



Lottery #maven

JMS, Message Driven Bean, JMX, Singleton Session Bean

Óbuda University, Java Enterprise Edition

John von Neumann Faculty of Informatics

Lab 6

Dávid Bedők
2018-01-17
v1.0



Task: let us create an Enterprise Java application around the weekly lottery game (numbers from 1 to 90, same probability). The application stored the weekly numbers (with timestamp, current prize pool and the name of the puller).

- ▷ The drawn numbers will be sent via a **JMS queue** (one message contains the five numbers separated by a comma) (machine-to-machine interface).
- ▷ The user can query the latest and all previously lot's results.
- ▷ The service maintains the distribution of the prize pool according to the current rules (e.g.: how much percentage (of the prize pool) does a three hit lottery ticket win?).
- ▷ Via RESTful interface there will be a service which calculates the available prize pool for a given lottery ticket (5 numbers)
- ▷ The application will have a **standard management (JMX)** interface through which an actor can fine-tune the application (prize pool, name of the puller, distribution).



- ▷ JBoss Management Console
 - admin/guest user creation
- ▷ **J**ava **M**essage **S**ervice (**JMS**)
 - JMS Destination
 - JMS Queue
 - JMS Topic (out of scope in this task)
 - **M**essage **D**riven **B**ean (**MDB**) (listener)
 - JMS Client
- ▷ **J**ava **M**anagement **eX**tension (**JMX**)
 - Standard **M**anagement **B**ean (MBean)
 - jconsole
- ▷ Singleton Session Bean
 - special Session Bean with state



▷ **lottery** (root project)

- **lot-webservice** (EAR web module)
 - Project for the RESTful services (presentation-tier).
- **lot-ejbservice** (EAR ejb module)
 - Business methods (service-tier)
- **lot-persistence** (EAR ejb module)
 - ORM layer, JPA (data-tier)
- **lot-jmsclient** (standalone)
 - JMS client application
- **lot-jmxclient** (standalone)
 - JMX client application

In case of Maven there is a **lot-ear** project too.

We do not need an e.g.: 'lot-ejb-serviceclient' project for the **JMS client** application, because the communication is standard (we do not need remote interface), and we will send text messages (we do not need stubs either).

- ▷ <http://localhost:9990/>
- ▷ We need an **admin** management user to reach that service (BASIC AUTH)
 - [JBOSS-HOME]/bin/add-user.[bat|sh]
 - Management User (enter)
 - Username: admin (are you sure? yes)
 - Password: AlmafA11#
 - Enter, Yes, Yes, Enter

```
1 > [JBOSS_HOME]/bin/add-user.[bat|sh] admin AlmafA11#
```

- [JBOSS-HOME]/standalone/configuration/mgmt-users.properties
- ▷ **guest** user creation (later it will be needed to authenticate a user, but we will not need any special permissions):
 - ```
1 > [JBOSS_HOME]/bin/add-user.[bat|sh] -a -u jmstestuser
-p User#70365 -g guest
```
  - [JBOSS-HOME]/standalone/configuration/application-users.properties
  - [JBOSS-HOME]/standalone/configuration/application-roles.properties



```
1 CREATE TABLE event (
2 event_id SERIAL NOT NULL,
3 event_puller CHARACTER VARYING(100) NOT NULL,
4 event_prizepool INTEGER NOT NULL,
5 event_date TIMESTAMP WITHOUT TIME ZONE NOT NULL,
6 CONSTRAINT PK_EVENT_ID PRIMARY KEY (event_id)
7);
8
9 ALTER TABLE event OWNER TO postgres;
10
11 CREATE TABLE drawnumber (
12 drawnumber_id SERIAL NOT NULL,
13 drawnumber_event_id INTEGER NOT NULL,
14 drawnumber_value INTEGER NOT NULL,
15 CONSTRAINT PK_DRAWNUMBER_ID PRIMARY KEY (drawnumber_id),
16 CONSTRAINT FK_DRAWNUMBER_EVENT FOREIGN KEY (drawnumber_event_id)
17 REFERENCES event (event_id) MATCH SIMPLE ON UPDATE RESTRICT ON DELETE RESTRICT
18);
19
20 ALTER TABLE drawnumber OWNER TO postgres;
```

create-schema.sql

# List the latest and all the previously lotteries

## RESTful services



- ▷ <http://localhost:8080/lottery/api/service/event/latest>
  - **LotteryRestService** (lot-webservice)
    - EventStub getLatestEvent() throws AdaptorException;
  - **LotteryFacade** (lot-ejbservice)
    - EventStub getLatestEvent() throws AdaptorException;
  - **EventService** (lot-persistence)
    - Event readLatest() throws PersistenceServiceException;
    - ```
1 SELECT e
2 FROM Event e
3   JOIN FETCH e.numbers
4 ORDER BY e.date DESC
```
 - SetMaxResult(1)
 - **Important!** Because of the drawn numbers this query fetches all the events from the database.
 - **EventConverter** (lot-ejbservice): EventStub to(Event event);
- ▷ <http://localhost:8080/lottery/api/service/event/list>
 - List<EventStub>...
 - List<Event>...

JOIN FETCH

repetition

