Piotr Nowicki's Homepage

"Simplicity is the ultimate sophistication."

Java EE 6 SCWCD Mock Exam

Posted on March 27, 2011 by Piotr | **47** Replies **Edit**

This test might help to test your knowledge before taking the Oracle Certified Expert, JEE 6: JSP and Servlet Developer Exam.

Note that there are some well-designed and good quality SCWCD 5 mock exams already available on the net. However, the below questions, focus mainly on the new features which comes along with the Servlets 3.0 FR specification.

These questions were made to help myself in preparations for the new exam, so I hope that they can be used not only for testing purposes, but also as a learning material.

At the end of this post, you can find an attachment with some of the questions source code, so you can always check it for yourself. More interestingly, all of those questions were uploaded at my GitHub repository, so feel free to check it.

Each directory is named after the question number (q02, q11, etc.). In each directory you'll find *webapp* subdirectory which consists of ready-to-deploy exemplary code. If the question was related to the JAR files (i.e. web fragment questions), these will be located in the question's main directory.

Most of the examples were tested on Apache Tomcat 7 and some were tested on Glassfish v.3.1 and Resin 4.0.16.

In the answers, the "Reference" section points relevant parts of the Servlets 3.0 Final Release specification which should be helpful to understand the particular topic.

If you can, after taking this test, please leave a comment about what do you think about the questions and/or click the "like button" if you liked it. It would be definitely nice to know that additional work put into this mock exam preparation was worth it! ;-)

Have fun and good luck!

Important!

**Manjula Weerasinghe **was kind enough to create an alternative form of this exam in which you can view the correct answer right below the question. If this form is more adequate for you, please proceed to this page.

Thanks a lot Manjula!

**1. **Considering Servlets 3.0 and listeners invocation order, choose statements that are true:

a. the order is always unspecified,
b. the order is specified by alphabetically sorted listener implementation class names,
c. the order is unspecified when using annotations to define listeners,
d. the order can be defined only when using Deployment Descriptor to define listeners.

**2. **Considering the following HTML form code snippet and the servlet code, what will be the result of servlet invocation after the form has been submitted?

```
<!-- form.html -->

    <form action="myServlet?var=q1&var=q2" method="POST"
>
        <input name="var" type="hidden" value="q3" />
        <input type="submit" />
    </form>
```

```
package com.nullhaus;

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

@WebServlet("/myServlet")
public class NullServlet extends HttpServlet {
     public void doPost(HttpServletRequest req,
                        HttpServletResponse resp) {
          String param = req.getParameter("var");
          resp.getWriter().println("[" + param + "]");
     }
}
```

a. [q1, q2],
b. [q3],
c. [q2],
d. [q1],
e. [q1, q2, q3],

f. [q3, q2, q1],
g. [null],
h. the above code doesn't compile.

**3. **Considering the following HTML form code snippet and the servlet code, what will be the result of servlet invocation after the form has been submitted?

```
<!-- form.html -->

<form action="myServlet?var=q1&var=q2" method="POST">
    <input name="var" type="hidden" value="q3" />
    <input type="submit" />
</form>
```

```
package com.nullhaus;

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

@WebServlet("/myServlet")
public class NullServlet extends HttpServlet {
    public void doPost(HttpServletRequest req,
                       HttpServletResponse resp) {
        String[] param = req.getParameterValues("var");

        resp.getWriter().println(Arrays.toString(param));
    }
}
```

a. [q1, q2],
b. [q3],
c. [q2],
d. [q1],
e. [q1, q2, q3],
f. [q3, q2, q1],
g. [null],
h. the above code doesn't compile.

**4. **Assume that the Deployment Descriptor consists of the following mapping rules:

> /security/* => MyServlet

What will be the values of `HttpServletRequest#getServletPath()` and `HttpServletRequest#getPathInfo()` for the following request:

> /myApp/security/p.html?var=q1

a.	ServletPath = /security/p.html	PathInfo = /p.html
b.	ServletPath = /security/p.html?var=q1	PathInfo = /p.html?var=q1
c.	ServletPath = /security/p.html?var=q1	PathInfo = null
d.	ServletPath = /security	PathInfo = /p.html
e.	ServletPath = /security	PathInfo = /p.html?var=q

f. This mapping is invalid because the “security” is a reserved mapping for container authentication and authorisation purposes.

**5. **Considering Servlets 3.0, you can programmatically:

a. add servlets,
b. add filters,
c. add listeners,
d. instantiate servlets class,
e. instantiate filters class,
f. access already registered servlets and filters,
g. modify url patterns the servlets / filters maps to.

---

6. Considering Servlets 3.0, you can programmatically add servlets / filters:

a. only from `ServletContextListener`,
b. only from `ServletContainerInitializer`,
c. only from class which is configured with DD element or loadOnStartup attribute of the `@WebServlet` annotation with value > 0,
d. only a and b are correct,
e. only a and c are correct,
f. only b and c are correct,
g. a, b and c are all correct.

**7. **Considering Servlets 3.0, you can access registered servlets:

a. which were registered programatically,
b. which were registered using annotations,
c. which were registered using deployment descriptor and web fragments,
d. you cannot access already registered servlets,
e. none of the above is correct.

**8. **Considering Servlets 3.0, which listeners can be added programatically:

a. `ServletContextListener`,
b. `ServletContextAttributeListener`,
c. `ServletRequestListener`,
d. `ServletRequestAttributeListener`,
e. `HttpSessionActivationListener`,
f. `HttpSessionAttributeListener`,
g. `HttpSessionBindingListener`,
h. you cannot add listeners programatically.

**9. ** Consider the `test.jsp` page code shown below:

```
<%@page contentType="plain/text" %>
Hello ${world}!
```

What will be the result of the request made to the following servlet:

```
package com.nullhaus;

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

@WebServlet("/foo/*")
public class NullServlet extends HttpServlet {
    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp)
                              throws IOException {

        ServletContext ctx = getServletContext();
        InputStream is = ctx.getResourceAsStream("/test.j
sp");
        byte[] b = new byte[is.available()];
        is.read(b);

        resp.getWriter().print(new String(b));
    }
}
```

a. <%@page contentType="plain/text" %>Hello ${world}!
b. <%@page contentType="plain/text" %>Hello !
c. <%@page contentType="plain/text" %>Hello null!
d. (Java servlet code which is a result of the translation of test.jsp file),
e. The above code doesn't compile.

---

10. Consider the following Deployment Descriptor code snippet:

```
...
<filter>
    <filter-name>MyFilter 1</filter-name>
    <filter-class>com.nullhaus.MyFilter</filter-class>
</filter>
<filter>
    <filter-name>MyFilter 2</filter-name>
    <filter-class>com.nullhaus.MyFilter</filter-class>
</filter>
...
```

Knowing that the com.nullhaus.MyFilter is a valid Filter class, the above snippet is the only filter-related DD part and that there aren't any annotations related to the filter configuration, how many filter instances will be created by the Servlet container?

a.
b. 1
c. 2
d. Deployment Descriptor is invalid and the runtime exception will be thrown

---

**11. **Consider the following Deployment Descriptor code snippet:

```
...
<filter>
    <filter-name>MyFilter 1</filter-name>
    <filter-class>com.nullhaus.MyFilter</filter-class>
</filter>
<filter>
    <filter-name>MyFilter 1</filter-name>
    <filter-class>com.nullhaus.MyFilter2</filter-class>
</filter>
...
```

Knowing that the com.nullhaus.MyFilter and com.nullhaus.MyFilter2 are valid Filter classes, the above snippet is the only filter-related DD part and that there aren't any annotations related to the filter configuration, how many filter instances will be created by the Servlet container?

a.
b. 1
c. 2
d. Deployment Descriptor is invalid and the runtime exception will be thrown

---

**12. **Consider the following Deployment Descriptor code snippet:

```
...
<filter>
    <filter-name>MyFilter 1</filter-name>
    <filter-class>com.nullhaus.MyFilter</filter-class>
</filter>
<filter>
    <filter-name>MyFilter 1</filter-name>
    <filter-class>com.nullhaus.MyFilter</filter-class>
</filter>
...
```

Knowing that the com.nullhaus.MyFilter is a valid Filter class, the above snippet is the only filter-related DD part and that there aren't any annotations related to the filter configuration, how many filter instances will be created by the Servlet container?

a.
b. 1
c. 2
d. Deployment Descriptor is invalid and the runtime exception will be thrown

---

**13. **Consider the following Deployment Descriptor code snippet:

```
...
<filter>
    <filter-name>NullFilter 1</filter-name>
    <filter-class>com.nullhaus.NullFilter</filter-class>
</filter>
...
```

and the following Filter code:

```
package com.nullhaus;

import javax.servlet.annotation.*;
import javax.servlet.*;

@WebFilter("/*")
public class NullFilter implements Filter {
        // necessary methods goes here
}
```

Knowing that above DD snippet is the only filter-related part, how many filter instances will be created by the Servlet container?

a.
b. 1
c. 2
d. Deployment Descriptor is invalid and the runtime exception will be thrown

---

**14. **Consider the following Deployment Descriptor code snippet:

```
...
<filter>
    <filter-name>NullFilter 1</filter-name>
    <filter-class>com.nullhaus.NullFilter</filter-class>
</filter>
...
```

and the following Filter code:

```
package com.nullhaus;

import javax.servlet.annotation.*;
import javax.servlet.*;

@WebFilter(urlPatterns = {"/*"}, filterName="NullFilter 1")
public class NullFilter implements Filter {
        // necessary methods goes here
}
```

Knowing that above DD snippet is the only filter-related part, how many filter instances will be created by the Servlet container?

a.
b. 1
c. 2
d. Deployment Descriptor is invalid and the runtime exception will be thrown

---

**15. **Consider the below filter-mapping definition in the Deployment Descriptor:

```
<filter>
    <filter-name>MyFilter 1</filter-name>
    <filter-class>com.nullhaus.MyFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>MyFilter 1</filter-name>
    <url-pattern>/bar/*</url-pattern>
    <servlet-name>YourServlet</servlet-name>
```

```
    <url-pattern>/baz/*</utl-pattern>
</filter-mapping>
```

What is true about the above DD snippet, assuming that MyFilter 1 is valid and runs without a grasp:

a. this DD is valid,
b. this DD is valid but will throw a runtime exception when “Myfilter 1″ url pattern will be matched,
c. this DD is invalid, because there can’t be a element in ,
d. this DD is invalid, because there can’t be a element together with element in ,
e. this DD is invalid, because there can’t be more than one element in .

**16. **Which of the following values are valid `RequestDispatcher` types:

a. REQUEST,
b. FORWARD,
c. ASYNC,
d. INCLUDE,
e. JSP,
f. ERROR.

