



UNIVERSIDADE FEDERAL  
DO ESPÍRITO SANTO

Centro Tecnológico  
Departamento de Informática

Prof. Vítor E. Silva Souza

<http://www.inf.ufes.br/~vitorsouza>

# Web Development in Java Part I



This material is licensed under the Creative Commons license Attribution-ShareAlike 4.0 International: <http://creativecommons.org/licenses/by-sa/4.0/>.

# Outline of part I

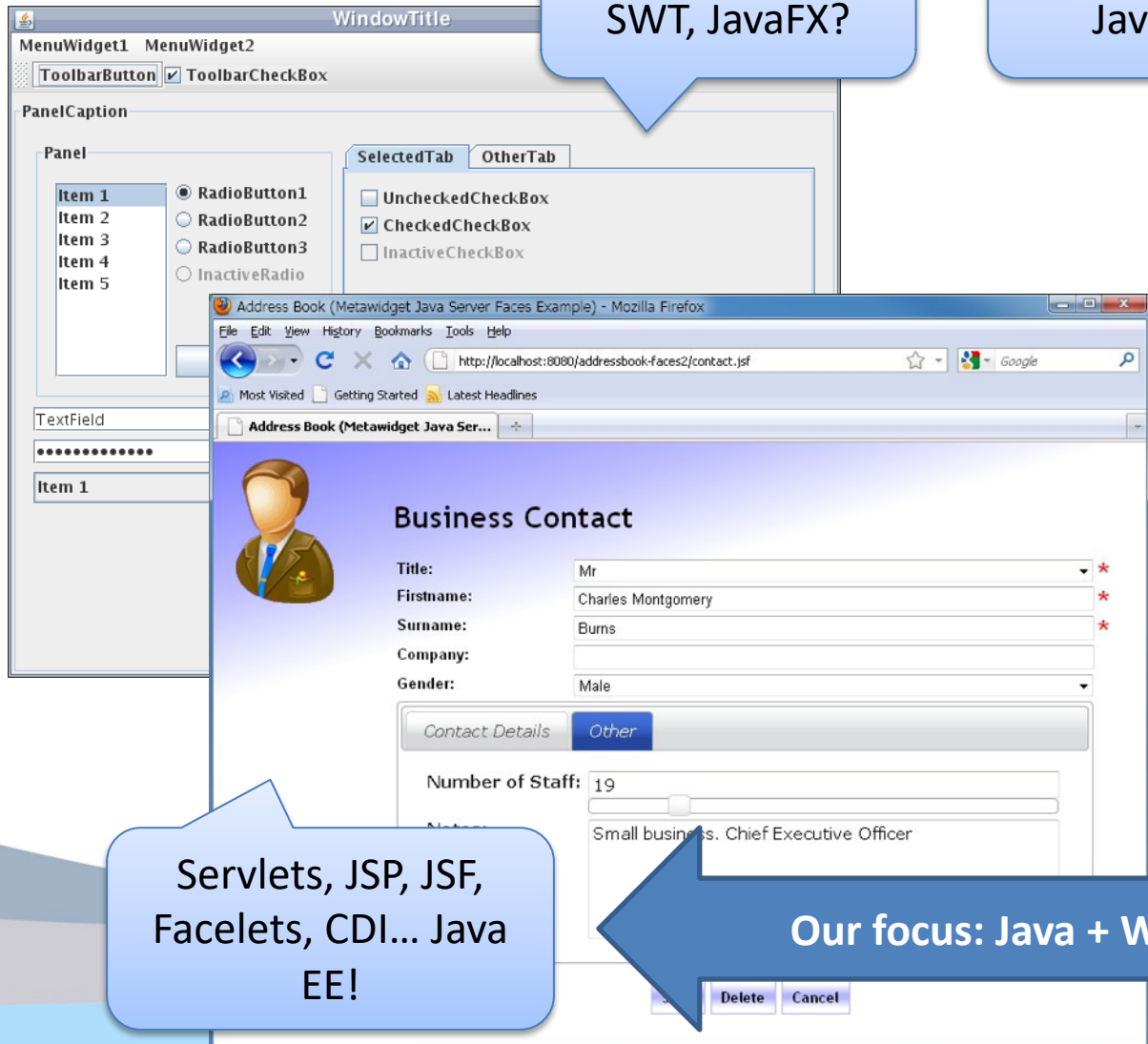
- Basic concepts: what are Web Applications?
- History & evolution of WebApps;
  - *Quick demonstrations;*
- Java EE overview;
- Tools for Java EE Web Profile development;
- Learning in practice – the Java Hostel demo:
  - *Decoration with Facelets;*
  - *Pages and controllers in JSF;*
  - *Wiring components with CDI;*
  - *Persistence with JPA.*

# So you've mastered Java...

## ■ What about...

AWT/Swing,  
SWT, JavaFX?

Android, Java ME,  
Java Card?



Servlets, JSP, JSF,  
Facelets, CDI... Java  
EE!

**Our focus: Java + Web**

# What is a Web Application?

- A distributed system;
- Access via a Web browser:
  - *HyperText Transfer Protocol (HTTP);*
  - *HTML, JavaScript, CSS, ...;*
  - *Popular and ubiquitous user client.*
- Similarity to “vanilla” (non-Web, regular) applications:
  - *Composed of GUI, business rules, domain data, persistence media, etc.;*
  - *Can range from a “Hello, world!” to very complex, millions-of-users, MLOCs systems.*

According to <http://www.informationisbeautiful.net/visualizations/million-lines-of-code/>, in 2015  
Facebook has ~ 62 MLOCs (including backend)

# It started with the (static) Web

- Born in 1989 as a more effective communication system for CERN (European Organization for Nuclear Research).

```
GET /index.html HTTP/1.0
Host: www.site.com
[...]
```



HTTP Request

HTTP Response



Looks for index.html file,  
merges with headers and  
sends back

```
HTTP/1.1 200 OK
Date: Fri, 15 Apr 2005 22:12:30 GMT
Server: Apache/1.3.26 [...]
Last-Modified: Wed, 23 Mar 2005 00:43:22 GMT
Content-Length: 11379
Content-Type: text/html
[...]
```



Tim Berners-Lee,  
creator of the WWW

# Soon enough pages were built dynamically

```
GET /index.cgi HTTP/1.0
Host: www.site.com
[...]
```



- 1: Runs a program associated with index.cgi;
- 2: Program returns text contents in HTML
- 3: Merges the result with HTTP headers and sends it back to client.

```
HTTP/1.1 200 OK
Date: Fri, 15 Apr 2005 22:12:30 GMT
Server: Apache/1.3.26 [...]
Last-Modified: Wed, 23 Mar 2005 00:43:22 GMT
Content-Length: 11379
Content-Type: text/html
[...]
```

# Web Development was born

- Web Development consists in writing programs that respond to requests using HTTP and produce results in a browser-compatible language (not just HTML!);
- Many components involved:
  - *The Web server;*
  - *The Web pages, style sheets, scripts, images, etc.;*
  - *Code in a programming language;*
  - *Existing APIs, frameworks, libraries.*



# Evolution of Web technologies

- 1993: CGI – Common Gateway Interface – C, C++, Fortran, Perl, etc.;
- 1994: Macromedia Coldfusion, PHP;
- 1995: Microsoft ASP;
- 1996: Java Servlets;
- 1999: JavaServer Pages (JSP);
- ...
- Today: Rich Internet Applications
  - *AJAX, HTML5, REST, responsive, single-page, etc.*



