Lottery #gradle

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

Lottery

Maintain lottery results

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 managemenet** (**JMX**) interface through which an actor can fine-tune the application (prize pool, name of the puller, distribution).

Acquiring knowledge

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

Project structure
Subprojects, modules

- ▷ **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-ejbserviceclient' 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).

JBoss management console

admin user creation

- 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

```
> [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):

  -
    ```
    > [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

Database schema

Lottery project

```sql
CREATE TABLE event (
  event_id SERIAL NOT NULL,
  event_puller CHARACTER VARYING(100) NOT NULL,
  event_prizepool INTEGER NOT NULL,
  event_date TIMESTAMP WITHOUT TIME ZONE NOT NULL,
  CONSTRAINT PK_EVENT_ID PRIMARY KEY (event_id)
);

ALTER TABLE event OWNER TO postgres;

CREATE TABLE drawnnumber (
  drawnnumber_id SERIAL NOT NULL,
  drawnnumber_event_id INTEGER NOT NULL,
  drawnnumber_value INTEGER NOT NULL,
  CONSTRAINT PK_DRAWNNUMBER_ID PRIMARY KEY (drawnnumber_id),
  CONSTRAINT FK_DRAWNNUMBER_EVENT FOREIGN KEY (drawnnumber_event_id)
    REFERENCES event (event_id) MATCH SIMPLE ON UPDATE RESTRICT ON DELETE RESTRICT
);

ALTER TABLE drawnnumber 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;

    -
      ```
      SELECT e
      FROM Event e
        JOIN FETCH e.numbers
      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

```
SELECT
  event0_.event_id AS event_id1_1_0_,
  numbers1_.drawnnumber_id AS drawnnum1_0_1_,
  event0_.event_date AS event_da2_1_0_,
  event0_.event_prizepool AS event_pr3_1_0_,
  event0_.event_puller AS event_pu4_1_0_,
  numbers1_.drawnnumber_event_id AS drawnnum3_0_1_,
  numbers1_.drawnnumber_value AS drawnnum2_0_1_,
  numbers1_.drawnnumber_event_id AS drawnnum3_1_0__,
  numbers1_.drawnnumber_id AS drawnnum1_0_0__
FROM
  event event0_
    INNER JOIN drawnnumber numbers1_ ON
      event0_.event_id=numbers1_.drawnnumber_event_id
ORDER BY
  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 `drawnumber` table would be missed), but it will not *attach* the entities (if we walk though 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.

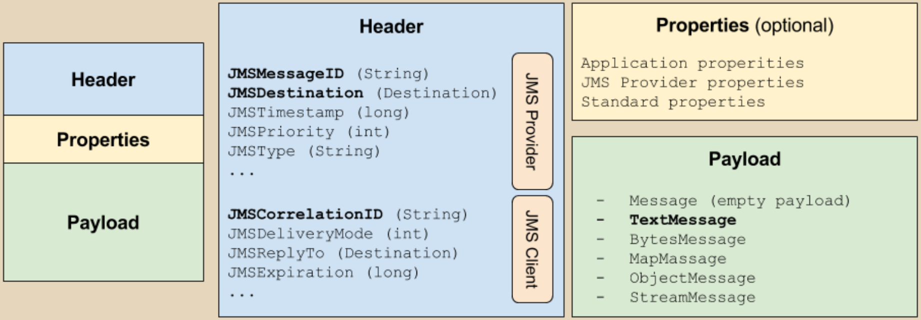
Java Message Service (JMS)
Message-Oriented Middleware (MOM)

- 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)

HornetQ
JBoss 6.x JMS Provider

- ▷ 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



JMSMessageID (String)
JMSDestination (Destination)
JMSTimestamp (long)
JMSPriority (int)
JMSType (String)
...

JMSCorrelationID (String)
JMSDeliveryMode (int)
JMSReplyTo (Destination)
JMSExpiration (long)
...

