Java EE Web Services SOAP and REST API

Java EE Web Services SOAP and REST API

1. Common characteristics of a web service:
   - A text-based data interchange format (such as XML)
   - A client accessible service description
     (Web services use XML for data representation and have self-describing interfaces accessible to clients)15
2. An XML schema simpleType may contain:
   - restriction
   - enumeration
     (simpleType defines restrictions and enumerations, while sequence and choice are for complex types)6
3. The tags and attributes allowed in an XML document can be constrained by:
   - DTDs
   - XML Schemas
     (Both DTDs and XML Schemas are used to constrain XML document structure)6
4. The default accessor type used by JAXB to obtain the state of an object is:
   - Properties – getter and setter methods are used
     (JAXB uses JavaBean getter/setter properties by default)6
5. A request to a SOAP web service endpoint must always be transferred using the HTTP protocol:
   - False
     (SOAP can be transported over protocols other than HTTP, such as SMTP)[general knowledge]
6. The definitions element in a completed WSDL should contain which sequence of elements?
   - types, message, portType, binding, service
     (This is the standard element order in WSDL)[general knowledge]
7. Which HTTP status code range represents client errors?

- 4xx
  (4xx codes indicate client errors)[general knowledge]

8. Which HTTP methods are idempotent?
   - GET
   - PUT
   - DELETE
     (GET, PUT, DELETE are idempotent; POST is not)[general knowledge]
9. Which Jersey class represents a REST resource that is located at a URL?
   - WebResource
     (In Jersey client API, WebResource represents a REST resource)[general knowledge]
10. Public methods in an endpoint must be annotated with @WebMethod in order to be exposed to clients:
    - False
      (By default, public methods are exposed unless annotated with @WebMethod(exclude=true))[general knowledge]
11. When using a bottom-up development approach to create SOAP endpoints you create the WSDL file first:
    - False
      (Bottom-up starts from code to generate WSDL; top-down starts from WSDL)[general knowledge]
12. When using a top-down development approach to create SOAP endpoints the resulting method names are determined solely by the WSDL file:
    - True
      (Top-down approach generates code from WSDL, so method names come from WSDL)[general knowledge]
13. To generate the service endpoint interface from a WSDL you would use:
    - wsimport
      (wsimport generates Java classes from WSDL)[general knowledge]
14. What are the three categories of methods that might be called to handle a HTTP GET by a JAX-RS implementation?
    - Resource
    - Sub-Resource
    - Locator
      (JAX-RS defines resource methods, sub-resource methods, and sub-resource locators)[general knowledge]
15. A sub-resource locator method is annotated with one or more HTTP method annotations such as @GET:

- False
  (Sub-resource locators are not annotated with HTTP method annotations; they return resource objects)[general knowledge]

16. A root resource class must be annotated with:
    - @Path
      (Root resource classes in JAX-RS must have @Path annotation)[general knowledge]
17. Which component of HTTP request contains metadata for the HTTP Request message as key-value pairs?
    - Request Header
      (Request headers contain metadata key-value pairs)[general knowledge]
18. Your API resource does not allow deletion, and a client application attempted to delete the resource. What HTTP response code should you return?
    - 405 Method Not Allowed
      (405 indicates the method is not allowed on the resource)[general knowledge]
19. Which REST constraint specifies that knowledge and understanding obtained from one component of the API should be generally applicable elsewhere in the API?
    - Uniform Interface
      (Uniform Interface constraint ensures consistent interaction across the API)[general knowledge]
20. Which is a common command-line tool for using or exploring an API?
    - curl
      (curl is widely used for testing and exploring APIs)[general knowledge]
21. Which of the following options are true for REST Services?
    - In REST Services, the client gets access to resources and the server provides access to them.
      (REST supports multiple representations, each resource has a unique URI, and clients access resources provided by servers)[general knowledge]
22. The ability to execute the same API request over and over again without changing the resource's state is an example of:
    - idempotency
      (Idempotency means multiple identical requests have the same effect as one)[general knowledge]
23. What must be enabled in order for a RESTful web service to receive invocations from different domains, subdomains or ports?

