EJB 3.1 - Killing The Top Eleven Myths, Tales and Biases

About me

- Expert Group Member (jcp.org) of Java EE 6, EJB 3.1, Time and Date and JPA 2.0
- Java Champion,Netbeans Dream Team Member, (JavaONE) speaker, freelancer, consultant and author: 7 German books + working on “Productive Java EE – Rethinking The Patterns”
- Trainer, Developer and Architect (since JDK 1.0)
- Project owner/committer: http://greenfire.dev.java.net, http://p4j5.dev.java.net, http://fishfarm.dev.java.net/

1. EJBs Are Heavyweight

EJBs Are Heavyweight

Configuration is mainly gone, because of conventions... (there was no XML in my last projects except persistence.xml)

EJB 3 are just annotated Java classes (if you love XML you can even use just Deployment Descriptors instead of annotation)

Container “services” like transactions, security, concurrency or state are implemented with aspects (often realized with dynamic proxies)

EJBs Are Heavyweight

"POJOs" are just JavaBeans maintained by another container, using similar techniques as EJB 3.1

EJB 3 containers are surprisingly small. Glassfish v3 EA comes with two jars (688kB + 8kB = 796kB). The EJB 3 container is an OSGI bundle...

The whole EJB 3.1 API is about 47 kB.

2. EJBs Are Hard To Test

EJBs Are Hard To Test

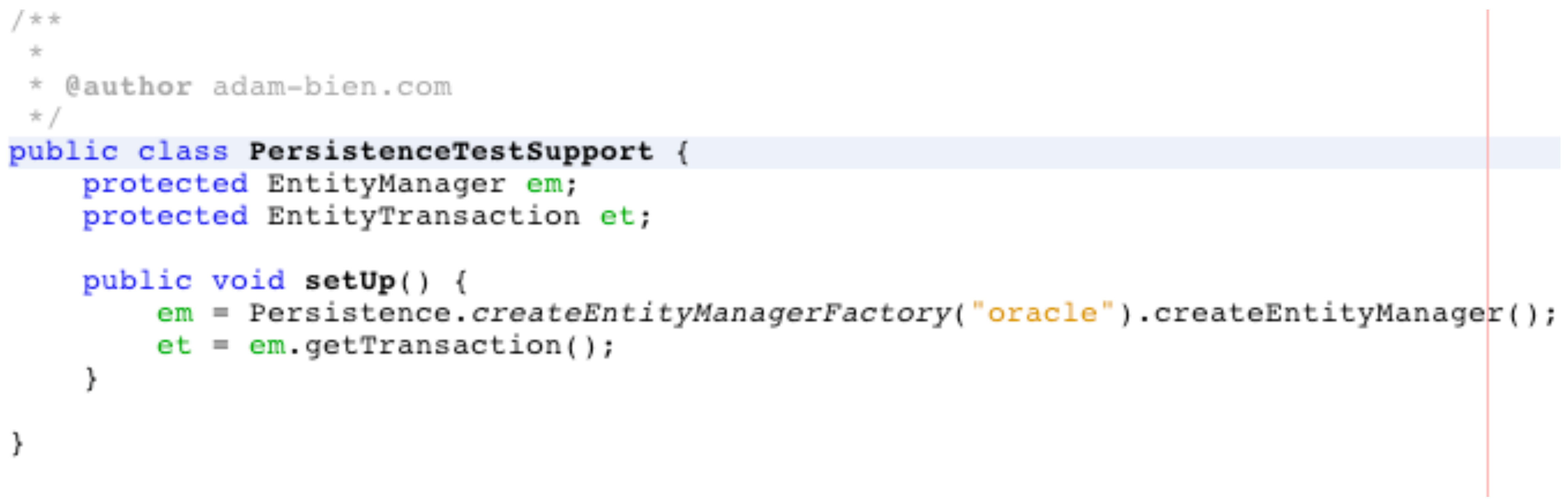
To test your EJBs outside the container, you will need to have the EJB 3.1 annotations available in your classpath.

In recent projects we deployed EJB 3.0 components directly to Netbeans RCP application. We could start and test the application without even having an application server involved.

If you can start EJBs outside the container, you can start them in a unit test as well...

EJBs Are Hard To Test

Create the EntityManager and fetch the EntityTransaction in the superclass...

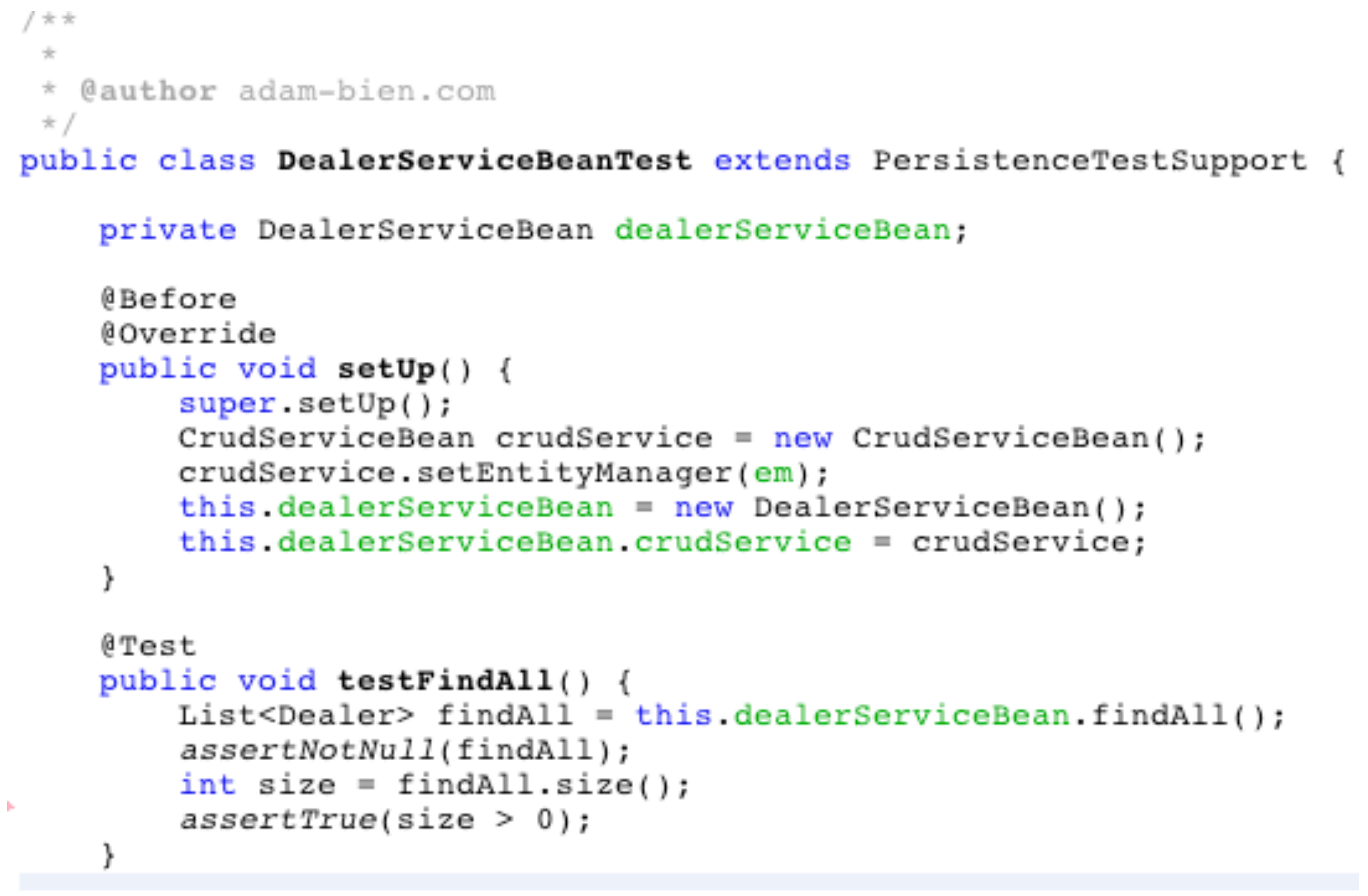


```
/**
 *
 * @author adam-bien.com
 */
public class PersistenceTestSupport {
    protected EntityManager em;
    protected EntityTransaction et;

    public void setUp() {
        em = Persistence.createEntityManagerFactory("oracle").createEntityManager();
        et = em.getTransaction();
    }

}
```