---

17. Consider the following Filter code:

```
package com.nullhaus;

import javax.servlet.annotation.*;
import javax.servlet.*;

@WebFilter(servletNames = {"*"},
           filterName="NullHaus Filter",
           dispatcherTypes = {DispatcherType.INCLUDE})
public class NullFilter implements Filter {
    // necessary methods goes here
}
```

Assuming that there aren’t any filter-related parts in the DD, which statements are true:

a. The “NullHaus Filter” will be accessed only if the `RequestDispatcher#include(-)` method is invoked from a `RequestDispatcher` obtained by name,
b. The “NullHaus Filter” will be accessed only if the `RequestDispatcher#include(-)` method is invoked from a `RequestDispatcher` obtained by path,

c. The “NullHaus Filter” will be accessed if the `RequestDispatcher#include(-)` method is invoked from a `RequestDispatcher` obtained either by name or path,
d. The “*” is not a valid `servletNames` attribute value and a runtime exception will be thrown.
e. The above filter annotation is functionally equal to the below DD part:

```
<filter>
    <filter-name>NullHaus Filter</filter-name>
    <filter-class>com.nullhaus.NullFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>NullHaus Filter</filter-name>
    <servlet-name>*</servlet-name>
    <dispatcher>INCLUDE</dispatcher>
</filter-mapping>
```

**18. **Consider the following element in the Deployment Descriptor:

```
<filter-mapping>
    <filter-name>NullHaus Filter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

a. This filter-mapping is valid and the “NullHaus Filter” will be accessed each time a request is made to the web application,
b. This filter-mapping is valid and the “NullHaus Filter” will be accessed only if the request is made to the web application root context (/),
c. This filter-mapping is valid and the “NullHaus Filter” will be accessed each time a request is dispatched from a `RequestDispatcher` obtained either by name or path,
d. This filter-mapping is invalid.

**19. **Considering the ordering of web fragments, which statements are true:

a. The order of web fragments scanning/discovering is always unspecified, and it depends on the container specific implementation,
b. The order of web fragments scanning/discovering is always specified, and it depends on the web-fragment.xml’s element value,
c. The order of web fragments scanning/discovering is always specified, and it depends on the web-fragment.xml’s element value,
d. The order of web fragments scanning/discovering is always specified, and it depends on the alphabetical order of JARs in which the web-fragments are located in,
e. The order of web fragments scanning/discovering is unspecified by

default, but the ordering rules can be specified in the Deployment Descriptor.

**20. **Considering the idea of web fragments, which statements are true:

a. web fragment's filename, to be discovered by the container, must be named "`web-fragment.xml`",
b. web fragment's filename, to be discovered by the container, must be named "`web_fragment.xml`",
c. If a web fragment is packaged as a JAR file, its web fragment XML file needs to be located at the top directory of the JAR file,
d. If a web fragment is packaged as a JAR file, its web fragment XML file needs to be located directly under META-INF/ directory of the JAR file,
e. If a web fragment packaged as a JAR file needs to be discovered by the container, it must be located somewhere in the application's classpath,
f. If a web fragment packaged as a JAR file needs to be discovered by the container, it must be located directly under WEB-INF/ directory of the application,
g. If a web fragment packaged as a JAR file needs to be discovered by the container, it must be located directly under WEB-INF/lib directory of the application.

**21. **What will be the order in which the container will scan and combine web fragments to form the effective Deployment Descriptor (attributes for and intentionally omitted; assume the default values):

web.xml

```
<web-app>
</web-app>
```

web-fragment.xml

```
<web-fragment>
    <name>Fragment 1</name>

    <absolute-ordering>
        <name>Fragment 2</name>
    </absolute-ordering>
</web-fragment>
```

web-fragment.xml

```
<web-fragment>
    <name>Fragment 2</name>
</web-fragment>
```

a. web.xml, Fragment 1, Fragment 2,
b. web.xml, Fragment 2, Fragment 1,
c. Fragment 1, Fragment 2, web.xml,
d. Fragment 2, Fragment 1, web.xml,
e. web.xml,
f. web.xml, Fragment 1,
g. web.xml, Fragment 2,
h. At least one of the above Deployment Descriptors is invalid.

**22. **What will be the order in which the container will scan and combine web fragments to form the effective Deployment Descriptor (attributes for and intentionally omitted; assume the default values):

web.xml

```
<web-app>
    <absolute-ordering>
        <name>Fragment 2</name>
    </absolute-ordering>
</web-app>
```

web-fragment.xml

```
<web-fragment>
    <name>Fragment 1</name>
</web-fragment>
```

web-fragment.xml

```
<web-fragment>
    <name>Fragment 2</name>
</web-fragment>
```

a. web.xml, Fragment 1, Fragment 2,
b. web.xml, Fragment 2, Fragment 1,
c. Fragment 1, Fragment 2, web.xml,
d. Fragment 2, Fragment 1, web.xml,
e. web.xml,
f. web.xml, Fragment 1,
g. web.xml, Fragment 2,
h. At least one of the above Deployment Descriptors is invalid.

**23. **What will be the order in which the container will scan and combine web fragments to form the effective Deployment Descriptor (attributes for and intentionally omitted; assume the default values):

web.xml

```
<web-app metadata-complete="false">
    <absolute-ordering>
        <name>Fragment 1</name>
        <name>Fragment 2</name>
    </absolute-ordering>
</web-app>
```

web-fragment.xml

```
<web-fragment>
    <name>Fragment 1</name>
    <ordering>
         <after>Fragment 2</after>
    </ordering>
</web-fragment>
```

web-fragment.xml

```
<web-fragment>
    <name>Fragment 2</name>
</web-fragment>
```

a. web.xml, Fragment 1, Fragment 2,
b. web.xml, Fragment 2, Fragment 1,
c. Fragment 1, Fragment 2, web.xml,
d. Fragment 2, Fragment 1, web.xml,
e. web.xml,
f. At least one of the above Deployment Descriptors is invalid.

**24. **What will be the order in which the container will scan and combine web fragments to form the effective Deployment Descriptor (attributes for and intentionally omitted; assume the default values):

web.xml

```
<web-app metadata-complete="true">
    <absolute-ordering>
        <name>Fragment 1</name>
        <name>Fragment 2</name>
    </absolute-ordering>
</web-app>
```

web-fragment.xml

```
<web-fragment>
    <name>Fragment 1</name>
    <ordering>
         <after>Fragment 2</after>
    </ordering>
</web-fragment>
```

web-fragment.xml

```
<web-fragment>
    <name>Fragment 2</name>
</web-fragment>
```

a. web.xml, Fragment 1, Fragment 2,
b. web.xml, Fragment 2, Fragment 1,
c. Fragment 1, Fragment 2, web.xml,
d. Fragment 2, Fragment 1, web.xml,
e. web.xml,
f. At least one of the above Deployment Descriptors is invalid and will thrown a runtime exception.

**25. **What will be the order in which the container will scan and combine web fragments to form the effective Deployment Descriptor (attributes for and intentionally omitted; assume the default values):

web.xml

```
<web-app>
    <absolute-ordering>
        <name>Fragment 1</name>
        <name>Fragment 2</name>
    </absolute-ordering>
</web-app>
```

web-fragment.xml

```
<web-fragment metadata-complete="true">
    <name>Fragment 1</name>
    <ordering>
         <after>Fragment 2</after>
    </ordering>
</web-fragment>
```

web-fragment.xml

```
<web-fragment>
```

```
    <name>Fragment 2</name>
</web-fragment>
```

a. web.xml, Fragment 1, Fragment 2,
b. web.xml, Fragment 2, Fragment 1,
c. Fragment 1, Fragment 2, web.xml,
d. Fragment 2, Fragment 1, web.xml,
e. web.xml,
f. At least one of the above Deployment Descriptors is invalid and will thrown a runtime exception.

**26. **Choose statements that are true about the `metadata-complete` attribute — for element in `web.xml`:

a. This attribute is of a boolean type which takes true/false values only,
b. This attribute can define if the container should scan for `web-fragments.xml` to create final Deployment Descriptor,
c. This attribute can define if the container should scan and process the new Servlets 3.0 annotations like `@WebFilter`, `@WebServlet`, etc.,
d. This attribute is purely informational and does not affect how container processes the final Deployment Descriptor,
e. There is no attribute metadata-complete, but element within the .

**27. **Considering the following Servlet code, choose the statements which are true:

```
package com.nullhaus;

import javax.servlet.annotation.*;
import javax.servlet.http.*;

@WebServlet("nullHausServlet")
public class NullServlet extends HttpServlet {
}
```

a. This is valid usage of `@WebServlet` annotations which creates a Servlet with “nullHausServlet” name,
b. This is valid usage of `@WebServlet` annotations which creates a Servlet with “nullHausServlet” url-pattern value,
c. This is an invalid usage of `@WebServlet` annotations because of the wrongly formed url-pattern value,
d. This code doesn’t compile, because NullHausServlet need to implement one of doGet(-), doPost(-), etc. methods,
e. This code doesn’t compile, because the value of `@WebServlet` annotation attribute (“nullHausServlet”) must be defined using `@WebServlet(value = “nullHausServlet”)` construct,

f. This code doesn't compile, because there is no `@WebServlet` annotation, but `@Servlet`.

**28. **Considering the following Servlet code, choose the statements which are true:

```
package com.nullhaus;

import javax.servlet.annotation.*;
import javax.servlet.http.*;

@WebServlet(value = "nullHausServlet")
public class NullServlet extends HttpServlet {
}
```

a. This is valid usage of `@WebServlet` annotations which creates a Servlet with "nullHausServlet" name,
b. This is valid usage of `@WebServlet` annotations which creates a Servlet with "nullHausServlet" url-pattern value,
c. This is an invalid usage of `@WebServlet` annotations because of the wrongly formed url-pattern value,
d. This is an invalid usage of `@WebServlet` annotations because the "value" attribute cannot be used explicitly in the annotation,
e. This code doesn't compile, because NullHausServlet need to implement one of doGet(-), doPost(-), etc. methods,
f. This code doesn't compile, because there is no `@WebServlet` annotation, but `@Servlet`.

**29. **Considering the following Servlet code, choose the statements which are true:

```
package com.nullhaus;

import javax.servlet.annotation.*;
import javax.servlet.http.*;

@WebServlet(urlPatterns="/nullHausServlet")
class NullHausServlet extends HttpServlet {
}
```

a. This is a valid usage of `@WebServlet` annotation which runs fine,
b. This is an invalid usage of `@WebServlet` annotation, because of the wrongly formed `url-pattern` value,
c. This is an invalid usage of `@WebServlet` annotation, because there is a "`urlPattern`" attribute – not "`urlPatterns`",
d. This is an invalid usage of `@WebServlet` annotation, because the

"`urlPatterns`" attribute should be an array of Strings – not a single String value,

e. This is a valid usage of `@WebServlet` annotation, but the servlet can't be accessed,
f. The name of this servlet is "`com.nullhaus.NullHausServlet`",
g. This code doesn't compile.

**30. **Considering the following Servlet code, choose the statements which are true:

```
package com.nullhaus;

import javax.servlet.annotation.*;
import javax.servlet.http.*;

@WebServlet(urlPatterns = {"/nullServlet"}, value="/nume
roDuo")
public class NullServlet extends HttpServlet {
}
```

a. This is a valid usage of `@WebServlet` annotation which runs fine,
b. This is an invalid usage of `@WebServlet` annotation, because of the wrongly formed `url-pattern` value,
c. This is an invalid usage of `@WebServlet` annotation, because there is a "`urlPattern`" attribute – not "`urlPatterns`",
d. This is an invalid usage of `@WebServlet` annotation, because the `urlPatterns` and `value` attributes cannot be defined together,
e. This code doesn't compile.

**31. **//\ **||Considering the following Servlet code, choose the statements which are true:

```
package com.nullhaus;

import javax.servlet.annotation.*;
import javax.servlet.http.*;

@WebServlet(name="NullServlet")
public class NullServlet extends HttpServlet {
}
```

a. This is a valid usage of `@WebServlet` annotation,
b. This is an invalid usage of `@WebServlet` annotation,
c. This code compiles,
d. This code doesn't compile.

**32. **Considering the following Servlet code and the Deployment Descriptor snippet, choose the statements which are true:

```
package com.nullhaus;

import javax.servlet.annotation.*;
import javax.servlet.http.*;

@WebServlet(urlPatterns={"/foo/*"}, name="NullHaus1")
public class NullHausServlet extends HttpServlet {
}
```

```
<servlet>
    <servlet-class>com.nullhaus.NullHausServlet</servlet-
class>
    <servlet-name>NullHaus1</servlet-name>
</servlet>

<servlet-mapping>
    <servlet-name>NullHaus1</servlet-name>
    <url-pattern>/baz/*</url-pattern>
</servlet-mapping>
```

a. There will be exactly one instance of the `NullHausServlet`,
b. There will be exactly two instances of the `NullHausServlet`,
c. There will be at least one instances of the `NullHausServlet`,
d. There will be at least two instances of the `NullHausServlet`,
e. The `NullHausServlet` will be accessible only from /foo/* url,
f. The `NullHausServlet` will be accessible only from /baz/* url,
g. The `NullHausServlet` will be accessible from /foo/* and /baz/* urls,
h. There will be a runtime exception thrown and NullHaus1 servlet will not be operational.

**33. **//\ **||Considering the following Servlet code and the Deployment Descriptor snippet, choose the statements which are true:

```
package com.nullhaus;

import javax.servlet.annotation.*;
import javax.servlet.http.*;

@WebServlet(urlPatterns={"/foo/*"}, name="NullHaus1")
public class NullHausServlet extends HttpServlet {
}
```

```
<servlet>
```

```
    <servlet-class>com.nullhaus.NullHausServlet</servlet-
class>
    <servlet-name>NullHaus2</servlet-name>
</servlet>

