

# Testdriven webbutveckling

- x Hur man kan använda ett **webbframverk** tillsammans med ett **mockramverk** så att man kan ta fram en webbapplikation
  - x utan att
- x skriva serverkod, driftsätta den eller konfigurera en databas.

# Innehåll

- x Testdriven?
- x Kort introduktion till Wicket
- x Kodning
- x Övning

# Testdriven?

- x Tester skrivna först
  - x Ger bättre gränssnitt
  - x Specificerar design
  - x Blir av
- x Tester skrivna efteråt
  - x Blir inte av
- x Automatgenererade tester
  - x Korkade men kan träffa rätt ändå



# Wicket

x Vi börjar med webbramverket

# Webframverk, alternativen

Echo Cocoon Millstone OXF  
Struts SOFIA Tapestry WebWork  
RIFE Spring MVC Canyamo Maverick  
JPublish JATO Folium Jucas  
Verge Niggle Bishop Barracuda  
Action Framework Shocks TeaServlet wingS  
Expresso Bento jStatemachine jZonic  
OpenEmcee Turbine Scope Warfare  
JWAA Jaffa Jacquard Macaw  
Smile MyFaces Chiba JBanana  
Jeenius JWarp Genie Melati  
Dovetail Cameleon JFormular Xoplon  
Japple Helma Dinamica WebOnSwing  
Nacho Cassandra Baritus Stripes  
Click GWT

# Wicket Goals

- x EASY (SIMPLE / CONSISTENT / OBVIOUS)
- x REUSABLE
- x NON-INTRUSIVE
- x SAFE
- x EFFICIENT / SCALABLE
- x COMPLETE
  - x Taget från deras egen sida – läs notsidan!

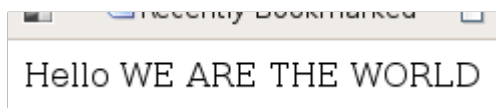
# Filosofi

- x Mer objekt-orienterat webbramverk
  - x Sidor är instansierade objekt
  - x Stöd för arv mellan sidor
    - x Typiskt en bassida för hela sajten
- x Java står för logiken
- x HTML för presentationen
  - x Redigerbar med t ex DreamWeaver
- x Javastruktur avspeglar sidstruktur
  - x Ex: ett formulär består av fält

# Java och HTML

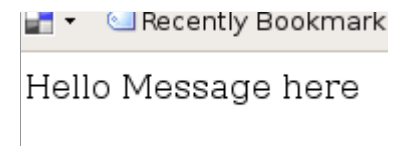
## x MyPage.java

```
x public class MyPage extends
  WebPage {
  x public MyPage(String id)
    {
    x super(id);
    x add(new
      Label("message", "WE
        ARE THE WORLD"));
    x }
  x }
```



## x MyPage.html

```
x <html>
  x <head/>
  x <body>
    x <p>
      x Hello <span
        wicket:id="message">
        Message here</span>
    x </p>
  x </body>
x </html>
```



# Do the Quick Start

- x Maven installerat
- x Repository i ~/.m2/settings.xml
- x <http://wicket.apache.org/quickstart.html>
- x cd blog
- x Editera pom.xml så att Junit blir 4.4
- x mvn eclipse:eclipse -DdownloadSources=true
- x Fixa M2\_REPO i Eclipse (~/.m2/repository)
- x Kör Start.java
- x <http://localhost:8080/>

# Notera

- x En testklass i src/test/java: TestHomePage
- x En HTML-sida och en Javaklass med samma namn: HomePage
- x HTML-sidan ren, bortsett från “wicket:id”

# Demo: en blogg

- x Dagens exempel är en blogg
- x Börja med att skapa en sida där man kan skriva ett inlägg

# Demo: en ny sida

- x En ny sida

```
x @Test
x public void shouldHaveCreatePostPage() {
x     tester.startPage(CreatePostPage.class);
x }
```

- x CreatePostPage är en ny klass som ärver WebPage

- x Kör testet, notera: “Markup of type 'html' for component ...”

- x Skapa CreatePostPage.html – green bar

# Demo: Webbläsaren

- x Plocka fram webbläsaren
  - x <http://localhost:8080/>
- x Men nu är vi ju på den torra startsidan ...
  - x <http://localhost:8080/CreatePostPage>
  - x Fungerar inte!
- x Vi måste lägga navigeringen i Java
  - x (Det finns bookmarkable page - överkurs)

# Testa efter länken

x HomePage ska ha en länk till CreatePost

```
x public void testShouldHaveLinkToCreatePostPage() {  
    x tester.startPage(HomePage.class);  
  
x  
  
    x HomePage homePage = (HomePage)  
      tester.getLastRenderedPage();  
  
    x Link link = (Link) homePage.get("create_post");  
  
    x assertNotNull(link);  
  
x }
```

x Red bar!

# En länk

## x Lägg till i HomePage.html

```
x <a href="#" wicket:id="create_post">Create a blog post</a>
```

## x Fortfarande red bar

## x Lägg till i HomePage.java

```
x add(new Link("create_post") {  
    x @Override  
    x public void onClick() {  
        x setResponsePage(CreatePostPage.class);  
    x }  
x });
```

## x Prova i webbläsaren igen!

30 min

# Vi ska kunna skapa ett inlägg

x Ett test verifierar att det finns ett inmatningsfält

x `@Test`

x `public void` `shouldCreatePost()` {

x `tester.startPage(CreatePostPage.class);`

x `CreatePostPage page = (CreatePostPage)`  
`tester.getLastRenderedPage();`

x `Form createPostForm = (Form) page.get("create_post_form");`

x `assertNotNull(createPostForm);`

x Red bar: Lägg till formuläret “create\_post\_form”

x `<form wicket:id="create_post_form">`

x `</form>`

# Formulär i klassen

x **Notera:** "Unable to find component ..."

