USN | | | | | | | | | | |

CMRIT
CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A+ GRADE BY NAAC

Internal Assessment Test 3 – November 2018

Sub:	Advanced JAVA and J2EE					Sub Code:	15CS553	Branch:	CSE
Date:	22/11/2018	Duration:	90 min's	Max Marks:	50	Sem / Sec:	5th A,B,C		OBE

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1	Explain all the methods defined by the collection interface.	[10]	CO2	L2
2	Explain the constructors of Treeset class and write a java program to create TreeSet collection and access it via an iterator.	[10]	CO2	L2
3	What are comparators? Write a Java program to sort the accounts by last name using comparator.	[10]	CO2	L3
4	List down all the legacy classes of java.util package and explain any four in detail with example and its constructors.	[10]	CO2	L3
5(a)	Explain the steps involved in executing servlet.	[06]	CO4	L2
5(b)	What are the advantages of servlet over traditional CGI?	[04]	CO4	L1
6	Define JSP. Explain different type of JSP tags by taking suitable example.	[10]	CO4	L2
7	Explain the classes and interfaces of javax.servlet package.	[10]	CO4	L2
8	What is Cookie? List out the methods defined by the cookie and write a java program to add a cookie.	[10]	CO4	L2

Scheme

Question #	Description	Marks Distribution		Max Marks
1	Explanation of atleast 10 methods of collection interface	1M * 10	10M	10M
2	Explanation on four constructors of Treeset with 1 mark for each. Program demonstrating the use of Treeset with iterator carrying 6 marks.	4M + 6M	10M	10M
3	Explanation on comparators carrying 2 marks. Program to sort the accounts by last name using comparator carrying 8 marks.	2M + 8M	10M	10M
4	List of all legacy classes carrying 2 marks. Explanation of four legacy classes carrying 8 marks.	2M + 8M	10M	10M
5 a	Explanation of basic steps involved in creating and executing servlet carrying 6 marks.	6M	6M	6M
5 b	Explanation on four advantages of Servlet over CGI with 1 mark for each	4M	4M	4M
6	JSP definition : 1 mark. Explanation on different JSP tags with example carrying 9 marks.	1M+9M	10M	10M
7	Explanation on classes of javax.servlet package carrying 5 marks. Explanation on interfaces of javax.servlet package carrying 5 marks.	5M+ 5M	10M	10M
8	Cookie explanation carrying : 1 mark Listing six methods of cookie : 3 marks Program to add cookie : 6 marks	1M+ 3M+ 6M	10M	10M

Solution

1. Explain all the methods defined by the collection interface.

interface Collection<E>

E specifies the type of objects that the collection will hold. Collection extends the Iterable interface.

Iterating through the list can be done through the Iteratable interface. Methods in collection interface

1. **add():** boolean add(E obj). Adds obj to the invoking collection. Returns true if obj was added to the collection. Returns false if obj is already a member of the collection and the collection does not allow duplicates.
2. **addAll():** boolean addAll(Collection<? extends E> c) Adds all the elements of c to the invoking collection. Returns true if the operation succeeded (i.e., the elements were added). Otherwise, returns false.
3. **clear():** void clear( ).Removes all elements from the invoking collection.
4. **contains():** boolean contains(Object obj).Returns true if obj is an element of the invoking collection. Otherwise, returns false.
5. **containsAll:** boolean containsAll(Collection<?> c) Returns true if the invoking collection contains all elements of c. Otherwise, returns false.
6. **equals():** boolean equals(Object obj) Returns true if the invoking collection and obj are equal. Otherwise, returns false.
7. **hashCode():** int hashCode( ).Returns the hash code for the invoking collection.
8. **isEmpty():** boolean isEmpty( ) . Returns true if the invoking collection is empty. Otherwise, returns false.
9. **iterator():** Iterator<E> iterator( ). Returns an iterator for the invoking collection.
10. **remove():** boolean remove(Object obj) Removes one instance of obj from the invoking collection. Returns true if the element was removed. Otherwise, returns false.
11. **removeAll():**boolean removeAll(Collection<?> c)Removes all elements of c from the invoking collection. Returns true if the collection changed (i.e., elements were removed).Otherwise, returns false.
12. **retainAll():**boolean retainAll(Collection<?> c)Removes all elements from the invoking collection except those in c. Returns true if the collection changed (i.e., elements were removed).Otherwise, returns false.
13. **size():**int size( ) Returns the number of elements held in the invoking collection.
14. **toArray**Object[ ] toArray( ).Returns an array that contains all the elements stored in the invoking collection. The array elements are copies of the collection elements. The array elements are copies of the collection elements. If the size of array equals the number of elements, these are returned in array.

