Oracle Certified Expert, Java Platform, Enterprise Edition 6 Web Services Developer Quiz

Mikalai Zaikin

IBA JV

Belarus
Minsk
<NZaikin[at]iba.by>

March 2011

Revision History		
Revision $Revision: 75 $	$Date: 2011-05-08 00:26:06 +0300 (нд, 08 Май 2011) $	$Author: mzaikin $
$Id: ocewsd6-quiz.xml 75 2011-05-07 21:26:06Z mzaikin $		

Abstract

The purpose of this document is to help in preparation for Java Platform, Enterprise Edition 6 Web Services Developer Certified Expert Exam (CX-310-232).

This document should not be used as the only study material for Oracle Certified Expert, Java Platform, Enterprise Edition 6 Web Services Developer Test. It might cover not all objective topics, and it might be not enough. I tried to make this document as much accurate as possible, but if you find any error, please let me know.

1.2.1. Use `@WebService` to indicate a service
1.2.2. Use `@WebMethod`, `@WebMethod(exclude)` to indicate service methods
1.2.3. Use `@SOAPBinding` to select doc/lit, doc/bare, rpc/lit style of web service
1.2.4. Use `@Oneway` where the service doesn't have any response
1.2.5. Use `@WebParam`, and `@WebResult` to customize parameter and operation names
1.2.6. Use checked exceptions to indicate service specific faults.
1.2.7. Use `wsgen` tool to generate artifacts in Java EE5 (optional in Java EE6, as artifacts are generated at run time).
1.2.8. Configure deployment descriptors (`web.xml`, `webservices.xml`) for URL patterns, HTTP security, container authorization, caller authentication, and message protection. JAX-WS runtime may also be configured to perform message layer authentication and protection.
1.2.9. Compile and package the Web Service into a WAR file
1.2.10. Deploy the web service into a Java EE servlet container

2. Create a RESTful web service in a servlet container
2.1. Create a web service using JAX-RS, refer to Jersey implementation for examples
2.1.1. Annotate a class with a `@Path` annotation to respond to URI templates.
2.1.2. Annotate the class's methods to respond to HTTP requests using the corresponding JAX-RS annotations (`@GET`, `@POST`, etc.).
2.1.3. Use the JAX-RS `@Consumes` and `@Produces` annotations to specify the input and output formats for the RESTful Web Service.
2.1.4. Use `@PathParam`, `@QueryParam`, `@MatrixParam` and `@HeaderParam` to extract request data.
2.1.5. Use the `UriInfo` and `UriBuilder` to create URIs that refer to resources in the service.
2.1.6. Use `ResponseBuilder` to create response with customized status and additional metadata.
2.1.7. Implement a `MessageBodyReader` and `MessageBodyWriter` to add support for custom request and response data types
2.1.8. Implement `ExceptionMapper` to map a custom `Exception` to a response.
2.1.9. Use `Request` to add support for HTTP preconditions.
2.1.10. Implement the functionality of the JAX-RS resource's methods.
2.1.11. Use `@Path` on a method to define a subresource.
2.1.12. Configure deployment descriptor (`web.xml`) for base URL pattern, HTTP security (via `security-constraints` in `web.xml`)
2.1.13. Compile and package
2.1.14. Deploy the web service in a Java EE servlet container

3. Create a SOAP based web service implemented by an EJB component
3.1. Create a web service starting from a WSDL file using JAX-WS
3.1.1. Use `wsimport` tool to generate artifacts and use customization files for wsimports if needed
3.1.2. Create an EJB web service implementations using annotations (`@Stateless` or `@Singleton`)
3.1.3. Configure deployment descriptors (`ejb-jar.xml`, `webservices.xml`) for transactions, etc.
3.1.4. Configure container role based access control via method-permissions in `ejb-jar.xml` or via access control annotations on EJB.
3.1.5. Configure caller authentication and message protection; either by Servlet Container via `web.xml`, and/or by JAX-WS message processing runtime.
3.1.6. Compile and package the web service into a EAR/WAR file (Java EE 6 - WAR can also have EJBs).
3.1.7. Deploy the web service into a Java EE container.
3.2. Create a web service starting from a Java source using JAX-WS
3.2.1. Use `wsgen` tool to generate artifacts in Java EE5 from EJB classes (optional in Java EE 6 - as artifacts are generated at run time).
3.2.2. Configure deployment descriptors (`ejb-jar.xml`, `webservices.xml`) for transactions, etc.
3.2.3. Configure container role based access control via method-permissions in

ejb-jar.xml or via access control annotations on EJB.
3.2.4. Configure caller authentication and message protection; either by Servlet Container via web.xml, and/or by JAX-WS message processing runtime.
3.2.5. Compile and package the Web Service into a WAR/EAR file.
3.2.6. Deploy the Web Service into a Java EE container.

4. Create a RESTful Web Service implemented by an EJB component
   4.1. Create a Web Service using JAX-RS from EJB classes.
      4.1.1. Annotate an enterprise bean class with a @Path annotation to respond to URL patterns.
      4.1.2. Annotate the class's methods to respond to HTTP requests using the corresponding JAX-RS annotations (@GET, @POST, etc.).
      4.1.3. Use the JAX-RS @Produces and @Consumes annotations to specify the input and output resources for the RESTful Web Service.
      4.1.4. Implement the functionality of the JAX-WS resource's methods.
      4.1.5. Configure container role based access control via method-permissions in ejb-jar.xml or via access control annotations on EJB.
      4.1.6. Configure caller authentication (for access control protected methods) and message protection by Servlet Container via web.xml.
      4.1.7. Compile and package.
      4.1.8. Deploy the web service in a Java EE servlet container.
5. Configure Java EE security for a SOAP web service
   5.1. Configure security requirements of service using Java EE-container based security (overlaps with steps in other tasks - repeated here for convenience)
      5.1.1. Configure security requirements through deployment descriptors (web.xml, webservices.xml) for a Servlet-based web service endpoint: container authorization, caller authentication, and message protection. JAX-WS runtime may also be configured to perform message layer authentication and protection.
      5.1.2. Configure security requirements through deployment descriptors (ejb-jar.xml, webservices.xml) for EJB-based web service endpoint:
      5.1.3. Configure security requirements through deployment descriptor (web.xml) for JAX-RS based web service endpoint.
6. Create a web service client for a SOAP based web service
   6.1. Create a standalone client.
      6.1.1. Use wsimport to generate artifacts.
      6.1.2. Create a client application using these artifacts.
      6.1.3. Package and deploy accordingly.
   6.2. Create a client in a managed component in a EE container.
      6.2.1. Use wsimport to generate artifacts.
      6.2.2. Using @WebserviceRef in the client application.
      6.2.3. Package and deploy accordingly.
7. Create a web service client for a RESTful web service
   7.1. Use a browser to access a JAX-RS resource
   7.2. Use the java.net.* APIs to access a JAX-RS resource.
   7.3. Use java.net.Authenticator to access a secure JAX-RS resource.
   7.4. Use Ajax to access a JAX-RS resource.
   7.5. Use the Jersey client API to access a JAX-RS resource.
   7.6. Use the JAX-WS HTTP binding to access a JAX-RS resource.
8. Create a SOAP based web service using Java SE platform.
   8.1. Create a web service starting from a WSDL file using JAX-WS.
      8.1.1. Use wsimport tool to generate artifacts and use customization files for wsimports if needed.
      8.1.2. Build the web service implementation using the above artifacts.
      8.1.3. Use Endpoint API to configure and deploy it in Java SE 6 platform.
   8.2. Create a web service starting from a Java source using JAX-WS.
      8.2.1. Use wsgen tool to generate artifacts in Java EE5 (optional in Java EE6 - as artifacts are generated at run time)
      8.2.2. Use Endpoint API to configure and deploy it in Java SE 6 platform.
9. Create handlers for SOAP web services.
   9.1. Configure SOAP and logical handlers on the server side.

2.2. Describe the SOAP Processing and Extensibility Model.
2.3. Describe SOAP Message Construct and create a SOAP message that contains an attachment.

3. Describing and Publishing (WSDL and UDDI)
   3.1. Explain the use of WSDL in Web Services, including a description of WSDL's basic elements, binding mechanisms and the basic WSDL operation types as limited by the WS-I Basic Profile 1.1.
   3.2. Describe how WSDL enables one to separate the description of the abstract functionality offered by a service from concrete details of a service description such as "how" and "where" that functionality is offered.
   3.3. Describe the Component Model of WSDL including Descriptions, Interfaces, Bindings, Services and Endpoints.
   3.4. Describe the basic functions provided by the UDDI Publish and Inquiry APIs to interact with a UDDI business registry.
4. JAX-WS
   4.1. Explain JAX-WS technology for building web services and client that communicate using XML.
   4.2. Given a set of requirements for a Web Service, such as transactional needs, and security requirements, design and develop Web Service applications that use JAX-WS technology.
   4.3. Describe the Integrated Stack (I-Stack) which consist of JAX-WS, JAXB, StAX, SAAJ.
   4.4. Describe and compare JAX-WS development approaches.
   4.5. Describe the features of JAX-WS including the usage of Java Annotations.
   4.6. Describe the architecture of JAX-WS including the Tools SPI that define the contract between JAX-WS tools and Java EE.
   4.7. Describe creating a Web Service using JAX-WS.
   4.8. Describe JAX-WS Client Communications Models.
   4.9. Given a set of requirements, design and develop a Web Service client, such as a Java EE client and a stand-alone client, using JAX-WS.
   4.10. Given a set of requirements, create and configure a Web Service client that accesses a stateful Web Service.
5. REST, JSON, SOAP and XML Processing APIs (JAXP, JAXB and SAAJ)
   5.1. Describe the characteristics of REST Web Services.
   5.2. Describe the characteristics of JSON Web Services.
   5.3. Compare SOAP Web Services to REST Web Services.
   5.4. Compare SOAP Web Services to JSON Web Services.
   5.5. Describe the functions and capabilities of the APIs included within JAXP.
   5.6. Describe the functions and capabilities of JAXB, including the JAXB process flow, such as XML-to-Java and Java-to-XML, and the binding and validation mechanisms provided by JAXB.
   5.7. Create and use a SOAP message with attachments using the SAAJ APIs.
6. JAXR
   6.1. Describe the function of JAXR in Web Service architectural model, the two basic levels of business registry functionality supported by JAXR, and the function of the basic JAXR business objects and how they map to the UDDI data structures.
   6.2. Create JAXR client to connect to a UDDI business registry, execute queries to locate services that meet specific requirements, and publish or update information about a business service.
7. J2EE Web Services
   7.1. Identify the characteristics of and the services and APIs included in the Java EE platform.
   7.2. Explain the benefits of using the Java EE platform for creating and deploying Web Service applications.
   7.3. Describe the functions and capabilities of the JAXP, DOM, SAX, StAX, JAXR, JAXB, JAX-WS and SAAJ in the Java EE platform.
   7.4. Describe the role of the WS-I Basic Profile when designing Java EE Web Services.
8. Security
   8.1. Explain basic security mechanisms including: transport level security, such as basic and mutual authentication and SSL, message level security, XML encryption, XML Digital Signature, and federated identity and trust.
   8.2. Identify the purpose and benefits of Web services security oriented initiatives and

Preface

If you believe you have found an error in the "Oracle Certified Expert, Java Platform, Enterprise

Edition 6 Web Services Developer Quiz" or have a suggestion to improve it, please send an e-mail to me. Please, indicate the topic and page URL.

Part I. Exam Objectives

Chapter 1. Create an SOAP web service in a servlet container

1.1. Create a web service starting from a WSDL file using JAX-WS

1.1.1. Use `wsimport` tool to generate artifacts from WSDL

Question 01010101

What statements are true about the following WSDL `operation`:

```
<portType name="SubmitPurchaseOrderPortType">
    <operation name="SubmitPurchaseOrder">
        <input name="order" message="SubmitPurchaseOrderMessage"/>
    </operation>
</portType>
```

Options (select 2):

1. This is a notification operation.
2. This is a one-way operation.
3. This is a two-way operation.
4. JAX-WS implementation may NOT throw any checked exceptions.
5. JAX-WS implementation may throw only unchecked exceptions.
6. JAX-WS implementation may throw any exceptions.

Question 01010102

Given the following WSDL fragment, which statement is true:

```
<portType name="BookQuote">
    <operation name="getBookPrice">
        <input name="isbn" message="ns1:GetBookPriceRequest"/>
        <output name="price" message="ns1:GetBookPriceResponse"/>
        <fault name="InvalidArgumentFault" message="ns1:InvalidArgumentFault"/>
        <fault name="SecurityFault" message="ns1:SecurityFault"/>
    </operation>
</portType>
```

Options (select 1):

1. The `BookQuote` operation is invalid, because it has two faults.
2. The `BookQuote` operation represents a solicit-response message exchange pattern.

3. Both `InvalidArgumentFault` and `SecurityFault` must be runtime exceptions.
4. None of the above.

Question 01010103

Snorcle, Inc. hired you to create a JAX-WS Web Service client for a Stock Quotes Web Service. You are given a WSDL file for the Web Service which implemented with C# on .NET platform. What is the correct sequence of steps?

Options (select 1):

1. Use RMI technology to look up `Remote` object in registry, then invoke object's methods from the client.
2. Use `wsimport` to generate Service Endpoint Interface object, then invoke SEI's methods from the client.
3. Use JAXP technology to parse WSDL, create Service Endpoint Interface from factory, then invoke SEI's methods from the client.
4. Use `wsgen` to generate Service Endpoint Interface object, then invoke SEI's methods from the client.
5. None of the above, because JAX-WS client can work only with Web Services deployed on JEE platform.

1.1.2. Use external and embedded `<jaxws:package>`, `<jaxws:enableWrapperStyle>`, `<jaxws:class>` customizations

blah-blah

1.1.3. Use JAXB customizations to configure mapping.

blah-blah

1.1.4. Build the web service implementation using the above artifacts.

blah-blah

1.1.5. Access `MessageContext.SERVLET_CONTEXT` from the injected `WebServiceContext`

Question 01010501

A developer is trying to access `WebServicesContext` object from the endpoint implementation class.

```
@WebService
public class EndpointJSE {

    // get web service context
    ...

    @WebMethod
    public String getUserPrincipal() {
        Principal principal = ctx.getUserPrincipal();
        return principal.getName();
    }
}
```

Which code fragment correctly demonstrates initializing of `WebServicesContext` in JAX-WS

endpoint?

Options (select 1):

1. ```
   @Resource
   private WebServiceContext ctx;
   ```

2. ```
   @MessageContext
   private WebServiceContext ctx;
   ```

3. ```
   @WebServiceContext
   private WebServiceContext ctx;
   ```

4. ```
   @Object
   private WebServiceContext ctx;
   ```

5. ```
   @WebServiceContext
   private Object ctx;
   ```

1.1.6. Configure deployment descriptors (`web.xml`, `webservices.xml`) for URL patterns, HTTP security, container authorization, caller authentication, and message protection. JAX-WS runtime may also be configured to perform message layer authentication and protection.

Question 01010601

A company wants to protect the access to their public Inventory Web Services using basic authentication. The Inventory Web Services are currently deployed in Java EE6 servlet container. What file should developer use to configure the security requirement?

Options (select 1):

1. `webservices.xml`
2. `web.xml`
3. `sun-web.xml`
4. `security.xml`

1.1.7. Compile and package the Web Service into a WAR file

blah-blah

1.1.8. Deploy the Web Service into a Java EE servlet container

blah-blah

1.2. Create a Web Service starting from a Java source using JAX-WS

1.2.1. Use `@WebService` to indicate a service

blah-blah

1.2.2. Use `@WebMethod`, `@WebMethod(exclude)` to indicate service methods
   ```

Question 01020201

Given the following Web Service code, what would be the name of the operation in the generated WSDL?

```
@WebService(serviceName="WeatherService",
            name="Weather",
            portName="WeatherServicePort")
public class WeatherWS {
    public String getWeather() {
        return "Sunny";
    }
}
```

Options (select 1):

1. `getWeatherMethod`
2. `getWeatherOperation`
3. `getWeatherWebMethod`
4. `getWeather`
5. None of the above.

Question 01020202

What statement is correct about the Weather Web Service:

```
@WebService(serviceName="WeatherService",
            name="Weather",
            portName="WeatherServicePort")
public class WeatherWS {

    @WebMethod()
    public String getWeather() {
        return "Sunny";
    }

    @WebMethod()
    public String getWeather(@WebParam(name = "city") String city) {
        return "Weather in " + city + ": Sunny";
    }
}
```

Options (select 1):

1. Weather Web Service is not well encapsulated, because `@WebMethod` annotation may not be applied to `public` methods.
2. Weather Web Service is invalid, because `@WebMethod` annotation may only be applied to SEI methods.
3. This is a correct example of Web Service method overloading.
4. You need to use `@WebMethod(operationName="getWeatherByCity")` for the second method to sucessfully run Weather Web Service.
5. None of the above.

Question 01020203

Given the following Web Services implementations:

```
@WebService
public class BaseCatalogWS {

    @WebMethod
    public int getIDByName(String name) {
        ...
    }

    public String getNameByID(int id) {
        ...
    }
}
```

and

```
@WebService
public class CatalogWS extends BaseCatalogWS {

    @WebMethod
    public String getTitle(String isbn)  {
        ...
    }

    public String getAuthor(String isbn)  {
        ...
    }

    private String getPublisher(String isbn)  {
        ...
    }
}
```

How many web methods the `CatalogWS` exposes?

Options (select 1):

1. 1
2. 2
3. 3
4. 4
5. 5
6. None of the above.

1.2.3. Use `@SOAPBinding` to select doc/lit, doc/bare, rpc/lit style of web service

Question 01020301

What statement is correct about the Hello Web Service:

```
@WebService()
public class HelloWS {

    @WebMethod(operationName="sayHelloToAnonymous")
    public String sayHello() {
        return "Hello, Anonymous !";
    }

    @WebMethod(operationName="sayHelloToPerson")
    public String sayHello(@WebParam(name="name") String name) {
```

```
        return "Hello, " + name + "!";
    }
}
```

Options (select 2):

1. Hello Web Service can be successfully deployed without changes.
2. Hello Web Service can be successfully deployed if you rename Java methods to different names.
3. Hello Web Service can be successfully deployed if you annotate class with `@SOAPBinding(style=Style.DOCUMENT, parameterStyle=ParameterStyle.BARE, use=Use.LITERAL)`
4. Hello Web Service can be successfully deployed if you annotate class with `@SOAPBinding(style=Style.DOCUMENT, parameterStyle=ParameterStyle.WRAPPED, use=Use.LITERAL)`
5. Hello Web Service can be successfully deployed if you annotate class with `@SOAPBinding()`
6. Hello Web Service can be successfully deployed if you annotate class with `@RequestWrapper(className="Hello")` `@ResponseWrapper(className="HelloResponse")`

1.2.4. Use `@Oneway` where the service doesn't have any response

blah-blah

1.2.5. Use `@WebParam`, and `@WebResult` to customize parameter and operation names

blah-blah

1.2.6. Use checked exceptions to indicate service specific faults.

blah-blah

1.2.7. Use `wsgen` tool to generate artifacts in Java EE5 (optional in Java EE6, as artifacts are generated at run time).

blah-blah

1.2.8. Configure deployment descriptors (`web.xml`, `webservices.xml`) for URL patterns, HTTP security, container authorization, caller authentication, and message protection. JAX-WS runtime may also be configured to perform message layer authentication and protection.

Question 01020801

A developer needs to create a Web Service that supports client's session. To avoid memory leaks on the server, developer wants user session to be expired after 20 minutes of inactivity. Which config file should the developer use?

Options (select 1):

1. `web.xml`
2. `webservices.xml`

3. `jax-ws.xml`
4. `sun-jaxws.xml`
5. None of the above.

1.2.9. Compile and package the Web Service into a WAR file

blah-blah

1.2.10. Deploy the web service into a Java EE servlet container

Question 01021001

Developer created and deployed the following JAX-WS Web Service using "Java-to-WSDL" approach:

```
@WebService
public class CalculatorWS {
        ...
}
```

At what URL this Web Service will be published by default (assume that servlet context "`CalculatorApp`" is correct and properly configured)?

Options (select 1):

1. `http://server.com/CalculatorApp/CalculatorWS`
2. `http://server.com/CalculatorApp/CalculatorWSPortBinding`
3. `http://server.com/CalculatorApp/CalculatorWSPort`
4. `http://server.com/CalculatorApp/CalculatorWSService`
5. `http://server.com/CalculatorApp/CalculatorWSPortType`

Question 01021002

Given the following Weather Web Service implementation:

```
@WebService(serviceName="WeatherService",
            name="Weather",
            portName="WeatherServicePort")
public class WeatherWS {
    public String getWeather() {
        return "Sunny";
    }
}
```

After deployment in JAX-WS enabled container, what URL can be used to access the Weather Web Service (assume that servlet context "`weather`" is correct and properly configured)?

Options (select 1):

1. `http://server.com/weather/Weather`
2. `http://server.com/weather/WeatherWS`
3. `http://server.com/weather/WeatherService`
4. `http://server.com/weather/WeatherServicePort`
5. `http://server.com/weather/WeatherWSService`

Chapter 2. Create a RESTful web service in a servlet container

2.1. Create a web service using JAX-RS, refer to Jersey implementation for examples

2.1.1. Annotate a class with a `@Path` annotation to respond to URI templates.

blah-blah

2.1.2. Annotate the class's methods to respond to HTTP requests using the corresponding JAX-RS annotations (`@GET`, `@POST`, etc.).

blah-blah

2.1.3. Use the JAX-RS `@Consumes` and `@Produces` annotations to specify the input and output formats for the RESTful Web Service.

Question 02010301

Snorcle, Inc. has deployed internal application to manage customers. To support interoperability with large base of standalone intranet applications, the services provided by this back-end application are exposed as XML-based RESTful Web Services. You are asked by management to add support of AJAX-based JSON-enabled web application clients which should consume the same RESTful Web Services. Estimate development efforts required to accomplish this requirement.

Options (select 1):

1. Development will not be able to accomplish this requirement, because same RESTful Web Service can not support in parallel both XML and JSON output formats.
2. Development can easily add new JSON output format to existing XML RESTful Web Service.
3. Development will be able to accomplish this requirement with significant amount of new code, in particular it will require create duplicate collection of resources: one for XML output, another for JSON output.
4. Development will not be able to accomplish this requirement, because RESTful Web Services do not support JSON output format.

2.1.4. Use `@PathParam`, `@QueryParam`, `@MatrixParam` and `@HeaderParam` to extract request data.

blah-blah

2.1.5. Use the `UriInfo` and `UriBuilder` to create URIs that refer to resources in the service.

blah-blah

2.1.6. Use `ResponseBuilder` to create response with customized status and additional metadata.

blah-blah

2.1.7. Implement a `MessageBodyReader` and `MessageBodyWriter` to add support for custom request and response data types

blah-blah

2.1.8. Implement `ExceptionMapper` to map a custom `Exception` to a response.

blah-blah

2.1.9. Use `Request` to add support for HTTP preconditions.

blah-blah

2.1.10. Implement the functionality of the JAX-RS resource's methods.

blah-blah

2.1.11. Use `@Path` on a method to define a subresource.

blah-blah

2.1.12. Configure deployment descriptor (`web.xml`) for base URL pattern, HTTP security (via `security-constraints` in `web.xml`)

Question 02011201

Given the following JAX-RS resource class:

```
@Path("/")
public class ClientResource {

    public static final HashMap<Integer, String> map = new HashMap<Integer, String>();
    static {
        map.put(1, "Mikalai Zaikin");
        map.put(2, "Volha Zaikina");
    }

    @GET
    @Path("/client/{id}")
    @Produces(MediaType.TEXT_PLAIN)
    public String retrieve(@PathParam("id") int clientNo) {
        return map.get(clientNo);
    }

    @POST
    @Path("/client")
    @Produces(MediaType.TEXT_HTML)
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    public String update(@FormParam("custno") int clientNo, @FormParam("custname") String name) {
        map.put(clientNo, name);
        return "'"+name+"' was added !";
    }
}
```

And a given `web.xml` deployment descriptor fragment:

```
...
<servlet>
    <servlet-name>ServletAdaptor</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
```

```
<servlet-mapping>
    <servlet-name>ServletAdaptor</servlet-name>
    <url-pattern>/resources/*</url-pattern>
</servlet-mapping>
...
```

How can you restrict unauthenticated users from accessing the `update(...)` method?

Options (select 1):

1.
```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>
            resources
        </web-resource-name>
        <url-pattern>/resources</url-pattern>
        <http-method>POST</http-method>
    </web-resource-collection>
    ...
</security-constraint>
```

2.
```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>
            resources
        </web-resource-name>
        <url-pattern>/client</url-pattern>
        <http-method>POST</http-method>
    </web-resource-collection>
    ...
</security-constraint>
```

3.
```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>
            resources
        </web-resource-name>
        <url-pattern>/update</url-pattern>
        <http-method>POST</http-method>
    </web-resource-collection>
    ...
</security-constraint>
```

4.
```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>
            resources
        </web-resource-name>
        <url-pattern>/resources/*</url-pattern>
        <http-method>POST</http-method>
    </web-resource-collection>
    ...
</security-constraint>
```

2.1.13. Compile and package

blah-blah

2.1.14. Deploy the web service in a Java EE servlet container

blah-blah

Chapter 3. Create a SOAP based web service implemented by an EJB component

3.1. Create a web service starting from a WSDL file using JAX-WS

3.1.1. Use `wsimport` tool to generate artifacts and use customization files for wsimports if needed

blah-blah

3.1.2. Create an EJB web service implementations using annotations (`@Stateless` or `@Singleton`)

blah-blah

3.1.3. Configure deployment descriptors (`ejb-jar.xml`, `webservices.xml`) for transactions, etc.

Question 03010301

The developer was asked to customize the following Weather Forecast Web Service:

```
package by.boot.java;

import javax.ejb.Stateless;
import javax.jws.WebService;

@Stateless(name="WeatherEJB")
@WebService(name="WeatherForecast", serviceName="WeatherService")
public class WeatherWS {
    public String getWeather() {
        return "Sunny";
    }
}
```

The new requirement is to add a handler chain to the existing Weather Forecast Web Service without changing of Java code. How this can be done?

Options (select 1):

1. Add `webservices.xml` deployment descriptor and link handler chain to the EJB using the following fragment:

```
...
<service-impl-bean>
    <ejb-link>
        WeatherEJB
    </ejb-link>
</service-impl-bean>
...
```

2. Add `webservices.xml` deployment descriptor and link handler chain to the EJB using the following fragment:

```
...
<service-impl-bean>
    <ejb-link>
        WeatherForecast
    </ejb-link>
</service-impl-bean>
...
```

3. Add `webservices.xml` deployment descriptor and link handler chain to the EJB using the following fragment:

```
...
<service-impl-bean>
    <ejb-link>
        WeatherService
    </ejb-link>
</service-impl-bean>
...
```

4. Add `webservices.xml` deployment descriptor and link handler chain to the EJB using the following fragment:

```
...
<service-impl-bean>
    <ejb-link>
        WeatherWS
    </ejb-link>
</service-impl-bean>
...
```

5. It is not possible without Java code modification.

3.1.4. Configure container role based access control via method-permissions in `ejb-jar.xml` or via access control annotations on EJB.

Question 03010401

A developer created a JAX-WS Web Service endpoint using a Singleton session bean:

```
@Singleton
@WebService
public class StatusBean {
    ...
    public String getState() {
        return state;
    }
}
```

Now developer needs to add role based access control to the Web Service business logic. What approaches can the developer use?

Options (select 2):

1. By using common annotations for the Java Platform.
2. By using `method-permission` element in `web.xml`
3. By using `method-permission` element in `ejb-jar.xml`
4. By using `security-constraint` element in `web.xml`
5. By using `auth-constraint` element in `ejb-jar.xml`

3.1.5. Configure caller authentication and message protection; either by Servlet Container via `web.xml`, and/or by JAX-WS message processing runtime.

blah-blah

3.1.6. Compile and package the web service into a EAR/WAR file (Java EE 6 - WAR can also have EJBs).

blah-blah

3.1.7. Deploy the web service into a Java EE container.

blah-blah

3.2. Create a web service starting from a Java source using JAX-WS

3.2.1. Use `wsgen` tool to generate artifacts in Java EE5 from EJB classes (optional in Java EE 6 - as artifacts are generated at run time).

blah-blah

3.2.2. Configure deployment descriptors (`ejb-jar.xml`, `webservices.xml`) for transactions, etc.

blah-blah

3.2.3. Configure container role based access control via method-permissions in `ejb-jar.xml` or via access control annotations on EJB.

blah-blah

3.2.4. Configure caller authentication and message protection; either by Servlet Container via `web.xml`, and/or by JAX-WS message processing runtime.

blah-blah

3.2.5. Compile and package the Web Service into a WAR/EAR file.

blah-blah

3.2.6. Deploy the Web Service into a Java EE container.

blah-blah

Chapter 4. Create a RESTful Web Service implemented by an EJB component

4.1. Create a Web Service using JAX-RS from EJB classes.

4.1.1. Annotate an enterprise bean class with a `@Path` annotation to respond to URL patterns.

blah-blah

4.1.2. Annotate the class's methods to respond to HTTP requests using the corresponding JAX-RS annotations (`@GET`, `@POST`, etc.).

blah-blah

4.1.3. Use the JAX-RS `@Produces` and `@Consumes` annotations to specify the input and output resources for the RESTful Web Service.

blah-blah

4.1.4. Implement the functionality of the JAX-WS resource's methods.

blah-blah

4.1.5. Configure container role based access control via method-permissions in `ejb-jar.xml` or via access control annotations on EJB.

Question 04010501

Given the following JAX-RS resource class:

```
@Path(value="/addresses")
@PermitAll
@Stateless
public class AddressBookResource {

    @GET
    @Produces(value="text/plain")
    public String getList() {
        ...
    }

    @RolesAllowed("admin")
    @PUT
    public void updateList(String addr) {
        ...
    }
}
```

Which statement describes the configuration which would be required to support the access control shown above?

Options (select 1):

1. No further configuration is required, Java EE runtime will read annotation and configure web container automatically.
2. Developer must configure web container to authenticate access to the resource.
3. Developer must configure EJB container to authenticate access to the resource.
4. No further configuration is required, because class level annotation specifies that all security roles are permitted to access the JAX-RS resource.

4.1.6. Configure caller authentication (for access control protected methods) and message protection by Servlet Container via `web.xml`.

blah-blah

4.1.7. Compile and package.

blah-blah

4.1.8. Deploy the web service in a Java EE servlet container.

blah-blah

Chapter 5. Configure Java EE security for a SOAP web service

5.1. Configure security requirements of service using Java EE-container based security (overlaps with steps in other tasks - repeated here for convenience)

5.1.1. Configure security requirements through deployment descriptors (`web.xml`, `webservices.xml`) for a Servlet-based web service endpoint: container authorization, caller authentication, and message protection. JAX-WS runtime may also be configured to perform message layer authentication and protection.

blah-blah

5.1.2. Configure security requirements through deployment descriptors (`ejb-jar.xml`, `webservices.xml`) for EJB-based web service endpoint:

5.1.2.1. Configure transactional support.

blah

5.1.2.2. Configure container role based access control via method-permissions in `ejb-jar.xml` or via access control annotations on EJB.

Question 0501020201

Given the following Web Service endpoint implementation:

```
@WebService
@Singleton(name="OrderBean")
public class SecureOrderBean {
    ...
}
```

and the following fragment from `ejb-jar.xml` deployment descriptor:

```
...
<assembly-descriptor>
    ...
    <security-role>
        <role-name>manager</role-name>
    </security-role>
    <method-permission>
        <role-name>manager</role-name>
        <method>
            <ejb-name>OrderBean</ejb-name>
```

```
            <method-name>*</method-name>
        </method>
    </method-permission>
    ...
</assembly-descriptor>
...
```

Which statement is true about security roles of the client of this SEI ?

Options (select 2):

1. Only EJB client must be in `manager` role.
2. Only JAX-WS client must be in `manager` role.
3. Both EJB and JAX-WS clients must be in `manager` role.
4. Both EJB and JAX-WS clients may be in any role.

5.1.2.3. Configure caller authentication and message protection; either by Servlet container via `web.xml`, and/or by JAX-WS message processing runtime.

blah

5.1.3. Configure security requirements through deployment descriptor (`web.xml`) for JAX-RS based web service endpoint.

blah-blah

Chapter 6. Create a web service client for a SOAP based web service

6.1. Create a standalone client.

Question 060101

Which statements are true about `javax.xml.ws.Service` type?

Options (select 2):

1. It belongs to JAX-RS client-side API.
2. It belongs to JAX-WS client-side API.
3. It belongs to JAX-RS server-side API.
4. It belongs to JAX-WS server-side API.
5. It is a concrete class.
6. It is an abstract interface.
7. It is an annotation.

6.1.1. Use `wsimport` to generate artifacts.

blah-blah

6.1.2. Create a client application using these artifacts.

Question 06010201

Which of the following can developer achieve by using `BindingProvider` interface of JAX-WS Web Service client?

Options (select 2):

1. Specify user name and password for BASIC authentication.
2. Set custom `HandlerResolver` to configure programmatically handler chain on client.
3. Validate against XML Schema the outgoing SOAP message.
4. Specify user-defined Web Service endpoint HTTP address.
5. Validate against XML Schema the incoming SOAP message.

6.1.2.1. Invoke web service synchronously or asynchronously.

blah-blah

6.1.3. Package and deploy accordingly.

blah-blah

6.2. Create a client in a managed component in a EE container.

6.2.1. Use `wsimport` to generate artifacts.

blah-blah

6.2.2. Using `@WebserviceRef` in the client application.

Question 06020201

Developer is asked to write a Servlet that must act as the client for existing JAX-WS Web Service:

```
@WebService
public interface Catalog {
    @WebMethod
    String getTitle(String id);
}
```

```
@WebService(serviceName="CatalogService", name="Catalog")
public class CatalogWS implements Catalog {
    @WebMethod
    public String getTitle(String id)  {
        ...
    }
}
```

The developer decided to use the `@WebServiceRef` annotation to inject a reference to the service he needs to invoke. Which code snipped demonstrates correct usage of the `@WebServiceRef` annotation in JAX-WS container-managed client?

Options (select 3):

1.
```
@WebServlet(name="CatalogServlet", urlPatterns={"/CatalogServlet"})
public class CatalogServlet extends HttpServlet {

    @WebServiceRef(value=CatalogService.class)
    Catalog ref;
```

```
    ...
```

2.
```
@WebServlet(name="CatalogServlet", urlPatterns={"/CatalogServlet"})
public class CatalogServlet extends HttpServlet {

    @WebServiceRef(type=CatalogService.class)
    Catalog ref;
    ...
```

3.
```
@WebServlet(name="CatalogServlet", urlPatterns={"/CatalogServlet"})
public class CatalogServlet extends HttpServlet {

    @WebServiceRef(CatalogService.class)
    Catalog ref;
    ...
```

4.
```
@WebServlet(name="CatalogServlet", urlPatterns={"/CatalogServlet"})
public class CatalogServlet extends HttpServlet {

    @WebServiceRef
    CatalogService ref;
    ...
```

5.
```
@WebServlet(name="CatalogServlet", urlPatterns={"/CatalogServlet"})
public class CatalogServlet extends HttpServlet {

    @WebServiceRef
    CatalogWS ref;
    ...
```

6.
```
@WebServlet(name="CatalogServlet", urlPatterns={"/CatalogServlet"})
public class CatalogServlet extends HttpServlet {

    @WebServiceRef
    Catalog ref;
    ...
```

6.2.3. Package and deploy accordingly.

blah-blah

Chapter 7. Create a web service client for a RESTful web service

7.1. Use a browser to access a JAX-RS resource

Question 070101

What HTTP methods are typically used by RESTful Web Services?

Options (select 4):

1. `OPTIONS`
2. `GET`
3. `HEAD`
4. `POST`
5. `PUT`

6. DELETE
7. TRACE
8. CONNECT

7.2. Use the `java.net.*` APIs to access a JAX-RS resource.

Question 070201

You are hired by Snorcle, Inc. to develop a REST Web Service client which retrieves information about Employee by his ID. The Web Service accepts HTTP POST requests. What code should be added at the line #5 to perform Employee information request? Assume that the rest of the code is valid.

```
 1
 2 ...
 3 URL url = new URL("http://snorcle.com/hr/employee");
 4 URLConnection connection = url.openConnection();
 5 // add new code here if needed
 6
 7 OutputStream os = connection.getOutputStream();
 8 OutputStreamWriter out = new OutputStreamWriter(os);
 9 out.write("id=" + id);
10 out.close();
11 ...
12
```

Options (select 1):

1. `connection.setRequestProperty("Method", "POST");`
2. `connection.setAllowUserInteraction(true);`
3. `connection.setDoOutput(true);`
4. `connection.setMethod("POST");`
5. `connection.setHttpMethod("POST");`
6. You don't need to add anything. The shown client code will send POST HTTP request by default.

Question 070202

You are hired by Snorcle, Inc. to develop a REST Web Service client which retrieves information about Employee by her ID. What code should be added at the line #5 to perform Employee information request? Assume that the rest of the code is valid.

```
 1
 2 ...
 3 URL url = new URL("http://snorcle.com/hr/employee/" + id);
 4 URLConnection connection = url.openConnection();
 5 // add new code here if needed
 6
 7 InputStream is = connection.getInputStream();
 8 InputStreamReader isr = new InputStreamReader(is);
 9 BufferedReader in = new BufferedReader(isr);
10 ...
11
```

Options (select 2):

1. ```
   connection.setAllowUserInteraction(true);
   ```

2. ```
   connection.setDoInput(true);
   ```

3. ```
   connection.setAllowInput(true);
   ```

4. You don't need to add anything. The shown client code will be able to read HTTP response by default.

7.3. Use `java.net.Authenticator` to access a secure JAX-RS resource.

blah-blah

7.4. Use Ajax to access a JAX-RS resource.

Question 070401

Snorcle, Inc. is refactoring its own intranet web site. Employees can retrieve and browse company customers in a web browser. Previously, customer information was retrieved in a separate window when user clicked the link for each customer. Now, the customer information must be generated by a Web Service deployed in the intranet, and must be displayed in the already opened window without reload. Which client-side technologies can be used for easiest refactoring?

Options (select 3):

1. JAXP
2. SAAJ
3. REST
4. XML
5. JSON
6. JavaScript

7.5. Use the Jersey client API to access a JAX-RS resource.

blah-blah

7.6. Use the JAX-WS HTTP binding to access a JAX-RS resource.

blah-blah

Chapter 8. Create a SOAP based web service using Java SE platform.

8.1. Create a web service starting from a WSDL file using JAX-WS.

Question 080101

You are hired by Snorcle, Inc. to develop a Stock Broker (SB) Web Service which will be consumed by partner companies. Clients use different platforms (JEE, .NET). In order to reduce load of the Web Service clients will be asked to implement client-side validation of requests. Which approach should you use?

Options (select 1):
   ```

1. Code-first ("Java-to-WSDL") approach.
2. Contract-first ("WSDL-to-Java") approach.
3. Meet in the middle approach.
4. None of the above.

8.1.1. Use `wsimport` tool to generate artifacts and use customization files for wsimports if needed.

blah-blah

8.1.2. Build the web service implementation using the above artifacts.

blah-blah

8.1.3. Use `Endpoint` API to configure and deploy it in Java SE 6 platform.

blah-blah

8.2. Create a web service starting from a Java source using JAX-WS.

Question 080201

You are hired by Snorcle, Inc. to develop an order-tracking Web Service which will be consumed by partner companies. Java team from another department would like to reuse your code as soon as possible. Which approach should you choose?

Options (select 1):

1. Code-first ("Java-to-WSDL") approach.
2. Contract-first ("WSDL-to-Java") approach.
3. Meet in the middle approach.
4. None of the above.

8.2.1. Use `wsgen` tool to generate artifacts in Java EE5 (optional in Java EE6 - as artifacts are generated at run time)

blah-blah

8.2.2. Use `Endpoint` API to configure and deploy it in Java SE 6 platform.

blah-blah

Chapter 9. Create handlers for SOAP web services.

9.1. Configure SOAP and logical handlers on the server side.

Question 090101

What statements are true about the intermediary node when processing SOAP message:

Options (select 2):

1. If `actor` attribute of SOAP header identifies the intermediary node, the node must not forward that header element.
2. An intermediary node must forward SOAP header element if its `actor` attribute identifies the

node.

3. An intermediary node may forward SOAP header element if its `actor` attribute identifies the node.
4. An intermediary node may insert new header elements in SOAP header.
5. An intermediary node may insert new elements in SOAP body.

9.1.1. Use `@HandlerChain` annotation.

Question 09010A01

Given the Web Service code:

```
@WebService(name = "Handler", targetNamespace = "http://java.boot.by")
@HandlerChain(file="handler-chain.xml")
public class HandlerWS {

    @Resource
    WebServiceContext ctx;

    @WebMethod()
    public String getProperty(String propertyName) {
        return (String) ctx.getMessageContext().get(propertyName);
    }
}
```

and the `handler-chain.xml` configuration file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<handler-chains xmlns="http://java.sun.com/xml/ns/javaee">
    <handler-chain>
        <handler>
            <handler-class>
                LogicalHandler1
            </handler-class>
        </handler>
        <handler>
            <handler-class>
                SOAPHandler2
            </handler-class>
        </handler>
    </handler-chain>
    <handler-chain>
        <handler>
            <handler-class>
                SOAPHandler3
            </handler-class>
        </handler>
    </handler-chain>
    <handler-chain>
        <handler>
            <handler-class>
                LogicalHandler4
            </handler-class>
        </handler>
    </handler-chain>
</handler-chains>
```

Which statement is true about incoming SOAP message processing?

Options (select 1):

1. Handlers executed in the following order: `LogicalHandler1`, `SOAPHandler2`, `SOAPHandler3`, `LogicalHandler4`.
2. Handlers executed in the following order: `LogicalHandler1`, `LogicalHandler4`, `SOAPHandler2`, `SOAPHandler3`.
3. Handlers executed in the following order: `SOAPHandler2`, `SOAPHandler3`, `LogicalHandler1`, `LogicalHandler4`.
4. `RuntimeException` thrown at runtime, because only one handler chain is allowed for a single Service Implementation Bean.
5. None of the above.

9.1.2. Use deployment descriptors.

Question 09010B01

Which approach can be used to configure JAX-WS handlers chain for Web Service?

Options (select 1):

1. Automatically load all handlers from the CLASSPATH by using in `web.xml`:

```
<servlet>
    <servlet-name>javax.ws.rs.core.Application</servlet-name>
    <load-on-startup>1</load-on-startup>
</servlet>
```

2. By using `Binding.setHandlerChain(handlerList)` method.
3. By using `Dispatch.setHandlerResolver(myHandlerResolver)` method.
4. By using `@SOAPMessageHandlers` annotation on service implementation bean.
5. None of the above.

Question 09010B02

Snorcle, Inc. asked you to add protocol handler which performs incoming SOAP 1.1 messages audit to the existing Web Services implementations. Which XML handler chain fragment can be used for this?

Options (select 1):

1.
```
<handler-chain>
    <protocol-bindings>##SOAP11_HTTP</protocol-bindings>
    <handler>
        <handler-name>AuditHandler</handler-name>
        <handler-class>server.AuditHandler</handler-class>
    </handler>
</handler-chain>
```

2.
```
<handler-chain>
    <protocol-pattern>##SOAP11_HTTP</protocol-pattern>
    <handler>
        <handler-name>AuditHandler</handler-name>
```

```
        <handler-class>server.AuditHandler</handler-class>
    </handler>
</handler-chain>
```

3.

```
<handler-chain>
    <protocol-patterns>##SOAP*</protocol-patterns>
    <handler>
        <handler-name>AuditHandler</handler-name>
        <handler-class>server.AuditHandler</handler-class>
    </handler>
</handler-chain>
```

4. None of the above.

Question 09010B03

You are asked to customize WSDL of the existing Web Service. The customization should allow logging handler chain to process only those SOAP messages, which have `TransactionID` header element. How this can be achieved?

Options (select 1):

1.

```
<bindings node="wsdl:definitions">
    <javaee:handler-chains xmlns:javaee="http://java.sun.com/xml/ns/javaee">
        <javaee:handler-chain>
            <javaee:header-name-pattern xmlns:ns1="http://java.boot.by/">
                ns1:TransactionID
            </javaee:header-name-pattern>
            <javaee:handler>
                <javaee:handler-class>
                    by.boot.java.SecuritySOAPHandler
                </javaee:handler-class>
            </javaee:handler>
        </javaee:handler-chain>
    </javaee:handler-chains>
</bindings>
```

2.

```
<bindings node="wsdl:definitions">
    <javaee:handler-chains xmlns:javaee="http://java.sun.com/xml/ns/javaee">
        <javaee:handler-chain>
            <javaee:header-name xmlns:ns1="http://java.boot.by/">
                ns1:TransactionID
            </javaee:header-name>
            <javaee:handler>
                <javaee:handler-class>
                    by.boot.java.SecuritySOAPHandler
                </javaee:handler-class>
            </javaee:handler>
        </javaee:handler-chain>
    </javaee:handler-chains>
</bindings>
```

3.

```
<bindings node="wsdl:definitions">
    <javaee:handler-chains xmlns:javaee="http://java.sun.com/xml/ns/javaee">
        <javaee:handler-chain>
            <javaee:header-name-pattern xmlns:ns1="http://java.boot.by/">
```

```
                ns1:TransactionID*
            </javaee:header-name-pattern>
            <javaee:handler>
                <javaee:handler-class>
                    by.boot.java.SecuritySOAPHandler
                </javaee:handler-class>
            </javaee:handler>
        </javaee:handler-chain>
    </javaee:handler-chains>
</bindings>
```

4. None of the above.

9.2. Configure SOAP and logical handlers on the client side.

9.2.1. Use deployment descriptors.

blah-blah

9.2.2. Use programmatic API.

Question 09020B01

Which approach can NOT be used to configure handler chains for Web Service clients?

Options (select 1):

1. By using `Dispatch.setHandlerResolver(myHandlerResolver)` method.
2. By extending `PackagesResourceConfig` class and providing in constructor handler classes packages names.
3. By using WSDL customization file.
4. By using `Binding.setHandlerChain(handlerList)` method.
5. By using `@HandlerChain` annotation.

Chapter 10. Create low-level SOAP web services.

10.1. Describe the functions and capabilities of the APIs included within JAXP.

blah-blah

10.2. Describe the functions and capabilities of JAXB, including the JAXB process flow, such as XML-to-Java and Java-to-XML, and the binding and validation mechanisms provided by JAXB.

Question 100201

You have been working on Weather Forecast Web Service using "Code first" ("Start from Java") approach.

Given the following Java bean which is used as parameter:

```
// -- Java code fragment
public enum USState {AK, CO}
```

Generated WSDL XML Schema fragment looks as follows:

```
// -- XML Schema fragment
<xs:simpleType name="usState">
    <xs:restriction base="xs:string">
        <xs:enumeration value="CO" />
        <xs:enumeration value="AK" />
    </xs:restriction>
</xs:simpleType>
```

Another developer creates a .NET platform Weather Forecast Web Service client using "Contract first" ("Start from WSDL") approach and generated parameter bean source code is as follows:

```
// .NET auto generated code from XML Schema
public enum usState { CO, AK }
```

Which statement is true about these classes and XML Schema ?

Options (select 1):

1. Both classes and XML Schema mapped well and Weather Forecast Web Service will work fine with the .NET client.
2. You need to add `@XmlEnum` and `@XmlEnumValue` annotation for mapping of `enum` types.
3. You need to define `@XmlType.propOrder()` class level annotation so order of properties in Java bean and in C# bean matches.
4. None of the above.

Question 100202

You are performing development of Web Service, followed by deployment. What is the correct sequence of steps in JAXB data binding process?

Options (select 1):

1. Generate classes, generate content tree, compile, marshal, process content, unmarshal.
2. Marshal, process content, generate classes, generate content tree, compile, unmarshal.
3. Generate classes, compile, unmarshal, generate content tree, process content, marshal.
4. Unarshal, process content, generate content tree, generate classes, compile, marshal.

10.3. Use `Provider` API to create a web service.

10.3.1. Process the entire SOAP message, using the SAAJ APIs.

blah-blah

10.3.2. Process only the SOAP body, using JAXB.

blah-blah

10.4. Use `Dispatch` API to create a dynamic web service client.

blah-blah

Chapter 11. Use MTOM and MIME in a SOAP Web Service.

11.1. Use MTOM on the service.

Question 1101

Developer uses MTOM optimized SOAP 1.1 messages for JAX-WS Web Service. What would be HTTP Content-Type header if MTOM is enabled?

Options (select 1):

1. `multipart/related`
2. `application/xop+xml`
3. `text/xml`
4. `text/plain`

11.1.1. Use `@MTOM` annotation with a web service.

blah-blah

11.1.2. Use MTOM policy in WSDL.

Question 11010B01

You are developing a Web Service using "WSDL-to-Java" approach. The Web Service uses a message which contains a binary field. The binary field is intended to carry a large image file, so it is not appropriate to send it as part of a normal SOAP message, but as optimized MTOM attachment. You have added the `xmime:expectedContentTypes` attribute to the element containing the binary data in WSDL:

```
...
<types>
    <schema targetNamespace="http://java.boot.by/types/"
            xmlns="http://www.w3.org/2001/XMLSchema"
            xmlns:xsd1="http://java.boot.by/types/"
            xmlns:xmime="http://www.w3.org/2005/05/xmlmime">
        <complexType name="ReportType">
            <sequence>
                <element name="imageData" type="xsd:base64Binary" xmime:expectedContentTypes="imag
            </sequence>
        </complexType>
        <element name="reportData" type="xsd1:ReportType" />
    </schema>
</types>
...
```

To which JAXB class the `reportData` element will be mapped?

Options (select 1):

1.
```
@XmlType
public class ReportType {
    @XmlMimeType("image/gif")
    protected javax.activation.DataHandler imageData;
    ...
}
```

2.
```
@XmlType
public class ReportType {
    @XmlMimeType("image/gif")
    protected javax.xml.transform.Source imageData;
    ...
}
```

3.
```
@XmlType
public class ReportType {
    @XmlMimeType("image/gif")
    protected byte[] imageData;
    ...
}
```

4.
```
@XmlType
public class ReportType {
    @XmlMimeType("image/gif")
    protected java.awt.Image imageData;
    ...
}
```

11.1.3. Use MTOM in the deployment descriptors

blah-blah

11.1.4. Use `MTOMFeature` with `javax.xml.ws.Endpoint` API

blah-blah

11.1.5. Use `swaRef` in WSDL.

Question 11010E01

Your team is working on a Web Service using "Java-to-WSDL" approach. The Web Service is required to send binary attachments. Also, you want to make sure that Web Service is WS-I Attachment Profile 1.0 compliant and decide to use mechanism of `ref:swaRef` in generated WSDL to reference MIME attachment parts. Which two independent approaches could you use to accomplish this requirement?

Options (select 2 best options):

1. Annotate payload bean with `@SwaRef`.
2. Annotate Web Service interface method with `@SwaRef`.
3. Annotate payload bean with `@XmlAttachmentRef`.
4. Annotate Web Service interface method with `@XmlAttachmentRef`.
5. Annotate Web Service interface method with `@MTOM`.
6. Annotate payload bean with `@XmlID`.

11.1.6. Use MIME binding in WSDL.

blah-blah

11.2. Use MTOM on the client.

11.2.1. Use `MTOMFeature` with `getPort()` methods.

Question 11020101

Given the `CatalogService` dynamic proxy factory and the `Catalog` Web Service proxy, how can developer enable MTOM on the client side?

Options (select 1):

1.
```
CatalogService service = new CatalogService(new MTOMFeature());
Catalog catalogPort = service.getCatalogPort();
```

2.
```
CatalogService service = new CatalogService();
Catalog catalogPort = service.getCatalogPort(new MTOMFeature());
```

3.
```
CatalogService service = new CatalogService();
service.setFeature(new MTOMFeature());
Catalog catalogPort = service.getCatalogPort();
```

4.
```
CatalogService service = new CatalogService();
Catalog catalogPort = service.getCatalogPort();
catalogPort.setFeature(new MTOMFeature());
```

11.2.2. Use MTOM in the deployment descriptors.

blah-blah

11.2.3. Sending any additional attachments using `MessageContext` properties.

blah-blah

Chapter 12. Use WS-Addressing with a SOAP web service

12.1. Use Addressing on the service

12.1.1. Use `@Addressing` annotation with a web service

blah-blah

12.1.2. Use `wsam:Addressing` policy in WSDL

blah-blah

12.1.3. Use Addressing in the deployment descriptors

blah-blah

12.1.4. Use `AddressingFeature` with `javax.xml.ws.Endpoint` API.

blah-blah

12.1.5. Use `@Action` and `@FaultAction` on the service methods.

blah-blah

12.1.6. Use `WebServiceContext.getEndpointReference()`

blah-blah

12.2. Use Addressing on the client.

12.2.1. Use `AddressingFeature` with `getPort()` methods.

blah-blah

12.2.2. Use Addressing in the deployment descriptors

blah-blah

12.2.3. Use `BindingProvider.getEndpointReference()`

blah-blah

12.2.4. Use `getPort(EndpointReference)` methods.

blah-blah

Chapter 13. Configure Message Level security for a SOAP web service

Configure SOAP message-layer security requirements of service as policy assertions in service description; this should be supported by an IDE such as is the case when Netbeans is used to select a Metro Security Profile.

13.1. Select the appropriate Security Profile for your service. The selection would be based on a match of the Protection guarantees offered by the profile and those required by the service.

Question 130101

Given the `BusinessCardService.wsdl` fragment, which is true about SOAP message security?

```
<wsdl:definitions name="BusinessCardService" ...>
    ...
    <wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
        ....
        <sp:SignedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
            <sp:Body />
            <sp:Header Name="To" Namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"/>
        </sp:SignedParts>
        ...
    </wsp:Policy>
    ...
</wsdl:definitions>
```

Options (select 1):

1. Both SOAP message `Body` element and header `To` must be encoded.
2. Only SOAP message `Body` element must be encoded.
3. SOAP message `Body` element and header `To` integrity must be protected.
4. Only SOAP message `Body` element integrity must be protected.
5. Both SOAP message `Body` element and header `To` must be encrypted.
6. Only SOAP message `Body` element must be encrypted.

Question 130102

Given the `StockQuote.wsdl` fragment, which is true about SOAP message security?

```
<definitions name="StockQuote" ...>
    ...
    <wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" wsu:Id="StockQuotePolicy">
        ....
        <sp:SignedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy" />
        ...
    </wsp:Policy>
    ...
    <binding name="StockQuoteSoap" type="tns:StockQuotePortType">
        <soap:binding style="document" transport="http://example.com/smtp"/>
        <wsp:PolicyReference URI="#StockQuotePolicy"/>
        <operation name="SubscribeToQuotes">
           <input message="tns:SubscribeToQuotes">
               <soap:body parts="body" use="literal"/>
               <soap:header message="tns:SubscribeToQuotes" part="subscribeheader" use="literal"/>
           </input>
        </operation>
    </binding>
    ...
</definitions>
```

Options (select 1):

1. Both SOAP message `Body` and `Header` elements must be encoded.
2. SOAP message `Body` and `Header` elements' integrity must be protected.
3. Both SOAP message `Body` and `Header` elements must be encrypted.
4. None of the above.

Question 130103

Given the `CalculatorWSService.wsdl` fragment, which is true about SOAP message security?

```
<wsdl:definitions name="CalculatorWSService" ...>
    ...
    <wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
        ....
        <sp:EncryptedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
            <sp:Body />
            <sp:Header Name="To" Namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"/>
        </sp:EncryptedParts>
        ...
    </wsp:Policy>
    ...
</wsdl:definitions>
```

Options (select 1):

1. Both SOAP message `Body` element and header `To` must be encoded.
2. SOAP message `Body` element and header `To` integrity must be protected.
3. Both SOAP message `Body` element and header `To` require confidentiality.
4. Only SOAP message `Body` element integrity must be protected.
5. Only SOAP message `Body` element must be encoded.
6. Only SOAP message `Body` element requires confidentiality.

Question 130104

Given the `StockQuoteWS.wsdl` fragment, which is true about SOAP message security?

```
<definitions name="StockQuoteWS" ...>
    ...
    <wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" wsu:Id="StockQuoteWSPolic
        ....
        <sp:EncryptedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy" />
        ...
    </wsp:Policy>
    ...

    <binding name="StockQuoteSoap" type="tns:StockQuotePortType">
        <soap:binding style="document" transport="http://example.com/smtp"/>
        <wsp:PolicyReference URI="#StockQuoteWSPolicy"/>
        <operation name="SubscribeToQuotes">
           <input message="tns:SubscribeToQuotes">
               <soap:body parts="body" use="literal"/>
               <soap:header message="tns:SubscribeToQuotes" part="subscribeheader" use="literal"/>
           </input>
        </operation>
    </binding>
    ...
</definitions>
```

Options (select 1):

1. Both SOAP message `Body` and `Header` elements must be encoded.
2. SOAP message `Body` and `Header` elements' integrity must be protected.
3. Both SOAP message `Body` and `Header` elements must be encrypted.
4. None of the above.

13.2. Configure Username/Password callbacks required by the Username Token Profile.

blah-blah

13.3. Configure any server side Validators as maybe applicable for the profile. There are defaults in GlassFish for most of them.

blah-blah

13.4. Optimize interaction between client and server by using WS-SecureConversation.

blah-blah

Chapter 14. Apply best practices to design and implement Web Services.

Question 1401

What is true about Service Implementation Layers?

Options (select 3):

1. The Service Interaction Layer consists of the endpoint interface that the service exposes to clients and through which it receives client requests.
2. Sometimes multiple layers make a service unnecessarily complicated and, in these cases, it

may be simpler to design the service as one layer.

3. The Service Processing Layer consists of the endpoint interface that the service exposes to clients and through which it receives client requests.
4. The Service Processing Layer holds all business logic used to process client requests.
5. Web Service Implementation is always separated in an Interaction Layer and a Processing Layer.
6. The Service Interaction Layer holds all business logic used to process client requests.

Question 1402

What Layer of Web Service Implementation is the best place to validate incoming XML document?

Options (select 1):

1. Service Processing Layer
2. Service Interaction Layer
3. Service Parsing Layer
4. Service Validation Layer

14.1. Use different encoding schemes - fast infoset.

blah

14.2. Use GZIP for optimiziing message sizes.

blah

14.3. Use catalog mechanism for WSDL access.

blah

14.4. Refer to WS-I sample app for best practices.

Question 140401

What statements are true about Web Services and SOA?

Options (select 2):

1. Both Web Servives and SOA can use UDDI.
2. Web Servives decrease CPU utilization on the server.
3. SOA decreases network bandwith requirements.
4. Web Services expose their interfaces as fine-grained components.
5. SOA is an architectural style and Web Services is a way to realize SOA.

Question 140402

What statements are true about WS-I BP 1.1?

Options (select 2):

1. All Java EE 6 Application Servers must support WS-I BP 1.1
2. Java EE 6 Application Servers are not required to support WS-I BP 1.1

3. All Java EE 6 applications must support WS-I BP 1.1
4. Java EE 6 applications are not required to support WS-I BP 1.1

Question 140403

What statements are correct about XML documents description standards for Web Services?

Options (select 2):

1. DTD language is the preffered way for Web Services description - according to WS-I BP 1.1
2. XML Schema and DTD are the preffered ways for Web Services description - according to WS-I BP 1.1
3. XML Schema is the preffered way for Web Services description - according to WS-I BP 1.1
4. World Wide Web Consortium (W3C) working group suggests XSD for Web Services.
5. World Wide Web Consortium (W3C) working group suggests SCH for Web Services.

Part II. Appendices

Appendix 1. XML Web Service Standards

1.1. Given XML documents, schemas, and fragments determine whether their syntax and form are correct (according to W3C schema) and whether they conform to the WS-I Basic Profile 1.1.

Question A010101

Snorcle, Inc. is creating an XML schema that describes various Java training materials available for purchase by students. Which XSD fragments correctly describe training material author's name, consisting of a first name and a last name?

Options (select 2):

1.
```
<xs:element name="author">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="firstName" type="xs:string" minOccurs="1"/>
            <xs:element name="lastName" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

2.
```
<xs:element name="author">
    <xs:complexType>
        <xs:choice>
            <xs:element name="firstName" type="xs:string"/>
            <xs:element name="lastName" type="xs:string" minOccurs="1"/>
        </xs:choice>
    </xs:complexType>
</xs:element>
```

3.
```
<xs:element name="author">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="firstName" type="xs:string" maxOccurs="1"/>
```

```
            <xs:element name="lastName" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

4.
```
<xs:element name="author">
    <xs:complexType>
        <xs:choice>
            <xs:element name="firstName" type="xs:string"/>
            <xs:element name="lastName" type="xs:string"/>
        </xs:choice>
    </xs:complexType>
</xs:element>
```

5.
```
<xs:element name="author">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="firstName" type="xs:string" minOccurs="0"/>
            <xs:element name="lastName" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

6.
```
<xs:element name="author">
    <xs:complexType>
        <xs:all>
            <xs:element name="firstName" type="xs:string"/>
            <xs:element name="lastName" type="xs:string" maxOccurs="5"/>
        </xs:all>
    </xs:complexType>
</xs:element>
```

Question A010102

Consider the following SOAP message fragment:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
        soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <soap:Body xmlns:ns1="http://example">
                ...
    </soap:Body>
</soap:Envelope>
```

Is the message WS-I BP 1.1 conformant?

Options (select 1):

1. Yes, it is comformant for all styles of SOAP messages.
2. Yes, it is comformant for "rpc-encoded" binding only.
3. Yes, it is comformant for "document-encoded" binding only.
4. Yes, it is comformant for "rpc-encoded" and "document-encoded" bindings only.
5. No, it is not comformant.

Question A010103

Which of the following WSDL Schema fragments contains a WS-I Basic Profile 1.1-conformant array declaration?

Options (select 1):

1.

```
<element name="myFavoriteStrings" type="ArrayOfStrings"/>

<complexType name="ArrayOfStrings">
    <complexContent>
        <restriction base="SOAP-ENC:Array">
            <sequence>
                <element name="x" type="string" minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
            <attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="string[]"/>
        </restriction>
    </complexContent>
</complexType>
```

2.

```
<element name="myFavoriteStrings" type="MyArrayType"/>

<xsd:complexType name="MyArrayType">
    <xsd:sequence>
        <xsd:element name="x" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="string[]"/>
</xsd:complexType>
```

3.

```
<element name="myFavoriteStrings" type="MyArrayType"/>

<xsd:complexType name="MyArrayType">
    <xsd:sequence>
        <xsd:element name="x" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
```

4.

```
<element name="myFavoriteStrings" type="ArrayOfStrings"/>

<complexType name="ArrayOfStrings">
    <complexContent>
        <restriction base="SOAP-ENC:Array">
            <sequence>
                <element name="x" type="string" minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
        </restriction>
    </complexContent>
</complexType>
```

Question A010104

Given the XML Schema fragment:

```
<xsd:element name="Customer">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="CustomerID" type="xsd:string"/>
            <xsd:element name="CompanyName" type="xsd:string"/>
            <xsd:element name="ContactName" type="xsd:string" nillable="true"/>
            <xsd:element name="Address" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
```

```
</xsd:element>
```

Which XML snippets are valid?

Options (select 3):

1.
```
<Customer>
    <CustomerID>123456</CustomerID>
    <CompanyName>IBM</CompanyName>
    <ContactName xsi:nil="1"></ContactName>
    <Address>5600 63rd Street</Address>
</Customer>
```

2.
```
<Customer>
    <CustomerID>123456</CustomerID>
    <CompanyName>IBM</CompanyName>
    <ContactName>Mikalai Zaikin</ContactName>
    <Address>5600 63rd Street</Address>
</Customer>
```

3.
```
<Customer>
    <CustomerID>123456</CustomerID>
    <CompanyName>IBM</CompanyName>
    <ContactName xsi:nil="true">Mikalai Zaikin</ContactName>
    <Address>5600 63rd Street</Address>
</Customer>
```

4.
```
<Customer>
    <CustomerID>123456</CustomerID>
    <CompanyName xsi:nil="true" />
    <ContactName xsi:nil="true" />
    <Address>5600 63rd Street</Address>
</Customer>
```

5.
```
<Customer>
    <CustomerID>123456</CustomerID>
    <CompanyName>IBM</CompanyName>
    <ContactName xsi:nil="true" />
    <Address>5600 63rd Street</Address>
</Customer>
```

Question A010105

Given the `sample.xsd` XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    targetNamespace="http://www.example.com">
```

```
<xsd:element name="Customer">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="CustomerID" type="xsd:string"/>
            <xsd:element name="CompanyName" type="xsd:string"/>
            <xsd:element name="ContactName" type="xsd:string"/>
            <xsd:element name="Address" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Which XML instance is valid?

Options (select 1):

1.
```
<?xml version="1.0" encoding="UTF-8"?>
<Customer xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.example.com sample.xsd ">
  <CustomerID>123456</CustomerID>
  <CompanyName>IBM</CompanyName>
  <ContactName>Mikalai Zaikin</ContactName>
  <Address>5600 63rd Street</Address>
</Customer>
```

2.
```
<?xml version="1.0" encoding="UTF-8"?>
<ns:Customer xmlns:ns="http://www.example.com"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.example.com sample.xsd ">
  <CustomerID>123456</CustomerID>
  <CompanyName>IBM</CompanyName>
  <ContactName>Mikalai Zaikin</ContactName>
  <Address>5600 63rd Street</Address>
</ns:Customer>
```

3.
```
<?xml version="1.0" encoding="UTF-8"?>
<Customer xmlns:ns="http://www.example.com"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.example.com sample.xsd ">
  <ns:CustomerID>123456</ns:CustomerID>
  <ns:CompanyName>IBM</ns:CompanyName>
  <ns:ContactName>Mikalai Zaikin</ns:ContactName>
  <ns:Address>5600 63rd Street</ns:Address>
</Customer>
```

4.
```
<?xml version="1.0" encoding="UTF-8"?>
<ns:Customer xmlns:ns="http://www.example.com"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.example.com sample.xsd ">
  <ns:CustomerID>123456</ns:CustomerID>
  <ns:CompanyName>IBM</ns:CompanyName>
  <ns:ContactName>Mikalai Zaikin</ns:ContactName>
  <ns:Address>5600 63rd Street</ns:Address>
</ns:Customer>
```

Question A010106

Given the following document-literal SOAP message:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:ns1="http://e-shop.com">
    <soapenv:Body>
        <ns1:name>Sony Reader Digital Book PRS-505</ns1:name>
        <ns1:color>Silver</ns1:color>
        <ns1:price >260.99</ns1:price>
    </soapenv:Body>
</soapenv:Envelope>
```

Which is true?

Options (select 1):

1. Yes, it is WS-I BP 1.1 compliant.
2. No, it is not WS-I BP 1.1 compliant, because document-literal enconding is restricted.
3. No, it is not WS-I BP 1.1 compliant, because message may not have more than one child elements of the SOAP `Body`.
4. No, it is not WS-I BP 1.1 compliant, because children of the SOAP `Body` may not be namespace qualified.

Question A010107

Given the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:ns1="http://e-shop.com"
        xmlns:wsr="http://routing.com">
    <soapenv:Header>
            <wsr:party id="uuid:791F4B43453573FB7D1204876983384">
    </soapenv:Header>
    <soapenv:Body>
        <ns1:order>
            <ns1:name>Sony Reader Digital Book PRS-505</ns1:name>
            <ns1:color>Silver</ns1:color>
            <ns1:price>260.99</ns1:price>
        </ns1:order>
    </soapenv:Body>
</soapenv:Envelope>
```

Which describes this SOAP message?

Options (select 1):

1. The message is WS-I BP 1.1 compliant.
2. The `ns1` namespace is not correctly declared.
3. The `wsr` namespace is not correctly declared.
4. None of the above.

Question A010108

Which SOAP message is WS-I BP 1.1 conformant?

Options (select 1):

1.
```
<soap:Envelope xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'>
    <soap:Body>
        <p:Process xmlns:p='http://example.org/operations' />
    </soap:Body>
    <m:Data xmlns:m='http://example.org/information' >
        <wsi:Claim conformsTo="http://ws-i.org/profiles/basic1.1/"
            xmlns:wsi="http://ws-i.org/schemas/conformanceClaim/" />
    </m:Data>
</soap:Envelope>
```

2.
```
<soap:Envelope xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'>
    <m:Data xmlns:m='http://example.org/information' >
        <wsi:Claim conformsTo="http://ws-i.org/profiles/basic1.1/"
            xmlns:wsi="http://ws-i.org/schemas/conformanceClaim/" />
    </m:Data>
    <soap:Body>
        <p:Process xmlns:p='http://example.org/operations' />
    </soap:Body>
</soap:Envelope>
```

3.
```
<soap:Envelope xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'>
    <soap:Header>
        <wsi:Claim conformsTo="http://ws-i.org/profiles/basic1.1/"
            xmlns:wsi="http://ws-i.org/schemas/conformanceClaim/" />
    </soap:Header>
    <m:Data xmlns:m='http://example.org/information' >
        <t:Trans xmlns:t="http://http://example.org/transaction/">
            15091974
        </t:Trans>
    </m:Data>
</soap:Envelope>
```

4.
```
<soap:Envelope xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'>
    <m:Data xmlns:m='http://example.org/information' >
        <t:Trans xmlns:t="http://http://example.org/transaction/">
            15091974
        </t:Trans>
    </m:Data>
    <soap:Header>
        <wsi:Claim conformsTo="http://ws-i.org/profiles/basic1.1/"
            xmlns:wsi="http://ws-i.org/schemas/conformanceClaim/" />
    </soap:Header>
</soap:Envelope>
```

5. None of the above.

1.2. Describe the use of XML schema in Java EE Web services.

Question A010201

Which statement is true about WSDL definition?

Options (select 1):

1. XML Schema is used as a formal definition of WSDL grammar.
2. XML Document Type Definition is used as a formal definition of WSDL grammar.
3. XML Stylesheet Language Transformation is used as a formal definition of WSDL grammar.
4. Document Object Model is used as a formal definition of WSDL grammar.
5. None of the above.

Appendix 2. SOAP 1.2 Web Service Standards

2.1. List and describe the encoding types used in a SOAP message.

Warning

The OCE WSD 6 exam questions are still based on the SOAP 1.1 specification.

Question A020101

Consider the following Java code and SOAP 1.1 message:

```
Name name = new Name("Mikalai", "Zaikin");
Author author = new Author(name);
Book book1 = new Book("SCDJWS 5.0 Guide", author);
Book book2 = new Book("OCE WSD 6 Guide", author);
// get Web Service endpoint interface [SEI]
...
boolean b = sei.compare(book1, book2);
```

```
<soap:Envelope xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'
       soap:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'>

    <soap:Body xmlns:ns1='http://example'>

        <ns1:compare>
            <e:book>
                <title>SCDJWS 5.0 Guide</title>
                <!-- 1 -->
            </e:book>
            <e:book>
                <title>OCE WSD 6 Guide</title>
                <!-- 2 -->
            </e:book>
        </ns1:compare>

        <!-- 3 -->
            <name>
                <fistName>Mikalai</firstName>
                <lastName>Zaikin</lastName>
            </name>
        </ns1:author>

    </soap:Body>

</soap:Envelope>
```

How can you refer to the same `author` element in several places ?

Options (select 1):

1.
```
1. <author href="#mz" />
2. <author href="#mz" />
3. <ns1:author id="mz" >
```

2.
```
1. <author href="#mz" />
2. <author href="#mz" />
3. <ns1:author cid="mz" >
```

3.
```
1. <author href="mz" />
2. <author href="mz" />
3. <ns1:author id="mz" >
```

4.
```
1. <author ref="mz" />
2. <author ref="mz" />
3. <ns1:author id="mz" >
```

Question A020102

Determine in the following XML Schema fragments which "`code`" element/attribute type maps to Java wrapper class or Java primitive:

1.
```
<xsd:element name="code" type="xsd:short" />
```

2.
```
<xsd:element name="code" type="xsd:short" maxOccurs="1" />
```

3.
```
<xsd:element name="code" type="xsd:short" minOccurs="0" />
```

4.
```
<xsd:element name="code" type="xsd:short" minOccurs="0" maxOccurs="1" />
```

5.
```
<xsd:element name="code" type="xsd:short" nillable="true" />
```

6.
```
<xsd:element name="description">
        <xsd:complexType>
```

```
                <xsd:sequence />
                <xsd:attribute name="code" type="xsd:short" use="optional"/>
        </xsd:complexType>
</xsd:element>
```

7.

```
<xsd:element name="description">
        <xsd:complexType>
                <xsd:sequence />
                <xsd:attribute name="code" type="xsd:short" default="0"/>
        </xsd:complexType>
</xsd:element>
```

Options:

a) `java.lang.Short`

b) `short`

Fill your answer here:

```
1 - [  ]; 2 - [  ]; 3 - [  ]; 4 - [  ]; 5 - [  ]; 6 - [  ]; 7 - [  ]
```

Question A020103

Compose the correct SOAP message from following elements.

Options:

1.

```
<soap:Headers>
```

2.

```
</soap:Header>
```

3.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
```

4.

```
</soap:Envelope>
```

5.

```
<soap:Header>
```

6.

```
</soap:Headers>
```

7.
```
<soap:Body>
```

8.
```
</soap:Body>
```

9.
```
<soap:Message xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
```

10.
```
</soap:Message>
```

11.
```
<soap:Content>
```

12.
```
</soap:Content>
```

Fill your answer here:

```
[       ]
    [       ]
        ...
    [       ]
    [       ]
        ...
    [       ]
[       ]
```

2.2. Describe the SOAP Processing and Extensibility Model.

Question A020201

When calling Web Service you received following SOAP message.

```
<soap:Fault xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/' >
    <faultcode>soap:Server</faultcode>
    <faultstring>Database is not accessible</faultstring>
</soap:Fault>
```

How can you get "`faultcode`" element value?

Options (select 2):

1.
```
SOAPBody body = message.getSOAPBody();
if (body.hasFault()) {
    SOAPFault fault = body.getFault();
    javax.xml.soap.Name code = fault.getFaultCodeAsName();
```

```
    ...
}
```

2.
```
SOAPBody body = message.getSOAPBody();
if (body.hasFault()) {
    SOAPFault fault = body.getFault();
    String code = fault.getElementByTagName("faultcode");
    ...
}
```

3.
```
SOAPBody body = message.getSOAPBody();
if (body.hasFault()) {
    SOAPFault fault = body.getFault();
    org.w3c.dom.Node faultCode = fault.getElementByTagName("faultcode");
    ...
}
```

4.
```
SOAPBody body = message.getSOAPBody();
if (body.hasFault()) {
    SOAPFault fault = body.getFault();
    org.w3c.dom.NodeList faultCodes = fault.getElementsByTagName("faultcode");
    ...
}
```

Question A020202

What are optional child elements of SOAP Fault?

Options (select 2):

1. `faultcode`
2. `faultstring`
3. `faultactor`
4. `detail`

Question A020203

According to WS-I Basic Profile 1.1, what are valid child elements of a `soap:Fault` element?

Options (select 3):

1. `detail`
2. `actor`
3. `faultcode`
4. `faultdetail`
5. `faultstring`

Question A020204

According to WS-I Basic Profile 1.1, which `soap:Fault` element is valid?

Options (select 1):

1.
```
<soap:Fault xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" >
```

```
  <soap:faultcode>soap:Client</soap:faultcode>
  <soap:faultstring>...</soap:faultstring>
  <soap:faultactor>...</soap:faultactor>
  <soap:detail>...</soap:detail>
</soap:Fault>
```

2.

```
<soap:Fault xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" >
  <faultcode>soap:Client</faultcode>
  <faultstring>...</faultstring>
  <faultactor>...</faultactor>
  <soap:detail>...</soap:detail>
</soap:Fault>
```

3.

```
<soap:Fault xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns="">
  <faultcode>Client</faultcode>
  <faultstring>...</faultstring>
  <faultactor>...</faultactor>
  <detail>...</detail>
</soap:Fault>
```

4.

```
<soap:Fault xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns="">
  <faultcode>soap:Client</faultcode>
  <faultstring>...</faultstring>
  <faultactor>...</faultactor>
  <detail>...</detail>
</soap:Fault>
```

Question A020205

What the features does SOAP have?

Options (select 4):

1. Support for SOAP XML messages incorporating attachments (using the multipart MIME structure).
2. It can use only HTTP or HTTPS as a transport protocol.
3. Is a platform and operating system independent.
4. Can use any transport protocol (HTTP, HTTPS, FTP, SMTP, JMS, etc.).
5. SOAP message can be constructed only using Java programming language.
6. Can be used only on Windows and Linux operating systems.
7. Can not be used for sending attachments (images, documents, etc.).
8. Programming language independent.

2.3. Describe SOAP Message Construct and create a SOAP message that contains an attachment.

Question A020301

You are sending a scanned image of business contract as SOAP 1.1 attachment:

```
--MIME_boundary
Content-Type: image/tiff
Content-Transfer-Encoding: binary
Content-ID: <business_contract>

d3d3Lm1hcmNoYWwuY29taesgfSEVFES45345sdvgfszd==
--MIME_boundary--
```

How can you refer to MIME attachment from SOAP body?

Options (select 1):

1.
```
<SOAP-ENV:Body>
    ..
    <contract href="cid:business_contract"/>
    ..
</SOAP-ENV:Body>
```

2.
```
<SOAP-ENV:Body>
    ..
    <contract href="business_contract"/>
    ..
</SOAP-ENV:Body>
```

3.
```
<SOAP-ENV:Body>
    ..
    <contract href="#business_contract"/>
    ..
</SOAP-ENV:Body>
```

4.
```
<SOAP-ENV:Body>
    ..
    <contract href="#cid:business_contract"/>
    ..
</SOAP-ENV:Body>
```

Question A020302

You need to send a scanned image of business contract as SOAP 1.1 attachment:

```
MessageFactory factory = MessageFactory.newInstance();
SOAPMessage message = factory.createMessage();

// add code here

attachment.setContentId("attached_contract");
message.addAttachmentPart(attachment);
```

How can you add to SOAP message an attachment with the image using SAAJ API?

Options (select 1):

1.
```
HttpDataSource dataSource = new HttpDataSource("http://localhost/contract.jpeg");
DataHandler dataHandler = new DataHandler(dataSource);
AttachmentPart attachment =  message.createAttachmentPart(dataHandler);
```

2.
```
FileDataSource dataSource = new FileDataSource("contract.jpeg");
DataHandler dataHandler = new DataHandler(dataSource);
AttachmentPart attachment =  message.createAttachmentPart(dataHandler);
```

3.
```
ImageDataSource dataSource = new ImageDataSource("contract.jpeg");
DataHandler dataHandler = new DataHandler(dataSource);
AttachmentPart attachment =  message.createAttachmentPart(dataHandler);
```

4.
```
AttachmentPart attachment = message.createAttachmentPart();
attachment.setContent("contract.jpeg", "image/jpeg");
```

Question A020303

Which statements about SOAP 1.1 message are true?

Options (select 3):

1. SOAP Header element is optional.
2. SOAP Header element is mandatory.
3. SOAP Body element is optional.
4. SOAP Body element is mandatory.
5. SOAP Envelope element is optional.
6. SOAP Envelope element is mandatory.

Appendix 3. Describing and Publishing (WSDL and UDDI)

3.1. Explain the use of WSDL in Web Services, including a description of WSDL's basic elements, binding mechanisms and the basic WSDL operation types as limited by the WS-I Basic Profile 1.1.

Question A030101

Snorcle, Inc. is developing a new Web Service using "WSDL-to-Java" approach. Development team is given the WSDL for the Weather Forecast Web Service. What is true about the following WSDL fragment?

```
...
<service name="WeatherService">
    <port name="WeatherServicePort" binding="tns:WeatherServicePortBinding">
        <soap:address location="http://java.boot.by/WeatherService"/>
    </port>
    <port name="WeatherWSPort" binding="tns:WeatherWSPortBinding">
        <soap:address location="http://java.boot.by/WeatherService"/>
    </port>
</service>
...
```

Options (select 1):

1. This fragment shows WS-I BP 1.1 compliant Web Service, because both `port`s have different `name` attributes.
2. This fragment shows NOT WS-I BP 1.1 compliant Web Service, because a single `service` has two `port` elements.
3. This fragment shows NOT WS-I BP 1.1 compliant Web Service, because both `port` elements point to the same location.
4. This fragment shows WS-I BP 1.1 compliant Web Service, because both `port`s have different `binding` attributes.
5. None of the above.

Question A030102

What is the purpose of `.wsdl` file?

Options (select 1):

1. It contains information that correlates the mapping between the Java interfaces and WSDL definition.
2. It describes a collection of servlets.
3. It identifies the services provided by the server and the set of operations or functionalities within each service that the server supports.
4. It is used to override existing JAX-WS annotations.

Question A030103

What SOAP bindings Basic Profile 1.1 allows?

Options (select 2):

1. rpc-literal
2. rpc-encoded
3. document-literal
4. document-encoded
5. rpc-wrapped

Question A030104

Which statement is true about WSDL namespace for WSDL SOAP binding?

Options (select 1):

1. Its URI is `http://schemas.xmlsoap.org/wsdl/`
2. Its URI is `http://schemas.xmlsoap.org/wsdl/http/`
3. Its URI is `http://schemas.xmlsoap.org/soap/encoding/`
4. Its URI is `http://schemas.xmlsoap.org/soap/envelope/`
5. None of the above.

3.2. Describe how WSDL enables one to separate the description of the abstract functionality offered by a service from concrete details of a service description

such as "how" and "where" that functionality is offered.

Question A030201

What statements are correct about WSDL 1.1 elements?

Options (select 3):

1. `types` – an abstract definition of the data being communicated.
2. `operation` – an abstract description of an action supported by the service.
3. `operation` – an abstract set of `portType`s supported by one or more endpoints.
4. `binding` - a single endpoint defined as a combination of a `portType` and a network address.
5. `types` – a container for data type definitions using some type system (such as XSD).
6. `binding` – a concrete protocol and data format specification for a particular port type.