# A brief word on AJAX

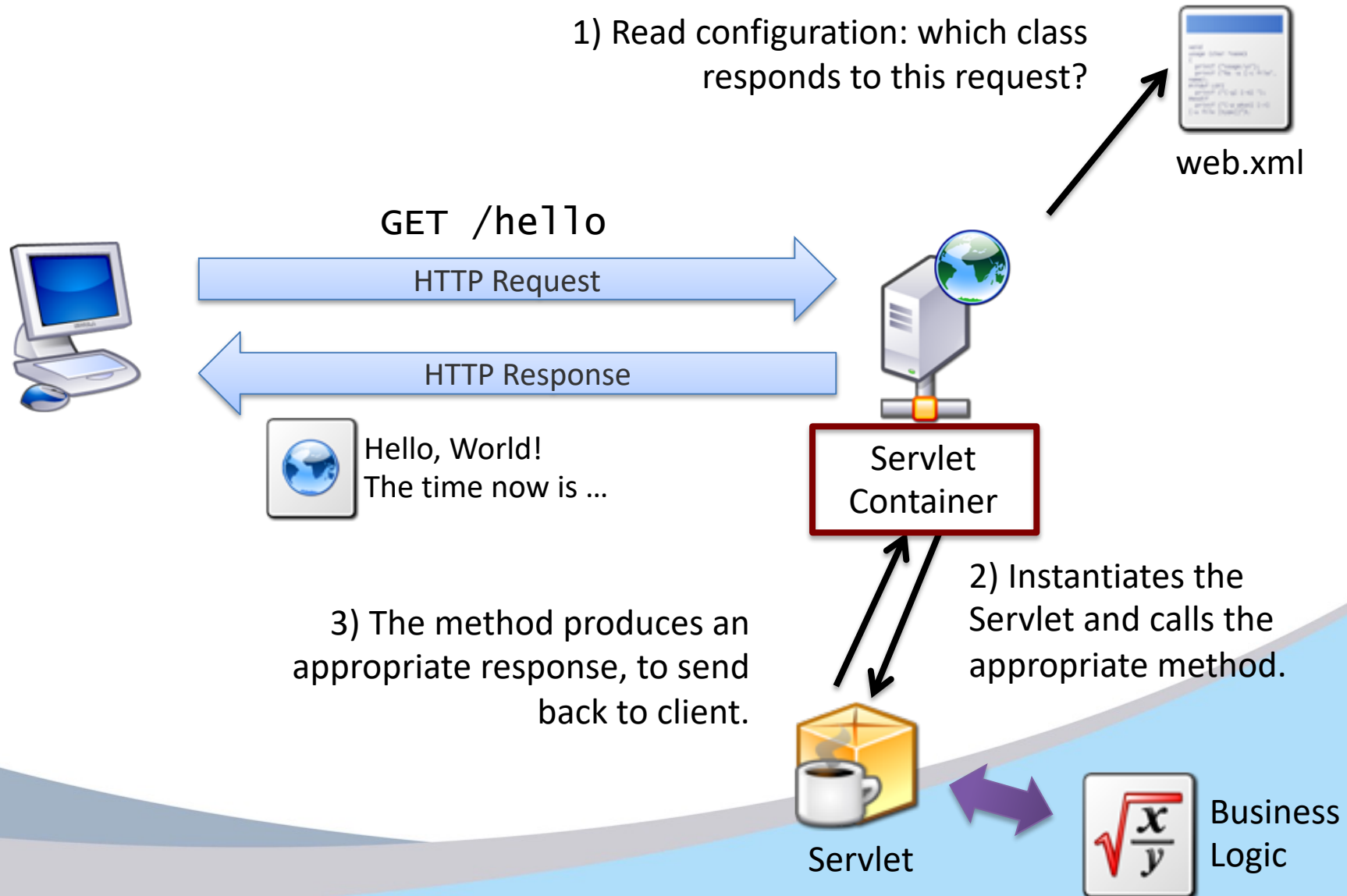
- AJAX = Asynchronous JavaScript and XML;
- Still request-response, that hasn't changed!
- JavaScript makes a smaller request, using an object called XMLHttpRequest;
- Small XML chunks are exchanged between browser and server, with partial data;
- The webpage's Document Object Model (DOM) is manipulated;
- Only parts of the page are changed as result.



# Evolution of Java Web technologies

- 1995: official launch, in the beginning was the Applet;
- 1996: Java Servlets;
- 1999: JavaServer Pages (JSP);
- 1999: J2EE platform (1.3 in 2001, 1.4 in 2003);
- 2000: Apache Struts, pioneer MVC framework;
- 2004: JavaServer Faces;
- 2006: Java EE 5;
- 2009: Java EE 6;
- 2013: Java EE 7;
- 2017: Java EE 8.

# Java Servlets





APACHE  
GERONIMO



WildFly

jetty://

resin®



WebSphere®



# Container???

- A container is a software that manages components with a very specific lifecycle;
- A Servlet (one of many such components):
  - *Is created when its URL is requested;*
  - *Executes a method that corresponds to the request;*
  - *Is collected by the GC after the response.*
- Combined with a software that “talks” in HTTP through a port (80, 8080, etc.), you get a Java Web Server.



# Back to Servlets...

- Not necessarily for the Web;
- The Web Servlet, the most used one, implements `javax.servlet.http.HttpServlet`:
  - *doGet(), doPost(), etc.;*
  - *init(), destroy();*
  - *HttpServletRequest/Response;*
  - *response: setContentType(), getWriter().*

