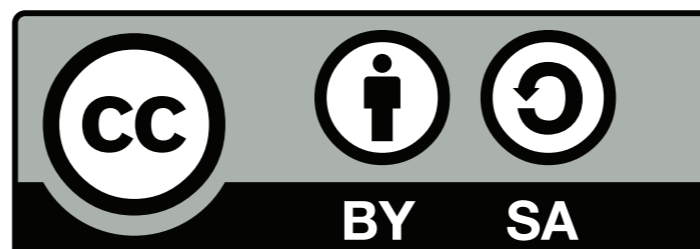


# Tecnologia e Applicazioni Internet 2009/10

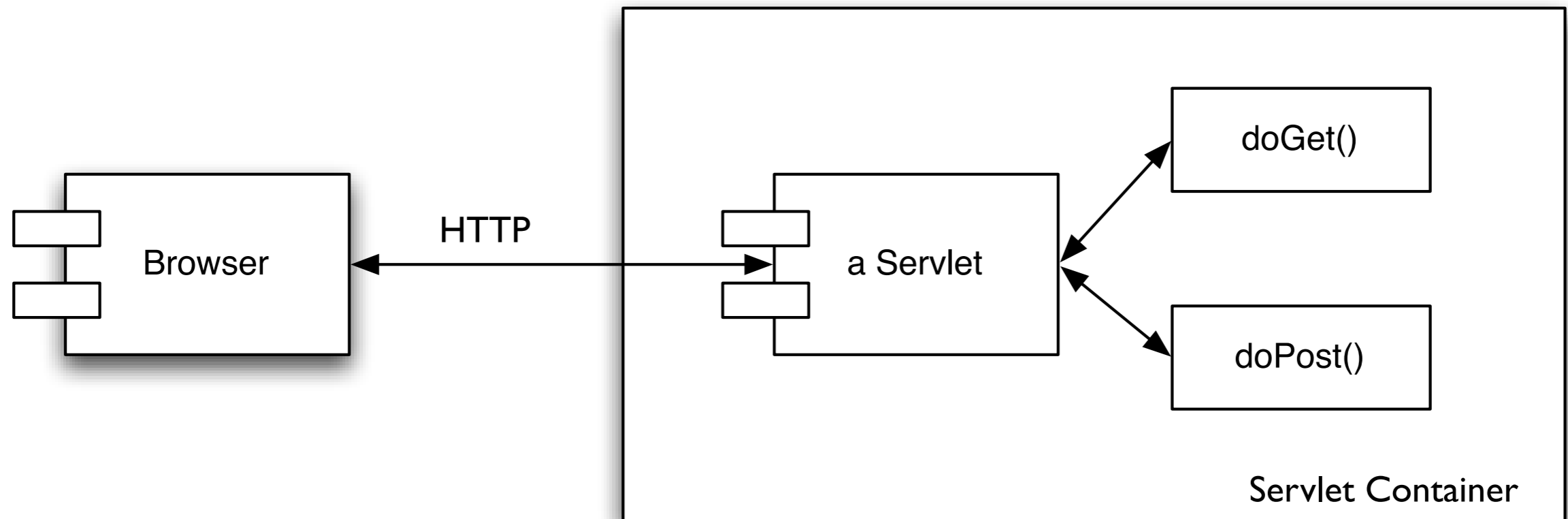
Lezione 2 - Web e Java

Matteo Vaccari

<http://matteo.vaccari.name/>  
[matteo.vaccari@uninsubria.it](mailto:matteo.vaccari@uninsubria.it)



# Servlets and container



# Simple servlet

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

public class HelloWorld extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        String name = request.getParameter("name");
        PrintWriter out = response.getWriter();

        out.println("<html>");
        out.println("<head><title>Hello world</title></head>");
        out.println("<body>");
        out.println("<big>Hello, " + name + "</big>");
        out.println("</body></html>");
    }
}
```

# Project structure

