Java EE 6
Servlet 3.0 Basics

Sang Shin
JPassion.com
"Code with Passion!"

Topics

- Ease of Development
- Dynamic registration of Servlets and Filters
- Pluggability
- Resources in bundled jar files

Advanced topics of Servlet 3.0 (AsyncServlet, Security enhancements, etc.) will be covered in “Servlet 3.0 Advanced”

Ease of Development

Ease of Development (EoD)

- Declarative style of programming through annotations
- *web.xml* is now optional
  - Everything that can be specified in web.xml now can be specified through annotations
- Better defaults
  - Convention over configuration

Use of Annotation

- Annotations to declare Servlets, Filters, Listeners and security constraints (replacing relevant configuration in *web.xml* file)
  - `@WebServlet` – Define a Servlet
  - `@WebFilter` – Define a Filter
  - `@WebListener` – Define a Listener
  - `@WebInitParam` – Define init params
  - `@MultipartConfig` – Define file upload properties
  - `@ServletSecurity` – Define security constraints

- Use *web.xml* to override values specified in the annotations

Use of Annotation (Continued)

- *@WebServlet* for defining a Servlet
  - The annotation MUST have at a minimum the URL pattern for the Servlet (as an annotation attribute)
  - All other attributes are optional with reasonable defaults.
    - For example, the default name of the Servlet is the name of the class
  - Class MUST still extend *HttpServlet*

Old Servlet 2.5 Example

```
public class SimpleSample extends
HttpServlet {
    public void doGet
    (HttpServletRequest req,
     HttpServletResponse res){


    }
}
```

web.xml (intentionally left unreadable)

```
<web-app>
  <servlet>
    <servlet-name>                MyServlet
    </servlet-name>
    <servlet-class>
     samples.SimpleSample
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>
      MyServlet
    </servlet-name>
    <url-pattern>
      /MyApp
    </url-pattern>
  </servlet-mapping>
...
</web-app>
```

@WebServlet Example #1

```
// The "value" attribute is recommended for use when
// the URL pattern is the only attribute being set,
// otherwise the "urlPattern" attribute should be
// used.
@WebServlet("/foo") //or @WebServlet(value = "/foo")
public class SimpleSample extends  HttpServlet {
    public void doGet(
            HttpServletRequest req,
            HttpServletResponse res){
      // Code
    }
 }
```

@WebServlet Example #2

```
@WebServlet(urlPatterns={"/foo", "bar"},
            name="MyServlet",
            asyncSupported=true)
public class SimpleSample extends HttpServlet {
    public void doGet(HttpServletRequest
        req,HttpServletResponse res){

    }
}
```

@WebFilter Example #1

```
@WebFilter(urlPatterns={"/"},
                initParams={ @WebInitParam(name="mesg", value="my filter") })
public class TestFilter implements Filter {
   String mesg = null;

   public void init(FilterConfig filterConfig) throws ServletException {
      mesg = filterConfig.getInitParameter("mesg"); // set the “mesg” variable with “my filter”
   }

   public void doFilter(ServletRequest req, ServletResponse res,
         FilterChain chain) throws IOException, ServletException {

      req.setAttribute("filterMessage", mesg);  // save it in request scope object
      chain.doFilter(req, res);                 // call next filter in the chain
   }

   public void destroy() {
   }
}
```

@WebFilter Example #2

```
@WebFilter(urlPatterns={ "/" }, dispatcherTypes={DispatcherType.REQUEST})
public class Wf3TestFilter implements Filter {
    public void init(FilterConfig filterConfig) throws ServletException {
    }

    public void doFilter(ServletRequest req, ServletResponse res,
            FilterChain chain) throws IOException, ServletException {

        // some code

        req.setAttribute("filterMessage", filterMessage);
        chain.doFilter(req, res);
    }

    public void destroy() {
    }
}
```