blog.adam-bien.com

EJBs Are Hard To Test

...inherit from it. Just inject / mock the dependencies by hand (EJB 3.0)



```
/**
 *
 * @author adam-bien.com
 */
public class DealerServiceBeanTest extends PersistenceTestSupport {

    private DealerServiceBean dealerServiceBean;

    @Before
    @Override
    public void setUp() {
        super.setUp();
        CrudServiceBean crudService = new CrudServiceBean();
        crudService.setEntityManager(em);
        this.dealerServiceBean = new DealerServiceBean();
        this.dealerServiceBean.crudService = crudService;
    }

    @Test
    public void testFindAll() {
        List<Dealer> findAll = this.dealerServiceBean.findAll();
        assertNotNull(findAll);
        int size = findAll.size();
        assertTrue(size > 0);
    }
```

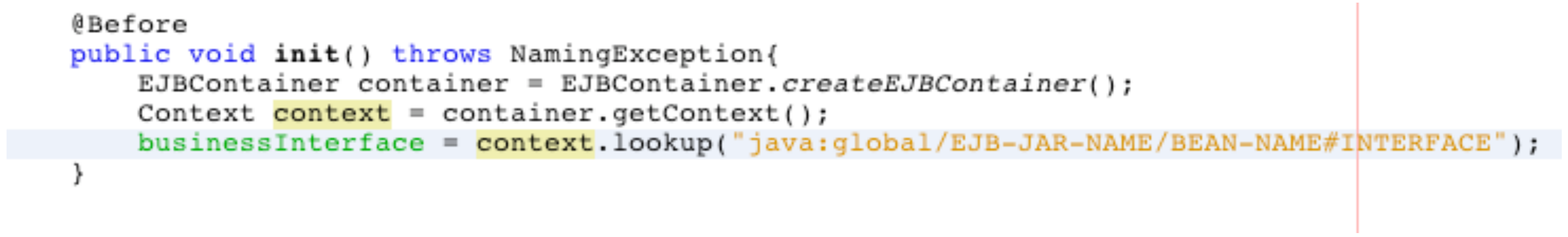
blog.adam-bien.com

EJBs Are Hard To Test

EJB 3.0 are just annotated POJOs - you can test them as such.

Transaction / Security can be harder to test - but integration tests are better for that purpose.

In EJB 3.1 the testing experience is even better with:



```
@Before
public void init() throws NamingException{
    EJBContainer container = EJBContainer.createEJBContainer();
    Context context = container.getContext();
    businessInterface = context.lookup("java:global/EJB-JAR-NAME/BEAN-NAME#INTERFACE");
}
```

3. EJBs Are Not Portable

EJBs Are Not Portable

J2EE 1.4 was underspecified :-) - EJB 3.X / JPA specs cover more real world stuff (locking, optimistic concurrency etc.).

Vendor specific deployment descriptor were painful for migration - they are basically gone.

In most cases a EJB-JAR module is nothing but a JAR without any XML descriptors (neither ejb-jar.xml nor vendor specific)

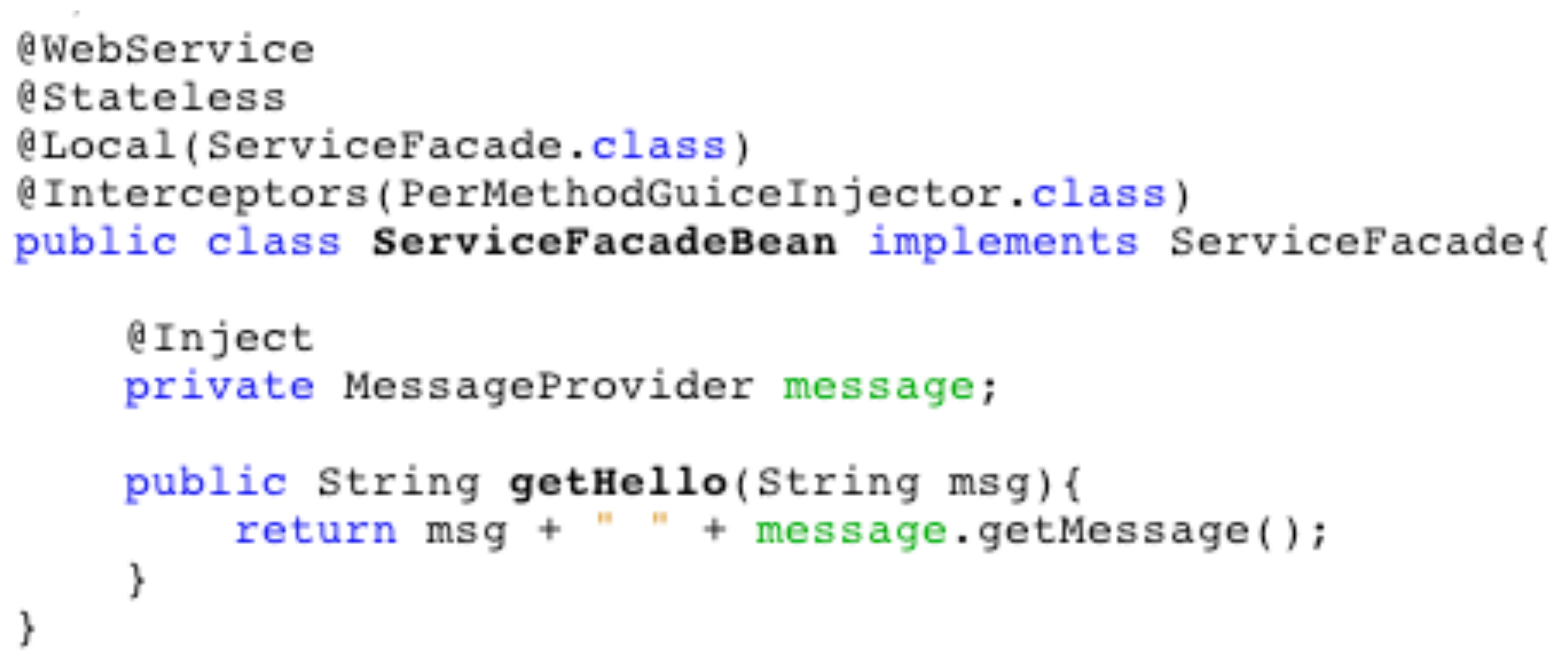
Vendor specific annotations are not needed to develop a Java EE application.

There is **NOTHING** vendor specific in an EAR. The portability is really good.