x **Lägg till ett formulär i klassen**

```
x public class CreatePostPage extends WebPage {
```

```
x
```

```
x     public CreatePostPage() {
```

```
x         add(new CreatePostForm("create_post_form"));
```

```
x     }
```

```
x }
```

x **Definiera CreatePostForm som en inre klass**

# Formulärsklassen

```
x public class CreatePostForm extends Form {  
    x private static final long serialVersionUID = 1L;  
x  
    x public CreatePostForm(String id) {  
        x super(id);  
    x }  
x }
```

x Green bar

# Lite fält till formuläret

x Testa att det finns två fält, ett för rubrik och ett för innehållet

x ...

```
x TextField titleField = (TextField)
createPostForm.get("title");
```

```
x assertNotNull(titleField);
```

```
x TextArea postArea = (TextArea) createPostForm.get("post");
```

```
x assertNotNull(postArea);
```

x Red bar...

x

# Fixa fälten

## x HTML

x `<p>`

x Title: `<input wicket:id="title" size="60" />`

x `</p>`

x `<p>`

x Post: `<textarea rows="24" cols="80" wicket:id="post" />`

x `</p>`

## x Java

x `add(new TextField("title"));`

x `add(new TextArea("post"));`

## x Green!

# Att testa ett formulär

```
x FormTester ft = tester.newFormTester("create_post_form");  
x ft.setValue("title", "TEST of title jada jada");  
x ft.setValue("post", "Oh la la IT is a test");  
x ft.submit();
```

x **Kör testet – kraschar fult:**

x ...

```
x Caused by: java.lang.IllegalStateException: Attempt to set  
model object on null model of component:  
create_post_form:title
```

x **Dags att berätta om modeller i Wicket**

# I, Model

- x IModel är gränssnittet
- x Modeller hanterar ett objekt – ditt domänobjekt

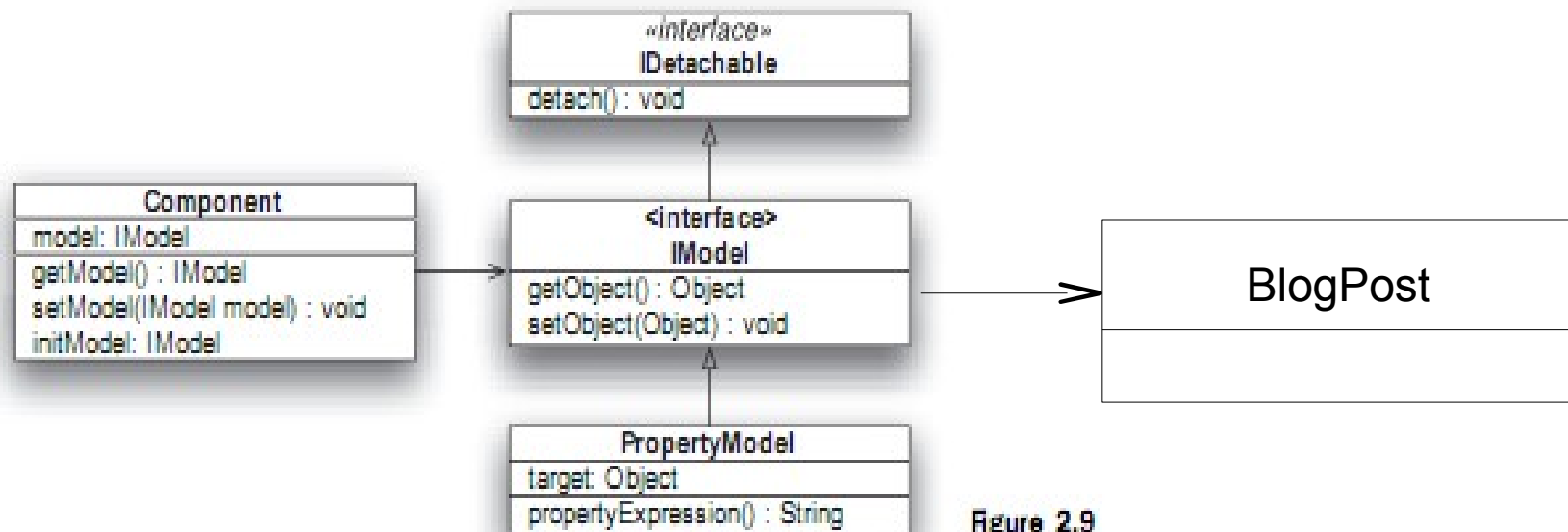


Figure 2.9  
The IModel interface

# BlogPost

x Låt oss skapa ett domänobjekt – BlogPost

```
x public class BlogPost implements Serializable{  
    x private String post;  
    x private String title;  
    x ... getters and setters
```

x Sedan använder vi det i formuläret

```
x private BlogPost blogPost = new BlogPost();  
x public CreatePostForm(String id) {  
    x super(id);  
    x setModel(new CompoundPropertyModel(this.blogPost));
```

x Green!

# CompoundPropertyModel

- x Koppla domänobjektet till komponenten med modellen

```
x setModel(new CompoundPropertyModel(this.blogPost));
```

```
x add(new TextField("title"));
```

```
x add(new TextArea("post"));
```

- x **HTML**

```
x <input wicket:id="title" size="60" />
```

```
x <textarea wicket:id="post" rows="24" cols="80">
```

# Ett inlägg som skapas visas

x Testa att vi visar ett blogginlägg som skapats

x `tester.assertRenderedPage (ShowPostPage.class) ;`

x `junit.framework.AssertionFailedError: expected:<ShowPostPage>  
but was:<CreatePostPage>`

x Antagligen borde vi göra något på submit

x `@Override`

x `protected void onSubmit() {`

x  `setResponsePage (ShowPostPage.class) ;`

x `}`

x Med ShowPostPage.html och dito java ...

# Bara visa att det visas

## x Verifiera att vårt inlägg visas

```
x ShowPostPage showPage = (ShowPostPage)  
  tester.getLastRenderedPage();
```

```
x Label titleLabel = (Label) showPage.get("title");
```

```
x assertNotNull(titleLabel);
```

```
x Label postLabel = (Label) showPage.get("post");
```

```
x assertNotNull(postLabel);
```

```
x
```

```
x
```

# Fixa visningen!

x Fixa så att sidan vet vad den ska visa

```
x protected void onSubmit() {  
    x ShowPostPage spp = new ShowPostPage(blogPost);  
    x setResponsePage(spp);  
x }
```

x Se till att sidan visar det den får

```
x public ShowPostPage(BlogPost blogPost) {  
    x setModel(new CompoundPropertyModel(blogPost));  
    x add(new Label("post")); // and the HTML  
    x add(new Label("title")); // and the HTML  
x }
```

60 min

# Testa själv

- x <http://localhost:8080/>
- x Fungerar ju
- x Nu är det dags att mocka
- x Va?

