INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal, Hyderabad - 500 043

INFORMATION TECHNOLOGY

PPT On
Web Technologies
B. Tech III Year II Semester

Prepared by:

Mr. A. Krishna Chaitanya, Associate Professor, IT

Mr. B. Venkateswara Rao, Associate Professor, IT

UNIT – I
Introduction to PHP

- A short history of php

- Parsing

- Variables

- Arrays

- Operators

- Functions

- Control Structures

- External Data Files

Background

- PHP is server side scripting system
  - PHP stands for "PHP: Hypertext Preprocessor"
  - Syntax based on Perl, Java, and C
  - Very good for creating dynamic content
  - Powerful, but somewhat risky!
  - If you want to focus on one system for dynamic content, this is a good one to choose

History

- Started as a Perl hack in 1994 by Rasmus Lerdorf (to handle his resume), developed to PHP/FI 2.0
- By 1997 up to PHP 3.0 with a new parser engine by Zeev Suraski and Andi Gutmans
- Version 5.2.4 is current version, rewritten by Zend (www.zend.com) to include a number of features, such as an object model
- Current is version 5
- php is one of the premier examples of what an open source project can be

About Zend

- A Commercial Enterprise
- Zend provides Zend engine for PHP for free
- They provide other products and services for a fee
  - Server side caching and other optimizations
  - Encoding in Zend's intermediate format to protectsource code
  - IDE-a developer's package with tools to make life easier
  - Support and training services
- Zend's web site is a great resource

PHP 5 Architecture

- Zend engine as parser (Andi Gutmans and Zeev Suraski)
- SAPI is a web server abstraction layer
- PHP components now self contained (ODBC, Java, LDAP, etc.)
- This structure is a good general design for software (compare to OSI model, and middleware applications)
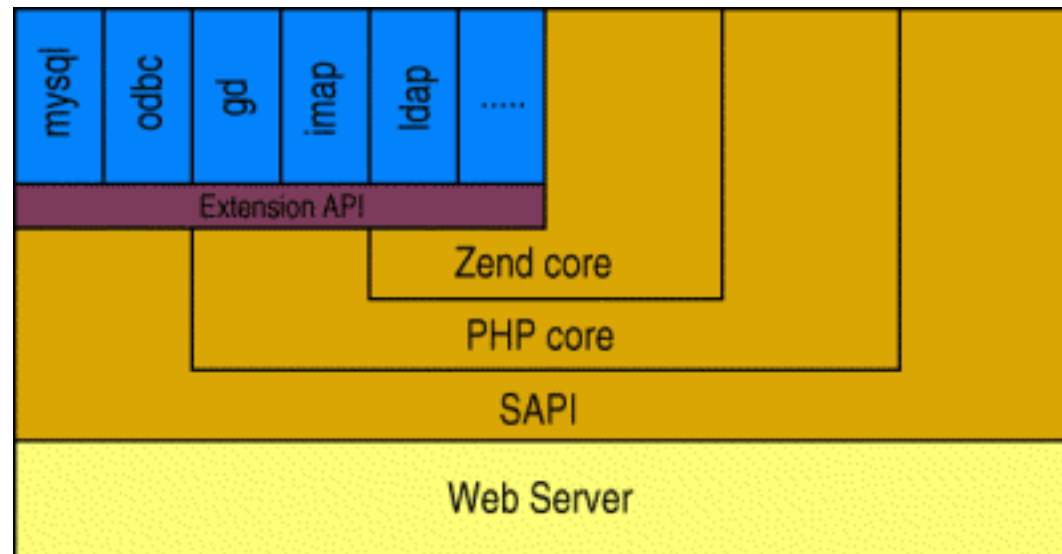
PHP Scripts

- Typically file ends in .php--this is set by the web server configuration
- Separated in files with the <?php   ?> tag
- php commands can make up an entire file, or can be contained in html--this is a choice….
- Program lines end in ";" or you get an error
- Server recognizes embedded script and executes
- Result is passed to browser, source isn't visible

```
<P>
<?php$myvar  = "Hello World!";
echo $myvar;
?>
</P>
```

Parsing

- We've talk about how the browser can read a text file and process it, that's a basic parsing method

- Parsing involves acting on relevant portions of a file and ignoring others

- Browsers parse web pages as they load

- Web servers with server side technologies like php parse web pages as they are being passed out to the browser

- Parsing does represent work, so there is a cost

Two Ways

- You can embed sections of php inside html:

```
<BODY>
<P>
<?php $myvar = "Hello World!";
echo $myvar;
</BODY>
```

- Or you can call html from php:

```
<?php
echo "<html><head><title>Howdy</title>
…
?>
```

What do we know already?

- Much of what we learned about javascript holds true in php (but not all!), and other languages as well

```php
$name = "bil";
echo "Howdy, my name is $name";
echo "What will $name be in this line?";
echo 'What will $name be in this line?';
echo 'What's wrong with this line?';
if ($name == "bil")
  {
  // Hey, what's this?
  echo "got a match!";
  }
```

phpinfo()

- The phpinfo() function shows the php environment

- Use this to read system and server variables, setting stored in php.ini, versions, and modules

- Notice that many of these data are in arrays

- This is the first script you should write...

Variable

- Using the value of a variable as the **name** of a second variable)

  ```
  $a = "hello";
  $$a = "world";
  ```

- Thus:

  ```
  echo "$a ${$a}";
  ```

- Is the same as:

  ```
  echo "$a $hello";
  ```

- But $$a echoes as "$hello"....

Operators

- Arithmetic (+, -, *, /, %) and String (.)
- Assignment (=) and combined assignment

  Bitwise (&, |, ^, ~, <<, >>)

- Comparison (==, ===, !=, !==, <, >, <=, >=)

Operators: The Movie

- Error Control (@)
  - When this precedes a command, errors generated are ignored (allows custom messages)

- Execution (` is similar to the shell_exec() function)
  - You can pass a string to the shell for execution:
    ```
    $output = `ls -al`;
    $output = shell_exec("ls -al");
    ```
  - This is one reason to be careful about user set variables!

- Incrementing/Decrementing
  ```
  ++$a  (Increments by one, then returns $a.)
  $a++  (Returns $a, then increments $a by one.)
  --$a   (Decrements $a by one, then returns $a.)
  $a--   (Returns $a, then decrements $a by one.)
  ```

Son of the Valley of Operators

- Logical

```
$a and $b     And      True if both $a and $b are true.
$a or $b      Or       True if either $a or $b is true.
$a xor $b     Xor      True if either $a or $b is true,
                       but not both.
! $a          Not      True if $a is not true.
$a && $b      And      True if both $a and $b are true.
$a || $b      Or       True if either $a or $b is true.
```

- The two ands and ors have different precedence rules, "and" and "or" are lower precedence than "&&" and "||"

- Use parentheses to resolve precedence problems or just to be clearer

Control Structures

- Wide Variety available
  - if, else, elseif
  - while, do-while
  - for, foreach
  - break, continue, switch
  - require, include, require_once, include_once
- Mostly parallel to what we've covered already in javascript
- if, elseif, else, while, for, foreach, break and continue

Switch

- Switch, which we've seen, is very useful
- These two do the same things….

```
if ($i == 0) {
    echo "i equals 0";
} elseif ($i == 1) {
    echo "i equals 1";
} elseif ($i == 2) {
    echo "i equals 2";
}
```

```
switch ($i) {
case 0:
    echo "i equals 0";
    break;
case 1:
    echo "i equals 1";
    break;
case 2:
    echo "i equals 2";
    break;
}
```

Example: A Dynamic Table

- I hate writing html tables
- You can build one in php
- This example uses pictures and builds a table with pictures in one column, and captions in another
- The captions are drawn from text files
- I'm using tables, but you could use css for placement easily…

Arrays

- Arrays are lists, or lists of lists, or list of lists of lists, you get the idea--Arrays can be multi-dimensional

- Array elements can be addressed by either by number or by name (strings)

- If you want to see the structure of an array, use the print_r function to recursively print an array inside of pre tags

Text versus Keys

- Text keys work like number keys (well, really, it's the other way around--number keys are just labels)

- You assign and call them the same way, except you have to assign the label to the value or variables, eg:
  echo "$my_text_array[third]";

```
$my_text_array = array(first=>1, second=>2, third=>3);
echo "<pre>";
print_r($my_text_array);
echo "</pre>";
```

Walking Arrays

- Use a loop, eg a foreach loop to walk through an array
- while loops also work for arrays with numeric keys--just set a variable for the loop, and make sure to increment that variable within the loop

```
$colors = array('red', 'blue', 'green', 'yellow');

foreach ($colors as $color) {
    echo"Do  you like $color?\n";
}
```

Cont..

- You can't echo an array directly...
  - You can walk through an echo or print() line by line
  - You can use print_r(), this will show you the structure of complex arrays--that output is to the right, and it's handy for learning the structure of an array

```
Array
(
    [1] => Array
        (
            [sku] => A13412
            [quantity] => 10
            [item] => Whirly Widgets
            [price] => .50
        )

    [2] => Array
        (
            [sku] => A43214
            [quantity] => 142
            [item] => Widget Nuts
            [price] => .05
        )
```

Multidimensional Arrays

- A one dimensional array is a list, a spreadsheet or other columnar data is two dimensional…

- Basically, you can make an array of arrays
  ```
  $multiD = array
      (
      "fruits" => array("myfavorite" => "orange", "yuck" => "banana", "yum" => "apple"),
      "numbers" => array(1, 2, 3, 4, 5, 6),
      "holes"   => array("first", 5 => "second", "third")
      );
  ```

- The structure can be built array by array, or declared with a single statement

- You can reference individual elements by nesting:
  ```
  echo "<p>Yes, we have no " . $multiD["fruits"]["yuck"] . " (ok by me).</p>";
  ```

- print_r() will show the entire structure, but don't forget the pre tags

Getting Data into arrays

- You can directly read data into individual array slots via a direct assignment:
$pieces[5] = "poulet resistance";
- From a file:
  - Use the file command to read a delimited file (the delimiter can be any unique char):
$pizza = file(./our_pizzas.txt)
  - Use explode to create an array from a line within a loop:
$pieces = explode(" ", $pizza);

The Surface

- The power of php lies partially in the wealth of functions---for example, the 40+ array functions
  - array_flip() swaps keys for values
  - array_count_values() returns an associative array of all values in an array, and their frequency
  - array_rand() pulls a random element
  - array_unique() removes duppies
  - array_walk() applies a user defined function to each element of an array (so you can dice all of a dataset)
  - count() returns the number of elements in an array
  - array_search() returns the key for the first match in an array

Forms: how they work

- We need to know..

1. How forms work.

2. How to write forms in XHTML.

3. How to access the data in PHP.

How forms work

User requests a particular URL

XHTML Page supplied with Form

User fills in form and submits.

Another URL is requested and the Form data is sent to this page either in URL or as a separate piece of data.

XHTML Response
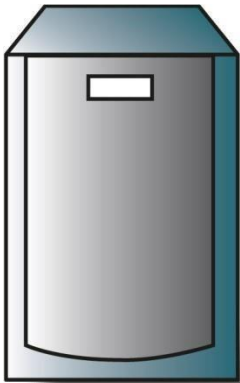
Web Server

User

XHTML Form

- The form is enclosed in form tags..

```
<form
  action="path/to/submit/page"
  method="get">
<!-- form contents -->
</form>
```

Form tags

- **`action=`**"…" is the page that the form should submit its data to.
- **`method=`**"…" is the method by which the form data is submitted. The option are either **`get`** or **`post`**. If the method is get the data is passed in the url string, if the method is post it is passed as a separate file.

Form fields: text input

- Use a text input within form tags for a single line freeform text input.

```
<label for="fn">First Name</label>
<input type="text"
        name="firstname"
        id="fn"
        size="20"/>
```

Form tags

- **name=**"…" is the name of the field. You will use this name in PHP to access the data.
- **id=**"…" is label reference string – this should be the same as that referenced in the **<label>** tag.
- **size=**"…" is the length of the displayed text box (number of characters).

Form fields: password input

- Use a starred text input for passwords.

```
<label for="pw">Password</label>
<input type="password"
       name="passwd"
       id="pw"
       size="20"/>
```
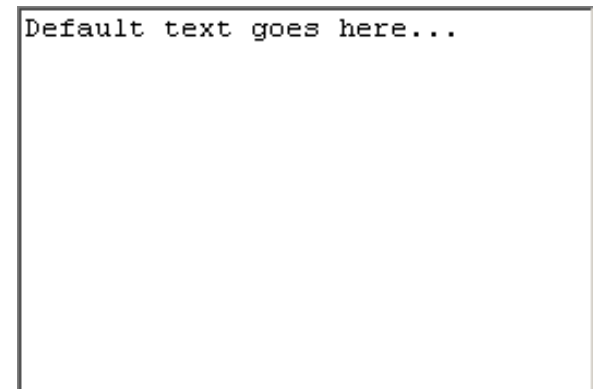
Form fields: text input

- If you need more than 1 line to enter data, use a textarea.

```
<label for="desc">Description</label>
<textarea name="description"
          id="desc"
          rows="10" cols="30">
Default text goes here…
</textarea>
```

Form fields: text area

- **name=**"…" is the name of the field. You will use this name in PHP to access the data.
- **id=**"…" is label reference string – this should be the same as that referenced in the **<label>** tag.
- **rows=**"…" **cols=**".." is the size of the displayed text box.

Form fields: drop down

```
<label for="tn">Where do you live?</label>
<select name="town" id="tn">
<option value="swindon">Swindon</option>
<option value="london"
        selected="selected">London</option>
<option value="bristol">Bristol</option>
</select>
```

Form fields: drop down

- **`name=`**"…" is the name of the field.

- **`id=`**"…" is label reference string.

- **`<option value=`**"…" is the actual data sent back to PHP if the option is selected.

- **`<option>`**…**`</option>`** is the value displayed to the user.

- **`selected="selected"`** this option is selected by default.

Form fields: radio buttons

```
<input type="radio"
    name="age"
    id="u30"
    checked="checked"
    value="Under30" />
<label for="u30">Under 30</label>
<br />
<input type="radio"
    name="age"
    id="thirty40"
    value="30to40" />
<label for="thirty40">30 to 40</label>
```

Form fields: radio buttons

- **name=**"…" is the name of the field. All radio boxes with the same name are grouped with only one selectable at a time.
- **id=**"…" is label reference string.
- **value=**"…" is the actual data sent back to PHP if the option is selected.
- **checked=**"checked" this option is selected by default.

Form fields: check boxes

```
What colours do you like?<br />
<inputtype="checkbox"
      name="colour[]"
      id="r"
      checked="checked"
      value="red" />
<label for="r">Red</label>
<br />
<inputtype="checkbox"
      name="colour[]"
      id="b"
      value="blue" />
<label for="b">Blue</label>
```

Form fields: check boxes

- **name=**"…" is the name of the field. Multiple checkboxes can be selected, so if the button are given the same name, they will overwrite previous values. The exception is if the name is given with square brackets – an array is returned to PHP.
- **id=**"…" is label reference string.
- **value=**"…" is the actual data sent back to PHP if the option is selected.
- **checked=**"**checked**" this option is selected by default.

Hidden Fields

```
<input type="hidden"
    name="hidden_value"
    value="My Hidden Value" />
```

- **name=**"…" is the name of the field.
- **value=**"…" is the actual data sent back to PHP.

Submit button..

- A submit button for the form can be created with the code:

```
<input type="submit"
       name="submit"
       value="Submit" />
```

Fieldset

- In XHTML 1.0, all inputs must be grouped within the form into <u>fieldsets</u>. These represent logical divisions through larger forms. For short forms, all inputs are contained in a single fieldset.

```
<form>
<fieldset>
<input … />
<input … />
</fieldset>
<fieldset>
<input … />
<input … />
</fieldset>
</form>
```

In PHP...

- The form variables are available to PHP in the page to which they have been submitted.
- The variables are available in two superglobal arrays created by PHP called $_POST and $_GET.

Access data

- Access submitted data in the relevant array for the submission type, using the input name as a key.

```
<form action="path/to/submit/page"
        method="get">
<input type="text" name="email">
</form>


$email = $_GET['email'];
```

A warning..

NEVER TRUST USER INPUT

- Always check what has been input.
- Validation can be undertaken using Regular expressions or in-built PHP functions.

A useful tip..

- I find that storing the validated data in a different array to the original useful.

- I often name this array 'clean' or something similarly intuitive.

- I then *only* work with the data in $clean, and never refer to $_POST/$_GET again.

Filter example

```php
$clean = array();

if (ctype_alnum($_POST['username']))

{

$clean['username'] = $_POST['username'];

}
```

Initialise an array to store filtered data.

Filter example

```php
$clean = array();

if (ctype_alnum($_POST['username'])))

{

$clean['username'] = $_POST['username'];

}
```

Inspect username to make sure
that it is alphanumeric.

Filter example

```
$clean = array();

if (ctype_alnum($_POST['username']))

{

$clean['username'] = $_POST['username'];

}
```

If it is, store it in the array.

Is it submitted?

- We also need to check before accessing data to see if the data is submitted, use isset() function.

```php
if (isset($_POST['username'])) {
  // perform validation
}
```

What is form validation?

- **validation:** ensuring that form's values are correct
- some types of validation:
    - preventing blank values (email address)
    - ensuring the type of values
        - integer, real number, currency, phone number, Social Security number, postal
    - address, email address, date, credit card number, …
    - ensuring the format and range of values (ZIP code must be a 5-digit integer)
    - ensuring that values fit together (user types email twice, and the two must match)

A real Form that uses validation

Client vs. server-side validation

- Validation can be performed:
  - **client-side** (before the form is submitted)
    - can lead to a better user experience, but not secure (why not?)
  - **server-side** (in PHP code, after the form is submitted)
    - needed for truly secure validation, but slower
  - both
  - best mix of convenience and security, but requires most effort to program

Basic server-side validation code

```php
$city = $_REQUEST["city"];
$state = $_REQUEST["state"];
$zip = $_REQUEST["zip"];
if (!$city || strlen($state) != 2 || strlen($zip) !=
5) {
?>
    <h2>Error, invalid city/state submitted.</h2>
<?php
}
?>
```
PHP

- basic idea: examine parameter values, and if they are bad, show an error message and abort

Basic server-side validation code

- validation code can take a lot of time / lines to write

  - How do you test for integers vs. real numbers vs. strings?

  - How do you test for a valid credit card number?

  - How do you test that a person's name has a middle initial?

  - How do you test whether a given string matches a particular complex format?

Regular expressions

```
[a-z]at              #cat, rat, bat…
[aeiou]
[a-zA-Z]
[^a-z]               #not a-z
[[:alnum:]]+         #at least one alphanumeric char
(very) *large        #large, very very very large…
(very){1, 3}            #counting "very" up to 3
^bob                 #bob at the beginning
com$                 #com at the end
```

- Regular expression: a pattern in a piece of text
  - PHP has:
    - POSIX
    - **Perl regular expressions**

Delimiters

```
/[a-z]/at              #cat, rat, bat…
#[aeiou]#
/[a-zA-Z]/
~[^a-z]~               #not a-z
/[[:alnum:]]+/         #at least onealphanumeric char
#(very) *#large        #large, very very very large…
~(very){1, 3}~              #counting "very" up to 3
/^bob/                      #bob at the beginning
/com$/                      #com at the end

/http:\/\
// #http://#           #better readability
```

- Used for Perl regular expressions (preg)

Basic Regular Expression

/abc/

- in PHP, regexes are strings that begin and end with /

- the simplest regexes simply match a particular substring

- the above regular expression matches any string containing "abc":
  - YES: "abc", "abcdef", "defabc", ".=.abc.=.", ...
  - NO: "fedcba", "ab c", "PHP", ...

Wildcards

- A dot . matches any character except a \n line break

  - "/.oo.y/" matches "Doocy", "goofy", "LooNy", …

- A trailing i at the end of a regex (after the closing /) signifies a case-insensitive match

  - "/xen/i" matches "Xenia", "xenophobic", "Xena the warrior princess", "XEN technologies" …

Special characters: |, (), ^, \

- | means *OR*
  - "/abc|def|g/" matches "abc", "def", or "g"
  - There's no *AND* symbol. Why not?
- () are for grouping
  - "/(Homer|Marge) Simpson/" matches "Homer Simpson" or "Marge Simpson"
- ^ matches the beginning of a line; $ the end
  - "/^<!--$/" matches a line that consists entirely of "<!--"

Special characters: |, (), ^, \

- \ starts an escape sequence
  - many characters must be escaped to match them literally: / \ $. [ ] ( ) ^ * + ?
  - "/<br \/>/" matches lines containing <br /> tags

Quantifiers: *, +, ?

- * means 0 or more occurrences
  - "/abc*/" matches "ab", "abc", "abcc", "abccc", …
  - "/a(bc)*/" matches "a", "abc", "abcbc", "abcbcbc", …
  - "/a.*a/" matches "aa", "aba", "a8qa", "a!?_a", …
- \+ means 1 or more occurrences
  - "/a(bc)+/" matches "abc", "abcbc", "abcbcbc", …
  - "/Goo+gle/" matches "Google", "Gooogle", "Goooogle", …
- ? means 0 or 1 occurrences
  - "/a(bc)?/" matches "a" or "abc"

More quantifiers: {min,max}

- {min,max} means between min and max occurrences (inclusive)
  - "/a(bc){2,4}/" matches "abcbc", "abcbcbc", or "abcbcbcbc"
- min or max may be omitted to specify any number
  - {2,} means 2 or more
  - {,6} means up to 6
  - {3} means exactly 3

Character sets: []

- [] group characters into a character set; will match any single character from the set
  - "/[bcd]art/" matches strings containing "bart", "cart", and "dart"
  - equivalent to "/(b|c|d)art/" but shorter
- inside [], many of the modifier keys act as normal characters
  - "/what[!*?]*/" matches "what", "what!", "what?**!", "what??!",
- What regular expression matches DNA (strings of A, C, G, or T)?

Character ranges: [start-end]

- inside a character set, specify a range of characters with -
  - "/[a-z]/" matches any lowercase letter
  - "/[a-zA-Z0-9]/" matches any lower- or uppercase letter or digit
- an initial ^ inside a character set negates it
  - "/[^abcd]/" matches any character other than a, b, c, or d

Character ranges: [start-end]

- inside a character set, - must be escaped to be matched
  - "/[+\-]?[0-9]+/" matches an optional + or -, followed by at least one digit
- What regular expression matches letter grades such as A, B+, or D- ?

Escape sequences

- special escape sequence character sets:
  - \d matches any digit (same as [0-9]); \D any non-digit ([^0-9])
  - \w matches any "word character" (same as [a-zA-Z_0-9]); \W any non-word
- char
  - \s matches any whitespace character (, \t, \n, etc.); \S any non-whitespace
- What regular expression matches dollar amounts of at least $100.00 ?

Regular expressions in PHP

- regex syntax: strings that begin and end with /, such as "/[AEIOU]+/"

function	description
preg_match(regex, string)	returns TRUE if string matches regex
preg_replace(regex, replacement, string)	returns a new string with all substrings that match regex replaced by replacement
preg_split(regex, string)	returns an array of strings from given string broken apart using the given regex as the delimiter (similar to explode but more powerful)

Regular expressions example

```php
echo preg_match ('/test/', "a testof preg_match");
echo preg_match ('/tutorial/',"a testof preg_match
");

$matchesarray[0] = "http://www.tipsntutorials.com/"
$matchesarray[1] = "http://"
$matchesarray[2] = "www.tipsntutorials.com/"
preg_match ('/(http://)(.*)/', "http://www.tipsntuto
rials.com/", $matchesarray)
```

PHP

Regular expressions example

```php
# replace vowels with stars
$str = "the quick brown fox";
$str = preg_replace("/[aeiou]/", "*", $str);
# "th* q**ck br*wn f*x"
# break apart into words
$words = preg_split("/[ ]+/", $str);
# ("th*","q**ck", "br*wn", "f*x")
# capitalize words that had 2+ consecutive vowels
for ($i = 0; $i < count($words); $i++) {
if (preg_match("/\\\*{2,}/", $words[$i])) {
$words[$i] = strtoupper($words[$i]);
}
} # ("th*", "Q**CK", "br*wn", "f*x")
```

PHP

PHP form validation

```php
$state = $_REQUEST["state"];
if (!preg_match("/[A-Z]{2}/", $state)) {
?>
<h2>Error, invalid state submitted.</h2>
<?php
}
```
PHP

- using preg_match and well-chosen regexes allows you to quickly validate query parameters against complex patterns

Another PHP experiment

- Write a PHP script that tests whether an e-mail address is input correctly. Test using valid and invalid addresses

- Use array

- Use function

UNIT-2

XML

Extensible Markup Language

XML

- **XML is based on SGML: Standard Generalized Markup Language**
- **HTML and XML are both based on SGML**
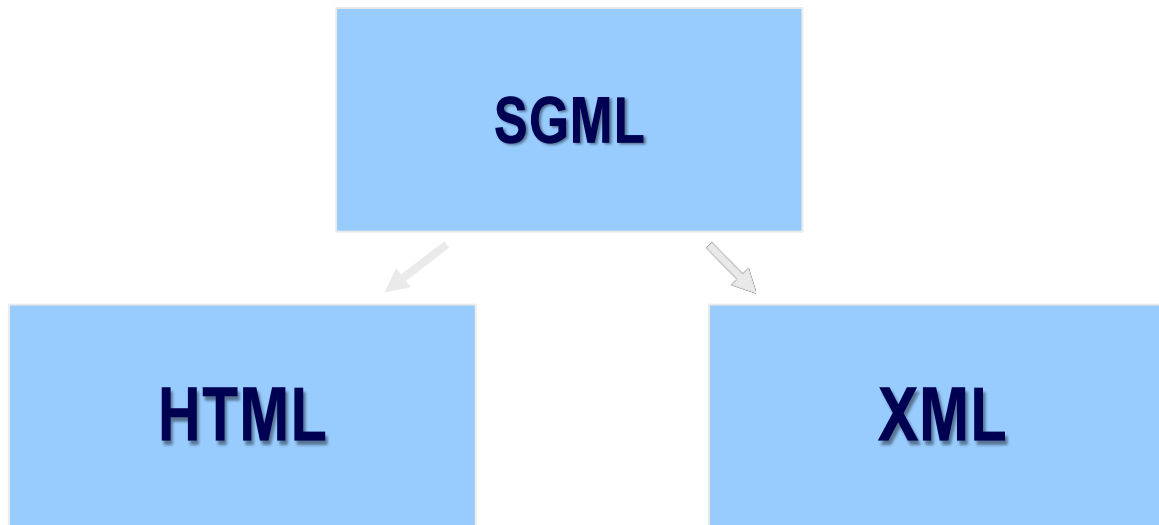
Diff b/w HTML & XML

- **HTML was designed to display data and to focus on how data looks.**
- **HTML is about displaying information, while XML is about describing information**

_ _

- **XML was designed to describe data and to focus on what data is.**
- **It is important to understand that XML was designed to store, carry, and exchange data. XML was not designed to display data.**

What is XML?

