INTRODUCTION TO COMPONENT DESIGN IN JAVA EE

COMPONENT VS. OBJECT,
JAVA EE
JAVA EE DEMO
JAVA ZOOLOGY

Java Standard Edition – Java SE
- Basic types, objects, classes, networking, security,
- Database access, XML parsing, user interfaces

Java Enterprise Edition – Java EE
- Large scale, multi-tier, scalable, reliable apps, components

Java Micro Edition – Java ME
- Mobile devices

Java FX
- Rich Internet Apps, high performance, modern look and feel,
- Clients for Java EE
OBJECT VS. COMPONENT

Object-based design
– construct app from objects

Component-based design
– construct app from preexisting service-providing components

Properties:
• Encapsulation
• Specification – interface
• Improved reuse and evolution
• Abstraction
**VALUE OBJECT VS REFERENCE OBJECT**

**Object-based design** - objects have identity

- **Reference object** – e.q. a Customer
  - One object identifies a customer in the real world
  - Any reference to the customer is a pointer to the Customer objects!
  - Changes to the customer object available to all users!
  - Compare identity

- **Value Object** - a small object that represents a simple entity like Date, Money
  - Multiple value objects represent the same real world thing
  - Hundred of objects that represent Jun 5\(^{th}\), 2015
  - Comparing dates does not compare identity but the value!
  - Its equality is not based on identity:
    - two value objects are equal when they have the same value,
    - not necessarily being the same object.

```
Person joel = getJoe();
Person joe2 = getJoe();
joel == joe2
Person bob = getBob();
bob.born.equals(joel.born)
```
OBJECT VS. COMPONENT

Component not language specific

- Organization unit, building block, functional element.
- Comparison
  - An object is a component
  - Collection of objects is a component

Components connect together, and usually have dependencies, although we think of a component as an independent functional block.

- e.g. OSGi standard – automobiles and industry automation

Component has usually specification and realization (Interfaces and implementation in the Object-based design)
OBJECT VS. COMPONENT

Object-based design – identity oriented – domain abstraction

Component-based design – service oriented – functional abstraction
Enterprise Application (EA)

- Tiered Applications
- Functionality separated into isolated functional areas – tiers

  - e.g.
    - **Client tier** – client app
    - **Web tier** – server-side controllers
    - **Business tier** – business functions
    - **EIS tier** – data store
    - ..
CLIENT TIER

Usually a different machine access to Java EE server.
Request – response communication
Client can be
- A web browser
- Standalone app
- Another server

Can use a different platform
WEB TIER

Components handling interaction between clients and business tier. Does the following tasks

- Dynamic content derivation in various formats
  - HTML, XML, JSON
- Collect user input, return results
- Control flow
- Maintain state of user session
- Basic logic

- Java EE Technologies
  - Servlets, Java ServerFaces (JSF), Facelets,
  - Expression language, Java Server Pages (JSP),
  - JSP Tag library, JavaBeans Components

Tomas Cerny, Software Engineering, FEE, CTU in Prague, 2016
WEB TIER
WEB TIER – JAVA EE TECHNOLOGIES

- **Servlets** - classes to dynamically process request and give response in HTML
- **Java ServerFaces (JSF)** – user interface component framework for web apps to include UI components on a page, convert, validate data, maintain state, save data
- **Facelets** – templating ang XHTML,
- **Expression Language** – reference Java EE components from JSP/Facelets
- **Java Server Pages (JSP)** – Text based document compiled to servlet, define dynamic content added to static pages – e.g. HTML
- **JSP Tag Library** – core functionality of tags
- **JavaBeans Components** – object that acts as temporary data store for app
BUSINESS TIER

Components that provide business logic of an application.

Business logic – is a code that provides functionality to a particular business domain.

