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

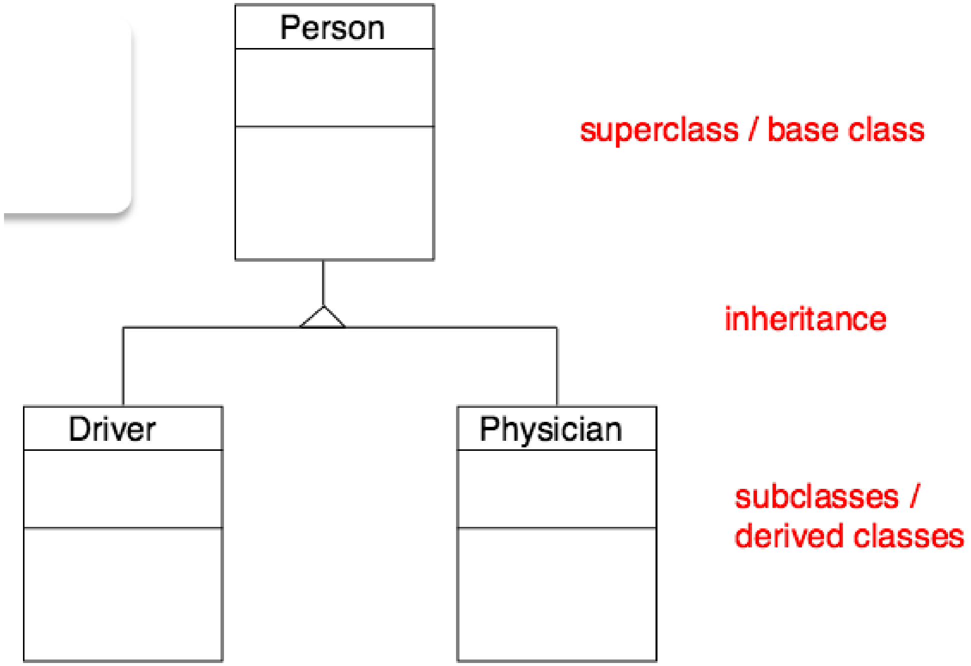
Java Micro Edition – Java ME

- Mobile devices

Java FX

- Rich Internet Apps, high performance, modern look and feel,
- Clients for Java EE

OBJECT VS. COMPONENT

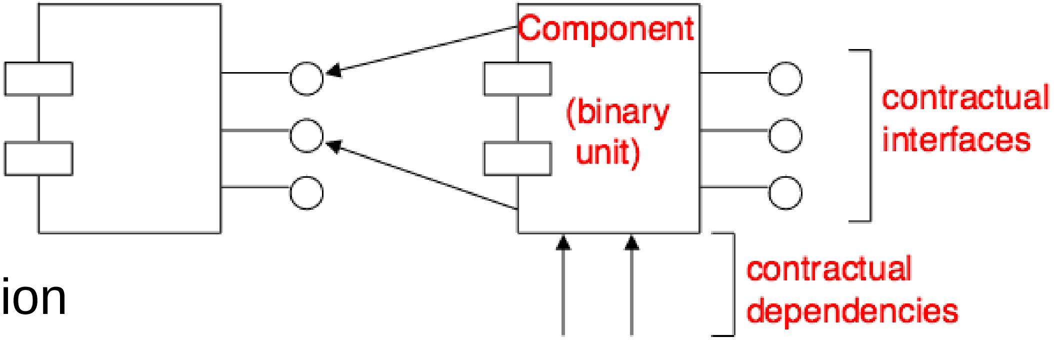


Object-based design

– 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

```
Person joe1 = getJoe();
Person joe2= getJoe();
joe1 == joe2
Person bob = getBob();
bob.born.equals(joe1.born)
```

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 5th, 2015
  - Comparing dates does not compare identify 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.

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**.**

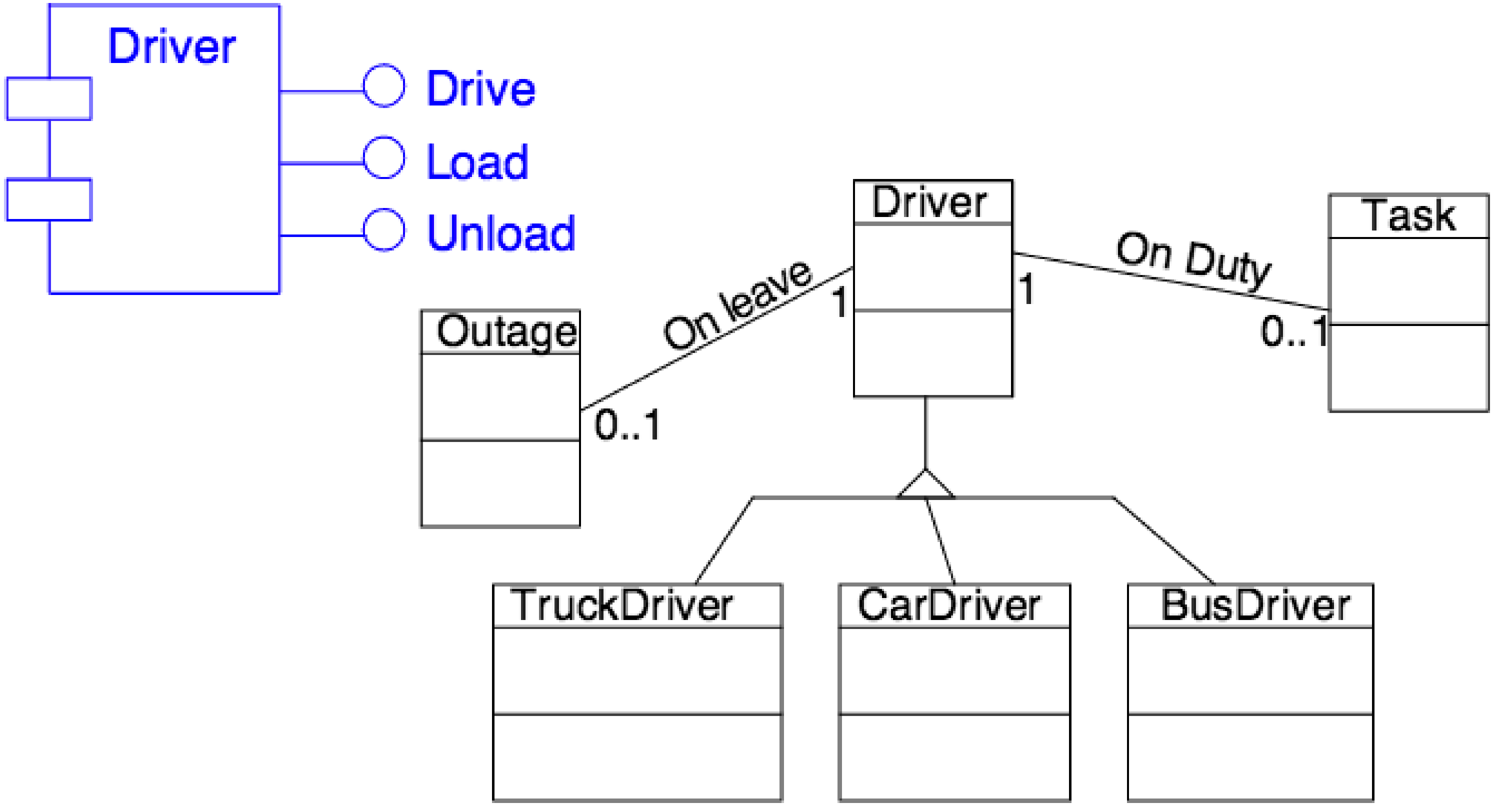
- e.g. OSGi standard – automobiles and industry automation

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

OBJECT VS. COMPONENT

Object-based design – identity oriented – domain abstraction

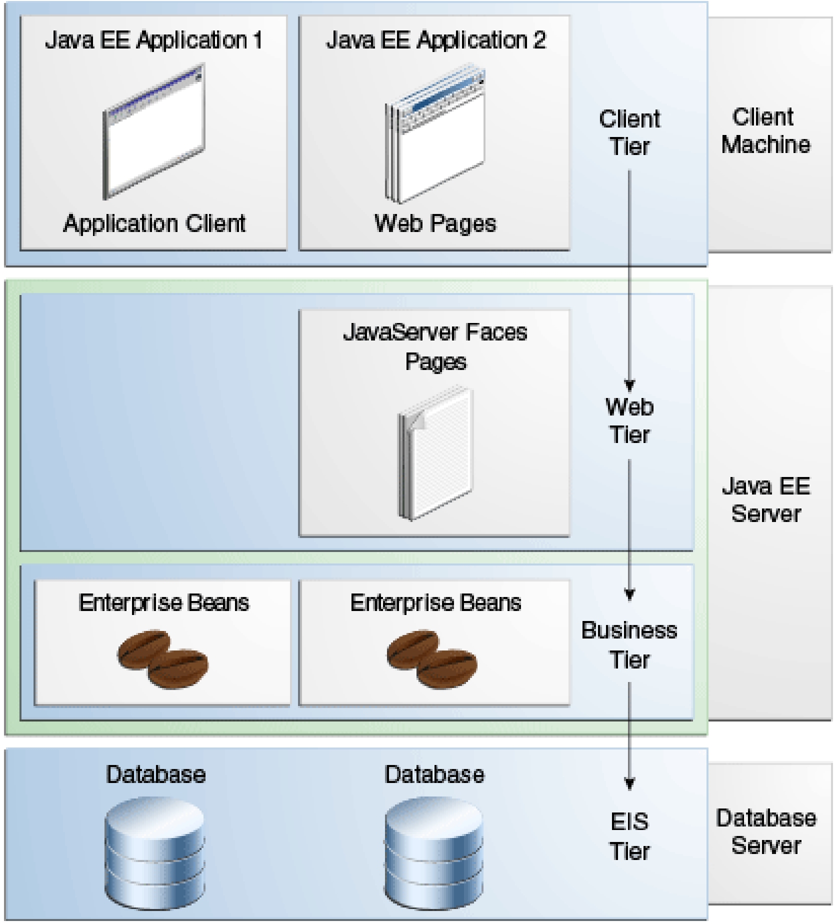
Component-based design – service oriented – functional abstraction



ENTERPRISE APP DESIGN
JAVA EE

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

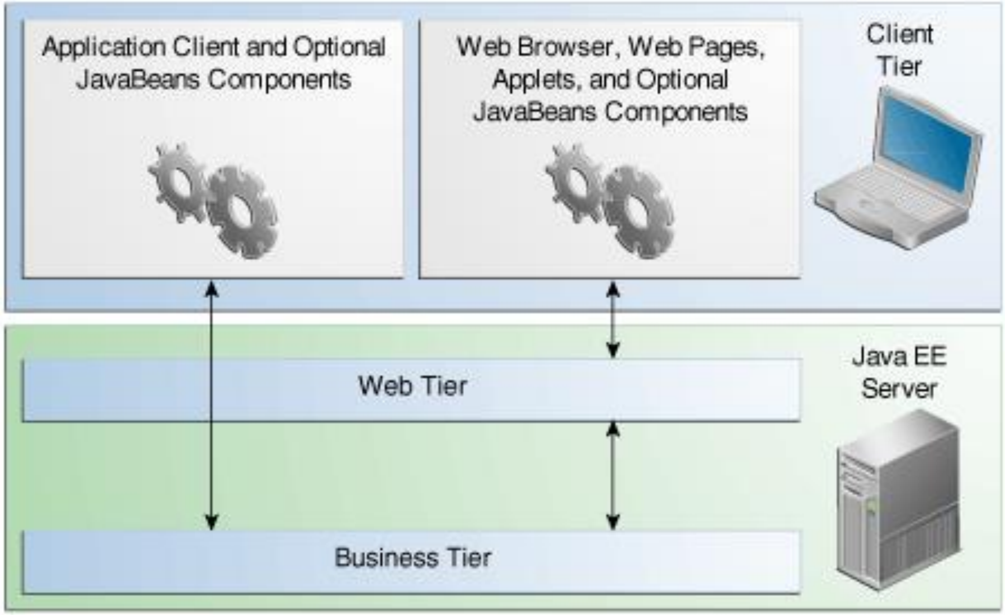
Usually a different machine access to Java EE server.