```
1 SELECT
2   event0_.event_id AS event_id1_1_0_,
3   numbers1_.drawnnumber_id AS drawnnum1_0_1_,
4   event0_.event_date AS event_da2_1_0_,
5   event0_.event_prizepool AS event_pr3_1_0_,
6   event0_.event_puller AS event_pu4_1_0_,
7   numbers1_.drawnnumber_event_id AS drawnnum3_0_1_,
8   numbers1_.drawnnumber_value AS drawnnum2_0_1_,
9   numbers1_.drawnnumber_event_id AS drawnnum3_1_0_-,
10  numbers1_.drawnnumber_id AS drawnnum1_0_0_-
11 FROM
12   event event0_
13     INNER JOIN drawnnumber numbers1_ ON
14       event0_.event_id=numbers1_.drawnnumber_event_id
15 ORDER BY
16   event0_.event_date DESC
```

The `Set<DrawnNumber>` is linked in **LAZY** way to the Event entity. The **JOIN** binds the table (in that case the query would be the same except that the columns of the drawnnumber table would be missed), but it will not *attach* the entities (if we walk through the values these will be fetched and bound with separate queries). The **JOIN FETCH** solves this (fetches and attaches the entities) without the additional queries.

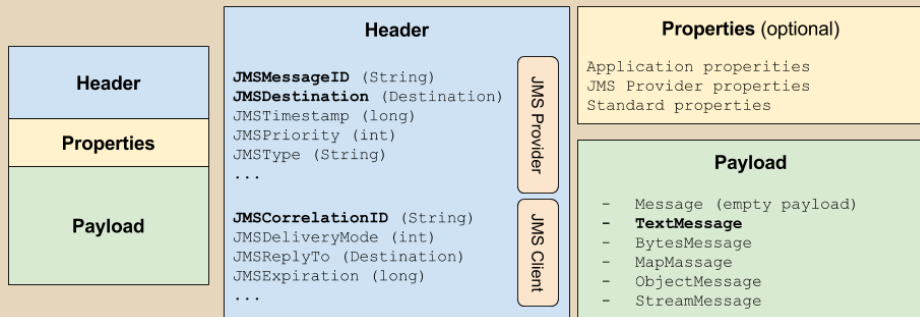
Java Message Service (JMS)

Message-Oriented Middleware (MOM)

- ▷ Message based communication
- ▷ 'loosely' coupled components (there is a component which handles and stores the messages)
- ▷ **JMS 1.1** (2002, JSR914, JEE6)
- ▷ **JMS 2.0** (2013, JSR343, JEE7)
- ▷ Types:
 - **point-to-point (queue)**
 - *producer* sends messages to the queue
 - *consumer* reads out messages from the queue
 - one message is processed by one consumer (*ack* sending)
 - there is no need that the *producer* and the *consumer* be 'online' at the same time
 - **publish-subscribe (topic)**
 - *publisher* sends messages to the topic
 - *subscriber(s)* get(s) the sent messages
 - one message can be processed by multiple consumers
 - there is a time dependency between the *publisher* and the *subscriber* (but there are special kind of subscriptions as well)

- ▷ <http://hornetq.jboss.org/>
- ▷ Deprecated, from JBoss 7.x the **JBoss A-MQ** will be used
 - <http://www.jboss.org/products/amq/overview/>
- ▷ **JMS 1.1** and **JMS 2.0** support
- ▷ Version: **2.3.25.Final** (in case of JBoss 6.4)

JMS message structure



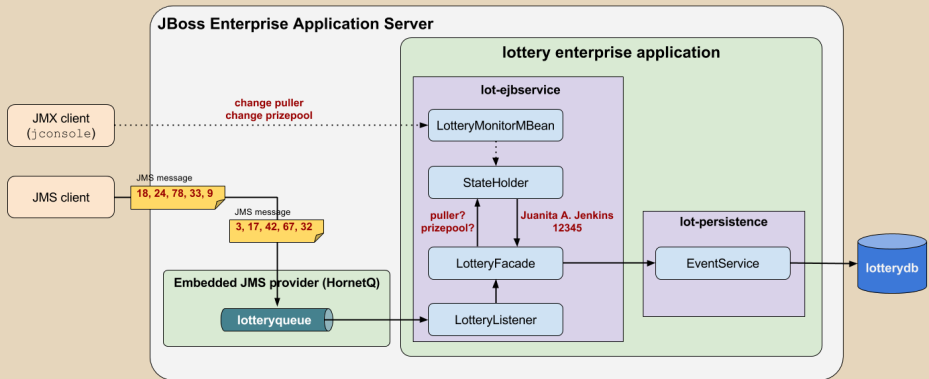
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <messaging-deployment xmlns="urn:jboss:messaging-deployment:1.0">
3   <hornetq-server>
4     <jms-destinations>
5       <jms-queue name="lotteryqueue">
6         <entry name="jms/queue/lotteryqueue" />
7         <entry name="java:jboss/exported/jms/queue/lotteryqueue"
8           />
9       </jms-queue>
10    </jms-destinations>
11  </hornetq-server>
12 </messaging-deployment>
```

The JNDI name will be `java:/jms/queue/lotteryqueue`. The `reamote` JMS client will put the `java:jboss/exported/` prefix automatically (in case of using the JBoss JMS Client jar).

lotteryqueue-jms.xml

The name of the file must end with `*-jms.xml` and we have to copy it into the `deployments` directory to create the destination. Another solution could be defining via the `standalone.xml`, or we can create a queue/topic in a programmed way at *runtime*.