# Mocka är inte skit

- x "Mock object" står för ett objekt som simulerar ett verkligt objekt
- x Vi vill ha det för att vänta med serverkod
- x ... av något skäl
- x ... testa
- x ... demo
- x

x Bild från Wikipedia



# Mockito is the shit

- x Mockito liknar JMock och EasyMock

- x Enklare att använda.

- x "mock" skapar ett mockobjekt

- x `List<String> listMock = mock(List.class);`

- x "stub" beskriver hur det beter sig – om man vill

- x `stub(listMock.get(0)).toReturn("nollan");`

- x "verify" kontrollerar att hur det använts

- x `verify(listMock).get(0);`

- x Igen: om man vill!

# När man skapar blogg ...

- x Skapar man ett blogginlägg så ska servern anropas
- x Vi skapar en tjänst "BlogService" som hittas via "BlogApplication"
- x Tjänsten ska användas av CreatePostPage när den skapar ett nytt inlägg.
- x Vi mockar upp en sådan tjänst

# Mockito läggs till pom.xml

```
x <dependency>  
  x <groupId>org.mockito</groupId>  
  x <artifactId>mockito-all</artifactId>  
  x <version>1.5</version>  
x </dependency>
```

## x Ny Maven till Eclipse

```
x > mvn eclipse:clean eclipse:eclipse
```

# Testa att servern anropas

x Vi har ingen server men klienten ska använda den ändå

x `@Test`

x `public void shouldCallServer() {`

x `BlogService blogServiceMock = mock(BlogService.class);`

x `BlogApplication.setBlogService(blogServiceMock);`

x `tester.startPage(CreatePostPage.class);`

x Skapa inlägg med formuläret som förut, sedan

x `verify(blogServiceMock).createPost(eq(expectedBlogPost));`

# Förväntat: en röding, såklart!

```
x org.mockito.exceptions.verificatio.WantedButNotInvoked:  
x Wanted but not invoked:  
x blogService.createPost(  
x     se.crisp.rd.BlogPost@12940b3  
x );
```

## x Ny version av onSubmit

```
x protected void onSubmit() {  
x     BlogService service = BlogApplication.getBlogService();  
x     service.createPost(blogPost);  
x     ShowPostPage spp = new ShowPostPage(blogPost);  
x     setResponsePage(spp);  
x }
```

# 90 min

x Efter pausen gör vi en tabell

# Hur gör man tabeller?

- x Om nu HTML ska vara statiskt och Java stå för logiken – hur skapar man tabeller och listor?
- x Svar: logiken ligger i Java (AbstractRepeater)
- x Antag att vi vill ha en sida med alla blogginlägg
- x `public class TestListPosts {`
- x `...`
- x `public void shouldListBlogPosts() {`
  - x `tester.startPage(ListBlogPostsPage.class);`
- x Kommer den inte att hämta alla inlägg från servern? Jo! `GetPosts()` - here we go!

# Låtsas att servern finns

x (fortsättning från föregående sida)

```
x List<BlogPost> testData = new ArrayList<BlogPost>();
```

```
x stub(blogServiceMock.getPosts()).toReturn(testData);
```

```
x
```

```
x verify(blogServiceMock).getPosts();
```

x **Skjut in testdata**

```
x BlogPost blogPost = new BlogPost();
```

```
x blogPost.setTitle("Da must hat lieben geworden");
```

```
x blogPost.setPost("Ich hat ein traum");
```

```
x testData.add(blogPost);
```

```
x BlogPost blogPost2 = new BlogPost();
```

```
x ...
```

# Implementera hämtning

```
x public ListBlogPostsPage() {  
    x super();  
    x BlogService service = BlogApplication.getBlogService();  
    x List<BlogPost> posts = service.getPosts();  
x }
```

x **Green! → Mera tester!**

```
x ListBlogPostsPage listPage = (ListBlogPostsPage)  
tester.getLastRenderedPage();  
x Component titleLabel1 = listPage.get("post_list:0:title");  
x assertNotNull(titleLabel1);
```

x **Red! → Implementera!**

# Knepigt?

## x PropertyListView är en "Repeater"

```
x add(new PropertyListView("post_list", posts) {  
  x @Override  
  x protected void populateItem(ListItem item) {  
    x item.add(new Label("title"));  
    x item.add(new MultiLineLabel("post"));  
  x }  
}
```

## x Lite HTML

```
x <div wicket:id="post_list">  
x <h3 wicket:id="title">Title</h3>  
x <p wicket:id="post">Post goes here.</p>  
x </div>
```

# Produktägaren kommer!

x Vi behöver en demo!



120 min

# Demodag

- x Vi kan skapa bloggar – men vi måste ju visa det
- x Kopiera Start.java till Demo.java
- x Vi använder Mockito för att simulera en server som tar emot och lagrar våra inlägg

# Bygg en "server"

```
x private static void initApplikationMocks() {  
    x BlogService blogServiceMock = mock(BlogService.class);  
    x BlogApplication.setBlogService(blogServiceMock);  
    x doAnswer(new Answer() {  
        x public Object answer(InvocationOnMock invocation) {  
            x BlogPost post = (BlogPost) invocation.getArguments()  
              [0];  
            x posts.add(post);  
            x return null;  
        x }  
    x }) .when(blogServiceMock).createPost((BlogPost) notNull());  
    x stub(blogServiceMock.getPosts()).toReturn(posts);  
x Testa http://localhost:8080
```

# Övningar

- x Välj en övning – säg högt vilken
- x Parprogrammera och jobba **testdrivet**
- x Övningarna bygger på det som gjorts
  - 1) Inloggning med skyddade sidor
  - 2) Använd Feedback för att kontrollera inmating
  - 3) Använd AjaxEditableLabel för att ändra ett inlägg
  - 4) Använd AjaxCompleteTextField för sökning
  - 5) Skapa en Panel med en klocka som uppdateras
  - 6) Skapa en bassida som ger sajtens utseende

# Övning: Inloggning med skyddade sidor

x Skapa en sida där man kan logga in

x Ärv `WebSession` och lägg in användaren där

```
x public class BlogSession extends WebSession {  
    x private User user = null;
```

x Definiera om i `WicketApplication`

```
x @Override
```

```
x public Session newSession(Request request, Response response)  
    {  
  
    x return new BlogSession(request);  
  
    x }
```

# Övning: Använd Feedback

- x Lägg till en FeedbackPanel på CreatePostPage
- x Ändra så att Title blir "RequiredTextField"

# Övning: Använd AjaxEditableLabel

- x Ändra så att man kan ändra ett inlägg "inline".