Request – response communication

Client can be

- A web browser
- Standalone app
- Another server

Can use a different platform


Application Client and Optional JavaBeans Components
Web Browser, Web Pages, Applets, and Optional JavaBeans Components
Client Tier
Web Tier
Java EE Server
Business Tier

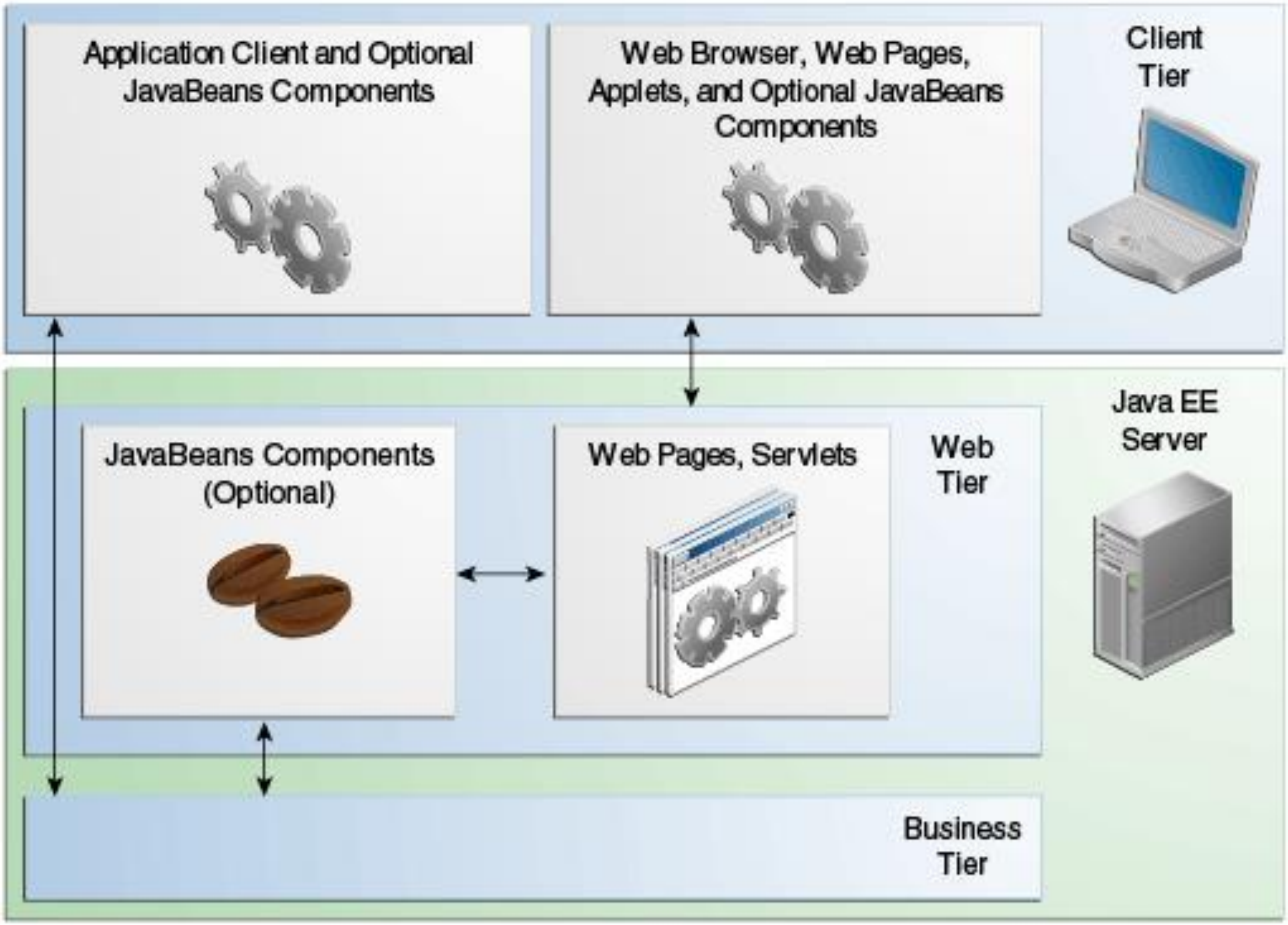
WEB TIER

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 (later in more detail..)**
  - Servlets, Java ServerFaces (JSF), Facelets,
  - Expression language, Java Server Pages (JSP),
  - JSP Tag library, JavaBeans Components

WEB 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
Business 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.

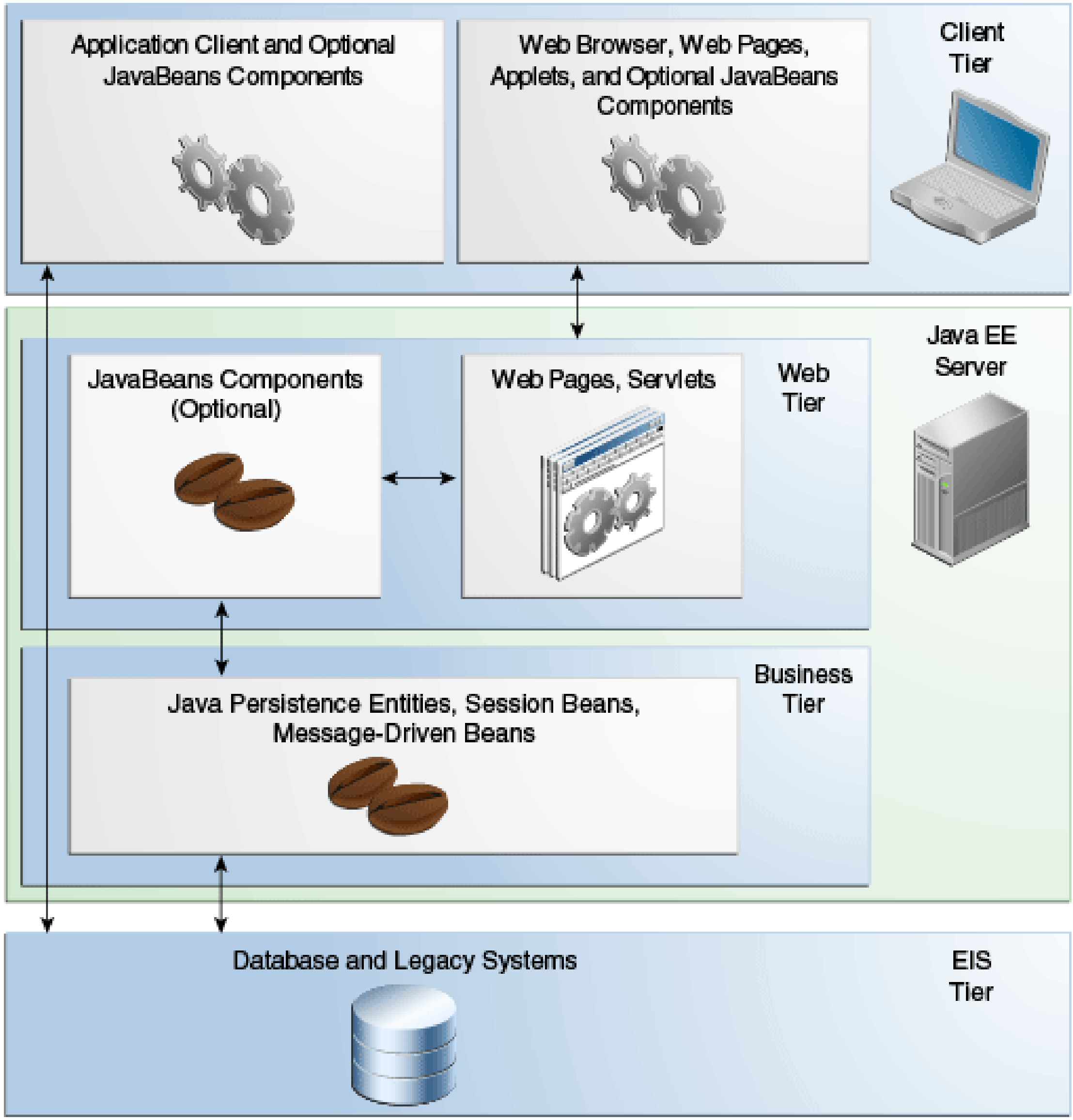
- **Financial industry**
- **E-commerce site**

Good design has the core functionality in business tier components

Java EE Technologies **(later in more detail..)**

- **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

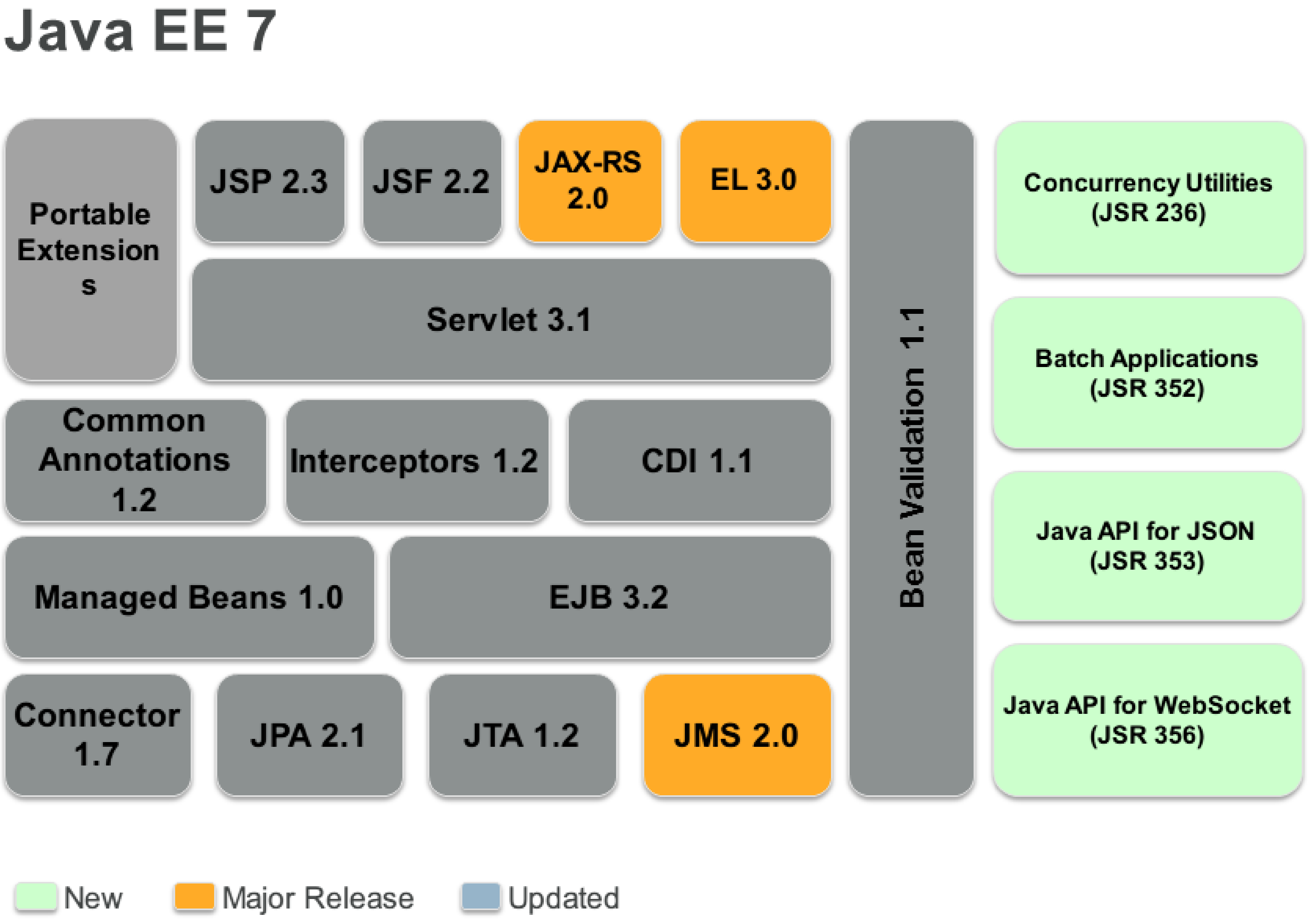
Java Database Connection API (JDBC) – low level API to access and retrieve data from data store. Connects to SQL relational database