Application properities
JMS Provider properties
Standard properties

- Message (empty payload)
- **TextMessage**
- BytesMessage
- MapMassage
- ObjectMessage
- StreamMessage

JMS Queue creation

HornetQ

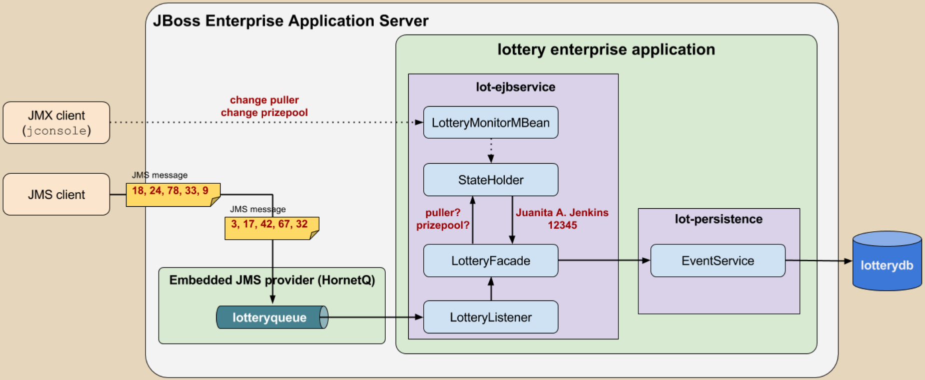
```
<?xml version="1.0" encoding="UTF-8"?>
<messaging-deployment xmlns="urn:jboss:messaging-deployment:1.0">
  <hornetq-server>
    <jms-destinations>
      <jms-queue name="lotteryqueue">
        <entry name="jms/queue/lotteryqueue" />
        <entry name="java:jboss/exported/jms/queue/lotteryqueue"
            />
      </jms-queue>
    </jms-destinations>
  </hornetq-server>
</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



LotteryListener

Message Driven Bean, (lot-ejbservice project)

```java
package hu.qwaevisz.lottery.ejbservice.listener;
[..]
@MessageDriven(name = "LotteryListener", activationConfig = { //
    @ActivationConfigProperty(propertyName = "destinationType", propertyValue =
         "javax.jms.Queue"),
    @ActivationConfigProperty(propertyName = "destination", propertyValue =
         "lotteryqueue"),
    @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue =
         "Auto-acknowledge") })
public class LotteryListener implements MessageListener {

  @EJB
  private LotteryFacade facade;

  @EJB
  private MessageConverter converter;

  @PostConstruct
  public void initialize() {
    LOGGER.info("Lottery Listener created...");
  }

  @Override
  public void onMessage(final Message message) {
    [..]
  }
}
```

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 rollbacked and the message will not removed from the queue by default!

LotteryListener.java

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

```java
@Override
public void onMessage(final Message message) {
  try {
    if (LOGGER.isDebugEnabled()) {
      final Queue destination = (Queue) message.getJMSDestination();
      final String queueName = destination.getQueueName();
      LOGGER.debug("New JMS message arrived into " + queueName + " queue (correlation
           id: " + message.getJMSCorrelationID() + ")");
    }
    if (message instanceof TextMessage) {
      final TextMessage textMessage = (TextMessage) message;
      String content = textMessage.getText();
      if (LOGGER.isDebugEnabled()) {
        LOGGER.debug("Received message: " + content);
      }
      this.facade.createNewEvent(this.converter.toNumbers(content));
    } else {
      LOGGER.error("Received message is not a TextMessage (" + message + ")");
    }
  } catch (final JMSException | AdaptorException | NumberFormatException e) {
    LOGGER.error(e, e);
  }
}
```

- BytesMessage
- MapMessage
- ObjectMessage
- StreamMessage
- TextMessage

LotteryListener.java

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

JMS Provider configuration - I

JMS Server and JMS Module

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

JMS Provider configuration - II

JMS Submodule and JMS Destination (queue)

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

JMS Provider configuration - III

Connection Factory

WebLogic Administration Console

▷ Services | Messaging | JMS Modules

- demoJMSmodule | Configuration | New
  - tpye: **Connection Factory**
  - name: **demoConnectionFactory**
  - JNDI name: **jms/demoConnectionFactory**
  - target: `myserver`

LotteryListener

Message Driven Bean (lot-ejbservice project)

```
package hu.qwaevisz.lottery.ejbservice.listener;
[..]
@MessageDriven(name = "LotteryListener", activationConfig = { //
    @ActivationConfigProperty(propertyName = "initialContextFactory", propertyValue =
         "weblogic.jndi.WLInitialContextFactory"),
    @ActivationConfigProperty(propertyName = "destinationType", propertyValue =
         "javax.jms.Queue"),
    @ActivationConfigProperty(propertyName = "destinationJndiName", propertyValue =
         "jms/queue/lotteryqueue"),
    @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue =
         "Auto-acknowledge") })
public class LotteryListener implements MessageListener, MessageDrivenBean {
  [..]
}
```

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/RemoteConnectionFact`
- 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