# Övning: Använd AjaxCompleteTextField

- x För varje text som matas in, indexera upp orden
- x Skapa en söksida som ger förslag på sökbegrepp vartefter man skriver

# Övning: Skapa en Panel

- x Panel är grunden för återanvändbara "widgets" i Wicket
- x Istället för en hel sida så bygger du upp en Panel i HTML och Java som distribueras i en Jar
- x Skapa en Panel med en kalender som man stoppa in på en sida
- x Använd `<wicket:panel>`

# Länkar

- x Hemsidan: <http://wicket.apache.org/>
- x Javadoc API: <http://wicketstuff.org/wicket13doc>
- x Exempel: <http://wicketstuff.org/wicket13>
- x Blog: <http://chillenious.wordpress.com/category/wicket>
- x Bok: Wicket in Action, <http://manning.com/dashorst/>
- x Snabbstart: <http://wicket.apache.org/quickstart.html>
- x Mockito: <http://code.google.com/p/mockito/>

# Extra Material

# How do we handle log in?

- x Typically, some pages require users to log in
- x If they try to reach a protected page, they should need to log in and *then* reach the page
- x It is called interception in Wicket
- x `public void shouldInterceptWhenNotLoggedIn() {`
  - x Test that we reach the log in page
- x `public void shouldNotInterceptWhenLoggedIn() {`
  - x Test that we reach the create post page
- x `public void shouldReachPageAfterLogIn() {`
  - x Test that we reach log in and then create post page

# The login form is ordinary

## x But we use a method on Component

x `@Override`

x **protected void** `onSubmit()` {

x `// Check that the log in is ok`

x `// if ...`

x `BlogSession session = (BlogSession) getSession();`

x `session.setUser(user);`

x **if** `(!continueToOriginalDestination())` {

x `setResponsePage(getApplication().getHomePage());`

x `}`

## x Destination is charged earlier by ...

# CreatePost guards itself

```
x public CreatePostPage() {  
    x if (!(BlogSession) getSession()).isAuthenticated() {  
        x redirectToInterceptPage(new LoginPage());  
    x }  
    x add(new CreatePostForm("create_post_form"));  
x }
```

# Intercept to handle log in

x As usual, credentials are stored in the session

x But we subclass it to get it type safe

```
x public class BlogSession extends WebSession {  
    x private User user = null;  
    x public BlogSession(Request request) { super(request); }  
    x public static BlogSession get() {  
        x return (BlogSession) WebSession.get();  
    x }  
    x public boolean isAuthenticated() { return (user != null); }  
x }
```

# Komponenter i Wicket

- x Består av
  - x Presentation i HTML
  - x Logik i Java
- x Packas i en jar
  - x ... nås via classpath

# Demo: en komponent

- x Vi vill skapa en klocka som visar tiden, hela tiden

- x Så här skulle vi vilja använda den:

- x `<div style="font-size: 24">`
  - x `<span wicket:id="clock">CLOCK</span>`
- x `</div>`

- x I Java:

- x `add(new ClockPanel("clock"));`

# Demo: klockpanelen

## \* Gör en panel i HTML (ClockPanel.html)

```
* <html>
  * <head></head>
  * <body>
    * <wicket:panel>
      * <span wicket:id="clock">Clock</span>
    * </wicket:panel>
  * </body>
* </html>
```

# Demo: Inkludera i panelen

## x HTML-koden backas upp med:

```
x public class ClockPanel extends Panel {  
    x public ClockPanel(String id) {  
        x super(id);  
        x Clock clock = new Clock("clock", TimeZone.getTimeZone("America/  
        Los_Angeles"));  
        x add(clock);  
        x clock.add(new  
        AjaxSelfUpdatingTimerBehavior(Duration.seconds(5)));  
    x }  
x }
```

# Demo: klockkoden

x Från <http://wicketstuff.org/wicket13/ajax/>

```
x public class Clock extends Label {
x     public Clock(String id, TimeZone tz) {
x         super(id, new ClockModel(tz));
x     }
x     private static class ClockModel extends AbstractReadOnlyModel {
x         private DateFormat df;
x         public ClockModel(TimeZone tz) {
x             df = DateFormat.getDateInstance(DateFormat.FULL, DateFormat.LONG);
x             df.setTimeZone(tz);
x         }
x         public Object getObject() {
x             return df.format(new Date());
x         }
x     }
x }
```

# Testdriven webbutveckling

- × Hur man kan använda ett **webbframverk** tillsammans med ett **mockramverk** så att man kan ta fram en webbapplikation
  - × utan att
- × skriva serverkod, driftsätta den eller konfigurera en databas.

# Innehåll

- x Testdriven?
- x Kort introduktion till Wicket
- x Kodning
- x Övning

# Testdriven?

- x **Tester skrivna först**
  - x Ger bättre gränssnitt
  - x Specificerar design
  - x Blir av
- x **Tester skrivna efteråt**
  - x Blir inte av
- x **Automatgenererade tester**
  - x Korkade men kan träffa rätt ändå



# Wicket

- x Vi börjar med webbramverket

# Webbramverk, alternativen

Echo Cocoon Millstone OXF  
Struts SOFIA Tapestry WebWork  
RIFE Spring MVC Canyamo Maverick  
JPublish JATO Folium Jucas  
Verge Niggle Bishop Barracuda  
Action Framework Shocks TeaServlet wingS  
Expresso Bento jStatemachine jZonic  
OpenEmcee Turbine Scope Warfare  
JWAA Jaffa Jacquard Macaw  
Smile MyFaces Chiba JBanana  
Jeenius JWarp Genie Melati  
Dovetail Cameleon JFormular Xoplon  
Japple Helma Dinamica WebOnSwing  
Nacho Cassandra Baritus Stripes  
Click GWT

# Wicket Goals

- x EASY (SIMPLE / CONSISTENT / OBVIOUS)
- x REUSABLE
- x NON-INTRUSIVE
- x SAFE
- x EFFICIENT / SCALABLE
- x COMPLETE
- x Target från deras egen sida – läs notsidan!

