JSP Custom Tags

- **Custom Tag Handler**

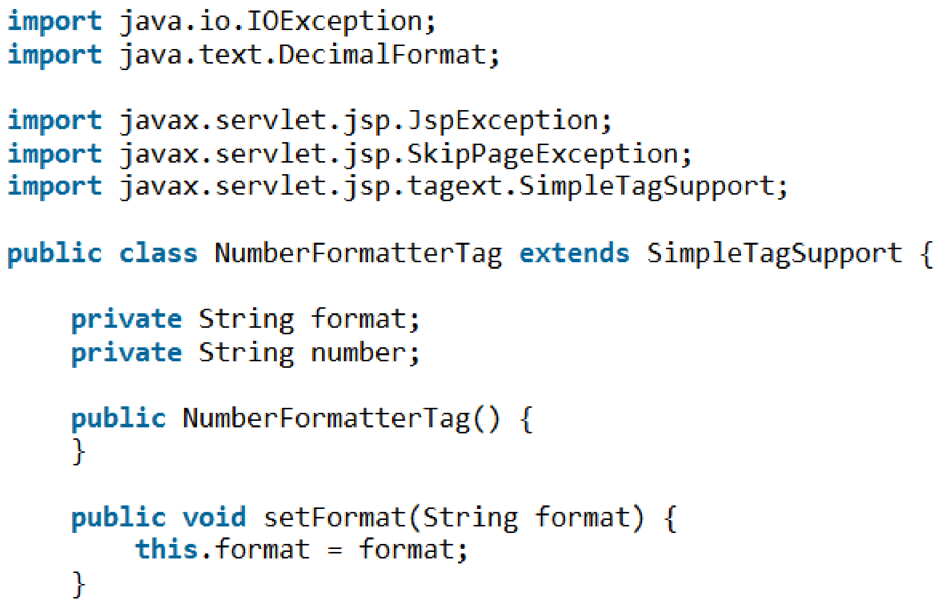
  We know that JSTL doesn't provide any inbuilt tags to achieve some special kind of formatting, so we will create our own custom tag implementation and use it in the JSP page. This is the first step in creating custom tags in JSP. All we need to do is extend javax.servlet.jsp.tagext.SimpleTagSupport class and override **doTag()** method.

  The important point to note is that we should have setter methods for the attributes we need for the tag. So we will define two setter methods – **setFormat(String format)** and **setNumber(String number)**.

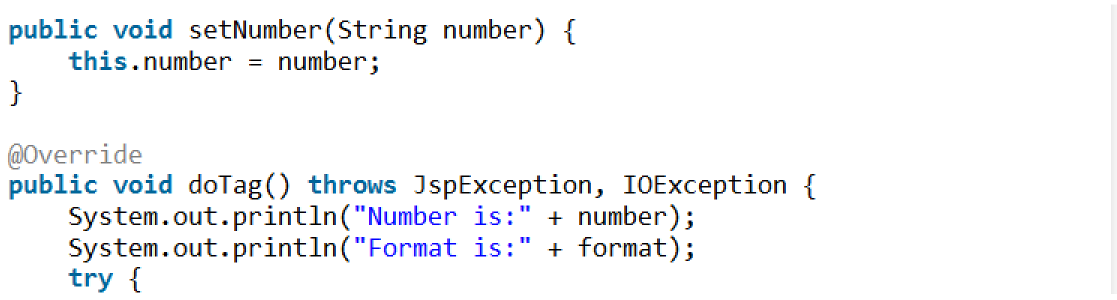
  SimpleTagSupport provide methods through which we can get JspWriter object and write data to the response.

  We will use DecimalFormat class to generate the formatted string and then write it to response. The final implementation is given below.

  NumberFormatterTag.java



```java
import java.io.IOException;
import java.text.DecimalFormat;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.SkipPageException;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class NumberFormatterTag extends SimpleTagSupport {

    private String format;
    private String number;

    public NumberFormatterTag() {
    }

    public void setFormat(String format) {
        this.format = format;
    }
```



```java
public void setNumber(String number) {
    this.number = number;
}

@Override
public void doTag() throws JspException, IOException {
    System.out.println("Number is:" + number);
    System.out.println("Format is:" + format);
    try {
```

```
                double amount = Double.parseDouble(number);
                DecimalFormat formatter = new DecimalFormat(format);
                String formattedNumber = formatter.format(amount);
                getJspContext().getOut().write(formattedNumber);
            } catch (Exception e) {
                e.printStackTrace();
                // stop page from loading further by throwing SkipPageException
                throw new SkipPageException("Exception in formatting " + number
                        + " with format " + format);
            }
        }

    }
```

Notice that I am catching exception thrown by format() method and then throwing SkipPageException to prevent further loading of the JSP page.

