATION\_EXCEPTION, CANCELLED, PENDING\_INVENTORY, DELIVERED, DELIVERY\_EXCEPTION.**

Reason:In web>model>BeerOrderDto we are using web>model>OrderStatusEnum as DataType.

package guru.sfg.beer.order.service.web.model;  
  
public enum OrderStatusEnum {  
 *NEW*, *READY*, *PICKED\_UP*}

@Data  
@NoArgsConstructor  
@EqualsAndHashCode(callSuper = true)  
public class BeerOrderDto extends BaseItem {  
  
 @Builder  
 public BeerOrderDto(UUID id, Integer version, OffsetDateTime createdDate, OffsetDateTime lastModifiedDate, UUID customerId, List<BeerOrderLineDto> beerOrderLines,  
 OrderStatusEnum orderStatus, String orderStatusCallbackUrl, String customerRef) {  
 super(id, version, createdDate, lastModifiedDate);  
 this.customerId = customerId;  
 this.beerOrderLines = beerOrderLines;  
 this.orderStatus = orderStatus;  
 this.orderStatusCallbackUrl = orderStatusCallbackUrl;  
 this.customerRef = customerRef;  
 }  
 private OrderStatusEnum orderStatus;

“””””””””  
}

Earlier we have refactored domain>OrderStatusEnum to BeerOrderStatusEnum, so web>model>OrderStatusEnum is not happy about that.

Mastruct generated code doesnot know how to handle this.

package guru.sfg.beer.order.service.web.mappers;

@Mapper(uses = {DateMapper.class, BeerOrderLineMapper.class})  
public interface BeerOrderMapper {  
 @Mapping(target = "customerId", source = "customer.id")  
 BeerOrderDto beerOrderToDto(BeerOrder beerOrder);  
  
 BeerOrder dtoToBeerOrder(BeerOrderDto dto);  
}

Solution:Either we can add all the Status to web>model>OrderStatusEnum from domain> BeerOrderStatusEnum

OR

We can Change its DataType to String

@Data  
@NoArgsConstructor  
@EqualsAndHashCode(callSuper = true)  
public class BeerOrderDto extends BaseItem {  
  
 “”””””””””  
 private String orderStatus;

“””””””””  
}

Suggestions: Its better to use DataType as String in the DTO instead of Enum (Validation advantage, can give better error message…etc).

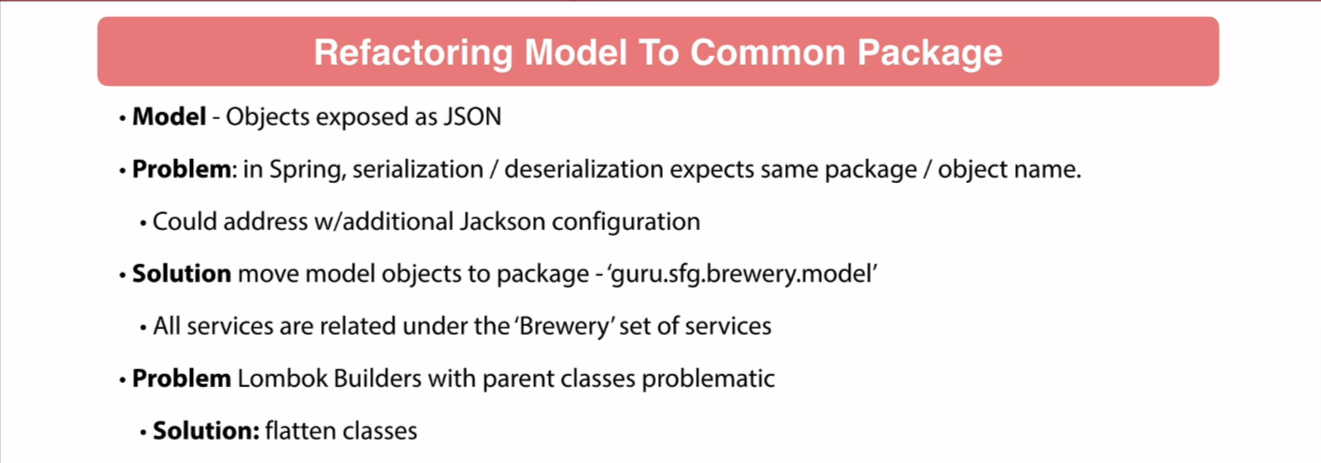
3.Create Transitions in SSM Config class

package guru.sfg.beer.order.service.sm;

@Configuration  
@EnableStateMachineFactory  
public class BeerOrderStateMachineConfig extends StateMachineConfigurerAdapter<BeerOrderStatusEnum, BeerOrderEventEnum> {  
  
 @Override  
 public void configure(StateMachineStateConfigurer<BeerOrderStatusEnum, BeerOrderEventEnum> states) throws Exception {  
 states.withStates()  
 .initial(BeerOrderStatusEnum.*NEW*)  
 .states(EnumSet.*allOf*(BeerOrderStatusEnum.class))  
 .end(BeerOrderStatusEnum.*PICKED\_UP*)  
 .end(BeerOrderStatusEnum.*DELIVERED*)  
 .end(BeerOrderStatusEnum.*CANCELLED*)  
 .end(BeerOrderStatusEnum.*DELIVERY\_EXCEPTION*)  
 .end(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)  
 .end(BeerOrderStatusEnum.*ALLOCATION\_EXCEPTION*);  
 }  
  
 @Override  
 public void configure(StateMachineTransitionConfigurer<BeerOrderStatusEnum,BeerOrderEventEnum> transitions)throws Exception{  
 transitions .withExternal().source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*NEW*).event(BeerOrderEventEnum.*VALIDATE\_ORDER*) //*TODO: add actions*

.and()  
.withExternal().source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATED*).event(BeerOrderEventEnum.*VALIDATION\_PASSED*) //*TODO: add actions*  
.and() .withExternal().source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*).event(BeerOrderEventEnum.*VALIDATION\_FAILED*);  
 }  
}

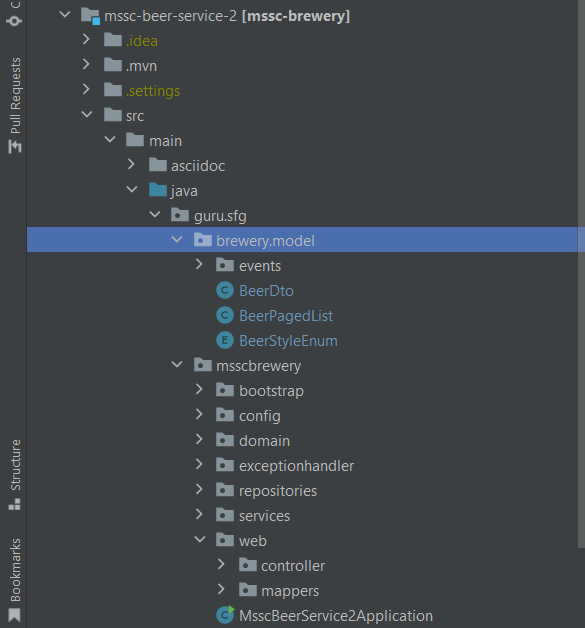
4. Refactor model to common package



Create a new package brewery.model

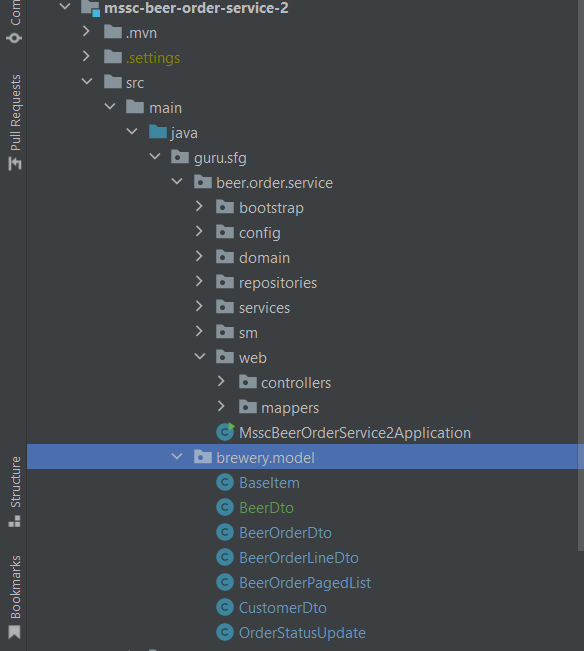
***Mssc-beer-service-2***

Move events from common>events to brewery>model>events & models from msscbrewery>web>model to brewery>model.



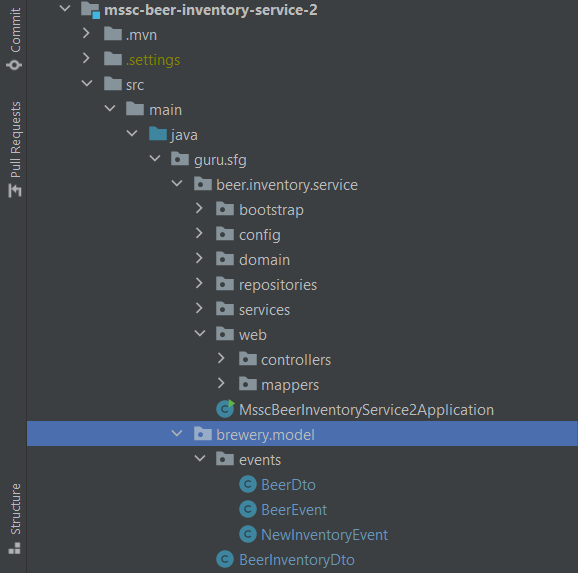
***Mssc-beer-order-service-2***

Move models form msscbrewery>web>model to brewery>model.



***Mssc-beer-inventory-service-2***

Move events from common>events to brewery>model>events & models from msscbrewery>web>model to brewery>model.



**Flattening the class**: In beer-order-service-2, inside web>model all the Classes were extending BaseEntity Class. So we will be replacing all the BaseEntity class properties to the Classes. Then delete the BaseEntity Class.

**Note:** model = DTO

5. Create BeerOrderManager Service. (sits on top of stateMachine)

public interface BeerOrderManager {  
  
 BeerOrder newBeerOrder(BeerOrder beerOrder);  
}

package guru.sfg.beer.order.service.services;  
  
import org.springframework.messaging.Message;

@RequiredArgsConstructor  
@Service  
public class BeerOrderManagerImpl implements BeerOrderManager {  
  
 private final StateMachineFactory<BeerOrderStatusEnum, BeerOrderEventEnum> stateMachineFactory;  
 private final BeerOrderRepository beerOrderRepository;

@Transactional  
 @Override  
 public BeerOrder newBeerOrder(BeerOrder beerOrder) {  
 beerOrder.setId(null);  
 beerOrder.setOrderStatus(BeerOrderStatusEnum.*NEW*);  
  
 BeerOrder savedBeerOrder = beerOrderRepository.save(beerOrder);  
 sendBeerEvent(savedBeerOrder, BeerOrderEventEnum.*VALIDATE\_ORDER*);  
  
 return savedBeerOrder;  
 }  
  
 private void sendBeerEvent(BeerOrder beerOrder, BeerOrderEventEnum eventEnum){  
 StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> sm = build(beerOrder);  
 Message msg = MessageBuilder.*withPayload*(eventEnum).build();  
 sm.sendEvent(msg);  
 }  
  
 private StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> build(BeerOrder beerOrder){  
 StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> sm = stateMachineFactory.getStateMachine(beerOrder.getId());  
 sm.stop();  
 sm.getStateMachineAccessor()  
 .doWithAllRegions(sma->{  
 sma.resetStateMachine(new DefaultStateMachineContext<>(beerOrder.getOrderStatus(),null, null,null));  
 });  
 sm.start();  
 return sm;  
 }  
}

6. Create StateChange Interceptor

package guru.sfg.beer.order.service.sm;   
  
@Slf4j  
@Component  
@RequiredArgsConstructor  
public class BeerOrderStateChangeInterceptor extends StateMachineInterceptorAdapter<BeerOrderStatusEnum, BeerOrderEventEnum> {  
  
 private final BeerOrderRepository beerOrderRepository;

@Transactional //while testing lazy initialization error was coming  
 @Override  
 public void preStateChange(State<BeerOrderStatusEnum, BeerOrderEventEnum> state, Message<BeerOrderEventEnum> message, Transition<BeerOrderStatusEnum, BeerOrderEventEnum> transition, StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> stateMachine, StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> rootStateMachine) {  
 *log*.debug("Pre-State Change");  
  
 Optional.*ofNullable*(message)  
 .flatMap(msg -> Optional.*ofNullable*((String) msg.getHeaders().getOrDefault(BeerOrderManagerImpl.*ORDER\_ID\_HEADER*, " ")))  
 .ifPresent(orderId -> {  
 *log*.debug("Saving state for order id: " + orderId + " Status: " + state.getId());  
  
 BeerOrder beerOrder = beerOrderRepository.getOne(UUID.*fromString*(orderId));  
 beerOrder.setOrderStatus(state.getId());  
 beerOrderRepository.saveAndFlush(beerOrder);  
 });  
 }  
}

package guru.sfg.beer.order.service.services;  
  
@RequiredArgsConstructor  
@Service  
public class BeerOrderManagerImpl implements BeerOrderManager {  
 public static final String *ORDER\_ID\_HEADER* = "ORDER\_ID\_HEADER" ;

private final StateMachineFactory<BeerOrderStatusEnum, BeerOrderEventEnum> stateMachineFactory;  
 private final BeerOrderRepository beerOrderRepository;  
 private final BeerOrderStateChangeInterceptor beerOrderStateChangeInterceptor;  
  
 @Transactional  
 @Override  
 public BeerOrder newBeerOrder(BeerOrder beerOrder) {  
 beerOrder.setId(null);  
 beerOrder.setOrderStatus(BeerOrderStatusEnum.*NEW*);  
  
 BeerOrder savedBeerOrder = beerOrderRepository.save(beerOrder);  
 sendBeerEvent(savedBeerOrder, BeerOrderEventEnum.*VALIDATE\_ORDER*);  
  
 return savedBeerOrder;  
 }  
  
 private void sendBeerEvent(BeerOrder beerOrder, BeerOrderEventEnum eventEnum){  
 StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> sm = build(beerOrder);  
 Message msg = MessageBuilder.*withPayload*(eventEnum)  
 .setHeader(*ORDER\_ID\_HEADER*, beerOrder.getId())  
 .build();  
 sm.sendEvent(msg);  
 }  
  
 private StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> build(BeerOrder beerOrder){  
 StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> sm = stateMachineFactory.getStateMachine(beerOrder.getId());  
 sm.stop();  
 sm.getStateMachineAccessor()  
 .doWithAllRegions(sma->{  
 sma.addStateMachineInterceptor(beerOrderStateChangeInterceptor);  
 sma.resetStateMachine(new DefaultStateMachineContext<>(beerOrder.getOrderStatus(),null, null,null));  
 });  
 sm.start();  
 return sm;  
 }  
}

7. Refactor OrderStates

Add Pending Status & AllocateOrder Event

package guru.sfg.beer.order.service.domain;  
  
public enum BeerOrderStatusEnum {  
 *NEW*, *VALIDATED*, *VALIDATION\_PENDING*, *VALIDATION\_EXCEPTION*,  
 *ALLOCATION\_PENDING*, *ALLOCATED*, *ALLOCATION\_EXCEPTION*, *CANCELLED*,  
 *PENDING\_INVENTORY*, *PICKED\_UP*, *DELIVERED*, *DELIVERY\_EXCEPTION*}

public enum BeerOrderEventEnum {  
 *VALIDATE\_ORDER*, *CANCEL\_ORDER*, *VALIDATION\_PASSED*, *VALIDATION\_FAILED*,  
 *ALLOCATE\_ORDER*, *ALLOCATION\_SUCCESS*, *ALLOCATION\_NO\_INVENTORY*, *ALLOCATION\_FAILED*,  
 *BEERORDER\_PICKED\_UP*}

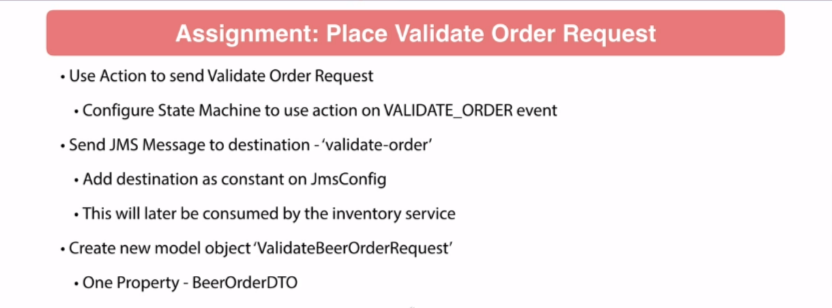
Change the configuration

package guru.sfg.beer.order.service.sm;

@Configuration  
@EnableStateMachineFactory  
public class BeerOrderStateMachineConfig extends StateMachineConfigurerAdapter<BeerOrderStatusEnum, BeerOrderEventEnum> {  
  
 @Override  
 public void configure(StateMachineStateConfigurer<BeerOrderStatusEnum, BeerOrderEventEnum> states) throws Exception {  
 states.withStates()  
 .initial(BeerOrderStatusEnum.*NEW*)  
 .states(EnumSet.*allOf*(BeerOrderStatusEnum.class))  
 .end(BeerOrderStatusEnum.*PICKED\_UP*)  
 .end(BeerOrderStatusEnum.*DELIVERED*)  
 .end(BeerOrderStatusEnum.*CANCELLED*)  
 .end(BeerOrderStatusEnum.*DELIVERY\_EXCEPTION*)  
 .end(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)  
 .end(BeerOrderStatusEnum.*ALLOCATION\_EXCEPTION*);  
 }  
  
 @Override  
 public void configure(StateMachineTransitionConfigurer<BeerOrderStatusEnum, BeerOrderEventEnum> transitions) throws Exception {  
 transitions  
 .withExternal().source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATION\_PENDING*)

.event(BeerOrderEventEnum.*VALIDATE\_ORDER*) //*TODO: add actions* .and()  
 .withExternal().source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATED*).event(BeerOrderEventEnum.*VALIDATION\_PASSED*)  
 .and()  
 .withExternal().source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*).event(BeerOrderEventEnum.*VALIDATION\_FAILED*);  
 }  
}

8. Add action for Beer Validation



8.1 add a event class to model

package guru.sfg.brewery.model.events;  
  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
@Builder  
public class ValidateOrderRequest {  
  
 private BeerOrderDto beerOrderDto;  
}

Img: ValidateOrderRequest

8.2 add destination constant to JmsConfig class

package guru.sfg.beer.order.service.config;  
  
@Configuration  
public class JmsConfig {  
  
 public static final String *VALIDATE\_ORDER\_QUEUE* = "validate-order";  
  
 //Convert Message to Object (or vice-versa)  
 @Bean  
 public MessageConverter messageConverter() {  
 MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter();  
 converter.setTargetType(MessageType.*TEXT*);  
 converter.setTypeIdPropertyName("\_type"); //need tobe set in order to convert incoming messge to java object (default is "none")  
 return converter;  
 }  
}

8.3 add action

package guru.sfg.beer.order.service.sm.actions;  
  
@Slf4j  
@Component  
@RequiredArgsConstructor  
public class ValidateOrderActions implements Action<BeerOrderStatusEnum, BeerOrderEventEnum> {  
  
 private final BeerOrderRepository beerOrderRepository;  
 private final BeerOrderMapper beerOrderMapper;  
 private final JmsTemplate jmsTemplate;  
  
 @Override  
 public void execute(StateContext<BeerOrderStatusEnum, BeerOrderEventEnum> stateContext) {  
 String beerOrderId = (String) stateContext.getMessage().getHeaders().get(BeerOrderManagerImpl.*ORDER\_ID\_HEADER*);

BeerOrder beerOrder = beerOrderRepository.getOne(UUID.*fromString*(beerOrderId));

jmsTemplate.convertAndSend(JmsConfig.*VALIDATE\_ORDER\_QUEUE*, ValidateOrderRequest.*builder*()  
 .beerOrderDto(beerOrderMapper.beerOrderToDto(beerOrder)) .build() );  
 *log*.debug("Sent Validate Order Request to queue for order id: ",beerOrderId);  
 }  
}

8.4 add action to BeerOrderStateMachineConfig

package guru.sfg.beer.order.service.sm;  
  
@Configuration  
@RequiredArgsConstructor  
@EnableStateMachineFactory  
public class BeerOrderStateMachineConfig extends StateMachineConfigurerAdapter<BeerOrderStatusEnum, BeerOrderEventEnum> {  
  
 private final Action<BeerOrderStatusEnum, BeerOrderEventEnum> ValidateOrderActions;  
  
 @Override  
 public void configure(StateMachineStateConfigurer<BeerOrderStatusEnum, BeerOrderEventEnum> states) throws Exception {  
 states.withStates()  
 .initial(BeerOrderStatusEnum.*NEW*)  
 .states(EnumSet.*allOf*(BeerOrderStatusEnum.class))  
 .end(BeerOrderStatusEnum.*PICKED\_UP*)  
 .end(BeerOrderStatusEnum.*DELIVERED*)  
 .end(BeerOrderStatusEnum.*CANCELLED*)  
 .end(BeerOrderStatusEnum.*DELIVERY\_EXCEPTION*)  
 .end(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)  
 .end(BeerOrderStatusEnum.*ALLOCATION\_EXCEPTION*);  
 }  
  
 @Override  
 public void configure(StateMachineTransitionConfigurer<BeerOrderStatusEnum, BeerOrderEventEnum> transitions) throws Exception {  
 transitions  
 .withExternal()

.source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATION\_PENDING*)

.event(BeerOrderEventEnum.*VALIDATE\_ORDER*).action(ValidateOrderActions)  
 .and()withExternal()

.source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATED*)

.event(BeerOrderEventEnum.*VALIDATION\_PASSED*)  
 .and().withExternal()

.source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)

.event(BeerOrderEventEnum.*VALIDATION\_FAILED*);  
 }  
}

**Note:** Why do we need to create objects for messages in the same package, with the same name?

We don't need to. Spring will de-serialize to a target type with Jackson. This is a fully qualified class - ie package and name. Optionally, we could use Jackson to deserialize to whatever type we wanted.

