

JAVA RealTime Tools

By- Mr. NATRAJ & TEAM

LOG4J

ANT

JUNIT

CVS

SVN

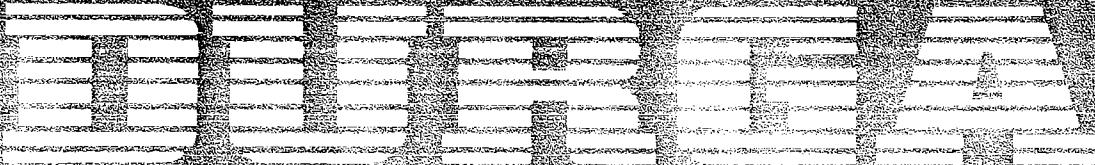
Jasper Reports

Maven

Installer

No.1 in JAVA TRAINING

AN ISO 9001:2008 CERTIFIED



Software Solutions®

SANDYAXEROX

Gayatrynagar, Ameerpet, HYD.

Cell: 8985928289

DURGA SOFTWARE SOLUTIONS
Tools Material

INDEX

- ANT TOOL -----→1
- LOG4J-----→18
- JUNIT-----→ 45
- CVS-----→81
- JasperReports-----→112
- maven-----→139
- SVN-----→206
- Applications-----→278

Ant Tool

Contents:

1. Introduction to Ant
2. Ant Installation
3. build.xml File Architecture
4. Ant Sample Build File - JAR
5. Ant Eclipse IDE Integration
6. Ant Property Task
7. Ant Properties File
8. Ant pre-defined Tasks

1. Introduction to Ant (Another Neat Tool):

Ant is an open source build technology developed by Apache intended to build processes in Java environment. It is a similar kind of tool like **make**, but it does not use shell commands to extend the functionality. The use of shell commands in *make* brings about the integrity with other languages too but this also makes it platform specific. In contrast, Ant is based on XML and uses java classes in automatic generation of build processes that makes it platform independent. It is applicable to any integrated development environment (IDE) that uses java. A build file is generally named as ***build.xml***.

The best features of the Ant technology can be summarized as below -

- **Easy to Use:** It is not a programming language, it is an XML based scripting tool, therefore easy to understand and implement.
- **Portable and Cross-platform based:** Use of Java classes makes it portable, i.e., it can be run on any operating system.
- **Extended Functionality:** Ant is based on java platform, that's why its functionality can be extended to any development environment based on java. It is easier to implement than any specific IDE because it is automated and ubiquitous.
- **Build Automation:** Ant provides automated build process that is faster and more efficient than manual procedures and other build tools can also be integrated with it.
- **Compilation of Source Code:** Ant can use and compile source code from a variety of version controls and packaging of the compiled code and resources can also be done.
- **Handling Dependencies between Targets:** An Ant Project describes the target and tasks associated with it and also handle dependencies between various targets and tasks.

Ant Definition

Apache Ant is an open source, cross-platform based build tool that is used to describe a build process and its dependencies and implemented in XML scripts using Java classes that ensures its

DURGA SOFTWARE SOLUTIONS
Tools Material

extensibility to any development environment (based on Java) and its integrity with other build tools.

2. Ant Installation:

Ant is free and open source build tool, written in Java, helps in automating the entire build process of a Java development project.

- Ant uses XML build files.
- By default, Ant looks for a build file named build.xml.
- The build file contains information about how to build a particular project.
- Each project contains multiple targets like creating directory, compiling source codes.
- Target can depend on other targets.
- Targets contain tasks.
- Behind each task is a Java class that performs the described work.

To install Ant follow the steps given below.

- a) Download latest or required version from Apache Foundation website
<http://ant.apache.org/bindownload.cgi>
- b) Extract Zip file to your local Disk say "E:" drive
D:\apache-ant-1.8.2
- c) Set the ANT_HOME environment variable to point to the ant installation directory.
ANT_HOME=E:\apache-ant-1.8.2
- d) Set the JAVA_HOME environment variable to point to the JDK location.
JAVA_HOME=E:\JDK1.5
- e) Add ANT_HOME/bin and JAVA_HOME/bin to your system's PATH environment variable.
PATH=E:\apache-ant-1.8.2\bin; D:\JDK1.5\bin;

To make sure the installation is proper, go to command prompt and execute the command **ant**.

```
C:\>ant
Buildfile: build.xml does not exist!
Build failed
C:\>
```

build.xml File

Ant is a build tool that means the main aim of Ant tool is to automation JAVA Project process.

