Technology Background

Development environment, Skeleton and Libraries

Christian Kroiß
(partly based on slides by Dr. Andreas Schroeder)

Outline

Lecture 1

I. Eclipse

II. Redmine, Jenkins, Git

Lecture 2

III. Short Intro to Java Web Applications

IV. A brief overview of Wicket (and AJAX)

V. TBIAL Skeleton Overview

VI. Testing & Logging: Junit, WicketTester, Mockito, Log4j

Part III. Short Intro to Java Web Applications

Java Servlets: CGI for Java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {

  public void doGet(HttpServletRequest req, HttpServletResponse res)
                                throws ServletException, IOException {

    res.setContentType("text/html");
    PrintWriter out = res.getWriter();

    out.println("<HTML>");
    out.println("<HEAD><TITLE>Hello World</TITLE></HEAD>");
    out.println("<BODY>");
    out.println("<BIG>Hello World</BIG>");
    out.println("</BODY></HTML>");
  }
}
```

Deployment Descriptors

WEB-INF/web.xml

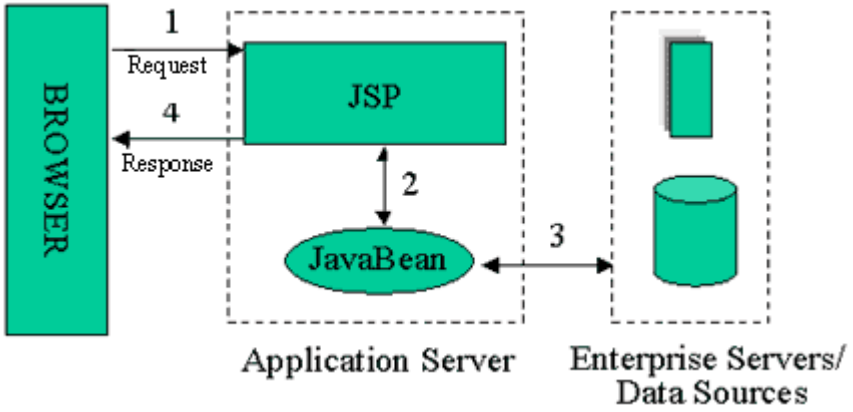
```
<web-app>
  <servlet>
    <servlet-name>hi</servlet-name>
    <servlet-class>HelloWorld</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>hi</servlet-name>
    <url-pattern>/hello.html</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>hi</servlet-name>
    <url-pattern>*.hello</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>hi</servlet-name>
    <url-pattern>/hello/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Java Server Pages (JSP): using Java like PHP

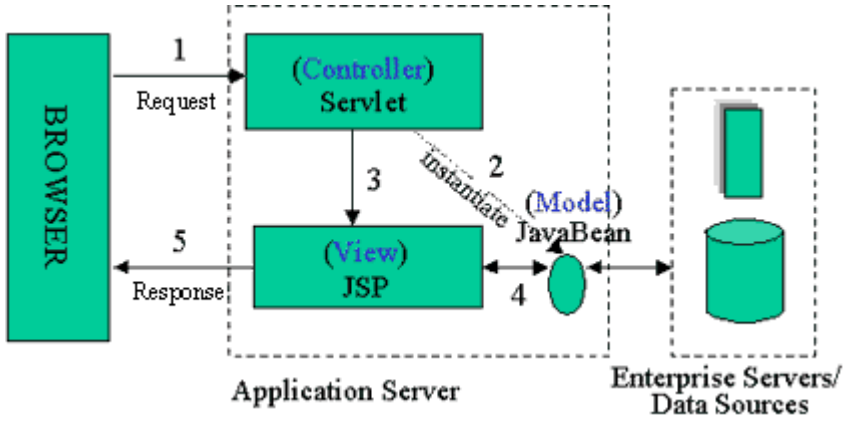
```
<%@ page import="java.util.*" %>
<HTML>
<BODY>
<%!
    Date theDate = new Date();
    Date getDate()
    {
         return theDate;
    }
%>
Hello!  The time is now <%= getDate() %>
</BODY>
</HTML>
```