- Financial industry
- E-commerce site

Good design has the core functionality in business tier components

Java EE Technologies

- Enterprise JavaBeans (EJB), JAX-RS RESTful web service endpoints, JAX-WS web service endpoints, Java Persistence API entities, Java EE manager beans.
BUSINESS TIER

Application Client and Optional JavaBeans Components

Web Browser, Web Pages, Applets, and Optional JavaBeans Components

Client Tier

JavaBeans Components (Optional)

Web Pages, Servlets

Web Tier

Java EE Server

Java Persistence Entities, Session Beans, Message-Driven Beans

Business Tier

Database and Legacy Systems

EIS Tier
BUSINESS TIER
JAVA EE TECHNOLOGIES

• Enterprise JavaBeans (EJB) – component that encapsulate the core functionality of an app

• JAX-RS RESTful web service endpoints – API to create web services on top of HTTP, REST – representational state transfer

• JAX-WS web service endpoints – creating and consuming SOAP web services

• Java Persistence API entities – API for accessing data in underlying data stores and mapping to Java objects

• Java EE managed beans – managed components that may provide business logic, but do not require transaction of security features of EJB
  • Light weight POJO with minimal requirements
  • Small set of basic services
ENTERPRISE INFORMATION SYSTEM (EIS) TIER

Usually contains, database servers, resource planning, legacy data sources, etc.

Resources usually distributed across different machines than the Java EE server and are accessed through components in business tier.

Java EE Technologies

Java Database Connection API (JDBC), Java Persistence API (JPA), Java EE Connector Architecture (JCA), Java Transition API (JTA)
ENTERPRISE INFORMATION SYSTEM (EIS) TIER

Usually contains, database servers, resource planning, legacy data sources, etc.

Resources usually distributed across different machines than the Java EE server and are accessed through components in business tier.

Java EE Technologies

Java Database Connection API (JDBC) – low level API to access and retrieve data from data store. Connects to SQL relational database

Java Persistence API (JPA) – Access the underlaying data stores through java Objects. On top of JDBC.

Java EE Connector Architecture (JCA) – API to connect to enterprise resources, like resource planning, customer management system, etc.

Java Transition API (JTA) – API to define and manage transitions, including distributes transactions across multiple data resources.
JAVA EE PLATFORM

Java EE 7

- JSP 2.3
- JSF 2.2
- JAX-RS 2.0
- EL 3.0
- Servlet 3.1
- Managed Beans 1.0
- EJB 3.2
- Connector 1.7
- JPA 2.1
- JTA 1.2
- JMS 2.0

- Common Annotations 1.2
- Interceptors 1.2
- CDI 1.1
- Bean Validation 1.1
- Concurrency Utilities (JSR 236)
- Batch Applications (JSR 352)
- Java API for JSON (JSR 353)
- Java API for WebSocket (JSR 356)

New | Major Release | Updated
JAVA EE

APPLICATION SERVERS

Implements the Java EE platform API
Provides standard services
Hosts several application components

Provides containers

- Interface between component and low-level functionality
- Web container
- Application client container
- EJB container
JAVA EE

APPLICATION SERVERS

Web container (1)
- Interface web component and web server
- Component Servlet/JSF/JSP page
- Container manages its lifecycle, dispatch request, provides context information

Application client container (2)
- Java EE app clients using Java EE server components
- Distinct machines

EJB container (3)
- Interface between EJB that provides business logic and the Java EE server
- EJB container manages the execution of the EJB
JAVA EE
APPLICATION SERVERS

Web container (1)

Application client container (2)

EJB cont. (3)
JAVA EE

APPLICATION SERVERS

[Diagram showing the components of an application server, including Application Client Container, Web Browser, Servlet, Web Container, EJB Container, and Database.]
JAVA EE API

CONTAINERS
Functional components

- **Enterprise beans** = Enterprise JavaBeans (EJB)
  - **Session beans** – transient conversation with client. Once client servers the session bean and its data are gone
  - **Message driven beans** – session bean features and message listener – receive messages asynchronously. Interacts with Java Message Service (JMS)
    - Multiple services can interact through messages

- Web page
- Servlet
- JSF/JSP
- Applet
JAVA EE
COMPONENTS

Many components needs to be connected

Introducing **high coupling**

**Contexts and Dependency Injection (CDI)**

- Contextual services in Java EE container
- Integration of components with **loose coupling and typesafety**
- Dependency injection
Example
JAVA EE PLATFORM

Java EE 7

- Portable Extensions
- Common Annotations 1.2
- Managed Beans 1.0
- Connector 1.7
- JSP 2.3
- JSF 2.0
- JAX-RS 2.0
- EL 3.0
- Servlet 3.1
- Interceptors 1.2
- CDI 1.1
- EJB 3.2