Per Lundholm, Crisp

6

- \* EASY (SIMPLE / CONSISTENT / OBVIOUS)
  - o POJO-centric
  - o **All code written in Java ala Swing**
  - o Minimize "conceptual surface area"
  - o **Avoid** overuse of **XML configuration** files
  - o Fully solve back button problem
  - o Easy to create bookmarkable pages
  - o Maximum **type safety** and compile-time problem diagnosis
  - o Maximum **diagnosis** of run-time problems
  - o Minimum reliance on special tools
  - o Components, containers and conventions should be **consistent**
- \* REUSABLE
  - o Components written in Wicket should be fully reusable
  - o Reusable components should be easily distributed in ordinary JAR files
- \* NON-INTRUSIVE
  - o **HTML** or other **markup not polluted** with programming semantics
  - o Only **one simple tagging** construct in markup
  - o Compatible with any ordinary HTML editor
  - o Easy for graphics designers to recognize and avoid framework tagging
  - o Easy to add tagging back to HTML if designers accidentally remove it
- \* SAFE
  - o Code is secure by default
  - o Only explicitly bookmarkable links can expose state in the page or URL
  - o All logic in Java with maximum type safety
  - o Easy to integrate with Java security
- \* EFFICIENT / SCALABLE
  - o **Efficient and lightweight, but not at the expense of other goals**
  - o Clustering through sticky sessions preferred
  - o Clustering via session replication is easy to accomplish and easy to tune by working with detachable models.
- \* COMPLETE
  - o The Wicket team is committed to deliver a feature complete, ready-to-use framework for developing Java web applications. The core framework was written and contributed by the author of this introduction, Jonathan Locke. The current team consists of a group of experienced programmers, some of which were active on some of the other frameworks stated above, and all of which have extensive experience building large scale Java web applications. We eat our own dogfood, and will thus work on Wicket from a framework user's perspective.

# Filosofi

- x Mer objekt-orienterat webbramverk
  - x Sidor är instansierade objekt
  - x Stöd för arv mellan sidor
    - x Typiskt en bassida för hela sajten
- x Java står för logiken
- x HTML för presentationen
  - x Redigerbar med t ex DreamWeaver
- x Javastruktur avspeglar sidstruktur
  - x Ex: ett formulär består av fält

# Java och HTML

## x MyPage.java

```
x public class MyPage extends
WebPage {
    x public MyPage(String id)
    {
        x super(id);
        x add(new
Label("message", "WE
ARE THE WORLD"));
    x }
x }
```

Recently Bookmarked

Hello WE ARE THE WORLD

## x MyPage.html

```
x <html>
    x <head/>
    x <body>
        x <p>
            x Hello <span
wicket:id="message">
Message here</span>
        x </p>
    x </body>
x </html>
```

Recently Bookmark

Hello Message here

## Do the Quick Start

- x Maven installerat
- x Repository i ~/.m2/settings.xml
- x <http://wicket.apache.org/quickstart.html>
- x cd blog
- x Editera pom.xml så att Junit blir 4.4
- x mvn eclipse:eclipse -DdownloadSources=true
- x Fixa M2\_REPO i Eclipse (~/.m2/repository)
- x Kör Start.java
- x <http://localhost:8080/>

```
<settings>
  <profiles>
    <profile>
      <id>default-repositories</id>
      <repositories>
        <repository>
          <id>therepo1</id>
          <url>http://repo1.maven.org</url>
        </repository>
      </repositories>
    </profile>
  </profiles>
</settings>
```

## Notera

- x En testklass i src/test/java: TestHomePage
- x En HTML-sida och en Javaklass med samma namn: HomePage
- x HTML-sidan ren, bortsett från "wicket:id"

## Demo: en blogg

- x Dagens exempel är en blogg
- x Börja med att skapa en sida där man kan skriva ett inlägg

## Demo: en ny sida

- x En ny sida

```
x @Test
x public void shouldHaveCreatePostPage() {
x     tester.startPage(CreatePostPage.class);
x }
```

- x CreatePostPage är en ny klass som ärver WebPage

- x Kör testet, notera: "Markup of type 'html' for component ..."

- x Skapa CreatePostPage.html – green bar

Vi inleder med ett nytt test i den befintliga testklassen.

FrontPage hittade vi just på. Med quickfix skapar vi den och ärver WebPage.

## Demo: Webbläsaren

- x Plocka fram webbläsaren
  - x <http://localhost:8080/>
- x Men nu är vi ju på den torra startsidan ...
  - x <http://localhost:8080/CreatePostPage>
  - x Fungerar inte!
- x Vi måste lägga navigeringen i Java
  - x (Det finns bookmarkable page - överkurs)

# Testa efter länken

## x HomePage ska ha en länk till CreatePost

```
x public void testShouldHaveLinkToCreatePostPage() {  
    x tester.startPage(HomePage.class);  
    x  
    x HomePage homePage = (HomePage)  
      tester.getLastRenderedPage();  
    x Link link = (Link) homePage.get("create_post");  
    x assertNotNull(link);  
x }
```

## x Red bar!

## En länk

- x Lägga till i HomePage.html

- x `<a href="#" wicket:id="create_post">Create a blog post</a>`

- x Fortfarande red bar

- x Lägga till i HomePage.java

- x 

```
add(new Link("create_post") {
    @Override
    public void onClick() {
        setResponsePage(CreatePostPage.class);
    }
});
```

- x Prova i webbläsaren igen!