4. EJBs Are Not Extensible

EJBs Are Not Extensible

How to inject a Guice component into an EJB 3:



```java
@WebService
@Stateless
@Local(ServiceFacade.class)
@Interceptors(PerMethodGuiceInjector.class)
public class ServiceFacadeBean implements ServiceFacade{

    @Inject
    private MessageProvider message;

    public String getHello(String msg){
        return msg + " " + message.getMessage();
    }
}
```

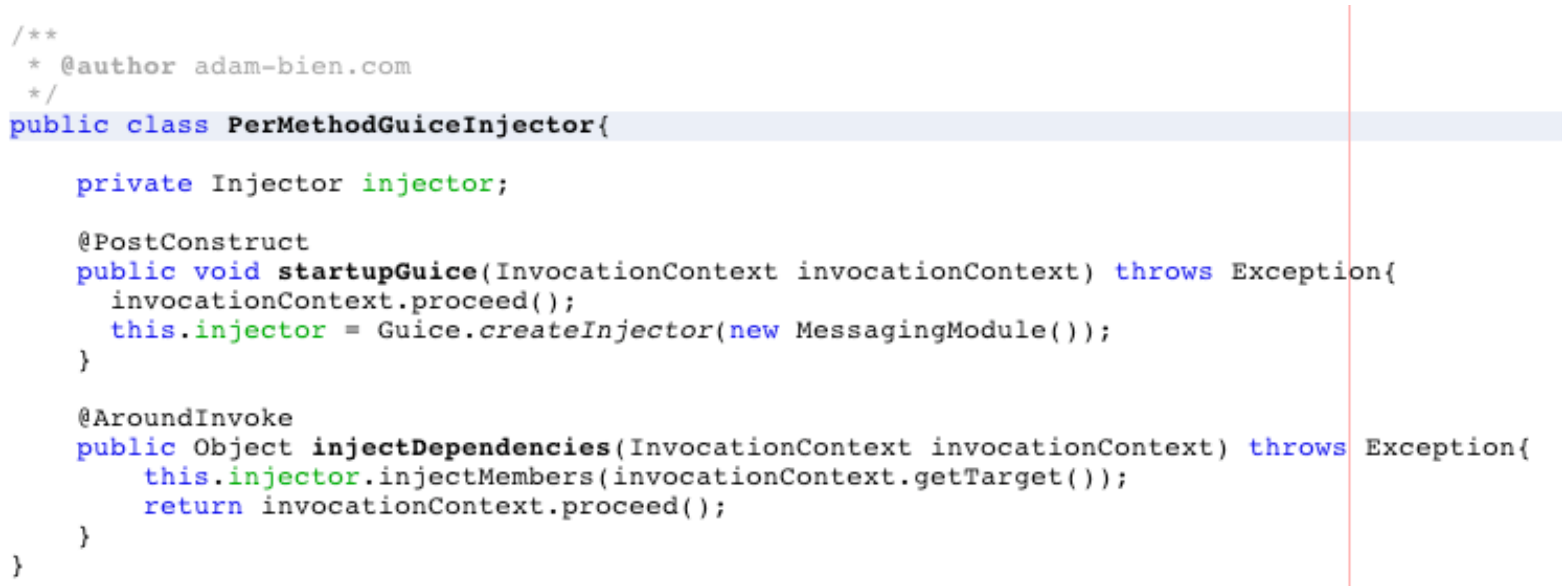
EJBs Are Not Extensible

The Guice (fluent) configuration:

```
import com.google.inject.Binder;
import com.google.inject.Module;

public class MessagingModule implements Module{

    public void configure(Binder binder) {
        binder.bind(MessageProvider.class).to(MessageProviderImpl.class);
    }
}
```

EJBs Are Not Extensible

You only need an interceptor:



```
/**
 * @author adam-bien.com
 */
public class PerMethodGuiceInjector{

    private Injector injector;

    @PostConstruct
    public void startupGuice(InvocationContext invocationContext) throws Exception{
      invocationContext.proceed();
      this.injector = Guice.createInjector(new MessagingModule());
    }

    @AroundInvoke
    public Object injectDependencies(InvocationContext invocationContext) throws Exception{
        this.injector.injectMembers(invocationContext.getTarget());
        return invocationContext.proceed();
    }
}
```

EJBs Are Not Extensible

Interceptors are able to access the Bean instance directly.

Having an instance available - you can manipulate it; inject members use reflection to invoke methods, or set fields...

It is very interesting for the integration of existing “legacy” IoC frameworks :-)

5. EJBs Are Slow

EJBs Are Slow

The throughput of the EJB 3 solution was 2391 transactions/second. The slowest method call took 7 milliseconds. The average wasn't measurable. Please keep in mind that in every request two session beans were involved - so the overhead is doubled.

POJO: The throughput of the POJO solution was 2562 requests/second (request - there are no transactions here). The slowest method call took 10 ms.

The difference is 171 requests / seconds, or 6.6%

6. EJBs Are Not Scaleable

EJBs Are Not Scalable

EJB (1.0, 1.1, 2.0, 2.1, 3.0 and 3.1) are always executed in a dedicated thread - they appear as single threaded for the developer.

For every thread (=user request, transaction) a new EJB instance is created (or reused from pool) - there is actually no shared state, except you access singletons, files etc. - which is not allowed.

Actually the spec even prohibits the usage of the synchronization primitives and threading functionality inside EJBs.

EJBs Are Not Scalable

Furthermore the programming model is rather procedural (or if you will "service oriented") - so in the @Stateless Session Beans the methods process some input data and give it back to the caller.

Even the @Stateful Session Beans do not brake the rule - the container still prohibits concurrent access to the @Stateful instance.

Because a @Stateful Session Bean can be only executed in a single thread - you do not have (you shouldn't!) care about the threading either.

EJBs Are Not Scalable

If you acess the JPA-persistence - the container synchronizes the data for you. Every transaction receives a copy of the JPA-entity (it is often implemented in this way), so nothing bad can happen even in this case.

Every thread is (hopefully) executed in a single core. So if you keep the transactions short, and do not "hack" the EJBs accessing static variables or singletons - you are on the bright side :-).

EJBs Are Not Scalable

Every bean is executed in a (pooled) thread.

Bean instances are bound to the thread for the duration of the execution.

Load balancing works perfectly - you can start as many servers as you like to. Even without having a cluster...

For SFSB “session affinity” can improve the scaleability significantly.

Programming restrictions do encourage “functional programming” (no shared state, blocking, threads)

EJBs Are Not Scalable (some load test results)

Environment:

Glassfish v2ur2 on Linux, Clustered, a 4 way, Intel based machine and JDK 1.5 (JDK 1.6 would be better, but was not available in this case). Two load-generators ran remotely (via IIOP, standard EJB 3 client), each with 50 virtual users. Settings: -Xmx to 512m: no further tuning was performed.

Configuration:

CRUD application as load generator, without think-time, and almost empty database, just to stress the appserver as much as possible.

Result:

250 - 500 Transactions / second (we weren't alone on the machine :-().

7. EJBs Are Too Complex

EJBs Are Too Complex

Java EE is distributed and concurrent platform per definition.

It mainly abstracts already existing products (messaging, EIS, relational databases)

Distributed programming with shared state is always a challenge.

In the Cloud Computing / SOA era non-functional requirements like: monitoring (JMX), management, fail-over or elasticity become more and more important.

Think about the ratio between the essential and accidential complexity...