- CORS
  (Cross-Origin Resource Sharing (CORS) enables cross-domain requests)[general knowledge]

24. How would you configure a RESTful URL parameter that supports a search for a book based on its ID?
    - GET /books/{id}
      (Path parameter style is RESTful and common for resource identification)[general knowledge]
25. The side-effects of N > 0 identical requests are the same as for a single request. This is:
    - Idempotency
      (Idempotency ensures repeated requests have the same side-effects as one)[general knowledge]

Section-II (Any 5 Question)
Marks: 25

1. What is the importance of Using XML Schemas in any application development. Create XML Schema to validate Employee Details.

Importance of Using XML Schemas in Application Development
XML Schemas (XSD) play a crucial role in application development by defining the structure, content, and data types of XML documents. Their importance includes:
1) Data Validation: XML Schemas help make sure that XML documents are correctly structured and follow the rules set for the data. This way, any mistakes can be caught early on, keeping the data accurate and reliable.
2) Support for Data Types: Unlike DTDs, XML Schemas allow you to define a wide variety of data types like strings, integers, and dates. You can also set restrictions such as length limits, specific patterns, or fixed sets of values (enumerations). This strong typing helps ensure that the data follows the expected format and stays consistent, which improves overall data quality.
3) Extensibility and Reusability: Since XML Schemas are themselves written in XML, they are easy to edit and extend using standard XML tools. They support namespaces, which means you can avoid naming conflicts, and schemas can be reused or combined. This modularity simplifies maintenance and makes it easier to manage complex data structures.
4) Interoperability: XML Schemas act as a clear, machine-readable contract that defines how XML data should look. This helps different systems or teams exchange data smoothly without misunderstandings—like mixing up date formats—and allows for easier integration across diverse environments.

5) Tool Support: Many XML parsers, editors, and transformation tools can read and work with XML Schemas. This enables automated validation and processing of XML data within development workflows, making it easier to build reliable applications.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="Employee">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="EmployeeID" type="xs:int"/>
        <xs:element name="Name" type="xs:string"/>
        <xs:element name="Age" type="xs:int">
          <xs:simpleType>
            <xs:restriction base="xs:int">
              <xs:minInclusive value="18"/>
              <xs:maxInclusive value="65"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="Email" type="xs:string"/>
        <xs:element name="Department" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

2. Explain the benefits of JAXB. Unmarshall XML data with JAXB and Marshall XML data with JAXB.

Benefits of JAXB
JAXB (Java Architecture for XML Binding) simplifies working with XML in Java by automating the conversion between Java objects and XML data. Its benefits include:
1) Automatic Conversion: JAXB automatically maps Java objects to XML and vice versa using annotations like @XmlRootElement and @XmlElement. This eliminates the need for manual XML parsing and serialization.

2) Seamless Integration: JAXB integrates with technologies such as JAX-RS (RESTful services) and JAX-WS (SOAP web services), making it ideal for handling XML payloads in web services.
3) Customization and Flexibility: Developers can customize how Java objects are marshalled to XML and unmarshalled back using annotations and XML schema bindings, allowing fine control over XML structure.
4) Integration: JAXB integrates well with Java web service technologies like JAX-RS and JAX-WS, making it ideal for handling XML in web services.
5) Efficiency: By automating XML binding, JAXB reduces boilerplate code and improves performance compared to traditional DOM or SAX parsing methods.
6) Validation Support: JAXB can validate XML documents against schemas during unmarshalling, ensuring data integrity.

