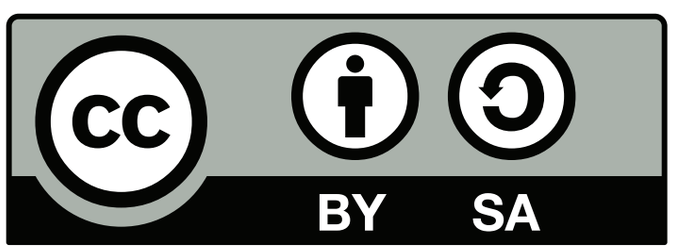
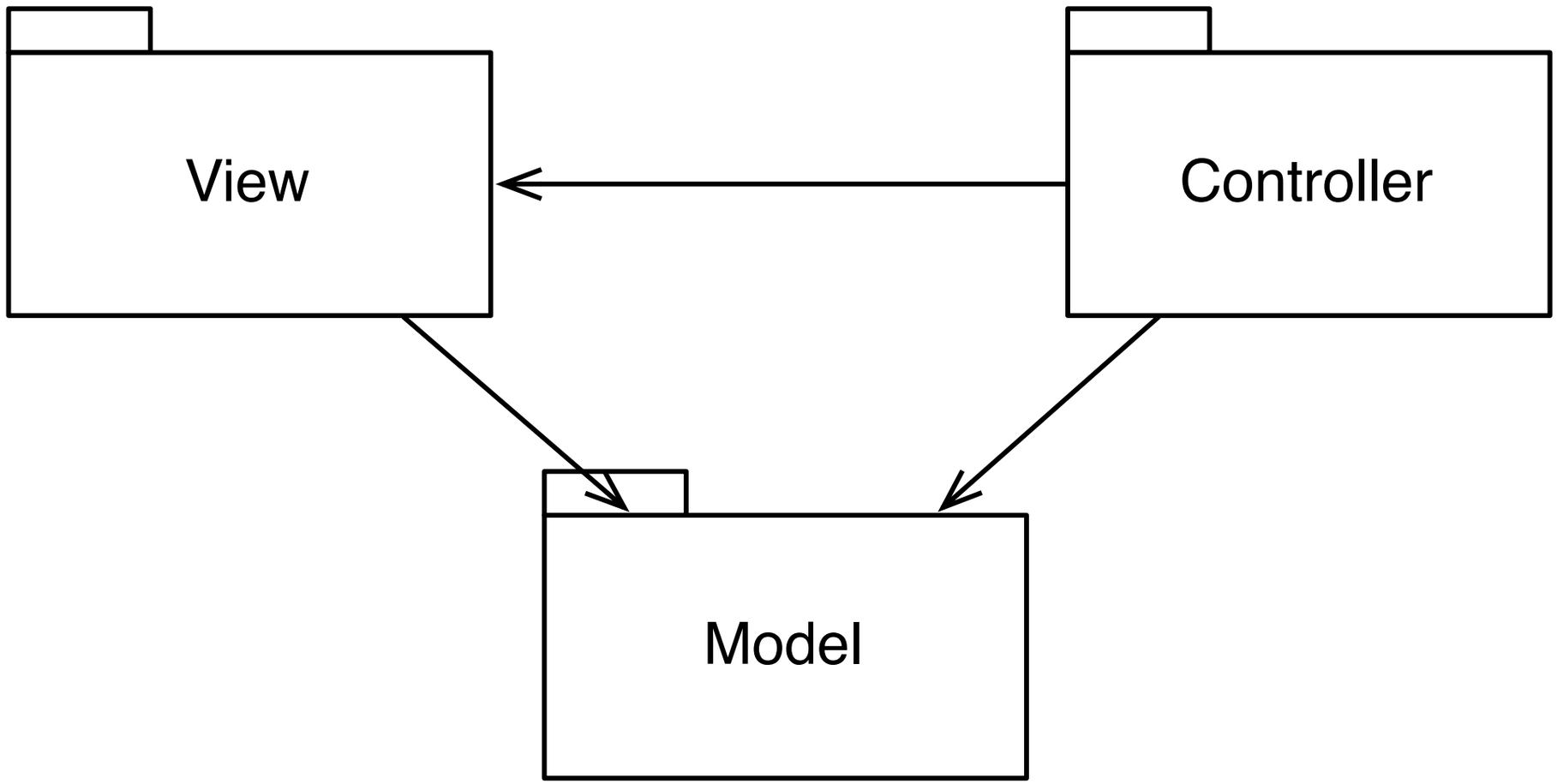
Tecnologia e Applicazioni Internet 2008/9