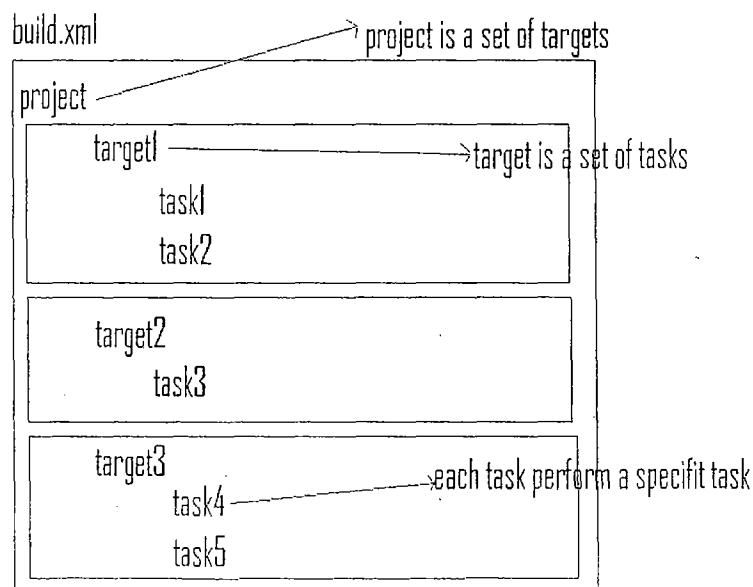
By default ant uses build.xml file for writing build script. It is an XML file.

It is a standard file name followed by everyone. It is not mandatory We can use any file name like banking.xml, bank-build.xml etc.

We can use ant tool to do the following things

- ✓ Create directories
- ✓ Delete directories
- ✓ Copy files from one directory to another directory
- ✓ Create new files
- ✓ Compile and run java files
- ✓ Create a war file
- ✓ Create a jar file
- ✓ Create an ear file
- ✓ Deploy war or ear into a server etc.

build.xml File Architecture



Content of build.xml file:

Each JAVA project contains one or more build script files. Each Build script file contains a project. Default build file name is 'build.xml', but we can use any name.

DURGA SOFTWARE SOLUTIONS
Tools Material

Ant's build files are written in XML.

Project:

We can represent this project by using `<project>` element and it contains some attributes like attribute "name" is used to specify project name etc.

That means build.xml contains `<project>` as root element.

Example:-

```
<project name="bank" .....>  
</project>
```

The `<project>` tag has three attributes:

- ✓ name
- ✓ default
- ✓ basedir

`<Project>` attributes description:

- ✓ The name attribute gives the project a name.
- ✓ The default attribute refers to a target name within the buildfile. If you run Ant without specifying a target on the command line, Ant executes the default target. If the default target doesn't exist, Ant returns an error.
- ✓ The basedir attribute defines the root directory of a project. Typically, it is ".", the directory in which the buildfile resides, regardless of the directory you're in when you run Ant. However, basedir can also define different points of reference.

Target:

Each `<project>` contains a set of targets. At least one target is mandatory.

We can represent this target using `<target>` element. Just like `<project>`, each target contains one name. We can represent this target name using "name" attribute.

We should provide unique target names within one build script file.

DURGA SOFTWARE SOLUTIONS
Tools Material

Example:-

```
<project name="bank" ....>  
  
    <target name="target-1" ...>  
    </target>  
  
    <target name="target-2" ...>  
    </target>  
  
    <target name="target-3" ...>  
    </target>  
  
</project>
```

The `<target>` tag has three attributes:

- ✓ name
- ✓ depends
- ✓ description

The name attribute gives the target a name.

The depends attribute are a comma-separated list of names of targets on which this target depends.

The description attribute a short description of this target's function.

Task:

Each target contains one or more number of tasks.

We can write targets without tasks also that means tasks are not mandatory.