Question A030202

What statements are true about elements of WSDL 1.1 document?

Options (select 2):

1. `portType` – a single endpoint defined as a combination of a binding and a network address.
2. `port` – a single endpoint defined as a combination of a binding and a network address.
3. `message` – an abstract set of types supported by one or more endpoints.
4. `message` - an abstract, typed definition of the data being communicated.

3.3. Describe the Component Model of WSDL including Descriptions, Interfaces, Bindings, Services and Endpoints.

blah-blah

3.4. Describe the basic functions provided by the UDDI Publish and Inquiry APIs to interact with a UDDI business registry.

Question A030401

User sends the following SOAP message to UDDI registry:

```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <delete_tModel generic="2.0" xmlns="urn:uddi-org:api_v2">
      <authInfo>uddiUser_AuthToken</authInfo>
      <tModelKey>uuid:6e090afa-33e5-36eb-81b7-1ca18373f457</tModelKey>
    </delete_tModel>
  </soap:Body>
</soap:Envelope>
```

What is true about the `tModel` upon successful completion?

Options (select 2):

1. The `tModel` is physically deleted as a result of this call.

2. The `tModel` is just hidden and is still accessible, via the `get_registeredInfo` call.
3. The `tModel` is just hidden and is still accessible, via the `find_tModel` call.
4. The `tModel` is just hidden and is still accessible, via the `get_tModelDetail` call.
5. UDDI registry is not changed. The `tModel` can not be deleted by anyone from UDDI registry, it can only be modified.

Question A030402

What is true about UDDI security?

Options (select 1):

1. Inquiry API requires HTTPS protocol, Publishing API requires HTTPS protocol
2. Inquiry API requires HTTPS protocol, Publishing API requires HTTP protocol
3. Inquiry API requires HTTP protocol, Publishing API requires HTTPS protocol
4. Inquiry API requires HTTP protocol, Publishing API requires HTTP protocol

Question A030403

Is the authentication required when you use UDDI Inquiry API?

Options (select 1):

1. Yes, authentication is always required for Inquiry API.
2. Yes, some Inquiry API functions require authentication.
3. No, Inquiry API does not require authentication.
4. None of the above.

Question A030404

Which statement about accessing UDDI registry is true?

Options (select 1):

1. Publisher API functions required to be exposed over FTP protocol.
2. Inquiry API functions required to be exposed over HTTP protocol.
3. Inquiry API functions are asynchronous.
4. Publisher API functions required to be exposed over SMTP protocol.
5. Publisher API functions are asynchronous.
6. Publisher API functions required to be exposed over HTTP protocol.

Question A030405

Which two are true about UDDI?

Options (select 2):

1. Publishing API functions are callable via POST method.
2. Publishing API functions are callable via GET method.
3. Publishing API functions are callable via PUT and DELETE method.

4. Inquiry API functions are callable via GET method.
5. Inquiry API functions are callable via HEAD method.
6. Inquiry API functions are callable via POST method.

Question A030406

Which is true about UDDI Inquiry API function?

Options (select 1):

1. `find_relatedEntity` function is used to locate related business entities and returns a `relatedEntitiesList` message.
2. `find_relatedBusinesses` function is used to locate related business entities and returns a `relatedBusinessesList` message.
3. `find_relatedBusinesses` function is used to locate related business entities and returns a `businessesList` message.
4. `find_relatedEntity` function is used to locate related business entities and returns a `entitiesList` message.

Appendix 4. JAX-WS

4.1. Explain JAX-WS technology for building web services and client that communicate using XML.

Question A040101

Web Service client program needs to add a security credential to the header of a SOAP message before sending the message. What would be the best approach?

Options (select 1):

1. Use client-side Header handler.
2. Use client-side Logical handler.
3. Use client-side SOAP handler.
4. Use client-side intermediary SEI.

4.2. Given a set of requirements for a Web Service, such as transactional needs, and security requirements, design and develop Web Service applications that use JAX-WS technology.

blah

4.3. Describe the Integrated Stack (I-Stack) which consist of JAX-WS, JAXB, StAX, SAAJ.

blah

4.4. Describe and compare JAX-WS development approaches.

Question A040401

A developer is implementing a Web Service using code-first ("Java-to-WSDL") approach. Which JAX-WS tools should be used to generate portable artifacts?

Options (select 2):

1. `xjc`
2. `apt`
3. `wsgen`
4. `wsimport`

Question A040402

You are hired by Snorcle, Inc. to develop an order-tracking Web Service which will be consumed by partner companies. Clients use different platforms (JEE, .NET). In order to reduce load of the Web Service clients will be asked to implement client-side validation of requests. Which approach should you use?

Options (select 1):

1. Code-first ("Java-to-WSDL") approach.
2. Contract-first ("WSDL-to-Java") approach.
3. Meet in the middle approach.
4. None of the above.

Question A040403

A developer is implementing a Web Service using contract-first ("WSDL-to-Java") approach. Which JAX-WS tool should be used to generate portable artifacts?

Options (select 1):

1. `xjc`
2. `apt`
3. `wsgen`
4. `wsimport`
5. None of the above.

4.5. Describe the features of JAX-WS including the usage of Java Annotations.

Question A040501

Given the SEI code:

```
package by.iba;

@WebService
@SOAPBinding(style = Style.RPC)
public interface Hello {
    @WebMethod
    String sayHello(String name);
}
```

What would be resulting contract fragment for the Web Service?

Options (select 1):

1.
```
<binding name="HelloImplPortBinding" type="tns:Hello">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="sayHello">
```

```
        <soap:operation soapAction=""></soap:operation>
        <input>
            <soap:body use="encoded" namespace="http://iba.by/"></soap:body>
        </input>
        <output>
            <soap:body use="encoded" namespace="http://iba.by/"></soap:body>
        </output>
    </operation>
</binding>
```

2.

```
<binding name="HelloImplPortBinding" type="tns:Hello">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="sayHello">
        <soap:operation soapAction=""></soap:operation>
        <input>
            <soap:body use="literal" namespace="http://iba.by/"></soap:body>
        </input>
        <output>
            <soap:body use="literal" namespace="http://iba.by/"></soap:body>
        </output>
    </operation>
</binding>
```

3.

```
<binding name="HelloImplPortBinding" type="tns:Hello">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="sayHello">
        <soap:operation soapAction=""></soap:operation>
        <input>
            <soap:body use="encoded" namespace="http://iba.by/"></soap:body>
        </input>
        <output>
            <soap:body use="encoded" namespace="http://iba.by/"></soap:body>
        </output>
    </operation>
</binding>
```

4.

```
<binding name="HelloImplPortBinding" type="tns:Hello">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="sayHello">
        <soap:operation soapAction=""></soap:operation>
        <input>
            <soap:body use="literal" namespace="http://iba.by/"></soap:body>
        </input>
        <output>
            <soap:body use="literal" namespace="http://iba.by/"></soap:body>
        </output>
    </operation>
</binding>
```

Question A040502

You are developing RESTful Web Service for the Snorcle, Inc. Order Tracking System.

Which annotation the endpoint implementation class must have?

Options (select 1):

1. `@WebServiceProvider`

2. `@WebServiceRef`
3. `@WebService`
4. `@RESTful`

Question A040503

You are developing a RESTful Web Service for the Snorcle, Inc. Order Managing System (OMS). The Web Service may return results in plain text, HTML or XML format, depending on the "`Accept:`" HTTP header sent by clients. Which code fragment demonstrates correct combination of JAX-WS annotations?

Options (select 1):

1.
```
@ServiceMode(value = Service.Mode.PAYLOAD);
@BindingType(value = SOAPBinding.SOAP11HTTP_BINDING);
```

2.
```
@ServiceMode(value = Service.Mode.PAYLOAD);
@BindingType(value = HTTPBinding.HTTP_BINDING);
```

3.
```
@ServiceMode(value = Service.Mode.MESSAGE);
@BindingType(value = HTTPBinding.HTTP_BINDING);
```

4.
```
@ServiceMode(value = Service.Mode.MESSAGE);
@BindingType(value = SOAPBinding.SOAP11HTTP_BINDING);
```

5. None of the above.

4.6. Describe the architecture of JAX-WS including the Tools SPI that define the contract between JAX-WS tools and Java EE.

blah-blah

4.7. Describe creating a Web Service using JAX-WS.

Question A040701

You are developing a wrapped-`document` Web Service using Java-to-WSDL approach. After implementing SIB you proceed to generating WSDL and required portable artifacts for the Web Service. Which statements are true?

Options (select 2):

1. You should use `wsgen` tool to generate portable artifacts.
2. You should use `wsimport` tool to generate portable artifacts.
3. You should use `wsgen` tool to generate WSDL.
4. You should use `wsimport` tool to generate WSDL.
5. You can not generate WSDL at development time as JAX-WS runtime will automatically generate a WSDL for you when you deploy your service.
6. Only unwrapped-`document` Web Service requires portable artifacts.

4.8. Describe JAX-WS Client Communications Models.

blah-blah

4.9. Given a set of requirements, design and develop a Web Service client, such as a Java EE client and a stand-alone client, using JAX-WS.

Question A040901

You are hired by Snorcle, Inc. to develop a Web Service client for the Order Managing System (OMS). The Web Service client must not block until either a response from the Web Service occurs or an exception is thrown. Which JAX-WS types can you use to make an asynchronous call against the service?

Options (select 2):

1. `SOAPHandler`
2. `AsynchronousHandler`
3. `AsyncHandler`
4. `LogicalHandler`
5. `Response`
6. `AsyncResponse`
7. `Request`
8. `AsynchronousRequest`

4.10. Given a set of requirements, create and configure a Web Service client that accesses a stateful Web Service.

blah-blah

Appendix 5. REST, JSON, SOAP and XML Processing APIs (JAXP, JAXB and SAAJ)

5.1. Describe the characteristics of REST Web Services.

Question A050101

You are developing a client to consume a RESTful Web Service. The Web Service can be accessed using the following URL:

```
http://java.boot.by/service
```

Which line generates client-support Java code?

Options (select 1):

1. `wsimport -keep -p client http://java.boot.by/service?wsdl`
2. `wsgen -keep -p client http://java.boot.by/service?wsdl`
3. `xjc -keep -p client http://java.boot.by/service?wsdl`
4. None of the above.

Question A050102

Snorcle, Inc. parners ask for an integration point in the corporate system to allow them to directly integrate their business processes into the Snorcle, Inc. environment. Since these partners use a variety of of enterprise development platforms for their infrastructure, the development team decides to develop a RESTful Web Service that exposes the necessary integration point to these partners in a platform- and message protocol-neutral way.

Which code fragment demonstrates the SIB declaration?

Options (select 1):

1.
```
public class IntegrationService implements Dispatch<Source> {
    ....
}
```

2.
```
public class IntegrationService implements Provider<Source> {
    ....
}
```

3.
```
public class IntegrationService implements IntegrationServiceSEI {
    ....
}
```

4.
```
public class IntegrationService implements Provider<SOAPMessage> {
    ....
}
```

Question A050103

Which statements are true about REST Web Services?

Options (select 2):

1. URIs act as identifying nouns which identify resources.
2. URIs act as verbs that specify operations on the resources.
3. HTTP methods act as identifying nouns which identify resources.
4. HTTP methods act as verbs that specify operations on the resources.

5.2. Describe the characteristics of JSON Web Services.

blah-blah

5.3. Compare SOAP Web Services to REST Web Services.

Question A050301

A developer is deciding what architectural style (SOAP or REST) would be an appropriate choice for a new application. Which statement is true?

Options (select 1):

1. The REST Web Services can maintain state.
2. The REST Web Services are useful for low bandwidth devices (such as PDAs and mobile phones).
3. The REST Web Services support formal contracts (such as messages, operations, bindings, and location of the Web Service).

4. The REST Web Services can address complex nonfunctional requirements (such as transactions, security).
5. The REST Web Services can handle asynchronous processing and invocation out of the box.

5.4. Compare SOAP Web Services to JSON Web Services.

blah-blah

5.5. Describe the functions and capabilities of the APIs included within JAXP.

Question A050501

You are parsing XML file which uses DTD with system ID pointing to remote server (see the XML fragment below). The network connection is very slow and it takes much time to download DTD and validate XML. How can you improve application's performance?

```
<!DOCTYPE quiz SYSTEM "http://java.boot.by/quiz.dtd">
<quiz>
    <name>OCE WSD 6 Quiz</name>
    <author>Mikalai Zaikin</author>
    ...
</quiz>
```

Options (select 1):

1. Use `ErrorHandler` interface.
2. It is not possible to improve performance in such situation without XML modification.
3. Use `EntityResolver` interface.
4. Use `ContentHandler` interface.
5. Use `XMLReader` interface.

Question A050502

Your company uses database as a repository for etalon DTDs. How can you validate XML document against DTD stored in BLOB field of database?

Options (select 1):

1. It is not possible.
2. Use `ErrorHandler` interface.
3. Use `ContentHandler` interface.
4. Use `EntityResolver` interface.

Question A050503

The `EntityResolver` was unable to find external entity. What is the best scenario to hanle this situation?

Options (select 1):

1. Throw `EntityNotFoundException` exception.
2. Return '`null`' value.

3. Throw `IOException` exception.
4. Throw `SAXException` exception.

5.6. Describe the functions and capabilities of JAXB, including the JAXB process flow, such as XML-to-Java and Java-to-XML, and the binding and validation mechanisms provided by JAXB.

Question A050601

What are the correct steps to build Java objects from XML file (with validation)?

1.
```
Unmarshaller um = context.createUnmarshaller();
```
2.
```
JAXBContext context = JAXBContext.newInstance("com.example");
```
3.
```
um.setValidating(true);
```
4.
```
JAXBContext context = new JAXBContext("com.example");
```
5.
```
context.setValidating(true);
```
6.
```
Unmarshaller um = context.getUnmarshaller();
```
7.
```
Item item = (Item)um.marshal(new File("item.xml"));
```
8.
```
Item item = (Item)um.unmarshal(new File("item.xml"));
```

Select 4 options in the correct order:

```
[   ]
[   ]
[   ]
[   ]
```

Question A050602

How can you validate Java objects content tree created with JAXB before marshalling it to XML file?

Options (select 1)

1.
```
Item item = ...
JAXBContext context = JAXBContext.newInstance("com.example");
Marshaller m = context.createMarshaller();
m.setValidating(true);
m.marshal(item,  new FileOutputStream("newItem.xml"));
```
2.
```
Item item = ...
JAXBContext context = JAXBContext.newInstance("com.example");
Marshaller m = context.createMarshaller();
Validator v = context.createValidator();
```

```
v.validate(item);
m.marshal(item,  new FileOutputStream("newItem.xml"));
```

3.
```
Item item = ...
JAXBContext context = JAXBContext.newInstance("com.example");
context.setValidating(true);
Marshaller m = context.createMarshaller();
m.marshal(item,  new FileOutputStream("newItem.xml"));
```

4.
```
Item item = ...
JAXBContext context = JAXBContext.newInstance("com.example");
Marshaller m = context.createMarshaller();
Validator v = context.createValidator();
context.validate(item);
m.marshal(item,  new FileOutputStream("newItem.xml"));
```

Question A050603

What are the correct steps to create XML document from Java objects content tree?

1.
```
Marshaller m = context.createMarshaller();
```

2.
```
JAXBContext context = JAXBContext.newInstance("com.example");
```

3.
```
m.setValidating(true);
```

4.
```
JAXBContext context = new JAXBContext("com.example");
```

5.
```
context.setValidating(true);
```

6.
```
Marshaller m = context.getMarshaller();
```

7.
```
m.marshal(item, new FileOutputStream("newItem.xml"));
```

8.
```
m.unmarshal(item, new FileOutputStream("newItem.xml"));
```

Select 3 options in the correct order:

```
[    ]
[    ]
[    ]
```

Question A050604

You are performing development of Web Service, followed by deployment. What is the correct sequence of steps in JAXB data binding process?

Options (select 1):

1. Generate classes, compile, marshal, process content, unmarshal.

2. Marshal, process content, generate classes, compile, unmarshal.
3. Generate classes, compile, unmarshal, process content, marshal.
4. Unarshal, process content, generate classes, compile, marshal.

5.7. Create and use a SOAP message with attachments using the SAAJ APIs.

Question A050701

A developer has a requirement to send a spreadsheet report to Web Service in a SOAP message. What SAAJ class the developer can use?

Options (select 1):

1. `Attachment`
2. `SOAPAttachment`
3. `MimeAttachment`
4. `DataHandler`
5. `None of the above`

Question A050702

You deployed collaborating handlers that sign outgoing messages by placing a signature in the header and verify that the signature on incoming messages is correct. After you verified signature, the header element is not needed. How can you get signature header and remove it from incoming SOAP message in one method call?

Options (select 1):

1. ```
   Iterator headerElements = header.examineHeaderElements("sign-handler");
   ```
2. ```
   Iterator headerElements = header.removeHeaderElements("sign-handler");
   ```
3. ```
   Iterator headerElements = header.extractHeaderElements("sign-handler");
   ```
4. ```
   Iterator headerElements = header.retrieveHeaderElements("sign-handler");
   ```
5. It is not possible to do with one method call.

Question A050703

Which SAAJ code fragment produces the following SOAP Header?

```
<SOAP-ENV:Header>
    <wsi:Claim
        xmlns:wsi="http://ws-i.org/schemas/conformanceClaim/"
        conformsTo="http://ws-i.org/profiles/basic/1.1/" />
</SOAP-ENV:Header>
```

Options (select 1):

1.

```
SOAPHeader header = message.getSOAPHeader();
Name headerName = message.createName("Claim", "wsi", "http://ws-i.org/schemas/conformanceClai
SOAPHeaderElement headerElement = header.addHeaderElement(headerName);
headerElement.addAttribute(message.createName("conformsTo"), "http://ws-i.org/profiles/basic/
```

2.
```
SOAPHeader header = message.getSOAPHeader();
Name headerName = soapFactory.createName("Claim", "wsi", "http://ws-i.org/schemas/conformance
header.addHeaderElement(headerName);
header.addAttribute(soapFactory.createName("conformsTo"), "http://ws-i.org/profiles/basic/1.1
```

3.
```
SOAPHeader header = message.getSOAPHeader();
Name headerName = soapFactory.createName("Claim", "wsi", "http://ws-i.org/schemas/conformance
SOAPHeaderElement headerElement = header.addHeaderElement(headerName);
headerElement.addAttribute(soapFactory.createName("conformsTo"), "http://ws-i.org/profiles/ba
```

4.
```
SOAPHeader header = message.getSOAPHeader();
Name headerName = message.createName("Claim", "wsi", "http://ws-i.org/schemas/conformanceClai
header.addHeaderElement(headerName);
header.addAttribute(message.createName("conformsTo"), "http://ws-i.org/profiles/basic/1.1/");
```

Question A050704

You are constructing a valid `SOAPMessage` object. What SAAJ types are optional?

Options (select 2):

1. `javax.xml.soap.SOAPPart`
2. `javax.xml.soap.SOAPBody`
3. `javax.xml.soap.SOAPHeader`
4. `javax.xml.soap.SOAPEnvelope`
5. `javax.xml.soap.AttachmentPart`

Question A050705

You are writing a SOAP handler which extends `javax.xml.ws.handler.soap.SOAPHandler`:

```
public boolean handleMessage(SOAPMessageContext smc) {
    SOAPMessage message = smc.getMessage();
    // Use SAAJ API to access SOAP header
    ...
    return true;
}
```

The handler must inspect SOAP header information. How can you access SOAP header using SAAJ API?

Options (select 2):

1.
```
SOAPPart soapPart = message.getSOAPPart();
SOAPEnvelope envelope = soapPart.getSOAPEnvelope();
SOAPHeader header = envelope.getSOAPHeader();
```

2.
```
SOAPHeader header = message.getHeader();
```

3.
```
SOAPPart soapPart = message.getSOAPPart();
SOAPEnvelope envelope = soapPart.getEnvelope();
SOAPHeader header = envelope.getHeader();
```

4.
```
SOAPHeader header = message.getSOAPHeader();
```

Appendix 6. JAXR

6.1. Describe the function of JAXR in Web Service architectural model, the two basic levels of business registry functionality supported by JAXR, and the function of the basic JAXR business objects and how they map to the UDDI data structures.

Question A060101

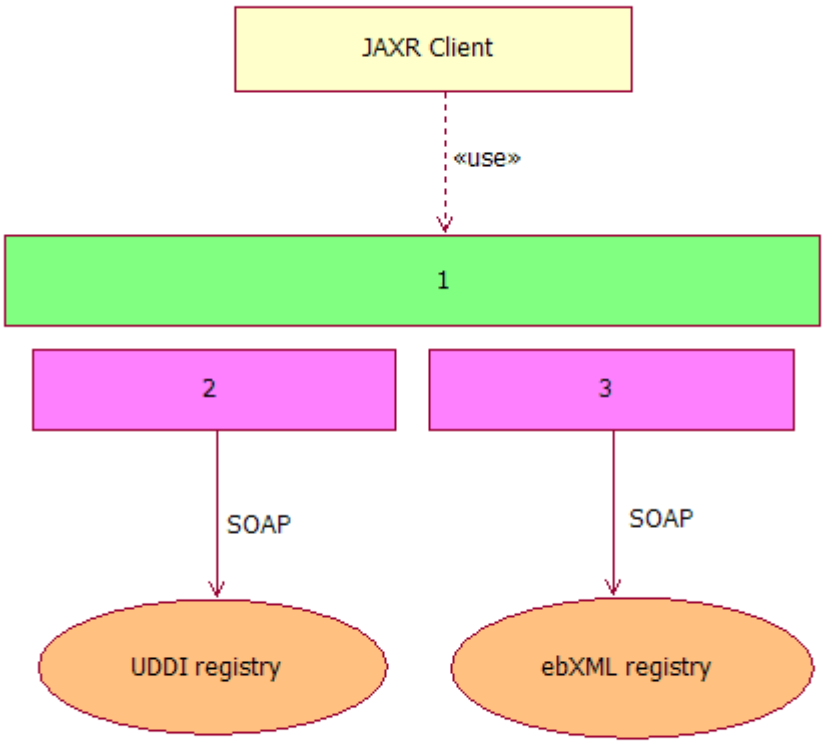
What statements are true about JAXR 1.0 ?

Options (select 2):

1. Each Registry Object can be identified by unique ID.
2. JAXR allows only to search Registry Objects.
3. JAXR allows to publish, search and modify Registry Objects.
4. JAXR clients can load pluggable Capability Profiles to use advanced search functions.

Question A060102

Looking at the JAXR architecture, name its components.



JAXR Architecture

Options (select 1):

1. 1 - JAXR API Capability Specific interfaces
   2 - Pluggable JAXR UDDI Provider
   3 - Pluggable JAXR ebXML Provider
2. 1 - JAXR API Capability Specific interfaces
   2 - Pluggable JAXR UDDI Client
   3 - Pluggable JAXR ebXML Client
3. 1 - SOAP API Capability Specific Interfaces
   2 - Pluggable JAXR Generic Provider
   3 - Pluggable JAXR Generic Provider
4. 1 - JAXB API Capability Specific interfaces
   2 - Pluggable JAXR UDDI Client
   3 - Pluggable JAXR ebXML Client

Question A060103

In the JAXR information model what interfaces map to the UDDI `tModel` XML data structure?

Options (select 1):

1. `Concept` and `Classification`
2. `Concept` and `ClassificationScheme`
3. `ConceptScheme` and `Classification`
4. `Concept` and `keyedReference`

Question A060104

What is true about JAXR?

Options (select 1):

1. It provides a uniform and standard Java API for accessing different kinds of XML registries.
2. It enables flexible XML processing from within Java programs.
3. It enables simplified XML processing using Java classes that are generated from XML schemas.
4. It provides an API for XML-based RPC communication in the Java platform.

Question A060105

Which capability profile level should support JAXR provider when publishing or discovering WS-I BP 1.1 conformant Web Service?

Options (select 1):

1. Capability Level 0
2. Capability Level 1
3. Capability Level 2

4. Capability Level 3

Question A060106

You are using JAXR for Web Services discovering and publishing. What is true about XML registry security?

Options (select 3):

1. JAXR Provider must authenticate with UDDI registry.
2. Authentication is performed when Client creates JAXR connection.
3. JAXR Client must authenticate with UDDI registry.
4. JAXR Provider sends queries to UDDI registry.
5. Authentication is performed only on certain queries types transparently for JAXR Client.
6. JAXR Client sends queries to UDDI registry.

Question A060107

Which JAXR method maps to UDDI Publisher API "`discard_authToken`" method(s)?

Options (select 1):

1. `BusinessLifeCycleManager.discardAuthToken()`
2. `BusinessLifeCycleManager.discard_authToken()`
3. `AuthenticationToken.discard()`
4. `AuthorizationToken.discard()`
5. Both `BusinessLifeCycleManager.discard_authToken()` and `AuthenticationToken.discard()`
6. None of the above.

Question A060108

Which methods allow the client to use SQL syntax for XML registry queries?

Options (select 1):

1. `BusinessQueryManager`'s Capability Level 0 methods
2. `DeclarativeQueryManager`'s Capability Level 0 methods
3. `BusinessQueryManager`'s Capability Level 1 methods
4. `DeclarativeQueryManager`'s Capability Level 1 methods

6.2. Create JAXR client to connect to a UDDI business registry, execute queries to locate services that meet specific requirements, and publish or update information about a business service.

Question A060201

You are testing JAXR client program which publishes service binding with bogus URL to local UDDI registry:

```
1
2 // Create, add, and configure a new ServiceBinding
```

```
3 ServiceBinding binding = lifeCycleMngr.createServiceBinding();
4 service.addServiceBinding(binding);
5 binding.setAccessURI("http://localhost/index.html");
6 ...
7 // Save
8
```

During saving, JAXR exception is thrown, indicating that access URL is invalid. How can you avoid this problem to test your JAXR client program locally?

Options (select 1):

1. It is not possible, all published URLs are verified by UDDI registry and must be valid.
2. Add the following code before line #5:

```
binding.setValidateURI(false);
```

3. Add the following code before line #5:

```
binding.setRedirectURI(true);
```

4. Add the following code before line #5:

```
binding.setIgnoreErrors(true);
```

Question A060202

You need to get all access URIs for Web Services which belong to your organization. How can you do this ?

Options (select 1):

1. Use `BusinessQueryManager.findOrganizations(...)` to find your organization in UDDI, `Organization.getServices()` to get all Web Services which belong to organization, `Service.getServiceBindings()` to get all Web Services bindings and get access URI from each `ServiceBinding`.
2. Use `BusinessQueryManager.findOrganizations(...)` to find your organization in UDDI, `BusinessQueryManager.getServices(organization)` to get all Web Services which belong to organization, `BusinessQueryManager.getServiceBindings(service)` to get all Web Service bindings and get access URI from each `ServiceBinding`.
3. Use `BusinessLifeCycleManager.findOrganizations(...)` to find your organization in UDDI, `BusinessLifeCycleManager.getServices(organization)` to get all Web Services which belong to organization, `BusinessLifeCycleManager.getServiceBindings(service)` to get all Web Service bindings and get access URI from each `ServiceBinding`.
4. Use `RegistryService.findOrganizations(...)` to find your organization in UDDI, `Organization.getServices()` to get all Web Services which belong to organization, `Service.getServiceBindings()` to get all Web Services bindings and get access URI from each `ServiceBinding`.

Question 060203

Until recent time your organization sold computer parts, now it start selling digital cameras. How can you update classification in JAXR registry to draw customers who are interested in digital cameras?

```
BusinessQueryManager     bqm = ... // get BusinessQueryManager
```

```
BusinessLifeCycleManager lcm = ... // get BusinessLifeCycleManager
Organization             org = ... // get your organization from registry
```

Options (select 1):

1.
```
ClassificationScheme scheme = new ClassificationScheme("ntis-gov:naics");
Classification classification = new Classification(scheme, "Digital Cameras", "123456");
Collection classifications = new ArrayList();
classifications.add(classification);
org.addClassifications(classifications);
Collection orgs = new ArrayList();
orgs.add(org);
lcm.saveOrganizations(orgs);
```

2.
```
ClassificationScheme scheme = bqm.findClassificationSchemeByName("ntis-gov:naics");
Classification classification = lcm.createClassification(scheme, "Digital Cameras", "123456")
Collection classifications = new ArrayList();
classifications.add(classification);
org.addClassifications(classifications);
Collection orgs = new ArrayList();
orgs.add(org);
lcm.saveOrganizations(orgs);
```

3.
```
Classification classification = lcm.createClassification("Digital Cameras", "123456");
Collection classifications = new ArrayList();
classifications.add(classification);
org.addClassifications(classifications);
Collection orgs = new ArrayList();
orgs.add(org);
lcm.saveClassifications(classification);
lcm.saveOrganizations(orgs);
```

4.
```
ClassificationScheme scheme = bqm.findClassificationSchemeByName("ntis-gov:naics");
Classification classification = lcm.createClassification(scheme, "Digital Cameras", "123456")
Collection classifications = new ArrayList();
classifications.add(classification);
org.addClassifications(classifications);
Collection orgs = new ArrayList();
orgs.add(org);
lcm.saveClassifications(classification);
lcm.saveOrganizations(orgs);
```

Question A060204

You are using JAXR registry to search for other businesses which sell digital cameras. In order to include more organizations in results you want to define in search criteria just name pattern, not exact organization name. Also, results should be sorted on the name field in ascending alphabetic sort order. How can you do this using JAXR API?

```
BusinessQueryManager bqm = ... // get BusinessQueryManager
```

Options (select 1):

1.
```
Collection<String> qualifiers = new ArrayList<String>();
qualifiers.add(FindQualifier.SORT_BY_NAME_ASC);
Collection<String> patterns = new ArrayList<String>();
patterns.add("*Digital Cameras*");
BulkResponse response = bqm.findOrganizations(qualifiers, patterns, null, null, null, null);
Collection<Organization> orgs = response.getCollection();
```

```

```

2.
```
Collection<String> qualifiers = new ArrayList<String>();
qualifiers.add(FindQualifier.SORT_BY_NAME_ASC);
Collection<String> patterns = new ArrayList<String>();
patterns.add("%Digital Cameras%");
BulkResponse response = bqm.findOrganizations(qualifiers, patterns, null, null, null, null);
Collection<Organization> orgs = response.getCollection();
```

3.
```
Collection<String> qualifiers = new ArrayList<String>();
qualifiers.add(FindQualifier.SORT_BY_NAME_ASC);
Collection<String> patterns = new ArrayList<String>();
patterns.add("*Digital Cameras*");
Collection<Organization> orgs = bqm.findOrganizations(qualifiers, patterns, null, null, null,
```

4.
```
Collection<String> qualifiers = new ArrayList<String>();
qualifiers.add(FindQualifier.SORT_BY_NAME_ASC);
Collection<String> patterns = new ArrayList<String>();
patterns.add("%Digital Cameras%");
Collection<Organization> orgs = bqm.findOrganizations(qualifiers, patterns, null, null, null,
```

5. It is not possible. You can not search with name patterns.
6. It is not possible. You can not sort the search results.

Question A060205

You are creating e-Business application which accesses third party Web Services. Unfortunately, after testing, you have discovered a problem with the code which responsible for Web Services lookup: you need to find all Web Services provided by "Amazon Books Co" and "Amazon Gadgets Co" companies, but you do not want search results include Web Services from "AMAZON SPRINGS WATER Co". How can you resolve this problem?

```
BusinessQueryManager bqm = ... // get BusinessQueryManager
```

Options (select 1):

1.
```
Collection<String> patterns = new ArrayList<String>();
patterns.add("Amazon");
BulkResponse response = bqm.findOrganizations(null, patterns, null, null, null, null);
Collection<Organization> orgs = response.getCollection();
```

2.
```
Collection<String> qualifiers = new ArrayList<String>();
qualifiers.add(FindQualifier.DISABLE_INSENSITIVE_MATCH);
Collection<String> patterns = new ArrayList<String>();
patterns.add("Amazon");
BulkResponse response = bqm.findOrganizations(qualifiers, patterns, null, null, null, null);
Collection<Organization> orgs = response.getCollection();
```

3.
```
Collection<String> qualifiers = new ArrayList<String>();
qualifiers.add(FindQualifier.CASE_SENSITIVE_MATCH);
Collection<String> patterns = new ArrayList<String>();
patterns.add("Amazon");
BulkResponse response = bqm.findOrganizations(qualifiers, patterns, null, null, null, null);
Collection<Organization> orgs = response.getCollection();
```

4. None of these.

Appendix 7. J2EE Web Services

7.1. Identify the characteristics of and the services and APIs included in the Java EE platform.

Question A070101

Which Java APIs are used for XML parsing?

Options (select 2):

1. SAAJ
2. StAX
3. JAX-RPC
4. SAX
5. JAXR

Question A070102

Which JEE 6.0 API supports asynchronous messaging between JEE components?

Options (select 1):

1. JMX
2. JMS
3. JAXP
4. JAXR
5. StAX

7.2. Explain the benefits of using the Java EE platform for creating and deploying Web Service applications.

blah

7.3. Describe the functions and capabilities of the JAXP, DOM, SAX, StAX, JAXR, JAXB, JAX-WS and SAAJ in the Java EE platform.

Question A070301

What is the purpose of JAXP API?

Options (select 1):

1. It enables Java software programmers to use a single, easy-to-use abstraction API to access a variety of XML registries.

2. It enables simplified XML processing using Java classes that are generated from XML schemas.
3. It enables applications to parse and transform XML documents independent of a particular XML processing implementation.
4. It enables Java clients to make remote procedure calls via XML and SOAP over HTTP.

Question A070302

You need to call a business method on Web Service which is already deployed in .NET environment. Which technology should you use?

Options (select 1):

1. DCOM
2. CORBA
3. JAX-WS
4. DOM
5. RMI

Question A070303

Which of the following APIs can be used in the application which requires vendor-independent XML parsing?

Options (select 1):

1. JAXB
2. JAXP
3. JAXR
4. XML4J
5. DOM4J
6. JAX-WS

Question A070304

Which SAAJ type is used to present SOAP message headers block?

Options (select 1):

1. `javax.xml.soap.SOAPPart`
2. `javax.xml.soap.AttachmentPart`
3. `javax.xml.soap.DetailEntry`
4. `javax.xml.soap.SOAPHeaderElement`

Question A070305

For a `javax.xml.soap.SOAPMessage` object, what can be in format other than XML?

Options (select 1):

1. `javax.xml.soap.SOAPPart`
2. `javax.xml.soap.AttachmentPart`

3. `javax.xml.soap.SOAPBody`
4. `javax.xml.soap.SOAPHeader`

Question A070306

Which SAAJ code snippets can be used to add the following XML fragment to SOAP message:

```
<name>Sony Reader Digital Book PRS-505</name>
```

(Assume that all variables are correctly initialized).

Options (select 2):

1.
```
Name name = soapFactory.createName("name");
SOAPElement element = orderElement.addChildElement(name);
element.addTextNode("Sony Reader Digital Book PRS-505");
```

2.
```
Name name = new Name("name");
SOAPElement element = orderElement.addChildElement(name);
element.addTextNode("Sony Reader Digital Book PRS-505");
```

3.
```
QName name = new QName("name");
SOAPElement element = orderElement.addChildElement(name);
element.addTextNode("Sony Reader Digital Book PRS-505");
```

4.
```
QName name = soapFactory.createName("name");
SOAPElement element = orderElement.addChildElement(name);
element.addTextNode("Sony Reader Digital Book PRS-505");
```

7.4. Describe the role of the WS-I Basic Profile when designing Java EE Web Services.

Question A070401

Web Services must be WS-I Basic Profile compliant. Is this statement true or false?

Options (select 1):

1. The statement is true only for Java EE 6.0 Web Services.
2. The statement is true for all Web Services.
3. The statement is true only for .NET Web Services.
4. The statement is false.

Appendix 8. Security

8.1. Explain basic security mechanisms including: transport level security, such as basic and mutual authentication and SSL, message level security, XML encryption, XML Digital Signature, and federated identity and trust.

Question A080101

What is true about XML Digital Signature?

Options (select 3):

1. Enveloping signature embeds the signed content into a signature container, the document is enclosed.
2. Detached signature is a signature which leaves the signed document in it's original state and provide the signature as additional data. Detached signature can be in separate XML file or reside within the same XML document but is sibling element of data object.
3. Enveloping signature is embedded in XML document that contains the signature as an element.
4. Detached signature is a signature which leaves the signed document in it's original state and provide the signature as additional data. Detached signature can be in separate XML file only, it can not reside with data object within the same XML document.
5. Enveloped signature is embedded in XML document that contains the signature as an element.
6. Enveloped signature embeds the signed content into a signature container, the document is enclosed.

Question A080102

What authentication forms are available for Java EE Web Services?

Options (select 2):

1. Basic Authentication
2. Digest Authentication
3. Form Based Authentication
4. HTTPS Based Authentication
5. Kerberos Authentication

8.2. Identify the purpose and benefits of Web services security oriented initiatives and standards such as Username Token Profile, SAML, XACML, XKMS, WS-Security, and the Liberty Project.

Question A080201

What are the key driving requirements for WS-Security communication protocol?

Options (select 3):

1. XML signature formats.
2. Non-repudiation.
3. Security token formats.
4. XML encryption technologies.
5. Establishing a security context.
6. Establishing authentication mechanisms.
7. Transport-level security.

Question A080202

Which initiative/standard enables single sign-on (SSO) service?

Options (select 1):

1. Username Token Profile
2. SAML
3. XACML
4. XKMS

Question A080203

Which specifications/recommendations WS-Security supports?

Options (select 3):

1. Username Token
2. XML Encoding
3. XML Signature
4. XML Encryption

8.3. Given a scenario, implement Java EE based web service web-tier and/or EJB-tier basic security mechanisms, such as mutual authentication, SSL, and access control.

Question A080301

Service endpoint uses HTTP BASIC authentication. How can you authenticate a user to use the specific Web Service?

Options (select 1):

1.
```
String username = "mikalai";
String password = "qwerty";
Calculator calculator = (new CalculatorService()).getCalculatorPort();
BindingProvider provider = (BindingProvider) calculator;
provider.getMessageContext().put(BindingProvider.USERNAME_PROPERTY, username);
provider.getMessageContext().put(BindingProvider.PASSWORD_PROPERTY, password);
```

2.
```
String username = "mikalai";
String password = "qwerty";
Calculator calculator = (new CalculatorService()).getCalculatorPort();
BindingProvider provider = (BindingProvider) calculator;
provider.getRequestContext().put(BindingProvider.USERNAME_PROPERTY, username);
provider.getRequestContext().put(BindingProvider.PASSWORD_PROPERTY, password);
```

3.
```
String username = "mikalai";
String password = "qwerty";
Calculator calculator = (new CalculatorService()).getCalculatorPort();
BindingProvider provider = (BindingProvider) calculator;
provider.getRequestContext().put(MessageContext.USERNAME_PROPERTY, username);
provider.getRequestContext().put(MessageContext.PASSWORD_PROPERTY, password);
```

4.
```
String username = "mikalai";
String password = "qwerty";
Calculator calculator = (new CalculatorService()).getCalculatorPort();
BindingProvider provider = (BindingProvider) calculator;
provider.getMessageContext().put(MessageContext.USERNAME_PROPERTY, username);
provider.getMessageContext().put(MessageContext.PASSWORD_PROPERTY, password);
```

Question A080302

You want to use HTTP BASIC authentication method for your service endpoint. Also, you know that in this case passwords can be intercepted. In addition to authentication you decided to use secure HTTPS connection (HTTP over SSL). How can you configure these options in deployment descriptor (`WEB-INF/web.xml`)?

Options (select 2):

1.
```
<!-- SECURITY CONSTRAINT -->
<security-constraint>
        ...
        <user-data-constraint>
                <transport-guarantee>SECURE</transport-guarantee>
        </user-data-constraint>
</security-constraint>

<!-- LOGIN AUTHENTICATION -->
<login-config>
        <auth-method>BASIC_AUTH</auth-method>
</login-config>
```

2.
```
<!-- SECURITY CONSTRAINT -->
<security-constraint>
        ...
        <user-data-constraint>
                <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
</security-constraint>

<!-- LOGIN AUTHENTICATION -->
<login-config>
        <auth-method>BASIC</auth-method>
</login-config>
```

3.
```
<!-- SECURITY CONSTRAINT -->
<security-constraint>
        ...
        <user-data-constraint>
                <transport-guarantee>SECURE</transport-guarantee>
        </user-data-constraint>
</security-constraint>

<!-- LOGIN AUTHENTICATION -->
<login-config>
        <auth-method>BASIC</auth-method>
</login-config>
```

4.
```
<!-- SECURITY CONSTRAINT -->
<security-constraint>
        ...
        <user-data-constraint>
                <transport-guarantee>INTEGRAL</transport-guarantee>
        </user-data-constraint>
</security-constraint>

<!-- LOGIN AUTHENTICATION -->
<login-config>
        <auth-method>BASIC</auth-method>
</login-config>
```

5.

```
<!-- SECURITY CONSTRAINT -->
<security-constraint>
        ...
        <user-data-constraint>
                <transport-guarantee>INTEGRAL</transport-guarantee>
        </user-data-constraint>
</security-constraint>

<!-- LOGIN AUTHENTICATION -->
<login-config>
        <auth-method>BASIC_AUTH</auth-method>
</login-config>
```

6.

```
<!-- SECURITY CONSTRAINT -->
<security-constraint>
        ...
        <user-data-constraint>
                <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
</security-constraint>

<!-- LOGIN AUTHENTICATION -->
<login-config>
        <auth-method>BASIC_AUTH</auth-method>
</login-config>
```

8.4. Describe factors that impact the security requirements of a Web Service, such as the relationship between the client and service provider, the type of data being exchanged, the message format, and the transport mechanism.

blah-blah

8.5. Describe WS-Policy that defines a base set of constructs that can be used and extended by other Web Services specifications to describe a broad range of service requirements and capabilities.

Question A080501

Which specification describes how senders and receivers (intermediaries and endpoints) can specify their security requirements and capabilities (e.g. required security tokens, supported encryption algorithms, privacy rules)?

Options (select 1):

1. WS-Security
2. WS-Policy
3. WS-Trust
4. WS-Privacy
5. WS-SecureConversation
6. WS-Federation
7. WS-Authorization

Appendix 9. Developing Web Services

9.1. Describe the steps required to configure, package, and deploy Java EE Web services and service clients, including a description of the packaging formats, such as .ear, .war, .jar, annotations and deployment descriptor settings.

Question A090101

Given the following Service Implementation Bean (SIB) code:

```
@WebService(serviceName="HelloService",
   name="HelloPortType",
   portName="HelloServicePort")
@Stateless(name="Hello")
public class Hello {
    public String sayHello(String name) {
        return "Hello, " + name;
    }
}
```

Which statements are correct about the Web Service packaging?

Options (select 4):

1. Web Service can be packaged in WAR module.
2. Web Service can be packaged in EJB-JAR module.
3. Web Service can be packaged in RAR module.
4. Compiled SIB class can placed in the "`/`" directory within archive.
5. Compiled SIB class can placed in the "`/WEB-INF/classes`" directory within archive.
6. Compiled SIB class can placed in the "`/META-INF/classes`" directory within archive.

Question A090102

Given the following Service Implementation Bean (SIB) code:

```
@WebService(serviceName="Hello")
public class Hello {
    public String sayHello(String name) {
        return "Hello, " + name;
    }
}
```

Which statements are correct about the Web Service packaging?

Options (select 2):

1. Web Service is packaged in WAR module.
2. Web Service is packaged in EJB-JAR module.
3. Web Service is packaged in RAR module.
4. Compiled SIB class is placed in the "`/`" directory within archive.
5. Compiled SIB class is placed in the "`/WEB-INF/classes`" directory within archive.
6. Compiled SIB class is placed in the "`/META-INF/classes`" directory within archive.

9.2. Given a set of requirements, develop code to process XML files using the SAX, StAX, DOM, XSLT, and JAXB APIs.

Question A090201

Your code is using vendor-neutral Java API for XML Processing (JAXP). You need to change parser factory implementation class, because current parser does not support namespaces. You know that JAXP API allows applications to plug in different JAXP compatible implementations of parsers or XSLT processors. What order is used when JAXP searches for the name of a concrete subclass of `DocumentBuilderFactory`?

```
import java.io.File;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;

public class TestDOMParsing {
        public static void main(String[] args) throws Exception {
                // Get Document Builder Factory
                DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
                DocumentBuilder builder = factory.newDocumentBuilder();
                Document doc = builder.parse(new File(args[0]));
        }
}
```

1. The value of a system property `javax.xml.parsers.DocumentBuilderFactory` if it exists and is accessible.
2. The contents of the file `$JAVA_HOME/jre/lib/jaxp.properties` if it exists.
3. The JAR Service Provider discovery mechanism specified in the JAR File Specification. A JAR file can have a resource (i.e. an embedded file) such as `META-INF/services /javax.xml.parsers.DocumentBuilderFactory` containing the name of the concrete class to instantiate.
4. The fallback platform default implementation.

Options (select 1):

1. 2, 1, 3, 4
2. 1, 2, 3, 4
3. 3, 2, 1, 4
4. 2, 3, 1, 4

Question A090202

What are the correct ways to parse XML file with SAX parser?

```
import java.io.File;

import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.InputSource;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.helpers.XMLReaderFactory;

public class ParseWithSAX {
        public static void main(String[] args) throws Exception {
```

```
                String fileName = "example.xml";

                // add code here

                System.out.println("XML file is parsed");
        }
}
```

Options (select 3):

1.
```
XMLReader xmlReader = XMLReaderFactory.createXMLReader();
xmlReader.setContentHandler(new MyContentHandler());
xmlReader.parse(fileName);
```

2.
```
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser saxParser = factory.newSAXParser();
XMLReader xmlReader = saxParser.getXMLReader();
InputSource source = new InputSource(fileName);
xmlReader.setContentHandler(new MyContentHandler());
xmlReader.parse(source);
```

3.
```
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser saxParser = factory.newSAXParser();
saxParser.parse(new File(fileName), new MyContentHandler());
```

4.
```
SAXParserFactory factory = new SAXParserFactory();
SAXParser saxParser = factory.newSAXParser();
XMLReader xmlReader = saxParser.getXMLReader();
InputSource source = new InputSource(fileName);
xmlReader.setContentHandler(new MyContentHandler());
xmlReader.parse(source);
```

5.
```
XMLReader xmlReader = XMLReaderFactory.newXMLReader();
xmlReader.setContentHandler(new MyContentHandler());
xmlReader.parse(fileName);
```

6.
```
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser saxParser = factory.createSAXParser();
saxParser.parse(new File(fileName), new MyContentHandler());
```

Question A090203

What is the correct way to parse XML file with DOM parser?

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;

public class ParseWithDOM {
        public static void main(String[] args) throws Exception {
                String fileName = "example.xml";

                // add code here

                System.out.println("XML file is parsed");
        }
}
```

Options (select 1):

1.
```
DocumentBuilderFactory factory = new DocumentBuilderFactory();
DocumentBuilder builder = factory.newDocumentBuilder();
Document document = builder.parse(fileName);
```

2.
```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document document = builder.parse(fileName);
```

3.
```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.createDocumentBuilder();
Document document = builder.parse(fileName);
```

4.
```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.createDocumentBuilder();
Document document = builder.parse(new File(fileName), new MyContentHandler());
```

Question A090204

The XML file you are parsing references multiple XML Schemas. As a result - it uses namespaces to avoid elements names ambiguity. How can you make your XML parsers support namespaces and provide namespace URIs, local element names and prefixed element names?

```
import java.io.File;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.w3c.dom.Document;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.DefaultHandler;

public class ParseWithNamespaces {

        public static void main(String[] args) throws Exception {
                String fileName = "example.xml";

                DocumentBuilderFactory domFactory = DocumentBuilderFactory.newInstance();
                SAXParserFactory saxFactory = SAXParserFactory.newInstance();

                // 1

                DocumentBuilder builder = domFactory.newDocumentBuilder();
                SAXParser saxParser = saxFactory.newSAXParser();
                XMLReader xmlReader = saxParser.getXMLReader();

                // 2

                Document document = builder.parse(fileName);
                saxParser.parse(new File(fileName), new MyContentHandler());
        }
}
```

Options (select 1):

1.
```
// 1
```

```
domFactory.setNamespaceAware(true);
saxFactory.setNamespaceAware(true);
xmlReader.setFeature("http://xml.org/sax/features/namespace-prefixes", true);
xmlReader.setFeature("http://xml.org/sax/features/namespaces", true);

// 2
// nothing
```

2.
```
// 1
// nothing

// 2
domFactory.setNamespaceAware(true);
saxFactory.setNamespaceAware(true);
xmlReader.setFeature("http://xml.org/sax/features/namespace-prefixes", true);
xmlReader.setFeature("http://xml.org/sax/features/namespaces", true);
```

3.
```
// 1
xmlReader.setFeature("http://xml.org/sax/features/namespace-prefixes", true);
xmlReader.setFeature("http://xml.org/sax/features/namespaces", true);

// 2
domFactory.setNamespaceAware(true);
saxFactory.setNamespaceAware(true);
```

4.
```
// 1
domFactory.setNamespaceAware(true);
saxFactory.setNamespaceAware(true);

// 2
xmlReader.setFeature("http://xml.org/sax/features/namespace-prefixes", true);
xmlReader.setFeature("http://xml.org/sax/features/namespaces", true);
```

Question A090205

You are parsing SOAP fault message with SAX parser and need to know namespace for `faulcode` element value.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" >
  <soap:Body>
    <soap:Fault>
      <faultcode>soap:Client</faultcode>
      <faultstring>Wrong parameter</faultstring>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

What method of `org.xml.sax.ContentHandler` can you use to retrieve element value namespace URI?

Options (select 1):

1. `startDocument()`
2. `processingInstruction(...)`
3. `startElement(...)`

4. `ignorableWhitespace(...)`
5. `startPrefixMapping(...)`
6. `characters(...)`

Question A090206

How can you create instance of `javax.xml.parsers.DocumentBuilderFactory`?

Options (select 1):

1.
```
DocumentBuilderFactory factory = DocumentBuilderFactory.init();
```

2.
```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
```

3.
```
DocumentBuilderFactory factory = new DocumentBuilderFactory();
```

4.
```
DocumentBuilderFactory factory = DocumentBuilderFactory.newFactory();
```

Question A090207

You are writing a Java Servlet which transforms business data in XML format to HTML presentation format. Which JAXP code fragment correctly performs XSLT transformation in multi-threaded environment (servlet container)?

Options (select 1):

1.
```
private Transformer transformer;
...
// init()
TransformerFactory factory  = TransformerFactory.newInstance();
Source xsl = new StreamSource("xml2html.xsl");
transformer = factory.newTransformer(xsl);
...
// doGet()
Source request  = new StreamSource(in);
Result response = new StreamResult(out);
transformer.transform(request, response);
```

2.
```
private Transformer transformer;
...
// init()
TransformerFactory factory  = TransformerFactory.newInstance();
transformer = factory.newTransformer();
...
// doGet()
Source request  = new StreamSource(in);
Result response = new StreamResult(out);
Source xsl = new StreamSource("xml2html.xsl");
transformer.transform(request, response, xsl);
```

3.
```
private Templates templates;
...
// init()
TransformerFactory factory = TransformerFactory.newInstance();
Source xsl = new StreamSource("xml2html.xsl");
templates = factory.newTemplates(xsl);
...
```

```
// doGet()
Source request    = new StreamSource(in);
Result response   = new StreamResult(out);
Transformer transformer = templates.newTransformer();
transformer.transform(request, response);
```

4.
```
private Templates templates;
...
// init()
TransformerFactory factory = TransformerFactory.newInstance();
templates = factory.newTemplates();
...
// doGet()
Source request    = new StreamSource(in);
Result response   = new StreamResult(out);
Source xsl = new StreamSource("xml2html.xsl");
Transformer transformer = templates.newTransformer();
transformer.transform(request, response, xsl);
```

Question A090208

Currently your application is using following code to transform incoming XML documents with the same XSLT template. You need to rewrite your application for multi-threaded environment to improve performance. What is the best approach to use?

```
// code part 1
TransformerFactory factory = TransformerFactory.newInstance();
Source xsl = new StreamSource("template.xsl");
Transformer transformer = factory.newTransformer(xsl);

// code part 2
Source src  = new StreamSource(in);
Result res = new StreamResult(out);
transformer.transform(src, res);
```

Options (select 1):

1. Leave code part 1 common, and fork multiple threads which will run code part 2.
2. Fork multiple threads with code part 1 and 2.
3. Leave code part 1 common, but create `Templates` object from the factory instead of `Transformer` object, and fork multiple threads which will create `Transformer` object from `Templates` object and then run code part 2.
4. Leave code part 1 common, but remove code which creates `Transformer` object, fork multiple threads which will create `Transformer` object from factory and then run code part 2.
5. None of these.

Question A090209

Which statements are true about SAX parser?

Options (select 2):

1. It is useful when developer wants only a small subset of information contained in XML document.
2. It is memory intensive.
3. It is not CPU intensive.

4. It provides random access to XML document.
5. It can be used for XML document generation.

9.3. Given an XML schema for a document style Web service create a WSDL file that describes the service and generate a service implementation.

blah-blah

9.4. Given a set of requirements, create code to create an XML-based, document style, Web service using the JAX-WS APIs.

blah-blah

9.5. Implement a SOAP logging mechanism for testing and debugging a Web service application using Java EE Web Service APIs.

Question A090501

Which statements are true about JAX-WS handler chain?

Options (select 3):

1. It can be defined declaratively.
2. It can NOT be defined declaratively.
3. It can be defined programmatically.
4. It can NOT be defined programmatically.
5. Handlers of any type are invoked in the order they defined in handler chain.
6. Handlers of the same type are invoked in the order they defined in handler chain.

9.6. Given a set of requirements, create code to handle system and service exceptions and faults received by a Web Services client.

blah-blah

Appendix 10. Web Services Interoperability Technologies

10.1. Describe WSIT, the features of each WSIT technology and the standards that WSIT.

Question A100101

Which of the following security technologies are included in the Metro Web Service stack?

Options (select 3):

1. XACML
2. WS-SecureConversation
3. WS-Trust
4. WS-SecurityPolicy
5. WS-Addressing

10.2. Describe how to create a WSIT client from a Web Service Description Language (WSDL) file.

blah

10.3. Describe how to configure Web Service providers and clients to use message optimization.

blah

10.4. Create a Microsoft Windows Communication Foundation (WCF) client that accesses a Java Web Service.

Question A100401

A developer creates .NET 3.0 platform client for already deployed stock quotes Web Service written in Java. The stock quotes Web Service provides access to its WSDL at the following location:

```
http://java.boot.by/stockquote?wsdl
```

How can the developer generate Web Service client proxy class in C#?

Options (select 1):

1. ```
   svc.exe /config:StockClient.exe.config http://java.boot.by/stockquote?wsdl
   ```

2. ```
   wsimport.exe /config:StockClient.exe.config http://java.boot.by/stockquote?wsdl
   ```

3. ```
   svcutil.exe /config:StockClient.exe.config http://java.boot.by/stockquote?wsdl
   ```

4. ```
   csc.exe /config:StockClient.exe.config http://java.boot.by/stockquote?wsdl
   ```

10.5. Describes the best practices for production and consumption of data for interoperability between WCF Web Services and Java Web Service clients or between Java WebServices and WCF Web Service clients.

Question A100501

A Java EE 6.0 Web Service was designed for a stock broker company. A developer used Transfer Object to get stock data:

```
public class StockItem {

    public String symbol;
    public Double wholeSalePrice;
    public double retailPrice;
    ...
}
```

Which code fragment demonstrates the correct stock item class in C#, when the Web Service is accessed from .NET platform ?

Options (select 1):

1. ```
   public partial class StockItem
   {
       public string symbol;
       private double wholeSalePrice;
       private double retailPrice;
   ```
   ```

```
    ...
}
```

2.
```
public partial class StockItem
{
    public String symbol;
    private Decimal wholeSalePrice;
    private double retailPrice;
    ...
}
```

3.
```
public partial class StockItem
{
    public char[] symbol;
    private Double wholeSalePrice;
    private double retailPrice;
    ...
}
```

4.
```
public partial class StockItem
{
    public string symbol;
    private decimal wholeSalePrice;
    private double retailPrice;
    ...
}
```

5.
```
public partial class StockItem
{
    public string symbol;
    private double wholeSalePrice;
    private decimal retailPrice;
    ...
}
```

Question 100502

You are developing interoperable Web Service using "Start from Java" approach. Which code snippet demonstrates correct WSDL type schema definition?

```
public enum USState {
    AL, AK, AS
}
```

Options (select 1):

1.
```
<xs:simpleType name="usState">
  <xs:restriction base="xs:enum">
    <xs:string value="AL" />
    <xs:string value="AK" />
    <xs:string value="AS" />
  </xs:restriction>
</xs:simpleType>
```

2.
```
<xs:simpleType name="usState">
  <xs:restriction base="xs:enumeration">
```

```
    <xs:pattern value="AL"/>
    <xs:pattern value="AK"/>
    <xs:pattern value="AS"/>
  </xs:restriction>
</xs:simpleType>
```

3.

```
<xs:simpleType name="usState">
  <xs:restriction base="xs:string">
    <xs:enumeration value="AL" />
    <xs:enumeration value="AK" />
    <xs:enumeration value="AS" />
  </xs:restriction>
</xs:simpleType>
```

4.

```
<xs:simpleType name="usState">
  <xs:restriction base="xs:string">
    <xs:enum value="AL" />
    <xs:enum value="AK" />
    <xs:enum value="AS" />
  </xs:restriction>
</xs:simpleType>
```

Appendix 11. General Design and Architecture

11.1. Describe the characteristics of a service-oriented architecture and how Web Services fit this model.

Question A110101

What are the characteristics of Web Services?

Options (select 2):

1. Web Services reduce coupling.
2. Web Services increase reliability.
3. Web Services reduce network load.
4. Web Services increase speed.
5. Web Services reduce complexity.
6. Web Services increase interoperability.

11.2. Given a scenario, design a Java EE Web Service using Web Services Design Patterns (Asynchronous Interaction, JMS Bridge, Web Service Cache, Web Serive Broker), and Best Practices.

Question A110201

You need to provide access to one or more services provided by heterogeneous applications, using XML and web protocols. What is the best approach?

Options (select 1):

1. Use a Service Facade to expose services using XML and web protocols.

2. Use a Web Service Broker to expose services using XML and web protocols.
3. Use a Mediator to expose services using XML and web protocols.
4. Use a Web Service Cache to expose services using XML and web protocols.

Question A110202

Which statements are true about JMS Bridge?

Options (select 2):

1. It increases overhead.
2. It reduces overhead.
3. It provides a single cross-platform JMS implementation.
4. It can guarantee messaging between any two JMS implementations.

11.3. Describe how to handle the various types of return values, faults, errors, and exceptions that can occur during a Web service interaction.

Question A110301

What happens when `SOAPHandler` on service endpoint throws `javax.xml.ws.soap.SOAPFaultException` exception in `handleMessage(...)` method?

Options (select 2):

1. This indicates a fault. `SOAPHandler` implementation class has the responsibility of setting the SOAP fault in the SOAP message.
2. This indicates a fault. JAX-WS Runtime has the responsibility of setting the SOAP fault in the SOAP message.
3. This does not indicate a fault.
4. Further processing of request handlers in this handler chain is not terminated. Handler chain invokes the `handleFault(...)` method on handlers registered in the handler chain, beginning with the next handler and going forward in execution.
5. Further processing of request handlers in this handler chain is terminated. Handler chain invokes the `handleFault(...)` method on handlers registered in the handler chain, beginning with the next handler in reverse direction and going backward in execution.
6. Further processing of request handlers in this handler chain is terminated. Handler chain invokes the `handleFault(...)` method on handlers registered in the handler chain, beginning with the handler instance that threw the exception and going backward in execution.
7. Handler chain continues execution of `SOAPHandler`s in forward order with original SOAP message.

Question A110302

Web Service received a SOAP message which is not well formed. As a result, a SOAP fault was returned. What is the fault code returned by the server?

Options (select 1):

1. `soap:Server`
2. `soap:Client`
3. `soap:Sender`

4. `soap:Receiver`
5. None of the above.

11.4. Describe the role that Web Services play when integrating data, application functions, or business processes in a Java EE application.

blah

Appendix 12. Endpoint Design and Architecture

12.1. Given a scenario, design Web Service applications using information models that are either procedure-style or document-style.

blah-blah

12.2. Describe the function of the service interaction and processing layers in a Web Service.

Question A120201ZZZZ

What is true about Service Implementation Layers?

Options (select 3):

1. The Service Interaction Layer consists of the endpoint interface that the service exposes to clients and through which it receives client requests.
2. Sometimes multiple layers make a service unnecessarily complicated and, in these cases, it may be simpler to design the service as one layer.
3. The Service Processing Layer consists of the endpoint interface that the service exposes to clients and through which it receives client requests.
4. The Service Processing Layer holds all business logic used to process client requests.
5. Web Service Implementation is always separated in an Interaction Layer and a Processing Layer.
6. The Service Interaction Layer holds all business logic used to process client requests.

Question A120202

What Layer of Web Servise Implementation is the best place to validate incoming XML document?

Options (select 1):

1. Service Processing Layer.
2. Service Interaction Layer.
3. Service Parsing Layer.
4. Service Validation Layer.

12.3. Design a Web Service for an asynchronous, document-style process and describe how to refactor a Web Service from a synchronous to an asynchronous model.

Question A120301

You are hired by Snorcle, Inc. to refactor an existing request-response synchronous Web Service for better performance. The new Web Service should receive order requests from customers and proceed to batch-processing. The client should not wait for the response because it may take a

significant amount of time, the results are sent back later via SMTP. Which Java EE technology can can be added to facilitate the refactoring?

Options (select 1):

1. JDBC
2. JMS
3. RMI
4. JAXB

Question A120302

You are hired by Snorcle, Inc. to refactor the Order Processing Center (OPC) application. The OPC has a Web Service invoker that sends a supplier purchase order as a request to the supplier application's Web Service endpoint. Currently the supplier application fulfills the request immediately, but due to business requirements changes, the supplier may take up to 14 days to perform fulfillment. Which design decisions should you implement?

Options (select 2):

1. Establish a correlation identifier between the purchase order message and the supplier response message.
2. Implement the "rpc-literal" style binding for supplier response message.
3. Have the OPC application provide Web Service and accept the response.
4. Use RESTful architectural style for OPC Web Service invoker.

12.4. Describe how the characteristics, such as resource utilization, conversational capabilities, and operational modes, of the various types of Web service clients impact the design of a Web service or determine the type of client that might interact with a particular service.

Question A120401

You are developing a Web Service which must manage multiple clients in thread-safe manner. Also, the Web Service must manage distributed transactions. Which design would be the best choice?

Options (select 1):

1. Java Servlet as a Web Service endpoint.
2. Stateful Session bean as a Web Service endpoint.
3. Stateless Session bean as a Web Service endpoint.
4. POJO as a Web Service endpoint.
5. None of the above.

Oracle Certified Expert, Java Platform, Enterprise Edition 6 Web Services Developer Quiz

Mikalai Zaikin

IBA JV

Belarus
Minsk
<NZaikin[at]iba.by>

March 2011

Revision History		
Revision $Revision: 75 $	$Date: 2011-05-08 00:26:06 +0300 (нд, 08 Май 2011) $	$Author: mzaikin $
$Id: ocewsd6-quiz.xml 75 2011-05-07 21:26:06Z mzaikin $		

Abstract

The purpose of this document is to help in preparation for Java Platform, Enterprise Edition 6 Web Services Developer Certified Expert Exam (CX-310-232).

This document should not be used as the only study material for Oracle Certified Expert, Java Platform, Enterprise Edition 6 Web Services Developer Test. It might cover not all objective topics, and it might be not enough. I tried to make this document as much accurate as possible, but if you find any error, please let me know.

1.2.1. Use `@WebService` to indicate a service
1.2.2. Use `@WebMethod`, `@WebMethod(exclude)` to indicate service methods
1.2.3. Use `@SOAPBinding` to select doc/lit, doc/bare, rpc/lit style of web service
1.2.4. Use `@Oneway` where the service doesn't have any response
1.2.5. Use `@WebParam`, and `@WebResult` to customize parameter and operation names
1.2.6. Use checked exceptions to indicate service specific faults.
1.2.7. Use `wsgen` tool to generate artifacts in Java EE5 (optional in Java EE6, as artifacts are generated at run time).
1.2.8. Configure deployment descriptors (`web.xml`, `webservices.xml`) for URL patterns, HTTP security, container authorization, caller authentication, and message protection. JAX-WS runtime may also be configured to perform message layer authentication and protection.
1.2.9. Compile and package the Web Service into a WAR file
1.2.10. Deploy the web service into a Java EE servlet container

2. Create a RESTful web service in a servlet container
2.1. Create a web service using JAX-RS, refer to Jersey implementation for examples
2.1.1. Annotate a class with a `@Path` annotation to respond to URI templates.
2.1.2. Annotate the class's methods to respond to HTTP requests using the corresponding JAX-RS annotations (`@GET`, `@POST`, etc.).
2.1.3. Use the JAX-RS `@Consumes` and `@Produces` annotations to specify the input and output formats for the RESTful Web Service.
2.1.4. Use `@PathParam`, `@QueryParam`, `@MatrixParam` and `@HeaderParam` to extract request data.
2.1.5. Use the `UriInfo` and `UriBuilder` to create URIs that refer to resources in the service.
2.1.6. Use `ResponseBuilder` to create response with customized status and additional metadata.
2.1.7. Implement a `MessageBodyReader` and `MessageBodyWriter` to add support for custom request and response data types
2.1.8. Implement `ExceptionMapper` to map a custom `Exception` to a response.
2.1.9. Use `Request` to add support for HTTP preconditions.
2.1.10. Implement the functionality of the JAX-RS resource's methods.
2.1.11. Use `@Path` on a method to define a subresource.
2.1.12. Configure deployment descriptor (`web.xml`) for base URL pattern, HTTP security (via `security-constraints` in `web.xml`)
2.1.13. Compile and package
2.1.14. Deploy the web service in a Java EE servlet container

3. Create a SOAP based web service implemented by an EJB component
3.1. Create a web service starting from a WSDL file using JAX-WS
3.1.1. Use `wsimport` tool to generate artifacts and use customization files for wsimports if needed
3.1.2. Create an EJB web service implementations using annotations (`@Stateless` or `@Singleton`)
3.1.3. Configure deployment descriptors (`ejb-jar.xml`, `webservices.xml`) for transactions, etc.
3.1.4. Configure container role based access control via method-permissions in `ejb-jar.xml` or via access control annotations on EJB.
3.1.5. Configure caller authentication and message protection; either by Servlet Container via `web.xml`, and/or by JAX-WS message processing runtime.
3.1.6. Compile and package the web service into a EAR/WAR file (Java EE 6 - WAR can also have EJBs).
3.1.7. Deploy the web service into a Java EE container.
3.2. Create a web service starting from a Java source using JAX-WS
3.2.1. Use `wsgen` tool to generate artifacts in Java EE5 from EJB classes (optional in Java EE 6 - as artifacts are generated at run time).
3.2.2. Configure deployment descriptors (`ejb-jar.xml`, `webservices.xml`) for transactions, etc.
3.2.3. Configure container role based access control via method-permissions in

ejb-jar.xml or via access control annotations on EJB.
3.2.4. Configure caller authentication and message protection; either by Servlet Container via web.xml, and/or by JAX-WS message processing runtime.
3.2.5. Compile and package the Web Service into a WAR/EAR file.
3.2.6. Deploy the Web Service into a Java EE container.

4. Create a RESTful Web Service implemented by an EJB component
   4.1. Create a Web Service using JAX-RS from EJB classes.
      4.1.1. Annotate an enterprise bean class with a @Path annotation to respond to URL patterns.
      4.1.2. Annotate the class's methods to respond to HTTP requests using the corresponding JAX-RS annotations (@GET, @POST, etc.).
      4.1.3. Use the JAX-RS @Produces and @Consumes annotations to specify the input and output resources for the RESTful Web Service.
      4.1.4. Implement the functionality of the JAX-WS resource's methods.
      4.1.5. Configure container role based access control via method-permissions in ejb-jar.xml or via access control annotations on EJB.
      4.1.6. Configure caller authentication (for access control protected methods) and message protection by Servlet Container via web.xml.
      4.1.7. Compile and package.
      4.1.8. Deploy the web service in a Java EE servlet container.
5. Configure Java EE security for a SOAP web service
   5.1. Configure security requirements of service using Java EE-container based security (overlaps with steps in other tasks - repeated here for convenience)
      5.1.1. Configure security requirements through deployment descriptors (web.xml, webservices.xml) for a Servlet-based web service endpoint: container authorization, caller authentication, and message protection. JAX-WS runtime may also be configured to perform message layer authentication and protection.
      5.1.2. Configure security requirements through deployment descriptors (ejb-jar.xml, webservices.xml) for EJB-based web service endpoint:
      5.1.3. Configure security requirements through deployment descriptor (web.xml) for JAX-RS based web service endpoint.
6. Create a web service client for a SOAP based web service
   6.1. Create a standalone client.
      6.1.1. Use wsimport to generate artifacts.
      6.1.2. Create a client application using these artifacts.
      6.1.3. Package and deploy accordingly.
   6.2. Create a client in a managed component in a EE container.
      6.2.1. Use wsimport to generate artifacts.
      6.2.2. Using @WebserviceRef in the client application.
      6.2.3. Package and deploy accordingly.
7. Create a web service client for a RESTful web service
   7.1. Use a browser to access a JAX-RS resource
   7.2. Use the java.net.* APIs to access a JAX-RS resource.
   7.3. Use java.net.Authenticator to access a secure JAX-RS resource.
   7.4. Use Ajax to access a JAX-RS resource.
   7.5. Use the Jersey client API to access a JAX-RS resource.
   7.6. Use the JAX-WS HTTP binding to access a JAX-RS resource.
8. Create a SOAP based web service using Java SE platform.
   8.1. Create a web service starting from a WSDL file using JAX-WS.
      8.1.1. Use wsimport tool to generate artifacts and use customization files for wsimports if needed.
      8.1.2. Build the web service implementation using the above artifacts.
      8.1.3. Use Endpoint API to configure and deploy it in Java SE 6 platform.
   8.2. Create a web service starting from a Java source using JAX-WS.
      8.2.1. Use wsgen tool to generate artifacts in Java EE5 (optional in Java EE6 - as artifacts are generated at run time)
      8.2.2. Use Endpoint API to configure and deploy it in Java SE 6 platform.
9. Create handlers for SOAP web services.
   9.1. Configure SOAP and logical handlers on the server side.

- 9.1.1. Use `@HandlerChain` annotation.
- 9.1.2. Use deployment descriptors.
- 9.2. Configure SOAP and logical handlers on the client side.
  - 9.2.1. Use deployment descriptors.
  - 9.2.2. Use programmatic API.
- 10. Create low-level SOAP web services.
  - 10.1. Describe the functions and capabilities of the APIs included within JAXP.
  - 10.2. Describe the functions and capabilities of JAXB, including the JAXB process flow, such as XML-to-Java and Java-to-XML, and the binding and validation mechanisms provided by JAXB.
  - 10.3. Use `Provider` API to create a web service.
    - 10.3.1. Process the entire SOAP message, using the SAAJ APIs.
    - 10.3.2. Process only the SOAP body, using JAXB.
  - 10.4. Use `Dispatch` API to create a dynamic web service client.
- 11. Use MTOM and MIME in a SOAP Web Service.
  - 11.1. Use MTOM on the service.
    - 11.1.1. Use `@MTOM` annotation with a web service.
    - 11.1.2. Use MTOM policy in WSDL.
    - 11.1.3. Use MTOM in the deployment descriptors
    - 11.1.4. Use `MTOMFeature` with `javax.xml.ws.Endpoint` API
    - 11.1.5. Use `swaRef` in WSDL.
    - 11.1.6. Use MIME binding in WSDL.
  - 11.2. Use MTOM on the client.
    - 11.2.1. Use `MTOMFeature` with `getPort()` methods.
    - 11.2.2. Use MTOM in the deployment descriptors.
    - 11.2.3. Sending any additional attachments using `MessageContext` properties.
- 12. Use WS-Addressing with a SOAP web service
  - 12.1. Use Addressing on the service
    - 12.1.1. Use `@Addressing` annotation with a web service
    - 12.1.2. Use `wsam:Addressing` policy in WSDL
    - 12.1.3. Use Addressing in the deployment descriptors
    - 12.1.4. Use `AddressingFeature` with `javax.xml.ws.Endpoint` API.
    - 12.1.5. Use `@Action` and `@FaultAction` on the service methods.
    - 12.1.6. Use `WebServiceContext.getEndpointReference()`
  - 12.2. Use Addressing on the client.
    - 12.2.1. Use `AddressingFeature` with `getPort()` methods.
    - 12.2.2. Use Addressing in the deployment descriptors
    - 12.2.3. Use `BindingProvider.getEndpointReference()`
    - 12.2.4. Use `getPort(EndpointReference)` methods.
- 13. Configure Message Level security for a SOAP web service
  - 13.1. Select the appropriate Security Profile for your service. The selection would be based on a match of the Protection guarantees offered by the profile and those required by the service.
  - 13.2. Configure Username/Password callbacks required by the Username Token Profile.
  - 13.3. Configure any server side Validators as maybe applicable for the profile. There are defaults in GlassFish for most of them.
  - 13.4. Optimize interaction between client and server by using WS-SecureConversation.
- 14. Apply best practices to design and implement Web Services.
  - 14.1. Use different encoding schemes - fast infoset.
  - 14.2. Use GZIP for optimiziing message sizes.
  - 14.3. Use catalog mechanism for WSDL access.
  - 14.4. Refer to WS-I sample app for best practices.

II. Appendices

- 1. XML Web Service Standards
  - 1.1. Given XML documents, schemas, and fragments determine whether their syntax and form are correct (according to W3C schema) and whether they conform to the WS-I Basic Profile 1.1.
  - 1.2. Describe the use of XML schema in Java EE Web services.
- 2. SOAP 1.2 Web Service Standards
  - 2.1. List and describe the encoding types used in a SOAP message.

- 2.2. Describe the SOAP Processing and Extensibility Model.
- 2.3. Describe SOAP Message Construct and create a SOAP message that contains an attachment.

3. Describing and Publishing (WSDL and UDDI)
   - 3.1. Explain the use of WSDL in Web Services, including a description of WSDL's basic elements, binding mechanisms and the basic WSDL operation types as limited by the WS-I Basic Profile 1.1.
   - 3.2. Describe how WSDL enables one to separate the description of the abstract functionality offered by a service from concrete details of a service description such as "how" and "where" that functionality is offered.
   - 3.3. Describe the Component Model of WSDL including Descriptions, Interfaces, Bindings, Services and Endpoints.
   - 3.4. Describe the basic functions provided by the UDDI Publish and Inquiry APIs to interact with a UDDI business registry.
4. JAX-WS
   - 4.1. Explain JAX-WS technology for building web services and client that communicate using XML.
   - 4.2. Given a set of requirements for a Web Service, such as transactional needs, and security requirements, design and develop Web Service applications that use JAX-WS technology.
   - 4.3. Describe the Integrated Stack (I-Stack) which consist of JAX-WS, JAXB, StAX, SAAJ.
   - 4.4. Describe and compare JAX-WS development approaches.
   - 4.5. Describe the features of JAX-WS including the usage of Java Annotations.
   - 4.6. Describe the architecture of JAX-WS including the Tools SPI that define the contract between JAX-WS tools and Java EE.
   - 4.7. Describe creating a Web Service using JAX-WS.
   - 4.8. Describe JAX-WS Client Communications Models.
   - 4.9. Given a set of requirements, design and develop a Web Service client, such as a Java EE client and a stand-alone client, using JAX-WS.
   - 4.10. Given a set of requirements, create and configure a Web Service client that accesses a stateful Web Service.
5. REST, JSON, SOAP and XML Processing APIs (JAXP, JAXB and SAAJ)
   - 5.1. Describe the characteristics of REST Web Services.
   - 5.2. Describe the characteristics of JSON Web Services.
   - 5.3. Compare SOAP Web Services to REST Web Services.
   - 5.4. Compare SOAP Web Services to JSON Web Services.
   - 5.5. Describe the functions and capabilities of the APIs included within JAXP.
   - 5.6. Describe the functions and capabilities of JAXB, including the JAXB process flow, such as XML-to-Java and Java-to-XML, and the binding and validation mechanisms provided by JAXB.
   - 5.7. Create and use a SOAP message with attachments using the SAAJ APIs.
6. JAXR
   - 6.1. Describe the function of JAXR in Web Service architectural model, the two basic levels of business registry functionality supported by JAXR, and the function of the basic JAXR business objects and how they map to the UDDI data structures.
   - 6.2. Create JAXR client to connect to a UDDI business registry, execute queries to locate services that meet specific requirements, and publish or update information about a business service.
7. J2EE Web Services
   - 7.1. Identify the characteristics of and the services and APIs included in the Java EE platform.
   - 7.2. Explain the benefits of using the Java EE platform for creating and deploying Web Service applications.
   - 7.3. Describe the functions and capabilities of the JAXP, DOM, SAX, StAX, JAXR, JAXB, JAX-WS and SAAJ in the Java EE platform.
   - 7.4. Describe the role of the WS-I Basic Profile when designing Java EE Web Services.
8. Security
   - 8.1. Explain basic security mechanisms including: transport level security, such as basic and mutual authentication and SSL, message level security, XML encryption, XML Digital Signature, and federated identity and trust.
   - 8.2. Identify the purpose and benefits of Web services security oriented initiatives and

Preface

If you believe you have found an error in the "Oracle Certified Expert, Java Platform, Enterprise

Edition 6 Web Services Developer Quiz" or have a suggestion to improve it, please send an e-mail to me. Please, indicate the topic and page URL.

Part I. Exam Objectives

Chapter 1. Create an SOAP web service in a servlet container

1.1. Create a web service starting from a WSDL file using JAX-WS

1.1.1. Use `wsimport` tool to generate artifacts from WSDL

Question 01010101

What statements are true about the following WSDL `operation`:

```
<portType name="SubmitPurchaseOrderPortType">
    <operation name="SubmitPurchaseOrder">
        <input name="order" message="SubmitPurchaseOrderMessage"/>
    </operation>
</portType>
```

Options (select 2):

1. This is a notification operation.
2. This is a one-way operation.
3. This is a two-way operation.
4. JAX-WS implementation may NOT throw any checked exceptions.
5. JAX-WS implementation may throw only unchecked exceptions.
6. JAX-WS implementation may throw any exceptions.

Answer:

Correct options are 2 and 4.

WSDL has four transmission primitives that an endpoint can support. WSDL refers to these primitives as `operation`s.

- One-way. The endpoint receives a message.

  The grammar for a one-way operation is:

```
<wsdl:definitions .... > <wsdl:portType .... > *
        <wsdl:operation name="nmtoken">
           <wsdl:input name="nmtoken"? message="qname"/>
        </wsdl:operation>
    </wsdl:portType >
</wsdl:definitions>
```

The `input` element specifies the abstract message format for the one-way operation.

- Request-response. The endpoint receives a message, and sends a correlated message.

  The grammar for a request-response operation is:

```
<wsdl:definitions .... >
    <wsdl:portType .... > *
        <wsdl:operation name="nmtoken" parameterOrder="nmtokens">
           <wsdl:input name="nmtoken"? message="qname"/>
           <wsdl:output name="nmtoken"? message="qname"/>
           <wsdl:fault name="nmtoken" message="qname"/>*
        </wsdl:operation>
    </wsdl:portType >
</wsdl:definitions>
```

  The `input` and `output` elements specify the abstract message format for the request and response, respectively. The optional `fault` elements specify the abstract message format for any error messages that may be output as the result of the operation (beyond those specific to the protocol).

  Note that a request-response operation is an abstract notion; a particular binding must be consulted to determine how the messages are actually sent: within a single communication (such as a HTTP request/response), or as two independent communications (such as two HTTP requests).

- Solicit-response. The endpoint sends a message, and receives a correlated message.

  The grammar for a solicit-response operation is:

```
<wsdl:definitions .... >
    <wsdl:portType .... > *
        <wsdl:operation name="nmtoken" parameterOrder="nmtokens">
           <wsdl:output name="nmtoken"? message="qname"/>
           <wsdl:input name="nmtoken"? message="qname"/>
           <wsdl:fault name="nmtoken" message="qname"/>*
        </wsdl:operation>
    </wsdl:portType >
</wsdl:definitions>
```

  The `output` and `input` elements specify the abstract message format for the solicited request and response, respectively. The optional `fault` elements specify the abstract message format for any error messages that may be output as the result of the operation (beyond those specific to the protocol).

  Note that a solicit-response operation is an abstract notion; a particular binding must be consulted to determine how the messages are actually sent: within a single communication (such as a HTTP request/response), or as two independent communications (such as two HTTP requests).

- Notification. The endpoint sends a message.

  The grammar for a notification operation is:

```
<wsdl:definitions .... >
    <wsdl:portType .... > *
        <wsdl:operation name="nmtoken">
           <wsdl:output name="nmtoken"? message="qname"/>
        </wsdl:operation>
```

```
    </wsdl:portType >
</wsdl:definitions>
```

The `output` element specifies the abstract message format for the notification operation.

In one-way messaging, the client sends a message to a Web Service, but doesn't expect a reply message. This Message Exchange Pattern (MEP) is typically thought of as asynchronous messaging.

If an operation is declared with a single `input` but no `output`, it defines a one-way operation. By listing only an `input` message, the operation indicates that clients will send messages to the Web Service without expecting a response.

Unlike request-response operations, one-way operations MAY NOT specify `fault` elements and DO NOT generate fault messages. The messaging model is strictly unidirectional — faults cannot be sent back to the client.

Sources:

Web Services Description Language (WSDL) 1.1 - [`http://www.w3.org/TR/wsdl#_one-way`]

Question 01010102

Given the following WSDL fragment, which statement is true:

```
<portType name="BookQuote">
    <operation name="getBookPrice">
        <input name="isbn" message="ns1:GetBookPriceRequest"/>
        <output name="price" message="ns1:GetBookPriceResponse"/>
        <fault name="InvalidArgumentFault" message="ns1:InvalidArgumentFault"/>
        <fault name="SecurityFault" message="ns1:SecurityFault"/>
    </operation>
</portType>
```

Options (select 1):

1. The `BookQuote` operation is invalid, because it has two faults.
2. The `BookQuote` operation represents a solicit-response message exchange pattern.
3. Both `InvalidArgumentFault` and `SecurityFault` must be runtime exceptions.
4. None of the above.

Answer:

Correct option is 4.

Option 1 is wrong, because request-response operation allows zero or many faults. The grammar for a request-response operation is:

```
<wsdl:definitions .... >
    <wsdl:portType .... > *
        <wsdl:operation name="nmtoken" parameterOrder="nmtokens">
           <wsdl:input name="nmtoken"? message="qname"/>
           <wsdl:output name="nmtoken"? message="qname"/>
           <wsdl:fault name="nmtoken" message="qname"/>*
        </wsdl:operation>
    </wsdl:portType >
</wsdl:definitions>
```

Option 2 is wrong, because the grammar for a solicit-response operation is:

```
<wsdl:definitions .... >
    <wsdl:portType .... > *
        <wsdl:operation name="nmtoken" parameterOrder="nmtokens">
           <wsdl:output name="nmtoken"? message="qname"/>
           <wsdl:input name="nmtoken"? message="qname"/>
           <wsdl:fault name="nmtoken" message="qname"/>*
        </wsdl:operation>
    </wsdl:portType >
</wsdl:definitions>
```

Option 3 is wrong. A `wsdl:fault` element refers to a `wsdl:message` that contains a single part. The global element declaration referred to by that part is mapped to a Java bean, henceforth called a fault bean. An implementation generates a wrapper exception class that extends `java.lang.Exception` and contains the following methods:

- `WrapperException(String message, FaultBean faultInfo)` A constructor where `WrapperException` is replaced with the name of the generated wrapper exception and `FaultBean` is replaced by the name of the generated fault bean.
- `WrapperException(String message, FaultBean faultInfo, Throwable cause)` A constructor where `WrapperException` is replaced with the name of the generated wrapper exception and `FaultBean` is replaced by the name of the generated fault bean. The last argument, `cause`, may be used to convey protocol specific fault information.
- `FaultBean getFaultInfo()` Getter to obtain the fault information, where `FaultBean` is replaced by the name of the generated fault bean.

Faults must map to an exception such that it:

- Directly or indirectly inherits from `java.lang.Exception`, but MUST NOT inherit from `RuntimeException` nor `RemoteException`.
- Has at most a single property called "`message`" of type `java.lang.String` with corresponding single `String` argument constructor.
- Must be SOAP encoded.

Sources:

Web Services Description Language (WSDL) 1.1 - [`http://www.w3.org/TR/wsdl#_one-way`]

Web Services for Java EE - [`http://www.jcp.org/en/jsr/detail?id=109`]

The Java API for XML-Based Web Services (JAX-WS) 2.2 - Section 2.5 (Fault) - [`http://www.jcp.org/en/jsr/detail?id=224`]

Question 01010103

Snorcle, Inc. hired you to create a JAX-WS Web Service client for a Stock Quotes Web Service. You are given a WSDL file for the Web Service which implemented with C# on .NET platform. What is the correct sequence of steps?

Options (select 1):

1. Use RMI technology to look up `Remote` object in registry, then invoke object's methods from the client.

2. Use `wsimport` to generate Service Endpoint Interface object, then invoke SEI's methods from the client.
3. Use JAXP technology to parse WSDL, create Service Endpoint Interface from factory, then invoke SEI's methods from the client.
4. Use `wsgen` to generate Service Endpoint Interface object, then invoke SEI's methods from the client.
5. None of the above, because JAX-WS client can work only with Web Services deployed on JEE platform.

Answer:

Correct option is 2.

Java API for XML-Based Web Services (JAX-WS) tooling supports generating Java artifacts you need to develop static JAX-WS Web services clients when starting with a Web Services Description Language (WSDL) file.

The JAX-WS client programming model supports both the Dispatch client API and the Dynamic Proxy client API. The Dispatch client API is a dynamic client programming model, whereas the static client programming model for JAX-WS is the Dynamic Proxy client. The Dispatch and Dynamic Proxy clients enable both synchronous and asynchronous invocation of JAX-WS Web Services.

- The Dynamic Proxy client invokes a Web Service based on a service endpoint interface (SEI) that is provided. The JAX-WS Dynamic Proxy instances leverage the dynamic proxy function in the base Java SE Runtime Environment (JRE) 6. You must begin with a WSDL file if you are developing a static client.
- In contrast, the Dispatch client API, `javax.xml.ws.Dispatch`, is an XML messaging-oriented client that is intended for advanced XML developers who prefer using XML constructs. The Dispatch API can send data in either `PAYLOAD` or `MESSAGE` mode. When using the `PAYLOAD` mode, the Dispatch client is only responsible for providing the contents of the `soap:Body` and JAX-WS includes the payload in a `soap:Envelope` element. When using the `MESSAGE` mode, the Dispatch client is responsible for providing the entire SOAP envelope. You do not need a WSDL file if you are developing a dynamic client.

To develop Web Services clients based on the JAX-WS programming model, you must determine the client model that best suits the needs of your Web Service application. If you want the Web Services client to invoke the service based on service endpoint interfaces with a dynamic proxy, use the Dynamic Proxy API to develop a static Web Service client. After the proxies are created, the client application can invoke methods on these proxies just like a standard implementation of the service endpoint interfaces. However, if you want to work directly with XML rather than a Java abstraction and work with either the message structure or the message payload structure, use the Dispatch API to develop a dynamic Web Service client.

The static client programming model for JAX-WS is the called the Dynamic Proxy client. The Dynamic Proxy client invokes a Web Service based on a service endpoint interface that is provided. After you create the proxy, the client application can invoke methods on the proxy just like a standard implementation of those interfaces. For JAX-WS Web Service clients using the dynamic proxy programming model, use the JAX-WS tool, `wsimport`, to process a WSDL file and generate portable Java artifacts that are used to create a Web Service client. Create the following portable Java artifacts using the `wsimport` tool:

- Service Endpoint Interface (SEI)
- Service class
- Exception class that is mapped from the `wsdl:fault` class (if any)
- Java Architecture for XML Binding (JAXB) generated type values which are Java classes mapped from XML schema types.

Option 1 is wrong. RMI technology is not used to invoke Web Service. RMI uses JRMP protocol over TCP/IP, and Web Services use XML over HTTP.

Option 3 is wrong. The Java API for XML Processing (JAXP) enables applications to parse, transform, validate and query XML documents using an API that is independent of a particular XML processor implementation.

Option 4 is wrong. The `wsgen` tool reads a Web Service endpoint class and generates all the required artifacts for Web Service deployment, and invocation.

Sources:

Implementing static JAX-WS Web services clients - [`http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/twbs_devwbsjaxwsclient.html`]

Developing a JAX-WS client from a WSDL file - [`http://publib.boulder.ibm.com/infocenter/wasinfo/fep/topic/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/twbs_jaxwsclientfromwsdl.html`]

1.1.2. Use external and embedded `<jaxws:package>`, `<jaxws:enableWrapperStyle>`, `<jaxws:class>` customizations

blah-blah

1.1.3. Use JAXB customizations to configure mapping.

blah-blah

1.1.4. Build the web service implementation using the above artifacts.

blah-blah

1.1.5. Access `MessageContext.SERVLET_CONTEXT` from the injected `WebServiceContext`

Question 01010501

A developer is trying to access `WebServicesContext` object from the endpoint implementation class.

