Cognizant Technology Solutions

Best Practices Trainer Guide

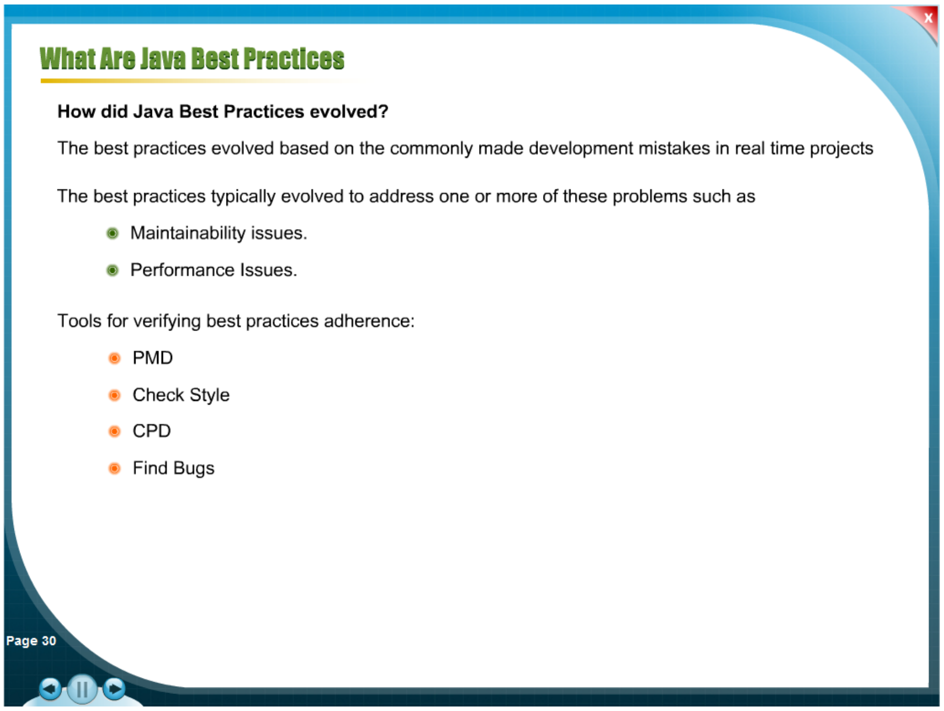
105110
4/5/2012

1. For the Trainers

This document is a guide for trainers to help them deliver the best practices session. This illustrates some sample best practices extracts which we follow in projects. The frequently made mistakes in projects have been transformed into best practices. It is always recommended to use tools like Findbugs & PMD to ensure that we develop application adhering to best practices. You will learn about PMD tool usage in the PMD session. This session focuses only on understanding the best practices.

The session has to parts to it, first part trainers should deliver the best practices. This will be followed by a game where associates identify the best practices violations.

2. Slide Details:



What Are Java Best Practices

How did Java Best Practices evolved?

The best practices evolved based on the commonly made development mistakes in real time projects

The best practices typically evolved to address one or more of these problems such as

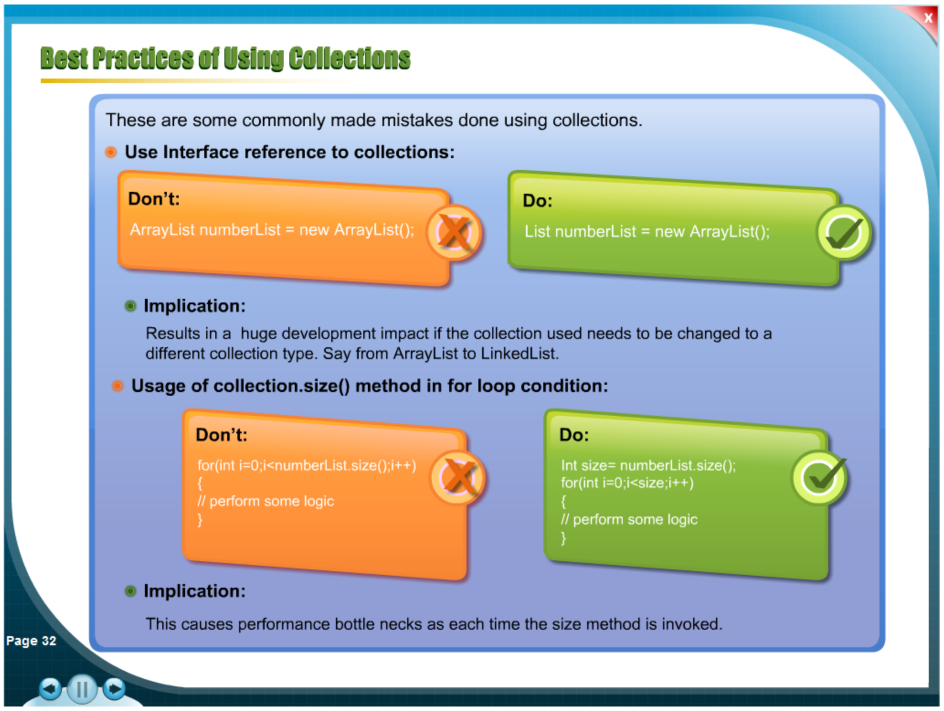
- Maintainability issues.
- Performance Issues.

Tools for verifying best practices adherence:

- PMD
- Check Style
- CPD
- Find Bugs

<Note>Trainers , please explain as below</Note>

The best practices evolved typically cause of the mistakes we have done in real time project. For example when developing application the developers used lost of string concatenation this resulted in performance issue. This was the ideation of the bets practices using String buffer rather than String concatenation using “+”. Similarly for all problems we make in real time projects evolved some best practices to avoid repealing the same mistakes again and again in the projects.



Best Practices of Using Collections

These are some commonly made mistakes done using collections.

- **Use Interface reference to collections:**

Don't:

```
ArrayList numberList = new ArrayList();
```

Do:

```
List numberList = new ArrayList();
```

- **Implication:**

Results in a huge development impact if the collection used needs to be changed to a different collection type. Say from ArrayList to LinkedList.

- **Usage of collection.size() method in for loop condition:**

Don't:

```
for(int i=0;i<numberList.size();i++)
{
// perform some logic
}
```

Do:

```
Int size= numberList.size();
for(int i=0;i<size;i++)
{
// perform some logic
}
```

- **Implication:**

This causes performance bottle necks as each time the size method is invoked.

Page 32

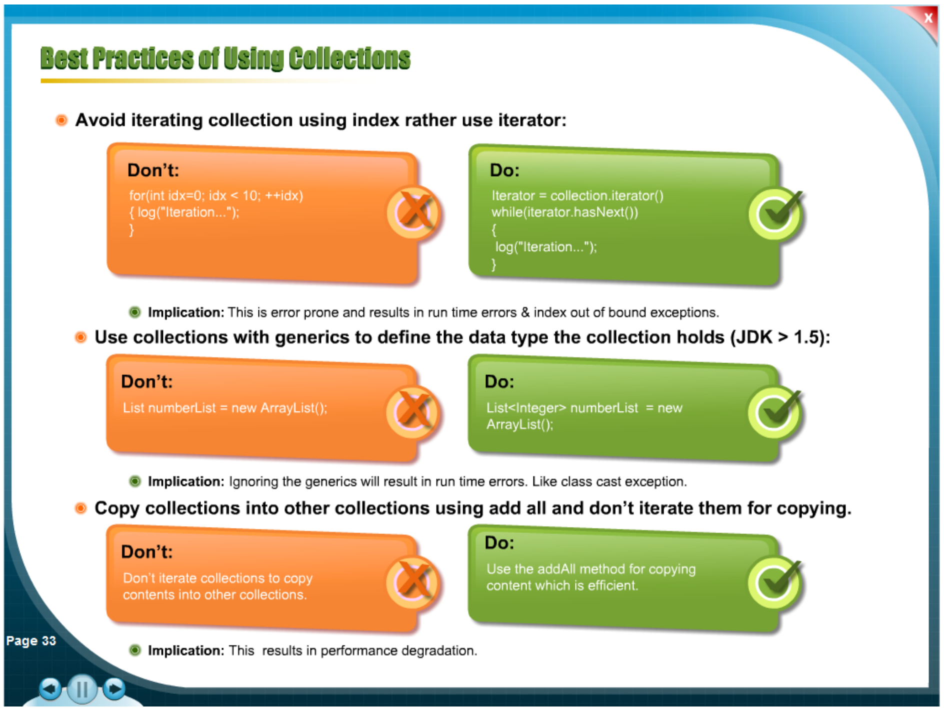
<NOTE> Trainers, use the notes give below to explain the content, please stress on the implication part, if needed use white board to explain too.</NOTE>

Overview of Collections

Collection is a group of objects. java.util package provides important types of collections. There are two fundamental types of collections they are Collection and Map. Collection types hold a group of objects, Eg. Lists and Sets where as Map types hold group of objects as key, value pairs Eg. HashMap and Hashtable.

Use Interface reference to collections: Use interface when declaring Collections. Lets take a example there is a method which accepts a ArrayList and 100 different clients are consuming this api. Lets say the array List is deprecated or we need to go for a linked List then change to this method will result in a change in all the 100 client a huge development revamp and testing effort. Assume that the method accepts a Interface Collection , in that case though the implementation changes from ArrayList to Linked list the method signature does not change and thus the change is done without changing any of the client code.

Usage of collection.size() method in for loop condition: Assume that the collection has 1000 entries and we are iterating through the list. Assume each size method call takes around .025 second (for the sake of dicssuion) then to complete iteration of 1000 records it would take 1000*0.25 = 25 seconds This will result in poor response time.



Best Practices of Using Collections

- Avoid iterating collection using index rather use iterator:

Don't:
```
for(int idx=0; idx < 10; ++idx)
{ log("Iteration...");
}
```

Do:
```
Iterator = collection.iterator()
while(iterator.hasNext())
{
 log("Iteration...");
}
```

- **Implication:** This is error prone and results in run time errors & index out of bound exceptions.