Java Persistence API (JPA) – Access the underplaying 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

Portable Extensions

JSP 2.3

JSF 2.2

JAX-RS 2.0

EL 3.0

Servlet 3.1

Common Annotations 1.2

Interceptors 1.2

CDI 1.1

Managed Beans 1.0

EJB 3.2

Connector 1.7

JPA 2.1

JTA 1.2

JMS 2.0

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

Who understands the Java EE components?

The interpret!

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 (large at server)**
- **Application client container (small at client)**
- **EJB container (middle at server)**

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)**

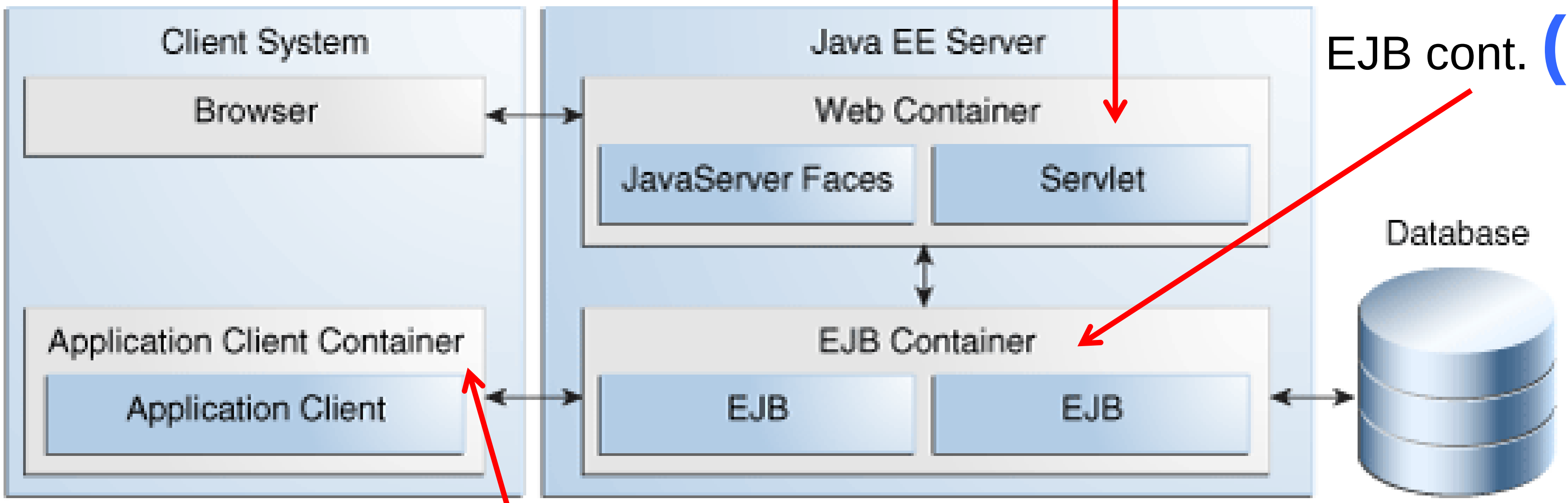
- Java EE app clients using Java EE server components
- Distinct machines