MVC for the Web

Model 1

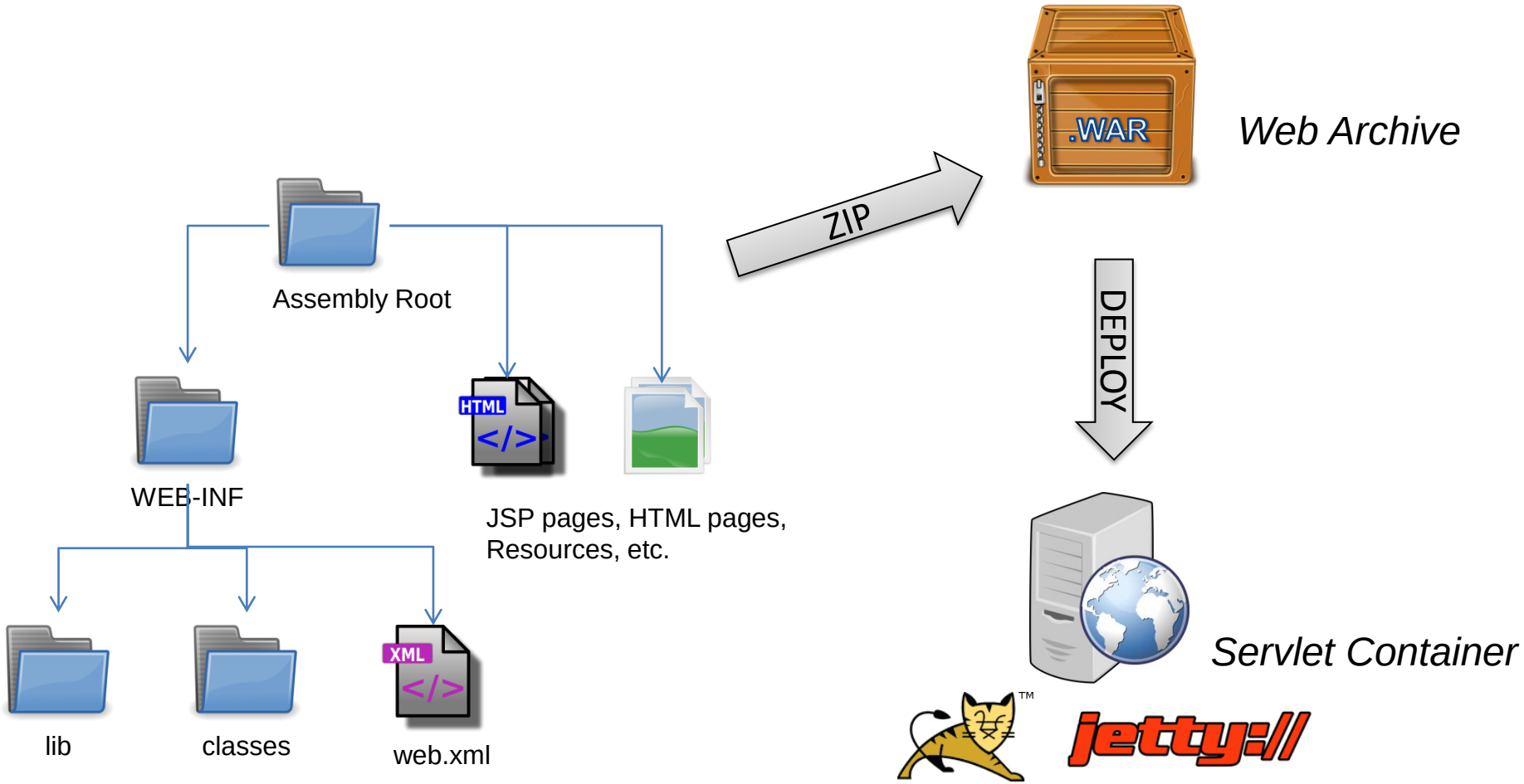


Model 2



Model 1 vs Model 2: http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html

Deployment: Web Modules



Part IV. Something Wicket this way comes…

State handling

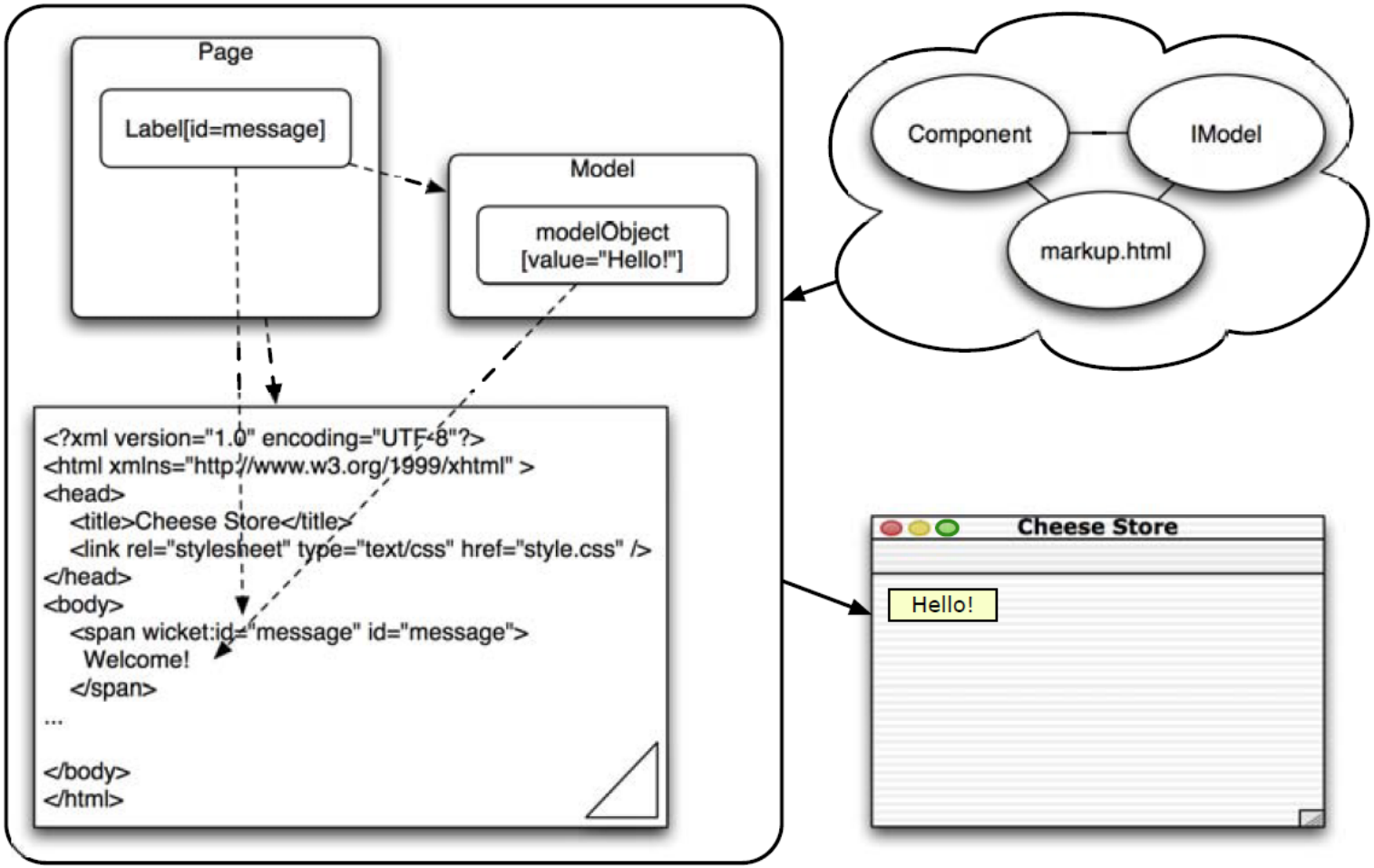
- Wicket aims to solve the **impedance mismatch between the stateless HTTP protocol and OO Java programming**.
  - State important, e.g. for tab-panels, etc.
  - Why not encoding state in request URLs?
    - security issues, hard to handle
  - Why not put state in session?
    - Back Button problem, etc.

➔ *Wicket handles state transparently*

Plain Java + Plain HTML

- Plain Java
  - Regular Java OOP that feels like Swing/SWT
  - Reusable widgets by inheritance and composition
  - Full IDE support
  - Refactoring
- Plain HTML
  - "Wicket doesn’t just reduce the likelihood of logic creeping into the presentation templates—it eliminates the possibility altogether."
  - Create layout with only HTML + CSS

The Wicket component triad



from [1]

A first Wicket example (1)

Source code in browser

```
<html>
<head>
<title>My Home Page</title>
</head>
<body>
  <b>Counter: </b>
  <span wicket:id="counter">7</span>
  <br />
  <a href="./?0-10.ILinkListener-link"
    wicket:id="link"><img
      src="images/button.png"
      width="32" border="0"></a>
</body>
</html>
```

WicketLabApplication.java

```
public class WicketLabApplication
  extends WebApplication {

  @Override
  public Class<? extends Page> getHomePage() {
    return MyHomePage.class;
  }

}
```

A first Wicket example (2)

