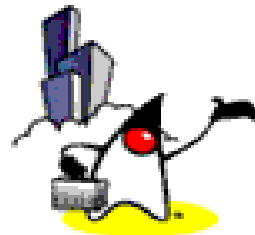




# **SAAJ** **(SOAP Attachment** **API for Java™)**

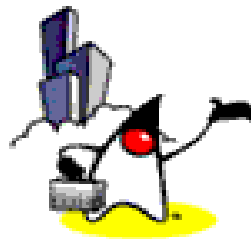


# Agenda

- What is SAAJ?
- SOAP message structure
- SAAJ programming APIs
- SAAJ and DOM
- SAAJ service
- SAAJ error handling



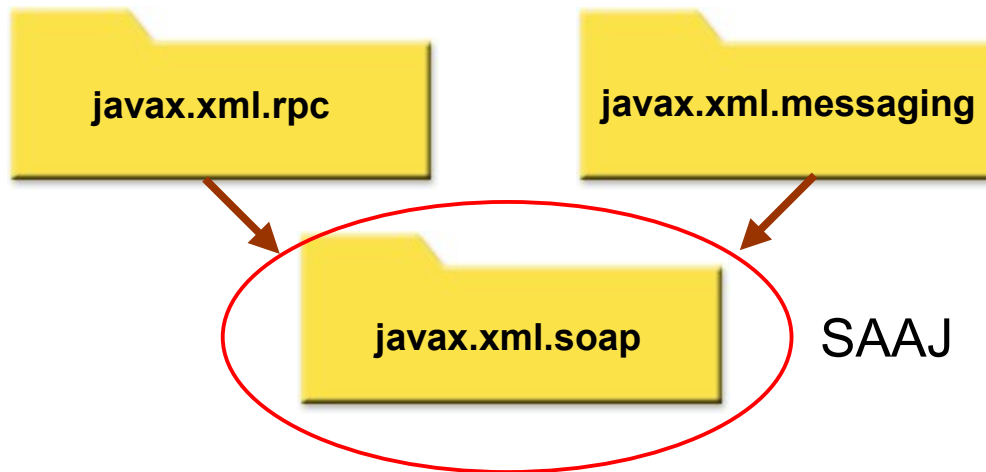
# What is SAAJ?



# SAAJ

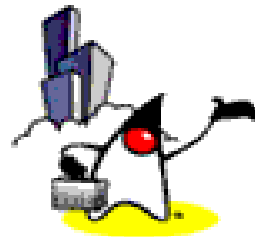
- Contains the API for **creating, populating, and accessing SOAP messages** conforming to SOAP 1.1 and SOAP with Attachment specifications
  - Used within SOAP message handlers for reading/modifying SOAP headers
- Contains API necessary for sending **request-response (non-provider-model)** messages