```
@WebService
public class EndpointJSE {

    // get web service context
    ...

    @WebMethod
    public String getUserPrincipal() {
        Principal principal = ctx.getUserPrincipal();
        return principal.getName();
    }
}
```

Which code fragment correctly demonstrates initializing of `WebServicesContext` in JAX-WS endpoint?

Options (select 1):

1.
```
@Resource
private WebServiceContext ctx;
```

2.
```
@MessageContext
private WebServiceContext ctx;
```

3.
```
@WebServiceContext
private WebServiceContext ctx;
```

4.
```
@Object
private WebServiceContext ctx;
```

5.
```
@WebServiceContext
private Object ctx;
```

Answer:

Correct option is 1.

The `javax.xml.ws.WebServiceContext` interface makes it possible for an endpoint implementation object and potentially any other objects that share its execution context to access information pertaining to the request being served.

The `WebServiceContext` is treated as an injectable resource that can be set at the time an endpoint is initialized. The `WebServiceContext` object will then use thread-local information to return the correct information regardless of how many threads are concurrently being used to serve requests addressed to the same endpoint object.

In Java SE, the resource injection denoted by the `WebServiceContext` annotation is REQUIRED to take place only when the annotated class is an endpoint implementation class.

The following code shows a simple endpoint implementation class which requests the injection of its `WebServiceContext`:

```
@WebService
public class EndpointJSE {

    // get web service context
    @Resource
    private WebServiceContext ctx;

    @WebMethod
    public String getUserPrincipal() {
        Principal principal = ctx.getUserPrincipal();
        return principal.getName();
    }
}
```

The `javax.annotation.Resource` annotation defined by JSR-250 is used to request injection of the `WebServiceContext`.

```
package javax.annotation;
import static java.lang.annotation.ElementType.*;
import static java.lang.annotation.RetentionPolicy.*;
@Target({TYPE, METHOD, FIELD})
@Retention(RUNTIME)
public @interface Resource {
    public enum AuthenticationType {
        CONTAINER,
        APPLICATION
    }
    String name() default "";
```

```
    Class type() default Object.class;
    AuthenticationType authenticationType() default AuthenticationType.CONTAINER;
    boolean shareable() default true;
    String mappedName() default "";
    String description() default "";
}
```

The following constraints apply to the annotation elements of a `@Resource` annotation used to inject a `WebServiceContext`:

- The `type` element MUST be either `java.lang.Object` (the default) or `javax.xml.ws.WebServiceContext`. If the former, then the resource MUST be injected into a field or a method. In this case, the type of the field or the type of the JavaBeans property defined by the method MUST be `javax.xml.ws.WebServiceContext`.
- The `authenticationType`, `shareable` elements, if they appear, MUST have their respective default values.

The above restriction on `type` guarantees that a resource type of `WebServiceContext` is either explicitly stated or can be inferred from the annotated field/method declaration. Moreover, the field/method type must be assignable from the type described by the annotation's `type` element.

Note: When using method-based injection, it is recommended that the method be declared as non-public, otherwise it will be exposed as a Web Service operation. Alternatively, the method can be marked with the `@WebMethod(exclude=true)` annotation to ensure it will not be part of the generated `portType` for the service.

Sources:

The Java API for XML-Based Web Services (JAX-WS) 2.2 - [`http://jcp.org/en/jsr/detail?id=224`]

Common Annotations for the Java Platform - [`http://jcp.org/en/jsr/detail?id=250`]

1.1.6. Configure deployment descriptors (`web.xml`, `webservices.xml`) for URL patterns, HTTP security, container authorization, caller authentication, and message protection. JAX-WS runtime may also be configured to perform message layer authentication and protection.

Question 01010601

A company wants to protect the access to their public Inventory Web Services using basic authentication. The Inventory Web Services are currently deployed in Java EE6 servlet container. What file should developer use to configure the security requirement?

Options (select 1):

1. `webservices.xml`
2. `web.xml`
3. `sun-web.xml`
4. `security.xml`

Answer:

Correct option is 2.

HTTP Basic Authentication, which is based on a username and password, is the authentication mechanism defined in the HTTP/1.0 specification. A web server requests a web client to authenticate the user. As part of the request, the web server passes the realm (a string) in which the user is to be authenticated. The web client obtains the username and the password from the user and transmits

them to the web server. The web server then authenticates the user in the specified realm.

The deployment descriptor, `web.xml` is the most important Java EE configuration piece of Java EE Web applications. The security configuration in this descriptor drives the semantics and operation of web container security. The `web.xml` deployment descriptor is located in `/WEB-INF/` directory.

A web container can authenticate a web client/user using either HTTP BASIC, HTTP DIGEST, HTTPS CLIENT or FORM based authentication schemes.

The BASIC authentication `login-config` element in `web.xml` would look like the following:

```
<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Example Basic Authentication</realm-name>
</login-config>
```

Option 1 is wrong.

Similar to Java API for XML-based RPC (JAX-RPC) Web services, you can use deployment descriptors to describe JAX-WS Web Services. For JAX-WS Web Services, the use of the `webservices.xml` deployment descriptor is optional because you can use annotations to specify all of the information that is contained within the deployment descriptor file. You can use the deployment descriptor file to augment or override existing JAX-WS annotations. Any information that you define in the `webservices.xml` deployment descriptor overrides any corresponding information that is specified by annotations.

The `webservices.xml` file describes the Web Service. It configures the WSDL file for the service, the name of the WSDL port that the service implements, the SEI, and also contains a link to the Servlet that implements the business logic.

Option 3 is wrong.

To deploy a WAR on the Application Server, the file must also contain a runtime deployment descriptor. The runtime deployment descriptor is an XML file that contains information such as the context root of the web application and the mapping of the portable names of an application's resources to the Application Server's resources. The GlassFish Application Server web application runtime DD is named `sun-web.xml` and is located in `/WEB-INF/` along with the web application DD (`web.xml`).

Option 4 is wrong. The file name is invalid.

Sources:

Java Servlet 3.0 Specification - [`http://jcp.org/en/jsr/detail?id=315`]

1.1.7. Compile and package the Web Service into a WAR file

blah-blah

1.1.8. Deploy the Web Service into a Java EE servlet container

blah-blah

1.2. Create a Web Service starting from a Java source using JAX-WS

1.2.1. Use `@WebService` to indicate a service

blah-blah

1.2.2. Use `@WebMethod`, `@WebMethod(exclude)` to indicate service methods

Question 01020201

Given the following Web Service code, what would be the name of the operation in the generated WSDL?

```
@WebService(serviceName="WeatherService",
            name="Weather",
            portName="WeatherServicePort")
public class WeatherWS {
    public String getWeather() {
        return "Sunny";
    }
}
```

Options (select 1):

1. `getWeatherMethod`
2. `getWeatherOperation`
3. `getWeatherWebMethod`
4. `getWeather`
5. None of the above.

Answer:

Correct option is 4. You can customize a method that is exposed as a Web Service operation. The `@WebMethod` annotation includes the following member-value pairs:

- `operationName`

  Name of the `wsdl:operation` matching this method. Default value is the name of the Java method.

- `action`

  The action for this operation. For SOAP bindings, this determines the value of the `soapAction`. By default is "".

- `exclude`

  Marks a method to NOT be exposed as a web method. Used to stop an inherited method from being exposed as part of this Web Service.

  If this element is specified, other elements MUST NOT be specified for the `@WebMethod`. By default is `false`.

Each `wsdl:operation` in a `wsdl:portType` is mapped to a Java method in the corresponding Java service endpoint interface.

In the absence of `@WebMethod` annotation, the `name` attribute of the `wsdl:operation` element will be the same as the name of a mapped Java method.

The generated WSDL fragment will look like:

```
...
<portType name="Weather">
    <operation name="getWeather">
        <input wsam:Action="http://java.boot.by/Weather/getWeatherRequest" message="tns:getWeather
```

```
        <output wsam:Action="http://java.boot.by/Weather/getWeatherResponse" message="tns:getWeath
    </operation>
</portType>
<binding name="WeatherServicePortBinding" type="tns:Weather">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="getWeather">
        <soap:operation soapAction=""/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
</binding>
...
```

Sources:

The Java API for XML-Based Web Services (JAX-WS) 2.2 - [`http://jcp.org/en/jsr/detail?id=224`]

Web Services Metadata for the Java Platform - [`http://jcp.org/en/jsr/detail?id=181`]

Question 01020202

What statement is correct about the Weather Web Service:

```
@WebService(serviceName="WeatherService",
            name="Weather",
            portName="WeatherServicePort")
public class WeatherWS {

    @WebMethod()
    public String getWeather() {
        return "Sunny";
    }

    @WebMethod()
    public String getWeather(@WebParam(name = "city") String city) {
        return "Weather in " + city + ": Sunny";
    }
}
```

Options (select 1):

1. Weather Web Service is not well encapsulated, because `@WebMethod` annotation may not be applied to `public` methods.
2. Weather Web Service is invalid, because `@WebMethod` annotation may only be applied to SEI methods.
3. This is a correct example of Web Service method overloading.
4. You need to use `@WebMethod(operationName="getWeatherByCity")` for the second method to sucessfully run Weather Web Service.
5. None of the above.

Answer:

Correct option is 5. The Weather Web Service will not deploy due to presence of overloaded Java methods.

Though Java permits method overloading, in WSDL the operation names have to be unique. By default, if `@WebMethod` annotation does not contain `operationName` attribute, the operation name is Java method's name. In our case this will make two operations with the same `getWeather` name. Therefore, the first thing is to use `@WebMethod(operationName="getWeatherByCity")` to avoid WSDL rules violation. However, this is not enough (and this is why option 4 is wrong).

Second problem is that in the Document/Literal/Wrapped mode (which is the default mode), JAX-WS generates wrapper JAXB beans based on the Java method name. For `RequestWrapper` the default name is the name of the Java method and for `ResponseWrapper`, the default name is the Java method name + "`Response`". Since in our case two Java methods have the same name, generating the wrappers is a problem.

With wrapper style, the `@RequestWrapper` annotation supplies the JAXB-generated request wrapper bean, the element name, and the namespace for serialization and deserialization with the request wrapper bean that is used at run time. When starting with a Java object, this element is used to resolve overloading conflicts in document literal mode. Only the `className` attribute is required in this case. To avoid clash, developer must add wrapper beans annotation to one of the overloaded methods. The correct Weather Web Service code will be as follows:

```
@WebService(serviceName="WeatherService",
            name="Weather",
            portName="WeatherServicePort")
public class WeatherWS {

    @WebMethod()
    public String getWeather() {
        return "Sunny";
    }

    @WebMethod(operationName="getWeatherByCity")
    @RequestWrapper(className="by.boot.java.Request")
    @ResponseWrapper(className="by.boot.java.Response")
    public String getWeather(@WebParam(name = "city") String city) {
        return "Weather in " + city + ": Sunny";
    }
}
```

Option 1 is wrong. The Web Service is well encapsulated (exposes business logic only through public interface)

Option 2 is wrong. Service Implementation Bean's and Service Endpoint Interface's methods can be annotated with `@WebMethod`.

Option 3 is wrong. This is invalid Web Service's method overloading.

Sources:

The Java API for XML-Based Web Services (JAX-WS) 2.2 - [`http://jcp.org/en/jsr/detail?id=224`]

Web Services Metadata for the Java Platform - [`http://jcp.org/en/jsr/detail?id=181`]

Question 01020203

Given the following Web Services implementations:

```
@WebService
public class BaseCatalogWS {

    @WebMethod
    public int getIDByName(String name) {
        ...
    }
```

```
    public String getNameByID(int id) {
        ...
    }
}
```

and

```
@WebService
public class CatalogWS extends BaseCatalogWS {

    @WebMethod
    public String getTitle(String isbn)  {
        ...
    }

    public String getAuthor(String isbn)  {
        ...
    }

    private String getPublisher(String isbn)  {
        ...
    }
}
```

How many web methods the `CatalogWS` exposes?

Options (select 1):

1. 1
2. 2
3. 3
4. 4
5. 5
6. None of the above.

Answer:

Correct option is 4. Totally, 4 methods are exposed (2 own methods and 2 methods inherited from `BaseCatalogWS`).

The `BaseCatalogWS` exposes both methods (refer JAX-WS 2.2 specification):

A Java class (not an interface) annotated with a `javax.jws.WebService` annotation can be used to define a Web Service.

In order to allow for a separation between Web Service interface and implementation, if the `WebService` annotation on the class under consideration has a `endpointInterface` element, then the interface referred by this element is for all purposes the SEI associated with the class.

Otherwise, the class implicitly defines a service endpoint interface (SEI) which comprises all of the public non-static or non-final methods that satisfy **one** of the following conditions:

1. They are annotated with the `javax.jws.WebMethod` annotation with the `exclude` element set to `false` or missing (since `false` is the default for this annotation element)
2. They are not annotated with the `javax.jws.WebMethod` annotation but their declaring class has a `javax.jws.WebService` annotation

The statement above means that following methods are exposed:

- `BaseCatalogWS.getIDByName` (because it has `@WebMethod` annotation)
- `BaseCatalogWS.getNameByID` (because it's `public` and defined in class with `@WebService` annotation)
- `CatalogWS.getTitle` (because it has `@WebMethod` annotation)
- `CatalogWS.getAuthor` (because it's `public` and defined in class with `@WebService` annotation)

Note: method `CatalogWS.getPublisher` is not exposed, because it's not `public`.

According to Common Annotations for the Java Platform specification (JSR-250), section 2.1:

Members inherited from a superclass and which are not hidden or overridden maintain the annotations they had in the class that declared them, including member-level annotations implied by class-level ones.

This means that inherited methods will also be exposed:

- `CatalogWS.getIDByName` (because it has `@WebMethod` annotation)
- `CatalogWS.getNameByID` (because it's `public` and defined in class with `@WebService` annotation)

So, totally 4 methods will be exposed:

1. `CatalogWS.getTitle(java.lang.String)`
2. `CatalogWS.getAuthor(java.lang.String)`
3. `CatalogWS.getIDByName(java.lang.String)`
4. `CatalogWS.getNameByID(int)`

Figure 1.1. JAX-WS web methods inheritance

Sources:

The Java API for XML-Based Web Services (JAX-WS) 2.2 [section 3.3] - [`http://jcp.org/en/jsr/detail?id=224`]

Common Annotations for the Java Platform 1.1 [section 2.1] - [`http://jcp.org/en/jsr/detail?id=250`]

1.2.3. Use `@SOAPBinding` to select doc/lit, doc/bare, rpc/lit style of web service

Question 01020301

What statement is correct about the Hello Web Service:

```
@WebService()
public class HelloWS {

    @WebMethod(operationName="sayHelloToAnonymous")
    public String sayHello() {
        return "Hello, Anonymous !";
    }

    @WebMethod(operationName="sayHelloToPerson")
    public String sayHello(@WebParam(name="name") String name) {
        return "Hello, " + name + "!";
    }
```

```
}
```

Options (select 2):

1. Hello Web Service can be successfully deployed without changes.
2. Hello Web Service can be successfully deployed if you rename Java methods to different names.
3. Hello Web Service can be successfully deployed if you annotate class with `@SOAPBinding(style=Style.DOCUMENT, parameterStyle=ParameterStyle.BARE, use=Use.LITERAL)`
4. Hello Web Service can be successfully deployed if you annotate class with `@SOAPBinding(style=Style.DOCUMENT, parameterStyle=ParameterStyle.WRAPPED, use=Use.LITERAL)`
5. Hello Web Service can be successfully deployed if you annotate class with `@SOAPBinding()`
6. Hello Web Service can be successfully deployed if you annotate class with `@RequestWrapper(className="Hello")` `@ResponseWrapper(className="HelloResponse")`

Answer:

Correct options are 2 and 3.

The main problem is that in the Document/Literal/Wrapped mode (which is the default mode), JAX-WS generates wrapper JAXB beans based on the Java method name. For `RequestWrapper` the default name is the name of the Java method and for `ResponseWrapper`, the default name is the Java method name + "`Response`". Since in our case two Java methods have the same names, generating default JAXB wrappers is a problem which will fail Hello Web Service deployment.

Document Wrapped style is identified by a `javax.jws.SOAPBinding` annotation with the following properties: a style of `DOCUMENT`, a use of `LITERAL` and a parameterStyle of `WRAPPED`.

For the purposes of utilizing the JAXB mapping, each method is converted to two Java bean classes: one for the method input (the request bean) and one for the method output (the response bean). Application's programming model doesn't use these bean classes, so the applications need not package these classes. JAX-WS implementations may generate these classes dynamically.

In the absence of customizations, the wrapper request bean class MUST be named the same as the method and the wrapper response bean class MUST be named the same as the method with a "Response" suffix. The first letter of each bean name is capitalized to follow Java class naming conventions.

Therefore, one of the solutions could be to rename methods to different names, so wrapper JAXB beans will have different class names (option 2).

Another solution for successful deployment could be to change Hello Web Service SOAP binding to Document/Literal/Bare. In this case, JAXB beans not generated for message wrapper element (option 3).

Option 1 is wrong. The Web Service will not deploy due to JAXB beans names confict.

Options 4 and 5 are wrong. They both represent default Document/Literal/Wrapped mode which will fail due to wrapper bean class name clash.

Option 6 is wrong. The `@RequestWrapper` and `@ResponseWrapper` are method-level annotations, they could fix the depoyment problem have they been applied to one of the two methods.

The `@RequestWrapper` annotation is applied to the methods of an SEI. It is used to capture the JAXB generated request wrapper bean and the element name and namespace for marshalling /

unmarshalling the bean. The default value of `localName` element is the `operationName` as defined in `@WebMethod` annotation and the default value for the `targetNamespace` element is the target namespace of the SEI. When starting from Java, this annotation is used to resolve overloading conflicts in document/literal/wrapped mode. Only the `className` element is required in this case.

The `@ResponseWrapper` annotation is applied to the methods of an SEI. It is used to capture the JAXB generated response wrapper bean and the element name and namespace for marshalling / unmarshalling the bean. The default value of the `localName` element is the `operationName` as defined in the `@WebMethod` appended with "`Response`" and the default value of the `targetNamespace` element is the target namespace of the SEI. When starting from Java, this annotation is used to resolve overloading conflicts in document/literal/wrapped mode. Only the `className` element is required in this case.

Sources:

The Java API for XML-Based Web Services (JAX-WS) 2.2 - [`http://jcp.org/en/jsr/detail?id=224`]

Web Services Metadata for the Java Platform - [`http://jcp.org/en/jsr/detail?id=181`]

1.2.4. Use `@Oneway` where the service doesn't have any response

blah-blah

1.2.5. Use `@WebParam`, and `@WebResult` to customize parameter and operation names

blah-blah

1.2.6. Use checked exceptions to indicate service specific faults.

blah-blah

1.2.7. Use `wsgen` tool to generate artifacts in Java EE5 (optional in Java EE6, as artifacts are generated at run time).

blah-blah

1.2.8. Configure deployment descriptors (`web.xml`, `webservices.xml`) for URL patterns, HTTP security, container authorization, caller authentication, and message protection. JAX-WS runtime may also be configured to perform message layer authentication and protection.

Question 01020801

A developer needs to create a Web Service that supports client's session. To avoid memory leaks on the server, developer wants user session to be expired after 20 minutes of inactivity. Which config file should the developer use?

Options (select 1):

1. `web.xml`
2. `webservices.xml`
3. `jax-ws.xml`
4. `sun-jaxws.xml`
5. None of the above.

Answer:

Correct option is 1.

In the HTTP protocol, there is no explicit termination signal when a client is no longer active. This means that the only mechanism that can be used to indicate when a client is no longer active is a time out period. The default time out period for sessions is defined by the servlet container and can be obtained via the `getMaxInactiveInterval` method of the `HttpSession` interface.

The `session-config` element defines the session parameters for this Web application. The sub-element `session-timeout` defines the default session time out interval for all sessions created in this Web application. The specified time out must be expressed in a whole number of minutes. If the time out is 0 or less, the container ensures the default behavior of sessions is never to time out. If this element is not specified, the container must set its default time out period.

The `web.xml` Deployment Descriptor Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
    ...
    <session-config>
        <session-timeout>20</session-timeout>
    </session-config>
    ...
</web-app>
```

Option 2 is wrong. The `webservices.xml` deployment descriptor file defines the set of Web Services that are to be deployed in a Web Services for Java EE enabled container. With JAX-WS the use of `webservices.xml` is optional since the annotations can be used to specify most of the information specified in this deployment descriptor file. The deployment descriptors are only used to override or augment the annotation member attributes. The deployment descriptor defines the WSDL port to Port component relationship.

Option 3 is wrong. The file name is invalid.

Option 4 is wrong. The `sun-jaxws.xml` is JAX-WS RI deployment descriptor. The file describes Web Services endpoints. Each endpoint represents a port in the WSDL and it contains all information about implementation class, servlet url-pattern, binding, WSDL, service, port QNames.

Sources:

Java Servlet 3.0 Specification - [`http://jcp.org/en/jsr/detail?id=315`]

Implementing Enterprise Web Services - [`http://jcp.org/en/jsr/detail?id=109`]

1.2.9. Compile and package the Web Service into a WAR file

blah-blah

1.2.10. Deploy the web service into a Java EE servlet container

Question 01021001

Developer created and deployed the following JAX-WS Web Service using "Java-to-WSDL" approach:

```
@WebService
public class CalculatorWS {
        ...
}
```

At what URL this Web Service will be published by default (assume that servlet context "`CalculatorApp`" is correct and properly configured)?

Options (select 1):

1. `http://server.com/CalculatorApp/CalculatorWS`
2. `http://server.com/CalculatorApp/CalculatorWSPortBinding`
3. `http://server.com/CalculatorApp/CalculatorWSPort`
4. `http://server.com/CalculatorApp/CalculatorWSService`
5. `http://server.com/CalculatorApp/CalculatorWSPortType`

Answer:

Correct option is 4.

Web Services are deployed so that they are reachable via URIs based on their **service name**. The Web Service shown below:

```
@WebService
public class CalculatorWS {
        ...
}
```

will be available at the URL:

```
http://server.com/CalculatorApp/CalculatorWSService
```

When deploying JAX-WS-based Web Services into a web container, the JAX-WS-based Web Services are just packaged into a web application to be deployed into that web container. Once deployed, they are available.

The WSDL generated for the Web Service above will be as follows:

```
...
<binding name="CalculatorWSPortBinding" type="tns:CalculatorWS">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    ...
</binding>

<service name="CalculatorWSService">
    <port name="CalculatorWSPort" binding="tns:CalculatorWSPortBinding">
       <soap:address location="http://server.com/CalculatorApp/CalculatorWSService"/>
    </port>
</service>
...
```

Option 1 is wrong, the URL will be based on the Web Service name which by default is implementation class short name (without package) with added "`Service`" word.

Option 2 is wrong, the "`CalculatorWSPortBinding`" will be used as default binding name.

Option 3 is wrong, the "`CalculatorWSPort`" will be used as default port name.

Option 5 is wrong, the "`CalculatorWSPortType`" is invalid.

Sources:

JSR-181. Web Services Metadata for the Java Platform. [http://jcp.org/aboutJava/communityprocess/mrel/jsr181/index.html]

JAX-WS annotations [http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.wsfep.multiplatform.doc/info/ae/ae/rwbs_jaxwsannotations.html]

Metro Guide [Creating a Client to Consume a WSIT-Enabled Web Service] [http://metro.java.net/guide/Creating_a_Client_to_Consume_a_WSIT_Enabled_Web_Service.html]

Question 01021002

Given the following Weather Web Service implementation:

```
@WebService(serviceName="WeatherService",
            name="Weather",
            portName="WeatherServicePort")
public class WeatherWS {
    public String getWeather() {
        return "Sunny";
    }
}
```

After deployment in JAX-WS enabled container, what URL can be used to access the Weather Web Service (assume that servlet context "weather" is correct and properly configured)?

Options (select 1):

1. http://server.com/weather/Weather
2. http://server.com/weather/WeatherWS
3. http://server.com/weather/WeatherService
4. http://server.com/weather/WeatherServicePort
5. http://server.com/weather/WeatherWSService

Answer:

Correct option is 3.

Web Services are deployed so that they are reachable via URIs based on their **service name** (@WebService.serviceName).

The Weather Web Service shown below:

```
@WebService(serviceName="WeatherService",
            name="Weather",
            portName="WeatherServicePort")
public class WeatherWS {
    public String getWeather() {
        return "Sunny";
    }
}
```

will be available at the URL:

```
http://server.com/weather/WeatherService
```

When deploying JAX-WS-based Web Services into a web container, the JAX-WS-based Web Services are just packaged into a web application to be deployed into that web container. Once deployed,

they are available.

The WSDL generated for the Weather Web Service above will be as follows:

```
...
<portType name="Weather">
    ...
</portType>

<binding name="WeatherServicePortBinding" type="tns:Weather">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    ...
</binding>

<service name="WeatherService">
    <port name="WeatherServicePort" binding="tns:WeatherServicePortBinding">
        <soap:address location="http://server.com/weather/WeatherService"/>
    </port>
</service>
...
```

Option 1 is wrong, the URL will be based on the Web Service name which is provided in `@WebService` annotation's `serviceName` attribute.

Option 2 is wrong, the "`WeatherWS`" is invalid.

Option 4 is wrong, the "`WeatherServicePort`" will be used as `wsdl:port` name.

Option 5 is wrong, the "`WeatherWSService`" will be overriden by `serviceName` attribute.

The `@WebService` annotation marks a Java class as implementing a Web Service or marks a service endpoint interface (SEI) as implementing a Web Service interface.

- A Java class that implements a Web Service must specify either the `@WebService` or `@WebServiceProvider` annotation. Both annotations cannot be present.

  This annotation is applicable on a client or server SEI or a server endpoint implementation class for a JavaBeans endpoint.

- If the annotation references an SEI through the `endpointInterface` attribute, the SEI must also be annotated with the `@WebService` annotation.

- If the `@WebService` annotation of an implementation class references an SEI, the implementation class MUST NOT have any `@WebMethod` annotations.

- All `public` methods for an SEI are considered exposed methods regardless of whether the `@WebMethod` annotation is specified or not. It is incorrect to have an `@WebMethod` annotation on an SEI that contains the `exclude` attribute.

- For an implementation class that does not reference an SEI, if the `@WebMethod` annotation is specified with a value of `exclude=true`, that method is not exposed. If the `@WebMethod` annotation is not specified, all `public` methods are exposed including the inherited methods with the exception of methods inherited from `java.lang.Object`.

Properties of the `@javax.jws.WebService` annotation:

- `name`

  The name of the Web Service. Used as the name of the `wsdl:portType` when mapped to WSDL 1.1

  By default is simple name of the Java class or interface.

- `targetNamespace`

  If the `@WebService.targetNamespace` annotation is on a service endpoint interface, the `targetNamespace` is used for the namespace for the `wsdl:portType` (and associated XML elements).

  If the `@WebService.targetNamespace` annotation is on a service implementation bean that does NOT reference a service endpoint interface (through the `endpointInterface` annotation element), the `targetNamespace` is used for both the `wsdl:portType` and the `wsdl:service` (and associated XML elements).

  If the `@WebService.targetNamespace` annotation is on a service implementation bean that does reference a service endpoint interface (through the `endpointInterface` annotation element), the `targetNamespace` is used for only the `wsdl:service` (and associated XML elements).

- `serviceName`

  The service name of the Web Service. Used as the name of the `wsdl:service` when mapped to WSDL 1.1.

  This member-value is NOT allowed on endpoint interfaces.

  By default it is simple name of the Java class + "`Service`"

- `portName`

  Used as the name of the `wsdl:port` when mapped to WSDL 1.1.

  This member-value is NOT allowed on endpoint interfaces.

  By default it is `@WebService.name` + "`Port`"

- `wsdlLocation`

  The location of a pre-defined WSDL describing the service. The `wsdlLocation` is a URL (relative or absolute) that refers to a pre-existing WSDL file. The presence of a `wsdlLocation` value indicates that the service implementation bean is implementing a pre-defined WSDL contract. The JSR-181 tool MUST provide feedback if the service implementation bean is inconsistent with the `portType` and bindings declared in this WSDL. Note that a single WSDL file might contain multiple `portTypes` and multiple bindings. The annotations on the service implementation bean determine the specific `portType` and bindings that correspond to the Web Service.

- `endpointInterface`

  The complete name of the service endpoint interface defining the service's abstract Web Service contract. This annotation allows the developer to separate the interface contract from the implementation. If this annotation is present, the service endpoint interface is used to determine the abstract WSDL contract (`portType` and bindings). The service endpoint interface MAY include JSR-181 annotations to customize the mapping from Java to WSDL. The service implementation bean MAY implement the service endpoint interface, but is not REQUIRED to do so.

  This member-value is NOT allowed on endpoint interfaces.

Sources:

JSR-181. Web Services Metadata for the Java Platform. [`http://jcp.org/aboutJava/communityprocess/mrel/jsr181/index.html`]

JAX-WS annotations [`http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.wsfep.multiplatform.doc/info/ae`

/ae/rwbs_jaxwsannotations.html]

Chapter 2. Create a RESTful web service in a servlet container

2.1. Create a web service using JAX-RS, refer to Jersey implementation for examples

2.1.1. Annotate a class with a `@Path` annotation to respond to URI templates.

blah-blah

2.1.2. Annotate the class's methods to respond to HTTP requests using the corresponding JAX-RS annotations (`@GET`, `@POST`, etc.).

blah-blah

2.1.3. Use the JAX-RS `@Consumes` and `@Produces` annotations to specify the input and output formats for the RESTful Web Service.

Question 02010301

Snorcle, Inc. has deployed internal application to manage customers. To support interoperability with large base of standalone intranet applications, the services provided by this back-end application are exposed as XML-based RESTful Web Services. You are asked by management to add support of AJAX-based JSON-enabled web application clients which should consume the same RESTful Web Services. Estimate development efforts required to accomplish this requirement.

Options (select 1):

1. Development will not be able to accomplish this requirement, because same RESTful Web Service can not support in parallel both XML and JSON output formats.
2. Development can easily add new JSON output format to existing XML RESTful Web Service.
3. Development will be able to accomplish this requirement with significant amount of new code, in particular it will require create duplicate collection of resources: one for XML output, another for JSON output.
4. Development will not be able to accomplish this requirement, because RESTful Web Services do not support JSON output format.

Answer:

Correct option is 2.

The JAX-RS specification has a few facilities that help you manage content negotiation. It does method dispatching based on `Accept` header values.

- **Method Dispatching**

  The `@Produces` annotation denotes which media type a JAX-RS method should respond with. JAX-RS also uses this information to dispatch requests to the appropriate Java method. It matches the preferred media types listed in the `Accept` HTTP header of the incoming request to the metadata specified in `@Produces` annotations:

```
@Path("/customers")
public class CustomerResource {

    @GET
    @Path("{id}")
```

```
    @Produces("application/xml")
    public Customer getCustomerXml(@PathParam("id") int id) {
        ...
    }

    @GET
    @Path("{id}")
    @Produces("text/plain")
    public String getCustomerText(@PathParam("id") int id) {
        ...
    }

    @GET
    @Path("{id}")
    @Produces("application/json")
    public Customer getCustomerJson(@PathParam("id") int id) {
        ...
    }
}
```

Here, we have three methods that all service the same URI but produce different data formats. JAX-RS can pick one of these methods based on what is in the `Accept` HTTP header. For example, a client made the following request:

```
GET http://java.boot.by/customers/1
Accept: application/json;q=1.0, application/xml;q=0.5
```

The JAX-RS provider would dispatch this request to the `getCustomerJson()` method.

- **Leveraging content negotiation with JAXB**

  You can to use JAXB annotations to map Java objects to and from XML and JSON. Assume you annotate `Customer` class with JAXB annotations:

```
@XmlRootElement(name="customer")
@XmlAccessorType(XmlAccessType.FIELD)
public class Customer {
    @XmlAttribute
    protected int id;
    @XmlElement
    protected String fullname;
    public Customer() {}
    public int getId() { return this.id; }
    public void setId(int id) { this.id = id; }
    public String getFullName() { return this.fullname; }
    public void setFullName(String name} { this.fullname = name; }
}
```

  If you leverage JAX-RS integration with content negotiation, you can implement one Java method that can service both formats. This can save you from writing a whole lot of boilerplate code:

```
@Path("/service")
public class MyService {

    @GET
    @Produces({"application/xml", "application/json"})
    public Customer getCustomer(@PathParam("id") int id) {
        ...
    }

}
```

In this example, the `getCustomer()` method produces either XML or JSON, as denoted by the `@Produces` annotation applied to it. The returned object is an instance of a Java class, `Customer`, which is annotated with JAXB annotations. Since most JAX-RS implementations support using JAXB to convert to XML or JSON, the information contained within `Accept` HTTP header can pick which `MessageBodyWriter` to use to marshal the returned Java object.

Option 1 is wrong, because a single RESTful Web Service can support output in multiple formats.

Sources:

JAX-RS: Java API for RESTful Web Services - [`http://jcp.org/aboutJava/communityprocess/mrel/jsr311/index.html`]

Jersey 1.5 User Guide [2.1.3. `@Produces`] [`http://jersey.java.net/nonav/documentation/latest/user-guide.html#d4e150`]

2.1.4. Use `@PathParam`, `@QueryParam`, `@MatrixParam` and `@HeaderParam` to extract request data.

blah-blah

2.1.5. Use the `UriInfo` and `UriBuilder` to create URIs that refer to resources in the service.

blah-blah

2.1.6. Use `ResponseBuilder` to create response with customized status and additional metadata.

blah-blah

2.1.7. Implement a `MessageBodyReader` and `MessageBodyWriter` to add support for custom request and response data types

blah-blah

2.1.8. Implement `ExceptionMapper` to map a custom `Exception` to a response.

blah-blah

2.1.9. Use `Request` to add support for HTTP preconditions.

blah-blah

2.1.10. Implement the functionality of the JAX-RS resource's methods.

blah-blah

2.1.11. Use `@Path` on a method to define a subresource.

blah-blah

2.1.12. Configure deployment descriptor (`web.xml`) for base URL pattern, HTTP security (via `security-constraints` in `web.xml`)

Question 02011201

Given the following JAX-RS resource class:

```
@Path("/")
public class ClientResource {

    public static final HashMap<Integer, String> map = new HashMap<Integer, String>();
    static {
        map.put(1, "Mikalai Zaikin");
        map.put(2, "Volha Zaikina");
    }

    @GET
    @Path("/client/{id}")
    @Produces(MediaType.TEXT_PLAIN)
    public String retrieve(@PathParam("id") int clientNo) {
        return map.get(clientNo);
    }

    @POST
    @Path("/client")
    @Produces(MediaType.TEXT_HTML)
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    public String update(@FormParam("custno") int clientNo, @FormParam("custname") String name) {
        map.put(clientNo, name);
        return "'"+name+"' was added !";
    }
}
```

And a given `web.xml` deployment descriptor fragment:

```
...
<servlet>
    <servlet-name>ServletAdaptor</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>ServletAdaptor</servlet-name>
    <url-pattern>/resources/*</url-pattern>
</servlet-mapping>
...
```

How can you restrict unauthenticated users from accessing the `update(...)` method?

Options (select 1):

1.
```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>
            resources
        </web-resource-name>
        <url-pattern>/resources</url-pattern>
        <http-method>POST</http-method>
    </web-resource-collection>
    ...
</security-constraint>
```

2.
```
<security-constraint>
    <web-resource-collection>
```

```
        <web-resource-name>
            resources
        </web-resource-name>
        <url-pattern>/client</url-pattern>
        <http-method>POST</http-method>
    </web-resource-collection>
    ...
</security-constraint>
```

3.

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>
            resources
        </web-resource-name>
        <url-pattern>/update</url-pattern>
        <http-method>POST</http-method>
    </web-resource-collection>
    ...
</security-constraint>
```

4.

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>
            resources
        </web-resource-name>
        <url-pattern>/resources/*</url-pattern>
        <http-method>POST</http-method>
    </web-resource-collection>
    ...
</security-constraint>
```

Answer:

Correct option is 4.

The Jersey Servlet `com.sun.jersey.spi.container.servlet.ServletContainer` was deployed to process all URLs which follow this pattern: `http://server/context`**`/resources/*`**

POST updates will be performed via the URL: `http://server/context`**`/resources/client`**

Option 1 is wrong, because URL `http://server/context`**`/resources`** will be protected. This URL is processed by Jersey Servlet, but it does not match POST updates URL (see above).

Option 2 is wrong, because URL `http://server/context`**`/client`** will be protected. This URL will not even be processed by Jersey Servlet.

Option 3 is wrong, because URL `http://server/context`**`/update`** will be protected. This URL will not even be processed by Jersey Servlet.

Option 4 is correct.

Other correct options to protect the `update(...)` method would be:

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>
            resources
        </web-resource-name>
```

```
        <url-pattern>/resources/client/*</url-pattern>
        <http-method>POST</http-method>
    </web-resource-collection>
    ...
</security-constraint>
```

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>
            resources
        </web-resource-name>
        <url-pattern>/resources/client</url-pattern>
        <http-method>POST</http-method>
    </web-resource-collection>
    ...
</security-constraint>
```

A security constraint is used to define the access privileges to a collection of resources using their URL mapping.

You should specify a `security-constraint` element in the deployment descriptor file.

The following subelements can be part of a `security-constraint`:

- `web-resource-collection`

  A list of URL patterns (the part of a URL after the host name and port you want to constrain) and HTTP operations (the methods within the files that match the URL pattern you want to constrain) that describe a set of resources to be protected.

- `auth-constraint`

  Specifies whether authentication is to be used and names the roles authorized to perform the constrained requests.

- `user-data-constraint`

  Specifies how data is protected when transported between a client and a server.

A web resource collection consists of the following subelements:

- `web-resource-name` is the name you use for this resource.
- `url-pattern` is used to list the request URI to be protected. Many applications have both unprotected and protected resources. To provide unrestricted access to a resource, do not configure a security constraint for that particular request URI.

  The request URI is the part of a URL after the host name and port. For example, let's say that you have an e-commerce site with a catalog that you would want anyone to be able to access and browse, and a shopping cart area for customers only. You could set up the paths for your web application so that the pattern `/cart/*` is protected but nothing else is protected. Assuming that the application is installed at context path `/myapp`, the following are true:

  - `http://localhost:8080/myapp/index.xhtml` is NOT protected.
  - `http://localhost:8080/myapp/cart/index.xhtml` is protected.

  A user will be prompted to log in the first time he or she accesses a resource in the `cart/` subdirectory.

- `http-method` or `http-method-omission` is used to specify which methods should be protected

or which methods should be omitted from protection. An HTTP method is protected by a `web-resource-collection` under any of the following circumstances:

- If no HTTP methods are named in the collection (which means that all are protected)
- If the collection specifically names the HTTP method in an `http-method` subelement
- If the collection contains one or more `http-method-omission` elements, none of which names the HTTP method

Full `web.xml` deployment descriptor will look like:

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="3.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/w

    <servlet>
        <servlet-name>ServletAdaptor</servlet-name>
        <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>ServletAdaptor</servlet-name>
        <url-pattern>/resources/*</url-pattern>
    </servlet-mapping>

    <security-constraint>
        <web-resource-collection>
            <web-resource-name>
                resources
            </web-resource-name>
            <url-pattern>/resources/*</url-pattern>
            <http-method>POST</http-method>
        </web-resource-collection>
        <auth-constraint>
            <role-name>CustUser</role-name>
        </auth-constraint>
    </security-constraint>
    <login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>resources</realm-name>
    </login-config>
</web-app>
```

Client HTML page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>Customer Service. Populate Page</title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    </head>
    <body>
        <h1>Snorcle, Inc. Customer Service. Populate Page</h1>
        <form action="http://localhost:8080/HelloWorld/resources/client" method="POST">
            <table>
                <tr>
                    <td>Enter Customer ID :  </td>
                    <td><input type="text" name="custno" size="10"/></td>
                </tr>
                <tr>
                    <td>Customer Name :  </td>
                    <td><input type="text" name="custname" size="20"/> </td>
                </tr>
                <tr>
```

```
                    <td colspan="2"><input type="submit" value="Add Customer"/></td>
                </tr>
            </table>
        </form>
    </body>
</html>
```

Sources:

Securing Web Applications [http://download.oracle.com/javaee/6/tutorial/doc/gkbaa.html]

2.1.13. Compile and package

blah-blah

2.1.14. Deploy the web service in a Java EE servlet container

blah-blah

Chapter 3. Create a SOAP based web service implemented by an EJB component

3.1. Create a web service starting from a WSDL file using JAX-WS

3.1.1. Use `wsimport` tool to generate artifacts and use customization files for wsimports if needed

blah-blah

3.1.2. Create an EJB web service implementations using annotations (`@Stateless` or `@Singleton`)

blah-blah

3.1.3. Configure deployment descriptors (`ejb-jar.xml`, `webservices.xml`) for transactions, etc.

Question 03010301

The developer was asked to customize the following Weather Forecast Web Service:

```
package by.boot.java;

import javax.ejb.Stateless;
import javax.jws.WebService;

@Stateless(name="WeatherEJB")
@WebService(name="WeatherForecast", serviceName="WeatherService")
public class WeatherWS {
    public String getWeather() {
        return "Sunny";
    }
}
```

The new requirement is to add a handler chain to the existing Weather Forecast Web Service without changing of Java code. How this can be done?

Options (select 1):

1. Add `webservices.xml` deployment descriptor and link handler chain to the EJB using the following fragment:

```
...
<service-impl-bean>
    <ejb-link>
        WeatherEJB
    </ejb-link>
</service-impl-bean>
...
```

2. Add `webservices.xml` deployment descriptor and link handler chain to the EJB using the following fragment:

```
...
<service-impl-bean>
    <ejb-link>
        WeatherForecast
    </ejb-link>
</service-impl-bean>
...
```

3. Add `webservices.xml` deployment descriptor and link handler chain to the EJB using the following fragment:

```
...
<service-impl-bean>
    <ejb-link>
        WeatherService
    </ejb-link>
</service-impl-bean>
...
```

4. Add `webservices.xml` deployment descriptor and link handler chain to the EJB using the following fragment:

```
...
<service-impl-bean>
    <ejb-link>
        WeatherWS
    </ejb-link>
</service-impl-bean>
...
```

5. It is not possible without Java code modification.

Answer:

Correct option is 1.

JAX-WS along with JSR-181 requires that the Service Implementation Beans must include `javax.jws.WebService` class-level annotation to indicate that it implements a Web Service. If member attributes of the annotation are not specified then server side deployment descriptors are used. The member attributes of the annotation can also be overridden by server side deployment

descriptors.

For Stateless or Singleton Session EJBs using this annotation, the `name` attribute of the `javax.ejb.Stateless` or `javax.ejb.Singleton` annotation on the Service Implementation Bean class must be used as the `<ejb-link>` element in the deployment descriptor to map the Port component to the actual EJB. If `name` attribute in `javax.ejb.Stateless` or `javax.ejb.Singleton` annotation is not specified, then the default value is used as defined in the section 4.4.1 of EJB 3.1 (unqualified name of the session bean class).

Following `webservices.xml` deployment dscriptor could be used to customize handler chain for JAX-WS Web Service:

```
<?xml version="1.0" encoding="UTF-8"?>
<webservices xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/ja

    <webservice-description>
        <webservice-description-name>WeatherService</webservice-description-name>
        <port-component>
            <port-component-name>WeatherForecast</port-component-name>
            <wsdl-service xmlns:pfx="http://java.boot.by/">pfx:WeatherService</wsdl-service>
            <wsdl-port xmlns:pfx="http://java.boot.by/">pfx:WeatherWSPort</wsdl-port>
            <enable-mtom>false</enable-mtom>

            <service-impl-bean>
                <ejb-link>
                    WeatherEJB
                </ejb-link>
            </service-impl-bean>

            <handler-chains>
                <handler-chain>
                    <handler>
                        <handler-name>Login Handler</handler-name>
                        <handler-class>by.boot.java.LoginHandler</handler-class>
                    </handler>
                </handler-chain>
            </handler-chains>
        </port-component>
    </webservice-description>
</webservices>
```

Option 2 is wrong. The `WeatherForecast` is port component name.

Option 3 is wrong. The `WeatherService` is Web Service description name.

Option 4 is wrong. It could be correct default value if `@Stateeless.name` attribute had not been provided.

Option 5 is wrong. For JAX-WS Web services, the use of the `webservices.xml` deployment descriptor is optional because you can use annotations to specify all of the information that is contained within the deployment descriptor file. You can use the deployment descriptor file to augment or override existing JAX-WS annotations. Any information that you define in the `webservices.xml` deployment descriptor **overrides** any corresponding information that is specified by annotations.

Sources:

Web Services for Java EE - [`http://jcp.org/en/jsr/detail?id=109`]

3.1.4. Configure container role based access control via method-permissions in `ejb-jar.xml` or via access control annotations on EJB.

Question 03010401

A developer created a JAX-WS Web Service endpoint using a Singleton session bean:

```
@Singleton
@WebService
public class StatusBean {
    ...
    public String getState() {
        return state;
    }
}
```

Now developer needs to add role based access control to the Web Service business logic. What approaches can the developer use?

Options (select 2):

1. By using common annotations for the Java Platform.
2. By using `method-permission` element in `web.xml`
3. By using `method-permission` element in `ejb-jar.xml`
4. By using `security-constraint` element in `web.xml`
5. By using `auth-constraint` element in `ejb-jar.xml`

Answer:

Correct options are 1 and 3.

EJB 3.1 introduces Singleton session bean component that provides an easy access to shared state. A Singleton session bean is instantiated once per application. A Singleton session bean must be annotated with the `javax.ejb.Singleton` annotation or denoted in the deployment descriptor as a singleton session bean.

The business logic of a Web Service is implemented by a service provider in one of three different ways:

1. A Stateless Session Bean: The service provider implements the Web Service business logic by creating a stateless session Bean that implements the methods of the Service Endpoint Interface as described in the Enterprise JavaBeans 3.0 specification.
2. A Java class: The service provider implements the Web Service business logic according to the requirements defined by the JAX-RPC or JAX-WS Servlet based service implementation model.
3. A Singleton Session Bean: The service provider implements the JAX-WS Web Service business logic by creating a singleton session bean that implements the methods of the Service Endpoint Interface as described in the EJB 3.1 specification.

The Bean Provider and/or Application Assembler can specify the methods of the business, home, and component interfaces, no-interface view, and/or Web Service endpoints that some security role is allowed to invoke.

Metadata annotations and/or the deployment descriptor can be used for this purpose.

- **Specification of Method Permissions with Metadata Annotations**

  The method permissions for the methods of a bean class may be specified on the class, the business methods of the class, or both.

  The `@RolesAllowed`, `@PermitAll`, and `@DenyAll` annotations are used to specify method permissions. The value of the `@RolesAllowed` annotation is a list of security role names to be

mapped to the security roles that are permitted to execute the specified method(s). The `@PermitAll` annotation specifies that all security roles are permitted to execute the specified method(s). The `@DenyAll` annotation specifies that no security roles are permitted to execute the specified method(s).

Specifying the `@RolesAllowed` or `@PermitAll` or `@DenyAll` annotation on the bean class means that it applies to all applicable business methods of the class.

Method permissions may be specified on a method of the bean class to override the method permissions value specified on the bean class.

The following example code demonstrates the use of the `@RolesAllowed` annotation:

```
@DeclareRoles({"Administrator", "Manager", "Employee"})
@Singleton
@WebService
public class StatusBean {
    ...
    @RolesAllowed("Administrator")
    public String getState() {
        return state;
    }
}
```

- **Specification of Method Permissions in the Deployment Descriptor**

The Bean Provider may use the `ejb-jar.xml` deployment descriptor as an alternative to metadata annotations to specify the method permissions (or as a means to supplement or override metadata annotations for method permission values). The application assembler is permitted to override the method permission values using the bean's deployment descriptor.

Any values explicitly specified in the deployment descriptor override any values specified in annotations. If a value for a method has not be specified in the deployment descriptor, and a value has been specified for that method by means of the use of annotations, the value specified in annotations will apply. The granularity of overriding is on the per-method basis.

The Bean Provider or Application Assembler defines the method permissions relation in the deployment descriptor using the `method-permission` elements as follows:

  - Each `method-permission` element includes a list of one or more security roles and a list of one or more methods. All the listed security roles are allowed to invoke all the listed methods. Each security role in the list is identified by the `role-name` element, and each method (or a set of methods, as described below) is identified by the `method` element. An optional description can be associated with a `method-permission` element using the `description` element.

  - The method permissions relation is defined as the union of all the method permissions defined in the individual `method-permission` elements.

  - A security role or a method may appear in multiple `method-permission` elements.

Example: `ejb-jar.xml` descriptor fragment that illustrates the `method-permission` element usage:

```
<ejb-jar>
    ...
    <assembly-descriptor>
        <method-permission>
            <description>
                The Administrator role may access the getState
                method of the StatusBean bean
            </description>
```

```
            <role-name>Administrator</role-name>
            <method>
                <ejb-name>StatusBean</ejb-name>
                <method-name>getState</method-name>
            </method>
        </method-permission>
    </assembly-descriptor>
</ejb-jar>
```

Option 4 is wrong. The `web.xml` deployment descriptor can be used to restrict access to servlet based endpoint, not EJB-based endpoint.

Options 2 and 5 are wrong. Elements names are invalid for the proposed deployment descriptors.

Sources:

Enterprise JavaBeans 3.1 - [`http://jcp.org/en/jsr/detail?id=318`]

Common Annotations for the Java Platform - [`http://jcp.org/en/jsr/detail?id=250`]

3.1.5. Configure caller authentication and message protection; either by Servlet Container via `web.xml`, and/or by JAX-WS message processing runtime.

blah-blah

3.1.6. Compile and package the web service into a EAR/WAR file (Java EE 6 - WAR can also have EJBs).

blah-blah

3.1.7. Deploy the web service into a Java EE container.

blah-blah

3.2. Create a web service starting from a Java source using JAX-WS

3.2.1. Use `wsgen` tool to generate artifacts in Java EE5 from EJB classes (optional in Java EE 6 - as artifacts are generated at run time).

blah-blah

3.2.2. Configure deployment descriptors (`ejb-jar.xml`, `webservices.xml`) for transactions, etc.

blah-blah

3.2.3. Configure container role based access control via method-permissions in `ejb-jar.xml` or via access control annotations on EJB.

blah-blah

3.2.4. Configure caller authentication and message protection; either by Servlet Container via `web.xml`, and/or by JAX-WS message processing runtime.

blah-blah

3.2.5. Compile and package the Web Service into a WAR/EAR file.

blah-blah

3.2.6. Deploy the Web Service into a Java EE container.

blah-blah

Chapter 4. Create a RESTful Web Service implemented by an EJB component

4.1. Create a Web Service using JAX-RS from EJB classes.

4.1.1. Annotate an enterprise bean class with a `@Path` annotation to respond to URL patterns.

blah-blah

4.1.2. Annotate the class's methods to respond to HTTP requests using the corresponding JAX-RS annotations (`@GET`, `@POST`, etc.).

blah-blah

4.1.3. Use the JAX-RS `@Produces` and `@Consumes` annotations to specify the input and output resources for the RESTful Web Service.

blah-blah

4.1.4. Implement the functionality of the JAX-WS resource's methods.

blah-blah

4.1.5. Configure container role based access control via method-permissions in `ejb-jar.xml` or via access control annotations on EJB.

Question 04010501

Given the following JAX-RS resource class:

```
@Path(value="/addresses")
@PermitAll
@Stateless
public class AddressBookResource {

    @GET
    @Produces(value="text/plain")
    public String getList() {
        ...
    }

    @RolesAllowed("admin")
    @PUT
    public void updateList(String addr) {
        ...
    }
}
```

Which statement describes the configuration which would be required to support the access control shown above?

Options (select 1):

1. No further configuration is required, Java EE runtime will read annotation and configure web container automatically.
2. Developer must configure web container to authenticate access to the resource.
3. Developer must configure EJB container to authenticate access to the resource.
4. No further configuration is required, because class level annotation specifies that all security roles are permitted to access the JAX-RS resource.

Answer:

Correct option is 2.

Annotations for security follow the declarative security model. Security constraints that are configured in the deployment descriptor, the `web.xml` file, take precedence over security constraints that are programmatically annotated in the application. It is important for developers of JAX-RS resources to consider a balance across configurable security constraints and annotated security constraints. Annotated constraints are additional to any configured security constraints. The JAX-RS runtime environment checks for annotated constraints after the web container runtime environment has checked for security constraints that are configured in the `web.xml` file.

Configure authentication constraints in the `web.xml` file. In the following example `web.xml` file, the `SecurityConstraint_1` security constraint is defined. This constraint is used to require authentication to the application. Additionally, the `SecurityConstraint_1` security constraint defines constraints on URL patterns corresponding to JAX-RS resources. When a JAX-RS resource is accessed that corresponds to one of these constraints, authorization checks are performed. Access checks are performed for the declarative security annotations only after the configured constraints are verified.

```
<web-app>
    ...
    <security-constraint id="SecurityConstraint_1">
        <web-resource-collection>
            <web-resource-name>AddressBookAppSample</web-resource-name>
            <url-pattern>/*</url-pattern>
            <http-method>PUT</http-method>
        </web-resource-collection>

        <auth-constraint>
            <role-name>admin</role-name>
        </auth-constraint>
    </security-constraint>

    <security-role>
        <role-name>admin</role-name>
    </security-role>

    <login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>Sample Realm Name</realm-name>
    </login-config>

</web-app>
```

Option 1 is wrong. When the JAX-RS resources have authorization constraints associatedwith them, the JAX-RS runtime relies on the web container to obtain authentication information – this means that the web container must be configured to require authentication data when receiving requests that will be forwarded to the JAX-RS runtime to process.

Option 3 is wrong. EJB container does not perform authentication.

Option 4 is wrong. Method level annotations take precedence over annotations at the class level.

Sources:

Securing JAX-RS resources using annotations - [http://publib.boulder.ibm.com/infocenter/wasinfo/beta/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/twbs_jaxrs_impl_securejaxrs_annotations.html]

4.1.6. Configure caller authentication (for access control protected methods) and message protection by Servlet Container via `web.xml`.

blah-blah

4.1.7. Compile and package.

blah-blah

4.1.8. Deploy the web service in a Java EE servlet container.

blah-blah

Chapter 5. Configure Java EE security for a SOAP web service

5.1. Configure security requirements of service using Java EE-container based security (overlaps with steps in other tasks - repeated here for convenience)

5.1.1. Configure security requirements through deployment descriptors (`web.xml`, `webservices.xml`) for a Servlet-based web service endpoint: container authorization, caller authentication, and message protection. JAX-WS runtime may also be configured to perform message layer authentication and protection.

blah-blah

5.1.2. Configure security requirements through deployment descriptors (`ejb-jar.xml`, `webservices.xml`) for EJB-based web service endpoint:

5.1.2.1. Configure transactional support.

blah

5.1.2.2. Configure container role based access control via method-permissions in `ejb-jar.xml` or via access control annotations on EJB.

Question 0501020201

Given the following Web Service endpoint implementation:

```
@WebService
@Singleton(name="OrderBean")
public class SecureOrderBean {
    ...
}
```

and the following fragment from `ejb-jar.xml` deployment descriptor:

```
...
<assembly-descriptor>
    ...
```

```
    <security-role>
        <role-name>manager</role-name>
    </security-role>
    <method-permission>
        <role-name>manager</role-name>
        <method>
            <ejb-name>OrderBean</ejb-name>
            <method-name>*</method-name>
        </method>
    </method-permission>
    ...
</assembly-descriptor>
...
```

Which statement is true about security roles of the client of this SEI ?

Options (select 2):

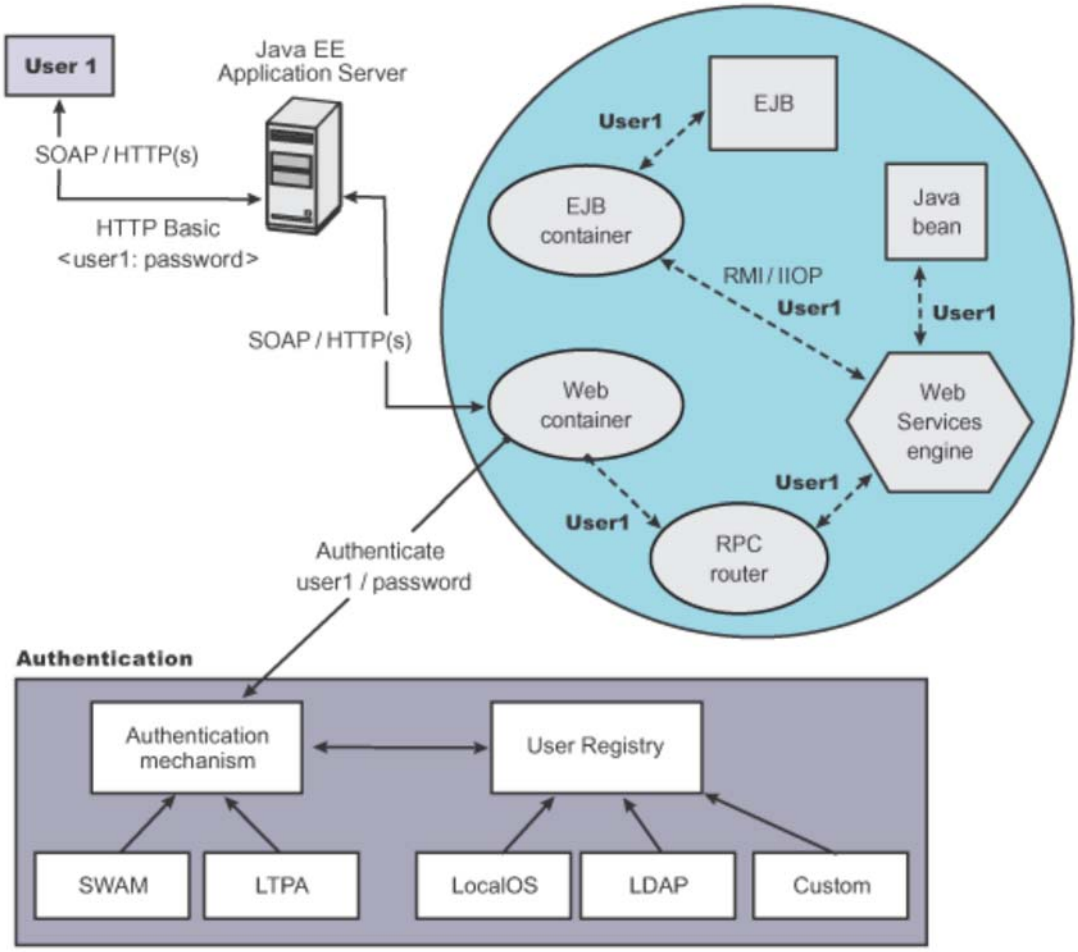
1. Only EJB client must be in `manager` role.
2. Only JAX-WS client must be in `manager` role.
3. Both EJB and JAX-WS clients must be in `manager` role.
4. Both EJB and JAX-WS clients may be in any role.

Answer:

Correct option is 3.

You can secure Web Services using the existing security infrastructure of Java EE Application Server, Java EE role-based security, and Secure Sockets Layer (SSL) transport level security.

Figure 5.1. SOAP message flow using existing security infrastructure



The Web Services port can be secured using Java EE role-based security. The Web Services sender sends the basic authentication data using the HTTP header. SSL (HTTPS) can be used to secure the transport. When the Java EE Application Server receives the SOAP message, the Web container authenticates the user (in this example, `user1`) and sets the security context for the call. After the security context is set, the SOAP router servlet sends the request to the implementation of the Web Services (the implementation can be JavaBeans or enterprise bean files). For enterprise bean implementations, the EJB container performs an authorization check against the identity of `user1`.

The Web Services port also can be secured using the Java EE role. Then, authorization is performed by the Web container before the SOAP request is dispatched to the Web Services implementation.

Sources:

Web Services Security and Java Platform, Enterprise Edition security relationship - [`http://publib.boulder.ibm.com/infocenter/wasinfo/beta/index.jsp?topic=/com.ibm.websphere.express.doc/info/exp/ae/cwbs_wssecurityj2ee.html`]

5.1.2.3. Configure caller authentication and message protection; either by Servlet container via `web.xml`, and/or by JAX-WS message processing runtime.

blah

5.1.3. Configure security requirements through deployment descriptor (`web.xml`) for JAX-RS based web service endpoint.

blah-blah

Chapter 6. Create a web service client for a SOAP based web service

6.1. Create a standalone client.

Question 060101

Which statements are true about `javax.xml.ws.Service` type?

Options (select 2):

1. It belongs to JAX-RS client-side API.
2. It belongs to JAX-WS client-side API.
3. It belongs to JAX-RS server-side API.
4. It belongs to JAX-WS server-side API.
5. It is a concrete class.
6. It is an abstract interface.
7. It is an annotation.

Answer:

Correct options are 2 and 5.

The `javax.xml.ws.Service` is a concrete class which extends `java.lang.Object`. `javax.xml.ws.Service` objects provide the client view of a Web Service.

`Service` acts as a factory of the following:

- Proxies for a target service endpoint.
- Instances of `Dispatch` for dynamic message-oriented invocation of a remote operation.

The ports available on a service can be enumerated using the `getPorts` method. Alternatively, you can pass a service endpoint interface to the unary `getPort` method and let the runtime select a compatible port:

```
// Specify the WSDL
URL wsdlLocation = new URL("http://localhost:8080/hello?wsdl");

// Create a Qualified Name that represents the namespace and local part of the service
QName serviceName = new QName("http://java.boot.by/", "HelloWSService");

//Create a proxy to get a port stub from
Service service = Service.create(wsdlLocation, serviceName);

// Return a list of QNames of ports
System.out.println("QNames of service endpoints:");
Iterator<QName> it = service.getPorts();
QName lastEndpoint = null;
while (it.hasNext()) {
    lastEndpoint = it.next();
    System.out.println("Name: " + lastEndpoint);
}

// Get the Hello stub
Hello hello = service.getPort(lastEndpoint, Hello.class);

// Invoke the business method
String result = hello.sayHello("Mikalai");
System.out.println("\nResponse: " + result);
```

Handler chains for all the objects created by a `Service` can be set by means of a `HandlerResolver`.

JAX-WS Service APIs, Client API, and Core APIs include the following types:

- Service APIs for service implementations:
  - `javax.xml.ws.Provider`
  - `javax.xml.ws.Endpoint`
  - `javax.xml.ws.WebServiceContext`
- Core APIs includes the following types that can be used both by the services and service clients:
  - `javax.xml.ws.Binding`
  - `javax.xml.ws.spi.Provider`
  - `javax.xml.ws.spi.ServiceDelegate`
  - Exceptions – `WebServiceException`, `ProtocolException` , `SOAPFaultException`, `HTTPException`.
- Client APIs to create proxies for remote service endpoints and dynamically construct operation invocations:
  - `javax.xml.ws.Service`
  - `javax.xml.ws.BindingProvider`
  - `javax.xml.ws.Dispatch`

Sources:

`javax.xml.ws.Service` JavaDoc - [`http://download.oracle.com/javase/6/docs/api/javax/xml/ws/Service.html`]

6.1.1. Use `wsimport` to generate artifacts.

blah-blah

6.1.2. Create a client application using these artifacts.

Question 06010201

Which of the following can developer achieve by using `BindingProvider` interface of JAX-WS Web Service client?

Options (select 2):

1. Specify user name and password for BASIC authentication.
2. Set custom `HandlerResolver` to configure programmatically handler chain on client.
3. Validate against XML Schema the outgoing SOAP message.
4. Specify user-defined Web Service endpoint HTTP address.
5. Validate against XML Schema the incoming SOAP message.

Answer:

Correct options are 1 and 4.

JAX-WS defines two service usage models:

- Proxy clients.

  In proxy-based client model, your applications work on local proxy objects that implement the SEI that is being exposed by the Web Service endpoint.

- Dispatch clients.

  The dispatch-client model offered by JAX-WS is a lower-level model that requires you to supply the necessary XML request yourself. This model can be used in the situations where you want to dynamically build up the SOAP request itself or where you must use a non-SOAP-based Web service endpoint.

Collectively, both client types are also known as BindingProviders because both clients realize the JAX-WS `javax.xml.ws.BindingProvider` interface.

Both proxy-based clients and dispatch-based clients share a common configuration model. The configuration is performed programmatically in the context of the Java Web service client code. By using this configuration, you can specify an explicit endpoint location, HTTP protocol session behavior, HTTP authentication credentials, and more.

The actual programmatic configuration is performed on the `javax.xml.ws.BindingProvider` client-side object. The dynamic proxies that are generated by the JAX-WS run time implement the `javax.xml.ws.BindingProvider` interface, where the `Dispatch` interface extends it. Therefore, dispatch objects produced by the JAX-WS Service class, by contract, also implement the `BindingProvider` behavior.

To configure the client `BindingProvider`, you add information to the request context, which is an ordinary `java.util.Map<String, Object>` that contains the actual configuration. The keys are strings and the values are objects. The map is available by using the `BindingProvider.getRequestContext()` method.

JAX-WS 2.1 specifies the following standard properties that can be used to configure the request context:

- `BindingProvider.ENDPOINT_ADDRESS_PROPERTY`:

  `javax.xml.ws.service.endpoint.adress` (value type: `String`)

  Target service endpoint address. The URI scheme for the endpoint address specification MUST correspond to the protocol/transport binding for the binding in use.

- `BindingProvider.USERNAME_PROPERTY`:

  `javax.xml.ws.security.auth.username` (value type: `String`)

  Specifies the user name in a set of HTTP basic authentication credentials.

- `BindingProvider.PASSWORD_PROPERTY`:

  `javax.xml.ws.security.auth.password` (value type: `String`)

  Specifies the password in a set of HTTP basic authentication credentials.

- `BindingProvider.SESSION_MAINTAIN_PROPERTY`:

  `javax.xml.ws.session.maintain` (value type: `Boolean`, default: `false`)

  This boolean property is used by a service client to indicate whether or not it wants to participate in a session with a service endpoint. If this property is set to `true`, the service client indicates that it wants the session to be maintained. If set to `false`, the session is not maintained. The default value for this property is `false`.

Example below illustrates how a client application casts the dynamic proxy object to a

`BindingProvider` and configures that object with an explicit endpoint location:

```
import javax.xml.ws.BindingProvider;
public class HelloClientCustomEndpoint {
    public static void main(String... args) throws Exception {
        HelloMessengerService service = new HelloMessengerService();
        HelloMessenger port = service.getHelloMessengerPort();
        ((BindingProvider)port).getRequestContext().put(
                    BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
                    "http://java.boot.by/Hello");
        String message = port.sayHello("Mikalai");
        System.out.println(message);
    }
}
```

Example below shows how to allow the Web Service client to participate in Web Service endpoint initiated sessions:

```
import javax.xml.ws.BindingProvider;
public class HelloClientHttpSession {
    public static void main(String... args) throws Exception {
        // Obtain the dynamic stub from the Service:
        HelloMessengerService service = new HelloMessengerService();
        HelloMessenger proxy = service.getHelloMessengerPort();

        // Cast the proxy to a BindingProvider:
        BindingProvider bindingProvider = (BindingProvider) proxy;

        // Get the request context
        Map<String, Object> requestContext = bindingProvider.getRequestContext();

        // Configure session preference
        requestContext.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);

        // Perform Web Service method invocation and print the result
        String message = proxy.sayHello("Volha");
        System.out.println(message);
    }
}
```

Knowing that all dynamic proxies also are `BindingProviders`, the application explicitly casts the proxy. It then obtains the request context map and specifies to the JAX-WS run time that it is willing to participate in server-side initiated session.

The request context map is available on the `BindingProvider` implementation:

```
Map<String,Object> getRequestContext()
```

It contains additional transport-level configuration options. These options are specific to the protocol binding that you use. If the application communicates by using SOAP/HTTP, for example, you have the option to configure HTTP headers in the request.

The `BindingProvider` implementation also exposes a response context map:

```
Map<String,Object> getResponseContext()
```

Similarly to the request context, this object gives you access to transport level data. By using SOAP/HTTP, you are most likely to find the HTTP response code inside the response context.

Option 2 is wrong. The programmatic handlers configuration works for all JAX-WS services, generated service interfaces, and the generic service. Handlers are added to the client-side service by using the following method:

```
Service.setHandlerResolver(HandlerResolver handlerResolver)
```

Example below shows a Web Service client that programmatically adds the `HelloMessengerProtocolHandler` to the client-side SEI:

```
import javax.xml.ws.handler.HandlerResolver;
import javax.xml.ws.handler.PortInfo;
public class HelloClientWithHandler {
    public static void main(String... args) throws Exception {
        HelloMessengerService service = new HelloMessengerService()
        service.setHandlerResolver(new HandlerResolver() {
            public List getHandlerChain(PortInfo inf) {
                return Collections.singletonList(new HelloMessengerProtocolHandler());
            }
        });
        HelloMessenger port = service.getHelloMessengerPort();
        String message = port.sayHello("Mikalai");
        System.out.println(message);
    }
}
```

Sources:

`javax.xml.ws.BindingProvider` JavaDoc - [`http://download.oracle.com/javase/6/docs/api/javax/xml/ws/BindingProvider.html`]

6.1.2.1. Invoke web service synchronously or asynchronously.

blah-blah

6.1.3. Package and deploy accordingly.

blah-blah

6.2. Create a client in a managed component in a EE container.

6.2.1. Use `wsimport` to generate artifacts.

blah-blah

6.2.2. Using `@WebserviceRef` in the client application.

Question 06020201

Developer is asked to write a Servlet that must act as the client for existing JAX-WS Web Service:

```
@WebService
public interface Catalog {
    @WebMethod
    String getTitle(String id);
}
```

```
@WebService(serviceName="CatalogService", name="Catalog")
public class CatalogWS implements Catalog {
    @WebMethod
```

```
    public String getTitle(String id)  {
        ...
    }
}
```

The developer decided to use the `@WebServiceRef` annotation to inject a reference to the service he needs to invoke. Which code snipped demonstrates correct usage of the `@WebServiceRef` annotation in JAX-WS container-managed client?

Options (select 3):

1.
```
@WebServlet(name="CatalogServlet", urlPatterns={"/CatalogServlet"})
public class CatalogServlet extends HttpServlet {

    @WebServiceRef(value=CatalogService.class)
    Catalog ref;
    ...
```

2.
```
@WebServlet(name="CatalogServlet", urlPatterns={"/CatalogServlet"})
public class CatalogServlet extends HttpServlet {

    @WebServiceRef(type=CatalogService.class)
    Catalog ref;
    ...
```

3.
```
@WebServlet(name="CatalogServlet", urlPatterns={"/CatalogServlet"})
public class CatalogServlet extends HttpServlet {

    @WebServiceRef(CatalogService.class)
    Catalog ref;
    ...
```

4.
```
@WebServlet(name="CatalogServlet", urlPatterns={"/CatalogServlet"})
public class CatalogServlet extends HttpServlet {

    @WebServiceRef
    CatalogService ref;
    ...
```

5.
```
@WebServlet(name="CatalogServlet", urlPatterns={"/CatalogServlet"})
public class CatalogServlet extends HttpServlet {

    @WebServiceRef
    CatalogWS ref;
    ...
```

6.
```
@WebServlet(name="CatalogServlet", urlPatterns={"/CatalogServlet"})
public class CatalogServlet extends HttpServlet {

    @WebServiceRef
    Catalog ref;
    ...
```

Answer:

Correct options are: 1, 3, and 4.

The `@WebServiceRef` annotation is used to declare a reference to a Web Service:

```
package javax.xml.ws;
@Target({TYPE, METHOD, FIELD}) @Retention(RUNTIME)
public @interface WebServiceRef {
     String name() default "";
     String wsdlLocation() default "";
     Class type default Object.class;
     Class value default Service.class; // If the reference type is an SEI, this value must be spe
     String mappedName() default "";
     String lookup() default "";
};
```

The `name()` attribute defines how the Web Service will be bound into the JNDI ENC. The `wsdlLocation()` attribute specifies the URL where the WSDL file lives (HTTP URLs are allowed). The `wsdlLocation` element, if present, overrides the WSDL location information specified in the `@WebService` annotation of the referenced generated service class. The `mappedName()` attribute is a vendor-specific global identifier for the Web Service. No use of a mapped name is portable. A defined reference can be resolved using a portable JNDI name provided by `lookup()` element. In this case, it is an error if there are any circular dependencies between entries of references. Similarly, it is an error if looking up the specified JNDI name results in a resource whose type is not compatible with the reference being created. Since this "lookup" functionality is just resolving to an already defined reference, only `name()` can be specified with `lookup()` (doesn't require any other metadata like `wsdlLocation`, etc.).

There are two uses to the `@WebServiceRef` annotation:

1. To define a reference whose type is a generated service class (the `CatalogService` in our example). In this case, the `type` and `value` element will both refer to the generated service class type. Moreover, if the reference type can be inferred by the field/method declaration the annotation is applied to, the type and value elements MAY have the default value (`Object.class`, that is). If the type cannot be inferred, then at least the `type()` element MUST be present with a non-default value.

   This makes option 4 correct.

2. To define a reference whose type is a SEI (the `Catalog` in our example). In this case, the `type` element MAY be present with its default value if the type of the reference can be inferred from the annotated field/method declaration, but the `value` element MUST always be present and refer to a generated service class (the `CatalogService`) type (a subtype of `javax.xml.ws.Service`).

   This makes options 1 and 3 correct (they are equivalent).

Here is fragment of generated JAX-WS `CatalogService` client-side class:

```
@WebServiceClient(name = "CatalogService",
        targetNamespace = "http://java.boot.by/",
        wsdlLocation = "http://localhost:8080/CatalogProject/CatalogService?wsdl")
public class CatalogService extends Service {
    ...
    @WebEndpoint(name = "CatalogPort")
    public Catalog getCatalogPort() {
        return super.getPort(new QName("http://java.boot.by/", "CatalogPort"), Catalog.class);
    }

    @WebEndpoint(name = "CatalogPort")
    public Catalog getCatalogPort(WebServiceFeature... features) {
        return super.getPort(new QName("http://java.boot.by/", "CatalogPort"), Catalog.class, feat
    }
    ...
}
```

Option 2 is wrong. We inject SEI type, not Service type, and the `WebServiceRef.value()` attribute

is missing.

Option 5 is wrong. We inject service implementation class, which is not available on the client. The code will not compile.

Option 6 is wrong. We inject SEI type, not Service type, and the `WebServiceRef.value()` attribute is missing. We must provide information about generated service class type (a subtype of `javax.xml.ws.Service`).