- Bean Validation 1.1
- Concurrency Utilities (JSR 236)
- Batch Applications (JSR 352)
- Java API for JSON (JSR 353)
- Java API for WebSocket (JSR 356)

Colors:
- New
- Major Release
- Updated
DEMO
SAMPLE CONFIGURATION

Get Eclipse Mars for Java EE + Install JBoss Tools Plugin*


WildFly Application Server 9/10

PostgreSQL + pgAdmin

Apache Maven

Java 8 JDK

Play examples here:

https://java.net/projects/firstcup/
https://github.com/wildfly/quickstart
https://java.net/downloads/glassfish-samples/javaee7-samples-1.0.zip

*http://tools.jboss.org/downloads/jbosstools/mars/4.3.0.Final.html#update_site
JBoss Sample Apps

1. Open Eclipse that has JBoss Tools installed *
2. File | New | Other
3. Examples | JBoss Tools | Project Examples | Next >>
4. Web Applications | helloworld | Next >> | select server/runtime
5. Download and Install.. | WildFly 9.0.1 | accept terms | fill path | Install
6. Wait until installs | Next | Use default location | Finish | wait | Finish*
7. Open readme.md and see “Run the Quickstart in JBoss Developer Studio or Eclipse”

*http://tools.jboss.org/downloads/jbosstools/mars/4.3.0.Final.html#update_site
This will deploy `target/jboss-helloworld.war` to the running instance of the server.

Access the application

The application will be running at the following URL: <http://localhost:8080/jboss-helloworld>.

Under deploy the Archive

1. Make sure you have started the JBoss EAP server as described above.
2. Open a command prompt and navigate to the root directory of this quickstart.
3. When you are finished testing,

CDI

Contexts and Dependency Injection (CDI) defines a set of complementary services that help improve the structure of application code. CDI provides type-safe dependency injection and shrinks the application code base by replace large sections of code with annotations. It also simplifies packaging and deployments and provides lifecycle management via contexts.

Servlet 3.0

Servlet 3.0 introduces @Servlet, @ServletFilter, @FilterMapping, and other annotations to ease development of servlet applications.

Click to Begin
**JBOS SAMPLE APPS**

8. Fix class dependencies is any [In my case pom.xml change]

```xml
<version.jboss.spec.javaee.6.0>3.0.2.Final-redhat-15</version.jboss.spec.javaee.6.0>
<version.jboss.spec.javaee.6.0>3.0.2.Final</version.jboss.spec.javaee.6.0>

Version 3.0.2.Final-redhat-13 to 3.0.2.Final
```

9. Right-click on WildFly | Start | go to web http://localhost:8080/

10. * See the running process in Unix 

```
ps aux | grep java
```

11. Right-click on jboss-helloworld project | Run As | Run on Server

12. Select WildFly 9 | Next | Verify jboss-helloworld | Finish

13. See console and web browser at

- http://localhost:8080/jboss-helloworld/HelloWorld

14. Servers | Right-click | Stop
14. Servers | Right-click | Stop

15. Servers | Right-click | **Debug**

16. Put debug break point (double click) to
   - HelloWorldService.java Line 28
   - HelloWorldServlet Line 55

17. Open web browser with address
   - [http://localhost:8080/jboss-helloworld/HelloWorld](http://localhost:8080/jboss-helloworld/HelloWorld)

18. Switch back to eclipse and see Confirmation on Debug View | Yes
This quickstart demonstrates the features of CDI 1.0 and 3.0. It is a simple application that can be used to verify the JBoss EAP server configuration and run correctly.