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



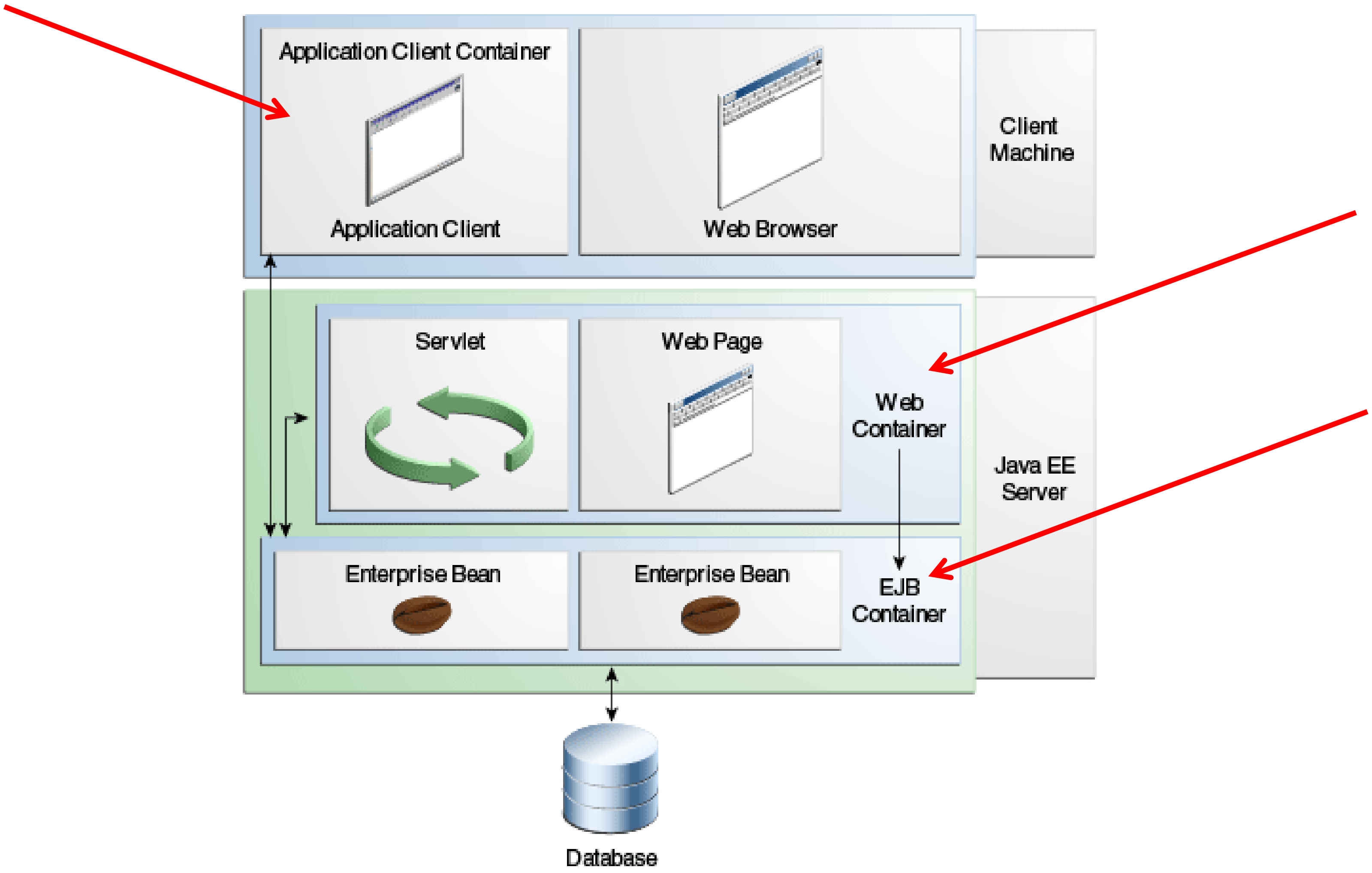
Web container **(1)**

EJB cont. **(3)**

Application client container **(2)**

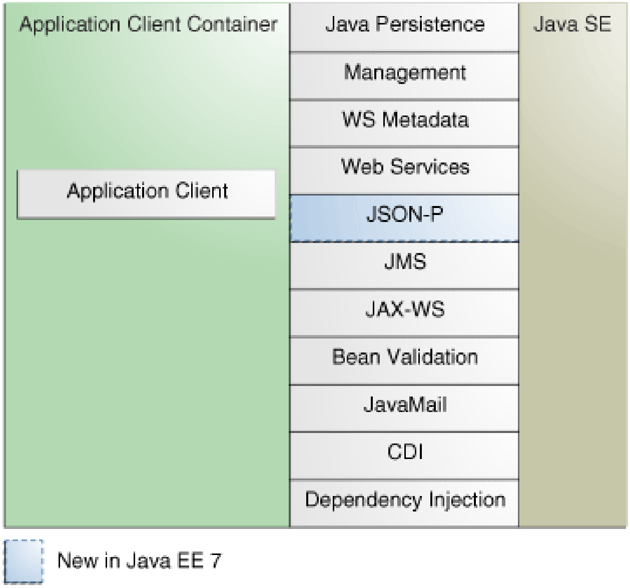
JAVA EE

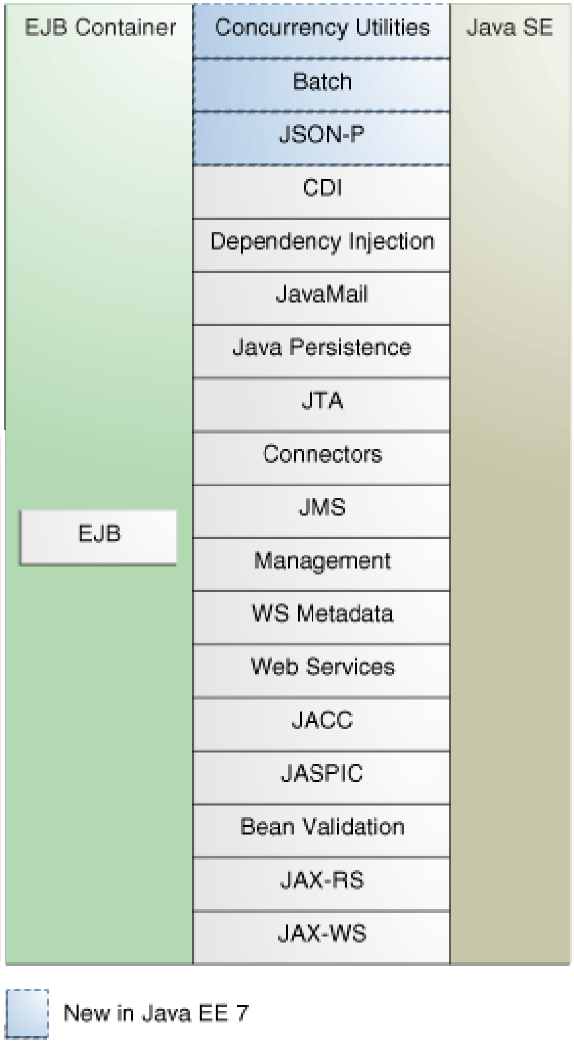
APPLICATION SERVERS

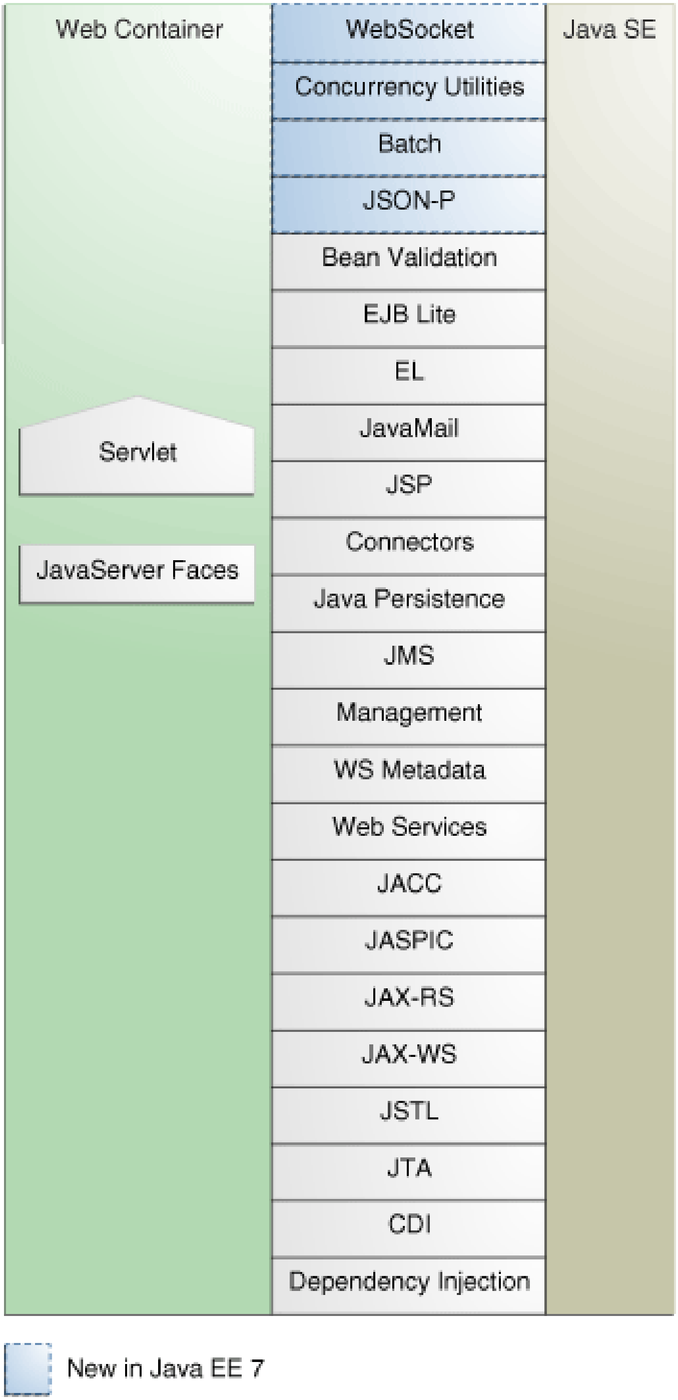


JAVA EE API

CONTAINERS







JAVA EE

COMPONENTS

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

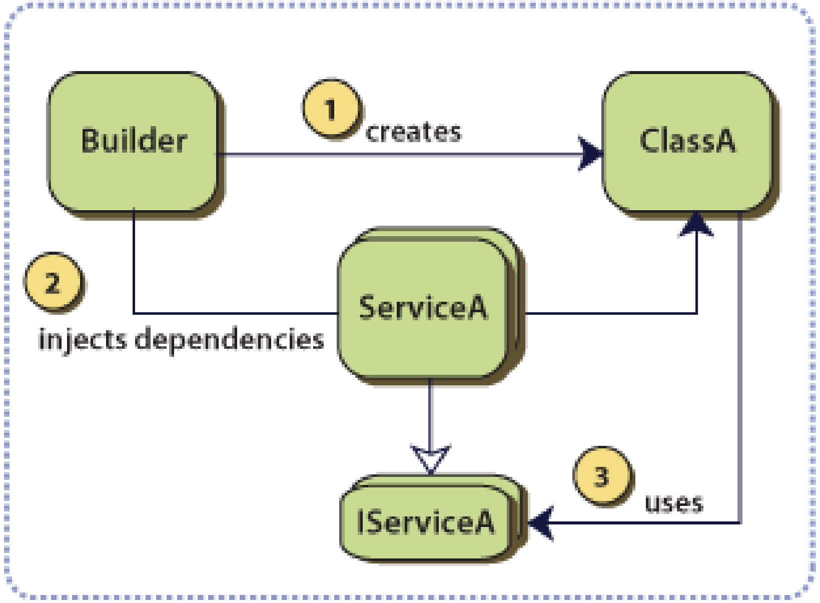
COMPONENTS

Many components needs to be connected

Introducing high coupling

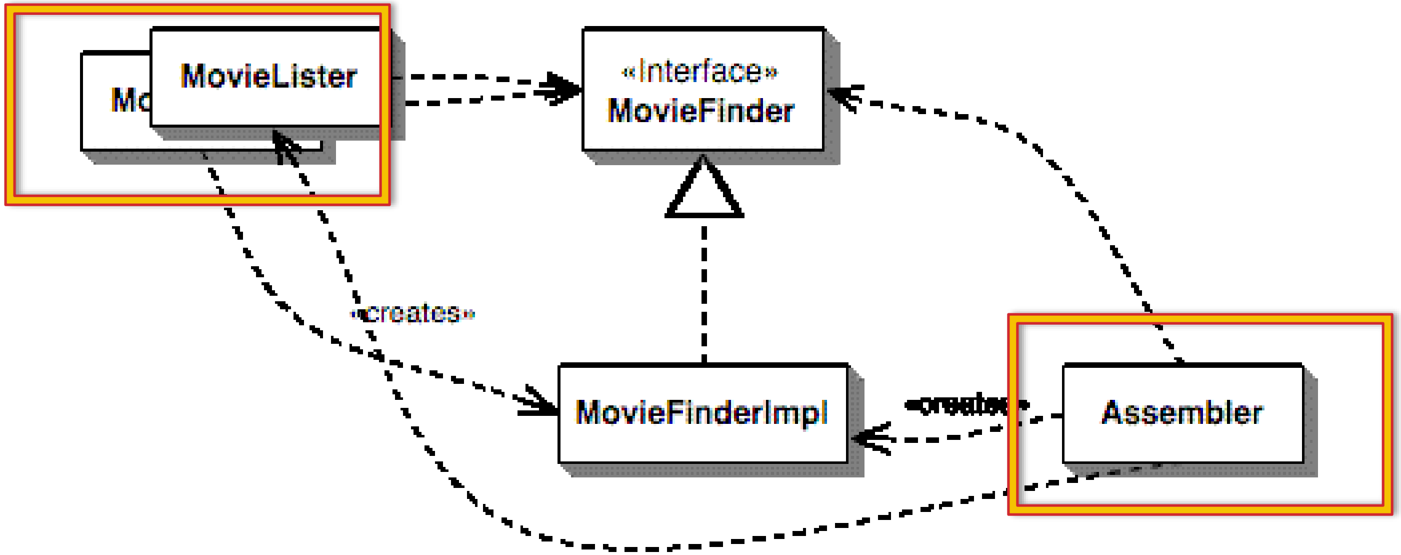
Contexts and Dependency Injection (CDI)

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

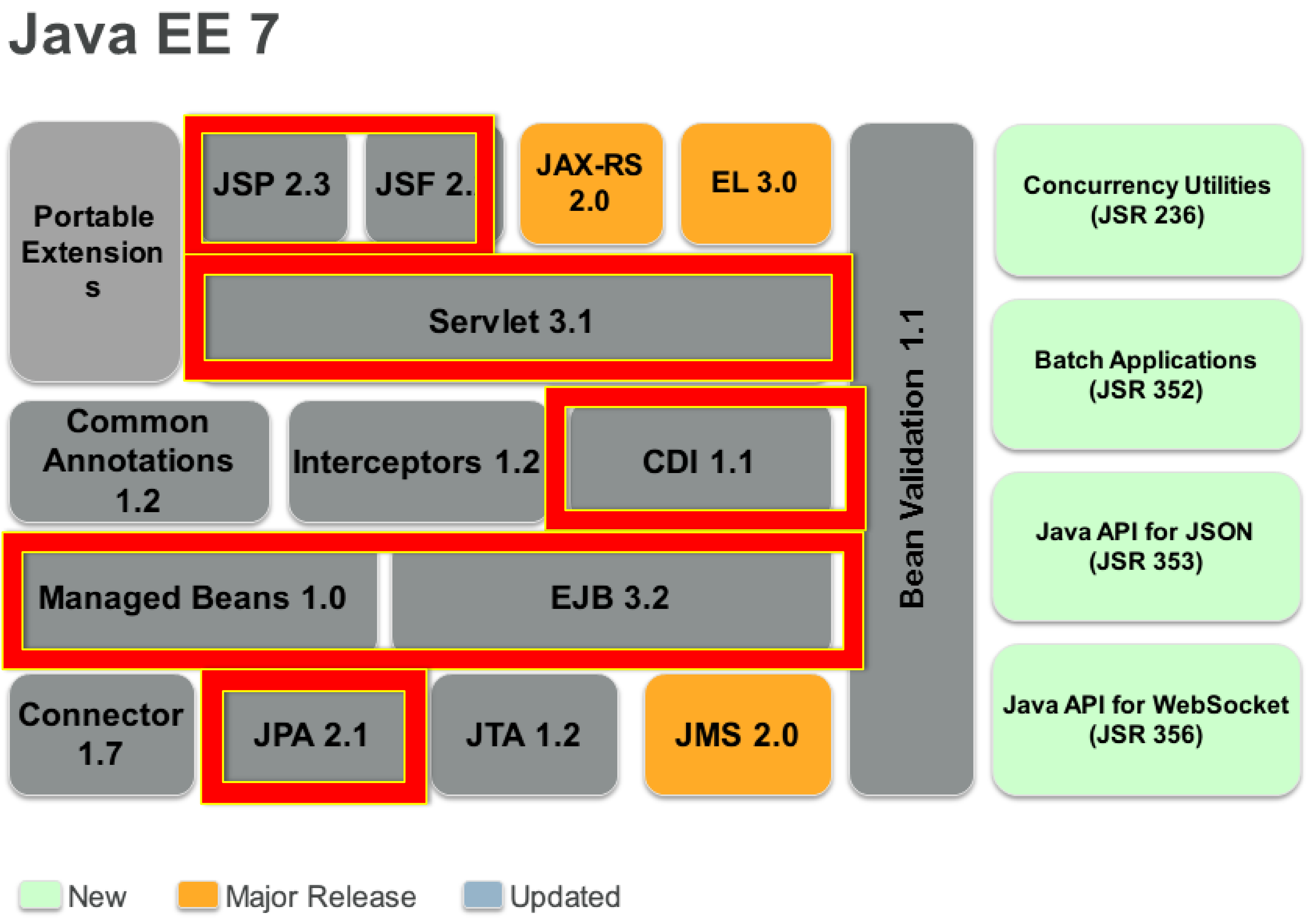


JAVA EE COMPONENTS - DEPENDENCY INJECTION

Example



JAVA EE PLATFORM



DEMO

SAMPLE CONFIGURATION

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

- http://tools.jboss.org/downloads/installation.html

WildFly Application Server 9/10

PostgreSQL + pgAdmin

Apache Maven

Java 8 JDK

Play examples here:

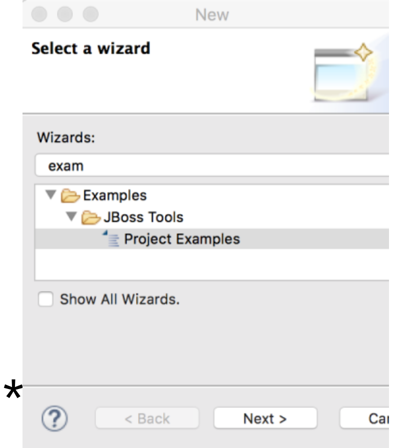
https://java.net/projects/firstcup/

https://github.com/wildfly/quickstart

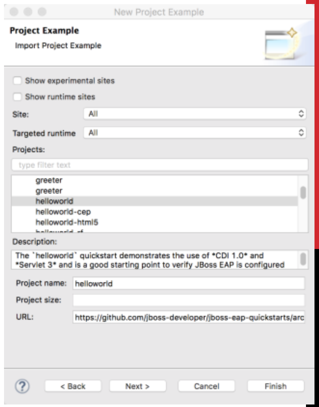
https://java.net/downloads/glassfish-samples/javaee7-samples-1.0.zip

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

JBOSS SAMPLE APPS



Wizards: exam



Site: All

Targeted runtime: All

Projects: helloworld

Description: The `helloworld` quickstart demonstrates the use of *CDI 1.0* and *Servlet 3* and is a good starting point to verify JBoss EAP is configured

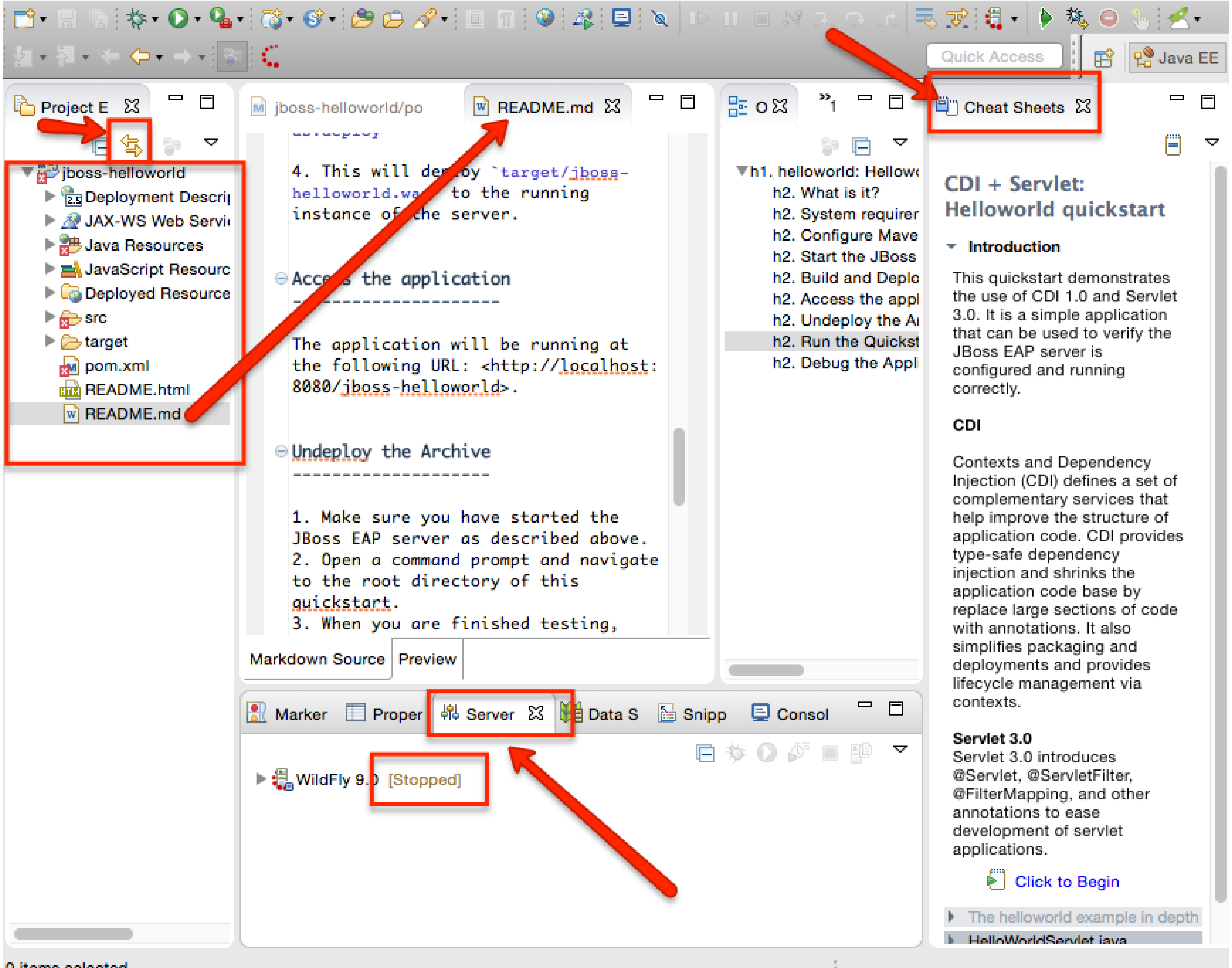
Project name: helloworld

Project size:

URL: https://github.com/jboss-developer/jboss-eap-quickstarts/arc

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. <u>Wait</u> until installs | Next | Use default location | Finish | <u>wait</u> | 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



```
4. 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>.


Undeploy 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 + Servlet: Helloworld quickstart

Introduction

This quickstart demonstrates the use of CDI 1.0 and Servlet 3.0. It is a simple application that can be used to verify the JBoss EAP server is configured and running correctly.

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.

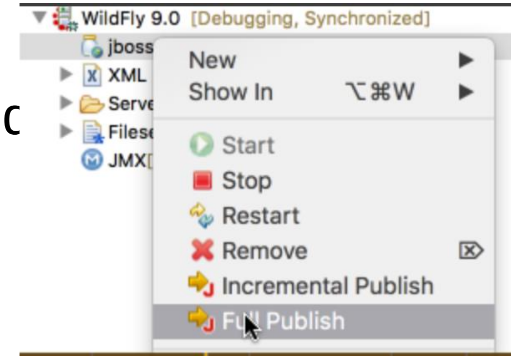
JBOSS SAMPLE APPS

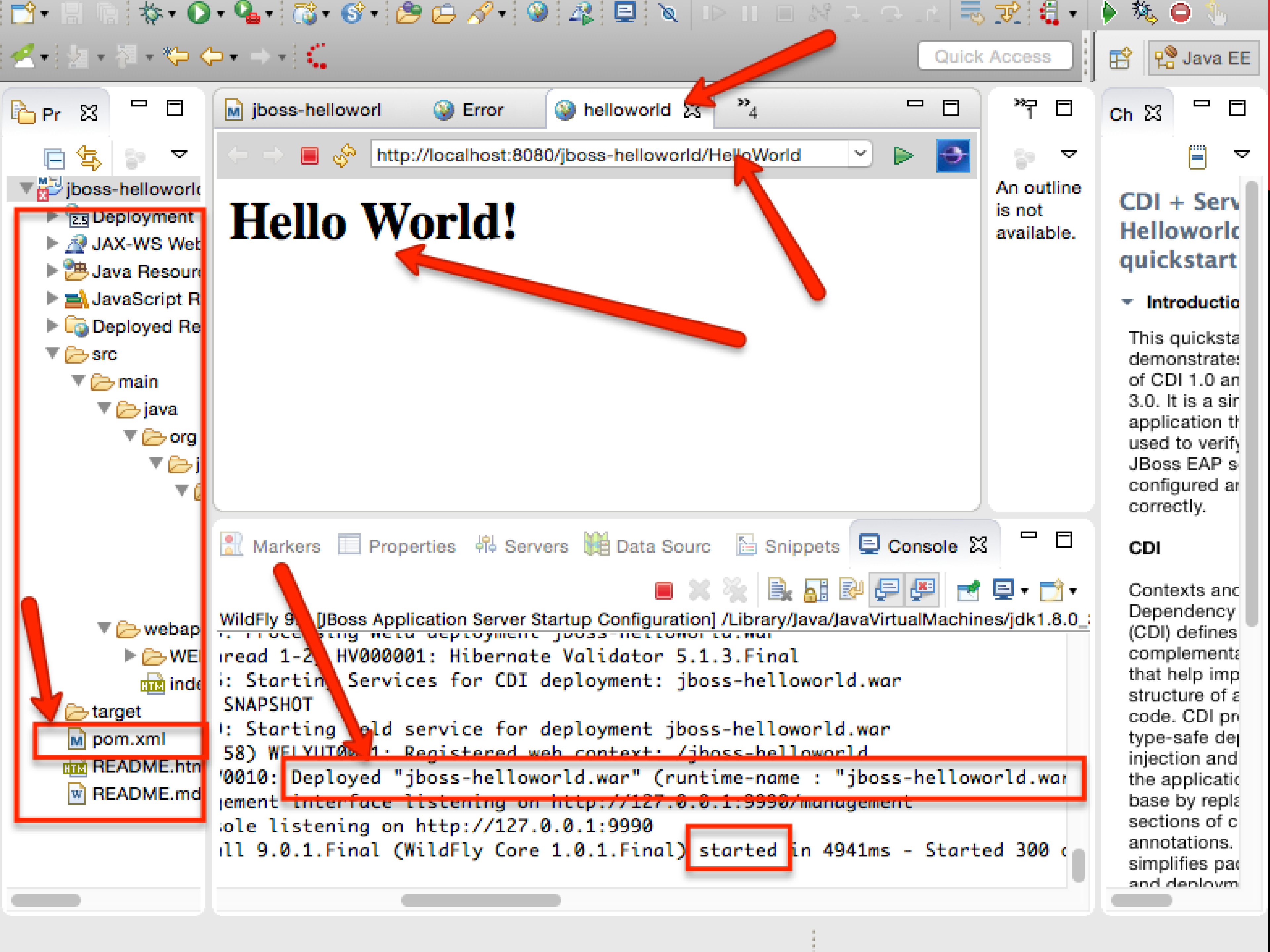
8. Fix class dependencies is any [In my case pom.xml change]

```
<version.jboss.spec.javaee.6.0>3.0.2.Final-redhat-15</version.jboss...6.0>
<version.jboss.spec.javaee.6.0>3.0.2.Final</version.jboss...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/HelloWorl
14. Servers | Right-click | Stop





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

Hello World!