MyHomePage.java

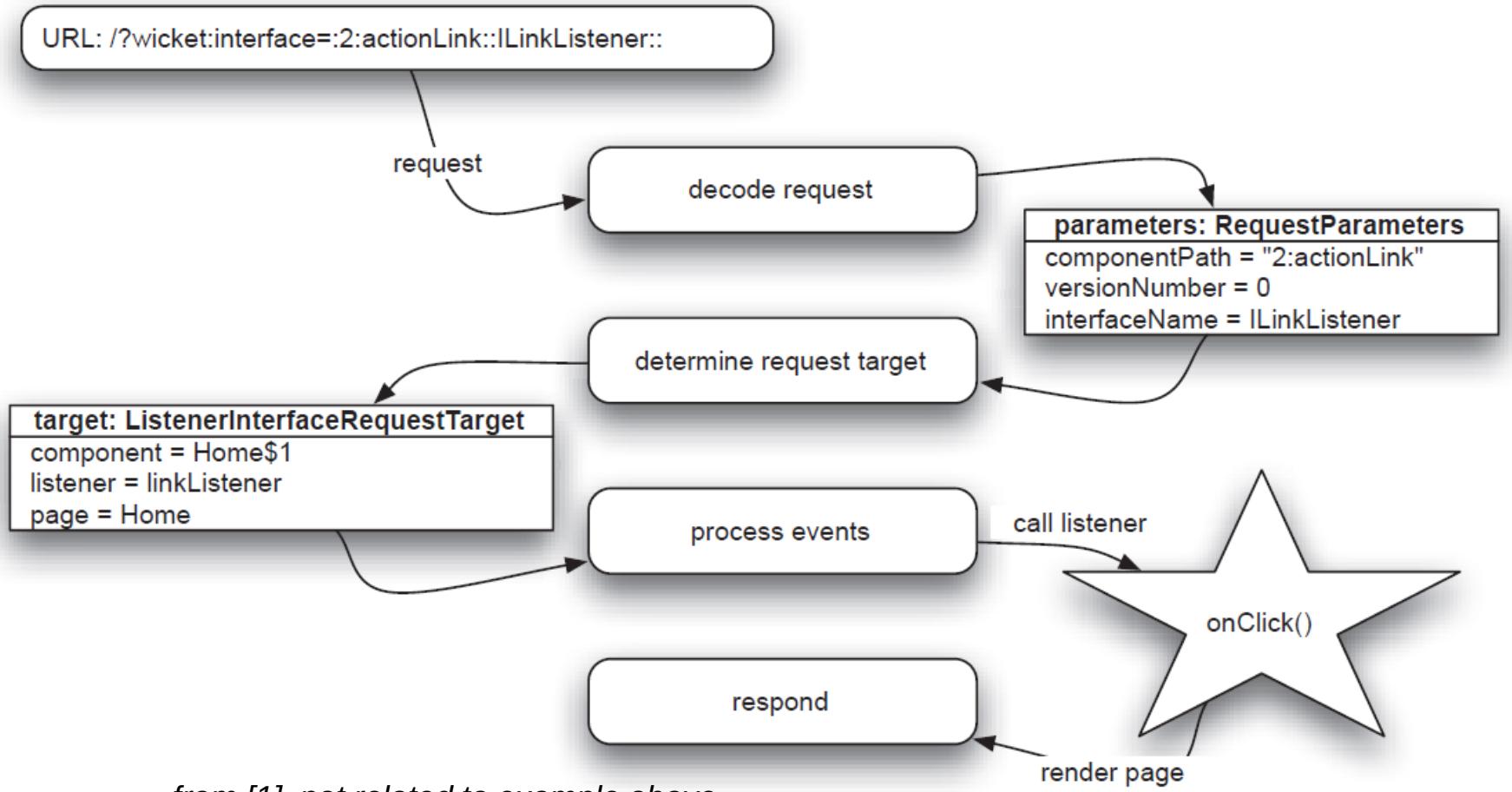
```
public class MyHomePage extends WebPage {
    private Label fLabel;
    private Link<Void> fLink;
    private IModel<Integer> fCounter;
    public MyHomePage() {
      fCounter= new Model<Integer>(0);
      fLabel= new Label("counter", fCounter);
      fLink= new Link<Void>("link") {
        public void onClick() {
          fCounter.setObject(fCounter.getObject() + 1);
        }
      };
      add(fLink);
      add(fLabel);
    }
}
```

MyHomePage.html

```
<html>
  <head>
    <title>My Home Page</title>
  </head>
  <body>
    <b>Counter: </b>
    <span wicket:id="counter">42</span>
    <br />

    <a href="#" wicket:id="link">
        <img src="images/button.png"
          width="32" border="0"></a>
  </body>
</html>
```

A glimpse behind the Wicket scenes (1)



from [1], not related to example above

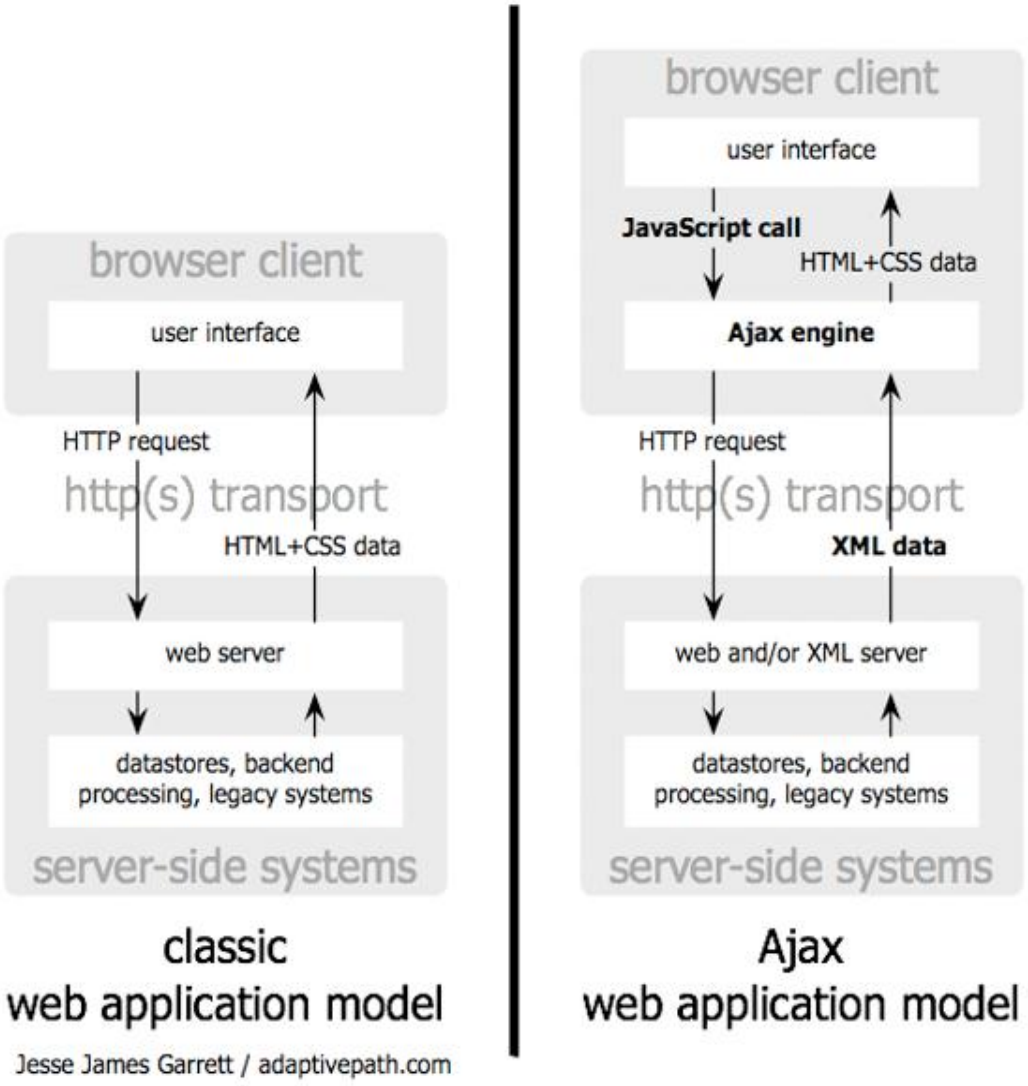
A glimpse behind the Wicket scenes (2)

```
<web-app>
...
  <display-name>WicketLab</display-name>
  <filter>
    <filter-name>WicketFilter</filter-name>
    <filter-class>org.apache.wicket.protocol.http.WicketFilter</filter-class>
    <init-param>
      <param-name>applicationClassName</param-name>
      <param-value>de.lmu.ifi.pst.WicketLabApplication</param-value>
    </init-param>
  </filter>
  <filter-mapping>
    <filter-name>WicketFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
...
</web-app>
```