<servlet-mapping>
    <servlet-name>NullHaus2</servlet-name>
    <url-pattern>/baz/*</url-pattern>
</servlet-mapping>
```

a. There will be exactly one instance of the `NullHausServlet`,
b. There will be exactly two instances of the `NullHausServlet`,
c. There will be at least two instances of the `NullHausServlet`,
d. There will be at most two instances of the `NullHausServlet`,
e. There will be a runtime exception thrown and NullHaus1 and NullHaus2 will not be operational.

**34. **Consider the following Servlet code and the `ServletContainerInitializer` code. Assume that the `MyInit` class is properly registered in the container as a `ServletContainerInitializer`. Choose the statements which are true:

```
package com.nullhaus;

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

@WebServlet(value = "/foo/*", name="NullHaus1")
public class MyJar1Servlet extends HttpServlet {
}
```

```
package com.nullhaus;

import javax.servlet.*;
import java.util.*;

public class MyInit implements ServletContainerInitializer
 {
  public void onStartup(Set<Class<?>> c, ServletContext c
tx)
                                        throws ServletExcep
tion {
    try {
```

```
        Class klass = Class.forName("com.nullhaus.MyJar1Servlet");
        Class<MyJar1Servlet> clazz = (Class<MyJar1Servlet>)klass;

        Servlet s = ctx.createServlet(clazz);
        ServletRegistration.Dynamic d =
                                    ctx.addServlet("NullHaus2", s);

        d.addMapping("/baz/*");
    } catch (ClassNotFoundException e) {
        // ...
    }
  }
}
```

a. There will be at least one instance of the `MyJar1Servlet` named NullHaus1,
b. There will be at least one instance of the `MyJar1Servlet` named NullHaus2,
c. There will be exactly two instances of the `MyJar1Servlet` named NullHaus1 and NullHaus2 respectively,
d. A runtime exception will be thrown,
e. This code doesn't compile.

**35. **Consider the following Servlet code and the `ServletContainerInitializer` code. Assume that the `MyInit` class is properly registered in the container as a `ServletContainerInitializer`. Choose the statements which are true:

```
package com.nullhaus;

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

@WebServlet(value = "/foo/*", name="NullHaus1")
public class MyJar1Servlet extends HttpServlet {
}
```

```
package com.nullhaus;

import javax.servlet.*;
import java.util.*;
```

```
public class MyInit implements ServletContainerInitializer {
  public void onStartup(Set<Class<?>> c, ServletContext ctx)
                                        throws ServletException
{
    try {
       Class klass = Class.forName("com.nullhaus.MyJar1Servlet"
);
       Class<MyJar1Servlet> clazz = (Class<MyJar1Servlet>)klass
;

       Servlet s = ctx.createServlet(clazz);
       ServletRegistration.Dynamic d =
                                  ctx.addServlet("NullHaus1", s)
;

       d.addMapping("/baz/*");
    } catch (ClassNotFoundException e) {
       // ...
    }
  }
}
```

a. There will be at least one instance of the `MyJar1Servlet` named NullHaus1,
b. The number of `MyJar1Servlet` instances is unspecified,
c. This code doesn't compile.

**36. **Which statements are true about classes annotated with `@WebServlet`?

a. they must extend the `javax.servlet.GenericServlet` class,
b. they must extend the `javax.servlet.http.HttpServlet` class,
c. they must implement the `javax.servlet.Servlet` interface,
d. none of the above is correct.

**37. **Consider the following servlet code:

```
package com.nullhaus;

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

@WebServlet(urlPatterns={"/foo/*"},
            name="NullHaus1",
            initParams=@WebInitParam(name="var1", value=
"Howdy!"))
```

```
public class NullHausServlet extends HttpServlet {
  public void doGet(HttpServletRequest req,
                    HttpServletResponse resp) throws IOEx
ception {
     String param1 = getInitParameter("var1");
     String param2 = getServletContext().getInitParamete
r("var1");
     resp.getWriter().print("Values: " + param1 + ", " +
 param2);
  }
}
```

Choose what will be the result of the code execution:

a. Values: null, null,
b. Values: null, Howdy!,
c. Values: Howdy!, null,
d. Values: Howdy!, Howdy!,
e. Runtime exception will be thrown,
f. This code doesn't compile.

**38. **Consider the following servlet code:

```
package com.nullhaus;

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

@WebInitParam(name="var1", value="Howdy!")
@WebServlet(urlPatterns={"/foo/*"},
            name="NullHaus1")
public class NullHausServlet extends HttpServlet {
  public void doGet(HttpServletRequest req,
                    HttpServletResponse resp) throws IOExceptio
n {
     String param1 = getInitParameter("var1");
     String param2 = getServletContext().getInitParameter("var1
");
     resp.getWriter().print("Values: " + param1 + ", " + param2
);
    }
}
```

Choose what will be the result of the code execution:

a. Values: null, null,
b. Values: null, Howdy!,
c. Values: Howdy!, null,
d. Values: Howdy!, Howdy!,

e. Runtime exception will be thrown,
f. This code doesn't compile.

**39. **Consider the following servlet code:

```
package com.nullhaus;

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

@WebInitParam(name="var1", value="Howdy!")
@WebInitParam(name="var2", value="Rancher!")
@WebServlet(urlPatterns={"/foo/*"},
            name="NullHaus1")
public class NullHausServlet extends HttpServlet {
  public void doGet(HttpServletRequest req,
                    HttpServletResponse resp) throws IOExceptio
n {
     String param1 = getInitParameter("var1");
     String param2 = getInitParameter("var2");
     resp.getWriter().print("Values: " + param1 + ", " + param2
);
  }
}
```

Choose what will be the result of the code execution:

a. Values: null, null,
b. Values: null, Rancher!,
c. Values: Howdy!, null,
d. Values: Howdy!, Rancher!,
e. Runtime exception will be thrown,
f. This code doesn't compile.

**40. **Considering the web fragments ordering rules, which statements are true:

a. The element can be placed within the `web.xml` file,
b. The element can be placed within the `web-fragment.xml` file,
c. The element can be placed within the `web.xml` file,
d. The element can be placed within the `web-fragment.xml` file,
e. The only possible elements of are , and ,
f. The only possible elements of the element are and ,
g. If there are no nor elements defined in `web.xml` and `web-fragment.xml`, the order of web fragments scanning is unspecified.

---

41. Which statements are true:

a. The element (subelement of) set to “false” forces the container to make the servlet unreachable for request to the defined url-pattern,
b. The element (subelement of) set to “false” forces the container to make the request for the servlet respond with HTTP Code 503 (Service unavailable),
c. The doesn’t have an subelement,
d. The web fragment is merged into the final Deployment Descriptor before the web fragment related annotations are processed,
e. The web fragment is merged into the final Deployment Descriptor after the web fragment related annotations are processed,
f. All web fragments are processed together (in a batch) and all are merged into the final Deployment Descriptor before the web fragments’ related annotations are processed.

---

42. Considering the following web fragments (attributes intentionally removed):

```
<web-fragment>
    <servlet>
        <servlet-name>NullHaus Servlet</servlet-name>
        <servlet-class>com.nullhaus.NullServlet</servlet-
class>
        <init-param>
            <param-name>myParam</param-name>
            <param-value>test1</param-name>
        </init-param>
    </servlet>
</web-fragment>
```

```
<web-fragment>
    <servlet>
        <servlet-name>NullHaus Servlet</servlet-name>
        <servlet-class>com.nullhaus.NullServlet</servlet-
class>
        <init-param>
            <param-name>myParam</param-name>
            <param-value>test2</param-name>
        </init-param>
    </servlet>
</web-fragment>
```

What will be the result of the request made to the following servlet:

```
package com.nullhaus;
```

```
// necessary imports goes here
@WebServlet(urlPatterns={"/foo/*"}, name="NullHaus Servl
et")
public class NullServlet extends HttpServlet {
    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp) {
        String myParam = getInitParameter("myParam");

        resp.getWriter().println(myParam);
    }
}
```

a. test1,
b. test2,
c. test1, test2,
d. test2, test1,
e. test1 and test2, but the order of these values is unspecified,
f. null,
g. The web fragments are invalid and the application will not be deployed,
h. The servlet code will not compile.

**43. **Choose the true statements about the Deployment Descriptor and annotations:

a. If servlet A defines init param named PARAM through annotations, and servlet B defines init param named PARAM through the Deployment Descriptor, the Deployment Descriptor value has precedence,
b. If servlet A defines init param named PARAM through annotations, and servlet B defines init param named PARAM through the Deployment Descriptor, the annotation value has precedence,
c. Init params with different names, defined in annotations and in the Deployment Descriptor are additive (all init params will be present in the final Deplyment Descriptor),
d. url-patterns with different values, defined in annotations and in the Deployment Descriptor are additive (all url patterns will be present in the final Deplyment Descriptor).

**44. **Consider the following `testJar.jar` structure:

> /META-INF/services
> /javax.servlet.initializer.ServletContainierInitializer
> /com/nullhaus/MyInitializer.class

Assuming that:

- `MyInitializer.class` is a class which properly implements

`ServletContainerInitializer` interface,
- `javax.servlet.initializer.ServletContainerInitializer` file consists of the following content:

> com.nullhaus.MyInitializer

Choose the statements which are true:

a. This is a correct usage of JAR Services API for the `ServletContainerInitializer`,
b. This is an incorrect usage of JAR Services API for the `ServletContainerInitializer`,
c. The MyInitializer class will be invoked if the `testJar.jar` will be put somewhere in the web application classpath,
d. The MyInitializer class will be invoked if the `testJar.jar` will be put in the web application WEB-INF/lib directory.

**45. **Which statements are true about the following request attributes:

> javax.servlet.forward.request_uri
> javax.servlet.forward.context_path
> javax.servlet.forward.servlet_path
> javax.servlet.forward.path_info
> javax.servlet.forward.query_string

a. These are all valid request attributes which must be set by the container after the request is forwarded using `RequestDispatcher#forward(-)` method,
b. Some or all of the above request attributes doesn't have to be set by the container during request forwarding using `RequestDispatcher#forward(-)` method,
c. Some or all of the above request attributes are valid if the word "forward" will be replaced with "include",
d. Some or all of the above request attributes are valid if the word "forward" will be replaced with "async",
e. Some or all of the above request attributes are always holding the original request information even if multiple forwarding operations were made.

**46. **Consider the following `testJar.jar` structure:

> /index0.html
> /resources/index1.html
> /META-INF/index2.html
> /META-INF/resources/index3.html

Assume that this `testJar.jar` is placed under the “myApp” web application WEB-INF/lib directory.

Which statements are true:

a. The resource index0.html is accessible by /myApp/index0.html request invocation,
b. The resource index0.html is accessible by /myApp/testJar/index0.html request invocation,
c. The resource index1.html is accessible by /myApp/testJar/resources /index1.html request invocation,
d. The resource index1.html is accessible by /myApp/resources /index1.html request invocation,
e. The resource index2.html is accessible by /myApp/testJar/index2.html request invocation,
f. The resource index2.html is accessible by /myApp/index2.html request invocation,
g. The resource index3.html is accessible by /myApp/testJar/resources /index3.html request invocation,
h. The resource index3.html is accessible by /myApp/index3.html request invocation.

**47. **Which statements are true about the web applications created with Servlets 3.0:

a. The web.xml file is required,
b. The web.xml file is not required,
c. The WEB-INF/ directory is required,
d. The WEB-INF/ directory is not required.

**48. **Which statements are true about the listeners in Servlets 3.0:

a. Every listener implementor must provide a public, no argument constructor,
b. Every listener implementor must not provide any constructor; a default, compiler generated one should be created,
c. All listeners can be registered within the Deployment Descriptor,
d. The listeners are invoked in the order in which they were registered.

**49. **Considering Servlets 3.0 Security, which statements are true:

a. Security constraints can be managed through Deployment Descriptor,
b. Security constraints can be managed programmatically, using `ServletRegistration.Dynamic#setServletSecurity(-)` method,
c. Security constraints can be managed using `@WebServletSecurity` annotation,
d. The `@WebServletSecurity` annotation have three attributes:

`value`, `httpMethodConstraints` and `rolesAllowed`.

**50. **Consider the following Servlet code:

```
package com.nullhaus;