2. Explain the constructors of Treeset class and write a java program to create TreeSet collection and access it via an iterator.

TreeSet extends **AbstractSet** and implements the **NavigableSet** interface. Objects are stored in sorted, ascending order. **TreeSet** is a generic class that has this declaration:

class TreeSet<E>

TreeSet has the following constructors:

TreeSet( )

TreeSet(Collection<? extends E> *c*)

TreeSet(Comparator<? super E> *comp*)

TreeSet(SortedSet<E> *ss*)

The first form constructs an empty tree set that will be sorted in ascending order according to the natural order of its elements. The second form builds a tree set that contains the elements of *c*. The third form constructs an empty tree set that will be sorted according to the comparator specified by *comp*. (Comparators are described later in this chapter.) The fourth form builds a tree set that contains the elements of *ss*.

Below program illustrates the use of Java.util.TreeSet.iterator() method:

```
// Java code to illustrate iterator()

import java.util.*;
import java.util.TreeSet;
public class TreeSetDemo {
   public static void main(String args[])
   {
      // Creating an empty TreeSet
      TreeSet<String> set = new TreeSet<String>();
      // Use add() method to add elements into the Set
      set.add("Java");
      set.add("Python");
      set.add("Algol");
      set.add("C");
      set.add("Fortron");
      // Displaying the TreeSet
      System.out.println("TreeSet: " + set);
      // Creating an iterator
      Iterator value = set.iterator();
```

```
        // Displaying the values after iterating through the set
        System.out.println("The iterator values are: ");
        while (value.hasNext()) {
            System.out.println(value.next());
        }
    }
}
```

Output of the above program

TreeSet: [Algol, C, Fortron, Java, Python]

The iterator values are:

Algol

C

Fortron

Java

Python

3. What are comparators? Write a Java program to sort the accounts by last name using comparator.

The comparator that defines precisely what “sorted order” means. By default, these classes store their elements by using what Java refers to as “natural ordering,” which is usually the ordering that you would expect (A before B, 1 before 2, and so forth). If you want to order elements a different way, then specify a Comparator when you construct the set or map. Doing so gives you then ability to govern precisely how elements are stored within sorted collections and maps.

Comparator is a generic interface that has this declaration:

interface Comparator<T>

Here, T specifies the type of objects being compared.

The Comparator interface defines two methods: compare( ) and equals( ). The compare( )

method, shown here, compares two elements for order:

int compare(T obj1, T obj2)

obj1 and obj2 are the objects to be compared. This method returns zero if the objects are equal.

It returns a positive value if obj1 is greater than obj2. Otherwise, a negative value is returned.

The equals( ) method, shown here, tests whether an object equals the invoking comparator:

boolean equals(Object obj)

Here, obj is the object to be tested for equality. The method returns true if obj and the invoking object are both Comparator objects and use the same ordering. Otherwise, it returns false.

```
// A class to represent a student.
import java.util.*;
// Compare last whole words in two strings.
class TComp implements Comparator<String> {
public int compare(String a, String b) {
int i, j, k;
String aStr, bStr;
aStr = a;
bStr = b;
// Find index of beginning of last name.
i = aStr.lastIndexOf(' ');
j = bStr.lastIndexOf(' ');
k = aStr.substring(i).compareTo(bStr.substring(j));
if(k==0) // last names match, check entire name
return aStr.compareTo(bStr);
else
return k;
}
// No need to override equals.
}
class TreeMapDemo2 {
public static void main(String args[]) {
// Create a tree map.
TreeMap<String, Double> tm = new TreeMap<String, Double>(new TComp());
// Put elements to the map.
tm.put("John Doe", new Double(3434.34));
tm.put("Tom Smith", new Double(123.22));
tm.put("Jane Baker", new Double(1378.00));
tm.put("Tod Hall", new Double(99.22));
tm.put("Ralph Smith", new Double(-19.08));
// Get a set of the entries.
Set<Map.Entry<String, Double>> set = tm.entrySet();
// Display the elements.
```

```
for(Map.Entry<String, Double> me : set) {
System.out.print(me.getKey() + ": ");
System.out.println(me.getValue());
}
System.out.println();
// Deposit 1000 into John Doe's account.
double balance = tm.get("John Doe");
tm.put("John Doe", balance + 1000);
System.out.println("John Doe's new balance: " +
tm.get("John Doe"));
}
}
```

Here is the output; notice that the accounts are now sorted by last name:

Jane Baker: 1378.0

John Doe: 3434.34

Todd Hall: 99.22

Ralph Smith: -19.08

Tom Smith: 123.22

John Doe's new balance: 4434.34

4. List down all the legacy classes of java.util package and explain any four in detail with example and its constructors.