Sources:

The Java API for XML-Based Web Services (JAX-WS) 2.2 - [Section 7.9 `javax.xml.ws.WebServiceRef`] - [`http://www.jcp.org/en/jsr/detail?id=224`]

6.2.3. Package and deploy accordingly.

blah-blah

Chapter 7. Create a web service client for a RESTful web service

7.1. Use a browser to access a JAX-RS resource

Question 070101

What HTTP methods are typically used by RESTful Web Services?

Options (select 4):

1. `OPTIONS`
2. `GET`
3. `HEAD`
4. `POST`
5. `PUT`
6. `DELETE`
7. `TRACE`
8. `CONNECT`

Answer:

Correct options are: 2, 4, 5, and 6.

A RESTful Web Service is a simple Web Service implemented using HTTP and the principles of REST. It is a collection of resources, with three defined aspects:

- The base URI for the Web Service, such as `http://java.boot.by/resources/`
- The Internet media type of the data supported by the Web Service. This is often JSON, XML or any other valid Internet media type.
- The set of operations supported by the Web Service using HTTP methods (e.g., POST, GET, PUT or DELETE).

The following table shows how the HTTP verbs are typically used to implement a Web Service:

Table 7.1. RESTful Web Service HTTP methods

Resource	GET	PUT	POST	DELETE

Resource	GET	PUT	POST	DELETE
Collection URI, such as `http://java.boot.by/resources/`	List the URIs and perhaps other details of the collection's members.	Replace the entire collection with another collection.	Create a new entry in the collection. The new entry's URL is assigned automatically and is usually returned by the operation.	Delete the entire collection.
Element URI, such as `http://java.boot.by/resources/142`	Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	Replace the addressed member of the collection, or if it doesn't exist, create it.	Treat the addressed member as a collection in its own right and create a new entry in it.	Delete the addressed member of the collection.

Methods PUT and DELETE are defined to be idempotent, meaning that multiple identical requests should have the same effect as a single request. Methods GET, HEAD, OPTIONS and TRACE, being prescribed as safe, should also be idempotent, as HTTP is a stateless protocol.

In contrast, the POST method is not necessarily idempotent, and therefore sending an identical POST request multiple times may further affect state or cause further side effects (such as financial transactions). In some cases this may be desirable, but in other cases this could be due to an accident, such as when a user does not realize that their action will result in sending another request, or they did not receive adequate feedback that their first request was successful. While web browsers may show alert dialog boxes to warn users in some cases where reloading a page may re-submit a POST request, it is generally up to the web application to handle cases where a POST request should not be submitted more than once.

Option 1 is wrong. The OPTIONS method represents a request for information about the communication options available on the request/response chain identified by the `Request-URI`. This method allows the client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating a resource retrieval.

Option 7 is wrong. The TRACE method is used to invoke a remote, application-layer loopback of the request message. The final recipient of the request SHOULD reflect the message received back to the client as the entity-body of a 200 (OK) response. The TRACE method allows the client to see what is being received at the other end of the request chain and use that data for testing or diagnostic information.

Option 8 is wrong. HTTP 1.1 specification reserves the method name CONNECT for use with a proxy that can dynamically switch to being a tunnel (e.g. SSL tunneling).

Sources:

RESTful Web Services by Leonard Richardson, Sam Ruby (O'Reilly Media, 2007) `[http://oreilly.com/catalog/9780596529260/]`

HTTP 1.1 Method Definitions `[http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html]`

7.2. Use the `java.net.*` APIs to access a JAX-RS resource.

Question 070201

You are hired by Snorcle, Inc. to develop a REST Web Service client which retrieves information about Employee by his ID. The Web Service accepts HTTP POST requests. What code should be added at the line #5 to perform Employee information request? Assume that the rest of the code is valid.

```
1
```

```
 2 ...
 3 URL url = new URL("http://snorcle.com/hr/employee");
 4 URLConnection connection = url.openConnection();
 5 // add new code here if needed
 6
 7 OutputStream os = connection.getOutputStream();
 8 OutputStreamWriter out = new OutputStreamWriter(os);
 9 out.write("id=" + id);
10 out.close();
11 ...
12
```

Options (select 1):

1. `connection.setRequestProperty("Method", "POST");`
2. `connection.setAllowUserInteraction(true);`
3. `connection.setDoOutput(true);`
4. `connection.setMethod("POST");`
5. `connection.setHttpMethod("POST");`
6. You don't need to add anything. The shown client code will send POST HTTP request by default.

Answer:

Correct option is 3. Setting the `URLConnection.setDoOutput()` to `true` implicitly sets the request method to POST.

Option 1 is wrong. The method `connection.setRequestProperty(...)` method adds a field to the header of the `URLConnection` with a specified name and value.

Option 2 is wrong. The `setAllowUserInteraction(true)` is a hint from one application to another that user interaction is OK (i.e. from HotJava to a lower-level module).

Options 4 and 5 are wrong. The methods names are invalid.

Option 6 is wrong. The default request method is GET.

Sources:

Java Tutorial. Working with URLs - `[http://download.oracle.com/javase/tutorial/networking/urls/readingWriting.html]`

Question 070202

You are hired by Snorcle, Inc. to develop a REST Web Service client which retrieves information about Employee by her ID. What code should be added at the line #5 to perform Employee information request? Assume that the rest of the code is valid.

```
 1
 2 ...
 3 URL url = new URL("http://snorcle.com/hr/employee/" + id);
 4 URLConnection connection = url.openConnection();
 5 // add new code here if needed
 6
 7 InputStream is = connection.getInputStream();
```

```
 8 InputStreamReader isr = new InputStreamReader(is);
 9 BufferedReader in = new BufferedReader(isr);
10 ...
11
```

Options (select 2):

1. ```
   connection.setAllowUserInteraction(true);
   ```

2. ```
   connection.setDoInput(true);
   ```

3. ```
   connection.setAllowInput(true);
   ```

4. You don't need to add anything. The shown client code will be able to read HTTP response by default.

Answer:

Correct options are 2 and 4. A URL connection can be used for input and/or output. Set the `doInput` flag to `true` if you intend to use the URL connection for input, `false` if not. The default is `true` (this means client will read response as it is shown in the question, without adding any code).

Option 1 is wrong. The `setAllowUserInteraction(true)` is a hint from one application to another that user interaction is OK (i.e. from HotJava to a lower-level module).

Option 4 is wrong. The method name is invalid.

Sources:

`java.net.URLConnection` JavaDoc - `[http://download.oracle.com/javase/6/docs/api/java/net/URLConnection.html]`

7.3. Use `java.net.Authenticator` to access a secure JAX-RS resource.

blah-blah

7.4. Use Ajax to access a JAX-RS resource.

Question 070401

Snorcle, Inc. is refactoring its own intranet web site. Employees can retrieve and browse company customers in a web browser. Previously, customer information was retrieved in a separate window when user clicked the link for each customer. Now, the customer information must be generated by a Web Service deployed in the intranet, and must be displayed in the already opened window without reload. Which client-side technologies can be used for easiest refactoring?

Options (select 3):

1. JAXP
2. SAAJ
3. REST
4. XML
5. JSON
6. JavaScript

Answer:
   ```

Correct options are 3, 5, and 6.

Option 1 is wrong. The Java API for XML Processing (JAXP) enables applications to parse, transform, validate and query XML documents using an API that is independent of a particular XML processor implementation. JAXP provides a pluggability layer to enable vendors to provide their own implementations without introducing dependencies in application code. JAXP requires Java Virtual Machine, it could be used within web browser by Java Applet, but this would not be the easiest refactoring.

Option 2 is wrong. The SOAP with Attachments API for Java or SAAJ provides a standard way to send XML documents over the Internet from the Java platform. SAAJ enables developers to produce and consume messages conforming to the SOAP 1.1 specification and SOAP with Attachments note. Using SOAP would not be the easiest refactoring.

Option 3 is correct. Representational State Transfer (REST) is a style of software architecture for distributed hypermedia systems such as the World Wide Web. REST-style architectures consist of clients and servers. Clients initiate requests to servers; servers process requests and return appropriate responses. Requests and responses are built around the transfer of representations of resources. A resource can be essentially any coherent and meaningful concept that may be addressed. A representation of a resource is typically a document that captures the current or intended state of a resource. It can be JSON, XML, plain text or YAML but can be any other valid Internet media type.

Option 4 is wrong. Extensible Markup Language (XML) is a set of rules for encoding documents in machine-readable form. XML is too verbose and complex for this refactoring.

Option 5 is correct. JSON (JavaScript Object Notation) is a lightweight text-based open standard designed for human-readable data interchange. It is derived from the JavaScript programming language for representing simple data structures and associative arrays, called objects. Despite its relationship to JavaScript, it is language-independent, with parsers available for most programming languages. The JSON format is often used for serializing and transmitting structured data over a network connection. It is primarily used to transmit data between a server and web application, serving as an alternative to XML.

Since JSON is a subset of JavaScript it is possible (but not recommended) to parse the JSON text into an object by invoking JavaScript's `eval()` function. For example, if the above JSON data is contained within a JavaScript string variable `contact`, one could use it to create the JavaScript object `customer` like so:

```
var customer = eval("(" + contact + ")");
```

The `contact` variable must be wrapped in parentheses to avoid an ambiguity in JavaScript's syntax.

The recommended way, however, is to use a JSON parser. Unless a client absolutely trusts the source of the text, or must parse and accept text which is not strictly JSON-compliant, one should avoid `eval()`. A correctly implemented JSON parser will accept only valid JSON, preventing potentially malicious code from running.

Modern browsers, such as Firefox 3.5 and Internet Explorer 8, include special features for parsing JSON. As native browser support is more efficient and secure than `eval()`, it is expected that native JSON support will be included in the next ECMAScript standard.

Option 6 is correct. JavaScript, also known as ECMAScript is a prototype-based object-oriented scripting language that is dynamic, weakly typed and has first-class functions. Because JavaScript code can run locally in a user's browser (rather than on a remote server), the browser can respond to user actions quickly, making an application more responsive. Furthermore, JavaScript code can detect user actions which HTML alone cannot, such as individual keystrokes. The wider trend of Ajax programming similarly exploits this strength.

Ajax is not one technology, but a group of technologies. Ajax uses a combination of HTML and CSS to mark up and style information. The DOM is accessed with JavaScript to dynamically display, and to

allow the user to interact with the information presented. JavaScript and the `XMLHttpRequest` object provide a method for exchanging data asynchronously between browser and server to avoid full page reloads.

Here is an example of combination of REST, JSON and JavaScript (AJAX).

1. Create a simple RESTFul Web Service which returns JSON content:

```
package by.boot.java;

import javax.ws.rs.Path;
import javax.ws.rs.GET;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;

@Path("/client/{id}")
public class ClientResource {

    @GET
    @Produces("application/json")
    public String getJson(@PathParam("id") int clientNo) {
        switch (clientNo) {
            case 1:
                return "{'name':'Mikalai Zaikin', 'age':36}";
            case 2:
                return "{'name':'Volha Zaikina', 'age':36}";
            default:
                return "{'name':'Anonymous', 'age':100}";
        }
    }
}
```

2. Configure Web Deployment Descriptor (`WEB-INF/web.xml`):

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="3.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/jav

    <servlet>
        <servlet-name>ServletAdaptor</servlet-name>
        <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>ServletAdaptor</servlet-name>
        <url-pattern>/resources/*</url-pattern>
    </servlet-mapping>

</web-app>
```

3. Create client - HTML page with JavaScript:

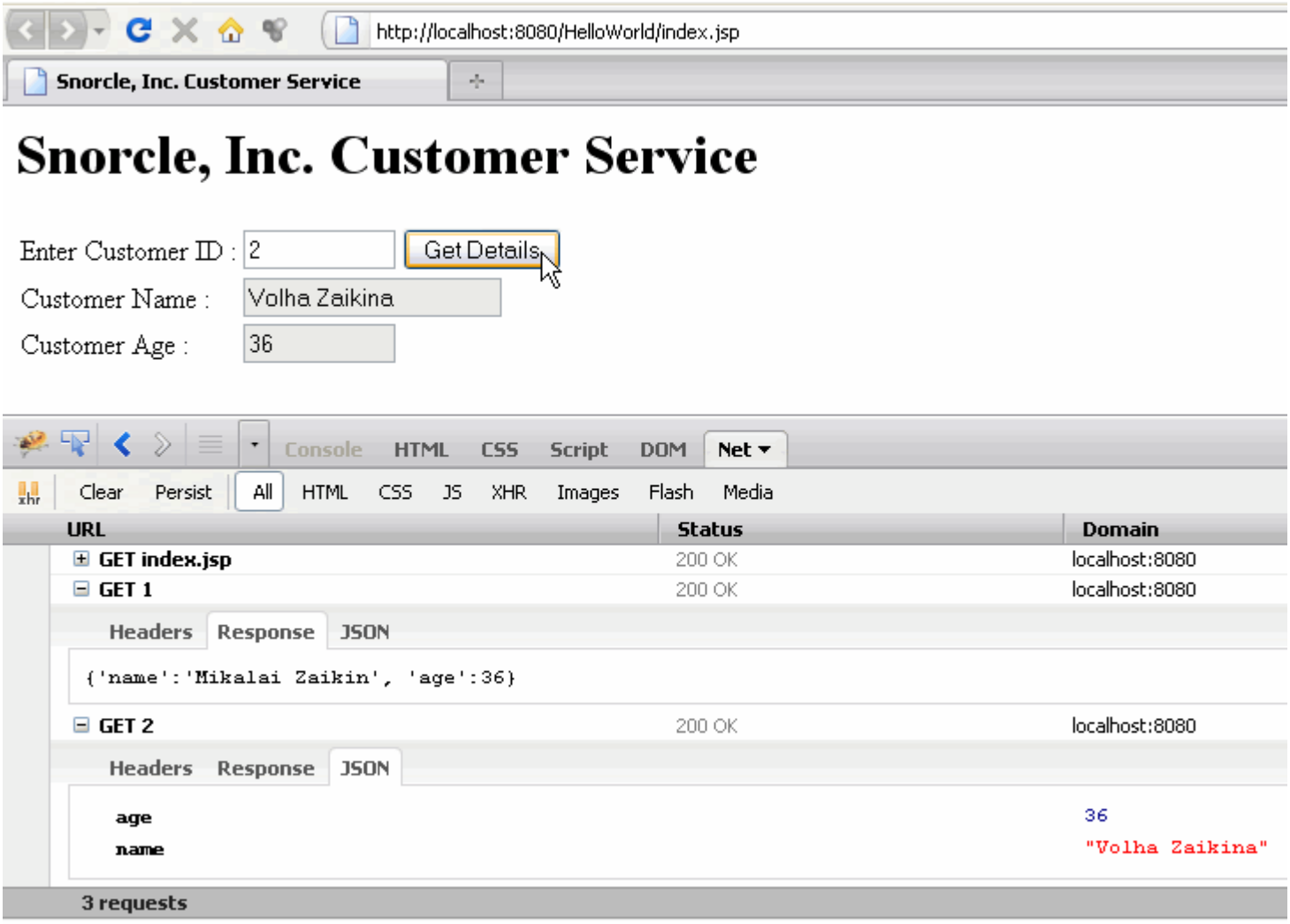
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>Snorcle, Inc. Customer Service</title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <script type="text/javascript"  language="JavaScript">
            var xmlhttp;

            function init() {
                xmlhttp = new XMLHttpRequest();
            }
```

```
            function getDetails() {
                var custno = document.getElementById("custno");
                var url = "http://localhost:8080/HelloWorld/resources/client/" + custno.value
                xmlhttp.open('GET', url, true);
                xmlhttp.send(null);
                xmlhttp.onreadystatechange = function() {

                    var custname =  document.getElementById("custname");
                    var age =  document.getElementById("age");
                    if (xmlhttp.readyState == 4) {
                        if ( xmlhttp.status == 200) {
                            var det = eval( "(" +  xmlhttp.responseText + ")");
                            if (det.age > 0 ) {
                                custname.value = det.name;
                                age.value = det.age;
                            } else {
                                custname.value = "";
                                age.value ="";
                                alert("Invalid Customer ID");
                            }
                        }
                        else {
                            alert("Error: " + xmlhttp.responseText);
                        }
                    }
                };
            }
        </script>
    </head>
    <body onload="init()">
        <h1>Snorcle, Inc. Customer Service</h1>
        <table>
            <tr>
                <td>Enter Customer ID :  </td>
                <td><input type="text" id="custno" size="10"/>
                <input type="button" value="Get Details" onclick="getDetails()"/>
            </tr>
            <tr>
                <td>Customer Name :  </td>
                <td><input type="text" readonly="true" id="custname" size="20"/> </td>
            </tr>
            <tr>
                <td>Customer Age : </td>
                <td><input type="text" readonly="true" id="age" size="10"/> </td>
            </tr>
        </table>
    </body>
</html>
```

Results:



Sources:

Real World REST Using Jersey, AJAX and JSON - [http://www.developer.com/print.php/3841046]

7.5. Use the Jersey client API to access a JAX-RS resource.

blah-blah

7.6. Use the JAX-WS HTTP binding to access a JAX-RS resource.

blah-blah

Chapter 8. Create a SOAP based web service using Java SE platform.

8.1. Create a web service starting from a WSDL file using JAX-WS.

Question 080101

You are hired by Snorcle, Inc. to develop a Stock Broker (SB) Web Service which will be consumed by partner companies. Clients use different platforms (JEE, .NET). In order to reduce load of the Web Service clients will be asked to implement client-side validation of requests. Which approach should you use?

Options (select 1):

1. Code-first ("Java-to-WSDL") approach.
2. Contract-first ("WSDL-to-Java") approach.
3. Meet in the middle approach.

4. None of the above.

Answer:

Correct option is 2.

If you start from Java SEI and the service changes, the WSDL automatically changes. The WSDL thereby loses some of its appeal for creating client artifacts, as this process may have to be repeated over and over. One of the first principles of software development is that an interface, once published, should be regarded as immutable so that code once written against a published interface never has to be rewritten. Also, the code-first approach seems to go against the language-neutral theme of SOAP-based Web Services.

Option 2 is correct.

The contract-first ("WSDL-to-Java") development approach is most suitable when you build a Web Service from scratch and the Web Service elements are the fundamental component model. You should also use this development approach when your primary focus is on client-service interoperability. Working from WSDL provides the following advantages:

- "WSDL-to-Java" approach is better suited to situations in which the service developer is not the author of the service semantics and workflow, such as when implementing to a certain industry's convention for Web Services.
- The strong typing is shared using the WSDL file itself. Starting from WSDL and XML Schema types puts strong types where they can be shared by services and clients.
- "WSDL-to-Java" approach offers you the best interoperability because it allows you to author the WSDL file directly and gives you control over data type and encoding choices.
- Serializable types are expressed first in XML Schema, and then mapped to one or more programming languages and object models.
- Client-side validation is easier to implement.

Sources:

Contract-First Web Services: 6 Reasons to Start with WSDL and Schema - `[http://soa.sys-con.com/node/143909]`

8.1.1. Use `wsimport` tool to generate artifacts and use customization files for wsimports if needed.

blah-blah

8.1.2. Build the web service implementation using the above artifacts.

blah-blah

8.1.3. Use `Endpoint` API to configure and deploy it in Java SE 6 platform.

blah-blah

8.2. Create a web service starting from a Java source using JAX-WS.

Question 080201

You are hired by Snorcle, Inc. to develop an order-tracking Web Service which will be consumed by partner companies. Java team from another department would like to reuse your code as soon as possible. Which approach should you choose?

Options (select 1):

1. Code-first ("Java-to-WSDL") approach.
2. Contract-first ("WSDL-to-Java") approach.
3. Meet in the middle approach.
4. None of the above.

Answer:

Correct option is 1.

It is most convenient to define a new Web Service by starting from a Java class when:

- When creating a Web Service that exposes the functionality of an existing application, or creating one from scratch.
- When you start with a suite of application components, generate the necessary Web Service descriptive and infrastructure files based on the functionality provided by the existing components.
- When you need the quickest development path and achieve the shortest time to market.

  This makes the option 1 correct.

Sources:

Designing Web Services with the J2EE 1.4 Platform - `[http://java.sun.com/blueprints/guidelines/designing_webservices/]`

8.2.1. Use `wsgen` tool to generate artifacts in Java EE5 (optional in Java EE6 - as artifacts are generated at run time)

blah-blah

8.2.2. Use `Endpoint` API to configure and deploy it in Java SE 6 platform.

blah-blah

Chapter 9. Create handlers for SOAP web services.

9.1. Configure SOAP and logical handlers on the server side.

Question 090101

What statements are true about the intermediary node when processing SOAP message:

Options (select 2):

1. If `actor` attribute of SOAP header identifies the intermediary node, the node must not forward that header element.
2. An intermediary node must forward SOAP header element if its `actor` attribute identifies the node.
3. An intermediary node may forward SOAP header element if its `actor` attribute identifies the node.
4. An intermediary node may insert new header elements in SOAP header.
5. An intermediary node may insert new elements in SOAP body.

Answer:

Correct options are 1 and 4.

A SOAP message travels from the originator to the ultimate destination, potentially by passing through a set of SOAP intermediaries along the message path. A SOAP intermediary is an application that is capable of both receiving and forwarding SOAP messages. Both intermediaries as well as the ultimate destination are identified by a URI.

Not all parts of a SOAP message may be intended for the ultimate destination of the SOAP message but, instead, may be intended for one or more of the intermediaries on the message path. The role of a recipient of a header element is similar to that of accepting a contract in that it cannot be extended beyond the recipient. That is, a recipient receiving a header element MUST NOT forward that header element to the next application in the SOAP message path. The recipient MAY insert a similar header element but in that case, the contract is between that application and the recipient of that header element.

Omitting the SOAP `actor` attribute indicates that the recipient is the ultimate destination of the SOAP message.

Option 2 is wrong. When a node processes a header block, it must remove it from the SOAP message.

Option 3 is wrong. The node may add new header blocks to the SOAP message. SOAP nodes frequently feign removal of a header block by simply modifying it, which is logically the same as removing it, modifying it, and then adding it back to the SOAP message.

Option 5 is wrong. Intermediaries in a SOAP message path must not modify the application-specific contents of the SOAP `Body` element, but they may, and often do, manipulate the SOAP header blocks.

Sources:

Simple Object Access Protocol (SOAP) 1.1 - [`http://www.w3.org/TR/2000/NOTE-SOAP-20000508/`]

9.1.1. Use `@HandlerChain` annotation.

Question 09010A01

Given the Web Service code:

```
@WebService(name = "Handler", targetNamespace = "http://java.boot.by")
@HandlerChain(file="handler-chain.xml")
public class HandlerWS {

    @Resource
    WebServiceContext ctx;

    @WebMethod()
    public String getProperty(String propertyName) {
        return (String) ctx.getMessageContext().get(propertyName);
    }
}
```

and the `handler-chain.xml` configuration file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<handler-chains xmlns="http://java.sun.com/xml/ns/javaee">
    <handler-chain>
        <handler>
            <handler-class>
                LogicalHandler1
            </handler-class>
```

```
        </handler>
        <handler>
            <handler-class>
                SOAPHandler2
            </handler-class>
        </handler>
    </handler-chain>
    <handler-chain>
        <handler>
            <handler-class>
                SOAPHandler3
            </handler-class>
        </handler>
    </handler-chain>
    <handler-chain>
        <handler>
            <handler-class>
                LogicalHandler4
            </handler-class>
        </handler>
    </handler-chain>
</handler-chains>
```

Which statement is true about incoming SOAP message processing?

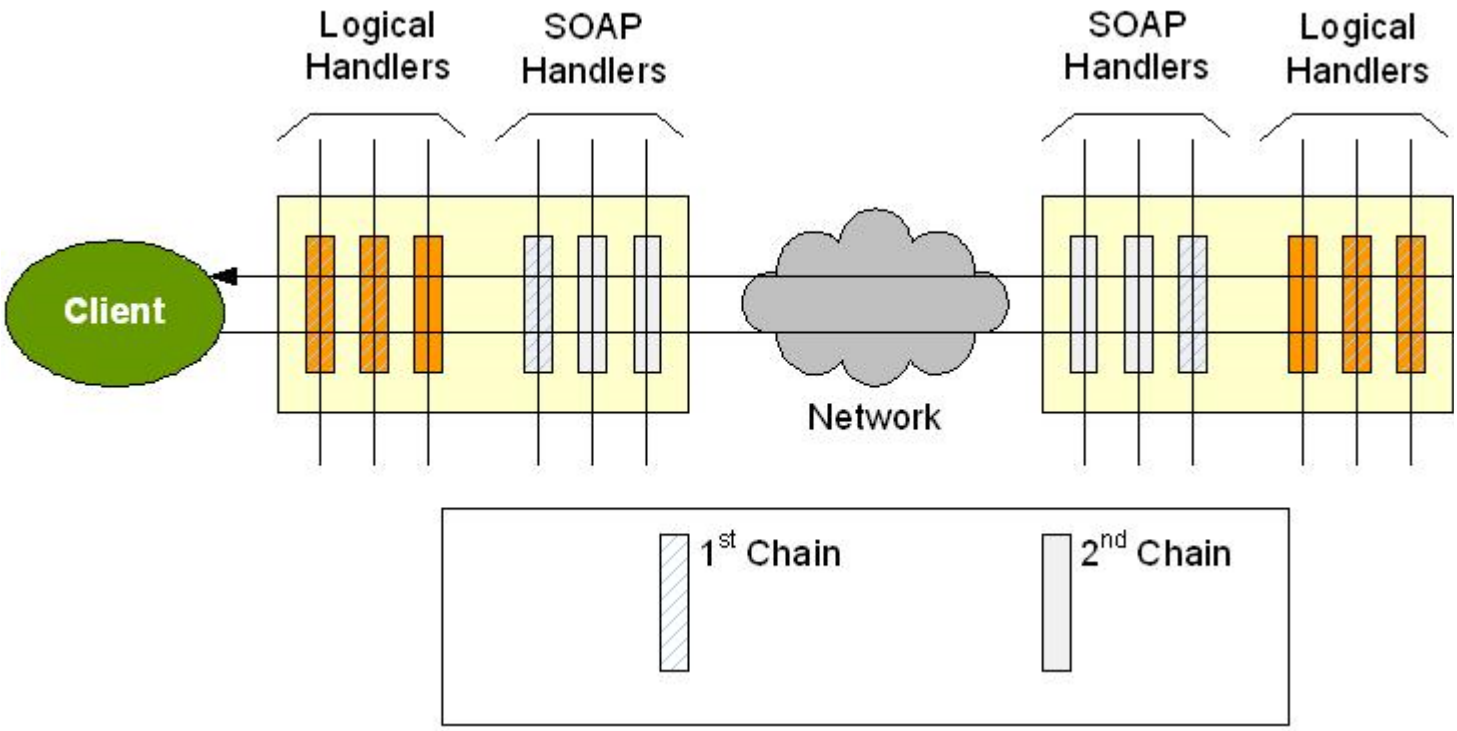
Options (select 1):

1. Handlers executed in the following order: `LogicalHandler1`, `SOAPHandler2`, `SOAPHandler3`, `LogicalHandler4`.
2. Handlers executed in the following order: `LogicalHandler1`, `LogicalHandler4`, `SOAPHandler2`, `SOAPHandler3`.
3. Handlers executed in the following order: `SOAPHandler2`, `SOAPHandler3`, `LogicalHandler1`, `LogicalHandler4`.
4. `RuntimeException` thrown at runtime, because only one handler chain is allowed for a single Service Implementation Bean.
5. None of the above.

Answer:

Correct option is 5. Correct order of handlers invocation for **incoming** SOAP message will be as follows: `SOAPHandler3`, `SOAPHandler2`, `LogicalHandler4`, `LogicalHandler1`.

Option 1 is wrong. For incoming SOAP messages, first SOAP (protocol) handlers will be invoked, and then logical handlers will be invoked. See the figure:



Option 2 is wrong. This order would be true for **outgoing** SOAP message.

Option 3 is wrong. For outbound messages handler processing starts with the first handler in the chain and proceeds in the same order as the handler chain. For inbound messages the order of processing is reversed: processing starts with the **last handler** in the chain and proceeds in the reverse order of the handler chain. E.g., consider a handler chain that consists of six handlers H1...H6 in that order: for outbound messages handler H1 would be invoked first followed by H2, H3, ... , and finally handler H6; for inbound messages H6 would be invoked first followed by H5, H4, ... , and finally H1.

Option 4 is wrong. In a handler-chain descriptor, there may be multiple chains, and in each of them, logical handlers may coexist with protocol handlers. Here is the behavior defined in the JAX-WS specification and as implemented in the reference implementation:

- Whether there are multiple handler chains or just one, all logical handlers are executed before protocol handlers on an outbound message, and all protocol handlers are executed before logical handlers on an inbound message.
- The reordering process in the first rule honors the relative positions of protocol handlers or logical handlers configured inside any handler chain on an outbound message; for an inbound message, the order is reversed.
- If multiple handler chains are configured, handlers (of the same kind) from chains in the front of the descriptor apply before handlers from those in the back on an outbound message, and handlers (of the same kind) from chains in the back of the descriptor apply before handlers from chains in the front on an inbound message.

Sources:

The Java API for XML-Based Web Services (JAX-WS) 2.2 - Section 9.3.2 (Handler Execution) - [http://www.jcp.org/en/jsr/detail?id=224]

Get a handle on the JAX-WS API's handler framework - [http://www.javaworld.com/javaworld/jw-02-2007/jw-02-handler.html]

9.1.2. Use deployment descriptors.

Question 09010B01

Which approach can be used to configure JAX-WS handlers chain for Web Service?

Options (select 1):

1. Automatically load all handlers from the CLASSPATH by using in `web.xml`:

```
<servlet>
    <servlet-name>javax.ws.rs.core.Application</servlet-name>
    <load-on-startup>1</load-on-startup>
</servlet>
```

2. By using `Binding.setHandlerChain(handlerList)` method.
3. By using `Dispatch.setHandlerResolver(myHandlerResolver)` method.
4. By using `@SOAPMessageHandlers` annotation on service implementation bean.
5. None of the above.

Answer:

Correct option is 5. None of these approaches can be used to configure server-side handler chain.

Option 1 is wrong. This `web.xml` deployment descriptor fragment is used to automatically load all root resource classes in JAX-RS application.

Option 2 is wrong. This approach can be used only to configure client-side handler chain.

Option 3 is wrong. This approach can be used only to configure client-side handler chain.

Option 4 is wrong. The `javax.jws.soap.SOAPMessageHandlers` annotation is deprecated as of JSR-181 2.0 with no replacement. Instead, `@HandlerChain` should be used.

In code-first service development starting from a service-endpoint implementation or a service endpoint interface (SEI), handler chains are configured with the `@HandlerChain` annotation defined in Java Specification Request 181, Web Services Metadata for the Java Platform. For example, we may use the following for `CreditCardService.java` and, from here, develop the Web Service with a handler chain configured:

```
@WebService
@HandlerChain(file="handler-chain.xml")
public class CreditCardService {
    ...
}
```

The `handler-chain.xml` referenced in the annotation is the handler chains' descriptor, and this is an example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<handler-chains xmlns="http://java.sun.com/xml/ns/javaee">
    <handler-chain>
        <handler>
            <handler-class>
                ServiceAuthenticationSOAPHandler
            </handler-class>
        </handler>
        <handler>
            <handler-class>
                ServicePerformanceMonitorLogicalHandler
            </handler-class>
```

```
        </handler>
    </handler-chain>
</handler-chains>
```

NOTE: JAX-WS only defines APIs for programmatic configuration of client side handler chains – server side handler chains are expected to be configured using deployment metadata.

Sources:

The Java API for XML-Based Web Services (JAX-WS) 2.2 - Section 9.2 (Configuration) - [http://www.jcp.org/en/jsr/detail?id=224]

Get a handle on the JAX-WS API's handler framework - [http://www.javaworld.com/javaworld/jw-02-2007/jw-02-handler.html]

Question 09010B02

Snorcle, Inc. asked you to add protocol handler which performs incoming SOAP 1.1 messages audit to the existing Web Services implementations. Which XML handler chain fragment can be used for this?

Options (select 1):

1.
```
<handler-chain>
    <protocol-bindings>##SOAP11_HTTP</protocol-bindings>
    <handler>
        <handler-name>AuditHandler</handler-name>
        <handler-class>server.AuditHandler</handler-class>
    </handler>
</handler-chain>
```

2.
```
<handler-chain>
    <protocol-pattern>##SOAP11_HTTP</protocol-pattern>
    <handler>
        <handler-name>AuditHandler</handler-name>
        <handler-class>server.AuditHandler</handler-class>
    </handler>
</handler-chain>
```

3.
```
<handler-chain>
    <protocol-patterns>##SOAP*</protocol-patterns>
    <handler>
        <handler-name>AuditHandler</handler-name>
        <handler-class>server.AuditHandler</handler-class>
    </handler>
</handler-chain>
```

4. None of the above.

Answer:

Correct option is 1.

To allow selection of handler chains based on service, port, and protocol binding in operation for a message exchange at the service runtime, we must specify constraints for handler chains in the

handler chains' descriptor (or the one embedded in the binding declaration for WSDL customization).

There are three such elements we can specify as sub-elements for `handler-chain`: `service-name-pattern`, `port-name-pattern`, or `protocol-bindings`, respectively restricting the chain defined by the parent element to apply for services and ports with certain name patterns, or for a specific binding protocol. If none of these elements are specified within the handler-chain element, the handlers specified in the chain are applied to everything.

Here is an example of a binding declaration file customizing WSDL for handler chains, with SOAP 1.1 protocol constraints:

```
<handler-chain>
    <protocol-bindings>##SOAP11_HTTP</protocol-bindings>
    <handler>
        <handler-name>AuditHandler</handler-name>
        <handler-class>server.AuditHandler</handler-class>
    </handler>
</handler-chain>
```

The `protocol-bindings` element requires a space-delimited list of protocol binding URIs or, for the standard binding types, their aliases. The following table lists the URIs of the standard binding types in JAX-WS and their corresponding aliases:

Table 9.1. Binding aliases for Message Bindings

URI	Alias
http://schemas.xmlsoap.org /wsdl/soap/http	##SOAP11_HTTP
http://schemas.xmlsoap.org /wsdl/soap/http?mtom=true	##SOAP11_HTTP_MTOM
http://www.w3.org/2003/05 /soap/bindings/HTTP/	##SOAP12_HTTP
http://www.w3.org/2003/05 /soap/bindings/HTTP/?mtom=true	##SOAP12_HTTP_MTOM
http://www.w3.org/2004/08 /wsdl/http	##XML_HTTP

Options 2 and 3 are wrong. Elements names (`protocol-pattern` and `protocol-patterns`) are wrong.

Sources:

Get a handle on the JAX-WS API's handler framework - [`http://www.javaworld.com/javaworld /jw-02-2007/jw-02-handler.html`]

Java API for XML Web Services (JAX-WS) Handler Framework - [`http://jax-ws.java.net /jax-ws-20-fcs/docs/handlers.html`]

Question 09010B03

You are asked to customize WSDL of the existing Web Service. The customization should allow logging handler chain to process only those SOAP messages, which have `TransactionID` header element. How this can be achieved?

Options (select 1):

1.

```
<bindings node="wsdl:definitions">
    <javaee:handler-chains xmlns:javaee="http://java.sun.com/xml/ns/javaee">
        <javaee:handler-chain>
            <javaee:header-name-pattern xmlns:ns1="http://java.boot.by/">
                ns1:TransactionID
            </javaee:header-name-pattern>
            <javaee:handler>
                <javaee:handler-class>
                    by.boot.java.SecuritySOAPHandler
                </javaee:handler-class>
            </javaee:handler>
        </javaee:handler-chain>
    </javaee:handler-chains>
</bindings>
```

2.

```
<bindings node="wsdl:definitions">
    <javaee:handler-chains xmlns:javaee="http://java.sun.com/xml/ns/javaee">
        <javaee:handler-chain>
            <javaee:header-name xmlns:ns1="http://java.boot.by/">
                ns1:TransactionID
            </javaee:header-name>
            <javaee:handler>
                <javaee:handler-class>
                    by.boot.java.SecuritySOAPHandler
                </javaee:handler-class>
            </javaee:handler>
        </javaee:handler-chain>
    </javaee:handler-chains>
</bindings>
```

3.

```
<bindings node="wsdl:definitions">
    <javaee:handler-chains xmlns:javaee="http://java.sun.com/xml/ns/javaee">
        <javaee:handler-chain>
            <javaee:header-name-pattern xmlns:ns1="http://java.boot.by/">
                ns1:TransactionID*
            </javaee:header-name-pattern>
            <javaee:handler>
                <javaee:handler-class>
                    by.boot.java.SecuritySOAPHandler
                </javaee:handler-class>
            </javaee:handler>
        </javaee:handler-chain>
    </javaee:handler-chains>
</bindings>
```

4. None of the above.

Answer:

Correct option is 4. It is not possible to do with binding declaration file customizing WSDL for handler chains.

You can restrict handler chain to apply for:

1. **services** with certain name patterns by using `javaee:service-name-pattern` element.
2. **ports** with certain name patterns by using `javaee:port-name-pattern` element.
3. specific binding **protocol** by using `javaee:protocol-bindings` element.

Both `javaee:service-name-pattern` and `javaee:port-name-pattern` need qualified names, and

the namespaces must be declared in the containing context. Also, you can use the wild card `*` in the names.

If none of these elements are specified within the `handler-chain` element, the handlers specified in the chain are applied to everything.

The following example demonstrates the `protocol-bindings`, `port-name-pattern`, and `service-name-pattern` elements that are used to restrict which services can apply the handlers:

```
<jws:handler-chains xmlns:jws="http://java.sun.com/xml/ns/javaee">
    <!-- NOTE: The '*' denotes a wildcard -->
    <jws:handler-chain>
        <jws:protocol-bindings>##SOAP11_HTTP</jws:protocol-bindings>
        <jws:port-name-pattern xmlns:ns1="http://java.boot.by/">
            ns1:MySampleP*
        </jws:port-name-pattern>
        <jws:service-name-pattern xmlns:ns1="http://java.boot.by/">
            ns1:MySampleS*
        </jws:service-name-pattern>
        <jws:handler>
            <jws:handler-class>
                by.boot.java.SampleLogicalHandler
            </jws:handler-class>
        </jws:handler>
        <jws:handler>
            <jws:handler-class>
                by.boot.java.SampleProtocolHandler
            </jws:handler-class>
        </jws:handler>
    </jws:handler-chain>
</jws:handler-chains>
```

Options 1, 2 and 3 are wrong. The `header-name-pattern` and `header-name` elements are invalid sub-elements of the `handler-chain` element.

Sources:

Get a handle on the JAX-WS API's handler framework - [`http://www.javaworld.com/javaworld/jw-02-2007/jw-02-handler.html`]

Java API for XML Web Services (JAX-WS) Handler Framework - [`http://jax-ws.java.net/jax-ws-20-fcs/docs/handlers.html`]

9.2. Configure SOAP and logical handlers on the client side.

9.2.1. Use deployment descriptors.

blah-blah

9.2.2. Use programmatic API.

Question 09020B01

Which approach can NOT be used to configure handler chains for Web Service clients?

Options (select 1):

1. By using `Dispatch.setHandlerResolver(myHandlerResolver)` method.
2. By extending `PackagesResourceConfig` class and providing in constructor handler classes packages names.

3. By using WSDL customization file.
4. By using `Binding.setHandlerChain(handlerList)` method.
5. By using `@HandlerChain` annotation.

Answer:

Correct option is 2. `PackagesResourceConfig` subclassing is the configuration strategy for scanning for resource and provider classes in JAX-RS environment.

Option 1 is wrong. You CAN implement a `javax.xml.ws.handler.HandlerResolver` and add to a `Service` instance on client side. For example:

```
public static class MyHandlerResolver implements HandlerResolver {
    public List<Handler> getHandlerChain(PortInfo portInfo) {
        List<Handler> handlers = new ArrayList<Handler>();
        handlers.add(new ServiceAuthenticationSOAPHandler());
        ...
        return handlers;
    }
}
```

Then add a handler resolver to the `Service` instance using the `setHandlerResolver(...)` method. In this case, the port proxy or `Dispatch` object created from the `Service` instance uses the `HandlerResolver` to determine the handler chain. For example:

```
HandlerWS hws = new HandlerWS("http://java.boot.by/HandlerWS?WSDL", new QName("http://java.boot.by
hws.setHandlerResolver(new MyHandlerResolver());
```

Option 3 is wrong. You CAN create a customization file that includes a `<binding>` element that contains a handler chain description. The schema for the `<handler-chains>` element is the same for both handler chain files (on the server) and customization files. For example `custom.xml`:

```
<bindings xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    wsdlLocation="http://localhost:7001/handler/HandlerWS?WSDL"
    xmlns="http://java.sun.com/xml/ns/jaxws">
    <bindings node="wsdl:definitions"
        xmlns:jws="http://java.sun.com/xml/ns/javaee">
    <handler-chains>
        <handler-chain>
            <handler>
                <handler-class>ServiceAuthenticationSOAPHandler</handler-class>
          </handler>
        </handler-chain>
        <handler-chain>
            <handler>
                <handler-class>ServicePerformanceMonitorLogicalHandler</handler-class>
            </handler>
        </handler-chain>
    </handler-chains>
</bindings>
```

Then use `wsimport` and binding customization file to configure handler chain:

```
wsimport -d outputdir -b custom.xml HandlerWS.wsdl
```

Option 4 is wrong. You CAN set a handler chain directly on the `javax.xml.ws.BindingProvider`, such as a port proxy or `javax.xml.ws.Dispatch` object. For example:

```
public class Main {
    public static void main(String[] args) {
        HandlerWS test;

        try {
            test = new HandlerWS("http://java.boot.by/HandlerWS?WSDL"),
                                 new QName("http://java.boot.by", "HandlerWS"));
        } catch (MalformedURLException murl) { throw new RuntimeException(murl); }

        HandlerWSPortType port = test.getHandlerWSPortTypePort();

        Binding binding = ((BindingProvider)port).getBinding();

        List<Handler> handlerList = binding.getHandlerChain();
        handlerList.add(new ServiceAuthenticationSOAPHandler());
        handlerList.add(new ServicePerformanceMonitorLogicalHandler());

        binding.setHandlerChain(handlerList);

        String result = null;
        result = port.sayHello("Mikalai");
        System.out.println( "Got result: " + result );
    }
}
```

Option 5 is wrong. In addition to appearing on a endpoint implementation class or a SEI, as specified by JSR-181, the `@HandlerChain` annotation MAY appear on a generated service class. In this case, it affects all the proxies and `Dispatch` instances created using any of the ports on the service.

Furthermore, the JAX-WS 2.0 specification requires the handler resolver to return handler chains consistent with the contents of the handler chains' descriptor referenced by the `@HandlerChain` annotation.

The reference implementation of JAX-WS 2.0 in Java SE 6 supports the `@HandlerChain` annotation on generated service class and service endpoint interfaces. While it is not as powerful or flexible as configuring handler chains programmatically, the `@HandlerChain` annotation allows service clients to more easily work with handler chains.

NOTE: if the service client is a web client or an application client that runs in a Java EE container, the `@HandlerChain` annotation can be specified on Web Service references. This sets the handlers on the injected service. For example:

```
public class WebClient extends HttpServlet {

    @javax.jws.HandlerChain(file="handler-chain.xml")
    @WebServiceRef HelloService service;

    public void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws javax.servlet.ServletException {
        ...
    }
}
```

Sources:

`PackagesResourceConfig` JavaDoc - [`http://jersey.java.net/nonav/apidocs/1.1.4/jersey/com/sun/jersey/api/core/PackagesResourceConfig.html`

Creating and Using SOAP Message Handlers - [`http://download.oracle.com/docs/cd/E15051_01`

/wls/docs103/webserv_adv/handlers.html]

Get a handle on the JAX-WS API's handler framework - [http://www.javaworld.com/javaworld/jw-02-2007/jw-02-handler.html]

The Java API for XML-Based Web Services (JAX-WS) 2.2 - Section 9.2.1.3 (HandlerChain annotation) - [http://www.jcp.org/en/jsr/detail?id=224]

Chapter 10. Create low-level SOAP web services.

10.1. Describe the functions and capabilities of the APIs included within JAXP.

blah-blah

10.2. Describe the functions and capabilities of JAXB, including the JAXB process flow, such as XML-to-Java and Java-to-XML, and the binding and validation mechanisms provided by JAXB.

Question 100201

You have been working on Weather Forecast Web Service using "Code first" ("Start from Java") approach.

Given the following Java bean which is used as parameter:

```
// -- Java code fragment
public enum USState {AK, CO}
```

Generated WSDL XML Schema fragment looks as follows:

```
// -- XML Schema fragment
<xs:simpleType name="usState">
    <xs:restriction base="xs:string">
        <xs:enumeration value="CO" />
        <xs:enumeration value="AK" />
    </xs:restriction>
</xs:simpleType>
```

Another developer creates a .NET platform Weather Forecast Web Service client using "Contract first" ("Start from WSDL") approach and generated parameter bean source code is as follows:

```
// .NET auto generated code from XML Schema
public enum usState { CO, AK }
```

Which statement is true about these classes and XML Schema ?

Options (select 1):

1. Both classes and XML Schema mapped well and Weather Forecast Web Service will work fine with the .NET client.
2. You need to add `@XmlEnum` and `@XmlEnumValue` annotation for mapping of `enum` types.
3. You need to define `@XmlType.propOrder()` class level annotation so order of properties in Java bean and in C# bean matches.
4. None of the above.

Answer:

Correct option is 1.

A Java `enum` type maps to an XML Schema type constrained by `enumeration` facets. This, in turn, binds to the .NET type `enum` type.

Enumerated types capture in Java the notion of a property with a fixed set of possible values associatedwith it – but such enumerations of values could be captured in one of two ways:

- Using the symbolic labels for each value.
- Using numbers corresponding to the position of the values in the set.

In our case `enum` constant names and enumeration facet's `value`s match, so mapping is correct. This makes option 1 correct.

The default mapping for a named atomic type that is derived by restriction with enumeration facet(s) and whose restriction base type (represented by {base type definition}) is `xs:string` or derived from it is mapped to an `enum` type.

Example binding:

XML Schema fragment:

```
<xs:simpleType name="USState">
    <xs:restriction base="xs:string">
        <xs:enumeration value="CO" />
        <xs:enumeration value="AK" />
    </xs:restriction>
</xs:simpleType>
```

The corresponding `enum` type binding is:

```
public enum USState {
    AK, AL;
    public String value() { return name(); }
    public static USState fromValue(String value) { ... }
}
```

For all cases where there exist at least one enumeration constant name that is not the same as the enumeration constant's value, the generated `enum` type must have a final value field that is set by the `enum` type's constructor.

Example: Schema-derived `enum` type when enumeration facet's value does NOT match `enum` constant name.

Given following schema fragment:

```
<xs:simpleType name="Coin">
    <!-- Assume jaxb customization that binds Coin to an enumType-->
    <xs:restriction base="xs:int">
        <!--Assume jaxb customization specifying enumConstantName-->
        <xs:enumeration value="1"/> <!-- name="penny"-->
        <xs:enumeration value="5"/> <!-- name="nickel"-->
        <xs:enumeration value="10"/><!-- name="dime"-->
        <xs:enumeration value="25"/><!-- name="quarter-->
    </xs:restriction>
</xs:simpleType>
```

Schema-derived enum type:

```
@XmlEnum(value="java.lang.Integer.class")
public enum Coin {
    @XmlEnumValue("1") PENNY(1),
    @XmlEnumValue("5") NICKEL(5),
    @XmlEnumValue("10") DIME(10),
    @XmlEnumValue("25") QUARTER(25);
    public int value(){ return value; }
    public static Coin fromValue(int value) { ... }

    final private Integer value;
    Coin(int value) { this.value = value; }
}
```

Sources:

The WSIT Tutorial. Enum Type - [http://docs.sun.com/app/docs/doc/820-1072/ahihw?l=en&a=view]

Java Architecture for XML Binding (JAXB) - [http://jcp.org/aboutJava/communityprocess/mrel/jsr222/index.html]

Question 100202

You are performing development of Web Service, followed by deployment. What is the correct sequence of steps in JAXB data binding process?

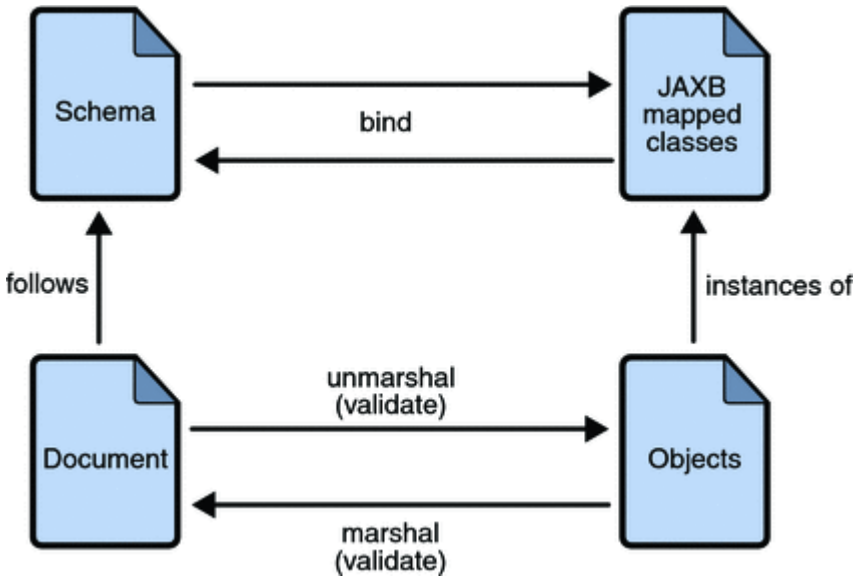
Options (select 1):

1. Generate classes, generate content tree, compile, marshal, process content, unmarshal.
2. Marshal, process content, generate classes, generate content tree, compile, unmarshal.
3. Generate classes, compile, unmarshal, generate content tree, process content, marshal.
4. Unarshal, process content, generate content tree, generate classes, compile, marshal.

Answer:

Correct option is 3.

The figure below shows what occurs during the JAXB binding process:



The general steps in the JAXB data binding process are:

1. Generate classes: An XML schema is used as input to the JAXB binding compiler to generate JAXB classes based on that schema.
2. Compile classes: All of the generated classes, source files, and application code must be compiled.
3. Unmarshal: XML documents written according to the constraints in the source schema are unmarshalled by the JAXB binding framework. Note that JAXB also supports unmarshalling XML data from sources other than files/documents, such as DOM nodes, string buffers, SAX Sources, and so forth.
4. Generate content tree: The unmarshalling process generates a content tree of data objects instantiated from the generated JAXB classes; this content tree represents the structure and content of the source XML documents.
5. Validate (optional): The unmarshalling process optionally involves validation of the source XML documents before generating the content tree. Note that if you modify the content tree in next step, you can also use the JAXB Validate operation to validate the changes before marshalling the content back to an XML document.
6. Process content: The client application can modify the XML data represented by the Java content tree by means of interfaces generated by the binding compiler.
7. Marshal: The processed content tree is marshalled out to one or more XML output documents. The content may be validated before marshalling.

Sources:

The Java EE5 Tutorial - `http://java.sun.com/javaee/5/docs/tutorial/doc/bnazg.html`

10.3. Use `Provider` API to create a web service.

10.3.1. Process the entire SOAP message, using the SAAJ APIs.

blah-blah

10.3.2. Process only the SOAP body, using JAXB.

blah-blah

10.4. Use `Dispatch` API to create a dynamic web service client.

blah-blah

Chapter 11. Use MTOM and MIME in a SOAP Web Service.

11.1. Use MTOM on the service.

Question 1101

Developer uses MTOM optimized SOAP 1.1 messages for JAX-WS Web Service. What would be HTTP Content-Type header if MTOM is enabled?

Options (select 1):

1. `multipart/related`
2. `application/xop+xml`
3. `text/xml`
4. `text/plain`

Answer:

Correct option is 1.

When sending a SOAP 1.1 message using the MIME Multipart/Related Serialization, the SOAP envelope Infoset is serialized into XML 1.0 as follows:

- The content-type of the outer package MUST be `multipart/related`.
- The type parameter of the content-type header of the outer package MUST have a value of `application/xop+xml`.
- The start-info parameter of the content-type header of the outer package MUST specify a content-type for the root part of `text/xml`.
- The content-type of the root part MUST be `application/xop+xml`.
- The type parameter of the content-type header of the root part MUST specify a content-type of `text/xml`.

The result is a MIME Multipart/Related XOP package: one body part, the root, containing an XML 1.0 representation of the modified SOAP 1.1 envelope, with an additional part used to contain the binary representation of each element that was optimized.

Sample MTOM message:

```
Content-Type: multipart/related; start-info="text/xml"; type="application/xop+xml"; boundary="----=
Content-Length: 3453
SOAPAction: ""

------=_Part_1_4558657.1118953559446
Content-Type: application/xop+xml; type="text/xml"; charset=utf-8

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <Detail xmlns="http://example.org/mtom/data">
      <image>
        <xop:Include xmlns:xop="http://www.w3.org/2004/08/xop/include" href="cid:5aeaa450-17f0-448
      </image>
    </Detail>
  </S:Body>
</S:Envelope>


------=_Part_1_4558657.1118953559446
Content-Type: image/jpeg
Content-ID: <5aeaa450-17f0-4484-b845-a8480c363444@example.org>


... binary data ...
```

The noteworthy points are:

1. The binary attachment is packaged in a MIME multi-part message (the same mechanism used in e-mail to handle attachments)
2. An `<xop:Include>` element is used to mark where the binary data is.
3. The actual binary data is kept in a different MIME part.

MTOM is efficient, in the sense that it doesn't have the 33% size increase penalty that `xs:base64Binary` has. It is interoperable, in the sense that it is a W3C standard. However, MIME multipart incurs a small cost proportional to the number of attachments, so it is not suitable for a large number of tiny attachments.

Sources:

SOAP 1.1 Binding for MTOM 1.0 - [http://www.w3.org/Submission/soap11mtom10/]

Binary Attachments (MTOM) - [http://metro.java.net/guide/Binary_Attachments__MTOM_.html]

11.1.1. Use @MTOM annotation with a web service.

blah-blah

11.1.2. Use MTOM policy in WSDL.

Question 11010B01

You are developing a Web Service using "WSDL-to-Java" approach. The Web Service uses a message which contains a binary field. The binary field is intended to carry a large image file, so it is not appropriate to send it as part of a normal SOAP message, but as optimized MTOM attachment. You have added the `xmime:expectedContentTypes` attribute to the element containing the binary data in WSDL:

```
...
<types>
    <schema targetNamespace="http://java.boot.by/types/"
            xmlns="http://www.w3.org/2001/XMLSchema"
            xmlns:xsd1="http://java.boot.by/types/"
            xmlns:xmime="http://www.w3.org/2005/05/xmlmime">
        <complexType name="ReportType">
            <sequence>
                <element name="imageData" type="xsd:base64Binary" xmime:expectedContentTypes="imag
            </sequence>
        </complexType>
        <element name="reportData" type="xsd1:ReportType" />
    </schema>
</types>
...
```

To which JAXB class the `reportData` element will be mapped?

Options (select 1):

1.
```
@XmlType
public class ReportType {
    @XmlMimeType("image/gif")
    protected javax.activation.DataHandler imageData;
    ...
}
```

2.
```
@XmlType
public class ReportType {
    @XmlMimeType("image/gif")
    protected javax.xml.transform.Source imageData;
    ...
}
```

3.
```
@XmlType
public class ReportType {
    @XmlMimeType("image/gif")
    protected byte[] imageData;
    ...
}
```

4.
```
@XmlType
public class ReportType {
    @XmlMimeType("image/gif")
    protected java.awt.Image imageData;
    ...
}
```

Answer:

Correct option is 4.

In WSDL, when defining a data type for passing along a block of binary data, such as an image file or a sound file, you define the element for the data to be of type `xsd:base64Binary`. By default, any element of type `xsd:base64Binary` results in the generation of a `byte[]` which can be serialized using MTOM. However, the default behavior of the code generators does not take full advantage of the serialization.

In order to fully take advantage of MTOM you must add annotations to either your service's WSDL document or the JAXB class that implements the binary data structure. Adding the annotations to the WSDL document forces the code generators to generate streaming data handlers for the binary data. Annotating the JAXB class involves specifying the proper content types and might also involve changing the type specification of the field containing the binary data.

If you want to use MTOM to send the binary part of the message as an optimized attachment you must add the `xmime:expectedContentTypes` attribute to the element containing the binary data. This attribute is defined in the `http://www.w3.org/2005/05/xmlmime` namespace and specifies the MIME types that the element is expected to contain. You can specify a comma separated list of MIME types. The setting of this attribute changes how the code generators create the JAXB class for the data. For most MIME types, the code generator creates a `DataHandler`. Some MIME types, such as those for images, have defined mappings.

The `wsimport` utility recognizes the `xmime:expectedContentType` attribute and will map the binary data to its proper Java representation instead. The table below shows the mapping rules:

Table 11.1. Default Binding for Known Media Type

MIME Type	Java Type
`image/gif`	`java.awt.Image`
`image/jpeg`	`java.awt.Image`
`text/xml` or `application/xml`	`javax.xml.transform.Source`
`text/plain`	`java.lang.String`
any other MIME type	`javax.activation.DataHandler`

Example 1:

```
<element name="image" type="base64Binary" />
```

is mapped to:

```
byte[]
```

Example 2:

```
<element name="image" type="base64Binary"
    xmime:expectedContentTypes="image/jpeg"
    xmlns:xmime="http://www.w3.org/2005/05/xmlmime" />
```

is mapped to:

```
java.awt.Image
```

Sources:

Metro Guide. Binary Attachments (MTOM) - [`http://metro.java.net/guide/Binary_Attachments__MTOM_.html`]

Java Architecture for XML Binding (JAXB) 2.0 - ENHANCED BINARY DATA HANDLING - [`http://jcp.org/aboutJava/communityprocess/mrel/jsr222/index.html`]

11.1.3. Use MTOM in the deployment descriptors

blah-blah

11.1.4. Use `MTOMFeature` with `javax.xml.ws.Endpoint` API

blah-blah

11.1.5. Use `swaRef` in WSDL.

Question 11010E01

Your team is working on a Web Service using "Java-to-WSDL" approach. The Web Service is required to send binary attachments. Also, you want to make sure that Web Service is WS-I Attachment Profile 1.0 compliant and decide to use mechanism of `ref:swaRef` in generated WSDL to reference MIME attachment parts. Which two independent approaches could you use to accomplish this requirement?

Options (select 2 best options):

1. Annotate payload bean with `@SwaRef`.
2. Annotate Web Service interface method with `@SwaRef`.
3. Annotate payload bean with `@XmlAttachmentRef`.
4. Annotate Web Service interface method with `@XmlAttachmentRef`.
5. Annotate Web Service interface method with `@MTOM`.
6. Annotate payload bean with `@XmlID`.

Answer:

Correct options are 3 and 4.

WS-I Attachment Profile 1.0 defines mechanism to reference MIME attachment parts using `swaRef`. In this mechanism the content of XML element of type `wsi:swaRef` is sent as MIME attachment and the element inside SOAP `Body` holds the reference to this attachment in the CID URI scheme as defined by RFC 2111.

JAX-WS endpoints delegate all marshalling/unmarshalling to the JAXB API. The most simple way to

enable SwaRef encoding for `DataHandler` types is to annotate a payload bean with the `@XmlAttachmentRef` annotation as shown below:

```
@XmlRootElement
class Foo {
    @XmlAttachmentRef
    @XmlAttribute
    DataHandler data;

    @XmlAttachmentRef
    @XmlElement
    DataHandler body;
}
```

The above code maps to the following XML:

```
<xs:element name="foo" xmlns:ref="http://ws-i.org/profiles/basic/1.1/xsd">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="body" type="ref:swaRef" minOccurs="0" />
        </xs:sequence>
        <xs:attribute name="data" type="ref:swaRef" use="optional" />
    </xs:complexType>
</xs:element>
```

This makes option 3 correct.

With document wrapped endpoints you may even specify the `@XmlAttachmentRef` annotation on the service endpoint interface:

```
@WebService
public interface DocWrappedEndpoint {

    @WebMethod
    @XmlAttachmentRef
    DataHandler parameterAnnotation(@XmlAttachmentRef DataHandler data, String test);

}
```

The message would then refer to the attachment part by `CID`:

```
<env:Envelope xmlns:env='http://schemas.xmlsoap.org/soap/envelope/'>
    <env:Header/>
    <env:Body>
        <ns2:parameterAnnotation xmlns:ns2='http://swaref.java.boot.by/'>
            <arg0>cid:0-1180017772935-32455963@java.boot.by</arg0>
            <arg1>Wrapped test</arg1>
        </ns2:parameterAnnotation>
    </env:Body>
</env:Envelope>
```

This makes option 4 correct.

There is no `@SwaRef` annotation.

The `@XmlID` annotation maps to `xsd:ID` Schema type. This makes option 6 wrong.

Sources:

Annotation Type `XmlAttachmentRef` JavaDoc - [`http://download.oracle.com/javase/6/docs/api/javax/xml/bind/annotation/XmlAttachmentRef.html`]

Java Architecture for XML Binding (JAXB) - [`http://jcp.org/aboutJava/communityprocess/mrel/jsr222/index.html`]

11.1.6. Use MIME binding in WSDL.

blah-blah

11.2. Use MTOM on the client.

11.2.1. Use `MTOMFeature` with `getPort()` methods.

Question 11020101

Given the `CatalogService` dynamic proxy factory and the `Catalog` Web Service proxy, how can developer enable MTOM on the client side?

Options (select 1):

1.
```
CatalogService service = new CatalogService(new MTOMFeature());
Catalog catalogPort = service.getCatalogPort();
```

2.
```
CatalogService service = new CatalogService();
Catalog catalogPort = service.getCatalogPort(new MTOMFeature());
```

3.
```
CatalogService service = new CatalogService();
service.setFeature(new MTOMFeature());
Catalog catalogPort = service.getCatalogPort();
```

4.
```
CatalogService service = new CatalogService();
Catalog catalogPort = service.getCatalogPort();
catalogPort.setFeature(new MTOMFeature());
```

Answer:

Correct option is 2.

You can enable MTOM on the client side by passing `javax.xml.ws.MTOMFeature` to the `javax.xml.ws.Service.getPort(...)` method.

Web Service client class generated by JAX-WS includes two `get<X>Port` methods: one that takes no arguments, and another that takes a variable length argument of `WebServiceFeature` objects. The `CatalogService`, generated by JAX-WS during `wsimport`, is shown below:

```
...
@WebEndpoint(name = "CatalogPort")
public Catalog getCatalogPort() {
    return super.getPort(new QName("http://java.boot.by",
            "CatalogPort"), Catalog.class);
}

@WebEndpoint(name = "CatalogPort")
public Catalog getCatalogPort(WebServiceFeature... features) {
    return super.getPort(new QName("http://java.boot.by",
            "CatalogPort"), Catalog.class, features);
}
...
```

A `WebServiceFeature` is a standard manner of enabling and disabling support for a variety of useful mechanisms at runtime. Some features come built in with JAX-WS, and vendors can provide additional features. You can also write your own.

The class `javax.xml.ws.soap.MTOMFeature` extends `WebServiceFeature`. The generated client interfaces make it easy to invoke your service operations using a built-in or custom feature. To use MTOM on the client, simply retrieve the port using the factory that accepts a variable-length argument of `WebServiceFeature` objects, passing in an instance of the `MTOMFeature`. Here's how you can enable MTOM using the `getCatalogPort` method:

```
CatalogService service = new CatalogService();
Catalog catalogPort = service.getCatalogPort(new MTOMFeature());
```

The implementation is handled for you by the runtime.

The `MTOMFeature` constructor also accepts an optional integer argument indicating the threshold, or the number of bytes that the binary data should be before being sent as an attachment:

```
MTOMFeature()
          Create an MTOMFeature.

MTOMFeature(boolean enabled)
          Creates an MTOMFeature.

MTOMFeature(boolean enabled, int threshold)
          Creates an MTOMFeature.

MTOMFeature(int threshold)
          Creates an MTOMFeature.
```

The default value for the `threshold` is 0.

SOAP MTOM/XOP mechanism on the client can be enabled or disabled by any one of the following ways:

- Programmatically passing `MTOMFeature` for a `Service` method that creates a SEI proxy or a `Dispatch` instance.
- Using `<port-component-ref>/<enable-mtom>` deployment descriptor element for a corresponding SEI proxy instance.
- Using `@MTOM` with a `@WebServiceRef` that creates a SEI proxy instance.

Sources:

Class `MTOMFeature` JavaDoc - [`http://download.oracle.com/javase/6/docs/api/javax/xml/ws/soap/MTOMFeature.html`]

The Java API for XML-Based Web Services (JAX-WS) 2.2 - [`http://jcp.org/aboutJava/communityprocess/mrel/jsr222/index.html`]

Web Services for Java EE - [`http://jcp.org/aboutJava/communityprocess/mrel/jsr109/index.html`]

11.2.2. Use MTOM in the deployment descriptors.

blah-blah

11.2.3. Sending any additional attachments using `MessageContext`

properties.

blah-blah

Chapter 12. Use WS-Addressing with a SOAP web service

12.1. Use Addressing on the service

12.1.1. Use `@Addressing` annotation with a web service

blah-blah

12.1.2. Use `wsam:Addressing` policy in WSDL

blah-blah

12.1.3. Use Addressing in the deployment descriptors

blah-blah

12.1.4. Use `AddressingFeature` with `javax.xml.ws.Endpoint` API.

blah-blah

12.1.5. Use `@Action` and `@FaultAction` on the service methods.

blah-blah

12.1.6. Use `WebServiceContext.getEndpointReference()`

blah-blah

12.2. Use Addressing on the client.

12.2.1. Use `AddressingFeature` with `getPort()` methods.

blah-blah

12.2.2. Use Addressing in the deployment descriptors

blah-blah

12.2.3. Use `BindingProvider.getEndpointReference()`

blah-blah

12.2.4. Use `getPort(EndpointReference)` methods.

blah-blah

Chapter 13. Configure Message Level security for a SOAP web service

Configure SOAP message-layer security requirements of service as policy assertions in service description; this should be supported by an IDE such as is the case when Netbeans is used to select a Metro Security Profile.

13.1. Select the appropriate Security Profile for your service. The selection would be based on a match of the Protection guarantees offered by the profile and those

required by the service.

Question 130101

Given the `BusinessCardService.wsdl` fragment, which is true about SOAP message security?

```
<wsdl:definitions name="BusinessCardService" ...>
    ...
    <wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
        ....
        <sp:SignedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
            <sp:Body />
            <sp:Header Name="To" Namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"/>
        </sp:SignedParts>
        ...
    </wsp:Policy>
    ...
</wsdl:definitions>
```

Options (select 1):

1. Both SOAP message `Body` element and header `To` must be encoded.
2. Only SOAP message `Body` element must be encoded.
3. SOAP message `Body` element and header `To` integrity must be protected.
4. Only SOAP message `Body` element integrity must be protected.
5. Both SOAP message `Body` element and header `To` must be encrypted.
6. Only SOAP message `Body` element must be encrypted.

Answer:

Correct option is 3.

WS-Security Policy specification defines a standard way to define how to secure messages exchanged between Web Services and clients. WS-Security policy language can be used to publish security requirements and constrains of a Web Service using the WSDL specification. That is, using WS-Security Policy language, we can drive a Web Service security engine to secure out going messages in a certain way and instruct the verification of incoming messages in a standard, defined way.

The `SignedParts` assertion is used to specify the parts of the message outside of security headers that require integrity protection. This assertion can be satisfied using WSS: SOAP Message Security mechanisms or by mechanisms out of scope of SOAP message security, for example by sending the message over a secure transport protocol like HTTPS. The binding specific token properties detail the exact mechanism by which the protection is provided.

For example, message part integrity can be checked using a signature (e.g. MD5 hash).

There may be multiple `SignedParts` assertions present. Multiple `SignedParts` assertions present within a policy alternative are equivalent to a single `SignedParts` assertion containing the union of all specified message parts. Note that this assertion does not require that a given part appear in a message, just that if such a part appears, it requires integrity protection.

Syntax:

```
<sp:SignedParts xmlns:sp="..." ... >
    <sp:Body />?
```

```
    <sp:Header Name="xs:NCName"? Namespace="xs:anyURI" ... />*
    <sp:Attachments />?
    ...
</sp:SignedParts>
```

The following describes the attributes and elements listed in the schema outlined above:

- `/sp:SignedParts`

  This assertion specifies the parts of the message that need integrity protection. If no child elements are specified, all message headers targeted at the UltimateReceiver role [SOAP12] or actor [SOAP11] and the body of the message MUST be integrity protected.

- `/sp:SignedParts/sp:Body`

  Presence of this optional empty element indicates that the entire body, that is the `soap:Body` element, it's attributes and content, of the message needs to be integrity protected.

- `/sp:SignedParts/sp:Header`

  Presence of this optional element indicates a specific SOAP header, its attributes and content (or set of such headers) needs to be protected. There may be multiple `sp:Header` elements within a single `sp:SignedParts` element. If multiple SOAP headers with the same local name but different namespace names are to be integrity protected multiple `sp:Header` elements are needed, either as part of a single `sp:SignedParts` assertion or as part of separate `sp:SignedParts` assertions.

  This element only applies to SOAP header elements targeted to the same actor/role as the Security header impacted by the policy. If it is necessary to specify a requirement to sign specific SOAP Header elements targeted to a different actor/role, that may be accomplished using the `sp:SignedElements` assertion.

- `/sp:SignedParts/sp:Header/@Name`

  This optional attribute indicates the local name of the SOAP header to be integrity protected. If this attribute is not specified, all SOAP headers whose namespace matches the Namespace attribute are to be protected.

- `/sp:SignedParts/sp:Header/@Namespace`

  This required attribute indicates the namespace of the SOAP header(s) to be integrity protected.

- `/sp:SignedParts/sp:Attachments`

  Presence of this optional empty element indicates that all SwA (SOAP Messages with Attachments) attachments [SwA] are to be integrity protected. When SOAP Message Security is used to accomplish this, all message parts other than the part containing the primary SOAP envelope are to be integrity protected as outlined in WSS: SOAP Message Security [WSS:SwAProfile1.1].

Sources:

WS-SecurityPolicy 1.2 `[http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html]`

Question 130102

Given the `StockQuote.wsdl` fragment, which is true about SOAP message security?

```
<definitions name="StockQuote" ...>
```

```
    ...
    <wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" wsu:Id="StockQuotePolicy"
        ....
        <sp:SignedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy" />
        ...
    </wsp:Policy>
    ...
    <binding name="StockQuoteSoap" type="tns:StockQuotePortType">
        <soap:binding style="document" transport="http://example.com/smtp"/>
        <wsp:PolicyReference URI="#StockQuotePolicy"/>
        <operation name="SubscribeToQuotes">
           <input message="tns:SubscribeToQuotes">
               <soap:body parts="body" use="literal"/>
               <soap:header message="tns:SubscribeToQuotes" part="subscribeheader" use="literal"/>
           </input>
        </operation>
    </binding>
    ...
</definitions>
```

Options (select 1):

1. Both SOAP message `Body` and `Header` elements must be encoded.
2. SOAP message `Body` and `Header` elements' integrity must be protected.
3. Both SOAP message `Body` and `Header` elements must be encrypted.
4. None of the above.

Answer:

Correct option is 2.

See for details the previous question explanation.

Syntax:

```
<sp:SignedParts xmlns:sp="..." ... >
    <sp:Body />?
    <sp:Header Name="xs:NCName"? Namespace="xs:anyURI" ... />*
    <sp:Attachments />?
    ...
</sp:SignedParts>
```

The following describes the attributes and elements listed in the schema outlined above:

- `/sp:SignedParts`

  This assertion specifies the parts of the message that need integrity protection. If no child elements are specified, all message headers targeted at the UltimateReceiver role [SOAP12] or actor [SOAP11] **and** the body of the message MUST be integrity protected.

  This makes option 2 correct.

Sources:

WS-SecurityPolicy 1.2 `[http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html]`

Question 130103

Given the `CalculatorWSService.wsdl` fragment, which is true about SOAP message security?

```
<wsdl:definitions name="CalculatorWSService" ...>
    ...
    <wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
        ....
        <sp:EncryptedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
            <sp:Body />
            <sp:Header Name="To" Namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"/>
        </sp:EncryptedParts>
        ...
    </wsp:Policy>
    ...
</wsdl:definitions>
```

Options (select 1):

1. Both SOAP message `Body` element and header `To` must be encoded.
2. SOAP message `Body` element and header `To` integrity must be protected.
3. Both SOAP message `Body` element and header `To` require confidentiality.
4. Only SOAP message `Body` element integrity must be protected.
5. Only SOAP message `Body` element must be encoded.
6. Only SOAP message `Body` element requires confidentiality.

Answer:

Correct option is 3.

The `EncryptedParts` assertion is used to specify the parts of the message that require confidentiality. This assertion can be satisfied with WSS: SOAP Message Security mechanisms or by mechanisms out of scope of SOAP message security, for example by sending the message over a secure transport protocol like HTTPS. The binding specific token properties detail the exact mechanism by which the protection is provided.

There may be multiple `EncryptedParts` assertions present. Multiple `EncryptedParts` assertions present within a policy alternative are equivalent to a single `EncryptedParts` assertion containing the union of all specified message parts. Note that this assertion does not require that a given part appear in a message, just that if such a part appears, it requires confidentiality protection.

Syntax:

```
<sp:EncryptedParts xmlns:sp="..." ... >
    <sp:Body/>?
    <sp:Header Name="xs:NCName"? Namespace="xs:anyURI" ... />*
    <sp:Attachments />?
    ...
</sp:EncryptedParts>
```

The following describes the attributes and elements listed in the schema outlined above:

- `/sp:EncryptedParts`

  This assertion specifies the parts of the message that need confidentiality protection. The single child element of this assertion specifies the set of message parts using an extensible dialect.

  If no child elements are specified, the body of the message MUST be confidentiality protected.

- `/sp:EncryptedParts/sp:Body`

  Presence of this optional empty element indicates that the entire body of the message needs to be confidentiality protected. In the case where mechanisms from WSS: SOAP Message Security are used to satisfy this assertion, then the `soap:Body` element is encrypted using the #Content encryption type.

- `/sp:EncryptedParts/sp:Header`

  Presence of this optional element indicates that a specific SOAP header (or set of such headers) needs to be protected. There may be multiple `sp:Header` elements within a single Parts element. Each header or set of headers MUST be encrypted. Such encryption will encrypt such elements using WSS 1.1 Encrypted Headers. As such, if WSS 1.1 Encrypted Headers are not supported by a service, then this element cannot be used to specify headers that require encryption using message level security. If multiple SOAP headers with the same local name but different namespace names are to be encrypted then multiple `sp:Header` elements are needed, either as part of a single `sp:EncryptedParts` assertion or as part of separate `sp:EncryptedParts` assertions.

- `/sp:EncryptedParts/sp:Header/@Name`

  This optional attribute indicates the local name of the SOAP header to be confidentiality protected. If this attribute is not specified, all SOAP headers whose namespace matches the Namespace attribute are to be protected.

- `/sp:EncryptedParts/sp:Header/@Namespace`

  This required attribute indicates the namespace of the SOAP header(s) to be confidentiality protected.

- `/sp:EncryptedParts/sp:Attachments`

  Presence of this optional empty element indicates that all SwA (SOAP Messages with Attachments) attachments [SwA] are to be confidentiality protected. When SOAP Message Security is used to accomplish this, all message parts other than the part containing the primary SOAP envelope are to be confidentiality protected as outlined in WSS: SOAP Message Security [WSS:SwAProfile1.1].

Sources:

WS-SecurityPolicy 1.2 `[http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html]`

Question 130104

Given the `StockQuoteWS.wsdl` fragment, which is true about SOAP message security?

```
<definitions name="StockQuoteWS" ...>
    ...
    <wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" wsu:Id="StockQuoteWSPolic
        ....
        <sp:EncryptedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy" />
        ...
    </wsp:Policy>
    ...

    <binding name="StockQuoteSoap" type="tns:StockQuotePortType">
        <soap:binding style="document" transport="http://example.com/smtp"/>
        <wsp:PolicyReference URI="#StockQuoteWSPolicy"/>
        <operation name="SubscribeToQuotes">
           <input message="tns:SubscribeToQuotes">
               <soap:body parts="body" use="literal"/>
               <soap:header message="tns:SubscribeToQuotes" part="subscribeheader" use="literal"/>
           </input>
        </operation>
```

```
    </binding>
    ...
</definitions>
```

Options (select 1):

1. Both SOAP message `Body` and `Header` elements must be encoded.
2. SOAP message `Body` and `Header` elements' integrity must be protected.
3. Both SOAP message `Body` and `Header` elements must be encrypted.
4. None of the above.

Answer:

Correct option is 4.

See for details the previous question explanation.

Syntax:

```
<sp:EncryptedParts xmlns:sp="..." ... >
    <sp:Body/>?
    <sp:Header Name="xs:NCName"? Namespace="xs:anyURI" ... />*
    <sp:Attachments />?
    ...
</sp:EncryptedParts>
```

The following describes the attributes and elements listed in the schema outlined above:

- `/sp:EncryptedParts`

  This assertion specifies the parts of the message that need confidentiality protection. The single child element of this assertion specifies the set of message parts using an extensible dialect.

  If no child elements are specified, the body of the message MUST be confidentiality protected.

  Therefore, only `soap:Body` will be encrypted. This makes option 2 wrong.

Sources:

WS-SecurityPolicy 1.2 `[http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html]`

13.2. Configure Username/Password callbacks required by the Username Token Profile.

blah-blah

13.3. Configure any server side Validators as maybe applicable for the profile. There are defaults in GlassFish for most of them.

blah-blah

13.4. Optimize interaction between client and server by using WS-SecureConversation.

blah-blah

Chapter 14. Apply best practices to design and implement Web Services.

Question 1401

What is true about Service Implementation Layers?

Options (select 3):

1. The Service Interaction Layer consists of the endpoint interface that the service exposes to clients and through which it receives client requests.
2. Sometimes multiple layers make a service unnecessarily complicated and, in these cases, it may be simpler to design the service as one layer.
3. The Service Processing Layer consists of the endpoint interface that the service exposes to clients and through which it receives client requests.
4. The Service Processing Layer holds all business logic used to process client requests.
5. Web Service Implementation is always separated in an Interaction Layer and a Processing Layer.
6. The Service Interaction Layer holds all business logic used to process client requests.

Answer:

Correct options are 1, 2, and 4.

Option 1 is correct. The Service Interaction Layer consists of the endpoint interface that the service exposes to clients and through which it receives client requests. The interaction layer also includes the logic for how the service delegates the requests to business logic and formulates responses. When it receives requests from clients, the interaction layer performs any required preprocessing before delegating requests to the business logic. When the business logic processing completes, the interaction layer sends back the response to the client. The interaction layer may be different for those scenarios where the service expects to receive XML documents from clients but the business logic deals with objects. In these cases, you map the XML documents to equivalent object representations in the interaction layer before delegating the request to the business logic.

Option 2 is correct. Although there are advantages to viewing a service in terms of Interaction and Processing Layers, a Web Service may opt to merge these two layers into a single layer. There are times when multiple layers make a service unnecessarily complicated and, in these cases, it may be simpler to design the service as one layer. Typically, this happens in scenarios where the logic in either layer is too small to merit a separate layer.

Option 3 is wrong. Processing Layer holds business logic.

Option 4 is correct. The Service Processing Payer holds all business logic used to process client requests. It is also responsible for integrating with EISs and other Web Services. In the case of existing applications adding a Web Service interoperable interface, the existing application itself typically forms the Service Processing Layer.

Option 5 is wrong. Service implementation may consist of one layer.

Option 6 is wrong. Interaction Layer is the starting point of a client's interaction with the service. It does not hold business logic.

Sources:

Designing Web Services with the J2EE 1.4 Platform - [`http://java.sun.com/blueprints/guidelines/designing_webservices/`]

Question 1402

What Layer of Web Service Implementation is the best place to validate incoming XML document?

Options (select 1):

1. Service Processing Layer
2. Service Interaction Layer
3. Service Parsing Layer
4. Service Validation Layer

Answer:

Correct option is 2.