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

@HttpConstraint(EmptyRoleSemantic.DENY)
@WebServlet(value = "/foo/*", name = "NullServlet")
public class NullServlet extends HttpServlet {
  public void doGet(HttpServletRequest req,
                    HttpServletResponse resp) throws IOExc
eption {
     resp.getWriter().print("Howdy Stragers!");
  }
}
```

Choose statements which are true about the GET HTTP request:

a. This servlet is accessible for all users,
b. This servlet is not accessible for any users,
c. The `EmptyRoleSemantic.DENY` is not a valid `@HttpConstraint` main (“value”) attribute value,
d. A runtime exception will be thrown while trying to access the servlet,
e. The above code doesn’t compile.

**51. **Consider the following Servlet code:

```
package com.nullhaus;

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

@ServletSecurity(value = @HttpConstraint(EmptyRoleSemantic.DENY
),
     httpMethodConstraints = {@HttpMethodConstraint(
                    methodName = "GET",
                    emptyRoleSemantic = EmptyRoleSemantic.ALLOW)
})

@WebServlet(value = "/foo/*", name = "NullServlet")
public class NullServlet extends HttpServlet {
```

```
    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp) throws IOExceptio
n {
        resp.getWriter().print("Howdy Stragers!");
    }
}
```

Choose statements which are true about the GET HTTP request made to the NullServlet:

a. This servlet is accessible for all users,
b. This servlet is not accessible for any users,
c. The `EmptyRoleSemantic.DENY` is not a valid `@HttpConstraint` main (“value”) attribute value,
d. A runtime exception will be thrown while trying to access the servlet,
e. The above code doesn’t compile.

**52. **Consider the following Servlet code:

```
package com.nullhaus;

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

@ServletSecurity(value = @HttpConstraint(EmptyRoleSemantic.DENY
),
     httpMethodConstraints = {@HttpMethodConstraint(value = "GE
T",
     emptyRoleSemantic = EmptyRoleSemantic.PERMIT)})

@WebServlet(value = "/foo/*", name = "NullServlet")
public class NullServlet extends HttpServlet {
    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp) throws IOExceptio
n {
        resp.getWriter().print("Howdy Stragers!");
    }
}
```

Choose statements which are true about the servlet’s HTTP GET request:

a. This servlet is accessible for all users,
b. This servlet is not accessible for any users,
c. The `EmptyRoleSemantic.DENY` is not a valid `@HttpConstraint` main (“value”) attribute value,
d. A runtime exception will be thrown while trying to access the servlet,
e. The above code doesn’t compile.

**53. **Considering Servlets 3.0 security constraints annotations, which statements are true:

a. Valid `@HttpConstraint` attributes are: value, rolesAllowed and transportGuarantee,
b. Valid `@HttpMethodConstraint` attributes are: value, rolesAllowed, transportGuarantee and emptyRoleSemantic,
c. The default value for `@HttpConstraint#transportGuarantee` is `TransportGuarantee.CONFIDENTIAL`,
d. The default value for `@HttpMethodConstraint#emptyRoleSemantic` is `EmptyRoleSemantic#DENY`.

**54. **Considering the following security constraints, which statements are true:

```
@ServletSecurity(
  value = @HttpConstraint(
            value = EmptyRoleSemantic.DENY,
            transportGuarantee = TransportGuarantee.CONFI
DENTIAL),
            httpMethodConstraints = {
                @HttpMethodConstraint(
                    value = "GET",
                    rolesAllowed = "manager"),
                @HttpMethodConstraint(
                    value = "POST",
                    rolesAllowed = "*",
                    emptyRoleSemantic = EmptyRoleSemantic
.PERMIT)
             }
          )
@WebServlet("/foo")
public class MyServlet extends HttpServlet {
}
```

a. For all HTTP methods other than GET and POST, and for all roles, access to this servlet is forbidden and the confidential transport is required,
b. For all HTTP methods and for all roles, access to this servlet is forbidden and the confidential transport is required,
c. For GET HTTP method request, the access to this servlet is forbidden if the role is different than “manager”,
d. For GET HTTP method request, the access to this servlet is forbidden if the role is equal to “manager”,
e. For POST HTTP method request, the access to this servlet is forbidden if the role is different than “manager”,
f. For POST HTTP method request, the access to this servlet is allowed

with no further roles restrictions,

**55. **Considering the following servlet code, choose statements which are true:

```
package com.nullhaus;

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

@DeclareRoles("Barbra")
public class MyBarbra extends HttpServlet {
}
```

a. This annotation defines a security role “Barbra” for the “MyBarbra” servlet,
b. This annotation defines a security role link for the “Barbra” role within the “MyBarbra” servlet,
c. This is an invalid usage of `@DeclareRoles` annotation,
d. This annotation is defined in Common Annotations, not Servlet 3.0,
e. This annotation is equal to the following DD fragment:

```
<web-app ...>
    <security-role>
         <role-name>Barbra</role-name>
    </security-role>
</web-app>
```

---

56. Considering the following asynchronous servlet code, choose which statements are true after a GET request is made:

```
package com.nullhaus;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;

@WebServlet(urlPatterns = "/foo/*",
            name="NullServlet")
public class NullServlet extends HttpServlet {
    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp) {
        System.out.println("I'm inside!");
```

```
        final AsyncContext ac = req.startAsync();

        ac.start(new Runnable() {
            public void run() {
                System.out.println("*** I'm an async thre
ad!");
                ac.complete();
            }
        });

        System.out.println("I'm leaving! Bye!");
    }
}
```

a. This code compiles and runs fine,
b. The modifier “final” in final AsyncContext ac = req.startAsync(); is not necessary and can be safely removed,
c. The guaranteed order of texts printed in the console/log file is: I’m inside!, *** I’m an async thread!, I’m leaving! Bye!,
d. There is no HttpServletRequest#startAsync() method – there is only a HttpServletRequest#startAsync(ServletRequest, ServletResponse) method,
e. A runtime exception will be thrown when accessing this servlet,
f. This code doesn’t compile.

**57. **Considering the following asynchronous servlet code, choose which statements are true after a GET request is made:

```
package com.nullhaus;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;

@WebServlet(urlPatterns = "/foo/*",
            name="NullServlet",
            asyncSupported = true)
public class NullServlet extends HttpServlet {
    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp) {
        System.out.println("I'm inside!");

        final AsyncContext ac = req.startAsync();

        ac.start(new Runnable() {
            public void run() {
                System.out.println("*** I'm an async thre
```

```
ad!");
                ac.complete();
            }
        });

        System.out.println("I'm leaving! Bye!");
    }
}
```

a. This code compiles and runs fine,
b. There is no “asyncSupported” attribute of @WebServlet,
c. The modifier “final” in final AsyncContext ac = req.startAsync(); is not necessary and can be safely removed,
d. The guaranteed order of texts printed in the console/log file is: I’m inside!, *** I’m an async thread!, I’m leaving! Bye!,
e. There is no HttpServletRequest#startAsync() method – there is only a HttpServletRequest#startAsync(ServletRequest, ServletResponse) method,
f. A runtime exception will be thrown when accessing this servlet,
g. This code doesn’t compile.

**58. **Considering the following asynchronous servlet code, choose which statements are true after a GET request is made:

```
package com.nullhaus;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;

@WebServlet(urlPatterns = "/foo/*",
            name = "NullServlet",
            asyncSupported = true)
public class NullServlet extends HttpServlet {
    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp) {

        final AsyncContext ac = req.startAsync();

        ac.start(new Runnable() {
            public void run() {
                ac.dispatch("/page.html");
                ac.complete();
            }
        });
    }
}
```

a. This code compiles,
b. The content of "/page.html" will be served as a response,
c. A runtime exception will be thrown when accessing this servlet,
d. This code doesn't compile.

---

59. Considering the following asynchronous servlets code, choose which statements are true after a GET request to the NullServlet is made:

```
package com.nullhaus;

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

@WebServlet(urlPatterns = "/foo/*",
            name = "NullServlet",
            asyncSupported = true)
public class NullServlet extends HttpServlet {
    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp)
                           throws IOException {

        resp.getWriter().println("Howdy from NullServlet
1!");

        final AsyncContext ac = req.startAsync();

        ac.start(new Runnable() {
            public void run() {
                ac.dispatch("/baz");
            }
        });
    }
}
```

```
package com.nullhaus;

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

@WebServlet(urlPatterns = "/baz/*",
            name = "NullServle2",
            asyncSupported = false)
public class NullServlet2 extends HttpServlet {
```

```
    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp)
                            throws IOException {

        resp.getWriter().println("Howdy from NullServlet
2!");
    }
}
```

a. This code compiles,
b. The “Howdy from NullServlet2″ will be included in the response,
c. The “Howdy from NullServlet1″ will be included in the response,
d. A runtime exception will be thrown when accessing this servlet,
e. This code doesn’t compile.

**60. **Considering the following Asynchronous Servlets code, choose which statements are true after a GET request to the NullServlet2 is made:

```
package com.nullhaus;

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

@WebServlet(urlPatterns = "/foo/*",
            name = "NullServlet",
            asyncSupported = true)
public class NullServlet extends HttpServlet {
    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp)
                            throws IOException {

        resp.getWriter().println("Howdy from NullServlet
1!");
    }
}
```

```
package com.nullhaus;

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

@WebServlet(urlPatterns = "/baz/*",
```

```
            name = "NullServle2",
            asyncSupported = false)
public class NullServlet2 extends HttpServlet {
    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp)
                        throws IOException, ServletExcept
ion {

        resp.getWriter().println("Howdy from NullServlet
2!");
        req.getRequestDispatcher("/foo").forward(req, re
sp);
    }
}
```

a. This code compiles,
b. The “Howdy from NullServlet2″ will be included in the response,
c. The “Howdy from NullServlet1″ will be included in the response,
d. A runtime exception will be thrown when accessing this servlet,
e. This code doesn’t compile.

**61. **Considering the following Asynchronous Servlets code, choose which statements are true after a GET request to the NullServlet2 is made:

```
package com.nullhaus;

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

@WebServlet(urlPatterns = "/foo/*",
            name = "NullServlet",
            asyncSupported = true)
public class NullServlet extends HttpServlet {
    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp)
                            throws IOException {

        resp.getWriter().println("Howdy from NullServlet
1!");

        final AsyncContext ac = req.startAsync();

        ac.start(new Runnable() {
            public void run() {
                System.out.println("Async!");
```

```
                ac.complete();
            }
        });
    }
}
```

```
package com.nullhaus;

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

@WebServlet(urlPatterns = "/baz/*",
            name="NullServle2",
            asyncSupported=false)
public class NullServlet2 extends HttpServlet {
    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp)
                          throws IOException, ServletExce
ption {

        resp.getWriter().println("Howdy from NullServlet
2!");

        req.getRequestDispatcher("/foo").forward(req, re
sp);
    }
}
```

a. This code compiles,
b. The “Howdy from NullServlet2″ will be included in the response,
c. The “Howdy from NullServlet1″ will be included in the response,
d. The “Async!” will be included in the response,
e. A runtime exception will be thrown when accessing this servlet,
f. This code doesn’t compile.

---

62. Considering the following Asynchronous Servlet code, choose which statements are true after a GET request to the servlet is made:

```
package com.nullhaus;

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

```
@WebServlet(urlPatterns = "/foo/*",
            name="NullServlet",
            asyncSupported=true)
public class NullServlet extends HttpServlet {
  public void doGet(HttpServletRequest req,
                    HttpServletResponse resp)
                            throws IOException {

    req.setAttribute("Hello", "World");

    final AsyncContext ac = req.startAsync();
    ac.setTimeout(-2);

    ac.start(new Runnable() {
      public void run() {
        String att = (String)ac.getRequest().getAttribute
("Hello");

        try {
          PrintWriter pw = ac.getResponse().getWriter();
          pw.println("Async! Value of Hello is: " + att)
;
        } catch (IOException e) {
          e.printStackTrace();
        }

        ac.complete();
      }
    });
  }
}
```

a. This code compiles,
b. The “Async! Value of Hello is: null” will be included in the response,
c. The “Async! Value of Hello is: World” will be included in the response,
d. The asynchronous operation will be timed out after the server default timeout value,
e. The asynchronous operation will be never timed out,
f. A runtime exception will be thrown when accessing this servlet,
g. This code doesn’t compile.