Create a new event



```
1 package hu.qwaevisz.lottery.ejbservice.listener;
2 [..]
3 @MessageDriven(name = "LotteryListener", activationConfig = { //
4     @ActivationConfigProperty(propertyName = "destinationType", propertyValue =
5         "javax.jms.Queue"),
6     @ActivationConfigProperty(propertyName = "destination", propertyValue =
7         "lotteryqueue"),
8     @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue =
9         "Auto-acknowledge") })
10 public class LotteryListener implements MessageListener {
11
12     @EJB
13     private LotteryFacade facade;
14
15     @EJB
16     private MessageConverter converter;
17
18     @PostConstruct
19     public void initialize() {
20         LOGGER.info("Lottery Listener created...");
21     }
22
23     @Override
24     public void onMessage(final Message message) {
25         [..]
26     }
27 }
```

When a message arrives into the lotteryqueue, the LotteryListener MDB will be activated and the onMessage() method will be called with the received message. If the method throws an exception the message processing will be rolled back and the message will not be removed from the queue by default!

LotteryListener

- ▷ **queue name**: extractable (it is useful when a *listener* handles multiple destinations)
- ▷ **correlation id**: typically it is set by the client to identify e.g. the async response of the sent message

```
1 @Override
2 public void onMessage(final Message message) {
3     try {
4         if (LOGGER.isDebugEnabled()) {
5             final Queue destination = (Queue) message.getJMSDestination();
6             final String queueName = destination.getQueueName();
7             LOGGER.debug("New JMS message arrived into " + queueName + " queue (correlation
8                 id: " + message.getJMSCorrelationID() + ")");
9         }
10        if (message instanceof TextMessage) {
11            final TextMessage textMessage = (TextMessage) message;
12            String content = textMessage.getText();
13            if (LOGGER.isDebugEnabled()) {
14                LOGGER.debug("Received message: " + content);
15            }
16            this.facade.createNewEvent(this.converter.toNumbers(content));
17        } else {
18            LOGGER.error("Received message is not a TextMessage (" + message + ")");
19        }
20    } catch (final JMSEException | AdaptorException | NumberFormatException e) {
21        LOGGER.error(e, e);
22    }
```

- ▷ BytesMessage
- ▷ MapMessage
- ▷ ObjectMessage
- ▷ StreamMessage
- ▷ TextMessage

RedHat JBoss vs. Oracle WebLogic

	JBoss 6.4	WebLogic 12.2.1
JavaEE version	JavaEE 6	JavaEE 6
Logging	JBoss Logging, Log4J, ..	JDK Logging, ..
RESTful	RESTEasy 2.3.10.Final	Jersey 1.18
JPA Provider	Hibernate 4.2.18.Final	EclipseLink 2.5.2
JMS Provider	HornetQ 2.3.25.Final	WebLogic JMS
JAXB	JAXB 2.2.5-redhat-9	JAXB 2.2.10 Oracle

WebLogic Administration Console

▷ Services | Messaging | JMS Servers

- New

- name: **demoJMSserver**
- persistence store: none
- target: myserver

▷ Services | Messaging | JMS Modules

- New

- name: **demoJMSmodule**
- location in domain: blank
- target: myserver
- Would you like to add resources to this JMS system module? → yes

WebLogic Administration Console

- ▷ Services | Messaging | JMS Modules
 - demoJMSmodule | Subdeployments | New
 - name: **demoJMSsubmodule**
 - target/server: demoJMSserver
 - target: myserver
 - Would you like to add resources to this JMS system module? → yes

- ▷ Services | Messaging | JMS Modules
 - demoJMSmodule | Configuration | New
 - tpye: **Queue**
 - name: **lotteryqueue**
 - JNDI name: **jms/queue/lotteryqueue**
 - subdeployments: demoJMSsubmodule
 - target/server: demoJMSserver

WebLogic Administration Console

- ▷ Services | Messaging | JMS Modules
 - demoJMSmodule | Configuration | New
 - type: **Connection Factory**
 - name: **demoConnectionFactory**
 - JNDI name: **jms/demoConnectionFactory**
 - target: myserver

```
1 package hu.qwaevisz.lottery.ejbsservice.listener;
2 [...]
3 @MessageDriven(name = "LotteryListener", activationConfig = { //
4     @ActivationConfigProperty(propertyName = "initialContextFactory", propertyValue =
5         "weblogic.jndi.WLInitialContextFactory"),
6     @ActivationConfigProperty(propertyName = "destinationType", propertyValue =
7         "javax.jms.Queue"),
8     @ActivationConfigProperty(propertyName = "destinationJndiName", propertyValue =
9         "jms/queue/lotteryqueue"),
10    @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue =
11        "Auto-acknowledge") })
12 public class LotteryListener implements MessageListener, MessageDrivenBean {
13     [...]
14 }
```

LotteryListener.java

Differences

- ▷ Implementing the `MessageDrivenBean` does not required, but it opens the opportunity to intervene the lifecycle (`setMessageDrivenContext(..)` and `ejbRemove()` methods).
- ▷ `initialContextFactory` property (primarily it is useful in case of outer JMS provider)
- ▷ usage of `destinationJndiName` property instead of `destination` (latter it is good for JBoss, and it contains the JBoss specific *friendly name*)

JMS Client Application

lot-jmsclient project

It sends JMS message to the `lotteryqueue`. This message will be handled by the **HornetQ** JMS MOM. HornetQ is started by JBoss EAS.