9 Add Jms Listener at **mssc-beer-service-2**

9.1 add constants to JmsConfig

package guru.sfg.msscbrewery.config;  
  
@Configuration  
public class JmsConfig {  
  
 public static final String *BREWING\_REQUEST\_QUEUE* = "brewing-request";  
 public static final String *NEW\_INVENTORY\_QUEUE* = "new-inventory";  
  
 //for beerOrder  
 public static final String *VALIDATE\_ORDER\_QUEUE* = "validate-order";  
 public static final String *VALIDATE\_ORDER\_RESPONSE\_QUEUE* = "validate-order-response";  
  
 //Strategy interface that specifies a converter between Java objects and JMS messages.  
 //Serialize message content to json using TextMessage  
 @Bean  
 public MessageConverter messageConverter(ObjectMapper objectMapper) {  
 MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter();  
 converter.setTargetType(MessageType.*TEXT*);  
 converter.setTypeIdPropertyName("\_type"); //need tobe set in order to convert incoming messge to java object (default is "none")  
 converter.setObjectMapper(objectMapper);  
 return converter;  
 }  
}

9.2 Copy BeerOrderDto, BeerOrderLineDto to brewery>model and ValidateOrderRequest(Img: ValidateOrderRequest) to brewery>model>events from **mssc-beer-order-service-2 .**

BeerOrderDto

package guru.sfg.brewery.model;   
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class BeerOrderDto {  
  
 @JsonProperty("id")  
 private UUID id = null;  
  
 @JsonProperty("version")  
 private Integer version = null;  
  
 @JsonFormat(pattern="yyyy-MM-dd'T'HH:mm:ssZ", shape=JsonFormat.Shape.*STRING*)  
 @JsonProperty("createdDate")  
 private OffsetDateTime createdDate = null;  
  
 @JsonFormat(pattern="yyyy-MM-dd'T'HH:mm:ssZ", shape=JsonFormat.Shape.*STRING*)  
 @JsonProperty("lastModifiedDate")  
 private OffsetDateTime lastModifiedDate = null;  
  
 private UUID customerId;  
 private String customerRef;  
 private List<BeerOrderLineDto> beerOrderLines;  
 private String orderStatus;  
 private String orderStatusCallbackUrl;  
}

BeerOrderLineDto

package guru.sfg.brewery.model;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class BeerOrderLineDto {  
  
 @JsonProperty("id")  
 private UUID id = null;  
  
 @JsonProperty("version")  
 private Integer version = null;  
  
 @JsonFormat(pattern="yyyy-MM-dd'T'HH:mm:ssZ", shape=JsonFormat.Shape.*STRING*)  
 @JsonProperty("createdDate")  
 private OffsetDateTime createdDate = null;  
  
 @JsonFormat(pattern="yyyy-MM-dd'T'HH:mm:ssZ", shape=JsonFormat.Shape.*STRING*)  
 @JsonProperty("lastModifiedDate")  
 private OffsetDateTime lastModifiedDate = null;  
  
 private String upc;  
 private String beerName;  
 private String beerStyle;  
 private UUID beerId;  
 private Integer orderQuantity = 0;  
 private BigDecimal price;  
}

ValidateOrderRequest

package guru.sfg.brewery.model.events;  
  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
@Builder  
public class ValidateOrderRequest {  
  
 private BeerOrderDto beerOrderDto;  
}

9.3 Create ValidateOrderResult

@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class ValidateOrderResult {  
 private UUID orderId;  
 private Boolean isValid;  
}

Img: ValidateOrderResult

9.4 Create Listener

Create a new packge order inside services.

package guru.sfg.msscbrewery.services.order;  
  
@RequiredArgsConstructor  
@Component  
public class BeerOrderValidationListeners {  
  
 private final BeerOrderValidator beerOrderValidator;  
 private final JmsTemplate jmsTemplate;  
  
 @JmsListener(destination = JmsConfig.*VALIDATE\_ORDER\_QUEUE*)  
 public void listen(ValidateOrderRequest validateOrderRequest){  
 Boolean isValid = beerOrderValidator.validateOrder(validateOrderRequest.getBeerOrderDto());  
  
 jmsTemplate.convertAndSend(JmsConfig.*VALIDATE\_ORDER\_RESPONSE\_QUEUE*, ValidateOrderResult.*builder*()  
 .orderId(validateOrderRequest.getBeerOrderDto().getId())  
 .isValid(isValid)  
 .build());  
 }  
}

9.5 Create Validator

package guru.sfg.msscbrewery.services.order;  
  
@RequiredArgsConstructor  
@Component  
public class BeerOrderValidator {  
 private final BeerRepository beerRepository;  
  
 public Boolean validateOrder(BeerOrderDto beerOrderDto) {  
 AtomicInteger beerNotFound = new AtomicInteger();  
  
 beerOrderDto.getBeerOrderLines().forEach(orderLine->{  
 if(beerRepository.findByUpc(orderLine.getUpc())==null){  
 beerNotFound.incrementAndGet();  
 }});  
 return beerNotFound.get() == 0; } }

**Note:** If the JMS Listener throws an unchecked exception, what will happen to the message on the broker?

JMS is transactional. In the event of an exception, the message will remain on the queue and available for another message consumer.

10 Add Jms Listener at mssc-beer-order-service-2 for listening to ValidateOrderResponse

10.1 add Queue constants (copied from mssc-beer-service-2)

package guru.sfg.beer.order.service.config;  
  
@Configuration  
public class JmsConfig {  
  
 public static final String *VALIDATE\_ORDER\_QUEUE* = "validate-order";  
 public static final String *VALIDATE\_ORDER\_RESPONSE\_QUEUE* = "validate-order-response";  
  
 //Convert Message to Object (or vice-versa)  
 @Bean  
 public MessageConverter messageConverter() {  
 MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter();  
 converter.setTargetType(MessageType.*TEXT*);  
 converter.setTypeIdPropertyName("\_type"); //need tobe set in order to convert incoming messge to java object (default is "none")  
 return converter;  
 }  
}

10.2 Copy ValidateOrderResult(Img: ValidateOrderResult) to brewery>model>event from mssc-beer-service-2.

ValidateOrderResult

@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class ValidateOrderResult {  
 private UUID orderId;  
 private Boolean isValid;  
}

10.3 Create a new Packge “listeners” inside service and create ValidationResultListener

package guru.sfg.beer.order.service.services.listeners;  
  
@Slf4j  
@RequiredArgsConstructor  
@Component  
public class ValidationResultListeners {  
  
 private final BeerOrderManager beerOrderManager;  
  
 @JmsListener(destination = JmsConfig.*VALIDATE\_ORDER\_RESPONSE\_QUEUE*)  
 public void listen(ValidateOrderResult result){  
 final UUID beerOrderId = result.getOrderId();  
  
 *log*.debug("Validation Result for Order Id:",beerOrderId);  
  
 beerOrderManager.processValidationResult(beerOrderId, result.getIsValid());  
 }  
}

10.4 create method for Processing Validation Result

package guru.sfg.beer.order.service.services;   
  
public interface BeerOrderManager {  
  
 BeerOrder newBeerOrder(BeerOrder beerOrder);  
  
 void processValidationResult(UUID beerOrderId, Boolean isValid);  
}

package guru.sfg.beer.order.service.services;  
  
import java.util.UUID;  
  
@RequiredArgsConstructor  
@Service  
public class BeerOrderManagerImpl implements BeerOrderManager {  
  
 public static final String *ORDER\_ID\_HEADER* = "ORDER\_ID\_HEADER" ;  
  
 private final StateMachineFactory<BeerOrderStatusEnum, BeerOrderEventEnum> stateMachineFactory;  
 private final BeerOrderRepository beerOrderRepository;  
 private final BeerOrderStateChangeInterceptor beerOrderStateChangeInterceptor;  
  
 @Transactional  
 @Override  
 public BeerOrder newBeerOrder(BeerOrder beerOrder) {  
 beerOrder.setId(null);  
 beerOrder.setOrderStatus(BeerOrderStatusEnum.*NEW*);  
  
 BeerOrder savedBeerOrder = beerOrderRepository.save(beerOrder);  
 sendBeerEvent(savedBeerOrder, BeerOrderEventEnum.*VALIDATE\_ORDER*);  
  
 return savedBeerOrder;  
 }  
  
 @Override  
 public void processValidationResult(UUID beerOrderId, Boolean isValid) {  
 BeerOrder beerOrder = beerOrderRepository.getOne(beerOrderId);  
 if(isValid){  
 sendBeerEvent(beerOrder, BeerOrderEventEnum.*VALIDATION\_PASSED*);  
 }else{  
 sendBeerEvent(beerOrder, BeerOrderEventEnum.*VALIDATE\_ORDER*);  
 }  
 }  
  
 private void sendBeerEvent(BeerOrder beerOrder, BeerOrderEventEnum eventEnum){  
 StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> sm = build(beerOrder);  
 Message msg = MessageBuilder.*withPayload*(eventEnum)  
 .setHeader(*ORDER\_ID\_HEADER*, beerOrder.getId())  
 .build();  
 sm.sendEvent(msg);  
 }  
  
 private StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> build(BeerOrder beerOrder){  
 StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> sm = stateMachineFactory.getStateMachine(beerOrder.getId());  
 sm.stop();  
 sm.getStateMachineAccessor()  
 .doWithAllRegions(sma->{  
 sma.addStateMachineInterceptor(beerOrderStateChangeInterceptor);  
 sma.resetStateMachine(new DefaultStateMachineContext<>(beerOrder.getOrderStatus(),null, null,null));  
 });  
 sm.start();  
 return sm;  
 }  
}

11 Initiate Allocation of Order

11.1 Create Constant for allocateOrder

package guru.sfg.beer.order.service.config;  
  
@Configuration  
public class JmsConfig {  
  
 public static final String *VALIDATE\_ORDER\_QUEUE* = "validate-order";  
 public static final String *VALIDATE\_ORDER\_RESPONSE\_QUEUE* = "validate-order-response";  
 public static final String *ALLOCATE\_ORDER\_QUEUE* = "allocate-order";  
  
 //Convert Message to Object (or vice-versa)  
 @Bean  
 public MessageConverter messageConverter() {  
 MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter();  
 converter.setTargetType(MessageType.*TEXT*);  
 converter.setTypeIdPropertyName("\_type"); //need tobe set in order to convert incoming messge to java object (default is "none")  
 return converter;  
 }  
}

11.2 create AllocateOrderRequest dto

@Data  
@AllArgsConstructor  
@NoArgsConstructor  
@Builder  
public class AllocateOrderRequest {  
 private BeerOrderDto beerOrderDto;  
}

11.3 Add AllocateOrder Action

package guru.sfg.beer.order.service.sm.actions;  
  
import java.util.UUID;  
  
@Slf4j  
@Component  
@RequiredArgsConstructor  
public class AllocateOrderAction implements Action<BeerOrderStatusEnum, BeerOrderEventEnum> {  
  
 private final BeerOrderRepository beerOrderRepository;  
 private final BeerOrderMapper beerOrderMapper;  
 private final JmsTemplate jmsTemplate;  
  
 @Override  
 public void execute(StateContext<BeerOrderStatusEnum, BeerOrderEventEnum> stateContext) {  
 String beerOrderId = (String) stateContext.getMessage().getHeaders().get(BeerOrderManagerImpl.*ORDER\_ID\_HEADER*);  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById(UUID.*fromString*(beerOrderId));  
  
 beerOrderOptional.ifPresentOrElse(beerOrder -> {  
 jmsTemplate.convertAndSend(JmsConfig.*ALLOCATE\_ORDER\_QUEUE*, AllocateOrderRequest.*builder*()  
 .beerOrderDto(beerOrderMapper.beerOrderToDto(beerOrder))  
 .build());  
  
 *log*.debug("Sent Allocation Request to queue for order id: ",beerOrderId);  
 }, ()-> *log*.error("Beer Order Not found!"));

}  
}

11.4 change BeerOrderStateMachineConfig (add transition from VALIDATED to ALLOCATION\_PENDING)

package guru.sfg.beer.order.service.sm;  
  
@Configuration  
@RequiredArgsConstructor  
@EnableStateMachineFactory  
public class BeerOrderStateMachineConfig extends StateMachineConfigurerAdapter<BeerOrderStatusEnum, BeerOrderEventEnum> {  
  
 private final Action<BeerOrderStatusEnum, BeerOrderEventEnum> ValidateOrderActions;  
 private final Action<BeerOrderStatusEnum, BeerOrderEventEnum> AllocateOrderAction;  
  
 @Override  
 public void configure(StateMachineStateConfigurer<BeerOrderStatusEnum, BeerOrderEventEnum> states) throws Exception {  
 states.withStates()  
 .initial(BeerOrderStatusEnum.*NEW*)  
 .states(EnumSet.*allOf*(BeerOrderStatusEnum.class))  
 .end(BeerOrderStatusEnum.*PICKED\_UP*)  
 .end(BeerOrderStatusEnum.*DELIVERED*)  
 .end(BeerOrderStatusEnum.*CANCELLED*)  
 .end(BeerOrderStatusEnum.*DELIVERY\_EXCEPTION*)  
 .end(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)  
 .end(BeerOrderStatusEnum.*ALLOCATION\_EXCEPTION*);  
 }  
  
 @Override  
 public void configure(StateMachineTransitionConfigurer<BeerOrderStatusEnum, BeerOrderEventEnum> transitions) throws Exception {  
 transitions  
 .withExternal()

.source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATION\_PENDING*)

.event(BeerOrderEventEnum.*VALIDATE\_ORDER*).action(ValidateOrderActions)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*VALIDATION\_PENDING*).target(BeerOrderStatusEnum.*VALIDATED*)  
 .event(BeerOrderEventEnum.*VALIDATION\_PASSED*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*VALIDATION\_PENDING*).target(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)  
 .event(BeerOrderEventEnum.*VALIDATION\_FAILED*)  
 .and().withExternal()

.source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATED*)

.event(BeerOrderEventEnum.*VALIDATION\_PASSED*)  
 .and().withExternal()

.source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)

.event(BeerOrderEventEnum.*VALIDATION\_FAILED*)  
 .and().withExternal()

.source(BeerOrderStatusEnum.*VALIDATED*).target(BeerOrderStatusEnum.*ALLOCATION\_PENDING*)

.event(BeerOrderEventEnum.*ALLOCATE\_ORDER*).action(AllocateOrderAction);  
 }  
}

11.5 Send AllocateOrder event after successful validation.

package guru.sfg.beer.order.service.services;   
  
@RequiredArgsConstructor  
@Service  
public class BeerOrderManagerImpl implements BeerOrderManager {  
  
 public static final String *ORDER\_ID\_HEADER* = "ORDER\_ID\_HEADER" ;  
  
 private final StateMachineFactory<BeerOrderStatusEnum, BeerOrderEventEnum> stateMachineFactory;  
 private final BeerOrderRepository beerOrderRepository;  
 private final BeerOrderStateChangeInterceptor beerOrderStateChangeInterceptor;  
  
 @Transactional  
 @Override  
 public BeerOrder newBeerOrder(BeerOrder beerOrder) {  
 beerOrder.setId(null);  
 beerOrder.setOrderStatus(BeerOrderStatusEnum.*NEW*);  
  
 BeerOrder savedBeerOrder = beerOrderRepository.save(beerOrder);  
 sendBeerEvent(savedBeerOrder, BeerOrderEventEnum.*VALIDATE\_ORDER*);  
  
 return savedBeerOrder;  
 }  
  
 @Override  
 public void processValidationResult(UUID beerOrderId, Boolean isValid) {  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById(beerOrderId);  
  
 beerOrderOptional.ifPresentOrElse(beerOrder -> {  
 if(isValid){  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*VALIDATION\_PASSED*);  
  
 //fetching new object bcz beerOrder becomes stale object  
 BeerOrder validatedOrder = beerOrderRepository.findById(beerOrderId).get();  
 sendBeerOrderEvent(validatedOrder, BeerOrderEventEnum.*ALLOCATE\_ORDER*);  
 }else{  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*VALIDATE\_ORDER*);  
 }  
 }, ()-> *log*.error("order not found, Id:"+ beerOrderId));

}  
  
 private void sendBeerEvent(BeerOrder beerOrder, BeerOrderEventEnum eventEnum){  
 StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> sm = build(beerOrder);  
 Message msg = MessageBuilder.*withPayload*(eventEnum)  
 .setHeader(*ORDER\_ID\_HEADER*, beerOrder.getId())  
 .build();  
 sm.sendEvent(msg);  
 }

private StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> build(BeerOrder beerOrder){  
 StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> sm = stateMachineFactory.getStateMachine(beerOrder.getId());  
 sm.stop();  
 sm.getStateMachineAccessor()  
 .doWithAllRegions(sma->{  
 sma.addStateMachineInterceptor(beerOrderStateChangeInterceptor);  
 sma.resetStateMachine(new DefaultStateMachineContext<>(beerOrder.getOrderStatus(),null, null,null));  
 });  
 sm.start();  
 return sm;  
 }  
}

Flow : after successful validation sendBeerEvent(validatedOrder, BeerOrderEventEnum.*ALLOCATE\_ORDER*); will execute, then in interceptor(preStateChange) new state will be saved to db ,then state changes, then action executes.