- **XML stands for EXtensible Markup Language**
- **XML is a markup language much like HTML**
- **XML was designed to describe data**
- **XML tags are not predefined. You must define your own tags**
- **Extensible: can be extended to lots of different applications.**
- **Markup language: language used to mark up data.**
- **Meta Language: Language used to create other languages.**
- **XML uses a Document Type Definition (DTD) or an XML Schema to describe the data**

HTML File

```
<HTML>
    <BODY>
            <H1>Harry Potter</H1>
            <H2>J. K. Rowling</H2>
            <H3>1999</H3>
            <H3>Scholastic</H3>
    </BODY>
</HTML>
```

XML File

```
<BOOK>
    <TITLE>Harry Potter</TITLE>
    <AUTHOR>J. K. Rowling</AUTHOR>
    <DATE>1999</DATE>
    <PUBLISHER>Scholastic</PUBLISHER>
</BOOK>
```

XML ROLE

XML Advantages

- **1. XML is used to Exchange Data**
  - **With XML, data can be exchanged between incompatible systems**
- **2. XML and B2B**
  - **With XML, financial information can be exchanged over the Internet.**
- **3. XML can be used to Share Data**
  - **With XML, plain text files can be used to share data.**

XML Advantages

- **4. XML is free and extensible**
  - **XML tags are not predefined. we must "invent" your own tags.**
- **5. XML can be used to Store Data**
  - **With XML, plain text files can be used to store data.**
- **6. XML can be used to Create new Languages**
  - **XML is the mother of WAP and WML.**
- **7. HTML focuses on "look and feel"**
  - **XML focuses on the structure of the data.**

XML Example

- <!-- This is a comment -->
- <?xml version="1.0" encoding="ISO-8859-1"?>
- <book>
  - <title>My First XML</title>
  - <prod id="33-657" Media="paper"></prod>
  - <chapter>Introduction to XML
    - <para>What is HTML</para>
    - <para>What is XML</para>
  - </chapter>
  - <chapter>XML Syntax
    - <para>Elements must have a closing tag</para>
    - <para>Elements must be properly nested</para>
  - </chapter>
- </book>

Elements can have different content types.

- **Element content → book**

- **Mixed content → Chapter**

- **Simple content → para**

- **Empty content → prod**

Element Naming rules

- **Names can contain letters, numbers, and other characters**
- **Names must not start with a number or punctuation character**
- **Names must not start with the letters xml (or XML, or Xml, etc)**
- **Names cannot contain spaces**
- **Avoid "-" and "." in names. For example, if you name something "first-name," it could be a mess if your software tries to subtract name from first. Or if you name something "first.name," your software may think that "name" is a property of the object "first."**

Element Naming rules

- **Names should be short and simple**

- **XML documents often have a corresponding database, in which fields exist corresponding to elements in the XML document.**

- **Attribute values must always be enclosed in quotes, but either single or double quotes can be used.**

  Ex: <person sex="female"></person>

XML Attributes

- **XML elements can have attributes. Attributes are used to provide additional information about elements.**

- **In HTML → <IMG SRC="computer.gif">.**

  - **The SRC attribute provides additional information about the IMG element.**

- **In XML →**

  - **<prod id="33-657" media="paper"></prod>**

Use of Elements vs. Attributes

- **Data can be stored in child elements or in attributes.**
- **Attributes:**
- **<person sex="female">**
-       **<firstname>Anna</firstname>**
        <lastname>Smith</lastname>
- **</person>**
- **Elements:**
- **<person>**
     <sex>female</sex>
     <firstname>Anna</firstname>
     <lastname>Smith</lastname>
      </person>

Well-formedness

- **A well-formed XML document conforms to XML syntax rules and constraints, such as:**
  - **The document must contain exactly one root element and all other elements are children of this root element.**
  - **All markup tags must be balanced; that is, each element must have a start and an end tag.**
  - **Elements may be nested but they must not overlap.**
  - **All attribute values must be in quotes.**

Validity

- **According to the XML specification, an XML document is considered valid if it has an associated DTD declaration and it complies with the constraints expressed in the DTD.**

- **To be valid, an XML document must meet the following criteria:**

  - **Be well-formed**

  - **Refer to an accessible DTD-based schema using a Document Type Declaration:**

  <!DOCTYPE>

Document Type Definitions

- **The Document Type Definition (DTD) forms the basis of valid documents because it establishes the grammar of an XML vocabulary, which in turn determines the structure of XML documents.**

- **A DTD is necessary for performing document validation, which is an important part of XML content development and deployment.**

Syntax of DTD

- **ELEMENT is used to declare element names**
- *<!ELEMENT element_name (subelement1,subelement2......subelement(n-1))>*
  - *Ex: <!ELEMENT product (name, type)>*

- ***ATTLIST To declare attributes***
- *<!ATTLIST element_name attr1_name att_type constraints [att2_name att_type constraints...........]>*
  - *<!ATTLIST product name CDATA #REQUIRED>*

Attribute Types

- **Types**   **Description**

- **CDATA**   **Unparsed character data**
- **Enumerated**   **a series of string values**
- **ID**   **A unique identifier**
- **IDREF**   **A reference to an ID declared somewhere**
- **NMTOKEN characters**   **A name consisting of XML token**
- **NMTOKENS XML token**   **Multiple names consisting of characters.**

Internal DTD

<!DOCTYPE root_ele_name [

DTD code

]>

- **InternalDTDs**
  Placing the DTD code in the DOCTYPE tag in this way

products.xml

- *<?xml version="1.0" encoding="UTF-8"?>*
- *<!DOCTYPE products [*
- *<!ELEMENT PRODUCTS (PRODUCT*)>*
- *<!ELEMENT PRODUCT(PID,PNAME,PRICE,DESCR,ISTOCK)>*
- *<!ELEMENT PID (#PCDATA)>*
- *<!ELEMENT PNAME (#PCDATA)>*
- *<!ELEMENT PRICE (#PCDATA)>*
- *<!ELEMENT DESCR (#PCDATA)>*
- *<!ELEMENT ISTOCK (#PCDATA)>*
- *]>*

- *<PRODUCTS>*
- *<PRODUCT>*
- *<PID>1</PID>*
- *<PNAME>XYZ</PNAME>*
- *<PRICE>300.00</PRICE>*
- *<DESCR>XYZ descr</DESCR>*
- *<ISTOCK>1000</ISTOCK>*
- *</PRODUCT>*
- *</PRODUCTS>*

External DTD

- **<!DOCTYPE root_ele_name SYSTEM 'dtd file name'>**
  - **SYSTEM →the definitions are developed and used by the same comp**

  or

- **<!DOCTYPE root_ele_name PUBLIC ' fpi string ' ' dtd url '>**
  - **PUBLIC →if the definition can be used by public**

- **products.dtd**
  <!ELEMENT products (product)>

- *<!ELEMENT product (name, type)>*

- *<!ELEMENT name (#PCDATA)>*

- *<!ELEMENT type (#PCDATA)>*

- **products.xml**
  <?xml version ="1.0"?>

- ***<!DOCTYPE products SYSTEM "products.dtd">***

Qualifier Name Meaning

Qualifier	Name	Meaning
?	Question Mark	Optional (zero or one)
*	Asterisk	Zero or more
+	Plus Sign	One or more

Ex : <!ELEMENT emp_details (emp+)>

Attribute Values

Specification	Specifies...
#REQUIRED	The attribute value must be specified in the document.
#IMPLIED	The value need not be specified in the document. If it isn't, the application will have a default value it uses.
"defaultValue"	The default value to use, if a value is not specified in the document.
#FIXED "fixedValue"	The value to use. If the document specifies any value at all, it must be the same.

<!ATTLIST product name CDATA #REQUIRED>

Prefix URI for namespaces

- Xml

  http://www.w3.org/XML/1998/namespace

- Xsl

  http://www.w3.org/1999/XSL/Transform

- Xsd

  http://www.w3.org/2001/XMLSchema

```
<root>                          <root>
<h:tablexmlns:h="http://www.w3.org/TR/html4/"
>
  <h:tr>
  <h:td>Apples</h:td>
  <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:tablexmlns:f="http://www.w3.org/TR/html4/"
>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
</root>
```

XML Schema

- **XML Schema is an XML-based alternative to DTD.**

- **An XML schema describes the structure of an XML document.**

- **The XML Schema language is also referred to as XML Schema Definition (XSD).**

XML Schema

- **An XML Schema:**
- **defines elements that can appear in a document**
- **defines attributes that can appear in a document**
- **defines which elements are child elements**
- **defines the order of child elements**
- **defines the number of child elements**
- **defines whether an element is empty or can include text**
- **defines data types for elements and attributes**

XML Schema elements

1.Simple elements

2.Complex elements

Simple Elements

- A simple element is an XML element that can contain only text. It cannot contain any other elements or attributes.

- Xml:

  <lastname> abc   </lastname>

- Xml schema:

  <xs:element name="lastname" type="xs:string“ />

Restrictions

<?xml version="1.0" encoding="utf-8" ?>

<xsd:schema
xmlns:xsd=http://www.w3.org/2001/XMLSchema>

<xsd:element name="age">

        <xsd:simpleType>

                <xsd:restriction base="xs:integer">

                        <xsd:minInclusive value="0"/>

                        <xsd:maxInclusive value="120"/>

                </xsd:restriction>

        </xsd:simpleType>

</xsd:element >

```xml
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema
xmlns:xsd=http://www.w3.org/2001/XMLSchema>
<xsd:element name="car">
  <xsd:simpleType>
   <xsd:restriction base="xsd:string">
     <xsd:enumeration value="Audi"/>
     <xsd:enumeration value="Golf"/>
     <xsd:enumeration value="BMW"/>
   </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

```xml
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="initials">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:pattern value="[A-Z][A-Z][A-Z]"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
```

Restriction on length

```
<xsd:element name="password">
  <xsd:simpleType>
    <xsd:restriction base="xs:string">
      <xsd:length value="8"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Complex Elements

- A complex element is an XML element that contains other elements and/or attributes.

```
<xsd:element name="employee">
   <xsd:complexType>
   <xsd:sequence>
<xsd:element name="firstname" type="xsd:string"/>        <xsd:element name="lastname" type="xsd:string"/>
   </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="employee" type="personinfo"/>
<xsd:complexType name="personinfo">
<xsd:sequence>
<xsd:element name="firstname" type="xs:string"/>
 <xsd:element name="lastname" type="xs:string"/>
 </xsd:sequence>
</xsd:complexType>
```

Xml file

```
<college name="snist">
    <strength>1000</strength>
    <branch>6</branch>
    <bname>
        <cse block="1">200</cse>
     <it block="2">200</it>
            <mech block="3">200</mech>
            <ece block="4">200</ece>
            <eee block="5">100</eee>
            <ecm block="6">100</ecm>
    </bname>
 </college>
<college>
..
</college>
```

Diff b/w DTD & XSD

- **DTD supports types ID,IDREF,CDATA etc.,**
- **Schema supports all primitive and user defined data types**
- **DTD supports No specifier, ?, *, + sign**
- **Schema hava minOccurs and maxOccurs attributes**
- **XML Schemas are extensible to future additions**
- **XML Schemas are richer and more powerful than DTDs**
- **XML Schemas are written in XML**

Parsers

- **An XML parser is a piece of code that reads a document and analyzes its structure.**

- **The parser is the engine for interpreting our XML documents**

- **The parser reads the XML and prepares the information for your application.**

How to use a parser

- **1. Create a parser object**

- **2. Pass your XML document to the parser**

- **3. Process the results**

Parser Example

```java
import com.ibm.xml.parser.*;
import java.io.*;
public class SimpleParser
{   public static void main (String a[]) throws Exception
    {     Parser p=new Parser("err");
          FileInputStream fis=new FileInputStream(a[0]);
          TXDocument doc=p.readStream(fis);
doc.printWithFormat(new      OutputStreamWriter(System.out)
    }
}
```

Types of Parsers

- **There are two common APIs that parsers use.**

    – **DOM is the Document Object Model API**

    – **SAX is the Simple API for XML**

Document Object Model (DOM)

- **DOM uses a tree based structure.**
- **DOM reads an entire XML document and builds a Document Object.**
  - The Document object contains the tree structure.
  - The top of the tree is the root node.
  - The tree grows down from this root, defining the child elements.
  - DOM is a W3C standard.
- **Using DOM, we can also perform insert nodes, update nodes, and deleting nodes.**

DOM interfaces

- **Node: The base data type of the DOM. Methods:**

- **Element:**
  - Attr: Represents an attribute of an element.
  - Text: The actual content of an Element or Attribute

- **Document: Represents the entire XML document. A Document object is often referred to as a _DOM tree_.**

Node

getChildNodes()
,
getNodeName(),
getNodeValue(),
hasChildNodes()

NodeList

getLength()
item()

Element

Document

createElement(),

createAttribute()
),

createTextNode(
)
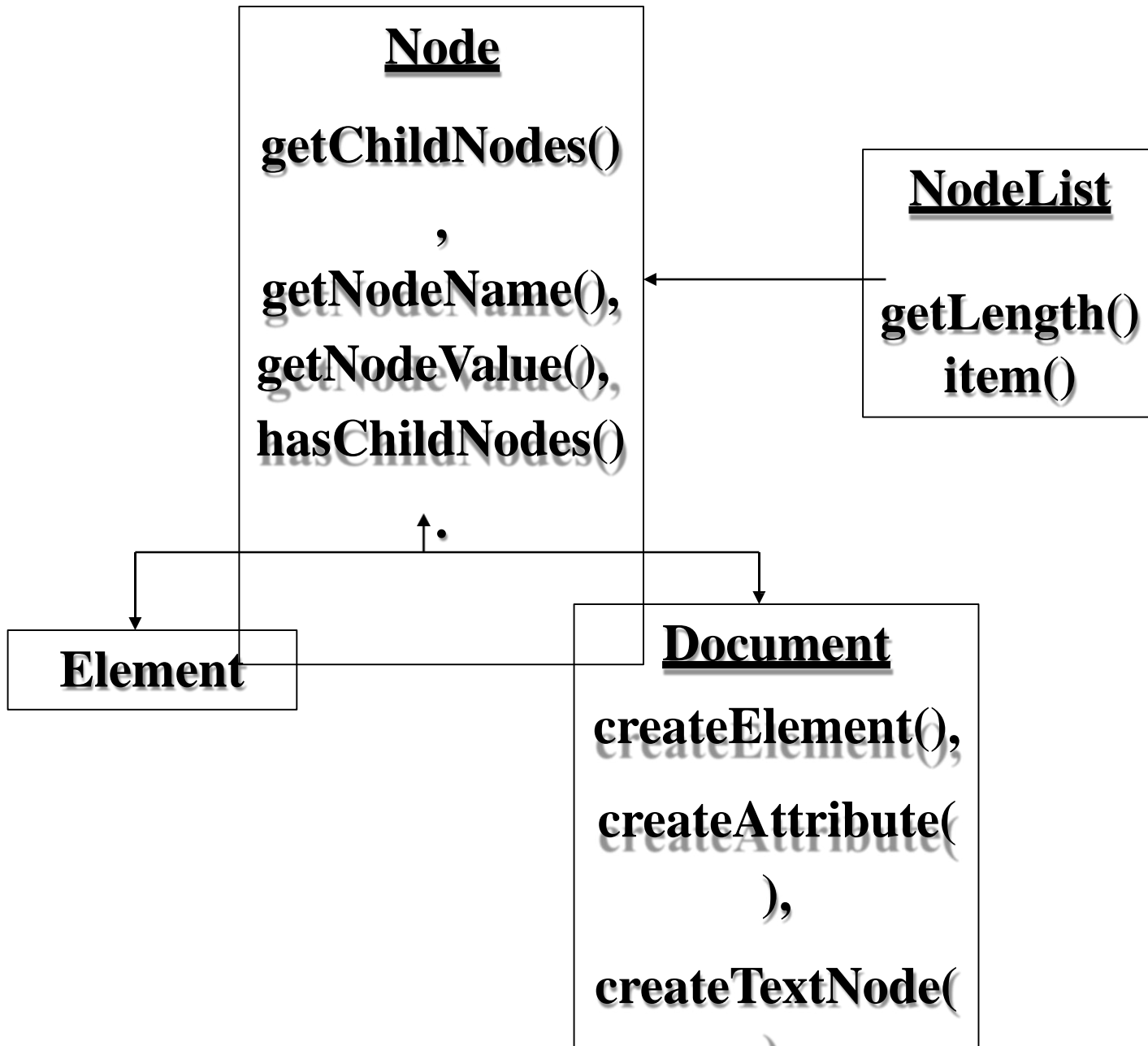
Simple API for XML Parsing (SAX)

- **SAX parsers are event-driven**
  - **The parser fires an event as it parses each XML item.**
  - **The developer writes a class that implements a handler interface for the events that the parser may fire.**

SAX Interface

- **DocumentHandler**
  - Functions in this interface
    - startDocument()
    - startElement()
    - endElement()
    - endDocument()
    - void setDocumentLocator(Locator)
    - void characters(char[ ],int start,intlength)
      - This event fires when text data is found in the XML Document

class HandlerBase is a sub class of DocumentHandler also called Adapter Class.

Diff b/w DOM & SAX

DOM	SAX
Uses more memory and has more functionality	**Uses less memory and provides less functionality**
The entire file is stored in an internal Document object. This may consume many resources	**The developer must handle each SAX event before the next event is fired.**
For manipulation of the document, DOM is best choice	**For simple parsing and display SAX will work great**

UNIT-3

Servlets

The Servlet Life Cycle

- Overview of the Life Cycle

- Birth of a Servlet

- Life of a Servlet

- Death of a Servlet

Life of a Servlet

- Birth: Create and initialize the servlet
  - Important method: init()
- Life: Handle 0 or more client requests
  - Important method: service()
- Death: Destroy the servlet
  - Important method: destroy()

The init() method

- The init() method is called when the servlet is first requested by a browser.
- It is <u>not</u> called again for each request.
- Used for <u>one-time initialization</u>.
- There are two versions of the init() method:
    - Version 1: takes no arguments
    - Version 2: takes a **servletConfig** object as an argument.
- We will focus only on the first option.

Simple Example

▢ The init() method is a good place to put any initialization variables.

▢ For example, the following servlet records its Birth Date/time…

```java
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Birth extends HttpServlet {
    Date birthDate;

    // Init() is called first
    public void init() throws ServletException {
        birthDate = new Date();
    }
```

```java
//  Handle an HTTP GET Request
public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException {
                response.setContentType("text/plain");
                PrintWriter out = response.getWriter();
                out.println ("I was born on:  "+birthDate);
                out.close();
    }
}
```

Service() Method

- Each time the server receives a request for a servlet, the server spawns a new thread and calls the servlet's service () method.
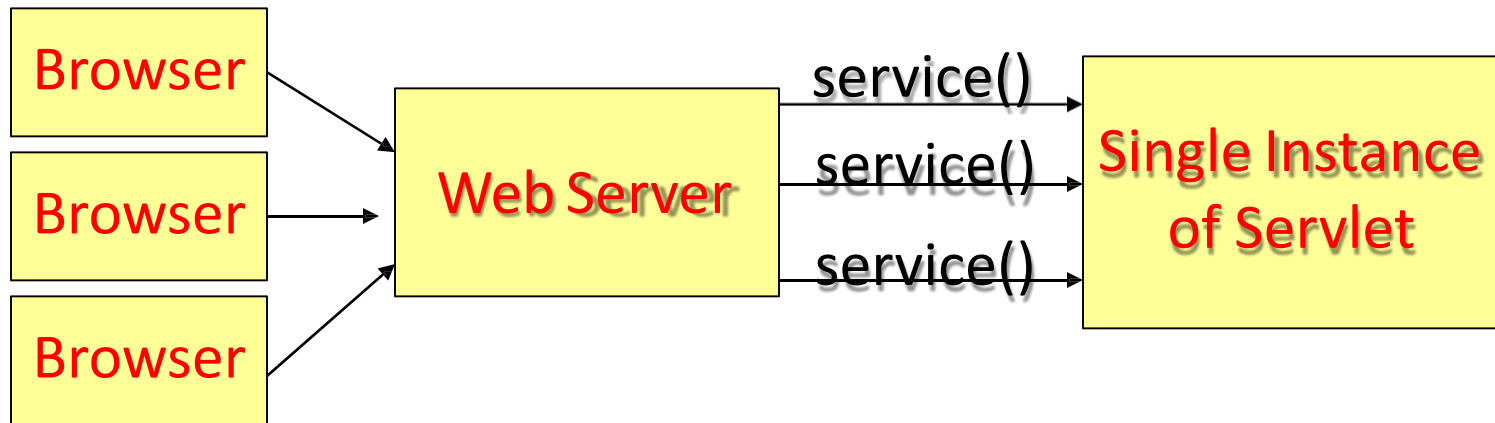
Let's Prove it…

- To prove that only one instance of a servlet is created, let's create a simple example.
- The Counter Servlet keeps track of the number of times it has been accessed.
- This example maintains a single instance variable, called count.
- Each time the servlet is called, the count variable is incremented.
- If the Server created a new instance of the Servlet for each request, count would always be 0!

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Counter extends HttpServlet {
        // Create an instance variable
        int count = 0;

        // Handle an HTTP GET Request
        public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws IOException,
ServletException  {
                response.setContentType("text/plain");
                PrintWriter out = response.getWriter();
                count++;
                out.println ("Since loading, this servlet has "
                        + "been accessed "+ count + " times.");
                out.close();
        }
}
```

Only one instance of the counter Servlet is created. Each browser request is therefore incrementing the same count variable.

The Service Method

?

- By default the service() method checks the
? HTTP Header.
- Based on the header, service calls either
? doPost() or doGet().
- doPost and doGet is where you put the
? majority of your code.
- If your servlets needs to handle both get and post identically, have your doPost() method call doGet() or vice versa.

Death of a Servlet

☐ Before a server shuts down, it will call the servlet's destroy() method.

☐ You can handle any servlet clean up here. For example:

☐ Updating log files.

☐ Closing database connections.

☐ Closing any socket connections.

```java
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Death extends HttpServlet
 {

  //  Handle an HTTP GET Request
  public void doGet(HttpServletRequest request,
HttpServletResponse  response)  throws IOException,
ServletException
   {
        response.setContentType("text/plain");
        PrintWriter out = response.getWriter();
        out.println ("I am alive!");
        out.close();
   }
 }
```

```java
//  This method is called when one stops
//  the Java Web Server
 public void destroy()
{

        try
         {

                FileWriter fileWriter = new FileWriter ("rip.txt");
                Date now = new Date();
                String rip = "I was destroyed at:   "+now.toString();
                fileWriter.write (rip);
                fileWriter.close();
        } catch (IOException e)
        {

                e.printStackTrace();

        }

 }
}
```

A Persistent Counter

- Now that we know all about the birth, life and death of a servlet, let's put this knowledge together to create a persistent counter.

- The Counter.java example we covered earlier has a big problem:

  - When you restart the web server, counting starts all over at 0.

  - It does not retain any persistent memory.

Persistent Counter

 To create a persistent record, we can store the count value within a "counter.txt" file.

   init(): Upon start-up, read in the current counter value from counter.txt.

   destroy(): Upon destruction, write out the new counter value to counter.txt

```java
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class CounterPersist extends HttpServlet {
  String fileName = "counter.txt";
  int count;

  public void init () {
        try {
          FileReader fileReader = new FileReader (fileName);
          BufferedReader bufferedReader = new BufferedReader (fileReader);
          String initial = bufferedReader.readLine();
          count = Integer.parseInt (initial);
        } catch (FileNotFoundException e) { count = 0; }
        catch (IOException e) { count = 0; }
        catch (NumberFormatException e) { count = 0; }
  }
}
```

At Start-up, load the counter from file.

In the event of any exception, initialize count to 0.

```
//  Handle an HTTP GET Request
public void doGet(HttpServletRequest request, HttpServletResponse resp
    throws IOException, ServletException  {
        response.setContentType("text/plain");
        PrintWriter out = response.getWriter();
        count++;
        out.println ("Since loading, this servlet has "
                +"been accessed "+ count + " times.");
        out.close();
 }
```

Each time the doGet() method is called, increment the count variable.

```java
// At Shutdown, store counter back to file
public void destroy() {
    try {
        FileWriter fileWriter = new FileWriter (fileName);
        String countStr = Integer.toString (count);
        fileWriter.write (countStr);
        fileWriter.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}
```

Can anybody foresee
any problems with
this code?

When destroy() is
called, store new
counter variable
back to counter.txt.

Steps for implementing servlet programs

1) Import the required packages
    1) Javax.servlet.http.*;
    2) Javax.servlet.*;
    3) Java.io.*;
    4) Any other packages if required
2) Define a class extends with either generic servlet or httpservlet class
3) Define the class with public access modifier hence save the file exactly same as class name
4) Override required lifecycle methods

Various phases of servlet during execution

1) Object instantiation phase
2) Object initialization
   1) Executing the init() method of servlet
3) Request processing phase
   1) Executing the service() method of servlet
4) Destruction phase
   1) Executing destroy() method of servlet
5) unavailability

Web.xml

- It is also known as web configuration file of deployment descriptor
- Web.xml is the fixed name to be given in the web application development
- It is always used to populate/ hide the technologies which are used in the web application development
- The web.xml file will be scanned by the server initially

contd..,

```
<web-app>
   <servlet>
        <servlet-name>logical name</servlet-name>
        <servlet-class>servlet class</servlet-class>
   </servlet>
   <servlet-mapping>
        <servlet-name>logical name</servlet-name>
        <url-pattern>/public url of web app</url-pattern>
   <servlet-mapping>
</web-app>
```

Deployment Structure

It is prescribed structure provided by sun microsystem in oracle to develop the web applications

→PR   JECT NAME
O
    →src

    → .java file

    → .html file

    → .js file

WEB-INF
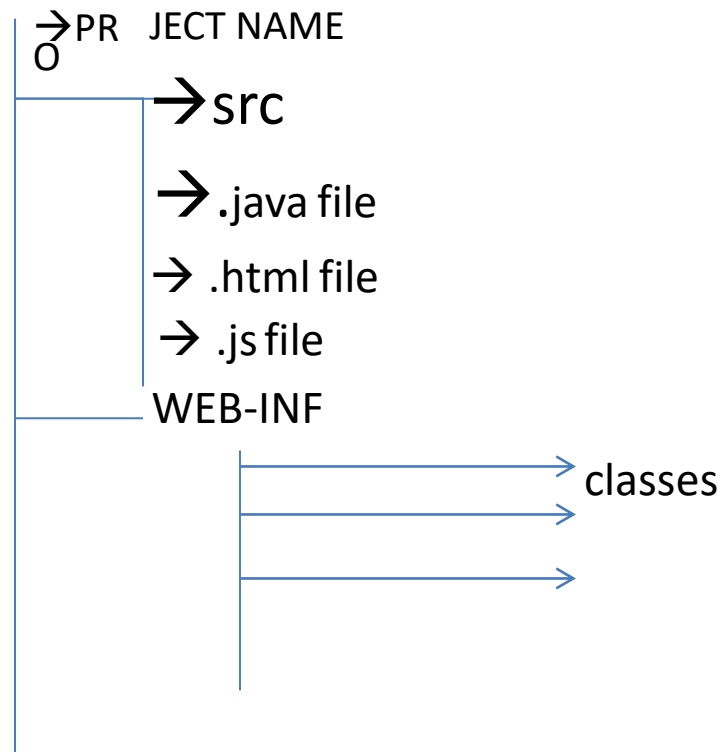
    classes

- Before compiling the servlet program we need to set the class path for servlet-api.jar

- Set classpath=c:\prgram files\apache software foundatin\tomcat7.0\lib\servlet-api.jar;

- After compiling the java program copy the .class file from src folder to classes folder.

- Define web.xml file

Deployment

- It is the process of copying the project from the current working directory to the context of server
- Copy the project from current directory and paste it into webapp
  - C:\program files\apache software foundation\tomcat 7.0\webapps
- Start the server
  - Start→programs→apache tomact 7.0→configured tomcat→start (ensure server status must be started)
- Open the browser type the following url
  - http://localhost:8080/projectname/servlet-name

UNIT-4

JSP-Java Server Page

Thread Synchronization

- After the Servlet is generated, one instance of it serves requests in different threads, just like any other Servlet

- In particular, the service method (_jspService) may be executed by several concurrent threads

- Thus, like Servlets, JSP programming requires concurrency management

Basic Elements in a JSP file

- HTML code: *<html-tag>content</html-tag>*

- JSP Comments: *<%-- comment --%>*

- Expressions: *<%= expression %>*

- Scriptlets: *<% code %>*

- Declarations: *<%! code %>*

- Directives: *<%@ directive attribute="value" %>*

- Actions: <jsp:forward.../>, <jsp:include.../>

- EL Expressions: ${*expression*}

Covered Later...

178

JSP Expressions

- A JSP expression is used to insert Java values directly into the output

- It has the form: *<%= expression %>* , where *expression* can be a Java object, a numerical expression, a method call that returns a value, etc...

- For example:

<%= new java.util.Date() %>

The **heading space** and the **following space** are not created in the result.

Use " " if you want a real space

<%= "Hello"+" World" %>

<%= (int)(100*Math.random()) %>

17-9

JSP Expressions

- Within the generated Java code
  - A JSP Expression is evaluated
  - The result is converted to a string
  - The string is inserted into the page
- This evaluation is performed at runtime (when the page is requested), and thus has full access to information about the request, the session, etc…

Expression Translation