# Java Servlets



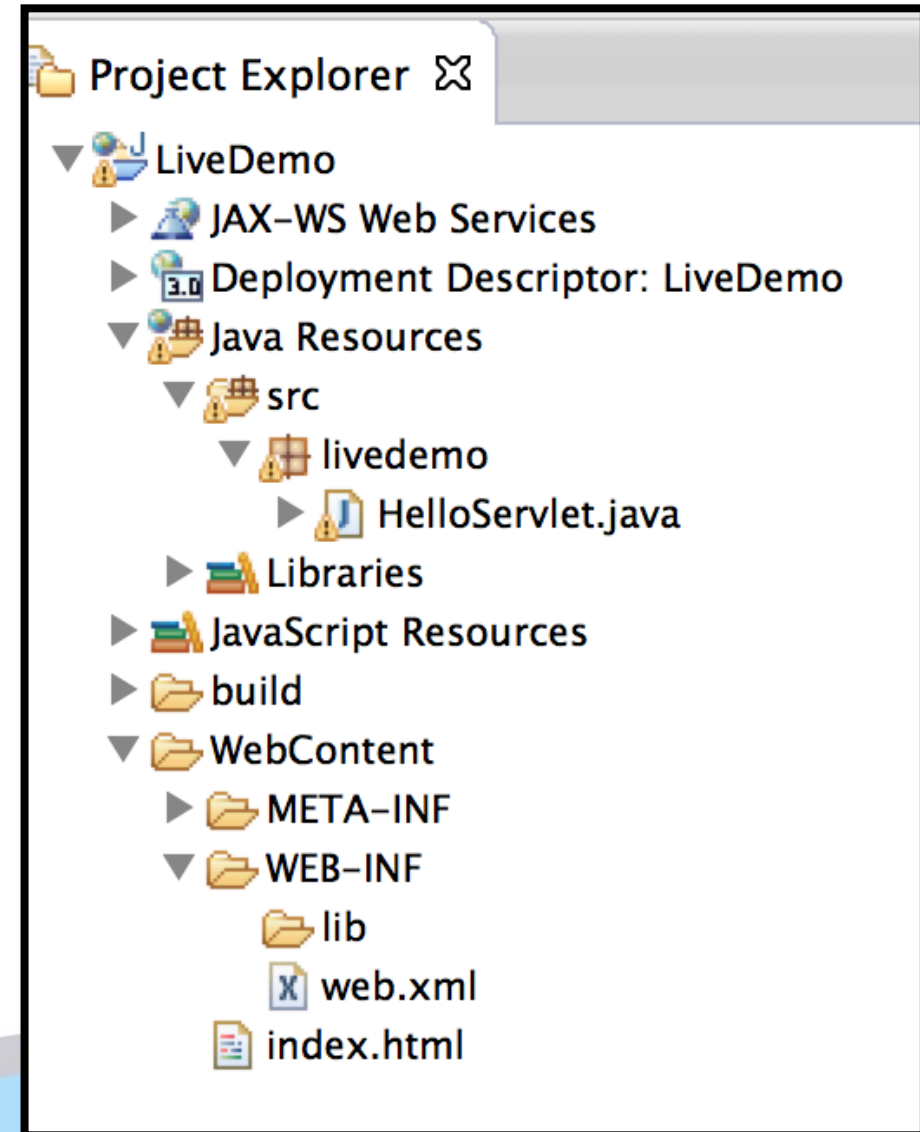
Live Demo





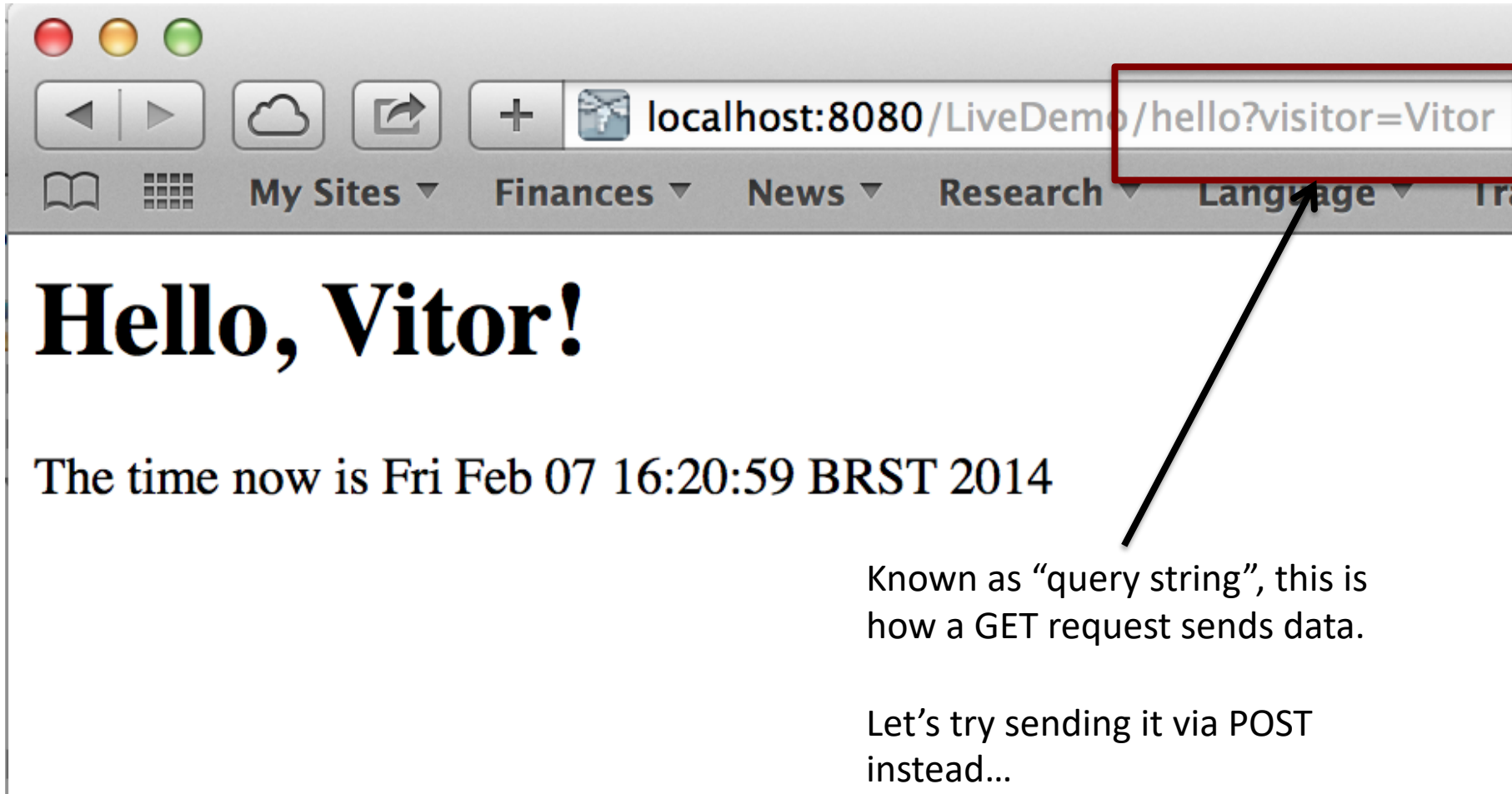
# Completing the Web Application

- A default *index.html* webpage;
- A form in the page sends data to the Servlet: name of the visitor;
- The Servlet now says “Hello, <visitor’s name>” instead.





# GET and POST requests



localhost:8080/LiveDemo/hello?visitor=Vitor

## Hello, Vitor!

The time now is Fri Feb 07 16:20:59 BRST 2014

Known as “query string”, this is how a GET request sends data.

Let's try sending it via POST instead...



# Fast forward to Servlets 3.1...

- Annotations for URL mapping;
- Asynchronous requests;
- Non-blocking I/O;
- HTTP upgrade protocol;
- Security upgrades against session fixation attacks;
- Etc.



# Servlets are annoying

- Writing an entire page with a PrintWriter:

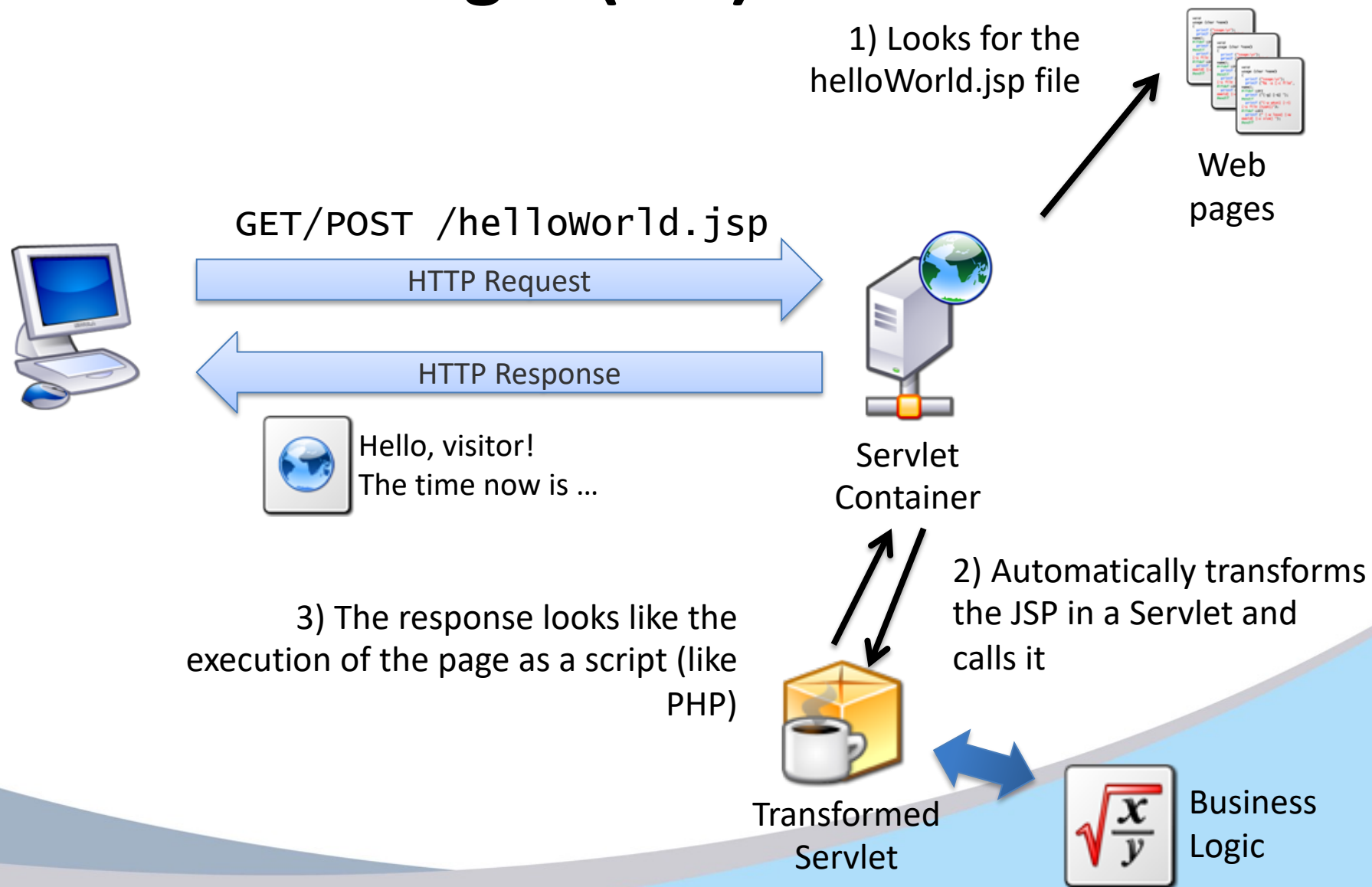
```
try (PrintWriter out = resp.getWriter()) {
    out.write("<html><head><title>Hello!</title></head><body>");
    out.write("<h1>Hello, " + req.getParameter("visitor") + "!</h1>");
    out.write("<p>The time now is " + new Date() + "</p>");
    out.write("</body></html>");
}
```

- Map Servlets to URLs:

```
@WebServlet(name = "HelloServlet", urlPatterns = {"/hello"})
```

(Also, remember that initially Servlets had to be mapped in web.xml!)

# JavaServer Pages (JSP)



# JSP Standard Tag Library

- Tag Libraries provide tags that encapsulate functionality common to many JSP applications;
- JSTL defines a standard set of tags that all Java Web Servers must implement (standardization effort):
  - *Core: variable support, flow control, URL management, miscellaneous;*
  - *XML: XML transformation and other functions;*
  - *I18N: locale setting, message formatting, number/date formatting;*
  - *SQL: database operations;*
  - *Functions: collection length, string manipulation.*

# JSP Standard Tag Library

## ■ Examples:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<c:forEach var="item" items="${sessionScope.cart.items}">
    ...
</c:forEach>

<c:set var="bookId" value="${param.Remove}"/>
<jsp:useBean id="bookId" type="java.lang.String" />
<% cart.remove(bookId); %>
<sql:query var="books" dataSource="${applicationScope.bookDS}">
    select * from PUBLIC.books where id = ?
    <sql:param value="${bookId}" />
</sql:query>

<c:url var="url" value="/catalog" >
    <c:param name="Add" value="${bookId}" />
</c:url>
<p><strong><a href="${url}">
```

# Something wrong with this model...

- Not the ideal place for business logic:

```
<html>
[...]
```

```
<%
Connection conn;
PreparedStatement stmt;
conn = Database.connect();
stmt = conn.prepareStatement("SQL");
ResultSet rs = stmt.executeQuery();