Unmarshalling XML Data with JAXB: Unmarshalling is the process of converting XML data into Java objects. Steps include:
1) Create Java classes (POJOs) annotated with JAXB annotations to represent the XML structure.
2) Create a JAXBContext instance for these classes.
3) Create an Unmarshaller from the JAXBContext.
4) Call unmarshal() on the XML input (file, stream, etc.) to convert XML to Java objects.
5) Access the data through the Java objects' getter methods

Marshalling Java Objects to XML with JAXB: Marshalling is the reverse process—converting Java objects into XML.
1) Create a JAXBContext for the Java classes.
2) Create a Marshaller from the JAXBContext.
3) Call marshal() on the Java object to produce XML output (to file, stream, or console).
4) Optionally, configure the marshaller to format the XML for readability

3. What is SOAP Based WebServices? Explain the need of RESTFul Web Services in Application Development.

SOAP (Simple Object Access Protocol) based web services are a protocol-driven approach for exchanging structured information in the implementation of web services. SOAP uses XML to format messages and relies on protocols such as HTTP, SMTP, or others for message transport. It defines a strict set of rules for message structure, encoding, and processing, including a formal contract using WSDL (Web Services Description Language) that describes the service interface and operations. SOAP web services are operation-centric, exposing functions like CreateEmployee that clients invoke by sending XML messages conforming to SOAP standards. SOAP web services are typically stateful, maintaining session information between requests, and support advanced features like built-in error handling, transaction compliance (ACID), and WS-Security for message integrity and confidentiality.

Need for RESTful Web Services in Application Development -
REST (Representational State Transfer) is an architectural style that addresses some limitations of SOAP by providing a simpler, more flexible approach to web services. RESTful web services focus on resources (data entities) rather than operations, using standard HTTP methods (GET, POST, PUT, DELETE) to perform actions on resources identified by URIs (Uniform Resource Identifiers).

The need for RESTful services arises from modern application development demands for:
1) Simplicity and Flexibility: REST uses standard HTTP and supports multiple data formats like JSON, XML, HTML, and plain text, making it easier to develop and consume across diverse clients including browsers and mobile devices.
2) Statelessness: Each REST request contains all necessary information, allowing servers to treat each request independently, which simplifies server design and improves scalability.
3) Performance: REST messages are generally smaller and can be cached, resulting in faster response times and reduced bandwidth usage.
4) Scalability: Due to statelessness and layered architecture, RESTful services scale more easily, allowing load balancing and intermediary caching.
5) Broad Adoption: REST is widely used for public APIs, mobile applications, and web services where lightweight, fast, and scalable communication is essential.
6) Security: REST leverages HTTPS for encryption and supports modern authentication mechanisms like OAuth without the additional overhead required by SOAP's WS-Security.

4. Create a sample application for Airline Reservation Application Representing SOAP Based
Web Services both the Approaches [Top Down and Bottom Up]

Top-Down Approach : WSDL -> Java classes -> service logic ->Deploy

```
<definitions name="AirlineReservationService"
            targetNamespace="http://example.com/airline"
            xmlns:tns="http://example.com/airline"
            xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
          xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
<xsd:schema targetNamespace="http://example.com/airline">
<xsd:element name="BookFlightRequest">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="flightNumber" type="xsd:string"/>
<xsd:element name="passengerName" type="xsd:string"/>
<xsd:element name="date" type="xsd:date"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="BookFlightResponse">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="confirmationNumber" type="xsd:string"/>
<xsd:element name="status" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
</types>

  <message name="BookFlightRequestMessage">
<part name="parameters" element="tns:BookFlightRequest"/>
</message>
<message name="BookFlightResponseMessage">
<part name="parameters" element="tns:BookFlightResponse"/>
</message>

  <portType name="AirlineReservationPortType">
<operation name="BookFlight">
<input message="tns:BookFlightRequestMessage"/>
<output message="tns:BookFlightResponseMessage"/>
</operation>
</portType>

  <binding name="AirlineReservationBinding" type="tns:AirlineReservationPortType">
<soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
```

```
<operation name="BookFlight">
<soap:operation soapAction="http://example.com/airline/BookFlight"/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
</binding>

  <service name="AirlineReservationService">
<port name="AirlineReservationPort" binding="tns:AirlineReservationBinding">
<soap:address location="http://localhost:8080/AirlineReservationService"/>
</port>
</service>

</definitions>
```

```
@WebService(endpointInterface =
"com.example.airline.wsdlclient.AirlineReservationPortType")
public class AirlineReservationServiceImpl implements AirlineReservationPortType {

    @Override
    public BookFlightResponse bookFlight(BookFlightRequest request) {
        BookFlightResponse response = new BookFlightResponse();
        // Simple logic: generate confirmation and status
        response.setConfirmationNumber("CONF12345");
        response.setStatus("Success");
        return response;
    }
}
```