```
<h1>A Random Number</h1>
<%= Math.random() %>
```

```java
public void _jspService(HttpServletRequest request,
        HttpServletResponse response)
        throws java.io.IOException, ServletException {

    ...
        response.setContentType("text/html");

    ...
        out.write("<h1>A Random Number</h1>");
        out.print( Math.random() );
    out.write("\r\n");

    ...
}
```

Default **content-type**

The generated **servlet** calls *out.write()* for **Strings**, and *out.print()* for **objects**

Predefined Variables (Implicit Objects)

- The following predefined variables can be used:
  - **request**: the HttpServletRequest
  - **response**: the HttpServletResponse
  - **session**: the HttpSession associated with the request
  - **out**: the PrintWriter (a buffered version of type JspWriter) used to fill the response content
  - **application**: The ServletContext
  - **config**: The ServletConfig
- These variables and more will be discussed in details

```html
<html>
  <head>
    <title>JSP Expressions</title>
  </head>
  <body>
    <h2>JSP Expressions</h2>
    <ul>
      <li>Current time: <%= new java.util.Date() %></li>
      <li>Your hostname:<%= request.getRemoteHost() %></li>
      <li>Your session ID: <%= session.getId() %></li>
      <li>The <code>testParam</code> form parameter:
        <%= request.getParameter("testParam") %></li>
    </ul>
  </body>
</html>
```
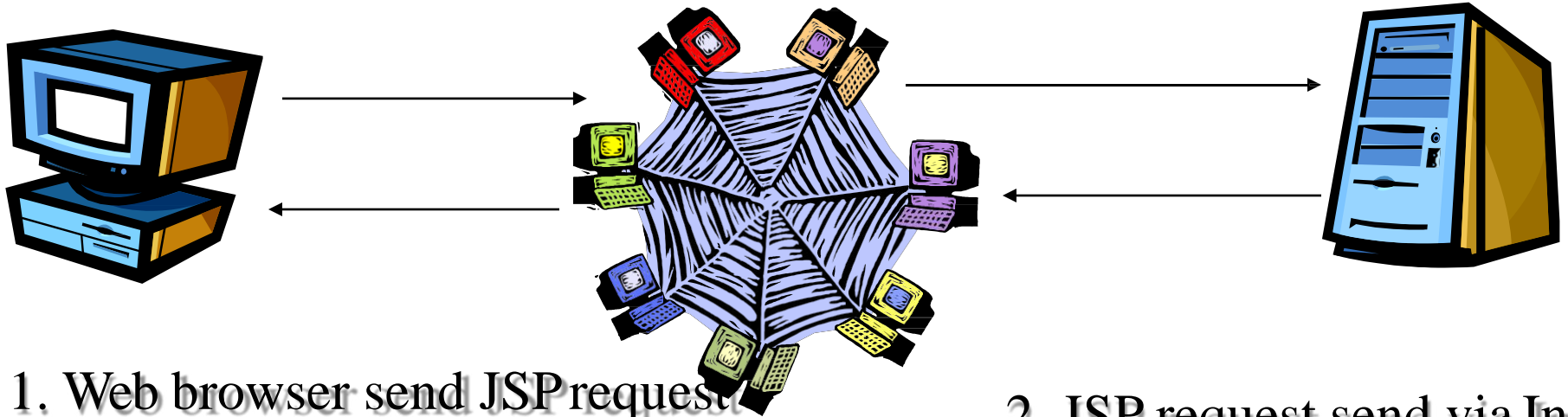
Computer-code style

▶

JSP Scriplets

- JSP scriptlets let you insert arbitrary code into the Servlet service method (_jspService)

- Scriptlets have the form: <% *Java Code* %>

- The code is inserted verbatim into the service method, according to the location of the scriptlet

- Scriptlets have access to the same automatically defined variables as expressions

Scriptlet Translation

```
<%= foo() %>
<% bar(); %>
```

```java
public void _jspService(HttpServletRequest request,
                        HttpServletResponse response)
throws ServletException, IOException {
        ...
        response.setContentType("text/html");
        ...
        out.print(foo());
        bar();
        ...
}
```

An Interesting Example

Scriptlets don't have to be complete code blocks:

```
<% if (Math.random() < 0.5) { %>
You <b>won</b> the game!
<% } else { %>
You <b>lost</b> the game!
<% } %>
```

⬇

```
if (Math.random() < 0.5) {
    out.write("You <b>won</b> the game!");
} else {
    out.write("You <b>lost</b> the game!");
}
```

JSP Declarations

- A JSP declaration lets you define methods or members that get inserted into the Servlet class (**outside** of all methods)
- It has the following form:

  - <%! Java Code %>

- For example:

  - <%! private int someField = 5; %>
  - <%! private void someMethod(...) {...} %>

- JSPs are intended to contain a minimal amount of code so it is usually of better design to define methods in a separate Java class…

Declaration Example

- Print the number of times the current page has been requested since the Servlet initialization:

```
<%! private int accessCount = 0; %>
<%! private synchronized int incAccess() {
            return ++accessCount;
    } %>
<h1>Accesses to page since Servlet init:
<%= incAccess() %> </h1>
```

```
public class serviceCount_jsp extends... implements...
         throws... {
         private int accessCount = 0;

         private synchronized int incAccess() {
           return ++accessCount;
         }

         public void _jspService(HttpServletRequest request,
               HttpServletResponse response)
                throws ServletException, IOException {
      ...
   ...
out.write("<h1>Accesses to page since Servlet init: ");
out.print(incAccess());
... } ... }
```

Java permits member Servlet Generate the JSP

initialization on declaration, even if the location is outside any method's scope

JSP Directives

- A JSP directive affects the structure of the Servlet class that is generated from the JSP page

- It usually has the following form:
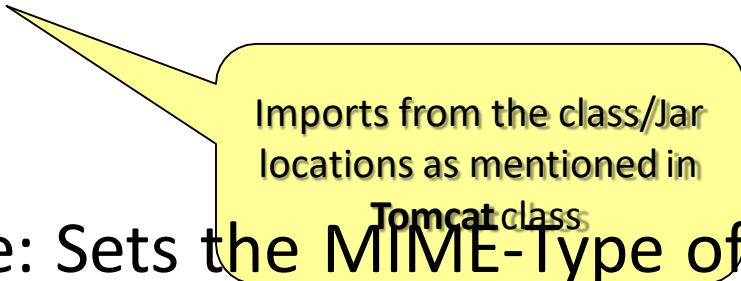
  <%@ directive attribute1="value1" **...** *attributeN="valueN" %>*

- Three important directives: page, include and taglib

- include and taglib will be discussed later

page-Directive Attributes

- import attribute: A comma separated list of classes/packages to import

<%@ page import="java.util.*, java.io.*" %>

Imports from the class/Jar locations as mentioned in **Tomcat** class

- contentType attribute: Sets the MIME-Type of the resulting document (default is text/html as already mentioned)

<%@ page contentType="text/plain" %>

- *What is the difference between setting the page contentType attribute, and writing*

  <%response.setContentType("...");%> *?*

  – In the latter case, the new servlet will call *response.setContentType()* twice

  – The first, impicit (from the JSP point of view), call will be with the default content type.

  – The second, explicit, call might even come after the buffer was flushed or after the writer was obtained…

Check: double-contenttype.jsp code using the explicit call, generated java code (servlet)

page-Directive Attributes (cont)

- session="**true**|false" - use a session?   .

- buffer="*sizekb*|none|**8kb**"

  - Specifies the content-buffer (out) size in kilobytes

- autoFlush="**true**|false"

  - Specifies whether the buffer should be flushed when it fills, or throw an exception otherwise

- isELIgnored ="**true**|false"

  - Specifies whether *JSP expression language* is used
  - EL is discussed later

Outline

- Introducing JavaServer Pages™ (JSP™)
- JSP scripting elements
  - Expressions, Scriptlets and declarations
- The JSP page Directive:
  - Structuring Generated Servlets™
- Including Files in JSP Documents
- Using JavaBeans™ components with JSP
- Creating custom JSP tag libraries
- Integrating servlets and JSP with the MVC architecture

The JSP Framework

- Idea:
  - Use regular HTML for most of page
  - Mark servlet code with special tags
  - Entire JSP page gets translated into a servlet (once), and servlet is what actually gets invoked (for each request)
- Example:
  - JSP
    - Thanks for ordering
      <I><%= request.getParameter("title") %></I>
  - URL
    - http://host/OrderConfirmation.jsp?title=Core+Web+Programming
  - Result
    - Thanks for ordering Core Web Programming

Setting Up Your Environment

✖ Set your CLASSPATH.

✖ Compile your code.

✖ Use packages to avoid name conflicts.

✖ Put JSP page in special directory.

  ✖ tomcat_install_dir/webapps/ROOT

✖ Use special URL to invoke JSP page.

- Caveats
  - Previous rules about CLASSPATH, install dirs, etc., still apply to regular classes used by JSP

```html
<HTML><HEAD>
<TITLE>JSP Expressions</TITLE>
<META NAME="author" CONTENT="Marty Hall">
<META NAME="keywords"
    CONTENT="JSP,expressions,JavaServer,Pages,servlets
    ">
<META NAME="description"
    CONTENT="A quick example of JSP expressions.">
<LINK REL=STYLESHEET HREF="JSP-Styles.css"
    TYPE="text/css">
</HEAD>
<BODY>
<H2>JSP Expressions</H2>
<UL>
 <LI>Current time: <%= new java.util.Date() %>
 <LI>Your hostname: <%= request.getRemoteHost() %>
 <LI>Your session ID: <%= session.getId() %>
 <LI>The <CODE>testParam</CODE> form parameter:
    <%= request.getParameter("testParam") %>
</UL></BODY></HTML>
```

Example Result

- With default setup, if location was
  - C:\<tomcatHome>\webapps\ROOT\Expressions.jsp

- URL would be
  - ht

JSP Expressions - Netscape

File　Edit　View　Go　Communicator　Help

Bookmarks　Location: http://webdev.apl.jhu.edu/~hall/JSP/Expressions.jsp?testParam=some+data

JSP Expressions

- Current time: Mon Jan 17 10:40:10 EST 2000
- Your hostname: pm4-s40.dial-up.abs.net
- Your session ID: YCKX3NIAAAA0XAG2MVSQAAA
- The testParam form parameter: some data

Document: Done
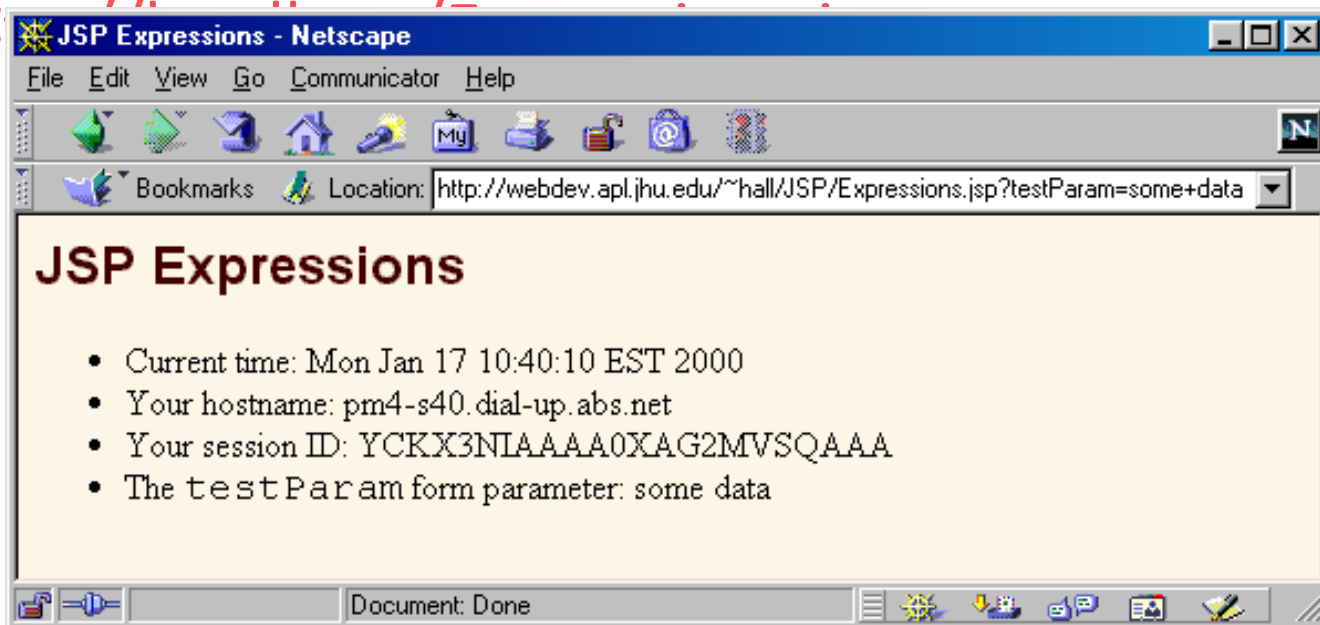
How JSP works?

1. Web browser send JSP request

8. HTML send back to the browser

2. JSP request send via Internet to the web server

3. The web server send the JSP file (template pages) to JSP servlet engine

4. Parse JSP file

5. Generate servlet source code

6. Compile servlet to class

JSP page translation and processing phases

JSP Life-cycle

Request for
JSP page

.jsp file newer than
Previously compiled
Servlet?

No

Yes

Translate and
(re-compile)
Servlet code

Execute the
Compiled Servlet

Response
an HTML
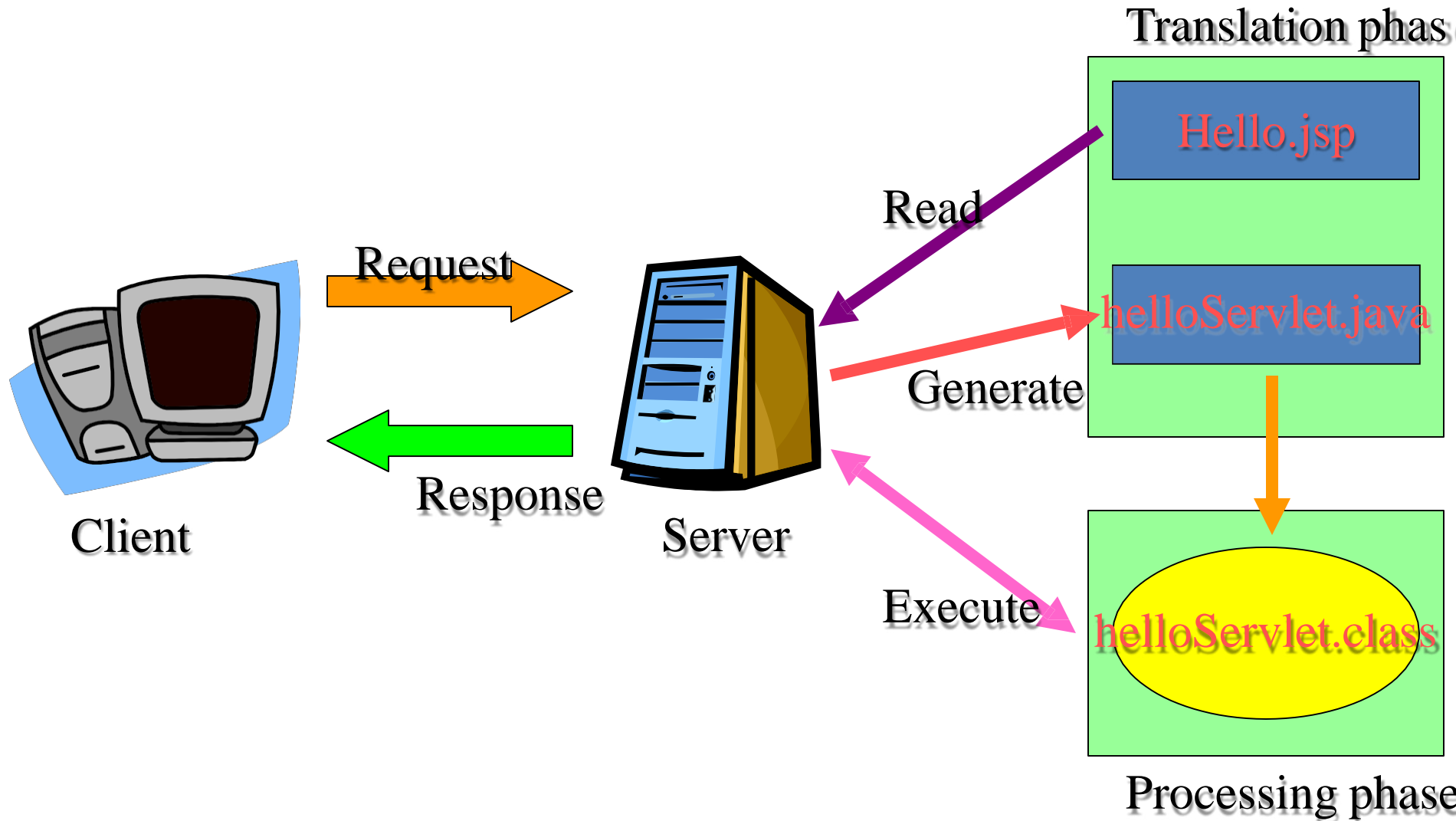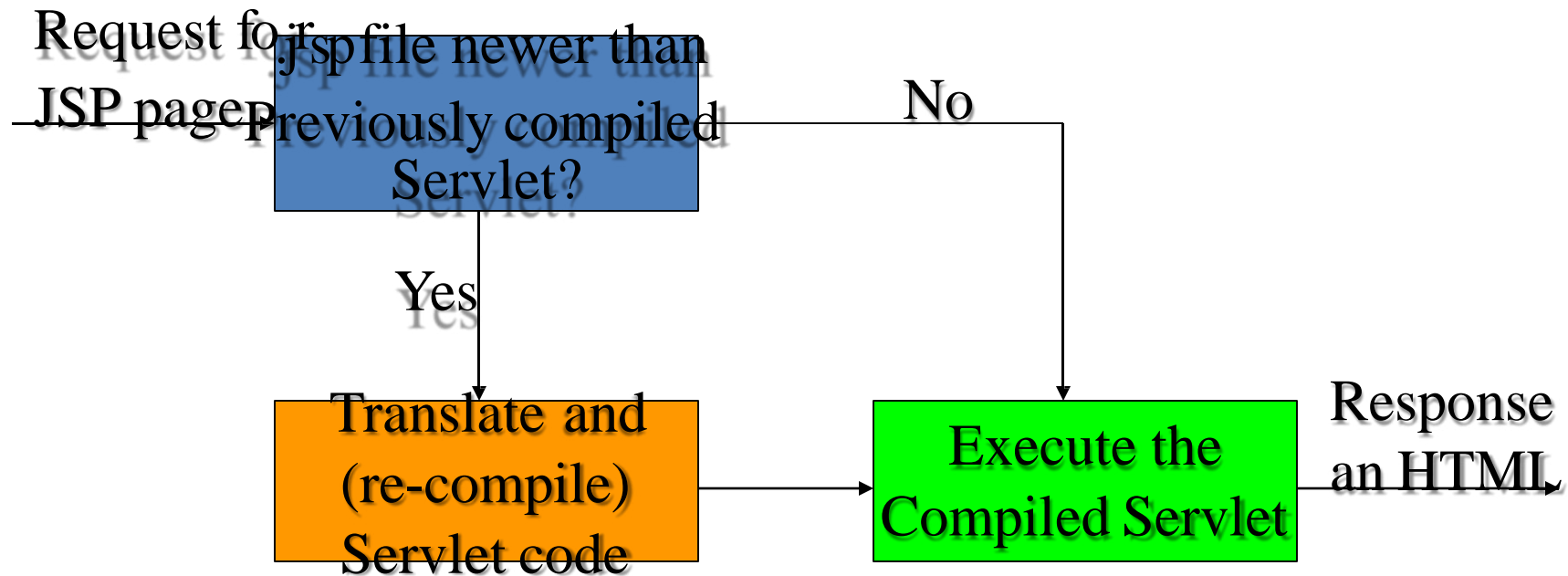
Template Pages

```
<html>
<title>
A simple example
</title>


<body color="#FFFFFF">
The time now is
<%= new java.util.Date() %>
</body>
</html>
```

translation →

```
<html>
<title>
A simple example
</title>


<body color="#FFFFFF">
The time now is
Tue Nov 5 16:15:11 PST 2002
</body>
</html>
```

Dividing Pure Servlets

```
Public class MySelect {

  public void doGet(…){

    if (isValid(..){

      saveRecord();

      out.println("<html>");

      ….

    }

  }

  private void isValid(…){…}

  private void saveRecord(…) {…}

}
```

Process request → **controller** Servlet

Presentation → **view** JSP

Business logic → **model** JavaBeans

Model-View-Controller (MVC)

Most Common Misunderstanding:
Forgetting JSP is Server-Side Technology

- **Very common question**
  - I can't do such and such with HTML.
    Will JSP let me do it?
- Similar questions
  - How do I put an applet in a JSP page?
    Answer: send an <APPLET…> tag to the client
  - How do I put an image in a JSP page?
    Answer: send an <IMG …> tag to the client
  - How do I use JavaScript/Acrobat/Shockwave/Etc?
    Answer: send the appropriate HTML tags

2nd Most Common Misunderstanding: Translation/Request Time Confusion

- **What happens at page translation time?**
  - JSP constructs get translated into servlet code
- **What happens at request time?**
  - Servlet code gets executed. No interpretation of JSP occurs at request time. The original JSP page is ignored at request time; only the servlet that resulted from it is used
- **When does page translation occur?**
  - Typically, the first time JSP page is accessed after it is modified. This should never happen to real user (developers should test all JSP pages they install).
  - Page translation does not occur for each request

JSP/Servlets in the Real World

- Delta Airlines: entire Web site, including real-time schedule info
- First USA Bank: largest credit card issuer in the world; most on-line banking customers

JSP/Servlets in the Real World

- Excite: one of the top five Internet portals; one of the ten busiest sites on the Web

Hidden / HTML Comment

- An HTML comment is sent to the client's browser, but is not displayed. The information can be reviewed from the source code.
  - <!-- comment [<%= expression%>] -->
- A hidden comment is discarded before any processing of the JSP page and is not sent to the web browser.
  - <%-- comment-->

JSP Components

- There are three main types of JSP constructs that you embed in a page.
  - Scripting elements
    - You can specify Java code
    - Expressions, Scriptlets, Declarations
  - Directives
    - Let you control the overall structure of the servlet
    - Page, include, Tag library
  - Actions
    - Enable the use of server side Javabeans
    - Transfer control between pages

Uses of JSP Constructs: Use of Scripting elements

Simple Application
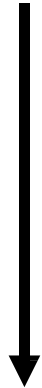
Complex Application

- Scripting elements calling servlet code directly
- Scripting elements calling servlet code indirectly (by means of utility classes)
- Beans
- Custom tags
- Servlet/JSP combo (MVC architecture)

Types of Scripting Elements

- You can insert code into the servlet that will be generated from the JSP page.
- Expressions: <%= expression %>
  - Evaluated and inserted into the servlet's output. i.e., results in something like out.println(expression)
- Scriptlets: <% code %>
  - Inserted verbatim into the servlet's _jspService method (called by service)
- Declarations: <%! code %>
  - Inserted verbatim into the body of the servlet class, outside of any existing methods

JSP Expressions

- Format
  - <%= Java Expression %>
- Result
  - Expression evaluated, converted to String, and placed into HTML page at the place it occurred in JSP page
  - That is, expression placed in _jspService inside out.print
- Examples
  - Current time: <%= new java.util.Date() %>
  - Your hostname: <%= request.getRemoteHost() %>
- XML-compatible syntax
  - <jsp:expression>Java Expression</jsp:expression>
  - XML version not supported by Tomcat 3. Until JSP 1.2, servers are not required to support it.

JSP/Servlet Correspondence

- Original JSP

  ```
  <H1>A Random Number</H1>
  <%= Math.random() %>
  ```

- Possible resulting servlet code

```java
public void _jspService(HttpServletRequest request,
                HttpServletResponse response)
   throws ServletException, IOException {
 request.setContentType("text/html");
 HttpSession session = request.getSession(true);
 JspWriter out = response.getWriter();
 out.println("<H1>A Random Number</H1>");
 out.println(Math.random());
 ...
 }
```

Example Using JSP Expressions

```
<BODY>
<H2>JSP Expressions</H2>
<UL>
  <LI>Current time: <%= new java.util.Date() %>
  <LI>Your hostname: <%= request.getRemoteHost() %>
  <LI>Your session ID: <%= session.getId() %>
  <LI>The <CODE>testParam</CODE> form parameter:
     <%= request.getParameter("testParam") %>
</UL>
</BODY>
```
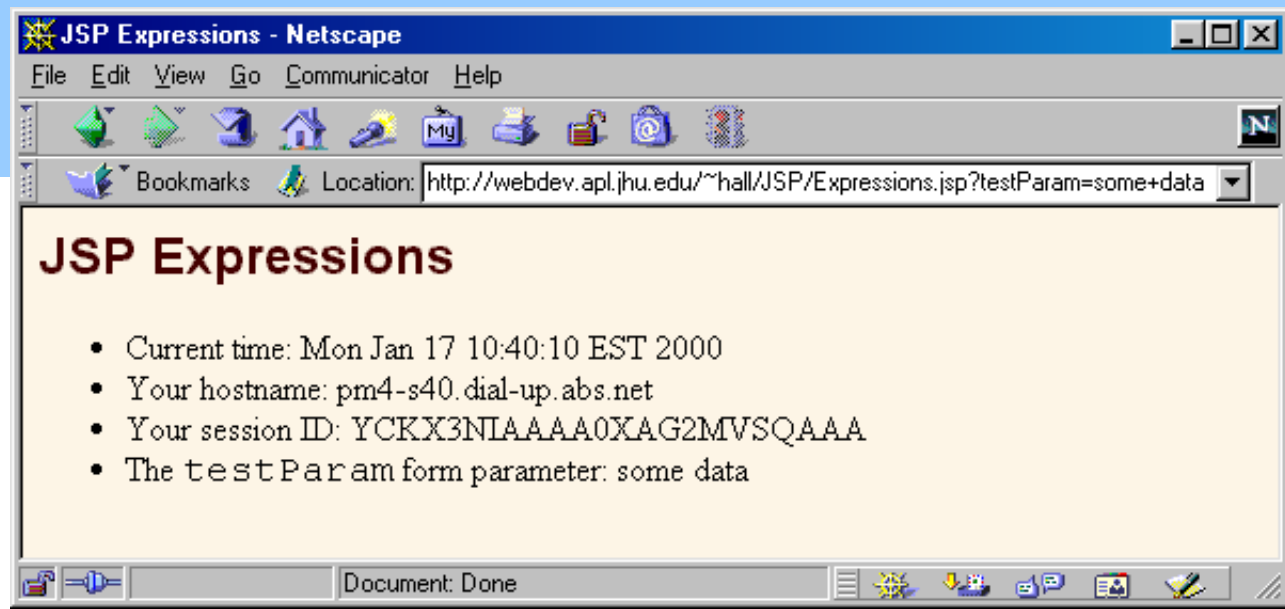
Predefined Variables (Implicit Objects)

- ## They are created automatically when a web server processes a JSP page.

- request: The HttpServletRequest (1st arg to doGet)

- response: The HttpServletResponse (2nd arg to doGet)

- session
  - The HttpSession associated with the request (unless disabled with the session attribute of the page directive)

- out
  - The stream (of type JspWriter) used to send output to the client

- application
  - The ServletContext (for sharing data) as obtained via getServletConfig().getContext().

- page, pageContext, config, exception

Implicit objects – Class files

- application: javax.servlet.ServletContext
- config: javax.servlet.ServletConfig
- exception: java.lang.Throwable
- out: javax.servlet.jsp.JspWriter
- page: java.lang.Object
- pageContext: javax.servlet.jsp.PageContext
- request: javax.servlet.ServletRequest
- response: javax.servlet.ServletResponse
- session: javax.servlet.http.HttpSession

Access Client Information

- The getRemoteHost method of the request object allows a JSP to retrieve the name of a client computer.