Early version of java did not include the Collections framework. It only defined several classes and interfaces that provide methods for storing objects. When Collections framework were added in J2SE 1.2, the original classes were reengineered to support the collection interface. These classes are also known as Legacy classes. All legacy classes and interface were redesign by JDK 5 to support Generics. In general, the legacy classes are supported because there is still some code that uses them.

The following are the legacy classes defined by **java.util** package

1. Dictionary
2. HashTable
3. Properties
4. Stack
5. Vector

There is only one legacy interface called **Enumeration**

1. Dictionary

Dictionary is an abstract class that represents a key/value storage repository and operates much like **Map**. Given a key and value, you can store the value in a **Dictionary** object. Although not currently deprecated, **Dictionary** is classified as obsolete, because it is fully superseded by **Map**. However, **Dictionary** is still in use and thus is fully discussed here.

With the advent of JDK 5, **Dictionary** was made generic. It is declared as shown here:

class Dictionary<K, V>

Some methods of Dictionary class are given below

V put(K *key*, V *value*)	Inserts a key and its value into the dictionary. Returns **null** if *key* is not already in the dictionary; returns the previous value associated with *key* if *key* is already in the dictionary.
V remove(Object *key*)	Removes *key* and its value. Returns the value associated with *key*. If *key* is not in the dictionary, a **null** is returned.

2. HashTable

Like HashMap, Hashtable also stores key/value pair. However neither **keys** nor **values** can be **null**. There is one more difference between **HashMap** and **Hashtable** that is Hashtable is synchronized while HashMap is not.

Hashtable has following four constructor.

- Hashtable() //This is the default constructor. The default size is 11.
- Hashtable(int size) //This creates a hash table that has an initial size specified by size.
- Hashtable(int size, float fillratio) //This creates a hash table that has an initial size specified by size and a fill ratio specified by fillRatio. This ratio must be between 0.0 and 1.0, and it determines how full the hash table can be before it is resized upward. Specifically, when the number of elements is greater than the capacity of the hash table multiplied by its fill ratio, the hash table is expanded. If you do not specify a fill ratio, then 0.75 is used.
- Hashtable(Map< ? extends K, ? extends V> m) //This creates a hash table that is initialized with the elements in m. The capacity of the hash table is set to twice the number of elements in m. The default load factor of 0.75 is used.

Program

```
import java.util.*;
```

```
class HashTableDemo
{
 public static void main(String args[])
 {
  Hashtable< String,Integer> ht = new Hashtable< String,Integer>();
  ht.put("a",new Integer(100));
  ht.put("b",new Integer(200));
  ht.put("c",new Integer(300));
  ht.put("d",new Integer(400));

  Set st = ht.entrySet();
  Iterator itr=st.iterator();
  while(itr.hasNext())
  {
   Map.Entry m=(Map.Entry)itr.next();
   System.out.println(itr.getKey()+" "+itr.getValue());
  }
 }
}
```

Output

a 100

b 200

c 300

d 400

3. Properties

Properties class extends Hashtable class. It is used to maintain list of value in which both key and value are String. Properties class defines two constructors.

- Properties() //This creates a Properties object that has no default values
- Properties (Properties propdefault) //This creates an object that uses propdefault for its default values.

One advantage of Properties over Hashtable is that we can specify a default property that will be useful when no value is associated with a certain key. In Properties class, you can specify a default property that will be returned if no value is associated with a certain key.

```
import java.util.*;
public class Test
```

```
{
  public static void main(String[] args)
  {
    Properties pr = new Properties();
    pr.put("Java", "James Ghosling");
    pr.put("C++", "Bjarne Stroustrup");
    pr.put("C", "Dennis Ritchie");
    pr.put("C#", "Microsoft Inc.");
    Set< ?> creator = pr.keySet();
    for(Object ob: creator)
    {
       System.out.println(ob+" was created by "+ pr.getProperty((String)ob) );
    } }}
```

Output

```
Java was created by James Ghosling
C++ was created by Bjarne Stroustrup
C was created by Dennis Ritchie
C# was created by Microsoft Inc
```

4. Stack

Stack class extends Vector. It follows last-in, first-out principle for the stack elements. It defines only one default constructor

Stack() //This creates an empty stack

If you want to put an object on the top of the stack, call push() method. If you want to remove and return the top element, call pop() method. An EmptyStackException is thrown if you call pop() method when the invoking stack is empty.

