Technology Background

Development environment, Skeleton and Libraries

Slides by Prof. Dr. Matthias Hölzl
(based on material by Dr. Andreas Schroeder and Christian Kroiß)

Outline

Lecture 1

I. Eclipse
II. Git

Lecture 2

III. Java Web Applications
IV. Wicket (and AJAX)
V. TBIAL Skeleton Overview
VI. Testing

Part III. Java Web Applications

Java Servlets: CGI for Java

```
package examples;

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

public class HelloWorld extends HttpServlet {

    private static final long serialVersionUID= 1L;

    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><h1>Hello World</h1></body>");
        out.println("</html>");
    }
}
```

Deployment Descriptors

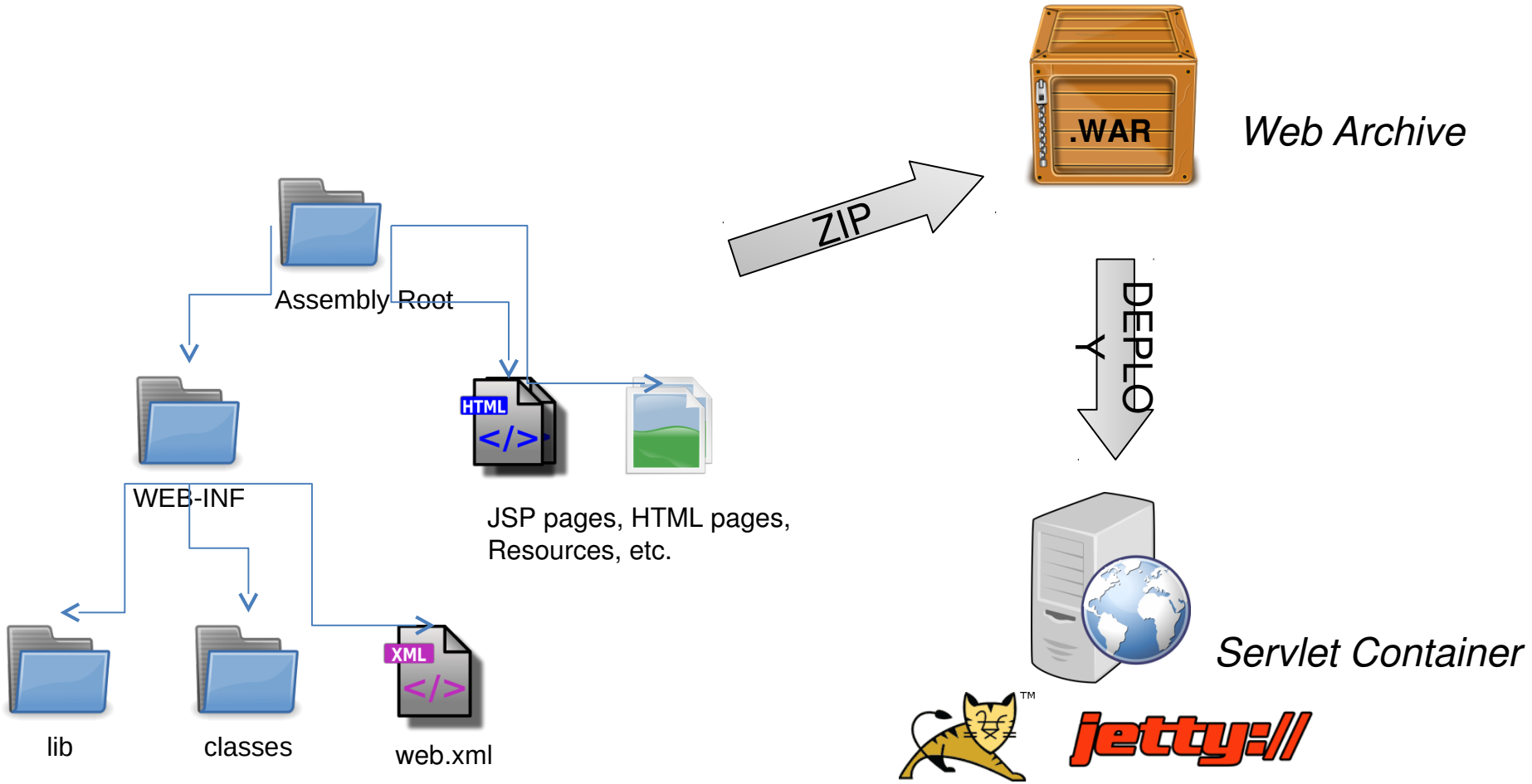
WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation=
        "http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="helloworld" version="3.0">
    <servlet>
        <servlet-name>hi</servlet-name>
        <servlet-class>examples.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 language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page import="java.util.*" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <body>
    <%! Date theDate = new Date();