Bottom-Up Approach : Java classes with annotations -> WSDL generation -> Deploy

```
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.xml.bind.annotation.XmlRootElement;

@WebService
```

```
public class AirlineReservationService {

    @WebMethod
    public BookingResponse bookFlight(BookingRequest request) {
        BookingResponse response = new BookingResponse();
        response.setConfirmationNumber("CONF67890");
        response.setStatus("Success");
        return response;
    }
}

@XmlRootElement
class BookingRequest {
    private String flightNumber;
    private String passengerName;
    private String date; // Use String or XMLGregorianCalendar for simplicity

    // Getters and setters
    public String getFlightNumber() { return flightNumber; }
    public void setFlightNumber(String flightNumber) { this.flightNumber = flightNumber; }
    public String getPassengerName() { return passengerName; }
    public void setPassengerName(String passengerName) { this.passengerName =
passengerName; }
    public String getDate() { return date; }
    public void setDate(String date) { this.date = date; }
}

@XmlRootElement
class BookingResponse {
    private String confirmationNumber;
    private String status;

    // Getters and setters
    public String getConfirmationNumber() { return confirmationNumber; }
    public void setConfirmationNumber(String confirmationNumber) {
this.confirmationNumber = confirmationNumber; }
    public String getStatus() { return status; }
    public void setStatus(String status) { this.status = status; }
}
```

5. Create an application for RESTFul Web Services for Airline Reservation Application Representing RESTFul Webservices [Both ServerSide and ClientSide (Using Jersey] Implementation.

```
package com.example.airline;

import jakarta.ws.rs.*;
import jakarta.ws.rs.core.MediaType;
import jakarta.ws.rs.core.Response;
import java.util.*;

@Path("/reservations")
public class AirlineReservationResource {

    @GET
    @Path("/{id}")
    public Response getReservation(String id) {
        Reservation reservation = reservations.get(id);
        return reservation;
    }

    // Update reservation by ID (PUT)
    @PUT
    @Path("/{id}")
    public Response updateReservation( String id, Reservation updatedReservation) {
        Reservation existing = reservations.get(id);
        updatedReservation.setId(id);
        reservations.put(id, updatedReservation);
        return updatedReservation;
    }
}

package com.example.airline;

import jakarta.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Reservation {
    private String id;
    private String passengerName;
    private String flightNumber;
    private String departureDate; // ISO date string e.g., "2025-07-01"

    // Getters and setters
    public String getId() { return id; }
    public void setId(String id) { this.id = id; }

    public String getPassengerName() { return passengerName; }
    public void setPassengerName(String passengerName) { this.passengerName =
passengerName; }

    public String getFlightNumber() { return flightNumber; }
    public void setFlightNumber(String flightNumber) { this.flightNumber = flightNumber; }

    public String getDepartureDate() { return departureDate; }
    public void setDepartureDate(String departureDate) { this.departureDate =
departureDate; }
}

package com.example.airline;

import org.glassfish.grizzly.http.server.HttpServer;
import org.glassfish.jersey.grizzly2.httpserver.GrizzlyHttpServerFactory;

import java.net.URI;

public class Main {
    public static final String BASE_URI = "http://localhost:8080/api/";

    public static void main(String[] args) {
        final HttpServer server =
GrizzlyHttpServerFactory.createHttpServer(URI.create(BASE_URI), new
AirlineApplication());
        System.out.println("Server started at " + BASE_URI + "\nPress enter to stop...");
        try {
            System.in.read();
        } catch (Exception e) {
            e.printStackTrace();
        }
        server.shutdownNow();
    }
}

package com.example.airline.client;

import com.example.airline.Reservation;
import jakarta.ws.rs.client.Client;
import jakarta.ws.rs.client.ClientBuilder;
import jakarta.ws.rs.client.Entity;
import jakarta.ws.rs.core.MediaType;
import jakarta.ws.rs.core.Response;

public class AirlineReservationClient {
    private static final String BASE_URI = "http://localhost:8080/api/reservations";

    public static void main(String[] args) {
        Client client = ClientBuilder.newClient();

        // Create a new reservation
        Reservation newReservation = new Reservation();
        newReservation.setPassengerName("Alice Johnson");
        newReservation.setFlightNumber("AI123");
        newReservation.setDepartureDate("2025-07-01");

        Response postResponse = client.target(BASE_URI)
                .request(MediaType.APPLICATION_JSON)
                .post(Entity.entity(newReservation, MediaType.APPLICATION_JSON));

        if (postResponse.getStatus() == 201) {
            Reservation created = postResponse.readEntity(Reservation.class);
            System.out.println("Created reservation ID: " + created.getId());

            // Get the reservation by ID
            Reservation fetched = client.target(BASE_URI).path(created.getId())
                    .request(MediaType.APPLICATION_JSON)
                    .get(Reservation.class);
            System.out.println("Fetched reservation for passenger: " +
fetched.getPassengerName());

            // Update reservation
            fetched.setPassengerName("Alice M. Johnson");
            Reservation updated = client.target(BASE_URI).path(fetched.getId())
                    .request(MediaType.APPLICATION_JSON)
                    .put(Entity.entity(fetched, MediaType.APPLICATION_JSON),
Reservation.class);
            System.out.println("Updated passenger name: " +
updated.getPassengerName());

            // Delete reservation
            Response deleteResponse = client.target(BASE_URI).path(fetched.getId())
                    .request()
                    .delete();
            System.out.println("Delete status: " + deleteResponse.getStatus());

        } else {
            System.out.println("Failed to create reservation. HTTP error code: " +
postResponse.getStatus());
        }

        client.close();
    }
}
```