You can use peek() method to return, but not remove, the top object. The empty() method returns true if nothing is on the stack. The search() method determines whether an object exists on the stack and returns the number of pops that are required to bring it to the top of the stack.

Example of Stack

```
import java.util.*;
class StackDemo {
public static void main(String args[]) {
Stack st = new Stack();
st.push(11);
st.push(22);
st.push(33);
```

```
st.push(44);
st.push(55);
Enumeration e1 = st.elements();
while(e1.hasMoreElements())
System.out.print(e1.nextElement()+" ");
st.pop();
st.pop();
System.out.println("\nAfter popping out two elements");
Enumeration e2 = st.elements();
while(e2.hasMoreElements())
System.out.print(e2.nextElement()+" ");
}
}
```

Output

```
11 22 33 44 55
After popping out two elements
11 22 33
```

5. Vector

Vector is similar to **ArrayList** which represents a dynamic array.There are two differences between **Vector** and **ArrayList**. First, Vector is synchronized while ArrayList is not, and Second, it contains many legacy methods that are not part of the Collections Framework. With the release of JDK 5, Vector also implements Iterable. This means that Vector is fully compatible with collections, and a Vector can have its contents iterated by the for-each loop.

Vector class has following four constructor

- Vector() //This creates a default vector, which has an initial size of 10.
- Vector(int size) //This creates a vector whose initial capacity is specified by size.
- Vector(int size, int incr) //This creates a vector whose initial capacity is specified by size and whose increment is specified by incr. The increment specifies the number of elements to allocate each time when a vector is resized for addition of objects.
- Vector(Collection c) //This creates a vector that contains the elements of collection c.

Method	**Description**
void addElement(E element)	adds element to the Vector

E elementAt(int index)	returns the element at specified index
Enumeration elements()	returns an enumeration of element in vector
E firstElement()	returns first element in the Vector
E lastElement()	returns last element in the Vector
void removeAllElements()	removes all elements of the Vector

Example of Vector

```
import java.util.*;
public class Test
{
  public static void main(String[] args)
   {
     Vector ve = new Vector();
      ve.add(10);
      ve.add(20);
      ve.add(30);
      ve.add(40);
      ve.add(50);
      ve.add(60);

      Enumeration en = ve.elements();
      while(en.hasMoreElements())
      {
         System.out.println(en.nextElement());
      }
   }
}
```

Output

```
10
20
30
40
```

50

60

5. A) Explain the steps involved in executing servlet.

Servlets Tomcat: Install Tomcat server of your choice version by downloading from Apache site. Many versions of Tomcat can be found in the following link.

http://tomcat.apache.org/download-60.cgi

Check with the documentation of what JDK or JRE version is compatible to the specific **Tomcat** version. For example I loaded **Tomcat 5.0** and is compatible with **JDK 6.0**.

While installation, it asks the port number and I have entered **8888** (default is displayed as 8080. I have not preferred 8080 for the reason on many systems **Oracle** server will be working on 8080. For this reason better choose a different port number). Later, give your own password.

After installing Tomcat Server on your machine follow the below mentioned steps :

a.Create directory structure for your application.

b.Create a Servlet

c.Compile the Servlet

d.Create Deployement Descriptor for your application

e.Start the server and deploy the application

Step 1: When installed, Tomcat gives many folders of which a few are given hereunder used for execution. See the following one.

C:\Program Files\Apache Software Foundation\Tomcat 5.0\common\lib\servlet-api.jar;

Keep the above JAR file in the **classpath** of **Windows Environment Variables**, else the servlet program will not be compiled.

Note: Now, a fresher should be careful here in the following steps. Steps are very simple but should be followed carefully. Any small mistake committed, Tomcat simply refuses to execute your servlet.

Step 2: Creating your own directory structure.

You also get the following folders.

C:\Program Files\Apache Software Foundation\Tomcat 5.0\webapps

In the above **webapps** folder create your own new folder. I created and named it as "**india**".

Step 3: Observe the following directory structure.

C:\Program Files\Apache Software Foundation\Tomcat 5.0\webapps\ROOT\WEB-INF

Just copy the above **WEB-INF** folder (ofcourse, along with its subdirectories) into **india** folder.

When you did, now you get the following structure. Check it.