12. Add Allocation Service in mssc-beer-inventory-service-2

12.1 copy BeerOrderDto and BeerOrderLineDto to brewery>model

BeerOrderDto

package guru.sfg.brewery.model;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class BeerOrderDto {  
  
 @JsonProperty("id")  
 private UUID id = null;  
  
 @JsonProperty("version")  
 private Integer version = null;  
  
 @JsonFormat(pattern="yyyy-MM-dd'T'HH:mm:ssZ", shape=JsonFormat.Shape.*STRING*)  
 @JsonProperty("createdDate")  
 private OffsetDateTime createdDate = null;  
  
 @JsonFormat(pattern="yyyy-MM-dd'T'HH:mm:ssZ", shape=JsonFormat.Shape.*STRING*)  
 @JsonProperty("lastModifiedDate")  
 private OffsetDateTime lastModifiedDate = null;  
  
 private UUID customerId;  
 private String customerRef;  
 private List<BeerOrderLineDto> beerOrderLines;  
 private String orderStatus;  
 private String orderStatusCallbackUrl;  
}

BeerOrderLineDto

package guru.sfg.brewery.model;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class BeerOrderLineDto {  
  
 @JsonProperty("id")  
 private UUID id = null;  
  
 @JsonProperty("version")  
 private Integer version = null;  
  
 @JsonFormat(pattern="yyyy-MM-dd'T'HH:mm:ssZ", shape=JsonFormat.Shape.*STRING*)  
 @JsonProperty("createdDate")  
 private OffsetDateTime createdDate = null;  
  
 @JsonFormat(pattern="yyyy-MM-dd'T'HH:mm:ssZ", shape=JsonFormat.Shape.*STRING*)  
 @JsonProperty("lastModifiedDate")  
 private OffsetDateTime lastModifiedDate = null;  
  
 private String upc;  
 private String beerName;  
 private String beerStyle;  
 private UUID beerId;  
 private Integer orderQuantity = 0;  
 private Integer quantityAllocated;  
 private BigDecimal price;  
}

12.2 add findByUpc to InventoryRepo

public interface BeerInventoryRepository extends JpaRepository<BeerInventory, UUID> {  
  
 List<BeerInventory> findAllByBeerId(UUID beerId);  
  
 List<BeerInventory> findAllByUpc(String upc);  
}

12.3 add AllocationService (interface and implementation)

package guru.sfg.beer.inventory.service.services;   
public interface AllocationService {

Boolean allocateOrder(BeerOrderDto beerOrderDto);  
}

package guru.sfg.beer.inventory.service.services;

@Slf4j  
@Service  
@RequiredArgsConstructor  
public class AllocationServiceImpl implements AllocationService {  
  
 private final BeerInventoryRepository beerInventoryRepository;  
  
 @Override  
 public Boolean allocateOrder(BeerOrderDto beerOrderDto) {  
 *log*.debug("Allocating OrderId: " + beerOrderDto.getId());  
  
 AtomicInteger totalOrdered = new AtomicInteger();  
 AtomicInteger totalAllocated = new AtomicInteger();  
  
 beerOrderDto.getBeerOrderLines().forEach(beerOrderLine -> {

//orderQuantity = customer ordered quantity, allocatedQuantity = pre allocated quantity  
 if ((((beerOrderLine.getOrderQuantity() != null ? beerOrderLine.getOrderQuantity() : 0)  
 - (beerOrderLine.getQuantityAllocated() != null ? beerOrderLine.getQuantityAllocated() : 0)) > 0)) {  
 allocateBeerOrderLine(beerOrderLine);  
 }  
 totalOrdered.set(totalOrdered.get() + beerOrderLine.getOrderQuantity());  
 totalAllocated.set(totalAllocated.get() + (beerOrderLine.getQuantityAllocated() != null ? beerOrderLine.getQuantityAllocated() : 0));  
 });  
  
 *log*.debug("Total Ordered: " + totalOrdered.get() + " Total Allocated: " + totalAllocated.get());  
  
 return totalOrdered.get() == totalAllocated.get();  
 }  
  
 private void allocateBeerOrderLine(BeerOrderLineDto beerOrderLine) {  
 List<BeerInventory> beerInventoryList = beerInventoryRepository.findAllByUpc(beerOrderLine.getUpc());  
  
 beerInventoryList.forEach(beerInventory -> {  
 int inventory = (beerInventory.getQuantityOnHand() == null) ? 0 : beerInventory.getQuantityOnHand();  
 int orderQty = (beerOrderLine.getOrderQuantity() == null) ? 0 : beerOrderLine.getOrderQuantity();  
 int allocatedQty = (beerOrderLine.getQuantityAllocated() == null) ? 0 : beerOrderLine.getQuantityAllocated();  
 int qtyToAllocate = orderQty - allocatedQty;  
  
 if (inventory >= qtyToAllocate) { // full allocation  
 inventory = inventory - qtyToAllocate;  
 beerOrderLine.setQuantityAllocated(orderQty);  
 beerInventory.setQuantityOnHand(inventory);  
  
 beerInventoryRepository.save(beerInventory);  
 } else if (inventory > 0) { //partial allocation  
 beerOrderLine.setQuantityAllocated(allocatedQty + inventory);  
 beerInventory.setQuantityOnHand(0);  
  
 }  
  
 if (beerInventory.getQuantityOnHand() == 0) {  
 beerInventoryRepository.delete(beerInventory);  
 }  
 });  
  
 }  
}

13 Completing Allocation Message Flow

Handle allocation request in mssc-beer-inventory-service-2 and allocation response in mssc-beer-order-service-2.

13.1 copy AllocateOrderRequest to mssc-beer-inventory-service-2 ,brewery>model>event from mssc-beer-order-service-2.

@Data  
@AllArgsConstructor  
@NoArgsConstructor  
@Builder  
public class AllocateOrderRequest {  
 private BeerOrderDto beerOrderDto;  
}

13.2 create AllocateOrderResult in mssc-beer-inventory-service-2 and also copy it to mssc-beer-order-service-2, brewery>model>event.

@Data  
@AllArgsConstructor  
@NoArgsConstructor  
@Builder  
public class AllocateOrderResult {  
 private BeerOrderDto beerOrderDto;  
 private Boolean allocationError = false;  
 private Boolean pendingInventory = false;  
}

13.3 create Queue Constants ( both in mssc-beer-order-service-2 & in mssc-beer-inventory-service-2 )

@Configuration  
public class JmsConfig {  
  
 public static final String *NEW\_INVENTORY\_QUEUE* = "new-inventory";  
 //beer-order  
 public static final String *ALLOCATE\_ORDER\_QUEUE* = "allocate-order";  
 public static final String *ALLOCATE\_ORDER\_RESPONSE\_QUEUE* = "allocate-order-response";  
  
 //Convert Message to Object (or vice-versa)  
 @Bean  
 public MessageConverter messageConverter(ObjectMapper objectMapper) {  
 MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter();  
 converter.setTargetType(MessageType.*TEXT*);  
 converter.setTypeIdPropertyName("\_type"); //need tobe set in order to convert incoming messge to java object (default is "none")  
 converter.setObjectMapper(objectMapper);  
 return converter;  
 }  
}

13.4 create AllocationListener in in mssc-beer-inventory-service-2

package guru.sfg.beer.inventory.service.services.listeners;

@Component  
@RequiredArgsConstructor  
@Slf4j  
public class AllocationListener {  
  
 private final AllocationService allocationService;  
 private final JmsTemplate jmsTemplate;  
  
 @JmsListener(destination = JmsConfig.*ALLOCATE\_ORDER\_QUEUE*)  
 public void listen(AllocateOrderRequest request){  
 AllocateOrderResult.AllocateOrderResultBuilder builder = AllocateOrderResult.*builder*();  
 builder.beerOrderDto(request.getBeerOrderDto());  
  
 try {  
 Boolean allocationResult = allocationService.allocateOrder(request.getBeerOrderDto());  
  
 if(allocationResult){  
 builder.pendingInventory(false);  
 }else {  
 builder.pendingInventory(true);  
 }  
 }catch (Exception e){  
 builder.allocationError(true);  
 }  
 jmsTemplate.convertAndSend(JmsConfig.*ALLOCATE\_ORDER\_RESPONSE\_QUEUE*, builder.build());  
 }  
}

13.5 Change SSM configuration (in mssc-beer-order-service-2)

Add transition from *ALLOCATION\_PENDING to ALLOCATION\_EXCEPTION, PENDING\_INVENTORY*

package guru.sfg.beer.order.service.sm;  
  
@Configuration  
@RequiredArgsConstructor  
@EnableStateMachineFactory  
public class BeerOrderStateMachineConfig extends StateMachineConfigurerAdapter<BeerOrderStatusEnum, BeerOrderEventEnum> {  
  
 private final Action<BeerOrderStatusEnum, BeerOrderEventEnum> validateOrderActions;  
 private final Action<BeerOrderStatusEnum, BeerOrderEventEnum> allocateOrderAction;  
  
 @Override  
 public void configure(StateMachineStateConfigurer<BeerOrderStatusEnum, BeerOrderEventEnum> states) throws Exception {  
 states.withStates()  
 .initial(BeerOrderStatusEnum.*NEW*)  
 .states(EnumSet.*allOf*(BeerOrderStatusEnum.class))  
 .end(BeerOrderStatusEnum.*PICKED\_UP*)  
 .end(BeerOrderStatusEnum.*DELIVERED*)  
 .end(BeerOrderStatusEnum.*CANCELLED*)  
 .end(BeerOrderStatusEnum.*DELIVERY\_EXCEPTION*)  
 .end(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)  
 .end(BeerOrderStatusEnum.*ALLOCATION\_EXCEPTION*);  
 }  
  
 @Override  
 public void configure(StateMachineTransitionConfigurer<BeerOrderStatusEnum, BeerOrderEventEnum> transitions) throws Exception {  
transitions  
 .withExternal()  
 .source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATION\_PENDING*)  
 .event(BeerOrderEventEnum.*VALIDATE\_ORDER*).action(validateOrderActions)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*VALIDATION\_PENDING*).target(BeerOrderStatusEnum.*VALIDATED*)  
 .event(BeerOrderEventEnum.*VALIDATION\_PASSED*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*VALIDATION\_PENDING*).target(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*).event(BeerOrderEventEnum.*VALIDATION\_FAILED*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATED*)  
 .event(BeerOrderEventEnum.*VALIDATION\_PASSED*)  
 .and() .withExternal()  
 .source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)  
 .event(BeerOrderEventEnum.*VALIDATION\_FAILED*)

.and().withExternal()

.source(BeerOrderStatusEnum.*VALIDATED*).target(BeerOrderStatusEnum.*ALLOCATION\_PENDING*).event(BeerOrderEventEnum.*ALLOCATE\_ORDER*).action(allocateOrderAction)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATION\_PENDING*).target(BeerOrderStatusEnum.*ALLOCATED*)  
 .event(BeerOrderEventEnum.*ALLOCATION\_SUCCESS*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATION\_PENDING*).target(BeerOrderStatusEnum.*ALLOCATION\_EXCEPTION*)  
 .event(BeerOrderEventEnum.*ALLOCATION\_FAILED*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATION\_PENDING*).target(BeerOrderStatusEnum.*PENDING\_INVENTORY*)  
 .event(BeerOrderEventEnum.*ALLOCATION\_NO\_INVENTORY*);  
 }  
}

13.6 add Listener for AllocationResult from inventoryService (in mssc-beer-order-service-2)

package guru.sfg.beer.order.service.services.listeners;  
  
@Slf4j  
@RequiredArgsConstructor  
@Component  
public class BeerOrderAllocationResultListener {  
  
 private final BeerOrderManager beerOrderManager;  
  
 @JmsListener(destination = JmsConfig.*ALLOCATE\_ORDER\_RESPONSE\_QUEUE*)  
 public void listen(AllocateOrderResult result){  
 if(!result.getAllocationError() && !result.getPendingInventory()){  
 //allocated normally  
 beerOrderManager.beerOrderAllocationPassed(result.getBeerOrderDto());  
 } else if(!result.getAllocationError() && result.getPendingInventory()) {  
 //pending inventory  
 beerOrderManager.beerOrderAllocationPendingInventory(result.getBeerOrderDto());  
 } else if(result.getAllocationError()){  
 //allocation error  
 beerOrderManager.beerOrderAllocationFailed(result.getBeerOrderDto());  
 }  
 }  
}

13.7 add methods in BeerOrderManager

public interface BeerOrderManager {  
  
 BeerOrder newBeerOrder(BeerOrder beerOrder);  
  
 void processValidationResult(UUID beerOrderId, Boolean isValid);  
  
 void beerOrderAllocationPassed(BeerOrderDto beerOrder);  
  
 void beerOrderAllocationPendingInventory(BeerOrderDto beerOrder);  
  
 void beerOrderAllocationFailed(BeerOrderDto beerOrder);  
}

package guru.sfg.beer.order.service.services;

@RequiredArgsConstructor  
@Service  
public class BeerOrderManagerImpl implements BeerOrderManager {  
  
 public static final String *ORDER\_ID\_HEADER* = "ORDER\_ID\_HEADER" ;  
  
 private final StateMachineFactory<BeerOrderStatusEnum, BeerOrderEventEnum> stateMachineFactory;  
 private final BeerOrderRepository beerOrderRepository;  
 private final BeerOrderStateChangeInterceptor beerOrderStateChangeInterceptor;  
  
 @Transactional  
 @Override  
 public BeerOrder newBeerOrder(BeerOrder beerOrder) {  
 beerOrder.setId(null);  
 beerOrder.setOrderStatus(BeerOrderStatusEnum.*NEW*);  
  
 BeerOrder savedBeerOrder = beerOrderRepository.save(beerOrder);  
 sendBeerOrderEvent(savedBeerOrder, BeerOrderEventEnum.*VALIDATE\_ORDER*);  
  
 return savedBeerOrder;  
 }  
  
 @Override  
 public void processValidationResult(UUID beerOrderId, Boolean isValid) {  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById(beerOrderId);  
  
 beerOrderOptional.ifPresentOrElse(beerOrder -> {  
 if(isValid){  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*VALIDATION\_PASSED*);  
  
 //fetching new object bcz beerOrder becomes stale object  
 BeerOrder validatedOrder = beerOrderRepository.findById(beerOrderId).get();  
 sendBeerOrderEvent(validatedOrder, BeerOrderEventEnum.*ALLOCATE\_ORDER*);  
 }else{  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*VALIDATE\_ORDER*);  
 }  
 }, ()-> *log*.error("order not found, Id:"+ beerOrderId));

}  
  
 **--------------------------------------------------------------------------------------------------------**

@Override  
 public void beerOrderAllocationPassed(BeerOrderDto beerOrderDto) {  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById((beerOrderDto.getId());  
  
 beerOrderOptional.ifPresentOrElse(beerOrder ->{  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*ALLOCATION\_SUCCESS*);  
 updateAllocatedQty(beerOrderDto);  
 }, ()-> *log*.error("OrderId not found:"+ beerOrderDto.getId()) );

}  
  
 @Override  
 public void beerOrderAllocationPendingInventory(BeerOrderDto beerOrderDto) {  
 BeerOrder beerOrder = beerOrderRepository.getOne(beerOrderDto.getId());  
  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*ALLOCATION\_NO\_INVENTORY*);  
 updateAllocatedQty(beerOrderDto);  
 }  
  
 private void updateAllocatedQty(BeerOrderDto beerOrderDto) {  
 BeerOrder allocatedOrder = beerOrderRepository.getOne(beerOrderDto.getId());  
  
 allocatedOrder.getBeerOrderLines().forEach(beerOrderLine -> {  
 beerOrderDto.getBeerOrderLines().forEach(beerOrderLineDto -> {  
 if(beerOrderLine.getId().equals(beerOrderLineDto.getId())) {  
 beerOrderLine.setQuantityAllocated(beerOrderLineDto.getQuantityAllocated());  
 }  
 });  
 });  
 beerOrderRepository.saveAndFlush(allocatedOrder);  
 }  
  
 @Override  
 public void beerOrderAllocationFailed(BeerOrderDto beerOrderDto) {  
 BeerOrder beerOrder = beerOrderRepository.getOne(beerOrderDto.getId());  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*ALLOCATION\_FAILED*);  
 }

**--------------------------------------------------------------------------------------------------------**  
  
 private void sendBeerOrderEvent(BeerOrder beerOrder, BeerOrderEventEnum eventEnum){  
 StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> sm = build(beerOrder);  
 Message msg = MessageBuilder.*withPayload*(eventEnum)  
 .setHeader(*ORDER\_ID\_HEADER*, beerOrder.getId())  
 .build();  
 sm.sendEvent(msg);  
 }  
  
 private StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> build(BeerOrder beerOrder){  
 StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> sm = stateMachineFactory.getStateMachine(beerOrder.getId());  
 sm.stop();  
 sm.getStateMachineAccessor()  
 .doWithAllRegions(sma->{  
 sma.addStateMachineInterceptor(beerOrderStateChangeInterceptor);  
 sma.resetStateMachine(new DefaultStateMachineContext<>(beerOrder.getOrderStatus(),null, null,null));  
 });  
 sm.start();  
 return sm;  
 }  
}

Img:BeerOrderManagerImpl\_LastModified

**Note:** For the autowired actions, why don't we need to use a qualifier annotation? How does Spring know which action to wire in?

When Spring creates a bean from an annotated stereotype, the default bean name is the classname of the bean (with leading lowercase). If the autowire target property matches the bean name, Spring will use this as a qualifier.

**RetroSpective** (what is SCRUM, Sprint,…) vid no: 233

refer: [RetroSpective.pdf](pdf-notes/Retrospective.pdf)

4.0 Integration Testing of SAGAS

4.1 Wire Mock vid:236

Using WireMock we can hit other localhost server as per user knowledge.