```
.
|-- lib
|   `-- servlet-api-2.4.jar
|-- src
|   `-- com
|       `-- example
|           `-- foobar
|               `-- FoobarServlet.java
`-- webapp
    |-- WEB-INF
    |   |-- classes
    |   |   `-- com
    |   |       `-- example
    |   |           `-- foobar
    |   |               `-- FoobarServlet.class
    |   `-- web.xml
    |-- images
    |   `-- logo.png
    |-- index.html
    |-- javascripts
    |   `-- prototype.js
    `-- stylesheets
        `-- style.css
```

# web.xml

```
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
```

```
<web-app>
  <servlet>
    <servlet-name>hello-world</servlet-name>
    <servlet-class>com.example.foobar.HelloWorld</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>hello-world</servlet-name>
    <url-pattern>/hello/*</url-pattern>
  </servlet-mapping>
</web-app>
```

/	equivale a /*
/hello	path esatto
/hello/*	tutti i path con quel prefisso
*.png	tutti i path con quella estensione

# Le servlet sono *singleton*

- Sono accedute da più thread contemporaneamente
- Evitare dunque le variabili di istanza

# HttpServletRequest

<http://www.example.com/context/hello/world>

String getParameter(String)  
Enumeration getParameterNames()

String getScheme() //=> http  
String getServerName() //=> www.example.com  
getServerPort() //=> 80  
getRequestURI() //=> /context/hello/world  
getContextPath() //=> /context  
getServletPath() //=> /hello  
getPathInfo() //=> /world

String getRemoteAddr()  
String getRemoteHost()

Enumeration getHeaderNames()  
String getHeader(String name)

Cookie[] getCookies()

# HttpServletResponse

```
void setContentType(String type)
void setContentLength(int length)
ServletOutputStream getOutputStream() // output binario
PrintWriter getWriter() // output testo

void setHeader(String name, String val)
void addCookie(Cookie cookie)

void sendRedirect(String url)
void setStatus(int code)
```

# Sessions

```
HttpSession session = request.getSession();  
Integer userId = (Integer) session.getValue("userId");
```

# Ant: like *make* for Java

```
<project name="simple-webapp" default="compile" basedir=". ">  
  
  <property name="build.dir"      location="target" />  
  <property name="build.prod.dir" location="${build.dir}/prod" />  
  <property name="build.test.dir" location="${build.dir}/test" />  
  <property name="src.dir"        location="src" />  
  <property name="lib.dir"        location="lib" />  
  <property name="web.dir"        location="webapp" />  
  <property name="test.dir"       location="unit" />  
  <property name="test.lib.dir"   location="lib/test" />  
  <property name="war.file"       location="${build.dir}/simple-webapp.war" />  
  <property name="conf.dir"       location="conf" />
```

# Ant: like *make* for Java

```
<path id="project.classpath">
  <pathelement location="${build.prod.dir}" />
  <pathelement location="${build.test.dir}" />
  <fileset dir="${lib.dir}">
    <include name="**/*.jar" />
  </fileset>
</path>

<target name="prepare">
  <mkdir dir="${build.prod.dir}" />
  <mkdir dir="${build.test.dir}" />
</target>

<target name="compile" depends="prepare">
  <javac source="1.5" target="1.5" destdir="${build.prod.dir}"
    debug="true" encoding="utf-8">
    <src path="${src.dir}" />
    <classpath refid="project.classpath" />
  </javac>
</target>
```

# Ant: like *make* for Java

```
<target name="prepare">
  <mkdir dir="${build.prod.dir}"/>
  <mkdir dir="${build.test.dir}"/>
</target>

<target name="compile" depends="prepare">
  <javac source="1.5" target="1.5" destdir="${build.prod.dir}"
    debug="true" encoding="utf-8">
    <src path="${src.dir}" />
    <classpath refid="project.classpath" />
  </javac>
</target>
```