C:\Program Files\Apache Software Foundation\Tomcat 5.0\webapps\india\WEB-INF

Note 1: *Do not confuse now with the WEB-INF folder as WEB-INF exists in two places – one in* ***ROOT*** *and one in* ***india****.*

Note 2: *Tomcat is case-sensitive. Do not write web-inf instead of WEB-INF etc.*

Remember, now onwards when I talk about ***WEB-INF*** *folder, I mean the* ***WEB-INF*** *available in* ***india*** *and not in* ***ROOT****. This is very important.*

In \webapps\india\WEB-INF folder you get automatically one "**classes**" folder and one "**web.xml**" file.

Note 3. Now what is to be done?

Following next steps for Servlets Tomcat deployment and execution.

Step 4: Deployment of Servlet

Write a servlet program, say Validation.java in your current directory and compile it as usual with **javac** command and obtain Validation.class. Copy the **Validation.class** file from your current directory to **classes** folder available in the following classpath.

C:\Program Files\Apache Software Foundation\Tomcat 5.0\webapps\india\WEB-INF\clssses

Copying the **.class file** of servlet to **classes** folder is known as deployment.

Step 5: Giving the alias name to the Servlet with Deployment Descriptor.

Open the **web.xml** file available in the following folder.

C:\Program Files\Apache Software Foundation\Tomcat 5.0\webapps\india\WEB-INF

In the **web.xml** file add the following code just before </web-app> tag.

```
  <servlet>
    <servlet-name>abcd</servlet-name>
    <servlet-class>Validation</servlet-class>
  </servlet>
<servlet-mapping>
    <servlet-name>abcd</servlet-name>
    <url-pattern>/roses</url-pattern>
  </servlet-mapping>
```

5. B) What are the advantages of servlet over traditional CGI?

Efficient. With traditional CGI, a new process is started for each HTTP request. If the CGI program does a relatively fast operation, the overhead of starting the process can dominate the execution time. With servlets, the Java Virtual Machine stays up, and each request is handled by a

lightweight Java thread, not a heavyweight operating system process. Similarly, in traditional CGI, if there are *N* simultaneous request to the same CGI program, then the code for the CGI program is loaded into memory N times. With servlets, however, there are *N* threads but only a single copy of the servlet class.

Convenient. Hey, you already know Java. Why learn Perl too? Besides the convenience of being able to use a familiar language, servlets have an extensive infrastructure for automatically parsing and decoding HTML form data, reading and setting HTTP headers, handling cookies, tracking sessions, and many other such utilities.

Powerful. Java servlets let you easily do several things that are difficult or impossible with regular CGI. For one thing, servlets can talk directly to the Web server (regular CGI programs can't). This simplifies operations that need to look up images and other data stored in standard places. Servlets can also share data among each other, making useful things like database connection pools easy to implement. They can also maintain

information from request to request, simplifying things like session tracking and caching of previous computations.

Portable. Servlets are written in Java and follow a well-standardized API. Consequently, servlets written for, say I-Planet Enterprise Server can run virtually unchanged on Apache, Microsoft IIS, or Web Star. Servlets are supported directly or via a plug in on almost every major Web server.

Inexpensive. There are a number of free or very inexpensive Web servers available that are good for "personal" use or low-volume Web sites. However, with the major exception of Apache, which is free, most commercial-quality Web servers are relatively expensive. Nevertheless, once you have a Web server, no matter the cost of that server, adding servlet support to it (if it doesn't come preconfigured to support servlets) is generally free or cheap.

6 Define JSP. Explain different type of JSP tags by taking suitable example.

JavaServer Pages (JSP) is a technology for developing Webpages that supports dynamic content. This helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.

A JavaServer Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.

Using JSP, you can collect input from users through Webpage forms, present records from a database or another source, and create Webpages dynamically.

JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages, and sharing information between requests, pages etc.

Declaration tag

Declaration tag is a block of java code for declaring class wide variables, methods and classes. Whatever placed inside these tags gets initialized during JSP initialization phase and has class scope. JSP container keeps this code outside of the service method (_jspService()) to make them class level variables and methods.

As we know that variables can be initialized using scriptlet too but those declaration being placed inside _jspService() method which doesn't make them class wide declarations. On the other side, **declaration tag** can be used for defining class level variables, methods and classes.