        Date getDate() {
          return theDate;
        }
    %>
    Hello! The time is now <%= getDate() %>
  </body>
</html>
```

Deployment: Web Modules



Part IV. Wicket

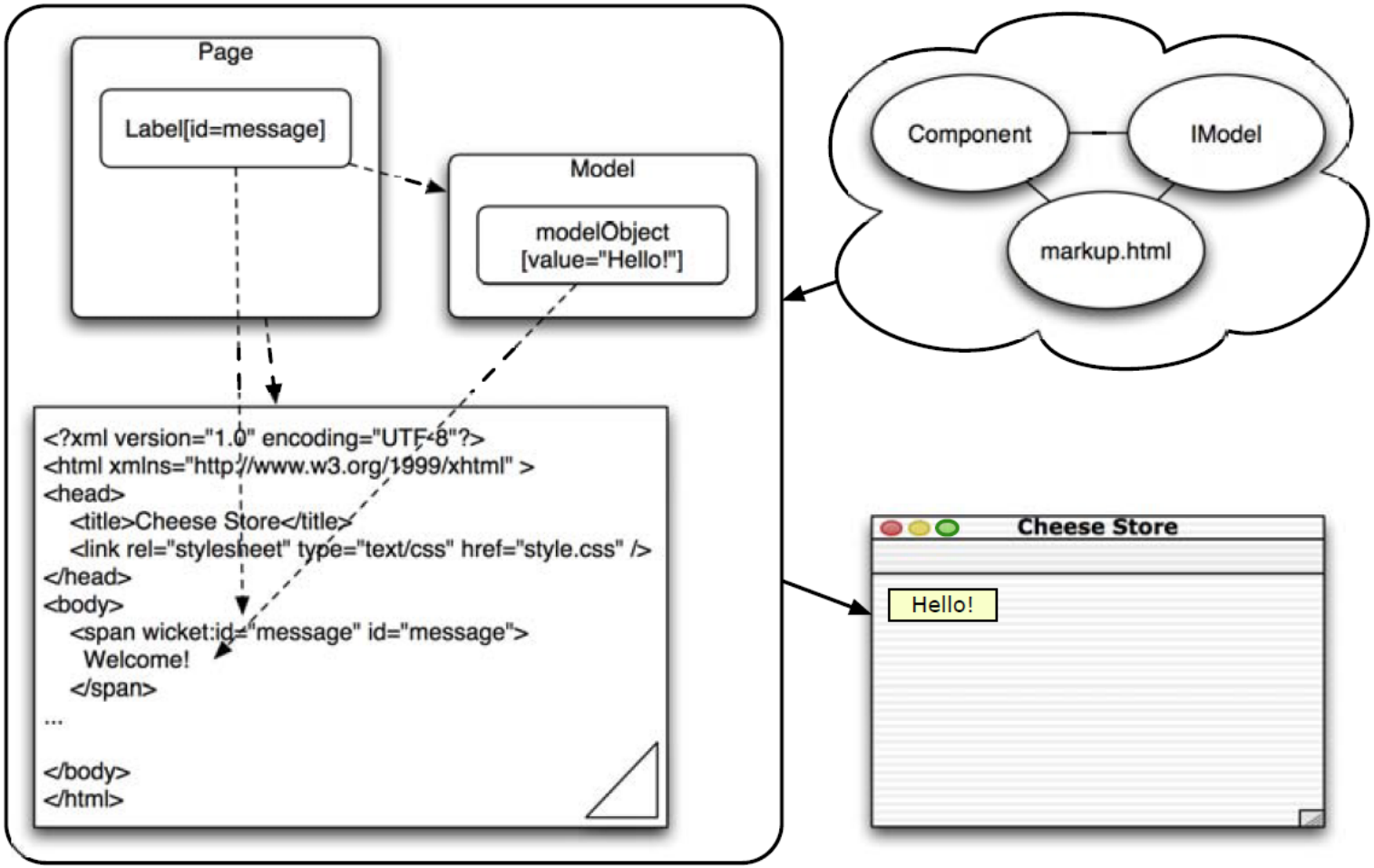
State handling

- 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

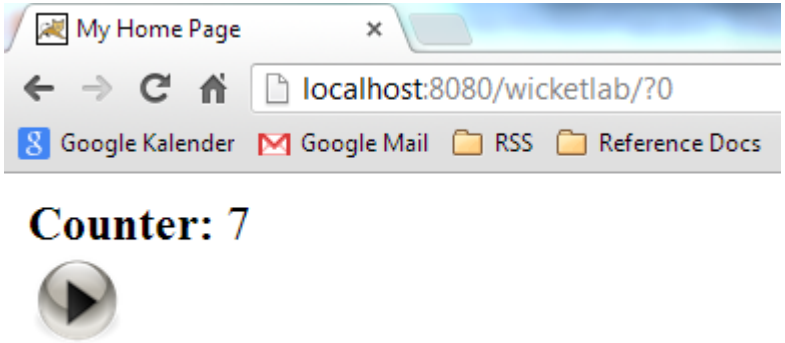
- Plain Java
  - Regular Java OOP that feels like Swing/SWT
  - Reusable widgets by inheritance and composition
  - Full IDE support
  - Refactoring
- Plain HTML
  - "Wicket doesn’t just reduce the likelihood of logic creeping into the presentation templates—it eliminates the possibility altogether."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

The Wicket component triad



from Wicket in Action

A first Wicket example (1)



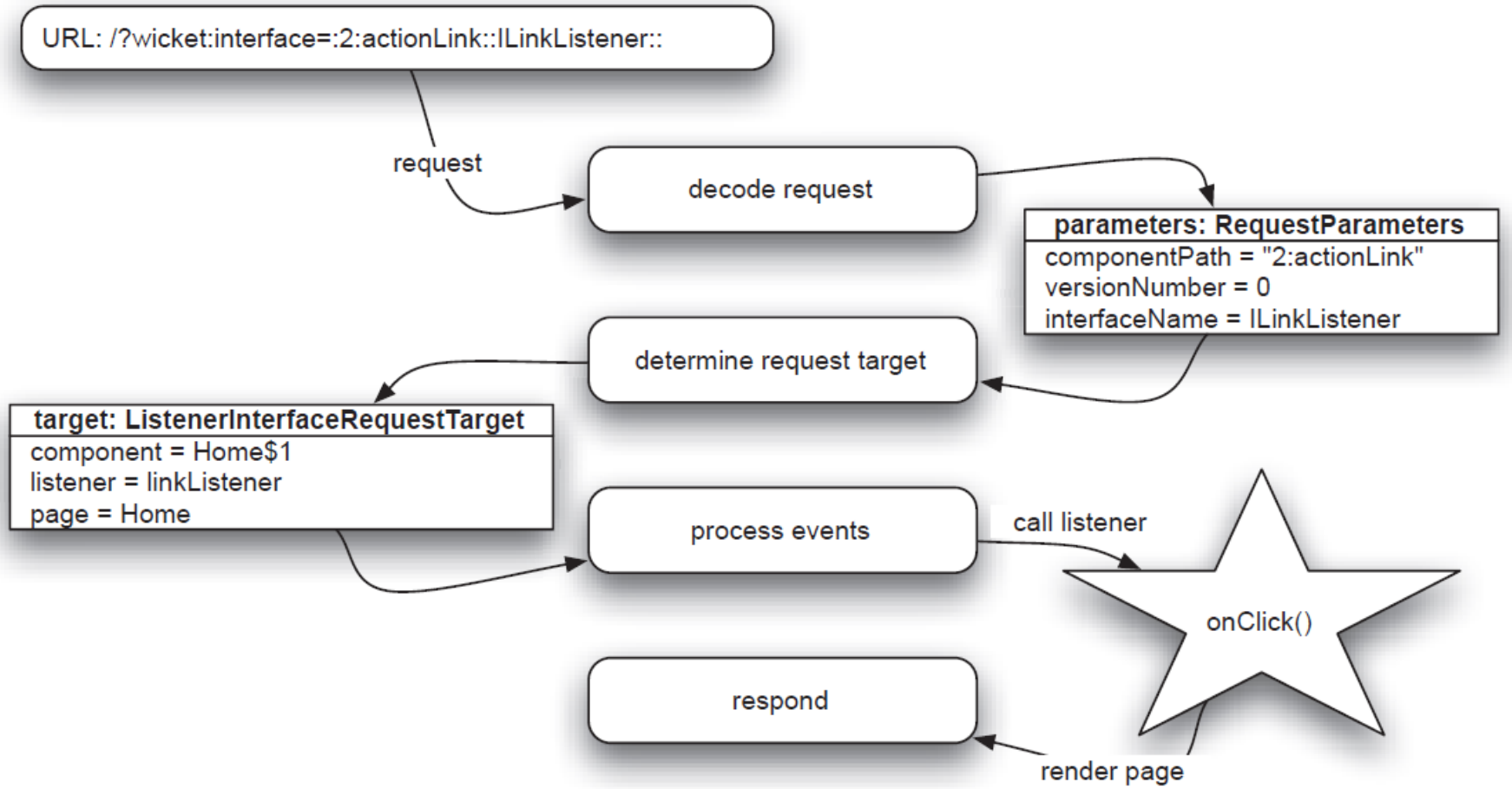
Source code in browser