Important!

We have to communicate with the *JMS Provider*, but HornetQ is an **embedded JMS Provider** so at first we have to reach JBoss's JNDI tree (via the `InitialContext`, like in case of the Remote EJB). In case of external *JMS Provider* these rules are not necessary.

We need the following to build communication:

▷ **JBoss initial context factory**

- required resources on the classpath:
`org.jboss.naming.remote.client.InitialContextFactory`
- compile group:
`'org.jboss.as', name: 'jboss-as-jms-client-bom', version: '7.2.0.Final'`

▷ Host (**localhost**) and remote port (def.: **4447**) of the JBoss EAS

- `standalone.xml` | `socket-binding-group` | `remoting socket-binding port: 4447`

▷ **JNDI name of the JMS Connection Factory** (`standalone.xml: jms/RemoteConnectionFactory`)

▷ Successful authentication a user with at least guest role (**username** and **password**)

▷ The target **JNDI name of the queue** (`lotteryqueue-jms.xml: jms/queue/lotteryqueue`)

▷ If we would like to send `ObjectMessage` instead of `TextMessage`, we would need a `'serviceclient.jar'` as well which contains the Serializable DTOs (like in case of the remote ejb client).

Messaging subsystem

standalone-full.xml



```
1 <subsystem xmlns="urn:jboss:domain:messaging:1.4">
2   <hornetq-server>
3     [..]
4     <connectors>[..]</connectors>
5     <acceptors>[..]</acceptors>
6     <security-settings>
7       <security-setting match="#">
8         <permission type="send" roles="guest"/>
9         <permission type="consume" roles="guest"/>
10        [..]
11      </security-setting>
12    </security-settings>
13    <address-settings>[..]</address-settings>
14    <jms-connection-factories>
15      <connection-factory name="InVmConnectionFactory">[..]</connection-factory>
16      <connection-factory name="RemoteConnectionFactory">
17        <connectors>[..]</connectors>
18        <entries>
19          <entry name="java:jboss/exported/jms/RemoteConnectionFactory"/>
20        </entries>
21      </connection-factory>
22      <pooled-connection-factory
23        name="hornetq-ra">[..]</pooled-
24    </jms-connection-factories>
25    <jms-destinations>[..]</jms-destinations
26  </hornetq-server>
</subsystem>
```

Because of the default security setting only the authenticated - at least **guest** - users can send/receive messages. The authentication process is managed by JBoss.

To build the *remote* connection we need the RemoteConnectionFactory, which we can get from the JNDI tree with the **jms/RemoteConnectionFactory** JNDI name.

standalone.xml

```
1 final Properties environment = new Properties();
2 environment.put(Context.INITIAL_CONTEXT_FACTORY,
3     "org.jboss.naming.remote.client.InitialContextFactory");
4 environment.put(Context.PROVIDER_URL, "remote://localhost:4447");
5 environment.put(Context.SECURITY_PRINCIPAL, "jmsstestuser");
6 environment.put(Context.SECURITY_CREDENTIALS, "User#70365");
7 final Context context = new InitialContext(environment);
8 final ConnectionFactory connectionFactory = (ConnectionFactory)
9     context.lookup("jms/RemoteConnectionFactory");
10 final Destination destination = (Destination) context.lookup("jms/queue/lotteryqueue");
11 final Connection connection = connectionFactory.createConnection("jmsstestuser",
12     "User#70365");
13 final Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
14 final MessageProducer producer = session.createProducer(destination);
15 connection.start();
16 final TextMessage textMessage = session.createTextMessage("1, 2, 3, 4, 5");
17 producer.send(textMessage);
18 connection.close();
```

SimpleClient.java

```
1 final Properties environment = new Properties();
2 environment.put(Context.INITIAL_CONTEXT_FACTORY,
3     "weblogic.jndi.WLInitialContextFactory");
4 environment.put(Context.PROVIDER_URL, "t3://localhost:7001");
5 environment.put(Context.SECURITY_PRINCIPAL, "weblogic");
6 environment.put(Context.SECURITY_CREDENTIALS, "AlmafA1#");
7 final Context context = new InitialContext(environment);
8 final ConnectionFactory connectionFactory = (ConnectionFactory)
9     context.lookup("jms/demoConnectionFactory");
10 final Destination destination = (Destination) context.lookup("jms/queue/lotteryqueue");
11 final Connection connection = connectionFactory.createConnection("weblogic",
12     "AlmafA1#");
13 final Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
14 final MessageProducer producer = session.createProducer(destination);
15 connection.start();
16 final TextMessage textMessage = session.createTextMessage("1, 2, 3, 4, 5");
17 producer.send(textMessage);
18 connection.close();
```