Syntax of declaration tag:

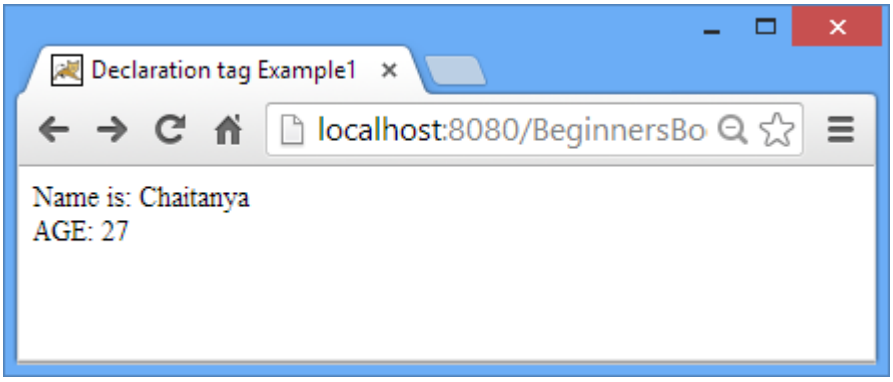
<%! Declaration %>

Example 1: Variables declaration

In this example we have declared two variables inside declaration tag and displayed them on client using expression tag.

```
<html>
<head>
<title>Declaration tag Example1</title>
</head>
<body>
<%! String name="Chaitanya"; %>
<%! int age=27; %>
<%= "Name is: "+ name %><br>
<%= "AGE: "+ age %>
</body>
</html>
```

Output:



SP Expression Tag – JSP Tutorial

Expression tag evaluates the expression placed in it, converts the result into String and send the result back to the client through response object. Basically it writes the result to the client(browser).

Syntax of expression tag in JSP:

<%= expression %>

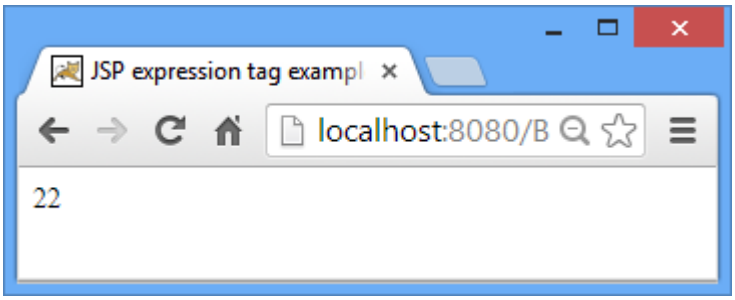
JSP expression tag Examples

Example 1: Expression of values

Here we are simply passing the expression of values inside expression tag.

```
<html>
<head>
<title>JSP expression tag example1</title>
</head>
<body>
<%= 2+4*5 %>
</body>
</html>
```

Output:



JSP Scriptlets

Scriptlets are nothing but java code enclosed within **<% and %> tags**. JSP container moves the statements enclosed in it to **_jspService()** method while generating servlet from JSP. The reason of copying this code to service method is: For each client's request the _jspService() method gets invoked, hence the code inside it executes for every request made by client.

Syntax of Scriptlet:

[code language="java"]<% Executable java code%>[/code]

JSP to Servlet transition for Scriptlet –

As I stated in my previous tutorials that JSP doesn't get executed directly, it first gets converted into a Servlet and then Servlet execution happens as normal. Also, I explained in first para that while translation from JSP to servlet, the java code is copied from scriptlet to _jspService() method. Lets see how that happens.

Sample JSP code:

```
[code language="html"]
<H3> Sample JSP </H3>
<% myMethod();%>
[/code]
```

Note: Semicolon at the end of scriptlet.

Corresponding translated Servlet code for above JSP code:

[code language=”java”]
public void _jspService(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html");
HttpSession session = request.getSession();
JspWriter out = response.getWriter();
out.println("<H2>Sample JSP</H2>");
myMethod();
}[/code]

JSP Directives – Page, Include and TagLib

Directives control the processing of an entire JSP page. It gives directions to the server regarding processing of a page.

Syntax of Directives:

<%@ directive name [attribute name=“value” attribute name=“value” .....]%>

There are three types of Directives in JSP:
1) Page Directive
2) Include Directive
3) TagLib Directive

1) Page Directive

There are several attributes, which are used along with Page Directives and these are –

import:

This attribute is used to import packages. While doing coding you may need to include more than one packages, In such scenarios this page directive’s attribute is very useful as it allows you to mention more than one packages at the same place separated by commas (,). Alternatively you can have multiple instances of page element each one with different package.

Syntax of import attribute –

<%@page import="value"%>

Here value is package name.

Example of import- The following is an example of how to import more than one package using import attribute of page directive.

<%@page import="java.io.*%>
<%@page import="java.lang.*%>
<%--Comment: OR Below Statement: Both are Same--%>
<%@page import="java.io.*, java.lang.*"%>

Include Directive

Include directive is used to copy the content of one JSP page to another. It's like including the code of one file into another.