Listener Method (just to refer) Note: It is a different project and user have not tested this

@Slf4j  
@Component  
public class BeerOrderStatusChangeEventListener {  
  
 RestTemplate restTemplate;  
 DateMapper dateMapper = new DateMapper();  
  
 public BeerOrderStatusChangeEventListener(RestTemplateBuilder restTemplateBuilder) {  
 this.restTemplate = restTemplateBuilder.build();  
 }  
  
 @Async  
 @EventListener  
 public void listen(BeerOrderStatusChangeEvent event){  
 System.*out*.println("I got an order status change event");  
 System.*out*.println(event);  
  
 OrderStatusUpdate update = OrderStatusUpdate.builder()  
 .id(event.getBeerOrder().getId())  
 .orderId(event.getBeerOrder().getId())  
 .version(event.getBeerOrder().getVersion() != null ? event.getBeerOrder().getVersion().intValue() : null)  
 .createdDate(dateMapper.asOffsetDateTime(event.getBeerOrder().getCreatedDate()))  
 .lastModifiedDate(dateMapper.asOffsetDateTime(event.getBeerOrder().getLastModifiedDate()))  
 .orderStatus(event.getBeerOrder().getOrderStatus().toString())  
 .customerRef(event.getBeerOrder().getCustomerRef())  
 .build();  
  
 try{  
 log.debug("Posting to callback url");  
 restTemplate.postForObject(event.getBeerOrder().getOrderStatusCallbackUrl(), update, String.class);  
 } catch (Throwable t){  
 log.error("Error Preforming callback for order: " + event.getBeerOrder().getId(), t);  
 }  
 }  
}

4.12 WireMock Maven Deps

<dependency>  
 <groupId>com.github.JensPiegsa</groupId>  
 <artifactId>wiremock-extension</artifactId>  
 <version>0.4.0</version>  
 <scope>test</scope>  
</dependency>

Add repository (refer wiremock project in google)

<repositories>  
 <repository>  
 <id>jitpack.io</id>  
 <url>https://jitpack.io</url>  
 </repository>  
</repositories>

4.13 Junit Test

@ExtendWith(WireMockExtension.class)  
class BeerOrderStatusChangeEventListenerTest {  
  
 BeerOrderStatusChangeEventListener listener;  
  
 @BeforeEach  
 void setUp() {  
 RestTemplateBuilder restTemplateBuilder = new RestTemplateBuilder();  
 listener = new BeerOrderStatusChangeEventListener(restTemplateBuilder);  
  
 }  
  
 @Test  
 void listen() {  
  
 BeerOrder beerOrder = BeerOrder.builder()  
 .orderStatus(OrderStatusEnum.READY)  
 .orderStatusCallbackUrl("http://localhost:" + wireMockServer.port() + "/update")  
 .createdDate(Timestamp.*valueOf*(LocalDateTime.*now*()))  
 .build();  
  
 BeerOrderStatusChangeEvent event = new BeerOrderStatusChangeEvent(beerOrder, OrderStatusEnum.NEW);  
  
 listener.listen(event);  
  
 }  
}

4.14 WireMock Server

@ExtendWith(WireMockExtension.class)  
class BeerOrderStatusChangeEventListenerTest {  
  
 @Managed  
 WireMockServer wireMockServer = with(wireMockConfig().dynamicPort());  
  
 BeerOrderStatusChangeEventListener listener;  
  
 @BeforeEach  
 void setUp() {  
 RestTemplateBuilder restTemplateBuilder = new RestTemplateBuilder();  
 listener = new BeerOrderStatusChangeEventListener(restTemplateBuilder);  
  
 }  
  
 @Test  
 void listen() {  
  
 wireMockServer.stubFor(post("/update").willReturn(ok()));  
  
 BeerOrder beerOrder = BeerOrder.builder()  
 .orderStatus(OrderStatusEnum.READY)  
 .orderStatusCallbackUrl("http://localhost:" + wireMockServer.port() + "/update")  
 .createdDate(Timestamp.valueOf(LocalDateTime.now()))  
 .build();  
  
 BeerOrderStatusChangeEvent event = new BeerOrderStatusChangeEvent(beerOrder, OrderStatusEnum.NEW);  
  
 listener.listen(event);  
  
 }  
}

4.15 Verfy Mock Interactions

@ExtendWith(WireMockExtension.class)  
class BeerOrderStatusChangeEventListenerTest {  
  
 @Managed  
 WireMockServer wireMockServer = with(wireMockConfig().dynamicPort());  
  
 BeerOrderStatusChangeEventListener listener;  
  
 @BeforeEach  
 void setUp() {  
 RestTemplateBuilder restTemplateBuilder = new RestTemplateBuilder();  
 listener = new BeerOrderStatusChangeEventListener(restTemplateBuilder);  
  
 }  
  
 @Test  
 void listen() {  
  
 wireMockServer.stubFor(post("/update").willReturn(ok()));  
  
 BeerOrder beerOrder = BeerOrder.builder()  
 .orderStatus(OrderStatusEnum.READY)  
 .orderStatusCallbackUrl("http://localhost:" + wireMockServer.port() + "/update")  
 .createdDate(Timestamp.*valueOf*(LocalDateTime.*now*()))  
 .build();  
  
 BeerOrderStatusChangeEvent event = new BeerOrderStatusChangeEvent(beerOrder, OrderStatusEnum.NEW);  
  
 listener.listen(event);  
  
 verify(1, postRequestedFor(urlEqualTo("/update"))); //1 is count of requests should matching  
  
 }  
}

Implementing the above in mssc-beer-order-service-2

Note: This Testing classs giving error as mentioned below (not resolved), user have created this below notes for knwoledge purpose only

Refer : Img:BeerOrderManagerImpl\_LastModified

Above Image TestClass:

@SpringBootTest  
public class BeerOrderManagerImplIT {  
  
 @Autowired  
 BeerOrderManager beerOrderManager;  
  
 @Autowired  
 BeerOrderRepository beerOrderRepository;  
  
 @Autowired  
 CustomerRepository customerRepository;  
  
 Customer testCustomer;  
  
 UUID beerId = UUID.*randomUUID*();  
  
 @BeforeEach  
 void setup(){  
 testCustomer = customerRepository.save(Customer.*builder*()  
 .customerName("Test Customer")  
 .build());  
 }  
  
 @Test  
 public void testNewToAllocated(){  
 BeerOrder beerOrder = createBeerOrder();  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder); // Line:77   
  
 *assertNotNull*(savedBeerOrder);  
 *assertEquals*(BeerOrderStatusEnum.*ALLOCATED*, savedBeerOrder.getOrderStatus());  
 }  
  
 public BeerOrder createBeerOrder(){  
 BeerOrder beerOrder = BeerOrder.*builder*()  
 .customer(testCustomer)  
 .build();  
  
 Set<BeerOrderLine> lines = new HashSet<>();  
 lines.add(BeerOrderLine.*builder*()  
 .beerId(beerId)  
 .orderQuantity(1)  
 .beerOrder(beerOrder)  
 .build());  
 beerOrder.setBeerOrderLines(lines);  
 return beerOrder;  
 }  
}

Error!!!: java.lang.NullPointerException, at guru.sfg.beer.order.service.services.BeerOrderManagerImplIT.testNewToAllocated(BeerOrderManagerImplIT.java:77)

*Implementing WireMock for it*

1. Adding WireMock dependency and repository to the pom (refer: 4.12 WireMock Maven Deps)
2. Create application.properties in test> resources> application.properties

sfg.brewery.beer-service-host=http://localhost:8083

1. Changes in the test classes

@ExtendWith(WireMockExtension.class)  
@SpringBootTest  
public class BeerOrderManagerImplIT {  
  
 @Autowired  
 BeerOrderManager beerOrderManager;  
  
 @Autowired  
 BeerOrderRepository beerOrderRepository;  
  
 @Autowired  
 CustomerRepository customerRepository;

@Autowired  
 WireMockServer wireMockServer;

@Autowired  
 ObjectMapper objectMapper;

Customer testCustomer;  
  
 UUID beerId = UUID.*randomUUID*();  
  
 @BeforeEach  
 void setup(){  
 testCustomer = customerRepository.save(Customer.*builder*()  
 .customerName("Test Customer")  
 .build());  
 }  
  
 @Test  
 public void testNewToAllocated() throws JsonProcessingException {  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *assertNotNull*(savedBeerOrder);  
 *assertEquals*(BeerOrderStatusEnum.*ALLOCATED*, savedBeerOrder.getOrderStatus());  
 }

public BeerOrder createBeerOrder(){  
 BeerOrder beerOrder = BeerOrder.*builder*()  
 .customer(testCustomer)  
 .build();  
  
 Set<BeerOrderLine> lines = new HashSet<>();  
 lines.add(BeerOrderLine.*builder*()  
 .beerId(beerId)  
 .orderQuantity(1)  
 .beerOrder(beerOrder)

.upc("12345")  
 .build());  
 beerOrder.setBeerOrderLines(lines);  
 return beerOrder;  
 }  
}

**Configuring Jms Server for Testing** vn: 238

Add dependency

<dependency>  
 <groupId>org.apache.activemq</groupId>  
 <artifactId>artemis-jms-server</artifactId>  
 <scope>test</scope>  
</dependency>

Create a Listener for testing test > java .> guru.sfg…service > services > testcomponents > BeerOrderValidationListener.java

@Slf4j  
@RequiredArgsConstructor  
@Component  
public class BeerOrderValidationListener {  
 private final JmsTemplate jmsTemplate;  
  
 @JmsListener(destination = JmsConfig.*VALIDATE\_ORDER\_QUEUE*)  
 public void list(Message msg){  
  
 ValidateOrderRequest request = (ValidateOrderRequest) msg.getPayload();  
  
 System.*out*.println("########### I RAN ########");  
  
 jmsTemplate.convertAndSend(JmsConfig.*VALIDATE\_ORDER\_RESPONSE\_QUEUE*,  
 ValidateOrderResult.builder()  
 .isValid(true)  
 .orderId(request.getBeerOrder().getId())  
 .build());  
  
 }  
}

Observation: If deserialization problem, there might be a problem with JmsConfiguration. For above code we were getting Deserialization error, then we added ObjectMapper to the configuration.

Set up Serialization properties (changes in normal properties file main > resources > application.properties)

#serialization properties  
spring.jackson.serialization.write-dates-as-timestamps=false  
spring.jackson.serialization.write-date-timestamps-as-nanoseconds=true

4.2 Awaitility

Add dependecny

<dependency>  
 <groupId>org.awaitility</groupId>  
 <artifactId>awaitility</artifactId>  
 <scope>test</scope>  
</dependency>

Changes in Test classes

package guru.sfg.beer.order.service.services;  
  
import com.fasterxml.jackson.core.JsonProcessingException;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.github.jenspiegsa.wiremockextension.WireMockExtension;  
import com.github.tomakehurst.wiremock.WireMockServer;  
import guru.sfg.beer.order.service.domain.BeerOrder;  
import guru.sfg.beer.order.service.domain.BeerOrderLine;  
import guru.sfg.beer.order.service.domain.BeerOrderStatusEnum;  
import guru.sfg.beer.order.service.domain.Customer;  
import guru.sfg.beer.order.service.repositories.BeerOrderRepository;  
import guru.sfg.beer.order.service.repositories.CustomerRepository;  
import guru.sfg.beer.order.service.services.beer.BeerServiceImpl;  
import guru.sfg.brewery.model.BeerDto;  
import org.junit.Test;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.extension.ExtendWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.boot.test.context.TestConfiguration;  
import org.springframework.context.annotation.Bean;  
  
import java.util.HashSet;  
import java.util.Set;  
import java.util.UUID;  
  
import static com.github.jenspiegsa.wiremockextension.ManagedWireMockServer.*with*;  
import static com.github.tomakehurst.wiremock.client.WireMock.*okJson*;  
import static com.github.tomakehurst.wiremock.core.WireMockConfiguration.*wireMockConfig*;  
import static com.github.tomakehurst.wiremock.client.WireMock.*get*;  
import static org.awaitility.Awaitility.*await*;  
import static org.junit.Assert.*assertEquals*;  
import static org.junit.Assert.*assertNotNull*;  
  
@ExtendWith(WireMockExtension.class)  
@SpringBootTest  
public class BeerOrderManagerImplIT {  
  
 @Autowired  
 BeerOrderManager beerOrderManager;  
  
 @Autowired  
 BeerOrderRepository beerOrderRepository;  
  
 @Autowired  
 CustomerRepository customerRepository;  
  
 @Autowired  
 WireMockServer wireMockServer;  
  
 @Autowired  
 ObjectMapper objectMapper;  
  
 Customer testCustomer;  
  
 UUID beerId = UUID.*randomUUID*();  
  
 @TestConfiguration  
 static class RestTemplateBuilderProvider {  
 @Bean(destroyMethod = "stop")  
 public WireMockServer wireMockServer(){  
 WireMockServer server = *with*(*wireMockConfig*().port(8083));  
 server.start();  
 return server;  
 }  
 }  
  
 @BeforeEach  
 void setup(){  
 testCustomer = customerRepository.save(Customer.*builder*()  
 .customerName("Test Customer")  
 .build());  
 }  
  
 @Test  
 public void testNewToAllocated() throws JsonProcessingException, InterruptedException {  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
  
 BeerOrderLine line = foundOrder.getBeerOrderLines().iterator().next();  
 *assertEquals*(line.getOrderQuantity(), line.getQuantityAllocated());  
 });  
  
 BeerOrder savedBeerOrder2 = beerOrderRepository.findById(savedBeerOrder.getId()).get();  
  
 *assertNotNull*(savedBeerOrder);  
 *assertEquals*(BeerOrderStatusEnum.*ALLOCATED*, savedBeerOrder2.getOrderStatus());

savedBeerOrder2.getBeerOrderLines().forEach(line->{  
 *assertEquals*(line.getOrderQuantity(), line.getQuantityAllocated());  
 });

}  
  
 public BeerOrder createBeerOrder(){  
 BeerOrder beerOrder = BeerOrder.*builder*()  
 .customer(testCustomer)  
 .build();  
  
 Set<BeerOrderLine> lines = new HashSet<>();  
 lines.add(BeerOrderLine.*builder*()  
 .beerId(beerId)  
 .orderQuantity(1)  
 .beerOrder(beerOrder)  
 .upc("12345")  
 .build());  
 beerOrder.setBeerOrderLines(lines);  
 return beerOrder;  
 }  
}

Create a Allocation Listener for testing test > java .> guru.sfg…service > services > testcomponents > BeerOrderAllocationListener.java

@Slf4j  
@RequiredArgsConstructor  
@Component  
public class BeerOrderAllocationListener {  
  
 private final JmsTemplate jmsTemplate;  
  
 @JmsListener(destination = JmsConfig.*ALLOCATE\_ORDER\_QUEUE*)  
 public void listen(Message msg){  
 AllocateOrderRequest request = (AllocateOrderRequest) msg.getPayload();  
  
 request.getBeerOrderDto().getBeerOrderLines().forEach(beerOrderLineDto -> {  
 beerOrderLineDto.setQuantityAllocated(beerOrderLineDto.getOrderQuantity());  
 });  
  
 jmsTemplate.convertAndSend(JmsConfig.*ALLOCATE\_ORDER\_RESPONSE\_QUEUE*,  
 AllocateOrderResult.*builder*()  
 .beerOrderDto(request.getBeerOrderDto())  
 .pendingInventory(false)  
 .allocationError(false)  
 .build());  
 }  
}

**Observation**:

* StateMachine will be running in different thread.
* ObjectOptimisticLockingFailureException: Object Updated before saving it.

Creating test cases for pickup order

Add transition from ALLOCATED to PICKEDUP

package guru.sfg.beer.order.service.sm;  
  
@Configuration  
@RequiredArgsConstructor  
@EnableStateMachineFactory  
public class BeerOrderStateMachineConfig extends StateMachineConfigurerAdapter<BeerOrderStatusEnum, BeerOrderEventEnum> {  
  
 private final Action<BeerOrderStatusEnum, BeerOrderEventEnum> validateOrderActions;  
 private final Action<BeerOrderStatusEnum, BeerOrderEventEnum> allocateOrderAction;  
  
 @Override  
 public void configure(StateMachineStateConfigurer<BeerOrderStatusEnum, BeerOrderEventEnum> states) throws Exception {  
 states.withStates()  
 .initial(BeerOrderStatusEnum.*NEW*)  
 .states(EnumSet.*allOf*(BeerOrderStatusEnum.class))  
 .end(BeerOrderStatusEnum.*PICKED\_UP*)  
 .end(BeerOrderStatusEnum.*DELIVERED*)  
 .end(BeerOrderStatusEnum.*CANCELLED*)  
 .end(BeerOrderStatusEnum.*DELIVERY\_EXCEPTION*)  
 .end(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)  
 .end(BeerOrderStatusEnum.*ALLOCATION\_EXCEPTION*);  
 }  
  
 @Override  
 public void configure(StateMachineTransitionConfigurer<BeerOrderStatusEnum, BeerOrderEventEnum> transitions) throws Exception {  
 transitions  
 .withExternal()  
 .source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATION\_PENDING*)  
 .event(BeerOrderEventEnum.*VALIDATE\_ORDER*).action(validateOrderActions)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*VALIDATION\_PENDING*).target(BeerOrderStatusEnum.*VALIDATED*)  
 .event(BeerOrderEventEnum.*VALIDATION\_PASSED*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*VALIDATION\_PENDING*).target(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)  
 .event(BeerOrderEventEnum.*VALIDATION\_FAILED*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATED*)  
 .event(BeerOrderEventEnum.*VALIDATION\_PASSED*)  
 .and() .withExternal()  
 .source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)  
 .event(BeerOrderEventEnum.*VALIDATION\_FAILED*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*VALIDATED*).target(BeerOrderStatusEnum.*ALLOCATION\_PENDING*)  
 .event(BeerOrderEventEnum.*ALLOCATE\_ORDER*).action(allocateOrderAction)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATION\_PENDING*).target(BeerOrderStatusEnum.*ALLOCATED*)  
 .event(BeerOrderEventEnum.*ALLOCATION\_SUCCESS*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATION\_PENDING*).target(BeerOrderStatusEnum.*ALLOCATION\_EXCEPTION*)  
 .event(BeerOrderEventEnum.*ALLOCATION\_FAILED*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATION\_PENDING*).target(BeerOrderStatusEnum.*PENDING\_INVENTORY*)  
 .event(BeerOrderEventEnum.*ALLOCATION\_NO\_INVENTORY*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATED*).target(BeerOrderStatusEnum.*PICKED\_UP*)  
 .event(BeerOrderEventEnum.*BEERORDER\_PICKED\_UP*)  
 ;  
 }  
}

Create beerOrderPickedUp() method in BeerOrderManagerImpl

public interface BeerOrderManager {  
  
 BeerOrder newBeerOrder(BeerOrder beerOrder);  
  
 void processValidationResult(UUID beerOrderId, Boolean isValid);  
  
 void beerOrderAllocationPassed(BeerOrderDto beerOrder);  
  
 void beerOrderAllocationPendingInventory(BeerOrderDto beerOrder);  
  
 void beerOrderAllocationFailed(BeerOrderDto beerOrder);  
  
 void beerOrderPickedUp(UUID id);  
}

package guru.sfg.beer.order.service.services;  
  
@Slf4j  
@RequiredArgsConstructor  
@Service  
public class BeerOrderManagerImpl implements BeerOrderManager {  
  
 public static final String *ORDER\_ID\_HEADER* = "ORDER\_ID\_HEADER" ;  
  
 private final StateMachineFactory<BeerOrderStatusEnum, BeerOrderEventEnum> stateMachineFactory;  
 private final BeerOrderRepository beerOrderRepository;  
 private final BeerOrderStateChangeInterceptor beerOrderStateChangeInterceptor;  
  
 @Transactional  
 @Override  
 public BeerOrder newBeerOrder(BeerOrder beerOrder) {  
 beerOrder.setId(null);  
 beerOrder.setOrderStatus(BeerOrderStatusEnum.*NEW*);  
  
 BeerOrder savedBeerOrder = beerOrderRepository.save(beerOrder);  
 sendBeerOrderEvent(savedBeerOrder, BeerOrderEventEnum.*VALIDATE\_ORDER*);  
  
 return savedBeerOrder;  
 }  
  
 @Transactional //while testing lazy initialization error was coming (vn:238)  
 @Override  
 public void processValidationResult(UUID beerOrderId, Boolean isValid) {  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById(beerOrderId);  
  
 beerOrderOptional.ifPresentOrElse(beerOrder -> {  
 if(isValid){  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*VALIDATION\_PASSED*);  
  