```
<subsystem xmlns="urn:jboss:domain:messaging:1.4">
    <hornetq-server>
        [..]
        <connectors>[..]</connectors>
        <acceptors>[..]</acceptors>
        <security-settings>
            <security-setting match="#">
                <permission type="send" roles="guest"/>
                <permission type="consume" roles="guest"/>
                [..]
            </security-setting>
        </security-settings>
        <address-settings>[..]</address-settings>
        <jms-connection-factories>
            <connection-factory name="InVmConnectionFactory">[..]</connection-factory>
            <connection-factory name="RemoteConnectionFactory">
                <connectors>[..]</connectors>
                <entries>
                    <entry name="java:jboss/exported/jms/RemoteConnectionFactory"/>
                </entries>
            </connection-factory>
            <pooled-connection-factory
                 name="hornetq-ra">[..]</pooled-
        </jms-connection-factories>
        <jms-destinations>[..]</jms-destinations
    </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

JMS Client application

RedHat® JBoss

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

SimpleClient.java

JMS Client application

Oracle® WebLogic

```java
final Properties environment = new Properties();
environment.put(Context.INITIAL_CONTEXT_FACTORY,
     "weblogic.jndi.WLInitialContextFactory");
environment.put(Context.PROVIDER_URL, "t3://localhost:7001");
environment.put(Context.SECURITY_PRINCIPAL, "weblogic");
environment.put(Context.SECURITY_CREDENTIALS, "AlmafA1#");
final Context context = new InitialContext(environment);
final ConnectionFactory connectionFactory = (ConnectionFactory)
     context.lookup("jms/demoConnectionFactory");
final Destination destination = (Destination) context.lookup("jms/queue/lotteryqueue");
final Connection connection = connectionFactory.createConnection("weblogic",
     "AlmafA1#");
final Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
final MessageProducer producer = session.createProducer(destination);
connection.start();
final TextMessage textMessage = session.createTextMessage("1, 2, 3, 4, 5");
producer.send(textMessage);
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*.s

Internal JMS Client

lot-webservice project

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

```java
[..]
InitialContext context = new InitialContext();
QueueConnectionFactory connectionFactory = (QueueConnectionFactory)
      context.lookup("ConnectionFactory");
QueueConnection connection = connectionFactory.createQueueConnection();
QueueSession session = connection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
Queue queue = (Queue) context.lookup("jms/queue/lotteryqueue");
QueueSender sender = session.createSender(queue);
TextMessage textMessage = session.createTextMessage();
connection.start();

textMessage.setText("1,2,3,4,5");
sender.send(textMessage);
sender.close();
session.close();
connection.close();
[..]
```

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 Session Bean

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)

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

```java
package hu.qwaevisz.lottery.ejbservice.holder;
[..]
@Singleton(mappedName = "ejb/lotteryState", name = "lotteryState")
@ConcurrencyManagement(ConcurrencyManagementType.CONTAINER)
public class LotteryStateHolderImpl implements LotteryStateHolder {
  private String puller;
  private Integer prizePool;
  private PrizeDistibution distribution;

  @PostConstruct
  public void initialize() {
    this.puller = "Juanita A. Jenkins";
    this.prizePool = 12345;
    this.distribution = new PrizeDistibution();
  }

  @Override
  @Lock(LockType.READ)
  public String getCurrentPuller() {
    return this.puller;
  }

  @Override
  @Lock(LockType.WRITE)
  public void setCurrentPuller(String name) {
    LOGGER.info("Change Puller: " + name);
    this.puller = name;
  }
  [..]
}
```

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.

tteryStateHolderImpl.jav

LotteryFacade

lot-ejbservice project

```java
package hu.qwaevisz.lottery.ejbservice.facade;
[..]
@Stateless(mappedName = "ejb/lotteryFacade")
public class LotteryFacadeImpl implements LotteryFacade {

  private static final Logger LOGGER = Logger.getLogger(LotteryFacadeImpl.class);

  @EJB
  private EventService eventService;

  @EJB
  private LotteryStateHolder stateHolder;

  @Override
  public void createNewEvent(int[] numbers) throws AdaptorException {
    try {
      String puller = this.stateHolder.getCurrentPuller();
      Integer prizePool = this.stateHolder.getCurrentPrizePool();
      this.eventService.create(puller, prizePool, numbers);
    } catch (final PersistenceServiceException e) {
      LOGGER.error(e, e);
      throw new AdaptorException(e.getLocalizedMessage());
    }
  }
  [..]
}
```

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.