Syntax of Include Directive:

<%@include file ="value"%>

here value is the JSP file name which needs to be included. If the file is in the same directory then just specify the file name otherwise complete URL(or path) needs to be mentioned in the value field.

Note: It can be used anywhere in the page.

Example:

<%@include file="myJSP.jsp"%>

You can use the above code in your JSP page to copy the content of myJSP.jsp file. However in this case both the JSP files must be in the same directory. If the myJSP.jsp is in the different directory then instead of just file name you would need to specify the complete path in above code.

Must Read: Include directive in detail with example.

3) Taglib Directive

This directive basically allows user to use Custom tags in JSP. we shall discuss about Custom tags in detail in coming JSP tutorials. Taglib directive helps you to declare custom tags in JSP page.

Syntax of Taglib Directive:

<%@taglib uri ="taglibURI" prefix="tag prefix"%>

Where URI is uniform resource locator, which is used to identify the location of custom tag and tag prefix is a string which can identify the custom tag in the location identified by uri.

Example of Taglib:

```
<%@ taglib uri="http://www.sample.com/mycustomlib" prefix="demotag" %>
<html>
<body>
<demotag:welcome/>
</body>
</html>
```

As you can see that uri is having the location of custom tag library and prefix is identifying the prefix of custom tag.
Note: In above example – <demotag: welcome> has a prefix demotag.

7. Explain the classes and interfaces of javax.servlet package.

The javax.servlet.http package contains a number of interfaces and classes that are commonly used by servlet developers. You will see that its functionality makes it easy to build servlets that work with HTTP requests and responses.

Interface	Description
HttpServletRequest	Enables servlets to read data from an HTTP request.
HttpServletResponse	Enables servlets to write data to an HTTP response.
HttpSession	Allows session data to be read and written.
HttpSessionBindingListener	Informs an object that it is bound to or unbound from a session.

Class	Description
Cookie	Allows state information to be stored on a client machine.
HttpServlet	Provides methods to handle HTTP requests and responses.
HttpSessionEvent	Encapsulates a session-changed event.
HttpSessionBindingEvent	Indicates when a listener is bound to or unbound from a session value, or that a session attribute changed.

The HttpServletRequest Interface

The HttpServletRequest interface enables a servlet to obtain information about a client request.

The HttpServletResponse Interface

The HttpServletResponse interface enables a servlet to formulate an HTTP response to a client. Several constants are defined. These correspond to the different status codes that can be assigned to an HTTP response. For example, SC_OK indicates that the HTTP request succeeded, and SC_NOT_FOUND indicates that the requested resource is not available.

The HttpSession Interface

The HttpSession interface enables a servlet to read and write the state information that is associated with an HTTP session.

The HttpSessionBindingListener Interface

The HttpSessionBindingListener interface is implemented by objects that need to be notified when they are bound to or unbound from an HTTP session. The methods that are invoked when an object is bound or unbound are

void valueBound(HttpSessionBindingEvent *e*)

void valueUnbound(HttpSessionBindingEvent *e*)

Here, *e* is the event object that describes the binding.

The Cookie Class

The Cookie class encapsulates a cookie. A *cookie* is stored on a client and contains state information. Cookies are valuable for tracking user activities. A servlet can write a cookie to a user's machine via the addCookie( ) method of the HttpServletResponse interface. The data for that cookie is then included in the header of the HTTP response that is sent to the browser. The names and values of cookies are stored on the user's machine. Some of the information that is saved for each cookie includes the following:

• The name of the cookie

• The value of the cookie

• The expiration date of the cookie

• The domain and path of the cookie

There is one constructor for **Cookie**. It has the signature shown here:

Cookie(String *name*, String *value*)

Here, the name and value of the cookie are supplied as arguments to the constructor.

The HttpServlet Class

The HttpServlet class extends GenericServlet. It is commonly used when developing servlets that receive and process HTTP requests.

The HttpSessionEvent Class

HttpSessionEvent encapsulates session events. It extends EventObject and is generated when a change occurs to the session. It defines this constructor:

HttpSessionEvent(HttpSession *session*)

Here, *session* is the source of the event.

HttpSessionEvent defines one method, getSession( ), which is shown here:

HttpSession getSession( )

It returns the session in which the event occurred.