# Package Relationship

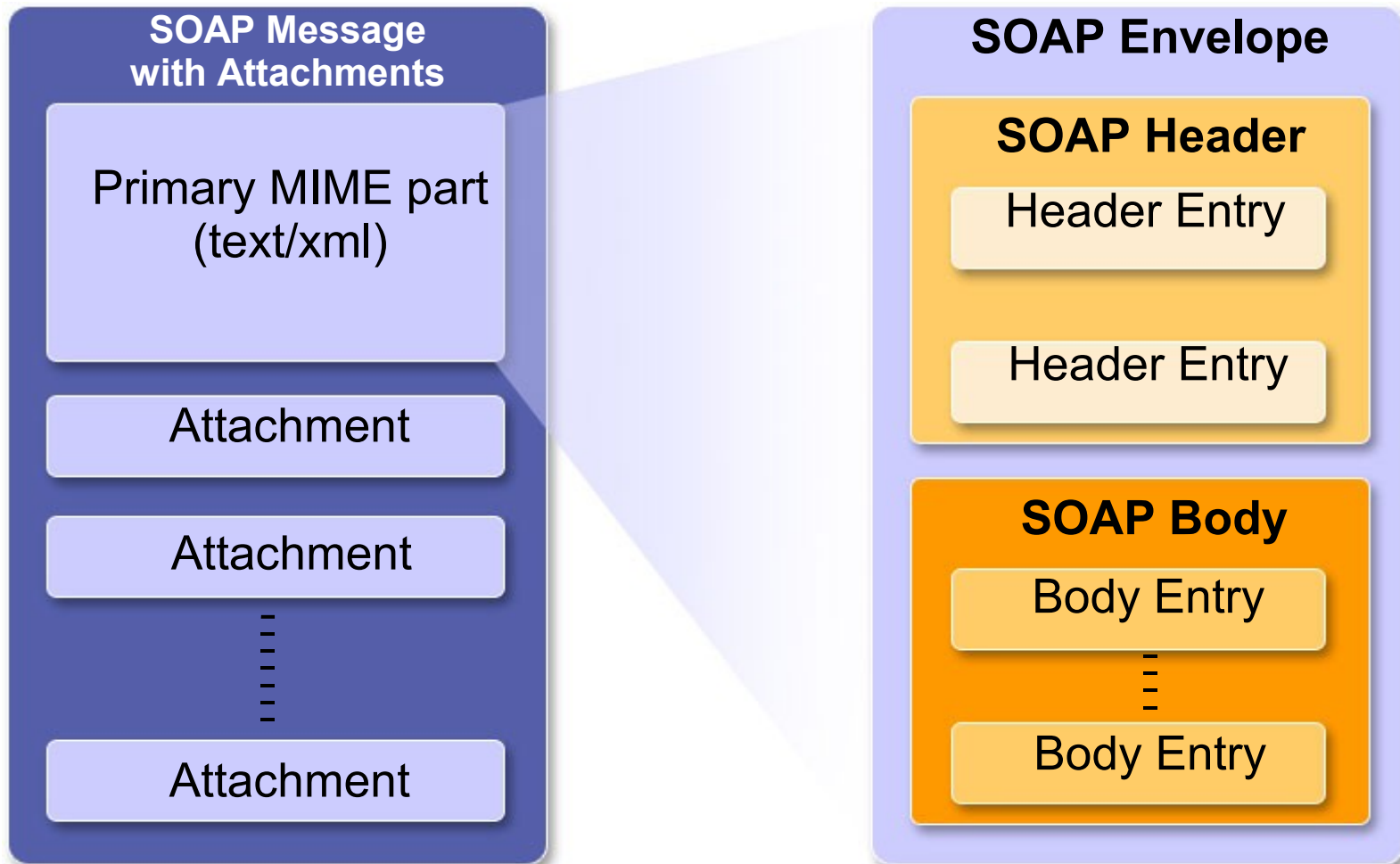




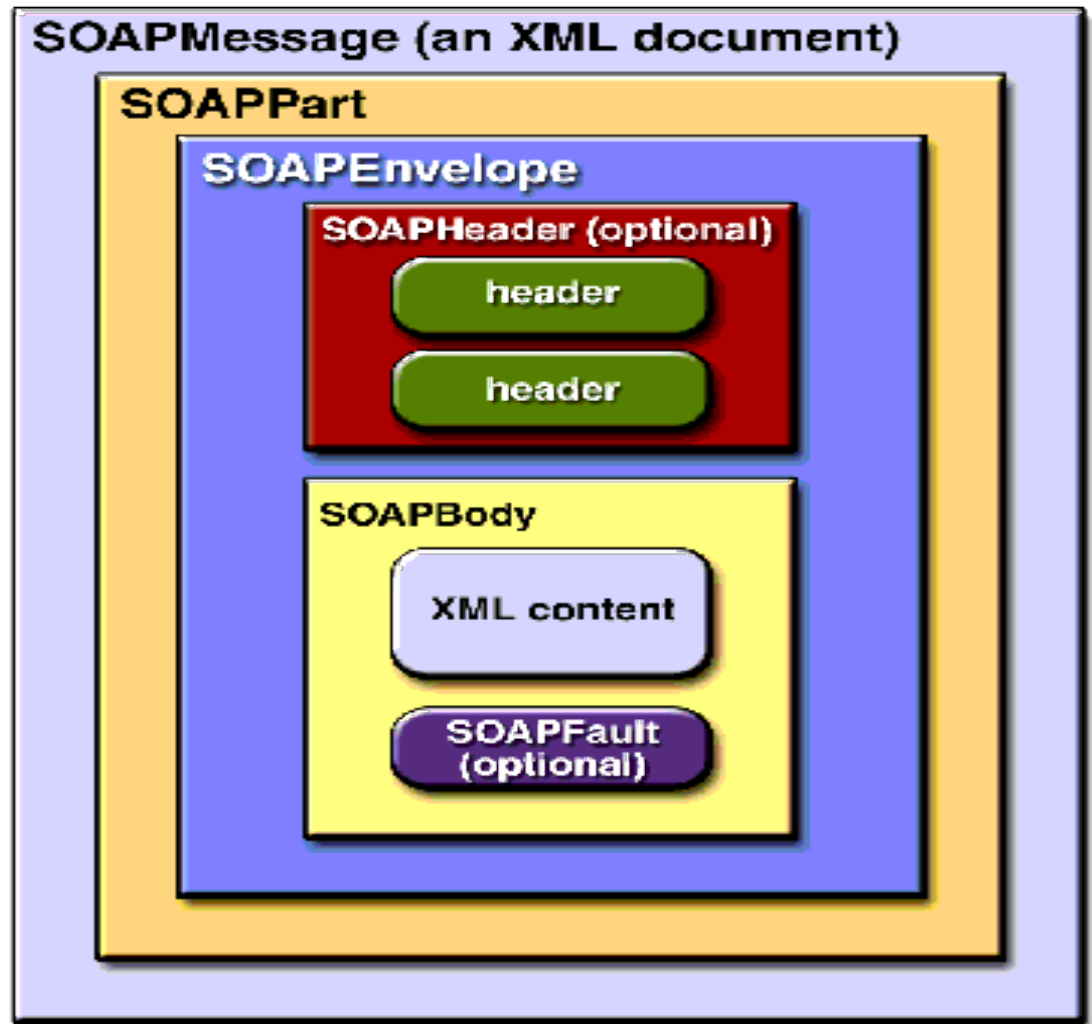
# SOAP Message Structure



# SOAP Message Structure

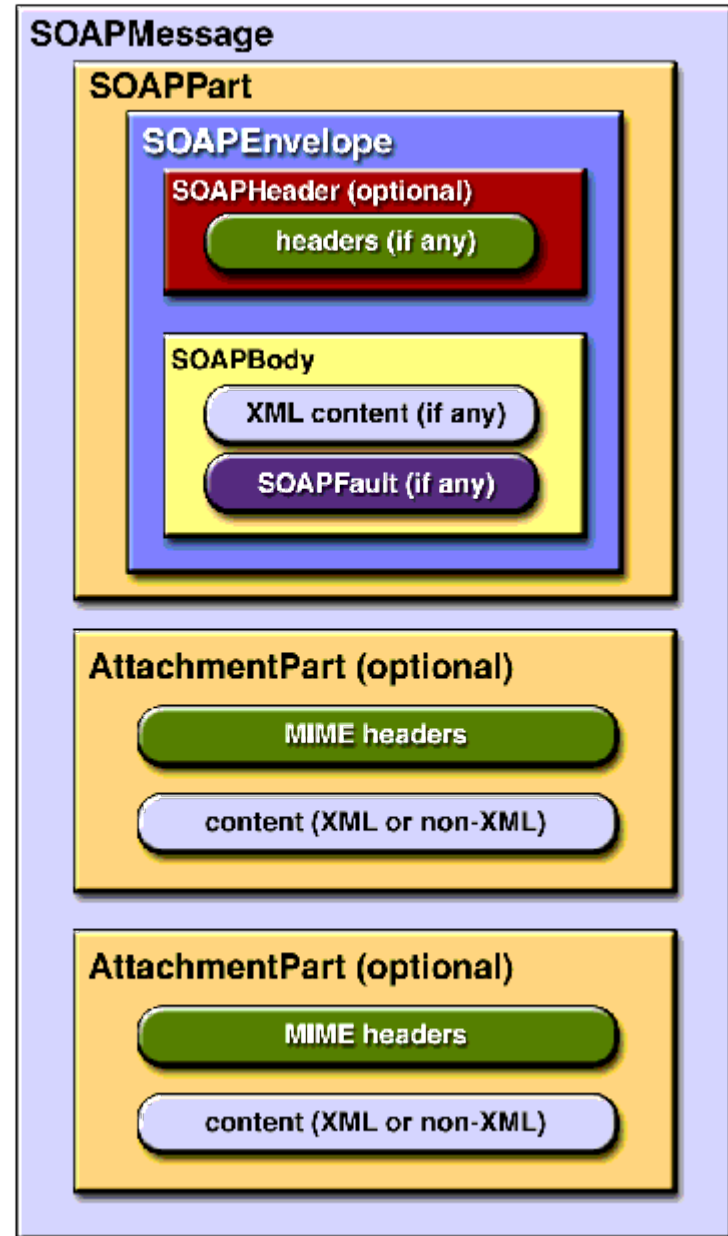


# SOAP Message Object Without Attachments



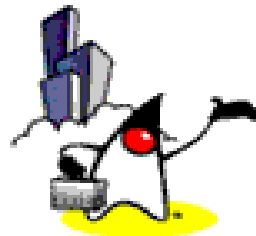