```
$ ant compile
Buildfile: build.xml

prepare:
  [mkdir] Created dir: /.../target/prod
  [mkdir] Created dir: /.../target/test

compile:
  [javac] Compiling 3 source files to /.../target/prod

BUILD SUCCESSFUL
Total time: 0 seconds
$
```

# Web ARchives: .war

```
$ jar tf target/simple-webapp.war
META-INF/
META-INF/MANIFEST.MF
WEB-INF/
WEB-INF/web.xml
WEB-INF/classes/
WEB-INF/classes/com/
WEB-INF/classes/com/example/
WEB-INF/classes/com/example/foobar/
WEB-INF/classes/com/example/foobar/HelloServlet.class
images/
javascripts/
stylesheets/
images/logo.png
javascripts/prototype.js
stylesheets/style.css
$
```

```
<target name="war" depends="compile">
  <war destfile="${war.file}" webxml="${conf.dir}/web.xml">
    <classes dir="${build.prod.dir}" />
    <classes dir="${src.dir}" includes="**/*.properties" />
    <fileset dir="${web.dir}" />
    <classes dir="${conf.dir}" includes="log4j.*" />
  </war>
</target>
```

```
$ ant war
Buildfile: build.xml

prepare:

compile:
    [javac] Compiling 2 source files to ../../target/prod

war:
    [war] Building war: ../../target/simple-webapp.war

BUILD SUCCESSFUL
Total time: 0 seconds
$
```

# Running a .war

```
$ cat script/start.sh
```

```
ant clean war || exit 1
```

```
java -jar script/winstone-0.9.10.jar --warfile target/simple-webapp.war
```

```
$
```

***Winstone*** is a very easy-to-use  
servlet container

# Servlets are hard to test

- Require to extend `HttpServlet`
- Require a no-arguments constructor
- Work with `HttpServletRequest` and `-Response` which are hard to instantiate
- `HttpServletRequest`: interface with 54 methods
- `HttpServletResponse`: interface with 32 methods

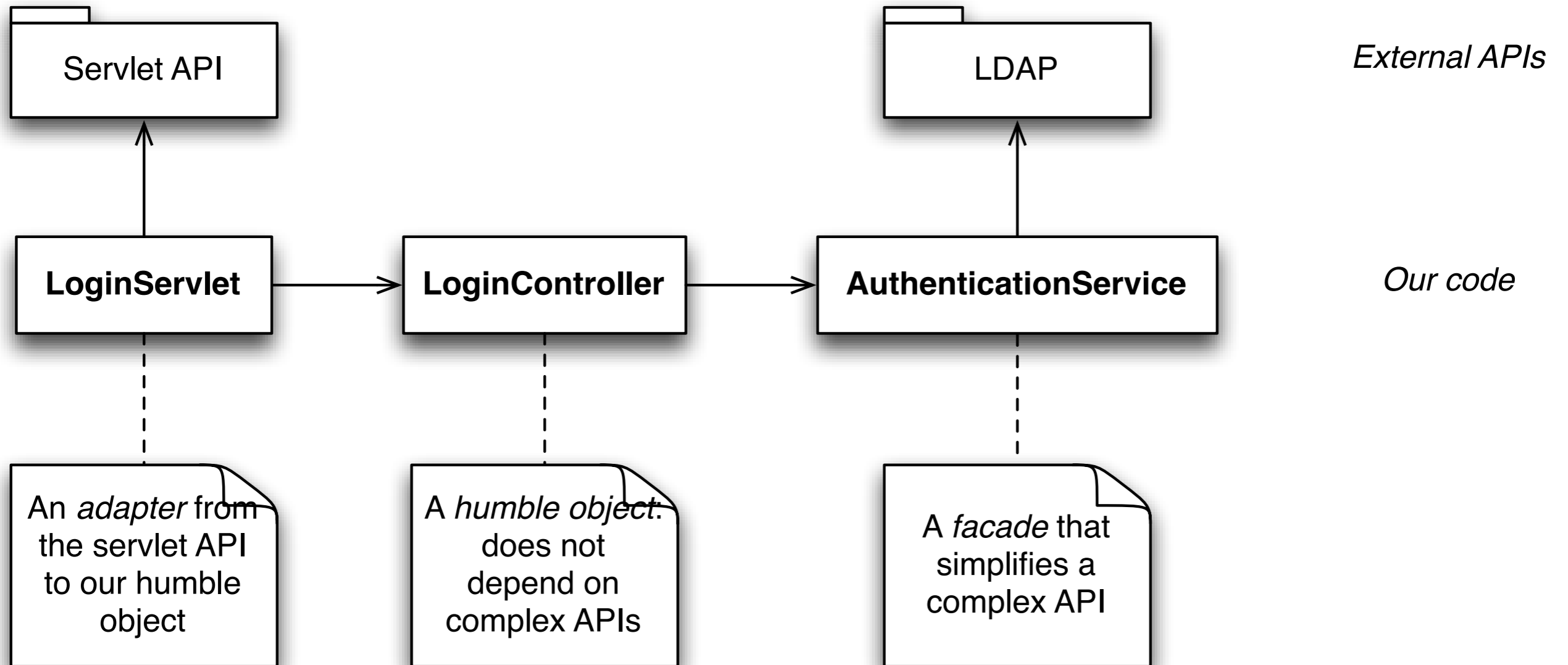
# What are test doubles?

- **Stub:** an object that implements a simplified version of the real object
- **Mock:** an object that verifies interaction

# Testing a servlet with stubs