```
<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<MyHomePage> getHomePage() {
        return MyHomePage.class;
    }
}
```

A glimpse behind the Wicket scenes (1)



from Wicket in Action*, not related to the example above*

A glimpse behind the Wicket scenes (2)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation=
        "http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="wicketlab" version="3.0">
  <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>
```

AJAX

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

- 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

Pure AJAX

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

Start typing a name in the input field below:

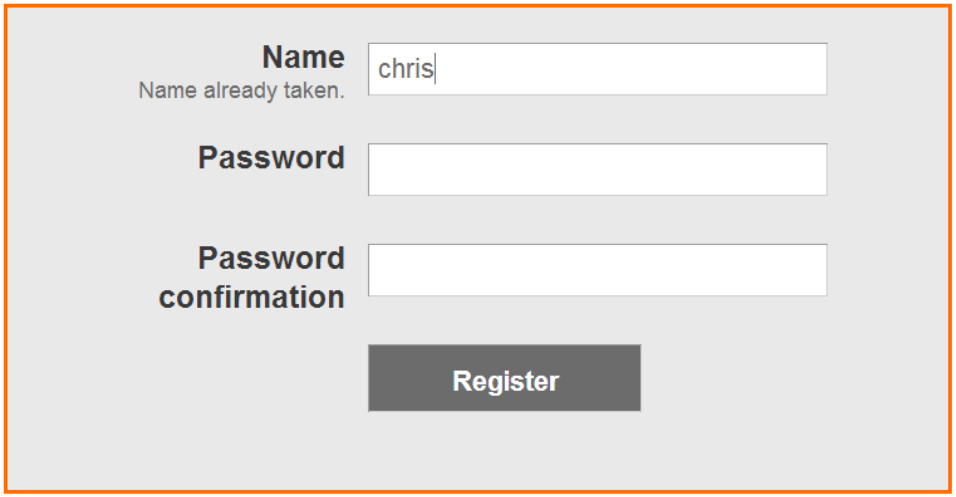
First name: El

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)

```
public Register() {

    // ...

    OnChangeAjaxBehavior onNameChange= new OnChangeAjaxBehavior() {
        @Override
        protected void onUpdate(AjaxRequestTarget target) {
            doNameFeedbackUpdate();
            target.add(fNameFeedback);
        }
    };

    fName.add(onNameChange);
}
```




```
private void doNameFeedbackUpdate() {
    String name= fName.getModelObject();
    if (getDatabase().nameTaken(name)) {
        fNameFeedback.setDefaultModelObject("Name already taken.");
    } else {
        fNameFeedback.setDefaultModelObject(" ");
    }
}
```