Syntax:

```
<project name="someprojectname">  
    <target name="dosomething">  
        <task1 param1="value1" param2="value2">  
        <task2 param3="value3" >  
        ...  
    </target>  
    <target name="target2">  
        <task1 param1="value1" param2="value2">  
        <task2 param3="value3" >  
        ...  
    </target>  
</project>
```

3. Ant Sample Build File - JAR:

Example 1:

In this example you will see how to compile a java program and compress it into a **.jar** file using Ant build file. The following listing shows the **build.xml** file.

```
01. <? xml version="1.0" ?>
02. <project name="Hello World" default="compress">
03.
04. <target name="compile">
05.   <javac srcdir=". />
06.   <echo> Compilation Complete! </echo>
07. </target>
08.
09. <target name="compress" depends="compile">
10.   <jar destfile="HelloWorld.jar" basedir=". "
        includes="*.class" />
11.   <echo> Building .jar file Complete! </echo>
12. </target>
13.
14. </project>
```

- The **<project>** element is the root element in Ant build files. The **name** attribute of the **<project>** element indicates the project name. Each project element can contain multiple **<target>** elements.
- A **<target>** represents a single stage in the build process. A build process can have multiple **<targets>**. Here we have two targets **compile** and **compress**.
- The **default** attribute of **<project>** element indicates the default target to be executed. Here the default target is **compress**.
- When you see the **<compress>** target, it in turn depends on the **<compile>** target, that is indicated by the **depends** attribute. So the **<compile>** target will be executed first.
- The **<compile>** target has two task elements **<javac>** and **<echo>**. The **javac** task is used to compile the java files. The attribute **srcdir=". /"** indicates all the java files in the current directory. Echo task is used to display message on the console..

- The compress target also performs two tasks, first the `<jar>` element as the name indicates, is used to build the jar file. The attributes `destfile="HelloWorld.jar"`, `basedir=".."` and `includes="*.class"` indicates all the `.class` files in the current directory should be compressed into `HelloWorld.jar` file. Later the `echo` task is used to display the success message on the console.

To run the `build.xml` file, open the command prompt, go to the example directory, type the command "ant". You will see the following information.

```
PS C:\WINDOWS\system32\cmd.exe
E:\ant examples\Example1>ant
Buildfile: build.xml

compile:
  [javac] Compiling 1 source file
  [echo] Compilation Complete!

compress:
  [jar] Building jar: E:\ant examples\Example1\HelloWorld.jar
  [echo] Building .jar file Complete!

BUILD SUCCESSFUL
Total time: 0 seconds
E:\ant examples\Example1>
```

Example2:

In this example we will see how to structure the project. If the grows bigger, it will become a problem to manage the files if all the files are there in the same directory.

For a easier maintenance all the source file should be kept in the `src` directory, the compressed jar file should be kept in the `dist` directory and all the intermediate class files should be kept in the `build/classes` directory.

By imposing structure cleaning the project becomes easy, we can just delete the directory and recreate it.

Using the `<mkdir>` task we create `build/classes` and `dist` directory.

DURGA SOFTWARE SOLUTIONS
Tools Material

```
1. <target name="init">
2.   <mkdir dir="build/classes" />
3.   <mkdir dir="dist" />
4. </target>
```

During the compilation process all the java files in the **src** directory will be compiled and the generated class files will be placed in the **build/classes** directory.

Since we placed all the class files under the **build/classes** directory, creating jar file becomes easier, you can simply specify the **basedir** attribute as "**build/classes**" instead of specifying **basedir=". "** and **includes="*.class"**. After creating the jar file we place it in the dist directory (**destfile="dist/HelloWorld.jar"**).

```
1. <target name="compile" depends="init">
2.   <javac srcdir="src" destdir="build/classes" />
3. </target>
4.
5. <target name="compress" depends="compile">
6.   <jar destfile="dist/HelloWorld.jar"
        basedir="build/classes"/>
7. </target>
```

You can use the **java** task to execute a class file as shown below. The **classname** attribute refers to **the java class to be executed** and the **classpath** attribute refers to **the directory in which the class is located**.

```
1. <target name="execute" depends="compile">
2. <java classname="com.vaannila.helloworld.HelloWorld"
       classpath="build/classes" />
3. </target>
```

Since all the class files and jar files are isolated, we can easily clean the project by deleting the respective directories.