# SOAP Message Object With Attachments

source: Java WSDP





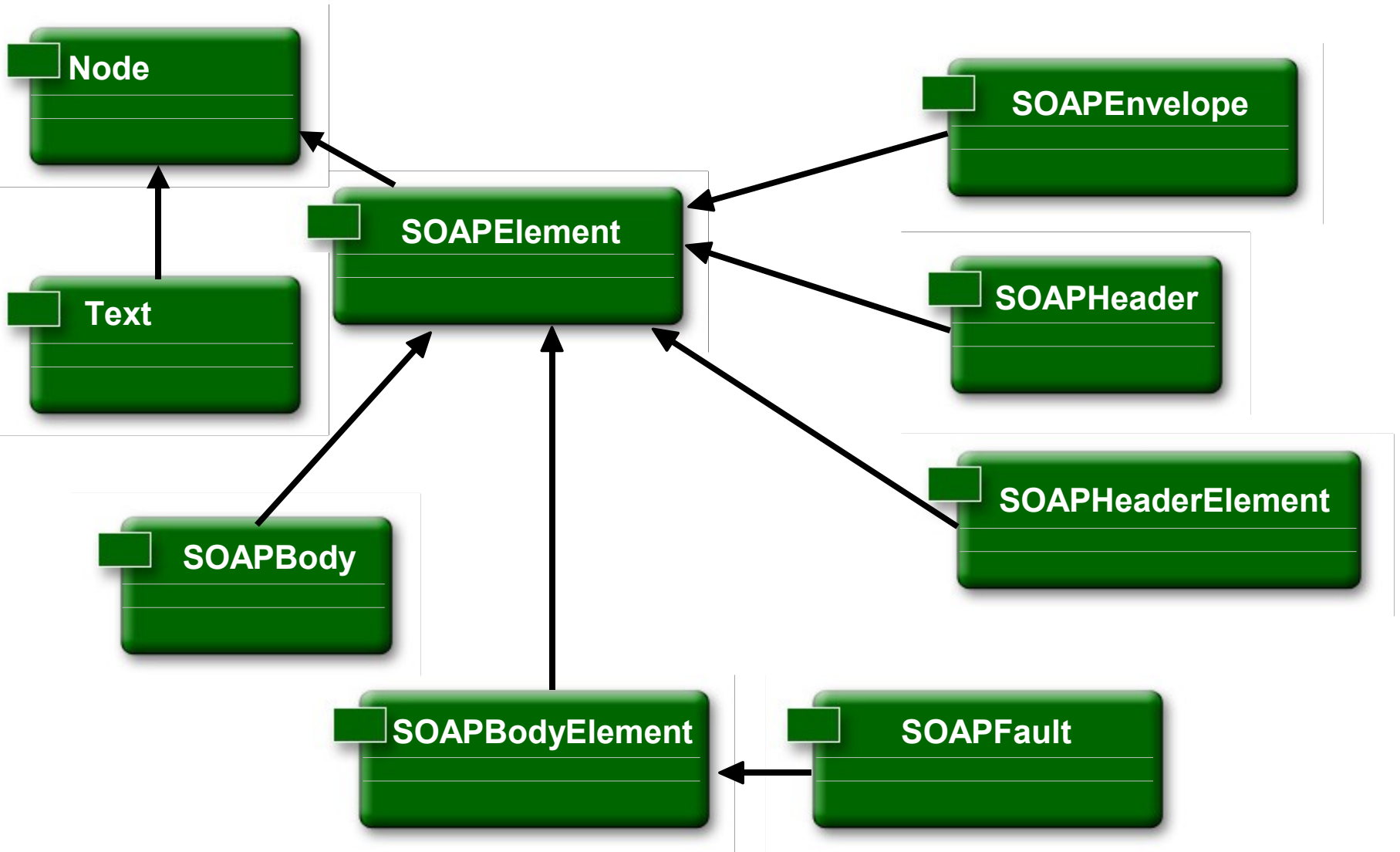
# SAAJ Programming APIs



# SAAJ

- `javax.xml.soap.*` package
- `SOAPConnection`
  - For request/response SOAP message exchange
- `SOAPMessage`
  - Creating and populating
- Constituent parts of a SOAP message
  - `SOAPPart`
  - `AttachmentPart`