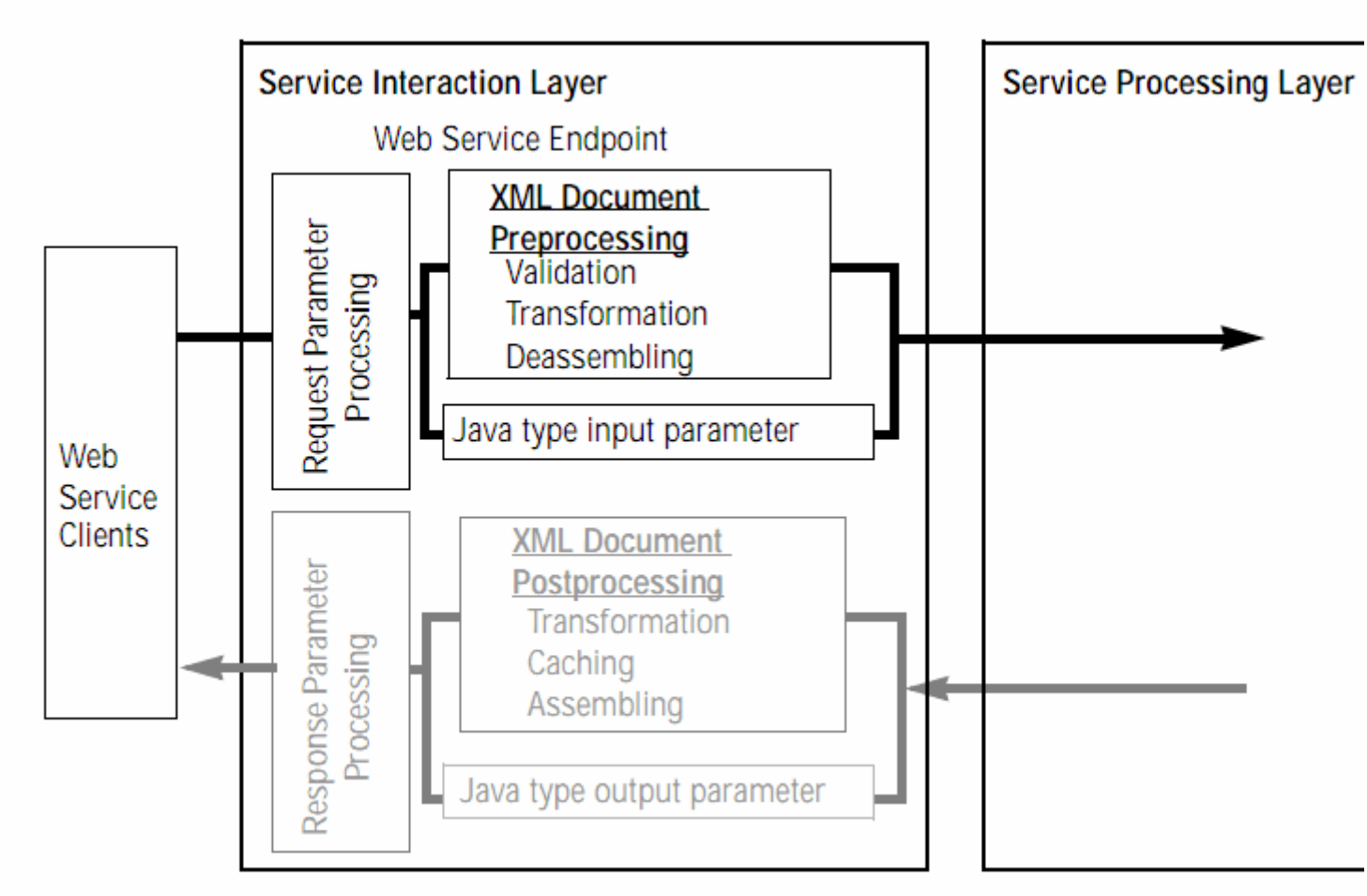
The Interaction Layer, through the Service Endpoint, receives client requests. The platform maps the incoming client requests, which are in the form of SOAP messages, to method calls present in the Web service interface. Before delegating these incoming client requests to the Web Service business logic, you should perform any required security validation, transformation of parameters, and other required pre-processing of parameters.

Web Service calls are basically method calls whose parameters are passed as either Java objects, XML documents (`javax.xml.transform.Source` objects), or even SOAP document fragments (`javax.xml.soap.SOAPElement` objects). For parameters that are passed as Java objects (such as `String`, `int`, JAX-WS value types, and so forth), do the application-specific parameter validation and map the incoming objects to domain-specific objects in the Interaction Layer before delegating the request to the Processing Layer. You may have to undertake additional steps to handle XML documents that are passed as parameters. These steps, which are best performed in the Interaction Layer of your service, are as follows:

1. The Service Endpoint should validate the incoming XML document against its schema.
2. When the service's Processing Layer and business logic are designed to deal with XML documents, you should transform the XML document to an internally supported schema, if the schema for the XML document differs from the internal schema, before passing the document to the processing layer.
3. When the Service Implementation deals with objects but the interface receives XML documents, then, as part of the Interaction Layer, map the incoming XML documents to domain objects before delegating the request to the processing layer.

It is important that these three steps - validation of incoming parameters or XML documents, translation of XML documents to internal supported schemas, and mapping documents to domain objects - be performed as close to the service endpoint as possible, and certainly in the Service Interaction Layer. A design such as this also helps to catch errors early, and thus avoids unnecessary calls and round-trips to the processing layer.

Figure 14.1. Web Service Request Processing



Option 1 is wrong. Processing Layer holds business logic. It deals with domain-specific business objects which usually created from incoming XML in Interaction Layer.

Options 3 and 4 are wrong. Layer names are invalid.

Sources:

Designing Web Services with the J2EE 1.4 Platform - [`http://java.sun.com/blueprints/guidelines/designing_webservices/html/webservdesign5.html`]

14.1. Use different encoding schemes - fast infoset.

blah

14.2. Use GZIP for optimiziing message sizes.

blah

14.3. Use catalog mechanism for WSDL access.

blah

14.4. Refer to WS-I sample app for best practices.

Question 140401

What statements are true about Web Services and SOA?

Options (select 2):

1. Both Web Servives and SOA can use UDDI.

2. Web Servives decrease CPU utilization on the server.
3. SOA decreases network bandwith requirements.
4. Web Services expose their interfaces as fine-grained components.
5. SOA is an architectural style and Web Services is a way to realize SOA.

Answer:

Correct options are 1 and 5.

There are four primary characteristics of an SOA that together differentiate it from previous architectural paradigms:

- **Interoperability** — Open interface standards, as opposed to systems or vendor specific standards.

  A basic characteristic of an SOA is the ability to deliver greater system interoperability. A service provider publishes a service description, which provides information regarding where to locate and how to invoke a service. This invocation information informs the service requester of the information the service requires and the information it will return. In effect, this function makes the service a "black box", where knowledge of its inner workings is not needed.

  As long as the service is trusted and you believe that the service will do what it describes, then the details of how the service has been implemented are unimportant. The physical location of the service, the programming language used to implement the service, and the hardware that the service runs on are all implementation details that can be hidden in an SOA.

  Although the implementation details of a service are hidden, that is not to say that they are unimportant. In fact, as an organization's SOA evolves, it will be critical to closely monitor and manage the availability of system resources, including the following:

  - **Network Bandwidth** — An SOA can significantly increase network bandwidth requirements. The use of XML can increase bandwidth requirements by three to four times; as such, it is important to monitor network usage as the SOA is implemented.

    This makes option 3 incorrect.

  - **Load Balancing** — From a Quality of Service (QoS) perspective, it is important to monitor the peak loading of machines that are running services to ensure that performance bottlenecks do not appear. For example, as systems come online, one single service might experience very heavy usage. This service could become a system bottleneck due to either network or processor bandwidth limitations.

  - **Identity and Security** — Even when implementing an SOA within the enterprise, it will be increasingly important to ensure that appropriate identification and security is implemented to ensure that access to services is secure and authenticated. Logging information regarding who is running which services and when will provide an important audit trail should a service be accessed without authorization.

- **Coarse Grained** — Business level services, as opposed to fine-grained code or components.

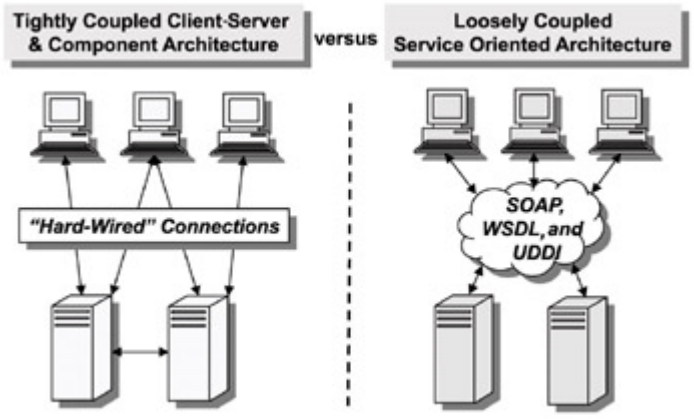
  Object-oriented languages such as Java, COM, and CORBA expose their interfaces as fine-grained components. These components are then aggregated into larger, coarse-grained services that more closely resemble real business functions.

  This makes option 4 incorrect.

- **Loosely Coupled** — Flexible, loosely coupled services, as opposed to tightly coupled units of code.

  Traditionally, systems are integrated using hard-coded connections in which the call parameters, interface name, network location, and so on are all hard-wired when an

application is implemented. This approach creates tightly coupled systems which are highly sensitive to even minor changes in system configuration. For example, changing something as simple as a server's network address could cause a number of systems to no longer operate correctly. This concept is illustrated in figure below:



From a Web Srvices perspective, loose coupling is achieved through three progressive levels of abstraction:

- **Services Communication** — SOAP provides the first level of abstraction. At this level, the details of which development language the service was implemented in is hidden. For example, a service implemented in Java can work hand-in-hand with a service implemented in .NET. However, the service parameters and network location of each service must still be specified manually.
- **Services Description** — WSDL provides the next level of abstraction. At this level, the service description contains the details of the service parameters and network address. The service description must itself be manually located, but once located, it can be manipulated programmatically to determine the additional information required to locate the service and run it.
- **Services Publishing and Discovery** — UDDI provides the final level of abstraction. At this level, the service description is published in a UDDI registry, from which it can be dynamically located using the registry's search capabilities. At this level of abstraction the only information required is access to the UDDI registry and enough information to search for the service (for example, the service's name, or type of service to be performed).

  This makes option 1 correct.

- **Dynamically Discoverable** — Distributed services that are dynamically located at run-time, as opposed to hard-wired references.

  In an SOA, services are published in a services registry. Services registries are central repositories, either within an organization or on the Web, which can be used to dynamically discover services. From a practical sense it is more than likely that business users will already know that the service they are looking for already exists, but the services registry provides an additional level of abstraction, allowing the service description to be dynamically retrieved when the service is run.

SOA is usually realized through Web Services. Web Services specifications may add to the confusion of how to best utilize SOA to solve business problems. In order for a smooth transition to SOA, managers and developers in organizations should known that:

- SOA is an architectural style that has been around for years. Web Services are the preferred way to realize SOA.

  This makes option 5 correct.

- SOA is more than just deploying software. Organizations need to analyze their design techniques and development methodology and partner/customer/supplier relationship.

- Moving to SOA should be done incrementally and this requires a shift in how we compose service-based applications while maximizing existing IT investments.

Sources:

Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI) [http://www.oracle.com/technetwork/articles/javase/index-142519.html]

Question 140402

What statements are true about WS-I BP 1.1?

Options (select 2):

1. All Java EE 6 Application Servers must support WS-I BP 1.1
2. Java EE 6 Application Servers are not required to support WS-I BP 1.1
3. All Java EE 6 applications must support WS-I BP 1.1
4. Java EE 6 applications are not required to support WS-I BP 1.1

Answer:

Correct options are 1 and 4.

All Java EE 6 compliant Application Servers must satisfy all the requirements defined in the Java EE 6 specification. Java EE 6 specification includes JAX-WS 2.2 specification (JSR 224)

JAX-WS 2.2 supports the Web Services Interoperability (WS-I) Basic Profile Version 1.1. The WS-I Basic Profile is a document that clarifies the SOAP 1.1 and WSDL 1.1 specifications to promote SOAP interoperability.

This makes option 1 correct.

WS-I BP 1.1 is a set of specifications that guarantee Web Services interoperability if users adhere to the profiles guidelines and if vendors include support for it in products. The profile identifies how Web Services specifications should be used together to create interoperable Web Services. Java EE 6 application is required to support WS-I BP 1.1 ONLY if developer wants the application be interoperable.

This makes option 4 correct.

Sources:

WS-I Basic Profile 1.1 [http://www.ws-i.org]

Question 140403

What statements are correct about XML documents description standards for Web Services?

Options (select 2):

1. DTD language is the preffered way for Web Services description - according to WS-I BP 1.1
2. XML Schema and DTD are the preffered ways for Web Services description - according to WS-I BP 1.1
3. XML Schema is the preffered way for Web Services description - according to WS-I BP 1.1
4. World Wide Web Consortium (W3C) working group suggests XSD for Web Services.
5. World Wide Web Consortium (W3C) working group suggests SCH for Web Services.

Answer:

Correct options are 3 and 4.

WSDL 1.1 uses XML Schema as one of its type systems. The Profile mandates the use of XML Schema as the type system for WSDL descriptions of Web Services.

WS-I BP 1.1: R2800 A DESCRIPTION MAY use any construct from XML Schema 1.0.

WS-I BP 1.1: R2801 A DESCRIPTION MUST use XML Schema 1.0 Recommendation as the basis of user defined datatypes and structures.

W3C suggests the following XSD Schema usage scenarios. Most of them can be performed by Web Services.

1. Publishing and syndication.

   Distribution of information through publishing and syndication services. Involves collections of XML documents with complex relations among them. Structural schemas describe the properties of headlines, news stories, thumbnail images, cross-references, etc. Document views under control of different versions of a schema.

2. Electronic commerce transaction processing.

   Libraries of schemas define business transactions within markets and between parties. A schema-aware processor is used to validate a business document, and to provide access to its information set.

3. Supervisory control and data acquisition.

   The management and use of network devices involves the exchange of data and control messages. Schemas can be used by a server to ensure outgoing message validity, or by the client to allow it to determine what part of a message it understands. In multi-vendor environment, discriminates data governed by different schemas (industry-standard, vendor-specific) and know when it is safe to ignore information not understood and when an error should be raised instead; provide transparency control. Applications include media devices, security systems, plant automation, process control.

4. Traditional document authoring/editing governed by schema constraints.

   One important class of application uses a schema definition to guide an author in the development of documents. A simple example might be a memo, whereas a more sophisticated example is the technical service manuals for a wide-body intercontinental aircraft. The application can ensure that the author always knows whether to enter a date or a part-number, and might even ensure that the data entered is valid.

5. Use schema to help query formulation and optimization.

   A query interface inspect XML schemas to guide a user in the formulation of queries. Any given database can emit a schema of itself to inform other systems what counts as legitimate and useful queries.

6. Open and uniform transfer of data between applications, including databases.

   XML has become a widely used format for encoding data (including metadata and control data) for exchange between loosely coupled applications. Such exchange is currently hampered by the difficulty of fully describing the exchange data model in terms of XML DTDs; exchange data model versioning issues further complicate such interactions. When the exchange data model is represented by the more expressive XML Schema definitions, the task of mapping the exchange data model to and from application internal data models will be simplified.

7. Metadata Interchange.

   There is growing interest in the interchange of metadata (especially for databases) and in the

use of metadata registries to facilitate interoperability of database design, DBMS, query, user interface, data warehousing, and report generation tools. Examples include ISO 11179 and ANSI X3.285 data registry standards, and OMG's proposed XMI standard.

Options 1 and 2 are incorrect, DTD (Document Type Definitions) language is NOT allowed by WS-BP 1.1.

Option 5 is incorrect, Schematron is an ISO/IEC Standard, not W3C recommendation. Schematron is a rule-based validation language for making assertions about the presence or absence of patterns in XML trees.

While Schematron is good at relational constructs, its ability to specify the basic structure of a document, that is, which elements can go where, results in a very verbose schema.

The typical way to solve this is to combine Schematron with RELAX NG or W3C XML Schema. There are several schema processors available for both languages that support this combined form. This allows Schematron rules to specify additional constraints to the structure defined by W3C XML Schema or RELAX NG.

Sources:

WS-I Basic Profile 1.1 [4.8 Use of XML Schema] [http://www.ws-i.org/profiles/basicprofile-1.1-2004-08-24.html]

XML Schema Requirements [http://www.w3.org/TR/NOTE-xml-schema-req]

Part II. Appendices

Appendix 1. XML Web Service Standards

1.1. Given XML documents, schemas, and fragments determine whether their syntax and form are correct (according to W3C schema) and whether they conform to the WS-I Basic Profile 1.1.

Question A010101

Snorcle, Inc. is creating an XML schema that describes various Java training materials available for purchase by students. Which XSD fragments correctly describe training material author's name, consisting of a first name and a last name?

Options (select 2):

1.
```
<xs:element name="author">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="firstName" type="xs:string" minOccurs="1"/>
            <xs:element name="lastName" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

2.
```
<xs:element name="author">
    <xs:complexType>
        <xs:choice>
            <xs:element name="firstName" type="xs:string"/>
            <xs:element name="lastName" type="xs:string" minOccurs="1"/>
        </xs:choice>
    </xs:complexType>
```

```
</xs:element>
```

3.
```
<xs:element name="author">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="firstName" type="xs:string" maxOccurs="1"/>
            <xs:element name="lastName" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

4.
```
<xs:element name="author">
    <xs:complexType>
        <xs:choice>
            <xs:element name="firstName" type="xs:string"/>
            <xs:element name="lastName" type="xs:string"/>
        </xs:choice>
    </xs:complexType>
</xs:element>
```

5.
```
<xs:element name="author">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="firstName" type="xs:string" minOccurs="0"/>
            <xs:element name="lastName" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

6.
```
<xs:element name="author">
    <xs:complexType>
        <xs:all>
            <xs:element name="firstName" type="xs:string"/>
            <xs:element name="lastName" type="xs:string" maxOccurs="5"/>
        </xs:all>
    </xs:complexType>
</xs:element>
```

Answer:

Correct options are 1 and 3.

The `<sequence>` indicator specifies that the child elements must appear in a specific order (i.e. `<firstName>`, `<lastName>`).

For all "Order" and "Group" indicators (`any`, `all`, `choice`, `sequence`, `group` name, and `group` reference) the default value for `maxOccurs` and `minOccurs` is 1.

Options 2 and 4 are wrong: the `<choice>` indicator specifies that either ONE child element OR another can occur.

Option 5 is wrong: the `minOccurs="0"` means that element is optional, but name must consist of a first name and a last name.

Option 6 is wrong: when using the `<all>` indicator you can set the `minOccurs` indicator to 0 or 1 and the `maxOccurs` indicator can only be set to 1. Another problem is that according to the requirement, only one `<lastName>` may present, and XSD fragment allows 5 `<lastName>` elements.

Sources:

XML Schema Part 0: Primer Second Edition - [`http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/`]

XSD Indicators - [`http://www.w3schools.com/Schema/schema_complex_indicators.asp`]

Question A010102

Consider the following SOAP message fragment:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
        soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <soap:Body xmlns:ns1="http://example">
                ...
    </soap:Body>
</soap:Envelope>
```

Is the message WS-I BP 1.1 conformant?

Options (select 1):

1. Yes, it is comformant for all styles of SOAP messages.
2. Yes, it is comformant for "rpc-encoded" binding only.
3. Yes, it is comformant for "document-encoded" binding only.
4. Yes, it is comformant for "rpc-encoded" and "document-encoded" bindings only.
5. No, it is not comformant.

Answer:

Correct option is 5.

The `soap:encodingStyle` attribute is used to indicate the use of a particular scheme in the encoding of data into XML. However, this introduces complexity, as this function can also be served by the use of XML Namespaces. As a result, the Basic Profile prefers the use of literal, non-encoded XML.

R1005 A MESSAGE MUST NOT contain `soap:encodingStyle` attributes on any of the elements whose namespace name is "`http://schemas.xmlsoap.org/soap/envelope/`".

R1006 A MESSAGE MUST NOT contain `soap:encodingStyle` attributes on any element that is a child of `soap:Body`.

R1007 An ENVELOPE described in an rpc-literal binding MUST NOT contain `soap:encodingStyle` attribute on any element that is a grandchild of `soap:Body`.

Options 2, 3 and 4 are also wrong because WS-I BP 1.1 explicitly prohibits the rpc-encoded and document-encoded modes.

Sources:

Basic Profile Version 1.1 - [`http://www.ws-i.org/Profiles/BasicProfile-1.1.html`]

Question A010103

Which of the following WSDL Schema fragments contains a WS-I Basic Profile 1.1-conformant array declaration?

Options (select 1):

1.
```
<element name="myFavoriteStrings" type="ArrayOfStrings"/>

<complexType name="ArrayOfStrings">
    <complexContent>
```

```
        <restriction base="SOAP-ENC:Array">
            <sequence>
                <element name="x" type="string" minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
            <attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="string[]"/>
        </restriction>
    </complexContent>
</complexType>
```

2.

```
<element name="myFavoriteStrings" type="MyArrayType"/>

<xsd:complexType name="MyArrayType">
    <xsd:sequence>
        <xsd:element name="x" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="string[]"/>
</xsd:complexType>
```

3.

```
<element name="myFavoriteStrings" type="MyArrayType"/>

<xsd:complexType name="MyArrayType">
    <xsd:sequence>
        <xsd:element name="x" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
```

4.

```
<element name="myFavoriteStrings" type="ArrayOfStrings"/>

<complexType name="ArrayOfStrings">
    <complexContent>
        <restriction base="SOAP-ENC:Array">
            <sequence>
                <element name="x" type="string" minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
        </restriction>
    </complexContent>
</complexType>
```

Answer:

Correct option is 3.

The recommendations in WSDL 1.1 Section 2.2 for declaration of array types have been interpreted in various ways, leading to interoperability problems. Further, there are other clearer ways to declare arrays.

R2110 In a DESCRIPTION, declarations MUST NOT extend or restrict the `soapenc:Array` type.

This means options 1 and 4 are wrong.

R2111 In a DESCRIPTION, declarations MUST NOT use `wsdl:arrayType` attribute in the type declaration.

This means option 1 and 2 are wrong.

R2112 In a DESCRIPTION, elements SHOULD NOT be named using the convention `ArrayOfXXX`.

Options 1 and 4 violate this requirement.

R2113 An ENVELOPE MUST NOT include the `soapenc:arrayType` attribute.

Sources:

Basic Profile Version 1.1 - [`http://www.ws-i.org/Profiles/BasicProfile-1.1.html`]

Question A010104

Given the XML Schema fragment:

```
<xsd:element name="Customer">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="CustomerID" type="xsd:string"/>
            <xsd:element name="CompanyName" type="xsd:string"/>
            <xsd:element name="ContactName" type="xsd:string" nillable="true"/>
            <xsd:element name="Address" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Which XML snippets are valid?

Options (select 3):

1.
```
<Customer>
    <CustomerID>123456</CustomerID>
    <CompanyName>IBM</CompanyName>
    <ContactName xsi:nil="1"></ContactName>
    <Address>5600 63rd Street</Address>
</Customer>
```

2.
```
<Customer>
    <CustomerID>123456</CustomerID>
    <CompanyName>IBM</CompanyName>
    <ContactName>Mikalai Zaikin</ContactName>
    <Address>5600 63rd Street</Address>
</Customer>
```

3.
```
<Customer>
    <CustomerID>123456</CustomerID>
    <CompanyName>IBM</CompanyName>
    <ContactName xsi:nil="true">Mikalai Zaikin</ContactName>
    <Address>5600 63rd Street</Address>
</Customer>
```

4.
```
<Customer>
    <CustomerID>123456</CustomerID>
    <CompanyName xsi:nil="true" />
    <ContactName xsi:nil="true" />
    <Address>5600 63rd Street</Address>
</Customer>
```

5.

```
<Customer>
    <CustomerID>123456</CustomerID>
    <CompanyName>IBM</CompanyName>
    <ContactName xsi:nil="true" />
    <Address>5600 63rd Street</Address>
</Customer>
```

Answer:

Correct options are 1, 2 and 5.

We can override the schema declaration when setting "`xsi:nil`" attribute to value "true" (or "1"). The schema must allow this by setting attribute "`nillable`" to "`true`" (default value is "`false`").

Option 1 is correct. The "`xsi:nil`" attribute set to "1" allows the element to be empty.

Option 2 is correct. The `ContactName` element contains text data

Option 3 is wrong. This document is not valid, because when "`xsi:nil`" is "`true`", the `ContactName` element must be empty.

Option 4 is wrong. This document is not valid, because "`nillable`" attribute in XML Schema has default value "`false`", so `CompanyName` can not contain "`xsi:nil`" attribute.

Option 5 is correct. The "`xsi:nil`" attribute set to "`true`" allows the element to be empty.

NOTE: The XML Schema specification says the `xsi:nil` attribute is of type `boolean`, and boolean allows "`0`" as equivalent to "`false`" and "`1`" as equivalent to "`true`".

Sources:

XML Schema Part 0 - [`http://www.w3.org/TR/xmlschema-0/#Nils`]

XML Schema Part 1 - [`http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/#xsi_nil`]

Question A010105

Given the `sample.xsd` XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    targetNamespace="http://www.example.com">
<xsd:element name="Customer">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="CustomerID" type="xsd:string"/>
            <xsd:element name="CompanyName" type="xsd:string"/>
            <xsd:element name="ContactName" type="xsd:string"/>
            <xsd:element name="Address" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Which XML instance is valid?

Options (select 1):

1.
```
<?xml version="1.0" encoding="UTF-8"?>
<Customer xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.example.com sample.xsd ">
  <CustomerID>123456</CustomerID>
  <CompanyName>IBM</CompanyName>
  <ContactName>Mikalai Zaikin</ContactName>
  <Address>5600 63rd Street</Address>
</Customer>
```

2.
```
<?xml version="1.0" encoding="UTF-8"?>
<ns:Customer xmlns:ns="http://www.example.com"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.example.com sample.xsd ">
  <CustomerID>123456</CustomerID>
  <CompanyName>IBM</CompanyName>
  <ContactName>Mikalai Zaikin</ContactName>
  <Address>5600 63rd Street</Address>
</ns:Customer>
```

3.
```
<?xml version="1.0" encoding="UTF-8"?>
<Customer xmlns:ns="http://www.example.com"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.example.com sample.xsd ">
  <ns:CustomerID>123456</ns:CustomerID>
  <ns:CompanyName>IBM</ns:CompanyName>
  <ns:ContactName>Mikalai Zaikin</ns:ContactName>
  <ns:Address>5600 63rd Street</ns:Address>
</Customer>
```

4.
```
<?xml version="1.0" encoding="UTF-8"?>
<ns:Customer xmlns:ns="http://www.example.com"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.example.com sample.xsd ">
  <ns:CustomerID>123456</ns:CustomerID>
  <ns:CompanyName>IBM</ns:CompanyName>
  <ns:ContactName>Mikalai Zaikin</ns:ContactName>
  <ns:Address>5600 63rd Street</ns:Address>
</ns:Customer>
```

Answer:

Correct option is 4.

While global elements and attributes must always be qualified, local elements may not need to be qualified. Qualification of local elements and attributes can be globally specified by a pair of attributes, "`elementFormDefault`" and "`attributeFormDefault`", on the "`schema`" element, or can be specified separately for each local declaration using the "`form`" attribute. All such attributes' values may each be set to "`unqualified`" or "`qualified`", to indicate whether or not locally declared elements and attributes must be unqualified.

Option 1 and 3 are wrong. Global elements must be namespace qualified.

Option 2 is wrong. The document instance XML Schema definition requires local elements be

namespace qualified.

Sources:

XML Schema Part 0 (Section 3.2 Qualified Locals) - [`http://www.w3.org/TR/xmlschema-0/#QualLocals`]

Question A010106

Given the following document-literal SOAP message:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:ns1="http://e-shop.com">
    <soapenv:Body>
        <ns1:name>Sony Reader Digital Book PRS-505</ns1:name>
        <ns1:color>Silver</ns1:color>
        <ns1:price >260.99</ns1:price>
    </soapenv:Body>
</soapenv:Envelope>
```

Which is true?

Options (select 1):

1. Yes, it is WS-I BP 1.1 compliant.
2. No, it is not WS-I BP 1.1 compliant, because document-literal enconding is restricted.
3. No, it is not WS-I BP 1.1 compliant, because message may not have more than one child elements of the SOAP `Body`.
4. No, it is not WS-I BP 1.1 compliant, because children of the SOAP `Body` may not be namespace qualified.

Answer:

Correct option is 3.

This question provides an example of a SOAP message using a document-literal **bare** style. The style does not use SOAP encoding rules completely and embeds an XML document based on a schema into the SOAP message.

Option 1 is wrong. The SOAP message is not WS-I BP 1.1 compliant.

Option 2 is wrong. WS-I Basic Profile 1.1 precludes the use of SOAP encoding, largely because it has proven to be a constant source of interoperability problems. The WS-I Basic Profile therefore requires use of either the rpc-literal or document-literal forms of the WSDL SOAP binding.

WS-I BP 1.1: R2706 A `wsdl:binding` in a DESCRIPTION MUST use the value of "`literal`" for the `use` attribute in all `soapbind:body`, `soapbind:fault`, `soapbind:header` and `soapbind:headerfault` elements.

Option 3 is correct. WS-I BP 1.1: R9981 An ENVELOPE MUST have exactly zero or one child elements of the `soap:Body` element. There are three child elements in the code example.

You can use document-literal pattern to make SOAP WS-I BP 1.1 compliant. In **wrapped** document style, each message has a single part. The message's part references a wrapper element defined in the `types` element of the WSDL contract.

Option 4 is wrong. The use of unqualified element names may cause naming conflicts, therefore

qualified names must be used for the children of SOAP `Body`.

WS-I BP 1.1: R1014 The children of the `soap:Body` element in an ENVELOPE MUST be namespace qualified.

Sources:

WS-I BP 1.1 (SOAP Envelope Structure, SOAP Body Namespace Qualification) - [`http://www.ws-i.org/Profiles/BasicProfile-1.1.html`]

Question A010107

Given the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:ns1="http://e-shop.com"
        xmlns:wsr="http://routing.com">
    <soapenv:Header>
            <wsr:party id="uuid:791F4B43453573FB7D1204876983384">
    </soapenv:Header>
    <soapenv:Body>
        <ns1:order>
            <ns1:name>Sony Reader Digital Book PRS-505</ns1:name>
            <ns1:color>Silver</ns1:color>
            <ns1:price>260.99</ns1:price>
        </ns1:order>
    </soapenv:Body>
</soapenv:Envelope>
```

Which describes this SOAP message?

Options (select 1):

1. The message is WS-I BP 1.1 compliant.
2. The `ns1` namespace is not correctly declared.
3. The `wsr` namespace is not correctly declared.
4. None of the above.

Answer:

Correct option is 4.

The SOAP message is not well-formed XML. The `wsr:party` element in message header is invalid empty tag.

XML element is a logical component of a document which either begins with a start-tag and ends with an end-tag, or consists only of an empty-element tag. The characters between the start- and end-tags, if any, are the element's content, and may contain markup, including other elements, which are called child elements. An example of an element is `<ns1:name>Sony Reader Digital Book PRS-505</ns1:name>`. Example of empty element is `<wsr:party id="uuid:791F4B43453573FB7D1204876983384" />`.

The XML specification defines an XML document as a text which is well-formed, i.e., it satisfies a list of syntax rules provided in the specification. The list is fairly lengthy; some key points are:

- It contains only properly-encoded legal Unicode characters.
- None of the special syntax characters such as "<" and "&" appear except when performing

their markup-delineation roles.

- The begin, end, and empty-element tags which delimit the elements are correctly nested, with none missing and none overlapping.

  Every start tag must have a corresponding end tag; for example, `<tag>content</tag>`. If the tag does not wrap any content, the end tag can be replaced with the shorthand "`/>`" notation; for example, `<tag />`.

- The element tags are case-sensitive; the beginning and end tags must match exactly.
- There is a single "root" element which contains all the other elements.

Option 1 is wrong. The SOAP message is not WS-I BP 1.1 compliant (because XML is not well-formed).

Options 2 and 3 are wrong. All namespaces are declared correctly.

Sources:

Extensible Markup Language (XML) 1.0 - [`http://www.w3.org/TR/2000/REC-xml-20001006`]

Question A010108

Which SOAP message is WS-I BP 1.1 conformant?

Options (select 1):

1.
```
<soap:Envelope xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'>
    <soap:Body>
        <p:Process xmlns:p='http://example.org/operations' />
    </soap:Body>
    <m:Data xmlns:m='http://example.org/information' >
        <wsi:Claim conformsTo="http://ws-i.org/profiles/basic1.1/"
            xmlns:wsi="http://ws-i.org/schemas/conformanceClaim/" />
    </m:Data>
</soap:Envelope>
```

2.
```
<soap:Envelope xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'>
    <m:Data xmlns:m='http://example.org/information' >
        <wsi:Claim conformsTo="http://ws-i.org/profiles/basic1.1/"
            xmlns:wsi="http://ws-i.org/schemas/conformanceClaim/" />
    </m:Data>
    <soap:Body>
        <p:Process xmlns:p='http://example.org/operations' />
    </soap:Body>
</soap:Envelope>
```

3.
```
<soap:Envelope xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'>
    <soap:Header>
        <wsi:Claim conformsTo="http://ws-i.org/profiles/basic1.1/"
            xmlns:wsi="http://ws-i.org/schemas/conformanceClaim/" />
    </soap:Header>
    <m:Data xmlns:m='http://example.org/information' >
        <t:Trans xmlns:t="http://http://example.org/transaction/">
            15091974
        </t:Trans>
    </m:Data>
</soap:Envelope>
```

4.

```
<soap:Envelope xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'>
    <m:Data xmlns:m='http://example.org/information' >
        <t:Trans xmlns:t="http://http://example.org/transaction/">
            15091974
        </t:Trans>
    </m:Data>
    <soap:Header>
        <wsi:Claim conformsTo="http://ws-i.org/profiles/basic1.1/"
            xmlns:wsi="http://ws-i.org/schemas/conformanceClaim/" />
    </soap:Header>
</soap:Envelope>
```

5. None of the above.

Answer:

Correct option is 5. None of the shown SOAP messages is WS-I BP 1.1 compliant.

Option 1 is wrong. According to WS-I BP 1.1, the interpretation of sibling elements following the `soap:Body` element is unclear. Therefore, such elements are disallowed.

R1011 An ENVELOPE MUST NOT have any element children of `soap:Envelope` following the `soap:Body` element.

This requirement clarifies a mismatch between the SOAP 1.1 specification and the SOAP 1.1 XML Schema.

Correct example of SOAP message could be:

```
<soap:Envelope xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'>
    <soap:Body>
        <p:Process xmlns:p='http://example.org/operations'>
            <m:Data xmlns:m='http://example.org/information' >
                <wsi:Claim conformsTo="http://ws-i.org/profiles/basic1.1/"
                    xmlns:wsi="http://ws-i.org/schemas/conformanceClaim/" />
            </m:Data>
        </p:Process>
    </soap:Body>
</soap:Envelope>
```

Options 2, 3 and 4 are wrong. According to SOAP 1.1 specification, the `soap:Body` element MUST be present in a SOAP message and MUST be an immediate child element of a `soap:Envelope` element. It MUST directly follow the `soap:Header` element if present. Otherwise it MUST be the first immediate child element of the `soap:Envelope` element.

Sources:

WS-I Basic Profile Version 1.1 - [`http://www.ws-i.org/Profiles/BasicProfile-1.1.html`]

SOAP 1.1 specification - [`http://www.w3.org/TR/2000/NOTE-SOAP-20000508`]

1.2. Describe the use of XML schema in Java EE Web services.

Question A010201

Which statement is true about WSDL definition?

Options (select 1):

1. XML Schema is used as a formal definition of WSDL grammar.
2. XML Document Type Definition is used as a formal definition of WSDL grammar.
3. XML Stylesheet Language Transformation is used as a formal definition of WSDL grammar.
4. Document Object Model is used as a formal definition of WSDL grammar.
5. None of the above.

Answer:

Correct option is 1.

XSD schemas are provided as a formal definition of WSDL grammar.

XML Schema: Structures specifies the XML Schema definition language, which offers facilities for describing the structure and constraining the contents of XML 1.0 documents, including those which exploit the XML Namespace facility. The schema language, which is itself represented in XML 1.0 and uses namespaces, substantially reconstructs and considerably extends the capabilities found in XML 1.0 document type definitions (DTDs). Unlike DTD, XML Schema has support for datatypes, and XML Schema uses XML Syntax.

Option 2 is wrong, because WSDL 1.1 specification uses only XML Scmemas (XSD) as a formal definition of WSDL grammar.

Option 3 is wrong. XSLT is the style sheet language for XML files. With XSLT you can transform XML documents into other formats, like XHTML.

Option 4 is wrong. The XML DOM defines a standard way for accessing and manipulating XML documents. The XML DOM is platform and language independent and can be used by any programming language like Java, JavaScript, and VBScript.

Source:

Web Services Description Language (WSDL) 1.1 (Section 1.2: Notational Conventions) - [http://www.w3.org/TR/2001/NOTE-wsdl-20010315]

Appendix 2. SOAP 1.2 Web Service Standards

2.1. List and describe the encoding types used in a SOAP message.

Warning

The OCE WSD 6 exam questions are still based on the SOAP 1.1 specification.

Question A020101

Consider the following Java code and SOAP 1.1 message:

```
Name name = new Name("Mikalai", "Zaikin");
Author author = new Author(name);
Book book1 = new Book("SCDJWS 5.0 Guide", author);
Book book2 = new Book("OCE WSD 6 Guide", author);
// get Web Service endpoint interface [SEI]
...
boolean b = sei.compare(book1, book2);
```

```
<soap:Envelope xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'
       soap:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'>
```

```
    <soap:Body xmlns:ns1='http://example'>

        <ns1:compare>
            <e:book>
                <title>SCDJWS 5.0 Guide</title>
                <!-- 1 -->
            </e:book>
            <e:book>
                <title>OCE WSD 6 Guide</title>
                <!-- 2 -->
            </e:book>
        </ns1:compare>

        <!-- 3 -->
            <name>
                <fistName>Mikalai</firstName>
                <lastName>Zaikin</lastName>
            </name>
        </ns1:author>

    </soap:Body>

</soap:Envelope>
```

How can you refer to the same `author` element in several places ?

Options (select 1):

1.
```
1. <author href="#mz" />
2. <author href="#mz" />
3. <ns1:author id="mz" >
```

2.
```
1. <author href="#mz" />
2. <author href="#mz" />
3. <ns1:author cid="mz" >
```

3.
```
1. <author href="mz" />
2. <author href="mz" />
3. <ns1:author id="mz" >
```

4.
```
1. <author ref="mz" />
2. <author ref="mz" />
3. <ns1:author id="mz" >
```

Answer:

Correct option is 1.

In cases when a field in a data structure is referred to in several places in that data structure (for example, in a doubly linked list), then the field is serialized as an independent element, an immediate child element of `Body`, and must have an `id` attribute of type `xsd:ID`. Such elements are called multireference accessors. They provide access to the data in the field from multiple locations

in the message. Each reference to the field in the data structure is serialized as an empty element with an `href` attribute of type `xsd:anyURI`, where the value of the attribute contains the identifier specified in the `id` attribute on the multireference accessor preceded by a fragment identifier, # (pound sign).

A multi-reference simple or compound value is encoded as an independent element containing a local, unqualified attribute named "`id`" and of type "`ID`" per the XML Specification. Each accessor to this value is an empty element having a local, unqualified attribute named "`href`" and of type "`uri-reference`" per the XML Schema Specification, with a "`href`" attribute value of a URI fragment identifier referencing the corresponding independent element.

Sources:

MSDN Magazine (March 2000) - [`http://msdn.microsoft.com/en-us/magazine/bb985060.aspx`]

MSDN SOAP Technical Article - [`http://msdn.microsoft.com/en-us/library/ms995710.aspx`]

Simple Object Access Protocol (SOAP) 1.1 (5.4.1 Compound Values, Structs and References to Values) - [`http://www.w3.org/TR/2000/NOTE-SOAP-20000508/#_Toc478383520`]

Question A020102

Determine in the following XML Schema fragments which "`code`" element/attribute type maps to Java wrapper class or Java primitive:

1.
```
<xsd:element name="code" type="xsd:short" />
```

2.
```
<xsd:element name="code" type="xsd:short" maxOccurs="1" />
```

3.
```
<xsd:element name="code" type="xsd:short" minOccurs="0" />
```

4.
```
<xsd:element name="code" type="xsd:short" minOccurs="0" maxOccurs="1" />
```

5.
```
<xsd:element name="code" type="xsd:short" nillable="true" />
```

6.
```
<xsd:element name="description">
        <xsd:complexType>
                <xsd:sequence />
                <xsd:attribute name="code" type="xsd:short" use="optional"/>
        </xsd:complexType>
</xsd:element>
```

7.
```
<xsd:element name="description">
        <xsd:complexType>
```

```
                <xsd:sequence />
                <xsd:attribute name="code" type="xsd:short" default="0"/>
        </xsd:complexType>
</xsd:element>
```

Options:

a) `java.lang.Short`

b) `short`

Fill your answer here:

```
1 - [  ]; 2 - [  ]; 3 - [  ]; 4 - [  ]; 5 - [  ]; 6 - [  ]; 7 - [  ]
```

Answer:

Correct answer is: 1 - b; 2 - b; 3 - a; 4 - a; 5 - a; 6 - a; 7 - b.

The following paragraph specifies the Java mapping for the built-in simple XML data types. These XML data types are as defined in the XML schema specification and the SOAP 1.1 encoding [`http://schemas.xmlsoap.org/soap/encoding/`].

```
xsd:string            String
xsd:integer           BigInteger
xsd:int               int
xsd:long              long
xsd:short             short
xsd:decimal           BigDecimal
xsd:float             float
xsd:double            double
xsd:boolean           boolean
xsd:byte              byte
xsd:QName             QName
xsd:dateTime          XMLGregorianCalendar
xsd:base64Binary      byte[]
xsd:hexBinary         byte[]
xsd:unsignedInt       long
xsd:unsignedShort     int
xsd:unsignedByte      short
xsd:time              XMLGregorianCalendar
xsd:date              XMLGregorianCalendar
xsd:g                 XMLGregorianCalendar
```

Mapping XML Schema primitive types to Java primitive types does not work for all possible XML Schema constructs. Several cases require that an XML Schema primitive type is mapped to the Java primitive type's corresponding wrapper type. These cases include:

1. an `element` declaration with the `nillable` attribute set to `true` as shown:

```
<xsd:element name="code" type="xsd:int" nillable="true"/>
```

2. an `element` declaration with the `minOccurs` attribute set to 0 (zero) and the `maxOccurs` attribute set to 1 (one) or absent, as shown:

```
<xsd:element name="code2" type="xsd:int" minOccurs="0"/>
```

3. an `attribute` declaration with the "`use`" attribute set to "`optional`" or absent and carrying neither the "`default`" nor the "`fixed`" attribute, as shown:

```
<xsd:element name="description">
    <xsd:complexType>
        <xsd:sequence />
        <xsd:attribute name="code3" type="xsd:int" use="optional"/>
    </xsd:complexType>
</xsd:element>
```

The `element`/`attribute` declarations for "`code`", "`code2`", "`code3`" above are all mapped to the `java.lang.Integer` type.

The following paragraph shows how XML Schema primitive types are mapped into Java **wrapper classes** in these cases:

```
xsd:int             java.lang.Integer
xsd:long            java.lang.Long
xsd:short           java.lang.Short
xsd:float           java.lang.Float
xsd:double          java.lang.Double
xsd:boolean         java.lang.Boolean
xsd:byte            java.lang.Byte
xsd:unsignedByte    java.lang.Short
xsd:unsignedShort   java.lang.Integer
xsd:unsignedInt     java.lang.Long
xsd:unsignedLong    java.math.BigInteger
```

Sources:

JSR-000101 Java(TM) API for XML-Based RPC 1.1 Specification (page 34) - [`http://jcp.org/en/jsr/detail?id=101`]

Question A020103

Compose the correct SOAP message from following elements.

Options:

1.
```
<soap:Headers>
```

2.
```
</soap:Header>
```

3.
```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
```

4.
```
</soap:Envelope>
```

5.

```
<soap:Header>
```

6.
```
</soap:Headers>
```

7.
```
<soap:Body>
```

8.
```
</soap:Body>
```

9.
```
<soap:Message xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
```

10.
```
</soap:Message>
```

11.
```
<soap:Content>
```

12.
```
</soap:Content>
```

Fill your answer here:

```
[       ]
    [       ]
        ...
    [       ]
    [       ]
        ...
    [       ]
[       ]
```

Answer:

Correct answer is: 3, 5, 2, 7, 8, 4.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header>
        ...
    </soap:Header>
    <soap:Body>
        ...
    </soap:Body>
</soap:Envelope>
```

A SOAP message consists of 3 main elements:

- `Envelope` - The `Envelope` element, as its name would suggest, serves as a container for the other elements of the SOAP message. The `Envelope` element is always the root element of a SOAP message.
- `Body` - The `Body` element of a SOAP message is the location for application-specific data. It contains the payload of the message, carrying the data that represents the purpose of the message. It could be a remote procedure call, a purchase order, a stylesheet, or any XML that needs to be exchanged using a message.

  The `Body` element must appear as an immediate child of the `Envelope` element. If there is no `Header` element, then the `Body` element is the first child; if a `Header` element does appear in the message, then the `Body` element immediately follows it. The payload of the message is represented as child elements of `Body` element.
- `Header` - The `Header` element is to encapsulate extensions to the message format without having to couple them to the payload or to modify the fundamental structure of SOAP. This allows extensions like transactions, encryption, object references, billing, and countless others to be added over time without breaking the specification.

  Like the `Body` element, the `Header` element must appear as an immediate child of the `Envelope` element. It is optional, but if it does appear, it must appear as the first child. The `Header` element contains one or more child elements known as header entries.

Sources:

Simple Object Access Protocol (SOAP) 1.1 (W3C Note) - [`http://www.w3.org/TR/2000/NOTE-SOAP-20000508/`]

2.2. Describe the SOAP Processing and Extensibility Model.

Question A020201

When calling Web Service you received following SOAP message.

```
<soap:Fault xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/' >
    <faultcode>soap:Server</faultcode>
    <faultstring>Database is not accessible</faultstring>
</soap:Fault>
```

How can you get "`faultcode`" element value?

Options (select 2):

1.
```
SOAPBody body = message.getSOAPBody();
if (body.hasFault()) {
    SOAPFault fault = body.getFault();
    javax.xml.soap.Name code = fault.getFaultCodeAsName();
    ...
}
```

2.
```
SOAPBody body = message.getSOAPBody();
if (body.hasFault()) {
    SOAPFault fault = body.getFault();
    String code = fault.getElementByTagName("faultcode");
    ...
```

```
}
```

3.
```
SOAPBody body = message.getSOAPBody();
if (body.hasFault()) {
    SOAPFault fault = body.getFault();
    org.w3c.dom.Node faultCode = fault.getElementByTagName("faultcode");
    ...
}
```

4.
```
SOAPBody body = message.getSOAPBody();
if (body.hasFault()) {
    SOAPFault fault = body.getFault();
    org.w3c.dom.NodeList faultCodes = fault.getElementsByTagName("faultcode");
    ...
}
```

Answer:

Correct options are 1 and 4.

The `SOAPFault` has the following methods which allow access its elements:

```
package javax.xml.soap;

public interface SOAPFault extends SOAPBodyElement {
    public abstract String getFaultCode();
    public abstract Name   getFaultCodeAsName();
    public abstract String getFaultString();
    public abstract String getFaultActor();
    public abstract Detail getDetail();
        ...
}
```

The `getFaultCodeAsName()` method gets the mandatory SOAP 1.1 fault code for this `SOAPFault` object as a SAAJ `Name` object. The SOAP 1.1 specification requires the value of the "`faultcode`" element to be of type `QName`. This method returns the content of the element as a `QName` in the form of a SAAJ `Name` object. This method should be used instead of the `getFaultCode` method since it allows applications to easily access the namespace name without additional parsing.

`SOAPFault` also has an `org.w3c.dom.Element` as a parent interface in its hierarchy, which provides method `public NodeList getElementsByTagName(String name)`

It returns a `NodeList` of all descendant `Element`s with a given tag name, in the order in which they are encountered in a preorder traversal of this `Element` tree. Method parameter is the name of the tag to match on. The special value "`*`" matches all tags.

Sources:

SOAP API JavaDoc - [`http://download.oracle.com/javase/6/docs/api/javax/xml/soap/SOAPFault.html#getFaultCodeAsName()`]

DOM Level 2. [Appendix D: Java Language Binding] - [`http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/java-binding.html`]

Question A020202

What are optional child elements of SOAP Fault?

Options (select 2):

1. `faultcode`
2. `faultstring`
3. `faultactor`
4. `detail`

Answer:

Correct options are 3 and 4. Only `faultcode` and `faultstring` elements MUST be present in SOAP Fault.

The `faultactor` element must be present only if fault was generated by not ultimate destination node. The ultimate destination node may OPTIONALLY include this element in fault.

The `detail` element MUST be included only when fault was caused by SOAP body content. It MUST NOT be included when fault was caused by SOAP header.

Below is a SOAP Fault message example when ultimate destination node did not understand mandatory header (NOTE, the `faultcode` element value belongs to SOAP `Envelope` namespace):

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
       <SOAP-ENV:Fault>
           <faultcode>SOAP-ENV:MustUnderstand</faultcode>
           <faultstring>SOAP Header Must Understand Error</faultstring>
       </SOAP-ENV:Fault>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Specification (4.4 SOAP Fault):

The SOAP Fault element is used to carry error and/or status information within a SOAP message. If present, the SOAP Fault element MUST appear as a body entry and MUST NOT appear more than once within a Body element.

The SOAP Fault element defines the following four subelements:

`faultcode` - The faultcode element is intended for use by software to provide an algorithmic mechanism for identifying the fault. The faultcode MUST be present in a SOAP Fault element and the faultcode value MUST be a qualified name. SOAP defines a small set of SOAP fault codes covering basic SOAP faults (`VersionMismatch`, `MustUnderstand`, `Client`, `Server`).

`faultstring` - The faultstring element is intended to provide a human readable explanation of the fault and is not intended for algorithmic processing. The faultstring element is similar to the 'Reason-Phrase' defined by HTTP. It MUST be present in a SOAP Fault element and SHOULD provide at least some information explaining the nature of the fault.

`faultactor` - The faultactor element is intended to provide information about who caused the fault to happen within the message path. It is similar to the SOAP actor attribute, but instead of indicating the destination of the header entry, it indicates the source of the fault. The value of the faultactor attribute is a URI identifying the source. Applications that do not act as the ultimate destination of the SOAP message MUST include the faultactor element in a SOAP Fault element. The ultimate destination of a message MAY use the faultactor element to indicate explicitly that it generated the fault.

`detail` - The detail element is intended for carrying application specific error information related to the Body element. It MUST be present if the contents of the Body element could not be successfully processed. It MUST NOT be used to carry information about error information belonging to header entries. Detailed error information belonging to header entries MUST be carried within header

entries. The absence of the detail element in the Fault element indicates that the fault is not related to processing of the Body element. This can be used to distinguish whether the Body element was processed or not in case of a fault situation.

Sources:

Simple Object Access Protocol (SOAP) 1.1 [4.4 SOAP Fault] - [`http://www.w3.org/TR/2000/NOTE-SOAP-20000508`]

Question A020203

According to WS-I Basic Profile 1.1, what are valid child elements of a `soap:Fault` element?

Options (select 3):

1. `detail`
2. `actor`
3. `faultcode`
4. `faultdetail`
5. `faultstring`

Answer:

Correct options are 1, 3 and 5.

SOAP Fault Structure [WS-I BP 1.1]

A SOAP Fault is a SOAP message that has a single child element of the `soap:Body` element, that element being a `soap:Fault` element.

R1000 When an ENVELOPE is a Fault, the `soap:Fault` element MUST NOT have element children other than `faultcode`, `faultstring`, `faultactor` and `detail`.

The SOAP Fault element defines the following four sub-elements:

`faultcode` - The faultcode element is intended for use by software to provide an algorithmic mechanism for identifying the fault. The faultcode MUST be present in a SOAP Fault element and the faultcode value MUST be a qualified name. SOAP defines a small set of SOAP fault codes covering basic SOAP faults.

`faultstring` - The faultstring element is intended to provide a human readable explanation of the fault and is not intended for algorithmic processing. The faultstring element is similar to the 'Reason-Phrase' defined by HTTP. It MUST be present in a SOAP Fault element and SHOULD provide at least some information explaining the nature of the fault.

`faultactor` - The faultactor element is intended to provide information about who caused the fault to happen within the message path. It is similar to the SOAP actor attribute but instead of indicating the destination of the header entry, it indicates the source of the fault. The value of the faultactor attribute is a URI identifying the source. Applications that do not act as the ultimate destination of the SOAP message MUST include the faultactor element in a SOAP Fault element. The ultimate destination of a message MAY use the faultactor element to indicate explicitly that it generated the fault.

`detail` - The detail element is intended for carrying application specific error information related to the Body element. It MUST be present if the contents of the Body element could not be successfully processed. It MUST NOT be used to carry information about error information belonging to header entries. Detailed error information belonging to header entries MUST be carried within header entries. The absence of the detail element in the Fault element indicates that the fault is not related to processing of the Body element. This can be used to distinguish whether the Body element was processed or not in case of a fault situation.

Sources:

WS-I Basic Profile 1.1 [SOAP Fault Structure] - [http://www.ws-i.org/Profiles/BasicProfile-1.1.html]

Simple Object Access Protocol (SOAP) 1.1 [4.4 SOAP Fault] - [http://www.w3.org/TR/2000/NOTE-SOAP-20000508]

Question A020204

According to WS-I Basic Profile 1.1, which `soap:Fault` element is valid?

Options (select 1):

1.
```
<soap:Fault xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" >
  <soap:faultcode>soap:Client</soap:faultcode>
  <soap:faultstring>...</soap:faultstring>
  <soap:faultactor>...</soap:faultactor>
  <soap:detail>...</soap:detail>
</soap:Fault>
```

2.
```
<soap:Fault xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" >
  <faultcode>soap:Client</faultcode>
  <faultstring>...</faultstring>
  <faultactor>...</faultactor>
  <soap:detail>...</soap:detail>
</soap:Fault>
```

3.
```
<soap:Fault xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns="">
  <faultcode>Client</faultcode>
  <faultstring>...</faultstring>
  <faultactor>...</faultactor>
  <detail>...</detail>
</soap:Fault>
```

4.
```
<soap:Fault xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns="">
  <faultcode>soap:Client</faultcode>
  <faultstring>...</faultstring>
  <faultactor>...</faultactor>
  <detail>...</detail>
</soap:Fault>
```

Answer:

Correct option is 4.

3.3 SOAP Faults [WS-I BP 1.1]

The children of the `soap:Fault` element are local to that element, therefore namespace qualification is unnecessary.

R1001 When an ENVELOPE is a Fault, the element children of the `soap:Fault` element MUST be unqualified.

This means that options 1 and 2 are incorrect.

3.3.6 SOAP Custom Fault Codes [WS-I BP 1.1]

R1004 When an ENVELOPE contains a `faultcode` element, the content of that element SHOULD be either one of the fault codes defined in SOAP 1.1.

Option 3 is wrong because it contains a SOAP Fault message with unqualified fault code. Also, it does not belong to SOAP 1.1 predefined fault codes (in this example - "`Client`") because default namespace is empty.

The set of predefined `faultcode` values is:

Table 2.1. SOAP Fault Codes

Error	Description
`VersionMismatch`	The processing party found an invalid namespace for the SOAP `Envelope` element.
`MustUnderstand`	An immediate child element of the SOAP `Header` element that was either not understood or not obeyed by the processing party contained a SOAP `mustUnderstand` attribute with a value of "`1`".
`Client`	The `Client` class of errors indicate that the message was incorrectly formed or did not contain the appropriate information in order to succeed. For example, the message could lack the proper authentication or payment information. It is generally an indication that the message should not be resent without change.
`Server`	The `Server` class of errors indicate that the message could not be processed for reasons not directly attributable to the contents of the message itself but rather to the processing of the message. For example, processing could include communicating with an upstream processor, which didn't respond. The message may succeed at a later point in time.

Sources:

WS-I Basic Profile 1.1 [3.3 SOAP Faults] - [`http://www.ws-i.org/Profiles/BasicProfile-1.1.html`]

Simple Object Access Protocol (SOAP) 1.1 [4.4 SOAP Fault] - [`http://www.w3.org/TR/2000/NOTE-SOAP-20000508`]

Question A020205

What the features does SOAP have?

Options (select 4):

1. Support for SOAP XML messages incorporating attachments (using the multipart MIME structure).
2. It can use only HTTP or HTTPS as a transport protocol.
3. Is a platform and operating system independent.
4. Can use any transport protocol (HTTP, HTTPS, FTP, SMTP, JMS, etc.).
5. SOAP message can be constructed only using Java programming language.
6. Can be used only on Windows and Linux operating systems.
7. Can not be used for sending attachments (images, documents, etc.).

8. Programming language independent.

Answer:

Correct options are: 1, 3, 4, 8.

The Simple Object Access Protocol (SOAP) is a lightweight, XML-based protocol for exchanging information in a decentralized, distributed environment. SOAP supports different styles of information exchange, including:

- Remote Procedure Call style (RPC), which allows for request-response processing, where an endpoint receives a procedure oriented message and replies with a correlated response message.
- Message-oriented information exchange, which supports organizations and applications that need to exchange business or other types of documents where a message is sent but the sender may not expect or wait for an immediate response.

SOAP has the following features:

- Transport protocol independence.
- Programming language independence.
- Platform and operating system independence.
- Support for SOAP XML messages incorporating attachments (using the multipart MIME structure).

Sources:

Overview of SOAP - [http://java.sun.com/developer/technicalArticles/xml/webservices/]

2.3. Describe SOAP Message Construct and create a SOAP message that contains an attachment.

Question A020301

You are sending a scanned image of business contract as SOAP 1.1 attachment:

```
--MIME_boundary
Content-Type: image/tiff
Content-Transfer-Encoding: binary
Content-ID: <business_contract>

d3d3Lm1hcmNoYWwuY29taesgfSEVFES45345sdvgfszd==
--MIME_boundary--
```

How can you refer to MIME attachment from SOAP body?

Options (select 1):

1.
```
<SOAP-ENV:Body>
    ..
    <contract href="cid:business_contract"/>
    ..
</SOAP-ENV:Body>
```

2.

```
<SOAP-ENV:Body>
    ..
    <contract href="business_contract"/>
    ..
</SOAP-ENV:Body>
```

3.

```
<SOAP-ENV:Body>
    ..
    <contract href="#business_contract"/>
    ..
</SOAP-ENV:Body>
```

4.

```
<SOAP-ENV:Body>
    ..
    <contract href="#cid:business_contract"/>
    ..
</SOAP-ENV:Body>
```

Answer:

Correct option is 1.

Both the header entries and body of the primary SOAP 1.1 message may need to refer to other entities in the message package. One of the methods to accomplish this is using existing mechanisms in SOAP and MIME. The data encoding rules allow the value of an accessor to be given by reference, i.e., as a resource referenced by a URI given as the value of an `href` attribute. We observe that the SOAP encoding schema allows the value of an `href` attribute to be any URI reference, and the attribute may therefore be used to reference not just XML fragments within a SOAP 1.1 message, but any resource whatsoever.

If a `Content-ID` header is present, then an absolute URI label for the part is formed using the CID URI scheme as described in RFC 2111.

The Uniform Resource Locator (URL) schema, "`cid:`" allows references to messages and the body parts of messages. For example, within a single SOAP multipart message, one SOAP body part might include embedded references to other parts of the same message.

RFC1738 [URL] reserves the "`cid`" schema for `Content-ID`. Below is shown the syntax for this URL.

The URL take the form:

```
content-id    = url-addr-spec
```

```
cid-url       = "cid" ":" content-id
```

Note: in Internet mail messages, the `url-addr-spec` in a `Content-ID` [MIME] header is enclosed in angle brackets (<>).

The `Content-ID` of a MIME body part is required to be globally unique.

Sources:

SOAP Messages With Attachments (W3C Note) - [`http://www.w3.org/TR/SOAP-attachments`]

RFC 2111 - [http://www.ietf.org/rfc/rfc2111.txt]

Question A020302

You need to send a scanned image of business contract as SOAP 1.1 attachment:

```
MessageFactory factory = MessageFactory.newInstance();
SOAPMessage message = factory.createMessage();

// add code here

attachment.setContentId("attached_contract");
message.addAttachmentPart(attachment);
```

How can you add to SOAP message an attachment with the image using SAAJ API?

Options (select 1):

1.
```
HttpDataSource dataSource = new HttpDataSource("http://localhost/contract.jpeg");
DataHandler dataHandler = new DataHandler(dataSource);
AttachmentPart attachment =  message.createAttachmentPart(dataHandler);
```

2.
```
FileDataSource dataSource = new FileDataSource("contract.jpeg");
DataHandler dataHandler = new DataHandler(dataSource);
AttachmentPart attachment =  message.createAttachmentPart(dataHandler);
```

3.
```
ImageDataSource dataSource = new ImageDataSource("contract.jpeg");
DataHandler dataHandler = new DataHandler(dataSource);
AttachmentPart attachment =  message.createAttachmentPart(dataHandler);
```

4.
```
AttachmentPart attachment = message.createAttachmentPart();
attachment.setContent("contract.jpeg", "image/jpeg");
```

Answer:

Correct option is 2.

There are several methods to creata `AttachmentPart`:

- Create an attachment with no content

  ```
  SOAPMessage.createAttachmentPart();
  ```

  Add content to attachment (string defines content type header)

  ```
  AttachmentPart.setContent(java.lang.Object, java.lang.String);
  ```

- The same as above (`String` defines content type header)

  ```
  SOAPMessage.createAttachmentPart(java.lang.Object, java.lang.String);
  ```

- Content takes a `javax.activation.DataHandler` object

  ```
  SOAPMessage.createAttachmentPart(javax.activation.DataHandler);
  ```

One of the methods for creating an `AttachmentPart` object with content takes a `DataHandler` object, which is part of the JavaBeans Activation Framework (JAF). Using a `DataHandler` object is fairly straightforward.

First you create a `FileDataSource` that represents a JPEG file. Then you create a `DataHandler` that delegates to the `FileDataSource`. Unlike the other methods for setting content, this one does not take a `String` for `Content-Type` header. This method takes care of setting the `Content-Type` header for you, which is possible because one of the things a `DataHandler` object does is determine the data type of the file it contains.

`DataHandler` can be created by 3 ways:

```
// Create a DataHandler instance referencing the specified DataSource.
DataHandler(DataSource ds)

// Create a DataHandler instance representing an object of this MIME type.
DataHandler(java.lang.Object obj, java.lang.String mimeType)

// Create a DataHandler instance referencing a URL.
DataHandler(java.net.URL url)
```

Options 1 and 3 are incorrect, because have invalid data source class names. JAF defines two standard `DataSource` objects: `javax.activation.FileDataSource` and `javax.activation.URLDataSource`.

Option 4 is correct syntactically, but incorrect logically.

Sources:

SAAJ Tutorial - [`http://java.sun.com/j2ee/1.4/docs/tutorial/doc/SAAJ3.html`]

JavaBeans Activation Framework (JAF) - [`http://java.sun.com/products/javabeans/jaf/index.jsp`]

Question A020303

Which statements about SOAP 1.1 message are true?

Options (select 3):

1. SOAP Header element is optional.
2. SOAP Header element is mandatory.
3. SOAP Body element is optional.
4. SOAP Body element is mandatory.
5. SOAP Envelope element is optional.
6. SOAP Envelope element is mandatory.

Answer:

Correct options are 1, 4 and 6.

A SOAP message is an XML document that consists of a mandatory SOAP envelope, an optional SOAP header, and a mandatory SOAP body. The namespace identifier for the elements and attributes is "`http://schemas.xmlsoap.org/soap/envelope/`". A SOAP message contains the following:

- The Envelope is the top element of the XML document representing the message.
- The Header is a generic mechanism for adding features to a SOAP message in a decentralized

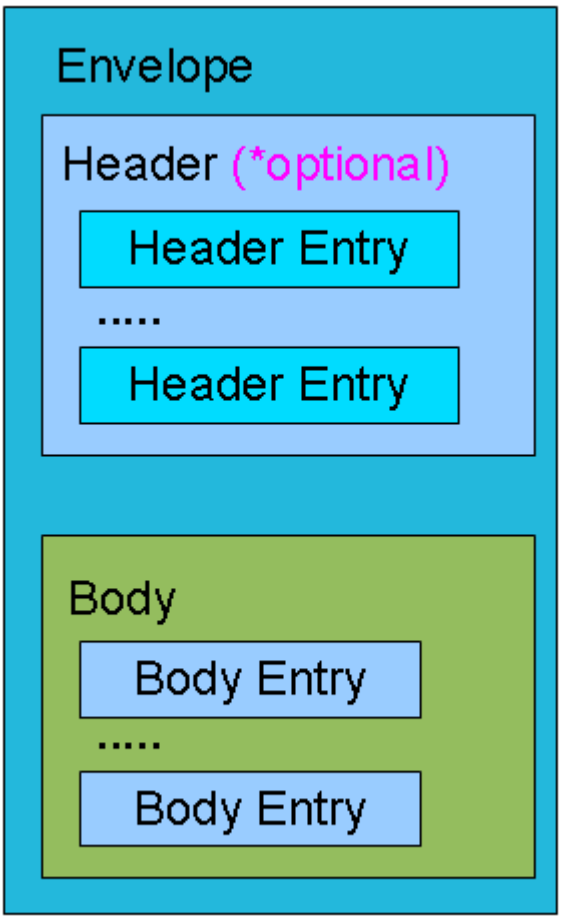
manner without prior agreement between the communicating parties.

- The Body is a container for mandatory information intended for the ultimate recipient of the message. SOAP defines one element for the body, which is the Fault element used for reporting errors.

The grammar rules are as follows:

1. Envelope
    - The element name is "Envelope".
    - The element MUST be present in a SOAP message.
    - The element MAY contain namespace declarations as well as additional attributes. If present, such additional attributes MUST be namespace-qualified. Similarly, the element MAY contain additional sub elements. If present these elements MUST be namespace-qualified and MUST follow the SOAP Body element (WARNING - this is not WS-I BP 1.1 conformant !).
2. Header
    - The element name is "Header".
    - The element MAY be present in a SOAP message. If present, the element MUST be the first immediate child element of a SOAP Envelope element.
    - The element MAY contain a set of header entries each being an immediate child element of the SOAP Header element. All immediate child elements of the SOAP Header element MUST be namespace-qualified.
3. Body
    - The element name is "Body".
    - The element MUST be present in a SOAP message and MUST be an immediate child element of a SOAP Envelope element. It MUST directly follow the SOAP Header element if present. Otherwise it MUST be the first immediate child element of the SOAP Envelope element.
    - The element MAY contain a set of body entries each being an immediate child element of the SOAP Body element. Immediate child elements of the SOAP Body element MAY be namespace-qualified. SOAP defines the SOAP Fault element, which is used to indicate error messages.

Structure of a SOAP Envelope:



Example of a SOAP message:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header>
        <t:Transaction xmlns:t="some-uri" soap:mustUnderstand="1">
            15091974
        </t:Transaction>
    </soap:Header>
    <soap:Body>
        <ns1:reserve xmlns:ns1="urn:reserve-exam-msg">
            <ns1:user>Mikalai Zaikin</ns1:user>
            <ns1:examID>CX-311-232</ns1:examID>
            <ns1:date>
                <ns1:year>2010</ns1:year>
                <ns1:month>09</ns1:month>
                <ns1:day>30</ns1:day>
            </ns1:date>
        </ns1:reserve>
    </soap:Body>
</soap:Envelope>
```

Source:

SOAP 1.1 specification - [http://www.w3.org/TR/2000/NOTE-SOAP-20000508]

Appendix 3. Describing and Publishing (WSDL and UDDI)

3.1. Explain the use of WSDL in Web Services, including a description of WSDL's basic elements, binding mechanisms and the basic WSDL operation types as

limited by the WS-I Basic Profile 1.1.

Question A030101

Snorcle, Inc. is developing a new Web Service using "WSDL-to-Java" approach. Development team is given the WSDL for the Weather Forecast Web Service. What is true about the following WSDL fragment?

```
...
<service name="WeatherService">
    <port name="WeatherServicePort" binding="tns:WeatherServicePortBinding">
        <soap:address location="http://java.boot.by/WeatherService"/>
    </port>
    <port name="WeatherWSPort" binding="tns:WeatherWSPortBinding">
        <soap:address location="http://java.boot.by/WeatherService"/>
    </port>
</service>
...
```

Options (select 1):

1. This fragment shows WS-I BP 1.1 compliant Web Service, because both `port`s have different `name` attributes.
2. This fragment shows NOT WS-I BP 1.1 compliant Web Service, because a single `service` has two `port` elements.
3. This fragment shows NOT WS-I BP 1.1 compliant Web Service, because both `port` elements point to the same location.
4. This fragment shows WS-I BP 1.1 compliant Web Service, because both `port`s have different `binding` attributes.
5. None of the above.

Answer:

Correct option is 3.

This fragment shows NOT WS-I BP 1.1 compliant Web Service because `location` value is the same for both ports.

When input messages destined for two different `wsdl:ports` at the same network endpoint are indistinguishable on the wire, it may not be possible to determine the `wsdl:port` being invoked by them. This may cause interoperability problems. However, there may be situations (e.g., SOAP versioning, application versioning, conformance to different profiles) where it is desirable to locate more than one port on an endpoint; therefore, the Profile allows this.

R2711 A DESCRIPTION SHOULD NOT have more than one `wsdl:port` with the same value for the `location` attribute of the `soapbind:address` element.

Option 1 is wrong, because the Web Service is not WS-I BP 1.1 compliant.

Option 2 is wrong. WS-I BP 1.1 allows more than one port on an endpoint

Sources:

Basic Profile Version 1.1 (4.7.7 Multiple Ports on an Endpoint) - [`http://www.ws-i.org/Profiles/BasicProfile-1.1.html`]

Question A030102

What is the purpose of `.wsdl` file?

Options (select 1):

1. It contains information that correlates the mapping between the Java interfaces and WSDL definition.
2. It describes a collection of servlets.
3. It identifies the services provided by the server and the set of operations or functionalities within each service that the server supports.
4. It is used to override existing JAX-WS annotations.

Answer:

Correct option is 3.

WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, however, the only bindings described in this document describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME.

Option 1 is wrong, this is the purpose of JAX-RPC mapping file.

Option 2 is wrong, this is the purpose of `web.xml` Servlet Container deployment descriptor.

Option 4 is wrong, this is the purpose of `webservices.xml` Web Services deployment descriptor.

Similar to Java API for XML-based RPC (JAX-RPC) Web services, you can use deployment descriptors to describe JAX-WS Web services. For JAX-WS Web services, the use of the `webservices.xml` deployment descriptor is optional because you can use annotations to specify all of the information that is contained within the deployment descriptor file. You can use the deployment descriptor file to augment or override existing JAX-WS annotations. Any information that you define in the `webservices.xml` deployment descriptor overrides any corresponding information that is specified by annotations.

Sources:

Web Services Description Language (WSDL) 1.1 - [`http://www.w3.org/TR/wsdl`]

Question A030103

What SOAP bindings Basic Profile 1.1 allows?

Options (select 2):

1. rpc-literal
2. rpc-encoded
3. document-literal
4. document-encoded
5. rpc-wrapped

Answer:

Correct options are 1 and 3.

The style, "document" or "rpc", of an interaction is specified at the `wsdl:operation` level, permitting `wsdl:binding`s whose `wsdl:operation`s have different styles. This has led to interoperability problems.

WS-I BP 1.1: A `wsdl:binding` in a DESCRIPTION MUST either be a `rpc-literal` binding or a `document-literal` binding.

The Profile prohibits the use of encodings, including the SOAP encoding.

WS-I BP 1.1: A `wsdl:binding` in a DESCRIPTION MUST use the value of "`literal`" for the use attribute in all `soapbind:body`, `soapbind:fault`, `soapbind:header` and `soapbind:headerfault` elements.

Option 5 is wrong, because "`rpc-wrapped`" binding does not exist.

Sources:

Basic Profile Version 1.1 (4.7 SOAP Binding) - [`http://www.ws-i.org/Profiles/BasicProfile-1.1.html`]

Question A030104

Which statement is true about WSDL namespace for WSDL SOAP binding?

Options (select 1):

1. Its URI is `http://schemas.xmlsoap.org/wsdl/`
2. Its URI is `http://schemas.xmlsoap.org/wsdl/http/`
3. Its URI is `http://schemas.xmlsoap.org/soap/encoding/`
4. Its URI is `http://schemas.xmlsoap.org/soap/envelope/`
5. None of the above.

Answer:

Correct option is 5.

WSDL namespace for WSDL SOAP binding URI is:

`http://schemas.xmlsoap.org/wsdl/soap/`

Below is WSDL fragment which demonstrates SOAP binding for "document-literal" style:

```
<?xml version="1.0" encoding="utf-8"?>

<definitions
    targetNamespace="http://java.boot.by"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
    xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
    xmlns:tns="http://java.boot.by"
    xmlns:ns0="http://java.boot.by/types"
    xmlns:ns1="http://java.boot.by"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
    ...
    <binding name="AddNumbersBinding" type="tns:AddNumbersPortType">
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="addNumbers">
            <input>
                <soap:body use="literal"/>
            </input>
```

```
            <output>
                <soap:body use="literal"/>
            </output>
            <fault name="error">
                <soap:fault name="error" use="literal"/>
            </fault>
        </operation>
    </binding>
</definitions>
```

Table 3.1. WSDL 1.1 namespaces and prefixes

prefix	namespace URI	definition
wsdl	http://schemas.xmlsoap.org/wsdl/	WSDL namespace for WSDL framework.
soap	http://schemas.xmlsoap.org/wsdl/soap/	WSDL namespace for WSDL SOAP binding.
http	http://schemas.xmlsoap.org/wsdl/http/	WSDL namespace for WSDL HTTP GET and POST binding.
mime	http://schemas.xmlsoap.org/wsdl/mime/	WSDL namespace for WSDL MIME binding.
soapenc	http://schemas.xmlsoap.org/soap/encoding/	Encoding namespace as defined by SOAP 1.1.
soapenv	http://schemas.xmlsoap.org/soap/envelope/	Envelope namespace as defined by SOAP 1.1.
xsi	http://www.w3.org/2000/10/XMLSchema-instance	Instance namespace as defined by XSD.
xsd	http://www.w3.org/2000/10/XMLSchema	Schema namespace as defined by XSD.
tns	(various)	The "this namespace" (tns) prefix is used as a convention to refer to the current document.
(other)	(various)	All other namespace prefixes are samples only. In particular, URIs starting with "http://example.com" represent some application-dependent or context-dependent URI.

Option 1 is wrong. This URI defines WSDL namespace for WSDL framework.

Option 2 is wrong. This URI defines WSDL namespace for WSDL HTTP GET and POST binding.

Option 3 is wrong. This URI defines encoding namespace as defined by SOAP 1.1.

Option 4 is wrong. This URI defines envelope namespace as defined by SOAP 1.1.

Sources:

Web Services Description Language (WSDL) 1.1 (Section 1.2: Notational Conventions) - [http://www.w3.org/TR/2001/NOTE-wsdl-20010315]

WSDL bindings - [https://jax-rpc.dev.java.net/docs/wsdl-bindings.html]

3.2. Describe how WSDL enables one to separate the description of the abstract functionality offered by a service from concrete details of a service description such as "how" and "where" that functionality is offered.

Question A030201

What statements are correct about WSDL 1.1 elements?

Options (select 3):

1. `types` – an abstract definition of the data being communicated.
2. `operation` – an abstract description of an action supported by the service.
3. `operation` – an abstract set of `portType`s supported by one or more endpoints.
4. `binding` - a single endpoint defined as a combination of a `portType` and a network address.
5. `types` – a container for data type definitions using some type system (such as XSD).
6. `binding` – a concrete protocol and data format specification for a particular port type.

Answer:

Correct options are 2, 5, and 6.

Option 1 is wrong. The `message` is an abstract, typed definition of the data being communicated.

The `message` element defines the data elements of an operation. Each message can consist of one or more parts. The parts can be compared to the parameters of a function call in a traditional programming language.

```
<message name="GetLastTradePriceInput">
    <part name="body" element="xsd1:TradePriceRequest"/>
</message>

<message name="GetLastTradePriceOutput">
    <part name="body" element="xsd1:TradePrice"/>
</message>
```

Option 2 is correct. The `operation` is an abstract description of an action supported by the service.

The `operation` element defines each operation that the port exposes. For each operation the corresponding SOAP action has to be defined. You must also specify how the input and output are encoded.

Option 3 is wrong. There is no element for a set of `portType`s. The `portType` is an abstract set of operations supported by one or more endpoints.

It defines a Web Service, the operations that can be performed, and the messages that are involved:

```
<portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
        <input message="tns:GetLastTradePriceInput"/>
        <output message="tns:GetLastTradePriceOutput"/>
    </operation>
</portType>
```

Option 4 is wrong. There is no element for combination of a `portType` and a network address. The `port` is a single endpoint defined as a combination of a binding and a network address.

```
<service name="StockQuoteService">
    <port name="StockQuotePort" binding="tns:StockQuoteBinding">
        <soap:address location="http://java.boot.by/stockquote"/>
```

```
    </port>
</service>
```

Option 5 is correct. The `types` is a container for data type definitions using some type system (such as XSD).

```
<types>
    <schema targetNamespace="http://example.com/stockquote.xsd"
            xmlns="http://www.w3.org/2000/10/XMLSchema">
        <element name="TradePriceRequest">
            <complexType>
                <all>
                    <element name="tickerSymbol" type="string"/>
                </all>
            </complexType>
        </element>
        <element name="TradePrice">
            <complexType>
                <all>
                    <element name="price" type="float"/>
                </all>
            </complexType>
        </element>
    </schema>
</types>
```

Option 6 is correct. The `binding` is a **concrete** protocol and data format specification for a particular port type.

The `binding` element has two attributes: `name` and `type`.

The `name` attribute (you can use any name you want) defines the name of the binding, and the `type` attribute points to the port for the binding.

The `soap:binding` element has two attributes: `style` and `transport`.

The `style` attribute can be "`rpc`" or "`document`". The `transport` attribute defines the SOAP protocol to use.

The `operation` element defines each operation that the port exposes.

For each `operation` the corresponding SOAP action has to be defined. You must also specify how the input and output are encoded.

```
<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetLastTradePrice">
        <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
</binding>
```

Sources:

Web Services Description Language (WSDL) 1.1 - [http://www.w3.org/TR/wsdl]

Question A030202

What statements are true about elements of WSDL 1.1 document?

Options (select 2):

1. `portType` – a single endpoint defined as a combination of a binding and a network address.
2. `port` – a single endpoint defined as a combination of a binding and a network address.
3. `message` – an abstract set of types supported by one or more endpoints.
4. `message` - an abstract, typed definition of the data being communicated.

Answer:

Correct options are 2 and 4.

Option 1 is wrong. The `portType` is an abstract set of operations supported by one or more endpoints.

```
<portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
        <input message="tns:GetLastTradePriceInput"/>
        <output message="tns:GetLastTradePriceOutput"/>
    </operation>
</portType>
```

Option 2 is correct. The `port` is a single endpoint defined as a combination of a binding and a network address.

```
<port name="StockQuotePort" binding="tns:StockQuoteBinding">
    <soap:address location="http://example.com/stockquote"/>
</port>
```

Option 3 is wrong. There is no WSDL element for abstract set of types supported by one or more endpoints.

Option 4 is correct. The `message` is an abstract, typed definition of the data being communicated.