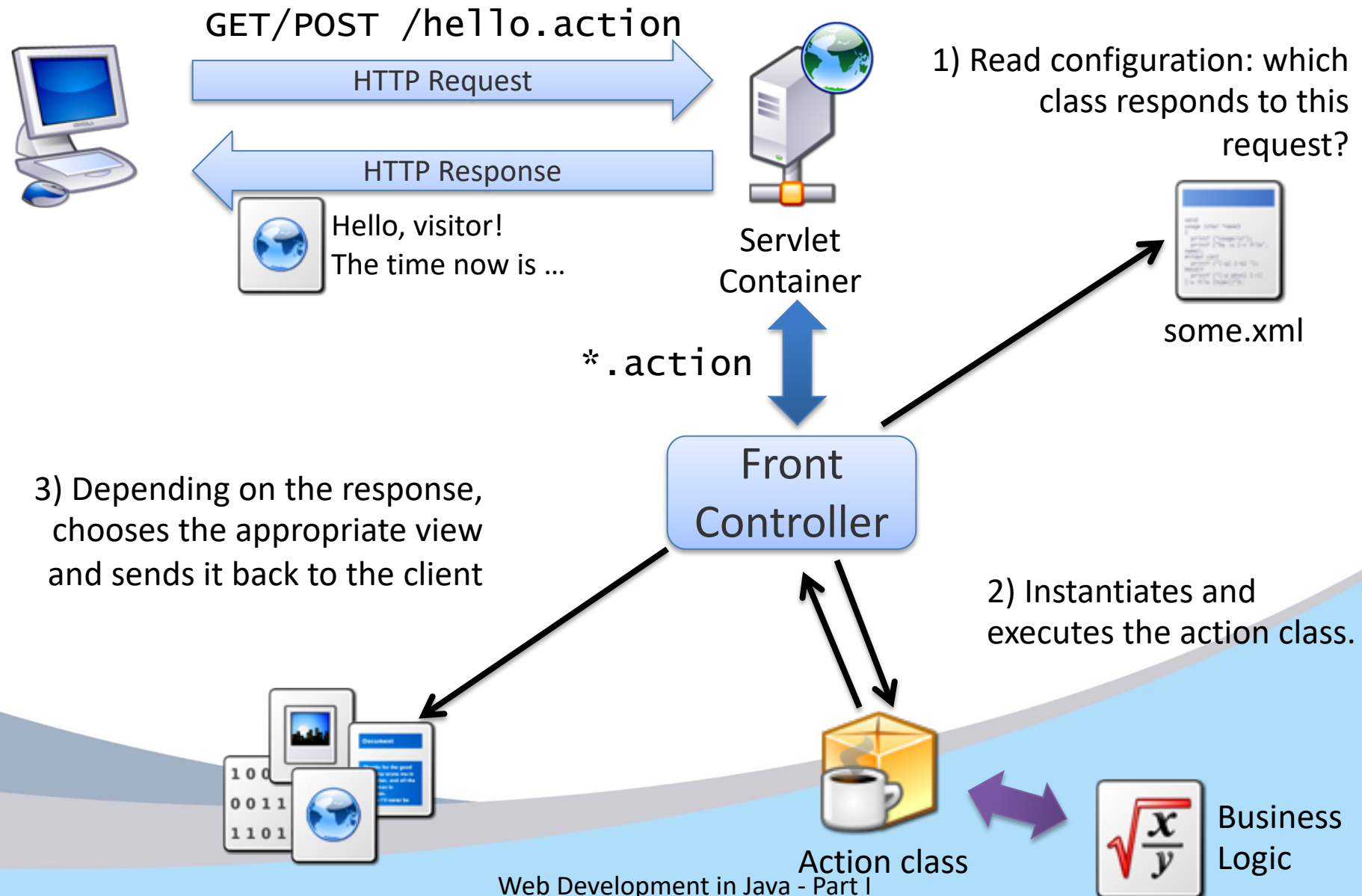
// Business logic here...

stmt = conn.prepareStatement("SQL");
stmt.executeUpdate();
%>
```

```
[...]
</html>
```



# Model 2 or MVC



# Model-View-Controller

- Born in the 1970s for Smalltalk in the Xerox PARC;
- Originally for GUIs (Swing also based on it);
- But also well-suited for the Web:
  - *Web pages and other Web resources are the **view**: focus on presentation;*
  - *Your business logic is the **model**: information and functionality, independent of presentation;*
  - *The framework and your action classes are the **controller**: mediator between model and view.*



apache  
**tapestry 5**  
*Code less, deliver more.*



# Example in Struts<sup>2</sup>

```
<s:form namespace="/examples" action="calculateAge" method="post">
  <s:textfield label="Name" name="name" />
  <s:textfield label="Birth date" name="birthDate" />
  <s:submit value="Calculate Age" />
</s:form>
```

From the Struts<sup>2</sup>  
tag library, by the  
way...

```
public class CalculateAgeAction extends ActionSupport {
    private String name;
    private Date birthDate;
    private int age;          // + getters and setters