# javax.xml.soap Package: Interfaces



# javax.xml.soap Package: Classes

MessageFactory

SOAPPart

SOAPElementFactory

SOAPMessage

AttachmentPart

SOAPConnection

SOAPConnectionFactory

Exception  
SOAPException

MimeHeader

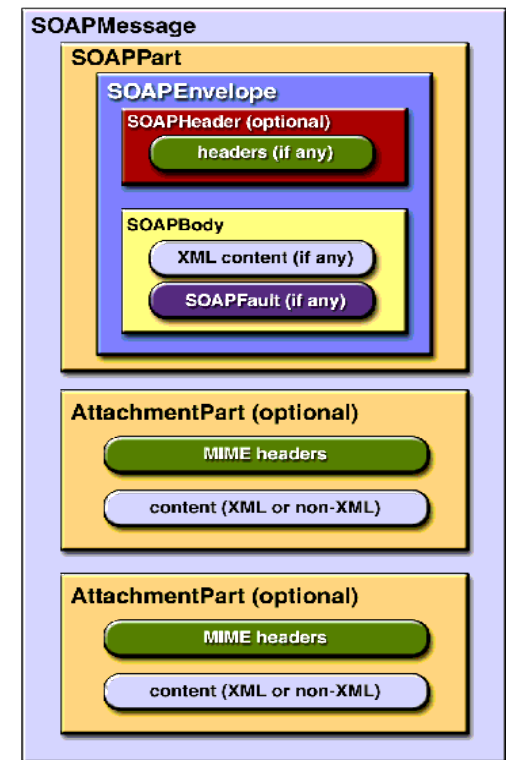
MimeHeaders

# javax.xml.soap.MessageFactory

- Factory for creating SOAP 1.1 and 1.2 -based messages

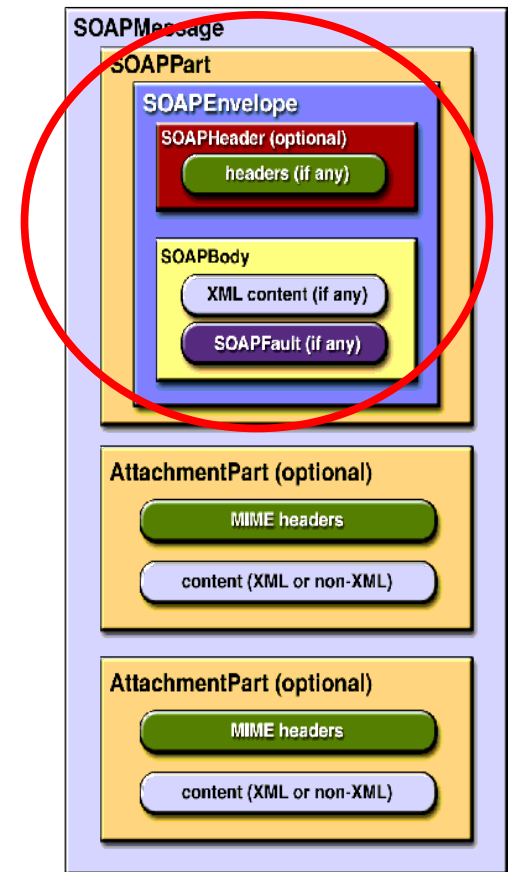
# javax.xml.soap.SOAPMessage

- Java technology abstraction for a SOAP 1.1 message
- **SOAPPart** + Zero or more **AttachmentPart**'s



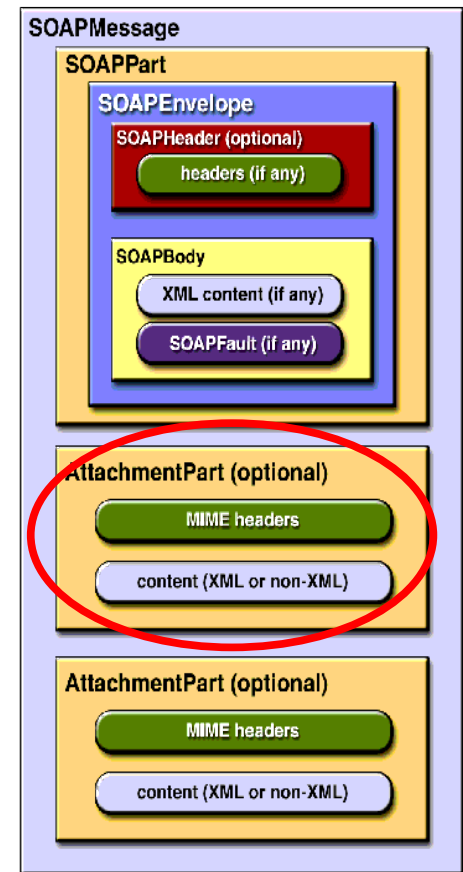
# javax.xml.soap.SOAPPart

- The first part of a multi-part message when there are attachments
- Contains a SOAP 1.1 header and body
- Support to create or examine the SOAP 1.1 header elements



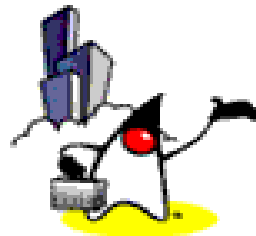
# javax.xml.soap.AttachmentPart

- Can contain any data (for example: JPEG images, XML business documents, etc.)
  - Each AttachmentPart has a MIME header to indicate the type of data it contains
  - An AttachmentPart may also have additional MIME headers to identify it or to give its location, which are optional but can be useful when there are multiple attachments





# Steps of SAAJ Programming



# Steps Sending and Receiving

1. Create a message (SOAPMessage)
2. Access elements of a message
3. Add contents to the body
4. Get a SOAPConnection object
5. Send and receive a message