Lezione 2 - Views Technology

Matteo Vaccari
http://matteo.vaccari.name/
matteo.vaccari@uninsubria.it



Model, view, controller



Mixing controller and view

```
public String doGet(String string) {
    String heading = "<h1>People Directory</h1>";
    String tableStart = "<table>";
    String rows = "";
    for (Person person : repository.getPeople(0, 10)) {
        rows += convertPersonToTableRow(person);
    }
    String tableEnd = "</table>";
    return heading  + tableStart  + rows + tableEnd ;
}
```

View code in blue, data access in red

Come realizzare la view?

- Templates
  - ✦ Freemarker, Velocity, StringTemplate...
  - ✦ Java Server Pages (JSP)
- Builder
  - ✦ Pattern poco usato in Java ma utile

Template + data-model = output

```
<html>
<head>
  <title>Welcome!</title>
</head>
<body>
  <h1>Welcome ${user}!</h1>
  <p>Our latest product:
  <a href="${latestProduct.url}">
   ${latestProduct.name}</a>!
</body>
</html>
```

+

```
(root)
  |
  +- user = "Big Joe"
  |
  +- latestProduct
      |
      +- url = "products/greenmouse.html"
      |
      +- name = "green mouse"
```

=

```
<html>
<head>
  <title>Welcome!</title>
</head>
<body>
  <h1>Welcome Big Joe!</h1>
  <p>Our latest product:
  <a href="products/greenmouse.html">
   green mouse</a>!
</body>
</html>
```

Examples from the Freemarker manual - http://freemarker.sourceforge.net/

The data model

- Is a simple map names to values
- The values can be any standard Java Maps, Lists, Arrays, or beans

```
(root)
  |
  +- animals
  |    |
  |    +- (1st)
  |    |    |
  |    |    +- name = "mouse"
  |    |    |
  |    |    +- size = "small"
  |    |    |
  |    |    +- price = 50
  |    |
  |    +- (2nd)
  |    |    |
  |    |    +- name = "elephant"
  |    |    |
  |    |    +- size = "large"
  |    |    |
  |    |    +- price = 5000
  |    |
  |    +- (3rd)
  |         |
  |         +- name = "python"
  |         |
  |         +- size = "medium"
  |         |
  |         +- price = 4999
  |
  +- whatnot
       |
       +- fruits
            |
            +- (1st) = "orange"
            |
            +- (2nd) = "banana"
```

```
class Animal {
    public String getName() {...}
    public String getSize() {...}
    public int getPrice() {...}
}

List other = Arrays.asList("orange", "banana");

Animal[] myAnimals = new Animal [] {
    new Animal("mouse", "small", 50),
    new Animal("elephant", "large", 5000),
    new Animal("python", "medimo", 4999),
};

Map modelRoot = new HashMap();
modelRoot.put("animals", myAnimals);
modelRoot.put("whatnot", other);
```

- The data model is a tree
- Scalars: numbers, strings, dates, bolean
- Non-scalars: arrays, lists, maps, any Java class with getters

Templates directives

```
<#if animals.python.price < animals.elephant.price>
  Pythons are cheaper than elephants today.
<#else>
  Pythons are not cheaper than elephants today.
</#if>
```

```
<p>We have these animals:
<table border=1>
  <tr><th>Name</th><th>Price</th></tr>
  <#list animals as animal>
  <tr><td>${animal.name}<td>&euro; ${animal.price}</td></tr>
  </#list>
</table>
```

```
<#include "/copyright_footer.html">
```

```
<p>We have these animals:
<table border=1>
  <tr><th>Name</th><th>Price</th>
  <tr><td>mouse<td>&euro; 50</td></tr>
  <tr><td>elephant<td>&euro; 5000</td></tr>
  <tr><td>python<td>&euro; 4999</td></tr>
</table>
```

Checking for null

```
<h1>Welcome ${user!"Anonymous"}!</h1>
```

Print value of “user” if not null, otherwise “Anonymous”

```
<#if user??><h1>Welcome ${user}!</h1></#if>
```