**63. **Considering the following Asynchronous Servlets code, choose which statements are true after a GET request to the NullServlet is made:

```
package com.nullhaus;
```

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

@WebServlet(urlPatterns = "/foo/*",
            name="NullServlet",
            asyncSupported=true)
public class NullServlet extends HttpServlet {
   public void doGet(HttpServletRequest req,
                     HttpServletResponse resp)
                        throws IOException, ServletExcept
ion {

     if (req.getDispatcherType() == DispatcherType.REQUES
T) {
         req.getRequestDispatcher("/baz").forward(req, r
esp);
     } else if (req.getDispatcherType() == DispatcherType
.ASYNC) {
         String hello = (String)req.getAttribute("Hello")
;

         resp.getWriter().println("Phew, that was a ride
!");
         resp.getWriter().println("Value of Hello is: "
+ hello);
     }
   }
}
```

```
package com.nullhaus;

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

@WebServlet(urlPatterns = "/baz/*",
            name="NullServlet2",
            asyncSupported=true)
public class NullServlet2 extends HttpServlet {
    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp)
                            throws IOException {

        req.setAttribute("Hello", "World");
```

```
        final AsyncContext ac = req.startAsync();
        ac.dispatch();
    }
}
```

a. This code compiles,
b. The “Value of Hello is: null” will be included in the response,
c. The “Value of Hello is: World” will be included in the response,
d. The inifite loop dispatch-loop will be created,
e. The request will be served fine, but no text will be included in the response,
f. A runtime exception will be thrown when accessing this servlet,
g. This code doesn’t compile.

---

64. Considering the following Asynchronous Servlets code, choose which statements are true after a GET request to the NullServlet is made:

```
package com.nullhaus;

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

@WebServlet(urlPatterns = "/foo/*",
            name="NullServlet",
            asyncSupported=true)
public class NullServlet extends HttpServlet {
  public void doGet(HttpServletRequest req,
                    HttpServletResponse resp)
                      throws IOException, ServletExceptio
n {

     if (req.getDispatcherType() == DispatcherType.REQUES
T) {
          AsyncContext ac = req.startAsync();
          ac.dispatch("/baz");
     } else if (req.getDispatcherType() == DispatcherType
.ASYNC) {
          resp.getWriter().println("Shotgun!");
     }
  }
}
```

```
package com.nullhaus;
```

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

@WebServlet(urlPatterns = "/baz/*",
            name="NullServlet2",
            asyncSupported=true)
public class NullServlet2 extends HttpServlet {
    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp)
                            throws IOException {
        AsyncContext ac = req.getAsyncContext();
        ac.dispatch("/foo");
    }
}
```

a. This code compiles,
b. The “Shotgun!” will be included in the response,
c. The inifite loop dispatch-loop will be created,
d. The request will be served fine, but no text will be included in the response,
e. A runtime exception will be thrown when accessing this servlet,
f. This code doesn’t compile.

---

65. Considering the following Asynchronous Servlets code, choose which statements are true after a GET request to the NullServlet is made:

```
package com.nullhaus;

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

@WebServlet(urlPatterns = "/foo/*",
            name="NullServlet",
            asyncSupported=true)
public class NullServlet extends HttpServlet {
  public void doGet(HttpServletRequest req,
                    HttpServletResponse resp)
                      throws IOException, ServletExceptio
n {

     if (req.getDispatcherType() == DispatcherType.REQUES
T) {
         AsyncContext ac = req.startAsync();
```

```
            ac.dispatch("/baz");
        } else if (req.getDispatcherType() == DispatcherType
.ASYNC) {
            resp.getWriter().println("Shotgun!");
        }
    }
}
```

```
package com.nullhaus;

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

@WebServlet(urlPatterns = "/baz/*",
            name="NullServlet2",
            asyncSupported=true)
public class NullServlet2 extends HttpServlet {
    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp)
                            throws IOException {
        AsyncContext ac = req.startAsync();
        ac.dispatch("/foo");
    }
}
```

a. This code compiles,
b. The “Shotgun!” will be included in the response,
c. The inifite loop dispatch-loop will be created,
d. The request will be served fine, but no text will be included in the response,
e. A runtime exception will be thrown when accessing this servlet,
f. This code doesn’t compile.

** *|||*For questions 66 and 67 assume that the MyListener class is defined as follows:

```
package com.nullhaus;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;

@WebListener
public class MyListener implements AsyncListener {
```

```
    public void onComplete(AsyncEvent event) {
        System.out.println("#Async listener [onComplete]"
);
    }

    public void onError(AsyncEvent event) {
        System.out.println("#Async listener [onError]");
    }

    public void onStartAsync(AsyncEvent event) {
        System.out.println("#Async listener [onStartAsync
]");
    }

    public void onTimeout(AsyncEvent event)  {
        System.out.println("#Async listener [onTimeout]")
;
    }
}
```

66. **What will be the result of the **first GET request to the following servlet:**

```
package com.nullhaus;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;

@WebServlet(urlPatterns = "/foo/*",
            name="NullServlet",
            asyncSupported=true)
public class NullServlet extends HttpServlet {
  public void doGet(HttpServletRequest req,
                    HttpServletResponse resp)
                        throws ServletException {

     final AsyncContext ac = req.startAsync();

     try {
         Class lClass = Class.forName("com.nullhaus.MyLis
tener");
         AsyncListener al = ac.createListener(lClass);

         ac.addListener(al);
     } catch (ClassNotFoundException e) {
         e.printStackTrace();
     }
```

```
        ac.start(new Thread() {
            public void run() {
                ac.complete();
            }
        });
    }
}
```

a. This code compiles,
b. The “#Async listener [onTimeout]” message will be to the console/log file,
c. The “#Async listener [onStartAsync]” message will be to the console/log file,
d. The “#Async listener [onError]” message will be to the console/log file,
e. The “#Async listener [onComplete]” message will be to the console/log file,
f. A runtime exception will be thrown when accessing this servlet,
g. This code doesn’t compile.

---

67. What will be the result of the **first** GET request to the following servlet:

```
package com.nullhaus;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;

@WebServlet(urlPatterns = "/foo/*",
            name="NullServlet",
            asyncSupported=true)
public class NullServlet extends HttpServlet {
  public void doGet(HttpServletRequest req,
                    HttpServletResponse resp)
                          throws ServletException {

        final AsyncContext ac = req.startAsync();

        try {
            Class lClass = Class.forName("com.nullhaus.MyLis
tener");
            AsyncListener al = ac.createListener(lClass);

            ac.addListener(al);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
```

```
        ac.setTimeout(3000);

        ac.start(new Thread() {
            public void run() {
                try {
                    this.sleep(5000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                ac.complete();
            }
        });
    }
}
```

a. This code compiles,
b. The “#Async listener [onTimeout]” message will be to the console/log file,
c. The “#Async listener [onStartAsync]” message will be to the console/log file,
d. The “#Async listener [onError]” message will be to the console/log file,
e. The “#Async listener [onComplete]” message will be to the console/log file,
f. A runtime exception will be thrown when accessing this servlet,
g. This code doesn’t compile.

Answers

1. c, d

Reference: page 72, section 8.2.3 “Assembling the descriptor from web.xml, web-fragment.xml and annotations”.

Explanation: The order of listeners when using `@WebListener` annotation is **unspecified**. If you need to define the ordering, you should use the Deployment Descriptor which follows a set of rules (defined in section 8.2.3, 2. c.) and works basically on **the order of listeners definition in the DD**.

2. d

Reference: page 22, 3.1 “HTTP Protocol Parameters”

Explanation: When the same variable name exists in the query string, as well as in the POST body, the resulting parameter values list will consists of the query string values, **followed by** the POST data. The`HttpServletRequest#getParameter(-)` returns **only the first parameter value**.

3. e** *|||*

Reference: page 22, 3.1 “HTTP Protocol Parameters”

Explanation: When the same variable name exists in the query string, as well as in the POST body, the resulting parameter values list will consists of the query string values, **followed by** the POST data.

4. d

Reference: page 25, 3.5 “Request Path Elements”

Explanation: The “/security” mapping doesn’t have any restriction policy.

Basically, the PathInfo part of the request path is the part which doesn’t belong to the ContextPath nor the ServletPath and ends **before** the query string.

5. a, b, c, d, e, f , g

Reference: pages 30 – 35**, **4.4 “Configuration Methods”

Explanation: Servlets 3.0 allows you to programatically add servlets, filters, listeners, as well as instantiate all of them. However mind that you can do this only when the `ServletContext` is **not initialized**. After it’s initialization, you’ll get a big, fat `IllegalStateException`.

6. d

Reference: page 30, 4.4 “Configuration methods”

Explanation: Programmatic addition of servlets / filters can be achieved only when the `ServletContext` is **not fully initialized**. Otherwise, the `IllegalStateException` will be thrown. This can be achieved from within the `ServletContextListener` or the `ServletContainerInitializer`.

The nor the loadOnStartup `@WebServlet` annotation attribute **doesn’t have any effect in this case**.

7. a, b, c

Reference: page 32, 4.4.1.5 “ServletRegistration getServletRegistration(String servletName)”

Explanation: You can access already registered servlet, no matter **how** it was registered using the ServletContext#getServletRegistration(-). You can obtain a ServletRegistration object which then can be used to i.e. alter the url mapping paths for the particular servlet.

8. a, b, c, d, f

Reference: page 34, 4.4.3 “Programmatically adding and configuring Listeners”

Explanation: As said before, the programmatically addition of listeners / servlets / filters can be made **only before** the `ServletContext` is fully initialized. This means that in common cases programmatic addition of `ServletContextListener` is **useless**. However, there is a special case when `ServletContextListener` **can be programmatically registered**

– it's when the `ServletContext` is obtained from the `ServletContainerInitializer#onStartup()` method.

The `HttpSessionActivationListener` and `HttpSessionBindingListener` are listeners related to the particular object and as-is doesn't need to be registered **at all**.

9. a.

Reference: page 37, 4.6 "Resources"

Explanation: The ServletContext#getResource(-) method is not intented to be used for dynamic content but for **static**. Therefore, the source code of the `test.jsp` file will be included as-is into the response.

10. c

Reference: page 49, 6.2.4 "Configuration of Filters in Web Application"

Explanation: If the programmer will define two filters with the same filter class, the container is **enforced** to create two instances of the filter.

11. d

Explanation: There **can't be a duplication of filter name** in the Deployment Descriptor. There is no difference if the filter class is different or not.

12. d

Explanation: There **can't be a duplication of filter name** in the Deployment Descriptor. There is no difference if the filter class is different or not.

13. c

Reference: page 63, 8.1.2 "@WebFilter"

Explanation: The default `@WebFilter` name (if not defined through the `filterName` attribute of the annotation) is **fully qualified class name** (`com.nullhaus.NullFilter` in this case). Therefore, there are two **differently named filters** which refers to the same class.

14. b

Explanation: If the `@WebFilter` name is specified, and it's the same as the DD one, the exception **will not be thrown** and the container will initialize only one instance of the filter. Moreover, the values specified in the annotation **will be added to the attributes defined in the DD**. From the logical point of view, there is only one Filter named "MyFilter 1″ with url-patterns "/*".

15. a

Reference: page 50, 6.2.4 "Configuration of Filters in Web Application"

Explanation: There **can** be multiple and elements in one element. The order of such combination of elements is standard, which means that first the

url-pattern hits are executed in the order in which they appear in DD** followed by **the hits are executed in order in which they appear in the DD.

16. a, b, c, d, f

Reference: page 52, 6.2.5 “Filters and the RequestDispatcher”

Explanation: The correct answer is a set of **5 valid `RequestDispatcher` types**. In Servlets 3.0 the new dispatcher type **`ASYNC`** has been introduced.

17. c, e

Reference: page 53, 6.2.5 “Filters and RequestDispatcher”

Explanation: The special filter name “*” has been introduced to express that the request match **either by the path or the servlet name** should be passed to this filter. It is a valid value of the element and servletNames attribute.

18. a

Explanation: The wildcard matches **all** requests made to the web application. The filter mapping configuration lacks the … element, so `RequestDispatcher` operations **will not pass the filter**.

19. e

Reference: page 66, 8.2.2 “Ordering of web.xml and web-fragment.xml”

Explanation: The order of web fragments discovery can be defined **only** if the ordering rules (or) are defined in the Deployment Descriptor.

