Q02 - Same GET request parameter behavior

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>
```

com.nullhaus.NullServlet:

```
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

Show answer

Q03 - Same POST request parameter behavior

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>
```

com.nullhaus.NullServlet:

```
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

Show answer

Q04 - Servlet paths

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 authorization purposes.

Hide answer

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.

Q05 - Programmatical features of Servlets 3.0

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

Hide answer

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`.

Q06 - Programmatical features of Servlets 3.0 [2]

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 `<load-on-startup>` 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

Hide answer

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 `<load-on-startup>` nor the `loadOnStartup` `@WebServlet` annotation attribute **doesn't have any effect in this case**.

Q07 - Accessing Servlets

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

Hide answer

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 e.g. to alter the URL mapping paths for the particular Servlet.

Q27 - Annotation Servlets Definition

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`

Hide answer

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.

Q28 - Annotation Servlets Definition [2]

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`

Hide answer

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 explicitly 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.

Q29 - Annotation Servlets Definition [3]

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

Hide answer

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.

Q30 - Annotation Servlets Definition [4]

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="/numeroDuo")
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

Hide answer

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.

Q31 - Annotation Servlets Definition [5]

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

Hide answer

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).

Q32 - Annotation and Declarative Servlets Definition

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

Hide answer

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.

Q33 - Annotation and Declarative Servlets Definition [2]

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

Hide answer

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.

Q34 - Programmatic Servlets Addition

Consider the following Servlet code and the `ServletContainerInitializer` code.

com.nullhaus.MyJar1Servlet

```
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 {
}
```

com.nullhaus.MyInit

```
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("NullHaus2", s);

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

Assume that the `MyInit` class is properly registered in the container as a `ServletContainerInitializer`. Choose the statements that are true:

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

Hide answer

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.

Q35 - Programmatic Servlets Addition [2]

Consider the following Servlet code and the `ServletContainerInitializer` code.

com.nullhaus.MyJar1Servlet

```
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 {
}
```

com.nullhaus.MyInit

```
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) {
            // ...
        }
    }
}
```

Assume that the `MyInit` class is properly registered in the container as a `ServletContainerInitializer`. Choose the statements which are true:

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

Hide answer

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.

Q36 - Annotation Servlets Definition [6]

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

Hide answer

b

Reference: page 62, 8.1.1 "@WebServlet"

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

Q37 - Servlet Init Parameters

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 IOException {
        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

Hide answer

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(-)`).

Q38 - Servlet Init Parameters [2]

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 IOException {
        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

Hide answer

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.

Q39 - Servlet Init Parameters [3]

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 IOException {
        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

Hide answer

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.