    public String execute() throws Exception {
        age = calculateDifference(birthDate, new Date());
        return SUCCESS;
    }
}
```

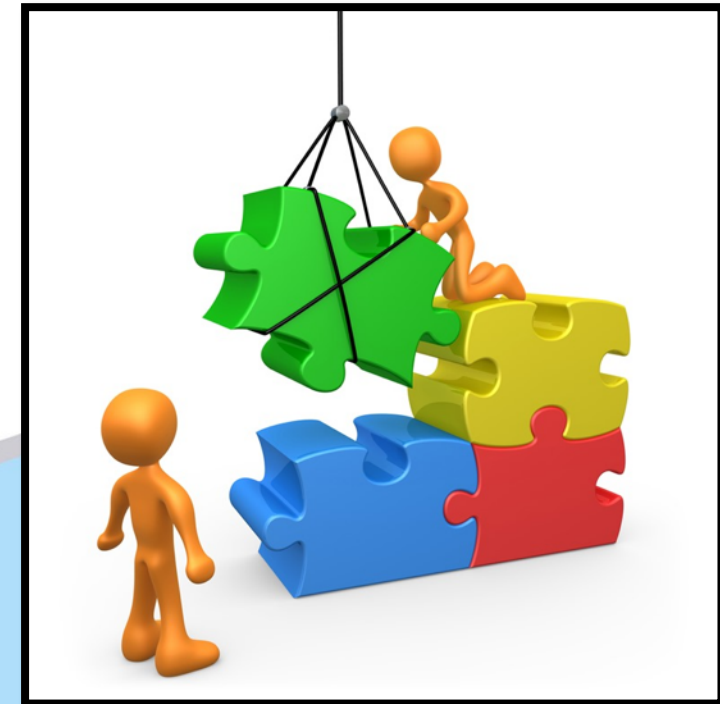
```
<p>Hello ${name}, you are ${age} years old.</p>
```

This, instead, is FreeMarker...



# Further room for improvement...

- Low productivity due to lack of reusable components;
- Vast range of frameworks, no standard;
- Lack of good IDE support (related to previous issues);
- Separation of roles could be further improved.

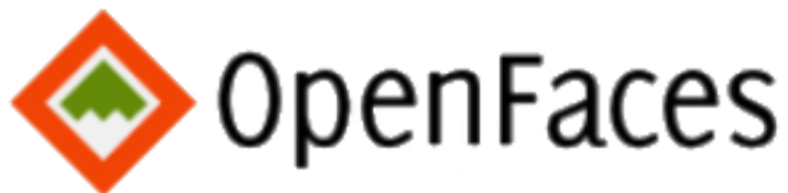


# JavaServer Faces

- Standard specification:
  - *1.0, 1.1 in 2004, 1.2 in 2006 (Java EE 5), 2.0 in 2008 (Java EE 6), 2.1 in 2010, 2.2 in 2013 (Java EE 7);*
  - *Currently in version 2.3, part of Java EE 8;*
- Builds on the idea of reusable components;
- Similar to an MVC framework:
  - *Component-based instead of action-based;*
  - *Represents UI components and manages their state;*
  - *Event handling;*
  - *Navigation control;*
  - *Component tag library.*

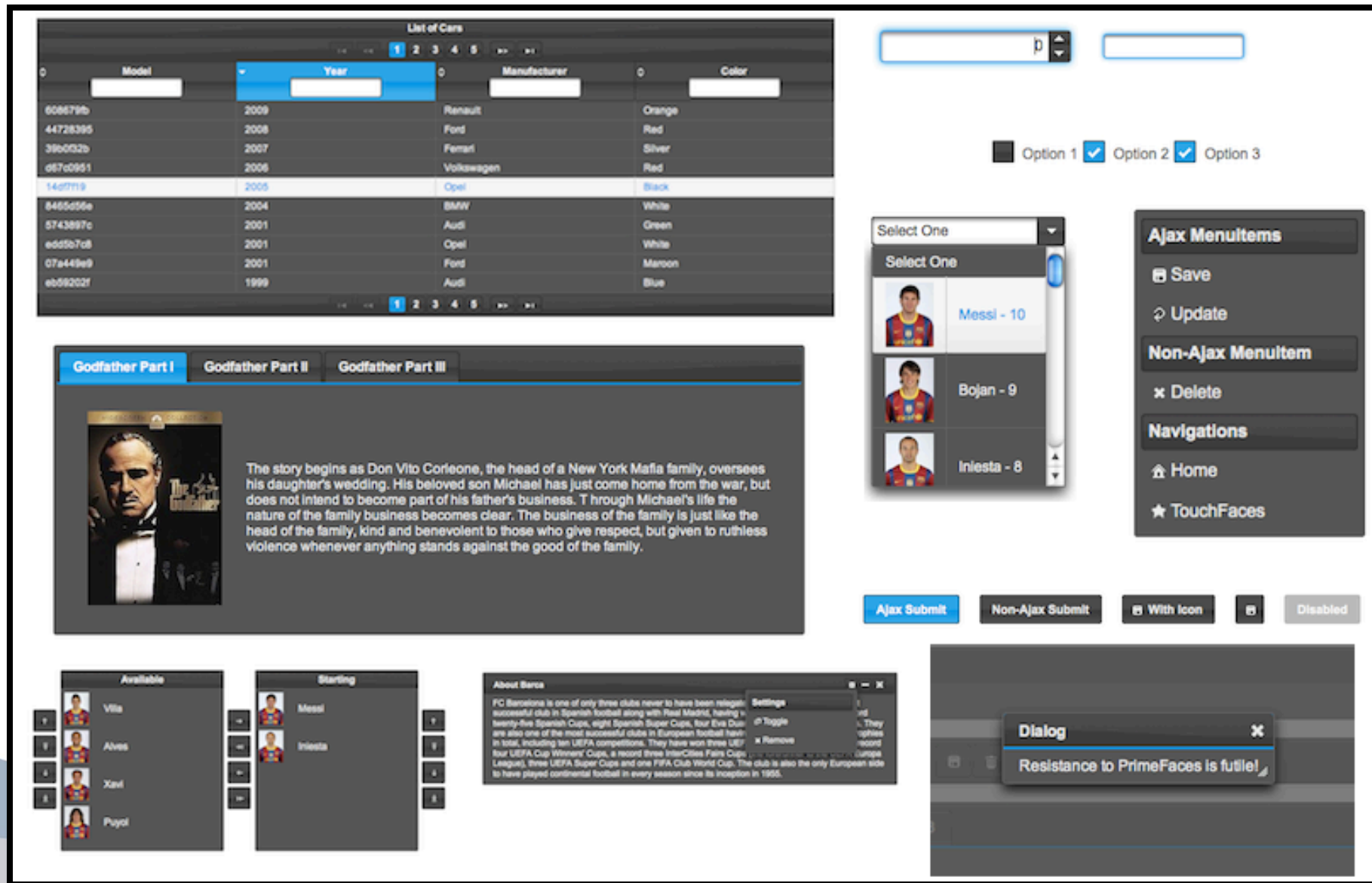


# JSF component libraries





# JSF component libraries



**List of Cars**

Model	Year	Manufacturer	Color
608679b	2009	Renault	Orange
44728395	2008	Ford	Red
39c0032b	2007	Ferrari	Silver
d67c09d1	2006	Volkswagen	Red
14d77119	2005	Opel	Black
8465c05e	2004	BMW	White
5743897c	2001	Audi	Green
edd5b7c8	2001	Opel	White
07a449e9	2001	Ford	Maroon
eb59202f	1999	Audi	Blue

**Godfather Part I** | Godfather Part II | Godfather Part III

The story begins as Don Vito Corleone, the head of a New York Mafia family, oversees his daughter's wedding. His beloved son Michael has just come home from the war, but does not intend to become part of his father's business. Through Michael's life the nature of the family business becomes clear. The business of the family is just like the head of the family, kind and benevolent to those who give respect, but given to ruthless violence whenever anything stands against the good of the family.

**Available**

- Villa
- Alves
- Xavi
- Puyol

**Starting**

- Messi
- Iniesta

**About Barca**

FC Barcelona is one of only three clubs never to have been relegated, successful club in Spanish football along with Real Madrid, having in twenty-five Spanish Cups, eight Spanish Super Cups, four Eva Duarte and also one of the most successful clubs in European football having in total, including ten UEFA competitions. They have won three UEFA four UEFA Cup Winners' Cups, a record three InterCities Fairs Cups (League), three UEFA Super Cups and one FIFA Club World Cup. The club is also the only European side to have played continental football in every season since its inception in 1995.

**Dialog**

Resistance to PrimeFaces is futile!



# JavaServer Faces



Live Demo



# Java Editions

## ■ Standard Editions:

- *Java 1.0 (1996);*
- *Java 1.1 (1997);*
- *J2SE 1.2 (1998);*
- *J2SE 1.3 (2000);*
- *J2SE 1.4 (2002);*
- *Java 1.5 / Java 5 (2004);*
- *Java SE 6 (2006);*
- *Java SE 7 (2011);*
- *Java SE 8 (2014);*
- *Java SE 9 (2017);*
- *Java SE 10 (2018.03);*
- *Java SE 11 (2018.09);*
- *Java SE 12 (2019.03).*

## ■ Enterprise Editions:

- *JPE project (1998);*
- *J2EE 1.2 (1999);*
- *J2EE 1.3 (2001);*
- *J2EE 1.4 (2003);*
- *Java EE 5 (2006);*
- *Java EE 6 (2009);*
- *Java EE 7 (2013);*
- *Java EE 8 (2017).*

# Java for the enterprise

- Sun Microsystems started with extensions for enterprise Java customers. Examples:
  - *JNDI: Java Naming and Directory Interface;*
  - *JTS: Java Transactions Service;*
- Loose extensions were confusing at first, so eventually Sun joined them together in a platform: J2EE;
- Sun provided the spec, a reference implementation and a compatibility test suite to certify application servers;
- [Application Programming | Service Provider] Interface:
  - *Programmers write code to the API;*
  - *Vendors implement the API via SPIs.*



# Remember containers?

- Java EE application servers are also containers to:
  - *Servlets;*
  - *Enterprise Java Beans;*
  - *Transactions;*
  - *Etc.*

J2EE 1.2	
JDBC Standard Extension API	2.0
Java Naming and Directory Interface Specification (JNDI)	1.2
RMI-IIOP	1.0
Java Servlets	2.2
JavaServer Pages (JSP)	1.1
Enterprise JavaBeans (EJB)	1.1
Java Message Service API (JMS)	1.0
Java Transaction API (JTA)	1.0
JavaMail API	1.1
JavaBeans Activation Framework (JAF)	1.0



# Enterprise Java Beans

- At the core of J2EE:
  - *Session EJBs provided functionality;*
  - *Entity EJBs represented domain concepts and provided persistence;*
  - *Message-driven EJBs respond to events.*
- Unfortunately, it suffered a lot of problems:
  - *Very complex API, drifted away from OO;*
  - *Persistence of Entity EJBs was terrible;*
  - *Poor performance.*
- Got eventually replaced by “lightweight” frameworks, such as Hibernate and Spring Framework.

Trivia:  
EJB-OSS  
↓  
JBoss  
↓  
JBoss

# De facto vs. de jure

- Popularity of frameworks led to a major change of standards;
- Beginning in Java EE 5, many of its technologies were “inspired” by frameworks:
  - *JPA is just like Hibernate (Gavin King led the spec);*
  - *CDI uses dependency injection, like Spring;*
  - *Facelets provides decoration like Sitemesh/Tiles;*
  - *JSF has MVC-like behavior.*

If you can't  
win...

# Java EE 6: pruning and Web Profile

Java EE 6 Web Profile	
<b>Bean Validation</b>	<b>Common Annotations for the Java Platform</b>
<b>CDI (Contexts and Dependency Injection) for the Java EE Platform</b>	<b>EJB (Enterprise Java Beans) Lite</b>
<b>EL (Expression Language)</b>	Entity Beans
JACC (Java Authorization Service Provider Contract for Containers)	<b>Interceptors</b>
Java EE Deployment API	JASPIC (Java Authentication Service Provider Interface for Containers)
JavaMail	Java EE Management API
JAX-RS (Java API for RESTful Web Services)	JAX-RPC (Java API for XML-based RPC)