- **Creating Tag Library Descriptor (TLD) File**

Once the tag handler class is ready, we need to define a TLD file inside the WEB-INF directory so that container will load it when application is deployed.

numberformatter.tld

```
<?xml version="1.0" encoding="UTF-8" ?>

<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
    version="2.0">
  <description>Number Formatter Custom Tag</description>
  <tlib-version>2.1</tlib-version>
  <short-name>mytags</short-name>
  <uri>http://journaldev.com/jsp/tlds/mytags</uri>
  <tag>
      <name>formatNumber</name>
      <tag-class>com.journaldev.jsp.customtags.NumberFormatterTag</tag-class>
      <body-content>empty</body-content>
      <attribute>
      <name>format</name>
      <required>true</required>
      </attribute>
      <attribute>
      <name>number</name>
      <required>true</required>
      </attribute>
  </tag>
  </taglib>
```

Notice the URI element, we will have to define it in our deployment descriptor file. Also notice the attributes format and number that are required. body-content is empty means that the tag will not have any body.

- **Deployment Descriptor Configuration**
  We will include the tag library in web application using jsp-config and taglib elements like below.
  web.xml

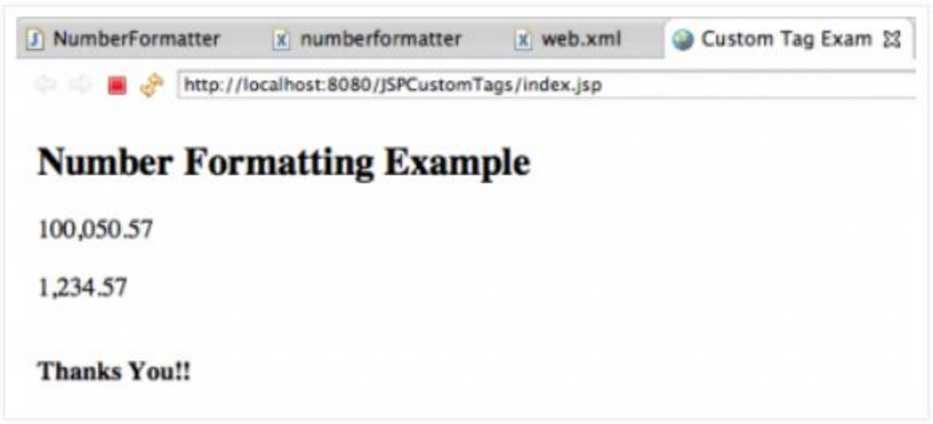
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">
    <display-name>JSPCustomTags</display-name>
    <welcome-file-list>
      <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

    <jsp-config>
    <taglib>
      <taglib-uri>http://journaldev.com/jsp/tlds/mytags</taglib-uri>
      <taglib-location>/WEB-INF/numberformatter.tld</taglib-location>
    </taglib>
    </jsp-config>
  </web-app>
```

  taglib-uri value should be same as defined in the TLD file. All the configuration is done now and we can use it in the JSP page.

- **JSP Page using Custom Tag**
  Here is a sample JSP page where I am using our number formatter custom tag.
  index.jsp

```
<%@ page language="java" contentType="text/html; charset=US-ASCII"
    pageEncoding="US-ASCII"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <html>
  <head>
  <meta http-equiv="Content-Type" content="text/html; charset=US-ASCII">
  <title>Custom Tag Example</title>
  <%@ taglib uri="http://journaldev.com/jsp/tlds/mytags" prefix="mytags"%>
  </head>
  <body>

  <h2>Number Formatting Example</h2>

  <mytags:formatNumber number="100050.574" format="#,###.00"/><br><br>

  <mytags:formatNumber number="1234.567" format="$# ###.00"/><br><br>

  <p><strong>Thanks You!!</strong></p>
  </body>
  </html>
```

Now when we run our web application, we get JSP page like below image.



http://localhost:8080/JSPCustomTags/index.jsp

Number Formatting Example

100,050.57

1,234.57

Thanks You!!

The JSP response page is showing the formatted number, similarly we can create more custom tag handler classes. We can have multiple tags defined in the tag library.
If you have a lot of custom tag handler classes or you want it to provide as a JAR file for others to use, you need to include TLD files in the META-INF directory of the JAR file and include it in the web application lib directory. In that case, we don’t need to configure them in web.xml also because JSP container takes care of that.