SimpleClient.java

Remote JMS Client

RedHat JBoss vs. Oracle WebLogic

	JBoss 6.4	WebLogic 12.2.1
Provider URL	remote://localhost:4447 6	t3://localhost:7001
User name	jmstestuser	weblogic
Password	User#70365	AlmafA1#
Destination JNDI	jms/queue/lotteryqueue	jms/queue/lotteryqueue
Initial context factory	org.jboss.naming.remote.client.InitialContextFactory	weblogic.jndi.WLInitialContextFactory
Connection factory JNDI	jms/RemoteConnectionFactory	jms/demoConnectionFactory
Classpath	org.jboss.as:jboss-as-jms-client-bom:7.2.0.Final	[WL-HOME]/wlserver/server/lib/wlthint3client.jar

Authentication

In case of JBoss we created a *guest user*, but in WebLogic we will use the *Admin user* for the same purpose. The Connection Factory comes from the `standalone-full.xml` in JBoss, but in WebLogic we configured it via the *Admin console*.

<http://localhost:8080/lottery/QueueServlet?message=1,2,3,4,5>

```
1  [...]
2  InitialContext context = new InitialContext();
3  QueueConnectionFactory connectionFactory = (QueueConnectionFactory)
   context.lookup("ConnectionFactory");
4  QueueConnection connection = connectionFactory.createQueueConnection();
5  QueueSession session = connection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
6  Queue queue = (Queue) context.lookup("jms/queue/lotteryqueue");
7  QueueSender sender = session.createSender(queue);
8  TextMessage textMessage = session.createTextMessage();
9  connection.start();
10
11 textMessage.setText("1,2,3,4,5");
12 sender.send(textMessage);
13 sender.close();
14 session.close();
15 connection.close();
16 [...]
```

In case of **WebLogic** the only difference is the JNDI name of the *Connection Factory*:
javax.jms.QueueConnectionFactory.
In case of **JBoss** the given JNDI name (ConnectionFactory) comes from the standalone-full.xml (InVmConnectionFactory).

SendQueueServlet.java

Sending a JMS message from inside a container is always simpler, because we can reach the `InitialContext` 'without configuration' (it is already initialized). In that case we can use the 'non-remote' *connection factory* as well.

Singleton nature

The EJB container guarantees that each thread will use the same Singleton Session Bean instance (its state will be the same).

Of course not all of the client (which would like to use the SSB) will stand in one line (it would be a huge bottleneck of the entire system). There are methods which have **READ** or **WRITE** lock (these annotation can be used only in case of **Container-Managed Concurrency (CMC)**).

- ▷ `LockType.READ`: can run simultaneously on multiple threads (state reading)
- ▷ `LockType.WRITE` (def.): only one thread can run at the same time (state writing)

Concurrency Management

Session Beans

Container-Managed Concurrency (CMC) (def.)

`@ConcurrencyManagement(ConcurrencyManagementType.CONTAINER)`

Bean-Managed Concurrency (BMC)

`@ConcurrencyManagement(ConcurrencyManagementType.BEAN)`

Bean managed

Only the **Singleton Session Beans** support the *Bean-Managed Concurrency*. In that case we are able to use the `synchronized` and `volatile` keywords.

StateHolder

Storing and managing the puller, the prize pool and the distribution

```
1 package hu.qwaevisz.lottery.ejb.service.holder;
2 [..]
3 @Singleton(mappedName = "ejb/lotteryState", name = "lotteryState")
4 @ConcurrencyManagement(ConcurrencyManagementType.CONTAINER)
5 public class LotteryStateHolderImpl implements LotteryStateHolder {
6     private String puller;
7     private Integer prizePool;
8     private PrizeDistribution distribution;
9
10    @PostConstruct
11    public void initialize() {
12        this.puller = "Juanita A. Jenkins";
13        this.prizePool = 12345;
14        this.distribution = new PrizeDistribution();
15    }
16
17    @Override
18    @Lock(LockType.READ)
19    public String getCurrentPuller() {
20        return this.puller;
21    }
22
23    @Override
24    @Lock(LockType.WRITE)
25    public void setCurrentPuller(String name) {
26        LOGGER.info("Change Puller: " + name);
27        this.puller = name;
28    }
29    [..]
30 }
```

Implementing the 'business logic' of prizePool is very similar. It is elegant if we create an interface for an SSB as well (LotteryStateHolder), which gets the @Local annotation.

LotteryFacade

lot-ejbservice project

```
1 package hu.qvaevisz.lottery.ejbservice.facade;
2 [...]
3 @Stateless(mappedName = "ejb/lotteryFacade")
4 public class LotteryFacadeImpl implements LotteryFacade {
5
6     private static final Logger LOGGER = Logger.getLogger(LotteryFacadeImpl.class);
7
8     @EJB
9     private EventService eventService;
10
11     @EJB
12     private LotteryStateHolder stateHolder;
13
14     @Override
15     public void createNewEvent(int[] numbers) throws AdaptorException {
16         try {
17             String puller = this.stateHolder.getCurrentPuller();
18             Integer prizePool = this.stateHolder.getCurrentPrizePool();
19             this.eventService.create(puller, prizePool, numbers);
20         } catch (final PersistenceServiceException e) {
21             LOGGER.error(e, e);
22             throw new AdaptorException(e.getLocalizedMessage());
23         }
24     }
25     [...]
26 }
```

In the *persistence* layer the EventService has to create the event and the associated five drawnumber rows in one transaction.