```
<message name="GetLastTradePriceInput">
    <part name="body" element="xsd1:TradePriceRequest"/>
</message>

<message name="GetLastTradePriceOutput">
    <part name="body" element="xsd1:TradePrice"/>
</message>
```

Sources:

Web Services Description Language (WSDL) 1.1 - [http://www.w3.org/TR/wsdl]

3.3. Describe the Component Model of WSDL including Descriptions, Interfaces, Bindings, Services and Endpoints.

blah-blah

3.4. Describe the basic functions provided by the UDDI Publish and Inquiry APIs to interact with a UDDI business registry.

Question A030401

User sends the following SOAP message to UDDI registry:

```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <delete_tModel generic="2.0" xmlns="urn:uddi-org:api_v2">
      <authInfo>uddiUser_AuthToken</authInfo>
      <tModelKey>uuid:6e090afa-33e5-36eb-81b7-1ca18373f457</tModelKey>
    </delete_tModel>
  </soap:Body>
</soap:Envelope>
```

What is true about the `tModel` upon successful completion?

Options (select 2):

1. The `tModel` is physically deleted as a result of this call.
2. The `tModel` is just hidden and is still accessible, via the `get_registeredInfo` call.
3. The `tModel` is just hidden and is still accessible, via the `find_tModel` call.
4. The `tModel` is just hidden and is still accessible, via the `get_tModelDetail` call.
5. UDDI registry is not changed. The `tModel` can not be deleted by anyone from UDDI registry, it can only be modified.

Answer:

Correct options are 2 and 4.

The `delete_tModel` API call is used to logically delete one or more `tModel` structures. Logical deletion hides the deleted `tModels` from find_tModel result sets but does not physically delete it. Deleting an already deleted `tModel` has no effect.

If a `tModel` is hidden in this way it will not be physically deleted as a result of this call. Any `tModel`s hidden in this way are still accessible, via the `get_registeredInfo` and `get_tModelDetail` messages, but will be omitted from any results returned by calls to `find_tModel`. The purpose of the `delete_tModel` behavior is to ensure that the details associated with a hidden `tModel` are still available to anyone currently using the `tModel`. A hidden `tModel` can be restored and made visible to search results by invoking the `save_tModel` API at a later time, passing the original data and the `tModelKey` value of the hidden `tModel`.

Sources:

UDDI v.2 API Specification [4.4.6 delete_tModel] - [`http://uddi.org/pubs/ProgrammersAPI_v2.htm`]

Question A030402

What is true about UDDI security?

Options (select 1):

1. Inquiry API requires HTTPS protocol, Publishing API requires HTTPS protocol
2. Inquiry API requires HTTPS protocol, Publishing API requires HTTP protocol
3. Inquiry API requires HTTP protocol, Publishing API requires HTTPS protocol
4. Inquiry API requires HTTP protocol, Publishing API requires HTTP protocol

Answer:

Correct option is 3.

Accessing UDDI programmatically is accomplished via API calls defined in this programmer's reference. Two types of APIs are defined. A publisher's API is provided for interactions between programs and the UDDI registry for the purpose of storing or changing data in the registry. An inquiry API is provided for programs that want to access the registry to read information from the registry.

Authenticated access is required to use the publishers API. Each Operator Site is responsible for selecting and implementing an authentication protocol that is compatible with the publishers API, as well as providing a new user sign-up mechanism. Before using any of the publisher API functions, the caller is responsible for signing up with one or more Operator Sites or compatible registries and establishing user credentials.

The Inquiry and Publishers API functions are exposed as SOAP messages over HTTP. HTTPS (specifically SSL 3.0) is used for all publisher API calls in order to assure wire privacy. No authentication is required to make use of the Inquiry API functions.

Sources:

UDDI v.2 API Specification - [`http://uddi.org/pubs/ProgrammersAPI_v2.htm`]

Question A030403

Is the authentication required when you use UDDI Inquiry API?

Options (select 1):

1. Yes, authentication is always required for Inquiry API.
2. Yes, some Inquiry API functions require authentication.
3. No, Inquiry API does not require authentication.
4. None of the above.

Answer:

Correct option is 3.

Accessing UDDI programmatically is accomplished via API calls defined in this programmer's reference. Two types of APIs are defined. A publisher's API is provided for interactions between programs and the UDDI registry for the purpose of storing or changing data in the registry. An inquiry API is provided for programs that want to access the registry to read information from the registry.

Authenticated access is required to use the publishers API. Each Operator Site is responsible for selecting and implementing an authentication protocol that is compatible with the publishers API, as well as providing a new user sign-up mechanism. Before using any of the publisher API functions, the caller is responsible for signing up with one or more Operator Sites or compatible registries and establishing user credentials.

The Inquiry and Publishers API functions are exposed as SOAP messages over HTTP. HTTPS (specifically SSL 3.0) is used for all publisher API calls in order to assure wire privacy. No authentication is required to make use of the Inquiry API functions.

Sources:

UDDI v.2 API Specification - `http://uddi.org/pubs/ProgrammersAPI_v2.htm`

Question A030404

Which statement about accessing UDDI registry is true?

Options (select 1):

1. Publisher API functions required to be exposed over FTP protocol.
2. Inquiry API functions required to be exposed over HTTP protocol.
3. Inquiry API functions are asynchronous.
4. Publisher API functions required to be exposed over SMTP protocol.
5. Publisher API functions are asynchronous.
6. Publisher API functions required to be exposed over HTTP protocol.

Answer:

Correct option is 2.

Inquiry API functions represent inquiries that anyone can make of any UDDI Operator Site at any time. These messages all behave synchronously and are required to be exposed via HTTP-POST only. Other synchronous or asynchronous mechanisms may be provided at the discretion of the individual UDDI Operator Site or UDDI compatible registry.

Publishing API functions represent commands that require authenticated access to an UDDI Operator Site, and are used to publish and update information contained in a UDDI compatible registry. Each business should initially select one Operator Site to host their information. Once chosen, information can only be updated at the site originally selected. UDDI provides no automated means to reconcile multiple or duplicate registrations.

Publishing API functions all behave synchronously and are callable via HTTP-POST only. HTTPS is used exclusively for all of the calls defined in this publisher's API.

This makes options 1, 3, 4, 5, and 6 incorrect.

Sources:

UDDI v.2 API Specification - [`http://uddi.org/pubs/ProgrammersAPI_v2.htm`]

Question A030405

Which two are true about UDDI?

Options (select 2):

1. Publishing API functions are callable via POST method.
2. Publishing API functions are callable via GET method.
3. Publishing API functions are callable via PUT and DELETE method.
4. Inquiry API functions are callable via GET method.
5. Inquiry API functions are callable via HEAD method.
6. Inquiry API functions are callable via POST method.

Answer:

Correct options are 1 and 6.

Inquiry API functions all behave synchronously and are required to be exposed via HTTP-POST only.

Publishing API functions all behave synchronously and are callable via HTTP-POST only. HTTPS is used exclusively for all of the calls defined in this publisher's API.

This makes options 2, 3, 4, and 5 incorrect.

Sources:

UDDI v.2 API Specification - [`http://uddi.org/pubs/ProgrammersAPI_v2.htm`]

Question A030406

Which is true about UDDI Inquiry API function?

Options (select 1):

1. `find_relatedEntity` function is used to locate related business entities and returns a `relatedEntitiesList` message.
2. `find_relatedBusinesses` function is used to locate related business entities and returns a `relatedBusinessesList` message.
3. `find_relatedBusinesses` function is used to locate related business entities and returns a `businessesList` message.
4. `find_relatedEntity` function is used to locate related business entities and returns a `entitiesList` message.

Answer:

Correct option is 2.

`find_relatedBusinesses` is used to locate information about `businessEntity` registrations that are related to a specific business entity whose key is passed in the inquiry. The Related Businesses feature is used to manage registration of business units and subsequently relate them based on organizational hierarchies or business partner relationships. Returns a `relatedBusinessesList` message.

Option 1 is wrong, because `find_relatedEntity` function does not exist.

Option 3 is wrong, because `find_relatedBusinesses` returns a `relatedBusinessesList` message. Moreover, `businessesList` name is invalid (do not be confused by `find_business` function which returns a `businessList` message.)

Option 4 is wrong, because `find_relatedEntity` function and `entitiesList` message do not exist in the UDDI Inquiry API.

Sources:

UDDI v.2 API Specification - [`http://uddi.org/pubs/ProgrammersAPI_v2.htm`]

Appendix 4. JAX-WS

4.1. Explain JAX-WS technology for building web services and client that communicate using XML.

Question A040101

Web Service client program needs to add a security credential to the header of a SOAP message before sending the message. What would be the best approach?

Options (select 1):

1. Use client-side Header handler.
2. Use client-side Logical handler.
3. Use client-side SOAP handler.
4. Use client-side intermediary SEI.

Answer:

Correct option is 3.

JAX-WS provides a handler framework that allows application code to inspect and manipulate outgoing and incoming SOAP messages.

Developer needs to create a handler class, which implements the `Handler` interface in the `javax.xml.ws.handler` package. JAX-WS provides two `Handler` subinterfaces, `LogicalHandler` and `SOAPHandler`. As the names suggest, the `LogicalHandler` is protocol-neutral but the `SOAPHandler` is SOAP-specific.

A `LogicalHandler` has access only to the message payload in the SOAP body, whereas a `SOAPHandler` has access to the entire SOAP message, including any optional headers and attachments.

The class that implements either the `LogicalHandler` or the `SOAPHandler` interface needs to define three methods for either interface type, including `handleMessage`, which gives the programmer access to the underlying message. The other two shared methods are `handleFault` and `close`. The `SOAPHandler` interface requires the implementation to define a fourth method, `getHeaders`.

```
public class CredentialsHandler implements SOAPHandler<SOAPMessageContext> {
    .....
    public boolean handleMessage(SOAPMessageContext ctx) {
        // Is this an outbound message, i.e., a request?
        Boolean req = (Boolean) ctx.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);

        // Manipulate the SOAP only if it is a request
        if (req) {
            // Prepare user credentials
            String credentials = ...

            try {
                SOAPMessage msg = ctx.getMessage();
                SOAPEnvelope env = msg.getSOAPPart().getEnvelope();
                SOAPHeader hdr = env.getHeader();

                // Ensure that the SOAP message has a header
                if (hdr == null) hdr = env.addHeader();

                QName qname = new QName("http://java.boot.by", "userID");
                SOAPHeaderElement helem = hdr.addHeaderElement(qname);

                helem.setActor(SOAPConstants.URI_SOAP_ACTOR_NEXT); // default
                helem.addTextNode(credentials);
                msg.saveChanges();
            } catch(SOAPException e) {
                System.err.println(e);
            } catch(IOException e) {
                System.err.println(e);
            }
        }
        return true; // continue down the chain
    }
    .....
}
```

Option 1 is wrong, because Header handler does not exist.

Option 2 is wrong, because Logical handler does not have access to SOAP header.

Option 4 is wrong, because it would require client program re-configuring. Handlers is the easiest way to implement the requested functionality. The JAX-WS handler framework encourages the chain of responsibility pattern, which Java servlet programmers encounter when using filters. The underlying idea is to distribute responsibility among various handlers so that the overall application is highly modular and, therefore, more easily maintainable.

Sources:

Get a handle on the JAX-WS API's handler framework - [`http://www.javaworld.com/javaworld/jw-02-2007/jw-02-handler.html`]

Handlers in JAX-WS - [`https://jax-ws.dev.java.net/articles/handlers_introduction.html`]

4.2. Given a set of requirements for a Web Service, such as transactional needs, and security requirements, design and develop Web Service applications that use JAX-WS technology.

blah

4.3. Describe the Integrated Stack (I-Stack) which consist of JAX-WS, JAXB, StAX, SAAJ.

blah

4.4. Describe and compare JAX-WS development approaches.

Question A040401

A developer is implementing a Web Service using code-first ("Java-to-WSDL") approach. Which JAX-WS tools should be used to generate portable artifacts?

Options (select 2):

1. `xjc`
2. `apt`
3. `wsgen`
4. `wsimport`

Answer:

Correct options are 2 and 3.

For creating a Web Service from a Java class, either you want to expose the functionality of an existing application or create the Web Service from scratch. To generate JAX-WS portable artifacts, use either the `apt` tool or the `wsgen` tool. Use the `apt` tool to generate the artifacts when you start from Java **source** files, because the JAX-WS tools then has full access to the source code and can use parameter names that are otherwise not available through the Java Reflection APIs. However, use the `wsgen` tool to generate the artifacts when you start from **compiled** Java classes.

Option 1 is wrong, because the `xjc` tool is a binding compiler. It creates JAXB Java classes from XML Schema.

Option 2 is correct. The `apt` tool provides a facility for programmatically processing the annotations added to Java by JSR 175, Metadata Facility for the Java Programming Language. In brief, JSR 175

allows programmers to declare new kinds of structured modifiers that can be associated with program elements, fields, methods, classes, etc.

The `apt` tool generates the portable artifacts used in JAX-WS services.

The following lists the process to create a Web Service starting from Java source files:

1. Use `apt` to generate the artifacts: a WSDL file, XML Schema documents, and other Java artifacts.
2. Package the `web.xml`, `sun-jaxws.xml`, Service Endpoint Interface (SEI) and implementation class, value types, and generated classes, if any, into a WAR file.
3. Deploy the WAR to a web container.

Option 3 is correct. The `wsgen` tool generates JAX-WS portable artifacts used in JAX-WS Web Services. The tool reads a Web Service endpoint class and generates all the required artifacts for web service deployment, and invocation.

The following lists the process to create a Web Service starting from compiled Java classes:

1. Use `wsgen` to generate portable artifacts.
2. Package the `web.xml`, `sun-jaxws.xml`, Service Endpoint Interface and implementation class, value types, and generated classes, if any, into a WAR file.
3. Deploy the WAR to a web container.

Option 4 is wrong, the `wsimport` tool is used when you creating a Web Service from a WSDL file, or when you need to create client-side artifacts.

Sources:

JAX-WS Tools - [`http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/docs/2.0/jaxws/jaxws-tools.html`]

Question A040402

You are hired by Snorcle, Inc. to develop an order-tracking Web Service which will be consumed by partner companies. Clients use different platforms (JEE, .NET). In order to reduce load of the Web Service clients will be asked to implement client-side validation of requests. Which approach should you use?

Options (select 1):

1. Code-first ("Java-to-WSDL") approach.
2. Contract-first ("WSDL-to-Java") approach.
3. Meet in the middle approach.
4. None of the above.

Answer:

Correct option is 2.

Option 1 is wrong.

If you start from Java SEI and the service changes, the WSDL automatically changes. The WSDL thereby loses some of its appeal for creating client artifacts, as this process may have to be repeated over and over. One of the first principles of software development is that an interface, once published, should be regarded as immutable so that code once written against a published interface never has to be rewritten. Also, the code-first approach seems to go against the language-neutral

theme of SOAP-based Web Services.

Option 2 is correct.

The contract-first ("WSDL-to-Java") development approach is most suitable when you build a Web Service from scratch and the Web Service elements are the fundamental component model. You should also use this development approach when your primary focus is on client-service interoperability. Working from WSDL provides the following advantages:

- "WSDL-to-Java" approach is better suited to situations in which the service developer is not the author of the service semantics and workflow, such as when implementing to a certain industry's convention for Web Services.
- The strong typing is shared using the WSDL file itself. Starting from WSDL and XML Schema types puts strong types where they can be shared by services and clients.
- "WSDL-to-Java" approach offers you the best interoperability because it allows you to author the WSDL file directly and gives you control over data type and encoding choices.
- Serializable types are expressed first in XML Schema, and then mapped to one or more programming languages and object models.
- Client-side validation is easier to implement.

Option 3 is wrong. There is no meet in the middle approach to create Web Service.

Sources:

Contract-First Web Services: 6 Reasons to Start with WSDL and Schema - [`http://soa.sys-con.com/node/143909`]

Question A040403

A developer is implementing a Web Service using contract-first ("WSDL-to-Java") approach. Which JAX-WS tool should be used to generate portable artifacts?

Options (select 1):

1. `xjc`
2. `apt`
3. `wsgen`
4. `wsimport`
5. None of the above.

Answer:

Correct option is 4.

The "WSDL-to-Java" technology approach in JAX-WS requires the following steps:

1. Create or obtain a WSDL file that specifies the characteristics of the service, such as service method name, service parameters, and return values.
2. Generate a Service Endpoint Interface (SEI).
3. Implement the required Web Service application components to satisfy the criteria specified in the service description.
4. Create a WAR file and deploy.

You should know the URL of the WSDL file, or have access to the relevant WSDL file. The WSDL file

is used to generate the SEI through the `wsimport` tool.

The `wsimport` tool generates JAX-WS portable artifacts, such as:

1. Service Endpoint Interface (SEI)
2. Service
3. Exception class mapped from `wsdl:fault` (if any)
4. Async Response Bean derived from response `wsdl:message` (if any)
5. JAXB generated value types (mapped Java classes from schema types)

Option 1 is wrong, because the `xjc` tool is a binding compiler. It creates JAXB Java classes from XML Schema.

Option 2 is wrong. The tool `apt`, annotation processing tool, includes a set of new reflective APIs and supporting infrastructure to process program annotations. The `apt` reflective APIs provide a build-time, source-based, read-only view of program structure. These reflective APIs are designed to cleanly model the Java programming language's type system after the addition of generics. First, `apt` runs annotation processors that can produce new source code and other files. Next, `apt` can cause compilation of both original and generated source files, easing development. The reflective APIs and other APIs used to interact with the tool are subpackages of `com.sun.mirror`.

Option 3 is wrong. The `wsgen` tool generates JAX-WS portable artifacts used in JAX-WS Web Services. The tool reads a Web Service endpoint implementation class and generates all the required artifacts for Web Service deployment, and invocation.

The `wsgen` tool is used for bottom up development to generate a WSDL and appropriate wrappers from a Web Service application.

Sources:

JAX-WS Tools - [`http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/docs/2.0/jaxws/jaxws-tools.html`]

The `wsimport` tool documentation - [`http://download.oracle.com/javase/6/docs/technotes/tools/share/wsimport.html`]

4.5. Describe the features of JAX-WS including the usage of Java Annotations.

Question A040501

Given the SEI code:

```
package by.iba;

@WebService
@SOAPBinding(style = Style.RPC)
public interface Hello {
    @WebMethod
    String sayHello(String name);
}
```

What would be resulting contract fragment for the Web Service?

Options (select 1):

1.
```
<binding name="HelloImplPortBinding" type="tns:Hello">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="sayHello">
```

```
        <soap:operation soapAction=""></soap:operation>
        <input>
            <soap:body use="encoded" namespace="http://iba.by/"></soap:body>
        </input>
        <output>
            <soap:body use="encoded" namespace="http://iba.by/"></soap:body>
        </output>
    </operation>
</binding>
```

2.

```
<binding name="HelloImplPortBinding" type="tns:Hello">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="sayHello">
        <soap:operation soapAction=""></soap:operation>
        <input>
            <soap:body use="literal" namespace="http://iba.by/"></soap:body>
        </input>
        <output>
            <soap:body use="literal" namespace="http://iba.by/"></soap:body>
        </output>
    </operation>
</binding>
```

3.

```
<binding name="HelloImplPortBinding" type="tns:Hello">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="sayHello">
        <soap:operation soapAction=""></soap:operation>
        <input>
            <soap:body use="encoded" namespace="http://iba.by/"></soap:body>
        </input>
        <output>
            <soap:body use="encoded" namespace="http://iba.by/"></soap:body>
        </output>
    </operation>
</binding>
```

4.

```
<binding name="HelloImplPortBinding" type="tns:Hello">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="sayHello">
        <soap:operation soapAction=""></soap:operation>
        <input>
            <soap:body use="literal" namespace="http://iba.by/"></soap:body>
        </input>
        <output>
            <soap:body use="literal" namespace="http://iba.by/"></soap:body>
        </output>
    </operation>
</binding>
```

Answer:

Correct option is 4.

If you are using a SOAP binding for your service, you can use JAX-WS annotations to specify a number of the bindings properties. These properties correspond directly to the properties you can specify in a service's WSDL contract.

The `@SOAPBinding` annotation is defined by the `javax.jws.soap.SOAPBinding` interface. It

provides details about the SOAP binding used by the service when it is deployed. If the `@SOAPBinding` annotation is not specified, a service is published using a wrapped doc/literal SOAP binding.

You can put the `@SOAPBinding` annotation on the SEI and any of the SEI's methods. When it is used on a method, setting of the method's `@SOAPBinding` annotation take precedence.

Table 4.1. @SOAPBinding Properties

Property	Values	Description
`style`	`Style.DOCUMENT` (default) | `Style.RPC`	Specifies the style of the SOAP message. If RPC style is specified, each message part within the SOAP body is a parameter or return value and will appear inside a wrapper element within the `soap:body` element. The message parts within the wrapper element correspond to operation parameters and must appear in the same order as the parameters in the operation. If `DOCUMENT` style is specified, the contents of the SOAP body must be a valid XML document, but its form is not as tightly constrained.
`use`	`Use.LITERAL` (default) | `Use.ENCODED`	Specifies how the data of the SOAP message is streamed.
`parameterStyle`	`ParameterStyle.WRAPPED` (default) | `ParameterStyle.BARE`	Specifies how the method parameters, which correspond to message parts in a WSDL contract, are placed into the SOAP message body. A parameter style of `BARE` means that each parameter is placed into the message body as a child element of the message root. A parameter style of `WRAPPED` means that all of the input parameters are wrapped into a single element on a request message and that all of the output parameters are wrapped into a single element in the response message.

The purpose of the SOAP binding element is to signify that the binding is bound to the SOAP protocol format: Envelope, Header and Body. This element makes no claims as to the encoding or format of the message.

The `soap:binding` element MUST be present when using the SOAP binding. The value of the `style` attribute is the default for the `style` attribute for each contained operation. If the `style` attribute is omitted, it is assumed to be "`document`".

```
<definitions .... >
    <binding .... >
        <soap:binding transport="uri"? style="rpc|document"?>
    </binding>
</definitions>
```

The `soap:body` element specifies how the message parts appear inside the SOAP Body element.

```
<definitions .... >
    <binding .... >
        <operation .... >
            <input>
                <soap:body parts="nmtokens"? use="literal|encoded"?
                           encodingStyle="uri-list"? namespace="uri"?>
            </input>
            <output>
```

```
                <soap:body parts="nmtokens"? use="literal|encoded"?
                           encodingStyle="uri-list"? namespace="uri"?>
            </output>
        </operation>
    </binding>
</definitions>
```

The required `use` attribute indicates whether the message parts are encoded using some encoding rules, or whether the parts define the concrete schema of the message.

If `use` is `encoded`, then each message part references an abstract type using the `type` attribute. These abstract types are used to produce a concrete message by applying an encoding specified by the `encodingStyle` attribute. The part names, types and value of the namespace attribute are all inputs to the encoding, although the namespace attribute only applies to content not explicitly defined by the abstract types. If the referenced encoding style allows variations in it's format (such as the SOAP encoding does), then all variations MUST be supported ("reader makes right").

If `use` is `literal`, then each part references a concrete schema definition using either the `element` or `type` attribute. In the first case, the element referenced by the part will appear directly under the Body element (for `document` style bindings) or under an accessor element named after the message part (in `rpc` style). In the second, the type referenced by the part becomes the schema type of the enclosing element (Body for `document` style or part accessor element for `rpc` style). The value of the `encodingStyle` attribute MAY be used when the `use` is `literal` to indicate that the concrete format was derived using a particular encoding (such as the SOAP encoding), but that only the specified variation is supported ("writer makes right").

Option 1 is wrong, because WSDL fragment demonstrates "`document/encoded`" binding.

Option 2 is wrong, because WSDL fragment demonstrates "`document/literal`" binding.

Option 3 is wrong, because WSDL fragment demonstrates "`rpc/encoded`" binding. Note, by default JAX-WS uses "`literal`" encoding.

Sources:

JSR 181: Web Services Metadata for the Java Platform - [`http://jcp.org/en/jsr/detail?id=181`]

Web Services Description Language (WSDL) 1.1 - [`http://www.w3.org/TR/2001/NOTE-wsdl-20010315`]

Question A040502

You are developing RESTful Web Service for the Snorcle, Inc. Order Tracking System.

Which annotation the endpoint implementation class must have?

Options (select 1):

1. `@WebServiceProvider`
2. `@WebServiceRef`
3. `@WebService`
4. `@RESTful`

Answer:

Correct option is 1.

JAX-WS enables building RESTful endpoints through a `javax.xml.ws.Provider` interface in the API.

`Provider` is a generic interface that can be implemented by a class as a dynamic alternative to a service endpoint interface (SEI), and a service implementing this interface can be deployed in a Java EE container or published in a stand-alone mode through the JAX-WS Endpoint API.

A `Provider` based service endpoint implementation MUST carry a `@WebServiceProvider` annotation.

The `@WebServiceProvider` annotation is specified on classes that implement a strongly typed `javax.xml.ws.Provider` interface. It is used to declare that a class that satisfies the requirements for a provider does indeed define a Web Service endpoint, much like the `@WebService` annotation does for SEI-based endpoints.

The `@WebServiceProvider` and `@WebService` annotations are mutually exclusive. A class annotated with the `@WebServiceProvider` annotation MUST NOT carry a `@WebService` annotation.

Option 2 is wrong. The `@WebServiceRef` annotation is used to declare a reference to a Web Service, example:

```
@Stateless
public ClientComponent {

    // WebServiceRef using the generated service interface type
    @WebServiceRef
    public StockQuoteService stockQuoteService;

    // WebServiceRef using the SEI type
    @WebServiceRef(StockQuoteService.class)
    private StockQuoteProvider stockQuoteProvider;

    ....
}
```

Option 3 is wrong. The `@WebService` annotation marks a Java class as implementing a SOAP based Web Service.

Option 4 is wrong, because the annotation name is invalid.

Sources:

RESTful Web Services - [`http://www.oracle.com/technetwork/articles/javase/index-137171.html`]

The Java API for XML-Based Web Services (JAX-WS) 2.2 - [`http://www.jcp.org/en/jsr/detail?id=224`]

Question A040503

You are developing a RESTful Web Service for the Snorcle, Inc. Order Managing System (OMS). The Web Service may return results in plain text, HTML or XML format, depending on the "`Accept:`" HTTP header sent by clients. Which code fragment demonstrates correct combination of JAX-WS annotations?

Options (select 1):

1.
```
@ServiceMode(value = Service.Mode.PAYLOAD);
@BindingType(value = SOAPBinding.SOAP11HTTP_BINDING);
```

2.
```
@ServiceMode(value = Service.Mode.PAYLOAD);
@BindingType(value = HTTPBinding.HTTP_BINDING);
```

3.
```
@ServiceMode(value = Service.Mode.MESSAGE);
```

```
@BindingType(value = HTTPBinding.HTTP_BINDING);
```

4.
```
@ServiceMode(value = Service.Mode.MESSAGE);
@BindingType(value = SOAPBinding.SOAP11HTTP_BINDING);
```

5. None of the above.

Answer:

Correct option is 3.

```
@ServiceMode(value = Service.Mode.MESSAGE);
```

There are two `@ServiceMode`s: `PAYLOAD`, the default, signals that the service wants access only to the underlying message payload (e.g., the body of an HTTP POST request); `MESSAGE` signals that the service wants access to entire message (e.g., the HTTP headers and body). Since the Web Service must have access to HTTP headers (`Accept:`), the correct choice is `MESSAGE` mode:

```
POST /oms HTTP/1.1
Content-Type: text/xml
Host: java.boot.by:8080
Accept: text/html,image/gif,image/jpeg,*;q=.2,*/*;q=.2
Connection: keep-alive
Content-length: 125

<?xml version="1.0" encoding="UTF-8" ?>
<ns1:request xmlns:ns1="...">
   ...
</ns1:request>
```

```
@BindingType(value = HTTPBinding.HTTP_BINDING);
```

The `@BindingType` annotation (`javax.xml.ws.BindingType`) is also defined by the JAX-WS 2.0 specification and is used to specify the binding that should be employed when publishing an endpoint. The property value indicates the actual binding. In this case, you can see that the value is specified as follow: `value=HTTPBinding.HTTP_BINDING`. This indicates that the XML/HTTP binding should be used, rather than the default SOAP 1.1/HTTP. This is how REST endpoints are specified in JAX-WS 2.0 -- by setting the `@BindingType`. If one were to leave the `@BindingType` annotation off, the Java EE 6 container would deploy it as a service that expects to receive a SOAP envelope, rather than straight XML over HTTP. Since the Web Service may return results in plain text, HTML or XML format, the correct choice is `HTTPBinding.HTTP_BINDING`.

Option 1 is wrong. `SOAPBinding.SOAP11HTTP_BINDING` represents SOAP 1.1 over HTTP binding, while the Web Service must use non-SOAP format. Also, with the `javax.xml.ws.Service.Mode.PAYLOAD` mode, the Web Service can only access the contents of the `<soap:Body>` element.

Option 2 is wrong, because the Web Service will not have access to HTTP headers.

Option 4 is wrong, because it uses SOAP 1.1 binding.

Sources:

RESTful Web Services - [`http://www.oracle.com/technetwork/articles/javase/index-137171.html`]

The Java API for XML-Based Web Services (JAX-WS) 2.0 - [http://www.jcp.org/en/jsr/detail?id=224]

Service.Mode enum JavaDoc - [http://download.oracle.com/javase/6/docs/api/javax/xml/ws/Service.Mode.html]

BindingType annotation JavaDoc - [http://download.oracle.com/javase/6/docs/api/javax/xml/ws/BindingType.html]

4.6. Describe the architecture of JAX-WS including the Tools SPI that define the contract between JAX-WS tools and Java EE.

blah-blah

4.7. Describe creating a Web Service using JAX-WS.

Question A040701

You are developing a wrapped-document Web Service using Java-to-WSDL approach. After implementing SIB you proceed to generating WSDL and required portable artifacts for the Web Service. Which statements are true?

Options (select 2):

1. You should use wsgen tool to generate portable artifacts.
2. You should use wsimport tool to generate portable artifacts.
3. You should use wsgen tool to generate WSDL.
4. You should use wsimport tool to generate WSDL.
5. You can not generate WSDL at development time as JAX-WS runtime will automatically generate a WSDL for you when you deploy your service.
6. Only unwrapped-document Web Service requires portable artifacts.

Answer:

Correct options are 1 and 3.

The wsgen tool generates JAX-WS portable artifacts used in JAX-WS Web Services. The tool reads a Web Service Endpoint class and generates all the required artifacts for Web Service deployment.

You can use -wsdl[:protocol] optional flag which will cause wsgen to generate a WSDL file and is usually only used so that the developer can look at the WSDL before the endpoint is deploy. The protocol is optional and is used to specify what protocol should be used in the wsdl:binding. Valid protocols include: soap1.1 and Xsoap1.2. The default is soap1.1.

NOTE: The current Metro release generates the wsgen artifacts automatically.

Options 2 and 4 are wrong. The wsimport tool supports the WSDL-to-Java approach.

The wsimport command-line tool processes an existing Web Services Description Language (WSDL) file and generates the required artifacts for developing Java API for XML-Based Web Services (JAX-WS) Web Service applications.

The wsimport command-line tool supports the top-down approach to developing JAX-WS Web Services. When you start with an existing WSDL file, use the wsimport command-line tool to generate the required JAX-WS artifacts.

Option 5 is wrong, because you can generate WSDL at development time, using -wsdl parameter.

Option 6 is wrong, because any document-style service, wrapped or unwrapped, requires the kind of artifacts that the `wsgen` utility produces.

Sources:

JAX-WS `wsimport` tool - [`http://download.oracle.com/javase/6/docs/technotes/tools/share/wsimport.html`]

JAX-WS `wsgen` tool - [`http://download.oracle.com/javase/6/docs/technotes/tools/share/wsgen.html`]

4.8. Describe JAX-WS Client Communications Models.

blah-blah

4.9. Given a set of requirements, design and develop a Web Service client, such as a Java EE client and a stand-alone client, using JAX-WS.

Question A040901

You are hired by Snorcle, Inc. to develop a Web Service client for the Order Managing System (OMS). The Web Service client must not block until either a response from the Web Service occurs or an exception is thrown. Which JAX-WS types can you use to make an asynchronous call against the service?

Options (select 2):

1. `SOAPHandler`
2. `AsynchronousHandler`
3. `AsyncHandler`
4. `LogicalHandler`
5. `Response`
6. `AsyncResponse`
7. `Request`
8. `AsynchronousRequest`

Answer:

Correct options are 3 and 5.

There are two types of models typically employed by clients for sending and receiving asynchronous messages: request-and-poll and request-and-register.

In a request-and-poll model, a sending application or service sends a message to a receiving application or service. Once the message has been sent, the sender spawns a new thread which polls for a response.

In a request-and-register model, the sender registers as an event listener with the receiver or a proxy of the receiver. Once the sender is registered as a listener, the sender sends the message and the receiver transmits a response to the event callback of the sender or a proxy of the sender.

- Using the polling model, a client can issue a request and receive a response object that can subsequently be polled to determine if the server has responded. When the server responds, the actual response can then be retrieved. The response object returns the response content when the `get()` method is called. The client receives an object of type `javax.xml.ws.Response` from the `invokeAsync` method. That `Response` object is used to monitor the status of the request to the server, determine when the operation has completed,

and to retrieve the response results.

- To implement an asynchronous invocation that uses the callback model, the client provides an `AsynchHandler` callback handler to accept and process the inbound response object. The client callback handler implements the `javax.xml.ws.AsynchHandler` interface, which contains the application code that is run when an asynchronous response is received from the server. The `javax.xml.ws.AsynchHandler` interface contains the `handleResponse(java.xml.ws.Response)` method that is called after the run time has received and processed the asynchronous response from the server. The response is delivered to the callback handler in the form of a `javax.xml.ws.Response` object. The response object returns the response content when the `get()` method is called. Additionally, if an error was received, then an exception is returned to the client during that call.

The following example illustrates a Web Service interface with methods for asynchronous requests from the client:

```
@WebService
public interface OrderManagingService {

    // Synchronous operation
    Order getOrder(Customer customer);

    // Asynchronous operation with polling
    Response<Order> getOrder(Customer customer);

    // Asynchronous operation with callback
    Future<?> getOrder(Customer customer, AsyncHandler<Order> handler);
}
```

- The following example illustrates an asynchronous polling client:

```
Response<Order> response = port.getOrder(cust);

while (!response.isDone()) {
    // do something while we wait
}

order = response.get();
// process the order
```

- The callback method requires a callback handler that is shown in the following example. When using the callback procedure, after a request is made, the callback handler is responsible for handling the response. The response value is a response or possibly an exception. The `Future<?>` method represents the result of an asynchronous computation and is checked to see if the computation is complete. When you want the application to find out if the request is completed, invoke the `Future.isDone()` method. Note that the `Future.get()` method does not provide a meaningful response and is not similar to the `Response.get()` method.

```
Future<?> invocation = port.getOrder(cust,
    new AsyncHandler<Order>() {
        public void handleResponse (Response<Order> response) {
            order = response.get();
            // process the order
        }
    }
);
```

Options 1 and 4 are wrong. `SOAPHandler` and `LogicalHandler` are message interceptors that can be easily plugged in to the JAX-WS runtime to do additional processing of the inbound and outbound messages.

Options 2, 6, 7 and 8 are wrong, because Java types names are invalid.

Sources:

JAX-WS 2.0 Specification - [`http://www.jcp.org/en/jsr/detail?id=224`]

Asynchronous Invocation Using Static Stub - [`http://jax-ws.java.net/jax-ws-20-fcs/docs/asynch.html`]

4.10. Given a set of requirements, create and configure a Web Service client that accesses a stateful Web Service.

blah-blah

Appendix 5. REST, JSON, SOAP and XML Processing APIs (JAXP, JAXB and SAAJ)

5.1. Describe the characteristics of REST Web Services.

Question A050101

You are developing a client to consume a RESTful Web Service. The Web Service can be accessed using the following URL:

```
http://java.boot.by/service
```

Which line generates client-support Java code?

Options (select 1):

1. `wsimport -keep -p client http://java.boot.by/service?wsdl`
2. `wsgen -keep -p client http://java.boot.by/service?wsdl`
3. `xjc -keep -p client http://java.boot.by/service?wsdl`
4. None of the above.

Answer:

Correct option is 4.

The WSDL 1.1 HTTP binding is inadequate to describe communications with HTTP and XML, so there is no way to formally describe REST Web services with WSDL.

Unlike SOAP-based Web Services, which have a standard vocabulary to describe the web service interface through WSDL, RESTful Web Services do not have such grammar. For a service consumer to understand the context and content of the data that must be sent to and received from the service, both the service consumer and service producer must have an out-of-band agreement. This takes the form of documentation, sample code, and an API that the service provider publishes for developers to use.

NOTE: The publication of WSDL 2.0, which was designed with REST Web services in mind, as a World Wide Web Consortium (W3C) recommendation means there is a language to describe REST Web services.

NOTE: OCE WSD 6 exam covers WSDL 1.1, not WSDL 2.0.

Option 1 is wrong. The `wsimport` utility eases the task of writing a Java client against a SOAP-based Web Service. The utility generates client-support code or artifacts from the service contract, the WSDL document.

Option 2 is wrong. The `wsgen` utility produces the classes required to build the WSDL, classes known as wsgen artifacts. The `wsgen` utility also can be used to generate a WSDL document for a Web Service.

Option 3 is wrong. The `xjc` tool accepts an XML schema and generates Java classes. The generated classes contain properties mapped to the XML elements and attributes defined in the XML Schema. The `xjc` tool is a part of the Java API for XML Binding (JAXB) API.

Sources:

RESTful Web Services - [`http://www.oracle.com/technetwork/articles/javase/index-137171.html`]

Describe REST Web services with WSDL 2.0 - [`http://www.ibm.com/developerworks/webservices/library/ws-restwsdl/`]

Question A050102

Snorcle, Inc. parners ask for an integration point in the corporate system to allow them to directly integrate their business processes into the Snorcle, Inc. environment. Since these partners use a variety of of enterprise development platforms for their infrastructure, the development team decides to develop a RESTful Web Service that exposes the necessary integration point to these partners in a platform- and message protocol-neutral way.

Which code fragment demonstrates the SIB declaration?

Options (select 1):

1.
```
public class IntegrationService implements Dispatch<Source> {
    ....
}
```

2.
```
public class IntegrationService implements Provider<Source> {
    ....
}
```

3.
```
public class IntegrationService implements IntegrationServiceSEI {
    ....
}
```

4.
```
public class IntegrationService implements Provider<SOAPMessage> {
    ....
}
```

Answer:

Correct option is 2.

JAX-WS enables building RESTful endpoints through a `javax.xml.ws.Provider` interface in the API. `Provider` is a generic interface that can be implemented by a class as a dynamic alternative to a service endpoint interface (SEI), and a service implementing this interface can be deployed in a Java EE container or published in a stand-alone mode through the JAX-WS Endpoint API. The `Provider` interface contains a single method with the following signature:

```
T invoke(T request)
```

`Provider` is a low-level generic API, but using it requires the endpoint to have an intimate knowledge of the desired message or payload structure being passed to the service. Depending on how the provider is implemented, the supported types for `T` and their uses are the following:

- `javax.xml.transform.Source`. Allows the `Rrovider` to generate and consume XML directly.
- `javax.activation.DataSource`. Works with MIME-typed messages.
- `javax.xml.soap.SOAPMessage`. Conveniently works with and manipulates the entire SOAP message.

All the `Provider` endpoints must have `@WebServiceProvider` annotation.

Option 1 is wrong, because `Dispatch` interface can be used to consume RESTful Web Services.

Option 3 is wrong.

Service Endpoint Interfaces (SEI) provides a high level Java-centric abstraction that hides the details of converting between Java objects and their XML representations for use in XML-based messages. However, in our case it is required for the Web Service to be able to operate at the XML message level. The `Provider` interface offers an alternative to SEIs and may be implemented by services wishing to work at the XML message level.

Option 4 is wrong, because the Web Service is required to work with any arbitrary XML, not only SOAP messages.

Sources:

RESTful Web Services - [`http://www.oracle.com/technetwork/articles/javase/index-137171.html`]

JSR 109: Web Services for Java EE, Version 1.2 (Section 5.3.2.2) - [`http://jcp.org/en/jsr/detail?id=109`]

Question A050103

Which statements are true about REST Web Services?

Options (select 2):

1. URIs act as identifying nouns which identify resources.
2. URIs act as verbs that specify operations on the resources.
3. HTTP methods act as identifying nouns which identify resources.
4. HTTP methods act as verbs that specify operations on the resources.

Answer:

Correct options are 1 and 4.

In RESTful services, URIs act as identifying nouns and HTTP methods act as verbs that specify operations on the resources identified by these nouns.

Principles of REST Web Service Design:

1. The key to creating RESTful Web Services is to identify all of the conceptual entities that you wish to expose as services, e.g. parts list, detailed part data, purchase order.
2. Create a URL to each resource. The resources should be nouns, not verbs. For example, do not

use this:

`http://www.server.com/parts/getPart?id=00345`

Note the verb, `getPart`. Instead, use a noun:

`http://www.server.com/parts/00345`

3. Categorize your resources according to whether clients can just receive a representation of the resource, or whether clients can modify (add to) the resource. For the former, make those resources accessible using an HTTP GET. For the later, make those resources accessible using HTTP POST, PUT, and/or DELETE.
4. All resources accessible via HTTP GET should be side-effect free. That is, the resource should just return a representation of the resource. Invoking the resource should not result in modifying the resource.
5. Put hyperlinks within resource representations to enable clients to drill down for more information, and/or to obtain related information.
6. Design to reveal data gradually. Don't reveal everything in a single response document. Provide hyperlinks to obtain more details.
7. Specify the format of response data using a schema (DTD, W3C Schema, RelaxNG, or Schematron). For those services that require a POST or PUT to it, also provide a schema to specify the format of the response.
8. Describe how your services are to be invoked using either a WSDL 2.0 document, or simply an HTML document.

Source:

RESTful Web Services - [`http://www.oracle.com/technetwork/articles/javase/index-137171.html`]

5.2. Describe the characteristics of JSON Web Services.

blah-blah

5.3. Compare SOAP Web Services to REST Web Services.

Question A050301

A developer is deciding what architectural style (SOAP or REST) would be an appropriate choice for a new application. Which statement is true?

Options (select 1):

1. The REST Web Services can maintain state.
2. The REST Web Services are useful for low bandwidth devices (such as PDAs and mobile phones).
3. The REST Web Services support formal contracts (such as messages, operations, bindings, and location of the Web Service).
4. The REST Web Services can address complex nonfunctional requirements (such as transactions, security).
5. The REST Web Services can handle asynchronous processing and invocation out of the box.

Answer:

Correct option is 2.

A RESTFul design may be appropriate when:

- The Web Services are completely stateless. A good test is to consider whether the interaction can survive a restart of the server. Therefore, option 1 is wrong.
- A caching infrastructure can be leveraged for performance. If the data that the Web Service returns is not dynamically generated and can be cached, then the caching infrastructure that web servers and other intermediaries inherently provide can be leveraged to improve performance. However, the developer must take care because such caches are limited to the HTTP GET method for most servers.
- The service producer and service consumer have a mutual understanding of the context and content being passed along. Because there is no formal way to describe the web services interface, both parties must agree out of band on the schemas that describe the data being exchanged and on ways to process it meaningfully. Therefore, option 3 is wrong.
- Bandwidth is particularly important and needs to be limited. REST is particularly useful for limited-profile devices such as PDAs and mobile phones, for which the overhead of headers and additional layers of SOAP elements on the XML payload must be restricted.
- Web service delivery or aggregation into existing web sites can be enabled easily with a RESTful style. Developers can use technologies such as Asynchronous JavaScript with XML (AJAX) and toolkits such as Direct Web Remoting (DWR) to consume the services in their web applications. Rather than starting from scratch, services can be exposed with XML and consumed by HTML pages without significantly refactoring the existing web site architecture.

A SOAP-based design may be appropriate when:

- A formal contract must be established to describe the interface that the web service offers. The Web Services Description Language (WSDL) describes the details such as messages, operations, bindings, and location of the Web Service.
- The architecture must address complex nonfunctional requirements. Many web services specifications address such requirements and establish a common vocabulary for them. Examples include Transactions, Security, Addressing, Trust, Coordination, and so on. Most real-world applications go beyond simple CRUD operations and require contextual information and conversational state to be maintained. With the RESTful approach, developers must build this plumbing into the application layer themselves. Therefore, option 4 is wrong.
- The architecture needs to handle asynchronous processing and invocation. In such cases, the infrastructure provided by standards such as WSRM and APIs such as JAX-WS with their client-side asynchronous invocation support can be leveraged out of the box. Therefore, option 5 is wrong.

Sources:

RESTful Web Services - [`http://www.oracle.com/technetwork/articles/javase/index-137171.html`]

5.4. Compare SOAP Web Services to JSON Web Services.

blah-blah

5.5. Describe the functions and capabilities of the APIs included within JAXP.

Question A050501

You are parsing XML file which uses DTD with system ID pointing to remote server (see the XML fragment below). The network connection is very slow and it takes much time to download DTD and validate XML. How can you improve application's performance?

```
<!DOCTYPE quiz SYSTEM "http://java.boot.by/quiz.dtd">
```

```
<quiz>
    <name>OCE WSD 6 Quiz</name>
    <author>Mikalai Zaikin</author>
    ...
</quiz>
```

Options (select 1):

1. Use `ErrorHandler` interface.
2. It is not possible to improve performance in such situation without XML modification.
3. Use `EntityResolver` interface.
4. Use `ContentHandler` interface.
5. Use `XMLReader` interface.

Answer:

Correct option is 3.

You can use the `EntityResolver` interface to improve performance in this situation.

Sometimes you want the parser to read from different URLs than the ones the document specifies. Also you might choose to replace that with a cached copy stored locally on the filesystem. A normal web server may be unavailable or may only be accessible through a slow or congested network link; such remote access can cause application slowdowns and failures.

Applications that handle documents with DTDs should plan to use an `EntityResolver` so they work robustly in the face of partial network failures, and so they avoid placing excessive loads on remote servers. That is, they should try to access local copies of DTD data even when the document specifies a remote one. There are many examples of sloppily written applications that broke when a remote system administrator moved a DTD file.

The `EntityResolver` interface allows you to filter the parser's requests for external parsed entities so you can replace the files it requests with your own copies. See the sample code below.

Client XML program:

```
...
XMLReader myXMLReader = XMLReaderFactory.createXMLReader();
myXMLReader.setEntityResolver(new QuizEntityResolver());
myXMLReader.setContentHandler(...);
myXMLReader.parse(...);
```

`QuizEntityResolver` class:

```
public class QuizEntityResolver implements EntityResolver {
        public InputSource resolveEntity(String publicId, String systemId)
                throws FileNotFoundException {

                if (systemId.equals("http://java.boot.by/quiz.dtd")) {
                        // use local copy from filesystem
                        return new InputSource(new FileInputStream("/local/DTD/quiz.dtd"));
                } else {
                        // use the default behaviour
                        return null;
                }
        }
}
```

Option 1 is wrong because `ErrorHandler` interface is used when SAX application needs to implement customized error handling.

Options 2 is wrong because option 3 provides XML parser performance improvement.

Option 4 is wrong because `ContentHandler` interface is used to receive basic document-related events like the start and end of elements and character data.

Option 5 is wrong because `XMLReader` interface allows an application to set and query features and properties in the parser, to register event handlers for document processing, and to initiate a document parse.

Sources:

Interface `EntityResolver` JavaDoc - [`http://www.saxproject.org/apidoc/org/xml/sax/EntityResolver.html`]

Question A050502

Your company uses database as a repository for etalon DTDs. How can you validate XML document against DTD stored in BLOB field of database?

Options (select 1):

1. It is not possible.
2. Use `ErrorHandler` interface.
3. Use `ContentHandler` interface.
4. Use `EntityResolver` interface.

Answer:

Correct option is 4.

Define in XML document a `systemId` with your custom URI scheme and use the `EntityResolver` interface to provide DTD to validate XML document.

When the entity's `systemId` uses a URI scheme that is not understood by the underlying JVM, the `EntityResolver` should be used.

Built-in schemes usually include '`http://`', '`file://`', '`ftp://`', and increasingly '`https://`'.

Schemes not supported by the JVM include '`urn:`' and application-specific schemes. (You may need to put such URI schemes into `publicID` values, in order to prevent problems resolving relative URIs.)

The `EntityResolver` interface allows you to filter the parser's requests for external parsed entities so you can replace the files it requests with your own copies. See the sample code below.

The XML fragment:

```
<!-- we defined our application-specific 'blob:' scheme -->
<!DOCTYPE quiz SYSTEM "blob:quiz.dtd">

<quiz>
        <name>OCE WSD 6 Quiz</name>
        <author>Mikalai Zaikin</author>
        ...
</quiz>
```

Client XML program:

```
...
XMLReader myXMLReader = XMLReaderFactory.createXMLReader();

// use BLOBEntityResolver class to find external DTDs
myXMLReader.setEntityResolver(new BLOBEntityResolver());

myXMLReader.setContentHandler(...);
myXMLReader.parse(...);
```

`BLOBEntityResolver` class:

```
public class BLOBEntityResolver implements EntityResolver {
        public InputSource resolveEntity(String publicId, String systemId)
                throws FileNotFoundException {

                // nonstandard URI scheme ?
                if (systemId.startsWith("blob:")) {
                InputSource retval = new InputSource(systemId);
                        String key = systemId.substring(5);
                        byte data [] = Storage.keyToBlob(key); // retrieve DTD from DB
                        retval.setInputSource(new ByteArrayInputStream(data));
                        return retval;
                } else {
                        ...
                }
        }
}
```

Options 1 is wrong because option 4 provides the requested functionality.

Option 2 is wrong because `ErrorHandler` interface is used when SAX application needs to implement customized error handling.

Option 3 is wrong because `ContentHandler` interface is used to receive basic document-related events like the start and end of elements and character data.

Sources:

Interface `EntityResolver` JavaDoc - [`http://www.saxproject.org/apidoc/org/xml/sax/EntityResolver.html`]

Question A050503

The `EntityResolver` was unable to find external entity. What is the best scenario to hanle this situation?

Options (select 1):

1. Throw `EntityNotFoundException` exception.
2. Return '`null`' value.
3. Throw `IOException` exception.
4. Throw `SAXException` exception.

Answer:

Correct option is 2.

The `EntityResolver` interface contains just a single method, `resolveEntity()`. If you register an `EntityResolver` with an `XMLReader`, then every time that `XMLReader` needs to load an external

parsed entity, it will pass the entity's public ID and system ID to `resolveEntity()` first.

```
package org.xml.sax;

public interface EntityResolver {
    public InputSource resolveEntity(String publicId, String systemId) throws SAXException, IOExce
}
```

The `resolveEntity()` method can either return an `InputSource` or `null`.

If it returns an `InputSource`, then this `InputSource` provides the entity's replacement text.

If it returns `null`, then the parser reads the entity in the same way it would have if there wasn't an `EntityResolver` - just by using the system ID and the `java.net.URL` class:

```
public class QuizEntityResolver implements EntityResolver {
        public InputSource resolveEntity(String publicId, String systemId) throws IOException {
                // try to resolve external entity
                ....

                // nothing helped - return null
                return null;
        }
}
```

Option 1 is wrong because the exception name is invalid.

Options 3 and 4 are wrong because `EntityResolver` should return `null` value to allow XML parser to recover from the situation.

Sources:

Interface `EntityResolver` JavaDoc - [`http://www.saxproject.org/apidoc/org/xml/sax/EntityResolver.html`]

5.6. Describe the functions and capabilities of JAXB, including the JAXB process flow, such as XML-to-Java and Java-to-XML, and the binding and validation mechanisms provided by JAXB.

Question A050601

What are the correct steps to build Java objects from XML file (with validation)?

1.
```
Unmarshaller um = context.createUnmarshaller();
```

2.
```
JAXBContext context = JAXBContext.newInstance("com.example");
```

3.
```
um.setValidating(true);
```

4.
```
JAXBContext context = new JAXBContext("com.example");
```

5.
```
context.setValidating(true);
```

6.
```
Unmarshaller um = context.getUnmarshaller();
```

7.
```
Item item = (Item)um.marshal(new File("item.xml"));
```

8.
```
Item item = (Item)um.unmarshal(new File("item.xml"));
```

Select 4 options in the correct order:

```
[   ]
[   ]
[   ]
[   ]
```

Answer:

Correct options order is 2, 1, 3, 8.

Unmarshalling an XML document means creating a tree of content objects that represents the content and organization of the document. The content tree is not a DOM-based tree. In fact, content trees produced through JAXB can be more efficient in terms of memory use than DOM-based trees. The content objects are instances of the classes produced by the binding compiler. In addition to providing a binding compiler, a JAXB implementation must provide runtime APIs for JAXB-related operations such as marshalling. The APIs are provided as part of a binding framework. The binding framework comprises three packages. The primary package, `javax.xml.bind`, contains classes and interfaces for performing operations such as unmarshalling, marshalling, and validation (marshalling and validation will be covered later). A second package, `javax.xml.bind.util`, contains a number of utility classes. The third package, `javax.xml.bind.helper`, is designed for JAXB implementation providers.

To unmarshal an XML document, you:

1. Create a `JAXBContext` object. This object provides the entry point to the JAXB API. When you create the object, you need to specify a context path. This is a list of one or more package names that contain interfaces generated by the binding compiler. By allowing multiple package names in the context path, JAXB allows you to unmarshal a combination of XML data elements that correspond to different schemas.

   For example, the following code snippet creates a `JAXBContext` object whose context path is `com.example`, the package that contains the interfaces generated for the `item.xsd` schema:

   ```
   import javax.xml.bind.JAXBContext;
   ...
   JAXBContext context = JAXBContext.newInstance("com.example");
   ```

2. Create an `Unmarshaller` object. This object controls the process of unmarshalling. In particular, it contains methods that perform the actual unmarshalling operation. For example, the following code snippet creates an `Unmarshaller` object:

   ```
   import javax.xml.bind.Unmarshaller;
   ...
   Unmarshaller um = context.createUnmarshaller();
   ```

3. You can have JAXB validate the source data against the associated schema as part of the unmarshalling operation. In this case, the statement asks JAXB to validate the source data against its schema. If the data is found to be invalid (that is, it doesn't conform to the schema) the JAXB implementation can report it and might take further action. JAXB providers have a lot of flexibility here. The JAXB specification mandates that all provider implementations report validation errors when the errors are encountered, but the implementation does not have to stop processing the data. Some provider implementations might stop processing when the first

error is found, others might stop even if many errors are found. In other words, it is possible for a JAXB implementation to successfully unmarshal an invalid XML document, and build a Java content tree. However, the result won't be valid. The main requirement is that all JAXB implementations must be able to unmarshal valid documents.

```
um.setValidating(true);
```

4. Call the `unmarshal(...)` method. This method does the actual unmarshalling of the XML document. For example, the following statement unmarshals the XML data in the `item.xml` file:

```
Item item = (Item)um.unmarshal(new File("item.xml"));
```

NOTE: There are other overloaded versions of `unmarshal(...)` that take different types of input: `File`, `java.io.InputStream`, `java.net.URL`, `javax.xml.transform.stream.StreamSource`, `org.w3c.dom.Node`.

Sources:

Java Architecture for XML Binding (JAXB) Technical Article - [`http://www.oracle.com/technetwork/articles/javase/index-140168.html`]

The Java Web Services Tutorial - [`http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/index.html`]

Java Architecture for XML Binding (JAXB) - [`http://jaxb.java.net/`]

Question A050602

How can you validate Java objects content tree created with JAXB before marshalling it to XML file?

Options (select 1)

1.
```
Item item = ...
JAXBContext context = JAXBContext.newInstance("com.example");
Marshaller m = context.createMarshaller();
m.setValidating(true);
m.marshal(item,  new FileOutputStream("newItem.xml"));
```

2.
```
Item item = ...
JAXBContext context = JAXBContext.newInstance("com.example");
Marshaller m = context.createMarshaller();
Validator v = context.createValidator();
v.validate(item);
m.marshal(item,  new FileOutputStream("newItem.xml"));
```

3.
```
Item item = ...
JAXBContext context = JAXBContext.newInstance("com.example");
context.setValidating(true);
Marshaller m = context.createMarshaller();
m.marshal(item,  new FileOutputStream("newItem.xml"));
```

4.
```
Item item = ...
JAXBContext context = JAXBContext.newInstance("com.example");
Marshaller m = context.createMarshaller();
Validator v = context.createValidator();
context.validate(item);
m.marshal(item,  new FileOutputStream("newItem.xml"));
```

```
```

Answer:

Correct option is 2.

Validation is not performed as part of the marshalling operation. Unlike the case for unmarshalling, there is no `setValidating(...)` method for marshalling. Instead, when marshalling data, you use the `Validator` class that is a part of the binding framework to validate a content tree against a schema. For example:

```
import javax.xml.bind.Validator;
...
Validator v = context.createValidator();
v.validate(item);
```

Validating the data as a separate operation from marshalling gives you a lot of flexibility. For example, you can do the validating at one point in time, and do the marshalling at another time. Or you can do some additional processing in between the two operations. Note that the JAXB specification doesn't require a content tree to be valid before it's marshalled. That doesn't necessarily mean that a JAXB implementation will allow invalid data to be marshalled - it might marshal part or all of the invalid data, or not. But all JAXB implementations must be able to marshal valid data.

Sources:

Java Architecture for XML Binding (JAXB) Technical Article - [`http://www.oracle.com/technetwork/articles/javase/index-140168.html`]

The Java Web Services Tutorial - [`http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/index.html`]

Java Architecture for XML Binding (JAXB) - [`http://jaxb.java.net/`]

Question A050603

What are the correct steps to create XML document from Java objects content tree?

1.
```
Marshaller m = context.createMarshaller();
```

2.
```
JAXBContext context = JAXBContext.newInstance("com.example");
```

3.
```
m.setValidating(true);
```

4.
```
JAXBContext context = new JAXBContext("com.example");
```

5.
```
context.setValidating(true);
```

6.
```
Marshaller m = context.getMarshaller();
```

7.
```
m.marshal(item, new FileOutputStream("newItem.xml"));
```

8.
```
m.unmarshal(item, new FileOutputStream("newItem.xml"));
```

Select 3 options in the correct order:

```
[    ]
[    ]
[    ]
```

Answer:

Correct options order is 2, 1, 7.

Marshalling is the opposite of unmarshalling. It creates an XML document from a content tree. To marshal a content tree, you:

1. Create a `JAXBContext` object, and specify the appropriate context path - that is, the package that contains the classes and interfaces for the bound schema. As is the case for unmarshalling, you can specify multiple package names in the context path. That gives you a way of building an XML document using a combination of XML data elements that correspond to different schemas:

   ```
   import javax.xml.bind.JAXBContext;
   ...
   JAXBContext context = JAXBContext.newInstance("com.example");
   ```

2. Create a `Marshaller` object. This object controls the process of marshalling. In particular, it contains methods that perform the actual marshalling operation:

   ```
   import javax.xml.bind.Marshaller;
   ...
   Marshaller m = context.createMarshaller();
   ```

3. Call the `marshal(...)` method. This method does the actual marshalling of the content tree. When you call the method, you specify an object that contains the root of the content tree, and the output target. For example, the following statement marshals the content tree whose root is in the `item` object and writes it as an output stream to the XML file `newItem.xml`:

   ```
   m.marshal(item, new FileOutputStream("newItem.xml"));
   ```

   NOTE: There are other overloaded versions of `marshal(...)` that take different types of output target: `org.xml.sax.ContentHandler`, `java.io.OutputStream`, `javax.xml.transform.Result`, `java.io.Writer`, `org.w3c.dom.Node`.

Sources:

Java Architecture for XML Binding (JAXB) Technical Article - [`http://www.oracle.com/technetwork/articles/javase/index-140168.html`]

The Java Web Services Tutorial - [`http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/index.html`]

Java Architecture for XML Binding (JAXB) - [`http://jaxb.java.net/`]

Question A050604

You are performing development of Web Service, followed by deployment. What is the correct sequence of steps in JAXB data binding process?

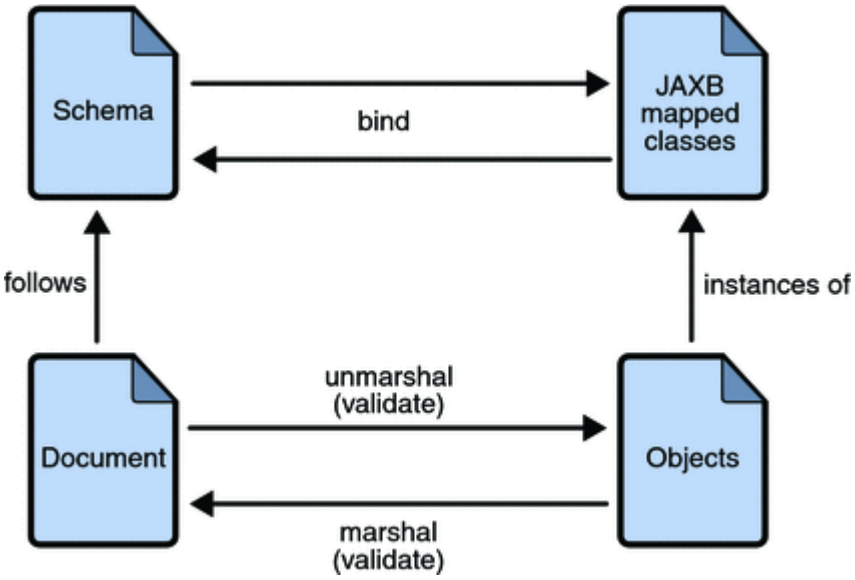
Options (select 1):

1. Generate classes, compile, marshal, process content, unmarshal.
2. Marshal, process content, generate classes, compile, unmarshal.
3. Generate classes, compile, unmarshal, process content, marshal.
4. Unarshal, process content, generate classes, compile, marshal.

Answer:

Correct option is 3.

The figure below shows what occurs during the JAXB binding process:



The general steps in the JAXB data binding process are:

1. Generate classes: An XML schema is used as input to the JAXB binding compiler to generate JAXB classes based on that schema.
2. Compile classes: All of the generated classes, source files, and application code must be compiled.
3. Unmarshal: XML documents written according to the constraints in the source schema are unmarshalled by the JAXB binding framework. Note that JAXB also supports unmarshalling XML data from sources other than files/documents, such as DOM nodes, string buffers, SAX Sources, and so forth.
4. Generate content tree: The unmarshalling process generates a content tree of data objects instantiated from the generated JAXB classes; this content tree represents the structure and content of the source XML documents.
5. Validate (optional): The unmarshalling process optionally involves validation of the source XML documents before generating the content tree. Note that if you modify the content tree in next step, you can also use the JAXB Validate operation to validate the changes before marshalling the content back to an XML document.
6. Process content: The client application can modify the XML data represented by the Java content tree by means of interfaces generated by the binding compiler.
7. Marshal: The processed content tree is marshalled out to one or more XML output documents. The content may be validated before marshalling.

Sources:

The Java EE5 Tutorial - [`http://download.oracle.com/javaee/5/tutorial/doc/bnazg.html`]

5.7. Create and use a SOAP message with attachments using the SAAJ APIs.

Question A050701

A developer has a requirement to send a spreadsheet report to Web Service in a SOAP message. What SAAJ class the developer can use?

Options (select 1):

1. `Attachment`
2. `SOAPAttachment`
3. `MimeAttachment`
4. `DataHandler`
5. `None of the above`

Answer:

Correct option is 5.

A SOAP message can include one or more attachment parts in addition to the SOAP part. The SOAP part must contain only XML content. As a result, if any content in a message is not in XML format (for example, spreadsheet document), it must be contained in an attachment part.

If you want a SOAP message to contain a binary file, the message MUST have an attachment part for it.

SAAJ provides the `javax.xml.soap.AttachmentPart` class to represent an attachment part of a SOAP message. If a `SOAPMessage` object has one or more attachments, each `AttachmentPart` object must have a MIME header to indicate the type of data it contains. The `AttachmentPart` object can also have additional MIME headers to identify it or to give its location. These headers are optional, but can be useful when there are multiple attachments.

Options 1, 2, and 3 are wrong, because such classes do not exist in SAAJ API.

Option 4 is wrong because `DataHandler` class does not belong to SAAJ API, it is part of Java Activation Framework (JAF).

The `DataHandler` class encapsulates a handler that can be mapped to a certain MIME type. The handler implements streaming, in and out, as well as activation using a command map.

```
MessageFactory factory = MessageFactory.newInstance();
SOAPMessage message = factory.createMessage();
...
URL url = new URL("file:///c:/documents/report.xls");
DataHandler datahandler = new DataHandler(url);
AttachmentPart attachment = message.createAttachmentPart(dataHandler);
attachment.setContentID ("attached_report");
message.addAttachmentPart(attachment);
```

Sources:

SAAJ Tutorial - [`http://download.oracle.com/javaee/5/tutorial/doc/bnbhr.html`]

`AttachmentPart` JavaDoc - [`http://download.oracle.com/javaee/5/api/javax/xml/soap/AttachmentPart.html`]

Question A050702

You deployed collaborating handlers that sign outgoing messages by placing a signature in the header and verify that the signature on incoming messages is correct. After you verified signature,

the header element is not needed. How can you get signature header and remove it from incoming SOAP message in one method call?

Options (select 1):

1. ```
   Iterator headerElements = header.examineHeaderElements("sign-handler");
   ```
2. ```
   Iterator headerElements = header.removeHeaderElements("sign-handler");
   ```
3. ```
   Iterator headerElements = header.extractHeaderElements("sign-handler");
   ```
4. ```
   Iterator headerElements = header.retrieveHeaderElements("sign-handler");
   ```
5. It is not possible to do with one method call.

Answer:

Correct option is 3.

The `SOAPHeader` interface provides two methods that return a `java.util.Iterator` object over all of the `SOAPHeaderElement` objects with an actor that matches the specified actor. The first method, `examineHeaderElements(String actor)`, returns an iterator over all of the elements with the specified actor.

```
java.util.Iterator headerElements = header.examineHeaderElements("sign-handler");
```

The second method, `extractHeaderElements(String actor)`, not only returns an iterator over all of the `SOAPHeaderElement` objects with the specified actor attribute but also detaches them from the `SOAPHeader` object.

```
java.util.Iterator headerElements = header.extractHeaderElements("sign-handler");
```

Option 1 is wrong because header elements are not detached.

Options 2 and 4 are wrong because methods names are invalid.

Option 5 is wrong because it poossible to do with one method call.

Below is fragment of `SOAPHeader` API:

```
package javax.xml.soap;

public interface SOAPHeader extends SOAPElement {

        /*
        Returns an Iterator over all the SOAPHeaderElement objects in this SOAPHeader object that
        is a global attribute that indicates the intermediate parties that should process a messag
        An actor receives the message and processes it before sending it on to the next actor. The
        for the message, so if no actor attribute is included in a SOAPHeader object, it is sent t
        */
        public Iterator examineHeaderElements(String actor);

        /*
        Returns an Iterator over all the SOAPHeaderElement objects in this SOAPHeader object that
        SOAPHeader object.
        This method allows an actor to process the parts of the SOAPHeader object that apply to it
        on to the next actor.
```

```
        */
        public Iterator extractHeaderElements(String actor);

        ...
}
```

Sources:

`SOAPHeader` API JavaDoc - [`http://download.oracle.com/javase/6/docs/api/javax/xml/soap/SOAPHeader.html`]

Question A050703

Which SAAJ code fragment produces the following SOAP Header?

```
<SOAP-ENV:Header>
    <wsi:Claim
        xmlns:wsi="http://ws-i.org/schemas/conformanceClaim/"
        conformsTo="http://ws-i.org/profiles/basic/1.1/" />
</SOAP-ENV:Header>
```

Options (select 1):

1.
```
SOAPHeader header = message.getSOAPHeader();
Name headerName = message.createName("Claim", "wsi", "http://ws-i.org/schemas/conformanceClai
SOAPHeaderElement headerElement = header.addHeaderElement(headerName);
headerElement.addAttribute(message.createName("conformsTo"), "http://ws-i.org/profiles/basic/
```

2.
```
SOAPHeader header = message.getSOAPHeader();
Name headerName = soapFactory.createName("Claim", "wsi", "http://ws-i.org/schemas/conformance
header.addHeaderElement(headerName);
header.addAttribute(soapFactory.createName("conformsTo"), "http://ws-i.org/profiles/basic/1.1
```

3.
```
SOAPHeader header = message.getSOAPHeader();
Name headerName = soapFactory.createName("Claim", "wsi", "http://ws-i.org/schemas/conformance
SOAPHeaderElement headerElement = header.addHeaderElement(headerName);
headerElement.addAttribute(soapFactory.createName("conformsTo"), "http://ws-i.org/profiles/ba
```

4.
```
SOAPHeader header = message.getSOAPHeader();
Name headerName = message.createName("Claim", "wsi", "http://ws-i.org/schemas/conformanceClai
header.addHeaderElement(headerName);
header.addAttribute(message.createName("conformsTo"), "http://ws-i.org/profiles/basic/1.1/");
```

Answer:

Correct option is 3.

To add content to the `SOAPHeader`, you need to create a `SOAPHeaderElement` object. As with all new elements, it must have an associated `Name` object, which you can create using the message's `SOAPEnvelope` object or a `SOAPFactory` object.

For example, suppose you want to add a conformance claim header to the message to state that your message conforms to the WS-I Basic Profile.

The following code fragment retrieves the `SOAPHeader` object from SOAP message:

```
SOAPHeader header = message.getSOAPHeader();
```

Then you create future header element `Name` object using `SOAPFactory`:

```
Name headerName = soapFactory.createName("Claim", "wsi", "http://ws-i.org/schemas/conformanceClaim
```

where "`Claim`" - element name, "`wsi`" - namespace prefix, "`http://ws-i.org/schemas/conformanceClaim/`" - namespace URI.

Then you add header element to SOAP header:

```
SOAPHeaderElement headerElement = header.addHeaderElement(headerName);
```

Finally, you add attribute for a WS-I conformance claim header:

```
headerElement.addAttribute(soapFactory.createName("conformsTo"), "http://ws-i.org/profiles/basic/1
```

For a different kind of header, you might want to add content to `headerElement`. The following line of code uses the method `addTextNode` to do this:

```
headerElement.addTextNode("order");
```

Now you have the `SOAPHeader` object header that contains a `SOAPHeaderElement` object whose content is "`order`".

Sources:

SAAJ Tutorial - [`http://download.oracle.com/javaee/5/tutorial/doc/bnbhr.html`]

`SOAPFactory` API JavaDoc - [`http://download.oracle.com/javase/6/docs/api/javax/xml/soap/SOAPFactory.html`]

Question A050704

You are constructing a valid `SOAPMessage` object. What SAAJ types are optional?

Options (select 2):

1. `javax.xml.soap.SOAPPart`
2. `javax.xml.soap.SOAPBody`
3. `javax.xml.soap.SOAPHeader`
4. `javax.xml.soap.SOAPEnvelope`
5. `javax.xml.soap.AttachmentPart`

Answer:

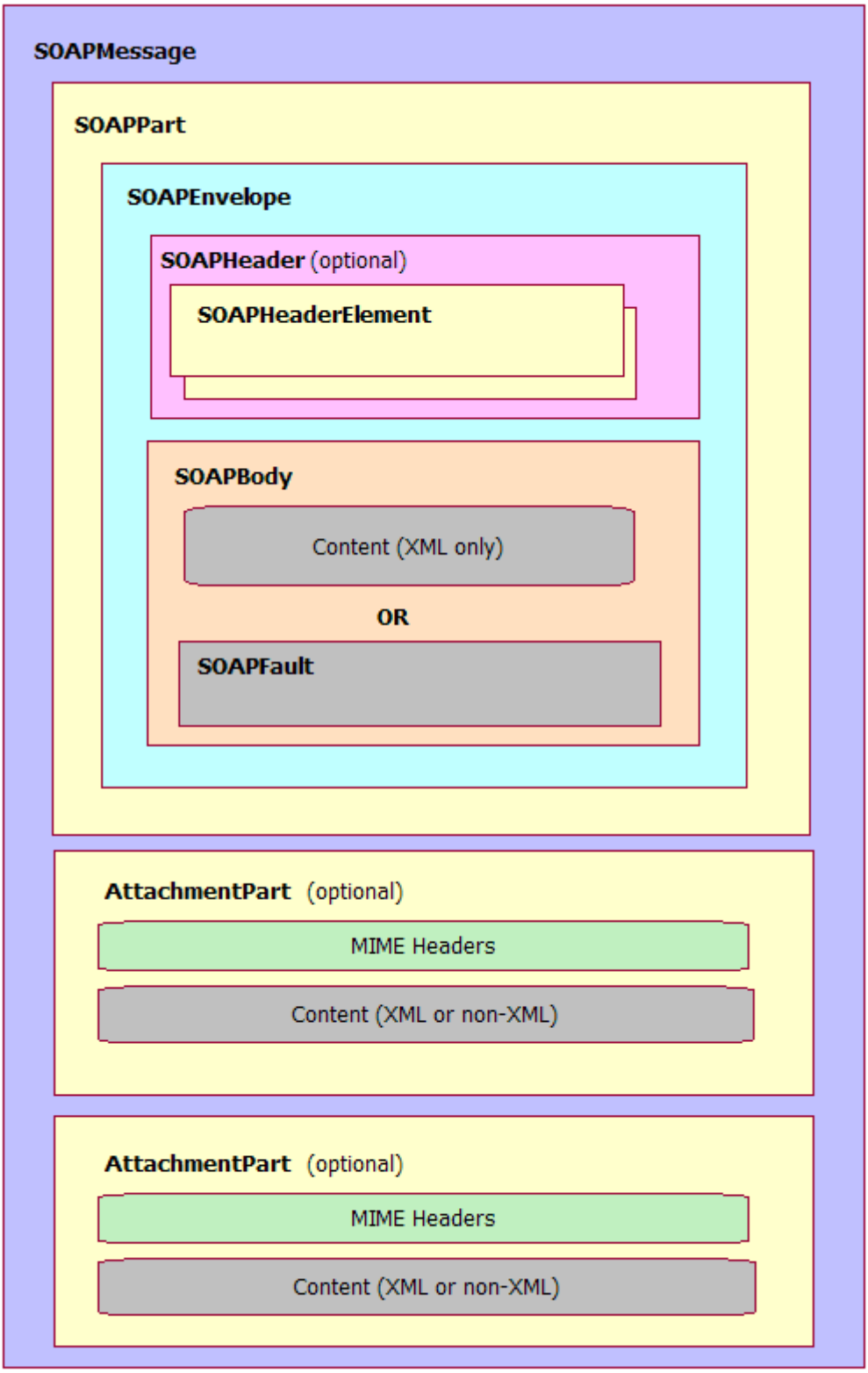
Correct options are 3 and 5.

The `SOAPEnvelope` object is a container object for the `SOAPHeader` and `SOAPBody` portions of a `SOAPPart` object. The `SOAPEnvelope` object must contain a `SOAPBody` object, but the `SOAPHeader` object is optional.

The `javax.xml.soap.AttachmentPart` presents a single attachment to a `SOAPMessage` object. A

`SOAPMessage` object may contain zero, one, or many `AttachmentPart` objects. Each `AttachmentPart` object consists of two parts, application-specific content and associated MIME headers. The MIME headers consists of name/value pairs that can be used to identify and describe the content. There are no restrictions on the content portion of an `AttachmentPart` object. The content may be anything from a simple plain text object to a complex XML document or image file.

Options 1, 2 and 4 are wrong because they must exist in `SOAPMessage` object.



Sources:

Overview of SAAJ - [http://download.oracle.com/javaee/5/tutorial/doc/bnbhg.html]

Question A050705

You are writing a SOAP handler which extends `javax.xml.ws.handler.soap.SOAPHandler`:

```
public boolean handleMessage(SOAPMessageContext smc) {
    SOAPMessage message = smc.getMessage();
    // Use SAAJ API to access SOAP header
    ...
    return true;
}
```

The handler must inspect SOAP header information. How can you access SOAP header using SAAJ API?

Options (select 2):

1.
```
SOAPPart soapPart = message.getSOAPPart();
SOAPEnvelope envelope = soapPart.getSOAPEnvelope();
SOAPHeader header = envelope.getSOAPHeader();
```

2.
```
SOAPHeader header = message.getHeader();
```

3.
```
SOAPPart soapPart = message.getSOAPPart();
SOAPEnvelope envelope = soapPart.getEnvelope();
SOAPHeader header = envelope.getHeader();
```

4.
```
SOAPHeader header = message.getSOAPHeader();
```

Answer:

Correct options are 3 and 4.

Option 1 is wrong, because `soapPart.getSOAPEnvelope()` and `envelope.getSOAPHeader()` method names are invalid.

Option 2 is wrong, because `message.getHeader()` method name is invalid.

Option 3 is correct. The first way to access the SOAP header is to get the `SOAPPart` object from the message object, then the `SOAPEnvelope` object, and finally the `SOAPHeader` object from the envelope:

```
SOAPPart sp = message.getSOAPPart();
SOAPEnvelope se = sp.getEnvelope();
SOAPHeader sh = se.getHeader();
```

Option 4 is correct. The second way to access the SOAP header is to retrieve the message header directly, without retrieving the `SOAPPart` or `SOAPEnvelope`. To do so, use the `getSOAPHeader` method of `SOAPMessage`, as follows:

```
SOAPHeader sh = message.getSOAPHeader();
```

Sources:

`SOAPEnvelope` JavaDoc API - [http://download.oracle.com/javaee/5/api/javax/xml/soap

/SOAPEnvelope.html]

SOAPMessage JavaDoc API - [http://download.oracle.com/javaee/5/api/javax/xml/soap/SOAPMessage.html]

SAAJ Tutorial - [http://download.oracle.com/javaee/5/tutorial/doc/bnbhr.html]

Appendix 6. JAXR

6.1. Describe the function of JAXR in Web Service architectural model, the two basic levels of business registry functionality supported by JAXR, and the function of the basic JAXR business objects and how they map to the UDDI data structures.

Question A060101

What statements are true about JAXR 1.0 ?

Options (select 2):

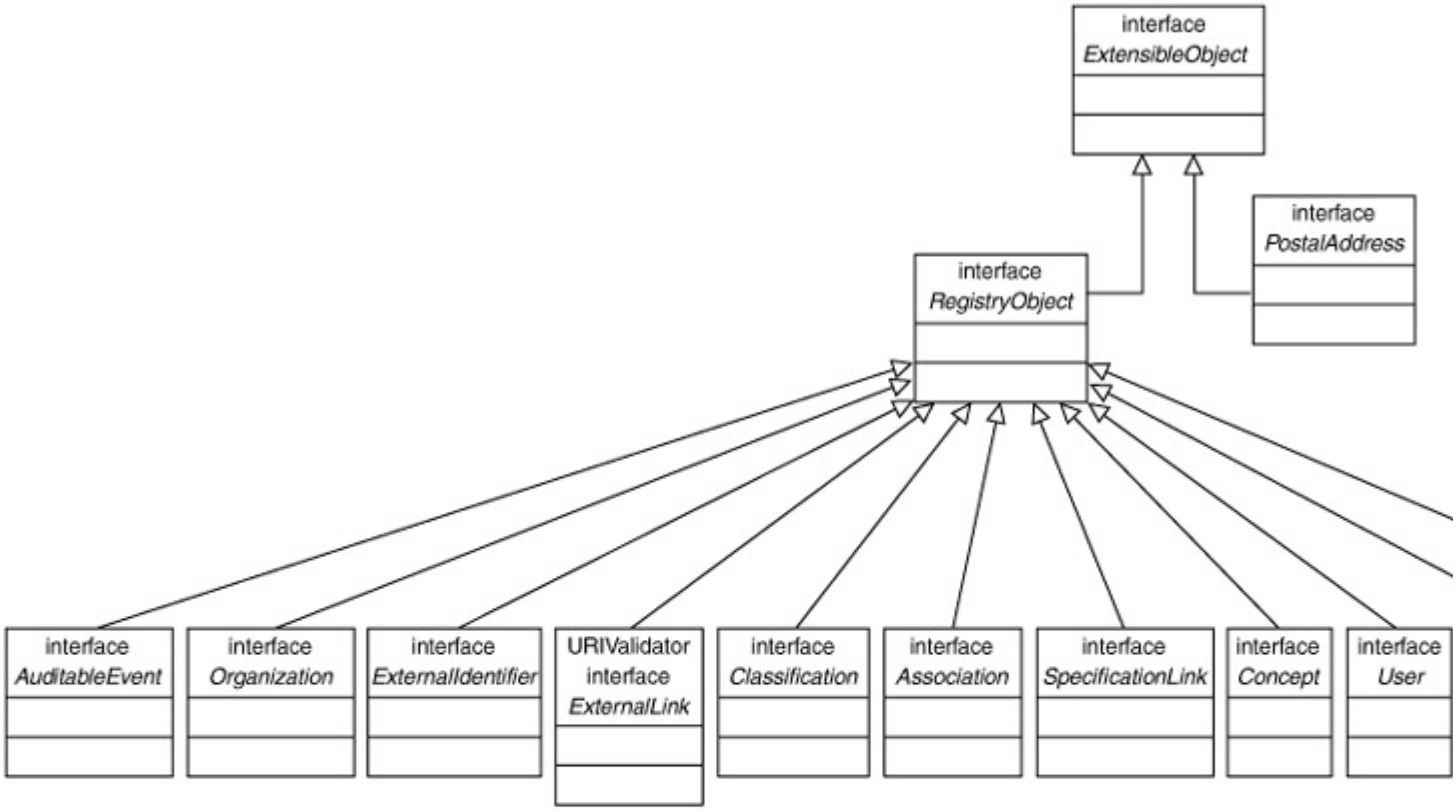
1. Each Registry Object can be identified by unique ID.
2. JAXR allows only to search Registry Objects.
3. JAXR allows to publish, search and modify Registry Objects.
4. JAXR clients can load pluggable Capability Profiles to use advanced search functions.

Answer:

Correct options are 1 and 3.

The JAXR information model has been developed with an object-oriented approach and is therefore easy to understand. It uses inheritance between different interfaces to abstract out the common behavior. At the core of the information model is a javax.xml.registry.infomodel.RegistryObject.

The figure below shows the physical generalization relationship. All the objects that conceptually must be represented or sent to the registry, implement this interface. In short, from an abstract perspective, everything in the registry is a RegistryObject:



`javax.xml.registry.infomodel.RegistryObject` has the following methods to get and set key representing the universally unique ID (UUID):

- `Key getKey()` - Gets the key representing the universally unique ID (UUID) for this object.
- `void setKey(Key key)` - Sets the key representing the universally unique ID (UUID) for this object.

Option 2 is wrong and option 3 is correct. The JAXR specification defines two life-cycle management interfaces:

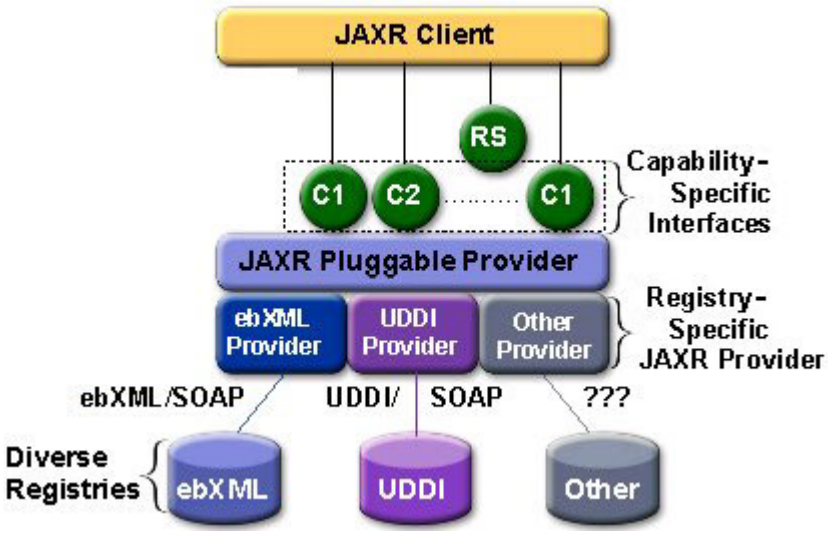
- `BusinessLifeCycleManager` (Capability Level 0)
- `LifeCycleManager` (Capability Level 1)

`BusinessLifeCycleManager` defines a simple business-level API for life-cycle management. This interface resembles the publisher's API in UDDI, which should prove familiar to the UDDI developer. For its part, `LifeCycleManager` interface provides complete support for all life-cycle management needs using a generic API.

Life-cycle management includes creating, saving, updating, deprecating, and deleting registry objects. In addition, the `LifeCycleManager` provides several factory methods to create JAXR information model objects. In general, life-cycle management operations are privileged, while a user

can use query management operations for browsing the registry.

Option 4 is wrong. Because some diversity exists among registry provider capabilities, the JAXR expert group decided to provide multilayer API abstractions through capability profiles. Each method of a JAXR interface is assigned a Capability Level, and those JAXR methods with the same Capability Level define the JAXR provider capability profile.



Currently, JAXR defines only two capability profiles: level 0 profile for basic features and level 1 profile for advanced features. Level 0's basic features support so-called business-focused APIs, while level 1's advanced features support generic APIs. At the minimum, all JAXR providers must implement a level 0 profile. A JAXR client application using only those methods of the level 0 profile can access any JAXR provider in a portable manner. JAXR providers for UDDI must be level 0 compliant.

JAXR providers can optionally support the level 1 profile. The methods assigned to this profile provide more advanced registry capabilities needed by more demanding JAXR clients. Support for the level 1 profile also implies full support for the level 0 profile. JAXR providers for ebXML must be level 1 compliant. A JAXR client can discover the capability level of a JAXR provider by invoking methods on the `CapabilityProfile` interface. If the client attempts to invoke capability level methods unsupported by the JAXR provider, the provider will throw an `UnsupportedCapabilityException`.

Sources:

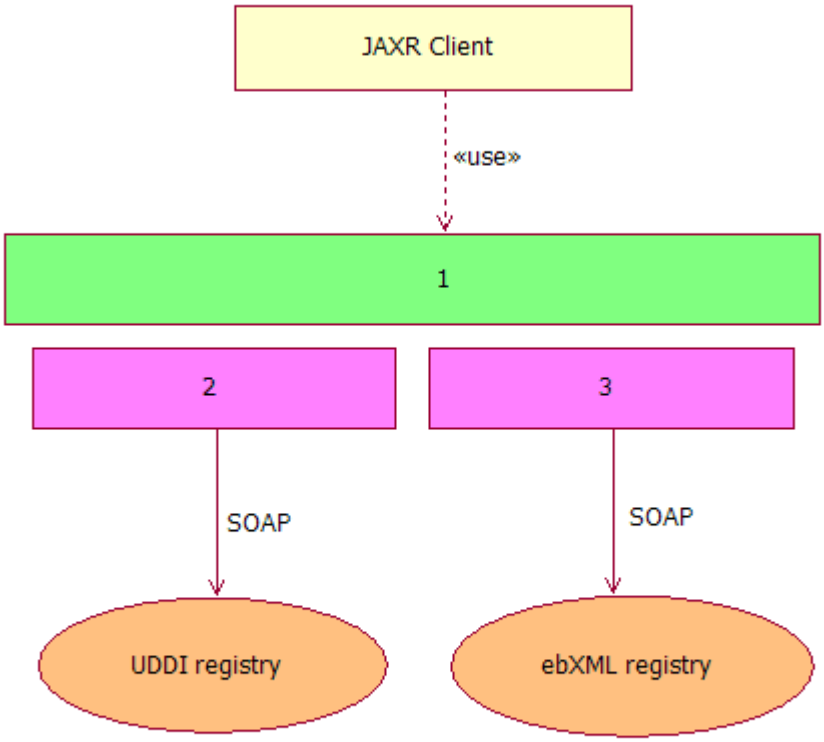
Overview of JAXR - [`http://download.oracle.com/javaee/1.4/tutorial/doc/JAXR2.html`]

`RegistryObject` Interface JavaDoc - [`http://download.oracle.com/javaee/6/api/javax/xml/registry/infomodel/RegistryObject.html`]

Discover and publish Web services with JAXR - [`http://www.javaworld.com/javaworld/jw-06-2002/jw-0614-jaxr.html`]

Question A060102

Looking at the JAXR architecture, name its components.



JAXR Architecture

Options (select 1):

1. 1 - JAXR API Capability Specific interfaces
   2 - Pluggable JAXR UDDI Provider
   3 - Pluggable JAXR ebXML Provider
2. 1 - JAXR API Capability Specific interfaces
   2 - Pluggable JAXR UDDI Client
   3 - Pluggable JAXR ebXML Client
3. 1 - SOAP API Capability Specific Interfaces
   2 - Pluggable JAXR Generic Provider
   3 - Pluggable JAXR Generic Provider
4. 1 - JAXB API Capability Specific interfaces
   2 - Pluggable JAXR UDDI Client
   3 - Pluggable JAXR ebXML Client

Answer:

Correct option is 1.

The high-level architecture of JAXR consists of the following parts:

- A JAXR Client: a client program that uses the JAXR API to access a business registry via a JAXR

Provider.

- JAXR API (Capability profile "0" and/or "1"): consists of 2 main packages `javax.xml.registry` and `javax.xml.registry.infomodel`.
- A JAXR Provider: an implementation of the JAXR API that provides access to a specific registry provider or to a class of registry providers that are based on a common specification.

JAXR Client uses the Capability Level "0" or "1" interfaces of the JAXR API to access the JAXR Provider. The JAXR Provider in turn accesses a registry (UDDI or ebXML).

Sources:

JAXR Home Page - [`http://java.sun.com/webservices/jaxr/index.jsp`]

Java Web Services Tutorial - [`http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/index.html`]

Question A060103

In the JAXR information model what interfaces map to the UDDI `tModel` XML data structure?

Options (select 1):

1. `Concept` and `Classification`
2. `Concept` and `ClassificationScheme`
3. `ConceptScheme` and `Classification`
4. `Concept` and `keyedReference`

Answer:

Correct option is 2.

In UDDI, `tModel` is an overloaded concept that can be used for a few different purposes. The following are two broad categories of purposes that `tModels` serve in UDDI:

- To serve as a namespace for a taxonomy (e.g. NAICS) or identification scheme (DUNS)
- To serve as a fingerprint or proxy for a technical specification that lives outside the registry in a `bindingTemplate`

In the JAXR the above two uses of `tModel` are modeled separately. The namespace use is modeled with the `ClassificationScheme` interface, while the technical fingerprint use is modeled with any `RegistryObject` which in case of a UDDI provider must be a `Concept`. The `SpecificationLink.getSpecificationObject` method must return a `Concept` instance for a UDDI provider.

Options 1 and 3 are wrong because they contain invalid interface names.

Option 4 is wrong because `keyedReference` element is used either to contain a group of classifications or to contain a group of identifiers for an object. To that end, `keyedReference` can map to either `ExternalIdentifiers` or `Classifications`.

Sources:

JAXR Specification [Appendix D] - [`http://jcp.org/en/jsr/detail?id=93`]

Question A060104

What is true about JAXR?

Options (select 1):

1. It provides a uniform and standard Java API for accessing different kinds of XML registries.
2. It enables flexible XML processing from within Java programs.
3. It enables simplified XML processing using Java classes that are generated from XML schemas.
4. It provides an API for XML-based RPC communication in the Java platform.

Answer:

Correct option is 1.

Option 1 is correct. The Java API for XML Registries (JAXR) API provides a uniform and standard Java API for accessing different kinds of XML Registries. XML registries are an enabling infrastructure for building, deployment, and discovery of Web services.

Currently there are a variety of specifications for XML registries including, most notably, the ebXML Registry and Repository standard, which is being developed by OASIS and U.N./CEFACT, and the UDDI specification, which is being developed by a vendor consortium.

JAXR enables Java software programmers to use a single, easy-to-use abstraction API to access a variety of XML registries. Simplicity and ease of use are facilitated within JAXR by a unified JAXR information model, which describes content and metadata within XML registries.

JAXR provides rich metadata capabilities for classification and association, as well as rich query capabilities. As an abstraction-based API, JAXR gives developers the ability to write registry client programs that are portable across different target registries. This is consistent with the Java philosophy of "Write Once, Run Anywhere." Similarly, JAXR also enables value-added capabilities beyond those of the underlying registries.

The current version of the JAXR specification includes detailed bindings between the JAXR information model and both the ebXML Registry and the UDDI Registry v2.0 specifications.

JAXR works in synergy with related Java APIs for XML, such as Java API for XML Processing (JAXP), Java Architecture for XML Binding (JAXB), Java API for XML-based RPC (JAX-RPC), and SOAP with Attachments API for Java (SAAJ), to enable Web services within the Java 2 Platform, Enterprise Edition (J2EE).

Option 2 is wrong. Java API for XML Processing or JAXP enables flexible XML processing from within Java programs. The JAXR API will make direct XML processing less important for JAXR clients. However, JAXP may be used by implementers of JAXR providers and JAXR clients for processing XML content that is submitted to or retrieved from the Registry. The JAXP API is likely to also be used in implementations of the JAXB API.

Option 3 is wrong. Java API for XML Data Binding or JAXB enables simplified XML processing using Java classes that are generated from XML schemas. The JAXR API will make direct XML processing less important for JAXR clients. However, JAXB may be used by implementers of JAXR providers and JAXR clients for processing XML content that is submitted to or retrieved from the Registry.

Option 4 is wrong. Java API for XML RPC or JAX-RPC provides an API for XML-based RPC communication in the Java platform. Implementations of the JAXR providers may use JAX-RPC for communication between JAXR providers and registry providers that export a SOAP-based RPC like interface (e.g. UDDI).

Sources:

JAXR Home Page - [`http://java.sun.com/webservices/jaxr/index.jsp`]

Java Web Services Tutorial - [`http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/index.html`]

Question A060105

Which capability profile level should support JAXR provider when publishing or discovering WS-I BP 1.1 conformant Web Service?

Options (select 1):

1. Capability Level 0
2. Capability Level 1
3. Capability Level 2
4. Capability Level 3

Answer:

Correct option is 1.

There are two capability profiles set up in the JAXR: Capability Level 0 and Capability Level 1.

A JAXR provider that supports the UDDI registry specification must be Level 0-compliant, whereas an ebXML registry must be Level 1-compliant. These capability levels make it easier to add and remove functionality without changing class structures because the capability profiles are based on industry agreement, not tied to checks make by the Java compiler or runtime. JAXR providers publish the capability level of their implementation, and must agree not to offer any capabilities that exceed their capability level so that portability can be ensured.

This means options 3 and 4 are wrong.

WS-I BP 1.1 mandates to use UDDI for Web Service publishing and discovering:

When publication or discovery of Web services is required, UDDI is the mechanism the Basic Profile has adopted to describe Web service providers and the Web services they provide. Business, intended use, and Web service type descriptions are made in UDDI terms; detailed technical descriptions are made in WSDL terms. Where the two specifications define overlapping descriptive data and both forms of description are used, the Profile specifies that the descriptions must not conflict.

Registration of Web service instances in UDDI registries is optional. By no means do all usage scenarios require the kind of metadata and discovery UDDI provides, but where such capability is needed, UDDI is the sanctioned mechanism.

This means option 2 is wrong.

Sources:

WS-I Basic Profile Version 1.1 - [`http://www.ws-i.org/Profiles/BasicProfile-1.1.html`]

Question A060106

You are using JAXR for Web Services discovering and publishing. What is true about XML registry security?

Options (select 3):

1. JAXR Provider must authenticate with UDDI registry.
2. Authentication is performed when Client creates JAXR connection.
3. JAXR Client must authenticate with UDDI registry.
4. JAXR Provider sends queries to UDDI registry.
5. Authentication is performed only on certain queries types transparently for JAXR Client.
6. JAXR Client sends queries to UDDI registry.

Answer:

Correct options are 1, 4 and 5.

JAXR Client does not compose and send SOAP messages directly to UDDI registry. JAXR Client invoke JAXR provider's methods, JAXR provider (on the client side) creates SOAP messages and send to XML registry.

JAXR Client <---[method call]---> JAXR Provider <---[SOAP]---> UDDI

From the perspective of the target registry provider, it is the JAXR Provider, not the JAXR Client, that is the registry providerв_Ts client. The JAXR Provider must authenticate with the registry provider as specified by the specification governing the registry provider (e.g. ebXML Registry, UDDI). Typically, such authentication occurs on certain privileged requests. For example, in UDDI, authentication is needed only on requests that use the UDDI publishing API to submit, update, or delete content.

In all cases, the JAXR Provider initiates the authentication requests, while the registry provider performs the actual authentication.

The JAXR Client does not directly initiate authentication. It does not need to know when authentication with the target registry is necessary nor how it must be done. Instead, this functionality is delegated to the JAXR Provider.

Option 2 is wrong because client might never be autenticated if it uses only Inquiry API.

Option 3 is wrong because JAXR provider sends messages to UDDI registry, so JAXR Provider must authenticate with UDDI registry.

Option 6 is wrong because JAXR Provider sends SOAP messages to UDDI registry, JAXR client just calls methods on JAXR Provider classes.

Sources:

JAXR Specification (10.3 Authentication) - [`http://jcp.org/en/jsr/detail?id=93`]

Question A060107

Which JAXR method maps to UDDI Publisher API "`discard_authToken`" method(s)?

Options (select 1):

1. `BusinessLifeCycleManager.discardAuthToken()`
2. `BusinessLifeCycleManager.discard_authToken()`
3. `AuthenticationToken.discard()`
4. `AuthorizationToken.discard()`
5. Both `BusinessLifeCycleManager.discard_authToken()` and `AuthenticationToken.discard()`
6. None of the above.

Answer:

Correct option is 6.

There is no method in JAXR API which maps to UDDI Publisher API "`discard_authToken`" call.

In all cases, the JAXR Provider initiates the authentication requests, while the registry provider performs the actual authentication.

The JAXR Client does not directly initiate authentication. It does not need to know when authentication with the target registry is necessary nor how it must be done. Instead, this

functionality is delegated to the JAXR Provider.

Since authentication is handled transparently by JAXR provider, the method to discard authentication token is not needed.

Sources:

JAXR Specification (D.2 Mapping of UDDI Publisher API Calls to JAXR) - [`http://jcp.org/en/jsr/detail?id=93`]

Question A060108

Which methods allow the client to use SQL syntax for XML registry queries?

Options (select 1):

1. `BusinessQueryManager`'s Capability Level 0 methods
2. `DeclarativeQueryManager`'s Capability Level 0 methods
3. `BusinessQueryManager`'s Capability Level 1 methods
4. `DeclarativeQueryManager`'s Capability Level 1 methods

Answer:

Correct option is 4.

`DeclarativeQueryManager` interface provides the ability to execute declarative queries (e.g. SQL) and the methods for SQL queries belong to Capability Level 1.

```
// Creates a Query object given a queryType (for example, QUERY_TYPE_SQL) and a String that repres
// a query in the syntax appropriate for queryType
Query createQuery(int queryType, String queryString) throws InvalidRequestException, JAXRException

// Executes a query as specified by query parameter
BulkResponse executeQuery(Query query) throws JAXRException
```

Query Management APIs:

- Interface `QueryManager`
  - supertype of `BusinessQueryManager` and `DeclarativeQueryManager`
- Interface `BusinessQueryManager`
  - mandatory
  - Capability Level 0
  - provides a simple business-level API
  - to query for the most important high-level interfaces in the information model
- Interface `DeclarativeQueryManager`
  - optional
  - Capability Level 1
  - provides a more flexible generic API
  - to perform ad hoc queries using a declarative query language like SQL

Sources:

Interface `javax.xml.registry.QueryManager` JavaDoc - [`http://download.oracle.com/javaee/5/api/javax/xml/registry/QueryManager.html`]

Interface `javax.xml.registry.DeclarativeQueryManager` JavaDoc - [`http://download.oracle.com/javaee/5/api/javax/xml/registry/DeclarativeQueryManager.html`]

The Java Web Services Tutorial [JAXR Architecture] - [`http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/JAXR-ebXML2.html`]

6.2. Create JAXR client to connect to a UDDI business registry, execute queries to locate services that meet specific requirements, and publish or update information about a business service.

Question A060201

You are testing JAXR client program which publishes service binding with bogus URL to local UDDI registry:

```
1
2 // Create, add, and configure a new ServiceBinding
3 ServiceBinding binding = lifeCycleMngr.createServiceBinding();
4 service.addServiceBinding(binding);
5 binding.setAccessURI("http://localhost/index.html");
6 ...
7 // Save
8
```

During saving, JAXR exception is thrown, indicating that access URL is invalid. How can you avoid this problem to test your JAXR client program locally?

Options (select 1):

1. It is not possible, all published URLs are verified by UDDI registry and must be valid.
2. Add the following code before line #5:

   ```
   binding.setValidateURI(false);
   ```

3. Add the following code before line #5:

   ```
   binding.setRedirectURI(true);
   ```

4. Add the following code before line #5:

   ```
   binding.setIgnoreErrors(true);
   ```

Answer:

Correct option is 2.

As shown below in the code snippet, the `LifeCycleManager` defines one method for creating a `ServiceBinding`, `createServiceBinding()`:

```
package javax.xml.registry;
import javax.xml.registry.infomodel.*;

public Interface LifeCycleManager {
```

```
    ...
    public ServiceBinding createServiceBinding() throws JAXRException
}
```

The `ServiceBinding` object does not have a name, but it will almost always have an `accessURI`, which is the electronic address of the service. The following code snippet shows a `ServiceBinding` being created, added to a service, and then configured with an access URI:

```
// Create, add, and configure a new ServiceBinding
ServiceBinding binding = lifeCycleMngr.createServiceBinding();
binding.setDescription(lifeCycleMngr.createInternationalString("Test Service Binding"));

service.addServiceBinding(binding);

binding.setValidateURI(false);
binding.setAccessURI("http://localhost/index.html");
```

In this case the access URI is the URL of static HTML Web page. If you were defining a Web Service, it would be the URL of the Web Service.

When a `ServiceBinding` is saved in the UDDI registry, the registry will verify that the `accessURI` is a valid URL - assuming it is a URL. Because you are making up an organization for this example, the method call `setValidateURI(false)` overrides this behavior. If you comment that line out of your code, and the URL you specified does not exist, the program will throw an exception indicating that the URL is invalid. This is an important added value of JAXR. Invalid URLs represent a significant problem in UDDI, and JAXR's ability to detect them on the client side, before they are saved in a UDDI registry, is important to ensuring the validity of data published to UDDI registries.

Sources:

Overview of JAXR - [`http://download.oracle.com/javaee/1.4/tutorial/doc/JAXR2.html`]

`URIValidator` Interface JavaDoc - [`http://download.oracle.com/javaee/6/api/javax/xml/registry/infomodel/URIValidator.html`]

Discover and publish Web services with JAXR - [`http://www.javaworld.com/javaworld/jw-06-2002/jw-0614-jaxr.html`]

Question A060202

You need to get all access URIs for Web Services which belong to your organization. How can you do this ?

Options (select 1):

1. Use `BusinessQueryManager.findOrganizations(...)` to find your organization in UDDI, `Organization.getServices()` to get all Web Services which belong to organization, `Service.getServiceBindings()` to get all Web Services bindings and get access URI from each `ServiceBinding`.

2. Use `BusinessQueryManager.findOrganizations(...)` to find your organization in UDDI, `BusinessQueryManager.getServices(organization)` to get all Web Services which belong to organization, `BusinessQueryManager.getServiceBindings(service)` to get all Web Service bindings and get access URI from each `ServiceBinding`.

3. Use `BusinessLifeCycleManager.findOrganizations(...)` to find your organization in UDDI, `BusinessLifeCycleManager.getServices(organization)` to get all Web Services which belong to organization, `BusinessLifeCycleManager.getServiceBindings(service)` to get all Web Service bindings and get access URI from each `ServiceBinding`.

4. Use `RegistryService.findOrganizations(...)` to find your organization in UDDI,

`Organization.getServices()` to get all Web Services which belong to organization, `Service.getServiceBindings()` to get all Web Services bindings and get access URI from each `ServiceBinding`.

Answer:

Correct option is 1.

The code example which demonstrates how to find access URIs for services from some organization is shown below:

```
// Get manager capabilities from RegistryService
BusinessQueryManager bqm = rs.getBusinessQueryManager();

// Define name patterns. Organization name contains qString.
Collection namePatterns = new ArrayList();
namePatterns.add("%" + qString + "%");

// Find organizations using the name
BulkResponse response = bqm.findOrganizations(null, namePatterns, null, null, null, null);
Collection orgs = response.getCollection();

Iterator orgIter = orgs.iterator();
while (orgIter.hasNext()) {
        Organization org = (Organization) orgIter.next();

        // Get a collection of Services from an Organization
        Collection services = org.getServices();

        // Iterate through the collection to get an individual Service
        Iterator svcIter = services.iterator();
        while (svcIter.hasNext()) {
                Service svc = (Service) svcIter.next();

                // Get a collection of ServiceBindings from a Service
                Collection serviceBindings = svc.getServiceBindings();

                // Iterate through the collection to get an individual ServiceBinding
                Iterator sbIter = serviceBindings.iterator();
                while (sbIter.hasNext()) {
                        ServiceBinding sb = (ServiceBinding) sbIter.next();

                        // Get URI of the service.  You can access the service through this URI.
                        String accessURI = sb.getAccessURI();
                        System.out.println("Access the service " + svc.getName().getValue() + " at
                }
        }
}
```

Option 2 is wrong because there are no such methods in `BusinessQueryManager`.

Option 3 is wrong because `BusinessLifeCycleManager` purpose is to create and modify data structures in UDDI registry.

Option 4 is wrong because `RegistryService` is access point to registry and it provides manager objects: `BusinessQueryManager` (JAXR query API) and `BusinessLifeCycleManager` (JAXR creation and modification API).

Sources:

The Java Web Services Tutorial [Querying a Registry] - [`http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/JAXR-ebXML4.html`]

Interface `RegistryService` JavaDoc - `http://download.oracle.com/javaee/5/api/javax/xml/registry/RegistryService.html`

Question 060203

Until recent time your organization sold computer parts, now it start selling digital cameras. How can you update classification in JAXR registry to draw customers who are interested in digital cameras?

```
BusinessQueryManager     bqm = ... // get BusinessQueryManager
BusinessLifeCycleManager lcm = ... // get BusinessLifeCycleManager
Organization             org = ... // get your organization from registry
```

Options (select 1):

1.
```
ClassificationScheme scheme = new ClassificationScheme("ntis-gov:naics");
Classification classification = new Classification(scheme, "Digital Cameras", "123456");
Collection classifications = new ArrayList();
classifications.add(classification);
org.addClassifications(classifications);
Collection orgs = new ArrayList();
orgs.add(org);
lcm.saveOrganizations(orgs);
```

2.
```
ClassificationScheme scheme = bqm.findClassificationSchemeByName("ntis-gov:naics");
Classification classification = lcm.createClassification(scheme, "Digital Cameras", "123456")
Collection classifications = new ArrayList();
classifications.add(classification);
org.addClassifications(classifications);
Collection orgs = new ArrayList();
orgs.add(org);
lcm.saveOrganizations(orgs);
```

3.
```
Classification classification = lcm.createClassification("Digital Cameras", "123456");
Collection classifications = new ArrayList();
classifications.add(classification);
org.addClassifications(classifications);
Collection orgs = new ArrayList();
orgs.add(org);
lcm.saveClassifications(classification);
lcm.saveOrganizations(orgs);
```

4.
```
ClassificationScheme scheme = bqm.findClassificationSchemeByName("ntis-gov:naics");
Classification classification = lcm.createClassification(scheme, "Digital Cameras", "123456")
Collection classifications = new ArrayList();
classifications.add(classification);
org.addClassifications(classifications);
Collection orgs = new ArrayList();
orgs.add(org);
lcm.saveClassifications(classification);
lcm.saveOrganizations(orgs);
```

Answer:

Correct option is 2.

An organization that uses the Java platform for its electronic business would use JAXR to register itself or update information in a standard registry (UDDI, ebXML).

Organzation has some classification, and classification should be withing some schema. For example, you can not say how big is number "121" until you know what notation it is: decimal notation, binary notation or hexadecimal notation. If you know, for example, that this number belongs to binary notation, you can say that it is quite small number, and if you know that it belongs to hexadecimal notation - this means the number is much bigger. So, to add new classification, you need classification schema. This makes option 3 wrong.

As any JAXR object, you are using `BusinessLifeCycleManager` to create (save and delete) new objects, you can not use simple Java constructors. This makes option 1 wrong.

JAXR supports cascade saving of objects to registry. Once you call "`saveOrganizations`" method of `BusinessLifeCycleManager`, it will manage the life cycle of the `Organization` objects contained in "`orgs`" collection. Also, there is no method `BusinessLifeCycleManager.saveClassifications`. This makes option 4 wrong.

Sources:

Interface `BusinessLifeCycleManager` API JavaDoc - [`http://download.oracle.com/javaee/5/api/javax/xml/registry/BusinessLifeCycleManager.html`]

Interface `Classification` API JavaDoc - [`http://download.oracle.com/javaee/5/api/javax/xml/registry/infomodel/Classification.html`]

Java Web Services Tutorial [Publishing Objects to the Registry] - [`http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/JAXR-ebXML5.html`]

Question A060204

You are using JAXR registry to search for other businesses which sell digital cameras. In order to include more organizations in results you want to define in search criteria just name pattern, not exact organization name. Also, results should be sorted on the name field in ascending alphabetic sort order. How can you do this using JAXR API?

```
BusinessQueryManager bqm = ... // get BusinessQueryManager
```

Options (select 1):

1.
```
Collection<String> qualifiers = new ArrayList<String>();
qualifiers.add(FindQualifier.SORT_BY_NAME_ASC);
Collection<String> patterns = new ArrayList<String>();
patterns.add("*Digital Cameras*");
BulkResponse response = bqm.findOrganizations(qualifiers, patterns, null, null, null, null);
Collection<Organization> orgs = response.getCollection();
```

2.
```
Collection<String> qualifiers = new ArrayList<String>();
qualifiers.add(FindQualifier.SORT_BY_NAME_ASC);
Collection<String> patterns = new ArrayList<String>();
patterns.add("%Digital Cameras%");
BulkResponse response = bqm.findOrganizations(qualifiers, patterns, null, null, null, null);
Collection<Organization> orgs = response.getCollection();
```

3.
```
Collection<String> qualifiers = new ArrayList<String>();
qualifiers.add(FindQualifier.SORT_BY_NAME_ASC);
Collection<String> patterns = new ArrayList<String>();
patterns.add("*Digital Cameras*");
Collection<Organization> orgs = bqm.findOrganizations(qualifiers, patterns, null, null, null,
```

4.
```
Collection<String> qualifiers = new ArrayList<String>();
qualifiers.add(FindQualifier.SORT_BY_NAME_ASC);
Collection<String> patterns = new ArrayList<String>();
```

```
patterns.add("%Digital Cameras%");
Collection<Organization> orgs = bqm.findOrganizations(qualifiers, patterns, null, null, null,
```

5. It is not possible. You can not search with name patterns.
6. It is not possible. You can not sort the search results.

Answer:

Correct option is 2.

Before query manager can invoke the method `findOrganizations`, the code needs to define the search criteria to be used. In our case, two of the possible six search parameters are supplied to `findOrganizations(...)`; because `null` is supplied for the third, fourth, fifth, and sixth parameters, those criteria are not used to limit the search. The first and second arguments are all `Collection<String>` objects, with find qualifiers and name patterns being defined here.

The only element in find qualifiers is a `String` specifying search results sort order. Another parameter is name patterns parameter. This parameter, which is also a `Collection<String>` object with only one element, says that businesses with "Digital Cameras" in their names are a match.

`BusinessQueryManager.findOrganizations` returns `BulkResponse`, not `Collection`. A returning `BulkResponse` contains `Collection` of `Organization` objects.

You can define name patterns in search criteria (which specify the strings to be searched): `%name-fragment%`.

Sources:

Interface `BusinessQueryManager` API JavaDoc - [`http://download.oracle.com/javaee/5/api/javax/xml/registry/BusinessQueryManager.html`]

Question A060205

You are creating e-Business application which accesses third party Web Services. Unfortunately, after testing, you have discovered a problem with the code which responsible for Web Services lookup: you need to find all Web Services provided by "Amazon Books Co" and "Amazon Gadgets Co" companies, but you do not want search results include Web Services from "AMAZON SPRINGS WATER Co". How can you resolve this problem?

```
BusinessQueryManager bqm = ... // get BusinessQueryManager
```

Options (select 1):

1.
```
Collection<String> patterns = new ArrayList<String>();
patterns.add("Amazon");
BulkResponse response = bqm.findOrganizations(null, patterns, null, null, null, null);
Collection<Organization> orgs = response.getCollection();
```

2.
```
Collection<String> qualifiers = new ArrayList<String>();
qualifiers.add(FindQualifier.DISABLE_INSENSITIVE_MATCH);
Collection<String> patterns = new ArrayList<String>();
patterns.add("Amazon");
BulkResponse response = bqm.findOrganizations(qualifiers, patterns, null, null, null, null);
Collection<Organization> orgs = response.getCollection();
```

3.
```
Collection<String> qualifiers = new ArrayList<String>();
qualifiers.add(FindQualifier.CASE_SENSITIVE_MATCH);
Collection<String> patterns = new ArrayList<String>();
patterns.add("Amazon");
BulkResponse response = bqm.findOrganizations(qualifiers, patterns, null, null, null, null);
Collection<Organization> orgs = response.getCollection();
```

4. None of these.

Answer:

Correct option is 3.

By default the following methods use case-insensitive behavior of a name match:

```
findObjects()
findOrganizations()
findServices()
findConcepts()
findClassificationScheme()
```

This makes option 1 wrong.

The `FindQualifier.CASE_SENSITIVE_MATCH` signifies that the default case-insensitive behavior of a name match should be overridden. When this behavior is specified, case is relevant in the search results and only entries that match the case of the value passed in the name argument will be returned.

Option 3 is wrong, because `FindQualifier.DISABLE_INSENSITIVE_MATCH` constant does not exist.

Sources:

The Java Web Services Tutorial [Querying a Registry] - [`http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/JAXR-ebXML4.html`]

Interface `FindQualifier` API JavaDoc - [`http://download.oracle.com/javaee/5/api/javax/xml/registry/FindQualifier.html`]

Appendix 7. J2EE Web Services

7.1. Identify the characteristics of and the services and APIs included in the Java EE platform.

Question A070101

Which Java APIs are used for XML parsing?

Options (select 2):

1. SAAJ
2. StAX
3. JAX-RPC
4. SAX
5. JAXR

Answer:

Correct options are 2 and 4.

The JAXP API is a part of Java EE 6.0 and Java SE 6.0.

JAXP includes the industry standard SAX and DOM APIs, as well as a pluggability API that allows SAX and DOM parsers and XSLT transform engines to be plugged into the framework, and allows applications to find parsers that support the features needed by the application.

JSE 6.0 must meet the JAXP conformance requirements and must provide at least one SAX 2 parser, at least one DOM 2 (DOM 3) parser, and at least one XSLT transform engine. There must be a SAX parser or parsers that support all combinations of validation modes and namespace support. There must be a DOM parser or parsers that support all combinations of validation modes and namespace support. All SAX and DOM parsers must support validation using either DTDs or XML Schemas, as described in the JAXP 1.4 specification.

The Streaming API for XML (StAX) specification defines a pull-parsing API for XML. The streaming API gives parsing control to the programmer by exposing a simple iterator based API. This allows the programmer to ask for the next event (pull the event) and allows state to be stored in a procedural fashion. All Java EE application client containers, web containers, and EJB containers are required to support the StAX API.

Sources:

Java EE 6.0 Specification - [`http://jcp.org/en/jsr/detail?id=316`]

JAXP Specification - [`http://jaxp.java.net/`]

Question A070102

Which JEE 6.0 API supports asynchronous messaging between JEE components?

Options (select 1):

1. JMX
2. JMS
3. JAXP
4. JAXR
5. StAX

Answer:

Correct option is 2.

The Java Message Service (JMS) is a Java API that allows applications to create, send, receive, and read messages. Designed by Sun and several partner companies, the JMS API defines a common set of interfaces and associated semantics that allow programs written in the Java programming language to communicate with other messaging implementations.

The JMS API minimizes the set of concepts a programmer must learn to use messaging products but provides enough features to support sophisticated messaging applications. It also strives to maximize the portability of JMS applications across JMS providers in the same messaging domain.

The JMS API enables communication that is not only loosely coupled but also:

- Asynchronous. A JMS provider can deliver messages to a client as they arrive; a client does not have to request messages in order to receive them.
- Reliable. The JMS API can ensure that a message is delivered once and only once. Lower levels of reliability are available for applications that can afford to miss messages or to receive duplicate messages.

A Java Message Service provider must be included in a Java EE product. The JMS implementation must provide support for both JMS point-to-point and publish/subscribe messaging, and thus must make those facilities available using the `ConnectionFactory` and `Destination` APIs.

Option 1 is wrong. The Java Management Extensions (JMX) technology is an open technology for management and monitoring that can be deployed wherever management and monitoring are needed. By design, this standard is suitable for adapting legacy systems, implementing new management and monitoring solutions and plugging into those of the future.

Option 3 is wrong. The Java API for XML Processing (JAXP) enables applications to parse and transform XML documents using an API that is independent of a particular XML processor implementation. JAXP also provides a pluggability feature which enables applications to easily switch between particular XML processor implementations.

Option 4 is wrong. The Java API for XML Registries (JAXR) gives you a uniform way to use business registries that are based on open standards (such as ebXML) or industry consortium-led specifications (such as UDDI).

Option 5 is wrong. The Streaming API for XML (StAX) specification defines a pull-parsing API for XML. The streaming API gives parsing control to the programmer by exposing a simple iterator based API. This allows the programmer to ask for the next event (pull the event) and allows state to be stored in a procedural fashion.

Sources:

JMS Specification - [`http://www.oracle.com/technetwork/java/jms/index.html`]

JAXP FAQ - [`http://jaxp.java.net/1.4/JAXP-FAQ.html`]

Java EE 6.0 Specification - [`http://jcp.org/en/jsr/detail?id=316`]

7.2. Explain the benefits of using the Java EE platform for creating and deploying Web Service applications.

blah

7.3. Describe the functions and capabilities of the JAXP, DOM, SAX, StAX, JAXR, JAXB, JAX-WS and SAAJ in the Java EE platform.

Question A070301

What is the purpose of JAXP API?

Options (select 1):

1. It enables Java software programmers to use a single, easy-to-use abstraction API to access a variety of XML registries.
2. It enables simplified XML processing using Java classes that are generated from XML schemas.
3. It enables applications to parse and transform XML documents independent of a particular XML processing implementation.
4. It enables Java clients to make remote procedure calls via XML and SOAP over HTTP.

Answer:

Correct option is 3.

The Java API for XML Processing (JAXP) enables applications to parse and transform XML documents independent of a particular XML processing implementation.

The Java API for XML Processing (JAXP), part of the Java SE platform, supports the processing of XML documents using Document Object Model (DOM), Simple API for XML (SAX), and Extensible

Stylesheet Language Transformations (XSLT). JAXP enables applications to parse and transform XML documents independent of a particular XML processing implementation.

JAXP includes the industry standard SAX and DOM APIs, as well as a pluggability API that allows SAX and DOM parsers and XSLT transform engines to be plugged into the framework, and allows applications to find parsers that support the features needed by the application.

All JSE 5.0 products must meet the JAXP conformance requirements and must provide at least one SAX 2 parser, at least one DOM 2 (DOM 3) parser, and at least one XSLT transform engine. There must be a SAX parser or parsers that support all combinations of validation modes and namespace support. There must be a DOM parser or parsers that support all combinations of validation modes and namespace support. All SAX and DOM parsers must support validation using either DTDs or XML Schemas, as described in the JAXP 1.4 specification.

Option 1 is wrong because Java API for XML Registries (JAXR) provides a uniform and standard Java API for accessing different kinds of XML Registries. XML registries are an enabling infrastructure for building, deployment, and discovery of Web Services.

Option 2 is wrong because Java API for XML Data Binding (JAXB) enables simplified XML processing using Java classes that are generated from XML schemas.

Option 4 is wrong because Java API for XML-based RPC (JAX-RPC) and JAX-WS use the SOAP standard and HTTP so client programs can make XML-based remote procedure calls (RPCs) over the Internet. JAX-RPC and JAX-WS rely on the HTTP transport protocol.

JAX-WS stands for Java API for XML Web Services. JAX-WS is a technology for building web services and clients that communicate using XML. JAX-WS allows developers to write message-oriented as well as RPC-oriented web services.

In JAX-WS, a web service operation invocation is represented by an XML-based protocol such as SOAP. The SOAP specification defines the envelope structure, encoding rules, and conventions for representing web service invocations and responses. These calls and responses are transmitted as SOAP messages (XML files) over HTTP.

Sources:

JAXP FAQ - [`http://jaxp.java.net/1.4/JAXP-FAQ.html`]

Java EE 6.0 Specification - [`http://jcp.org/en/jsr/detail?id=316`]

The Java EE 6 Tutorial - [`http://download.oracle.com/javaee/6/tutorial/doc/index.html`]

Question A070302

You need to call a business method on Web Service which is already deployed in .NET environment. Which technology should you use?

Options (select 1):

1. DCOM
2. CORBA
3. JAX-WS
4. DOM
5. RMI

Answer:

Correct option is 3.

Web Services clients are using SOAP protocol to call methods on Web Services.

JAX-WS provides run-time services for marshalling and demarshalling Java data and objects to and from XML SOAP messages.

A service client uses a JAX-WS service by invoking remote methods on a service endpoint. Note that a JAX-WS service client can call a service endpoint that has been defined and deployed on a non-Java platform. The converse is also true.

Sources:

JAX-WS API - [`http://jax-ws.java.net/`]

Web Services Interoperability Technologies - [`http://wsit.java.net/`]

Question A070303

Which of the following APIs can be used in the application which requires vendor-independent XML parsing?

Options (select 1):

1. JAXB
2. JAXP
3. JAXR
4. XML4J
5. DOM4J
6. JAX-WS

Answer:

Correct option is 2.

The Java API for XML Processing (JAXP) enables applications to parse and transform XML documents using an API that is independent of a particular XML processor implementation. JAXP also provides a pluggability feature which enables applications to easily switch between particular XML processor implementations.

To achieve the goal of XML processor independence, an application should limit itself to the JAXP API and avoid implementation-dependent APIs and behavior. JAXP includes industry standard APIs such as DOM and SAX.

The reason for the existance of JAXP is to facilitate the use of XML on the Java platform.

Sources:

Java API for XML Processing (JAXP) FAQ - [`http://jaxp.java.net/1.4/JAXP-FAQ.html`]

Question A070304

Which SAAJ type is used to present SOAP message headers block?

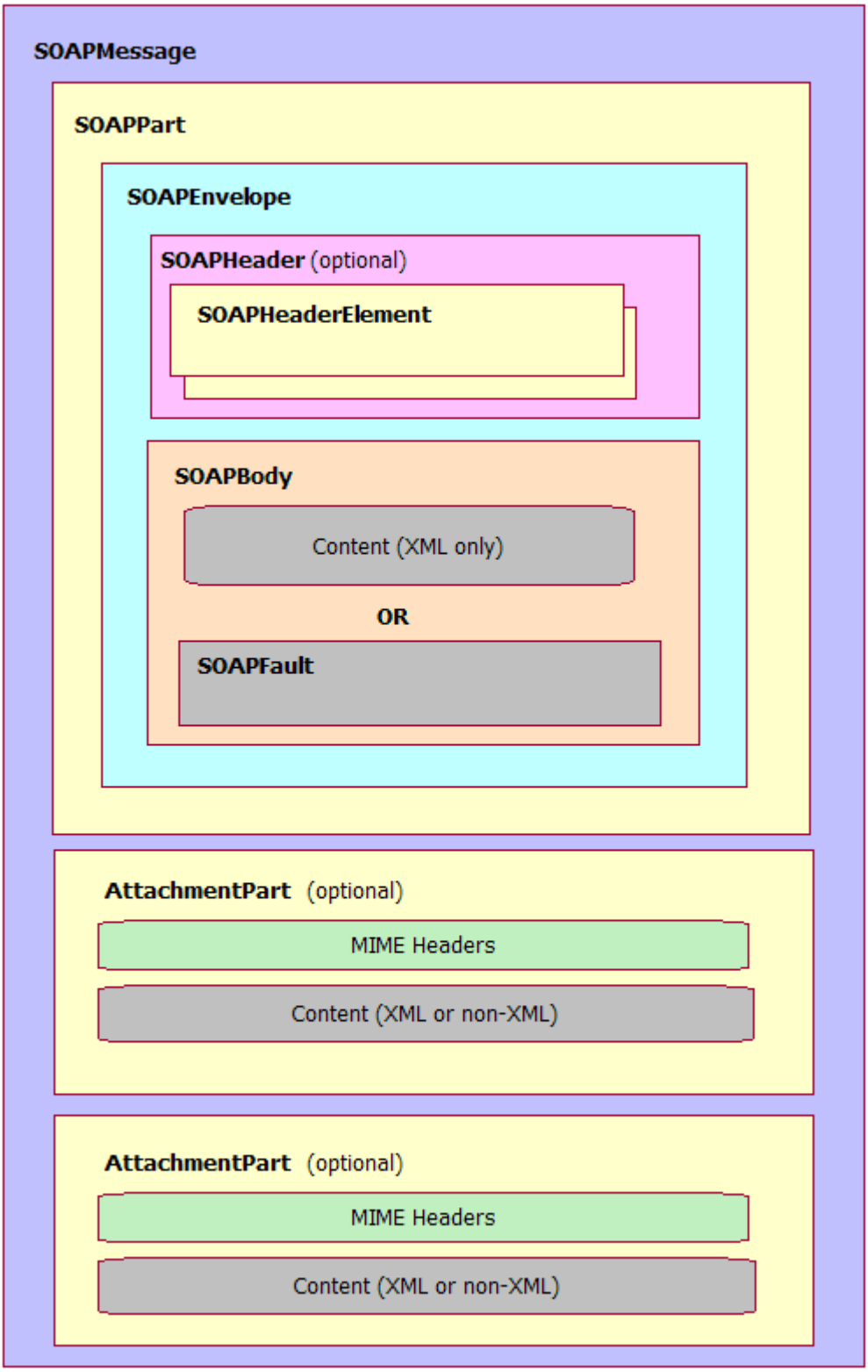
Options (select 1):

1. `javax.xml.soap.SOAPPart`
2. `javax.xml.soap.AttachmentPart`
3. `javax.xml.soap.DetailEntry`
4. `javax.xml.soap.SOAPHeaderElement`

Answer:

Correct option is 1.

SOAP headers block is presented by `javax.xml.soap.SOAPPart` type. The `SOAPPart` object is containing the `SOAPEnvelope` object. The `SOAPEnvelope` object must contain a single `SOAPBody` object and may contain a `SOAPHeader` object.



Option 2 is wrong because `javax.xml.soap.AttachmentPart` contains application-specific content and corresponding MIME headers.

Option 3 is wrong because `javax.xml.soap.DetailEntry` carries the content for a `Detail` object, giving details for a `SOAPFault` object.

Option 4 is wrong because `javax.xml.soap.SOAPHeaderElement` models the content of only a single SOAP header of a SOAP envelope.

Sources:

SOAP with Attachments API for Java (SAAJ) Tutorial - [`http://download.oracle.com/javaee/5/tutorial/doc/bnbhr.html`]

Overview of SAAJ - [`http://download.oracle.com/javaee/5/tutorial/doc/bnbhg.html`]

Question A070305

For a `javax.xml.soap.SOAPMessage` object, what can be in format other than XML?

Options (select 1):

1. `javax.xml.soap.SOAPPart`
2. `javax.xml.soap.AttachmentPart`
3. `javax.xml.soap.SOAPBody`
4. `javax.xml.soap.SOAPHeader`

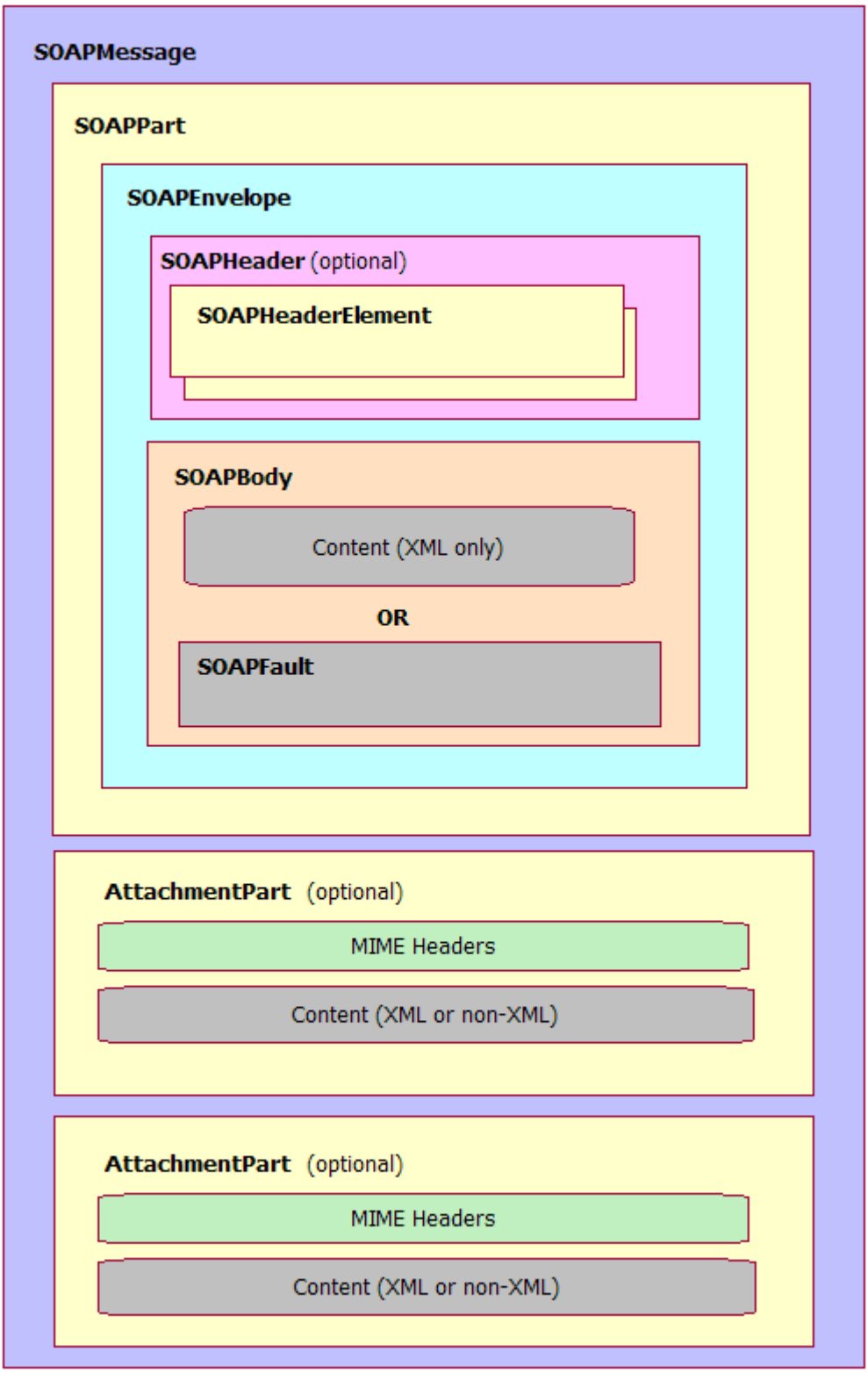
Answer:

Correct option is 2.

An `javax.xml.soap.AttachmentPart` object can contain any type of content, including XML. And because the SOAP part can contain only XML content, you must use an `javax.xml.soap.AttachmentPart` object for any content that is not in XML format.

For example, code below demonstrates creating an attachment with image:

```
URL url = new URL("http://sample.com/image.jpg");
DataHandler dataHandler = new DataHandler(url);
AttachmentPart attachment = message.createAttachmentPart(dataHandler);
attachment.setContentId("attached_image");
message.addAttachmentPart(attachment);
```



Option 1 is wrong because `javax.xml.soap.SOAPPart` must be in XML format.

Option 3 is wrong because `javax.xml.soap.SOAPBody` must be in XML format.

Option 4 is wrong because `javax.xml.soap.SOAPHeader` must be in XML format.

Sources:

SOAP with Attachments API for Java (SAAJ) Tutorial - [`http://download.oracle.com/javaee`

/5/tutorial/doc/bnbhr.html]

Question A070306

Which SAAJ code snippets can be used to add the following XML fragment to SOAP message:

```
<name>Sony Reader Digital Book PRS-505</name>
```

(Assume that all variables are correctly initialized).

Options (select 2):

1.
```
Name name = soapFactory.createName("name");
SOAPElement element = orderElement.addChildElement(name);
element.addTextNode("Sony Reader Digital Book PRS-505");
```

2.
```
Name name = new Name("name");
SOAPElement element = orderElement.addChildElement(name);
element.addTextNode("Sony Reader Digital Book PRS-505");
```

3.
```
QName name = new QName("name");
SOAPElement element = orderElement.addChildElement(name);
element.addTextNode("Sony Reader Digital Book PRS-505");
```

4.
```
QName name = soapFactory.createName("name");
SOAPElement element = orderElement.addChildElement(name);
element.addTextNode("Sony Reader Digital Book PRS-505");
```

Answer:

Correct options are 1 and 3.

When you create any new element, you also need to create an associated `javax.xml.namespace.QName` or `javax.xml.soap.Name` object so that it is uniquely identified.

Option 1 is correct. You can use `Name` objects instead of `QName` objects. `Name` objects are specific to the SAAJ API, and you create them using either `SOAPEnvelope` methods or `SOAPFactory` methods. However, the `Name` interface may be deprecated at a future release.

The `SOAPFactory` class also lets you create XML elements when you are not creating an entire message or do not have access to a complete `SOAPMessage` object. For example, when you work with XML fragments rather than complete `SOAPMessage` objects. Consequently, you do not have access to a `SOAPEnvelope` object, and this makes using a `SOAPFactory` object to create `Name` objects very useful.

The following SAAJ code snippet:

```
SOAPBody body = envelope.getBody();
Name bodyName = soapFactory.createName("purchase", "p", "http://sony.com");
SOAPBodyElement purchase = body.addBodyElement(bodyName);
Name childName = soapFactory.createName("order");
SOAPElement order = purchase.addChildElement(childName);
childName = soapFactory.createName("product");
SOAPElement product = order.addChildElement(childName);
product.addTextNode("Sony Reader Digital Book PRS-505");
childName = soapFactory.createName("price");
SOAPElement price = order.addChildElement(childName);
```

```
price.addTextNode("260.99");
```

will result in the following XML snippet:

```
<p:purchase xmlns:p="http://sony.com">
    <order>
        <product>Sony Reader Digital Book PRS-505</product>
        <price>260.99</price>
    </order>
</p:purchase>
```

NOTE: In SAAJ 1.1 you can only use `Name` interface to indetify elements.

Option 2 is wrong. The `javax.xml.soap.Name` is an interface and cannot be instantiated with constructor. You should use either `SOAPEnvelope.createName(...)` or `SOAPFactory.createName(...)` methods.

Option 3 is correct. `QName` represents a qualified name as defined in the XML specifications: XML Schema Part2: Datatypes specification, Namespaces in XML, Namespaces in XML Errata.

When you create any new element, you also need to create an associated `javax.xml.namespace.QName` object so that it is uniquely identified.

Note: `QName` objects associated with `SOAPBodyElement` or `SOAPHeaderElement` objects must be fully qualified; that is, they must be created with a namespace URI, a local part, and a namespace prefix. Specifying a namespace for an element makes clear which one is meant if more than one element has the same local name.

Option 4 is wrong. The `QName` is a concrete class and must be instantiated with one of its public constructors:

```
QName(String localPart)
QName(String namespaceURI, String localPart)
QName(String namespaceURI, String localPart, String prefix)
```

Both `SOAPEnvelope.createName(...)` and `SOAPFactory.createName(...)` return `Name` instance, not `QName`.

Note: `QName` class was added in SAAJ 1.3.

Sources:

SAAJ Tutorial - [`http://download.oracle.com/javaee/5/tutorial/doc/bnbhr.html`]

`SOAPFactory` JavaDoc API - [`http://download.oracle.com/javase/6/docs/api/javax/xml/soap/SOAPFactory.html`]

`SOAPElement` JavaDoc API (SAAJ 1.3) - [`http://download.oracle.com/javase/6/docs/api/javax/xml/soap/SOAPElement.html`]

`QName` JavaDoc API - [`http://download.oracle.com/javase/1.5.0/docs/api/javax/xml/namespace/QName.html`]

7.4. Describe the role of the WS-I Basic Profile when designing Java EE Web Services.

Question A070401

Web Services must be WS-I Basic Profile compliant. Is this statement true or false?

Options (select 1):

1. The statement is true only for Java EE 6.0 Web Services.
2. The statement is true for all Web Services.
3. The statement is true only for .NET Web Services.
4. The statement is false.

Answer:

Correct option is 4.

Web Service does not have to be Basic Profile 1.1 conformant. You should support Basic Profile recommendations only if you need to develop interoperable across several platforms (Windows, UNIX, Linux) and languages (Java, C#, Perl) Web Service.

The Basic Profile 1.1 (BP 1.1) consists of implementation guidelines recommending how a set of core Web Services specifications should be used together to develop interoperable Web Services.

BP 1.1 will significantly simplify the task of implementing interoperable Web services solutions within and across enterprise boundaries, by allowing companies to focus resources on delivering new products and services to their customers, instead of dealing with basic plumbing. The greater confidence in Web services interoperability that BP 1.1 brings will reduce the risk of adopting Web services technologies or beginning Web services projects. BP 1.1 will also simplify purchasing decisions for customers concerned about interoperability with partners, customers, and suppliers. BP 1.1 delivers on the key promise of WS-I, to promote consistent and reliable interoperability among Web services across platforms, applications, and programming languages.

Sources:

WS-I Basic Profile 1.1 - [`http://www.ws-i.org/Profiles/BasicProfile-1.1.html`]

WS-I Basic Profile 1.0 FAQ - [`http://www.ws-i.org/Profiles/Basic/2003-08/BasicProfile-1.0faq.pdf`]

Appendix 8. Security

8.1. Explain basic security mechanisms including: transport level security, such as basic and mutual authentication and SSL, message level security, XML encryption, XML Digital Signature, and federated identity and trust.

Question A080101

What is true about XML Digital Signature?

Options (select 3):

1. Enveloping signature embeds the signed content into a signature container, the document is enclosed.
2. Detached signature is a signature which leaves the signed document in it's original state and provide the signature as additional data. Detached signature can be in separate XML file or reside within the same XML document but is sibling element of data object.
3. Enveloping signature is embedded in XML document that contains the signature as an element.
4. Detached signature is a signature which leaves the signed document in it's original state and provide the signature as additional data. Detached signature can be in separate XML file only, it can not reside with data object within the same XML document.

5. Enveloped signature is embedded in XML document that contains the signature as an element.
6. Enveloped signature embeds the signed content into a signature container, the document is enclosed.

Answer:

Correct options are 1,2, and 5.

Enveloping Signature

Enveloping signatures embed the signed content into a signature container, the document is enclosed. The main document is the signature itself, which contains the signed document as integral part of the signature. The main problem of this approach is that the signed application data has to be extracted from the container before it can be processed by the application. To integrate enveloping signatures into an automated business process of signing B2B messages at the sender's side and verifying the messages on the receiver's side before processing the message, a usual filter has to be set up. The problem is that after verification of the signature and extracting the document, the signature is no longer available to the document processing application. So the system has to keep track of the document and the signature in two different application domains. This makes data handling complicated.

An enveloping signature is useful when the signing facility wants to add its own metadata (such as a timestamp) to a signature - it doesn't have to modify the source document, but can include additional data covered by the signature within the signature document it generates. (An XML Digital Signature can sign multiple objects at once, so enveloping is usually combined with another format).

The signature is over content found within an `Object` element of the signature itself. The `Object` (or its content) is identified via a `Reference` (via a URI fragment identifier or transform).

`Signature` envelopes the contents to be signed:

```
<Signature>
        ...
        <Reference URI="#myRefObjectID">
                ...
        </Reference>

        <Object Id="myRefObjectID">
                <doc>
                        <myElement>
                                ...
                        </myElement>
                        ...
                </doc>
        </Object>
</Signature>
```

Detached Signature

Detached signatures are signatures which leave the signed document in it's original state and provide the signature as additional data. The two entities, document and signature, have to be transferred together. In terms of files, the signature is stored in a separate file. Applications can read the document as before the signature generation. The use of a separate object for signature storage makes it even more complicated to integrate the system into an IT system because of the need to transfer two objects between both parties.

A detached signature is useful when you can't modify the source; the downside is that it requires two XML documents - the source and its signature - to be carried together. In other words, it requires a packaging format - enter SOAP headers.

The signature is over content external to the `Signature` element, and can be identified via a URI or

transform. Consequently, the signature is "detached" from the content it signs. This definition typically applies to separate data objects, but it also includes the instance where the `Signature` and data object reside within the same XML document but are SIBLING elements.

`Signature` is external to the content that is signed:

```
<Signature>
        ...
        <Reference URI="http://www.buy.com/books/purchaseWS"/>
        ...
</Signature>
```

Enveloped Signature

Enveloped XML Signature offers a third choice of signature mechanism, which is closely related to human hand written signatures: When a hand written signature is applied to a document, the document itself remains readable and usable, but the signature is visible (embedded) on the document. XML allows to produce enclosed signatures, where the digital signature is enclosed in the document. The advantage of this solution is that the document contains the signature, remains in it's native data format and can be processed without further attention to the signature. This enables IT systems to have a continuous workflow with additional security. An eCommerce application can be enhanced by using digital signatures without disturbing normal business operations. Even more important, since signed XML objects are in their native format, they can be imported into enterprise databases and business processes without loosing their digital signature. This continuous binding between signed data and signature ensures that the signature can be verified at every time after receiving XML data. In case of a dispute between business partners, the signatures are available to solve the problems.

An enveloped signature is useful when you have a simple XML document which you to guarantee the integrity of. For example, XKMS messages can use enveloped signatures to convey "trustable" answers from a server back to a client.

The signature is over the XML content that contains the signature as an element. The content provides the root XML document element. Obviously, enveloped signatures must take care not to include their own value in the calculation of the `SignatureValue`.

`Signature` is enveloped within the content been signed:

```
<doc Id="myID">
        <myElement>
                ...
        </myElement>
        <Signature>
                ...
                <Reference URI="#myID"/>
                ...
        </Signature>
</doc>
```

Sources:

XML-Signature Syntax and Processing - [`http://www.w3.org/TR/xmldsig-core/`]

Question A080102

What authentication forms are available for Java EE Web Services?

Options (select 2):

1. Basic Authentication
2. Digest Authentication
3. Form Based Authentication
4. HTTPS Based Authentication
5. Kerberos Authentication

Answer:

Correct options are 1 and 4.

Two forms of authentication are available for use within Web Services for Java EE based on existing Java EE functionality. These are **Basic Authentication** and **HTTPS Based Authentication (Symmetric HTTP)**, which are defined by the Servlet specification.

Using the authentication models above, the container can also perform a credential mapping of incoming credentials at any point along the execution path. The mapping converts the external user credentials into a credential used within a specific security domain, for example by using Kerberos or other imbedded third party model.

HTTP Basic Authentication, which is based on a username and password, is the authentication mechanism defined in the HTTP/1.0 specification. A web server requests a web client to authenticate the user. As part of the request, the web server passes the realm (a string) in which the user is to be authenticated. The realm string of Basic Authentication does not have to reflect any particular security policy domain (confusingly also referred to as a realm). The web client obtains the username and the password from the user and transmits them to the web server. The web server then authenticates the user in the specified realm.

Basic Authentication is not a secure authentication protocol. User passwords are sent in simple BASE64 encoding, and the target server is not authenticated. Additional protection can alleviate some of these concerns: a secure transport mechanism (HTTPS), or security at the network level (such as the IPSEC protocol or VPN strategies) is applied in some deployment scenarios.

End user authentication using HTTPS (HTTP over SSL) is a strong authentication mechanism. This mechanism requires the user to possess a Public Key Certificate (PKC). Currently, PKCs are useful in e-commerce applications and also for a single-sign-on from within the browser. Servlet containers that are JEE technology compliant are required to support the HTTPS protocol.

Sources:

Web Services for Java EE Specification (9.1.1 Authentication) - [`http://www.jcp.org/en/jsr/detail?id=109`]

Java Servlet Specification 3.0 (Section 13.6 Authentication) - [`http://jcp.org/en/jsr/detail?id=315`]

8.2. Identify the purpose and benefits of Web services security oriented initiatives and standards such as Username Token Profile, SAML, XACML, XKMS, WS-Security, and the Liberty Project.

Question A080201

What are the key driving requirements for WS-Security communication protocol?

Options (select 3):

1. XML signature formats.
2. Non-repudiation.
3. Security token formats.

4. XML encryption technologies.
5. Establishing a security context.
6. Establishing authentication mechanisms.
7. Transport-level security.

Answer:

Correct options are 1, 3 and 4.

WS-Security describes enhancements to SOAP messaging to provide quality of protection through message integrity and message confidentiality. As well, this specification defines how to attach and include security tokens within SOAP messages. Finally, a mechanism is provided for specifying binary encoded security tokens (e.g. X.509 certificates). These mechanisms can be used independently or in combination to accommodate a wide variety of security models and encryption technologies.

WS-Security provides a general-purpose mechanism for associating security tokens with messages. No specific type of security token is required by WS-Security. It is designed to be extensible (e.g. support multiple security token formats). For example, a requester might provide proof of identity and proof that they have a particular business certification.

Message integrity is provided by leveraging XML Signature in conjunction with security tokens (which may contain or imply key data) to ensure that messages are transmitted without modifications. The integrity mechanisms are designed to support multiple signatures, potentially by multiple actors, and to be extensible to support additional signature formats. The signatures may reference (i.e. point to) a security token.

Similarly, message confidentiality is provided by leveraging XML Encryption in conjunction with security tokens to keep portions of SOAP messages confidential. The encryption mechanisms are designed to support additional encryption technologies, processes, and operations by multiple actors. The encryption may also reference a security token.

Finally, WS-Security describes a mechanism for encoding binary security tokens. Specifically, the specification describes how to encode X.509 certificates and Kerberos tickets as well as how to include opaque encrypted keys. It also includes extensibility mechanisms that can be used to further describe the characteristics of the security tokens that are included with a message.

The Web services security language must support a wide variety of security models. The following list identifies the key driving requirements for WS-Security specification:

- Multiple security token formats.
- Multiple trust domains.
- Multiple signature formats.
- Multiple encryption technologies.
- End-to-end message content security and not just transport-level security.

The following topics are outside the scope of WS-Security specification document:

- Establishing a security context or authentication mechanisms.
- Key derivation.
- Advertisement and exchange of security policy.
- How trust is established or determined.
- Non-repudiation.

Sources:

WS-Security: SOAP Message Security - [http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf]

Security in a Web Services World: A Proposed Architecture and Roadmap - [http://msdn.microsoft.com/en-us/library/ms977312.aspx]

Question A080202

Which initiative/standard enables single sign-on (SSO) service?

Options (select 1):

1. Username Token Profile
2. SAML
3. XACML
4. XKMS

Answer:

Correct option is 2.

SAML is an OASIS authentication and authorization standard based upon an XML framework for exchanging security information. This security information is expressed in the form of assertions about subjects, where a subject is an entity (either human or computer) that has an identity in some security domain. A typical example of a subject is a person, identified by his or her email address in a particular Internet DNS domain. One major design goal for SAML is Single Sign-On (SSO), the ability of a user to authenticate in one domain and use resources in other domains without re-authenticating.

Option 1 is wrong. The Web Services Security Username Token Profile document describes how to use the UsernameToken with the Web Services Security (WSS) specification; more specifically, it describes how a Web Service consumer can supply a UsernameToken as a means of identifying the requestor by 'username', and optionally using a password, to authenticate that identity to the Web Service producer.

Option 3 is wrong. XACML stands for Extensible Access Control Markup Language, and its primary goal is to standardize access control language in XML syntax. A standard access control language results in lower costs because there is no need to develop an application-specific access control language or write the access control policy in multiple languages. Plus, system administrators need to understand only one language. With XACML, it is also possible to compose access control policies from the ones created by different parties.

In a nutshell, XACML is a general-purpose access control policy language. This means that it provides a syntax (defined in XML) for managing access to resources.

XACML is an OASIS standard that describes both a policy language and an access control decision request/response language (both written in XML). The policy language is used to describe general access control requirements, and has standard extension points for defining new functions, data types, combining logic, etc. The request/response language lets you form a query to ask whether or not a given action should be allowed, and interpret the result. The response always includes an answer about whether the request should be allowed using one of four values: Permit, Deny, Indeterminate (an error occurred or some required value was missing, so a decision cannot be made) or Not Applicable (the request can't be answered by this service).

Option 4 is wrong. XKMS stands for the XML Key Management Specification and consists of two parts: XKISS (XML Key Information Service Specification) and XKRSS (XML Key Registration Service Specification). XKISS defines a protocol for resolving or validating public keys contained in signed and encrypted XML documents, while XKRSS defines a protocol for public key registration, revocation, and recovery. The key aspect of XKMS is that it serves as a protocol specification between an XKMS client and an XKMS server in which the XKMS server provides trust services to its

clients (in the form of Web services) by performing various PKI (public key infrastructure) operations, such as public key validation, registration, recovery, and revocation on behalf of the clients.

Sources:

Achieving SSO With Sun Java System Access Manager and SAML - [http://developers.sun.com/identity/reference/techart/sso.html]

OASIS Security Services - [http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security]

Security Assertion Markup Language (SAML) - [http://xml.coverpages.org/saml.html]

Question A080203

Which specifications/recommendations WS-Security supports?

Options (select 3):

1. Username Token
2. XML Encoding
3. XML Signature
4. XML Encryption

Answer:

Correct options are 1, 3 and 4.

WS-Security defines a standard set of SOAP extensions that implement message-level integrity and confidentiality for secure message exchanges.

WS-Security describes enhancements to SOAP messaging to provide quality of protection through the following features:

- Message integrity (XML Signature).
- Message confidentiality (XML Encryption).
- Single message authentication (Security Username Tokens).

These mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

Protecting the message content from being intercepted (confidentiality) or illegally modified (integrity) are primary security concerns. WS-Security specification provides a means to protect a message by encrypting and/or digitally signing a body, a header, an attachment, or any combination of them (or parts of them).

Message integrity is provided by leveraging XML Signature in conjunction with security tokens to ensure that messages are transmitted without modifications. The integrity mechanisms are designed to support multiple signatures, potentially by multiple actors, and to be extensible to support additional signature formats.

Message confidentiality leverages XML Encryption in conjunction with security tokens to keep portions of a SOAP message confidential. The encryption mechanisms are designed to support additional encryption processes and operations by multiple actors.

Sources:

Web Services Security (WS-Security) Specification (3.2. Message Protection) -

[http://www.ibm.com/developerworks/library/specification/ws-secure/]

WS-Security. Security for Web Services Article - [http://www.webservicesarchitect.com/content/articles/apshankar04.asp]

8.3. Given a scenario, implement Java EE based web service web-tier and/or EJB-tier basic security mechanisms, such as mutual authentication, SSL, and access control.

Question A080301

Service endpoint uses HTTP BASIC authentication. How can you authenticate a user to use the specific Web Service?

Options (select 1):

1.
```
String username = "mikalai";
String password = "qwerty";
Calculator calculator = (new CalculatorService()).getCalculatorPort();
BindingProvider provider = (BindingProvider) calculator;
provider.getMessageContext().put(BindingProvider.USERNAME_PROPERTY, username);
provider.getMessageContext().put(BindingProvider.PASSWORD_PROPERTY, password);
```

2.
```
String username = "mikalai";
String password = "qwerty";
Calculator calculator = (new CalculatorService()).getCalculatorPort();
BindingProvider provider = (BindingProvider) calculator;
provider.getRequestContext().put(BindingProvider.USERNAME_PROPERTY, username);
provider.getRequestContext().put(BindingProvider.PASSWORD_PROPERTY, password);
```

3.
```
String username = "mikalai";
String password = "qwerty";
Calculator calculator = (new CalculatorService()).getCalculatorPort();
BindingProvider provider = (BindingProvider) calculator;
provider.getRequestContext().put(MessageContext.USERNAME_PROPERTY, username);
provider.getRequestContext().put(MessageContext.PASSWORD_PROPERTY, password);
```

4.
```
String username = "mikalai";
String password = "qwerty";
Calculator calculator = (new CalculatorService()).getCalculatorPort();
BindingProvider provider = (BindingProvider) calculator;
provider.getMessageContext().put(MessageContext.USERNAME_PROPERTY, username);
provider.getMessageContext().put(MessageContext.PASSWORD_PROPERTY, password);
```

Answer:

Correct option is 2.

Sometimes, invoking Web Services require exchange of additional information or metadata (not captured in SOAP message). This metadata forms the context of message exchange and may in-turn change the way the message is processed. JAX-WS Specification defines some standard properties to describe the metadata and also provides a standard way to manipulate or exchange such information.

A client application can configure metadata through request context and response context of the `BindingProvider` (instance of `Proxy` or `Dispatch`). The request and response contexts are of type `java.util.Map<String,Object>` and are obtained using `getRequestContext` and `getResponseContext` methods of the `BindingProvider`.

You can implement BASIC HTTP authentication in JAX-WS using the following example:

```
HelloService service = new HelloService();
Hello proxy = service.getHelloPort();
((BindingProvider)proxy).getRequestContext().put(BindingProvider.USERNAME_PROPERTY, "username");
((BindingProvider)proxy).getRequestContext().put(BindingProvider.PASSWORD_PROPERTY, "password");
```

`USERNAME_PROPERTY`, `PASSWORD_PROPERTY` are used primarily for service requests.

When you instantiate `Service`, it fetches WSDL. If the server is returning 401 error, you could try any one of the following solutions:

- Use `java.net.Authenticator` class in your client application.
- Provide a local access to the WSDL using catalog.
- Configure `WEB-INF/web.xml` to allow GET requests without authentication.

Options 1 and 4 are wrong because there is no `getMessageContext` method in `BindingProvider` interface.

Option 3 is wrong because there are no `USERNAME_PROPERTY` and `PASSWORD_PROPERTY` constants defined in `MessageContext` interface.

Sources:

JAX-WS FAQ - [`http://jax-ws.java.net/faq/index.html`]

`BindingProvider` Interface JavaDoc - [`http://download.oracle.com/javaee/6/api/javax/xml/ws/BindingProvider.html`]

`Dispatch` Interface JavaDoc - [`http://download.oracle.com/javaee/6/api/javax/xml/ws/Dispatch.html`]

Question A080302

You want to use HTTP BASIC authentication method for your service endpoint. Also, you know that in this case passwords can be intercepted. In addition to authentication you decided to use secure HTTPS connection (HTTP over SSL). How can you configure these options in deployment descriptor (`WEB-INF/web.xml`)?

Options (select 2):

1.
```
<!-- SECURITY CONSTRAINT -->
<security-constraint>
        ...
        <user-data-constraint>
                <transport-guarantee>SECURE</transport-guarantee>
        </user-data-constraint>
</security-constraint>

<!-- LOGIN AUTHENTICATION -->
<login-config>
        <auth-method>BASIC_AUTH</auth-method>
</login-config>
```

2.
```
<!-- SECURITY CONSTRAINT -->
<security-constraint>
        ...
        <user-data-constraint>
                <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
```

```
</security-constraint>

<!-- LOGIN AUTHENTICATION -->
<login-config>
        <auth-method>BASIC</auth-method>
</login-config>
```

3.

```
<!-- SECURITY CONSTRAINT -->
<security-constraint>
        ...
        <user-data-constraint>
                <transport-guarantee>SECURE</transport-guarantee>
        </user-data-constraint>
</security-constraint>

<!-- LOGIN AUTHENTICATION -->
<login-config>
        <auth-method>BASIC</auth-method>
</login-config>
```

4.

```
<!-- SECURITY CONSTRAINT -->
<security-constraint>
        ...
        <user-data-constraint>
                <transport-guarantee>INTEGRAL</transport-guarantee>
        </user-data-constraint>
</security-constraint>

<!-- LOGIN AUTHENTICATION -->
<login-config>
        <auth-method>BASIC</auth-method>
</login-config>
```

5.

```
<!-- SECURITY CONSTRAINT -->
<security-constraint>
        ...
        <user-data-constraint>
                <transport-guarantee>INTEGRAL</transport-guarantee>
        </user-data-constraint>
</security-constraint>

<!-- LOGIN AUTHENTICATION -->
<login-config>
        <auth-method>BASIC_AUTH</auth-method>
</login-config>
```

6.

```
<!-- SECURITY CONSTRAINT -->
<security-constraint>
        ...
        <user-data-constraint>
                <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
</security-constraint>

<!-- LOGIN AUTHENTICATION -->
<login-config>
        <auth-method>BASIC_AUTH</auth-method>
```

```
</login-config>
```

Answer:

Correct options are 2 and 4.

When the login authentication method is set to BASIC, passwords are not protected, meaning that passwords sent between a client and a server on a non-protected session can be viewed and intercepted by third parties.

To configure HTTP BASIC authentication over SSL, specify `CONFIDENTIAL` or `INTEGRAL` within the `transport-guarantee` element.

The `transport-guarantee` element specifies that the communication between client and server should be `NONE`, `INTEGRAL`, or `CONFIDENTIAL`.

This means options 1 and 3 are wrong.

`NONE` means that the application does not require any transport guarantees. A value of `INTEGRAL` means that the application requires that the data sent between the client and server be sent in such a way that it can't be changed in transit. `CONFIDENTIAL` means that the application requires that the data be transmitted in a fashion that prevents other entities from observing the contents of the transmission. In most cases, the presence of the `INTEGRAL` or `CONFIDENTIAL` flag will indicate that the use of SSL is required.

The `auth-method` element is used to configure the authentication mechanism for the web application. As a prerequisite to gaining access to any web resources which are protected by an authorization constraint, a user must have authenticated using the configured mechanism. Legal values for this element are "`BASIC`", "`DIGEST`", "`FORM`", or "`CLIENT-CERT`".

This means options 5 and 6 are wrong.

Sources:

Java Servlet Specification 3.0 - [`http://jcp.org/en/jsr/detail?id=315`]

8.4. Describe factors that impact the security requirements of a Web Service, such as the relationship between the client and service provider, the type of data being exchanged, the message format, and the transport mechanism.

blah-blah

8.5. Describe WS-Policy that defines a base set of constructs that can be used and extended by other Web Services specifications to describe a broad range of service requirements and capabilities.

Question A080501

Which specification describes how senders and receivers (intermediaries and endpoints) can specify their security requirements and capabilities (e.g. required security tokens, supported encryption algorithms, privacy rules)?

Options (select 1):

1. WS-Security
2. WS-Policy
3. WS-Trust
4. WS-Privacy

5. WS-SecureConversation
6. WS-Federation
7. WS-Authorization

Answer:

Correct option is 2.

WS-Security: describes how to attach signature and encryption headers to SOAP messages. In addition, it describes how to attach security tokens, including binary security tokens such as X.509 certificates and Kerberos tickets, to messages.

WS-Policy: describes the capabilities and constraints of the security (and other business) policies on intermediaries and endpoints (e.g. required security tokens, supported encryption algorithms, privacy rules).

WS-Trust: describes a framework for trust models that enables Web services to securely interoperate.

WS-Privacy: describes a model for how Web services and requesters state privacy preferences and organizational privacy practice statements.

WS-SecureConversation: describes how to manage and authenticate message exchanges between parties including security context exchange and establishing and deriving session keys.

WS-Federation: describes how to manage and broker the trust relationships in a heterogeneous federated environment including support for federated identities.

WS-Authorization: describes how to manage authorization data and authorization policies.

Sources:

MSDN [Security in a Web Services World] - [http://msdn.microsoft.com/en-us/library/ms977312.aspx]

Appendix 9. Developing Web Services

9.1. Describe the steps required to configure, package, and deploy Java EE Web services and service clients, including a description of the packaging formats, such as .ear, .war, .jar, annotations and deployment descriptor settings.

Question A090101

Given the following Service Implementation Bean (SIB) code:

```
@WebService(serviceName="HelloService",
   name="HelloPortType",
   portName="HelloServicePort")
@Stateless(name="Hello")
public class Hello {
    public String sayHello(String name) {
        return "Hello, " + name;
    }
}
```

Which statements are correct about the Web Service packaging?

Options (select 4):

1. Web Service can be packaged in WAR module.
2. Web Service can be packaged in EJB-JAR module.

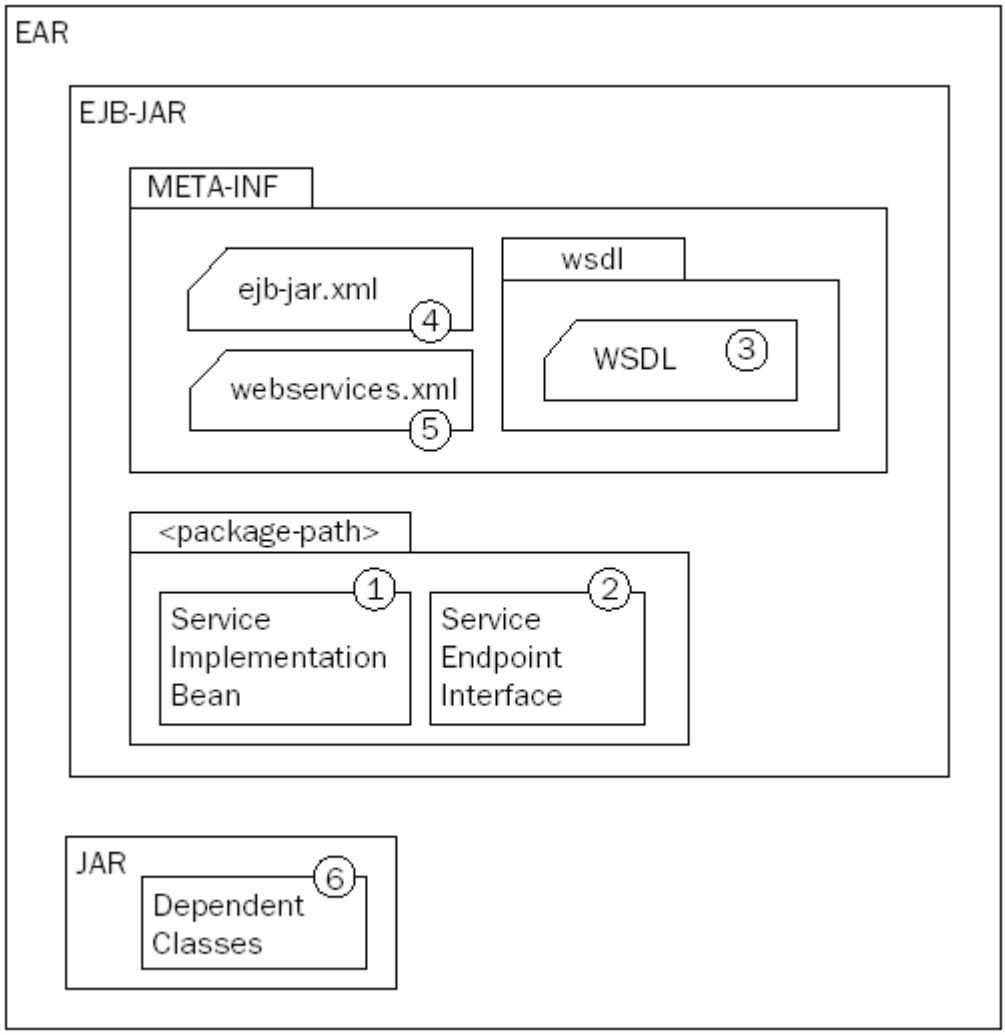
3. Web Service can be packaged in RAR module.
4. Compiled SIB class can placed in the "`/`" directory within archive.
5. Compiled SIB class can placed in the "`/WEB-INF/classes`" directory within archive.
6. Compiled SIB class can placed in the "`/META-INF/classes`" directory within archive.

Answer:

Correct options are: 1, 2, 4, and 5.

Stateless or Singleton Session EJB Service Implementation Beans are packaged in an EJB-JAR that contains the class files and WSDL files. In addition, the Web Services Deployment Descriptor location within the EJB-JAR file is `META-INF/webservices.xml`. The wsdl directory is located at `META-INF/wsdl`

When you package a SIB for deployment as a Stateless Session Bean endpoint, your EJB-JAR should be structured as shown in the figure below:



1. Service Implementation Bean (required) is contained in the `/<package-path>` directory, where `<package-path>` is determined by the class' package name. This class file must be annotated with `@Stateless`, and `@WebService` or `@WebServiceProvider`. Its methods contain the business logic that implements the Web Service operations.
2. Service Endpoint Interface (optional) is contained in the `/<package-path>` directory, where `<package-path>` is determined by the interface's package name. When used, the SIB's

`@WebService.endpointInterface` attribute's value must equal the complete name of this SEI.

3. WSDL (optional) is contained in the `/META-INF/wsdl` directory.
4. `ejb-jar.xml` (optional) is contained in the `/META-INF` directory.
5. `webservices.xml` (optional) is contained in the `/META-INF` directory.
6. Dependent Classes (optional) are contained in a separate JAR ╖le at the root of the enclosing EAR where they will be available on the application classpath. These are any classes the SIB or SEI depends on. This is not a mandatory location. The dependent classes may be located anywhere on the application classpath.

JAX-RPC/JAX-WS Service Endpoints and Stateless/Singleton EJB as JAX-WS Service endpoints CAN be packaged (as of Java EE 6) in a **WAR file** that contains the class files and WSDL files. An enterprise bean class with a component-defining annotation defines an enterprise bean component when packaged within the `WEB-INF/classes` directory or in a `.jar` file within `WEB-INF/lib`. An enterprise bean can also be defined via `WEB-INF/ejb-jar.xml`.

Sources:

Web Services for Java EE, Version 1.3 (Section 5.4.2, EJB Module Packaging; Section 5.4.3, WebApp Module Packaging) - [`http://jcp.org/en/jsr/detail?id=109`]

Question A090102

Given the following Service Implementation Bean (SIB) code:

```
@WebService(serviceName="Hello")
public class Hello {
    public String sayHello(String name) {
        return "Hello, " + name;
    }
}
```

Which statements are correct about the Web Service packaging?

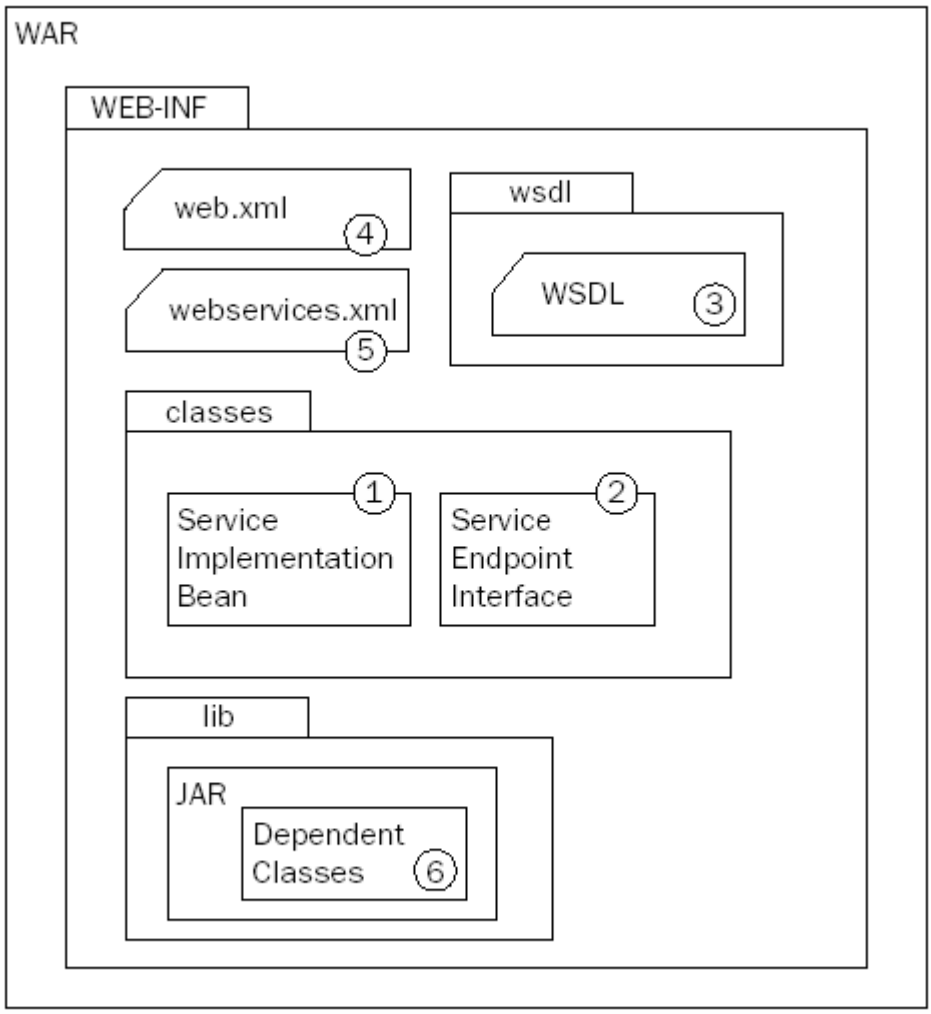
Options (select 2):

1. Web Service is packaged in WAR module.
2. Web Service is packaged in EJB-JAR module.
3. Web Service is packaged in RAR module.
4. Compiled SIB class is placed in the "`/`" directory within archive.
5. Compiled SIB class is placed in the "`/WEB-INF/classes`" directory within archive.
6. Compiled SIB class is placed in the "`/META-INF/classes`" directory within archive.

Answer:

Correct options are 1 and 5.

When you package a SIB for deployment as a servlet endpoint, your WAR is structured as shown in the figure below:



1. Service Implementation Bean (required) is contained in the `/WEB-INF/classes/<package-path>` directory, where `<package-path>` is determined by the class' package name. This location is not mandatory. Alternatively, the SIB could be contained in a JAR under the `/WEB-INF/lib`. This class file must be annotated with `@WebService` or `@WebServiceProvider`. Its methods contain the business logic that implements the Web Service operations.
2. Service Endpoint Interface (optional) is contained in the `/WEB-INF/classes/<package-path>` directory, where `<package-path>` is determined by the interface's package name. When used, the SIB's `@WebService.endpointInterface` attribute's value must equal the complete name of this SEI.
3. WSDL (optional) is contained in the `/WEB-INF/wsdl` directory.
4. `web.xml` (optional) is contained in the `/WEB-INF` directory.
5. `webservices.xml` (optional) is contained in the `/WEB-INF` directory.
6. Dependent Classes (optional) are bundled in a JAR and contained under the `/WEB-INF/lib` directory where they are available on the application classpath. These are any classes the SIB or SEI depend on. This is not a mandatory location, but one possible approach when the SIB depends on a library of classes that may already be packaged in a JAR. The dependent classes may be located anywhere on the application classpath.

Sources:

JSR 109: Web Services for Java EE, Version 1.3 (Section 5.4.3, WebApp Module Packaging) - [http://jcp.org/en/jsr/detail?id=109]

9.2. Given a set of requirements, develop code to process XML files using the SAX, StAX, DOM, XSLT, and JAXB APIs.

Question A090201

Your code is using vendor-neutral Java API for XML Processing (JAXP). You need to change parser factory implementation class, because current parser does not support namespaces. You know that JAXP API allows applications to plug in different JAXP compatible implementations of parsers or XSLT processors. What order is used when JAXP searches for the name of a concrete subclass of `DocumentBuilderFactory`?