```
import com.mockobjects.servlet.MockHttpServletRequest;  
import com.mockobjects.servlet.MockHttpServletResponse;  
  
public class AdderServletTest {  
  
    @Test  
    public void testAddingTwoNumbers() throws Exception {  
        MockHttpServletRequest request = new MockHttpServletRequest();  
        request.setupAddParameter("a", "3");  
        request.setupAddParameter("b", "2");  
        MockHttpServletResponse response = new MockHttpServletResponse();  
        response.setExpectedContentType("text/plain");  
  
        new AdderServlet().doGet(request, response);  
  
        assertEquals("5\n", response.getOutputStreamContents());  
        response.verify();  
    }  
}
```

# Testing with humble objects



# Simplify complex APIs with a *facade*

```
interface AuthenticationService {  
    boolean authenticate(String username,  
                        String password);  
}
```

```
class LdapAuthenticationService implements AuthenticationService {  
    boolean authenticate(String username,  
                        String password) {  
        // do complex stuff  
    }  
}
```

```
class StubAuthenticationService implements AuthenticationService {  
    boolean authenticate(String username,  
                        String password) {  
        return username.equals("foo") && password.equals("bar");  
    }  
}
```

# Testing with *humble objects*

```
class LoginController {
    AuthenticationService authenticator;
    boolean success;
    String errorMessage;

    LoginController(AuthenticationService authenticator) {
        this.authenticator = authenticator;
    }

    String execute(Map<String, String> parameters,
                  String cookie) {
        // do some work
        success = ...;
        errorMessage = ...;
    }

    String render(Writer writer) {
        if (success)
            return "redirect URL";
        else
            writer.write(...);
    }
}
```

<http://misko.hevery.com/2009/01/04/interfacing-with-hard-to-test-third-party-code/>

# An *adapter* from Servlet

```
class LoginServlet extends HttpServlet {
    Provider<LoginController> provider;

    // no arg constructor required by Servlet Framework
    LoginServlet() {
        this(Global.injector.getProvider(LoginController.class));
    }

    // Dependency injected constructor used for testing
    LoginServlet(Provider<LoginController> provider) {
        this.provider = provider;
    }

    void doPost(HttpServletRequest req, HttpServletResponse resp) {
        LoginController controller = provider.get();
        controller.execute(req.getParameterMap(), req.getCookies());
        String redirect = controller.render(resp.getWriter());
        if (redirect != null)
            resp.sendRedirect(redirect);
    }
}
```

<http://misko.hevery.com/2009/01/04/interfacing-with-hard-to-test-third-party-code/>

# Key points

- *Facade* is used when we are calling a complex API
- *Adapter* is used when we are called by a complex API
- Humble objects contain application logic & are easy to test
- Adapter is hard to test but contains *no logic, only wiring*
- Facades can be interfaces that are easy to stub