- Use collections with generics to define the data type the collection holds (JDK > 1.5):

Don't:
```
List numberList = new ArrayList();
```

Do:
```
List<Integer> numberList  = new
ArrayList();
```

- **Implication:** Ignoring the generics will result in run time errors. Like class cast exception.

- Copy collections into other collections using add all and don't iterate them for copying.

Don't:
Don't iterate collections to copy contents into other collections.

Do:
Use the addAll method for copying content which is efficient.

- **Implication:** This results in performance degradation.

Page 33

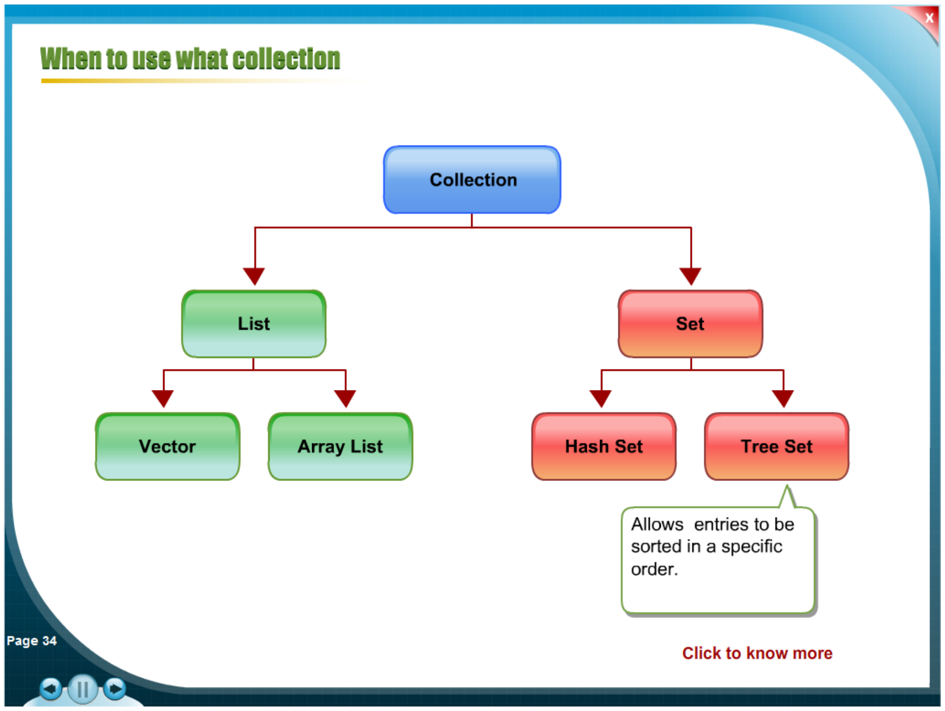
<NOTE> Trainers, use the notes give below to explain the content, please stress on the implication part, if needed use white board to explain too.</NOTE>

Avoid iterating collection using index rather use Iterator:

Using index is error prone many instance we end up in array index out of bound exception. Also assume that the list is manipulated inside the for loop this can result in index changing and some functional error too.

Use collections with generics to define the data type the collection holds (JDK > 1.5):

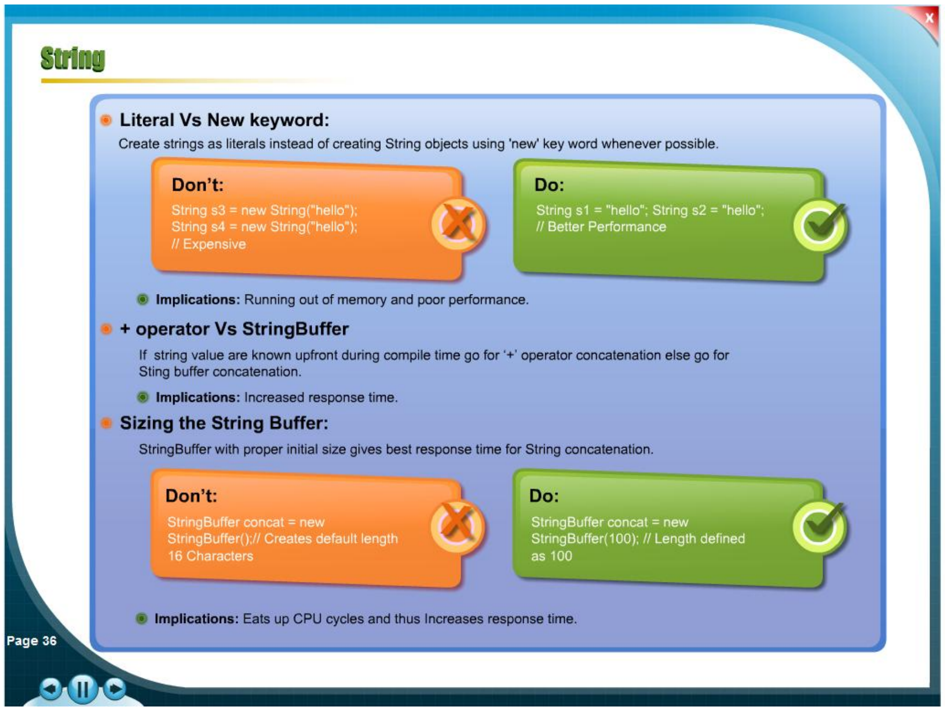
Adding generics will be easy to maintain and understand the code. The consumers looking at the api will be easily able to know what needs to be retrieved or added in the list. For example lets say that in a method a list of Integer objects is added. And the client retrieves this and typecast it to a String you will get a cast exception at run time. Using generics if there is any data type mismatch the error will be shown as compilation errors. So the errors can be found upfront before deployment.


When to use what collection
Collection
List
Set
Vector
Array List
Hash Set
Tree Set
Allows entries to be sorted in a specific order.
Page 34
Click to know more

<NOTE> Explain each collection types and as mentioned below , please provide a overview on thread safe before you explain things**<NOTE>**

So in a nut shell If you want list to hold unique values go for Set, if it needs to be sorted go for Tree Set else Hash Set.

If duplicate values can be held go for List. In case the access of the list entries should be thread safe go for vector, but it is slow when compared to ArrayList.



String

- **Literal Vs New keyword:**
  Create strings as literals instead of creating String objects using 'new' key word whenever possible.

  Don't:
  ```
  String s3 = new String("hello");
  String s4 = new String("hello");
  // Expensive
  ```
  Do:
  ```
  String s1 = "hello"; String s2 = "hello";
  // Better Performance
  ```
  - **Implications:** Running out of memory and poor performance.
- **+ operator Vs StringBuffer**
  If string value are known upfront during compile time go for '+' operator concatenation else go for Sting buffer concatenation.
  - **Implications:** Increased response time.
- **Sizing the String Buffer:**
  StringBuffer with proper initial size gives best response time for String concatenation.

  Don't:
  ```
  StringBuffer concat = new
  StringBuffer();// Creates default length
  16 Characters
  ```
  Do:
  ```
  StringBuffer concat = new
  StringBuffer(100); // Length defined
  as 100
  ```
  - **Implications:** Eats up CPU cycles and thus Increases response time.

Page 36

<NOTE> Please explain as mentioned below,

Overview of String and StringBuffer

Immutable objects cannot be modified once they are created. Mutable objects can be modified after their creation. String objects are immutable where as StringBuffer objects are mutable.

You need to carefully choose between these two objects depending on the situation for better performance.

Create String as Literals:

Because the content is same s1 and s2 refer to the same object where as s3 and s4 do not refer to the same object. The 'new' key word creates new objects for s3 and s4 which is expensive. The more objects we create the more the memory utilized which can make the application slow or result in out of memory error. So use String literals wisely and avoid unnecessary creation of String objects using new().

+ operator Vs String Buffer:

String object is resolved at compile time where as StringBuffer object is resolved at run time. Run time resolution takes place when the value of the string is not known in advance where as compile time resolution happens when the value of the string is known in advance .

String buffer reduces response time and reduces unnecessary object creations.

You can write this in the board and explain

```
private  void sayHello(String s1, String s) {

String concat = s1+s;

}// The string S1 and S are values passed In runtime fetched from DB. This can be written as
```

```
private void sayHello(String s1, String s) {

StringBuffer concat = new StringBuffer().append(s1).append(s);

}
```

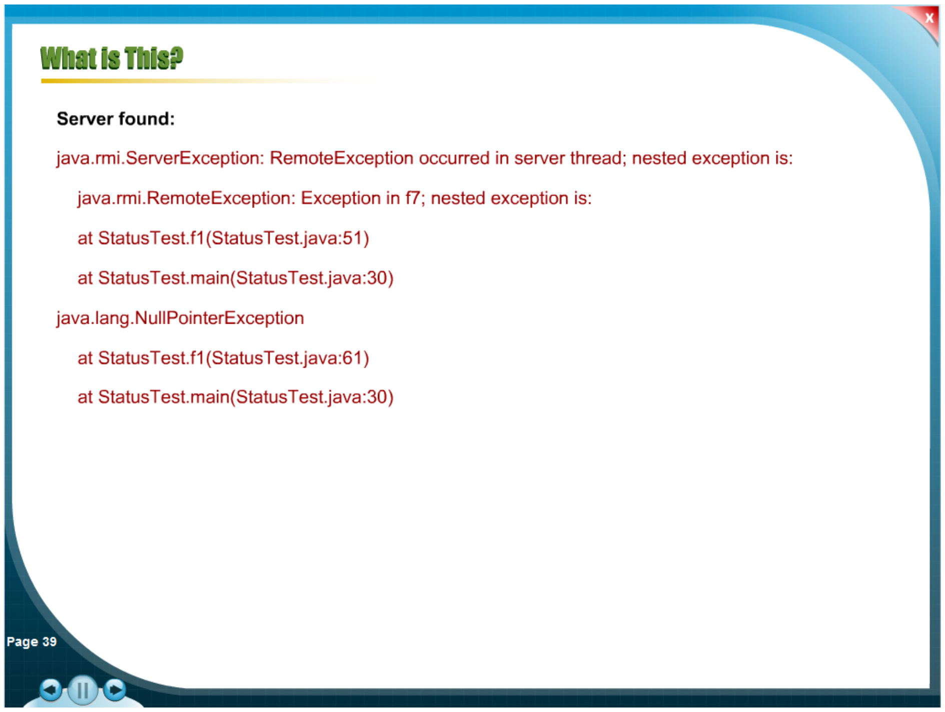
Sizing the String buffer:

If size not specified in String buffer the string buffer needs to be expanded every time the String grows resulting in wastage of CPU cycles and reducing the response time. So it is always advisable to specify the buffer size if you know the approximate String length say length of column from DB etc.