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 http://wicket.apache.org/

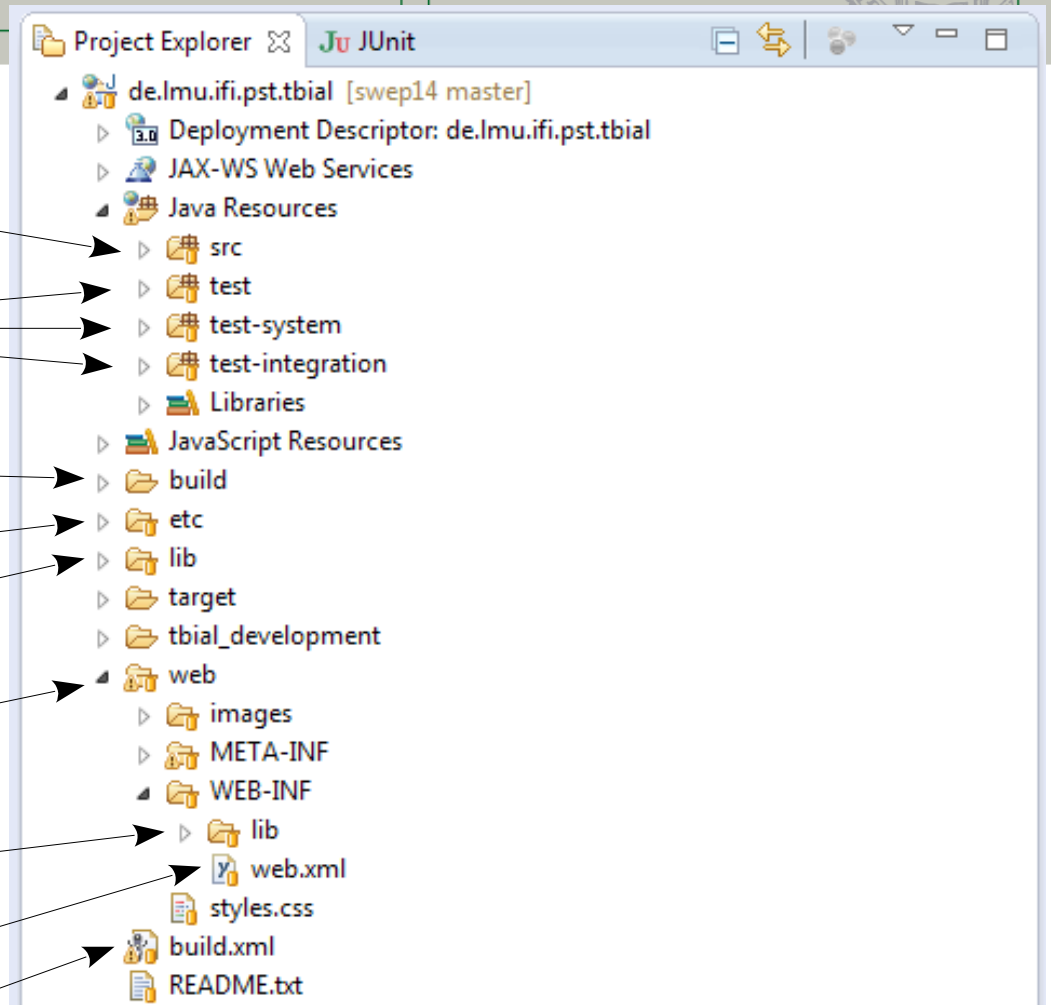
Part V. Skeleton Overview

Part V: Learning Targets

Learning Targets

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

Project structure



main source folder

test source folders

compiler output folder

config files etc.

development libraries (e.g. testing)

web root folder

application libraries (deployed)

deployment descriptor

Ant buildfile

Project structure (2)