CDI

Contexts and Dependency Injection (CDI) defines a set of annotations and stereotypes that help improve the structure of application code. CDI provides a type-safe dependency injection and shrinks the application code base by replacing hand-rolled sections of code with annotations. It also simplifies packaging and deployment of applications by providing lifecycle management via contexts.
19. Step next in debug view until line 58 then step into (out/in few time)
20. See the stack that corresponds to HelloService.createHelloMessage
21. See the parameter value: name = “World”
22. In the Debug panel click on HelloWorldServlet.doGet(....
23. Change line 58 servlet param to Your Name! and repeat step 17
24. Click Resume (F8) in debug view
25. Nothing happen!?
   • Right-click server | publish
   • Restart server and try again
   • Still broken?
     • Right-click project | run as | run on server
     • Open server and pick the project | Full Publish | restart OR Remove
JBOSS SAMPLE APPS

• Components
  • HelloWorldServlet = @WebServlet("/HelloWorld")
    • Extends HttpServlet
    • Implements doGet (HTTP GET) similarly doPost
  • Open in Eclipse HelloWorldServlet
    • hold ctrl and left click on HttpServlet
  • **Maven** fetches the source code for you!

• CDI
  • Notice the connection HelloWorldServlet and HelloService
  • @Inject
HELLOWORLD-HTML5

Try: example helloworld-html5

• See index.html
• HelloWorld no longer servlet instead a web service

@Path("/")
public class HelloWorld {
    @Inject
    HelloService helloService;

    @GET
    @Path("/json/{name}"),
    @Produces("application/json")
    public String getHelloWorldJSON(@PathParam("name") String name) {
        return "\"result\":\"" + helloService.createHelloMessage(name) + "\"\";
    }
}
HELLOWORLD-HTML5

Try: example helloworld-html5

• See index.html
• HelloWorld no longer servlet instead a web service

• Access http://localhost:8080/jboss-helloworld-html5/

• Then go to
  • http://localhost:8080/jboss-helloworld-html5/hello/json/aa
  • http://localhost:8080/jboss-helloworld-html5/hello/xml/aa
JAVA EE WEB PROJECT

Try: example JBoss Maven Archetypes / Java EE Web Project

Finish deploy and go to

http://localhost:8080/jboss-javaee6-webapp/index.jsf

Add yourself to the form
Welcome to WildFly!

You have successfully deployed a Java EE 7 Enterprise Application.

Your application can run on:

![WildFly](image)

### Member Registration
Enforces annotation-based constraints defined on the model class.

**Name:**

**Email:**

**Phone #:**

[Register]

- Registered!

### Members

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Email</th>
<th>Phone #</th>
<th>REST URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bob</td>
<td><a href="mailto:lala@foo.la">lala@foo.la</a></td>
<td>5645644545</td>
<td>/rest/members/1</td>
</tr>
<tr>
<td>0</td>
<td>John Smith</td>
<td><a href="mailto:john.smith@mailinator.com">john.smith@mailinator.com</a></td>
<td>2125551212</td>
<td>/rest/members/0</td>
</tr>
</tbody>
</table>

REST URL for all members: /rest/members

This project was generated from a Maven archetype from WildFly.
SEE THE STRUCTURE
SEE THE STRUCTURE

Data Definition and Access

- Member.java
  - See field annotations - validation
- JPA + XML
- EntityManager Producer - Resources.java
  - MemberRepository.java – finder!

Presentation

- index.html
- JSF + Facelets

REST

- MemberResourceRESTService.java + JaxRsActivator.java
  - http://localhost:8080/jboss-javaee6-webapp/rest/members

Business EJB

- MemberRegistration.java @Stateless EJB
  - CDI inject entityManager = events

Controllers

- MemberListProducer.java – observer events
- MemberController.java see #register() and binding to Member.java
WHERE TO GO NEXT?

See all sources at:

https://github.com/jboss-developer/jboss-eap-quickstarts
HOMEWORK

1. Add person removal function
2. Add person info update feature
3. Make page transition to person detail though JSF dispatch
4. Connect to PostgreSQL database
5. Make a named query
INSTALL POSTGRES

Install postgres server
Make postgres user a password
Make a user testuser with somepass

```
$ sudo -u postgres psql
postgres=> alter user postgres password 'XXX';
posgres=> create user testuser createdb createuser password 'somepass';
posgres=> create database testdb owner testuser ;
posgres=> \q
$ ...
```
INSTALL POSTGRES PGADMIN

http://www.pgadmin.org/

Install and connect to localhost:5432
Use your user / password
Or postgres / password

Add database testdb
Owner testuser
Definition|Template template0
SERVER ADMIN CONSOLE