Print message only if value of “user” is not null

Programming Freemarker

```
// configure
Configuration configuration = new Configuration();
configuration.setDirectoryForTemplateLoading(
        new File("/where/you/store/templates"));

// load the template from file
// /where/you/store/templates/test.ftl
Template template = configuration.getTemplate("test.ftl");

// create the root model
Map root = new HashMap();
root.put("user", "Big Joe");

// merge data and template
Writer out = new OutputStreamWriter(System.out);
template.process(root, out);
out.flush();
```

Testing views

- Voglio testare la *logica della view* e solo quella
- Voglio passare informazioni al template e vedere che le renderizzi senza eccezioni
- *Non voglio* dover lanciare un web server per eseguire i test

Simple test

```
@Test
public void testFreemarker() throws Exception {
    String templateString = "Hello, ${user}";
    Template template = new Template("my template",
        new StringReader(templateString), new Configuration());

    Map model = new HashMap();
    model.put("user", "Tizius");

    StringWriter writer = new StringWriter();
    template.process(model, writer);
    assertEquals("Hello, Tizius", writer.toString());
}
```

Not so simple test

```
@Test
public void bigTest() throws Exception {
    Template template = configuration.getTemplate("bigjoe.ftl");

    Map model = new HashMap();
    model.put("user", "Big Joe");

    StringWriter writer = new StringWriter();
    template.process(model, writer);

    String expected = "<html>\n" +
    "<head>\n" +
    "  <title>Welcome!</title>\n" +
    "</head>\n" +
    "<body>\n" +
    "  <h1>Welcome Big Joe!</h1>\n" +
    "  <p>Our latest product:\n" +
    "  <a href=\"products/greenmouse.html\">\n" +
    "green mouse</a>!\n" +
    "</body>\n" +
    "</html>\n";
    assertEquals(expected, writer.toString());
}
```

Testare l'html come stringa non conviene

- Test fragile: basta aggiungere uno spazio e si rompe
- Test rigido: modifiche non significative (es. cambio il titolo della pagina) rompono il test

Testare la view con *XPath*

```
@Test
public void betterTest() throws Exception {
    ...
    Document document = xmlDocumentFromString(writer.toString());
    assertEquals("Welcome!", getNodeContent(document, "/html/head/title"));
    assertEquals("green mouse",
        getNodeContent(document, "//a[@href='greenmouse.html']"));
}
```

```
<html>
<head>
  <title>Welcome!</title>
</head>
<body>
  <h1>Welcome Big Joe!</h1>
  <p>Our latest product:
    <a href="greenmouse.html">green mouse</a>!
  </p>
</body>
</html>
```

Introduzione a XPath

- http://www.lonerunners.net/slides/xpath-injection, slides 4-12

Perché non JSP?

- Perché è difficile scrivere test *unitari*
- Occorre un compilatore Java
- Si può fare (Lasse Koskela ha scritto una libreria) ma vale la pena solo per progetti *legacy*

Builder style

```
html do
  head do
    title 'Big products!'
    stylesheet_link_tag 'scaffold'
  end

  body do
    h1 "Big products!", :style => "color: green"
    p "first paragraph"
  end
end
```

```
new Html().add(
  new Head().add(
    new Title("My Title!"),
    new StylesheetLinkTag("scaffold")
  ),
  new Body().add(
    new H1("Big Products!"),
    new Paragraph("first paragraph").with("style", "color: green")
  )
).build();
```

Come funziona?

```
public interface HtmlBuilder {
    public HtmlBuilder add(HtmlBuilder ... children);
    public String build();
}

public class Title implements HtmlBuilder {

    private final String content;

    public Title(String content) {
        this.content = content;
    }

    @Override
    public String build() {
        return "<title>" + content + "</title>";
    }
}
```

Come funziona?

```
public interface HtmlBuilder {
    public HtmlBuilder add(HtmlBuilder ... children);
    public String build();
}

public class Head implements HtmlBuilder {
    private List<HtmlBuilder> children = new ArrayList<HtmlBuilder>();

    public HtmlBuilder add(HtmlBuilder ... moreChildren) {
        this.children.addAll(Arrays.asList(moreChildren));
        return this;
    }

    private String processChildren() {
        String result = "";
        for (HtmlBuilder child : children) {
            result += child.build();
        }
        return result;
    }
}
```

Vantaggi del *builder*

- L'html è valido *per costruzione*
- Posso generare diversi linguaggi (html 4.0, xhtml 1.0,...) cambiando un opzione