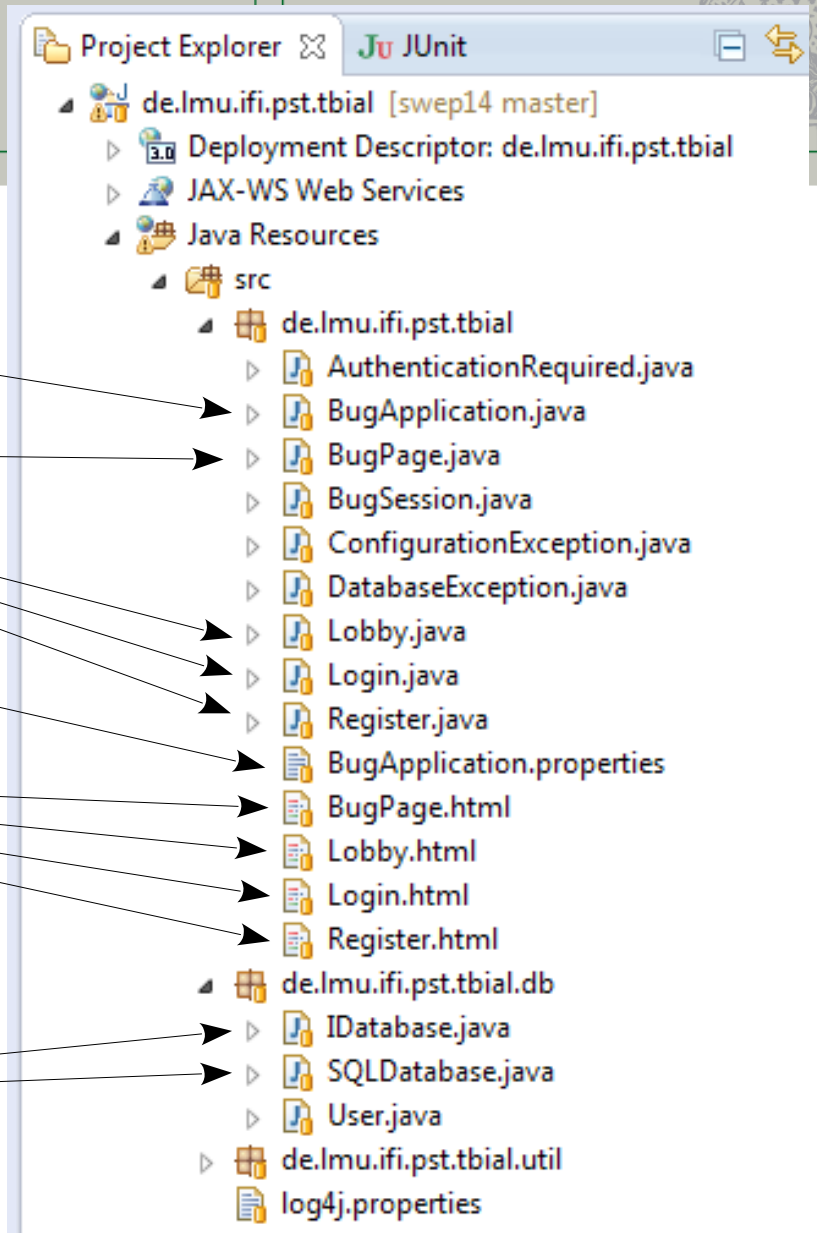
main application class

web page classes

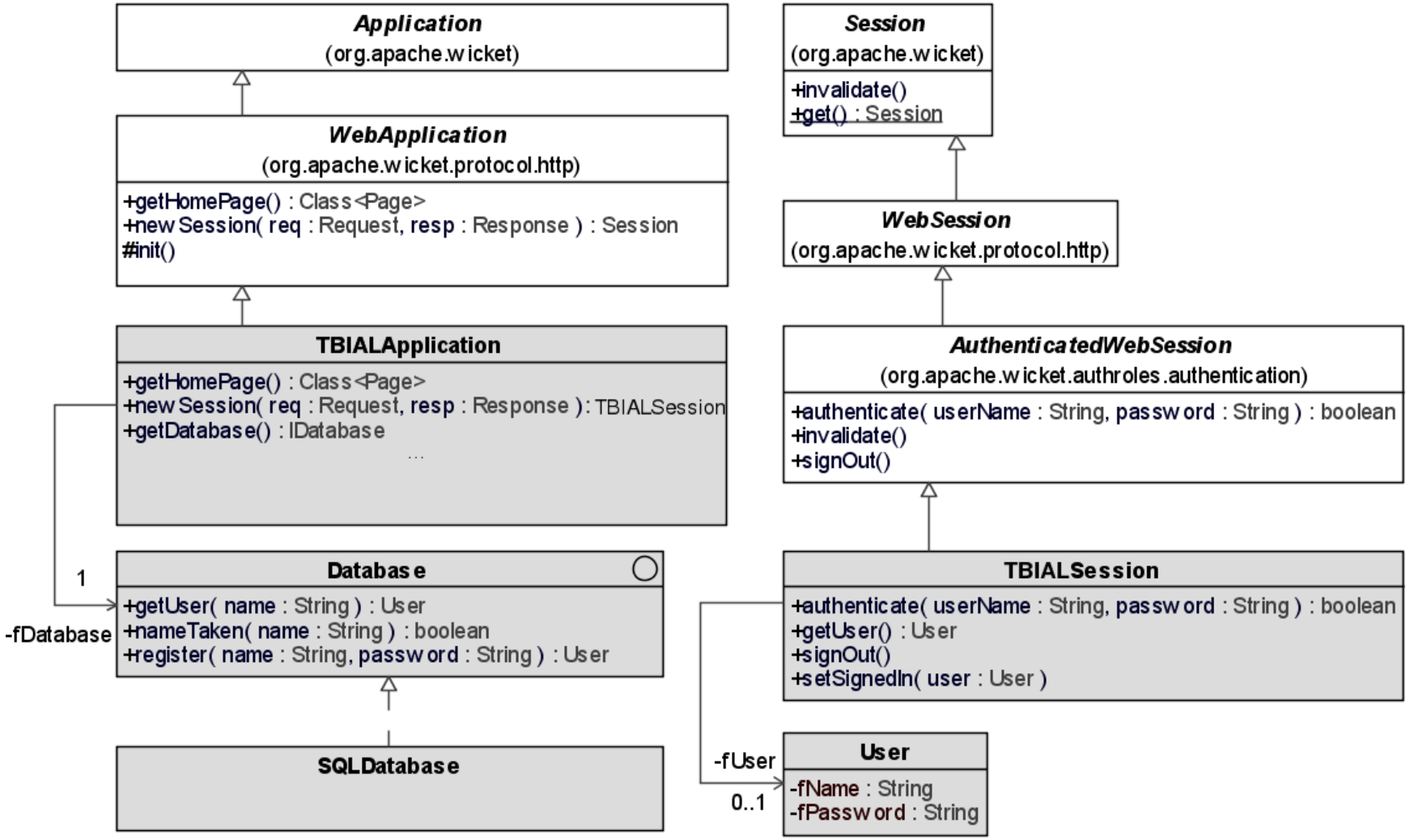
I18N messages

web page markup files

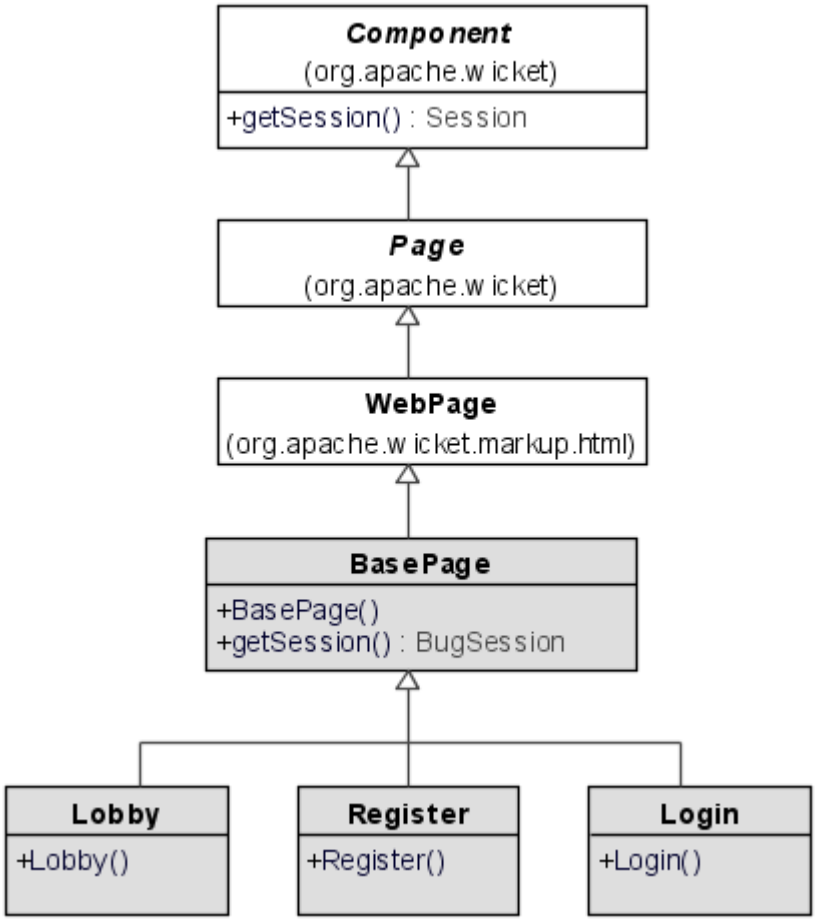
database access facade



Main classes in the skeleton



Web Pages with inheritance



BasePage.html

```
…
<div class="content">
   <wicket:child />
</div>
…
```

[^BasePage].html

```
…
<wicket:extend>
…Main Content…
</wicket:extend>
…
```

Authentication & Authorization

- **Authentication** is done in the `authenticate()` method of TBIALSession, 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 BugPage {
```

Application.java

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.
- PostgreSQL is used for staging.

… 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 J

You need to know the code base!

Part VI. Testing:
JUnit, Mockito, WicketTester

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

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!
- 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) for automated web page tests without starting a server

```
@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"));
}
```

Summary

Summary

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

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

V. **Skeleton Overview**
- Project structure
- Authentication

VI. **Testing**
- Junit, Mockito
- WicketTester

Rules and Task

Task to work on

User Counter **Prio: 20**

As a player, I want to know how many other players are currently online so that I can see how large the current player base is.

The counter should be displayed in the footer, and should be updated every time a player logs in or out.

- Select a peer for code review
- Create your ticket for working on the task (use version “Programming Exercise”)