An outline is not available.

```
WildFly 9 [JBoss Application Server Startup Configuration] /Library/Java/JavaVirtualMachines/jdk1.8.0_
: Processing weld deployment jboss-helloworld.war
read 1-2) HV000001: Hibernate Validator 5.1.3.Final
: Starting Services for CDI deployment: jboss-helloworld.war
 SNAPSHOT
: Starting weld service for deployment jboss-helloworld.war
58) WFLYUT0021: Registered web context: /jboss-helloworld
0010: Deployed "jboss-helloworld.war" (runtime-name : "jboss-helloworld.war
ement interface listening on http://127.0.0.1:9990/management
ole listening on http://127.0.0.1:9990
ll 9.0.1.Final (WildFly Core 1.0.1.Final) started in 4941ms - Started 300 o
```

CDI + Serv
Helloworld
quickstart

Introductio

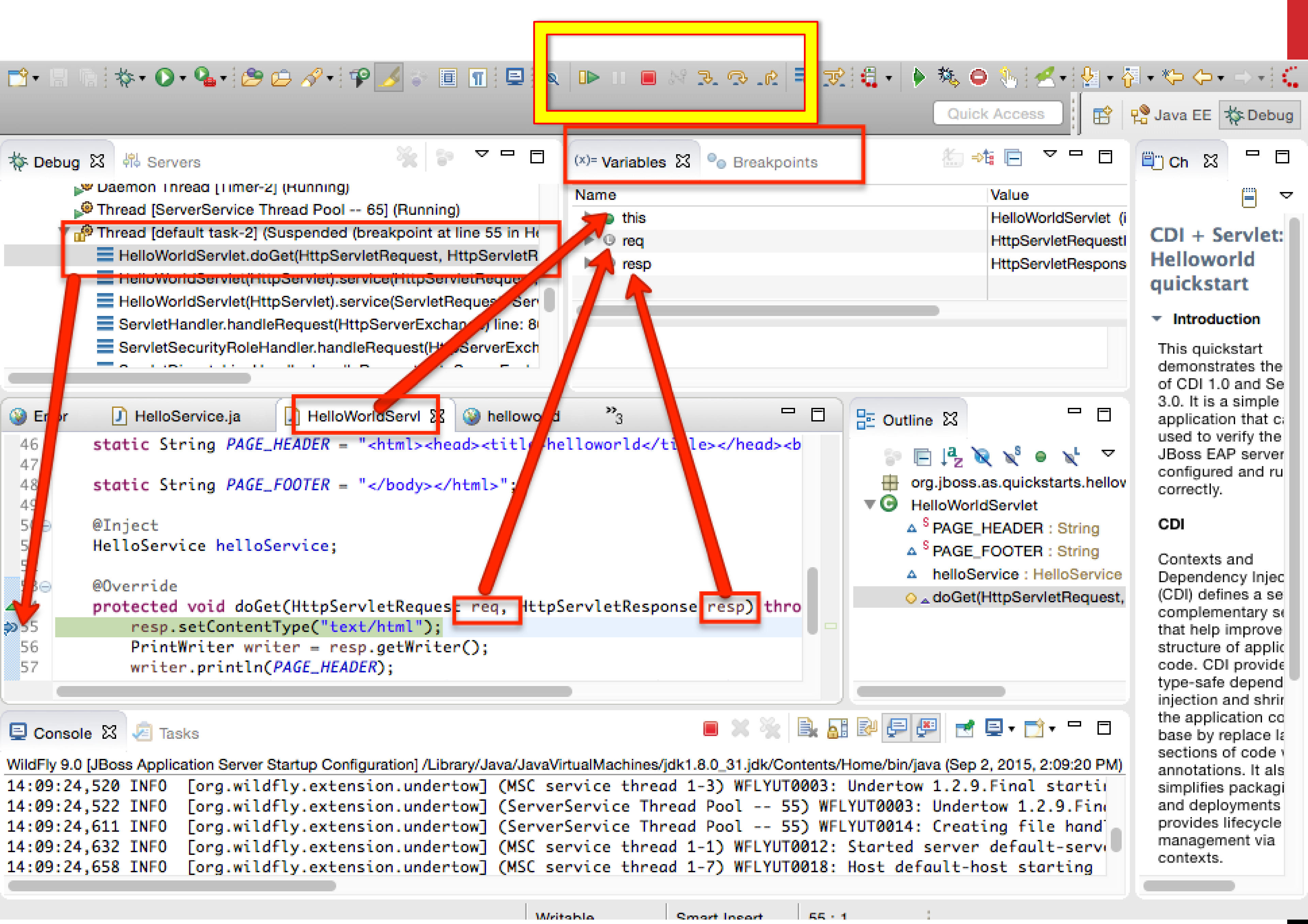
This quicksta
demonstrates
of CDI 1.0 an
3.0. It is a sir
application th
used to verify
JBoss EAP s
configured ar
correctly.

CDI

Contexts and
Dependency
(CDI) defines
complementa
that help imp
structure of a
code. CDI pr
type-safe dep
injection and
the applicatio
base by repla
sections of c
annotations.
simplifies pac
and deploym

JBOSS SAMPLE APPS DEBUG

14. Servers | Right-click | Stop
15. Servers | Right-click | **Debug**
16. Put debug break point (double click) to
    - `HelloService.java` Line 28
    - `HelloWolrdServlet` Line 55
17. Open web browser with address
    - http://localhost:8080/jboss-helloworld/HelloWorld
18. Switch back to eclipse and see Confirmation on Debug View | Yes



```
Daemon Thread [Timer-2] (Running)
Thread [ServerService Thread Pool -- 65] (Running)
Thread [default task-2] (Suspended (breakpoint at line 55 in He
    HelloWorldServlet.doGet(HttpServletRequest, HttpServletR
    HelloWorldServlet(HttpServlet).service(HttpServletReque
    HelloWorldServlet(HttpServlet).service(ServletReques  Ser
    ServletHandler.handleRequest(HttpServerExchan   ) line: 8
    ServletSecurityRoleHandler.handleRequest(H  ServerExch
```

Name	Value
this	HelloWorldServlet (i
req	HttpServletRequestI
resp	HttpServletRespons

```java
    static String PAGE_HEADER = "<html><head><titl  helloworld</ti  le></head><b

    static String PAGE_FOOTER = "</body></html>";

    @Inject
    HelloService helloService;

    @Override
    protected void doGet(HttpServletReques  req, HttpServletResponse resp) thro
        resp.setContentType("text/html");
        PrintWriter writer = resp.getWriter();
        writer.println(PAGE_HEADER);
```

```
org.jboss.as.quickstarts.hellov
HelloWorldServlet
    PAGE_HEADER : String
    PAGE_FOOTER : String
    helloService : HelloService
    doGet(HttpServletRequest,
```

CDI + Servlet: Helloworld quickstart

Introduction

This quickstart demonstrates the of CDI 1.0 and Se 3.0. It is a simple application that c used to verify the JBoss EAP server configured and ru correctly.

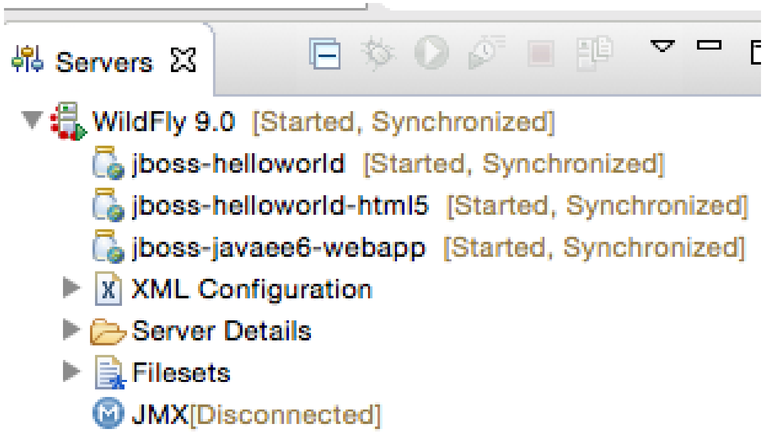
CDI

Contexts and Dependency Injec (CDI) defines a se complementary se that help improve structure of applic code. CDI provide type-safe depend injection and shrir the application co base by replace la sections of code annotations. It als simplifies packagi and deployments provides lifecycle management via contexts.

```
WildFly 9.0 [JBoss Application Server Startup Configuration] /Library/Java/JavaVirtualMachines/jdk1.8.0_31.jdk/Contents/Home/bin/java (Sep 2, 2015, 2:09:20 PM)
14:09:24,520 INFO  [org.wildfly.extension.undertow] (MSC service thread 1-3) WFLYUT0003: Undertow 1.2.9.Final startii
14:09:24,522 INFO  [org.wildfly.extension.undertow] (ServerService Thread Pool -- 55) WFLYUT0003: Undertow 1.2.9.Fin
14:09:24,611 INFO  [org.wildfly.extension.undertow] (ServerService Thread Pool -- 55) WFLYUT0014: Creating file hand
14:09:24,632 INFO  [org.wildfly.extension.undertow] (MSC service thread 1-1) WFLYUT0012: Started server default-serv
14:09:24,658 INFO  [org.wildfly.extension.undertow] (MSC service thread 1-7) WFLYUT0018: Host default-host starting
```

JBOSS SAMPLE APPS DEBUG

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



WildFly 9.0 [Started, Synchronized]
jboss-helloworld [Started, Synchronized]
jboss-helloworld-html5 [Started, Synchronized]
jboss-javaee6-webapp [Started, Synchronized]
XML Configuration
Server Details
Filesets
JMX[Disconnected]

JBOSS SAMPLE APPS

- Components
  - `HelloWorldServlet_= @WebServlet("/HelloWorld")`
    - Extends HttpServlet
    - Implementes 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