# 1. Creating a Message

- Use `MessageFactory` as a factory of messages
- `SOAPMessage` object has the following
  - 1 `SOAPPart` object
    - 1 `SOAPEnvelope` object
      - 1 empty `SOAPHeader` object
      - 1 empty `SOAPBody` object

- Example

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

# XML Fragment that gets created

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

## 2. Accessing SOAPHeader & SOAPBody

- Approach 1: from SOAPEnvelope object

```
SOAPPart soapPart = message.getSOAPPart();
SOAPEnvelope envelope = soapPart.getEnvelope();
SOAPHeader header = envelope.getHeader();
SOAPBody body = envelope.getBody();
```
- Approach 2: from SOAPMessage object

```
SOAPHeader header = message.getSOAPHeader();
SOAPBody body = message.getSOAPBody();
```

# 3. Add contents to the body

```
SOAPFactory soapFactory = SOAPFactory.newInstance();  
Name bodyName = soapFactory.createName("GetLastTradePrice",  
    "m", "http://wombat.ztrade.com");
```

```
SOAPBody body = message.getSOAPBody();  
SOAPBodyElement bodyElement = body.addBodyElement(bodyName);
```

Generates XML fragment like following:

```
<SOAP-ENV:Body>  
  <m:GetLastTradePrice  
    xmlns:m="http://wombat.ztrade.com">  
    . . . .  
  </m:GetLastTradePrice>  
</SOAP-ENV:Body>
```

# Add contents to the body

```
Name name = soapFactory.createName("symbol");  
SOAPElement symbol = bodyElement.addChildElement(name);  
symbol.addTextNode("SUNW");
```

Generates XML fragment like following:

```
...  
<symbol>SUNW</symbol>  
...
```

## 4. Get a SOAPConnection Object

```
SOAPConnectionFactory soapConnectionFactory =  
    SOAPConnectionFactory.newInstance();  
SOAPConnection connection =  
    soapConnectionFactory.createConnection();
```

## 5. Send a Message & Receive a response

// Create an endpoint point which is either URL or String type

java.net.URL endpoint =

new URL("http://wombat.ztrade.com/quotes");

// Send a SOAPMessage (request) and then wait for

// SOAPMessage (response)

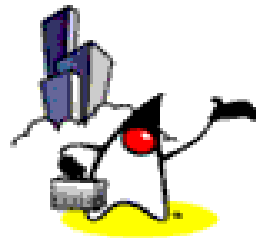
SOAPMessage response = connection.call(message, endpoint);

# Reading a response

```
SOAPBody soapBody = response.getSOAPBody();
java.util.Iterator iterator =
    soapBody.getChildElements(bodyName);
SOAPBodyElement bodyElement =
    (SOAPBodyElement)iterator.next();
String lastPrice = bodyElement.getValue();
System.out.print("The last price for SUNW is ");
System.out.println(lastPrice);
```

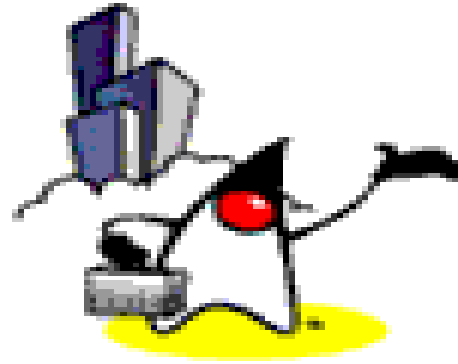


# Constructing SOAP Message



# Constructing SOAP Message

- Adding elements to SOAPHeader
- Adding elements to SOAPBody
- Adding contents of a file to SOAPBody
- Adding contents of a file to SOAPPart
- Adding attachments
- Adding attributes



# Constructing SOAPMessage **Adding Content to SOAPHeader**

# Adding Contents to SOAPHeader

- Create SOAPHeaderElement
  - Must have an associated Name object

- Java Code

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

```
headerElement.addAttribute(soapFactory.createName(  
    "conformsTo"), "http://ws-i.org/profiles/basic1.0/");
```

# Adding Contents to SOAPHeader

- XML fragment generated

```
<SOAP-ENV:Header>
```

```
<wsi:Claim conformsTo="http://ws-i.org/profiles/basic1.0/"  
  xmlns:wsi="http://ws-i.org/schemas/conformanceClaim/">
```

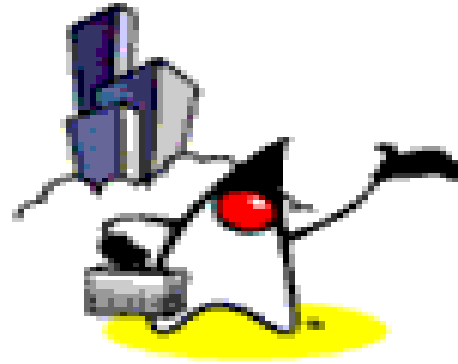
```
</wsi:Claim>
```

```
</SOAP-ENV:Header>
```

# Adding Contents to SOAPHeader

- Adding a Textnode to SOAPHeaderElement

```
headerElement.addTextNode("order");
```



# Constructing SOAPMessage **Adding Content to SOAPBody**

# Adding Contents to Empty SOAPBody

```
SOAPBody body = soapFactory.getSOAPBody();
Name bodyName = soapFactory.createName("PurchaseLineItems",
    "PO", "http://sonata.fruitsgalore.com");
SOAPBodyElement purchaseLineItems =
    body.addBodyElement(bodyName);
```