 //fetching new object bcz beerOrder becomes stale object  
 BeerOrder validatedOrder = beerOrderRepository.findById(beerOrderId).get();  
 sendBeerOrderEvent(validatedOrder, BeerOrderEventEnum.*ALLOCATE\_ORDER*);  
 }else{  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*VALIDATE\_ORDER*);  
 }  
 }, ()-> *log*.error("order not found, Id:"+ beerOrderId));  
 }  
  
 @Override  
 public void beerOrderAllocationPassed(BeerOrderDto beerOrderDto) {  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById(beerOrderDto.getId());  
  
 beerOrderOptional.ifPresentOrElse(beerOrder ->{  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*ALLOCATION\_SUCCESS*);  
 updateAllocatedQty(beerOrderDto);  
 }, ()-> *log*.error("OrderId not found:"+ beerOrderDto.getId()) );  
 }  
  
 @Override  
 public void beerOrderAllocationPendingInventory(BeerOrderDto beerOrderDto) {  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById(beerOrderDto.getId());  
  
 beerOrderOptional.ifPresentOrElse(beerOrder ->{  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*ALLOCATION\_NO\_INVENTORY*);  
 updateAllocatedQty(beerOrderDto);  
 }, ()-> *log*.error("OrderId not found:"+ beerOrderDto.getId()) );  
 }  
  
 private void updateAllocatedQty(BeerOrderDto beerOrderDto) {  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById(beerOrderDto.getId());  
  
 beerOrderOptional.ifPresentOrElse(allocatedOrder ->{  
 allocatedOrder.getBeerOrderLines().forEach(beerOrderLine -> {  
 beerOrderDto.getBeerOrderLines().forEach(beerOrderLineDto -> {  
 if(beerOrderLine.getId().equals(beerOrderLineDto.getId())) {  
 beerOrderLine.setQuantityAllocated(beerOrderLineDto.getQuantityAllocated());  
 }  
 });  
 });  
 beerOrderRepository.saveAndFlush(allocatedOrder);  
 }, ()-> *log*.error("OrderId not found:"+ beerOrderDto.getId()) );  
  
 }  
  
 @Override  
 public void beerOrderAllocationFailed(BeerOrderDto beerOrderDto) {  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById(beerOrderDto.getId());  
 beerOrderOptional.ifPresentOrElse(beerOrder ->{  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*ALLOCATION\_FAILED*);  
 }, ()-> *log*.error("OrderId not found:"+ beerOrderDto.getId()) );  
 }  
**------------------------------------------------------------------------------------------**  
 @Override  
 public void beerOrderPickedUp(UUID id) {  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById(id);  
  
 beerOrderOptional.ifPresentOrElse(beerOrder ->{  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*BEERORDER\_PICKED\_UP*);  
 }, ()-> *log*.error("OrderId not found:"+ id) );  
  
 }  
**------------------------------------------------------------------------------------------**  
 private void sendBeerOrderEvent(BeerOrder beerOrder, BeerOrderEventEnum eventEnum){  
 StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> sm = build(beerOrder);  
 Message msg = MessageBuilder.*withPayload*(eventEnum)  
 .setHeader(*ORDER\_ID\_HEADER*, beerOrder.getId())  
 .build();  
 sm.sendEvent(msg);  
 }  
  
 private StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> build(BeerOrder beerOrder){  
 StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> sm = stateMachineFactory.getStateMachine(beerOrder.getId());  
 sm.stop();  
 sm.getStateMachineAccessor()  
 .doWithAllRegions(sma->{  
 sma.addStateMachineInterceptor(beerOrderStateChangeInterceptor);  
 sma.resetStateMachine(new DefaultStateMachineContext<>(beerOrder.getOrderStatus(),null, null,null));  
 });  
 sm.start();  
 return sm;  
 }  
}

Create testCase for beerOrderPickedUp

package guru.sfg.beer.order.service.services;  
  
import com.fasterxml.jackson.core.JsonProcessingException;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.github.jenspiegsa.wiremockextension.WireMockExtension;  
import com.github.tomakehurst.wiremock.WireMockServer;  
import guru.sfg.beer.order.service.domain.BeerOrder;  
import guru.sfg.beer.order.service.domain.BeerOrderLine;  
import guru.sfg.beer.order.service.domain.BeerOrderStatusEnum;  
import guru.sfg.beer.order.service.domain.Customer;  
import guru.sfg.beer.order.service.repositories.BeerOrderRepository;  
import guru.sfg.beer.order.service.repositories.CustomerRepository;  
import guru.sfg.beer.order.service.services.beer.BeerServiceImpl;  
import guru.sfg.brewery.model.BeerDto;  
import org.junit.Test;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.extension.ExtendWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.boot.test.context.TestConfiguration;  
import org.springframework.context.annotation.Bean;  
  
import static com.github.jenspiegsa.wiremockextension.ManagedWireMockServer.*with*;  
import static com.github.tomakehurst.wiremock.client.WireMock.*okJson*;  
import static com.github.tomakehurst.wiremock.core.WireMockConfiguration.*wireMockConfig*;  
import static com.github.tomakehurst.wiremock.client.WireMock.*get*;  
import static org.awaitility.Awaitility.*await*;  
import static org.junit.Assert.*assertEquals*;  
import static org.junit.Assert.*assertNotNull*;  
  
@ExtendWith(WireMockExtension.class)  
@SpringBootTest  
public class BeerOrderManagerImplIT {  
  
 @Autowired  
 BeerOrderManager beerOrderManager;  
  
 @Autowired  
 BeerOrderRepository beerOrderRepository;  
  
 @Autowired  
 CustomerRepository customerRepository;  
  
 @Autowired  
 WireMockServer wireMockServer;  
  
 @Autowired  
 ObjectMapper objectMapper;  
  
 Customer testCustomer;  
  
 UUID beerId = UUID.*randomUUID*();  
  
 @TestConfiguration  
 static class RestTemplateBuilderProvider {  
 @Bean(destroyMethod = "stop")  
 public WireMockServer wireMockServer(){  
 WireMockServer server = *with*(*wireMockConfig*().port(8083));  
 server.start();  
 return server;  
 }  
 }  
  
 @BeforeEach  
 void setup(){  
 testCustomer = customerRepository.save(Customer.*builder*()  
 .customerName("Test Customer")  
 .build());  
 }  
  
 @Test  
 public void testNewToAllocated() throws JsonProcessingException, InterruptedException {  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 BeerOrderLine line = foundOrder.getBeerOrderLines().iterator().next();  
 *assertEquals*(line.getOrderQuantity(), line.getQuantityAllocated());  
 });  
  
 BeerOrder savedBeerOrder2 = beerOrderRepository.findById(savedBeerOrder.getId()).get();  
  
 *assertNotNull*(savedBeerOrder);  
 *assertEquals*(BeerOrderStatusEnum.*ALLOCATED*, savedBeerOrder2.getOrderStatus());  
 savedBeerOrder2.getBeerOrderLines().forEach(line->{  
 *assertEquals*(line.getOrderQuantity(), line.getQuantityAllocated());  
 });  
 }  
**------------------------------------------------------------------------------------------**  
 void testNewToPickedUp() throws JsonProcessingException {  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*ALLOCATED*, foundOrder.getOrderStatus());  
 });  
  
 beerOrderManager.beerOrderPickedUp(savedBeerOrder.getId());  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*PICKED\_UP*, foundOrder.getOrderStatus());  
 });  
  
 BeerOrder pickedUpOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
  
 *assertEquals*(BeerOrderStatusEnum.*PICKED\_UP*, pickedUpOrder.getOrderStatus());  
 }  
**------------------------------------------------------------------------------------------**  
  
 public BeerOrder createBeerOrder(){  
 BeerOrder beerOrder = BeerOrder.*builder*()  
 .customer(testCustomer)  
 .build();  
  
 Set<BeerOrderLine> lines = new HashSet<>();  
 lines.add(BeerOrderLine.*builder*()  
 .beerId(beerId)  
 .orderQuantity(1)  
 .beerOrder(beerOrder)  
 .upc("12345")  
 .build());  
 beerOrder.setBeerOrderLines(lines);  
 return beerOrder;  
 }  
}

Refactor beerOrderService to use Saga (beer-order-service-2)

Use BeerOrderManager methods instead of hardcoding

package guru.sfg.beer.order.service.services;  
  
@Slf4j  
@RequiredArgsConstructor  
@Service  
public class BeerOrderServiceImpl implements BeerOrderService {  
  
 private final BeerOrderRepository beerOrderRepository;  
 private final CustomerRepository customerRepository;  
 private final BeerOrderMapper beerOrderMapper;  
 private final ApplicationEventPublisher publisher;  
 private final BeerOrderManager beerOrderManager;  
  
 @Override  
 public BeerOrderPagedList listOrders(UUID customerId, Pageable pageable) {  
 Optional<Customer> customerOptional = customerRepository.findById(customerId);  
  
 if (customerOptional.isPresent()) {  
 Page<BeerOrder> beerOrderPage =  
 beerOrderRepository.findAllByCustomer(customerOptional.get(), pageable);  
  
 return new BeerOrderPagedList(beerOrderPage  
 .stream()  
 .map(beerOrderMapper::beerOrderToDto)  
 .collect(Collectors.*toList*()), PageRequest.*of*(  
 beerOrderPage.getPageable().getPageNumber(),  
 beerOrderPage.getPageable().getPageSize()),  
 beerOrderPage.getTotalElements());  
 } else {  
 return null;  
 }  
 }  
  
 @Transactional  
 @Override  
 public BeerOrderDto placeOrder(UUID customerId, BeerOrderDto beerOrderDto) {  
 Optional<Customer> customerOptional = customerRepository.findById(customerId);  
  
 if (customerOptional.isPresent()) {  
 BeerOrder beerOrder = beerOrderMapper.dtoToBeerOrder(beerOrderDto);  
 beerOrder.setId(null); //should not be set by outside client  
 beerOrder.setCustomer(customerOptional.get());  
 beerOrder.setOrderStatus(BeerOrderStatusEnum.*NEW*);  
  
 beerOrder.getBeerOrderLines().forEach(line -> line.setBeerOrder(beerOrder));  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder); //before we used to save it in db  
  
 return beerOrderMapper.beerOrderToDto(savedBeerOrder);  
 }  
 //*todo add exception type* throw new RuntimeException("Customer Not Found");  
 }  
  
 @Override  
 public BeerOrderDto getOrderById(UUID customerId, UUID orderId) {  
 return beerOrderMapper.beerOrderToDto(getOrder(customerId, orderId));  
 }  
  
 @Override  
 public void pickupOrder(UUID customerId, UUID orderId) {

//Before  
// BeerOrder beerOrder = getOrder(customerId, orderId);  
// beerOrder.setOrderStatus(BeerOrderStatusEnum.PICKED\_UP);

beerOrderManager.beerOrderPickedUp(orderId);  
 }  
  
 private BeerOrder getOrder(UUID customerId, UUID orderId){  
 Optional<Customer> customerOptional = customerRepository.findById(customerId);  
  
 if(customerOptional.isPresent()){  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById(orderId);  
  
 if(beerOrderOptional.isPresent()){  
 BeerOrder beerOrder = beerOrderOptional.get();  
  
 // fall to exception if customer id's do not match - order not for customer  
 if(beerOrder.getCustomer().getId().equals(customerId)){  
 return beerOrder;  
 }  
 }  
 throw new RuntimeException("Beer Order Not Found");  
 }  
 throw new RuntimeException("Customer Not Found");  
 }  
}

5.0 Componsating transactions with sagas

**5.1 Testing Failed Validation**

package guru.sfg.beer.order.service.services;  
  
import com.fasterxml.jackson.core.JsonProcessingException;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.github.jenspiegsa.wiremockextension.WireMockExtension;  
import com.github.tomakehurst.wiremock.WireMockServer;  
import guru.sfg.beer.order.service.domain.BeerOrder;  
import guru.sfg.beer.order.service.domain.BeerOrderLine;  
import guru.sfg.beer.order.service.domain.BeerOrderStatusEnum;  
import guru.sfg.beer.order.service.domain.Customer;  
import guru.sfg.beer.order.service.repositories.BeerOrderRepository;  
import guru.sfg.beer.order.service.repositories.CustomerRepository;  
import guru.sfg.beer.order.service.services.beer.BeerServiceImpl;  
import guru.sfg.brewery.model.BeerDto;  
import org.junit.Test;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.extension.ExtendWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.boot.test.context.TestConfiguration;  
import org.springframework.context.annotation.Bean;  
  
import java.util.HashSet;  
import java.util.Set;  
import java.util.UUID;  
  
import static com.github.jenspiegsa.wiremockextension.ManagedWireMockServer.*with*;  
import static com.github.tomakehurst.wiremock.client.WireMock.*get*;  
import static com.github.tomakehurst.wiremock.client.WireMock.*okJson*;  
import static com.github.tomakehurst.wiremock.core.WireMockConfiguration.*wireMockConfig*;  
import static org.awaitility.Awaitility.*await*;  
import static org.junit.Assert.*assertEquals*;  
import static org.junit.Assert.*assertNotNull*;  
  
@ExtendWith(WireMockExtension.class)  
@SpringBootTest  
public class BeerOrderManagerImplIT {  
  
 @Autowired  
 BeerOrderManager beerOrderManager;  
  
 @Autowired  
 BeerOrderRepository beerOrderRepository;  
  
 @Autowired  
 CustomerRepository customerRepository;  
  
// @MockBean  
// BeerOrderManager beerOrderManager;  
//  
// @MockBean  
// BeerOrderRepository beerOrderRepository;  
//  
// @MockBean  
// CustomerRepository customerRepository;  
  
 @Autowired  
 WireMockServer wireMockServer;  
  
 @Autowired  
 ObjectMapper objectMapper;  
  
 Customer testCustomer;  
  
 UUID beerId = UUID.*randomUUID*();  
  
 @TestConfiguration  
 static class RestTemplateBuilderProvider {  
 @Bean(destroyMethod = "stop")  
 public WireMockServer wireMockServer(){  
 WireMockServer server = *with*(*wireMockConfig*().port(8083));  
 server.start();  
 return server;  
 }  
 }  
  
 @BeforeEach  
 void setup(){  
 testCustomer = customerRepository.save(Customer.*builder*()  
 .customerName("Test Customer")  
 .build());  
 }  
  
 @Test  
 public void testNewToAllocated() throws JsonProcessingException, InterruptedException {  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 BeerOrderLine line = foundOrder.getBeerOrderLines().iterator().next();  
 *assertEquals*(line.getOrderQuantity(), line.getQuantityAllocated());  
 });  
  
 BeerOrder savedBeerOrder2 = beerOrderRepository.findById(savedBeerOrder.getId()).get();  
  
 *assertNotNull*(savedBeerOrder);  
 *assertEquals*(BeerOrderStatusEnum.*ALLOCATED*, savedBeerOrder2.getOrderStatus());  
 savedBeerOrder2.getBeerOrderLines().forEach(line->{  
 *assertEquals*(line.getOrderQuantity(), line.getQuantityAllocated());  
 });  
 }  
  
 @Test  
 public void testNewToPickedUp() throws JsonProcessingException {  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*ALLOCATED*, foundOrder.getOrderStatus());  
 });  
  
 beerOrderManager.beerOrderPickedUp(savedBeerOrder.getId());  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*PICKED\_UP*, foundOrder.getOrderStatus());  
 });  
  
 BeerOrder pickedUpOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
  
 *assertEquals*(BeerOrderStatusEnum.*PICKED\_UP*, pickedUpOrder.getOrderStatus());  
 }  
**------------------------------------------------------------------------------------------**

@Test  
 public void testFailedValidation() throws JsonProcessingException{  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
 beerOrder.setCustomerRef("fail-validation");  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*, foundOrder.getOrderStatus());  
 });  
 }  
**------------------------------------------------------------------------------------------**  
  
 public BeerOrder createBeerOrder(){  
 BeerOrder beerOrder = BeerOrder.*builder*()  
 .customer(testCustomer)  
 .build();  
  
 Set<BeerOrderLine> lines = new HashSet<>();  
 lines.add(BeerOrderLine.*builder*()  
 .beerId(beerId)  
 .orderQuantity(1)  
 .beerOrder(beerOrder)  
 .upc("12345")  
 .build());  
 beerOrder.setBeerOrderLines(lines);  
 return beerOrder;  
 }  
}

Changes in test jmsListener (Listener created inside test package)

package guru.sfg.beer.order.service.services.testcomponents;  
  
@Slf4j  
@RequiredArgsConstructor  
@Component  
public class BeerOrderValidationListener {  
 private final JmsTemplate jmsTemplate;  
  
 @JmsListener(destination = JmsConfig.*VALIDATE\_ORDER\_QUEUE*)  
 public void list(Message msg){  
  
 boolean isValid = true;  
  
 ValidateOrderRequest request = (ValidateOrderRequest) msg.getPayload();  
  
 //condition to fail validation  
 if(request.getBeerOrderDto().getCustomerRef() != null &&

request.getBeerOrderDto().getCustomerRef().equals("fail-validation")){  
 isValid=false;  
 }  
  
 System.*out*.println("########### I RAN ########");  
  
 jmsTemplate.convertAndSend(JmsConfig.*VALIDATE\_ORDER\_RESPONSE\_QUEUE*,  
 ValidateOrderResult.*builder*()  
 .isValid(isValid)  
 .orderId(request.getBeerOrderDto().getId())  
 .build());  
  
 }  
}

Configure StateMachine

package guru.sfg.beer.order.service.sm;  
  
@Configuration  
@RequiredArgsConstructor  
@EnableStateMachineFactory  
public class BeerOrderStateMachineConfig extends StateMachineConfigurerAdapter<BeerOrderStatusEnum, BeerOrderEventEnum> {  
  
 private final Action<BeerOrderStatusEnum, BeerOrderEventEnum> validateOrderActions;  
 private final Action<BeerOrderStatusEnum, BeerOrderEventEnum> allocateOrderAction;  
 private final Action<BeerOrderStatusEnum, BeerOrderEventEnum> validationFailureAction;  
  
 @Override  
 public void configure(StateMachineStateConfigurer<BeerOrderStatusEnum, BeerOrderEventEnum> states) throws Exception {  
 states.withStates()  
 .initial(BeerOrderStatusEnum.*NEW*)  
 .states(EnumSet.*allOf*(BeerOrderStatusEnum.class))  
 .end(BeerOrderStatusEnum.*PICKED\_UP*)  
 .end(BeerOrderStatusEnum.*DELIVERED*)  
 .end(BeerOrderStatusEnum.*CANCELLED*)  
 .end(BeerOrderStatusEnum.*DELIVERY\_EXCEPTION*)  
 .end(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)  
 .end(BeerOrderStatusEnum.*ALLOCATION\_EXCEPTION*);  
 }  
  