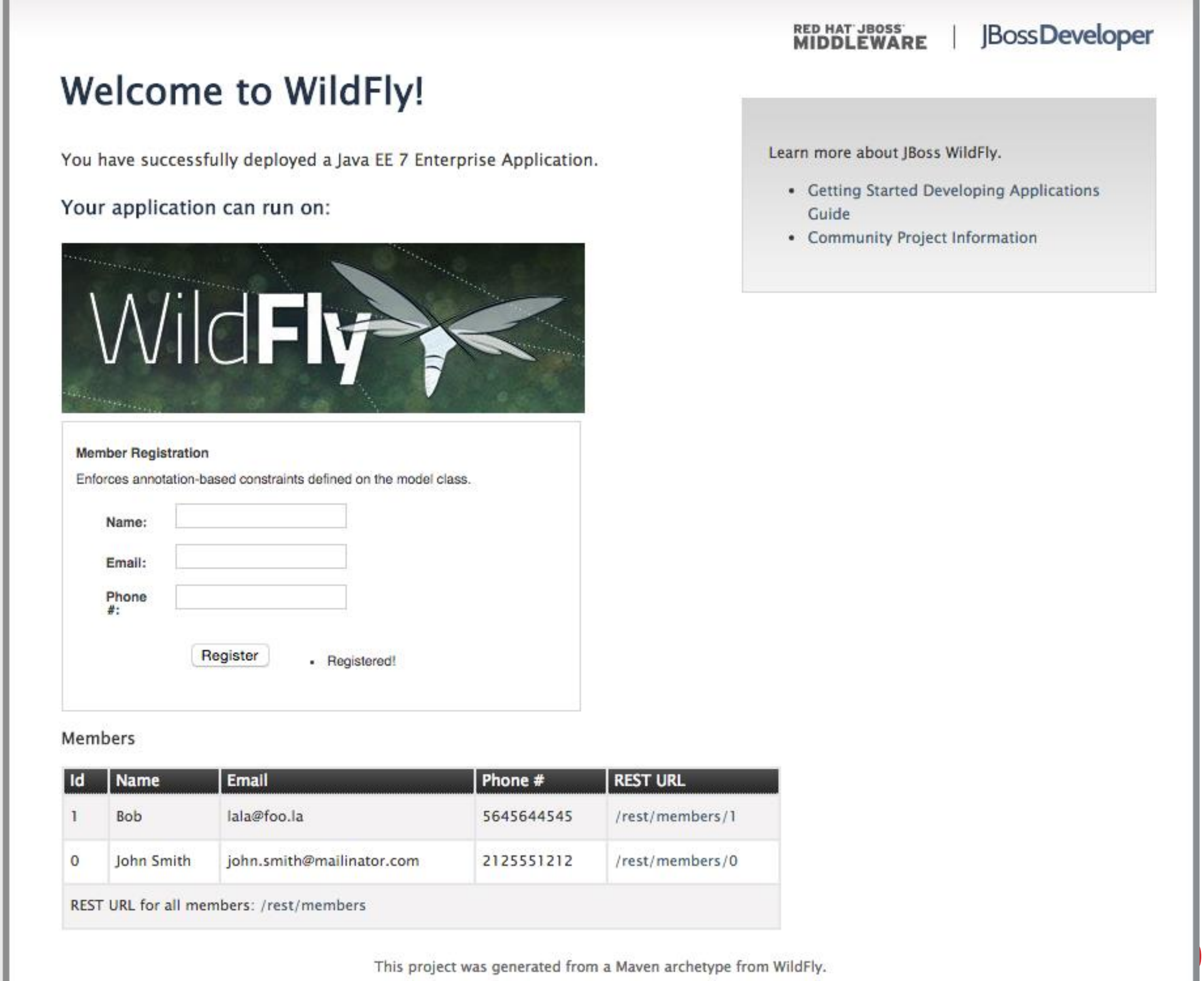
JAVA EE WEB PROJECT

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:

Learn more about JBoss WildFly.

- Getting Started Developing Applications Guide
- Community Project Information

Member Registration

Enforces annotation-based constraints defined on the model class.

Name:	
Email:	
Phone #:	

- Registered!

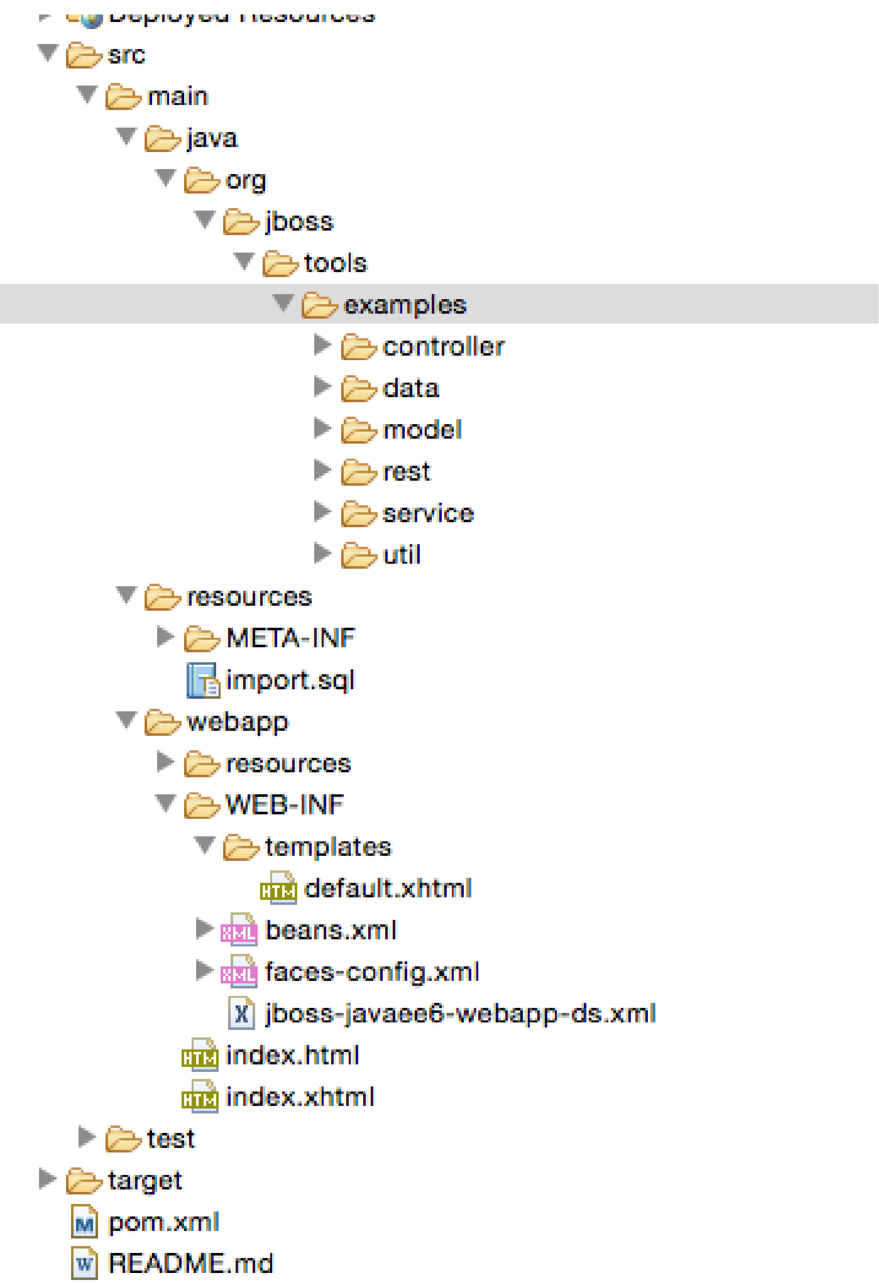
Members

Id	Name	Email	Phone #	REST URL
1	Bob	lala@foo.la	5645644545	/rest/members/1
0	John Smith	john.smith@mailinator.com	2125551212	/rest/members/0

REST URL for all members: /rest/members

This project was generated from a Maven archetype from WildFly.

SEE THE STRUCTURE



- Deployed Resources
- src
  - main
    - java
      - org
        - jboss
          - tools
            - examples
              - controller
              - data
              - model
              - rest
              - service
              - util
    - resources
      - META-INF
      - import.sql
    - webapp
      - resources
      - WEB-INF
        - templates
          - default.xhtml
        - beans.xml
        - faces-config.xml
        - jboss-javaee6-webapp-ds.xml
      - index.html
      - index.xhtml
  - test
- target
- pom.xml
- README.md

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

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**
   1. https://github.com/javaee-samples/javaee7-samples/blob/master/jpa/storedprocedure/src/main/java/org/javaee7/jpa/storedprocedure/Movie.java
   2. https://github.com/javaee-samples/javaee7-samples/blob/master/jpa/storedprocedure/src/main/java/org/javaee7/jpa/storedprocedure/MovieBean.java

INSTALL POSTGRES

Install portgres server

Make `postgres` user a password

Make a user `testuser` with `somepass`

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

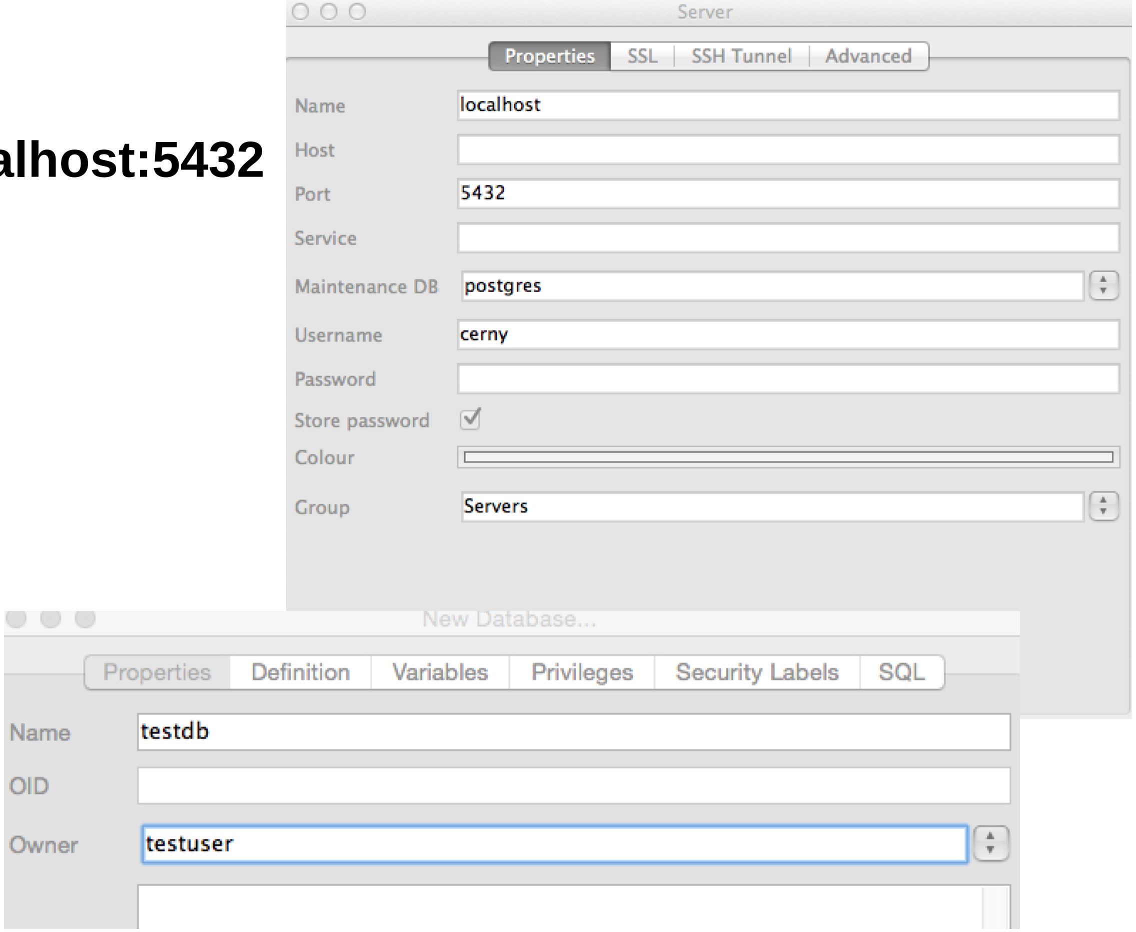
INSTALL POSTGRES PGADMIN

http://www.pgadmin.org/

Install and connect to localhost:5432

Use your user / password

Or postgress / password



Name: localhost
Host:
Port: 5432
Service:
Maintenance DB: postgres
Username: cerny
Password:
Store password: ☑
Colour:
Group: Servers

Name: testdb
OID:
Owner: testuser

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

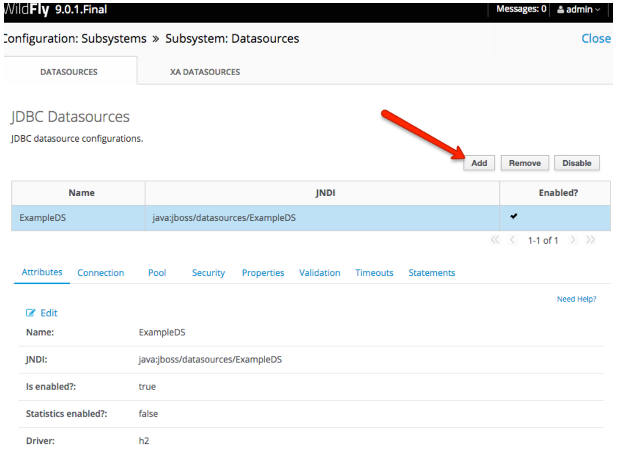
See the management instructions for console follow steps below - to add user

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

```
~/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
```

REGISTER POSTGRES TO SERVER I.