DURGA SOFTWARE SOLUTIONS
Tools Material

```
1. <target name="clean">
2. <delete dir="build" />
3. <delete dir="dist" />
4. </target>
```

The default target is **compress**, so when you run the build file the **compress** target will be executed. The **compress** target depends on **compile** target which in turn depends on the **init** target, so first the **init** target will be executed and the two directories will be created, then the **compile** target will be execute, later the jar file is created.

```
C:\WINDOWS\system32\cmd.exe

E:\ant examples\Example2>ant
Buildfile: build.xml

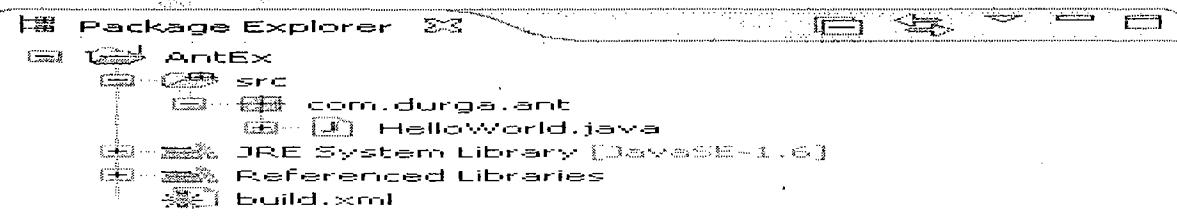
init:
  [mkdir] Created dir: E:\ant examples\Example2\build\classes
  [mkdir] Created dir: E:\ant examples\Example2\dist

compile:
  [javac] Compiling 1 source file to E:\ant examples\Example2\build\classes

compress:
  [jar] Building jar: E:\ant examples\Example2\dist\HelloWorld.jar

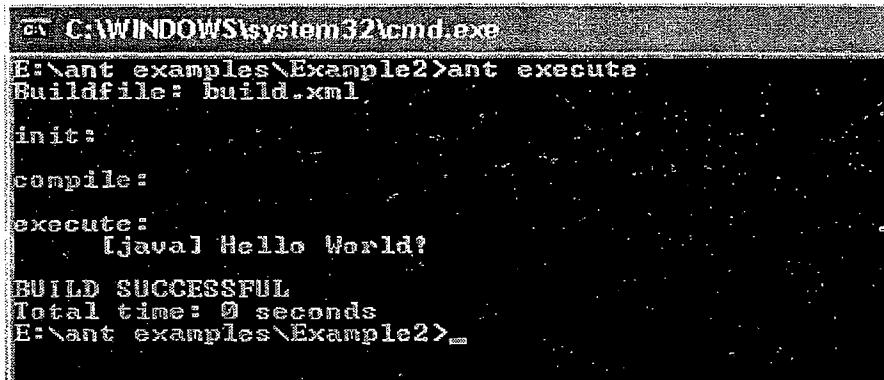
BUILD SUCCESSFUL
Total time: 0 seconds
E:\ant examples\Example2>..
```

When you run the "**ant execute**" command the **execute** target will be invoked. **Execute** target also depends on **compile** target which in turn depends on **init** target, but now the directories won't be created again because it already exist. Since the java file is already compiled it won't be compiled again. Ant checks the timestamp of the file and compiles only the updated files. The *HelloWorld.class* file will be executed and the "Hello World!" message will be displayed on