 @Override  
 public void configure(StateMachineTransitionConfigurer<BeerOrderStatusEnum, BeerOrderEventEnum> transitions) throws Exception {  
 transitions  
 .withExternal()  
 .source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATION\_PENDING*)  
 .event(BeerOrderEventEnum.*VALIDATE\_ORDER*).action(validateOrderActions)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*VALIDATION\_PENDING*).target(BeerOrderStatusEnum.*VALIDATED*)  
 .event(BeerOrderEventEnum.*VALIDATION\_PASSED*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*VALIDATION\_PENDING*).target(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)  
 .event(BeerOrderEventEnum.*VALIDATION\_FAILED*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATED*)  
 .event(BeerOrderEventEnum.*VALIDATION\_PASSED*)  
 .and() .withExternal()  
 .source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)  
 .event(BeerOrderEventEnum.*VALIDATION\_FAILED*).action(validationFailureAction)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*VALIDATED*).target(BeerOrderStatusEnum.*ALLOCATION\_PENDING*)  
 .event(BeerOrderEventEnum.*ALLOCATE\_ORDER*).action(allocateOrderAction)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATION\_PENDING*).target(BeerOrderStatusEnum.*ALLOCATED*)  
 .event(BeerOrderEventEnum.*ALLOCATION\_SUCCESS*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATION\_PENDING*).target(BeerOrderStatusEnum.*ALLOCATION\_EXCEPTION*)  
 .event(BeerOrderEventEnum.*ALLOCATION\_FAILED*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATION\_PENDING*).target(BeerOrderStatusEnum.*PENDING\_INVENTORY*)  
 .event(BeerOrderEventEnum.*ALLOCATION\_NO\_INVENTORY*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATED*).target(BeerOrderStatusEnum.*PICKED\_UP*)  
 .event(BeerOrderEventEnum.*BEERORDER\_PICKED\_UP*)  
 ;  
 }  
}

Assignment

Develop Tests to handle failed allocation and partial allocation

* Integration Tests to validate Spring State Machine configuration
* Update Allocation Listener to fail allocation and do partial allocation

Create test for allocationFailure and partialAllocation

package guru.sfg.beer.order.service.services;  
  
import com.fasterxml.jackson.core.JsonProcessingException;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.github.jenspiegsa.wiremockextension.WireMockExtension;  
import com.github.tomakehurst.wiremock.WireMockServer;  
import guru.sfg.beer.order.service.domain.BeerOrder;  
import guru.sfg.beer.order.service.domain.BeerOrderLine;  
import guru.sfg.beer.order.service.domain.BeerOrderStatusEnum;  
import guru.sfg.beer.order.service.domain.Customer;  
import guru.sfg.beer.order.service.repositories.BeerOrderRepository;  
import guru.sfg.beer.order.service.repositories.CustomerRepository;  
import guru.sfg.beer.order.service.services.beer.BeerServiceImpl;  
import guru.sfg.brewery.model.BeerDto;  
import org.junit.Test;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.extension.ExtendWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.boot.test.context.TestConfiguration;  
import org.springframework.context.annotation.Bean;  
  
import java.util.HashSet;  
import java.util.Set;  
import java.util.UUID;  
  
import static com.github.jenspiegsa.wiremockextension.ManagedWireMockServer.*with*;  
import static com.github.tomakehurst.wiremock.client.WireMock.*get*;  
import static com.github.tomakehurst.wiremock.client.WireMock.*okJson*;  
import static com.github.tomakehurst.wiremock.core.WireMockConfiguration.*wireMockConfig*;  
import static org.awaitility.Awaitility.*await*;  
import static org.junit.Assert.*assertEquals*;  
import static org.junit.Assert.*assertNotNull*;  
  
@ExtendWith(WireMockExtension.class)  
@SpringBootTest  
public class BeerOrderManagerImplIT {  
  
 @Autowired  
 BeerOrderManager beerOrderManager;  
  
 @Autowired  
 BeerOrderRepository beerOrderRepository;  
  
 @Autowired  
 CustomerRepository customerRepository;  
  
// @MockBean  
// BeerOrderManager beerOrderManager;  
//  
// @MockBean  
// BeerOrderRepository beerOrderRepository;  
//  
// @MockBean  
// CustomerRepository customerRepository;  
  
 @Autowired  
 WireMockServer wireMockServer;  
  
 @Autowired  
 ObjectMapper objectMapper;  
  
 Customer testCustomer;  
  
 UUID beerId = UUID.*randomUUID*();  
  
 @TestConfiguration  
 static class RestTemplateBuilderProvider {  
 @Bean(destroyMethod = "stop")  
 public WireMockServer wireMockServer(){  
 WireMockServer server = *with*(*wireMockConfig*().port(8083));  
 server.start();  
 return server;  
 }  
 }  
  
 @BeforeEach  
 void setup(){  
 testCustomer = customerRepository.save(Customer.*builder*()  
 .customerName("Test Customer")  
 .build());  
 }  
  
 @Test  
 public void testNewToAllocated() throws JsonProcessingException, InterruptedException {  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 BeerOrderLine line = foundOrder.getBeerOrderLines().iterator().next();  
 *assertEquals*(line.getOrderQuantity(), line.getQuantityAllocated());  
 });  
  
 BeerOrder savedBeerOrder2 = beerOrderRepository.findById(savedBeerOrder.getId()).get();  
  
 *assertNotNull*(savedBeerOrder);  
 *assertEquals*(BeerOrderStatusEnum.*ALLOCATED*, savedBeerOrder2.getOrderStatus());  
 savedBeerOrder2.getBeerOrderLines().forEach(line->{  
 *assertEquals*(line.getOrderQuantity(), line.getQuantityAllocated());  
 });  
 }  
  
 @Test  
 public void testNewToPickedUp() throws JsonProcessingException {  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*ALLOCATED*, foundOrder.getOrderStatus());  
 });  
  
 beerOrderManager.beerOrderPickedUp(savedBeerOrder.getId());  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*PICKED\_UP*, foundOrder.getOrderStatus());  
 });  
  
 BeerOrder pickedUpOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
  
 *assertEquals*(BeerOrderStatusEnum.*PICKED\_UP*, pickedUpOrder.getOrderStatus());  
 }  
  
 @Test  
 public void testFailedValidation() throws JsonProcessingException{  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
 beerOrder.setCustomerRef("fail-validation");  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*, foundOrder.getOrderStatus());  
 });  
 }  
**------------------------------------------------------------------------------------------**  
 @Test  
 public void testAllocationFailure() throws JsonProcessingException {  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
 beerOrder.setCustomerRef("fail-allocation");  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*ALLOCATION\_EXCEPTION*, foundOrder.getOrderStatus());  
 });  
 }  
  
 @Test  
 public void testPartialAllocation() throws JsonProcessingException {  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
 beerOrder.setCustomerRef("partial-allocation");  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*PENDING\_INVENTORY*, foundOrder.getOrderStatus());  
 });  
 }  
**------------------------------------------------------------------------------------------**  
 public BeerOrder createBeerOrder(){  
 BeerOrder beerOrder = BeerOrder.*builder*()  
 .customer(testCustomer)  
 .build();  
  
 Set<BeerOrderLine> lines = new HashSet<>();  
 lines.add(BeerOrderLine.*builder*()  
 .beerId(beerId)  
 .orderQuantity(1)  
 .beerOrder(beerOrder)  
 .upc("12345")  
 .build());  
 beerOrder.setBeerOrderLines(lines);  
 return beerOrder;  
 }  
}

BeerOrderAllocationListener changes (inside test package)

package guru.sfg.beer.order.service.services.testcomponents;  
  
@Slf4j  
@RequiredArgsConstructor  
@Component  
public class BeerOrderAllocationListener {  
  
 private final JmsTemplate jmsTemplate;  
  
 @JmsListener(destination = JmsConfig.*ALLOCATE\_ORDER\_QUEUE*)  
 public void listen(Message msg){  
 AllocateOrderRequest request = (AllocateOrderRequest) msg.getPayload();

**------------------------------------------------------------------------------------------**  
 boolean pendingInventory = false;  
 boolean allocationError = false;  
  
 //condition to fail validation  
 if(request.getBeerOrderDto().getCustomerRef() != null

&& request.getBeerOrderDto().getCustomerRef().equals("partial-allocation")){  
 pendingInventory=true;  
 }  
 //condition to fail validation  
 if(request.getBeerOrderDto().getCustomerRef() != null

&& request.getBeerOrderDto().getCustomerRef().equals("fail-allocation")){  
 allocationError=true;  
 }

// -Code Before-  
// request.getBeerOrderDto().getBeerOrderLines().forEach(beerOrderLineDto -> {  
// beerOrderLineDto.setQuantityAllocated(beerOrderLineDto.getOrderQuantity());  
// });  
 boolean finalPendingInventory = pendingInventory;  
 request.getBeerOrderDto().getBeerOrderLines().forEach(beerOrderLineDto -> {  
 if(finalPendingInventory){  
 beerOrderLineDto.setQuantityAllocated(beerOrderLineDto.getOrderQuantity()-1);  
 }else{  
 beerOrderLineDto.setQuantityAllocated(beerOrderLineDto.getOrderQuantity()-1);  
 }  
 });  
**------------------------------------------------------------------------------------------**  
 jmsTemplate.convertAndSend(JmsConfig.*ALLOCATE\_ORDER\_RESPONSE\_QUEUE*,  
 AllocateOrderResult.*builder*()  
 .beerOrderDto(request.getBeerOrderDto())  
 .pendingInventory(pendingInventory)  
 .allocationError(allocationError)  
 .build());  
 }  
}

**Note:**

1. How do you modify the allocation listener to return the desired result for the test condition?

Add a property to the message payload with can be used for conditional logic for the return action.

1. If we are using @SpringBootTest we will be having access to full spring context.

Assignment – Failed Allocation Componsating transaction

* Use Spring State Machine Action to Send Failed Allocation Compensating Transaction
* Create New Message with Order Id Property
* Create new Message queue
* Send Message
* Verify Message in Integration Test
  + Use JmsTemplate receiveAndConvert method

Create AllocationFailureEvent class

@Data  
@AllArgsConstructor  
@NoArgsConstructor  
@Builder  
public class AllocationFailureEvent {  
 private UUID orderId;  
}

Add AllocationFailure Queue name to JmsConfig

@Configuration  
public class JmsConfig {  
  
 public static final String *VALIDATE\_ORDER\_QUEUE* = "validate-order";  
 public static final String *VALIDATE\_ORDER\_RESPONSE\_QUEUE* = "validate-order-response";  
 public static final String *ALLOCATE\_ORDER\_QUEUE* = "allocate-order";  
 public static final String *ALLOCATE\_ORDER\_RESPONSE\_QUEUE* = "allocate-order-response";  
 public static final String *ALLOCATE\_FAILURE\_QUEUE* = "allocation-failure";  
  
 //Convert Message to Object (or vice-versa)  
 @Bean  
 public MessageConverter messageConverter(ObjectMapper objectMapper) {  
 MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter();  
 converter.setTargetType(MessageType.*TEXT*);  
 converter.setObjectMapper(objectMapper);  
 converter.setTypeIdPropertyName("\_type"); //need tobe set in order to convert incoming messge to java object (default is "none")  
 return converter;  
 }  
}

Create AllocationFailureAction class

@Slf4j  
@Component  
@RequiredArgsConstructor  
public class AllocationFailureAction implements Action<BeerOrderStatusEnum, BeerOrderEventEnum> {  
  
 private final JmsTemplate jmsTemplate;  
  
 @Override  
 public void execute(StateContext<BeerOrderStatusEnum, BeerOrderEventEnum> stateContext) {  
 String beerOrderId = (String) stateContext.getMessage().getHeaders().get(BeerOrderManagerImpl.*ORDER\_ID\_HEADER*);  
  
 jmsTemplate.convertAndSend(JmsConfig.*VALIDATE\_ORDER\_QUEUE*, AllocationFailureEvent.*builder*()  
 .orderId(UUID.*fromString*(beerOrderId))  
 .build());  
  
 *log*.debug("Sent Allocation Failure Message for order id: ",beerOrderId);  
 }  
}

Configure StateMachineConfigure class

package guru.sfg.beer.order.service.sm;  
  
@Configuration  
@RequiredArgsConstructor  
@EnableStateMachineFactory  
public class BeerOrderStateMachineConfig extends StateMachineConfigurerAdapter<BeerOrderStatusEnum, BeerOrderEventEnum> {  
  
 private final Action<BeerOrderStatusEnum, BeerOrderEventEnum> validateOrderActions;  
 private final Action<BeerOrderStatusEnum, BeerOrderEventEnum> allocateOrderAction;  
 private final Action<BeerOrderStatusEnum, BeerOrderEventEnum> validationFailureAction;  
 private final Action<BeerOrderStatusEnum, BeerOrderEventEnum> allocationFailureAction;  
  
 @Override  
 public void configure(StateMachineStateConfigurer<BeerOrderStatusEnum, BeerOrderEventEnum> states) throws Exception {  
 states.withStates()  
 .initial(BeerOrderStatusEnum.*NEW*)  
 .states(EnumSet.*allOf*(BeerOrderStatusEnum.class))  
 .end(BeerOrderStatusEnum.*PICKED\_UP*)  
 .end(BeerOrderStatusEnum.*DELIVERED*)  
 .end(BeerOrderStatusEnum.*CANCELLED*)  
 .end(BeerOrderStatusEnum.*DELIVERY\_EXCEPTION*)  
 .end(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)  
 .end(BeerOrderStatusEnum.*ALLOCATION\_EXCEPTION*);  
 }  
  
 @Override  
 public void configure(StateMachineTransitionConfigurer<BeerOrderStatusEnum, BeerOrderEventEnum> transitions) throws Exception {  
 transitions  
 .withExternal()  
 .source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATION\_PENDING*)  
 .event(BeerOrderEventEnum.*VALIDATE\_ORDER*).action(validateOrderActions)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*VALIDATION\_PENDING*).target(BeerOrderStatusEnum.*VALIDATED*)  
 .event(BeerOrderEventEnum.*VALIDATION\_PASSED*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*VALIDATION\_PENDING*).target(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)  
 .event(BeerOrderEventEnum.*VALIDATION\_FAILED*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATED*)  
 .event(BeerOrderEventEnum.*VALIDATION\_PASSED*)  
 .and() .withExternal()  
 .source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)  
 .event(BeerOrderEventEnum.*VALIDATION\_FAILED*).action(validationFailureAction)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*VALIDATED*).target(BeerOrderStatusEnum.*ALLOCATION\_PENDING*)  
 .event(BeerOrderEventEnum.*ALLOCATE\_ORDER*).action(allocateOrderAction)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATION\_PENDING*).target(BeerOrderStatusEnum.*ALLOCATED*)  
 .event(BeerOrderEventEnum.*ALLOCATION\_SUCCESS*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATION\_PENDING*).target(BeerOrderStatusEnum.*ALLOCATION\_EXCEPTION*)  
 .event(BeerOrderEventEnum.*ALLOCATION\_FAILED*).action(allocationFailureAction)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATION\_PENDING*).target(BeerOrderStatusEnum.*PENDING\_INVENTORY*)  
 .event(BeerOrderEventEnum.*ALLOCATION\_NO\_INVENTORY*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATED*).target(BeerOrderStatusEnum.*PICKED\_UP*)  
 .event(BeerOrderEventEnum.*BEERORDER\_PICKED\_UP*)  
 ;  
 }  
}

Add JmsMessage integration testing to testClass

package guru.sfg.beer.order.service.services;  
  
import com.fasterxml.jackson.core.JsonProcessingException;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.github.jenspiegsa.wiremockextension.WireMockExtension;  
import com.github.tomakehurst.wiremock.WireMockServer;  
import guru.sfg.beer.order.service.config.JmsConfig;  
import guru.sfg.beer.order.service.domain.BeerOrder;  
import guru.sfg.beer.order.service.domain.BeerOrderLine;  
import guru.sfg.beer.order.service.domain.BeerOrderStatusEnum;  
import guru.sfg.beer.order.service.domain.Customer;  
import guru.sfg.beer.order.service.repositories.BeerOrderRepository;  
import guru.sfg.beer.order.service.repositories.CustomerRepository;  
import guru.sfg.beer.order.service.services.beer.BeerServiceImpl;  
import guru.sfg.brewery.model.BeerDto;  
import guru.sfg.brewery.model.events.AllocationFailureEvent;  
import org.assertj.core.api.Assertions;  
import org.junit.Test;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.extension.ExtendWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.boot.test.context.TestConfiguration;  
import org.springframework.context.annotation.Bean;  
import org.springframework.jms.core.JmsTemplate;  
  
import java.util.HashSet;  
import java.util.Set;  
import java.util.UUID;  
  
import static com.github.jenspiegsa.wiremockextension.ManagedWireMockServer.*with*;  
import static com.github.tomakehurst.wiremock.client.WireMock.*get*;  
import static com.github.tomakehurst.wiremock.client.WireMock.*okJson*;  
import static com.github.tomakehurst.wiremock.core.WireMockConfiguration.*wireMockConfig*;  
import static org.awaitility.Awaitility.*await*;  
import static org.junit.Assert.*assertEquals*;  
import static org.junit.Assert.*assertNotNull*;  
  
@ExtendWith(WireMockExtension.class)  
@SpringBootTest  
public class BeerOrderManagerImplIT {  
  
 @Autowired  
 BeerOrderManager beerOrderManager;  
  
 @Autowired  
 BeerOrderRepository beerOrderRepository;  
  
 @Autowired  
 CustomerRepository customerRepository;  
  
// @MockBean  
// BeerOrderManager beerOrderManager;  
//  
// @MockBean  
// BeerOrderRepository beerOrderRepository;  
//  
// @MockBean  
// CustomerRepository customerRepository;  
  
 @Autowired  
 WireMockServer wireMockServer;  
  
 @Autowired  
 ObjectMapper objectMapper;  
  
 @Autowired  
 JmsTemplate jmsTemplate;  
  
 Customer testCustomer;  
  
 UUID beerId = UUID.*randomUUID*();  
  
 @TestConfiguration  
 static class RestTemplateBuilderProvider {  
 @Bean(destroyMethod = "stop")  
 public WireMockServer wireMockServer(){  
 WireMockServer server = *with*(*wireMockConfig*().port(8083));  
 server.start();  
 return server;  
 }  
 }  
  
 @BeforeEach  
 void setup(){  
 testCustomer = customerRepository.save(Customer.*builder*()  
 .customerName("Test Customer")  
 .build());  
 }  
  
 @Test  
 public void testNewToAllocated() throws JsonProcessingException, InterruptedException {  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 BeerOrderLine line = foundOrder.getBeerOrderLines().iterator().next();  
 *assertEquals*(line.getOrderQuantity(), line.getQuantityAllocated());  
 });  
  
 BeerOrder savedBeerOrder2 = beerOrderRepository.findById(savedBeerOrder.getId()).get();  
  
 *assertNotNull*(savedBeerOrder);  
 *assertEquals*(BeerOrderStatusEnum.*ALLOCATED*, savedBeerOrder2.getOrderStatus());  
 savedBeerOrder2.getBeerOrderLines().forEach(line->{  
 *assertEquals*(line.getOrderQuantity(), line.getQuantityAllocated());  
 });  
 }  
  
 @Test  
 public void testNewToPickedUp() throws JsonProcessingException {  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*ALLOCATED*, foundOrder.getOrderStatus());  
 });  
  
 beerOrderManager.beerOrderPickedUp(savedBeerOrder.getId());  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*PICKED\_UP*, foundOrder.getOrderStatus());  
 });  
  
 BeerOrder pickedUpOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
  
 *assertEquals*(BeerOrderStatusEnum.*PICKED\_UP*, pickedUpOrder.getOrderStatus());  
 }  
  
 @Test  
 public void testFailedValidation() throws JsonProcessingException{  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
 beerOrder.setCustomerRef("fail-validation");  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*, foundOrder.getOrderStatus());  
 });  
 }  
  
 @Test  
 public void testAllocationFailure() throws JsonProcessingException {  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
 beerOrder.setCustomerRef("fail-allocation");  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*ALLOCATION\_EXCEPTION*, foundOrder.getOrderStatus());  
 });  
  
 AllocationFailureEvent allocationFailureEvent =

(AllocationFailureEvent)jmsTemplate.receiveAndConvert(JmsConfig.*ALLOCATE\_FAILURE\_QUEUE*);

*assertNotNull*(allocationFailureEvent);  
 Assertions.*assertThat*(allocationFailureEvent.getOrderId()).isEqualTo(savedBeerOrder.getId());  
 }  
  
 @Test  
 public void testPartialAllocation() throws JsonProcessingException {  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
 beerOrder.setCustomerRef("partial-allocation");  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*PENDING\_INVENTORY*, foundOrder.getOrderStatus());  
 });  
 }  
  
 public BeerOrder createBeerOrder(){  
 BeerOrder beerOrder = BeerOrder.*builder*()  
 .customer(testCustomer)  
 .build();  
  
 Set<BeerOrderLine> lines = new HashSet<>();  
 lines.add(BeerOrderLine.*builder*()  
 .beerId(beerId)  
 .orderQuantity(1)  
 .beerOrder(beerOrder)  
 .upc("12345")  
 .build());  
 beerOrder.setBeerOrderLines(lines);  
 return beerOrder;  
 }  
}

**Note:**

Why are we using JMS rather than just using Mockito?