StringBuffer maintains a character array internally.When you create StringBuffer with default constructor StringBuffer() without setting initial length, then the StringBuffer is initialized with 16 characters. The default capacity is 16 characters. When the StringBuffer reaches its maximum capacity, it will increase its size by twice the size plus 2 (2*old size +2).

If you use default size, initially and go on adding characters, then it increases its size by 34(2*16 +2) after it adds 16th character and it increases its size by 70(2*34+2) after it adds 34th character. Whenever it reaches its maximum capacity it has to create a new character array and recopy old and new characters. It is obviously expensive. So it is always good to initialize with proper size that gives very good performance.

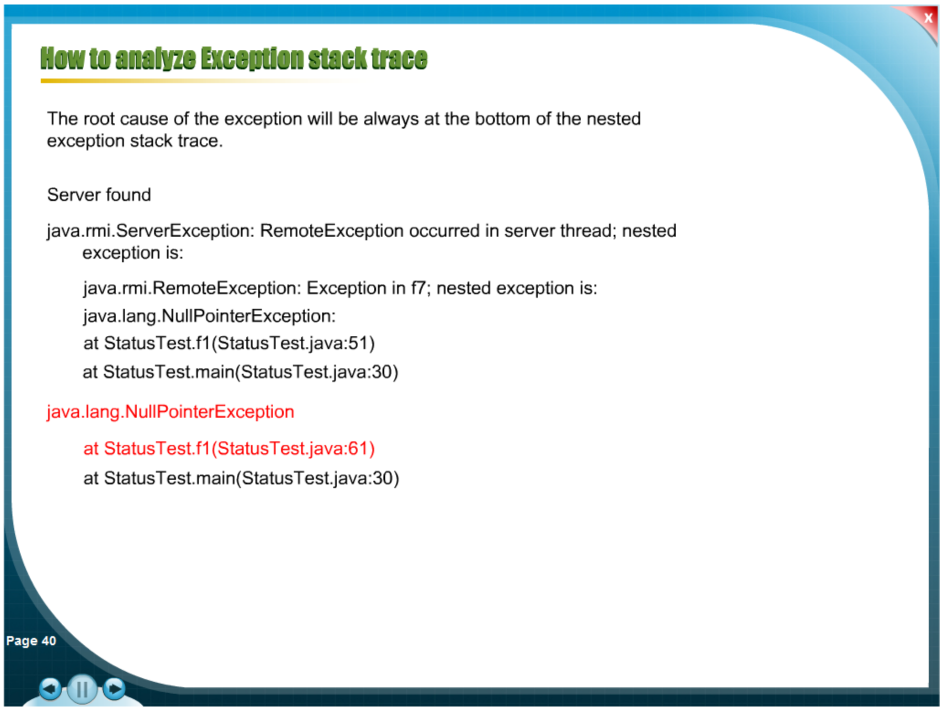
Trainers can quickly recap the session content delivered till now.


What is This?
Page 39

Server found:

```
java.rmi.ServerException: RemoteException occurred in server thread; nested exception is:
    java.rmi.RemoteException: Exception in f7; nested exception is:
    at StatusTest.f1(StatusTest.java:51)
    at StatusTest.main(StatusTest.java:30)
java.lang.NullPointerException
    at StatusTest.f1(StatusTest.java:61)
    at StatusTest.main(StatusTest.java:30)
```

<NOTE> Please ask this question to the audience and await a response</NOTE>



How to analyze Exception stack trace

The root cause of the exception will be always at the bottom of the nested exception stack trace.

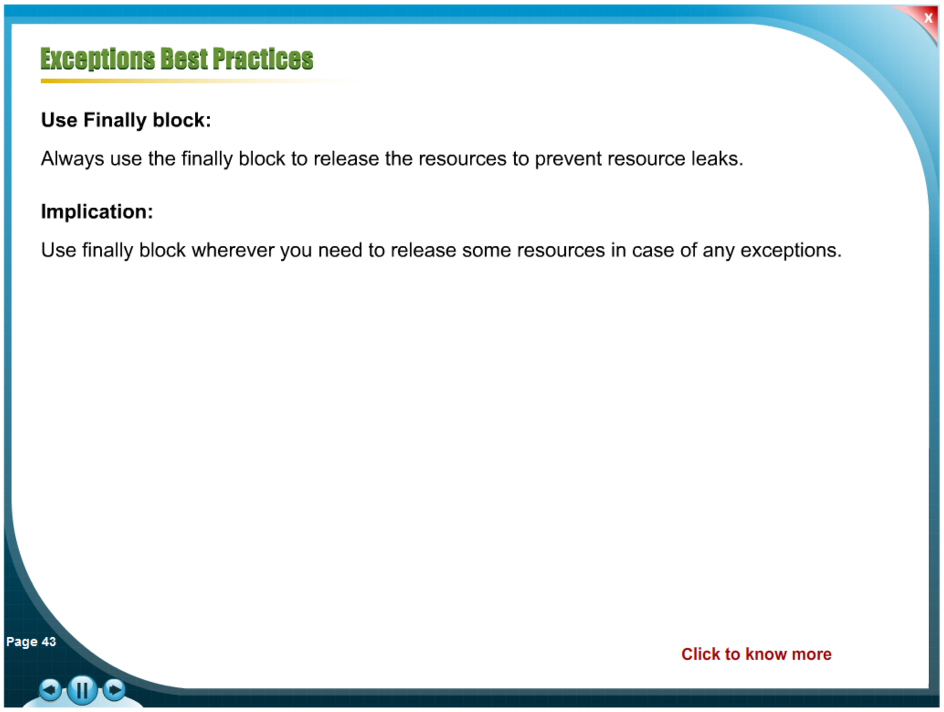
Server found

```
java.rmi.ServerException: RemoteException occurred in server thread; nested
      exception is:
      java.rmi.RemoteException: Exception in f7; nested exception is:
      java.lang.NullPointerException:
      at StatusTest.f1(StatusTest.java:51)
      at StatusTest.main(StatusTest.java:30)
java.lang.NullPointerException
      at StatusTest.f1(StatusTest.java:61)
      at StatusTest.main(StatusTest.java:30)
```

Page 40

<NOTE> Explain as mentioned below</NOTE>

The root exception is always at the bottom so read the exception from the bottom the exception states the root cause and the line number. In the example it states the message as null pointer exception in StatusTest.java line # 61. this is the line which needs to be fixed for the null issue.



Exceptions Best Practices

Use Finally block:

Always use the finally block to release the resources to prevent resource leaks.

Implication:

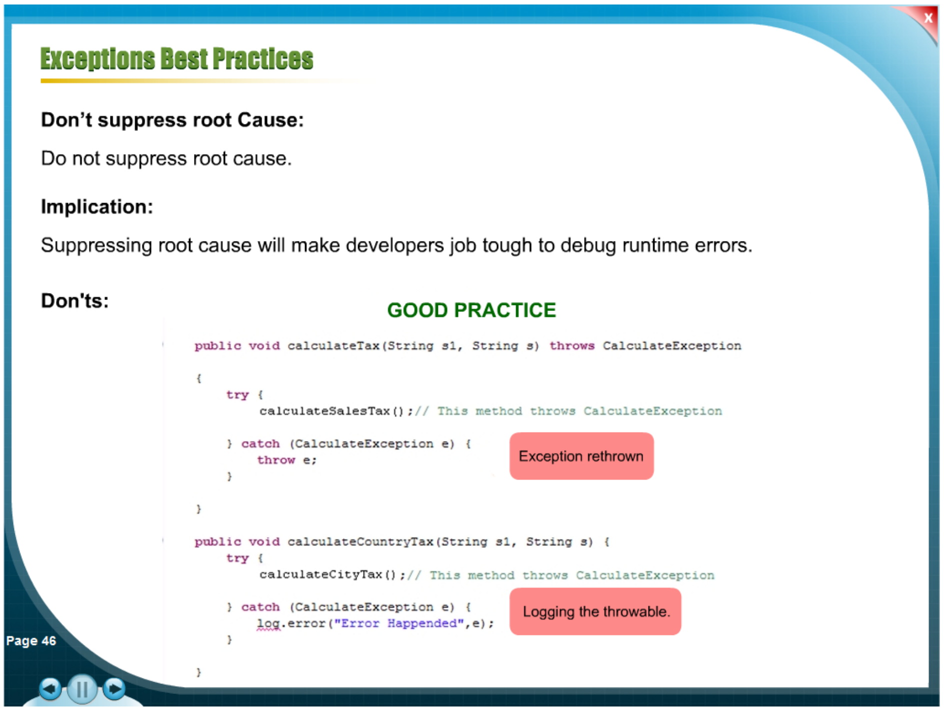
Use finally block wherever you need to release some resources in case of any exceptions.

Page 43

Click to know more

<NOTE> Explain as mentioned below</NOTE>

The bad sample code illustrated resulted in a application crash in one of the projects in cognizant resulted in a huge loss for our customers due to the application down time. This is how it happened. In a reporting functionality due to some data error there was a exception raised. Every exception raised did not release the DB connection that the application opened. As the number of users using it peaked up the number of available DB connections went down and finally there was no DB connections available for the application to consume resulting in system crash. This happened in one big project account. A simple mistake proved to be so costly. So always remember to free resource in the finally block as mentioned in the good practice sample code.



Exceptions Best Practices

Don't suppress root Cause:

Do not suppress root cause.