DURGA SOFTWARE SOLUTIONS
Tools Material

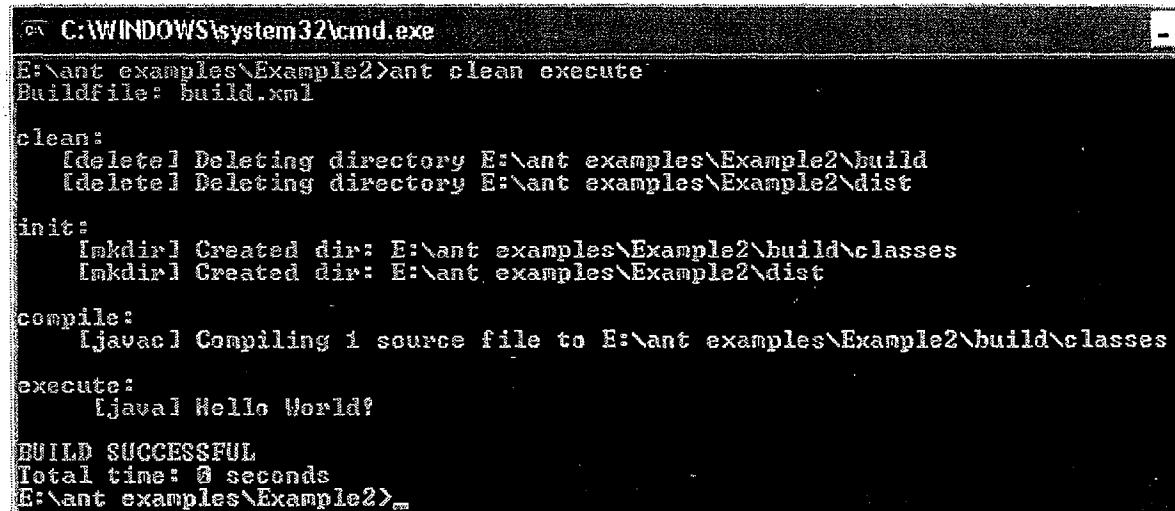
the console.



```
PS C:\WINDOWS\system32\cmd.exe
E:\ant examples\Example2>ant execute
Buildfile: build.xml

init:
compile:
execute:
    [java] Hello World!
BUILD SUCCESSFUL
Total time: 0 seconds
E:\ant examples\Example2>_
```

To clean and execute the program run the "**ant clean execute**" command. First the **clean** target will be executed and the directories will be deleted, later the execute task will be invoked.



```
PS C:\WINDOWS\system32\cmd.exe
E:\ant examples\Example2>ant clean execute
Buildfile: build.xml

clean:
    [delete] Deleting directory E:\ant examples\Example2\build
    [delete] Deleting directory E:\ant examples\Example2\dist

init:
    [mkdir] Created dir: E:\ant examples\Example2\build\classes
    [mkdir] Created dir: E:\ant examples\Example2\dist

compile:
    [javac] Compiling 1 source file to E:\ant examples\Example2\build\classes

execute:
    [java] Hello World!

BUILD SUCCESSFUL
Total time: 0 seconds
E:\ant examples\Example2>_
```

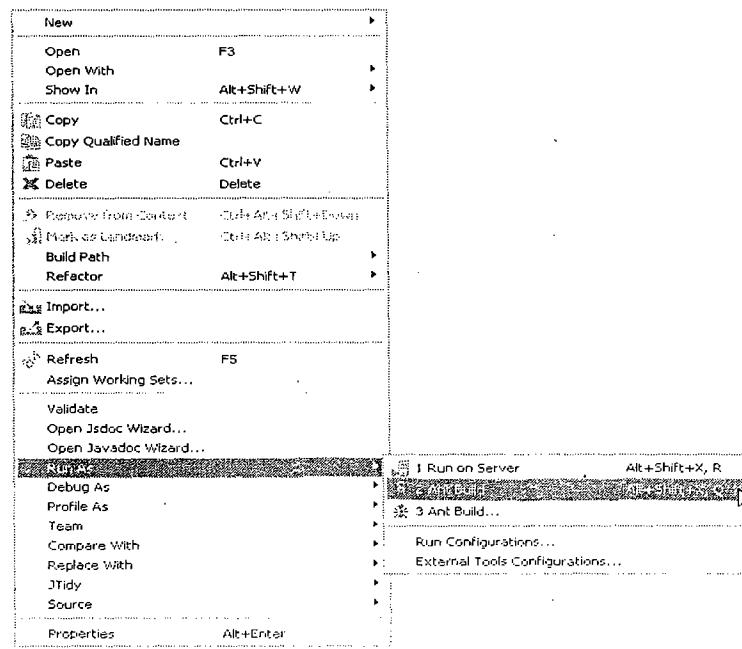
4. Ant Eclipse IDE Integration

To integrate Ant build file with the Eclipse IDE, first create a **build.xml** file, right click the project select **New->Other->XML**, enter the name as **build.xml** and click *Finish*.