This is an integration test, thus we wish to test the interactions of the components. Mockito is typically for testing a single class.

5.2 Cancel Order Configuration

Add CANCEL\_ORDER event to BeerOrderEventEnum

public enum BeerOrderEventEnum {  
 *VALIDATE\_ORDER*, *CANCEL\_ORDER*, *VALIDATION\_PASSED*, *VALIDATION\_FAILED*,  
 *ALLOCATE\_ORDER*, *ALLOCATION\_SUCCESS*, *ALLOCATION\_NO\_INVENTORY*, *ALLOCATION\_FAILED*,  
 *BEERORDER\_PICKED\_UP*}

Add CANCELED status to BeerOrderStatusEnum

public enum BeerOrderStatusEnum {  
 *NEW*, *VALIDATED*, *VALIDATION\_PENDING*, *VALIDATION\_EXCEPTION*,  
 *ALLOCATION\_PENDING*, *ALLOCATED*, *ALLOCATION\_EXCEPTION*, *CANCELLED*,  
 *PENDING\_INVENTORY*, *PICKED\_UP*, *DELIVERED*, *DELIVERY\_EXCEPTION*}

Configure BeerOrderStateMachineConfig

ValidationPending -> cancel

Validated -> cancel

Allocation Pending -> cancel

Allocated -> cancel

@Configuration  
@RequiredArgsConstructor  
@EnableStateMachineFactory  
public class BeerOrderStateMachineConfig extends StateMachineConfigurerAdapter<BeerOrderStatusEnum, BeerOrderEventEnum> {  
  
 private final Action<BeerOrderStatusEnum, BeerOrderEventEnum> validateOrderActions;  
 private final Action<BeerOrderStatusEnum, BeerOrderEventEnum> allocateOrderAction;  
 private final Action<BeerOrderStatusEnum, BeerOrderEventEnum> validationFailureAction;  
 private final Action<BeerOrderStatusEnum, BeerOrderEventEnum> allocationFailureAction;  
  
 @Override  
 public void configure(StateMachineStateConfigurer<BeerOrderStatusEnum, BeerOrderEventEnum> states) throws Exception {  
 states.withStates()  
 .initial(BeerOrderStatusEnum.*NEW*)  
 .states(EnumSet.*allOf*(BeerOrderStatusEnum.class))  
 .end(BeerOrderStatusEnum.*PICKED\_UP*)  
 .end(BeerOrderStatusEnum.*DELIVERED*)  
 .end(BeerOrderStatusEnum.*CANCELLED*)  
 .end(BeerOrderStatusEnum.*DELIVERY\_EXCEPTION*)  
 .end(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)  
 .end(BeerOrderStatusEnum.*ALLOCATION\_EXCEPTION*);  
 }  
  
 @Override  
 public void configure(StateMachineTransitionConfigurer<BeerOrderStatusEnum, BeerOrderEventEnum> transitions) throws Exception {  
 transitions  
 .withExternal()  
 .source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATION\_PENDING*)  
 .event(BeerOrderEventEnum.*VALIDATE\_ORDER*).action(validateOrderActions)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*VALIDATION\_PENDING*).target(BeerOrderStatusEnum.*VALIDATED*)  
 .event(BeerOrderEventEnum.*VALIDATION\_PASSED*)

.and().withExternal()  
 .source(BeerOrderStatusEnum.*VALIDATION\_PENDING*).target(BeerOrderStatusEnum.*CANCELLED*)  
 .event(BeerOrderEventEnum.*CANCEL\_ORDER*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*VALIDATED*).target(BeerOrderStatusEnum.*CANCELLED*)  
 .event(BeerOrderEventEnum.*CANCEL\_ORDER*)

.and().withExternal()  
 .source(BeerOrderStatusEnum.*VALIDATION\_PENDING*).target(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)  
 .event(BeerOrderEventEnum.*VALIDATION\_FAILED*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATED*)  
 .event(BeerOrderEventEnum.*VALIDATION\_PASSED*)  
 .and() .withExternal()  
 .source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)  
 .event(BeerOrderEventEnum.*VALIDATION\_FAILED*).action(validationFailureAction)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*VALIDATED*).target(BeerOrderStatusEnum.*ALLOCATION\_PENDING*)  
 .event(BeerOrderEventEnum.*ALLOCATE\_ORDER*).action(allocateOrderAction)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATION\_PENDING*).target(BeerOrderStatusEnum.*ALLOCATED*)  
 .event(BeerOrderEventEnum.*ALLOCATION\_SUCCESS*)

.and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATION\_PENDING*).target(BeerOrderStatusEnum.*CANCELLED*)  
 .event(BeerOrderEventEnum.*CANCEL\_ORDER*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATED*).target(BeerOrderStatusEnum.*CANCELLED*)  
 .event(BeerOrderEventEnum.*CANCEL\_ORDER*) //*todo add action*

.and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATION\_PENDING*).target(BeerOrderStatusEnum.*ALLOCATION\_EXCEPTION*)  
 .event(BeerOrderEventEnum.*ALLOCATION\_FAILED*).action(allocationFailureAction)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATION\_PENDING*).target(BeerOrderStatusEnum.*PENDING\_INVENTORY*)  
 .event(BeerOrderEventEnum.*ALLOCATION\_NO\_INVENTORY*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATED*).target(BeerOrderStatusEnum.*PICKED\_UP*)  
 .event(BeerOrderEventEnum.*BEERORDER\_PICKED\_UP*)  
 ;  
 }  
}

5.3 Cancel Order Componsating Transaction (sending jms message for calcellation on ALLOCATED state)

Add Jms Queue for DeAllocation

package guru.sfg.beer.order.service.config;  
  
@Configuration  
public class JmsConfig {  
  
 public static final String *VALIDATE\_ORDER\_QUEUE* = "validate-order";  
 public static final String *VALIDATE\_ORDER\_RESPONSE\_QUEUE* = "validate-order-response";  
 public static final String *ALLOCATE\_ORDER\_QUEUE* = "allocate-order";  
 public static final String *ALLOCATE\_ORDER\_RESPONSE\_QUEUE* = "allocate-order-response";  
 public static final String *ALLOCATE\_FAILURE\_QUEUE* = "allocation-failure";  
 public static final String *DEALLOCATE\_ORDER\_QUEUE* = "deallocate-order";  
  
 //Convert Message to Object (or vice-versa)  
 @Bean  
 public MessageConverter messageConverter(ObjectMapper objectMapper) {  
 MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter();  
 converter.setTargetType(MessageType.*TEXT*);  
 converter.setObjectMapper(objectMapper);  
 converter.setTypeIdPropertyName("\_type"); //need tobe set in order to convert incoming messge to java object (default is "none")  
 return converter;  
 }  
}

Add DeallocateOrderRequest class (brewery>model>events)

package guru.sfg.brewery.model.events;  
  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
@Builder  
public class DeallocateOrderRequest {  
 private BeerOrderDto beerOrderDto;  
}

Add DeallocateOrderAction

@Slf4j  
@Component  
@RequiredArgsConstructor  
public class DeallocateOrderAction implements Action<BeerOrderStatusEnum, BeerOrderEventEnum> {  
  
 private final BeerOrderRepository beerOrderRepository;  
 private final BeerOrderMapper beerOrderMapper;  
 private final JmsTemplate jmsTemplate;  
  
 @Override  
 public void execute(StateContext<BeerOrderStatusEnum, BeerOrderEventEnum> stateContext) {  
 String beerOrderId = (String) stateContext.getMessage().getHeaders().get(BeerOrderManagerImpl.*ORDER\_ID\_HEADER*);  
 BeerOrder beerOrder = beerOrderRepository.getOne(UUID.*fromString*(beerOrderId));  
 jmsTemplate.convertAndSend(JmsConfig.*DEALLOCATE\_ORDER\_QUEUE*, DeallocateOrderRequest.*builder*()  
 .beerOrderDto(beerOrderMapper.beerOrderToDto(beerOrder))  
 .build()  
 );  
 *log*.debug("Sent DeAllocated Order Request to queue for order id: ",beerOrderId);  
  
 }  
}

Add action to StatemachineConfig class

@Configuration  
@RequiredArgsConstructor  
@EnableStateMachineFactory  
public class BeerOrderStateMachineConfig extends StateMachineConfigurerAdapter<BeerOrderStatusEnum, BeerOrderEventEnum> {  
  
 private final Action<BeerOrderStatusEnum, BeerOrderEventEnum> validateOrderActions;  
 private final Action<BeerOrderStatusEnum, BeerOrderEventEnum> allocateOrderAction;  
 private final Action<BeerOrderStatusEnum, BeerOrderEventEnum> validationFailureAction;  
 private final Action<BeerOrderStatusEnum, BeerOrderEventEnum> allocationFailureAction;  
 private final Action<BeerOrderStatusEnum, BeerOrderEventEnum> deallocateOrderAction;  
  
 @Override  
 public void configure(StateMachineStateConfigurer<BeerOrderStatusEnum, BeerOrderEventEnum> states) throws Exception {  
 states.withStates()  
 .initial(BeerOrderStatusEnum.*NEW*)  
 .states(EnumSet.*allOf*(BeerOrderStatusEnum.class))  
 .end(BeerOrderStatusEnum.*PICKED\_UP*)  
 .end(BeerOrderStatusEnum.*DELIVERED*)  
 .end(BeerOrderStatusEnum.*CANCELLED*)  
 .end(BeerOrderStatusEnum.*DELIVERY\_EXCEPTION*)  
 .end(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)  
 .end(BeerOrderStatusEnum.*ALLOCATION\_EXCEPTION*);  
 }  
  
 @Override  
 public void configure(StateMachineTransitionConfigurer<BeerOrderStatusEnum, BeerOrderEventEnum> transitions) throws Exception {  
 transitions  
 .withExternal()  
 .source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATION\_PENDING*)  
 .event(BeerOrderEventEnum.*VALIDATE\_ORDER*).action(validateOrderActions)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*VALIDATION\_PENDING*).target(BeerOrderStatusEnum.*VALIDATED*)  
 .event(BeerOrderEventEnum.*VALIDATION\_PASSED*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*VALIDATION\_PENDING*).target(BeerOrderStatusEnum.*CANCELLED*)  
 .event(BeerOrderEventEnum.*CANCEL\_ORDER*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*VALIDATED*).target(BeerOrderStatusEnum.*CANCELLED*)  
 .event(BeerOrderEventEnum.*CANCEL\_ORDER*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*VALIDATION\_PENDING*).target(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)  
 .event(BeerOrderEventEnum.*VALIDATION\_FAILED*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATED*)  
 .event(BeerOrderEventEnum.*VALIDATION\_PASSED*)  
 .and() .withExternal()  
 .source(BeerOrderStatusEnum.*NEW*).target(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*)  
 .event(BeerOrderEventEnum.*VALIDATION\_FAILED*).action(validationFailureAction)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*VALIDATED*).target(BeerOrderStatusEnum.*ALLOCATION\_PENDING*)  
 .event(BeerOrderEventEnum.*ALLOCATE\_ORDER*).action(allocateOrderAction)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATION\_PENDING*).target(BeerOrderStatusEnum.*ALLOCATED*)  
 .event(BeerOrderEventEnum.*ALLOCATION\_SUCCESS*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATION\_PENDING*).target(BeerOrderStatusEnum.*CANCELLED*)  
 .event(BeerOrderEventEnum.*CANCEL\_ORDER*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATED*).target(BeerOrderStatusEnum.*CANCELLED*)  
 .event(BeerOrderEventEnum.*CANCEL\_ORDER*).action(deallocateOrderAction)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATION\_PENDING*).target(BeerOrderStatusEnum.*ALLOCATION\_EXCEPTION*)  
 .event(BeerOrderEventEnum.*ALLOCATION\_FAILED*).action(allocationFailureAction)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATION\_PENDING*).target(BeerOrderStatusEnum.*PENDING\_INVENTORY*)  
 .event(BeerOrderEventEnum.*ALLOCATION\_NO\_INVENTORY*)  
 .and().withExternal()  
 .source(BeerOrderStatusEnum.*ALLOCATED*).target(BeerOrderStatusEnum.*PICKED\_UP*)  
 .event(BeerOrderEventEnum.*BEERORDER\_PICKED\_UP*)  
 ;  
 }  
}

5.4 Integration Test for Cancel Order

Create cancelOrder() method in BeerOrderManager

public interface BeerOrderManager {  
  
 BeerOrder newBeerOrder(BeerOrder beerOrder);  
  
 void processValidationResult(UUID beerOrderId, Boolean isValid);  
  
 void beerOrderAllocationPassed(BeerOrderDto beerOrder);  
  
 void beerOrderAllocationPendingInventory(BeerOrderDto beerOrder);  
  
 void beerOrderAllocationFailed(BeerOrderDto beerOrder);  
  
 void beerOrderPickedUp(UUID id);  
  
 void cancelOrder(UUID id);  
}

BeerOrderManagerImpl

@Slf4j  
@RequiredArgsConstructor  
@Service  
public class BeerOrderManagerImpl implements BeerOrderManager {  
  
 public static final String *ORDER\_ID\_HEADER* = "ORDER\_ID\_HEADER" ;  
  
 private final StateMachineFactory<BeerOrderStatusEnum, BeerOrderEventEnum> stateMachineFactory;  
 private final BeerOrderRepository beerOrderRepository;  
 private final BeerOrderStateChangeInterceptor beerOrderStateChangeInterceptor;  
 private final EntityManager entityManager;  
  
 @Transactional  
 @Override  
 public BeerOrder newBeerOrder(BeerOrder beerOrder) {  
 beerOrder.setId(null);  
 beerOrder.setOrderStatus(BeerOrderStatusEnum.*NEW*);  
  
 BeerOrder savedBeerOrder = beerOrderRepository.save(beerOrder);  
 sendBeerOrderEvent(savedBeerOrder, BeerOrderEventEnum.*VALIDATE\_ORDER*);  
  
 return savedBeerOrder;  
 }  
  
 @Transactional //while testing lazy initialization error was coming (vn:238)  
 @Override  
 public void processValidationResult(UUID beerOrderId, Boolean isValid) {

-----------------------------------------------------------------------------  
 //In BeerOrderStateChangeInterceptor we are doing saveAndFlush() which is running in different thread, so here  
 //status is not changing, to avoid this  
 // we will flush entity manager first, so the session is flushed and things are written into database.  
 entityManager.flush();  
 -----------------------------------------------------------------------------  
  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById(beerOrderId);  
  
 beerOrderOptional.ifPresentOrElse(beerOrder -> {  
 if(isValid){  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*VALIDATION\_PASSED*);  
  
 //fetching new object bcz beerOrder becomes stale object  
 BeerOrder validatedOrder = beerOrderRepository.findById(beerOrderId).get();  
 sendBeerOrderEvent(validatedOrder, BeerOrderEventEnum.*ALLOCATE\_ORDER*);  
 }else{  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*VALIDATE\_ORDER*);  
 }  
 }, ()-> *log*.error("order not found, Id:"+ beerOrderId));  
 }  
  
 @Override  
 public void beerOrderAllocationPassed(BeerOrderDto beerOrderDto) {  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById(beerOrderDto.getId());  
  
 beerOrderOptional.ifPresentOrElse(beerOrder ->{  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*ALLOCATION\_SUCCESS*);  
 updateAllocatedQty(beerOrderDto);  
 }, ()-> *log*.error("OrderId not found:"+ beerOrderDto.getId()) );  
 }  
  
 @Override  
 public void beerOrderAllocationPendingInventory(BeerOrderDto beerOrderDto) {  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById(beerOrderDto.getId());  
  
 beerOrderOptional.ifPresentOrElse(beerOrder ->{  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*ALLOCATION\_NO\_INVENTORY*);  
 updateAllocatedQty(beerOrderDto);  
 }, ()-> *log*.error("OrderId not found:"+ beerOrderDto.getId()) );  
 }  
  
 private void updateAllocatedQty(BeerOrderDto beerOrderDto) {  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById(beerOrderDto.getId());  
  
 beerOrderOptional.ifPresentOrElse(allocatedOrder ->{  
 allocatedOrder.getBeerOrderLines().forEach(beerOrderLine -> {  
 beerOrderDto.getBeerOrderLines().forEach(beerOrderLineDto -> {  
 if(beerOrderLine.getId().equals(beerOrderLineDto.getId())) {  
 beerOrderLine.setQuantityAllocated(beerOrderLineDto.getQuantityAllocated());  
 }  
 });  
 });  
 beerOrderRepository.saveAndFlush(allocatedOrder);  
 }, ()-> *log*.error("OrderId not found:"+ beerOrderDto.getId()) );  
  
 }  
  
 @Override  
 public void beerOrderAllocationFailed(BeerOrderDto beerOrderDto) {  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById(beerOrderDto.getId());  
 beerOrderOptional.ifPresentOrElse(beerOrder ->{  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*ALLOCATION\_FAILED*);  
 }, ()-> *log*.error("OrderId not found:"+ beerOrderDto.getId()) );  
 }  
  
 @Override  
 public void beerOrderPickedUp(UUID id) {  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById(id);  
  
 beerOrderOptional.ifPresentOrElse(beerOrder ->{  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*BEERORDER\_PICKED\_UP*);  
 }, ()-> *log*.error("OrderId not found:"+ id) );  
 }  
  
 -----------------------------------------------------------------------------  
 @Override  
 public void cancelOrder(UUID id) {  
 beerOrderRepository.findById(id).ifPresentOrElse(beerOrder ->{  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*CANCEL\_ORDER*);  
 }, ()-> *log*.error("OrderId not found:"+ id) );  
 }  
 -----------------------------------------------------------------------------

private void sendBeerOrderEvent(BeerOrder beerOrder, BeerOrderEventEnum eventEnum){  
 StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> sm = build(beerOrder);  
 Message msg = MessageBuilder.*withPayload*(eventEnum)  
 .setHeader(*ORDER\_ID\_HEADER*, beerOrder.getId())  
 .build();  
 sm.sendEvent(msg);  
 }  
  
 private StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> build(BeerOrder beerOrder){  
 StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> sm = stateMachineFactory.getStateMachine(beerOrder.getId());  
 sm.stop();  
 sm.getStateMachineAccessor()  
 .doWithAllRegions(sma->{  
 sma.addStateMachineInterceptor(beerOrderStateChangeInterceptor);  
 sma.resetStateMachine(new DefaultStateMachineContext<>(beerOrder.getOrderStatus(),null, null,null));  
 });  
 sm.start();  
 return sm;  
 }  
}