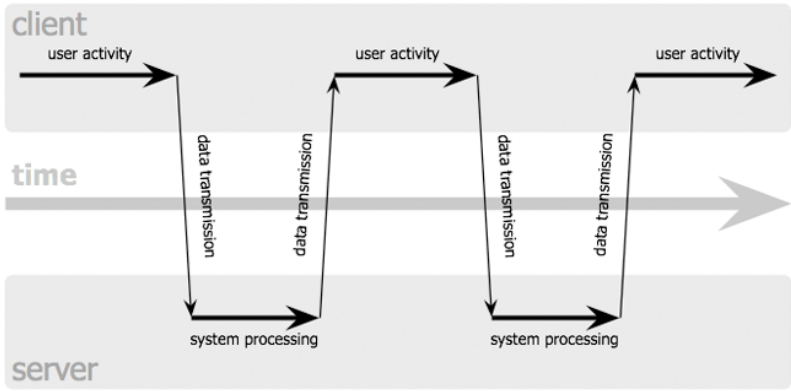
Asynchronous **J**avaScript **a**nd **X**ML

- Coined in 2005 article (see [2])
- Originally meant to Incorporate
  - standards-based presentation using XHTML and CSS;
  - dynamic display and interaction using the Document Object Model;
  - data interchange and manipulation using XML and XSLT;
  - asynchronous data retrieval using `XMLHttpRequest`;
  - and JavaScript binding everything together.
- Now often used with JSON instead of XML

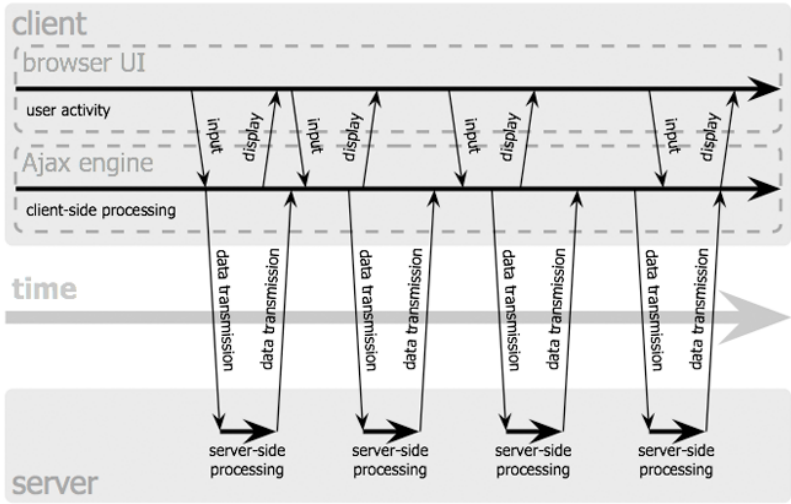
The AJAX web application model



classic web application model (synchronous)



Ajax web application model (asynchronous)



Jesse James Garrett / adaptivepath.com

Pure AJAX

http://www.w3schools.com/ajax/tryit.asp?filename=tryajax_suggest

Start typing a name in the input field below:

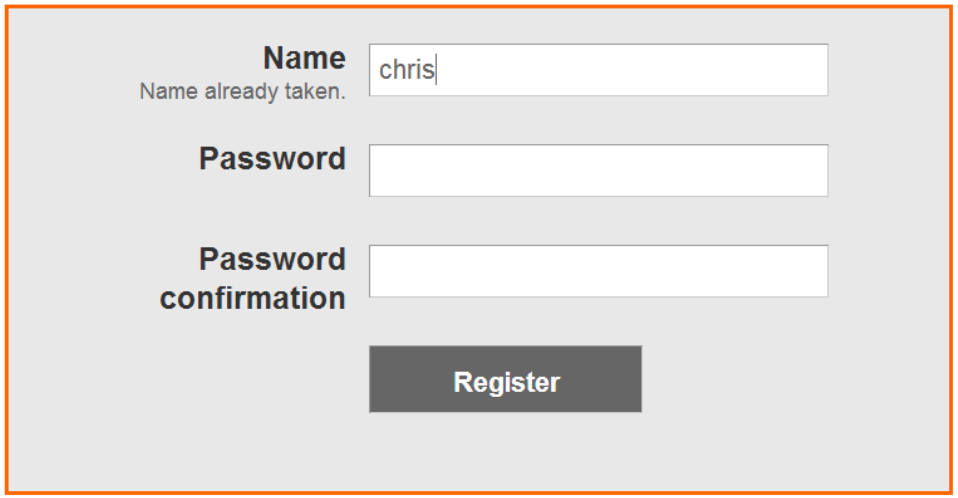
First name: El

Suggestions: Elizabeth , Ellen

```
<html><head><script>
function showHint(str) {
  var xmlhttp;
  if (str.length==0) {
    document.getElementById("txtHint").innerHTML="";
    return;
  }
  if (window.XMLHttpRequest) {
    // code for IE7+, Firefox, Chrome, Opera, Safari
    xmlhttp=new XMLHttpRequest();
  }
  else {// code for IE6, IE5
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
  xmlhttp.onreadystatechange=function() {
    if (xmlhttp.readyState==4 &&
      xmlhttp.status==200) {
        document.getElementById(
          "txtHint").innerHTML=
            xmlhttp.responseText;
    }
  }
  xmlhttp.open("GET",
    "gethint.asp?q="+str,true);
  xmlhttp.send();
}
</script></head>
<body>
<h3>Start typing a name in the input
field below:</h3>
<form action="">
First name: <input type="text" id="txt1"
onkeyup="showHint(this.value)" />
</form>
<p>Suggestions: <span id="txtHint">
</span></p>
</body></html>
```