@WebListener Example

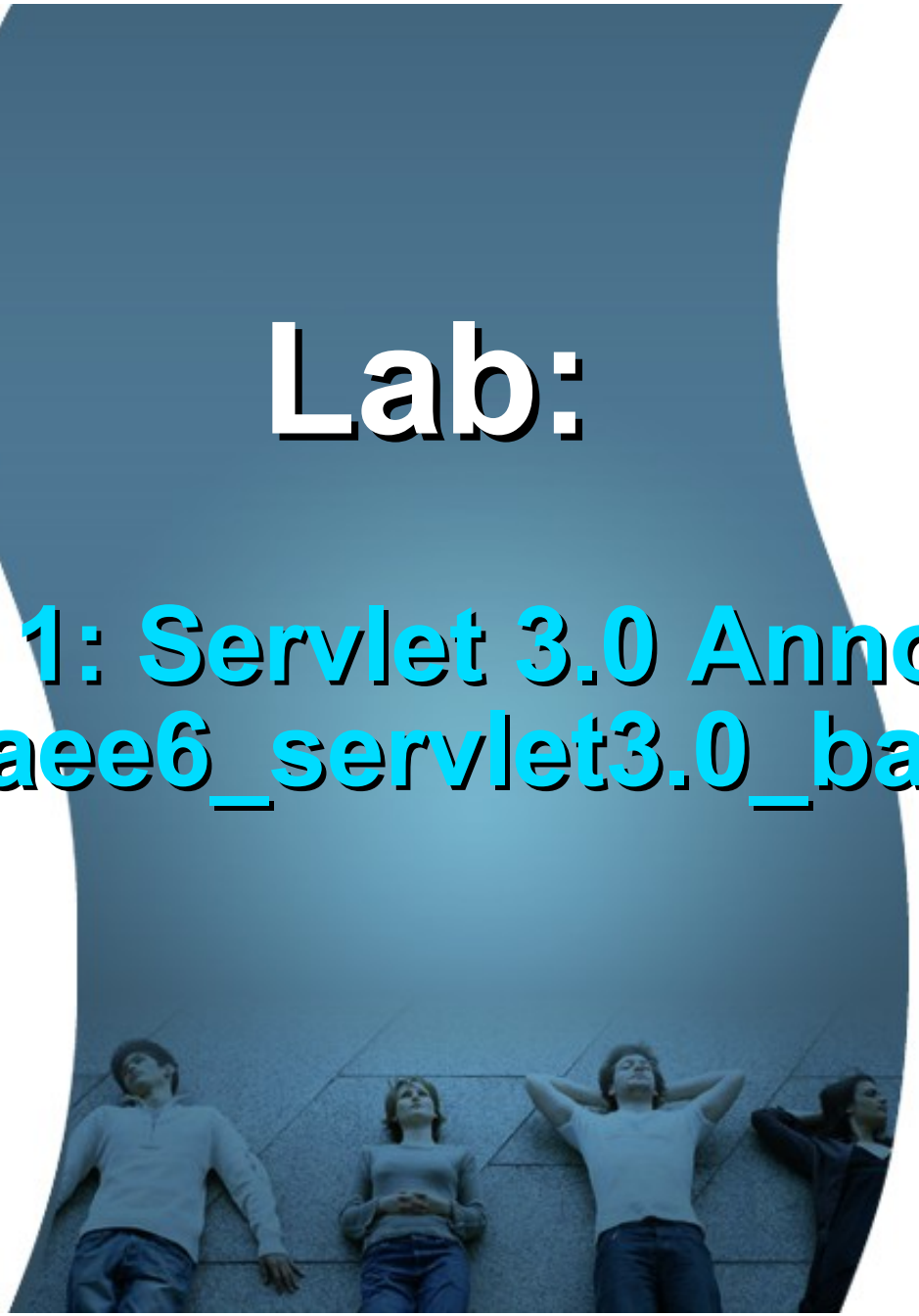
```
@WebListener
public class TestServletContextListener implements ServletContextListener {
    public void contextInitialized(ServletContextEvent sce) {

        ServletContext context = sce.getServletContext();
        context.setAttribute("listenerMessage", "my listener");
    }

    public void contextDestroyed(ServletContextEvent sce) {
    }
}
```