```java
package hu.qwaevisz.lottery.persistence.service;
[..]
@Stateless(mappedName = "ejb/eventService")
@TransactionManagement(TransactionManagementType.CONTAINER)
@TransactionAttribute(TransactionAttributeType.REQUIRES_NEW)
public class EventServiceImpl implements EventService {

  @PersistenceContext(unitName = "lot-persistence-unit")
  private EntityManager entityManager;

  @Override
  public void create(String puller, Integer prizePool, int[] numbers) throws
       PersistenceServiceException {
    try {
      final Event event = new Event(puller, prizePool);
      for (final int number : numbers) {
        event.addNumber(number);
      }
      this.entityManager.persist(event);
    } catch (final Exception e) {
      throw new PersistenceServiceException("Unknown error when persisting Events! " +
           e.getLocalizedMessage(), e);
    }
  }
  [..]
}
```

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)

```java
public void addNumber(Integer number) {
  this.numbers.add(new DrawnNumber(number, this));
}
```

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`).

LotteryMonitor

lot-ejbservice project

```java
package hu.qwaevisz.lottery.ejbservice.management;
[..]
public class LotteryMonitor implements LotteryMonitorMBean {
  @EJB
  private LotteryStateHolder stateHolder;

  @Override
  public String getPuller() { return this.stateHolder.getCurrentPuller(); }

  @Override
  public void setPuller(String name) { this.stateHolder.

  @Override
  public Integer getPrizePool() { [..] }

  @Override
  public void setPrizePool(Integer value) { [..] }

  @Override
  public int getDistribution(int hit) { return this.stateHolder.getDistribution(hit); }

  @Override
  public int[] getDistributions() { [..]}
  @Override
  public void setDistribution(int hit, int value) { [..] }
  public void start() throws Exception { LOGGER.info("Start Lottery MBean"); }
  public void stop() throws Exception { LOGGER.info("Stop Lottery MBean"); }
}
```

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

LotteryMonitor.java

LotteryMonitor - I

lot-ejbservice project

```java
package hu.qwaevisz.lottery.ejbservice.management;
[..]
public class LotteryMonitor extends StandardMBean implements LotteryMonitorMBean {

  public LotteryMonitor() {
    super(LotteryMonitorMBean.class, false);
  }

  @Override
  public ObjectName preRegister(MBeanServer server, ObjectName name) throws Exception {
    return name;
  }

  @Override
  public void postRegister(Boolean registrationDone) {}

  @Override
  public void preDeregister() throws Exception {}

  @Override
  public void postDeregister() {}

}
```

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

LotteryMonitor - II

lot-ejbservice project

```java
package hu.qwaevisz.lottery.ejbservice.management;
[..]
public class LotteryMonitor extends StandardMBean implements LotteryMonitorMBean {

  private static final String LOTTERY_STATE_HOLDER_JNDI =
        "java:global.lottery-1.0.lot-ejbservice.lotteryState";

  private LotteryStateHolder getStateHolder() {
    LotteryStateHolder holder = null;
    try {
      InitialContext context = new InitialContext();
      holder = LotteryStateHolder.class.cast( context.lookup(LOTTERY_STATE_HOLDER_JNDI)
           );
    } catch (NamingException e) {
      LOGGER.log(Level.SEVERE, e.getMessage(), e);
    }
    return holder;
  }

  @Override
  public String getPuller() {
    LotteryStateHolder holder = this.getStateHolder()
    return holder != null ? holder.getCurrentPuller() : "";
  }
}
```

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

JNDI tree

Browse the JNDI names

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 stucture is vendor specific, but the standardization is under construction):

`java:global.lottery-1.0.lot-ejbservice.lotteryState`

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

MBean registration

lot-ejbservice project

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

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

MBean registration

root project

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

```
<?xml version="1.0" encoding="UTF-8"?>
<weblogic-application xmlns="http://xmlns.oracle.com/weblogic/weblogic-application">
  <listener>
    <listener-class>hu.qwaevisz.lottery.ejbservice.management.ApplicationMBeanLifeCycleList
  </listener>
</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).