8. EJBs Are Hard To Integrate With Web Frameworks (and POJOs)

Wicket

DI in Wicket

Configuration in Wicket application required:

```
import
org.wicketstuff.javaee.injection.JavaEEComponentInjector;

import org.apache.wicket.protocol.http.WebApplication;

public class WicketApplication extends WebApplication{

  @Override

  protected void init() {

   addComponentInstantiationListener(

new JavaEEComponentInjector(this));

    }

}
```

DI in Wicket

Extension of web.xml:

```
<ejb-local-ref>
   <ejb-ref-name>profileService</ejb-ref-name>
   <ejb-ref-type>Session</ejb-ref-type>
    <local-home/>
   <local>....facade.ProfileService</local>
</ejb-local-ref>
```

DI in Wicket

…then EJB 3.X can be just injected into pages:

```
import org.apache.wicket.markup.html.WebPage;

public class ProfileView extends WebPage {

    @EJB(name = "profileService")

    private ProfileService profileService;


    @EJB(name = "dealerService")

    private DealerService dealerService;
```

JSF

JSF

EJB 3 injection is even easier:

1. No additional declaration in web.xml
2. No programmatic configuration needed.
3. Just inject SLSB into request-scoped and SFSB into session-scoped backing beans

```
public class StatefulBackingBean{
    @EJB
    private OrderGateway orderGateway;
}
```

Servlets

Servlets

Same story as in JSF:

1. No additional declaration in web.xml
2. No programmatic configuration needed.
3. Just inject SLSB into the servlet and lookup SFSB and put them into the HTTPSession

```
public class LoadServlet extends HttpServlet{

    @EJB

    private PersonService service;

}
```

Servlets

Servlets can be used to implement simple load test scenarios for the EJB layer:

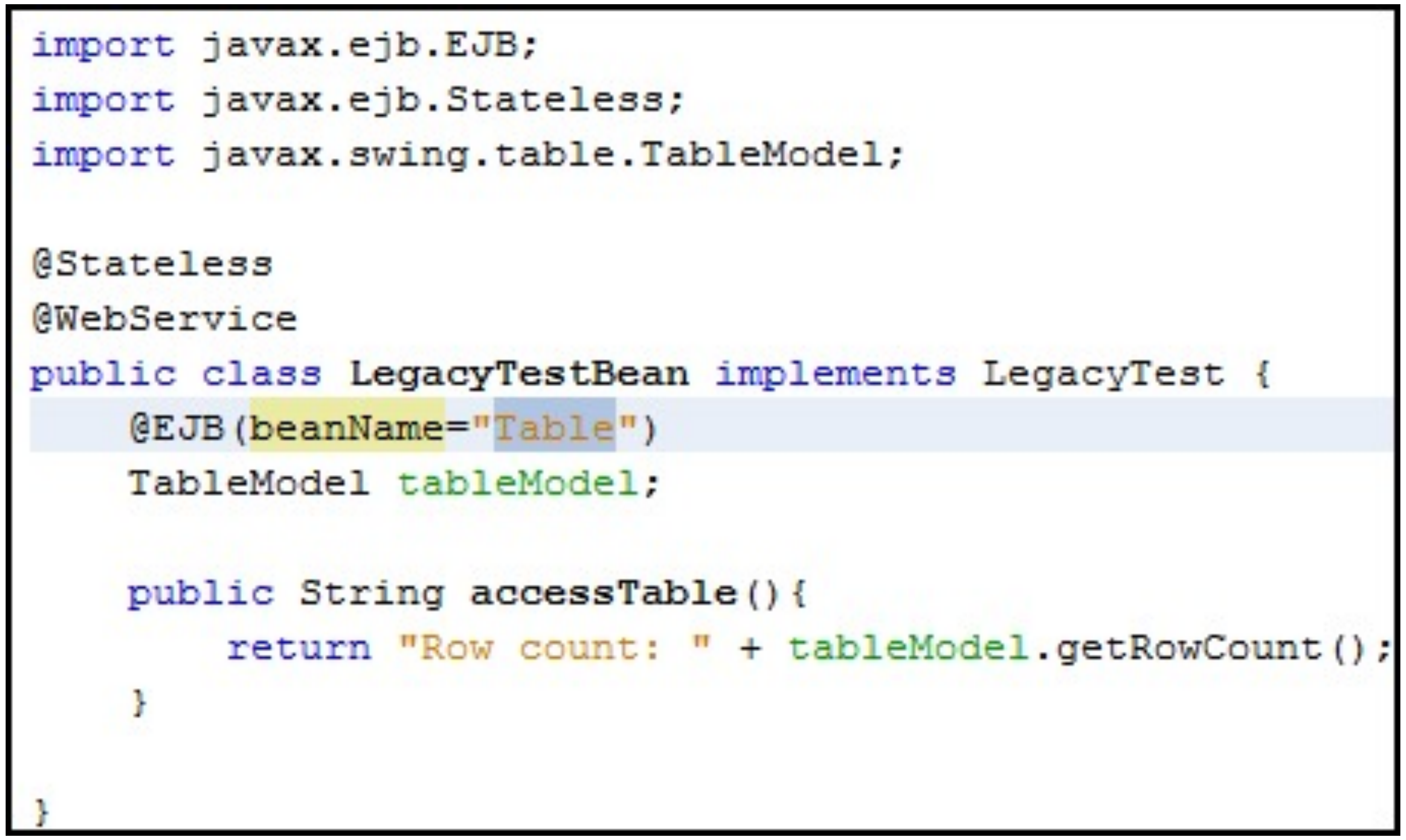
- Inject your bean into the servlet
- Implement the test-flow in doGet method
- Use e.g. http://jakarta.apache.org/jmeter to generate the load
- Use JConsole / VisualVM to monitor server
- Watch the resource consumption and throughput.

...and POJOs (into EJB 3)

Legacy POJO injection

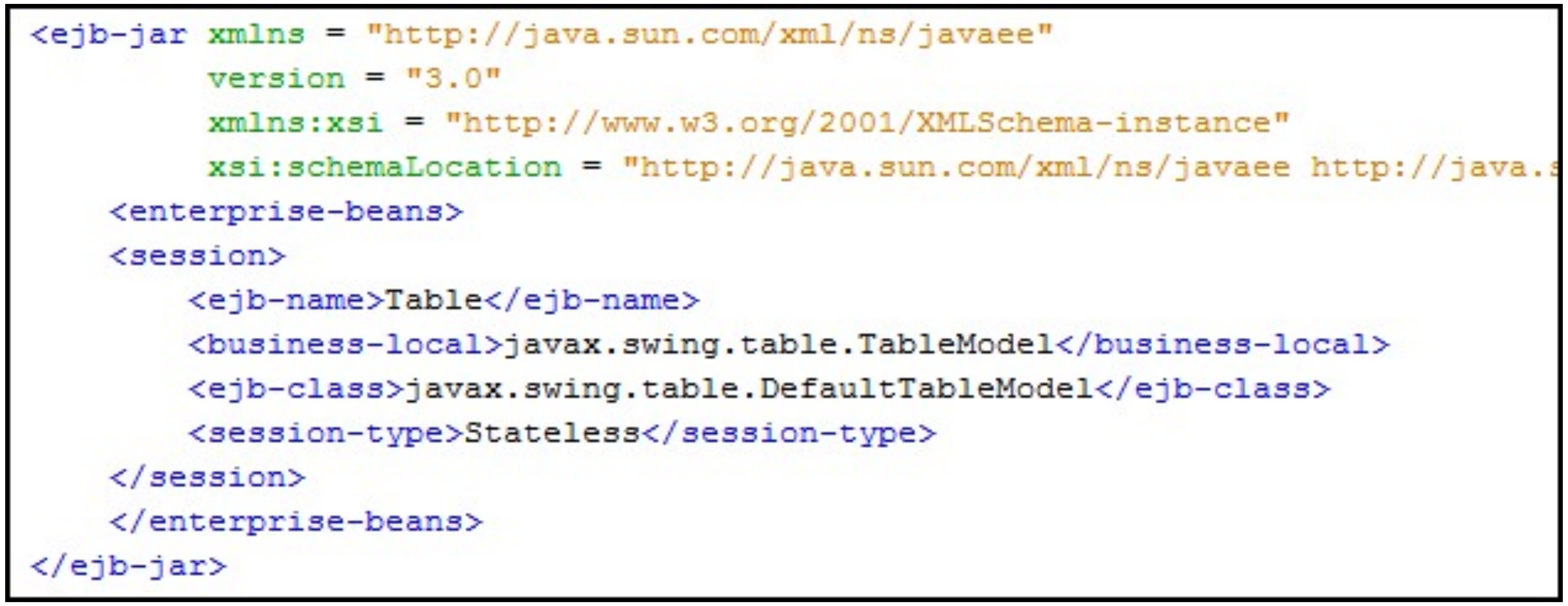
- Not everything is an EJB
- Sometimes it is convenient to inject „legacy POJOs“
- But we do not always have the source for the legacy POJO…