Register.java (in TBIAL Skeleton)

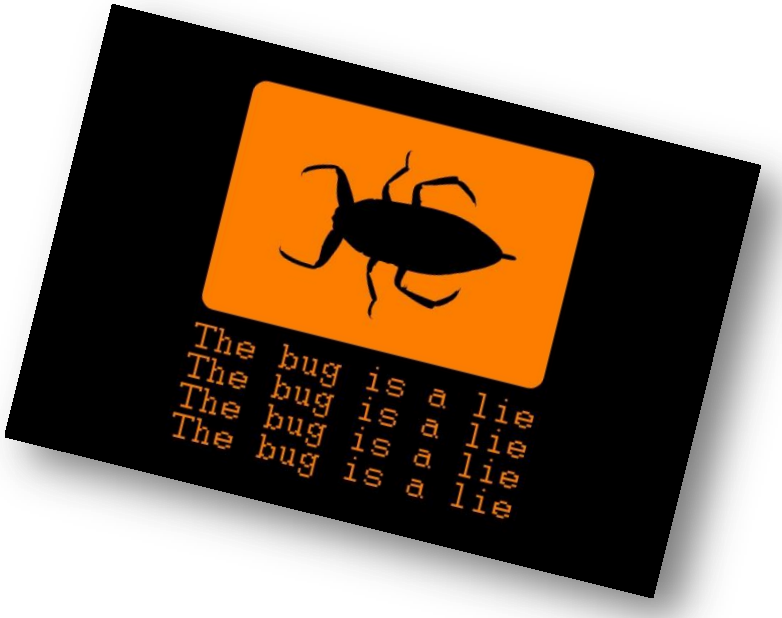
```
fName= new TextField<String>("name", new Model<>(""));
…
OnChangeAjaxBehavior onNameChange= new OnChangeAjaxBehavior() {
  @Override
  protected void onUpdate(AjaxRequestTarget target) {
    String name= fName.getModelObject();
    if (getDatabase().hasUserWithName(name)) {
      fNameFeedback.setDefaultModelObject(
        "Name already taken.");
    } else {
      fNameFeedback.setDefaultModelObject(" ");
    }
    target.add(fNameFeedback);
  }
};
fName.add(onNameChange);
```



Summary

- Wicket…
  - offers a light-weight object-oriented programming model for web applications
  - enforces clear separation of Java and HTML
  - has pretty neat AJAX support
- For further information, see [1] and http://svn.pst.ifi.lmu.de/redmine/projects/swep13/wiki/Wicket_Howtos

Part V. The Skeleton (which is not a real skeleton)



Learning Targets

- Understand the structure of the skeleton
- Know what is done where
- Have a starting point for inspecting source and programming spikes

Project structure

main source folder

test source folders

compiler output folder

config files etc.

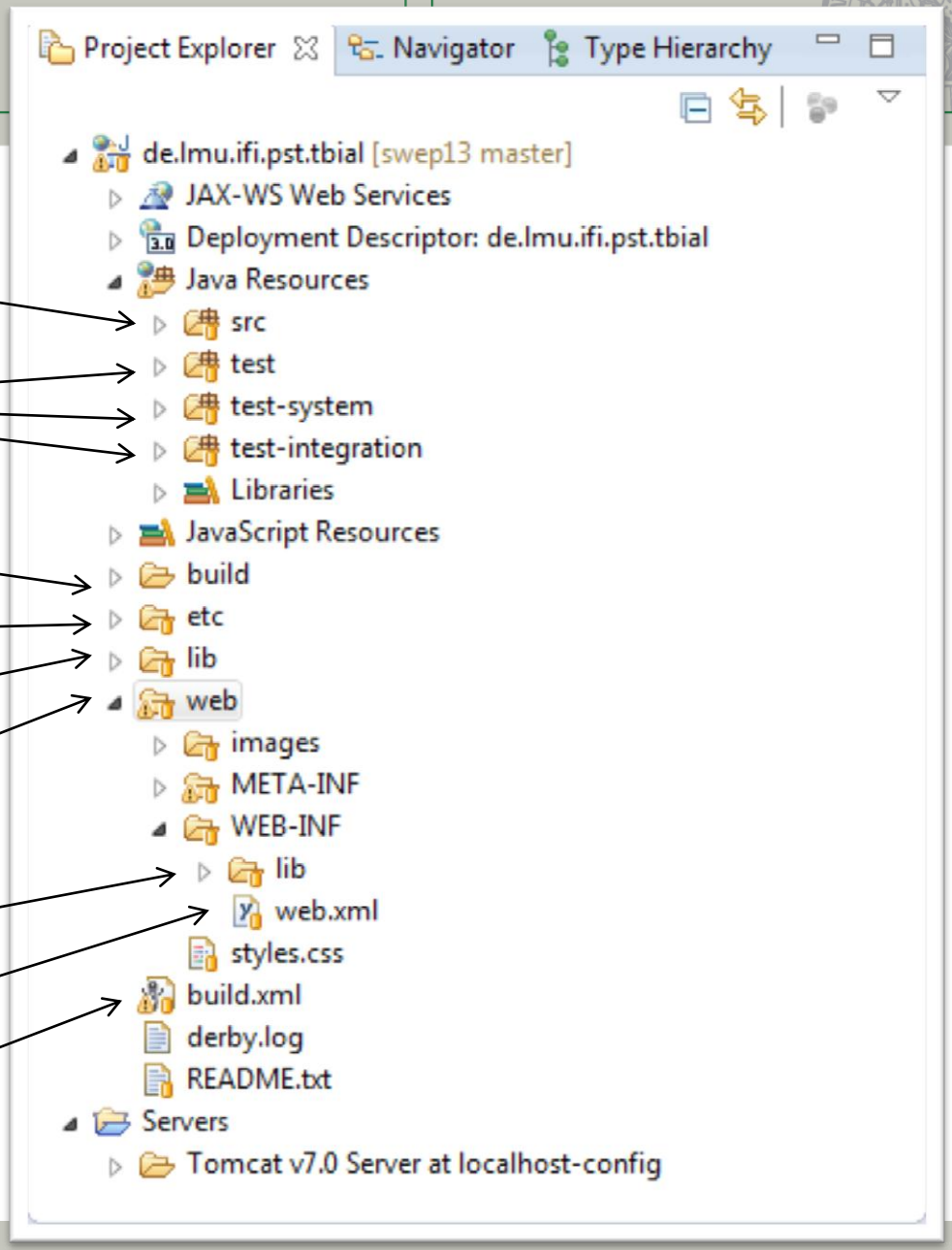
development libraries (e.g. testing)