1. Go again to http://localhost:8080/console and login `admin admin`
2. Read through 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
6. Go to http://localhost:9990/console/App.html#profile/datasources
7. Click Add



JDBC Datasources

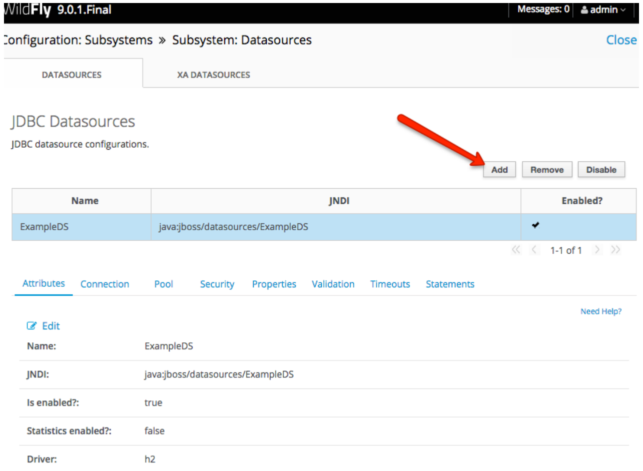
JDBC datasource configurations.

Name	JNDI	Enabled?
ExampleDS	java:jboss/datasources/ExampleDS	✓

Name: ExampleDS
JNDI: java:jboss/datasources/ExampleDS
Is enabled?: true
Statistics enabled?: false
Driver: h2

REGISTER POSTGRES TO SERVER II.

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



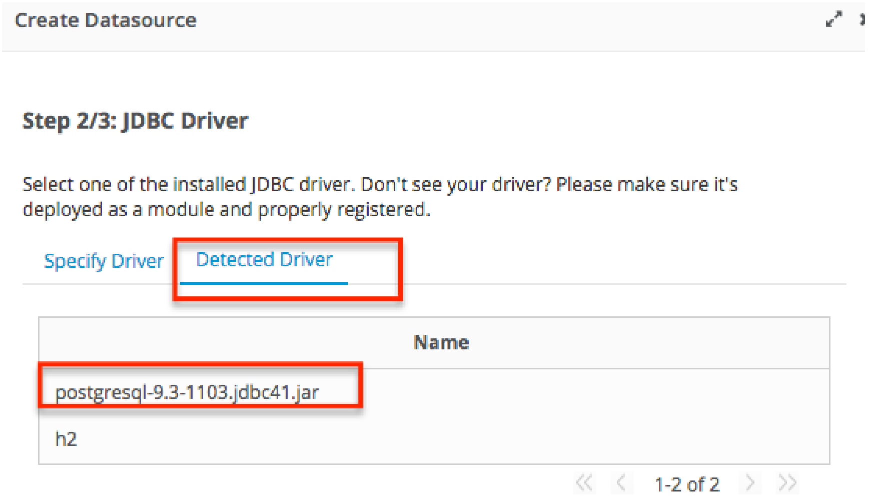
JDBC Datasources

JDBC datasource configurations.

Name	JNDI	Enabled?
ExampleDS	java:jboss/datasources/ExampleDS	✓

1-1 of 1

Name: ExampleDS
JNDI: java:jboss/datasources/ExampleDS
Is enabled?: true
Statistics enabled?: false
Driver: h2



Step 2/3: JDBC Driver

Select one of the installed JDBC driver. Don't see your driver? Please make sure it's deployed as a module and properly registered.

Name
postgresql-9.3-1103.jdbc41.jar
h2

1-2 of 2

8. Pick Postgresql-9.3.. | Setup connection

`URL: jdbc:postgresql://localhost:5432/testdb`

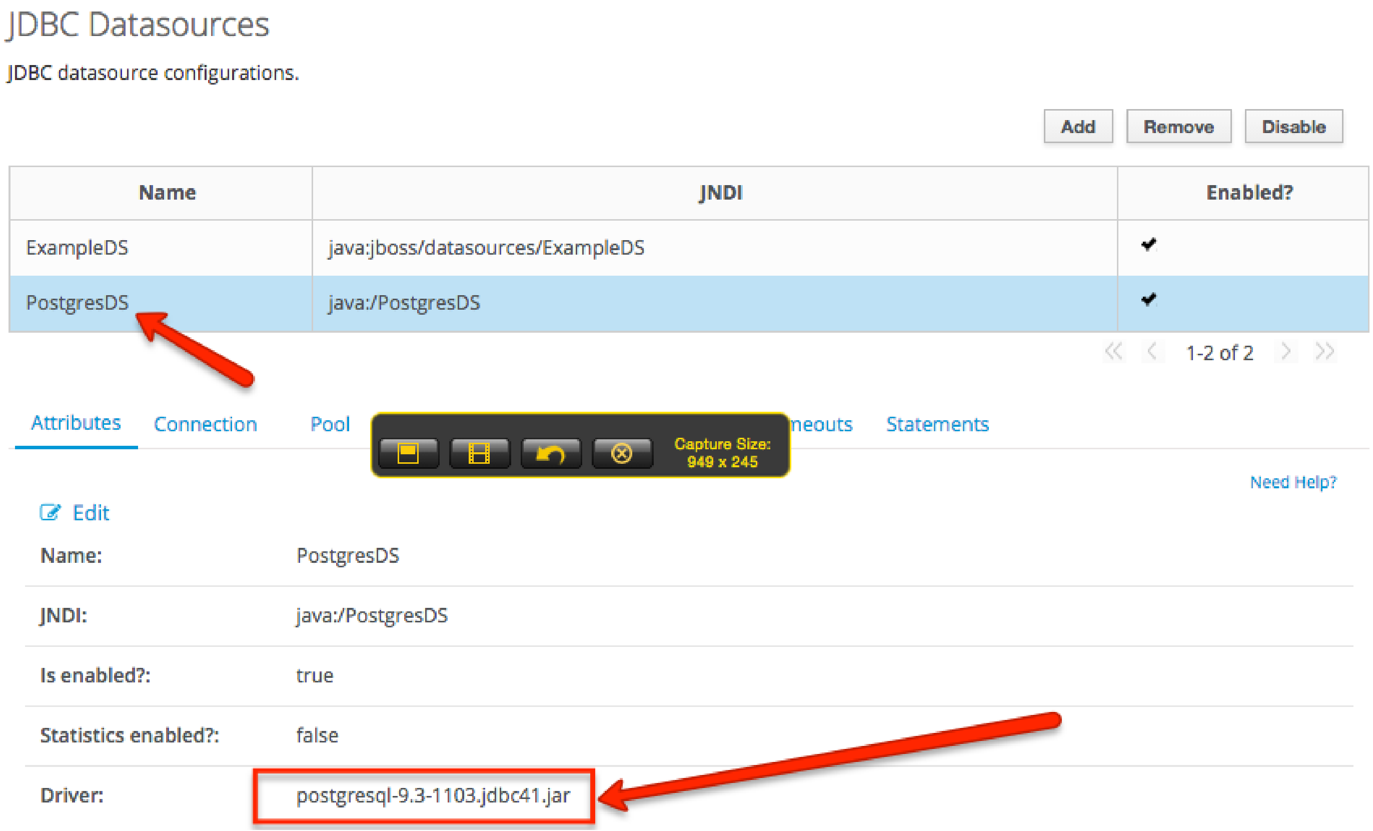
`Username: testuser`

`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



JDBC Datasources

JDBC datasource configurations.

Name	JNDI	Enabled?
ExampleDS	java:jboss/datasources/ExampleDS	✔
PostgresDS	java:/PostgresDS	✔

1-2 of 2

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

```
<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>
```

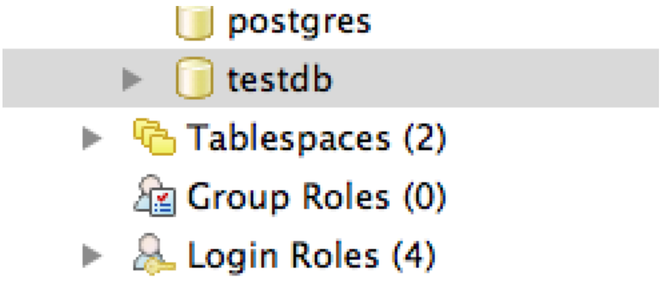
Add dialect to persistence.xml

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

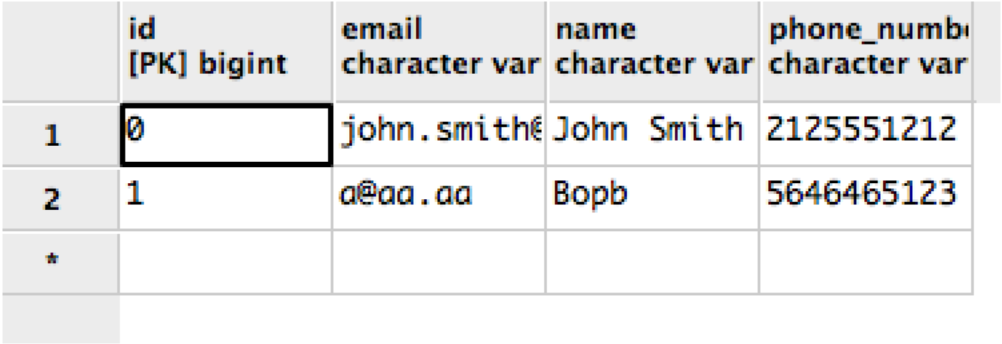
Go to http://localhost:8080/jboss-javaee6-webapp/index.jsf & add user

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**



postgres
testdb
Tablespaces (2)
Group Roles (0)
Login Roles (4)



	id [PK] bigint	email character var	name character var	phone_numb character var
1	0	john.smith	John Smith	2125551212
2	1	a@aa.aa	Bopb	5646465123
*				

IF YOU STOP SERVER DATA DISAPPEAR

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

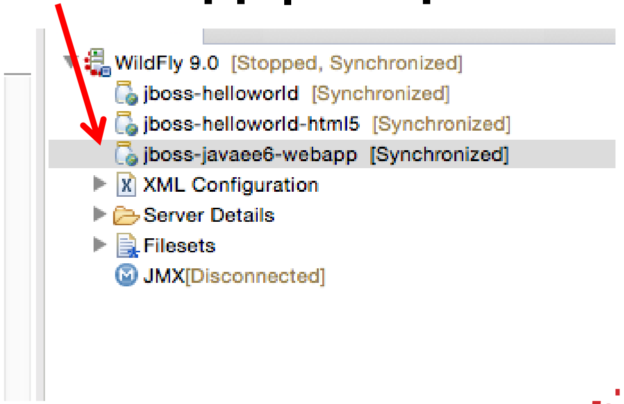
```
<property name="hibernate.hbm2ddl.auto" value="update" />

<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


WildFly 9.0 [Stopped, Synchronized]
jboss-helloworld [Synchronized]
jboss-helloworld-html5 [Synchronized]
jboss-javaee6-webapp [Synchronized]
XML Configuration
Server Details
Filesets
JMX[Disconnected]

NAMED QUERY HINT

1. Go to `Member.java`

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

2. Go to `MemberRepository.java` replace `findAllOrderedByName`

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