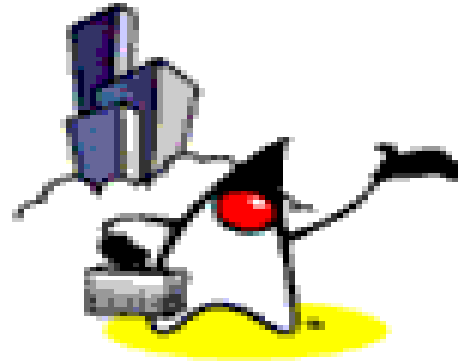
```
Name childName = soapFactory.createName("Order");
SOAPElement order =
    purchaseLineItems.addChildElement(childName);
```

```
childName = soapFactory.createName("Product");
SOAPElement product = order.addChildElement(childName);
product.addTextNode("Apple");
```

```
childName = soapFactory.createName("Price");
SOAPElement price = order.addChildElement(childName);
price.addTextNode("1.56");
```

# XML fragment

```
<PO:PurchaseLineItems  
  xmlns:PO="http://www.sonata.fruitsgalore/order">  
  <Order>  
    <Product>Apple</Product>  
    <Price>1.56</Price>  
  </Order>  
  
  ...  
</PO:PurchaseLineItems>
```



# Constructing SOAPMessage Adding a File to SOAPBody

# Adding an XML File to SOAPBody as DOM object

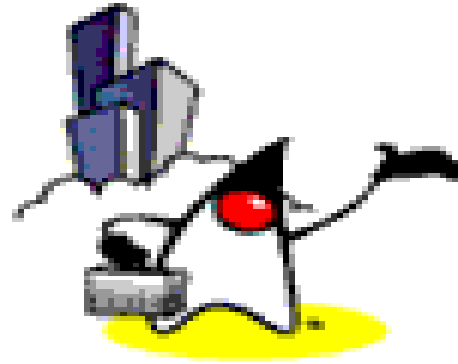
```
static Document document;
```

```
...
```

```
DocumentBuilderFactory factory =  
    DocumentBuilderFactory.newInstance();  
factory.setNamespaceAware(true);  
try {  
    DocumentBuilder builder =  
        factory.newDocumentBuilder();  
    document = builder.parse( new File(args[0]) );  
}
```

```
...
```

```
// document can be any XML document  
SOAPBody body = message.getSOAPBody();  
SOAPBodyElement docElement = body.addDocument(document);
```



# Constructing SOAPMessage

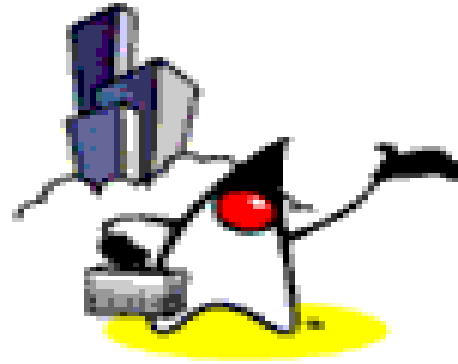
## **Adding a File to SOAPPart**

# Adding a File to SOAPPart Object

- Create a `javax.xml.transform.Source` object
  - using APIs from JAXP
- The XML document must be a **SOAP message format** since it is setting SOAPPart
- Three types of Source objects
  - SAXSource or DOMSource
    - hold content along with the instructions for transforming the content into an XML document
  - StreamSource
    - holds content as an XML document

# Adding a file using DOMSource Object

```
DocumentBuilderFactory dbFactory =  
    DocumentBuilderFactory.newInstance();  
dbFactory.setNamespaceAware(true);  
DocumentBuilder builder = dbFactory.newDocumentBuilder();  
Document document =  
    builder.parse("file:///music/order/soap.xml");  
DOMSource domSource = new DOMSource(document);  
  
// Document must be a SOAP message  
SOAPPart soapPart = message.getSOAPPart();  
soapPart.setContent(domSource);
```



# Constructing SOAPMessage **Adding & Accessing Attachments**

# Creating an AttachmentPart

- Create from `SOAPMessage` object

- Example

```
AttachmentPart attachment =  
    message.createAttachmentPart();
```

- AttachmentPart is made of two parts
  - associated MIME header
    - Content-Type and ID
    - `attachment.setMimeHeader("Content-Type", "application/xml"); //Setting Content-type`
  - application-specific content

# Adding Contents to Attachment (Option 1: Setting Content & ContentId)

```
String stringContent = "Update address for Sunny Skies " +  
    "Inc., to 10 Upbeat Street, Pleasant Grove, CA 95439";
```

```
// Set content and content type
```

```
attachment.setContent(stringContent, "text/plain");
```

```
// Set content id
```

```
attachment.setContentId("update_address");
```

```
// Add an attachment to a SOAP message
```

```
message.addAttachmentPart(attachment);
```

# Possible ContentType

- String
- Stream
- javax.xml.transform.Source
- javax.activation.DataHandler

# Adding Contents to Attachment (Option 2: Using DataHandler)

```
// Create DataHandler object
```

```
URL url =
```

```
    new URL ("http://greatproducts.com/gizmos/img.jpg");
```

```
DataHandler dataHandler = new DataHandler(url);
```

```
AttachmentPart attachment =
```

```
    message.createAttachmentPart(dataHandler);
```

```
// Note that there is no need to set Content Type
```

```
// because DataHandler object already has that info
```

```
attachment.setContentId("attached_image");
```

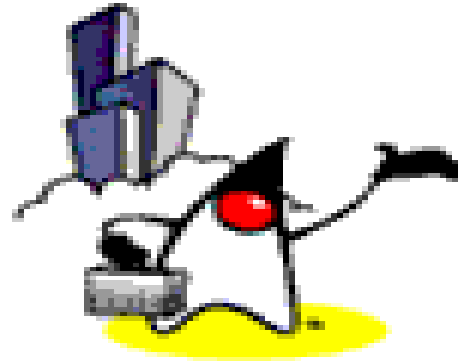
```
message.addAttachmentPart(attachment);
```

# Accessing Attachments

```
java.util.Iterator iterator = message.getAttachments();
while (iterator.hasNext()) {
    AttachmentPart attachment =
        (AttachmentPart)iterator.next();

    // Get content id
    String id = attachment.getContentId();
    // Get content type
    String type = attachment.getContentType();

    System.out.print("Attachment " + id + " has content type " + type);
    if (type == "text/plain") {
        Object content = attachment.getContent();
        System.out.println("Attachment " + "contains:\n" + content);
    }
}
```