Implication:

Suppressing root cause will make developers job tough to debug runtime errors.

Don'ts:

GOOD PRACTICE

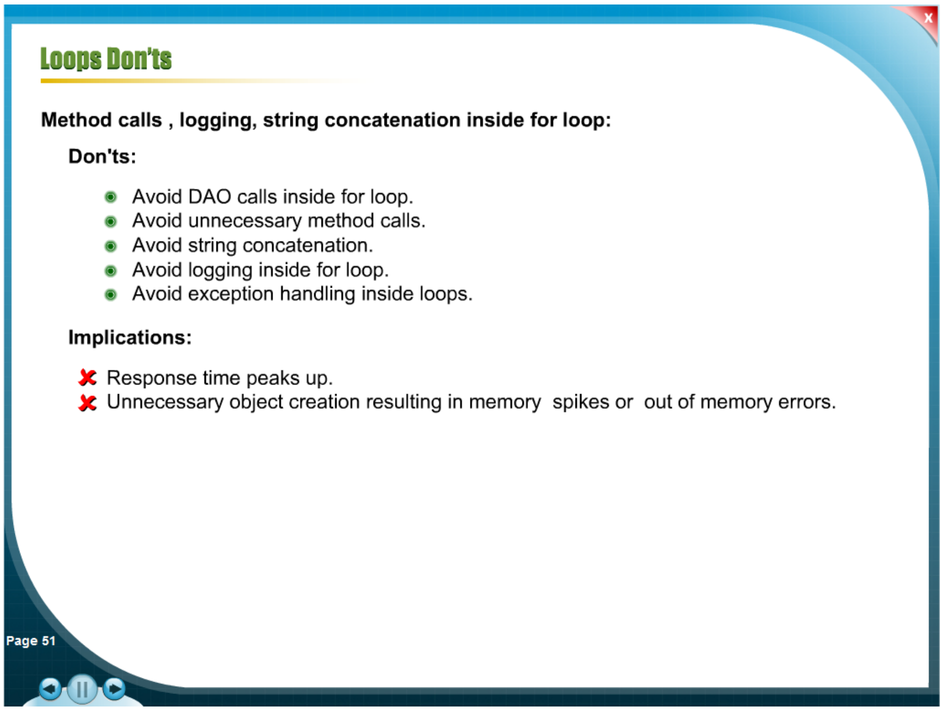
```
public void calculateTax(String s1, String s) throws CalculateException

{
    try {
        calculateSalesTax();// This method throws CalculateException

    } catch (CalculateException e) {
        throw e;
    }

}

public void calculateCountryTax(String s1, String s) {
    try {
        calculateCityTax();// This method throws CalculateException

    } catch (CalculateException e) {
        log.error("Error Happended",e);
    }

}
```

Exception rethrown

Logging the throwable.

Illustrate this with a real time story given below.

This happened again in a real time project, The project cost was 3 Million dollar and 30 developers developed it in 1.5 years time. One fine day at production user reported a major issue SLA was 24 hours. We started going through the log were unable to find the bug, we almost spend 12 hours in vain. We went back to the client saying it is non reproducible and let us know when the issue happens again. The client was not happy with our response. The same error happened a week after but this time the customer gave us all the proof the screen shot , the data etc. We got exclusive permission to process the data at production. We switched it on in debug mode and started debugging. As the code got executed we noticed a null exception being raised in a method, due to a mandatory master data not available. Instead of this being handled and logged, this was suppressed. This small null issue was the root cause of the problem for 2 weeks, if we have had handled the exception properly we could have nailed it down in first day. We could have avoided the bad encounter with the customer.



Loops Don'ts

Method calls , logging, string concatenation inside for loop:

Don'ts:

- Avoid DAO calls inside for loop.
- Avoid unnecessary method calls.
- Avoid string concatenation.
- Avoid logging inside for loop.
- Avoid exception handling inside loops.

Implications:

- ✗ Response time peaks up.
- ✗ Unnecessary object creation resulting in memory spikes or out of memory errors.

<NOTE> Trainers please explain as mentioned below</NOTE>

1.DAO calls inside loops causes huge performance issues, advisable approach is to ensure that the DAO calls can be clubbed in one call using IN parameter.

2. If a CPU intensive method can be invoked once instead of placing it inside the for loop, go for it.

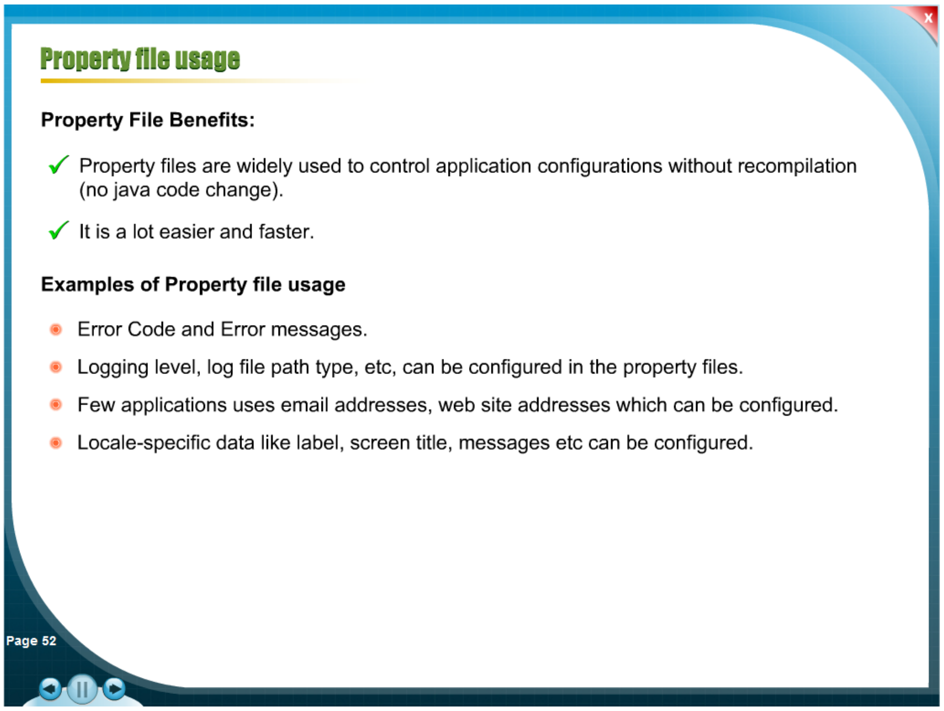
3. Avoid string concatenation and logging inside for loop this will peak up the number of objects, creates memory spikes. Resulting in GC running constantly. Resulting in CPU utilization peaking and downgrading the performance of application.

4. Avoid exception handling inside loops assume any exception happens and in catch block the exception is suppressed. The loop will continue and we will end up getting the same exception again and again resulting in a huge performance degradation.

A quick real time happening: At Cognizant there was one project delivered to client. The project was developed by 70 people in one year period time.

The application was delivered to production and was rolled back cause of performance issue. This got into serious of escalation and we lost customers confidence.

On trouble shooting we noticed that there was a small method developed by a developer which had a logging inside for loop. The loop ran every time the user logged in as the number of users increased the logging increased resulting in performance degradation and application crash. The developers was forced to fix this problem overnight, the entire team was asked to stay to smoke test the application and delivery released. It was so stressful for the entire team.



Property file usage

Property File Benefits:

- ✓ Property files are widely used to control application configurations without recompilation (no java code change).
- ✓ It is a lot easier and faster.

Examples of Property file usage

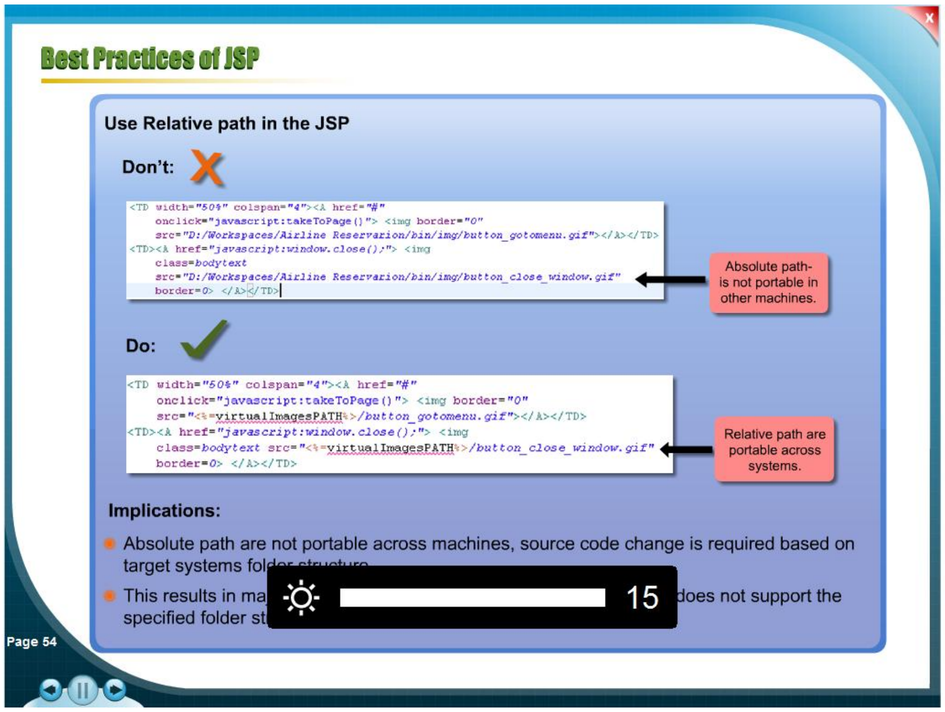
- Error Code and Error messages.
- Logging level, log file path type, etc, can be configured in the property files.
- Few applications uses email addresses, web site addresses which can be configured.
- Locale-specific data like label, screen title, messages etc can be configured.