Legacy POJO injection



```
import javax.ejb.EJB;
import javax.ejb.Stateless;
import javax.swing.table.TableModel;

@Stateless
@WebService
public class LegacyTestBean implements LegacyTest {
    @EJB(beanName="Table")
    TableModel tableModel;

    public String accessTable(){
        return "Row count: " + tableModel.getRowCount();
    }

}
```

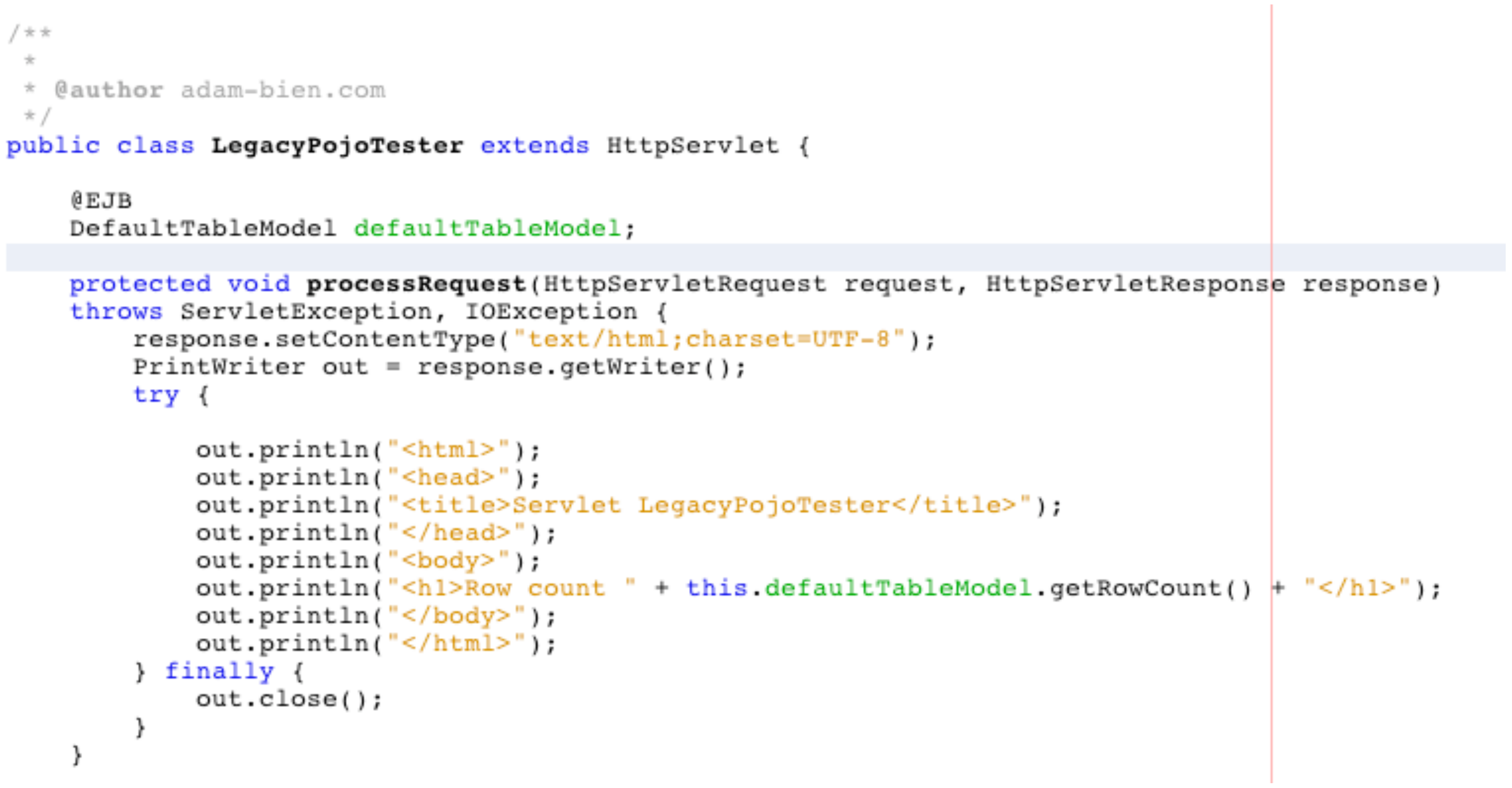
Legacy POJO injection


```
<ejb-jar xmlns = "http://java.sun.com/xml/ns/javaee"
         version = "3.0"
         xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation = "http://java.sun.com/xml/ns/javaee http://java.s
    <enterprise-beans>
    <session>
        <ejb-name>Table</ejb-name>
        <business-local>javax.swing.table.TableModel</business-local>
        <ejb-class>javax.swing.table.DefaultTableModel</ejb-class>
        <session-type>Stateless</session-type>
    </session>
    </enterprise-beans>
</ejb-jar>
```

...and POJOs (into EJB 3.1)

...and POJOs (into EJB 3.1)

Servlet in WAR:



```
/**
 *
 * @author adam-bien.com
 */
public class LegacyPojoTester extends HttpServlet {

    @EJB
    DefaultTableModel defaultTableModel;

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {

            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet LegacyPojoTester</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Row count " + this.defaultTableModel.getRowCount() + "</h1>");
            out.println("</body>");
            out.println("</html>");
        } finally {
            out.close();
        }
    }
```

...and POJOs (into EJB 3.1)

...and some XML:

```
<ejb-jar xmlns = "http://java.sun.com/xml/ns/javaee"
         version = "3.0"
         xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation = "http://java.sun.com/xml/ns/javaee http://java.su
    <enterprise-beans>
    <session>
       <ejb-name>Table</ejb-name>
        <ejb-class>javax.swing.table.DefaultTableModel</ejb-class>
        <session-type>Stateless</session-type>
    </session>
    </enterprise-beans>
</ejb-jar>
```

9. EJBs Are Hard To Configure

EJBs Are Hard To Configure

EJB 3.0 Session Beans can be configured using XML in different ways.

You can use dependency injection, as well as programmatic lookup for this purpose.

The dependency injection approach allows you the definition of default values in the bean class, which can be overwritten in the deployment descriptor (the XML-file).