```html
<html><head>
<title>Your Information</title>
</head><body>
Your computer's IP address is
<b><%= request.getRemoteAddr() %></b>
<br>Your computer's name is
<b><%= request.getRemoteHost() %></b>
<br>Your computer is accessing port number
<b><%= request.getServerPort() %></b>
</body></html>
```

Your Information - Netscape

File   Edit   View   Go   Bookmarks   Tools   Window   Help

Search

Your computer's IP address is **127.0.0.1**
Your computer's name is **127.0.0.1**
Your computer is accessing port number **80**

Document: Done (3.285 secs)
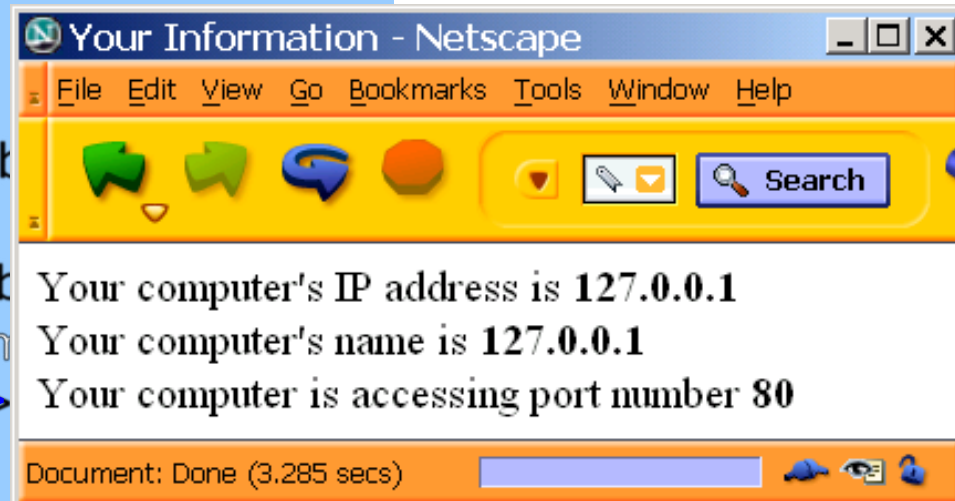
Work with the Buffer

☑ When the page is being processed, the datais stored in the buffer instead of being directly sent to the client browser.

```
<html>
This is a test of the buffer<br/>
<%
out.flush();
for (int x=0; x < 100000000; x++);
out.print("This test is generated abo
    later.");
out.flush();
%>
</html>
```

Netscape

File   Edit   View   Go   Bookmarks   Tools   Window   Help

Search

This is a test of the buffer
This test is generated about 5 seconds later.

Document: Done (8.272 secs)
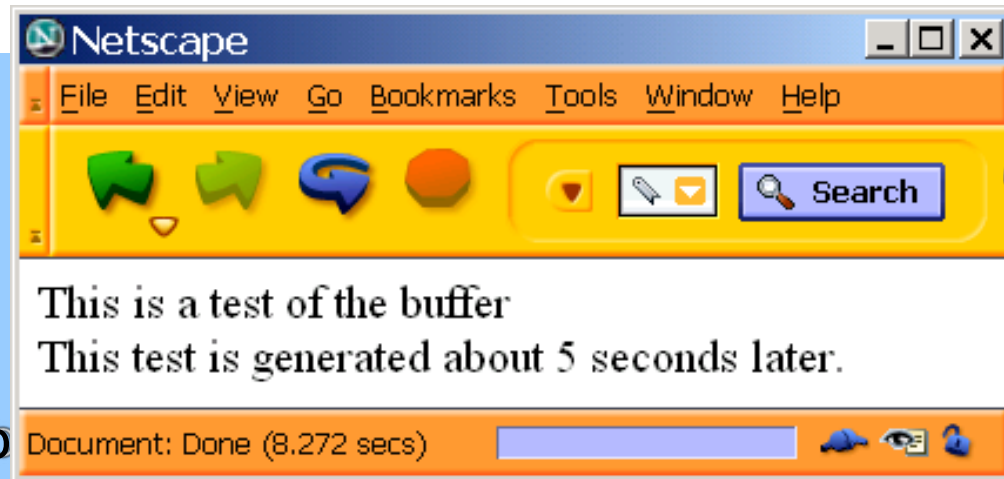
Working with Session object

- The session object has many useful methods that can alter or obtain information about the current session.
  - setMaxInactiveInterval(second)

```
<html><head>
<title>Session Values</title>
</head><body>
<%
session.setMaxInactiveInterval(10);
String name = (String)
    session.getAttribute("username");
out.print("Welcome to my site " + name + "<br>");
%>
</body></html>
```

Session Values ...
File Edit View Go Bookmarks
Directo...   Sessio...

Welcome to my site null

Session Values ...
File Edit View Go Bookmarks

Welcome to my site Ray
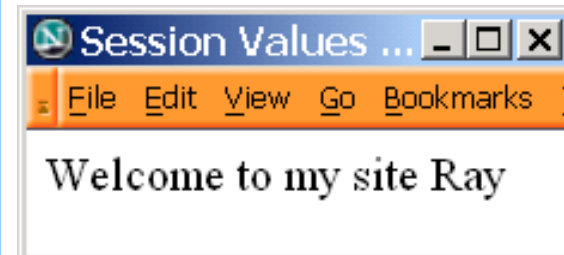
JSP Scriptlets

- Format: <% Java Code %>
- Result
  - Code is inserted verbatim into servlet's _jspService
- Example
  - <%
    String queryData = request.getQueryString();
    out.println("Attached GET data: " + queryData);
    %>
  - <% response.setContentType("text/plain"); %>
- XML-compatible syntax
  - <jsp:scriptlet>Java Code</jsp:scriptlet>

JSP/Servlet Correspondence

- Original JSP

```
<%= foo() %>
<% bar(); %>
```

- Possible resulting servlet code

```
public void _jspService(HttpServletRequest request,
                        HttpServletResponse response)
    throws ServletException, IOException {
  request.setContentType("text/html");
  HttpSession session = request.getSession(true);
  JspWriter out = response.getWriter();
  out.println(foo());
  bar();
  ...
}
```

JSP Scriptlets Example

```
<%
for (int i=100; i>=0; i--)
{
%>
  <%= i %> bottles of beer on the
    wall.<br>
<%
}
%>
```


Netscape — http://localhost/myJSP/Chapter01

100 bottles of beer on the wall.
99 bottles of beer on the wall.
98 bottles of beer on the wall.
97 bottles of beer on the wall.
96 bottles of beer on the wall.
95 bottles of beer on the wall.
94 bottles of beer on the wall.
93 bottles of beer on the wall.
92 bottles of beer on the wall.
91 bottles of beer on the wall.

Document: Done (0.13 secs)
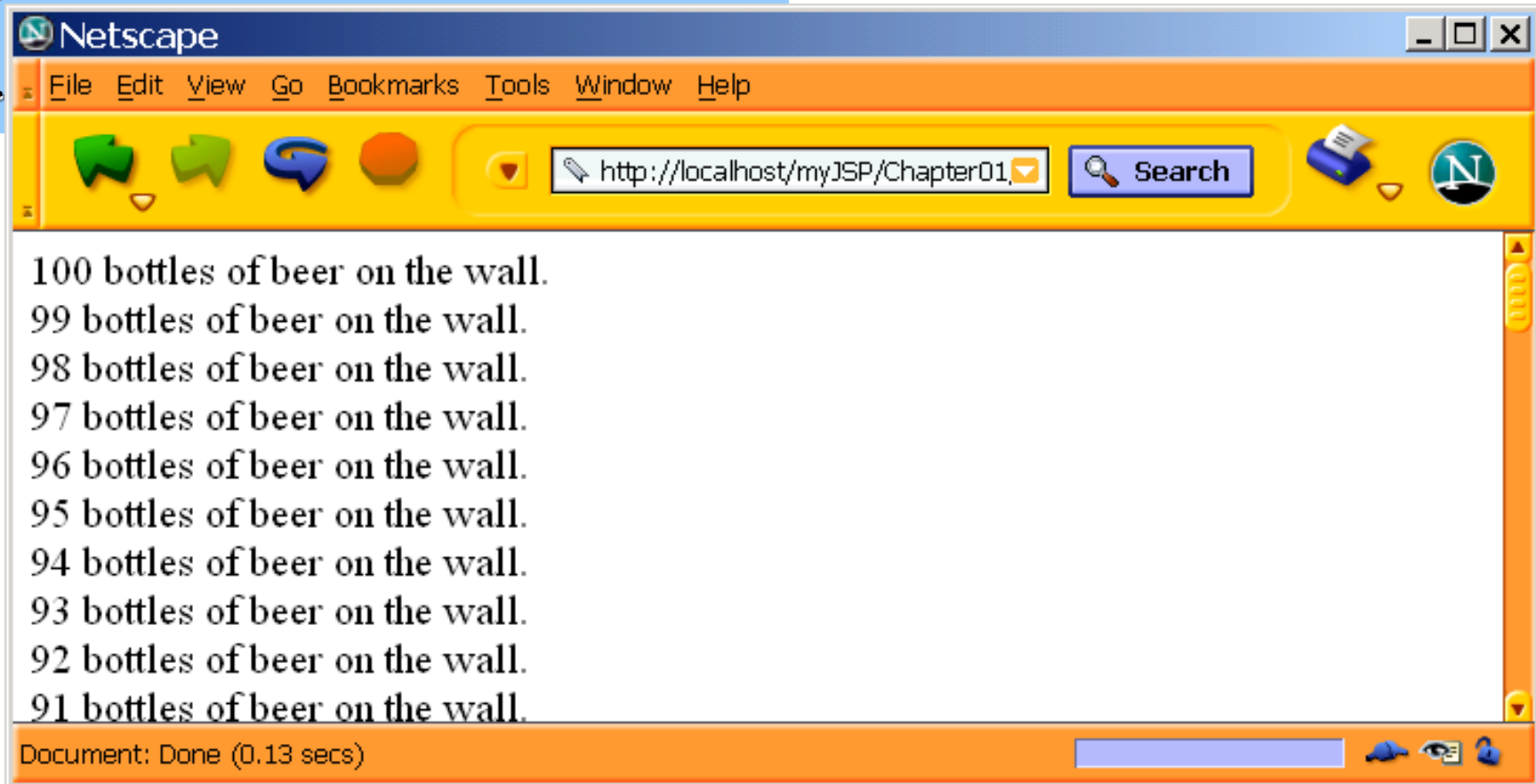
Example Using JSP Scriptlets

```
<HTML>
<HEAD>
  <TITLE>Color Testing</TITLE>
</HEAD>
<%
String bgColor =
    request.getParameter("bgColor");
boolean hasExplicitColor;
if (bgColor != null) {
hasExplicitColor = true;
} else {
  hasExplicitColor = false;
  bgColor = "WHITE";
}
%>
<BODY BGCOLOR="<%= bgColor %>">
```

Color Testing - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://localhost/JSP/BGColor.jsp

Color Testing

Color Testing - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://localhost/JSP/BGColor.jsp?bgColor=papayawhip

Color Testing

You supplied an explicit background color of papayawhip.

Color Testing - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://localhost/JSP/BGColor.jsp?bgColor=C0C0C0

Color Testing

You supplied an explicit background color of C0C0C0.

Done   Local intranet

JSP Declarations

- Format
  - <%! Java Code %>
- Result
  - Code is inserted verbatim into servlet's class definition, outside of any existing methods
- Examples
  - <%! private int someField = 5; %>
  - <%! private void someMethod(...) {...} %>
- XML-compatible syntax
  - <jsp:declaration>Java Code</jsp:declaration>

Scriptlets vs. Declarations

<%! int count=100; %>
<%= ++count %>

```
public final class
    _scopeExpermnt1_xjsp
{
  int count=100;

  public void _jspService
    (HttpServletRequest request,
     HttpServletResponse response)
    throws java.io.IOException
  {
    JspWriter out =
   pageContext.getOut();

   out.print( "\r\n" );

   out.print( String.valueOf(  ++
   count  ) );
    out.print( "\r\n" );
  }
}
```

<% int count=100; %>
<%= ++count %>

```
public final class
    _scopeExpermnt2_xjsp
{
  public void _jspService
    (HttpServletRequest request,
     HttpServletResponse response)
     throws java.io.IOException
  {
     JspWriter out =
   pageContext.getOut();

     int count=100;

     out.print( "\r\n" );
     out.print( String.valueOf(
   ++count  ) );
     out.print( "\r\n" );

  }
}
```

- Original JSP

```
<H1>Some Heading</H1>
<%!
  private String randomHeading() {
    return("<H2>" + Math.random() + "</H2>");
  }
%>
<%= randomHeading() %>
```

- Possible resulting servlet code

```
public class xxxx implements HttpJspPage {
 private String randomHeading() {
 return("<H2>" + Math.random() + "</H2>");
 }
 public void _jspService(HttpServletRequest request,
       HttpServletResponse response) throws ServletException, IOException {
  request.setContentType("text/html");
  HttpSession session = request.getSession(true);
  JspWriter out = response.getWriter();
  out.println("<H1>Some Heading</H1>");
  out.println(randomHeading());
  ...
 }
}
```

Example Using JSP Declarations
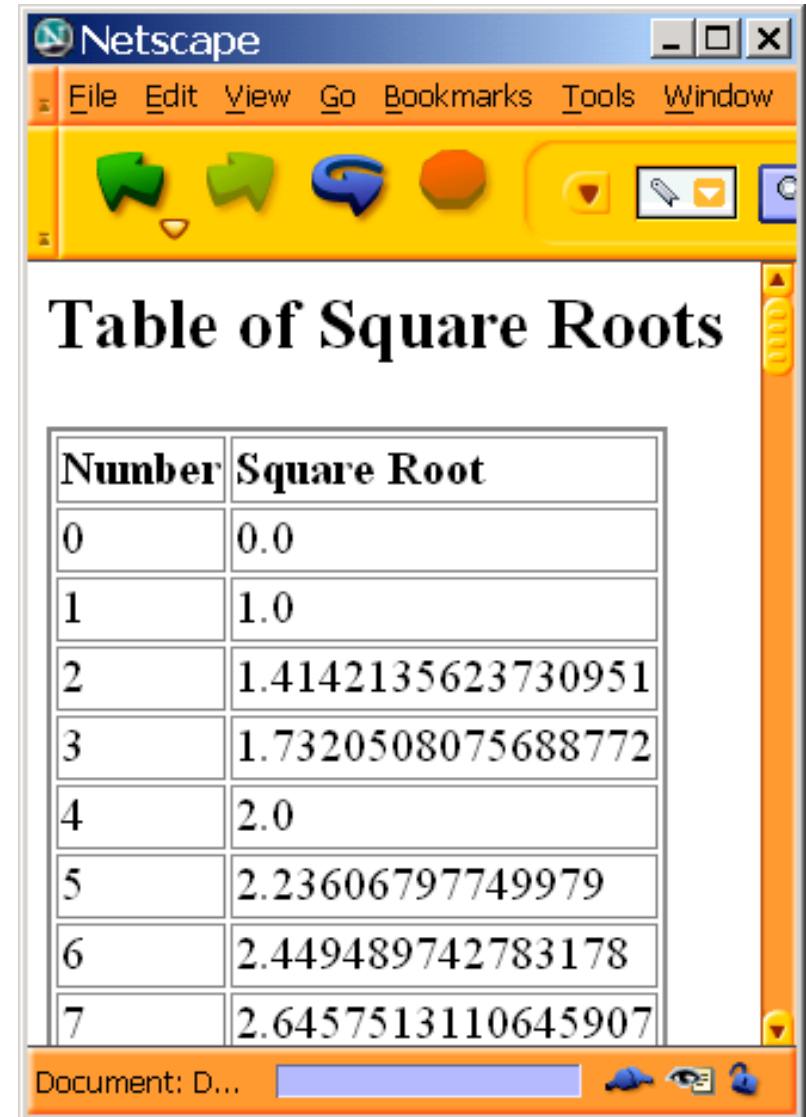
```
...
<body>
<h1>JSP Declarations</h1>
<%! private int accessCount = 0; %>
<h2>Accesses to page since server reboot:
<%= ++accessCount %></h2>
</body></html>
```

- After 15 total visits by an arbitrary number of different clients
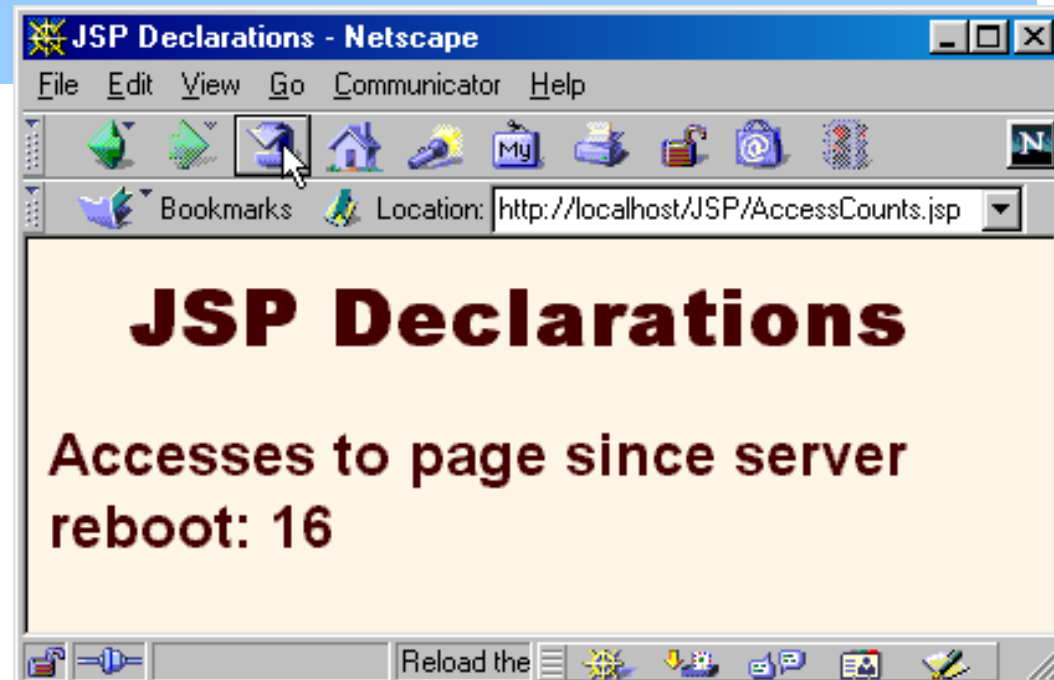
JSP Tags + HTML Tags

```
<h2>Table of Square Roots</h2>
<table border=2>
 <tr>
  <td><b>Number</b></td>
  <td><b>Square Root</b></td>
 </tr>
 <%
 for (int n=0; n<=100; n++)
 {
 %>
  <tr>
   <td><%=n%></td>
   <td><%=Math.sqrt(n)%></td>
  </tr>
 <%
 }
 %>
</table>
```

Netscape   _ □ ✕

File Edit View Go Bookmarks Tools Window

Table of Square Roots

Number	Square Root
0	0.0
1	1.0
2	1.4142135623730951
3	1.7320508075688772
4	2.0
5	2.23606797749979
6	2.449489742783178
7	2.6457513110645907

Document: D...

Purpose of the page Directive

- Give high-level information about the servlet that will result from the JSP page
- Can control
  - Which classes are imported
  - What class the servlet extends
  - What MIME type is generated
  - How multithreading is handled
  - If the servlet participates in sessions
  - The size and behavior of the output buffer
  - What page handles unexpected errors

The import Attribute

- **Format**
  - <%@ page import="package.class" %>
  - <%@ page import="package.class1,...,package.classN" %>

- **Purpose**
  - Generate import statements at top of servlet

- **Notes**
  - Although JSP pages can be almost anywhere on server, classes used by JSP pages must be in normal servlet dirs
  - For Tomcat, this is install_dir/webapps/ROOT/WEB-INF/classes or …/ROOT/WEB-INF/classes/directoryMatchingPackage

```
...
<BODY><H2>The import Attribute</H2>
<%-- JSP page directive --%>
<%@ page import="java.util.*,cwp.*" %>

<%-- JSP Declaration --%>
<%!
private String randomID() {
  int num = (int)(Math.random()*10000000.0);
  return("id" + num);
}
private final String NO_VALUE = "<I>No Value</I>";
%>
<%
Cookie[] cookies = request.getCookies();
String oldID = ServletUtilities.getCookieValue(cookies, "userID",
    NO_VALUE);
String newID;
if (oldID.equals(NO_VALUE)) { newID = randomID();
} else { newID = oldID; }
LongLivedCookie cookie = new LongLivedCookie("userID", newID);
response.addCookie(cookie);
%>
<%-- JSP Expressions --%>
This page was accessed at <%= new Date() %> with a userID
cookie of <%= oldID %>.
 </BODY></HTML>
```

Example of import Attribute

- First access
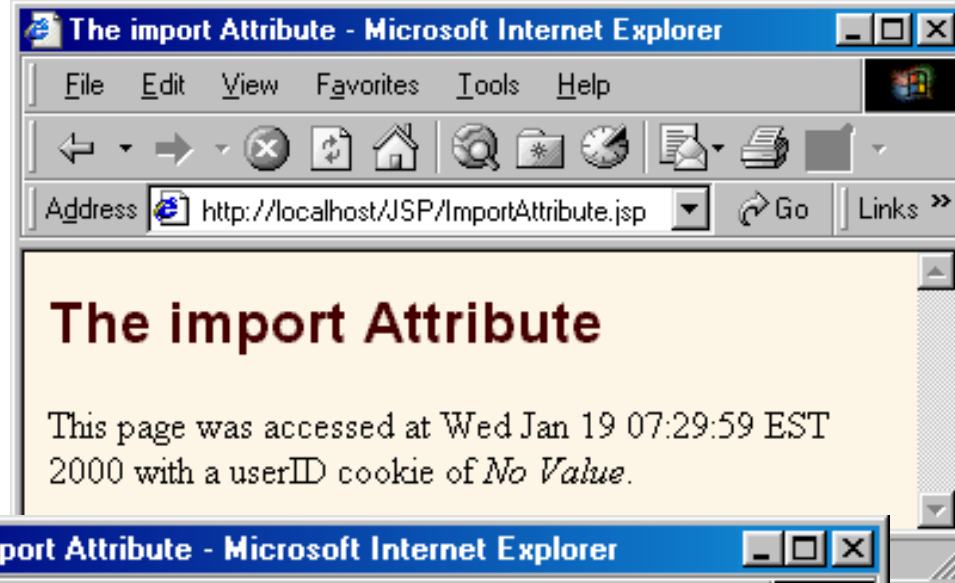
- Subsequent accesses
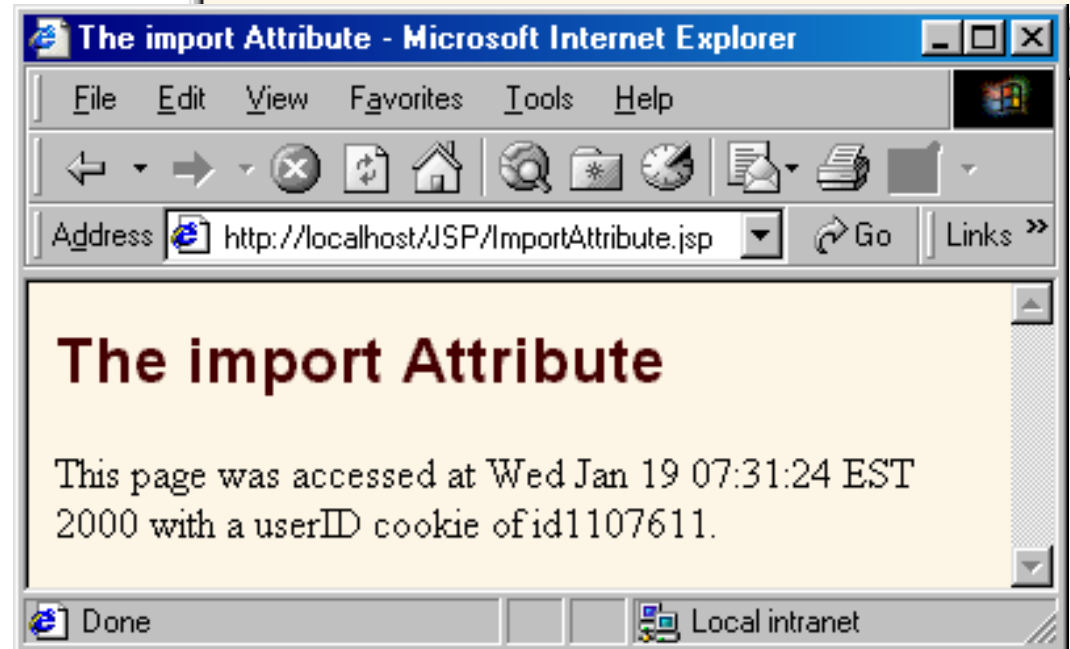
The contentType Attribute

- Format

  - <%@ page contentType="MIME-Type" %>

  - <%@ page contentType="MIME-Type; charset=Character-Set"%>

- Purpose

  - Specify the MIME type of the page generated by the servlet that results from the JSP page

First	Last	Email Address
Marty	Hall	hall@corewebprogramming.com
Larry	Brown	brown@corewebprogramming.com
Bill	Gates	gates@sun.com
Larry	Ellison	ellison@microsoft.com

```
<%@ page contentType="application/vnd.ms-excel" %>
<%-- There are tabs, not spaces, between columns. --%>
```

Another Example

Form Processing