LotteryFacadeImpl.java

EventService

lot-persistence project

Important! Let the cascade value of the @OneToMany annotation be CascadeType.ALL or PERSIST at the Event's Set<DrawnNumber> numbers field.

```
1 package hu.qvaevisz.lottery.persistence.service;
2 [...]
3 @Stateless(mappedName = "ejb/eventService")
4 @TransactionManagement(TransactionManagementType.CONTAINER)
5 @TransactionAttribute(TransactionAttributeType.REQUIRES_NEW)
6 public class EventServiceImpl implements EventService {
7
8     @PersistenceContext(unitName = "lot-persistence-unit")
9     private EntityManager entityManager;
10
11     @Override
12     public void create(String puller, Integer prizePool, int[] numbers) throws
13         PersistenceServiceException {
14         try {
15             final Event event = new Event(puller, prizePool);
16             for (final int number : numbers) {
17                 event.addNumber(number);
18             }
19             this.entityManager.persist(event);
20         } catch (final Exception e) {
21             throw new PersistenceServiceException("Unknown error when persisting Events! " +
22                 e.getLocalizedMessage(), e);
23         }
24     }
25
26     public void addNumber(Integer number) {
27         this.numbers.add(new DrawnNumber(number, this));
28     }
29 }
```

persist: create a new (or a marked as delete) entity, and put the given entity into *managed* state // **merge:** create a *detached* (non managed) entity (the method gives back the *managed* entity, the given instance will not be touched)

EventServiceImpl.java

Java Management eXtension (JMX)

JavaSE

- ▶ The **JMX** technology is part of the JavaSE but of course the JavaEE also supports it to monitor server side components.
- ▶ We need to create **Managed Bean(s)** (**MBean**) which are handled by the *MBean server*.
- ▶ We can write JMX client easily but we can use prewritten application as well because of the standards (e.g.: **jconsole** is a built-in client application of the Java SE).
- ▶ All MBean have to meet for the JMX specification rules.

Rules of MBean creation

- ▷ If the name of the implementation is `Something` class, then the name of the interface should be `SomethingMBean`.
- ▷ We can define **operations** and **attributes** in the MBean(s).
- ▷ Definition of MBean **attribute**:
 - In case of a read-only `A` typed `xyz` attribute there should be an `A getXyz()` method.
 - In case of read- and writeable `A` types `xyz` attribute there should be an `A getXyz()` and a `void setXyz(A value)` method.
 - We cannot use an attribute-related getter/setter method to any other purpose, we cannot use overload getters, we cannot use other typed return values, etc.
- ▷ Definition of MBean **operation**:
 - All the methods which are not standard accessor (getter) or mutator (setter) become an operation. (e.g.: `B getItem(C value)`).
- ▷ In simple cases you can use java **primitives**, **arrays** or **Strings**, but there is some complex type as well (e.g.: `TabularData`).

```
1 package hu.qwaevisz.lottery.ejbsservice.management;
2 [...]
3 public class LotteryMonitor implements LotteryMonitorMBean {
4     @EJB
5     private LotteryStateHolder stateHolder;
6
7     @Override
8     public String getPuller() { return this.stateHolder.getCurrentPuller(); }
9
10    @Override
11    public void setPuller(String name) { this.stateHolder.
12
13    @Override
14    public Integer getPrizePool() { [...] }
15
16    @Override
17    public void setPrizePool(Integer value) { [...] }
18
19    @Override
20    public int getDistribution(int hit) { return this.stateHolder.getDistribution(hit); }
21
22    @Override
23    public int[] getDistributions() { [...] }
24    @Override
25    public void setDistribution(int hit, int value) { [...] }
26    public void start() throws Exception { LOGGER.info("Start Lottery MBean"); }
27    public void stop() throws Exception { LOGGER.info("Stop Lottery MBean"); }
28 }
```

The start() and the stop() methods will be called during the lifecycle of the JMX MBean. Their use is optional.

```
1 package hu.qwaevisz.lottery.ejbservice.management;
2 [...]
3 public class LotteryMonitor extends StandardMBean implements LotteryMonitorMBean {
4
5     public LotteryMonitor() {
6         super(LotteryMonitorMBean.class, false);
7     }
8
9     @Override
10    public ObjectName preRegister(MBeanServer server, ObjectName name) throws Exception {
11        return name;
12    }
13
14    @Override
15    public void postRegister(Boolean registrationDone) {}
16
17    @Override
18    public void preDeregister() throws Exception {}
19
20    @Override
21    public void postDeregister() {}
22
23 }
```

The ctor of the ancestor (StandardMBean) expects the interface of the MBean.

Usage of the MBeanRegistration interface allows the interference in their life cycle.