JAXR (Java API for XML Registries)	JAXB (Java Architecture for XML Binding)
JMS (Java Messaging Service)	JCA (Java EE Connector Architecture)
<b>JSF (JavaServer Faces)</b>	<b>JPA (Java Persistence API)</b>
<b>JSTL (Standard Tag Library for JavaServer Pages)</b>	<b>JSP (JavaServer Pages)</b>
<b>Managed Beans</b>	<b>JTA (Java Transaction API)</b>
Web Services Metadata for the Java Platform	<b>Servlet</b>



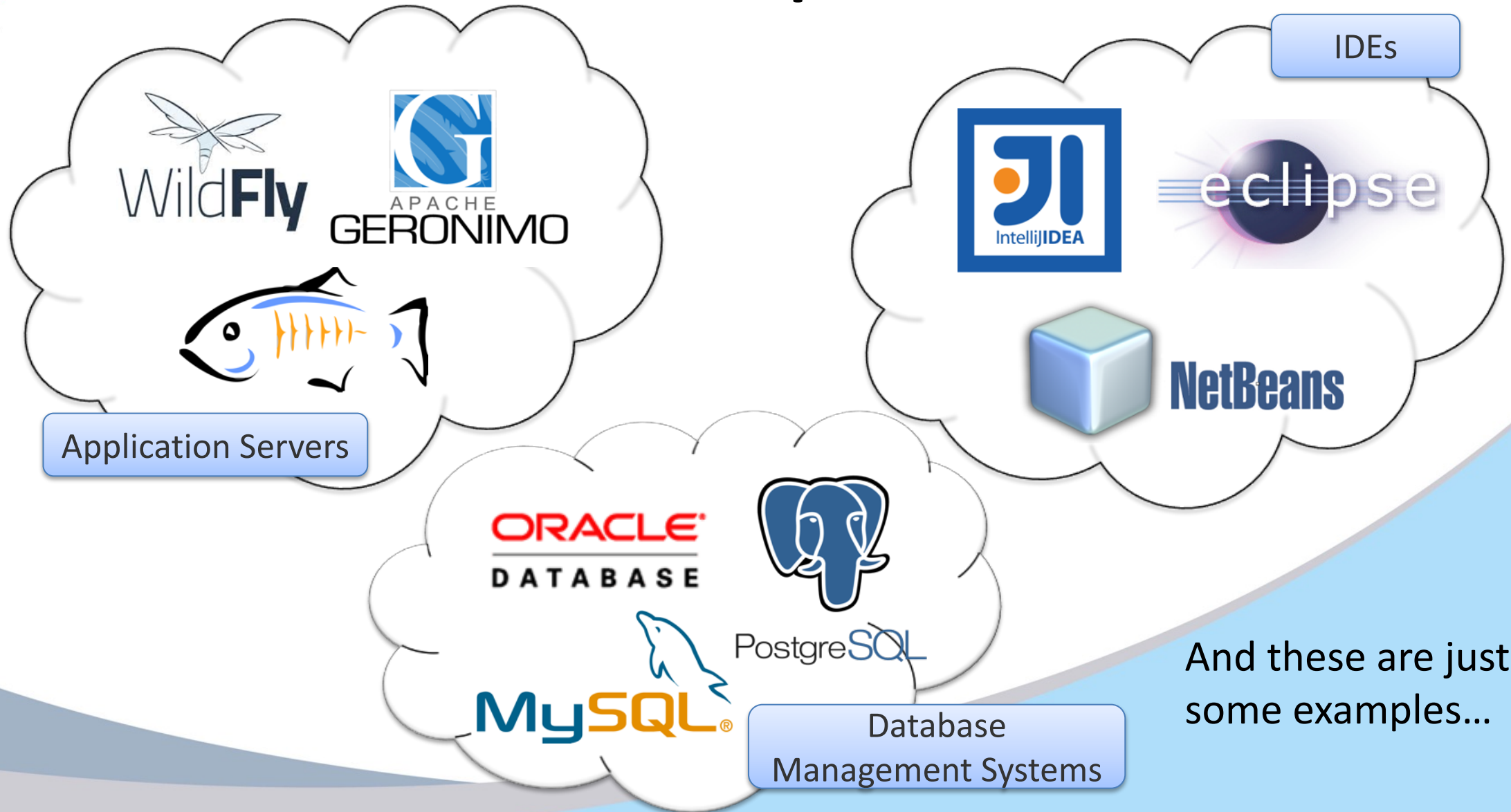


# The JavaHostel Tutorial

# Learn by example

- Overview of some of the main Java EE (Web Profile) technologies: JSF, Facelets, CDI, JPA;
- Development of a simple but real application: the website for a hostel;
- Detailed instructions (a bit outdated) in my blog:
  - <http://www.inf.ufes.br/~vitorsouza/blog/developing-a-java-ee-6-web-profile-application-from-scratch>
- Other resources:
  - <https://github.com/feees/Sigme/wiki/Como-obter-e-executar-o-Sigme> (in Portuguese);
  - <https://github.com/dwws-ufes/jbutler/wiki>.

# Tools for Java EE development



And these are just  
some examples...

# Our choice for this tutorial



WildFly 17 (formerly JBoss  
Application Server)  
<http://wildfly.org>



Eclipse IDE 2019.06  
<http://www.eclipse.org>

MySQL Community Server 8.0  
<http://www.mysql.com>

Workbench

Connector/J



# Installation instructions

<https://github.com/dwws-ufes/jbutler/wiki/Tutorial00>

## Tool installation and configuration

### Set-up Eclipse to work with WildFly

The follow  
also work

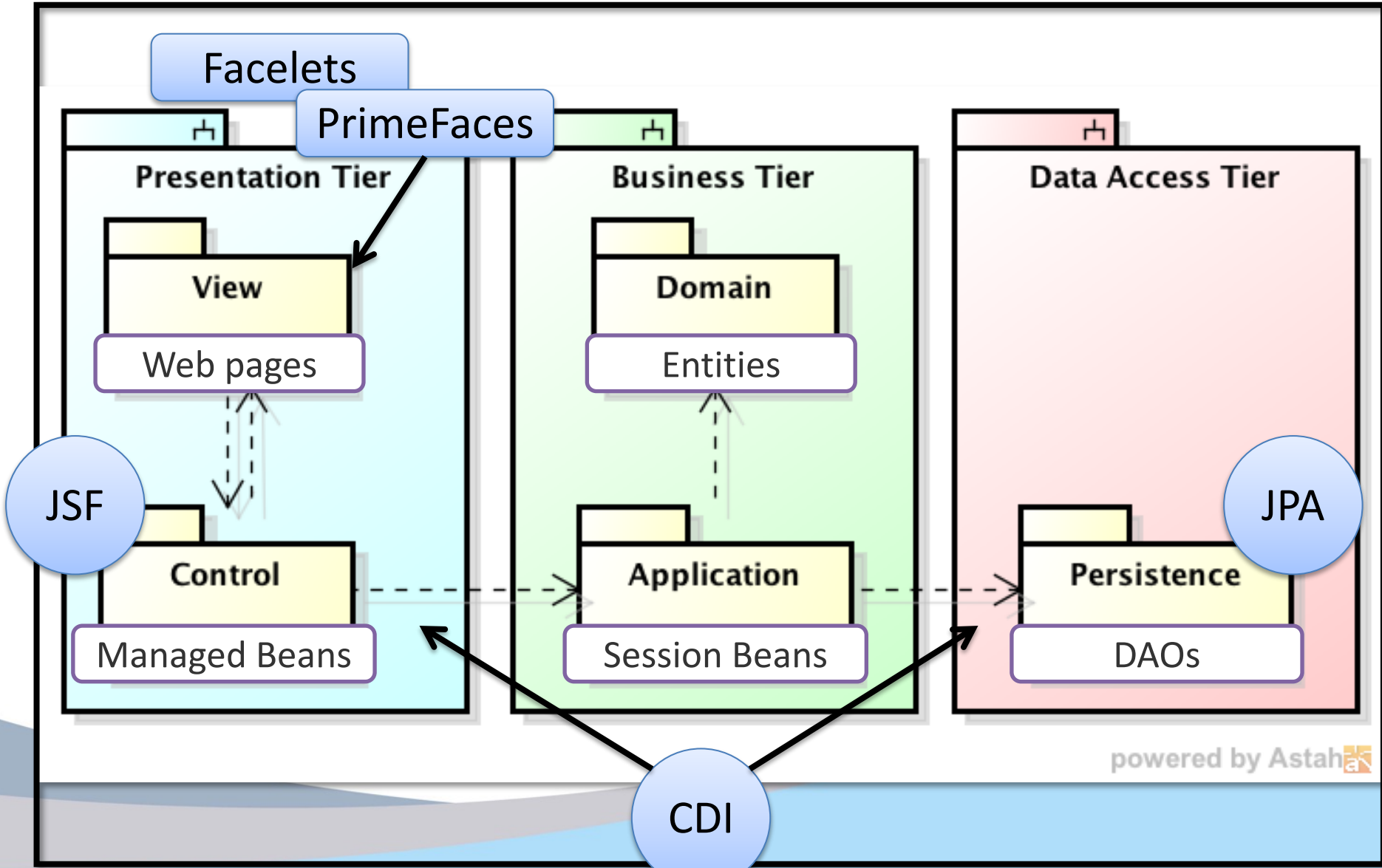
- Oracle
- Eclipse
- WildFly
- MySQL
- MySQL
- MySQL

Installation  
we have  
Eclipse s  
Eclipse I  
your har  
\$ECLIPS  
workben  
system's  
distribut

To deploy our application in WildFly during development more easily, it's recommended to integrate the IDE Eclipse with the application server WildFly. This is done with some tools provided by JBoss/Red Hat, which can now be installed in a more straightforward way in the latest Eclipse version:

1. Open Eclipse;
2. Click at the menu *Help > Eclipse Marketplace...*;
3. In the *Find:* field, type `JBoss` and click *Go*;
4. Locate *JBoss Tools 4.12.0.Final* (or a more recent version) and click *Install*, then *Confirm >*, select *I accept the terms of the license agreements* and click *Finish*. Wait for installation and restart Eclipse afterwards;
5. Open the *Servers* view (if not visible, use the *Window > Show View* menu);
6. Click on the *"No servers are available. Click this link to create a new server..."* link, which is shown when the *Servers* view is empty. Alternatively, right-click the blank space at the *Servers* view and select *New > Server*;
7. Under *JBoss Community* folder, select *WildFly 17*. Click *Next* twice;
8. Fill in the server's directory (pointing it to `$WILDFLY_HOME`, whatever that is in your system) and click *Finish*.

# My proposed architecture





# Database creation and connection

<https://github.com/dwws-ufes/jbutler/wiki/Tutorial00>

## Set-up WildFly to connect to MySQL

WildFly  
we need  
server i

### Database creation and set-up

We will use JPA (Java Persistence API), one of the Java EE standards, persisting our objects in a relational database stored in the MySQL server. We will, therefore:

1. In the console, run the following command to create the database schema named `oldenburg` ;
2. Use the following command to create a database user named `dwws` with password `dwws` ;
3. Give user `dwws` full permission for the schema `oldenburg` .

(For all example applications that I develop during the *Web Development and the Semantic Web* course I teach --- whose acronym, in Portuguese, is **DWWS** --- I use the same database user `dwws` . You can use whatever username and password suits you best, as long as you configure the datasource later with the same credentials.)

To do that, use MySQL Workbench. Once you open it, connect to the server using the *root* user (the administrator) and you should see a screen similar to the figure below (it's from an older version, so you might need to adapt a bit). If you see an error message at the bottom of the screen indicating that a connection to the server could not be established, click on *Server > Startup/Shutdown* and click the button to start the server.



# Overview of the tutorial

- Create the project with the appropriate configuration;

- Apply a decorator so it looks good right away;

Facelets

- Write domain classes and create the DB schema;

JPA

- Implement the guest registration feature.

JSF

CDI

JPA



# I'm confused...



# 1 - Initial project configuration

- Some Java EE technologies require configuration:
- CDI:
  - *WEB-INF/beans.xml;*
- JSF:
  - *WEB-INF/faces-config.xml;*
  - *Resource bundles, locale, navigation rules, etc.;*
  - *Not used in JavaHostel;*
- JPA:
  - *META-INF/persistence.xml;*
  - *Data source, persistence provider and its properties.*

# How does it work?

- Multiple JPA providers:
  - *The container has one (or more) implementation(s): WildFly → Hibernate, GlassFish → EclipseLink;*
  - *You can add your favorite one to the container;*
- JPA Providers are not standard (persistence.xml is):
  - *Hibernate: hibernate.hbm2ddl.auto = create;*
  - *EclipseLink: eclipselink.ddl-generation = create-tables;*
- Once JPA is configured, the container is ready to provide us with an Entity Manager, which can perform ORM operations (we'll get there);
- CDI and JSF just needs the configuration file to be there...

## 2 - Decoration with Facelets

- Download an existing template (or make your own);
- Create a decorator file: HTML file with placeholders for parts of the page;
- Apply the decorator to all pages;
- You can have different decorators for different sections, different users/roles, etc.;
- We will use XHTML (HTML + XML) from now on. Trust me, it's better.

# How does it work?

- Facelets is based on JSF:

```
xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:h="http://java.sun.com/jsf/html"
```

- The decorator is a standard (X)HTML page, but:
  - *<h:body> and <h:head> allow JSF features to work;*
  - *Tags in the decorator show where content goes:*

```
<ui:insert name="title" />
<ui:insert name="contents">Blank page.</ui:insert>
```

Default value if page doesn't provide  
contents for a specific section.

# How does it work?

- Pages that use the decorator are not HTML documents, but Facelets compositions:

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ui="http://java.sun.com/jsf/facelets"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:h="http://java.sun.com/jsf/html"
  template="/resources/decorator.xhtml">
```

- Pages only need to worry about their contents. No need to repeat layout code:

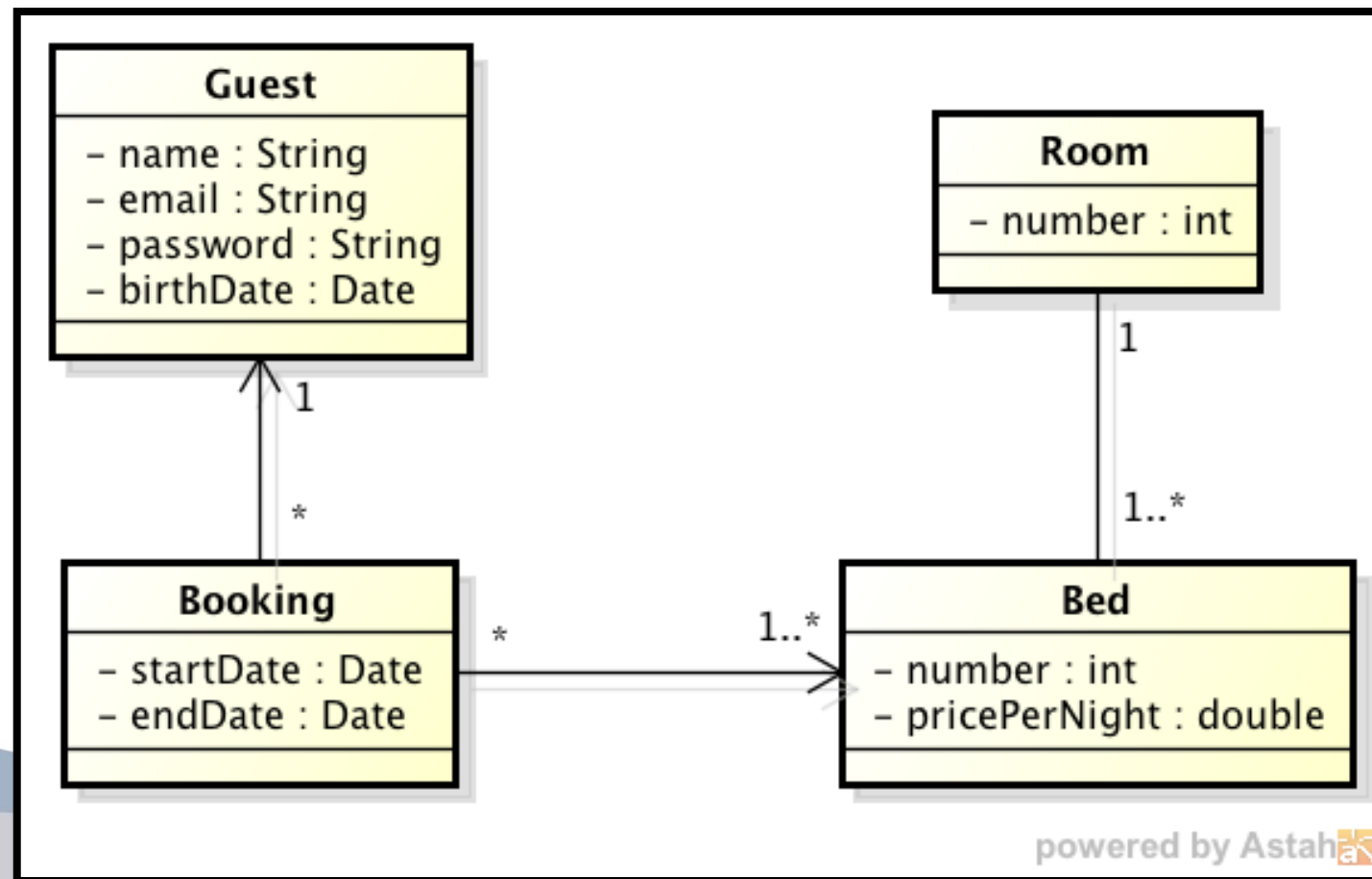
```
<ui:define name="title">Welcome</ui:define>
<ui:define name="contents">
  <h1>Welcome to JavaHostel</h1>
  <p>Under development.</p>
</ui:define>
```

- CSS also helps a lot...



## 3 - Let's talk about the domain

- Implement the following domain classes, plus tell JPA how to map them to relational database tables:



# How does it work?

- Each class that maps to the RDB is called an Entity:

```
@Entity
public class ...
```

- Although not mandatory, entities should have artificial IDs to serve as PKs (again, trust me, it's better):

```
@Id @GeneratedValue(strategy = GenerationType.AUTO)
private Long id;
```

- Simple attributes need no annotation (but if must, you can use @Basic):

```
@Basic
private String name;
```

# How does it work?

- Dates can store date-only, time-only and date/time:

```
@Temporal(TemporalType.DATE)
private Date birthDate;
```

- Associations can be one-to-many, many-to-one and many-to-many. Use standard collections:

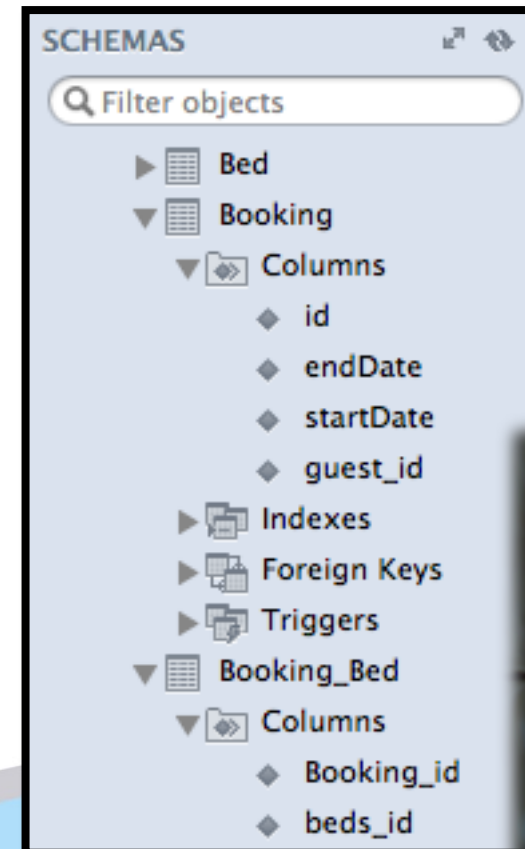
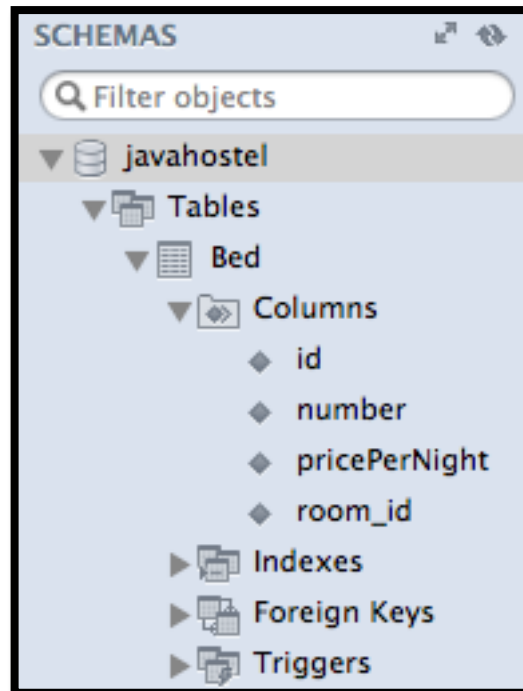
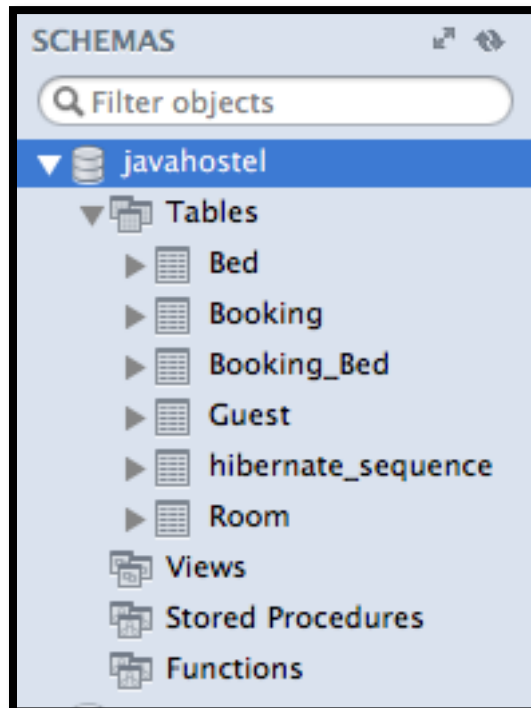
```
@ManyToOne private Guest guest;
@OneToMany private Set<Bed> beds;
```