Lab:

Exercise 1: Servlet 3.0 Annotations
4532_javaee6_servlet3.0_basics.zip



Dynamic Registration of Servlets and Filters

Dynamic/Programmatic Servlet registration

- Performed during ServletContext initialization
- ServletContext#add[Servlet|Filter]
  - Use returned *Registration* handle to configure all aspects of [Servlet|Filter]

Dynamic registration of a Servlet

```
public class TestServletContextListener
             implements ServletContextListener {
    @Override
    public void contextInitialized(ServletContextEvent sce) {

        ServletContext servletContext = sce.getServletContext();

        ServletRegistration.Dynamic dynamic =
                    servletContext.addServlet(
                                "DynamicServlet",        // Name
                                "com.mycom.MyServlet"); // Class
        dynamic.addMapping("/dynamicServlet"); // URL mapping
        dynamic.setAsyncSupported(true);
    }
}
```

Lookup & Configure

- `ServletContext#get[Servlet|Filter]Registration`
  - Takes [Servlet|Filter] name as argument
  - Returned Registration handle provides subset of configuration methods
    - May only be used to add initialization parameters and mappings

```
ServletRegistration declared = ServletContext
    .getServletRegistration("DeclaredServlet");
declared.addMapping("/declaredServlet");
declared.setInitParameter("param", "value");
```

Lab:

Exercise 2: Dynamic Registration
4532_javaee6_servlet3.0_basics.zip

Pluggability

Pluggability

- Enable use of 3rd party framework without extra configuration in web.xml
- Modularize *web.xml* to allow frameworks to be self-contained within their own JAR file
- Programmatic configuration APIs

Motivation of Pluggability

- Use of 3rd-party frameworks required (possibly complex) configuration in *web.xml* (in pre Servlet 3.0)
- For example, the framework had to
  - Declare framework controller Servlet
  - Declare framework logging and security Filters
  - Declare framework listeners to perform actions at various points in the lifecycle of the application
- Can get complex as dependencies increase
- Frameworks also need to document all the configuration that needs to be done
  - User of the framework has to set the configuration

web-fragment.xml (in Servlet 3.0)

- *web-fragment.xml* is descriptor for framework / library
- Included in *META-INF* directory of JAR file
- Container is then responsible for discovering fragments and assembling the effective deployment descriptor
- Almost identical to *web.xml*
  - Ordering related elements different
- Only JAR files in *WEB-INF/lib* considered as fragments