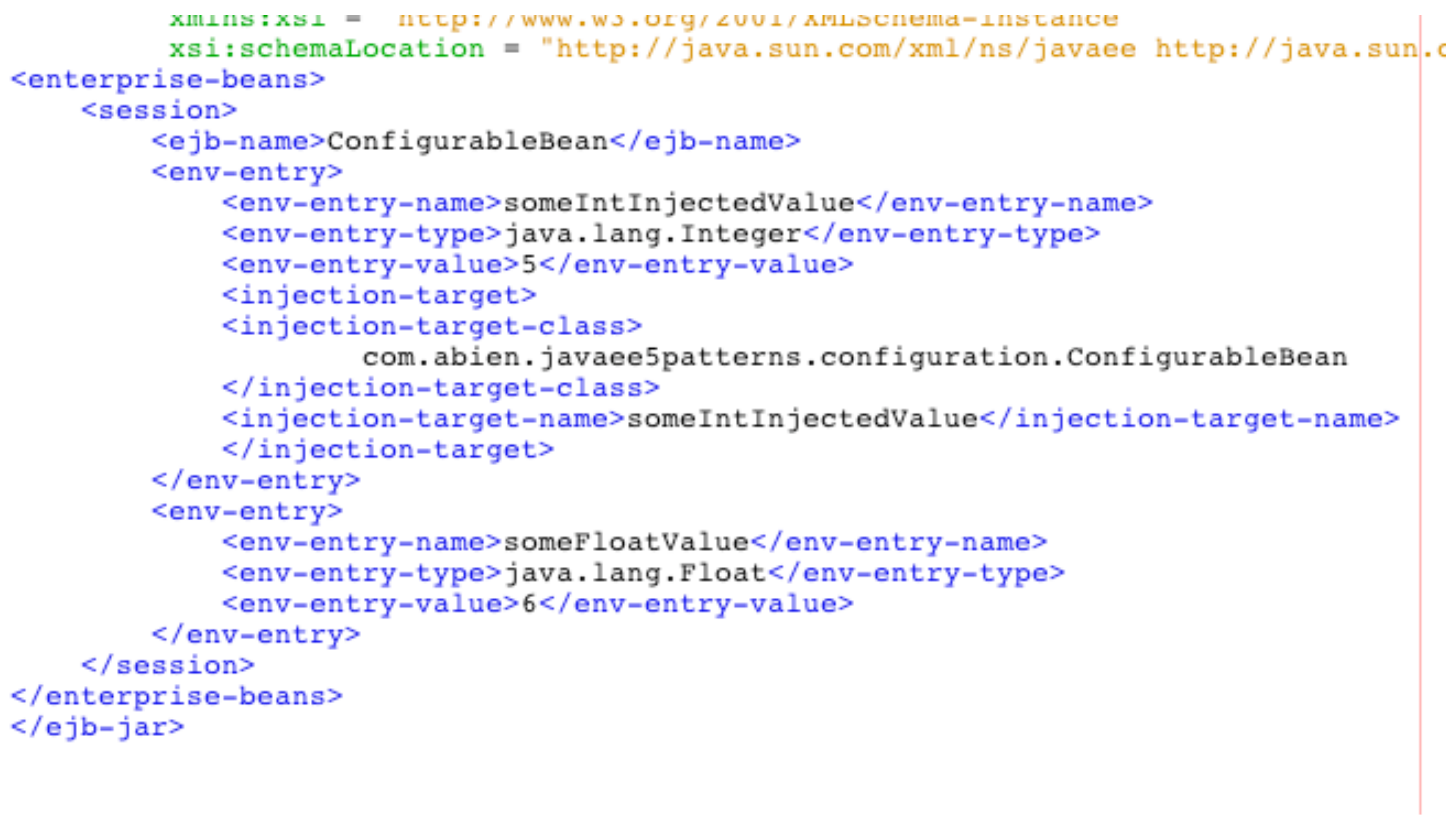
EJBs Are Hard To Configure

Injecting variables:

```java
/**
 *
 * @author adam-bien.com
 */
@Stateless
@WebService
public class ConfigurableBean implements Configurable {

    @Resource int someIntInjectedValue = 100;
    float someFloatValue = 2;

    @Resource SessionContext sessionContext;

    public int getSomeIntInjectedValue() {
        return this.someIntInjectedValue;
    }

    public float fetchSomeFloatValue(){
        return (Float)this.sessionContext.lookup("someFloatValue");
    }


}
```

EJBs Are Hard To Configure

...and the configuration:



```
         xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation = "http://java.sun.com/xml/ns/javaee http://java.sun.c
<enterprise-beans>
    <session>
        <ejb-name>ConfigurableBean</ejb-name>
        <env-entry>
            <env-entry-name>someIntInjectedValue</env-entry-name>
            <env-entry-type>java.lang.Integer</env-entry-type>
            <env-entry-value>5</env-entry-value>
            <injection-target>
            <injection-target-class>
                    com.abien.javaee5patterns.configuration.ConfigurableBean
            </injection-target-class>
            <injection-target-name>someIntInjectedValue</injection-target-name>
            </injection-target>
        </env-entry>
        <env-entry>
            <env-entry-name>someFloatValue</env-entry-name>
            <env-entry-type>java.lang.Float</env-entry-type>
            <env-entry-value>6</env-entry-value>
        </env-entry>
    </session>
</enterprise-beans>
</ejb-jar>
```

blog.adam-bien.com

EJBs Are Hard To Configure

The ejb-jar.xml isn’t beautiful - but portable.

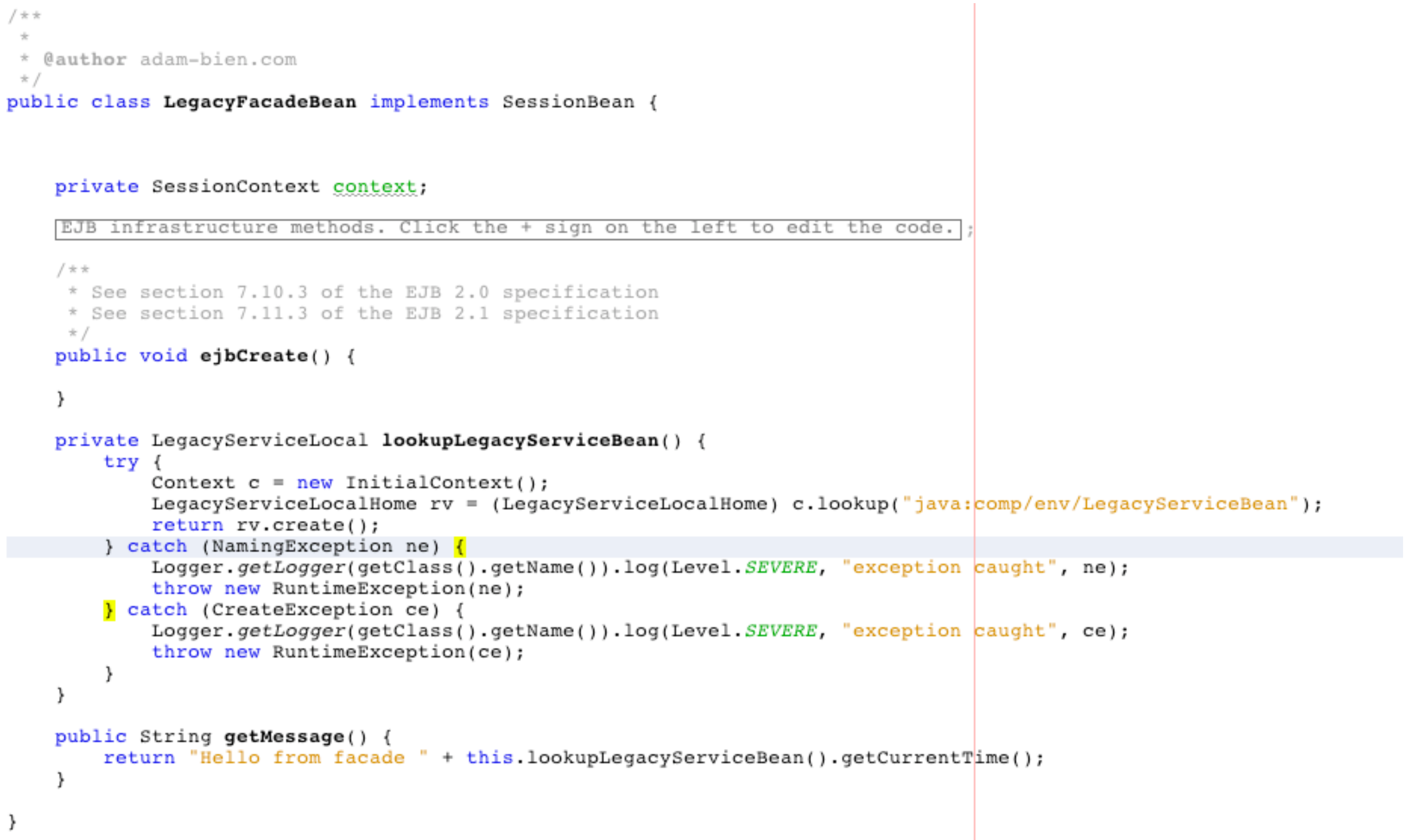
The verbose syntax will improve in EJB 3.1