LotteryMonitor.java

```
1 package hu.qvaevisz.lottery.ejbservice.management;
2 [..]
3 public class LotteryMonitor extends StandardMBean implements LotteryMonitorMBean {
4
5     private static final String LOTTERY_STATE_HOLDER_JNDI =
6         "java:global.lottery-1.0.lot-ejbservice.lotteryState";
7
8     private LotteryStateHolder getStateHolder() {
9         LotteryStateHolder holder = null;
10        try {
11            InitialContext context = new InitialContext();
12            holder = LotteryStateHolder.class.cast( context.lookup(LOTTERY_STATE_HOLDER_JNDI)
13                );
14        } catch (NamingException e) {
15            LOGGER.log(Level.SEVERE, e.getMessage(), e);
16        }
17        return holder;
18    }
19
20    @Override
21    public String getPuller() {
22        LotteryStateHolder holder = this.getStateHolder()
23        return holder != null ? holder.getCurrentPuller() : "";
24    }
25 }
```

In case of WebLogic the MBean will not be part of the *EJB context* (in case of JBoss it will be), so we cannot use the *@EJB* annotation. In all such cases we have to obtain the EJB from the JNDI tree.

LotteryMonitor.java

WebLogic Administration Console

- ▷ Environment | Server | Configuration
 - General tab → **VIEW JNDI Tree** link
 - *Overview tab* Binding Name property value

JNDI name of the EJBs (the JNDI tree structure is vendor specific, but the standardization is under construction):

```
java:global.lottery-1.0.lot-ejb-service.lotteryState
```

- ▷ `java:global`
- ▷ `lottery-1.0` → name of the deployed EAR (because of the 'dot' the browser splits it))
- ▷ `lot-ejb-service` → name of the EJB module inside the EAR
- ▷ `lotteryState` → the name attribute of the EJB (def: name of the implementation class)

src | main | **resources** | META-INF | jboss-service.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <server xmlns="urn:jboss:service:7.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="urn:jboss:service:7.0 jboss-service_7_0.xsd">
5     <mbean code="hu.qwaevisz.lottery.ejbsservice.management.LotteryMonitor"
6         name="lottery.mbean:service=LotteryMonitorMBean"></mbean>
</server>
```

jboss-service.xml

lottery.mbean (name attribute) will be the name in the topology,
LotteryMonitorMBean will be the name of the MBean (name attribute).
We have to set the JMX MBean class in the **code** attribute.
This is a *JBoss specific* file, the name has to be `jboss-service.xml`.

In case of **Gradle**'s ear plugin if we put something in the `src/main/application` directory, it will become part of the *archive* (EAR).

src | main | **application** | META-INF | weblogic-application.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <weblogic-application xmlns="http://xmlns.oracle.com/weblogic/weblogic-application">
3   <listener>
4     <listener-class>hu.qwaevisz.lottery.ejb.service.management.ApplicationMBeanLifecycleList
5   </listener>
6 </weblogic-application>
```

weblogic-application.xml

In case of WebLogic the MBean registration is source code driven (programmed), and we need to refer the configuration class in the container specific EAR deployment descriptor (`weblogic-application.xml`).

```
1 package hu.qwaevisz.lottery.ejbservice.management; [..]
2 public class ApplicationMBeanLifecycleListener extends ApplicationLifecycleListener {
3     private static final String MBEAN_SERVER_JNDI = "java:comp/jmx/runtime";
4     private static final String OBJECT_PACKAGE =
5         "hu.qwaevisz.lottery.ejbservice.management";
6     @Override
7     public void postStart(ApplicationLifecycleEvent evt) throws ApplicationException {
8         try {
9             InitialContext context = new InitialContext();
10            MBeanServer mbeanServer = MBeanServer.class.cast(
11                context.lookup(MBEAN_SERVER_JNDI) );
12            LotteryMonitor mbean = new LotteryMonitor();
13            ObjectName oname = this.buildObjectName();
14            mbeanServer.registerMBean(mbean, oname);
15        }
16        catch (Exception e) {
17            LOGGER.log(Level.SEVERE, e.getMessage(), e);
18        }
19    }
20    private ObjectName buildObjectName() throws MalformedObjectNameException {
21        return new ObjectName(OBJECT_PACKAGE +
22            ":type="+LotteryMonitor.class.getSimpleName()+",name="+LotteryMonitor.class.getName());
23    }
24    @Override
25    public void preStop(ApplicationLifecycleEvent evt) throws ApplicationException {
26        [..]
27    }
28 }
```

Because of the ApplicationLifecycleListener class we need to put the [WL-HOME]/wlserver/server/lib/wls-api.jar into the classpath.


```
1 > [JRE_HOME]/bin/jconsole.[bat|sh]
```

BUT: In case of JBoss we have to add further classes into the classpath of the jconsole (e.g.: jboss-cli-client.jar), so use the following command:

```
1 > [JBOSS_HOME]/bin/jconsole.[bat|sh]
```

It refers the [JRE_HOME]'s jconsole.

The JBoss AS will appear among the *Local processes* (but with the same client we can connect to *Remote JVM* as well).

Note

On MAC OS (or any other system) if the JBoss does not find the [JRE_HOME], we can use the jconsole.sh to set the CLASSPATH variable for the current terminal, then we can run the jconsole as usual (in the same terminal).