MBean registration

lot-ejbservice project

```java
package hu.qwaevisz.lottery.ejbservice.management; [..]
public class ApplicationMBeanLifeCycleListener extends ApplicationLifecycleListener {
    private static final String MBEAN_SERVER_JNDI = "java:comp/jmx/runtime";
    private static final String OBJECT_PACKAGE =
         "hu.qwaevisz.lottery.ejbservice.management";
    @Override
    public void postStart(ApplicationLifecycleEvent evt) throws ApplicationException {
        try {
            InitialContext context = new InitialContext();
            MBeanServer mbeanServer = MBeanServer.class.cast(
                 context.lookup(MBEAN_SERVER_JNDI) );
            LotteryMonitor mbean = new LotteryMonitor();
            ObjectName oname = this.buildObjectName();
            mbeanServer.registerMBean(mbean, oname);
        }
        catch (Exception e) {
            LOGGER.log(Level.SEVERE, e.getMessage(), e);
        }
    }
    private ObjectName buildObjectName() throws MalformedObjectNameException {
        return new ObjectName(OBJECT_PACKAGE +
             ":type="+LotteryMonitor.class.getSimpleName()+",name="+LotteryMonitor.class.ge
    }
    @Override
    public void preStop(ApplicationLifecycleEvent evt) throws ApplicationException {
        [..]
    }
}
```

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

ApplicationMBeanLifeCycleListener.java

jconsole

JMX client application

```
> [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:

```
> [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).

jconsole

JMX client application

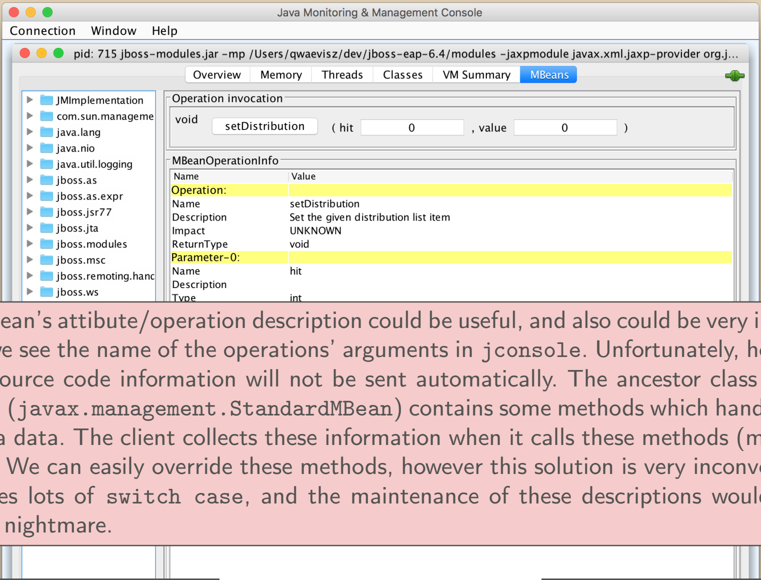
Similar to JBoss in case of WebLogic we need to put some special JAR into the classpath of `jconsole`.

```
> 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`.

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

MBean descriptions, argument names



Java Monitoring & Management Console
Connection Window Help
pid: 715 jboss-modules.jar -mp /Users/qwaevisz/dev/jboss-eap-6.4/modules -jaxpmodule javax.xml.jaxp-provider org.j...
Overview Memory Threads Classes VM Summary MBeans
JMImplementation
com.sun.manageme
java.lang
java.nio
java.util.logging
jboss.as
jboss.as.expr
jboss.jsr77
jboss.jta
jboss.modules
jboss.msc
jboss.remoting.hanc
jboss.ws
Operation invocation
void setDistribution (hit 0 , value 0)
MBeanOperationInfo

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 attibute/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

```java
package hu.qwaevisz.lottery.ejbservice.management;

@Description("Lottery Monitor")
public interface LotteryMonitorMBean {

  @Description("Get current Puller")
  String getPuller();
  [..]
  @Description("Set the given distribution list item")
  void setDistribution(@PName("hit") int hit, @PName("value") int value);
}
```

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):

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

HTTP Response (application/json):

```
370
```

```java
@Path("/service")
public interface LotteryRestService {
  @POST
  @Path("/verify")
  @Consumes("application/json")
  @Produces("application/json")
  int verifyTicket(int[] numbers) throws AdaptorException;
}
```

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

Deploy application in the design environment (root project)

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

```
[..]
ext {
  [..]
  deployLocation = System.getenv('JBOSS_HOME') + '/standalone/deployments/'
}
[..]
task deployClean ( type: Delete ) {
  delete deployLocation + "${project.name}-${version}.ear"
  sleep(2000)
}

task deployEar ( type: Copy ) {
  dependsOn 'deployClean'
  from "build/libs/${project.name}-${version}.ear"
  into deployLocation
}
```

build-root.gradle

```
> gradle clean build deployEar
```