Page 52

<NOTE> Explain as below **</NOTE>**

This allows for a non-developer (such as a power-user, system admin, or operations person) to make the change. using the properties file in a jar file does not require the source code to make the change. As such: it can be done on the system at hand rather then needing to be done on a development system Tech support can walk someone through it if needed It better facilitates emergency changes that may only be temporary and as such saves you from a 3am page waking you up to recompile some code for a simple config change. Sometimes you don't want to make a change, but simply want to see what something is set to. Again, it is much easier to unjar a properties file and read it than to decompile a class or tracking down the source code.

Trainers can quickly recap the session content delivered till now.



Best Practices of JSP

Use Relative path in the JSP

Don't:

```
<TD width="50%" colspan="4"><A href="#"
    onclick="javascript:takeToPage()"> <img border="0"
    src="D:/Workspaces/Airline Reservarion/bin/img/button_gotomenu.gif"></A></TD>
<TD><A href="javascript:window.close();"> <img
    class=bodytext
    src="D:/Workspaces/Airline Reservarion/bin/img/button_close_window.gif"
    border=0> </A></TD>
```

Absolute path- is not portable in other machines.

Do:

```
<TD width="50%" colspan="4"><A href="#"
    onclick="javascript:takeToPage()"> <img border="0"
    src="<%=virtualImagesPATH%>/button_gotomenu.gif"></A></TD>
<TD><A href="javascript:window.close();"> <img
    class=bodytext src="<%=virtualImagesPATH%>/button_close_window.gif"
    border=0> </A></TD>
```

Relative path are portable across systems.

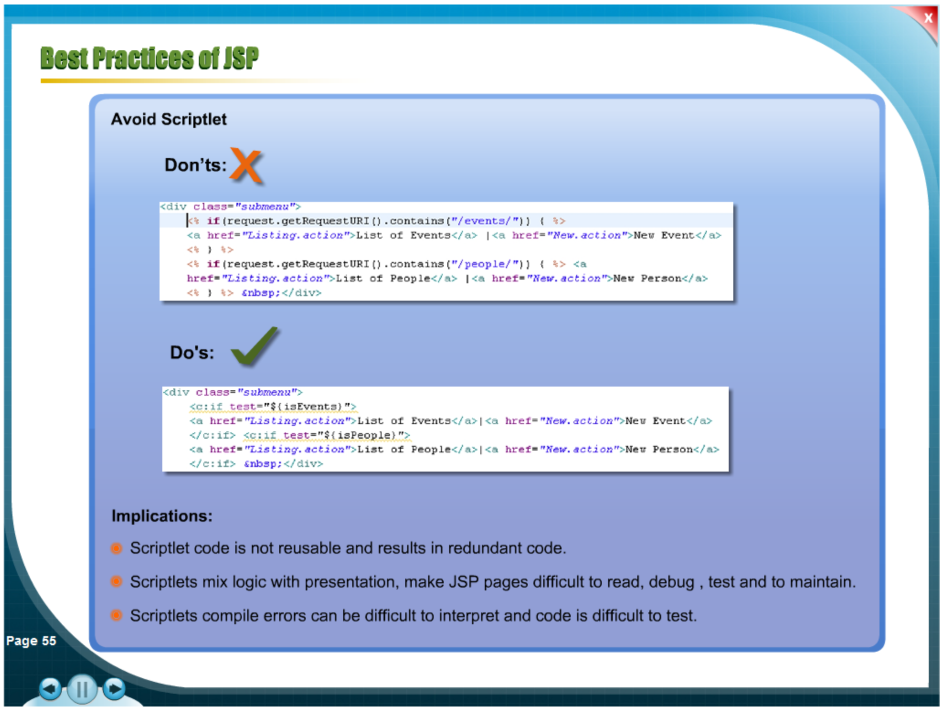
Implications:

- Absolute path are not portable across machines, source code change is required based on target systems fol
- This results in ma does not support the specified folder st

<NOTE> Please explain as below<NOTE>

If relative path is used then Source Code modification is not required if project is deployed to any other machine.

This can result in a huge development effort to tweak the code test the same. So avoid using absolute path.



Best Practices of JSP

Avoid Scriptlet

Don'ts: ✗

```
<div class="submenu">
    <% if(request.getRequestURI().contains("/events/")) { %>
    <a href="Listing.action">List of Events</a> |<a href="New.action">New Event</a>
    <% } %>
    <% if(request.getRequestURI().contains("/people/")) { %> <a
    href="Listing.action">List of People</a> |<a href="New.action">New Person</a>
    <% } %> &nbsp;</div>
```

Do's: ✓

```
<div class="submenu">
    <c:if test="${isEvents}">
    <a href="Listing.action">List of Events</a>|<a href="New.action">New Event</a>
    </c:if> <c:if test="${isPeople}">
    <a href="Listing.action">List of People</a>|<a href="New.action">New Person</a>
    </c:if> &nbsp;</div>
```

Implications:

- Scriptlet code is not reusable and results in redundant code.
- Scriptlets mix logic with presentation, make JSP pages difficult to read, debug , test and to maintain.
- Scriptlets compile errors can be difficult to interpret and code is difficult to test.

Page 55

Scriptlet code is not reusable. If the same logic is needed elsewhere, it must be either included (decreasing readability) or copied and pasted into the new context. Custom tags can be reused by reference.

Scriptlets encourage copy/paste coding--Because scriptlet code appears in only one place, it is often copied to a new context. When the scriptlet needs to be modified, usually all the copies need updating. Finding all copies of the scriptlet and updating them is an error-prone maintenance headache. With time, the copies tend to diverge, making it difficult to determine which scriptlets are copies of others, further frustrating maintenance. Custom tags centralize code in one place. When a tag handler class changes, the tag's behavior changes everywhere it is used.

Scriptlets mix logic with presentation--Scriptlets are islands of program code in a sea of presentation code. Changing either requires some understanding of what the other is doing to avoid breaking the relationship between the two. Scriptlets can easily confuse the intent of a JSP page by expressing program logic within the presentation. Custom tags encapsulate program logic so that JSP pages can focus on presentation.

Scriptlets break developer role separation--Because scriptlets mingle programming and Web content, Web page designers need to know either how to program or which parts of their pages to avoid modifying. Poorly implemented scriptlets can have subtle dependencies on the surrounding template data. Consider, for example, the following line of code: <% out.println("<a \"href=\"" + url + "\">" + text); %> </a> It would be very easy to change this line in a way that breaks the page, especially for someone who does not understand what the line is doing.

Custom tags help the separation of developer roles, because programmers create the tags, and page authors use them.

Scriptlets make JSP pages difficult to read and to maintain--JSP pages with scriptlets mix structured tags with JSP page delimiters and Java language code. The Java language code in scriptlets often uses "implicit" objects, which are not declared anywhere except in the JavaServer Pages specification. Also, even consistent indentation does not help readability much for nontrivial pages. JSP pages with custom tags are composed of tags and character data, which is much easier to read. JSP pages that use XML syntax can be validated as well.

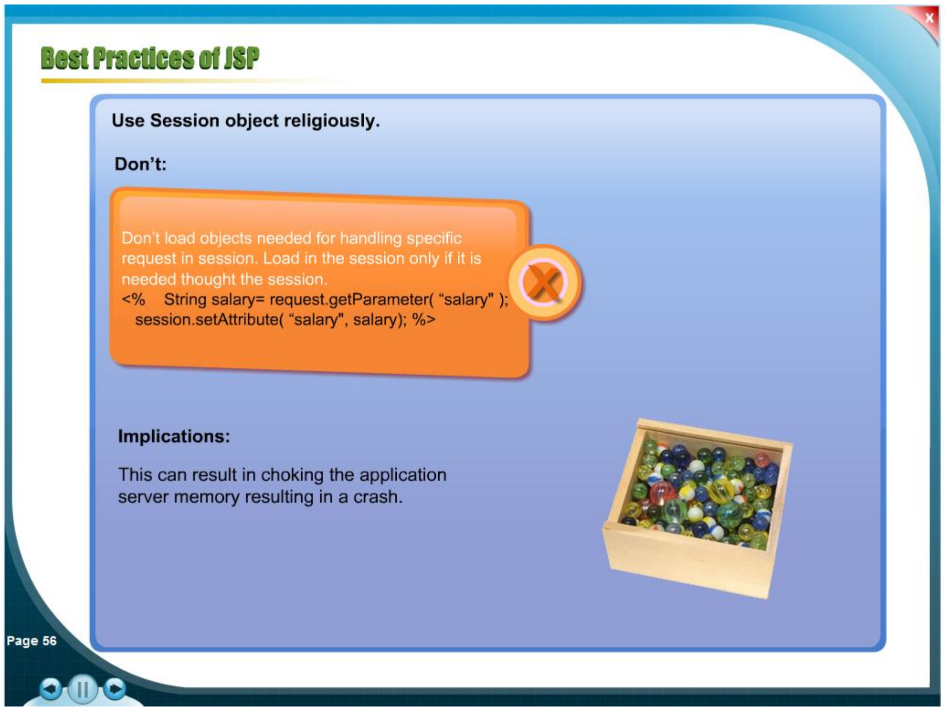
Scriptlet compile errors can be difficult to interpret--Many JSP page compilers do a poor job of translating line numbers between the source page and the generated servlet. Even those that emit error messages often depend on invisible context, such as implicit objects or surrounding template data. With poor error reporting, a missed semicolon can cost hours of development time. Erroneous code in custom tags will not compile either, but all of the context for determining the problem is present in the custom tag code, and the line numbers do not need translation.

Scriptlet code is difficult to test--Unit testing of scriptlet code is virtually impossible. Because scriptlets are embedded in JSP pages, the only way to execute them is to execute the page and test the results. A custom tag can be unit tested, and errors can be isolated to either the tag or the page in which it is used.

Expressions sometimes also suffer from these problems, but they are somewhat less problematic than scriptlets because they tend to be small.

Custom tags maintain separation between developer roles. They encourage reuse of logic and state within a single source file. They also improve source code readability, and improve both testability and error reporting.

Some projects have Web page authors but few or no programmers. Page authors with limited programming skill can use scriptlets effectively if they use scriptlets for display logic only. Business logic should never be implemented in scriptlets.