Changes in BeerOrderValidationListener

@Slf4j  
@RequiredArgsConstructor  
@Component  
public class BeerOrderValidationListener {  
 private final JmsTemplate jmsTemplate;  
  
 @JmsListener(destination = JmsConfig.*VALIDATE\_ORDER\_QUEUE*)  
 public void list(Message msg){  
  
 boolean isValid = true;  
 boolean sendResponse = true;  
  
 ValidateOrderRequest request = (ValidateOrderRequest) msg.getPayload();  
  
 //condition to fail validation  
 if(request.getBeerOrderDto().getCustomerRef() != null){  
 if(request.getBeerOrderDto().getCustomerRef().equals("fail-validation")){  
 isValid=false;  
 }else if(request.getBeerOrderDto().getCustomerRef().equals("dont-validate")){  
 sendResponse = false;  
 }  
 }  
  
 System.*out*.println("########### I RAN ########");  
  
 if(sendResponse){  
 jmsTemplate.convertAndSend(JmsConfig.*VALIDATE\_ORDER\_RESPONSE\_QUEUE*,  
 ValidateOrderResult.*builder*()  
 .isValid(isValid)  
 .orderId(request.getBeerOrderDto().getId())  
 .build());  
 }  
  
 }  
}

Changes in BeerOrderAllocationListener

@Slf4j  
@RequiredArgsConstructor  
@Component  
public class BeerOrderAllocationListener {  
  
 private final JmsTemplate jmsTemplate;  
  
 @JmsListener(destination = JmsConfig.*ALLOCATE\_ORDER\_QUEUE*)  
 public void listen(Message msg){  
 AllocateOrderRequest request = (AllocateOrderRequest) msg.getPayload();  
 boolean pendingInventory = false;  
 boolean allocationError = false;  
 boolean sendResponse = true;  
  
 //condition to fail validation  
 if(request.getBeerOrderDto().getCustomerRef() != null){  
 if(request.getBeerOrderDto().getCustomerRef().equals("fail-allocation")){  
 allocationError=true;  
 }else if(request.getBeerOrderDto().getCustomerRef().equals("dont-allocate")){  
 sendResponse = false;  
 }else if(request.getBeerOrderDto().getCustomerRef().equals("partial-allocation")){  
 pendingInventory=true;  
 }  
 }  
  
// -Code Before-  
// request.getBeerOrderDto().getBeerOrderLines().forEach(beerOrderLineDto -> {  
// beerOrderLineDto.setQuantityAllocated(beerOrderLineDto.getOrderQuantity());  
// });  
 boolean finalPendingInventory = pendingInventory;  
 request.getBeerOrderDto().getBeerOrderLines().forEach(beerOrderLineDto -> {  
 if(finalPendingInventory){  
 beerOrderLineDto.setQuantityAllocated(beerOrderLineDto.getOrderQuantity()-1);  
 }else{  
 beerOrderLineDto.setQuantityAllocated(beerOrderLineDto.getOrderQuantity()-1);  
 }  
 });  
  
 if(sendResponse){  
 jmsTemplate.convertAndSend(JmsConfig.*ALLOCATE\_ORDER\_RESPONSE\_QUEUE*,  
 AllocateOrderResult.*builder*()  
 .beerOrderDto(request.getBeerOrderDto())  
 .pendingInventory(pendingInventory)  
 .allocationError(allocationError)  
 .build());  
 }  
 }  
}

Test Case Generation

package guru.sfg.beer.order.service.services;  
  
import com.fasterxml.jackson.core.JsonProcessingException;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.github.jenspiegsa.wiremockextension.WireMockExtension;  
import com.github.tomakehurst.wiremock.WireMockServer;  
import guru.sfg.beer.order.service.config.JmsConfig;  
import guru.sfg.beer.order.service.domain.BeerOrder;  
import guru.sfg.beer.order.service.domain.BeerOrderLine;  
import guru.sfg.beer.order.service.domain.BeerOrderStatusEnum;  
import guru.sfg.beer.order.service.domain.Customer;  
import guru.sfg.beer.order.service.repositories.BeerOrderRepository;  
import guru.sfg.beer.order.service.repositories.CustomerRepository;  
import guru.sfg.beer.order.service.services.beer.BeerServiceImpl;  
import guru.sfg.brewery.model.BeerDto;  
import guru.sfg.brewery.model.events.AllocationFailureEvent;  
import guru.sfg.brewery.model.events.DeallocateOrderRequest;  
import org.assertj.core.api.Assertions;  
import org.junit.Test;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.extension.ExtendWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.boot.test.context.TestConfiguration;  
import org.springframework.context.annotation.Bean;  
import org.springframework.jms.core.JmsTemplate;  
  
import java.util.HashSet;  
import java.util.Set;  
import java.util.UUID;  
  
import static com.github.jenspiegsa.wiremockextension.ManagedWireMockServer.*with*;  
import static com.github.tomakehurst.wiremock.client.WireMock.*get*;  
import static com.github.tomakehurst.wiremock.client.WireMock.*okJson*;  
import static com.github.tomakehurst.wiremock.core.WireMockConfiguration.*wireMockConfig*;  
import static org.awaitility.Awaitility.*await*;  
import static org.junit.Assert.*assertEquals*;  
import static org.junit.Assert.*assertNotNull*;  
  
@ExtendWith(WireMockExtension.class)  
@SpringBootTest  
public class BeerOrderManagerImplIT {  
  
 @Autowired  
 BeerOrderManager beerOrderManager;  
  
 @Autowired  
 BeerOrderRepository beerOrderRepository;  
  
 @Autowired  
 CustomerRepository customerRepository;  
  
// @MockBean  
// BeerOrderManager beerOrderManager;  
//  
// @MockBean  
// BeerOrderRepository beerOrderRepository;  
//  
// @MockBean  
// CustomerRepository customerRepository;  
  
 @Autowired  
 WireMockServer wireMockServer;  
  
 @Autowired  
 ObjectMapper objectMapper;  
  
 @Autowired  
 JmsTemplate jmsTemplate;  
  
 Customer testCustomer;  
  
 UUID beerId = UUID.*randomUUID*();  
  
 @TestConfiguration  
 static class RestTemplateBuilderProvider {  
 @Bean(destroyMethod = "stop")  
 public WireMockServer wireMockServer(){  
 WireMockServer server = *with*(*wireMockConfig*().port(8083));  
 server.start();  
 return server;  
 }  
 }  
  
 @BeforeEach  
 void setup(){  
 testCustomer = customerRepository.save(Customer.*builder*()  
 .customerName("Test Customer")  
 .build());  
 }  
  
 @Test  
 public void testNewToAllocated() throws JsonProcessingException, InterruptedException {  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 BeerOrderLine line = foundOrder.getBeerOrderLines().iterator().next();  
 *assertEquals*(line.getOrderQuantity(), line.getQuantityAllocated());  
 });  
  
 BeerOrder savedBeerOrder2 = beerOrderRepository.findById(savedBeerOrder.getId()).get();  
  
 *assertNotNull*(savedBeerOrder);  
 *assertEquals*(BeerOrderStatusEnum.*ALLOCATED*, savedBeerOrder2.getOrderStatus());  
 savedBeerOrder2.getBeerOrderLines().forEach(line->{  
 *assertEquals*(line.getOrderQuantity(), line.getQuantityAllocated());  
 });  
 }  
  
 @Test  
 public void testNewToPickedUp() throws JsonProcessingException {  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*ALLOCATED*, foundOrder.getOrderStatus());  
 });  
  
 beerOrderManager.beerOrderPickedUp(savedBeerOrder.getId());  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*PICKED\_UP*, foundOrder.getOrderStatus());  
 });  
  
 BeerOrder pickedUpOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
  
 *assertEquals*(BeerOrderStatusEnum.*PICKED\_UP*, pickedUpOrder.getOrderStatus());  
 }  
  
 @Test  
 public void testFailedValidation() throws JsonProcessingException{  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
 beerOrder.setCustomerRef("fail-validation");  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*VALIDATION\_EXCEPTION*, foundOrder.getOrderStatus());  
 });  
 }  
  
 @Test  
 public void testAllocationFailure() throws JsonProcessingException {  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
 beerOrder.setCustomerRef("fail-allocation");  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*ALLOCATION\_EXCEPTION*, foundOrder.getOrderStatus());  
 });  
  
 AllocationFailureEvent allocationFailureEvent = (AllocationFailureEvent) jmsTemplate.receiveAndConvert(JmsConfig.*ALLOCATE\_FAILURE\_QUEUE*);  
 *assertNotNull*(allocationFailureEvent);  
 Assertions.*assertThat*(allocationFailureEvent.getOrderId()).isEqualTo(savedBeerOrder.getId());  
 }  
  
 @Test  
 public void testPartialAllocation() throws JsonProcessingException {  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
 beerOrder.setCustomerRef("partial-allocation");  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*PENDING\_INVENTORY*, foundOrder.getOrderStatus());  
 });  
 }

**--------------------------------------------------------------------------------**  
 @Test  
 public void testValidationPendingToCancel() throws JsonProcessingException {  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
 beerOrder.setCustomerRef("dont-validate");  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*VALIDATION\_PENDING*, foundOrder.getOrderStatus());  
 });  
  
 beerOrderManager.cancelOrder(savedBeerOrder.getId());  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*CANCELLED*, foundOrder.getOrderStatus());  
 });  
 }  
  
 @Test  
 public void testAllocationPendingToCancel() throws JsonProcessingException {  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
 beerOrder.setCustomerRef("dont-allocate");  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*ALLOCATION\_PENDING*, foundOrder.getOrderStatus());  
 });  
  
 beerOrderManager.cancelOrder(savedBeerOrder.getId());  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*CANCELLED*, foundOrder.getOrderStatus());  
 });  
 }  
  
 @Test  
 public void testAllocatedToCancel() throws JsonProcessingException {  
 BeerDto beerDto = BeerDto.*builder*().id(beerId).upc("12345").build();  
  
 wireMockServer.stubFor(*get*(BeerServiceImpl.*BEER\_UPC\_PATH\_V1* + "12345")  
 .willReturn(*okJson*(objectMapper.writeValueAsString(beerDto))));  
  
 BeerOrder beerOrder = createBeerOrder();  
  
 BeerOrder savedBeerOrder = beerOrderManager.newBeerOrder(beerOrder);  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*ALLOCATED*, foundOrder.getOrderStatus());  
 });  
  
 beerOrderManager.cancelOrder(savedBeerOrder.getId());  
  
 *await*().untilAsserted(() -> {  
 BeerOrder foundOrder = beerOrderRepository.findById(beerOrder.getId()).get();  
 *assertEquals*(BeerOrderStatusEnum.*CANCELLED*, foundOrder.getOrderStatus());  
 });  
  
 DeallocateOrderRequest deallocateOrderRequest = (DeallocateOrderRequest) jmsTemplate.receiveAndConvert(JmsConfig.*DEALLOCATE\_ORDER\_QUEUE*);  
 *assertNotNull*(deallocateOrderRequest);  
 Assertions.*assertThat*(deallocateOrderRequest.getBeerOrderDto().getId()).isEqualTo(savedBeerOrder.getId());  
 }

**--------------------------------------------------------------------------------**  
  
 public BeerOrder createBeerOrder(){  
 BeerOrder beerOrder = BeerOrder.*builder*()  
 .customer(testCustomer)  
 .build();  
  
 Set<BeerOrderLine> lines = new HashSet<>();  
 lines.add(BeerOrderLine.*builder*()  
 .beerId(beerId)  
 .orderQuantity(1)  
 .beerOrder(beerOrder)  
 .upc("12345")  
 .build());  
 beerOrder.setBeerOrderLines(lines);  
 return beerOrder;  
 }  
}

**Note:**

Can you think of other things that would need a compensating transaction?

Yes - any step that could encounter an error or other unexpected condition should have a compensating transaction.

5.5 Refactoring code for Persisting status change (remove EntityManager)

This is not the best way

Changes in BeerOrderManagerImpl

package guru.sfg.beer.order.service.services;  
  
import guru.sfg.beer.order.service.domain.BeerOrder;  
import guru.sfg.beer.order.service.domain.BeerOrderEventEnum;  
import guru.sfg.beer.order.service.domain.BeerOrderStatusEnum;  
import guru.sfg.beer.order.service.repositories.BeerOrderRepository;  
import guru.sfg.beer.order.service.sm.BeerOrderStateChangeInterceptor;  
import guru.sfg.brewery.model.BeerOrderDto;  
import lombok.RequiredArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.messaging.Message;  
import org.springframework.messaging.support.MessageBuilder;  
import org.springframework.statemachine.StateMachine;  
import org.springframework.statemachine.config.StateMachineFactory;  
import org.springframework.statemachine.support.DefaultStateMachineContext;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.util.Optional;  
import java.util.UUID;  
import java.util.concurrent.atomic.AtomicBoolean;  
import java.util.concurrent.atomic.AtomicInteger;  
  
@Slf4j  
@RequiredArgsConstructor  
@Service  
public class BeerOrderManagerImpl implements BeerOrderManager {  
  
 public static final String *ORDER\_ID\_HEADER* = "ORDER\_ID\_HEADER" ;  
  
 private final StateMachineFactory<BeerOrderStatusEnum, BeerOrderEventEnum> stateMachineFactory;  
 private final BeerOrderRepository beerOrderRepository;  
 private final BeerOrderStateChangeInterceptor beerOrderStateChangeInterceptor;  
  
 @Transactional  
 @Override  
 public BeerOrder newBeerOrder(BeerOrder beerOrder) {  
 beerOrder.setId(null);  
 beerOrder.setOrderStatus(BeerOrderStatusEnum.*NEW*);  
  
 BeerOrder savedBeerOrder = beerOrderRepository.save(beerOrder);  
 sendBeerOrderEvent(savedBeerOrder, BeerOrderEventEnum.*VALIDATE\_ORDER*);  
  
 return savedBeerOrder;  
 }  
  
 @Transactional //while testing lazy initialization error was coming (vn:238)  
 @Override  
 public void processValidationResult(UUID beerOrderId, Boolean isValid) {  
  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById(beerOrderId);  
  
 beerOrderOptional.ifPresentOrElse(beerOrder -> {  
 if(isValid){  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*VALIDATION\_PASSED*);  
  
 //wait for status change  
 awaitForStatus(beerOrderId, BeerOrderStatusEnum.*VALIDATED*);  
  
 //fetching new object bcz beerOrder becomes stale object  
 BeerOrder validatedOrder = beerOrderRepository.findById(beerOrderId).get();  
 sendBeerOrderEvent(validatedOrder, BeerOrderEventEnum.*ALLOCATE\_ORDER*);  
 }else{  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*VALIDATE\_ORDER*);  
 }  
 }, ()-> *log*.error("order not found, Id:"+ beerOrderId));  
 }  
  
 @Override  
 public void beerOrderAllocationPassed(BeerOrderDto beerOrderDto) {  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById(beerOrderDto.getId());  
  
 beerOrderOptional.ifPresentOrElse(beerOrder ->{  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*ALLOCATION\_SUCCESS*);  
 awaitForStatus(beerOrder.getId(), BeerOrderStatusEnum.*ALLOCATED*);  
 updateAllocatedQty(beerOrderDto);  
 }, ()-> *log*.error("OrderId not found:"+ beerOrderDto.getId()) );  
 }  
  
 @Override  
 public void beerOrderAllocationPendingInventory(BeerOrderDto beerOrderDto) {  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById(beerOrderDto.getId());  
  
 beerOrderOptional.ifPresentOrElse(beerOrder ->{  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*ALLOCATION\_NO\_INVENTORY*);  
 awaitForStatus(beerOrder.getId(), BeerOrderStatusEnum.*PENDING\_INVENTORY*);  
 updateAllocatedQty(beerOrderDto);  
 }, ()-> *log*.error("OrderId not found:"+ beerOrderDto.getId()) );  
 }  
  
 private void updateAllocatedQty(BeerOrderDto beerOrderDto) {  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById(beerOrderDto.getId());  
  
 beerOrderOptional.ifPresentOrElse(allocatedOrder ->{  
 allocatedOrder.getBeerOrderLines().forEach(beerOrderLine -> {  
 beerOrderDto.getBeerOrderLines().forEach(beerOrderLineDto -> {  
 if(beerOrderLine.getId().equals(beerOrderLineDto.getId())) {  
 beerOrderLine.setQuantityAllocated(beerOrderLineDto.getQuantityAllocated());  
 }  
 });  
 });  
 beerOrderRepository.saveAndFlush(allocatedOrder);  
 }, ()-> *log*.error("OrderId not found:"+ beerOrderDto.getId()) );  
  
 }  
  
 @Override  
 public void beerOrderAllocationFailed(BeerOrderDto beerOrderDto) {  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById(beerOrderDto.getId());  
 beerOrderOptional.ifPresentOrElse(beerOrder ->{  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*ALLOCATION\_FAILED*);  
 }, ()-> *log*.error("OrderId not found:"+ beerOrderDto.getId()) );  
 }  
  
 @Override  
 public void beerOrderPickedUp(UUID id) {  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById(id);  
  
 beerOrderOptional.ifPresentOrElse(beerOrder ->{  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*BEERORDER\_PICKED\_UP*);  
 }, ()-> *log*.error("OrderId not found:"+ id) );  
 }  
  
 //new  
 @Override  
 public void cancelOrder(UUID id) {  
 beerOrderRepository.findById(id).ifPresentOrElse(beerOrder ->{  
 sendBeerOrderEvent(beerOrder, BeerOrderEventEnum.*CANCEL\_ORDER*);  
 }, ()-> *log*.error("OrderId not found:"+ id) );  
 }

**-------------------------------------------------------------------------------------------------------**  
 private void awaitForStatus(UUID beerOrderId, BeerOrderStatusEnum statusEnum){  
  
 AtomicBoolean found = new AtomicBoolean(false);  
 AtomicInteger loopCount = new AtomicInteger(0);  
  
 while (!found.get()){  
 if(loopCount.get()>10){  
 found.set(true);  
 *log*.debug("Loop retires exceeded");  
 }  
 beerOrderRepository.findById(beerOrderId).ifPresentOrElse(beerOrder -> {  
 if(beerOrder.getOrderStatus().equals(statusEnum)){  
 found.set(true);  
 *log*.debug("Order found");  
 }else{  
 *log*.debug("Status order not equal, Expected "+statusEnum.name()+" Found "+beerOrder.getOrderStatus());  
 }  
 }, ()->{  
 *log*.debug("Order Id not found");  
 });  
 if(!found.get()){  
 try {  
 *log*.debug("sleeping for debug");  
 Thread.*sleep*(100);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }

**---------------------------------------------------------------------------------------------------**  
  
 private void sendBeerOrderEvent(BeerOrder beerOrder, BeerOrderEventEnum eventEnum){  
 StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> sm = build(beerOrder);  
 Message msg = MessageBuilder.*withPayload*(eventEnum)  
 .setHeader(*ORDER\_ID\_HEADER*, beerOrder.getId())  
 .build();  
 sm.sendEvent(msg);  
 }  
  
 private StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> build(BeerOrder beerOrder){  
 StateMachine<BeerOrderStatusEnum, BeerOrderEventEnum> sm = stateMachineFactory.getStateMachine(beerOrder.getId());  
 sm.stop();  
 sm.getStateMachineAccessor()  
 .doWithAllRegions(sma->{  
 sma.addStateMachineInterceptor(beerOrderStateChangeInterceptor);  
 sma.resetStateMachine(new DefaultStateMachineContext<>(beerOrder.getOrderStatus(),null, null,null));  
 });  
 sm.start();  
 return sm;  
 }  
}