Java står som sagt för logiken, där ingår hur sidor hänger ihop. Var på sidan som länken sitter och hur den presenteras bestäms dock av HTML.

30 min

x Click to add an outline

# Vi ska kunna skapa ett inlägg

x Ett test verifierar att det finns ett inmatningsfält

```
x @Test
```

```
x public void shouldCreatePost() {
```

```
    x tester.startPage(CreatePostPage.class);
```

```
    x CreatePostPage page = (CreatePostPage)
      x tester.getLastRenderedPage();
```

```
    x Form createPostForm = (Form) page.get("create_post_form");
```

```
    x assertNotNull(createPostForm);
```

x Red bar: Lägg till formuläret "create\_post\_form"

```
x <form wicket:id="create_post_form">
```

```
x </form>
```

# Formulär i klassen

x Notera: "Unable to find component ..."

x Lägga till ett formulär i klassen

```
x public class CreatePostPage extends WebPage {  
x  
x     x public CreatePostPage() {  
x         x add(new CreatePostForm("create_post_form"));  
x     }  
x }
```

x Definiera CreatePostForm som en inre klass

# Formulärsklassen

```
x public class CreatePostForm extends Form {  
    x private static final long serialVersionUID = 1L;  
x  
    x public CreatePostForm(String id) {  
        x super(id);  
    x }  
x }
```

x Green bar

## Lite fält till formuläret

x Testa att det finns två fält, ett för rubrik och ett för innehållet

x ...

```
x TextField titleField = (TextField)  
createPostForm.get("title");
```

```
x assertNotNull(titleField);
```

```
x TextArea postArea = (TextArea) createPostForm.get("post");
```

```
x assertNotNull(postArea);
```

x Red bar...

.

# Fixa fälten

## x HTML

```
x <p>  
x Title: <input wicket:id="title" size="60" />  
x </p>  
x <p>  
x Post: <textarea rows="24" cols="80" wicket:id="post" />  
x </p>
```

## x Java

```
x add(new TextField("title"));  
x add(new TextArea("post"));
```

## x Green!

# Att testa ett formulär

```
x FormTester ft = tester.newFormTester("create_post_form");  
x ft.setValue("title", "TEST of title jada jada");  
x ft.setValue("post", "Oh la la IT is a test");  
x ft.submit();
```

x **Kör testet – kraschar fult:**

x ...

```
x Caused by: java.lang.IllegalStateException: Attempt to set  
model object on null model of component:  
create_post_form:title
```

x **Dags att berätta om modeller i Wicket**

# I, Model

- x IModel är gränssnittet
- x Modeller hanterar ett objekt – ditt domänobjekt

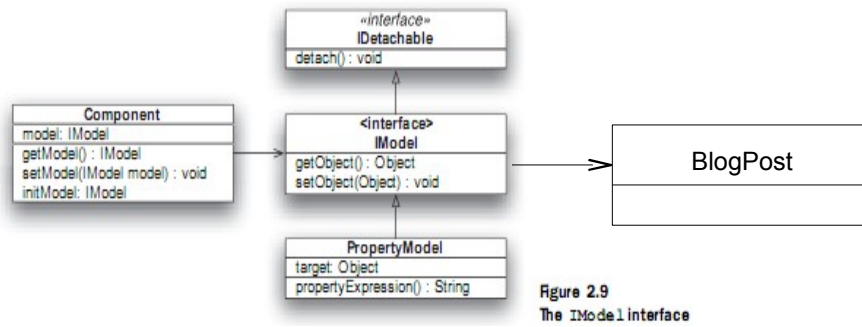


Figure 2.9  
The IModel interface

# BlogPost

- x Låt oss skapa ett domänobjekt – BlogPost

- x 

```
public class BlogPost implements Serializable{  
    x private String post;  
    x private String title;  
    x ... getters and setters
```

- x Sedan använder vi det i formuläret

- x 

```
private BlogPost blogPost = new BlogPost();
```
- x 

```
public CreatePostForm(String id) {  
    x super(id);  
    x setModel(new CompoundPropertyModel(this.blogPost));
```

- x Green!

# CompoundPropertyModel

- x Koppla domänobjektet till komponenten med modellen

```
x setModel(new CompoundPropertyModel(this.blogPost));
```

```
x add(new TextField("title"));
```

```
x add(new TextArea("post"));
```

- x HTML

```
x <input wicket:id="title" size="60" />
```

```
x <textarea wicket:id="post" rows="24" cols="80">
```

## Ett inlägg som skapas visas

x Testa att vi visar ett blogginlägg som skapats

```
x tester.assertRenderedPage (ShowPostPage.class);
```

```
x junit.framework.AssertionFailedError: expected:<ShowPostPage>  
but was:<CreatePostPage>
```

x Antagligen borde vi göra något på submit

```
x @Override
```

```
x protected void onSubmit() {
```

```
    x setResponsePage (ShowPostPage.class);
```

```
x }
```

x Med ShowPostPage.html och dito java ...

# Bara visa att det visas

## \* Verifiera att vårt inlägg visas

```
* ShowPostPage showPage = (ShowPostPage)
  tester.getLastRenderedPage();
* Label titleLabel = (Label) showPage.get("title");
* assertNotNull(titleLabel);
* Label postLabel = (Label) showPage.get("post");
* assertNotNull(postLabel);
```

\*

\*

# Fixa visningen!

x Fixa så att sidan vet vad den ska visa

```
x protected void onSubmit() {  
    x ShowPostPage spp = new ShowPostPage(blogPost);  
    x setResponsePage(spp);  
x }
```

x Se till att sidan visar det den får

```
x public ShowPostPage(BlogPost blogPost) {  
    x setModel(new CompoundPropertyModel(blogPost));  
    x add(new Label("post")); // and the HTML  
    x add(new Label("title")); // and the HTML  
x }
```

En sida är ett objekt, så vi ger det ett blogginlägg att presentera. Känns naturligt.

60 min

x Click to add an outline

# Testa själv

- x <http://localhost:8080/>
- x Fungerar ju
- x Nu är det dags att mocka
- x Va?