```jsp
<%-- processOrder.jsp --%>
<%@ page errorPage="orderError.jsp"
       import="java.text.NumberFormat" %>
<h3>Your order:</h3>
<%
 String numTees = request.getParameter("t-shirts");
 String numHats = request.getParameter("hats");
 NumberFormat currency =
     NumberFormat.getCurrencyInstance();
%>
Number of tees:
 <%= numTees %><br>
Your price:
 <%= currency.format(Integer.parseInt(numTees)*15.00)
     %><p>
Number of hats:
 <%= numHats %><br>
Your price:
 <%= currency.format(Integer.parseInt(numHats)*10.00)
     %><p>
<!-- orderForm.htm -->
<h1>Order Form</h1>
What would you like to purchase?<p>
<form name=orders action=processOrder.jsp
<table border=0>
 <tr><th>Item</th>
   <th>Quantity</th>
   <th>Unit Price</th>
 <tr><tr>
...
```

Order Form

What would you like to purchase?

Item	Quantity	Unit Price
T-Shirts	10	@ $15.00 ea.
Hats	5	@ $10.00 ea.

Place order   Clear all fields

Your order:

Number of tees: 10
Your price: NT$150.00

Number of hats: 5
Your price: NT$50.00

Please enter a number in each quantity field.
Return to the main order form.

Other Attributes of the page Directive

- **session**
  - Lets you choose not to participate in sessions
- **buffer**
  - Changes min size of buffer used by JspWriter
- **autoflush**
  - Requires developer to explicitly flush buffer
- **extends**
  - Changes parent class of generated servlet
- **errorPage**
  - Designates a page to handle unplanned errors
- **isErrorPage, isThreadSafe, language, …**

JSP Actions

- There are *seven* standard JSP actions.
  - Include, param, forward, plugin, …
  - Include action is similar to include directive.
  - You can add additional parameters to the existing request by using the param action.
  - The plugin action inserts object and embed tags (such as an applet) into the response to the client.
  - In the coming slides, we will talk about "include" and "plugin" actions.

Including Pages at Request Time

- **Format**
  - <jsp:include page="Relative URL" flush="true" **/>**
- **Purpose**
  - To reuse JSP, HTML, or plain text content
  - JSP content cannot affect main page: only output of included JSP page is used
  - To permit updates to the included content without changing the main JSP page(s)

Including Pages: Example Code

```
...
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
     What's New at JspNews.com</TABLE>
<P>
Here is a summary of our three most recent news stories:
<OL>
  <LI><jsp:include page="news/Item1.html" flush="true" />
  <LI><jsp:include page="news/Item2.html" flush="true" />
  <LI><jsp:include page="news/Item3.html" flush="true" />
</OL>
</BODY></HTML>
```
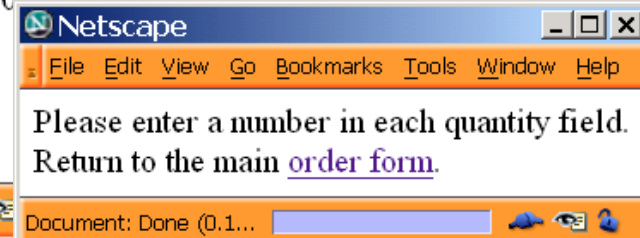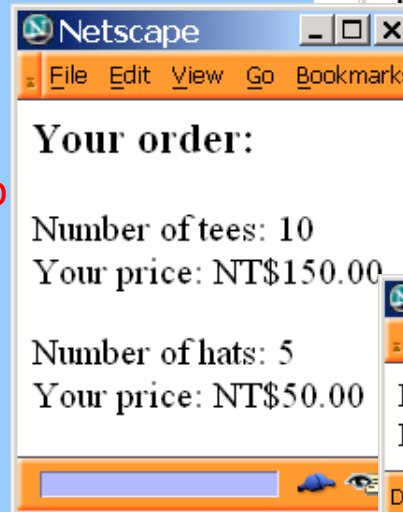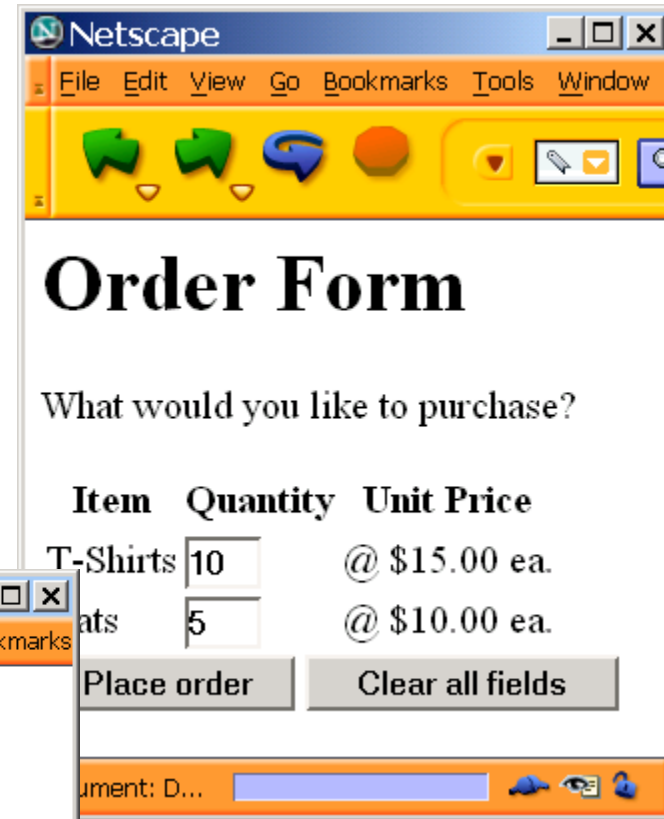
Including Pages: Result

Including Files at Page Translation Time

- **Format**
  - <%@ include file="Relative URL" %>
- **Purpose**
  - To reuse JSP content in multiple pages, where JSP content affects main page
- **Notes**
  - Servers are not required to detect changes to the included file, and in practice many don't
  - Thus, you need to change the JSP files whenever the included file changes
  - You can use OS-specific mechanisms such as the Unix "touch" command, or
    - <%-- Navbar.jsp modified 3/1/02 --%>
      <%@ include file="Navbar.jsp" %>

Reusable JSP Content: ContactSection.jsp

```jsp
<%@ page import="java.util.Date" %>
<%-- The following become fields in each servlet that
     results from a JSP page that includes this file. --%>
<%!
private int accessCount = 0;
private Date accessDate = new Date();
private String accessHost = "<I>No previous access</I>";
%>
<P><HR>
This page &copy; 2000
<A HREF="http//www.my-company.com/">my-company.com</A>.
This page has been accessed <%= ++accessCount %>
times since server reboot. It was last accessed from
<%= accessHost %> at <%= accessDate %>.
<% accessHost = request.getRemoteHost(); %>
<% accessDate = new Date(); %>
```

```
...
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
    Some Random Page</TABLE>
<P> Information about our products and services.
<P> Blah, blah, blah.
<P> Yadda, yadda, yadda.
<%@ include file="ContactSection.jsp" %>
</BODY>
</HTML>
```
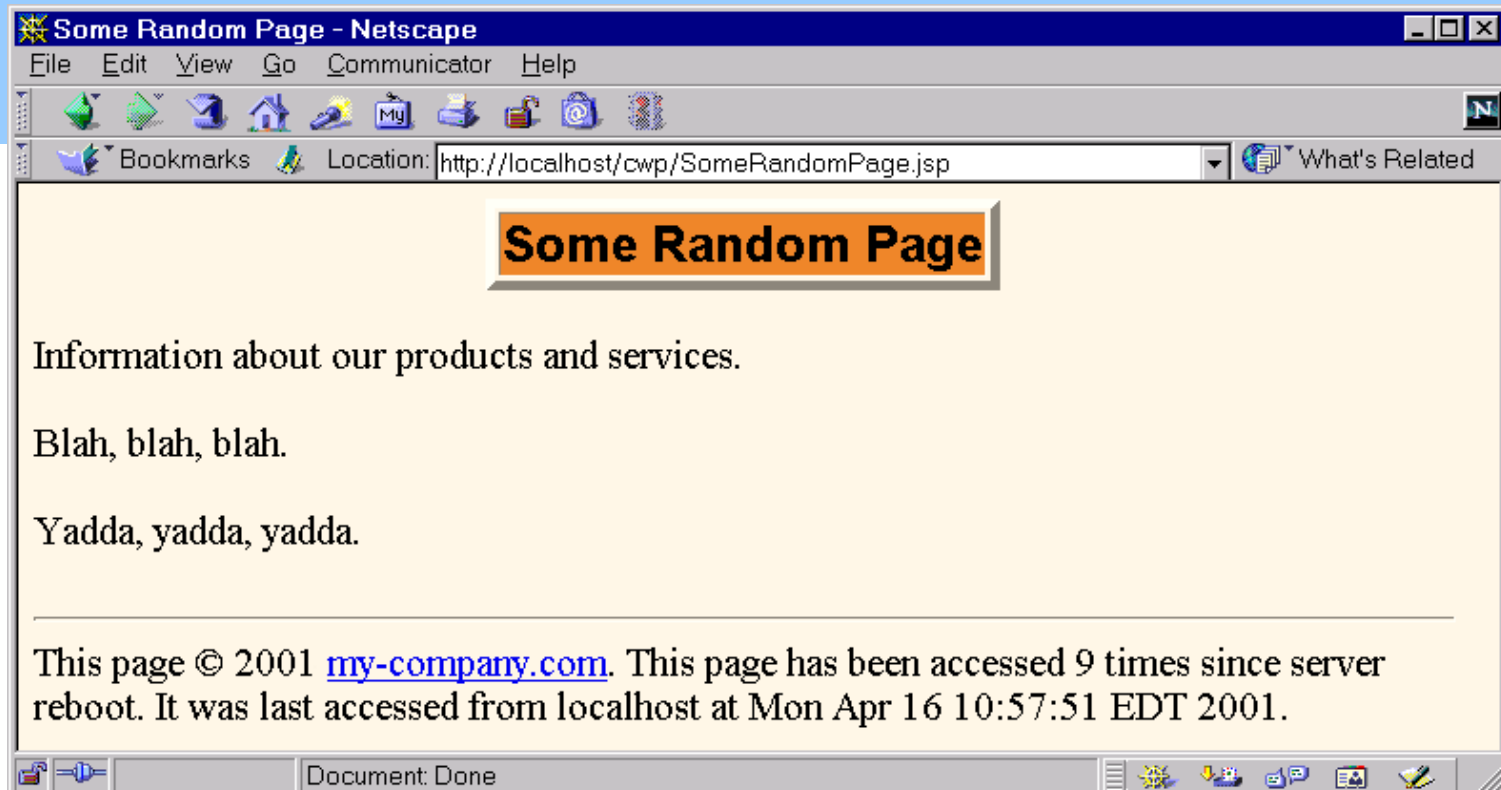
Include directive vs. Include action

A request-time include action

| `<jsp:include page="test.jsp"` `…` | `<test.jsp>` `&copy …` |

translation → translation →

| `pageContext.include(test.jsp)` ·····> | `<test.jsp>` `&copy …` |
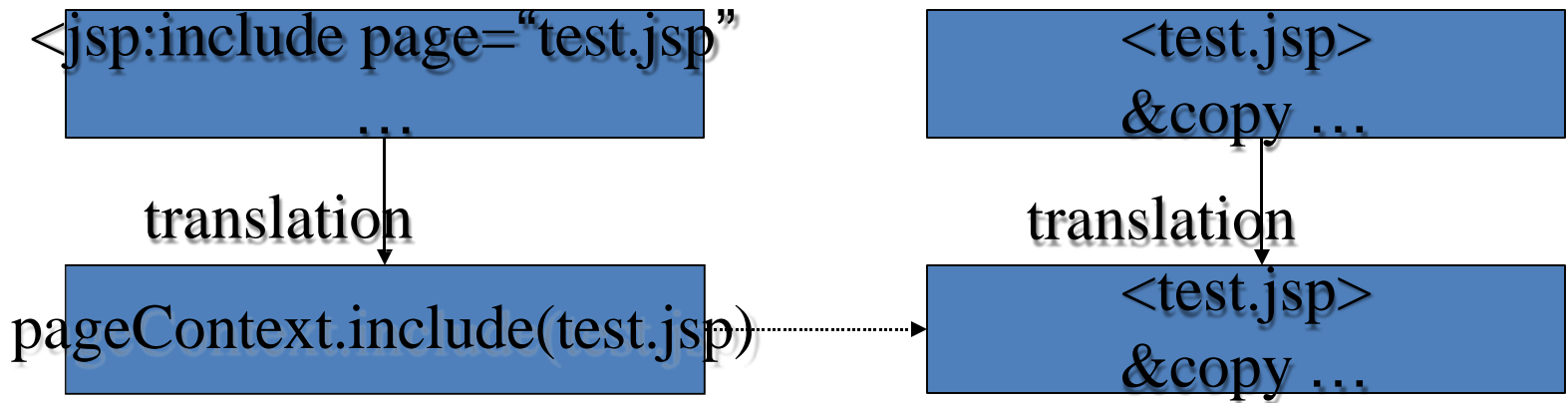
A translation-time include directive

| `<%@ include file="test.jsp"` `…` | `<test.jsp>` `&copy …` |

translation

| `&copy` `…` |
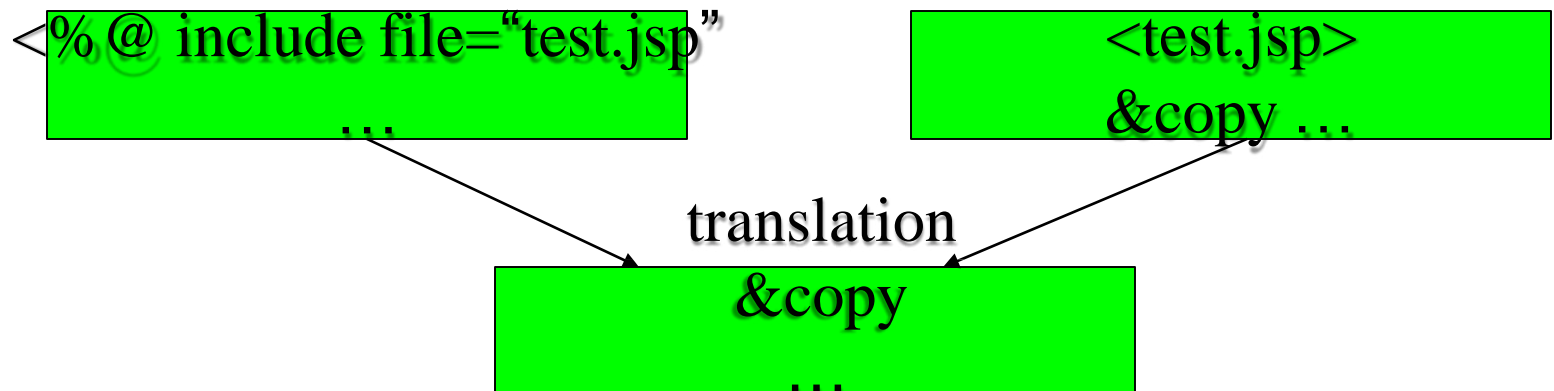
The plugin action's attribute

- \<jsp:plugin type="applet" code="myBox" codebase="path/myClass" width="200" height=200">… params \</jsp:plugin>
- We usually use
  - type: to specify we place an applet or others onto a web page.
  - Code: to give the name of the Java class to be run.
  - Width/Height: to define the size of the rectangle set aside for displaying the applet in the browser's window.

jsp:forward action

- Used to instruct a web server to stop processing the current page and start another one.

  <jsp:forward page="another.jsp">

  <jsp:param name="callingPage" value="current.jsp">

  </jsp:forward>

  – Another page:

    - <%= request.getParameter("callingPage")%>

    - Returns "current.jsp"

Uses of JSP Constructs: Using JavaBeans

Simple Application
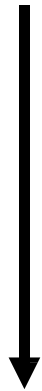
↓

Complex Application

- Scripting elements calling servlet code directly
- Scripting elements calling servlet code indirectly (by means of utility classes)
- Beans
- Custom tags
- Servlet/JSP combo (MVC architecture)

Background: What Are Beans?

- Classes that follow certain conventions
  - Must have a zero-argument (empty) constructor
  - Should have no public instance variables (fields)
  - Persistent values should be accessed through methods called getXxx and setXxx
    - If class has method getTitle that returns a String, class is said to have a String property named title
    - Boolean properties use isXxx instead of getXxx

- For more on beans, see http://java.sun.com/beans/docs/

Basic Bean Use in JSP

- Format: <jsp:useBean id="name"
                    class="package.Class" />
- Purpose: Allow instantiation of classes without explicit Java syntax
- Notes
  - Simple interpretation: JSP action
    ```
    <jsp:useBean id="book1" class="cwp.Book" />
    ```
    can be thought of as equivalent to the scriptlet
    ```
    <% cwp.Book book1 = new cwp.Book(); %>
    ```
  - But useBean has two additional features
    - Simplifies setting fields based on incoming request params
    - Makes it easier to share beans

Accessing Bean Properties

- Format: <jsp:getProperty
  name="name"
  property="property" />

- Purpose: Allow access to bean properties (i.e., calls to getXxx methods) without explicit Java code

- Notes

  - `<jsp:getProperty name="book1" property="title" />`

    is equivalent to the following JSP expression
    `<%= book1.getTitle() %>`

Setting Bean Properties: Simple Case

- **Format:** <jsp:setProperty name="name" property="property" value="value" />
- **Purpose**
  - Allow setting of bean properties (i.e., calls to setXxx) without explicit Java code
- **Notes**
  - ```
    <jsp:setProperty name="book1"
                property="title"
                value="Core Servlets and JSP" />
    ```
    is equivalent to the following scriptlet
    ```
    <% book1.setTitle("Core Servlets and JSP"); %>
    ```

Example: StringBean

```java
public class StringBean {
  private String message = "No message specified";
  public String getMessage() {
    return(message);
  }
  public void setMessage(String message) {
    this.message = message;
  }
}
```

- Installed in normal servlet directory

```
<jsp:useBean id="stringBean" class="cwp.StringBean" />
<OL>
<LI>Initial value (getProperty):
    <I><jsp:getProperty name="stringBean"
                property="message" /></I>
<LI>Initial value (JSP expression):
    <I><%= stringBean.getMessage() %></I>
<LI><jsp:setProperty name="stringBean"
                property="message"
                value="Best string bean: Fortex" />
    Value after setting property with setProperty:
    <I><jsp:getProperty name="stringBean"
                property="message" /></I>
<LI>
<% stringBean.setMessage("My favorite: Kentucky Wonder"); %>
    Value after setting property with scriptlet:
    <I><%= stringBean.getMessage() %></I>
</OL>
```

JSP Page That Uses StringBean
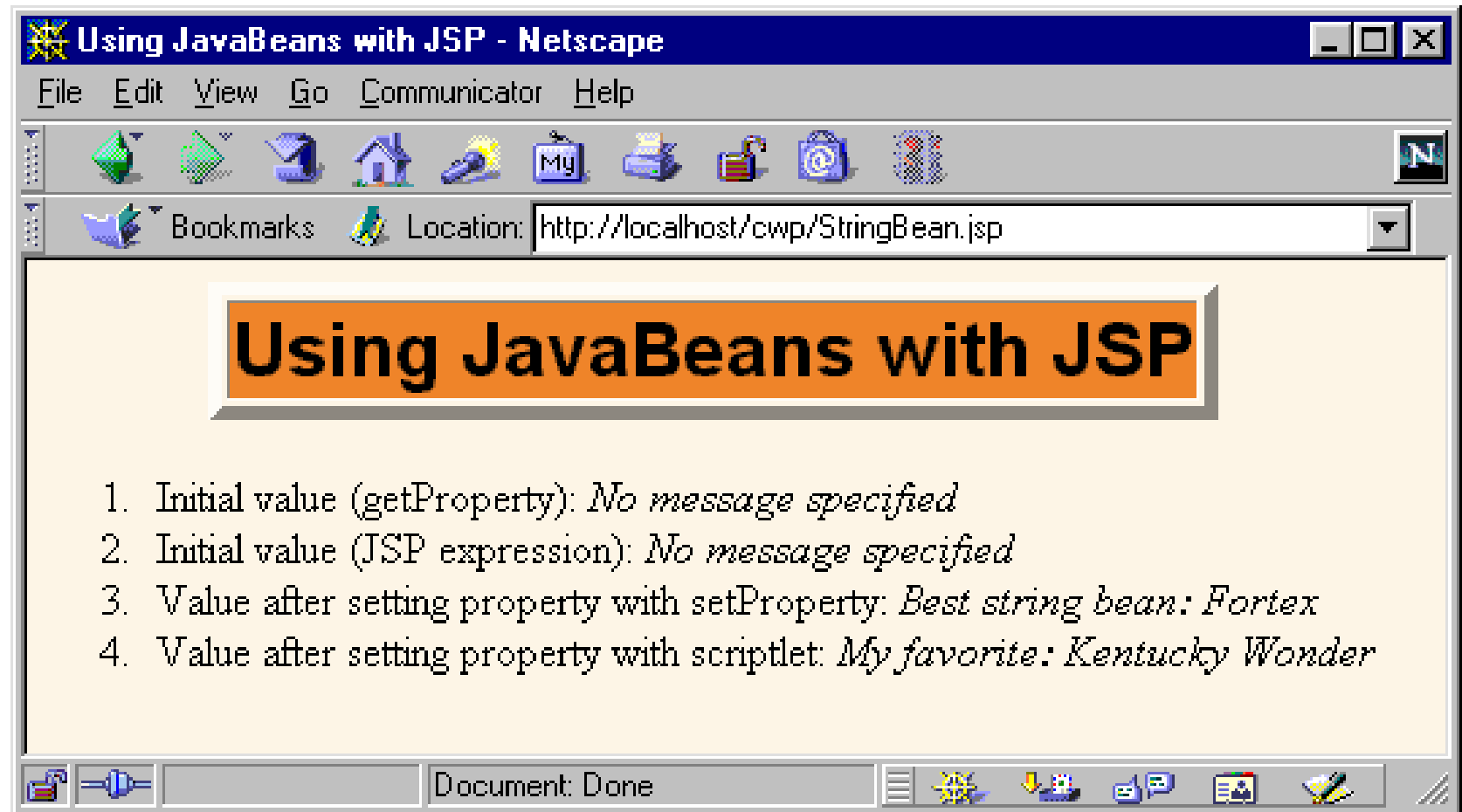
Associating Bean Properties with Request (Form) Parameters

- If property is a String, you can do
  - `<jsp:setProperty ... value='`<%= request.getParameter("...") %>`' />`
- Scripting expressions let you convert types, but you have to use Java syntax
- The param attribute indicates that:
  - Value should come from specified request param
  - Simple automatic type conversion performed
- Using "*" for the property attribute indicates that:
  - Value should come from request parameter whose name matches property name
  - Simple type conversion should be performed

Setting Bean Properties Case 1:
Explicit Conversion & Assignment

```
<!DOCTYPE ...>
...
<jsp:useBean id="entry"
          class="cwp.SaleEntry" />

<%-- getItemID expects a String --%>
<jsp:setProperty
    name="entry"
    property="itemID"
    value='<%=
```

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib ...>
<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>cwp</shortname>
  <info>
    A tag library from Core Web Programming 2nd Edition,
    http://www.corewebprogramming.com/.
  </info>
  <tag>
    <name>simplePrime</name>
    <tagclass>cwp.tags.SimplePrimeTag</tagclass>
    <info>Outputs a random 50-digit prime.</info>
  </tag>
</taglib>
```

Accessing Custom Tags From JSP Files

- Import the tag library
  - Specify location of TLD file

    <%@ taglib uri= "cwp-taglib.tld" prefix= "cwp" %>
  - Define a tag prefix (namespace)

    <%@ taglib uri="cwp-taglib.tld" prefix= "cwp" %>
- Use the tags
  - <prefix:tagName />
    - Tag name comes from TLD file
    - Prefix comes from taglib directive
  - E.g., <cwp:simplePrime />

...

Some 50-Digit Primes

```
<%@ taglib uri="cwp-taglib.tld" prefix="cwp" %>
<UL>
  <LI><cwp:simplePrime />   <LI><cwp:simplePrime />
  <LI><cwp:simplePrime />   <LI><cwp:simplePrime />
</UL>
```


Some 50-Digit Primes

- 85525203807603181442992536487526237946236109329293
- 94866654313655363893733986180399466323070643332731
- 34410418758257965902314053586112120565127145728483
- 14536436218832698568115498683116771875392186140487
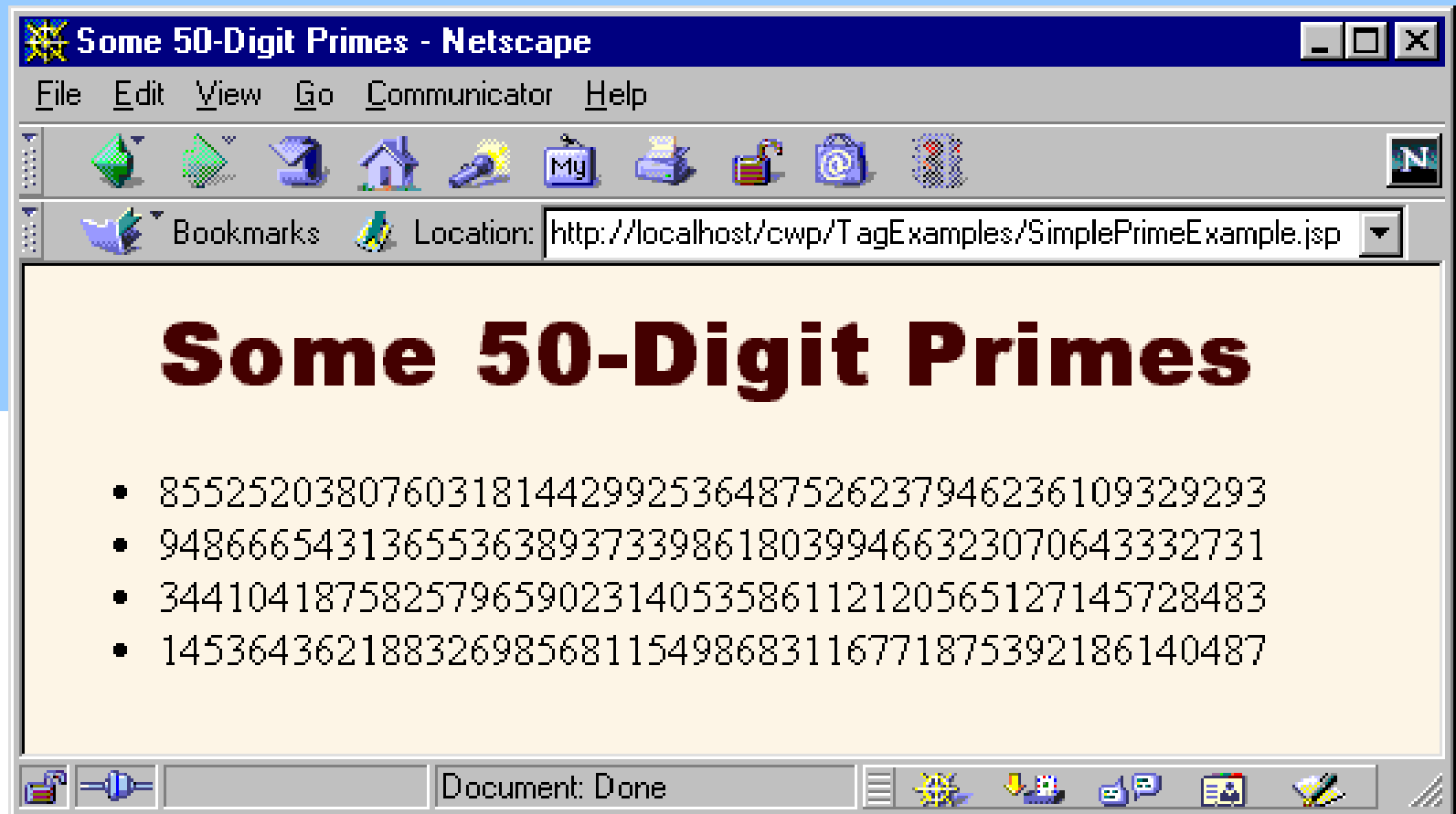
Intermediate and Advanced Custom Tags

- Tags with attributes
- Tags that include their body content
- Tags that optionally include their body
- Tags that manipulate their body
- Tags that manipulating their body multiple times (looping tags)
- Nested tags
- See book for details (related chapter online in PDF at Java Developer's Connection)
  - http://developer.java.sun.com/developer/Books/cservletsjsp/

Tags with Attributes: Prime Tag