Start server | Go to http://localhost:8080 and click administr. console

It takes you to http://localhost:8080/console

and redirects to http://localhost:9990/error/index.html

See the management instructions for console follow steps below - to add user

Go to your server bin folder such as cd ~/wildfly-9.0.1.Final/bin/

```bash
~/wildfly-9.0.1.Final/bin$ chmod +x ./add-user.sh
~/wildfly-9.0.1.Final/bin$ ./add-user.sh
a [enter]
admin [enter]
admin [enter]
*yes to all
```
1. Go again to [http://localhost:8080/console](http://localhost:8080/console) and login admin admin
2. Read through [https://developer.jboss.org/wiki/JBossAS7-DatasourceConfigurationForPostgresql](https://developer.jboss.org/wiki/JBossAS7-DatasourceConfigurationForPostgresql)
3. Download postgresql-9.3-1103.jdbc41.jar
4. And move it to `~/wildfly-9.0.1.Final/standalone/deployments`
5. Restart server
7. Click Add
REGISTER POSTGRES TO SERVER II.

7. Click Add | select PostgreSQL | Next | Next | Detected Driver

8. Pick Postgresql-9.3.. | Setup connection
URL: jdbc:postgresql://localhost:5432/testdb
Username: testuser
Password: somepass
9. Click Test Connection if it passed then click Done
REGISTER POSTGRES TO SERVER III.

10. See the new data source and the driver name

<table>
<thead>
<tr>
<th>Name</th>
<th>JNDI</th>
<th>Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExampleDS</td>
<td>java:jboss/datasources/ExampleDS</td>
<td>✔️</td>
</tr>
<tr>
<td>PostgresDS</td>
<td>java:/PostgresDS</td>
<td>✔️</td>
</tr>
</tbody>
</table>

**Attributes**
- **Name**: PostgresDS
- **JNDI**: java:/PostgresDS
- **Is enabled?**: true
- **Statistics enabled?**: false
- **Driver**: postgresql-9.3-1103.jdbc41.jar
CONNECT WEB APP TO POSTGRES VIA DRIVER

Go to `jboss-javaee6-webapp-ds.xml` and replace H2 with Postgres

```xml
<datasources xmlns="..">
    <datasource jndi-name="java:jboss/datasources/jboss-javaee6-webappDS"
        pool-name="jboss-javaee6-webapp" enabled="true"
        use-java-context="true">
        <connection-url>jdbc:postgresql://localhost:5432/testdb</connection-url>
        <driver>postgresql-9.3-1103.jdbc41.jar</driver>
        <security>
            <user-name>testuser</user-name>
            <password>somepass</password>
        </security>
    </datasource>
</datasources>
```

Add dialect to `persistence.xml`

```xml
<property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect"/>
```

SEE YOUR PGADMIN

1. Open pgAdmin and right click databases node and refresh
2. The testdb appears
3. Open it | schemas | public
4. Open Tables | members
5. Right click | view data

<table>
<thead>
<tr>
<th>id [PK] bigint</th>
<th>email character var</th>
<th>name character var</th>
<th>phone_number character var</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>john.smith@email</td>
<td>2125551212</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td><a href="mailto:bob@example.com">bob@example.com</a></td>
<td>5646465123</td>
</tr>
<tr>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
IF YOU STOP SERVER
DATA DISAPPEAR

1. Go to persistence.xml
2. Replace properties with

```
<property name="hibernate.hbm2ddl.auto" value="update" />
<property name="hibernate.show_sql" value="true" />
<property name="hibernate.format_sql" value="true" />
<property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect" />
```

3. In Eclipse open Server | WildFly | right click webapp | Full publish
4. Restart server
5. Go to http://localhost:8080/jboss-javaee6-webapp
6. See console SQL
7. Add person and see console SQL
NAMED QUERY HINT

1. Go to Member.java

```java
@Entity
@XmlElement
@Table(uniqueConstraints = @UniqueConstraint(columnNames = "email"))
@NamedQueries({
    @NamedQuery(name = "Member.findAll", query = "SELECT m FROM Member m"),
})
public class Member implements Serializable { ..

public List<Member> findAllOrderedByName() {
    return em.createNamedQuery("Member.findAll", Member.class).getResultList();
}
```