```
import java.io.File;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;

public class TestDOMParsing {
        public static void main(String[] args) throws Exception {
                // Get Document Builder Factory
                DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
                DocumentBuilder builder = factory.newDocumentBuilder();
                Document doc = builder.parse(new File(args[0]));
        }
}
```

1. The value of a system property `javax.xml.parsers.DocumentBuilderFactory` if it exists and is accessible.
2. The contents of the file `$JAVA_HOME/jre/lib/jaxp.properties` if it exists.
3. The JAR Service Provider discovery mechanism specified in the JAR File Specification. A JAR file can have a resource (i.e. an embedded file) such as `META-INF/services /javax.xml.parsers.DocumentBuilderFactory` containing the name of the concrete class to instantiate.
4. The fallback platform default implementation.

Options (select 1):

1. 2, 1, 3, 4
2. 1, 2, 3, 4
3. 3, 2, 1, 4
4. 2, 3, 1, 4

Answer:

Correct option is 2.

The JAXP classes in standard extension package `javax.xml.parsers` are all marked "`abstract`". This is because the parsers and parser factories are implemented by classes outside of the package. JAXP's vendor independence comes from this separation. The `abstract` classes `DocumentBuilderFactory` and `SAXParserFactory` each implement one method: `newInstance`. This method creates a new instance of the factory class. The package is vendor-neutral because the `newInstance` method looks in several places for the name of a class that fully implements the abstract factory interface.

JAXP comes pre-configured to use a default set of classes for parsers and parser factories. Which class is used depends on the application vendor. To change the parser used by an application, you need to tell JAXP the fully-qualified class name of the parser factory (`SAXParserFactory` or `DocumentBuilderFactory`) to use. There are three ways to do this (search is doing in the following order too):

- Set a system property. The system property `javax.xml.parsers.SAXParserFactory` can be set to the fully-qualified class name of your alternative SAX parser factory. The system property `javax.xml.parsers.DocumentBuilderFactory` can be set to the class name of the alternative DOM parser factory. System properties are set when the application starts. For the JEE Reference Implementation, you can simply set the system property on the command line using the `-D` option:

```
java -Dxml.parsers.DocumentBuilderFactory=com.example.parsers.MyDOMFactory TestDOMParsing tes
```

- Specify classes in the file `$JAVA_HOME/lib/jaxp.properties`. You can set the same two properties (`javax.xml.parsers.SAXParserFactory` and `javax.xml.parsers.DocumentBuilderFactory`) using standard properties notation in a file called `jaxp.properties` in the `lib` directory of your JEE platform main directory. For example, here are two lines the the files that identify classes for alternative SAX and DOM parsers:

```
javax.xml.parsers.SAXParserFactory=com.example.parsers.MySAXFactory
javax.xml.parsers.DocumentBuilderFactory=com.example.parsers.MyDOMFactory
```

- Specify the classes as an entry in the services directory of a JAR file. The JAR file specification defines a directory called `META-INF/services` that JAXP checks to find definitions of parser factories. JAXP uses the fully-qualified class names found in the files `META-INF/services /javax.xml.parsers.SAXParserFactory` and `META-INF/services /javax.xml.parsers.DocumentBuilderFactory` as the parser factory classes, if those files exist.

If JAXP Runtime does not find a parser in any of these locations, it instantiates the default parser provided by the JAXP vendor.

Sources:

`javax.xml.parsers.DocumentBuilderFactory` JavaDoc - [`http://download.oracle.com /javase/1.5.0/docs/api/javax/xml/parsers/DocumentBuilderFactory.html`]

`javax.xml.parsers.SAXParserFactory` JavaDoc - [`http://download.oracle.com/javase /1.5.0/docs/api/javax/xml/parsers/SAXParserFactory.html`]

Question A090202

What are the correct ways to parse XML file with SAX parser?

```
import java.io.File;

import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.InputSource;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.helpers.XMLReaderFactory;

public class ParseWithSAX {
        public static void main(String[] args) throws Exception {
                String fileName = "example.xml";

                // add code here
```

```
                System.out.println("XML file is parsed");
        }
}
```

Options (select 3):

1.
```
XMLReader xmlReader = XMLReaderFactory.createXMLReader();
xmlReader.setContentHandler(new MyContentHandler());
xmlReader.parse(fileName);
```

2.
```
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser saxParser = factory.newSAXParser();
XMLReader xmlReader = saxParser.getXMLReader();
InputSource source = new InputSource(fileName);
xmlReader.setContentHandler(new MyContentHandler());
xmlReader.parse(source);
```

3.
```
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser saxParser = factory.newSAXParser();
saxParser.parse(new File(fileName), new MyContentHandler());
```

4.
```
SAXParserFactory factory = new SAXParserFactory();
SAXParser saxParser = factory.newSAXParser();
XMLReader xmlReader = saxParser.getXMLReader();
InputSource source = new InputSource(fileName);
xmlReader.setContentHandler(new MyContentHandler());
xmlReader.parse(source);
```

5.
```
XMLReader xmlReader = XMLReaderFactory.newXMLReader();
xmlReader.setContentHandler(new MyContentHandler());
xmlReader.parse(fileName);
```

6.
```
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser saxParser = factory.createSAXParser();
saxParser.parse(new File(fileName), new MyContentHandler());
```

Answer:

Correct options are 1, 2, 3.

Option 1. SAX represents parsers as instances of the `XMLReader` interface (interface for reading an XML document using callbacks). The specific class that implements this interface varies from parser to parser. For example, in Xerces it's `org.apache.xerces.parsers.SAXParser`. In Crimson it's `org.apache.crimson.parser.XMLReaderImpl`. Most of the time you don't construct instances of this interface directly. Instead you use the static `XMLReaderFactory.createXMLReader()` factory method to create a parser-specific instance of this class. Then you pass `InputSource` or `String` object containing XML documents to the `parse()` method of `XMLReader`. The parser reads the document, and throws an exception if it detects any well-formedness errors.

NOTE: You will probably use most of the time this way to parse XML with SAX parser.

Option 2. This example uses a pair of classes, `SAXParserFactory` and `SAXParser`, to create the parser, so you don't have to know the name of the driver itself. First get instance if `SAXParserFactory` using `newInstance()` method. Then, use `SAXParserFactory` to create a `SAXParser`. It's the `SAXParser` that gives you the `XMLReader`. Note, `XMLReader` has two

`parse(...)` methods: `parse(org.xml.sax.InputSource s)` and `parse(java.lang.String systemId)`.

NOTE: `SAXParserFactory` is an obsolete class for building and configuring `SAXParser` objects in an implementation independent fashion. The concrete subclass to load is read from the `javax.xml.parsers.SAXParserFactory` Java system property. This class has been replaced by the `org.xml.sax.helpers.XMLReaderFactory` class in SAX2, and there's little reason to use it any more.

Option 3. You first need to create a `SAXParser` object from a `SAXParserFactory` object. You would call the method `parse` on it, passing it an XML file and an instance of your new handler class. In this example, we pass XML file, but the `parse` method can also take a variety of other input sources, including an `InputStream` object, a `URL` object, and an `InputSource` object.

NOTE: `SAXParserFactory` is an obsolete class for building and configuring `SAXParser` objects in an implementation independent fashion. The concrete subclass to load is read from the `javax.xml.parsers.SAXParserFactory` Java system property. This class has been replaced by the `org.xml.sax.helpers.XMLReaderFactory` class in SAX2, and there's little reason to use it any more.

Option 4. The line:

```
SAXParserFactory factory = new SAXParserFactory();
```

is invalid. The `newInstance();` method is used to instantiate factory object.

Option 5. The line:

```
XMLReader xmlReader = XMLReaderFactory.newXMLReader();
```

is invalid. The `XMLReaderFactory.createXMLReader();` method is used to create XML reader.

Option 6. The line:

```
SAXParser saxParser = factory.createSAXParser();
```

is invalid. The `newSAXParser();` method is used to create SAX parser.

Sources:

Official SAX website - [`http://sax.sourceforge.net/`]

JAXP Tutorial - [`http://java.sun.com/webservices/reference/tutorials/jaxp/html/sax.html`]

Question A090203

What is the correct way to parse XML file with DOM parser?

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;

public class ParseWithDOM {
        public static void main(String[] args) throws Exception {
                String fileName = "example.xml";

                // add code here
```

```
                System.out.println("XML file is parsed");
        }
}
```

Options (select 1):

1.
```
DocumentBuilderFactory factory = new DocumentBuilderFactory();
DocumentBuilder builder = factory.newDocumentBuilder();
Document document = builder.parse(fileName);
```

2.
```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document document = builder.parse(fileName);
```

3.
```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.createDocumentBuilder();
Document document = builder.parse(fileName);
```

4.
```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.createDocumentBuilder();
Document document = builder.parse(new File(fileName), new MyContentHandler());
```

Answer:

Correct option is 2.

Option 1 is wrong because `DocumentBuidelrFactory` is created not with constructor (this class is `abstract` and can't be instantiated with class constructor), but with static method `newInstance()`.

Option 2. If your parser implements JAXP you can use the `javax.xml.parsers.DocumentBuilderFactory` and `javax.xml.parsers.DocumentBuilder` classes to parse the documents. The basic approach is as follows:

- Use the static `DocumentBuilderFactory.newInstance()` factory method to return a `DocumentBuilderFactory` object.
- Use the `newDocumentBuilder()` method of this `DocumentBuilderFactory` object to return a parser-specific instance of the abstract `DocumentBuilder` class.
- Use one of the five `parse(...)` methods of `DocumentBuilder` to read the XML document and return an `org.w3c.dom.Document` object.

Our program creates a `DocumentBuilderFactory` object, which is then used to create the `DocumentBuilder` object builder. The code then calls the `parse(...)` method on builder, passing it the XML file. Other possible arguments for `parse(...)` are:

```
public abstract Document parse(InputSource is)  throws  SAXException, IOException;

public Document parse(File f) throws SAXException, IOException {...}
public Document parse(InputStream is) throws SAXException, IOException {...}
public Document parse(InputStream is, String systemId)  throws SAXException, IOException {...}
public Document parse(String uri)  throws SAXException, IOException {...}
```

Option 3 is wrong because the method `createDocumentBuilder()` is invalid. You can use `newDocumentBuilder()` to create document builder.

Option 4 is wrong because the method `createDocumentBuilder()` is invalid. You can use `newDocumentBuilder()` to create document builder. Also, you can not use `ContentHandler` classes with DOM parsers, they are intended to be used with SAX parsers.

Sources:

Code sample. Java API for XML Processing API (JAXP) Using DOM - [`http://developers.sun.com/sw/building/codesamples/dom/`]

JAXP Tutorial - [`http://java.sun.com/webservices/reference/tutorials/jaxp/html/dom.html`]

Question A090204

The XML file you are parsing references multiple XML Schemas. As a result - it uses namespaces to avoid elements names ambiguity. How can you make your XML parsers support namespaces and provide namespace URIs, local element names and prefixed element names?

```
import java.io.File;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.w3c.dom.Document;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.DefaultHandler;

public class ParseWithNamespaces {

        public static void main(String[] args) throws Exception {
                String fileName = "example.xml";

                DocumentBuilderFactory domFactory = DocumentBuilderFactory.newInstance();
                SAXParserFactory saxFactory = SAXParserFactory.newInstance();

                // 1

                DocumentBuilder builder = domFactory.newDocumentBuilder();
                SAXParser saxParser = saxFactory.newSAXParser();
                XMLReader xmlReader = saxParser.getXMLReader();

                // 2

                Document document = builder.parse(fileName);
                saxParser.parse(new File(fileName), new MyContentHandler());
        }
}
```

Options (select 1):

1.
```
// 1
domFactory.setNamespaceAware(true);
saxFactory.setNamespaceAware(true);
xmlReader.setFeature("http://xml.org/sax/features/namespace-prefixes", true);
xmlReader.setFeature("http://xml.org/sax/features/namespaces", true);

// 2
// nothing
```

2.
```
// 1
// nothing

// 2
```

```
domFactory.setNamespaceAware(true);
saxFactory.setNamespaceAware(true);
xmlReader.setFeature("http://xml.org/sax/features/namespace-prefixes", true);
xmlReader.setFeature("http://xml.org/sax/features/namespaces", true);
```

3.
```
// 1
xmlReader.setFeature("http://xml.org/sax/features/namespace-prefixes", true);
xmlReader.setFeature("http://xml.org/sax/features/namespaces", true);

// 2
domFactory.setNamespaceAware(true);
saxFactory.setNamespaceAware(true);
```

4.
```
// 1
domFactory.setNamespaceAware(true);
saxFactory.setNamespaceAware(true);

// 2
xmlReader.setFeature("http://xml.org/sax/features/namespace-prefixes", true);
xmlReader.setFeature("http://xml.org/sax/features/namespaces", true);
```

Answer:

Correct option is 4.

Option 1 is wrong because code will not compile.

Option 2 is syntactically correct, but parsers will ignore namespaces.

Option 3 is wrong because code will not compile.

Option 4 is correct. First we call `setNamespaceAware(true)` on parser factories. These methods determine whether the parsers produced by this factory are "namespace aware". A namespace aware parser will set the prefix and namespace URI properties of element and attribute nodes that are in a namespace. A non-namespace aware parser will not, it will interpret the "`foo:bar`" as the element's local name.

Then we set features (properties) on SAX `XMLReader`: the namespace URI and local name are required when the `http://xml.org/sax/features/namespaces` property is `true` (the default), and are optional when the `http://xml.org/sax/features/namespaces` property is `false` (if one is specified, both must be); the qualified name is required when the `http://xml.org/sax/features/namespace-prefixes` property is `true`, and is optional when the `http://xml.org/sax/features/namespace-prefixes` property is `false` (the default).

Sources:

`DocumentBuilderFactory` API JavaDoc - [`http://download.oracle.com/javase/6/docs/api/javax/xml/parsers/DocumentBuilderFactory.html`]

`SAXParserFactory` API JavaDoc - [`http://download.oracle.com/javase/6/docs/api/javax/xml/parsers/SAXParserFactory.html`]

`ContentHandler` API JavaDoc - [`http://download.oracle.com/javase/6/docs/api/org/xml/sax/ContentHandler.html`]

`XMLReader` API JavaDoc - [`http://www.saxproject.org/apidoc/org/xml/sax/XMLReader.html`]

Question A090205

You are parsing SOAP fault message with SAX parser and need to know namespace for `faulcode`

element value.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" >
  <soap:Body>
    <soap:Fault>
      <faultcode>soap:Client</faultcode>
      <faultstring>Wrong parameter</faultstring>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

What method of `org.xml.sax.ContentHandler` can you use to retrieve element value namespace URI?

Options (select 1):

1. `startDocument()`
2. `processingInstruction(...)`
3. `startElement(...)`
4. `ignorableWhitespace(...)`
5. `startPrefixMapping(...)`
6. `characters(...)`

Answer:

Correct option is 5

Option 1. This option is wrong. The method `startDocument()` receives notification of the beginning of a document.

Option 2. This option is wrong. The method `processingInstruction(String target, String data)` receives notification of a processing instruction. The Parser will invoke this method once for each processing instruction found: note that processing instructions may occur before or after the main document element. A SAX parser must never report an XML declaration.

Option 3. This option is wrong. The method `startElement(String namespaceURI, String localName, String qName, Attributes atts)` receives notification of the beginning of an element. The Parser will invoke this method at the beginning of every element in the XML document; there will be a corresponding `endElement` event for every `startElement` event (even when the element is empty). All of the element's content will be reported, in order, before the corresponding `endElement` event. This event allows up to three name components for each element: the Namespace URI; the local name; and the qualified (prefixed) name. Another argument is the attributes attached to the element. If there are no attributes, it shall be an empty `Attributes` object.

Option 4. This option is wrong. The method `ignorableWhitespace(char[] ch, int start, int length)` receives notification of ignorable whitespace in element content. Validating Parsers must use this method to report each chunk of whitespace in element content, non-validating parsers may also use this method if they are capable of parsing and using content models. SAX parsers may return all contiguous whitespace in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity, so that the `Locator` provides useful information. The application must not attempt to read from the array outside of the specified range.

Option 5. This option is correct. The method `startPrefixMapping(String prefix, String uri)`

begins the scope of a prefix-URI Namespace mapping. There are cases when applications need to use the prefixes and XML namespaces URIs of attribute values and element values. The `startPrefixMapping` event supplies the information to the application to expand prefixes in those contexts itself, if necessary. All `startPrefixMapping` events will occur before the corresponding `startElement` event, and all `endPrefixMapping` events will occur after the corresponding `endElement` event, but their order is not otherwise guaranteed.

Option 6. This option is wrong. The method `characters(char[] ch, int start, int length)` receive notification of character data. The Parser will call this method to report each chunk of character data. SAX parsers may return all contiguous character data in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity so that the `Locator` provides useful information. The application must not attempt to read from the array outside of the specified range.

Sources:

`org.xml.sax.ContentHandler` API JavaDoc - [`http://download.oracle.com/javase/6/docs/api/org/xml/sax/ContentHandler.html`]

Official SAX website - [`http://sax.sourceforge.net/`]

Question A090206

How can you create instance of `javax.xml.parsers.DocumentBuilderFactory`?

Options (select 1):

1. ```
   DocumentBuilderFactory factory = DocumentBuilderFactory.init();
   ```
2. ```
   DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
   ```
3. ```
   DocumentBuilderFactory factory = new DocumentBuilderFactory();
   ```
4. ```
   DocumentBuilderFactory factory = DocumentBuilderFactory.newFactory();
   ```

Answer:

Correct option is 2

Options 1 and 4 are wrong because methods are invalid.

Option 2 is correct. The `newInstance()` method obtains a new instance of a `DocumentBuilderFactory`. This static method creates a new factory instance. This method uses the following ordered lookup procedure to determine the `DocumentBuilderFactory` implementation class to load:

- Use the `javax.xml.parsers.DocumentBuilderFactory` system property.
- Use the properties file "`lib/jaxp.properties`" in the JRE directory. This configuration file is in standard `java.util.Properties` format and contains the fully qualified name of the implementation class with the key being the system property defined above.
- Use the Services API (as detailed in the JAR specification), if available, to determine the classname. The Services API will look for a classname in the file `META-INF/services/javax.xml.parsers.DocumentBuilderFactory` in JARs available to the runtime.
- Platform default `DocumentBuilderFactory` instance.

Once an application has obtained a reference to a `DocumentBuilderFactory` it can use the factory to configure and obtain parser instances.

Option 3. This option is wrong. The `DocumentBuilderFactory` is `abstract`, so you can not instantiate it using constructor.

Sources:

`javax.xml.parsers.DocumentBuilderFactory` API JavaDoc - [`http://download.oracle.com/javase/6/docs/api/javax/xml/parsers/DocumentBuilderFactory.html`]

Question A090207

You are writing a Java Servlet which transforms business data in XML format to HTML presentation format. Which JAXP code fragment correctly performs XSLT transformation in multi-threaded environment (servlet container)?

Options (select 1):

1.
```
private Transformer transformer;
...
// init()
TransformerFactory factory  = TransformerFactory.newInstance();
Source xsl = new StreamSource("xml2html.xsl");
transformer = factory.newTransformer(xsl);
...
// doGet()
Source request  = new StreamSource(in);
Result response = new StreamResult(out);
transformer.transform(request, response);
```

2.
```
private Transformer transformer;
...
// init()
TransformerFactory factory  = TransformerFactory.newInstance();
transformer = factory.newTransformer();
...
// doGet()
Source request  = new StreamSource(in);
Result response = new StreamResult(out);
Source xsl = new StreamSource("xml2html.xsl");
transformer.transform(request, response, xsl);
```

3.
```
private Templates templates;
...
// init()
TransformerFactory factory = TransformerFactory.newInstance();
Source xsl = new StreamSource("xml2html.xsl");
templates = factory.newTemplates(xsl);
...
// doGet()
Source request    = new StreamSource(in);
Result response   = new StreamResult(out);
Transformer transformer = templates.newTransformer();
transformer.transform(request, response);
```

4.
```
private Templates templates;
...
// init()
TransformerFactory factory = TransformerFactory.newInstance();
templates = factory.newTemplates();
...
// doGet()
Source request    = new StreamSource(in);
```

```
Result response   = new StreamResult(out);
Source xsl = new StreamSource("xml2html.xsl");
Transformer transformer = templates.newTransformer();
transformer.transform(request, response, xsl);
```

Answer:

Correct option is 3.

TrAX, the Transformations API for XML, is a Java API for performing XSLT transforms. TrAX is a standard part of JAXP, and is bundled with Java 1.4 and later.

These are main TrAX classes and interfaces that you need to use, all in the `javax.xml.transform` package:

- `Transformer` - The class that represents the style sheet. It transforms a `Source` into a `Result`. `Transformer` is NOT guaranteed to be thread-safe.
- `TransformerFactory` - The class that represents the XSLT processor. This is a factory class that reads a stylesheet to produce a new `Transformer` or new `Templates`. `TransformerFactory` is NOT guaranteed to be thread-safe.
- `Source` - The interface that represents the input XML document to be transformed, whether presented as a DOM tree, an `InputStream`, or a SAX event sequence.
- `Result` - The interface that represents the XML document produced by the transformation, whether generated as a DOM tree, an `OutputStream`, or a SAX event sequence.
- `Templates` - The `Templates` class represents the parsed stylesheet. You can then ask the `Templates` class to give you as many separate `Transformer` objects as you need, each of which can be created very quickly by copying the processor's in-memory data structures rather than by reparsing the entire stylesheet from disk or the network. The `Templates` class itself can be safely used across multiple threads. This technique is particularly important in server environments where the transform may be applied to thousands of different input documents with potentially dozens being processed in parallel in separate threads at the same time.

To transform in thread-safe environment an input document into an output document follow these steps:

1. Load the `TransformerFactory` with the static `TransformerFactory.newInstance()` factory method.
2. Form a `Source` object from the XSLT stylesheet.
3. Pass this `Source` object to the factory's `newTransformer()` factory method to build a `Transformer` object.
4. Build a `Source` object from the input XML document you wish to transform.
5. Build a `Result` object for the target of the transformation.
6. Pass both the source and the result to the `Transformer` object's `transform()` method.

To transform in multi-threaded environment an input document into an output document follow these steps:

1. Load the `TransformerFactory` with the static `TransformerFactory.newInstance()` factory method.
2. Form a `Source` object from the XSLT stylesheet.
3. Pass this `Source` object to the factory's `newTemplates()` factory method to build a `Templates`

object.

4. Build a `Source` object from the input XML document you wish to transform.
5. Build a `Result` object for the target of the transformation.
6. Call `Templates.newTransformer()` method to build a `Transformer` object.
7. Pass both the source and the result to the `Transformer` object's `transform()` method.

Option 1 is not valid for multi-threaded environment because `Transformer` is not thread safe. For thread safe environment this code works fine.

Options 2 and 4 are wrong because method signatures (passed parameters) are invalid.

Sources:

TrAX - [`http://www.cafeconleche.org/books/xmljava/chapters/ch17s02.html`]

Question A090208

Currently your application is using following code to transform incoming XML documents with the same XSLT template. You need to rewrite your application for multi-threaded environment to improve performance. What is the best approach to use?

```
// code part 1
TransformerFactory factory = TransformerFactory.newInstance();
Source xsl = new StreamSource("template.xsl");
Transformer transformer = factory.newTransformer(xsl);

// code part 2
Source src  = new StreamSource(in);
Result res = new StreamResult(out);
transformer.transform(src, res);
```

Options (select 1):

1. Leave code part 1 common, and fork multiple threads which will run code part 2.
2. Fork multiple threads with code part 1 and 2.
3. Leave code part 1 common, but create `Templates` object from the factory instead of `Transformer` object, and fork multiple threads which will create `Transformer` object from `Templates` object and then run code part 2.
4. Leave code part 1 common, but remove code which creates `Transformer` object, fork multiple threads which will create `Transformer` object from factory and then run code part 2.
5. None of these.

Answer:

Correct option is 3

Option 1. This option is wrong. Neither `TransformerFactory` nor `Transformer` is guaranteed to be thread-safe. You can not use one common `Transformer` object simultaneously across multiple threads.

With this approach race condition may happen.

Option 2. This option is wrong. If your program is multi-threaded, the simplest solution is just to give each separate thread its own `TransformerFactory` and `Transformer` objects. However, this can be expensive, especially if you frequently reuse the same large stylesheet, since it will need to be read

from disk or the network and parsed every time you create a new `Transformer` object. There is also likely to be some overhead in building the processor's internal representation of an XSLT stylesheet from the parsed XML tree.

With this approach race condition will not happen, but performance will not be improved too.

Option 3. This option is correct.

The best approach is to ask the `TransformerFactory` to build a `Templates` object instead. The `Templates` class represents the parsed stylesheet. You can then ask the `Templates` class to give you as many separate `Transformer` objects as you need, each of which can be created very quickly by copying the processor's in-memory data structures rather than by reparsing the entire stylesheet from disk or the network. The `Templates` class itself can be safely used across multiple threads.

Follwoing code demonstrates this approach:

```
// code part 1 (common)
TransformerFactory factory = TransformerFactory.newInstance();
Source xsl = new StreamSource("template.xsl");
Templates templates = factory.newTemplates(xsl);

// code part 2 (separate thread)
Transformer transformer = templates.newTransformer();
Source src  = new StreamSource(in);
Result res = new StreamResult(out);
stylesheet.transform(src, res);
```

Since the thread-unsafe `Transformer` object is local to the thread, references to it don't escape into other threads. This prevents the `transform()` method from being called concurrently. The `Templates` object may be shared among multiple threads. However, it is thread safe so this isn't a problem. Furthermore, all the time-consuming work is done when the `Templates` object is created. Calling `templates.newTransformer()` is very quick by comparison.

This technique is particularly important in server environments where the transform may be applied to thousands of different input documents with potentially dozens being processed in parallel in separate threads at the same time.

Option 4. This option is wrong. Neither `TransformerFactory` nor `Transformer` is guaranteed to be thread-safe. You can not use one common `TransformerFactory` object simultaneously across multiple threads.

With this approach race condition may happen.

Sources:

Processing XML with Java: A Guide to SAX, DOM, JDOM, JAXP, and TrAX - [`http://my.safaribooksonline.com/book/xml/0201771861`]

JAXP Tutorial [Extensible Stylesheet Language Transformations] - `http://java.sun.com/webservices/reference/tutorials/jaxp/html/xslt.html`

Question A090209

Which statements are true about SAX parser?

Options (select 2):

1. It is useful when developer wants only a small subset of information contained in XML document.
2. It is memory intensive.
3. It is not CPU intensive.

4. It provides random access to XML document.
5. It can be used for XML document generation.

Answer:

Correct options are 1 and 3.

Option 1. This option is correct.

SAX is fast and efficient, but its event model makes it most useful for such state-independent filtering. For example, a SAX parser calls one method in your application when an element tag is encountered and calls a different method when text is found. If the processing you're doing is state-independent (meaning that it does not depend on the elements have come before), then SAX works fine.

On the other hand, for state-dependent processing, where the program needs to do one thing with the data under element `A` but something different with the data under element `B`, then a pull parser such as the Streaming API for XML (StAX) would be a better choice. With a pull parser, you get the next node, whatever it happens to be, at any point in the code that you ask for it. So it's easy to vary the way you process text (for example), because you can process it multiple places in the program.

Option 2. This option is wrong.

SAX requires much less memory than DOM, because SAX does not construct an internal representation (tree structure) of the XML data, as a DOM does. Instead, SAX simply sends data to the application as it is read; your application can then do whatever it wants to do with the data it sees.

Option 3. This option is correct.

Option 4. This option is wrong.

SAX API acts like a serial I/O stream. You see the data as it streams in, but you can't go back to an earlier position or leap ahead to a different position. In general, SAX parsers work well when you simply want to read data and have the application act on it.

Option 5. This option is wrong.

When you need to modify an XML structure or generate new XML document, an in-memory structure makes more sense. DOM is one such model.

Source:

The Java EE Tutorial [Simple API for XML] - [`http://download.oracle.com/javaee/1.4/tutorial/doc/JAXPSAX.html`]

9.3. Given an XML schema for a document style Web service create a WSDL file that describes the service and generate a service implementation.

blah-blah

9.4. Given a set of requirements, create code to create an XML-based, document style, Web service using the JAX-WS APIs.

blah-blah

9.5. Implement a SOAP logging mechanism for testing and debugging a Web service application using Java EE Web Service APIs.

Question A090501

Which statements are true about JAX-WS handler chain?

Options (select 3):

1. It can be defined declaratively.
2. It can NOT be defined declaratively.
3. It can be defined programmatically.
4. It can NOT be defined programmatically.
5. Handlers of any type are invoked in the order they defined in handler chain.
6. Handlers of the same type are invoked in the order they defined in handler chain.

Answer:

Correct options are 1, 3 and 6.

JAX-WS provides a handler framework that allows application code to inspect and manipulate outgoing and incoming SOAP messages. A handler can be injected into the framework in two steps:

1. The first step is to create a handler class, which implements the `Handler` interface in the `javax.xml.ws.handler` package. JAX-WS provides two `Handler` subinterfaces, `LogicalHandler` and `SOAPHandler`. As the names suggest, the `LogicalHandler` is protocol-neutral but the `SOAPHandler` is SOAP-specific. A `LogicalHandler` has access only to the message payload in the SOAP body, whereas a `SOAPHandler` has access to the entire SOAP message, including any optional headers and attachments.
2. The second step is to place a handler within a handler chain. This is typically done through a configuration file, although handlers also can be managed through code.

The handler chain for a binding is constructed by starting with the handler chain as returned by the `HandlerResolver` for the service in use and sorting its elements so that all logical handlers precede all protocol handlers. In performing this operation, the order of handlers of any given type (logical or protocol) in the original chain is maintained.

Option 2 is wrong. Below is an example of deployment file for a handler chain:

```
<?xml version="1.0" encoding="UTF-8"?>
<handler-chains xmlns="http://java.sun.com/xml/ns/javaee">
    <handler-chain>
        <handler>
            <handler-name>ServerLogicalHandler</handler-name>
            <handler-class>by.iba.ws.ServerLogicalHandler</handler-class>
        </handler>
        <handler>
            <handler-name>ServerSOAPHandler</handler-name>
            <handler-class>by.iba.ws.ServerSOAPHandler</handler-class>
        </handler>
    </handler-chain>
</handler-chains>
```

Option 4 is wrong, because handler chain can be configured programmatically (on the client only):

```
class WSClient {
    public static void main(String[ ] args) {
        MyService service = new MyService();
        service.setHandlerResolver(new ClientHandlerResolver());
        ...
    }
}
```

```
class ClientHandlerResolver implements HandlerResolver {
    public List<Handler> getHandlerChain(PortInfo info) {
        List<Handler> chain = new ArrayList<Handler>();
        chain.add(new ClientLogicalHandler());
        chain.add(new ClientSOAPHandler());
        return chain;
    }
}
```

Option 5 is wrong, because all logical handlers precede all protocol handlers.

Sources:

Handler example using JAX-WS 2.0 - [http://blogs.sun.com/sdimilla/entry/implementing_handlers_using_jaxws_2]

The Java API for XML-Based Web Services (JAX-WS) 2.0 (Chapter 9) - [http://www.jcp.org/en/jsr/detail?id=224]

9.6. Given a set of requirements, create code to handle system and service exceptions and faults received by a Web Services client.

blah-blah

Appendix 10. Web Services Interoperability Technologies

10.1. Describe WSIT, the features of each WSIT technology and the standards that WSIT.

Question A100101

Which of the following security technologies are included in the Metro Web Service stack?

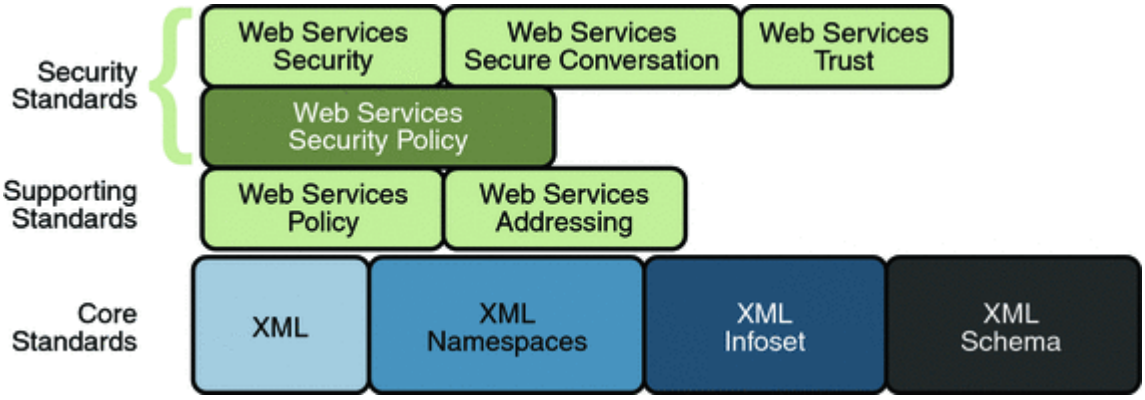
Options (select 3):

1. XACML
2. WS-SecureConversation
3. WS-Trust
4. WS-SecurityPolicy
5. WS-Addressing

Answer:

Correct options are 2, 3, and 4.

The figure below shows Metro specifications implemented to secure communication between two Web Service endpoints and across intermediate endpoints:



In addition to the Core XML specifications, the security feature is implemented using the following specifications:

- Web Services Security

  This specification defines a standard set of SOAP extensions that can be used when building secure web services to implement message content integrity and confidentiality. The implementation provides message content integrity and confidentiality even when communication traverses intermediate nodes, thus overcoming a short coming of SSL. The implementation can be used within a wide variety of security models including PKI, Kerberos, and SSL and provides support for multiple security token formats, multiple trust domains, multiple signature formats, and multiple encryption technologies.

- Web Services Policy

  This specification provides a flexible and extensible grammar for expressing the capabilities, requirements, and general characteristics of a web service. It provides a framework and a model for the expression of these properties as policies and is a building block for Web Services Security policy.

- Web Services Trust

  This specification supports the following capabilities in a standardized way:

  - Defines extensions to Web Services Security that provide methods for issuing, renewing, and validating security tokens used by Web services security.
  - Establishes, assesses the presence of, and brokers trust relationships.

- Web Services Secure Conversation

  This specification defines a standardized way to provide better message-level security and efficiency in multiple-message exchanges. The Metro implementation provides basic mechanisms on top of which secure messaging semantics can be defined for multiple-message exchanges and allows for contexts to be established along with more efficient keys or new key material. This approach increases the overall performance and security of the subsequent exchanges. While the Web Services Security specification, described above, focuses on the message authentication model, it does leave openings for several forms of attacks. The Secure Conversation authentication specification defines a standardized way to authenticate a series of messages, thereby addressing the short comings of Web Services Security. With the Web Services Security Conversation model, the security context is defined as a new Web Services security token type that is obtained using a binding of Web Services Trust.

- Web Services Security Policy

  This specification defines a standard set of patterns or sets of assertions that represent common ways to describe how messages are secured on a communications path. The Metro implementation allows flexibility in terms of tokens, cryptography, and mechanisms used, including leveraging transport security, but is specific enough to ensure interoperability based on assertion matching by web service clients and web services providers.

Option 1 is wrong, XACML stands for eXtensible Access Control Markup Language. It is a declarative access control policy language implemented in XML and a processing model, describing how to interpret the policies.

Option 5 is wrong. Web Services Addressing belongs to Message Optimization Specifications. This specification defines a endpoint reference representation. A web service endpoint is an entity, processor, or resource that can be referenced and to which web services messages can be addressed. Endpoint references convey the information needed to address a web service endpoint. The specification defines two constructs: message addressing properties and endpoint references, that normalize the information typically provided by transport protocols and messaging systems in a way that is independent of any particular transport or messaging system. This is accomplished by defining XML tags for including web service addresses in the SOAP message, instead of the HTTP header. The implementation of these features enables messaging systems to support message transmission in a transport-neutral manner through networks that include processing nodes such as endpoint managers, firewalls, and gateways.

Sources:

Metro Web Services Technologies - [http://www.oracle.com/technetwork/java/index-jsp-137051.html]

Metro Specifications - [http://metro.java.net/guide/Metro_Specifications.html]

10.2. Describe how to create a WSIT client from a Web Service Description Language (WSDL) file.

blah

10.3. Describe how to configure Web Service providers and clients to use message optimization.

blah

10.4. Create a Microsoft Windows Communication Foundation (WCF) client that accesses a Java Web Service.

Question A100401

A developer creates .NET 3.0 platform client for already deployed stock quotes Web Service written in Java. The stock quotes Web Service provides access to its WSDL at the following location:

```
http://java.boot.by/stockquote?wsdl
```

How can the developer generate Web Service client proxy class in C#?

Options (select 1):

1.
```
svc.exe /config:StockClient.exe.config http://java.boot.by/stockquote?wsdl
```

2.
```
wsimport.exe /config:StockClient.exe.config http://java.boot.by/stockquote?wsdl
```

3.
```
svcutil.exe /config:StockClient.exe.config http://java.boot.by/stockquote?wsdl
```

4.
```
csc.exe /config:StockClient.exe.config http://java.boot.by/stockquote?wsdl
```

Answer:

Correct option is 3.

The ServiceModel Metadata Utility tool (`svcutil.exe`) is used to generate service model code from metadata documents and metadata documents from service model code.

`svcutil.exe` can generate code for service contracts, clients and data types from metadata documents. These metadata documents can be on a durable storage, or be retrieved online. Online retrieval follows either the WS-Metadata Exchange protocol or the DISCO protocol.

Usage:

```
svcutil.exe [/t:code] <metadataDocumentPath>* | <url>* | <epr>
```

where:

`url` - The URL to a service endpoint that provides metadata or to a metadata document (WSDL or XSD) hosted online.

`/config:<configFile>` - Specifies the filename for the generated configuration file. Default: `output.config`

Option 1 is wrong, because `svc.exe` does not exist in .NET platform.

Option 2 is wrong, because `wsimport.exe` does not exist in .NET platform.

NOTE: do not be confused by `wsimport` command for JAX-WS applications.

The `wsimport` command-line tool processes an existing Web Services Description Language (WSDL) file and generates the required portable artifacts for developing Java API for XML-Based Web Services (JAX-WS) Web service applications.

The `wsimport` command-line tool supports the top-down approach to developing JAX-WS Web services. When you start with an existing WSDL file, use the `wsimport` command-line tool to generate the required JAX-WS portable artifacts.

The `wsimport` tool reads an existing WSDL file and generates the following portable artifacts:

- Service Endpoint Interface (SEI) - The SEI is the annotated Java representation of the WSDL file for the Web service. This interface is used for implementing JavaBeans endpoints or creating dynamic proxy client instances.
- `javax.xml.ws.Service` extension class - This is a generated class that extends the `javax.xml.ws.Service` class. This class is used to configure and create both dynamic proxy and dispatch instances.
- required data beans, including any Java Architecture for XML Binding (JAXB) beans that are required to model the Web service data.

Option 4 is wrong, because `csc.exe` is a command line compiler for Microsoft C#. The `csc.exe` is used with the .NET SDK.

Sources:

The WSIT Tutorial [Creating a WCF Client] - [`http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/reference/tutorials/wsit/doc/WCFClient2.html`]

JAX-WS 2.0 wsimport - [`http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/docs/2.0/jaxws/wsimport.html`]

MSDN [ServiceModel Metadata Utility] - `http://msdn.microsoft.com/en-us/library`

/aa347733.aspx

10.5. Describes the best practices for production and consumption of data for interoperability between WCF Web Services and Java Web Service clients or between Java WebServices and WCF Web Service clients.

Question A100501

A Java EE 6.0 Web Service was designed for a stock broker company. A developer used Transfer Object to get stock data:

```
public class StockItem {

    public String symbol;
    public Double wholeSalePrice;
    public double retailPrice;
    ...
}
```

Which code fragment demonstrates the correct stock item class in C#, when the Web Service is accessed from .NET platform ?

Options (select 1):

1.
```
public partial class StockItem
{
    public string symbol;
    private double wholeSalePrice;
    private double retailPrice;
    ...
}
```

2.
```
public partial class StockItem
{
    public String symbol;
    private Decimal wholeSalePrice;
    private double retailPrice;
    ...
}
```

3.
```
public partial class StockItem
{
    public char[] symbol;
    private Double wholeSalePrice;
    private double retailPrice;
    ...
}
```

4.
```
public partial class StockItem
{
    public string symbol;
    private decimal wholeSalePrice;
    private double retailPrice;
    ...
}
```

5.
```
public partial class StockItem
{
    public string symbol;
    private double wholeSalePrice;
    private decimal retailPrice;
```

```
    ...
}
```

Answer:

Correct option is 1.

Below is the table showing some Java types and mapped C# aliases with CLR types:

Java type	C# alias	CLR type
java.lang.Double	double	System.Double
double	double	System.Double
int	int	System.Int32
java.math.BigDecimal	decimal	System.Decimal
java.lang.String	string	System.String

Options 2, 4 and 5 are wrong, because in C# `decimal` (`System.Decimal`) type matches the `java.math.BigDecimal` Java type.

Option 4 is wrong because `java.lang.String` type matches the `string` (`System.String`) type in C#, not `char[]`.

C# defines a number of aliases for intrinsic Common Language Runtime (CLR) types.

`string` is a type in C#.

`System.String` is a type in the CLR.

When you use C# together with the CLR `string` will be mapped to `System.String`.

They may be used interchangably, and even mixed together, e.g.:

```
string x = new System.String(' ', 10);
```

With the said above, the following example would be valid also:

```
using System;
public partial class StockItem
{
    public String symbol;
    private Double wholeSalePrice;
    private Double retailPrice;
    ...
}
```

Sources:

The WSIT Tutorial [Data Contracts] - [`http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/reference/tutorials/wsit/doc/DataBinding2.html`]

C# Language Reference - string (C# Reference) - [`http://msdn.microsoft.com/en-us/library/362314fe(VS.80).aspx`]

C# Language Reference - Built-In Types Table (C# Reference) - [`http://msdn.microsoft.com/en-us/library/ya5y69ds.aspx`]

Question 100502

You are developing interoperable Web Service using "Start from Java" approach. Which code snippet

demonstrates correct WSDL type schema definition?

```
public enum USState {
    AL, AK, AS
}
```

Options (select 1):

1.
```
<xs:simpleType name="usState">
  <xs:restriction base="xs:enum">
    <xs:string value="AL" />
    <xs:string value="AK" />
    <xs:string value="AS" />
  </xs:restriction>
</xs:simpleType>
```

2.
```
<xs:simpleType name="usState">
  <xs:restriction base="xs:enumeration">
    <xs:pattern value="AL"/>
    <xs:pattern value="AK"/>
    <xs:pattern value="AS"/>
  </xs:restriction>
</xs:simpleType>
```

3.
```
<xs:simpleType name="usState">
  <xs:restriction base="xs:string">
    <xs:enumeration value="AL" />
    <xs:enumeration value="AK" />
    <xs:enumeration value="AS" />
  </xs:restriction>
</xs:simpleType>
```

4.
```
<xs:simpleType name="usState">
  <xs:restriction base="xs:string">
    <xs:enum value="AL" />
    <xs:enum value="AK" />
    <xs:enum value="AS" />
  </xs:restriction>
</xs:simpleType>
```

Answer:

Correct option is 3.

A Java `enum` type maps to an `xs:simpleType` XML schema type constrained by `enumeration` facets.

`xs:restriction` defines constraints on a simple content or complex content definition. `base` attribute is the name of a built-in data type, simple type, or complex type. If a complex type, this type must be of one of the following: built-in data type (excluding `xs:boolean`), simple type, or simple content.

Options 1 and 2 are wrong, because `enum` and `enumeration` types do not exist in XSD.

Built-in XML data types list (incomplete) includes:

```
string
boolean (enumeration facet can not appear on this type)
decimal
float
double
```

Option 4 is wrong, because `enum` facet name does not exist in XML Schema.

List (incomplete) of XML Schema facets includes:

```
length
minLength
maxLength
pattern
enumeration
maxInclusive
maxExclusive
minInclusive
minExclusive
```

Sources:

MSDN [Data Type Facets] - [http://msdn.microsoft.com/en-us/library/ms256149(VS.85).aspx]

MSDN [Primitive XML Data Types] - [http://msdn.microsoft.com/en-us/library/ms256220(VS.85).aspx]

The WSIT Tutorial [Data Contracts] - [http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/reference/tutorials/wsit/doc/DataBinding2.html]

Appendix 11. General Design and Architecture

11.1. Describe the characteristics of a service-oriented architecture and how Web Services fit this model.

Question A110101

What are the characteristics of Web Services?

Options (select 2):

1. Web Services reduce coupling.
2. Web Services increase reliability.
3. Web Services reduce network load.
4. Web Services increase speed.
5. Web Services reduce complexity.
6. Web Services increase interoperability.

Answer:

Correct options are 1 and 6.

Option 1 is correct.

Web Services are used to make application to be available to as many clients as possible, regardless

of the platform or technologies that the client is using. Web Services also provide a good interface for separating and decoupling software systems.

Option 2 is wrong.

Reliability is the aspect of a Web service representing how well it maintains its service and service quality. Often, reliability is measured by the number of failures that occur in a given time period. Reliability may be more difficult to achieve because of the unreliable nature of HTTP; HTTP provides only best-effort delivery and does not guarantee packet or in-order delivery.

Option 3 is wrong.

Web Services do not reduce network load. Because every RPC call is packaged in XML SOAP message, this increases network load.

Option 4 is wrong.

Since in most cases HTTP protocol is used - speed is reduced comparing to regular RPC calls (CORBA or JRMI).

Option 5 is wrong.

Comlexity is not decreasing when you implement Web Services.

Option 6 is correct.

Perhaps the most important reason for the increased use of Web Services - the main force for their widespread adaptation - is that Web Services promote interoperability across different platforms, systems, and languages.

Unlike traditional distributed environments, Web Services emphasize interoperability. Web Services are independent of a particular programming language, whereas traditional environments tend to be bound to one language or another. Similarly, since they can be easily bound to different transport mechanisms, Web Services offer more flexibility in the choice of these mechanisms. Furthermore, unlike traditional environments, Web Services are often not bound to particular client or server frameworks. Overall, Web Services are better suited to a loosely coupled, coarse-grained set of relationships. Traditional RPC environments, on the other hand, do better with more tightly-coupled, fine-grained contracts. Relying on XML gives Web Services an additional advantage, since XML makes it possible to use documents across heterogeneous environments.

Sources:

Designing Web Services with the J2EE 1.4 Platform - [`http://java.sun.com/blueprints/guidelines/designing_webservices/`]

11.2. Given a scenario, design a Java EE Web Service using Web Services Design Patterns (Asynchronous Interaction, JMS Bridge, Web Service Cache, Web Serive Broker), and Best Practices.

Question A110201

You need to provide access to one or more services provided by heterogeneous applications, using XML and web protocols. What is the best approach?

Options (select 1):

1. Use a Service Facade to expose services using XML and web protocols.
2. Use a Web Service Broker to expose services using XML and web protocols.
3. Use a Mediator to expose services using XML and web protocols.
4. Use a Web Service Cache to expose services using XML and web protocols.

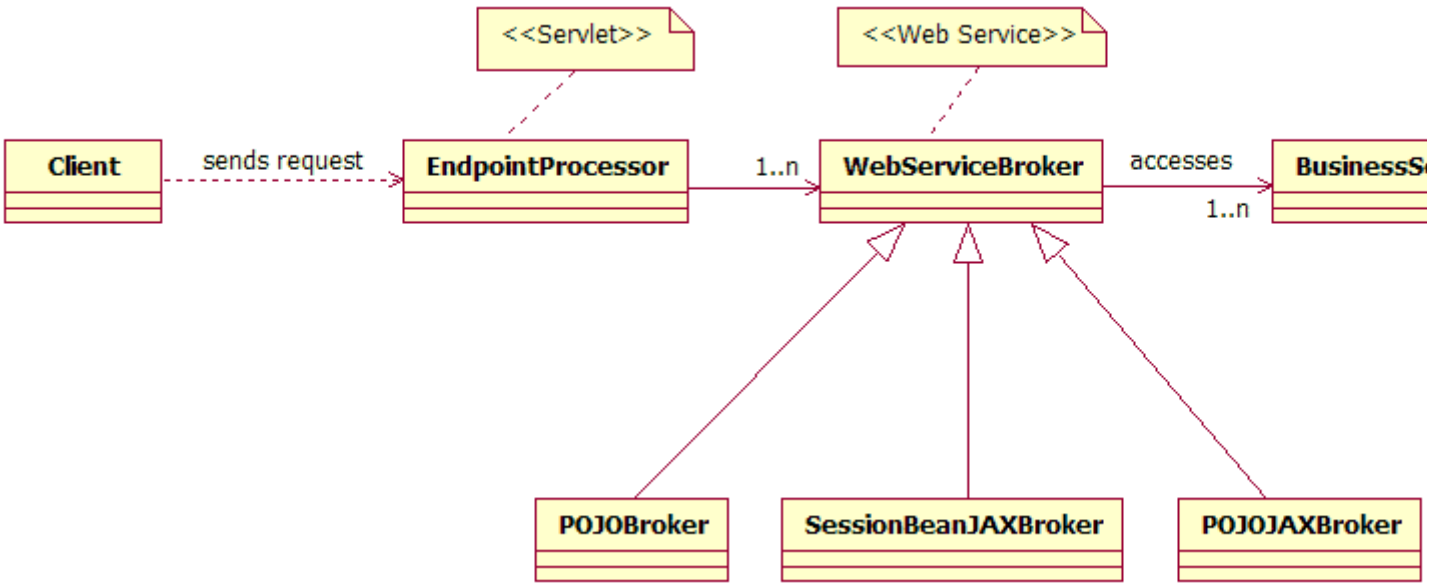
Answer:

Correct option is 2.

Use a Web Service Broker to expose and broker one or more services using XML and web protocols to enable integration of heterogeneous applications.

A Web Service Broker is a coarse-grained service exposed as a Web Service. It coordinates interactions among one or more services, aggregates responses and may demarcate and compensate transactions. A Web Service Broker is exposed either using an RPC (remote procedure call)-style interface such as WSDL, or a messaging interface. For RPC style interface, you can use a JEE Stateless Session Bean Web Service Endpoint or a JAX-WS implementation object as a Web Service Endpoint. Both of these are exposed using WSDL.

Web Services are used to make application to be available to as many clients as possible, regardless of the platform or technologies that the client is using. Web Services also provide a good interface for separating and decoupling software systems.



Option 1 is wrong.

JEE applications expose coarse-grained business services using Service Facade. However, these services might be too fine grained to expose as a Web Service or may not be designed to be exposed outside of their application.

Another factor is that enterprise services run on various platforms and are implemented in a variety of languages. This heterogeneity often introduces incompatibilities, increasing the amount of complexity involved in seamlessly integrating these systems, such as with JEE and .NET applications. This means that whether you have existing JEE services, .NET services, or legacy services, you rarely wish to expose each one as a Web Service. Even with coarse-grained JEE services, businesses typically want to expose only a limited set of the JEE service methods for Web Service access. Additionally, systems often have a requirement to coordinate and aggregate several existing services.

Option 3 is wrong.

Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.

Sources:

Core J2EE Patterns 2nd edition - [`http://www.corej2eepatterns.com/index.htm`]

Web Service Broker pattern - [`http://www.corej2eepatterns.com/Patterns2ndEd`

/WebServiceBroker.htm]

Question A110202

Which statements are true about JMS Bridge?

Options (select 2):

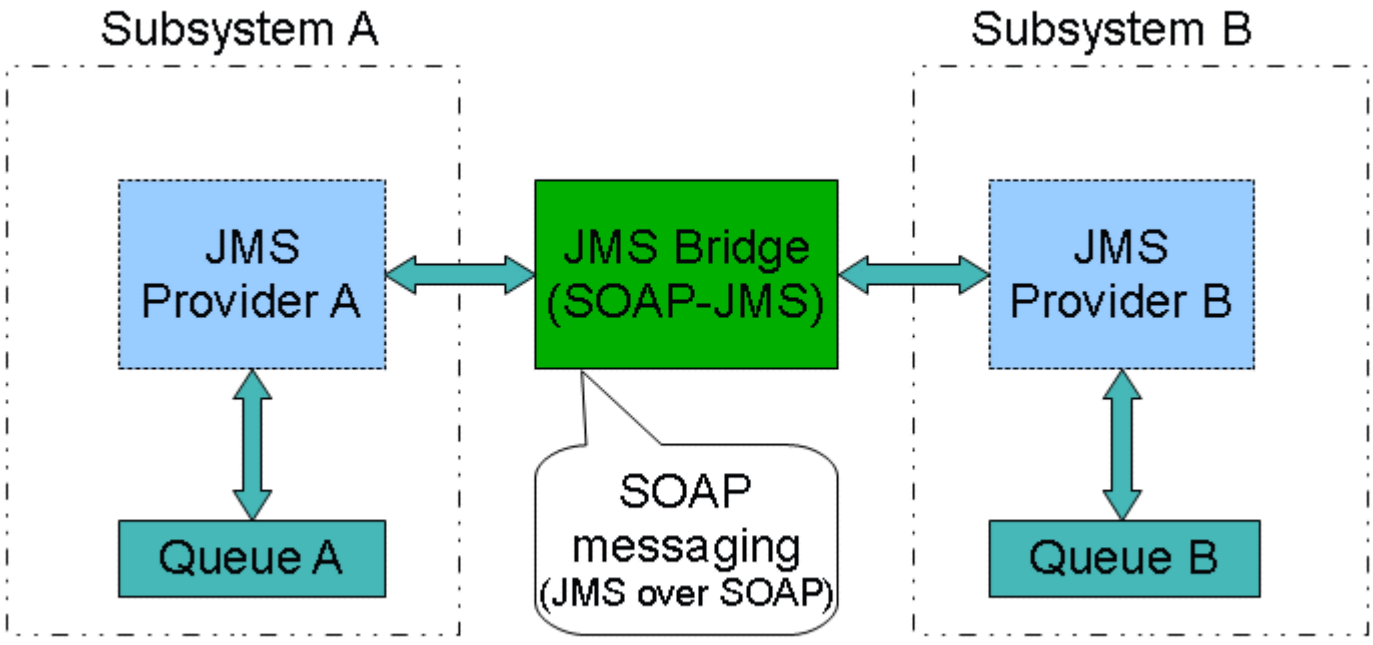
1. It increases overhead.
2. It reduces overhead.
3. It provides a single cross-platform JMS implementation.
4. It can guarantee messaging between any two JMS implementations.

Answer:

Correct options are 1 and 4.

The main disadvantage of the JMS Bridge pattern is that there is overhead associated with the act of relaying messages across JMS implementations. Presence of a JMS bridge in a design means there is overhead corresponding to the XML encoding, transmission, and XML decoding associated with the relaying of the message.

There is no need to develop or implement a vendor-specific adapter to bridge two underlying middleware vendors: a JMS Bridge is vendor and JMS-independent, so it can guarantee messaging between any two JMS implementations.



Option 2 is wrong. JMS Bridge increases overhead.

Option 3 is wrong. JMS Bridge does not provide a single cross-platform JMS imlementation.

Source:

Developing Web Services Using Java Technology, Java EE 6 - [http://education.oracle.com /pls/web_prod-plq-dad/db_pages.getcoursedesc?dc=D65185GC10&p_org_id=15947&lang=US]

11.3. Describe how to handle the various types of return values, faults, errors, and exceptions that can occur during a Web service interaction.

Question A110301

What happens when `SOAPHandler` on service endpoint throws `javax.xml.ws.soap.SOAPFaultException` exception in `handleMessage(...)` method?

Options (select 2):

1. This indicates a fault. `SOAPHandler` implementation class has the responsibility of setting the SOAP fault in the SOAP message.
2. This indicates a fault. JAX-WS Runtime has the responsibility of setting the SOAP fault in the SOAP message.
3. This does not indicate a fault.
4. Further processing of request handlers in this handler chain is not terminated. Handler chain invokes the `handleFault(...)` method on handlers registered in the handler chain, beginning with the next handler and going forward in execution.
5. Further processing of request handlers in this handler chain is terminated. Handler chain invokes the `handleFault(...)` method on handlers registered in the handler chain, beginning with the next handler in reverse direction and going backward in execution.
6. Further processing of request handlers in this handler chain is terminated. Handler chain invokes the `handleFault(...)` method on handlers registered in the handler chain, beginning with the handler instance that threw the exception and going backward in execution.
7. Handler chain continues execution of `SOAPHandler`s in forward order with original SOAP message.

Answer:

Correct options are 2 and 5.

The `handleMessage` method is called for normal message processing. It can throw `ProtocolException` or a subclass (e.g. `SOAPFaultException`). This indicates that normal message processing should stop. Subsequent actions depend on whether the MEP in use requires a response to the message currently being processed or not:

- If response is required: Normal message processing stops, fault message processing starts. The message direction is reversed, if the message is not already a fault message then it is replaced with a fault message, and the runtime invokes `handleFault` on the next (means the next handler taking into account the message direction reversal) handler or dispatches the message if there are no further handlers.
- If no response is requied: Normal message processing stops, `close` is called on each previously invoked handler in the chain, the exception is dispatched.

The binding is responsible for catching runtime exceptions thrown by handlers and respecting any resulting message direction and message type change. Outbound exceptions are converted to protocol fault messages and dispatched using whatever means the protocol binding uses for communication.

R1028 When a fault is generated by a RECEIVER, further processing SHOULD NOT be performed on the SOAP envelope aside from that which is necessary to rollback, or compensate for, any effects of processing the envelope prior to the generation of the fault.

R1029 Where the normal outcome of processing a SOAP envelope would have resulted in the transmission of a SOAP response, but rather a fault is generated instead, a RECEIVER MUST transmit a fault in place of the response.

R1030 A RECEIVER that generates a fault SHOULD notify the end user that a fault has been generated when practical, by whatever means is deemed appropriate to the circumstance.

Sources:

Basic Profile Version 1.1 - [`http://www.ws-i.org/Profiles/BasicProfile-1.1.html`]

JAX-WS 2.2 Specification - `http://www.jcp.org/en/jsr/detail?id=224`

Question A110302

Web Service received a SOAP message which is not well formed. As a result, a SOAP fault was returned. What is the fault code returned by the server?

Options (select 1):

1. `soap:Server`
2. `soap:Client`
3. `soap:Sender`
4. `soap:Receiver`
5. None of the above.

Answer:

Correct option is 2.

Option 1 is wrong. The `Server` fault code indicates that the message could not be processed for reasons not directly attributable to the contents of the message itself but rather to the processing of the message. For example, processing could include communicating with an upstream processor, which did not respond. The message may succeed at a later point in time.

Option 2 is correct. The `Client` fault code indicates that the message was incorrectly formed or did not contain the appropriate information in order to succeed. For example, the message could lack the proper authentication or payment information. It is generally an indication that the message should not be resent without change.

Two other fault codes defined in SOAP 1.1 specification:

The `VersionMismatch` fault code means that the processing party found an invalid namespace for the SOAP `Envelope` element.

The `MustUnderstand` fault code means that an immediate child element of the SOAP `Header` element that was either not understood or not obeyed by the processing party contained a SOAP `mustUnderstand` attribute with a value of "1".

Option 3 is wrong. Although, the `Sender` fault code indicates a problem with SOAP message (e.g. not well formed), but it belongs to SOAP 1.2 specification, while OCE WSD 6 exam covers SOAP 1.1 specification.

Moreover, the WS-I BP 1.1 mandates the use of SOAP 1.1: An ENVELOPE MUST conform to the structure specified in SOAP 1.1 Section 4, "SOAP Envelope".

Option 4 is wrong. The `Receiver` faul code belongs to SOAP 1.2 and indicates that SOAP message could not be processed because of processing error, not of a problem with the message. Again, OCE WSD 6 exam covers SOAP 1.1, not SOAP 1.2.

Sources:

Basic Profile Version 1.1 - `http://www.ws-i.org/Profiles/BasicProfile-1.1.html`

SOAP 1.1 Specification - `http://www.w3.org/TR/2000/NOTE-SOAP-20000508/`

11.4. Describe the role that Web Services play when integrating data, application

functions, or business processes in a Java EE application.

blah

Appendix 12. Endpoint Design and Architecture

12.1. Given a scenario, design Web Service applications using information models that are either procedure-style or document-style.

blah-blah

12.2. Describe the function of the service interaction and processing layers in a Web Service.

Question A120201ZZZZ

What is true about Service Implementation Layers?

Options (select 3):

1. The Service Interaction Layer consists of the endpoint interface that the service exposes to clients and through which it receives client requests.
2. Sometimes multiple layers make a service unnecessarily complicated and, in these cases, it may be simpler to design the service as one layer.
3. The Service Processing Layer consists of the endpoint interface that the service exposes to clients and through which it receives client requests.
4. The Service Processing Layer holds all business logic used to process client requests.
5. Web Service Implementation is always separated in an Interaction Layer and a Processing Layer.
6. The Service Interaction Layer holds all business logic used to process client requests.

Answer:

Correct options are 1, 2, and 4.

Option 1 is correct. The Service Interaction Layer consists of the endpoint interface that the service exposes to clients and through which it receives client requests. The interaction layer also includes the logic for how the service delegates the requests to business logic and formulates responses. When it receives requests from clients, the interaction layer performs any required preprocessing before delegating requests to the business logic. When the business logic processing completes, the interaction layer sends back the response to the client. The interaction layer may be different for those scenarios where the service expects to receive XML documents from clients but the business logic deals with objects. In these cases, you map the XML documents to equivalent object representations in the interaction layer before delegating the request to the business logic.

Option 2 is correct. Although there are advantages to viewing a service in terms of Interaction and Processing Layers, a Web Service may opt to merge these two layers into a single layer. There are times when multiple layers make a service unnecessarily complicated and, in these cases, it may be simpler to design the service as one layer. Typically, this happens in scenarios where the logic in either layer is too small to merit a separate layer.

Option 3 is wrong. Processing Layer holds business logic.

Option 4 is correct. The Service Processing Payer holds all business logic used to process client requests. It is also responsible for integrating with EISs and other Web services. In the case of existing applications adding a Web Service interoperable interface, the existing application itself typically forms the service processing layer.

Option 5 is wrong. Service implementation may consist of one layer.

Option 6 is wrong. Interaction Layer is the starting point of a client's interaction with the service. It does not hold business logic.

Sources:

Designing Web Services with the J2EE 1.4 Platform - [`http://java.sun.com/blueprints/guidelines/designing_webservices/`]

Question A120202

What Layer of Web Servise Implementation is the best place to validate incoming XML document?

Options (select 1):

1. Service Processing Layer.
2. Service Interaction Layer.
3. Service Parsing Layer.
4. Service Validation Layer.

Answer:

Correct option is 2:

The Interaction Layer, through the Service Endpoint, receives client requests. The platform maps the incoming client requests, which are in the form of SOAP messages, to method calls present in the Web service interface. Before delegating these incoming client requests to the Web Service business logic, you should perform any required security validation, transformation of parameters, and other required pre-processing of parameters.

Web Service calls are basically method calls whose parameters are passed as either Java objects, XML documents (`javax.xml.transform.Source` objects), or even SOAP document fragments (`javax.xml.soap.SOAPElement` objects). For parameters that are passed as Java objects (such as `String`, `int`, JAX-WS value types, and so forth), do the application-specific parameter validation and map the incoming objects to domain-specific objects in the Interaction Layer before delegating the request to the Processing Layer. You may have to undertake additional steps to handle XML documents that are passed as parameters. These steps, which are best performed in the Interaction Layer of your service, are as follows:

1. The Service Endpoint should validate the incoming XML document against its schema.
2. When the service's Processing Layer and business logic are designed to deal with XML documents, you should transform the XML document to an internally supported schema, if the schema for the XML document differs from the internal schema, before passing the document to the processing layer.
3. When the Service Implementation deals with objects but the interface receives XML documents, then, as part of the Interaction Layer, map the incoming XML documents to domain objects before delegating the request to the processing layer.

It is important that these three steps - validation of incoming parameters or XML documents, translation of XML documents to internal supported schemas, and mapping documents to domain objects - be performed as close to the service endpoint as possible, and certainly in the service interaction layer. A design such as this also helps to catch errors early, and thus avoids unnecessary calls and round-trips to the processing layer.

Option 1 is wrong. Processing Layer holds business logic. It deals with domain-specific business objects which usually created from incoming XML in Interaction Layer.

Options 3 and 4 are wrong. Layer names are invalid.

Sources:

Designing Web Services with the J2EE 1.4 Platform - [http://java.sun.com/blueprints/guidelines/designing_webservices/]

12.3. Design a Web Service for an asynchronous, document-style process and describe how to refactor a Web Service from a synchronous to an asynchronous model.

Question A120301

You are hired by Snorcle, Inc. to refactor an existing request-response synchronous Web Service for better performance. The new Web Service should receive order requests from customers and proceed to batch-processing. The client should not wait for the response because it may take a significant amount of time, the results are sent back later via SMTP. Which Java EE technology can can be added to facilitate the refactoring?

Options (select 1):

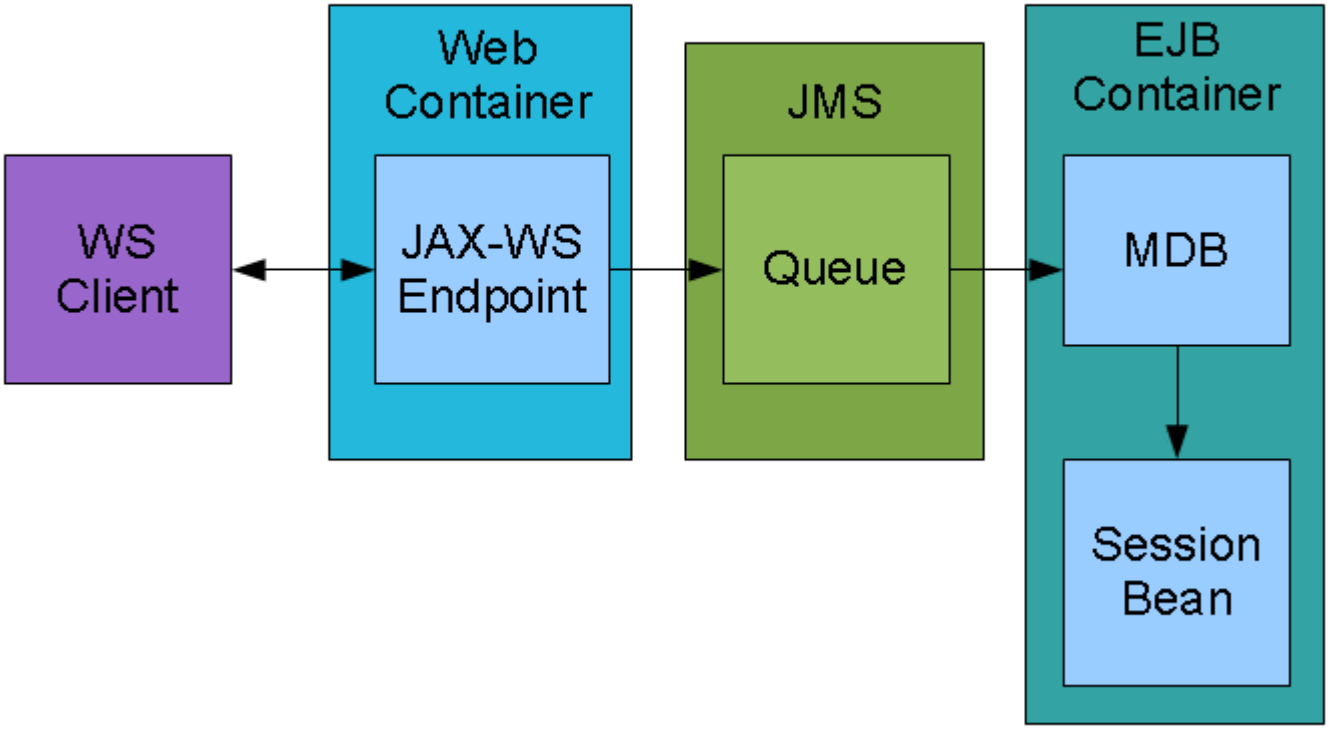
1. JDBC
2. JMS
3. RMI
4. JAXB

Answer:

Correct option is 2.

With asynchronous services, the client invokes the service but does not wait for the response. Often, with these services, the client does not want to wait for the response because it may take a significant amount of time for the service to process the request. The client can continue with some other processing rather than wait for the response. Later, when it does receive the response, it resumes whatever processing initiated the service request.

The figure below shows how an asynchronous Web Service might be architected. This figure shows one recommended approach for architecting a Web Service to achieve asynchronous communication in the JEE setting. In this architecture, the client sends a request to the JAX-WS servlet endpoint on the Web container. The servlet endpoint delegates the client's request to the appropriate business logic of the service. It does so by sending the request as a Java Message Service (JMS) message to a designated JMS Queue or Topic. The JMS layer (along with Message-Driven Beans) makes asynchronous communication possible.



Option 1 is wrong. The JDBC API is the industry standard for database-independent connectivity between the Java programming language and a wide range of databases. The JDBC API provides a call-level API for SQL-based database access. JDBC technology allows you to use the Java programming language to exploit "Write Once, Run Anywhere" capabilities for applications that require access to enterprise data.

Option 3 is wrong. The Java Remote Method Invocation (RMI) system allows an object running in one Java virtual machine to invoke methods on an object running in another Java virtual machine. RMI provides for remote communication between programs written in the Java programming language. Java RMI promotes synchronous invocation model.

Option 4 is wrong. Java Architecture for XML Binding (JAXB) allows Java developers to map Java classes to XML representations. JAXB provides two main features: the ability to marshal Java objects into XML and the inverse, i.e. to unmarshal XML back into Java objects. In other words, JAXB allows storing and retrieving data in memory in any XML format, without the need to implement a specific set of XML loading and saving routines for the program's class structure.

Source:

Using Web Services Effectively (Web Service Interaction Architectures) - [`http://java.sun.com/blueprints/webservices/using/webservbp3.html`]

Question A120302

You are hired by Snorcle, Inc. to refactor the Order Processing Center (OPC) application. The OPC has a Web Service invoker that sends a supplier purchase order as a request to the supplier application's Web Service endpoint. Currently the supplier application fulfills the request immediately, but due to business requirements changes, the supplier may take up to 14 days to perform fulfillment. Which design decisions should you implement?

Options (select 2):

1. Establish a correlation identifier between the purchase order message and the supplier response message.

2. Implement the "rpc-literal" style binding for supplier response message.
3. Have the OPC application provide Web Service and accept the response.
4. Use RESTful architectural style for OPC Web Service invoker.

Answer:

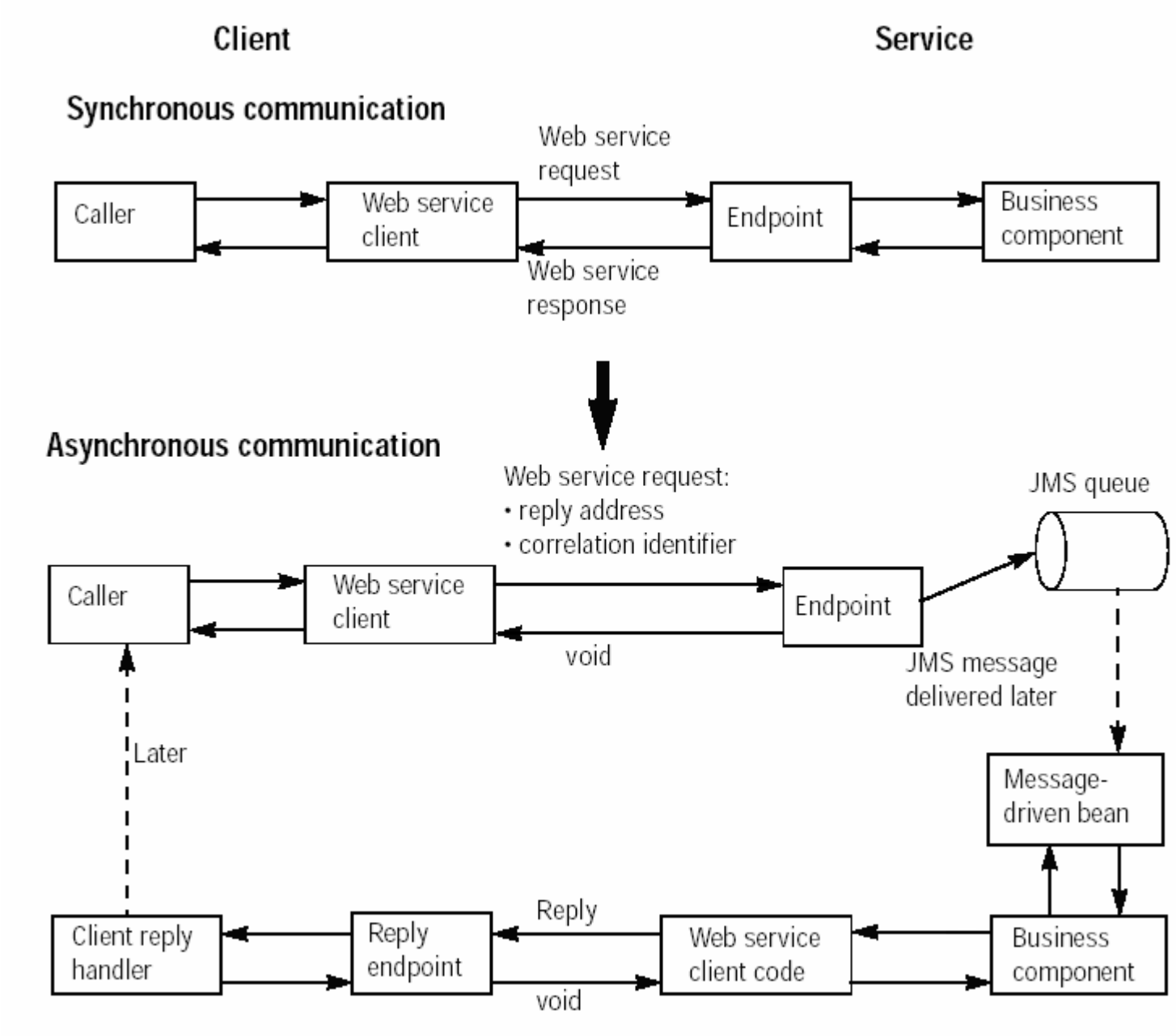
Correct options are 1 and 3.

When a component (OPC application) makes a synchronous call to another component, the calling component must wait for the receiving component to finish its processing. If the calling component (OPC application) instead makes the call asynchronously, the caller can proceed with its own work without waiting for the receiving component (supplier's application) to do its job.

Let's look at what needs to be done to convert a synchronous endpoint to an asynchronous endpoint:

- Change a synchronous request/reply interaction into a request that returns `void` as the immediate reply.
- Change the client code to send the message request and immediately return.
- Have the client establish a reply address for receiving the response and include this reply address in the request. By including a reply address, the service knows where to send the response message.
- A correlation identifier needs to be established between the request message and the service response message. Both the client and the service use this identifier to associate the request and the reply.
- Refactor the server endpoint to accept the request, send it to a JMS queue for processing later, and immediately return an acknowledgement to the client. The service retains the correlation identifier.
- Have the service processing layer use a message-driven bean to process the received request. When a response is ready, the service sends the response to the reply address. In addition to the results, the response message contains the same correlation identifier so that the service requestor can identify the request to which this response relates.
- Have the client accept the response at a Web service that it establishes at the reply address.

Converting from Synchronous to Asynchronous Communication:



Source:

Designing Web Services with the JEE 1.4 Platform [Web Service Communication Patterns] - [http://java.sun.com/blueprints/guidelines/designing_webservices/html/architecture5.html]

12.4. Describe how the characteristics, such as resource utilization, conversational capabilities, and operational modes, of the various types of Web service clients impact the design of a Web service or determine the type of client that might interact with a particular service.

Question A120401

You are developing a Web Service which must manage multiple clients in thread-safe manner. Also, the Web Service must manage distributed transactions. Which design would be the best choice?

Options (select 1):

1. Java Servlet as a Web Service endpoint.
2. Stateful Session bean as a Web Service endpoint.

3. Stateless Session bean as a Web Service endpoint.
4. POJO as a Web Service endpoint.
5. None of the above.

Answer:

Correct option is 3.

Java EE Application Server (JBoss, GlassFish, WebSphere) allows Web Services to be deployed in either WAR or EJB-JAR files.

In the first case, Application Server characterizes the implementation as servlet-based because, a `WSServlet` instance intercepts that Web Service requests and passes these on to the Web Service packaged in the WAR file:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="2.5"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema"
     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
     http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <display-name>HelloService</display-name>
    <listener>
        <listener-class>
            com.sun.xml.ws.transport.http.servlet.WSServletContextListener
        </listener-class>
    </listener>
    <servlet>
        <display-name>HelloService</display-name>
        <servlet-name>HelloService</servlet-name>
        <servlet-class>com.sun.xml.ws.transport.http.servlet.WSServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloService</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>
</web-app>
```

In the second case, the Web Service is a Stateless (or a Singleton) Session EJB that the EJB container manages. In the servlet implementation, the Web Container handles requests against the Web Service; in the EJB implementation, the EJB container handles requests against the Web Service. A given Web Service thus can be implemented in two different ways. At issue, then, are the tradeoffs between the two implementations of a Web Service.

The main advantage of the EJB implementation is that the EJB container provides more services than does the servlet container. For one thing, a Web Service implemented as an EJB is thereby thread-safe, whereas the operationally identical service implemented as a POJO is not thread-safe.

The EJB container also automatically wraps database operations in transactions, which remain transparent. The EJB container manages the persistence so that the application does not have to do so. In a servlet-based service, the application would take on these responsibilities.

Option 1 is wrong. Servlet-based endpoint is not thread-safe and does not support distributed transactions.

Option 2 is wrong. Only Stateless and Singleton Session Beans, as defined by the Enterprise JavaBeans specification, can be used to implement a Web Service to be deployed in the EJB container.

Option 4 is wrong. POJO-based endpoint does not support distributed transactions.

Sources:

JSR 109: Web Services for Java EE, Version 1.3 (Section 5.3.2.3 and 5.3.2.4) - [http://jcp.org/en/jsr/detail?id=109]