- Create your solution in your own code branch
- Review the code of your peer until May 15th

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Technology Background

Development environment, Skeleton and Libraries

Slides by Prof. Dr. Matthias Hölzl
(based on material by Dr. Andreas Schroeder and Christian Kroiß)

Outline

Lecture 1

I. Eclipse
II. Git

Lecture 2

III. Java Web Applications
IV. Wicket (and AJAX)
V. TBIAL Skeleton Overview
VI. Testing

Part III. Java Web Applications

Java Servlets: CGI for Java

```java
package examples;

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

public class HelloWorld extends HttpServlet {

    private static final long serialVersionUID= 1L;

    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><h1>Hello World</h1></body>");
        out.println("</html>");
    }
}
```

Deployment Descriptors

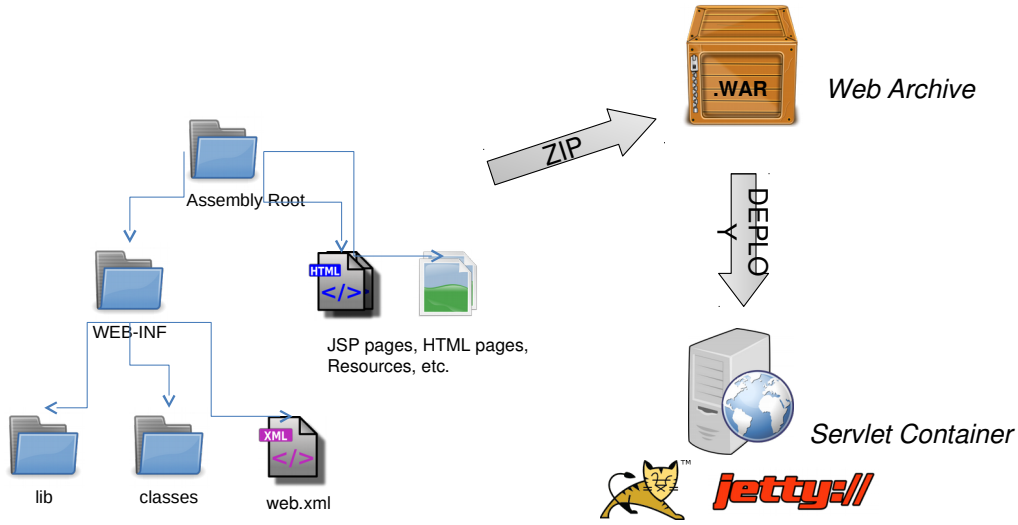
WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation=
        "http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="helloworld" version="3.0">
    <servlet>
        <servlet-name>hi</servlet-name>
        <servlet-class>examples.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 language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page import="java.util.*" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <body>
    <%! Date theDate = new Date();

        Date getDate() {
          return theDate;
        }
    %>
    Hello! The time is now <%= getDate() %>
  </body>
</html>
```

Deployment: Web Modules



Part IV. Wicket

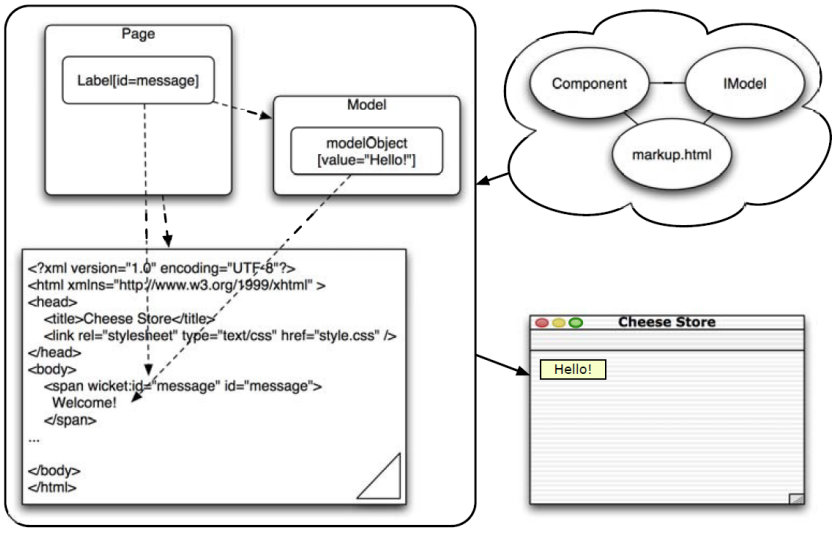
State handling

- 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.
- è *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