Best Practices of JSP

Use Session object religiously.

Don't:

Don't load objects needed for handling specific request in session. Load in the session only if it is needed thought the session.

```
<%   String salary= request.getParameter( "salary" );
  session.setAttribute( "salary", salary); %>
```

Implications:

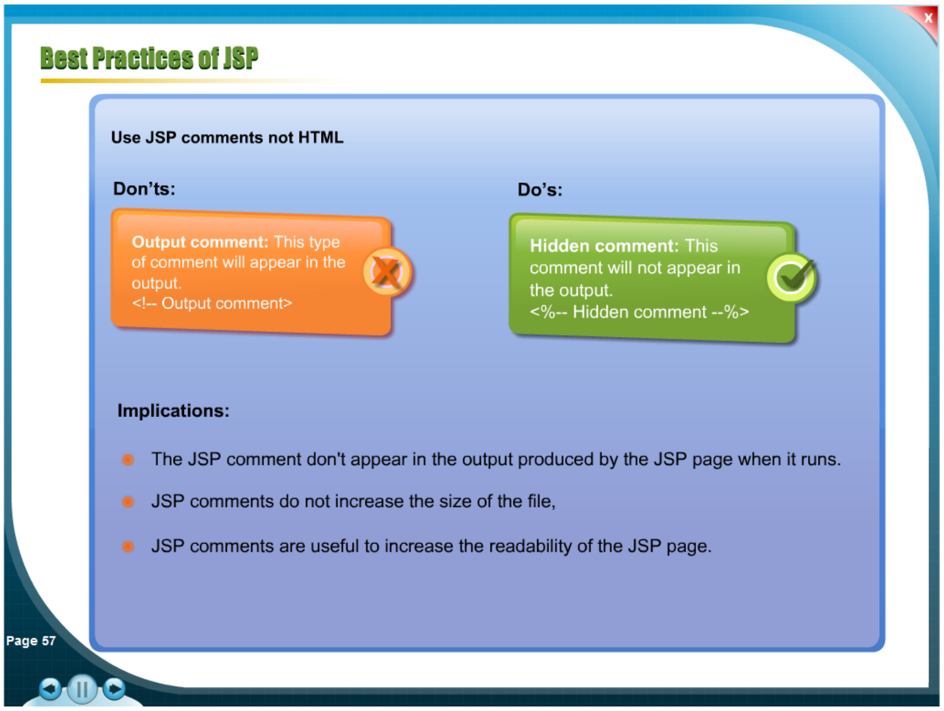
This can result in choking the application server memory resulting in a crash.

<NOTE> Please explain as below<NOTE>

Use session object carefully, Developers tend to put many values is session which could have been processed in request scope.

The memory is like a empty box it becomes full as you load more and more objects and finally when it is full it overflows which is the out of memory error.

In one of the projects in Cognizant the application crashed due to a small value object loaded in memory. What happened is the developer developed in such a way that when user logs into the system the login credential object was loaded in the session, this could have been well done in request scope. During production day one 3000 users logged in resulting in 3000 session objects created. The application crashed within few minutes after inauguration. On analyzing we found out the object size was 500Kb and the total memory peaked to3000*500= 1.5 GB resulting in the crash. This was a major setback for us and we had to roll back the application and the roll out was cancelled. Bad day. So lets be very careful when loading session objects, don't load unnecessary objects.



Best Practices of JSP

Use JSP comments not HTML

Don'ts:

Output comment: This type of comment will appear in the output.
<!-- Output comment>

Do's:

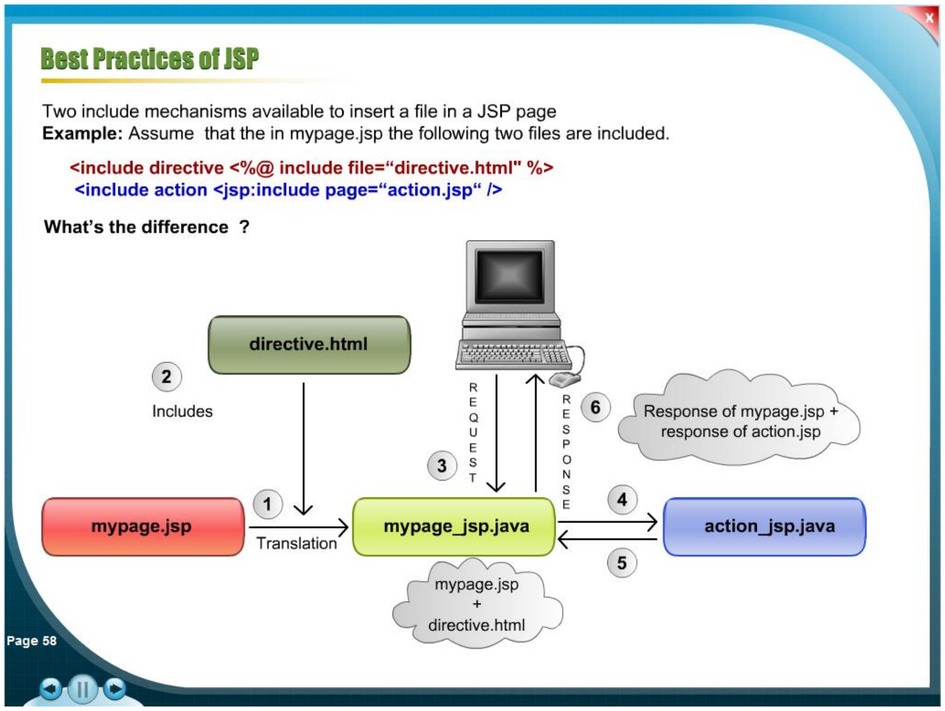
Hidden comment: This comment will not appear in the output.
<%-- Hidden comment --%>

Implications:

- The JSP comment don't appear in the output produced by the JSP page when it runs.
- JSP comments do not increase the size of the file,
- JSP comments are useful to increase the readability of the JSP page.

Page 57

In a jsp we should always try to use jsp- style comments unless you want the comments to appear in the HTML. Jsp comments are converted by the jsp engine into java comments in the source code of the servlet that implements the Jsp page. The jsp comment don't appear in the output produced by the jsp page when it runs. Jsp comments do not increase the size of the file, jsp page are useful to increase the readability of the jsp page.



Best Practices of JSP

Two include mechanisms available to insert a file in a JSP page
Example: Assume that the in mypage.jsp the following two files are included.

<include directive <%@ include file="directive.html" %>
<include action <jsp:include page="action.jsp" />

What's the difference ?

There are two include mechanisms available to insert a file in a JSP page. They are

1. include directive <%@ include file="child.jsp" %>

2. include action <jsp:include page="child.jsp" flush="true" />

The include directive includes the content of the file during the translation phase where as include action includes the content of the file during execution/request processing phase.

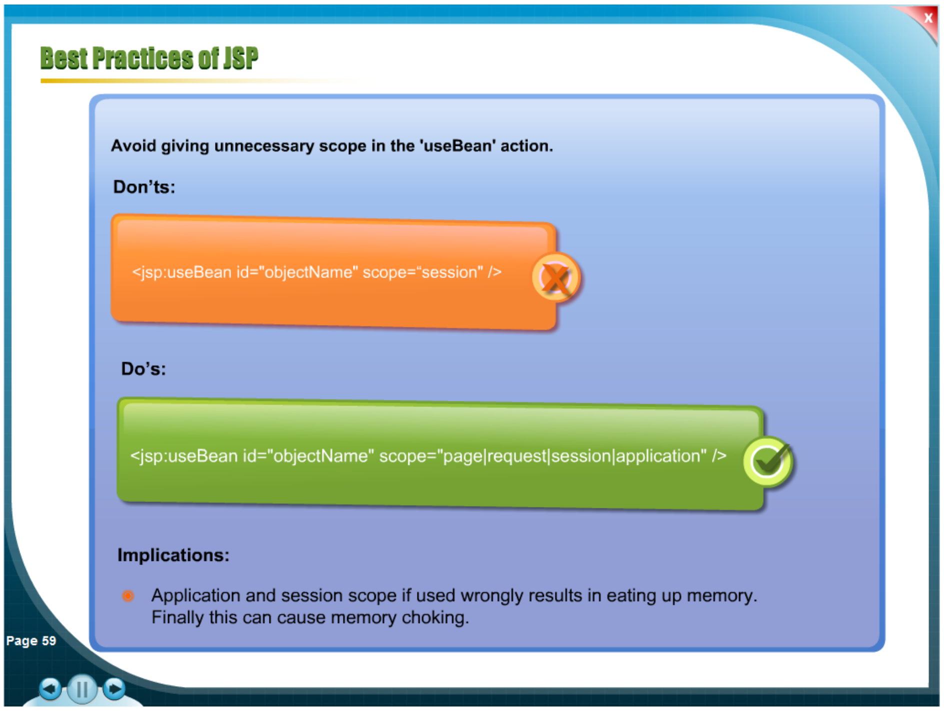
For include directive, JSP Engine adds the content of the inserted page at translation phase, so it does not have an impact on performance.

For include action, JSP Engine adds the content of the inserted page at run time which imposes extra overhead.

In the case of action include the page gets included when a request comes. This is a dynamic include.

So while including files make intelligent includes. If you go for performance and your included files do not change.

If you want to include something based on your input go for action include.



Best Practices of JSP

Avoid giving unnecessary scope in the 'useBean' action.

Don'ts:

```
<jsp:useBean id="objectName" scope="session" />
```

Do's:

```
<jsp:useBean id="objectName" scope="page|request|session|application" />
```

Implications:

- Application and session scope if used wrongly results in eating up memory. Finally this can cause memory choking.