# Mocka är inte skit

- x "Mock object" står för ett objekt som simulerar ett verkligt objekt
- x Vi vill ha det för att vänta med serverkod
- x ... av något skäl
- x ... testa
- x ... demo
- x



x Bild från Wikipedia

# Mockito is the shit

- x Mockito liknar JMock och EasyMock

- x Enklare att använda.

- x "mock" skapar ett mockobjekt

- x `List<String> listMock = mock(List.class);`

- x "stub" beskriver hur det beter sig – om man vill

- x `stub(listMock.get(0)).thenReturn("nollan");`

- x "verify" kontrollerar att hur det använts

- x `verify(listMock).get(0);`

- x Igen: om man vill!

## När man skapar blogg ...

- x Skapar man ett blogginlägg så ska servern anropas
- x Vi skapar en tjänst "BlogService" som hittas via "BlogApplication"
- x Tjänsten ska användas av CreatePostPage när den skapar ett nytt inlägg.
- x Vi mockar upp en sådan tjänst

# Mockito läggs till pom.xml

```
x <dependency>  
  x <groupId>org.mockito</groupId>  
  x <artifactId>mockito-all</artifactId>  
  x <version>1.5</version>  
x </dependency>
```

## x Ny Maven till Eclipse

```
x > mvn eclipse:clean eclipse:eclipse
```

## Testa att servern anropas

- x Vi har ingen server men klienten ska använda den ändå

```
x @Test
```

```
x public void shouldCallServer() {
```

```
    x BlogService blogServiceMock = mock(BlogService.class);
```

```
    x BlogApplication.setBlogService(blogServiceMock);
```

```
    x tester.startPage(CreatePostPage.class);
```

- x Skapa inlägg med formuläret som förut, sedan

```
x verify(blogServiceMock).createPost(eq(expectedBlogPost));
```

# Förväntat: en röding, såklart!

```
x org.mockito.exceptions.verIFICATION.WantedButNotInvoked:  
x Wanted but not invoked:  
x   blogService.createPost(  
x     se.crisp.rd.BlogPost@12940b3  
x   );
```

## x Ny version av onSubmit

```
x protected void onSubmit() {  
x   BlogService service = BlogApplication.getBlogService();  
x   service.createPost(blogPost);  
x   ShowPostPage spp = new ShowPostPage(blogPost);  
x   setResponsePage(spp);  
x }
```

90 min

x Efter pausen gör vi en tabell

## Hur gör man tabeller?

- x Om nu HTML ska vara statiskt och Java stå för logiken – hur skapar man tabeller och listor?
- x Svar: logiken ligger i Java (AbstractRepeater)
- x Antag att vi vill ha en sida med alla blogginlägg
- x 

```
public class TestListPosts {  
    ...  
    public void shouldListBlogPosts() {  
        tester.startPage(ListBlogPostsPage.class);  
    }  
}
```
- x Kommer den inte att hämta alla inlägg från servern? Jo! GetPosts() - here we go!

# Låtsas att servern finns

\* (fortsättning från föregående sida)

```
* List<BlogPost> testData = new ArrayList<BlogPost>();  
* stub(blogServiceMock.getPosts()).thenReturn(testData);  
*  
* verify(blogServiceMock).getPosts();
```

\* **Skjut in testdata**

```
* BlogPost blogPost = new BlogPost();  
* blogPost.setTitle("Da must hat lieben geworden");  
* blogPost.setPost("Ich hat ein traum");  
* testData.add(blogPost);  
* BlogPost blogPost2 = new BlogPost();  
* ...
```

# Implementera hämtning

```
x public ListBlogPostsPage() {  
    x super();  
    x BlogService service = BlogApplication.getBlogService();  
    x List<BlogPost> posts = service.getPosts();  
x }
```

x **Green! → Mera tester!**

```
x ListBlogPostsPage listPage = (ListBlogPostsPage)  
    tester.getLastRenderedPage();  
x Component titleLabel1 = listPage.get("post_list:0:title");  
x assertNotNull(titleLabel1);
```

x **Red! → Implementera!**

# Knepigt?

- x **PropertyListView** är en "Repeater"

```
x add(new PropertyListView("post_list", posts) {  
  x @Override  
  x protected void populateItem(ListItem item) {  
    x item.add(new Label("title"));  
    x item.add(new MultiLineLabel("post"));  
  x }  
}
```

- x **Lite HTML**

```
x <div wicket:id="post_list">  
x <h3 wicket:id="title">Title</h3>  
x <p wicket:id="post">Post goes here.</p>  
x </div>
```

# Produktägaren kommer!

x Vi behöver en demo!



120 min

x Click to add an outline

# Demodag

- × Vi kan skapa bloggar – men vi måste ju visa det
- × Kopiera Start.java till Demo.java
- × Vi använder Mockito för att simulera en server som tar emot och lagrar våra inlägg

## Bygg en "server"

```
x private static void initApplikationMocks() {  
    x BlogService blogServiceMock = mock(BlogService.class);  
    x BlogApplication.setBlogService(blogServiceMock);  
    x doAnswer(new Answer() {  
        x public Object answer(InvocationOnMock invocation) {  
            x BlogPost post = (BlogPost) invocation.getArguments()  
              [0];  
            x posts.add(post);  
            x return null;  
        x }  
    x }).when(blogServiceMock).createPost((BlogPost) notNull());  
    x stub(blogServiceMock.getPosts()).toReturn(posts);  
x Testa http://localhost:8080
```

45

Prova att ändra HTML utan att starta om, t ex:

```
<div wicket:id="post_list" style="border:  
    solid">
```

# Övningar

- \* Välj en övning – säg högt vilken
- \* Parprogrammera och jobba **testdrivet**
- \* Övningarna bygger på det som gjorts
  - 1) Inloggning med skyddade sidor
  - 2) Använd Feedback för att kontrollera inmating
  - 3) Använd AjaxEditableLabel för att ändra ett inlägg
  - 4) Använd AjaxCompleteTextField för sökning
  - 5) Skapa en Panel med en klocka som uppdateras
  - 6) Skapa en bassida som ger sajtens utseende

## Övning: Inloggning med skyddade sidor

x Skapa en sida där man kan logga in

x Ärv WebSession och lägg in användaren där

```
x public class BlogSession extends WebSession {  
    x private User user = null;
```

x Definiera om i WicketApplication

```
x @Override
```

```
x public Session newSession(Request request, Response response)  
{
```

```
x     return new BlogSession(request);
```

```
x }
```

## Övning: Använd Feedback

- x Lägg till en FeedbackPanel på CreatePostPage
- x Ändra så att Title blir "RequiredTextField"

## Övning: Använd AjaxEditableLabel

- x Ändra så att man kan ändra ett inlägg "inline".