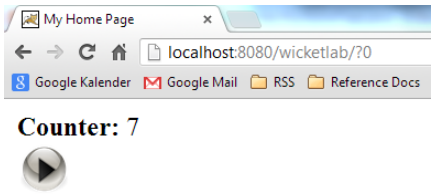
The Wicket component triad



from Wicket in Action

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

A first Wicket example (1)



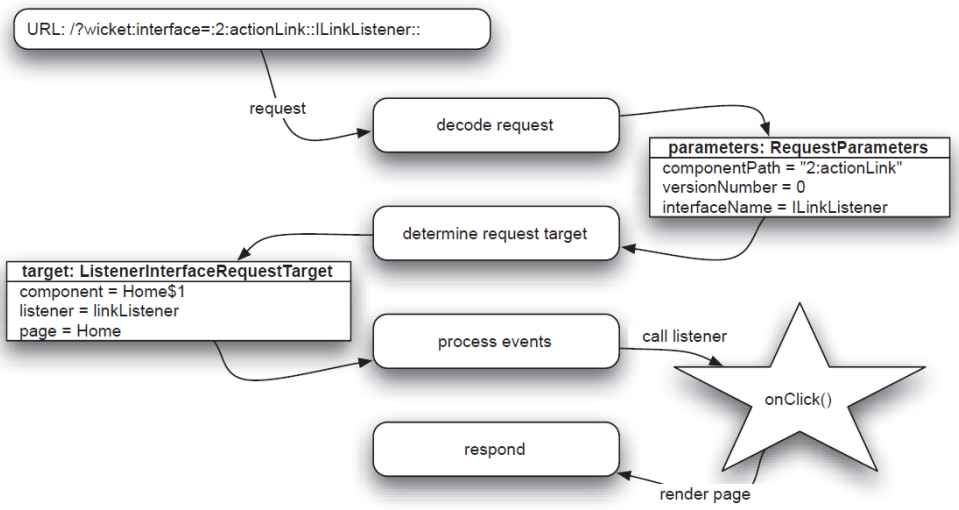
Source code in browser

```
<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<MyHomePage> getHomePage() {
        return MyHomePage.class;
    }
}
```

A glimpse behind the Wicket scenes (1)



from Wicket in Action*, not related to the example above*

A glimpse behind the Wicket scenes (2)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation=
        "http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="wicketlab" version="3.0">
  <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

- 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

Pure AJAX

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

Start typing a name in the input field below:

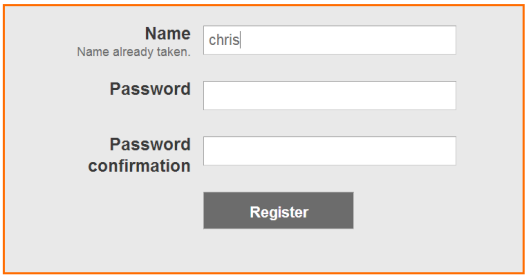
First name: E

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)

```
public Register() {

    // ...

    OnChangeAjaxBehavior onNameChange= new OnChangeAjaxBehavior() {
        @Override
        protected void onUpdate(AjaxRequestTarget target) {
            doNameFeedbackUpdate();
            target.add(fNameFeedback);
        }
    };

    fName.add(onNameChange);
}



private void doNameFeedbackUpdate() {
    String name= fName.getModelObject();
    if (getDatabase().nameTaken(name)) {
        fNameFeedback.setDefaultModelObject("Name already taken.");
    } else {
        fNameFeedback.setDefaultModelObject(" ");
    }
}
```