6. Explain Jersey Client API. Explore Different Types of Parameter Passing in RESTFul Webservices.

The Jersey Client API is a Java library that makes it easier to build RESTful web service clients. As the reference implementation of JAX-RS, it provides a fluent and type-safe API for sending HTTP requests like GET, POST, PUT, and DELETE, and for handling responses from REST services. By abstracting the complexities of managing HTTP connections and parsing responses, Jersey Client allows developers to focus on working with REST APIs more efficiently and intuitively.

Path Parameter: Parameters embedded directly in the URI path to identify specific resources.
Example: /employees/{id} where {id} is a path parameter.
Used to uniquely identify a resource.

```
@GET
@Path("/employees/{id}")
```

```
public Employee getEmployee(@PathParam("id") String id) {
}
```

Query Parameter: Parameters appended to the URI after a question mark (?) to filter or modify the request.
Example: /employees?department=HR&age=30
Used for optional filtering or searching.

```
@GET
@Path("/employees")
public List<Employee> getEmployees(@QueryParam("department") String dept) {
   // Fetch employees filtered by department
}
```

Matrix Parameters (@MatrixParam):
Parameters embedded within the path segments separated by semicolons (;).
Example: /employees;department=HR;age=30
Less commonly used but supported by JAX-RS.

Header parameter: Parameters passed in HTTP headers.
Example: Authorization tokens or custom headers.

```
@GET
@Path("/secure-data")
public Response getSecureData(@HeaderParam("Authorization") String authHeader) {
}
```

Form Parameter: Parameters sent as part of form data in POST requests.

```
@POST
@Path("/employees")
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
public Response createEmployee(@FormParam("name") String name,
@FormParam("email") String email) {
}
```

Cookie Parameters (@CookieParam): Parameters passed via HTTP cookies.

```
@GET
@Produces(MediaType.APPLICATION_JSON)
public Response getItems(@CookieParam("cookieParam1") String cookieParam1,
@CookieParam("cookieParam2") String cookieParam2) {
      System.out.println("cookieParam1 is :: " + cookieParam1);
      System.out.println("cookieParam2 is :: " + cookieParam2);
```