Java Best Practices

Elango Sundaram

TABLE OF CONTENTS

Document Objective

This document aims to capture some of the best practices for developing Java Enterprise Application. This should be ideally used during Design Cycle.

Java

Resource Handling

- It is preferable to have a common mechanism to get and return resources. For example a common class /method should be used by all to get and release Database Connections, Result Sets, Statements.
- The resources must be released only in the ***finally*** block in java code and not in the try block. This will enable proper resource release even in the event an exception is thrown in try block. For example, the database connection and result sets should be closed in the finally block
- Streams (like File) should be closed after use
- Hard coding of strings in database queries must be avoided. For example Table/View/Schema Names should preferably be made constants and placed in a common file. The SQL Queries across application can reuse these name strings
- For SQL Queries, the Results should preferably be obtained using something like, lResultSet.getString ("column-name") instead of something like lResultSet.getString (1)

Comments

- In-line comments should be placed wherever needed
- Javadoc Comments must be available for all classes and methods
- Standard version numbers should be used across the application
- Author Names must be available for all classes
- All the commented out code must be removed before code release
- If any SQL procedures are called from within a java class, then documentation should be available in the java code as to what the procedure is doing
- If the SQL queries inside of DAO (Data Access Objects) become too long, they must be properly commented; fully explaining the intricacies of what the SQL is doing

NLS

- If the application is NLS (National Language Support) compatible, then a common approach must be followed to implement the same across the application. All the NLS compatible message have to be placed in a common place (preferably in XML files). Common contract must be established among developers to reduce redundant NLS messages. One way to avoid redundancy is to segregate the messages as *common* and *module* specific messages. All the application level common messages can be placed in a single file and each module can have a module specific message file
- All the IMG (image) attributes in HTML should have the ALT (Alternate Text) tag specified and the ALT message should be an NLS messages
- If the application is NLS compatible, then the Alert and confirm messages should be NLS.

Error Handling

- Error messages must be identified and placed in a common file (preferably XML), so that the error messages can be re-used. One way to avoid redundancy is to segregate the messages as *common* errors and *module* specific errors. All the application level common error messages can be placed in a single file and each module can have a module specific error message file. This will reduce redundant messages
- A common approach must be used to handle error display. For example errors like, “Failed to initialize Java Bean” should be logged and not displayed on the screen

- Null Pointer Exceptions should not be thrown in the application. If there is a variable, that probably could come un-initialized, checking should made if it is null and logging must be done. No operation should be made on a null which would make the application throw Null Pointer Exception.
- Empty catch blocks should be avoided. It should at least have a log, so that debugging will be easy in the future
- All Exceptions must be documented in Javadoc using the @throws tag

Others/ general

- In projects involving heavy calculations, a commonly established contract must be used with reference to rounding off the numbers. Across the application, common class could be used to round off numbers. So global changes can be made if needed, easily.
- Sometimes a number could be obtained from the database and in other instances a derivative (resultant of a calculation) could be used. Its better to have a clear guideline on when a number should be obtained from the database as against obtaining from previous screens/ calculations.
- Redundant coding should be reduced. For example, common code must be used across application for getting EJB references, accessing Resources etc
- Transaction rollback (), commit () should be in the separate try-catch block as it can also throw an exception. If we do not want to do anything about a transaction, which fails to roll back, at least a log must be made on the failure.
- Logging should be based on some standard mechanism (like using Log4J). The logging level should be set based on the type of messages logged. This is important, since the production deployment may be done with a log level like “INFO” or “WARN”, instead of the standard “DEBUG”. Correct logging will be useful when we attempt to debug a live application using the log file.
- Looping should not be done using hard coded variables.
- Duplicate class names (across package) should be avoided/ reduced. (This is based on project standards)
- Hard coding of strings must be avoided esp., is the string is used in more than once.
- No System.out.println in the java code. Standard logging mechanism should be used instead.
- There should be space between the package and the import statements
- The classes must be named according to project level specification.
- Package must named in accordance to project level specification.
- Class and local attributes must be named in accordance to project level specification.
- Constants must be named in accordance to project level specification.
- Complicated looping structure must be avoided. In case the looping is complicated, then proper commenting should be done to explain the logic.
- In applications involves currency exchanges, a common class/method may be written to handle conversion.
- Common code should be conceived and built as much as possible. For example, common code should be used for applications like Calendar. This can be re-used at different parts of the application

JSP

- If there are multiple modules in the application, the JSP's must preferably be arranged according to the modules
- Based on complexity, its preferable to have one JSP per web page
- JSPs should have presentation logic only. If any logic like currency conversion etc., is needed, then the logic should be coded in the Java Beans/Class and not directly in JSP

- Verifying the user input can be made in a common class / bean and can be used across application
- JSPs should not allow caching at the browser. This can be done by using something like response.setHeader("Pragma", "no-cache") & response.setHeader("Expires", "-1");
- No System.out.println shall be present in JSP
- Special characters like (<,>) should be properly escaped. For example '<' should be made as &lt, and '>' should be made as &gt.
- Comments must be available for the Java Script functions being used in the JSP file
- The JSPs should preferably have comment block, which defines the purpose of the JSP, version, author etc
- Out.println() must be reduced as much as possible and instead HTML should be used along with Java variables
- All the java script functions should be present at one place instead of being placed at different places in the code
- All the Java code in JSP must be placed in Java Beans. Straight Java Coding in JSP must be avoided
- Commonly used code like paginations should be wrapped in a Tag Library and may be used across application
- During Logout, remove unneeded information from the session.

Author:
Elango Sundaram,
www.geocities.com/esundara
Chennai,India