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 http://wicket.apache.org/

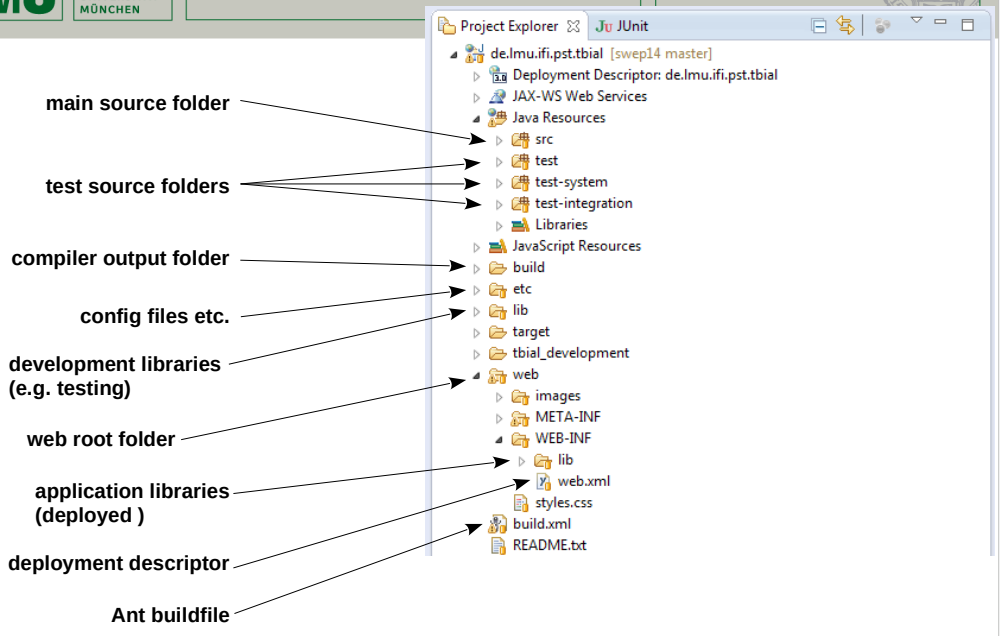
Part V. Skeleton Overview

Learning Targets

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

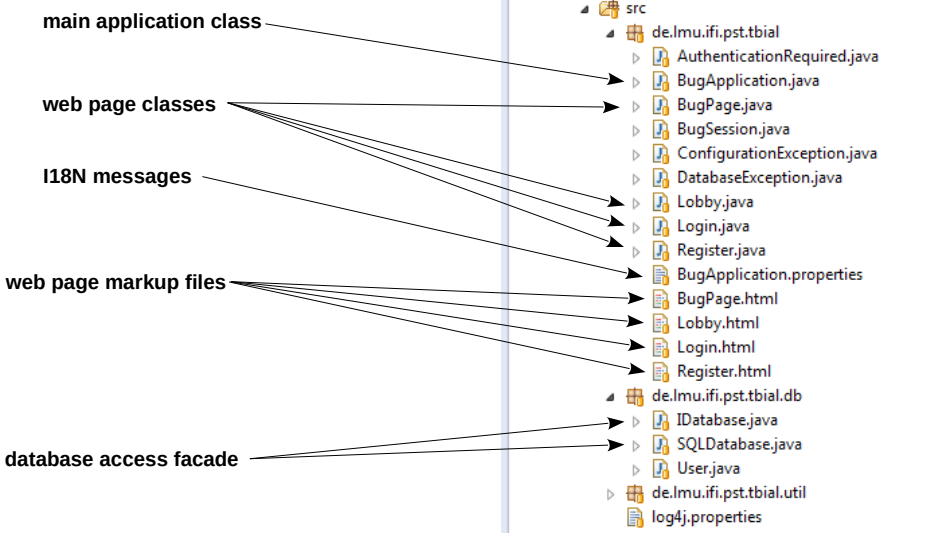
LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Project structure



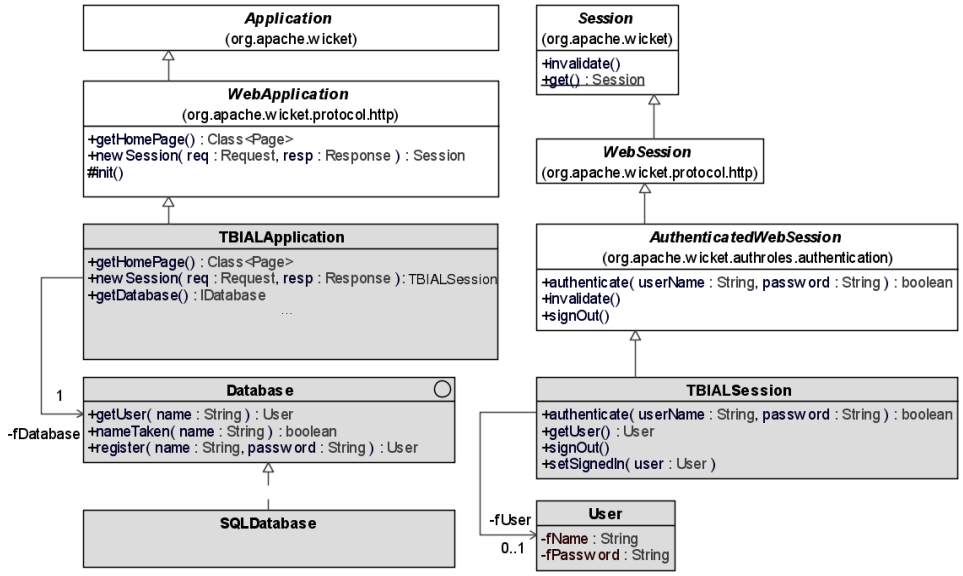
LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Project structure (2)

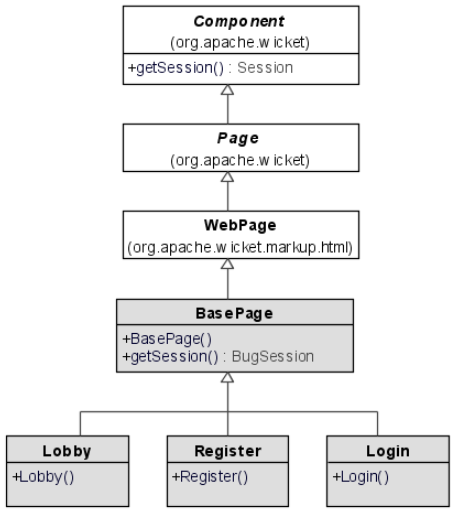


LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Main classes in the skeleton



Web Pages with inheritance



BasePage.html

```
…
<div class="content">
   <wicket:child />
</div>
…
```

[^BasePage].html

```
…
<wicket:extend>
…Main Content…
</wicket:extend>
…
```

- **Authentication** is done in the `authenticate()` method of TBIALSession, 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 BugPage {
```

Application.java

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.
- PostgreSQL is used for staging.

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 J

You need to know the code base!

Part VI. Testing:
JUnit, Mockito, WicketTester

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

→ Soweit so einfach, aber warum denn set up und tear down?

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

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

→ Fixture-code kann ziemlich umfangreich werden. Um zu vermeiden dass er überbordet gibt es Mockito.

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Mockito

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();
```

→ Test-Struktur ist jetzt klar, fixture-support haben wir jetzt auch. Welche Richtlinien gibt es für das Testen?

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Unit Test Best Practices

Testing best practices

- Test **behavior**, not methods;
  Behaviors are paths through code!
- 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

→ Das war's zum Thema testen, jetzt zum nächsten großen Thema: UI.

Testing Wicket Pages with WicketTester

- Use WicketTester (integrated in Wicket) for automated web page tests without starting a server

```
@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"));
}
```

Summary

Summary

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

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

V. **Skeleton Overview**
- § Project structure
- § Authentication

VI. **Testing**
- § Junit, Mockito
- § WicketTester

Rules and Task

Task to work on

User Counter **Pts: 20**

As a player, I want to know how many other players are currently online so that I can see how large the current player base is.

The counter should be displayed in the footer, and should be updated every time a player logs in or out.

- Select a peer for code review
- Create your ticket for working on the task (use version “Programming Exercise”)
- Create your solution in your own code branch
- Review the code of your peer until May 15th