20. a, d, g

Reference: page 65, 8.2.1 “Modularity of web.xml”

Explanation: The name of the web fragment descriptor is “`web-fragment.xml`“, so b is incorrect.

If this `web-fragment.xml` is a part of a JAR file, it **must** be located under JAR’s META-INF/ directory, so c is incorrect.

If this `web-fragment.xml` is a part of a JAR file and it’s supposed to alter or contribute to the final Deployment Descriptor, it **must** be located under application’s WEB-INF/lib directory. The container can, but **is not required** to discover web fragments which are **not** located under WEB-INF/lib.

21. h

Reference: page 66, 8.2.2 “Ordering of web.xml and web-fragment.xml”

Explanation: The element can only be present in the web.xml. It cannot be defined within the web-fragment.xml file.

22. g

Reference: page 67, 8.2.2 “Ordering of web.xml and web-fragment.xml”

Explanation: If the element is present in the web.xml, it takes under consideration only the fragments which are specified by the … element. If the

fragment name is not present within the … element **and** the element is not present, the fragment will not be added to the result DD.

The element defines that every web fragments which are not defined explicitely within the element, will be automatically added by the container to the result DD.

23. a

Reference: page 66, 8.2.2 “Ordering of web.xml and web-fragment.xml”

Explanation: When the element is present in the web.xml, it defines the effective order of the web-fragments. Any occurences of element within the web-fragment.xml will be **ignored**.

The `metadata-complete` attribute of the element in web.xml is by default set to **false**, which means that** the container will scan** for web-fragments that can be combined into final Deployment Descriptor.

24. e

Reference: page 66, 8.2.2 “Ordering of web.xml and web-fragment.xml”

Explanation: If the `metadata-complete` attribute is present in the `web.xml` and it is set to **true**, it informs the container that the whole Deployment Descriptor data is complete and is presented in the `web.xml`. Therefore no `web-fragment.xml` files should be scanned, and no annotations should be taken under consideration.

The default value of the `metadata-complete` attribute is **false** which means that both the `web-fragment.xml` files should be scanned and the annotations should be combined into the final DD.

25. a

Reference: page 66, 8.2.2 “Ordering of web.xml and web-fragment.xml”

Explanation: If the `metadata-complete` attribute is present in the `web-fragment.xml` and it is set to **true**, it informs the container that the particular **web fragment** is complete and no annotations **related with this web fragment** should be scanned.

Note that the `metadata-complete` within the `web-fragment.xml` has a **different behaviour** than the same attribute within the web.xml file!

The default value for `metadata-complete` is **false**, which means that the annotations for the particular **web fragment** will be scanned.

Check the exemplary code provided for this question in the bundle at the end of this test.

26. a, b, c

Reference: page 61, 8.1 “Annotations and pluggability”

Explanation: The metadata-complete is a and elements attribute, which can be used to determine if the data presented in the `web.xml` is complete (“true”) or incomplete (“false”). If the data presented by the `web.xml` is

incomplete, the container will scan for additional `web-fragment.xml` files and for annotated classes to form the effective, final, Deployment Descriptor. If the value is set to "false" the container will ignore any `web-fragment.xml` files **as well as** any used annotations.

By default, this attribute takes value **false**.

27. c

Reference: page 62, 8.1.1 "@WebServlet"

Explanation: There is a `@WebServlet` annotation, so f is incorrect.

This code compiles fine, as the HttpServlet is an abstract class, but **none of the methods are abstract**; therefore empty class implementation is perfectly valid, so d is incorrect.

The `@WebServlet` annotation defines two ways of specifying url-pattern for the annotated Servlet – directly into the `@WebServlet` annotation (as in the example – implicitely using **value** attrbute) or using an **urlPatterns** attribute. So, the e and a are incorrect.

The url-pattern should start with "/", so this url-pattern is invalid, therefore b is incorrect.

28. c

Reference: page 62, 8.1.1 "@WebServlet"

Explanation: For the main explanation, refer to the previous question.

The only difference here is that the "value" attribute is used explicitely in the `@WebServlet` annotation. This is perfectly valid, but — however — is no different than specifying the value implicitly as in the `@WebServlet("nullHausServlet")` construct. So, the url-pattern is still invalid, and if it would be `@WebServlet(value = "/nullHausServlet")` it would be correct.

29. e, f

Reference: page 62, 8.1.1 "@WebServlet"

Explanation: The only catch in this example is that the `NullHausServlet` has a **package (default) access modifier**, which makes the Servlet **useless for the container**.

The default name of the servlet — if it's "name" attribute is not specified — is a **fully-qualified class name**.

Also note that even that the `urlPatterns` operates on `String` array, you can define **only one `String` element**, which you pass as a value.

30. d

Reference: page 62, 8.1.1 "@WebServlet"

Explanation: The `urlPatterns` and `value` attributes are **required**, but **only one of them** can be present in the `@WebServlet` annotation.

The specification suggest using the default `value` attribute, when this is the only attribute in the annotation, and `url-patterns` if there are more attributes in the annotation.

31. b, c

Reference: page 62, 8.1.1 “@WebServlet”

Explanation: One of the `urlPatterns` or `value` attributes are **required**. The servlet will **not be deployed** and will **not be accessible** even by the name from the DD. Note that both Tomcat 7 and GlassFish Server 3.1 will not throw any exceptions **unless** you try to access the servlet (i.e. using its name).

32. c, f

Reference: page 62, 8.1.1 “@WebServlet” and page 81, 8.2.3 “Assembling the descriptor from web.xml, web-fragment.xml and annotations”

Explanation: When the same servlet class is defined in the DD with the same name, the container **is not required to create a new instance of the servlet class**, however the exact number of servlet instances is unpredictable and it’s **container-dependent**.

If the url-pattern is defined in both: the DD and the annotations, the DD takes precedence.

33. c

Reference: page 62, 8.1.1 “@WebServlet”

Explanation: When the same servlet class is defined in the DD with another name, **the container is forced to create a new instance of the servlet class**.

However, **the exact number of servlet instances is undefined**, as each container may decide what policy it will follow, as some kind of servlet-pools are allowed to be present.

34. a,b

Reference: page 62, 8.1.1 “@WebServlet”

Explanation: When the same servlet class — but with different name — is instantiated using programmatic addition,** the container will create two instances of the servlet**. The annotated one will have configuration as defined using the annotations, and the programmatic one will have its own configuration.

35. b

Reference: page 62, 8.1.1 “@WebServlet”

Explanation: When the servlet is **programmatically** created using the **same name** as previously defined (using annotations or DD), the behaviour is **not specified**.

Tomcat 7 and Glassfish 3.1 (both uses Catalina servlet container

implementation) will throw NullPointerException — **ctx.addServlet(-) returns null** and none of the url patterns will be mapped to the servlet.

Rasin 4.0.16 will create **one instance** of the servlet and **add the two url-patterns**, so the single servlet instance will respond to both: /baz/* and /foo/* url patterns.

36. b

Reference: page 62, 8.1.1 "@WebServlet"

Explanation: Classes annotated with `@WebServlet` annotation must extend the **`javax.servlet.http.HttpServlet`** class.

37. c

Reference: page 63, 8.1.3 "@WebInitParam"

Explanation: The `@WebInitParam` defines the **servlet config init param** which is correctly accessed in this code (`HttpServlet#getInitParameter(-)` is in fact an invocation of `getServletConfig().getInitParameter(-)`).

38. a

Reference: page 63, 8.1.3 "@WebInitParam"

Explanation: The `@WebInitParam` must be used as a `@WebServlet`'s `initParams` attribute value. If it's used as a direct annotation of the servlet class, it will not have any impact.

39. f

Reference: page 63, 8.1.3 "@WebInitParam"

Explanation: This code doesn't compile, because there is a annotation duplication error. A concrete Java annotation cannot be used more than once in a single class.

40. b, c, f, g

Reference: page 69, 8.2.2 "Ordering of web.xml and `web-fragment.xml`"

Explanation: The element is used to define a relative ordering of web fragments and it's possible location is within the `web-fragment.xml` file. The element is used to define an absolute ordering of web fragments and the main DD (`web.xml`) is the place where this element must be placed.

The is **not** a valid subelement of element. The only valid elements are and which can have or subelements.

41. a, d

Reference: page 73, 8.2.3 "Assembling the descriptor from web.xml, web-fragment.xml and annotations"

Explanation: The element set to "false" forces the container to **disable** the specified servlet, which means that the defined will **not** forward the request

to the servlet.

Each web fragment is processed **separately**. The container will firstly process the particular web fragment (web-fragment.xml), than it'll process this web fragment annotations **and only after this**, the container will process the **next** web fragment. Therefore, the f option is incorrect.

42. g

Reference: page 74, 8.2.3 "Assembling the descriptor from web.xml, web-fragment.xml and annotations"

Explanation: The situation in which two **different web fragments** defines the **same name** of a but with different values is as a **configuration conflict**. It will result in an application deploy **failure**.

43. a, c

Reference: page 81, 8.2.3 "Assembling the descriptor from web.xml, web-fragment.xml and annotations"

Explanation: Basically, the **DD has precedence over the annotations**, so the init-params with the same name, defined in DD has precedence over the ones defined in annotations and will replace them.

The init params with different names are **additive** in the final DD.

If the url-pattern is defined in the DD it **replaces** the urlPatterns annotation for the particular servlet.

44. b

Reference: page 84, 8.2.4 "Shared libraries / runtimes pluggability"

Explanation: The service file is **incorrectly named**. It should be **`javax.servlet.ServletContainerInitializer`**. If this filename would be corrected, option d would be also correct. The container **doesn't have to** run initializer which is in a JAR file in the web application classpath but **not** in the WEB-INF/lib.

45. a, c, d, e

Reference: pages 90 – 95, 9.2 "Using a Request Dispatcher"

Explanation: The Servlets 3.0 brings **new set of request attributes** which are set after dispatching the asynchronous processing using `AsyncContext#dispatch(-)`. These attributes replaces the "forward" word in the above example with "**async**", just as option d suggests.

All of those attributes always hold the information about the **original** request.

46. h

Reference: page 99, 10.5 "Directory Structure"

Explanation: The WEB-INF directory in a web application is **basically a private directory**. An exception is made, however, to **static resources** which are located in the** /META-INF/resources** directory of a JAR file located in WEB-INF/lib directory of the web application. Such resources are

served **directly to the client**.

47. b, d

Reference: page 107, 10.13 “Inclusion of a web.xml Deployment Descriptor”

Explanation: An application which is serving only a static content, **doesn’t have to contain neither the WEB-INF directory, nor the web.xml file**.

48. a, d

Reference: pages 111 – 112, 11.3 “Listener Class Configuration”

Explanation: Every listener class must provide a **public, no argument constructor**. There is no requirement that this constructor must be compiler generated.

The **`HttpSessionActivationListener`** and **`HttpSessionBindingListener`** are **not** registered in the DD.

The **AsyncListener** cannot be registered in the DD – it **must be registered programmatically**.

49. a, b

Reference: page 122, 13.4 “Programmatic Access Control Annotations”

Explanation: All three ways (DD, programmatically, annotations) of managing security constraints are **valid**, but the annotation name is `@ServletSecurity` – **not `@WebServletSecurity`**. Hence, the a and b are correct and c is incorrect.

The `@ServletSecurity` annotation has two attributes – `value` and `httpMethodConstraint`.

The `rolesAllowed` attribute is a part of `@HttpConstraint` or `@HttpMethodConstraint`, not the `@ServletSecurity`.

50. a

Explanation: This might seem like a valid Http constraint which denies access for all users, but in fact it is **inappropriate usage of `@HttpConstraint`** annotation. The compiler wont’ complain, a runtime exception will not be thrown, but the servlet will act like **there are no security constraints defined**. This is because the `@HttpConstraint` and `@HttpMethodConstraint` can be used only as **the `@ServletConstraint` annotation attributes**.

51. e

Explanation: There is **no** `methodName` attribute for `@HttpMethodConstraint` annotation. The value should be used instead.

There is **no** `EmptyRoleSemantic.ALLOW` enum value. The `PERMIT` value should be used instead.

52. a

Explanation: This is the correct usage of the annotations. The servlet is

allowed only for HTTP GET requests and **blocked for all other HTTP methods** requests despite the user's role.