- Bi-directional associations need to be connected with the other side (or there will be 2 associations!):

```
@OneToMany(cascade = CascadeType.ALL, mappedBy = "room")
private Set<Bed> beds;
```

## 4 - Database schema creation

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



## 5 - Finally, a feature!

- Guest registration:
  - *Guest enters her name, birthDate, email and password;*
  - *If guest is underage, display a message explaining minors cannot register;*
  - *Otherwise, save the information in the database and display a success message.*

# How does it work?



JSF Page

## JSF Expression Language / CDI



JSF Controller

```
<h:inputText id="name"
value="#{registrationController
.guest.name}" size="30" />

<h:inputText id="birthDate"
value="#{registrationController
.guest.birthDate}" size="10">
  <f:convertDateTime
  pattern="dd/MM/yyyy" />
</h:inputText>

<h:commandButton
action="#{registrationControll
r.register}" value="Register"
/>
```

JSF Converters!

```
@Named
public class
RegistrationController
implements Serializable {
```

```
    Guest getGuest() ...
    ... String register() ...
```

```
    ... getName() ...
    ... setName() ...
    ... getBirthDate() ...
    ... setBirthDate() ...
```



Guest.java

# JSF page <-> controller communication

- Whenever a JSF page is rendered, EL is interpreted:
  - *I.e., `#{registrationController.guest.name}` becomes `registrationController.getGuest().getName()`;*
  - *The `registrationController` object is created for you;*
- When you send the next request to the server, JSF will update the state of the components at the controller:
  - *I.e., `registrationController.getGuest().setName(n)` is called, where  $n$  = contents of the field;*
  - *This happens in **any** request, not only the ones that call `registrationController.someMethod()`!*
- If the field has a formatter, it parses/formats the data.

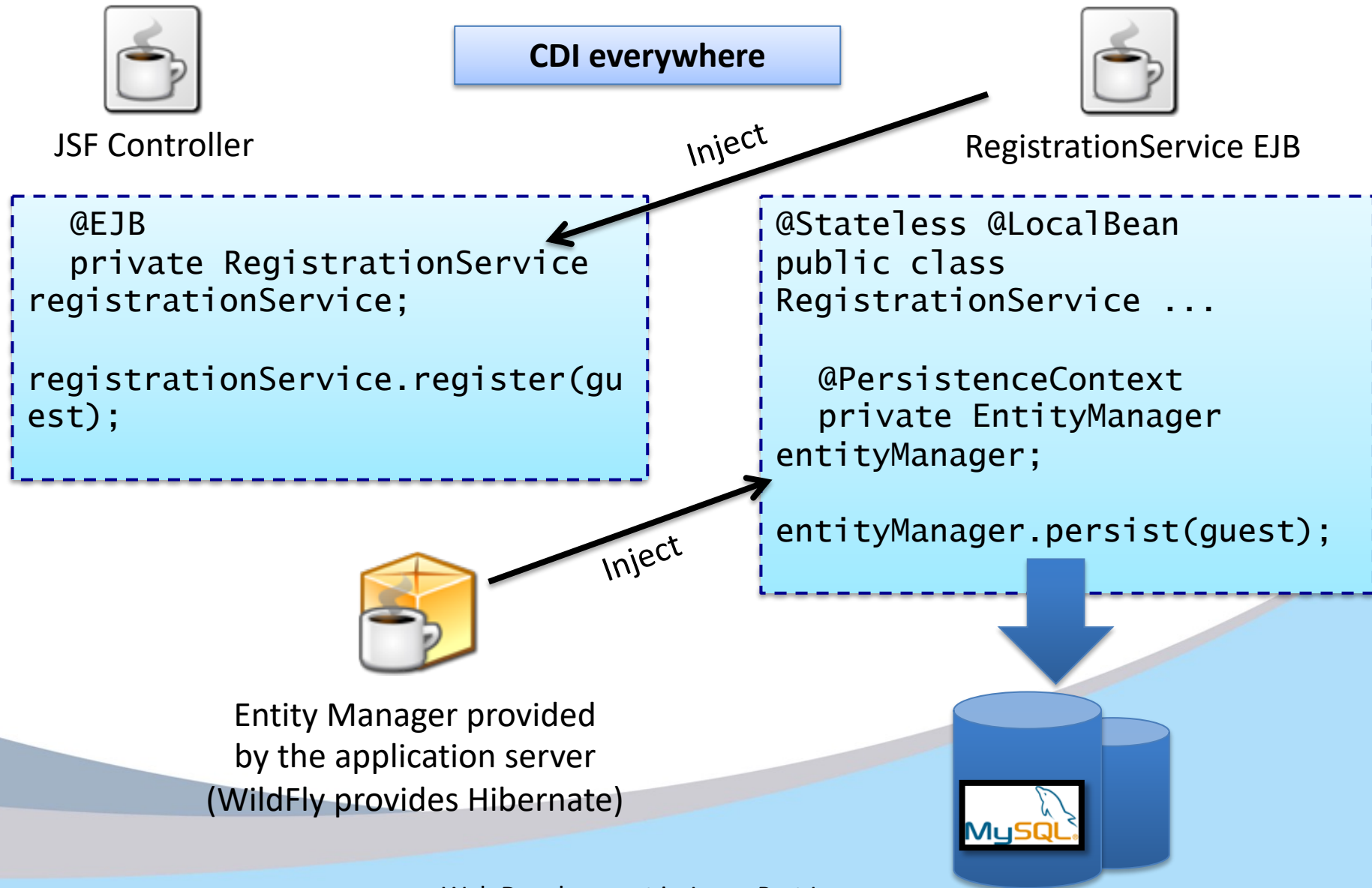


# Enters CDI



- Why is this particular class instantiated when the EL refers to registrationController?
  - *@Named creates a default EL reference;*
  - *You can change the default: @Named("regCtrl");*
- When is this object created? How many are created?
  - *@SessionScoped = one per session (client);*
  - *There are other scopes: request, conversation, application, flash (this last one is JSF, not CDI);*
- There is much more to CDI...

# CDI meets EJBs



# EJB stuff

- @Stateless = no state = no object attributes;
  - *The container can have a share pool of instances;*
  - *To store state, use @Stateful, then give it a scope;*
  - *An EJB can also be @Singleton now;*
- @LocalBean means the EJB is not split between interface and implementation;
  - *Now that I think about it, it should have been...*
  - *Remote vs. local interface.*



# Back to JSF

- Where to go after the controller executes?

```
catch (UnderAgeGuestException e) {
    age = e.getAge();
    return "/registration/underage.xhtml";
}
return "/registration/success.xhtml";
```

- *Returning null or void, the same page is reloaded;*
- *You can also have navigation rules in faces-config.*

- Data from the controller is shown in the result page:

```
<p>Dear <h:outputText value="#{registrationController.guest.name}"/>,
unfortunately underage people are not allowed to register as guests and,
according to your birth date, you have only <h:outputText
value="#{registrationController.age}" /> years.</p>
```

```
<p>Dear <h:outputText value="#{registrationController.guest.name}"/>, welcome to
JavaHostel.</p>
```

## Next

- More on JSF: AJAX support, converters, formatters, i18n, Facelets components, etc.;
- More on CDI: scopes, producers, qualifiers, interceptors, decorators, events, etc.;
- More on JPA: element collections, bean validation, JPQL, criteria API, etc.;
- More on EJBs: authorization, local/remote interfaces, singleton EJBs, asynchronous invocations, etc.;
- Use a JSF component library (e.g., PrimeFaces).