DURGA SOFTWARE SOLUTIONS
Tools Material

```
01. <?xml version="1.0" ?>
02. <project name="Ant Example" default="execute">
03.
04. <target name="init" depends="clean">
05.   <mkdir dir="build/classes" />
06. </target>
07.
08. <target name="compile" depends="init">
09. <javac srcdir="src" destdir="build/classes" />
10. </target>
11.
12. <target name="execute" depends="compile">
13. <java classname="com.durga.ant.HelloWorld"
      classpath="build/classes" />
14. </target>
15.
16. <target name="clean">
17. <delete dir="build" />
18. </target>
19.
20. </project>
```

To build the project right click the *build.xml* file and select ***Ant Build***.



DURGA SOFTWARE SOLUTIONS
Tools Material

The **HelloWorld.java** file will be compiled and executed. The following message shows the sequence of events that happen once the build file is executed.

```
01. Buildfile: E:\Eclipse Workspace\AntExample\build.xml
02. clean:
03. [delete] Deleting directory E:\Eclipse
   Workspace\AntExample\build
04. init:
05. [mkdir] Created dir: E:\Eclipse
   Workspace\AntExample\build\classes
06. compile:
07. [javac] Compiling 1 source file to E:\Eclipse
   Workspace\AntExample\build\classes
08. execute:
09. [java] Hello World!
10. BUILD SUCCESSFUL
11. Total time: 625 milliseconds
```

5. Ant Property Task:

The **<property>** task is used to set the Ant properties. The property value is immutable, once the value is set you cannot change it. To set a property to a specific value you use Name/value assignment.

Property		
Attribute	Description	Requirement
name	name of the property, which is case-sensitive	not necessary
value	name of task attribute, this is done by placing the property name between "\${name}" in the attribute value	necessary
location	it contain property name	not necessary
file	name of the property file	not necessary

DURGA SOFTWARE SOLUTIONS
Tools Material

```
1. <property name="project.name" value="AntExample2" />
```

To set a property to a location you use Name/location assignment.

```
1. <property name="web.dir" location="WebContent"/>
2. <property name="web.lib.dir" location="${web.dir}/WEB-
   INF/lib"/>
3. <property name="build.classes.dir"
   location="build/classes"/>
4. <property name="dist.dir" location="dist"/>
```

To use the properties surround them with \${}.

6. Ant Properties File:

You can also group all the property values in a separate properties file and include it in the ant build file. Here the **build.properties** file contains all the property values. Remember the property value is immutable, so if you set a property value in the properties file you cannot change it in the build file. This gives more control over the build process.

The **build. Properties** file:

```
1. Web.dir=WebContent
2. web.lib.dir=${web.dir}/WEB-INF/lib
3. build.classes.dir=build/classes
4. dist.dir=dist
5. project.name=AntExample3
```

Use the **property** task to include the **properties file** in the **Ant build file**.

```
1.<property file="build.properties" />
```

(Q) How to ear your Java application using Ant?

DURGA SOFTWARE SOLUTIONS
Tools Material

We use ant tasks **<ear>** to create ear of your Java Application

<ear> task syntax:

```
<ear destfile="ear-file-name-and-location"
     appxml="applicaton-config-file ">
  <fileset dir="files-dir" includes="ListofJarsAndWars"/>
</ear>
```

Parameters

destFile

The EAR file to create.

appxml

Display name to insert into the application.xml

Example:

```
<ear destfile="${build.dir}/myapp.ear"
      appxml="${src.dir}/metadata/application.xml">
  <fileset dir="${build.dir}"
    includes="*.jar,*.war"/>
</ear>
```

7. Ant pre-defined Tasks:

The following List represents the available pre-defined task libraries in ant tool.