53. a, b

Reference: pages 122 – 125, 13.4 "Programmatic Access Control Annotations"

Explanation: The default value for `@HttpConstraint#transportGuarantee` is **`TransportGuarantee.NONE`**.

The default value for `@HttpMethodConstraint#emptyRoleSemantic` is **`EmptyRoleSemantic.PERMIT`**.

54.a, c

Reference: pages 123 – 126, 13.4 "Programmatic Access Control Annotations"

Explanation: The `value` attribute defines the constraint which affects the HTTP methods which are **not** listed using the `httpMethodConstraints` attribute.

`EmptyRoleSemantic` defines the default action which should be undertaken if the `rolesAllowed` returns an** empty list** (no roles are defined). As a side not, you **should not** define `emptyRoleSemantic` when `rolesAllowed` list is not empty.

The "*" as a value of `rolesAllowed` attribute **doesn't have any special meaning**. In the DD, when is used it means "all roles", but in the `@ServletSecurity` annotation, it doesn't.

55. a, d, e

Reference: page 181, 15.5.1 "@DeclareRoles"

Explanation: This is a valid usage of the `@DeclareRoles` annotation which defines a security role with name passed as the annotation `value` attribute.

It is **impossible** to create a security role link using `@DeclareRoles` annotation. To do so, one should **use the DD**.

This annotation is defined in Common Annotations which means that if you're using i.e. Tomcat 7, you need to add the annotations-api.jar to your classpath.

56. e

Reference: pages 10 – 20, 2.3.3.3 "Asynchronous processing"

Explanation: To use the Asynchronous Servlets features, the servlet **must** support this type of processing either by defining **`asyncSupported = true`** attribute of the `@WebServlet`, or by **`true`** element in the DD.

57. a

Reference: pages 10 – 20, 2.3.3.3 “Asynchronous processing”

Explanation: This code runs fine – there is an `asyncSupported` attribute and in fact it’s the **only asynchronous related attribute in the `@WebServlet` annotation**. Things like timeouts must be dealt programmatically by using the `AsyncContext`.

The modifier **`final` is necessary**, because the “ac” variable must be accessible from the anonymous inner class.

The order of messages is **not guaranteed**. There are two threads, and the Container/JVM can choose how and when execute them; the only guaranteed thing is that the “I’m inside” message will be printed first.

58. a, b, c

Reference: pages 10 – 20, 2.3.3.3 “Asynchronous processing”

Explanation: The container will immediately dispatch the request to the given resource (“/page.html”) and implicitely treat it like the **`AsyncContext#complete()`** method invocation.

The following `ac.complete()` invoacation will result in a** runtime exception** — `IllegalStateException` — thrown, as the request has already been dispatched.

**59. **a, b, c

Reference: pages 10 – 20, 2.3.3.3 “Asynchronous processing”

Explanation: It is fine to dispatch from an **asynchronous servlet to the synchronous**. By using req.startAsync() the original request and response objects are passed to the asynchronous thread, so the “Howdy from NullServlet1″ **will not be lost **and will be included in the response.

60. a, c

Reference: pages 11, 2.3.3.3 “Asynchronous processing”

Explanation: It is **illegal** to dispatch the request from the synchronous servlet to the asynchronouse, but the exception throwing is **delayed to the moment of actually using the asynchronous nature of the servlet** – like `AsyncContext.startAsync(-)`. If the asynchronous servlet doesn’t use any of the asynchronousity features, it is legal to do such dispatch.

The “Howdy from NullServlet2″ will **not** be included in the response, as the uncommited output in **response buffer is cleared** during the forwarding.

61. e

Reference: pages 11, 2.3.3.3 “Asynchronous processing”

Explanation: Refer to the previous question’s explanaition; this code explicitely uses the Asynchronous features, so the `IllegalStateException` will be thrown.

62. a, c, e

Reference: page 14, 2.3.3.3 “Asynchronous processing”

Explanation: If the `AsyncContext#setTimeout(-)` argument is **<= 0**, it means that the asynchronous operation will** never be timed out**.

The `ServletRequest#startAsync()` method uses the unwrapped `ServletRequest` and `ServletResponse`, so the attributes set before asynchronous thread start are **accessible** from within the thread.

**63. **a, c

Reference: page 15, 2.3.3.3 “Asynchronous processing”

Explanation: This code is an example of how to use the `ServletRequest#getDispatcherType()` to recognize if the request was made initialy (`REQUEST`) or through asynchronous dispatch (`ASYNC`).

Another thing is that if the **unwrapped** `ServletRequest` and `ServletResponse` are used in the `ServletRequest#startAsync(-)`, the following `AsyncContext#dispatch()` will dispatch the to the URL of the **original** request (/foo/*).

The last thing shown is that the `ServletRequest` and the `ServletResponse` are the same objects in the dispatch chain, so the attributes are saved properly.

**64. **a, e

Reference: page 16, 2.3.3.3 “Asynchronous processing”

Explanation: It is **illegal** to use more than one asynchronous dispatch from **one AsyncContext**. A runtime exception will be thrown.

**65. **a, b

Reference: page 16, 2.3.3.3 “Asynchronous processing”

Explanation: It is **illegal** to use more than one asynchronous dispatch from** one AsyncContext**, but in this case a AsyncContext **has been reinitialized** (`req.startAsync()` instead of `req.getAsyncContext()`).

**66. **a, e

Reference: pages 17 – 18, 2.3.3.3 “Asynchronous processing”

Explanation: The listener will be notified after the asynchronous processing will be completed. The timeout, error and onStartAsync (it’s the **first** invocation of startAsync()) will not occur in this case.

**67. **a, b, d, e, f

Reference: pages 17 – 18, 2.3.3.3 “Asynchronous processing”

Explanation: After the given timeout, the listener `onTimeout(-)` method will be invoked, as well as the `onError(-)`.

After that, the `onComplete(-)` method will be invoked (response for the

`ac.complete()` invocation). This invocation will be followed by the `IllegalStateException`, as it's illegal to complete the asynchronous processing** if the request has already been dispatched (timeout)**.

Source code

[OCE-JSP-and-Servlets-Developer-Mock-Exam](https://github.com/PiotrNowicki/OCE-JSP-and-Servlets-Developer-Mock-Exam) at GitHub.

SCWCD JEE6 Mock Exam - This package consists of source code of ~30 questions from my SCWCD JEE6 mock exam.

Posted in Java.
Tagged certification, exam, glassfish, java, jee, jee 6, mock, oracle, resin, scwcd, sevlets, tomcat.

47 thoughts on "Java EE 6 SCWCD Mock Exam"

Roy Pozarelli said on May 20, 2011 at 02:09: **Edit**

In question #29, you say that f is valid.
@WebServlet(urlPatterns="/nullHausServlet")
class NullServlet extends HttpServlet {
}
since the default name of the WebServlet is the fully qualified name. But in the question f states: The name of this servlet is "com.nullhaus.NullHausServlet" either this should be "com.nullhaus.NullServlet" or the class name needs to change to NullHausServlet.

Aside, this is EXCELLENT review for me. Thanks for the effort it is appreciated.

Reply ↓

NullHaus said on May 20, 2011 at 09:49: **Edit**

Hi Roy,

You're right – the name of the class should be

NullHausServlet instead of NullServlet. I just fixed it.
Thanks a lot for pointing this out!
I'm glad you found this exam useful :-)

Cheers!

Reply ↓

Roy Pozarelli said on May 23, 2011 at 16:26: **Edit**

In question 41, statement F. In reading over Sec. 8.2.3 – 5.d "Web fragments are merged into the … The merging takes place before annotation processing on the corresponding fragment". So in reasoning about just that it seems that F should be valid. Or is there another section of the spec. that also applies that I'm missing?

Reply ↓

NullHaus said on May 23, 2011 at 17:07: **Edit**

Hi Roy,

Well I guess it's just an unfortunate sentence construction… The meaning of this question is somewhat:

"Are the web fragments scanned all at the same time or one after another?"

As you quoted:
"The merging takes place before annotation processing on the *corresponding fragment*".

Which means that there is no 'scan all *web fragments* and put them in the web.xml and then process all annotations for those fragments'. There is a sequential process: 'scan one web fragment, put it in the DD, read it's annotation… take next web fragment, put it in the DD, read it's annotation, …' and so on.

Does it answer your question? Maybe the last answer should be rephrased. What do you think?

Reply ↓

Roy Pozarelli said on May 23, 2011 at 19:57: **Edit**

That answers it. Maybe it was my ignorance that caused the confusion about the answer.

Check out Answer 49, there seems to be an inconsistency A,C or A,B

Aside, w.r.t. ServletRegistration#setServletSecurity(-) method,

does it make a difference in that it is really the ServletRegistration.Dynamic#setServletSecurity(-) is the actual method?

Note: Thanks for the responses! It is always helpful when you are in the learning mode.

Reply ↓

NullHaus said on May 23, 2011 at 20:09: **Edit**

Hi Roy,

Another point for being on guard! You're right, the answer should be A and B as the answer description tells.

And second time you're right – it is my mistake – it should have been ServletRegistration.Dynamic#setServletSecurity(-) which I fixed right now.

Glad you are so observant! :-)

Cheers!

Reply ↓

Roy Pozarelli said on May 23, 2011 at 21:09: **Edit**

Question 56, 57, 58, 59. I had "code does not compile" since there is NO throws declaration in the doGet(…) {.

protected void doGet(HttpServletRequest req,
HttpServletResponse resp)
throws ServletException,
java.io.IOException

Clearly that wasn't the intent for all these questions. I didn't check the other code samples to see if this issue is repeated else where.

Reply ↓

Piotr said on May 24, 2011 at 10:02: **Edit**

Hi Roy,

Did you try downloading and compiling the source code for questions 56 – 59?

Cheers!

Reply ↓

Roy Pozarelli said on May 24, 2011 at 17:39: **Edit**

No, sorry, that is my next step to see what my server

implementation does with some of the examples.

Reply ↓

Piotr said on May 30, 2011 at 13:41: **Edit**

Roy, did you have the time to check the code for questions 56 – 59? Is it really not compiling?

Cheers!

Reply ↓

Roy Pozarelli said on June 8, 2011 at 19:50: **Edit**

In question 13:
com.nullhaus.MyFilter
with
…
public class NullFilter implements Filter {

Was the intent to use the same class in the question or 2 different classes? Also check question 14 too.

Reply ↓

Piotr said on June 9, 2011 at 00:16: **Edit**

You're right – both, the DD and Filter class should have the same class name (NullFilter instead of MyFilter). Thanks for pointing this out!

Reply ↓

Roy Pozarelli said on June 8, 2011 at 20:11: **Edit**

Piotr, just a question about #18.
The *, I haven't coded up an example for this yet, but in reading the spec. sec. 12.2 "Specification of Mappings", should this have a /* instead? I'm not clear yet on what is actually needed.

Reply ↓

Piotr said on June 9, 2011 at 00:22: **Edit**

Once again right – I've copied the example from the above question where it was "*" but after changing it to I forgot to change the wildcard to "/*".

Thanks for being on guard!

Cheers!

Reply ↓

Roy Pozarelli said on June 8, 2011 at 20:16: **Edit**

Actually, in NetBeans 7.0, Glassfish server, it does need the /* for question 18 to be deployed. Just using * failed to deploy for me.

Reply ↓

Roy Pozarelli said on June 9, 2011 at 01:52: **Edit**

In question 61, isn't a) The code compiles, also valid given e) throws an exception?

Reply ↓

Roy Pozarelli said on June 9, 2011 at 02:25: **Edit**

Never mind about the last (q. 61) I forgot the … and runs fine part.

Reply ↓

Goldest said on July 1, 2011 at 17:38: **Edit**

I am a SCWCD seeker and have recently started the preparation. This site has really made my confidence high. Now I know that if I attempt the latest 6th version for this exam, I have this place to refer to regarding the new additions to the exam.

Thanks a lot for taking your time and making these things available for us.

Very Well Done…!!!
(I will soon be back after finishing with my preparation to take a final look here.)

Goldest

Reply ↓

Piotr said on July 2, 2011 at 03:32: **Edit**

Thanks a lot mate!

Glad I could help and good luck with your exam.

Cheers!

Reply ↓

hi said on August 2, 2011 at 03:54: **Edit**

Any Ebook or study material available to prepair for this exam ?
Can i just study Servlets 3.0 FR specification ?