web-fragment.xml example

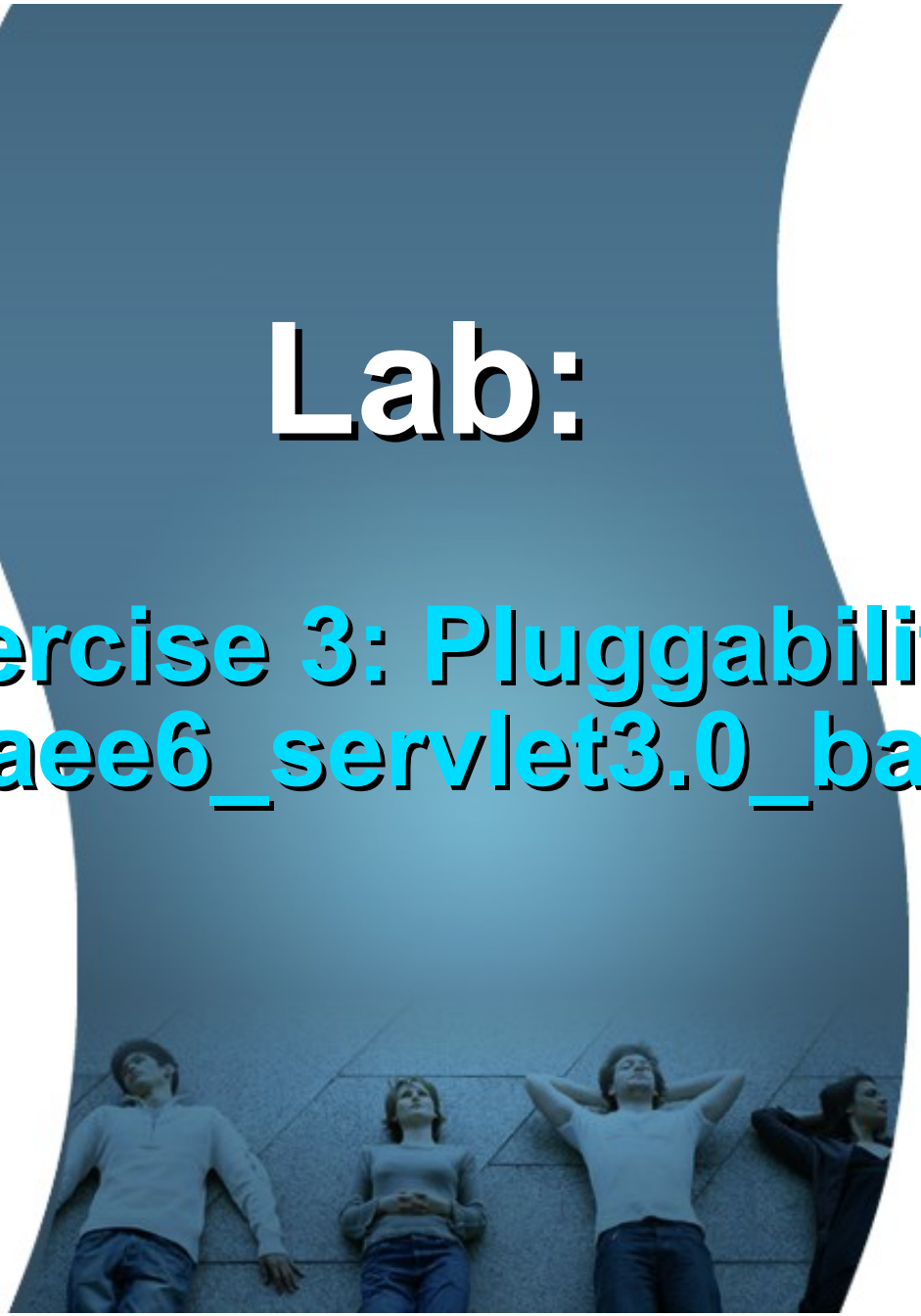
```
<web-fragment>
  <servlet>
    <servlet-name>welcome</servlet-name>
    <servlet-class>com.mycom.WelcomeServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>welcome</servlet-name>
    <url-pattern>/Welcome</url-pattern>
  </servlet-mapping>
  ...
</web-fragment>
```

Ordering of Fragments

- Fragments identified by <*name*>
- *web.xml* may declare absolute ordering of fragments via <*absolute-ordering*>
- Fragments may declare ordering preferences relative to other fragments via <*ordering*> with nested <*before*> and <*after*>
  - Ignored if <*absolute-ordering*> is specified
- Special <*others/*> element moves fragment to beginning or end of list of sorted fragments

Lab:

Exercise 3: Pluggability
4532_javaee6_servlet3.0_basics.zip



Resources in bundled jar files

Resources in bundled jar files

- Static and JavaServer™ Pages (JSP) resources no longer confined to web application's document root
- May be placed inside ***`WEB-INF/lib/[*.jar]/META-INF/resources`***
- Container must honor this new location when processing HTTP requests and calls to **`ServletContext#getResource or ServletContext#getResourceAsStream methods`**
- Resources in document root take precedence over those in bundled JAR files, however

Resources in bundled jar files: Example

`mywebapp.war` packaging:

`/index.jsp`

`/WEB-INF/lib/shared.jar`

`/META-INF/resources/shared.jsp`

Request for:

`http://localhost:8080/mywebapp/shared.jsp`

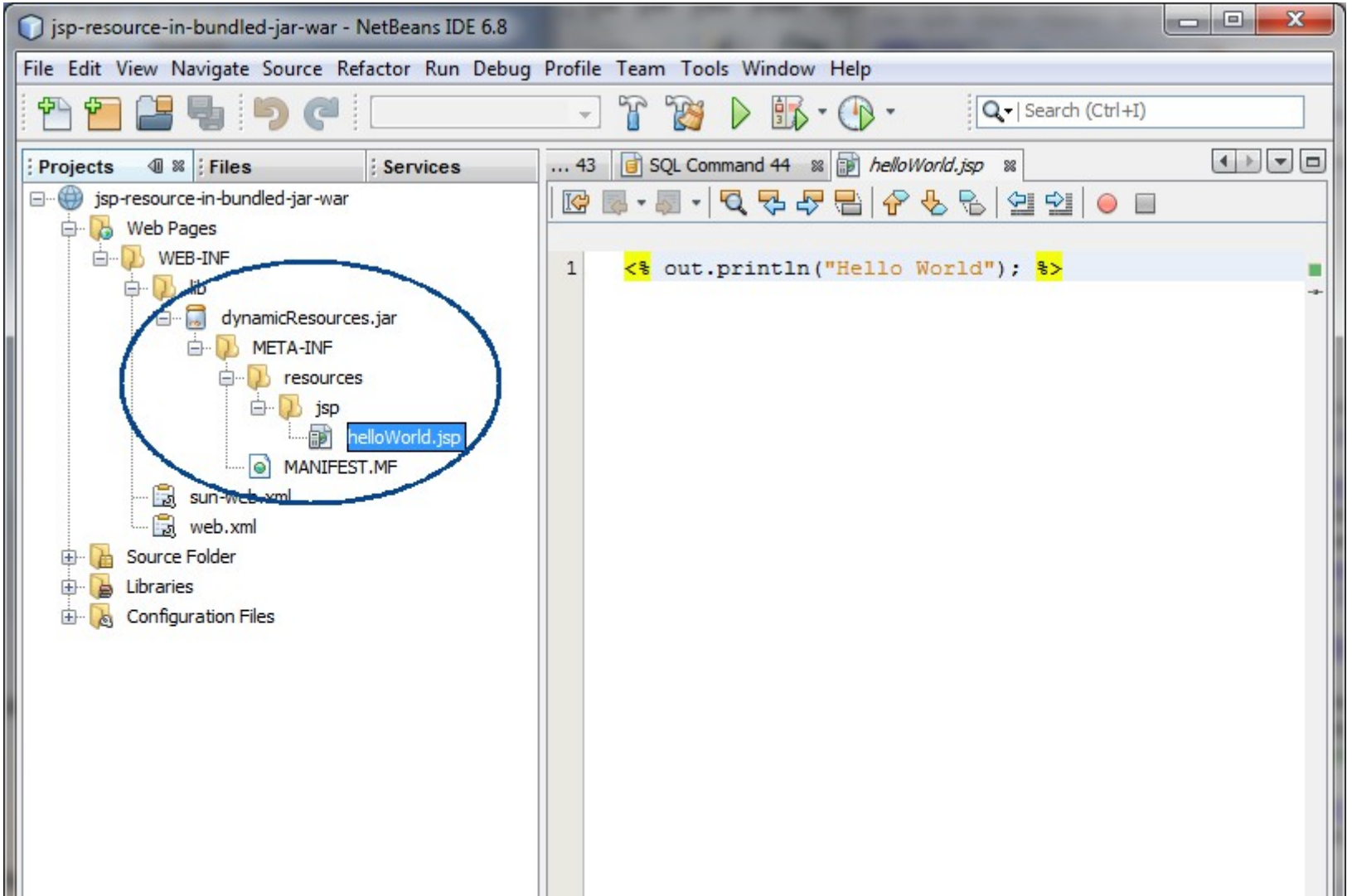
will be served from:

`/path/to/mywebapp/WEB-INF/lib/shared.jar`

`/META-INF/resources/shared.jsp`

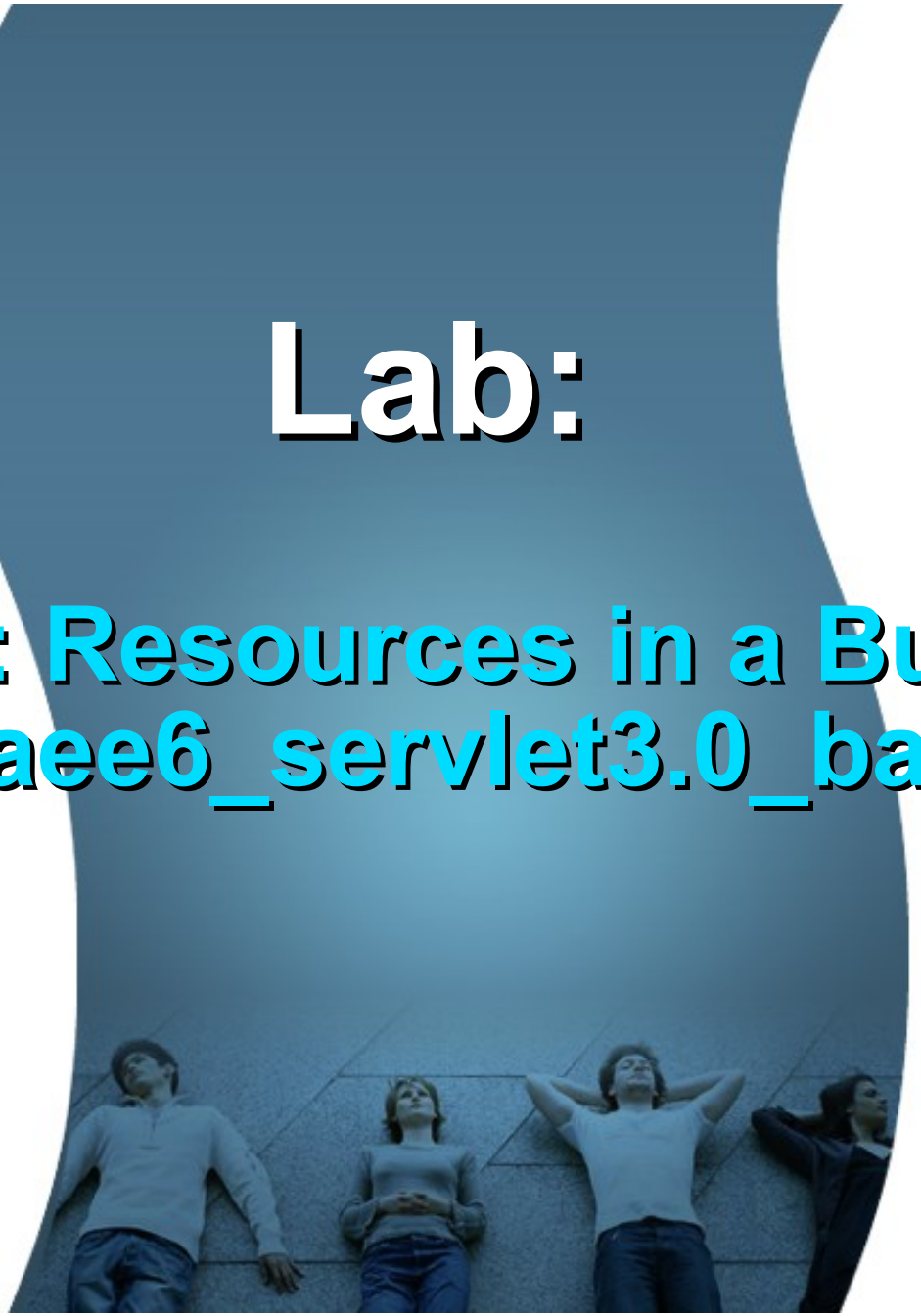
Resources in a Bundled jar

- The following can be accessible <URL>/jsp/helloWorld.jsp



Lab:

Exercise 4: Resources in a Bundled jar 4532_javaee6_servlet3.0_basics.zip



Code with Passion!
JPassion.com