assembly root folder

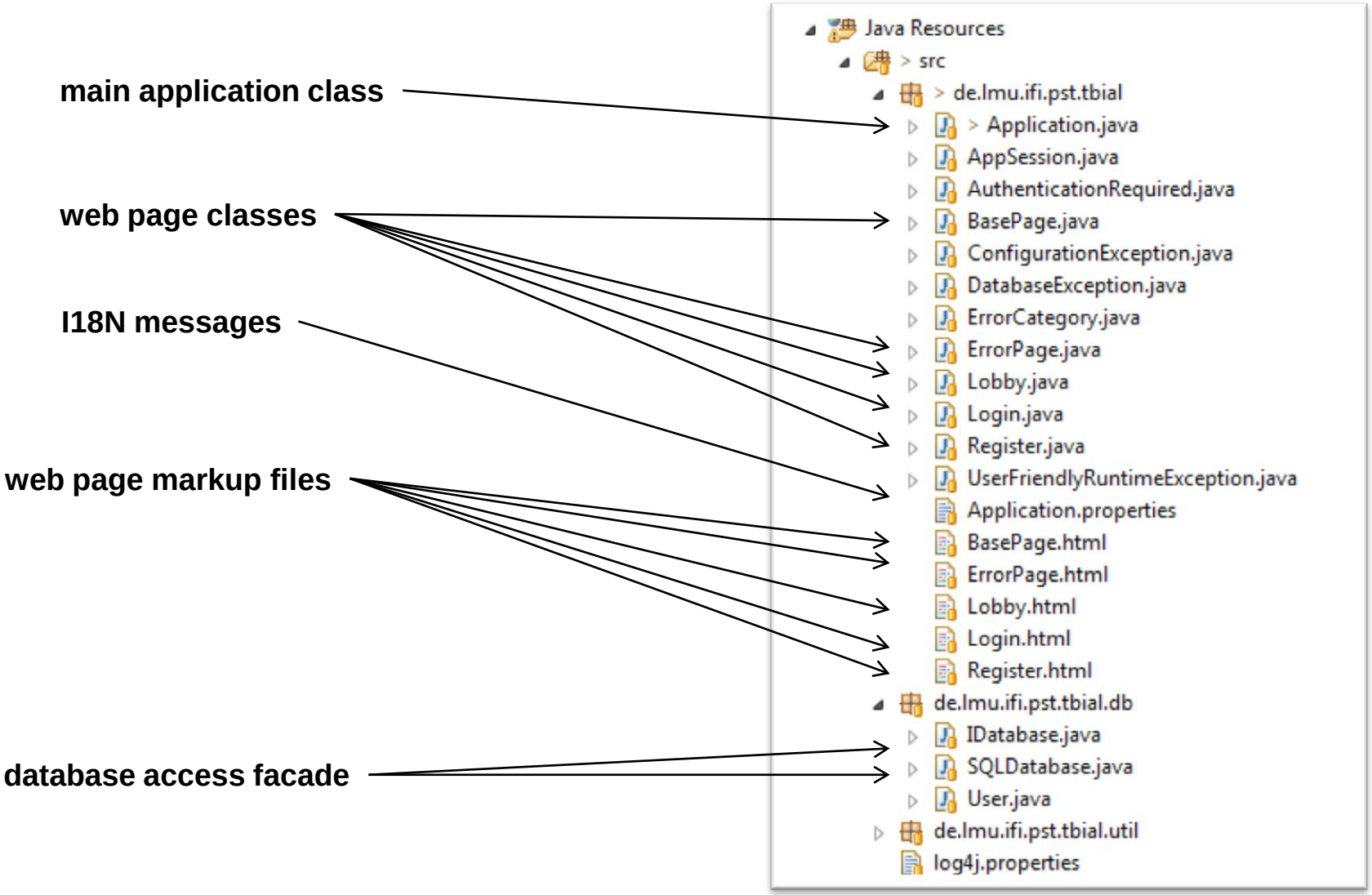
application libraries (deployed)