Page 59

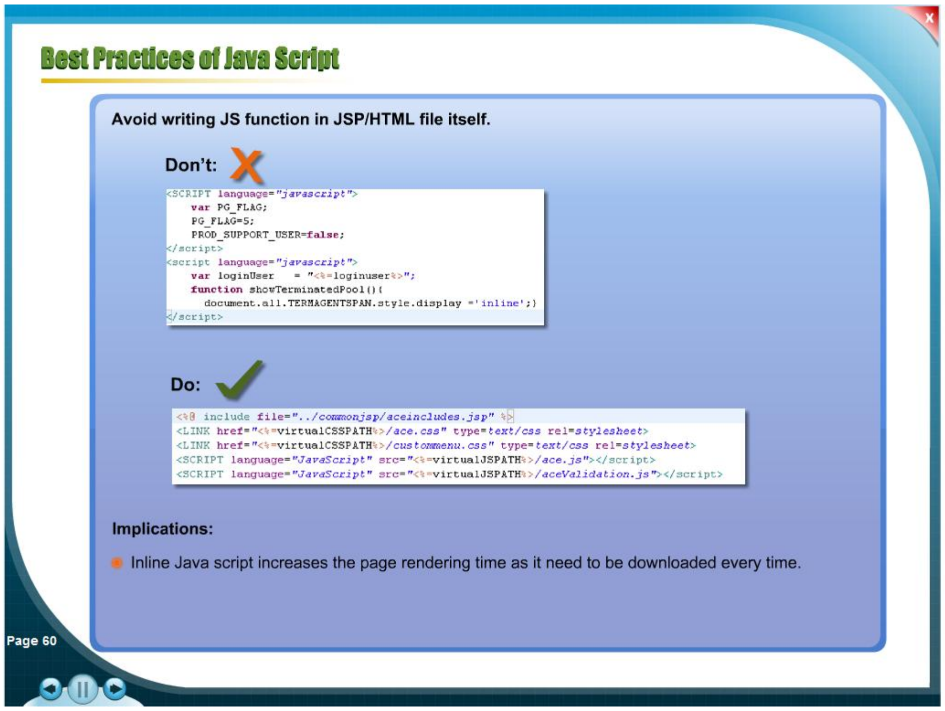
When you want to create a bean using *useBean* action tag you can set scope for that bean

<jsp:useBean id="objectName" scope="**page**|request|session|application" />

default value is 'page' for any bean if you don't specify the scope explicitly.

By defining scope attribute, you are defining the life time of that object, when it has to be created and when its life time ends. To be precise, you are defining the availability of that object to a page, request, session (that is across multiple requests to a user) or application (across multiple users). Here the scope effects the performance if you don't specify exact scope as per your requirement. What will happen if you set a session scope for an object which is needed only a request? The object will unnecessary reside in the memory even after your work is done. When using the session or application scope object you have to explicitly remove it after you are done. Otherwise the session object will be there in the memory till you explicitly remove the object or your server removes it after a configured time limit (typically it is 30 minutes). It reduces the performance by imposing overhead on memory and garbage collector. The same is the problem with the application scope objects.

So set exact scope for an object and also remove those scope objects immediately whenever you are done with them.



Best Practices of Java Script

Avoid writing JS function in JSP/HTML file itself.

Don't:

```
<SCRIPT language="javascript">
    var PG_FLAG;
    PG_FLAG=5;
    PROD_SUPPORT_USER=false;
</script>
<script language="javascript">
    var loginUser    = "<%=loginuser%>";
    function showTerminatedPool(){
      document.all.TERMAGENTSPAN.style.display ='inline';}
</script>
```

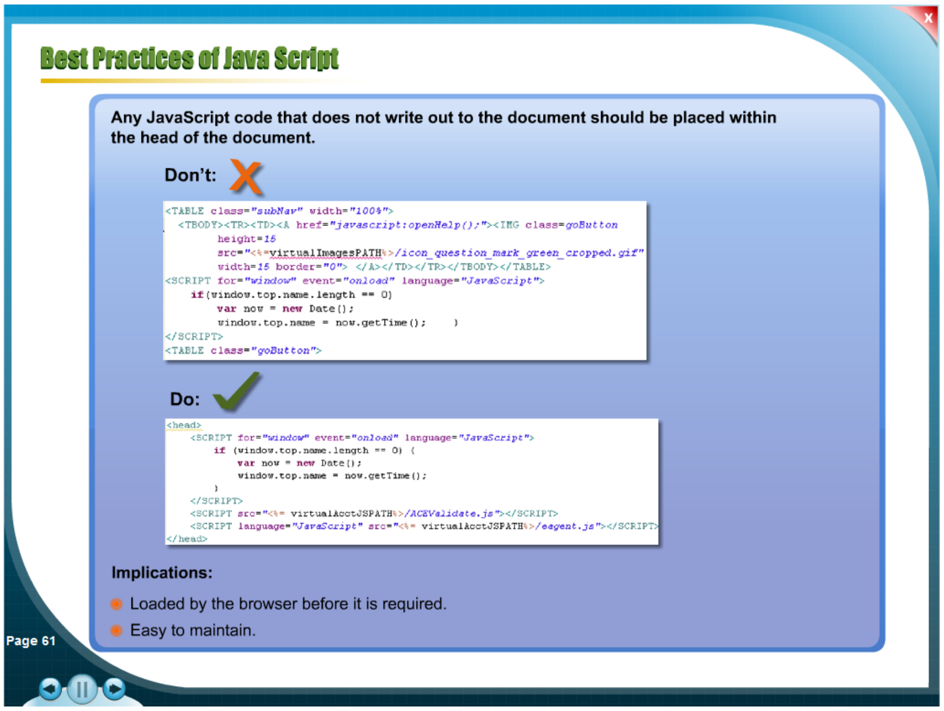
Do:

```
<%@ include file="../commonjsp/aceincludes.jsp" %>
<LINK href="<%=virtualCSSPATH%>/ace.css" type=text/css rel=stylesheet>
<LINK href="<%=virtualCSSPATH%>/custommenu.css" type=text/css rel=stylesheet>
<SCRIPT language="JavaScript" src="<%=virtualJSPATH%>/ace.js"></script>
<SCRIPT language="JavaScript" src="<%=virtualJSPATH%>/aceValidation.js"></script>
```

Implications:

- Inline Java script increases the page rendering time as it need to be downloaded every time.

Using external files in the real world generally produces faster pages because the JavaScript and CSS files are cached by the browser. JavaScript and CSS that are inlined in HTML documents get downloaded every time the HTML document is requested. This reduces the number of HTTP requests that are needed, but increases the size of the HTML document. On the other hand, if the JavaScript and CSS are in external files cached by the browser, the size of the HTML document is reduced without increasing the number of HTTP requests.



Best Practices of Java Script

Any JavaScript code that does not write out to the document should be placed within the head of the document.

Don't:

```
<TABLE class="subNav" width="100%">
  <TBODY><TR><TD><A href="javascript:openHelp();"><IMG class=goButton
        height=15
        src="<%=virtualImagesPATH%>/icon_question_mark_green_cropped.gif"
        width=15 border="0"> </A></TD></TR></TBODY></TABLE>
<SCRIPT for="window" event="onload" language="JavaScript">
    if(window.top.name.length == 0)
        var now = new Date();
        window.top.name = now.getTime();     }
</SCRIPT>
<TABLE class="goButton">
```

Do:

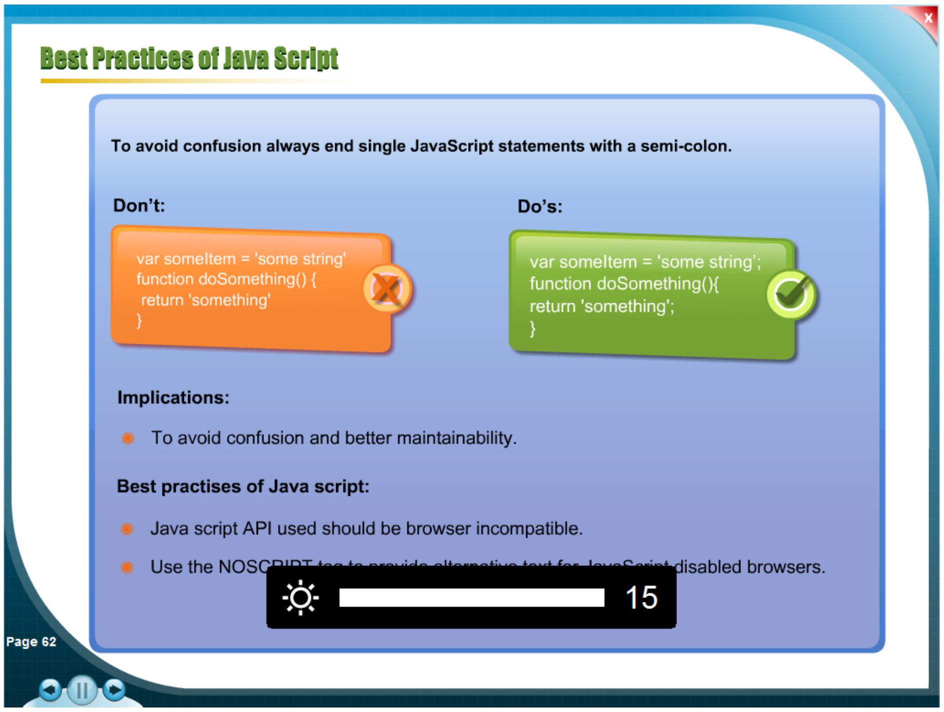
```
<head>
    <SCRIPT for="window" event="onload" language="JavaScript">
        if (window.top.name.length == 0) {
            var now = new Date();
            window.top.name = now.getTime();
        }
    </SCRIPT>
    <SCRIPT src="<%= virtualAcctJSPATH%>/ACEValidate.js"></SCRIPT>
    <SCRIPT language="JavaScript" src="<%= virtualAcctJSPATH%>/eagent.js"></SCRIPT>
</head>
```

Implications:

- Loaded by the browser before it is required.
- Easy to maintain.

Page 61

This ensures that the JavaScript function definitions have been loaded by the browser before it is required. It also makes it slightly easier to maintain the JavaScript code if it can always be found in the head of the document.



Best Practices of Java Script

To avoid confusion always end single JavaScript statements with a semi-colon.