# Constructing SOAPMessage **Adding Attributes**

# Adding an Attribute to any SOAPElement

- Java code

```
Name attributeName = envelope.createName("id");  
person.addAttribute(attributeName, "Person7");
```

- XML fragment

```
<person id="Person7">
```

```
...
```

```
</person>
```

# Retrieving a value of an Attribute from any SOAPElement

- Return a value of single attribute

```
String attributeValue =  
    person.getAttributeValue(attributeName);
```

- Return values for all of the attributes

```
Iterator iterator = person.getAllAttributes();  
while (iterator.hasNext()){  
    Name attributeName = (Name) iterator.next();  
    System.out.println("Attribute name is " +  
        attributeName.getQualifiedName());  
    System.out.println("Attribute value is " +  
        element.getAttributeValue(attributeName));  
}
```

# Removeal of an Attribute from any SOAPElement

- Java code

```
boolean successful =  
    person.removeAttribute(attributeName);
```

# Adding Attributes to SOAPHeaderElement

- Header attributes
  - actor
  - mustUnderstand

# Retrieving All SOAPHeaderElement's

```
Iterator allHeaders =
    header.examineAllHeaderElements();
while (allHeaders.hasNext()) {
    SOAPHeaderElement headerElement =
        (SOAPHeaderElement)allHeaders.next();
    Name headerName =
        headerElement.getElementName();
    System.out.println("\nHeader name is " +
        headerName.getQualifiedName());
    System.out.println("Actor is " +
        headerElement.getActor());
}
```

# Adding Actor Attribute

```
SOAPHeader header = message.getSOAPHeader();  
SOAPFactory soapFactory = SOAPFactory.newInstance();
```

```
String nameSpace = "ns";  
String nameSpaceURI = "http://gizmos.com/NSURI";
```

```
Name order = soapFactory.createName("orderDesk",  
    nameSpace, nameSpaceURI);  
SOAPHeaderElement orderHeader =  
    header.addHeaderElement(order);  
orderHeader.setActor("http://gizmos.com/orders");
```

```
Name shipping =  
    soapFactory.createName("shippingDesk",  
        nameSpace, nameSpaceURI);  
SOAPHeaderElement shippingHeader =  
    header.addHeaderElement(shipping);  
shippingHeader.setActor("http://gizmos.com/shipping");
```

# Retrieving All SOAPHeaderElements with a particular Actor (Option 1)

```
java.util.Iterator headerElements =
```

```
header.examineHeaderElements("http://gizmos.com/orders");
```

# Extracting All SOAPHeaderElements with a particular Actor (Option 2)

```
java.util.Iterator headerElements =  
  
    header.extractHeaderElements("http://gizmos.com/order  
s");
```

# Creating SOAPHeaderElement with mustUnderstand attribute (page 1)

- Java code

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

```
Name name = soapFactory.createName("Transaction", "t",  
    "http://gizmos.com/orders");
```

```
SOAPHeaderElement transaction =  
    header.addHeaderElement(name);
```

```
transaction.setMustUnderstand(true);
```

```
transaction.addTextNode("5");
```

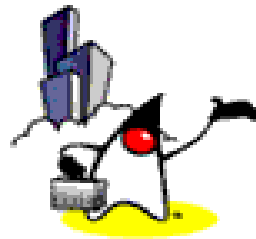
# Creating SOAPHeaderElement with mustUnderstand attribute (page 2)

- XML fragment

```
<SOAP-ENV:Header>  
  <t:Transaction  
    xmlns:t="http://gizmos.com/orders"  
    SOAP-ENV:mustUnderstand="1">  
    5  
  </t:Transaction>  
</SOAP-ENV:Header>
```



# SAAJ & DOM



# SAAJ nodes are DOM nodes

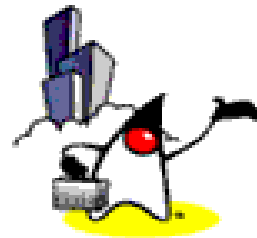
- At SAAJ 1.2, the SAAJ APIs extend their counterparts in the `org.w3c.dom` package
  - The `Node` interface extends the `org.w3c.dom.Node` interface.
  - The `SOAPElement` interface extends both the `Node` interface and the `org.w3c.dom.Element` interface.
  - The `SOAPPart` class implements the `org.w3c.dom.Document` interface.
  - The `Text` interface extends the `org.w3c.dom.Text` interface.

# SAAJ nodes are DOM nodes

- SAAJ nodes and elements implement the DOM Node and Element interfaces, you have several options for adding or changing message content:
  - Use only DOM APIs
  - Use only SAAJ APIs
  - Use SAAJ APIs and then switch to using DOM APIs
  - Use DOM APIs and then switch to using SAAJ APIs (with a caveat)



# SAAJ Service



# SAAJ Service

- Provides the response part of the request-response paradigm
- Server code is in the form of a servlet
- `javax.servlet.HttpServlet`
  - `Init()`, `doPost()`: set up the response message
  - `OnMessage()`: gives the message its content

# SAAJ Distributor Service

```
public class PriceListServlet extends HttpServlet {
    static MessageFactory fac = null;

    static {
        try {
            fac = MessageFactory.newInstance();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
};
```

# SAAJ Distributor Service

```
public void doPost(HttpServletRequest req,
                  HttpServletResponse resp)
    throws ServletException, IOException {
    try {
        // Get all the headers from the HTTP request
        MimeHeaders headers = getHeaders(req);

        // Get the body of the HTTP request
        InputStream is = req.getInputStream();

        // Now internalize the contents of the HTTP request
        // and create a SOAPMessage
        SOAPMessage msg = fac.createMessage(headers, is);
```