deployment descriptor

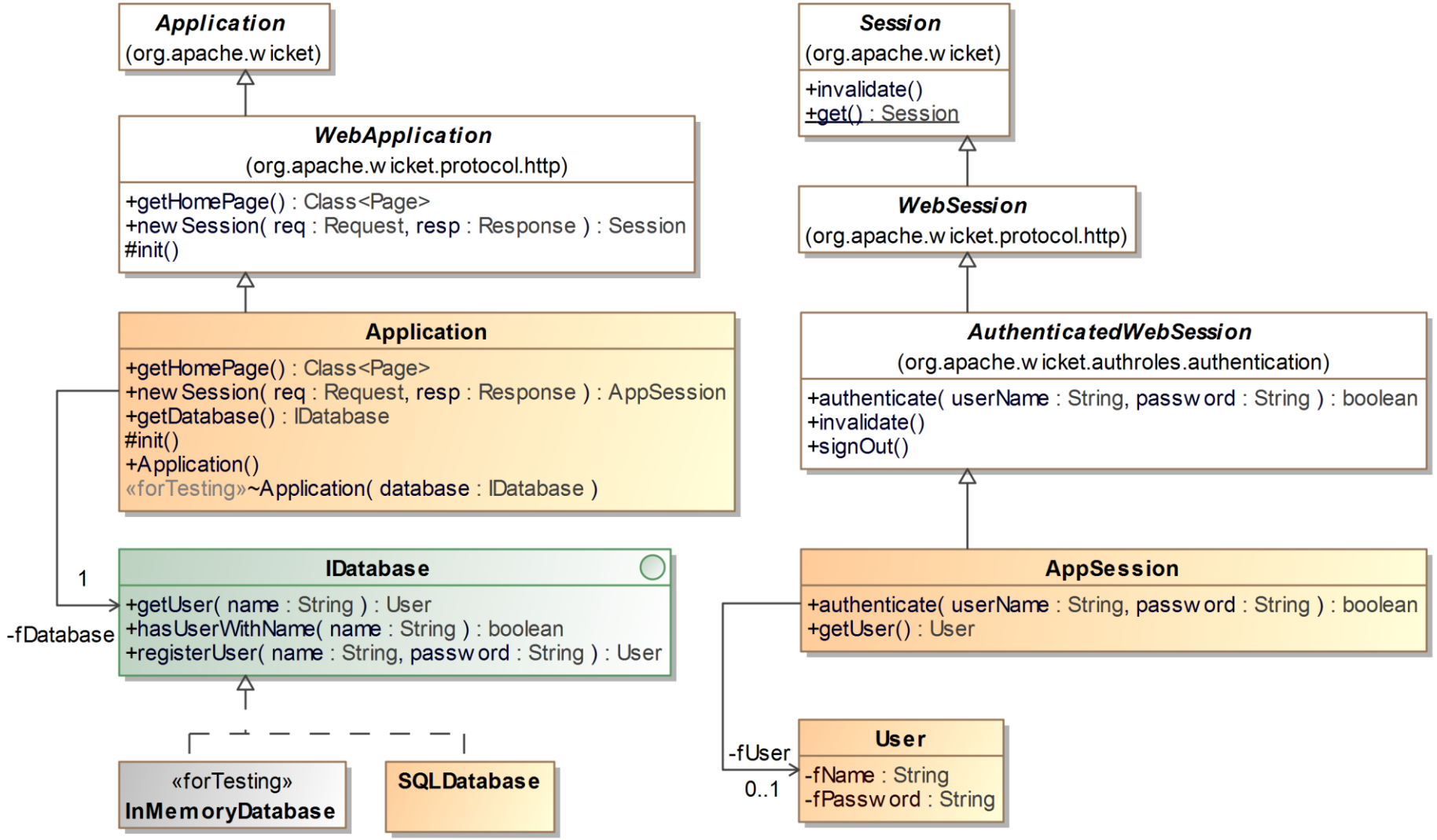
Ant buildfile



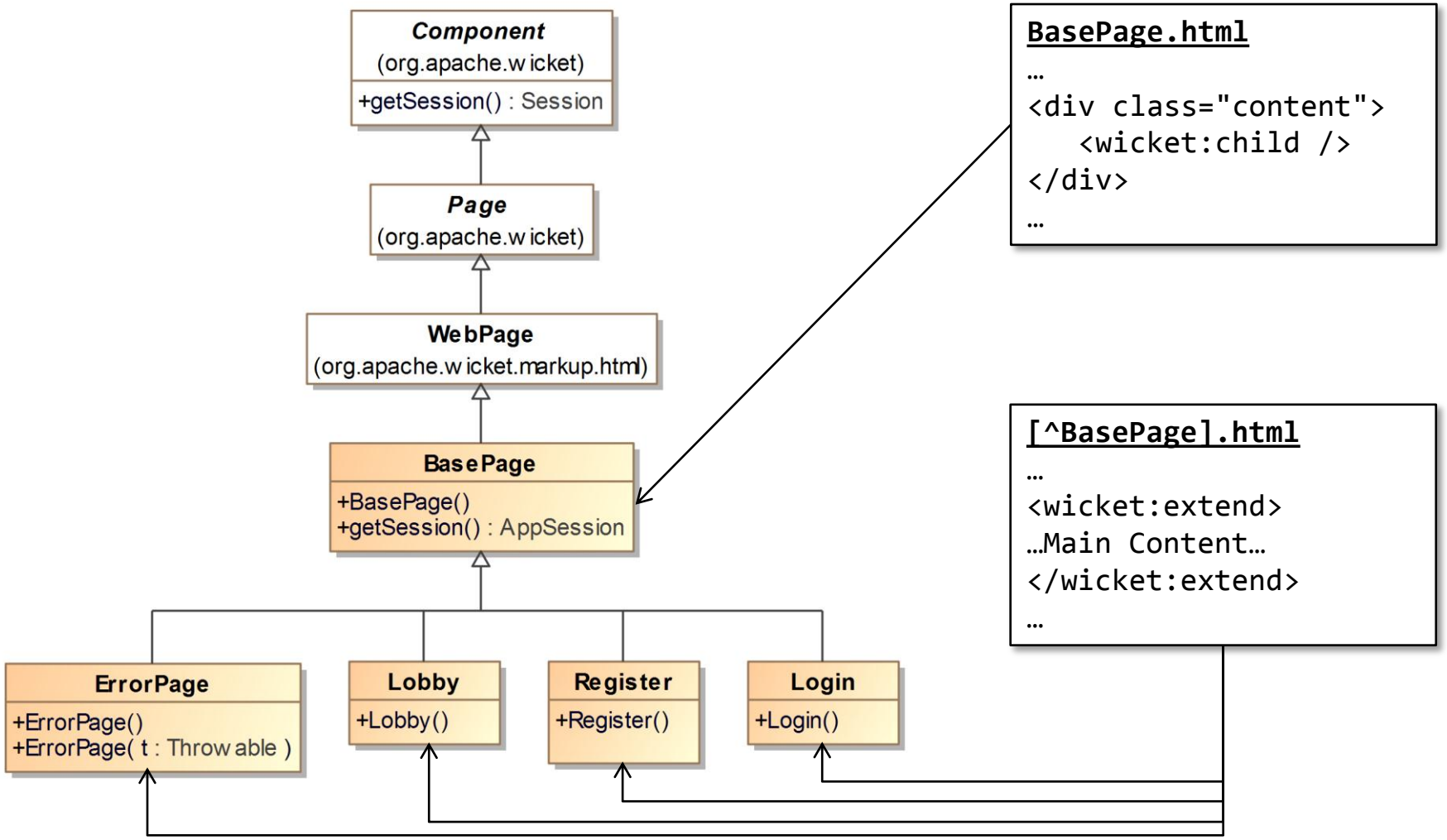
Project structure (2)



Main classes in the skeleton



Web Pages with inheritance



Authentication & Authorization

- **Authentication** is done in the `authenticate()` method of `AppSession`, which is called from the Login and Register page.
  1. simple lookup of User from the database
  2. check if password matches
  3. if successful, store user object in session
     otherwise redirect
- **Authorization** can be handled very comfortably with an annotation:
  - If a class is annotated with @AuthenticationRequired then it is only rendered if a user is signed in.