Don't:

```
var someItem = 'some string'
function doSomething() {
 return 'something'
}
```

Do's:

```
var someItem = 'some string';
function doSomething(){
return 'something';
}
```

Implications:

- To avoid confusion and better maintainability.

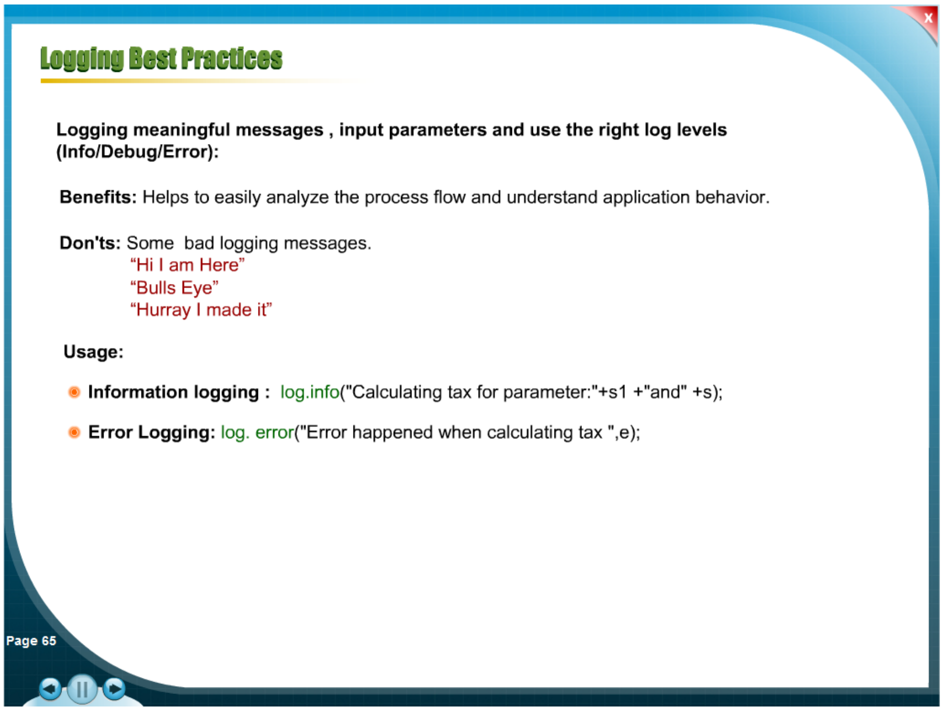
Best practises of Java script:

- Java script API used should be browser incompatible.
- Use the NOSCRIPT tag to provide alternative text for JavaScript-disabled browsers.

Page 62

<u>Java script API compatibility:</u> This happened in one project where we developed a java script sorting functionality for reporting module. The functionality worked in IE perfectly but few

customers started to netscape where the functionality failed. We had to revamp the entire design and had to change the application. This costed us lots of money and effort.



Logging Best Practices

Logging meaningful messages , input parameters and use the right log levels (Info/Debug/Error):

Benefits: Helps to easily analyze the process flow and understand application behavior.

Don'ts: Some bad logging messages.

"Hi I am Here"
"Bulls Eye"
"Hurray I made it"

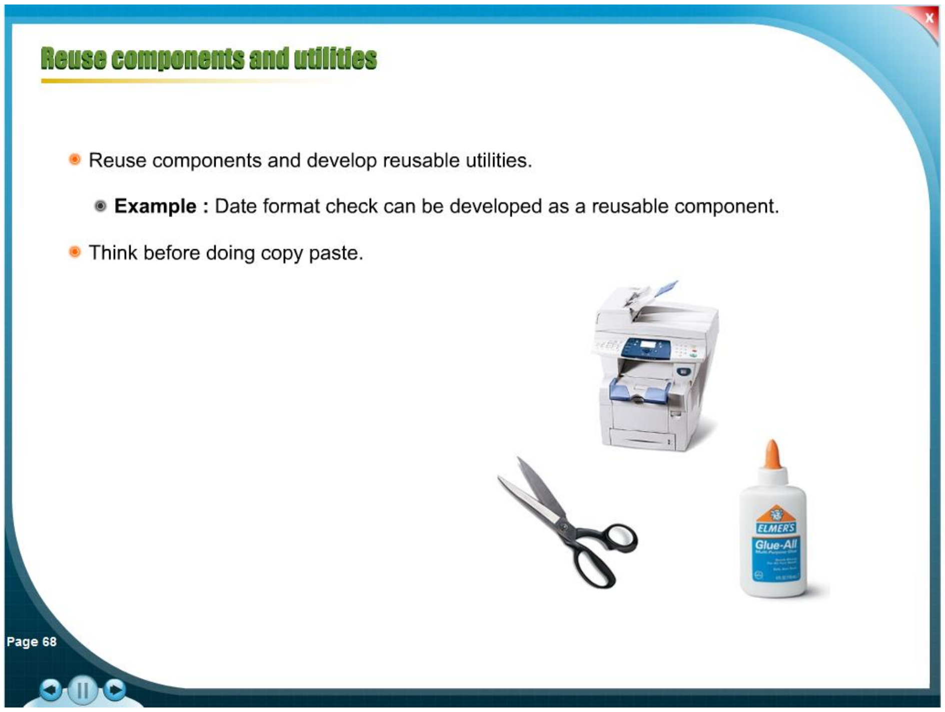
Usage:

- **Information logging :** `log.info("Calculating tax for parameter:"+s1 +"and" +s);`
- **Error Logging:** `log. error("Error happened when calculating tax ",e);`

<NOTE>Please stress the importance of logging meaningful message. Explain as below <NOTE>

When developing application please ensure to use the right log level. Use info for debugging application in production mod. In production mode the debug will be switched off. Log the method parameters, in case of value objects override the to string method and print the value objects. When logging errors log the throwable object.

Trainers can quickly recap the session content delivered till now.



Reuse components and utilities

- Reuse components and develop reusable utilities.
  - **Example :** Date format check can be developed as a reusable component.
- Think before doing copy paste.

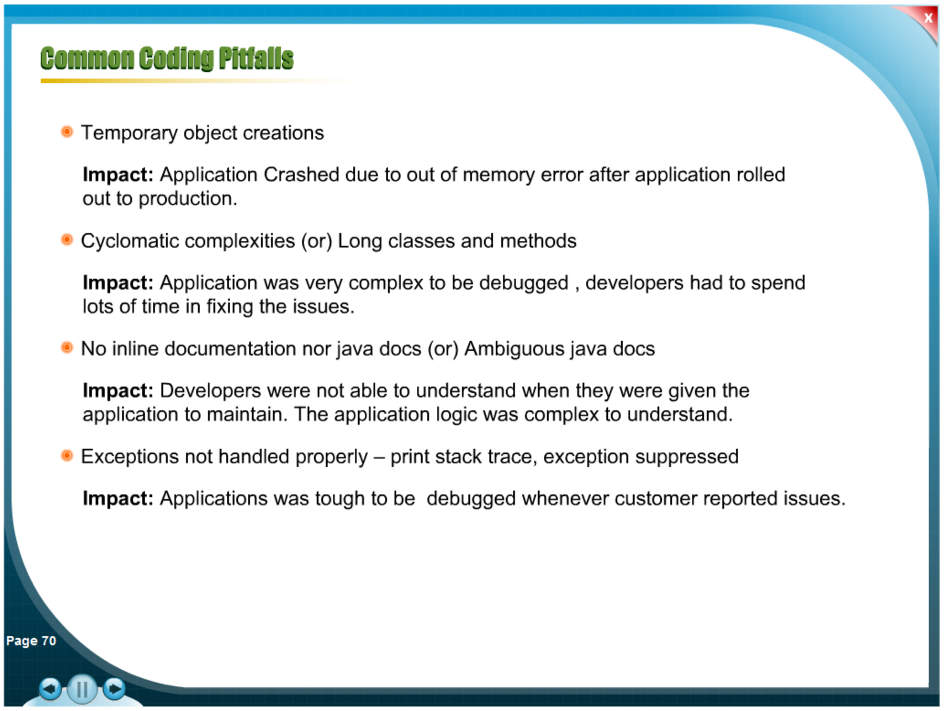
<NOTE>Explain as mentioned below</NOTE>

- Reuse components and develop reusable utilities.

Let me illustrate a example, in one of the projects in cognizant the developers developed an application which had more than 500+ screens. Many of these screens had date control whose format needs to be validated. Many of the developers wrote their own logic of checking the common format. This resulted in we developing lots of methods doing the same logic in different ways. After an year the customer changed the date format, we ended up changing and testing all the methods, it was a nightmare. We spent lots of effort and money to do this. Imagine if we have done this as one component and one method all it needs was few hours of effort. But we ended up in burning 3 weeks to fix the issues. So always develop reusable component.

- Think before doing copy paste.

Before you copy paste one or more lines of code see if you can move it to a common method and reuse.



Common Coding Pitfalls

- Temporary object creations

  Impact: Application Crashed due to out of memory error after application rolled out to production.
- Cyclomatic complexities (or) Long classes and methods

  Impact: Application was very complex to be debugged , developers had to spend lots of time in fixing the issues.
- No inline documentation nor java docs (or) Ambiguous java docs

  Impact: Developers were not able to understand when they were given the application to maintain. The application logic was complex to understand.
- Exceptions not handled properly – print stack trace, exception suppressed

  Impact: Applications was tough to be debugged whenever customer reported issues.

Page 70

The cyclomatic complexity of a section of source code is the count of the number of linearly independent paths through the source code. For instance, if the source code contained no decision points such as IF statements or FOR loops, the complexity would be 1, since there is only a single path through the code. If the code had a single IF statement containing a single condition there would be two paths through the code, one path where the IF statement is evaluated as TRUE and one path where the IF statement is evaluated as FALSE.

Best Practices Game:

End of the session the trainers need to deliver a game where in trainees need to find the best practices violation in few problems. The objective is that the CAT recaps the best practices learnt as part of the session.

The game can be individually played. The associates can download the game and play it independently. Duration for the entire game is 30 minutes.