# SAAJ Distributor Service

```
SOAPMessage reply = null;  
reply = onMessage(msg);  
if (reply != null) {
```

```
    /*  
    * Need to call saveChanges because we're  
    * going to use the MimeHeaders to set HTTP  
    * response information. These MimeHeaders  
    * are generated as part of the save.  
    */  
    if (reply.saveRequired()) {  
        reply.saveChanges();  
    }
```

# SAAJ Distributor Service

```
resp.setStatus(HttpServletResponse.SC_OK);
putHeaders(reply.getMimeHeaders(), resp);

// Write out the message on the response stream
OutputStream os = resp.getOutputStream();
reply.writeTo(os);
os.flush();
} else {
    resp.setStatus(
        HttpServletResponse.SC_NO_CONTENT);
}
} catch (Exception ex) {
    throw new ServletException( "SAAJ POST failed: " +
        ex.getMessage());
}
}
```

# SAAJ Distributor Service

```
public SOAPMessage onMessage(SOAPMessage msg) {
    SOAPMessage message = null;

    try {
        message = fac.createMessage();

        SOAPPart part = message.getSOAPPart();
        SOAPEnvelope envelope = part.getEnvelope();
        SOAPBody body = envelope.getBody();

        Name bodyName = envelope.createName("price-list",
            "PriceList", "http://sonata.coffeebreak.com");
        SOAPBodyElement list = body.addBodyElement(bodyName);

        Name coffeeN = envelope.createName("coffee");
        SOAPElement coffee = list.addChildElement(coffeeN);

        Name coffeeNm1 = envelope.createName("coffee-name");
        SOAPElement coffeeName =
            coffee.addChildElement(coffeeNm1);
        coffeeName.addTextNode("Arabica");
    }
}
```

# SAAJ Distributor Service

```
Name priceName3 = envelope.createName("price");  
SOAPElement price3 = coffee.addChildElement(priceName3);  
price3.addTextNode("6.00");
```

```
Name coffeeNm4 = envelope.createName("coffee-name");  
SOAPElement coffeeName4 =  
    coffee.addChildElement(coffeeNm4);  
coffeeName4.addTextNode("House Blend");
```

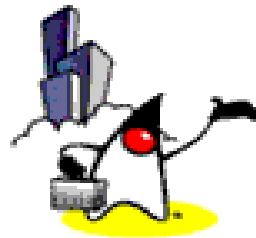
```
Name priceName4 = envelope.createName("price");  
SOAPElement price4 = coffee.addChildElement(priceName4);  
price4.addTextNode("5.00");
```

```
message.saveChanges();
```

```
} catch(Exception e) {  
    e.printStackTrace();  
}  
return message;  
}
```



# SAAJ Error Handling



# SOAPFault Object

- Represents SOAP Fault element in SOAP body
- Parties that can generate SOAPFault
  - Recipient of a message in point-to-point messaging
    - Indicates missing address information in purchase order SOAP message
  - Messaging provider in messaging provider-based messaging
    - Messaging provider cannot deliver the message due to server failure

# SOAPFault Object Contains

- fault code (mandatory)
  - Required in **SOAPFault** Object
  - **VersionMismatch**, **MustUnderstand**, **Client**, **Server**
- fault string (mandatory)
  - Human readable explanation of the fault
- fault actor (conditional)
  - Required if **SOAPHeader** object has one or more **actor** attributes
- Detail object (conditional)
  - Required if the fault is error related to the **SOAPBody** object
  - If not present in Client fault, **SOAPBody** is assumed to be OK

# Example 1: SOAPFault with No Detail object

```
// Create SOAPBody object
```

```
SOAPEnvelope envelope = msg.getSOAPPart().getEnvelope();  
SOAPBody body = envelope.getBody();
```

```
// Create and fill up SOAPFault object.
```

```
// Note that Detail object is not being set here since the fault has
```

```
// nothing to do with SOAPBody
```

```
SOAPFault fault = body.addFault();  
fault.setFaultCode("Server");  
fault.setFaultActor("http://gizmos.com/orders");  
fault.setFaultString("Server not responding");
```

# Example 2: SOAPFault with Detail object

```
// Create SOAPFault object
SOAPFault fault = body.addFault();
// Set fault code and fault string
fault.setFaultCode("Client");
fault.setFaultString("Message does not have necessary info");

// Detail object contains two DetailEntry's
Detail detail = fault.addDetail();
Name entryName =
    envelope.createName("order", "PO", "http://gizmos.com/orders/");
DetailEntry entry = detail.addDetailEntry(entryName);
entry.addTextNode("quantity element does not have a value");

Name entryName2 =
    envelope.createName("confirmation", "PO",
        "http://gizmos.com/confirm");
DetailEntry entry2 = detail.addDetailEntry(entryName2);
entry2.addTextNode("Incomplete address: no zip code");
```

# Example 3: Retrieving SOAPFault

```
// Get SOAPBody object
```

```
SOAPBody body = newmsg.getSOAPPart().getEnvelope().getBody();
```

```
// Check if SOAPFault is present in the message
```

```
if ( body.hasFault() ) {
```

```
    SOAPFault newFault = body.getFault();
```

```
    String code = newFault.getFaultCode();
```

```
    String string = newFault.getFaultString();
```

```
    String actor = newFault.getFaultActor();
```

```
    if ( actor != null ) { System.out.println(" fault actor = " + actor); }
```

```
}
```

# Example 4: Retrieving Detail Object

```
// Get Detail object
```

```
Detail newDetail = newFault.getDetail();
```

```
// Get the list of DetailEntry's
```

```
if ( newDetail != null) {
```

```
    Iterator it = newDetail.getDetailEntries();
```

```
    while ( it.hasNext() ) {
```

```
        DetailEntry entry = (DetailEntry)it.next();
```

```
        String value = entry.getValue();
```

```
        System.out.println(" Detail entry = " + value);
```

```
    }
```

```
}
```



# Passion!