Think about the Guice integration - there are no limits here...

10. EJBs Are Hard To Migrate

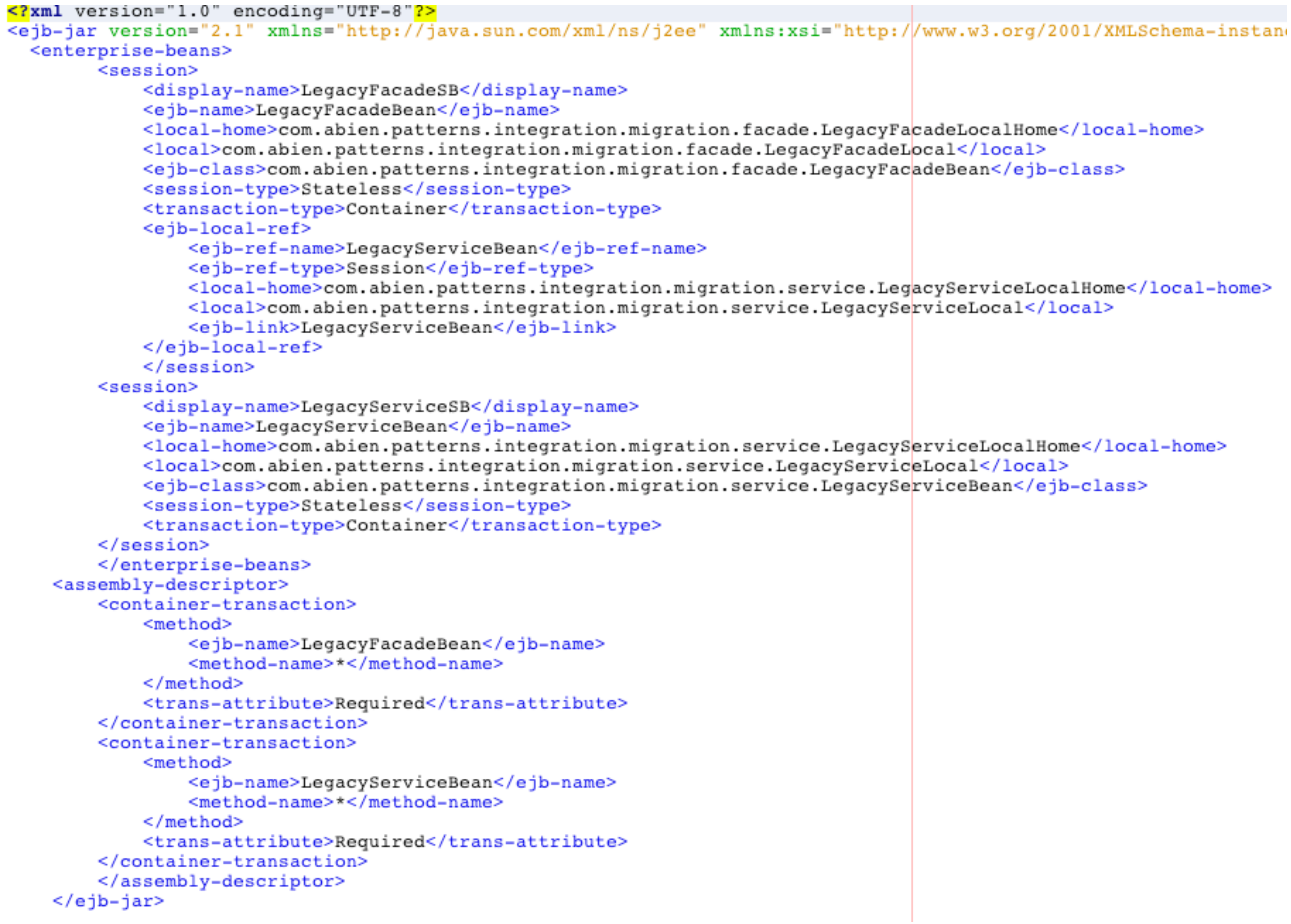
EJBs Are Hard To Migrate

A legacy EJB 2:



```
/**
 *
 * @author adam-bien.com
 */
public class LegacyFacadeBean implements SessionBean {


    private SessionContext context;

    EJB infrastructure methods. Click the + sign on the left to edit the code. ;

    /**
     * See section 7.10.3 of the EJB 2.0 specification
     * See section 7.11.3 of the EJB 2.1 specification
     */
    public void ejbCreate() {

    }

    private LegacyServiceLocal lookupLegacyServiceBean() {
        try {
            Context c = new InitialContext();
            LegacyServiceLocalHome rv = (LegacyServiceLocalHome) c.lookup("java:comp/env/LegacyServiceBean");
            return rv.create();
        } catch (NamingException ne) {
            Logger.getLogger(getClass().getName()).log(Level.SEVERE, "exception caught", ne);
            throw new RuntimeException(ne);
        } catch (CreateException ce) {
            Logger.getLogger(getClass().getName()).log(Level.SEVERE, "exception caught", ce);
            throw new RuntimeException(ce);
        }
    }

    public String getMessage() {
        return "Hello from facade " + this.lookupLegacyServiceBean().getCurrentTime();
    }

}
```