```
...
<H1>Some N-Digit Primes</H1>
<%@ taglib uri="cwp-taglib.tld" prefix="cwp" %>
<UL>
  <LI>20-digit: <cwp:prime length="20" />
  <LI>40-digit: <cwp:prime length="40" />
  <LI>80-digit: <cwp:prime length="80" />
  <LI>Default (50-digit): <cwp:prime />
</UL>
```


Some N-Digit Primes - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://localhost/cwp/TagExamples/PrimeExample.jsp

Some N-Digit Primes

- 20-digit: 32202893832239491679
- 40-digit: 6657206684091561727357099946005524053317
- 80-digit: 41557069270593072733338187916203903965223542415375887442486332222481331561431357
- Default (50-digit): 67648745674655715012021344027143131834812065849301

Done   Local intranet

Including Body Content: heading Tag

```
…
<%@ taglib uri="cwp-taglib.tld" prefix="cwp" %>
<cwp:heading bgColor="#C0C0C0">
Default Heading
</cwp:heading>
<P>
<cwp:heading bgColor="BLACK" color="WHITE">
White on Black Heading
</cwp:heading>
<P>
<cwp:heading bgColor="#EF8429" fontSize="60" border="5">
Large Bordered Heading
</cwp:heading>
…
```
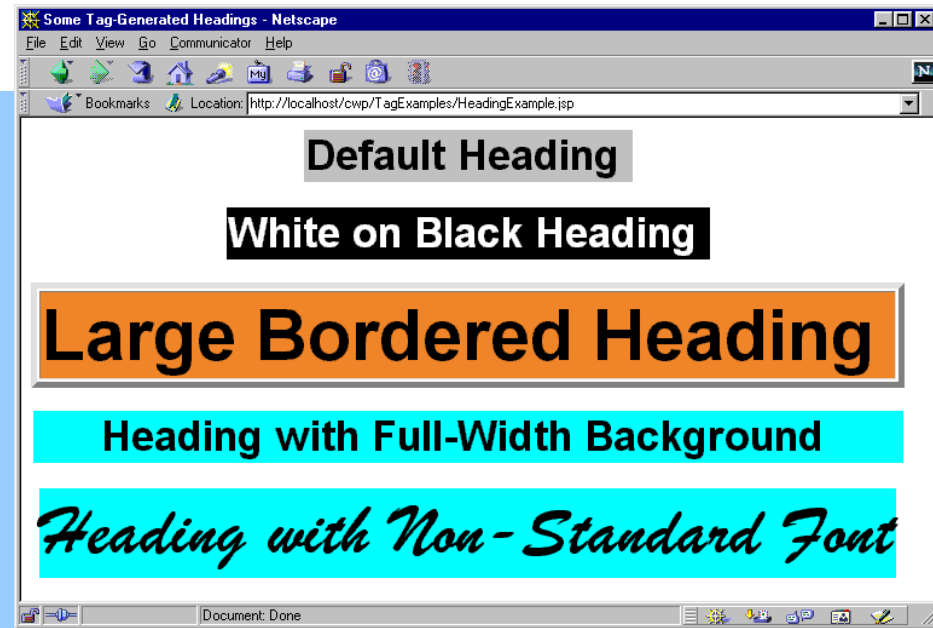
Optionally Including Tag Body: debug Tag

```
<%@ taglib uri="cwp-taglib.tld" prefix="cwp" %>
Top of regular page. Blah, blah, blah.
Yadda, yadda, yadda.
<P>
<cwp:debug>
<B>Debug:</B>
<UL>
  <LI>Current time: <%= new java.util.Date() %>
   <LI>Requesting hostname: <%= request.getRemoteHost()%>
   <LI>Session ID: <%= session.getId() %>
</UL>
</cwp:debug>
<P>
Bottom of regular page. Blah, blah, blah.
 Yadda, yadda, yadda.
```
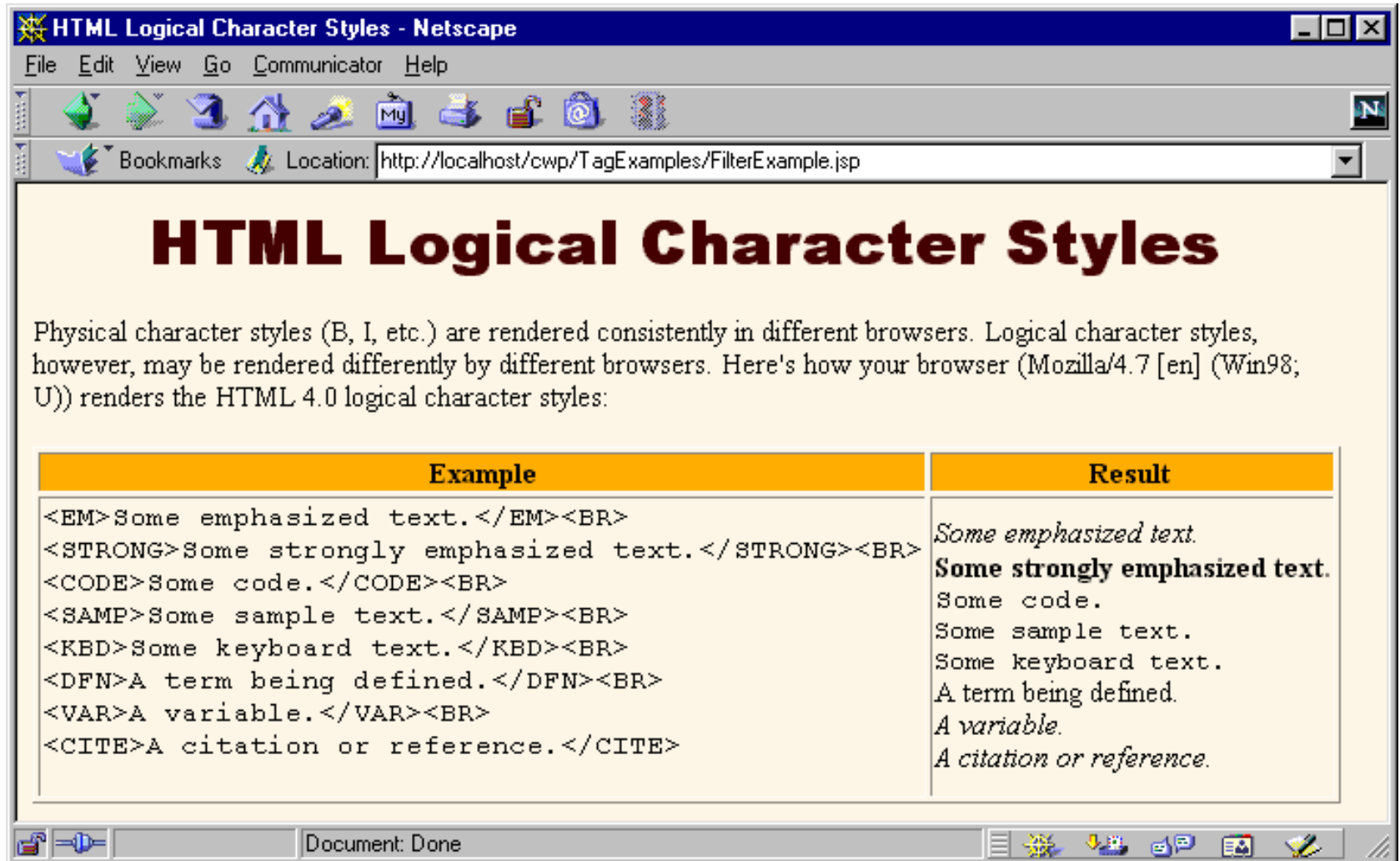
Using debug Tag: Results
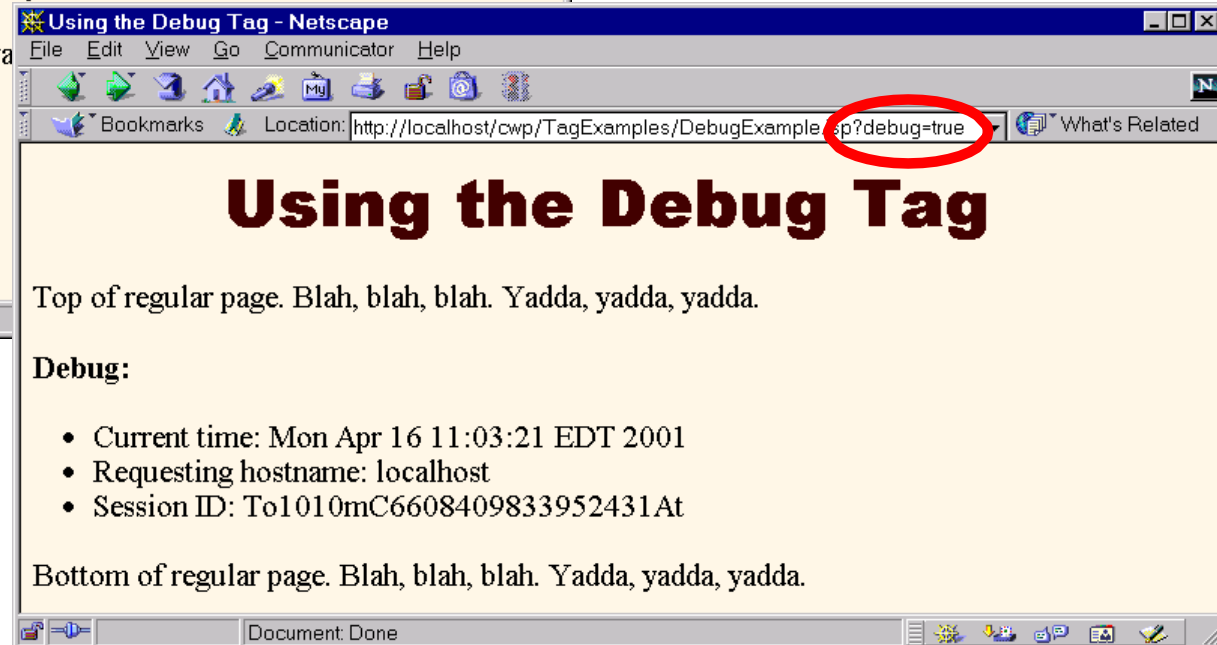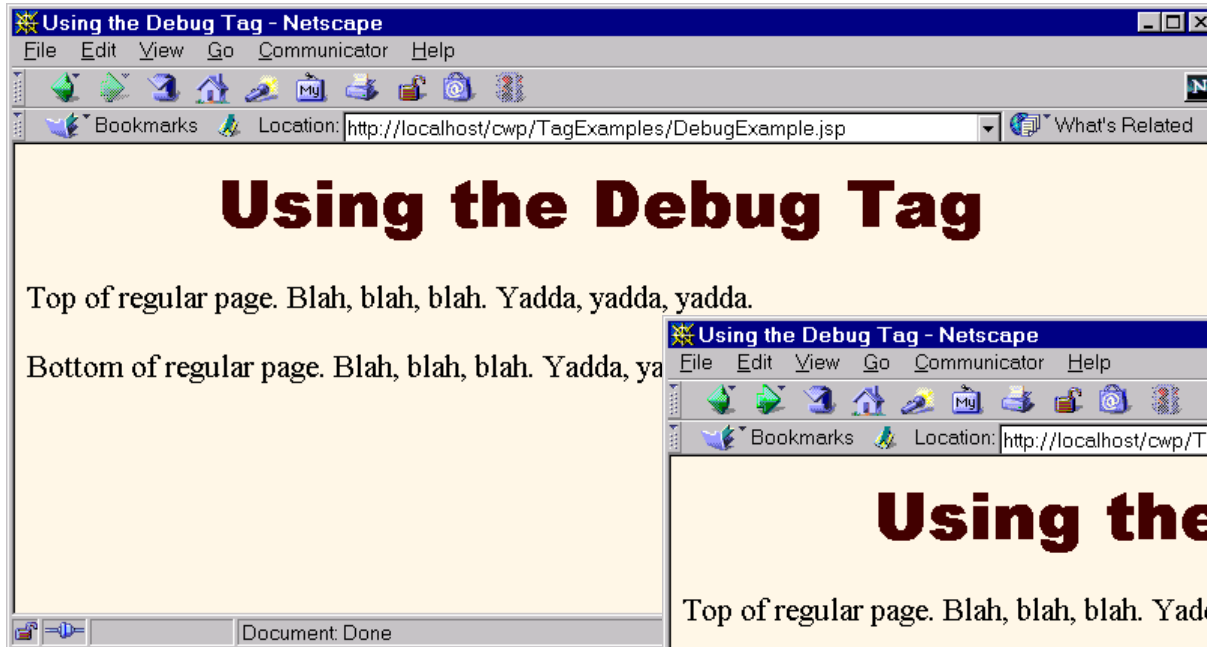
Manipulating Tag Body: the filter Tag

```
<%@ taglib uri="cwp-taglib.tld" prefix="cwp" %>
<TABLE BORDER=1 ALIGN="CENTER">
<TR CLASS="COLORED"><TH>Example<TH>Result
<TR>
<TD><PRE><cwp:filter>
<EM>Some emphasized text.</EM><BR>
<STRONG>Some strongly emphasized text.</STRONG><BR>
<CODE>Some code.</CODE><BR>

...
</cwp:filter></PRE>
<TD>
<EM>Some emphasized text.</EM><BR>
<STRONG>Some strongly emphasized text.</STRONG><BR>
<CODE>Some code.</CODE><BR>

...
</TABLE>
```

Using the filter Tag: Results

```jsp
<%@ taglib uri="cwp-taglib.tld" prefix="cwp" %>
<OL>
<!-- Repeats N times. A null reps value
     means repeat once. -->
<cwp:repeat
  reps='<%= request.getParameter("repeats") %>'>
  <LI><cwp:prime length="40" />
</cwp:repeat>
</OL>
```

Manipulating the
Body Multiple Times:
the repeat Tag

Some 40-Digit Primes — Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://localhost/cwp/TagExamples/RepeatExample.jsp?repeats=7

Some 40-Digit Primes

Each entry in the following list is the first prime number higher than
a randomly selected 40-digit number.

1. 7790427884846922906964969059906599223259
2. 5654911225096957336845971755232574136573
3. 4440250773563950124888671855331783776667
4. 4557511042237325396986257256572883741037
5. 5844763612419953952527117862322464484577
6. 2878822708441674101514196529821727599631
7. 9665985263198820747432458513545913598401

Done   Local intranet

Open Source Tag Libraries
http://jakarta.apache.org/taglibs/

- Internationalization (I18N)
- Database access
- Sending email
- JNDITM
- Date/time
- Populating/validating form fields
- Perl regular expressions
- Extracting data from other Web pages
- XSL transformations
- Etc

Break Time – 15 minutes

Uses of JSP Constructs:
Integrating Servlets and JSP

Simple Application
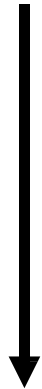
Complex Application

- Scripting elements calling servlet code directly
- Scripting elements calling servlet code indirectly (by means of utility classes)
- Beans
- Custom tags
- Servlet/JSP combo (MVC architecture)

Why Combine Servlets & JSP?

- Typical picture: use JSP to make it easier to develop and maintain the HTML content
  - For simple dynamic code, call servlet code from scripting expressions
  - For moderately complex cases, use custom classes called from scripting expressions
  - For more complicated cases, use beans and custom tags
- But, that's not enough
  - For complex processing, JSP is awkward
  - Despite the convenience of separate classes, beans, and custom tags, *the assumption behind JSP is that a single page gives a single basic look*

Integrating Servlets and JSP : Architecture

- Approach
  - Original request is answered by a servlet
  - Servlet processes request data, does database lookup, accesses business logic, etc.
  - Results are placed in beans
  - Request is forwarded to a JSP page to format result
  - Different JSP pages can be used to handle different types of presentation
- Terminology
  - Often called the "Model View Controller" architecture or "Model2" approach to JSP
  - Formalized further with Apache "Struts" framework
    - See http://jakarta.apache.org/struts/

Dispatching Requests

- First, call the getRequestDispatcher method of ServletContext
  - Supply a URL relative to the Web application root
  - Example
    - String url = "/presentations/presentation1.jsp";
      RequestDispatcher dispatcher =
      getServletContext().getRequestDispatcher(url);
- Second
  - Call forward to completely transfer control to destination page. See following example
  - Call include to insert output of destination page and then continue on.

```java
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
  String operation = request.getParameter("operation");
  if (operation == null) {
    operation = "unknown";
  }
  if (operation.equals("operation1")) {
    gotoPage("/operations/presentation1.jsp",
          request, response);
  } else if (operation.equals("operation2")) {
    gotoPage("/operations/presentation2.jsp",
          request, response);
  } else {
    gotoPage("/operations/unknownRequestHandler.jsp",
          request, response);
  }
}

private void gotoPage(String address,
      HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
  RequestDispatcher dispatcher =
  getServletContext().getRequestDispatcher(address);
  dispatcher.forward(request, response);
}
```

Reminder: JSP useBean Scope Alternatives

- request
  - `<jsp:useBean id="..." class="..." scope="request" />`

- session
  - `<jsp:useBean id="..." class="..." scope="session" />`

- application
  - `<jsp:useBean id="..." class="..." scope="application" />`

- page
  - `<jsp:useBean id="..." class="..." scope="page" />`
    or just
    `<jsp:useBean id="..." class="..." />`
  - This scope is not used in MVC architecture

Storing Data for Later Use: The Servlet Request

- Purpose
  - Storing data that servlet looked up and that JSP page will use only in this request.
- Servlet syntax to store data

  SomeClass value = new SomeClass(…);

  request.setAttribute("key", value);

  // Use RequestDispatcher to forward to JSP page

- JSP syntax to retrieve data

  <jsp:useBean id="key"
      class="SomeClass" scope="request" />

Storing Data for Later Use: The Session Object

- Purpose
  - Storing data that servlet looked up and that JSP page will use in this request and in later requests from same client.

- Servlet syntax to store data

  SomeClass value = new SomeClass(…);

  HttpSession session = request.getSession(true);

  session.setAttribute("key", value);

  // Use RequestDispatcher to forward to JSP page

- JSP syntax to retrieve data

  ```
  <jsp:useBean id="key"
              class="SomeClass" scope="session" />
  ```

Storing Data for Later Use: The Servlet Context

- Purpose
  - Storing data that servlet looked up and that JSP page will use in this request and in later requests from any client.

- Servlet syntax to store data

  SomeClass value = new SomeClass(…);

  getServletContext().setAttribute("key", value);

  // Use RequestDispatcher to forward to JSP page

- JSP syntax to retrieve data

  <jsp:useBean
      id="key"
      class="SomeClass"
      scope="application" />

An On-Line Travel Agent

Review: JSP Introduction

- JSP makes it easier to create/maintain HTML, while still providing full access to servlet code
- JSP pages get translated into servlets
  - It is the servlets that run at request time
  - Client does not see anything JSP-related
- You still need to understand servlets
  - Understanding how JSP really works
  - Servlet code called from JSP
  - Knowing when servlets are better than JSP
  - Mixing servlets and JSP

Uses of JSP Constructs

Simple Application
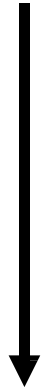
↓

Complex Application

- Scripting elements calling servlet code directly
- Scripting elements calling servlet code indirectly (by means of utility classes)
- Beans
- Custom tags
- Servlet/JSP combo (MVC architecture)

Review: Calling Java Code Directly: JSP Scripting Elements

- ▣ JSP Expressions
  - Format: <%= expression %>
  - Evaluated and inserted into the servlet's output.
- ▣ JSP Scriptlets
  - Format: <% code %>
  - Inserted verbatim into the _jspService method
- ▣ JSP Declarations
  - Format: <%! code %>
  - Inserted verbatim into the body of the servlet class
- ▣ Predefined variables
  - request, response, out, session, application
- ▣ Limit the Java code in page
  - Use helper classes, beans, custom tags, servlet/JSP combo

Review: The JSP page Directive: Structuring Generated Servlets

- **The import attribute**
  - Changes the packages imported by the servlet that results from the JSP page
- The contentType attribute
  - Specifies MIME type of result
  - Cannot be used conditionally

Review: Including Files in JSP Documents

- **<jsp:include page="Relative URL" flush="true"/>**
  - Output inserted into JSP page at request time
  - Cannot contain JSP content that affects entire page
  - Changes to included file do not necessitate changes to pages that use it
- **<%@ include file="Relative URL" %>**
  - File gets inserted into JSP page prior to page translation
  - Thus, file can contain JSP content that affects entire page (e.g., import statements, declarations)
  - Changes to included file might require you to manually update pages that use it

Review: Using JavaBeans Components with JSP

- Benefits of jsp:useBean
  - Hides the Java programming language syntax
  - Makes it easier to associate request parameters with objects (bean properties)
  - Simplifies sharing objects among multiple requests or servlets/JSPs
- jsp:useBean
  - Creates or accesses a bean
- jsp:getProperty
  - Puts bean property (i.e. getXxx call) into output
- jsp:setProperty
  - Sets bean property (i.e. passes value to setXxx)

Review: Creating Custom JSP Tag Libraries

- For each custom tag, you need
  - A tag handler class (usually extending TagSupport or BodyTagSupport)
  - An entry in a Tag Library Descriptor file
  - A JSP file that imports library, specifies prefix, and uses tags
- Simple tags
  - Generate output in doStartTag, return SKIP_BODY
- Attributes
  - Define setAttributeName method. Update TLD file
- Body content
  - doStartTag returns EVAL_BODY_INCLUDE
  - Add doEndTag method

Review: Integrating Servlets and JSP

- Use MVC (Model 2) approach when:
  - One submission will result in multiple basic looks
  - Several pages have substantial common processing
- Architecture
  - A servlet answers the original request
  - Servlet does the real processing & stores results in beans
    - Beans stored in HttpServletRequest, HttpSession, or ServletContext
  - Servlet forwards to JSP page via forward method of RequestDispatcher
  - JSP page reads data from beans by means of jsp:useBean with appropriate scope (request, session, or application)

UNIT-5

Java script

Introduction

- JavaScript is the most popular scripting language on the internet, and works in all major browsers, such as Internet Explorer, Firefox, Chrome, Opera, and Safari.

- **What is JavaScript?**
  - JavaScript is a scripting language
  - JavaScript is usually embedded directly into HTML pages
  - JavaScript was designed to add interactivity to HTML pages
  - JavaScript is an interpreted language (means that scripts execute without preliminary compilation)
  - Everyone can use JavaScript without purchasing a license

Introduction

What can a JavaScript do?

- **JavaScript gives HTML designers a programming tool -** HTML authors are normally not programmers, but JavaScript is a scripting language with a very simple syntax! Almost anyone can put small "snippets" of code into their HTML pages

- **JavaScript can put dynamic text into an HTML page -** A JavaScript statement like this: document.write("<h1>" + name + "</h1>") can write a variable text into an HTML page

- **JavaScript can react to events -** A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element

Introduction

- **JavaScript can be used to validate data -** A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing

- **JavaScript can be used to create cookies** - A JavaScript can be used to store and retrieve information on the visitor's computer

- **JavaScript can read and write HTML elements -** A JavaScript can read and change the content of an HTML element

Introduction

Put a JavaScript into an HTML page

- The example below shows how to use JavaSript to write text on a web page:
- The HTML <script> tag is used to insert a JavaScript into an HTML page.
- **Example**
  <html>
  <body>
  <script type="text/javascript">
  document.write("Hello World!");
  </script>
  </body>
  </html>

Introduction
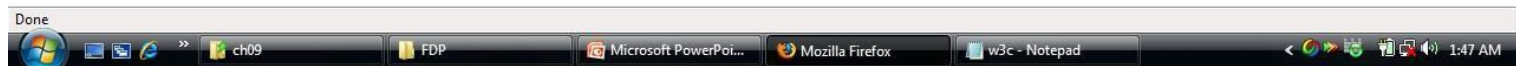
Introduction

The example below shows how to add HTML tags to the JavaScript:

Example

```
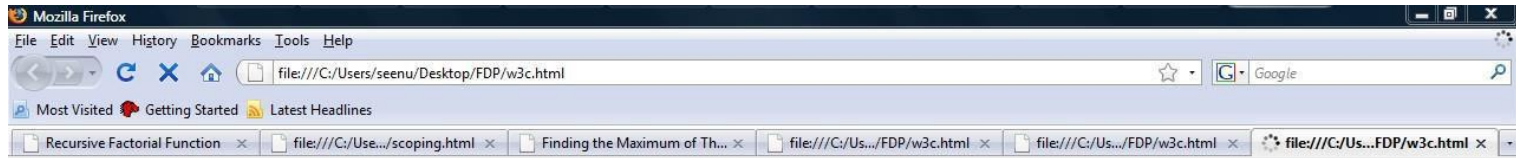<html>
<body>
<script type="text/javascript">
document.write("<h1>Hello World!</h1>");
</script>
</body>
</html>
```

– The **document.write** command is a standard JavaScript commandfor writing output to a page.

– JavaScripts in the body section will be executed WHILE the page loads.

– JavaScripts in the head section will be executed when CALLED.

Introduction
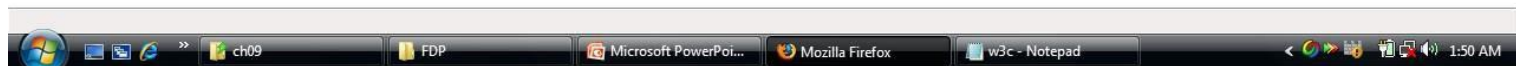
Introduction

Scripts in <body>

- Scripts to be executed when the page loads go in the body section.
- If you place a script in the body section, it generates the content of a page.

Example

```
<html>
<head>
</head>

<body>
Introduction<script type="text/javascript">
document.write("This message is written by JavaScript");
</script>
</body>

</html>
```

Introduction

Introduction

Scripts in <head> and <body>

- You can place an unlimited number of scripts in your document, so you can have scripts in both the body and the head section.