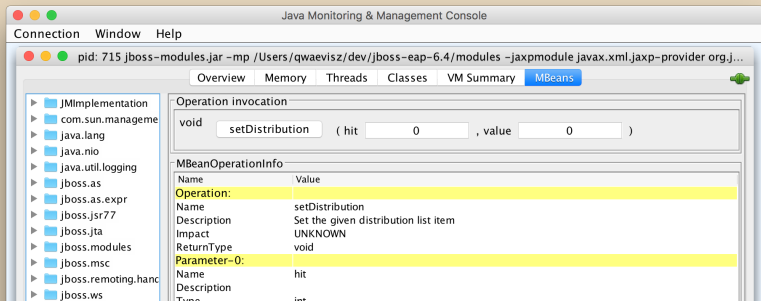
Similar to JBoss in case of WebLogic we need to put some special JAR into the classpath of jconsole.

```
1 > jconsole
   -J-Djava.class.path=%JAVA_HOME%\lib\jconsole.jar;%JAVA_HOME%\lib\
   -J-Djmx.remote.protocol.provider.pkgs=weblogic.management.remote
   -debug
```

The easy way to set the classpath if we execute the `setDomainEnv.[sh|cmd]` command then in the same terminal/command window we run the jconsole.

```
1 >
   [WL-HOME]/user_projects/domains/mydomain/bin/setDomainEnv.[sh|cmd]
2 > jconsole
```

MBean descriptions, argument names



The screenshot shows the Java Monitoring & Management Console interface. The left sidebar displays a tree view of MBeans under the path `com.sun.management.jmxremote`. The main panel is titled "MBeans" and shows details for the `setDistribution` operation. The "Operation invocation" section shows `void setDistribution (hit 0 , value 0)`. Below this, the "MBeanOperationInfo" section provides a table of details:

Name	Value
Operation:	
Name	setDistribution
Description	Set the given distribution list item
Impact	UNKNOWN
ReturnType	void
Parameter-0:	
Name	hit
Description	
Type	int

An MBean's attribute/operation description could be useful, and also could be very informing if we see the name of the operations' arguments in `jconsole`. Unfortunately, however these source code information will not be sent automatically. The ancestor class of the MBean (`javax.management.StandardMBean`) contains some methods which handle these meta data. The client collects these information when it calls these methods (multiple times). We can easily override these methods, however this solution is very inconvenient, it causes lots of `switch` case, and the maintenance of these descriptions would have been a nightmare.

Annotated MBean creation

lot-ejbservice project

```
1 package hu.qvaevisz.lottery.ejbservice.management;
2
3 @Description("Lottery Monitor")
4 public interface LotteryMonitorMBean {
5
6     @Description("Get current Puller")
7     String getPuller();
8     [...]
9     @Description("Set the given distribution list item")
10    void setDistribution(@PName("hit") int hit, @PName("value") int value);
11 }
```

LotteryMonitorMBean.java

Techniques

With the help of some custom annotations (@Description and @PName) and an intermediate ancestor class (AnnotatedStandardMBean extends StandardMBean) we are able to create a maintainable solution. The implementation uses the **Reflection API**.

Verify lottery ticket

POST `http://localhost:8080/lottery/api/service/verify`

HTTP Request (application/json):

```
1 [5, 15, 20, 42, 40]
```

HTTP Response (application/json):

```
1 370
```

```
1 @Path("/service")
2 public interface LotteryRestService {
3     @POST
4     @Path("/verify")
5     @Consumes("application/json")
6     @Produces("application/json")
7     int verifyTicket(int[] numbers) throws AdaptorException;
8 }
```

LotteryRestService.java

Calculation of the ticket's prize pool

Current winning numbers: 10, 20, 30, 40, 50

Prize pool: 12345

Prize pool distribution: { 1, 3, 6, 10, 80 }

Two-hits lottery ticket: $12345 * 0.03 = 370.35$

Deploy - I

Deploy application in the design environment (lot-ear module)



```
1 <project [..]>
2   [..]
3   <build>
4     <finalName>${project.parent.artifactId}-${project.version}</finalName>
5     <plugins>
6       [..]
7       <plugin>
8         <groupId>org.apache.maven.plugins</groupId>
9         <artifactId>maven-dependency-plugin</artifactId>
10        <version>${version.plugin.dependency}</version>
11        <executions>
12          <execution>
13            <id>copy</id>
14            <phase>package</phase>
15            <goals>
16              <goal>copy</goal>
17            </goals>
18          </execution>
19        </executions>
20        <configuration>
21          [CONFIGURATION]
22        </configuration>
23      </plugin>
24    </plugins>
25  </build>
26 </project>
```

The version number of the **maven-dependency-plugin** is defined in the *root's pom.xml* (3.0.2).

It will be linked to the **package** Maven phase.

pom.xml

Deploy - II

Deploy application in the design environment (lot-ear module)



Set the [JBOSS_HOME] environment variable depending on the used OS:
NIX: export JBOSS_HOME=~ /dev/jboss-eap-6.4 (.bash_profile)
WIN: JBOSS_HOME=c:\apps\jboss-eap-6.4 (environment variable)

```
1 <configuration>
2   <artifactItems>
3     <artifactItem>
4       <groupId>${project.groupId}</groupId>
5       <artifactId>${project.artifactId}</artifactId>
6       <version>${project.version}</version>
7       <type>${project.packaging}</type>
8       <overwrite>true</overwrite>
9       <destFileName>${project.parent.artifactId}-${project.version}.ear</destFileName>
10    </artifactItem>
11  </artifactItems>
12  <outputDirectory>${env.JBOSS_HOME}/standalone/deployments/</outputDirectory>
13 </configuration>
```

pom.xml

```
1 > mvn clean package
```