Authorization mechanism

Lobby.java

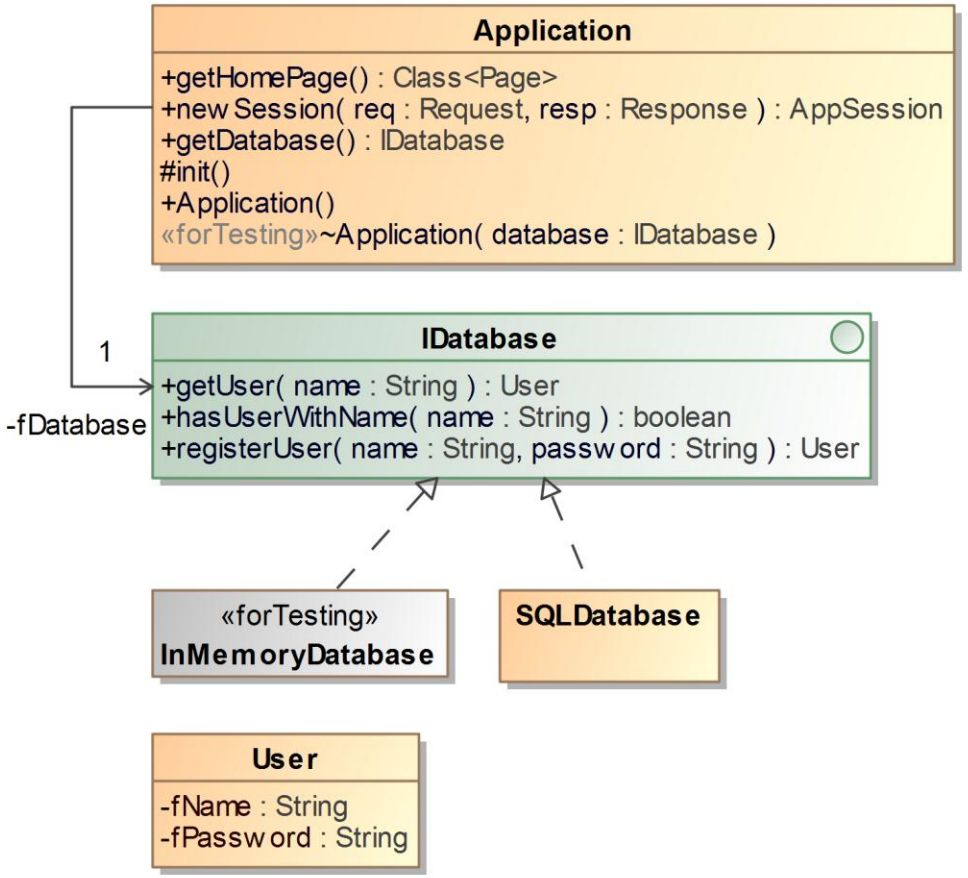
```
@AuthenticationRequired
public class Lobby extends BasePage {}
```

Application.java

```
…
getSecuritySettings().setAuthorizationStrategy(new IAuthorizationStrategy() {
  @Override
  public <T extends IRequestableComponent> boolean isInstantiationAuthorized(
    Class<T> componentClass) {
    boolean requiresAuthentication= componentClass.isAnnotationPresent(
      AuthenticationRequired.class);
    boolean isSignedIn= ((AppSession) Session.get()).isSignedIn();
    if (requiresAuthentication && !isSignedIn)
      throw new RestartResponseAtInterceptPageException(Login.class); // redirect to Login
    return true;
  }
… });
```

Database Access

- At the moment, only user names and plain passwords are stored in the database
- An in-memory database stub is used for unit tests
- Apache Derby is used for development (see [3]).
- **If** you have to change something, read [4].



… but first

Have no fear to experiment!

- Everything is safely stored in Git
- Eclipse has a local history, get familiar with it
- Not breaking things (locally) at least one time is (almost) a bad sign ☺

You need to know the code base!

Part VI. Testing & Logging: Junit, Mockito, WicketTester, and Log4j

http://svn.pst.ifi.lmu.de/redmine/projects/swep13/wiki/JUnit_Howtos

Goals of unit testing

- Increase confidence
- Show that the code works
- Facilitate change and feature integration

- Five steps make a unit test
  1. Set up fixture
  2. Create input
  3. Execute
  4. Check output
  5. Tear down

Fixtures

- Code worth testing has **dependencies**
  - Database, Config files, Environment variables
- A **test fixture** is the baseline for running the test
  - Goal: create a known and controlled environment
  - Data and environment is tailored to the test
- Setup and tear down
  - JUnit offers @Before and @After annotations for setup and tear down
  - **Setup**: setup code that is re-used among tests
  - **Tear down**: clean-up performed regardless of test result

Mockito

http://svn.pst.ifi.lmu.de/redmine/projects/swep13/wiki/Mockito_Howtos

Writing fixtures can be a lot of work, but

- Over time, a set of re-usable fixtures will emerge
- Mockito allows to quickly create one-shot fixture mocks

- Mockito lifecycle
  - Create mock object

```
fNetwork= mock(IManagedClientNetworkController.class);
```

  - Record behavior

```
when(fNetwork.isConnected()).thenReturn(true);
```

  - Use

```
fApplication= new ApplicationController(fNetwork);
```

  - Verify

```
verify(fNetwork).start();
verify(fNetwork).isConnected();
```

Unit Test Best Practices

Testing best practices

- Test **behavior**, not methods;
  Behaviors are paths through code!
- Method name: method_doesWhatIWant_underProperConditions
- Do not test code that cannot break
- Use **OO principles** for your tests (stay SOLID and DRY)
- Keep tests **orthogonal**
  - Check only one behavior in one test
  - Do not check the same behavior in several tests
- Keep the **architecture testable**
  - Test one code unit at a time
- Use fixtures and mocks

Testing Wicket Pages with WicketTester

- Use WicketTester (integrated in Wicket) to test web pages without starting a server. For further info, see [4].

```
@Test
public void echoForm() {
  WicketTester tester = new WicketTester();
  tester.startPage(EchoPage.class);
  tester.assertLabel("message", "");
  FormTester formTester = tester.newFormTester("form");
  assertEquals("", formTester.getTextComponentValue("field"));
  formTester.setValue("field", "Echo message");
  formTester.submit("button");
  tester.assertLabel("message", "Echo message");
  assertEquals("", formTester.getTextComponentValue("field"));
}
```

Example from [1]

Logging

- Sensible logging is a very important aspect of software quality.
  - In production, logfiles are the most important (or only) information source for problem analysis!
- The skeleton currently uses jog4j 1.2 (http://logging.apache.org/log4j/1.2/). This could be easily changed to something else using SLF4J (http://www.slf4j.org/) , which is used by Wicket anyway.
- For further indformation, see http://svn.pst.ifi.lmu.de/redmine/projects/swep13/wiki/Log4J_Howtos

Summary

Summary

I. **Eclipse**
   - Why, installation, getting help

II. **Git, Redmine, Jenkins**
   - Knowledge base, tool support for Scrum, continuous integration

III. **Java Web Applications**
   - **The very basics**

IV. **Wicket introduction**
   - **Basic architecture, AJAX support**

Summary

V. Skeleton Overview

- Project structure
- Authentication & Authorization
- Web page "inheritance"

VI. Testing & Logging

- Junit
- Mockito
- WicketTester
- log4j

References

[1] Martijn Dashorst and Eelco Hillenius. *Wicket in Action.* Manning, 2009.

[2] Jesse James Garrett. *Ajax: A New Approach to Web Applications.* 2005. http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications .

[3] http://svn.pst.ifi.lmu.de/redmine/projects/swep13/wiki/Eclipse_Setup#5-Database-Setup

[4] https://cwiki.apache.org/WICKET/testing-pages.html