```
<html>
<head>
<script type="text/javascript">
....
</script>
</head>
<body>
<script type="text/javascript">
....
</script>
</body>
```

Introduction

JavaScript Statements

- JavaScript is a sequence of statements to be executed by the browser.

JavaScript is Case Sensitive

- Unlike HTML, JavaScript is case sensitive - therefore watch your capitalization closely when you write JavaScript statements, create or call variables, objects and functions

- The semicolon is optional (according to the JavaScript standard), and the browser is supposed to interpret the end of the line as the end of the statement. Because of this you will often see examples without the semicolon at the end.

- **Note:** Using semicolons makes it possible to write multiple statements on one line.

Introduction

JavaScript Code

- JavaScript code (or just JavaScript) is a sequence of JavaScript statements.

- Each statement is executed by the browser in the sequence they are written.

- This example will write a heading and two paragraphs to a web page:

Example

```
<script type="text/javascript">
document.write("<h1>This is a heading</h1>");
document.write("<p>This is a paragraph.</p>");
document.write("<p>This is another paragraph.</p>");
</script>
```

Introduction

JavaScript Blocks

- JavaScript statements can be grouped together in blocks.
- Blocks start with a left curly bracket {, and ends with a right curly bracket }.
- The purpose of a block is to make the sequence of statements execute together.
- This example will write a heading and two paragraphs to a web page:

Example

```
<script type="text/javascript">
{
document.write("<h1>This is a heading</h1>");
document.write("<p>This is a paragraph.</p>");
document.write("<p>This is another paragraph.</p>");
}
</script>
```

Introduction

Introduction

JavaScript Comments

- Comments can be added to explain the JavaScript, or to make the code more readable.

- Single line comments start with //.

- The following example uses single line comments to explain the code:

Example

```
<script type="text/javascript">
// Write a heading
document.write("<h1>This is a heading</h1>");
// Write two paragraphs:
document.write("<p>This is a paragraph.</p>");
document.write("<p>This is another paragraph.</p>");
</script>
```

Introduction

JavaScript Multi-Line Comments

- Multi line comments start with /* and end with */.

- The following example uses a multi line comment to explain the code:

Example

```
<script type="text/javascript">
/*
The code below will write
one heading and two paragraphs
*/
document.write("<h1>This is a heading</h1>");
document.write("<p>This is a paragraph.</p>");
document.write("<p>This is another paragraph.</p>");
</script>
```

Introduction

Using Comments at the End of a Line

- In the following example the comment is placed at the end of a code line:

Example

```
<script type="text/javascript">
document.write("Hello"); // Write "Hello"
document.write(" Dolly!"); // Write " Dolly!"
</script>
```

Introduction

Additional problems

```
<html>
  <head><title> Printing Multiple Lines</title>

    <script type = "text/javascript">
      <!--
      document.writeln( "<h1>Welcome to<br/>JavaScript" +          "<br
    />Programming!</h1>" );
</script>

  </head><body></body>
</html>
```

Introduction
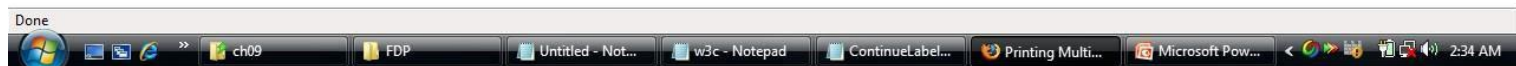
Introduction

JavaScript Variables

- As with algebra, JavaScript variables are used to hold values or expressions.
- A variable can have a short name, like x, or a more descriptive name, like carname.

Rules for JavaScript variable names:

- Variable names are case sensitive (y and Y are two different variables)
- Variable names must begin with a letter or the underscore character

Variables

Declaring (Creating) JavaScript Variables

- Creating variables in JavaScript is most often referred to as "declaring" variables.
- You can declare JavaScript variables with the var statement:

  var x;

  var carname;

- After the declaration shown above, the variables are empty (they have no values yet).
- However, you can also assign values to the variables when you declare them:

  var x=5;

  var carname="Volvo";

Cont..

Assigning Values to Undeclared JavaScript Variables

- If you assign values to variables that have not yet been declared, the variables will automatically be declared.

These statements:

x=5;

carname="Volvo";

Redeclaring JavaScript Variables

- If you redeclare a JavaScript variable, it will not lose its original value.

    var x=5;

    var x;

- After the execution of the statements above, the variable x will still have the value of 5. The value of x is not reset (or cleared) when you redeclare it.

Operators

JavaScript Arithmetic Operators

- Arithmetic operators are used to perform arithmetic between variables and/or values.

JavaScript Assignment Operators

- Assignment operators are used to assign values to JavaScript variables.

The + Operator Used on Strings

- The + operator can also be used to add string variables or text values together.

Operators

Comparison Operators

- Comparison operators are used in logical statements to determine equality or difference between variables or values.

Logical Operators

- Logical operators are used to determine the logic between variables or values.

Arithmatic operators

```
<html>
  <head>
    <title>An Addition Program</title>

    <script type = "text/javascript">
      var firstNumber,    // first string entered by user
          secondNumber, // second string entered by user
          number1,
            number2,
          sum;          // sum of number1 and number2

      // read in first number from user as a string
      firstNumber =
        window.prompt( "Enter first integer", "0" );

      // read in second number from user as a string
      secondNumber =
        window.prompt( "Enter second integer", "0" );
```

Operators

```
// convert numbers from strings to integers
    number1 = parseInt( firstNumber );
    number2 = parseInt( secondNumber );

    // add the numbers
    sum = number1 + number2;

    // display the results
    document.writeln( "<h1>The sum is " + sum + "</h1>" );
  </script>

  </head>
  <body>
    <p>Click Refresh (or Reload) to run the script again</p>
  </body>
</html>
```

Introduction-JavaScript Popup Boxes

Alert Box

- An alert box is often used if you want to make sure information comesthrough to the user.
- When an alert box pops up, the user will have to click "OK" to proceed.

```html
<html>
 <head><title>Printing Multiple Lines in a Dialog Box</title>

    <script type = "text/javascript">
      <!--
      window.alert( "Welcome to\nJavaScript\nProgramming!" );
      // -->
    </script>
 </head>

  <body>
    <p>Click Refresh (or Reload) to run this script again.</p>
  </body>
</html>
```

Introduction-JavaScript Popup Boxes
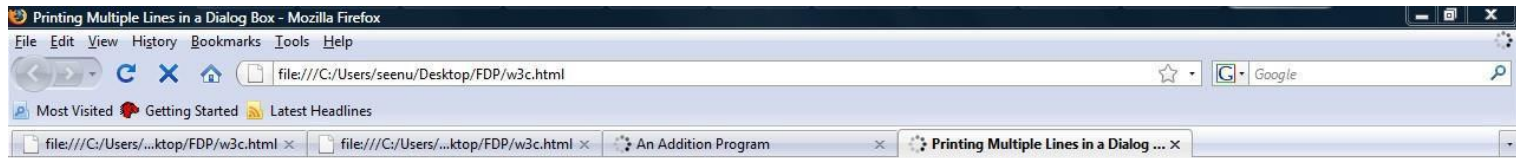
Introduction-JavaScript Popup Boxes

- **Confirm Box**

- A confirm box is often used if you want the user to verify or accept something.

- When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

- If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Introduction-JavaScript Popup Boxes

Example

```html
<html>
<head>
<script type="text/javascript">
function show_confirm()
{
var r=confirm("Press a button");
if (r==true)
  {
  document.write("You pressed OK!");
  }
else
  {
  document.write("You pressed Cancel!");
  }
}
</script>
</head>
<body>
<input type="button" onclick="show_confirm()" value="Show confirm box" />
</body>
</html>
```

Introduction-JavaScript Popup Boxes

Introduction-JavaScript Popup Boxes

Introduction-JavaScript Popup Boxes

Prompt Box

- A prompt box is often used if you want the user to input a value before entering a page.

- When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

- If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

```html
<html >
  <head>
    <title>Using Prompt and Alert Boxes</title>
    <script type = "text/javascript">
     var name; // string entered by the user
      // read the name from the prompt box as a string
      name = window.prompt( "Please enter your name", "GalAnt" );
      document.writeln( "<h1>Hello " + name +
        ", welcome to JavaScript programming!</h1>" );
     </script>

  </head>

  <body>
    <p>Click Refresh (or Reload) to run this script again.</p>
  </body>
</html>
```

Control structures-if

```
if ( hour < 12 )
        document.write( "<h1>Good Morning, " );

     // determine whether the time is PM
     if ( hour >= 12 )
     {
             hour = hour - 12;

             if ( hour < 6 )
         document.write( "<h1>Good Afternoon, " );

             if ( hour >= 6 )
         document.write( "<h1>Good Evening, " );
     }

      document.writeln( name +
        ", welcome to JavaScript programming!</h1>" );
     </script> </head>

<body> <p>Click Refresh (or Reload) to run this script again.</p> </body>
</html>
```

JavaScript Loops
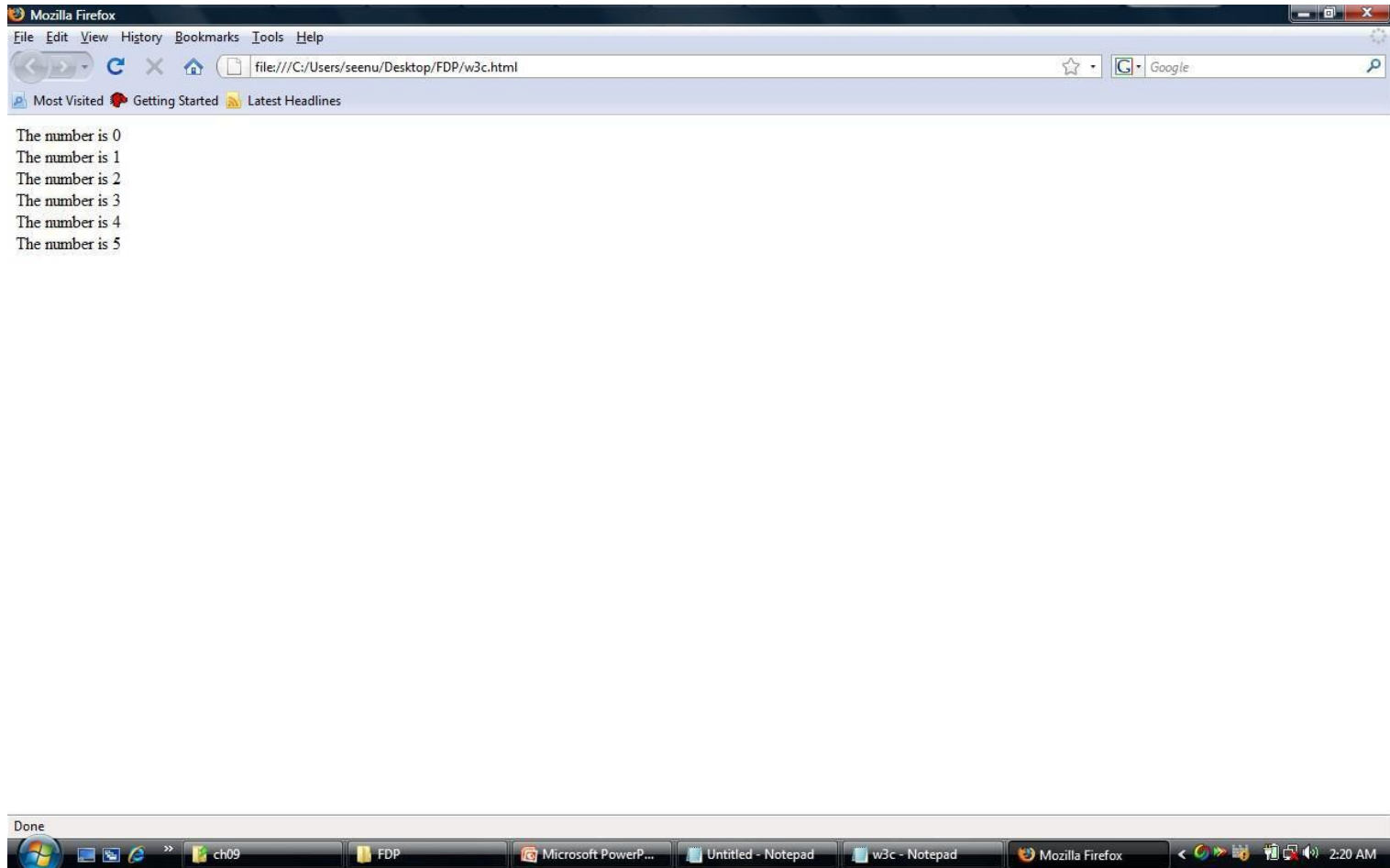
JavaScript Loops

JavaScript While Loop

- oops execute a block of code a specified number of times, or while a specified condition is true.

Example

```
<html>
<body>
<script type="text/javascript">
var i=0;
while (i<=5)
  {
  document.write("The number is " + i);
  document.write("<br />");
  i++;
  }
</script>
</body>
</html>
```

JavaScript Loops

The do...while Loop

- This loop will execute the block of code ONCE, and then it will repeat the loop as long as the specified condition is true.

Example

```
<html>
<body>
<script type="text/javascript">
var i=0;
do
  {
  document.write("The number is " + i);
  document.write("<br />");
  i++;
  }
while (i<=5);
</script>
</body>
</html>
```

JavaScript Break

The break Statement
- The break statement will break the loop and continue executing the code that follows after the loop(if any).

```
<html>
  <head>

    <script type = "text/javascript">
      <!--
      for ( var count = 1; count <= 10; ++count ) {
        if ( count == 5 )
          break; // break loop only if count == 5

        document.writeln( "Count is: " + count + "<br />" );
      }

      document.writeln(
        "Broke out of loop at count = " + count );
      // -->
    </script>

  </head><body></body>
</html>
```

JavaScript Break
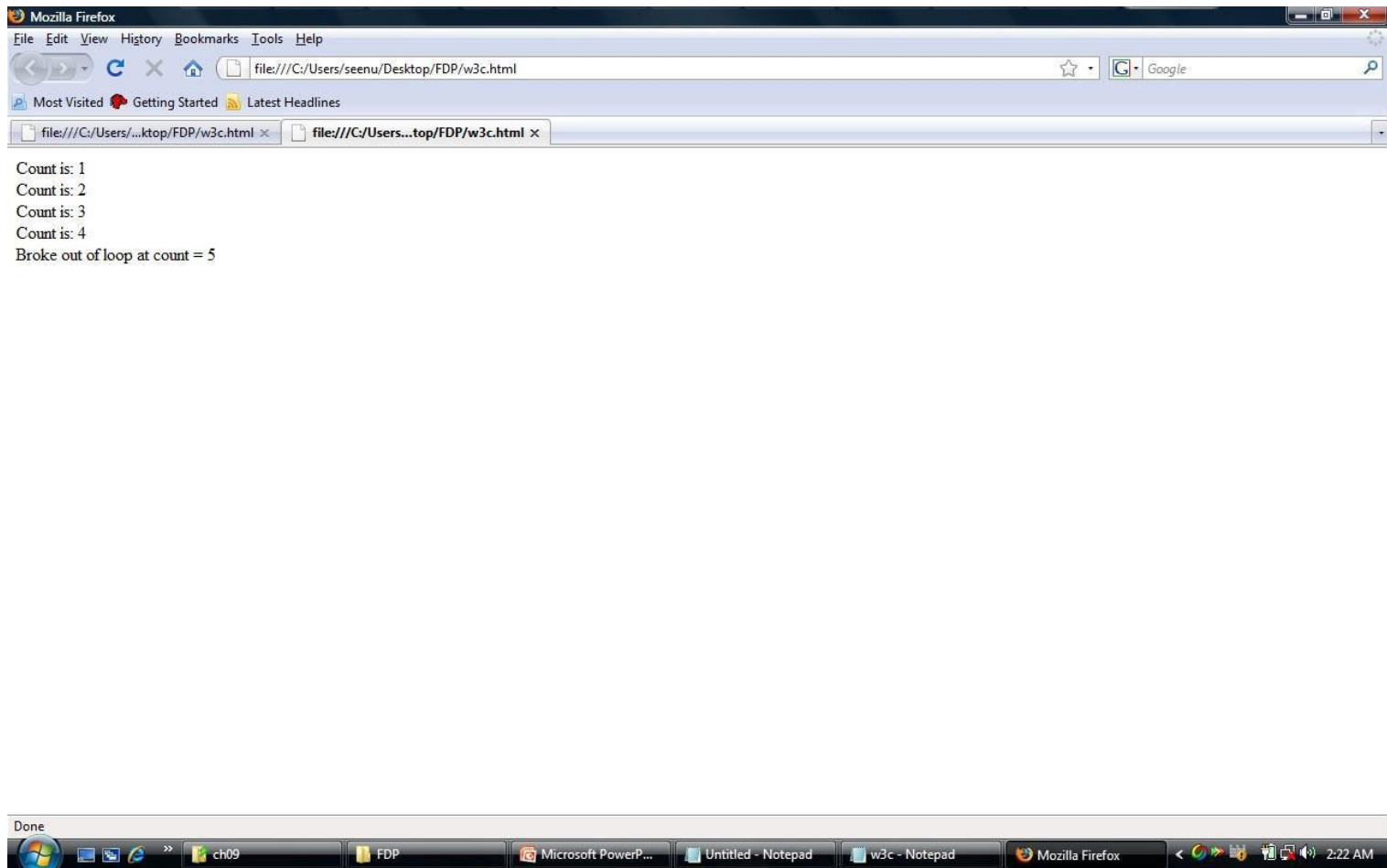
Continue Statements

The continue Statement
- The continue statement will break the current loop and continue with the next value

```
<html>
  <head>
      <script type = "text/javascript">
          for ( var count = 1; count <= 10; ++count ) {
      if ( count == 5 )
        continue; // skip remaining code in loop
                // only if count == 5

      document.writeln( "Count is: " + count + "<br />" );
    }
    document.writeln( "Used continue to skip printing 5" );
</script>
  </head><body></body>
</html>
```

Continue Statements
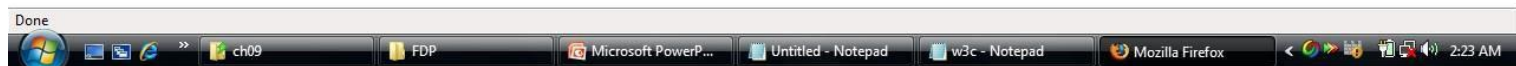
The labeled continue Statement

```html
<html >
  <head>
    <title>Using the continue Statement with a Label</title>
    <script type = "text/javascript">
        nextRow:   // target label of continue statement
          for ( var row = 1; row <= 5; ++row ) {
            document.writeln( "<br />" );
            for ( var column = 1; column <= 10; ++column ) {

              if ( column > row )
                continue nextRow; // next iteration of labeled loop
              document.write( "* " );
            }
         }
</script>
  </head><body></body>
</html>
```

The labeled continue Statement
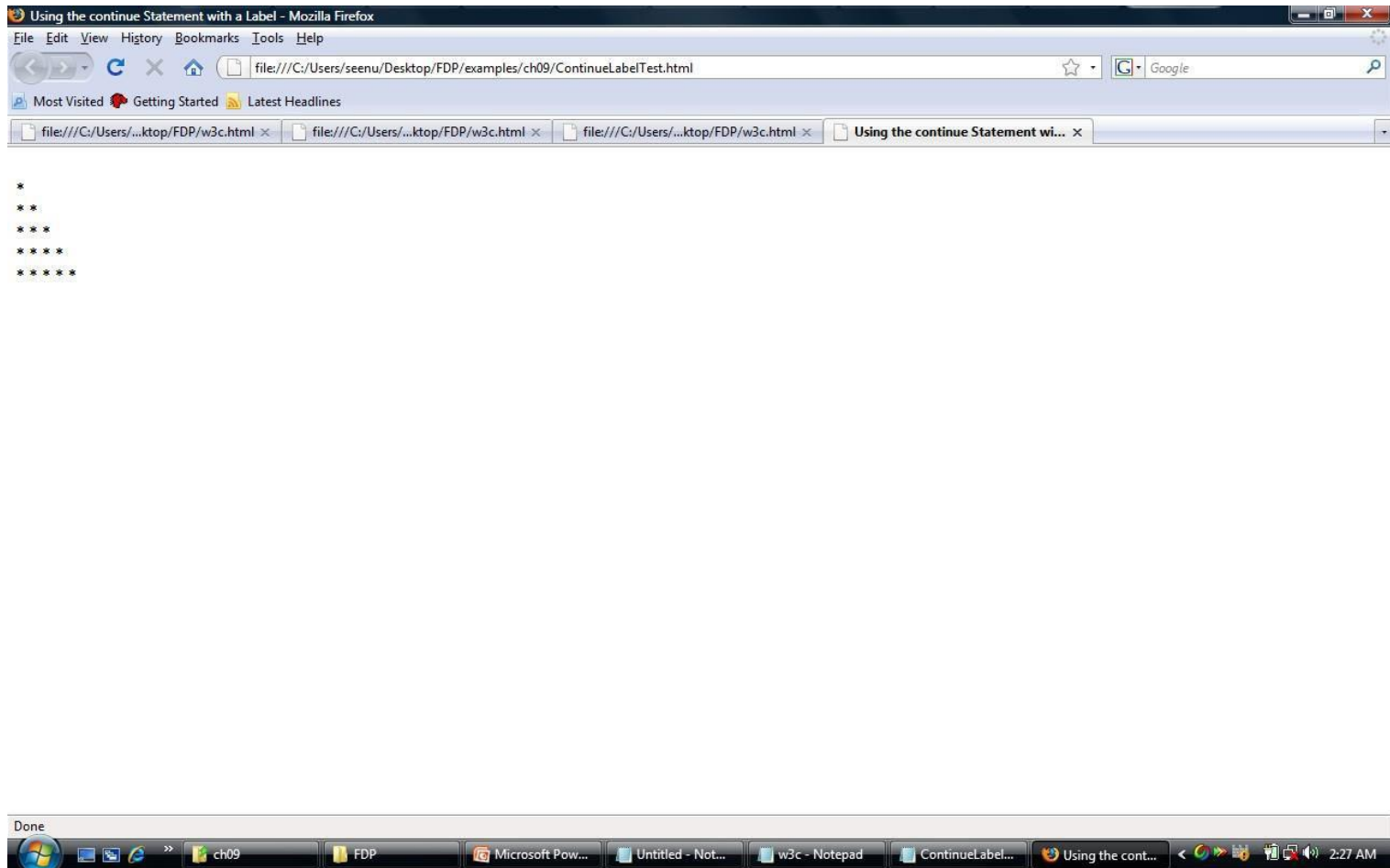
The labeled break Statement

```
<html>
  <head>
      <script type = "text/javascript">
      stop: { // labeled block
        for ( var row = 1; row <= 10; ++row ) {
          for ( var column = 1; column <= 5 ; ++column ) {

              if ( row == 5 )
                break stop; // jump to end of stop block

              document.write( "* " );
            }

            document.writeln( "<br />" );
          }

          // the following line is skipped
          document.writeln( "This line should not print" );
        }

      document.writeln( "End of script" );
</script>

  </head><body></body>
</html>
```

The labeled break Statement
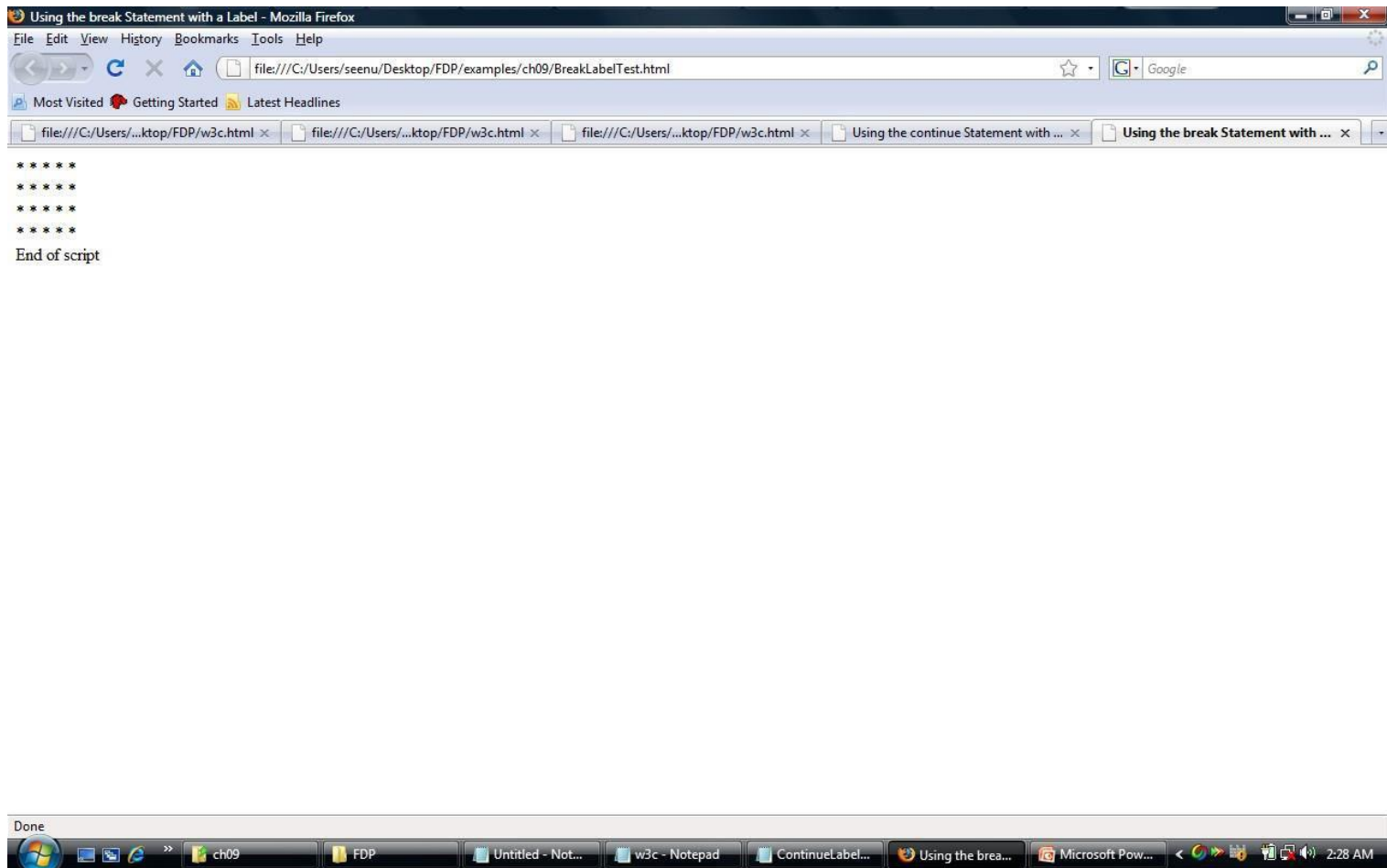
JavaScript Try...Catch Statement

- The try...catch statement allows you to test a block of code for errors.

JavaScript - Catching Errors

- When browsing Web pages on the internet, we all have seen a JavaScript alert box telling us there is a runtime error and asking "Do you wish to debug?". Error message like this may be useful for developers but not for users. When users see errors, they often leave the Web page.

- This chapter will teach you how to catch and handle JavaScript error messages, so you don't lose your audience.

JavaScript Try...Catch Statement

Example

- ```html
  <html>
  <head>
  <script type="text/javascript">
  var txt="";
  function message()
  {
  try
    {
    adddlert("Welcome guest!");
    }
  catch(err)
    {
    txt="There was an error on this page.\n\n";
    txt+="Error description: " + err.description + "\n\n";
    txt+="Click OK to continue.\n\n";
    alert(txt);
    }
  }
  </script></head>

  <body>
  <input type="button" value="View message" onclick="message()" />
  </body></html>
  ```

The Throw Statement

- The throw statement allows you to create an exception. If you use this statement together with the try...catch statement, you can control program flow and generate accurate error messages.

The Throw Statement

```
<html>
<body>
<script type="text/javascript">
var x=prompt("Enter a number between 0 and 10:","");
try
  {
  if(x>10)
    {
    throw "Err1";
    }
  else if(x<0)
    {
    throw "Err2";
    }
    }
catch(er)
  {
  if(er=="Err1")
    {
    alert("Error! The value is too high");
    }
  if(er=="Err2")
    {
    alert("Error! The value is too low");
    }
  }
</script>
</body>
</html>
```

JavaScript Functions

- A function will be executed by an event or by a call to the function.

- Functions can be defined both in the <head> and in the <body> section of a document. However, to assure that a function is read/loaded by the browser before it is called, it could be wise to put functions in the <head> section.

JavaScript Functions

JavaScript Function Example
Example

```html
<html>
<head>
<script type="text/javascript">
function displaymessage()
{
alert("Hello World!");
}
</script>
</head>

<body>
<form>
<input type="button" value="Click me!" onclick="displaymessage()" />
</form>
</body>
</html>
```

JavaScript Functions
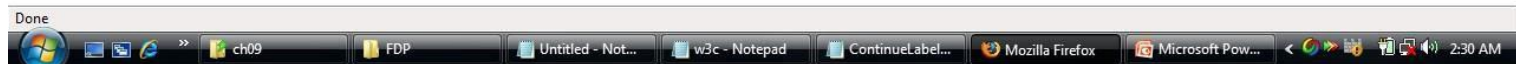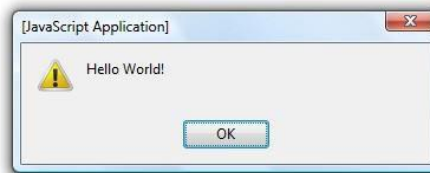
JavaScript Functions

The return Statement

- The return statement is used to specify the value that is returned from the function.

Example
```
<html>
<head>
<script type="text/javascript">
function product(a,b)
{
return a*b;
}
</script>
</head>