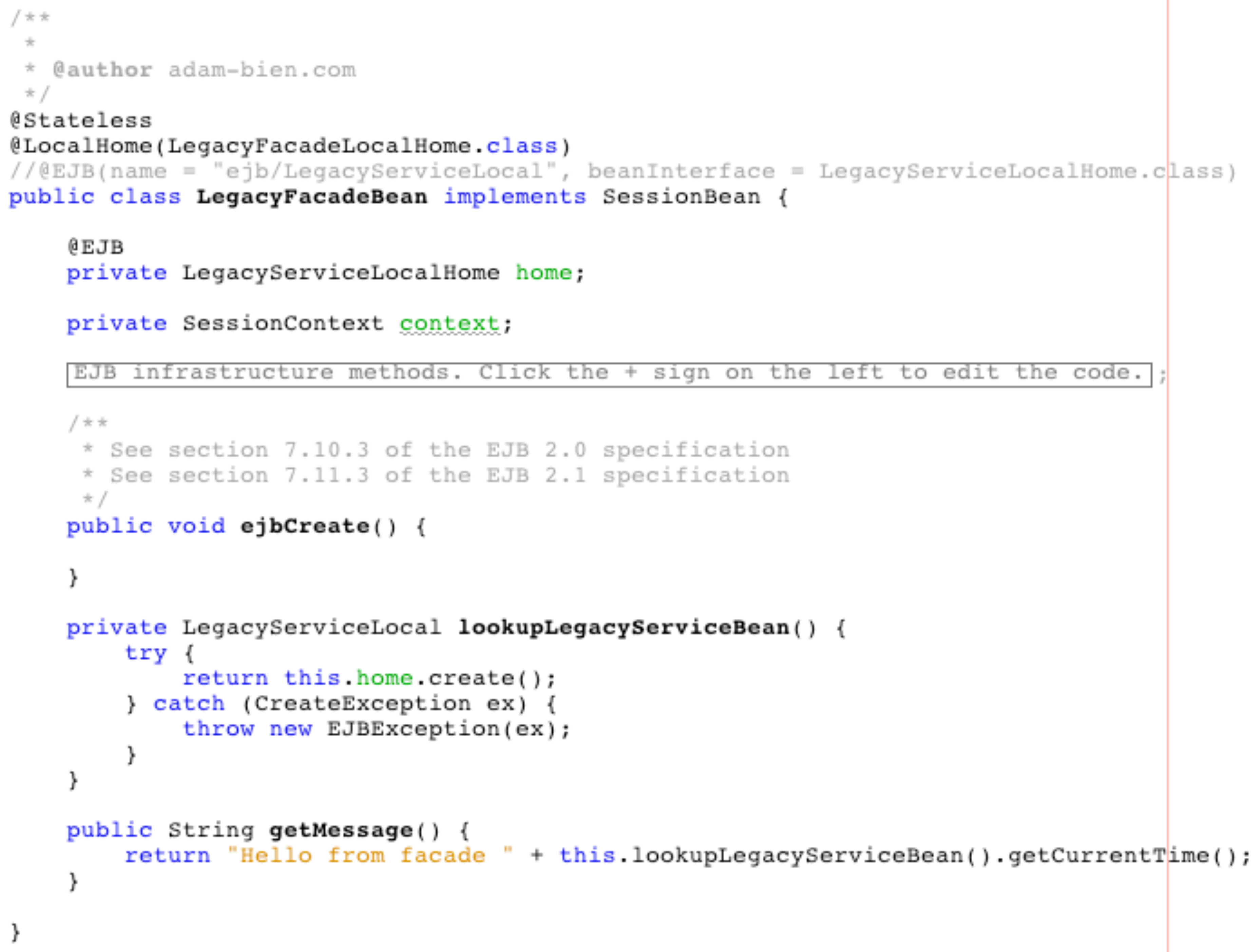
EJBs Are Hard To Migrate (deployment descriptor)



```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar version="2.1" xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instan
  <enterprise-beans>
        <session>
            <display-name>LegacyFacadeSB</display-name>
            <ejb-name>LegacyFacadeBean</ejb-name>
            <local-home>com.abien.patterns.integration.migration.facade.LegacyFacadeLocalHome</local-home>
            <local>com.abien.patterns.integration.migration.facade.LegacyFacadeLocal</local>
            <ejb-class>com.abien.patterns.integration.migration.facade.LegacyFacadeBean</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
            <ejb-local-ref>
                <ejb-ref-name>LegacyServiceBean</ejb-ref-name>
                <ejb-ref-type>Session</ejb-ref-type>
                <local-home>com.abien.patterns.integration.migration.service.LegacyServiceLocalHome</local-home>
                <local>com.abien.patterns.integration.migration.service.LegacyServiceLocal</local>
                <ejb-link>LegacyServiceBean</ejb-link>
            </ejb-local-ref>
            </session>
        <session>
            <display-name>LegacyServiceSB</display-name>
            <ejb-name>LegacyServiceBean</ejb-name>
            <local-home>com.abien.patterns.integration.migration.service.LegacyServiceLocalHome</local-home>
            <local>com.abien.patterns.integration.migration.service.LegacyServiceLocal</local>
            <ejb-class>com.abien.patterns.integration.migration.service.LegacyServiceBean</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
        </session>
        </enterprise-beans>
    <assembly-descriptor>
        <container-transaction>
            <method>
                <ejb-name>LegacyFacadeBean</ejb-name>
                <method-name>*</method-name>
            </method>
            <trans-attribute>Required</trans-attribute>
        </container-transaction>
        <container-transaction>
            <method>
                <ejb-name>LegacyServiceBean</ejb-name>
                <method-name>*</method-name>
            </method>
            <trans-attribute>Required</trans-attribute>
        </container-transaction>
        </assembly-descriptor>
    </ejb-jar>
```

EJBs Are Hard To Migrate

After a “smooth” migration, with minimal amount of changes:



```
/**
 *
 * @author adam-bien.com
 */
@Stateless
@LocalHome(LegacyFacadeLocalHome.class)
//@EJB(name = "ejb/LegacyServiceLocal", beanInterface = LegacyServiceLocalHome.class)
public class LegacyFacadeBean implements SessionBean {

    @EJB
    private LegacyServiceLocalHome home;

    private SessionContext context;

    EJB infrastructure methods. Click the + sign on the left to edit the code.;

    /**
     * See section 7.10.3 of the EJB 2.0 specification
     * See section 7.11.3 of the EJB 2.1 specification
     */
    public void ejbCreate() {

    }

    private LegacyServiceLocal lookupLegacyServiceBean() {
        try {
            return this.home.create();
        } catch (CreateException ex) {
            throw new EJBException(ex);
        }
    }

    public String getMessage() {
        return "Hello from facade " + this.lookupLegacyServiceBean().getCurrentTime();
    }

}
```

EJBs Are Hard To Migrate

EJB 3 are compatible with EJB 2

You can deploy EJB 2 beans as EJB 3

You can then inject EJB 2 instances straight into EJB 3

After this step you can just delete the XML etc.

Is up to you, whether you would like to delete the superfluous artifacts like home-, remote interfaces.

XML deployment descriptors are no more needed after the migration...

11. EJBs Are Hard To Develop

EJBs Are Hard To Develop

Simplest possible EJB 3.1:

```
@Stateless
public class SimpleSample{
    public void doSomething() { /*business logic*/  }
}
```

EJBs Are Hard To Develop

How to compile:

You will need the the EJB 3.0 / 3.1 API in the classpath, or at least the @Stateless annotation.

How to deploy:

Just JAR the class and put the JAR into e.g: [glassfishv3-prelude-b23]\glassfish\domains\domain1\autodeploy

How to use:

```
import javax.ejb.EJB;

public class MyServlet extends HttpServlet{

@EJB

private SimpleSample sample;

}
```

Questions?

Thank you!

Interested in „highend“ trainings, coaching, consulting?

…just send me an email => abien@adam-bien.com