The HttpSessionBindingEvent Class

The HttpSessionBindingEvent class extends HttpSessionEvent. It is generated when a listener is bound to or unbound from a value in an HttpSession object. It is also generated when an attribute is bound or unbound. Here are its constructors:

HttpSessionBindingEvent(HttpSession *session*, String *name*)

HttpSessionBindingEvent(HttpSession *session*, String *name*, Object *val*)

Here, *session* is the source of the event, and *name* is the name associated with the object that is being bound or unbound. If an attribute is being bound or unbound, its value is passed in *val.*

The getName( ) method obtains the name that is being bound or unbound. It is shown here:

String getName( )

The getSession( ) method, shown next, obtains the session to which the listener is being bound or unbound:

HttpSession getSession( )

The getValue( ) method obtains the value of the attribute that is being bound or unbound. It is shown here:

Object getValue( )

8. What is Cookie? List out the methods defined by the cookie and write a java program to add a cookie.

A *cookie* is stored on a client and contains state information. Cookies are valuable for tracking user activities. A servlet can write a cookie to a user's machine via the addCookie( ) method of the HttpServletResponse interface. The data for that cookie is then included in the header of the HTTP response that is sent to the browser. The names and values of cookies are stored on the user's machine. Some of the information that is saved for each cookie includes the following:

• The name of the cookie

• The value of the cookie

• The expiration date of the cookie

• The domain and path of the cookie

There is one constructor for **Cookie**. It has the signature shown here:

Cookie(String *name*, String *value*)

Here, the name and value of the cookie are supplied as arguments to the constructor.

Methods defined for the cookie class are as follows

Method	Description
Object clone( )	Returns a copy of this object.
String getComment( )	Returns the comment.
String getDomain( )	Returns the domain.
int getMaxAge( )	Returns the maximum age (in seconds).
String getName( )	Returns the name.
String getPath( )	Returns the path.
boolean getSecure( )	Returns **true** if the cookie is secure. Otherwise, returns **false**.
String getValue( )	Returns the value.
int getVersion( )	Returns the version.
void setComment(String *c*)	Sets the comment to *c*.
void setDomain(String *d*)	Sets the domain to *d*.
void setMaxAge(int *secs*)	Sets the maximum age of the cookie to *secs*. This is the number of seconds after which the cookie is deleted.
void setPath(String *p*)	Sets the path to *p*.
void setSecure(boolean *secure*)	Sets the security flag to *secure*.
void setValue(String *v*)	Sets the value to *v*.
void setVersion(int *v*)	Sets the version to *v*.

The HTML source code for **AddCookie.htm** is shown in the following listing. This page contains a text field in which a value can be entered. There is also a submit button on the page. When this button is pressed, the value in the text field is sent to **AddCookieServlet.java** via an HTTP POST request.

<html>

```
<body>
<center>
<form name="Form1"
method="post"
action="http://localhost:8080/servlets-examples/servlet/AddCookieServlet">
<B>Enter a value for MyCookie:</B>
<input type=textbox name="data" size=25 value="">
<input type=submit value="Submit">
</form>
</body>
</html>
```

The source code for **AddCookieServlet.java** is shown in the following listing. It gets the value of the parameter named “data”. It then creates a **Cookie** object that has the name “MyCookie” and contains the value of the “data” parameter. The cookie is then added to the header of the HTTP response via the **addCookie( )** method. A feedback message is then written to the browser.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class AddCookieServlet extends HttpServlet {
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
// Get parameter from HTTP request.
String data = request.getParameter("data");
// Create cookie.
Cookie cookie = new Cookie("MyCookie", data);
// Add cookie to HTTP response.
response.addCookie(cookie);
// Write output to browser.
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
pw.println("<B>MyCookie has been set to");
pw.println(data);
pw.close();
}}
```

The source code for **GetCookiesServlet.java** is shown in the following listing. It invokes the **getCookies( )** method to read any cookies that are included in the HTTP GET request. The names and values of these

cookies are then written to the HTTP response. Observe that the **getName( )** and **getValue( )** methods are called to obtain this information.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class GetCookiesServlet extends HttpServlet {
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
// Get cookies from header of HTTP request.
Cookie[] cookies = request.getCookies();
// Display these cookies.
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
pw.println("<B>");
for(int i = 0; i < cookies.length; i++) {
String name = cookies[i].getName();
String value = cookies[i].getValue();
pw.println("name = " + name +
"; value = " + value);
}
pw.close();
}
}
```

Observe that the name and value of the cookie are displayed in the browser. In this example, an expiration date is not explicitly assigned to the cookie via the **setMaxAge( )** method of **Cookie**. Therefore, the cookie expires when the browser session ends.