## Övning: Använd AjaxCompleteTextField

- x För varje text som matas in, indexera upp orden
- x Skapa en söksida som ger förslag på sökbegrepp vartefter man skriver

## Övning: Skapa en Panel

- x Panel är grunden för återanvändbara "widgets" i Wicket
- x Istället för en hel sida så bygger du upp en Panel i HTML och Java som distribueras i en Jar
- x Skapa en Panel med en kalender som man stoppa in på en sida
- x Använd `<wicket:panel>`

## Länkar

- × Hemsidan: <http://wicket.apache.org/>
- × Javadoc API: <http://wicketstuff.org/wicket13doc>
- × Exempel: <http://wicketstuff.org/wicket13>
- × Blog: <http://chillenious.wordpress.com/category/wicket>
- × Bok: Wicket in Action, <http://manning.com/dashorst/>
- × Snabbstart: <http://wicket.apache.org/quickstart.html>
- × Mockito: <http://code.google.com/p/mockito/>

# Extra Material

x Click to add an outline

# How do we handle log in?

- x Typically, some pages require users to log in
- x If they try to reach a protected page, they should need to log in and *then* reach the page
- x It is called interception in Wicket
- x `public void` `shouldInterceptWhenNotLoggedIn()` {
  - x Test that we reach the log in page
- x `public void` `shouldNotInterceptWhenLoggedIn()` {
  - x Test that we reach the create post page
- x `public void` `shouldReachPageAfterLogIn()` {
  - x Test that we reach log in and then create post page

Per Lundholm, Crisp

54

```
@Test
public void shouldInterceptWhenNotLoggedIn() {
    tester.startPage(HomePage.class);

    tester.clickLink("create_post");

    tester.assertRenderedPage(LoginPage.class);
}

@Test
public void shouldNotInterceptWhenLoggedIn() {
    tester.startPage(HomePage.class);
    User testUser = new User();
    testUser.setUserId("test.user");
    testUser.setPassword("test.password");
    session.setUser(testUser);

    tester.clickLink("create_post");

    tester.assertRenderedPage(CreatePostPage.class);
}

@Test
public void shouldReachPageAfterLogIn() {
    tester.startPage(HomePage.class);

    tester.clickLink("create_post");

    tester.assertRenderedPage(LoginPage.class);

    FormTester ft = tester.newFormTester("login_form");
    ft.setValue("userId", "test.user");
    ft.setValue("password", "test.password");
    ft.submit();

    tester.assertRenderedPage(CreatePostPage.class);
}
}
```

# The login form is ordinary

- x But we use a method on Component

```
x @Override
x protected void onSubmit() {
x     // Check that the log in is ok
x     // if ...
x     BlogSession session = (BlogSession) getSession();
x     session.setUser(user);
x     if(!continueToOriginalDestination()) {
x         setResponsePage(getApplication().getHomePage());
x     }
```

- x Destination is charged earlier by ...

# CreatePost guards itself

```
* public CreatePostPage () {  
  * if (!(BlogSession) getSession()).isAuthenticated() {  
    * redirectToInterceptPage(new LoginPage());  
  * }  
  * add(new CreatePostForm("create_post_form"));  
* }
```

# Intercept to handle log in

- x As usual, credentials are stored in the session
- x But we subclass it to get it type safe
- x 

```
public class BlogSession extends WebSession {  
    x private User user = null;  
    x public BlogSession(Request request) { super(request); }  
    x public static BlogSession get() {  
        x return (BlogSession) WebSession.get();  
    x }  
    x public boolean isAuthenticated() { return (user != null); }  
x }
```

Per Lundholm, Crisp

57

Test set up creates a session:

```
@Before  
public void setUp() {  
    tester = new WicketTester(new WicketApplication() {  
  
        @Override  
        public Session newSession(Request request, Response response) {  
            session = new BlogSession(request);  
            return session;  
        }  
  
    });  
}
```

# Komponenter i Wicket

- x Består av
  - x Presentation i HTML
  - x Logik i Java
- x Packas i en jar
  - x ... nås via classpath

# Demo: en komponent

- x Vi vill skapa en klocka som visar tiden, hela tiden

- x Så här skulle vi vilja använda den:

- x `<div style="font-size: 24">`
  - x `<span wicket:id="clock">CLOCK</span>`
  - x `</div>`

- x I Java:

- x `add(new ClockPanel("clock"));`

# Demo: klockpanelen

## \* Gör en panel i HTML (ClockPanel.html)

```
* <html>
  * <head></head>
  * <body>
    * <wicket:panel>
      * <span wicket:id="clock">Clock</span>
    * </wicket:panel>
  * </body>
* </html>
```

# Demo: Inkludera i panelen

## \* HTML-koden backas upp med:

```
* public class ClockPanel extends Panel {  
    * public ClockPanel(String id) {  
        * super(id);  
        * Clock clock = new Clock("clock", TimeZone.getTimeZone("America/  
        * Los_Angeles"));  
        * add(clock);  
        * clock.add(new  
        * AjaxSelfUpdatingTimerBehavior(Duration.seconds(5)));  
    * }  
* }
```

# Demo: klockkoden

x Från <http://wicketstuff.org/wicket13/ajax/>

```
*, public class Clock extends Label {
*,     public Clock(String id, TimeZone tz) {
*,         super(id, new ClockModel(tz));
*,     }
*,     private static class ClockModel extends AbstractReadOnlyModel {
*,         *, private DateFormat df;
*,         *, public ClockModel(TimeZone tz) {
*,             *, df = DateFormat.getDateInstance(DateFormat.FULL, DateFormat.LONG);
*,             *, df.setTimeZone(tz);
*,         }
*,         *, public Object getObject() {
*,             *, return df.format(new Date());
*,         }
*,     }
*, }
```