Reply ↓

Piotr said on August 2, 2011 at 20:05: **Edit**

Hi,

Well, I'm not sure if there any other materials… I would say that the Servlets specification is the best source of knowledge.

Of course you can always check for some new books at the Java Ranch: http://www.coderanch.com/how-to/java/ScwcdLinks

BTW: There is an Enthuware new mock exam simulator available for all you OCE-JSP/Servlet Java EE candidates: http://enthuware.com/index.php/mock-exams/oracle-sun-java-certifications/jsp-servlet/oce-jsp-servlet-mock-questions

Cheers!

Reply ↓

Roy Pozarelli said on August 9, 2011 at 18:22: **Edit**

Question 67:
@WebListener
public class MyListener implements AsyncListener …

In the Java Docs for this annotation it does NOT list AsyncListener as a valid listener. In coding this up using NetBeans IDE 7.0 and using GlassFish v.3.1 it reports back to me a severe error during deployment:
SEVERE: The Class MyListeners.Listener67 having annotation javax.servlet.annotation.WebListener need to implement one of the following interfaces: javax.servlet.ServletContextListener, javax.servlet.ServletContextAttributeListener, javax.servlet.ServletRequestListener, javax.servletServletRequestAttributeListener, javax.servlet.http.HttpSessionListener, javax.servlet.http.HttpSessionAttributeListener.
as well as:
SEVERE: Annotations processing failed for file …

Commenting out the annotation and using an element in the DD (web.xml) also results in the same deployment error.

Commenting out the annotation and having no entry in the DD results in a good deployment.

So is there an error here? Is my ignorance showing? Is a correction needed?

Reply ↓

Mindmap said on October 4, 2011 at 23:59: **Edit**

Congratulations Piotr, This test is superb. It's really helpful when you want to prepare for the new version of the exam.

Reply ↓

Pingback: OCPWCD 6, pierwsze wrażenia z przygotowań | CaveRed **Edit**

Vimal Kumar Venugopal said on February 18, 2012 at 07:38: **Edit**

Howdy Piotrno,

.. Thanks a lot for all the effort you put in here!!! This is totally useful..

Reply ↓

Piotr said on February 18, 2012 at 21:30: **Edit**

Thanks for kind words. Glad you find it useful! :-)

Cheers!

Reply ↓

wpchia said on March 1, 2012 at 10:11: **Edit**

Question 54:

I think g is the correct answer, since rolesAllowed = "*" will returns a non-emtpty array, hence the emptyRolesSemantic should not be specified, this will cause an invalid usage of @ServeltSecurity.

Reply ↓

Piotr said on March 1, 2012 at 10:27: **Edit**

You're right – thanks for pointing this out!

I've removed the "g" answer from the question, as I decided that this was not the real purpose it was asked and you can still gain some knowledge from it. I've added additional explanation in the answers section.

Once again – thanks for your awareness!

Reply ↓

Swapnil said on May 12, 2012 at 08:21: **Edit**

Howdy Piotr?

I am starting preparations for this exam. Currently I have started with HFSJ. Could you please let me know how to learn the specs? I there any simple way / trick to learn it? I mean if I read HFSJ then

it covers Servlet 2.4. How can I find exact differences between Servlet 2.4 and 3.0 from learning perspective? One more question: There are three more specs other than that of Servlet 3.0 namely: EL, JSTL and JSP. I think those are also in the syllabus but never seen anyone mentioning those who have passed this exam.

Could you please guide me?

Cheers,
Swapnil

Reply ↓

Piotr said on May 12, 2012 at 11:08: **Edit**

Hi Swapnil,

Well, there is no real trick in learning the specs – I just found it interesting to read it and test some of the discussed features. I do believe that there are few websites talking what changes were made between 2.4, 2.5 and 3.0 like here: http://www.javaworld.com/javaworld/jw-01-2006/jw-0102-servlet.html and here: http://stackoverflow.com/questions/1638865/what-are-the-differences-between-servlet-2-5-and-3

After reading HFSJ, I think you can just skim the specs and you'll fast see what are the differences. I've skimmed the JSTL and JSP specifications but didn't really use it for the exam. The EL is more important so I took some time with it (although e.g. deferred expressions weren't at the exam…)

If you have more time, I'd just advise to read the Servlets 3.0 specs after the HFSJ (it'll give you the big picture) and then go with the Enthuware's mock exam. If don't have much time, I'd just take the Enthuware's mock exam after the HFSJ.

Good luck mate!

Reply ↓

Swapnil said on May 12, 2012 at 15:55: **Edit**

Thanks a lot Piotr!!

I believe reading JSP, EL specs is not that much important from exam point of view.

I would thoroughly study Servlet specs though and would definitely try your mock exam.

Cheers,
SWapnil

Reply ↓

Michael said on July 25, 2012 at 20:48: **Edit**

Hi Piotr!
Many thanks for this exam preparation question/answers. I am currently in preparation and they are of great use for me. Please let me know what is HFSJ? I was taking the course from Oracle and went through the servlet spec 3.0 in detail (apart from learning JSP/EL). And apart from some bits and pieces (e.g. asynchronous processing) I am confident I could take the exam in some time.
Regards, Michael

Reply ↓

Piotr said on July 25, 2012 at 21:10: **Edit**

Hello Michael!

I'm glad you find the exam useful :-)
HFSJ is an abbreviation for "Head First: Servlets and JSP" – a book by Kathy Sierra and Bert Bates (http://www.amazon.com/Head-First-Servlets-JSP-Certified/dp/0596005407).
If you took the training and read the specs (or at least skimmed over it), just do some mock exams and I'm more than sure that you'll pass it easily :-) I would definitely recommend http://enthuware.com/ exam – it's definitely worth its price!

Good luck and don't forget to tell us about the result! :-)

Reply ↓

Michael said on July 26, 2012 at 09:39: **Edit**

Hi Piotr,
thanks for the info! Much appreciated. For two of the tasks I have questions:
Nr. 58 I rerun the given example and you are right, however, according to servlet spec 9.7 "All the variations of the dispatch methods returns immediately and do not commit the response." So commit could theoretically be called AFTER the dispatch. I couldn't find in the spec a statement saying that AsyncContext#complete will lead to a runtime exception if the request was dispatched before!?
Nr. 59 when in NullServlet response.flushBuffer is called BEFORE the AsyncContext is created and started the response is already committed to the client. According to the spec "It is illegal to call startAsync [...] if the response has been committed [...]" However, the example then

still would run and both strings be printed.
Why?
In which case would “Howdy from NullServlet1″ be lost – just be using a Wrapper for the original request and response?
Regards,
Michael

Reply ↓

Piotr said on July 26, 2012 at 12:25: **Edit**

Michael,

As for your first question, take a look at the Javadoc for AsyncContext#dispatch():
http://docs.oracle.com/javaee/6/api/javax/servlet/AsyncContext.html#dispatch%28%29

Throws: IllegalStateException – if one of the dispatch methods has been called and the startAsync method has not been called during the resulting dispatch, **or if complete() was called**

You’d need to give me some time for the second question, as I haven’t recently used asynchronous features of Servlets 3.0 :-)

Reply ↓

Piotr said on July 26, 2012 at 13:34: **Edit**

Well, Michael, about question 59. I’ve just tested it on Tomcat 7.0.27, and it seems a bit strange indeed.

The flushBuffer() does not make any changes to the response – both texts are included. However, section 5.5. Closure of Response Object says when the response should be trated as ‘closed’ (note that sometimes they use ‘commited’ and sometimes ‘closed’ – not sure if these are the same). Flushing the buffer is not one of the cases when the response object is “closed”.

So if instead of flushing the buffer, you'll sendError(-), you won't get anything printed out BUT the request will be dispatched correctly (wicked!)

It's a bit strange for me, as the response has already been commited (isCommited() returns true) but you still can create an AsyncContext despite the following information in the javadoc for ServletRequest#startAsync(): IllegalStateException – (…) or if the response has already been closed

If you'll dig into it, it would be great if you could post some info here :-)

Cheers!

Reply ↓

Michael said on August 1, 2012 at 18:11: **Edit**

Hi Piotr,
many thanks for your answers! Today I successfully passed the exam – with the help of your questions as well! Thanks again,
Michael

Reply ↓

Piotr said on August 1, 2012 at 18:19: **Edit**

Congratulations Michael! Great job :-)

Reply ↓

Rashko Rejmer said on August 3, 2012 at 21:31: **Edit**

Hi Piotr,
I have a tiny concern about #8. Isn't it possible to add ServletContextListener programatically inside ServletContainerInitializer#onStartup(). I know that it is actually pointes but I am asking just out of curiosity.
btw, this test is really great source of information.
Thanks

Rashko

Reply ↓

Piotr said on August 4, 2012 at 12:34: **Edit**

Hello Rashko.

You're totally right. The listeners can be added before the ServletContext is fully initialized. If you specify a ServletContainerInitializer, you can still register the regular listeners and the ServletContextListener as well.

I've just edited the answer for this question.

Javadoc: http://docs.oracle.com/javaee/6/api/javax/servlet/ServletContext.html#addListener%28T%29

Thanks for pointing this out!

Reply ↓

Vimal Kumar said on August 12, 2012 at 16:33: **Edit**

Hi Piotr,

Nice work. Appreciate your tedious efforts. This helps us all a lot.

The answer for Q31 seems confusing. I read from the specs (Servlet 3.0 – 8.2.3 – pg 74) " *v. elements with the same are additive*". So I was thinking whether both /foo/* and /bar/* url patterns will be invoking the servlet.

Kindly help me out.
Thanks a ton. Continue your good work

Reply ↓

Vimal Kumar said on August 12, 2012 at 16:36: **Edit**

Sorry for the typo. I meant Q32.

Reply ↓

Piotr said on August 12, 2012 at 22:01: **Edit**

Hello Vimal!

Take a look at mentioned page 81, 8.3 Annotations and pluggability, point enumerated ‘n. iv’:

“url-patterns, when specified in a descriptor for a given servlet name overrides the url patterns specified via the annotation.”

I do believe that the part you cited, refers to the combination of web-fragments. Although I think it should be more precisely split as it can be hard to find the relevant information in all those multilevel bullet points.

Hope it helps

Cheers!

PS. I also encourage you to download the sample code from my github account I mentioned in the post and try deploying the app yourself. It’s quite helpful!

Reply ↓

Valentina said on September 3, 2012 at 20:05: **Edit**

Hi Piotr,

I was wondering about question 35.Part of the explanation is:

“Explanation: When the servlet is programmatically created using the same name as previously
defined (using annotations or DD), the behaviour is not specified.”

If we had the same scenario but instead of annotations we used DD and programatic addition then
wouldn’t this always yield null(as opposed to unspecified behaviour)?
ServletRegistration.Dynamic d = ctx.addServlet(“NullHaus1″, s);

In the API:”Returns:a ServletRegistration object that may be used to further configure the
registered servlet, or null if this ServletContext already contains a complete ServletRegistration
for the given servletName”

Maybe I am missing something here…Thanks for your time and great test:)

Reply ↓

Piotr said on September 13, 2012 at 10:39: **Edit**

Hello Valentina.

Did you try using DD and programmatic addition? I wonder how Resin would interpret this situation. It might

be interesting to see if `metadata-complete="true"` would change this behavior in any way.

Cheers!

Reply ↓

Avik Ganguly said on September 8, 2012 at 06:13: **Edit**

Shouldnt the option b in 36 be javax.servlet.http.HttpServlet? When do you think Oracle will update their online documentation (field,method and classes summary) of the Servlets 3.0 related parts?

Reply ↓

Piotr said on September 13, 2012 at 10:16: **Edit**

Hello Avik.

You're right – that is an obvious mistake of mine. It should have the *.http.* package in it. I've already tried to fix it but the markdown and wordpress started to do weird things, so the formatting of the whole exam might be a bit messed up.

Nevertheless, about the update of the online docs – what exactly do you mean? The Java EE 6 Javadoc API is already published here: http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServlet.html

Is it what you're looking for?

Cheers!

Reply ↓

Leave a reply

Logged in as Piotr. Log out?

Comment

You may use these `<a href="" title=""> <abbr title="">`

HTML tags and attributes:

```
<acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <strike> <strong>
```

Post Comment

← Previous Post

Next Post →