<body>
<script type="text/javascript">
document.write(product(4,3));
</script>
</body>
</html>
```

JavaScript Functions
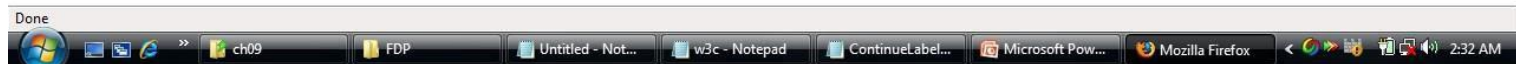
JavaScript Functions-another example

```
<head>
   <script type = "text/javascript">
     var input1 = window.prompt( "Enter first number", "0" );
     var input2 = window.prompt( "Enter second number", "0" );
     var input3 = window.prompt( "Enter third number", "0" );

     var value1 = parseFloat( input1 );
     var value2 = parseFloat( input2 );
     var value3 = parseFloat( input3 );

     var maxValue = maximum( value1, value2, value3 );

     document.writeln( "First number: " + value1 +

        "<br />Second number: " + value2 +
        "<br />Third number: " + value3 +
        "<br />Maximum is: " + maxValue );
```

JavaScript Functions

```
// maximum method definition (called from line 25)
    function maximum( x, y, z )
    {
      return Math.max( x, Math.max( y, z ) );
    }
  </script>

  </head>
  <body>
    <p>Click Refresh (or Reload) to run the script again</p>
  </body>
</html>
```
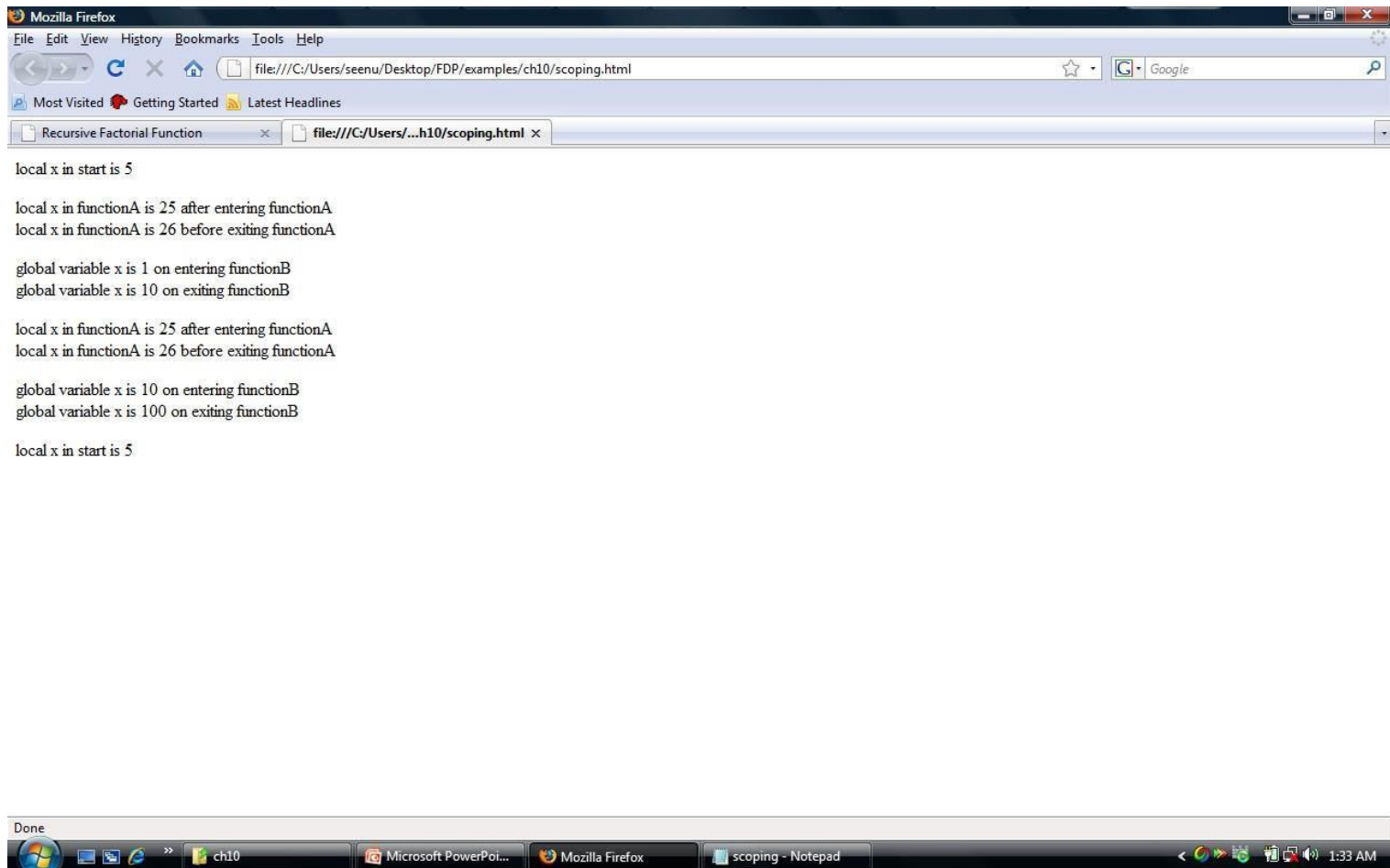
JavaScript Functions

JavaScript Functions

JavaScript Functions-scope

Array Object

An array is a special variable, which can hold more than one value, at a time.

```html
<html>
 <head>
     <script type = "text/javascript">
    function initializeArrays()
    {
      var n1 = new Array( 5 ); // allocate 5-element Array
      var n2 = new Array();      // allocate empty Array

          for ( var i = 0; i < n1.length; ++i )
        n1[ i ] = i;
          for ( i = 0; i < 5; ++i )
        n2[ i ] = i;

      outputArray( "Array n1 contains", n1 );
      outputArray( "Array n2 contains", n2 );
    }
```
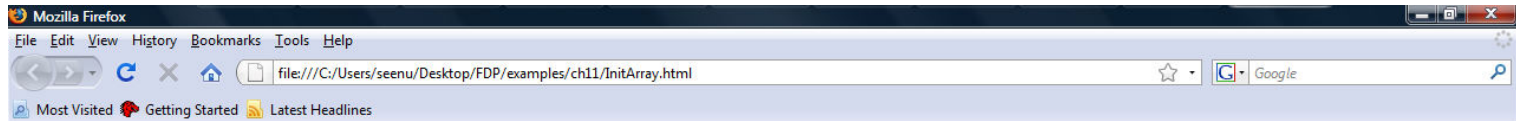
Array Object

```
function outputArray( header, theArray )
    {
        document.writeln( "<h2>" + header + "</h2>" );
        document.writeln( "<table border = \"1\" width =" +          "\"100%\">" );
            document.writeln( "<thead><th width = \"100\"" +
            "align = \"left\">Subscript</th>" +
            "<th align =\"left\">Value</th></thead><tbody>" );
            for ( var i = 0; i < theArray.length; i++ )
            document.writeln( "<tr><td>" + i + "</td><td>" +
                theArray[ i ] + "</td></tr>" );
        document.writeln( "</tbody></table>");
    }
</script>
    </head><body onload = "initializeArrays()"></body>
</html>
```

Array Object

Array Object

```
<html>
  <head>
    <title>Initializing an Array with a Declaration</title>
    <script type = "text/javascript">
      function start()
      {
        // Initializer listspecifies number of elements and
        // value for each element.
        var colors = new Array( "cyan", "magenta",
          "yellow", "black" );
            var integers1 = [ 2, 4, 6, 8 ];
        var integers2 = [ 2, , , 8 ];
         outputArray( "Array colors contains", colors );
            outputArray( "Array integers1 contains", integers1 );
        outputArray( "Array integers2 contains", integers2 );
      }
```

Array Object

```
function outputArray( header, theArray )
    {
        document.writeln( "<h2>" + header + "</h2>" );
        document.writeln( "<table border = \"1\"" +
            "width = \"100%\">" );
        document.writeln( "<thead><th width = \"100\" " +
            "align = \"left\">Subscript</th>" +
            "<th align = \"left\">Value</th></thead><tbody>" );
        for ( var i = 0; i < theArray.length; i++ )
            document.writeln( "<tr><td>" + i + "</td><td>" +
                theArray[ i ] + "</td></tr>" );
        document.writeln( "</tbody></table>");
    }
</script>


    </head><body onload = "start()"></body>
</html>
```

Array Object

Array Object

```
<html >
  <head>
    <title>Initializing MultidimensionalArrays</title>

    <script type = "text/javascript">
     function start()
      {
        var array1 = [ [ 1, 2, 3 ],    // first row
                  [ 4, 5, 6 ] ];  // second row
        var array2 = [ [ 1, 2 ],       // first row
                  [ 3 ],         // second row
                  [ 4, 5, 6 ] ];  // third row

        outputArray( "Values in array1 by row", array1 );
        outputArray( "Values in array2 by row", array2 );
      }
```

Array Object

```
function outputArray( header, theArray )
    {
        document.writeln( "<h2>" + header + "</h2><tt>" );

        for ( var i in theArray ) {

            for ( var j in theArray[ i ] )
                document.write( theArray[ i ][ j ] + " " );
            document.writeln( "<br />" );
        }

}
</script>

    </head><body onload = "start()"></body>
</html>
```

Array Object

Array Object

```html
<html>
  <head>
    <title>Sum the Elements of an Array</title>
    <script type = "text/javascript">
     function start()
      {
        var theArray = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ];
        var total1 = 0, total2 = 0;
        for ( var i = 0; i < theArray.length; i++ )
          total1 += theArray[ i ];
            document.writeln( "Total using subscripts: " + total1);
        for ( var element in theArray )
          total2 += theArray[ element ];
            document.writeln( "<br />Total using for...in: " +
          total2 );
      }
</script>   </head><body onload = "start()"></body></html>
```

Array Object

Array Object

```
<html>
  <head>
    <title>Passing Arrays and Individual Array
          Elements to Functions</title>
    <script type = "text/javascript">
      function start()
      { var a = [ 1, 2, 3, 4, 5 ];
        document.writeln( "<h2>Effects of passing entire " +
          "array by reference</h2>" );
        outputArray( "The values of the original array are: ", a );
        modifyArray( a ); // array a passed by reference
        outputArray( "The values of the modified array are: ", a );
        document.writeln( "<h2>Effects of passing array " +
       "element by value</h2>" +"a[3] before modifyElement: " + a[ 3 ] );
        modifyElement( a[ 3 ] );
        document.writeln("<br />a[3] aftermodifyElement: " + a[ 3 ] );
      }
```

Array Object

```
function outputArray( header, theArray )
    {
        document.writeln( header + theArray.join( " " ) + "<br />" );
    }
function modifyArray( theArray )
    {
        for ( var j in theArray )
            theArray[ j ] *= 2;
    }
function modifyElement( e )
    {
        e *= 2;
        document.writeln( "<br />value in modifyElement: " + e );
    }
</script>
</head><body onload = "start()"></body>
</html>
```

Array Object

Array Object

```html
<html >
  <head>
    <title>Sorting an Array with Array Method sort</title>

    <script type = "text/javascript">

      function start()

      {
        var a = [ 10, 1, 9, 2, 8, 3, 7, 4, 6, 5 ];

        document.writeln( "<h1>Sorting an Array</h1>" );
        outputArray( "Data items in original order: ", a );
        a.sort( compareIntegers ); // sort the array
        outputArray( "Data items in ascending order: ", a );
      }
```

Array Object

```
// outputs "header" followed by the contents of "theArray"

    function outputArray( header, theArray )

    {
      document.writeln( "<p>" + header +
        theArray.join( " " ) + "</p>" );

    }


    // comparison function for use with sort
    function compareIntegers( value1, value2)

    {
      return parseInt( value1 ) - parseInt( value2 );
    }
</script>


  </head><body onload = "start()"></body>
</html>
```

Array Object

String object

```
<html >
  <head>
    <title>Character Processing Methods</title>
    <script type = "text/javascript">
     var s = "ZEBRA";
      var s2 = "AbCdEfG";
      document.writeln( "<p>Character at index 0 in '" +
        s + "' is " + s.charAt( 0 ) );
      document.writeln( "<br />Character code at index 0 in '"
        + s + "' is " + s.charCodeAt( 0 ) + "</p>" );
      document.writeln( "<p>'" + String.fromCharCode( 87, 79, 82, 68 )+
        "' contains character codes 87, 79, 82 and 68</p>" )
      document.writeln( "<p>'" + s2 + "' inlowercase is '" +
        s2.toLowerCase() + "'" );
      document.writeln( "<br />'" + s2 + "' in uppercase is '"
        + s2.toUpperCase() + "'</p>" );
    </script>   </head><body></body></html>
```

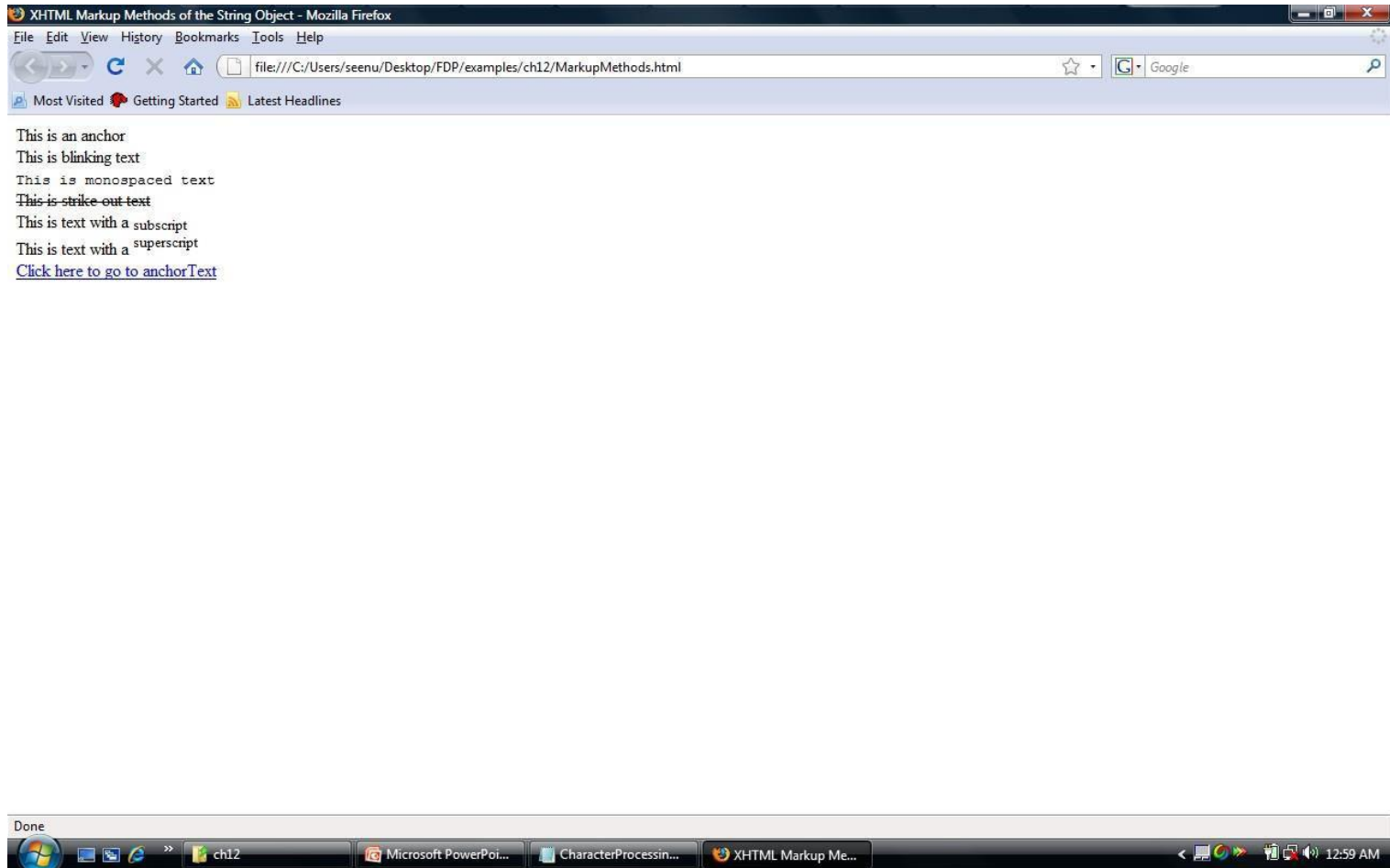# String object
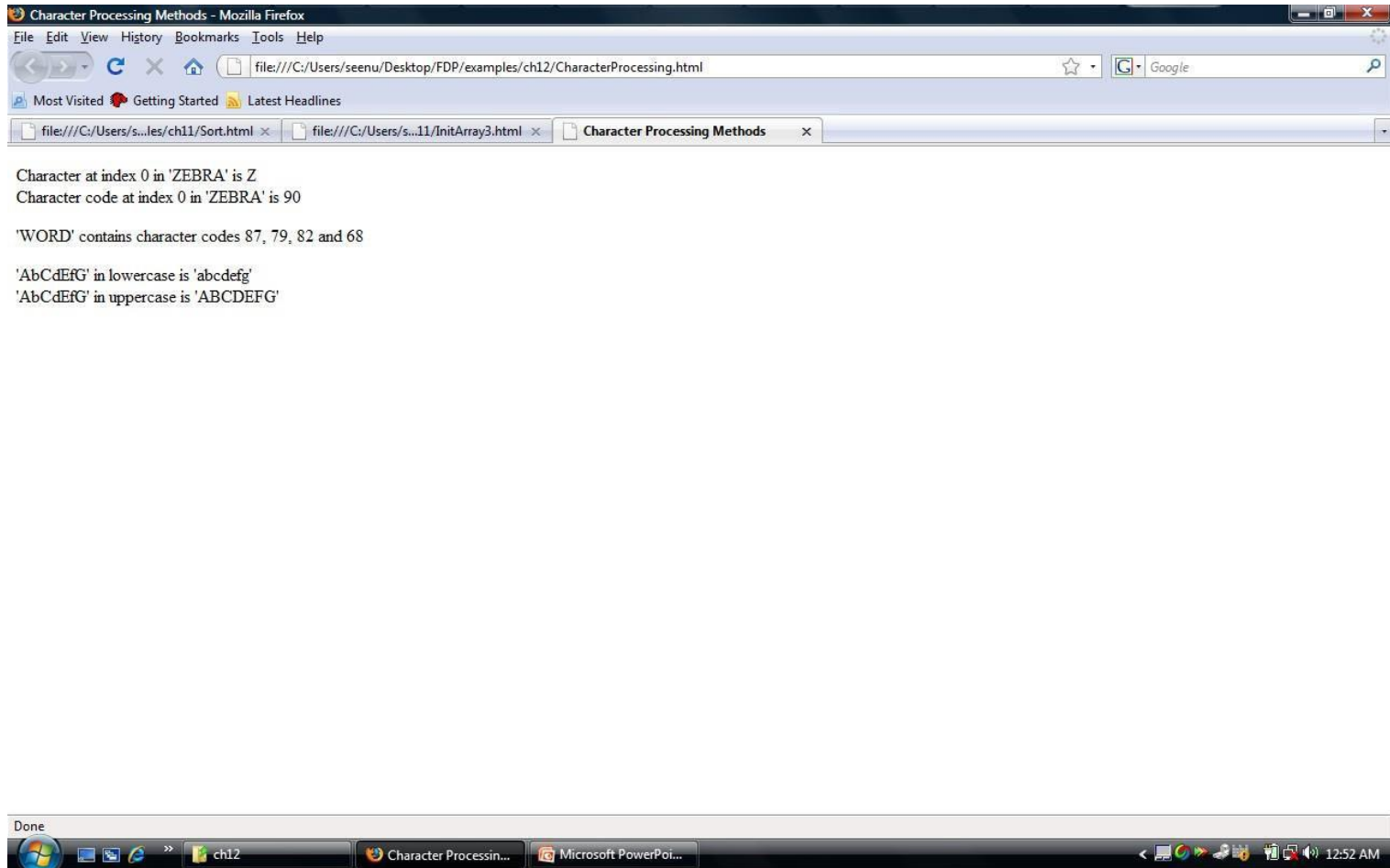
String object

```
<html >
  <head><title>XHTML Markup Methods of the String Object</title>
    <script type = "text/javascript">
        var anchorText = "This is an anchor",
        blinkText = "This is blinking text",  fixedText
        = "This is monospaced text",  linkText =
        "Click here to go to anchorText",  strikeText
        = "This is strike out text",
        subText = "subscript",    supText = "superscript";
    document.writeln( anchorText.anchor( "top" ) );
    document.writeln( "<br />" + blinkText.blink() );
    document.writeln( "<br />" + fixedText.fixed() );
    document.writeln( "<br />" + strikeText.strike() );
    document.writeln("<br />This is text with a " + subText.sub() );
    document.writeln( "<br />This is text with a " + supText.sup());
    document.writeln( "<br />" + linkText.link( "#top" ));
    </script> </head><body></body></html>
```

String object
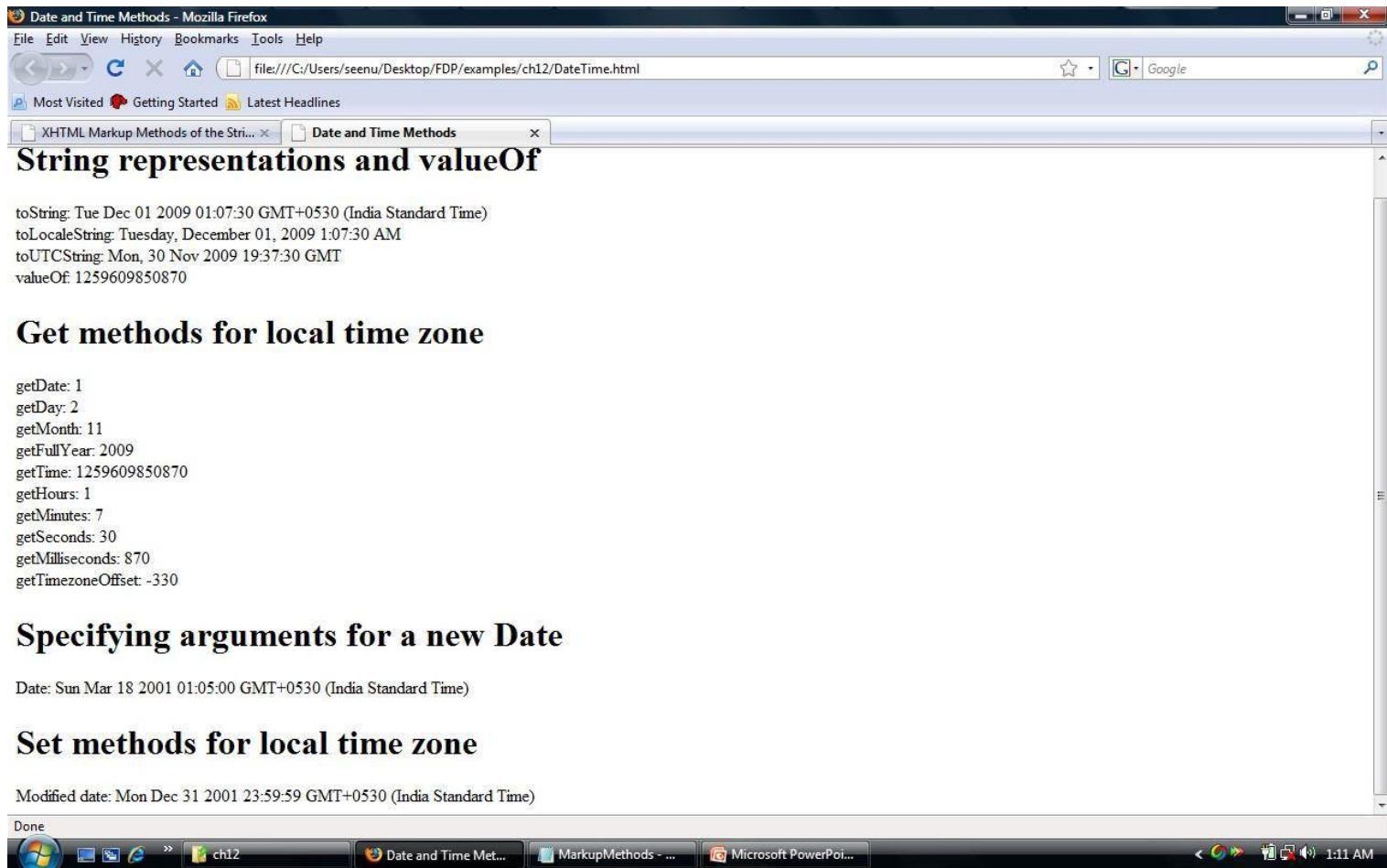
Date object

```
<html >
 <head><title>Date and Time Methods</title>
    <script type = "text/javascript">
      var current = new Date();
      document.writeln(
        "<h1>String representations and valueOf</h1>" );
      document.writeln( "toString: " + current.toString() +
        "<br />toLocaleString: " + current.toLocaleString() +
        "<br />toUTCString: " + current.toUTCString() +
        "<br />valueOf: " + current.valueOf());
      document.writeln("<h1>Get methods for local time zone</h1>" );
      document.writeln( "getDate: " + current.getDate() +
        "<br />getDay: " + current.getDay() +
        "<br />getMonth: " + current.getMonth() +
        "<br />getFullYear: " + current.getFullYear() +
        "<br />getTime: " + current.getTime() +
        "<br />getHours: " + current.getHours() +
```

Date object

Compare Two Dates

- The Date object is also used to compare two dates.

- The following example compares today's date with the 14th January 2010:

- ```
var myDate=new Date();
myDate.setFullYear(2010,0,14);
var today = new Date();

if (myDate>today)
  {
  alert("Today is before 14th January 2010");
  }
else
  {
  alert("Today is after 14th January 2010");
  }
```

# Document object

```
<html >
  <head>
    <title>Using Cookies</title>
    <script type = "text/javascript">
      var now = new Date(); // current date and time
      var hour = now.getHours(); // current hour (0-23)
      var name;
      if ( hour < 12 ) // determine whether it is morning
        document.write( "<h1>Good Morning, " );
      else
      { hour = hour - 12; // convert from 24 hour clock to PM time
        // determine whether it is afternoon or evening
        if ( hour < 6 )
          document.write( "<h1>Good Afternoon, " );
        else
          document.write( "<h1>Good Evening, " );
      }
```

Document object

```
// determine whether there is a cookie
    if ( document.cookie )
    {// convert escape characters in the cookie string to their
                    // english notation
      var myCookie = unescape( document.cookie );
      // split the cookie into tokens using = as delimiter
      var cookieTokens = myCookie.split( "=" );
      // set name to the part of the cookie that follows the = sign
      name = cookieTokens[ 1 ];
    }
    else
    { // if there was no cookie then ask the user to input a name
      name = window.prompt( "Please enter your name", "GalAnt" );
            // escape non-alphanumeric characters in the name string
                    // and add name to the cookie
      document.cookie = "name=" + escape( name );
    }
```

Document object

```
document.writeln(
        name + ", welcome to JavaScript programming! </h1>" );
      document.writeln( "<a href= \" JavaScript:wrongPerson() \" > " +
        "Click here if you are not " + name + "</a>" );
      // reset the document's cookie if wrong person
      function wrongPerson()
      {
        // reset the cookie
        document.cookie= "name=null;" +
          " expires=Thu, 01-Jan-95 00:00:01 GMT";
        // after removing the cookie reload the page to get a new name
        location.reload();
      }
</script>
  </head>
  <body><p>Click Refresh (or Reload) to run the script again</p>
  </body></html>
```