- 1. Archive Tasks**
- 2. Audit/Coverage Tasks**
- 3. Compile Tasks**
- 4. Deployment Tasks**
- 5. Documentation Tasks**
- 6. EJB Tasks**
- 7. Execution Tasks**
- 8. File Tasks**
- 9. Java2 Extensions Tasks**
- 10. Logging Tasks**
- 11. Mail Tasks**
- 12. Miscellaneous Tasks**
- 13. Pre-process Tasks**
- 14. Property Tasks**

DURGA SOFTWARE SOLUTIONS
Tools Material

15. Remote Tasks

16. SCM Tasks

17. Testing Task

Note: if you want to know more about Ant pre-defined tasks, please refer the site <http://ant.apache.org/manual/tasksOverview.html>

All the best.....

Real-Time Log4J

Log4J Content

- ✓ Introduction to Logging
 - What is Logging?
 - Importance of Logging
 - Available Logging Frameworks for JAVA
- ✓ Introduction to Log4J
 - What is Log4J?
 - Why we need to use Log4J?
 - Log4J Features
 - Log4J Advantages
- ✓ Log4J Setup for Java: Standalone and Web Applications
 - Download Log4J
 - log4J Jar file
 - log4j.properties
- ✓ Log4J Development Approaches
 - Programmatic
 - Declarative
- ✓ Log4J Programmatic Implementation
 - Logger
 - BasicConfigurator, PropertyConfigurator
 - Setting Logging Level
- ✓ Log4J Logging Levels
 - Default Logging Level
 - Available Logging Levels
 - DEBUG, INFO, WARN, ERROR and FATAL
 - Log4J Levels Hierarchy
- ✓ Log4J Logger

DURGA SOFTWARE SOLUTIONS
Tools Material

- RootLogger
- Logger
- Relationship between RootLogger and Logger

✓ Log4J Declarative Implementation: log4j.properties Configuration

- Set Debug Level
- Set Appender
- Set PatternLayout
- Set ConversionPattern

✓ Log4J Appenders

- ConsoleAppender
- FileAppender
- RollingFileAppender
- AdminFileAppender
- ReportFileAppender
- Setting Single Appender
- Setting Multiple Appenders

✓ Log4J Implementation with xml file

- log4j.xml
- <log4j:configuration>
- Setting Appenders
- Setting Log Level
- Setting Conversion Pattern

✓ Conversion Pattern Syntax

- TTCC
- TTCCLayout
- Time Elapsed
- Thread Information
- Logging Priority
- Logging Category

DURGA SOFTWARE SOLUTIONS
Tools Material

- NDC - Nested Diagnostic Context
- Application Message
- ✓ Log4J Integration
 - Java Standalone Project
 - Servlets/JSP Project
- ✓ Log4J Issues
 - Log4J Drawbacks
 - Alternative to Log4J
 - Introduction to SLF4J
 - Advantages of SLF4J
 - Migration of Logging Framework
 - Log4J to SLF4J

Log4J

Introduction to Logging:

What is Logging?

Logging is a technique or process to record the development activities to a console or a log file.

Several Logging frameworks simplify and standardize the process of logging for the Java platform.

Why we need logging in an application?

- ✓ To understand the flow of the application logic
- ✓ To find out root cause of an issue
- ✓ To track transactions (In Banking Domain) permanently by storing log into a Data base.

Advantages of Logging:

- ✓ Easy to debug application
- ✓ Easy to find out root cause of the problem
- ✓ Easy to find out flow of the logic

Drawbacks of Logging:

- ✓ Logging severely affects the performance of the application
- ✓ It consumes some memory at server side to store data.

DURGA SOFTWARE SOLUTIONS
Tools Material

In Realtime, most of the developers uses different logging techniques during development to understand his/her code execution process and also to find out root cause of the problem.

As a less experience developer or not working people, you may like to use

`System.out.println(logMessage)`

Statement to log your statements to a console(Command prompt).

But it is NOT recommended to use `System.out.println(logMessage)` in real time. It has lot of advantages.

What is the simple logging technique available for a Java Program?

Using `System.out.println(logMessage)`

What is the major drawback of `System.out.println()`; logging technique?

When we want to deliver code from development to next phase or into production, we have to remove or comment all SOP statements one by one manually.

- ✓ It is very time consuming process
- ✓ It kills our development time
- ✓ It increases development time and cost

That's why most of the people uses one of the available logging frameworks in their application.

Popular Logging Frameworks for Java Applications:

S.No.	Logging Frameworks for JAVA
1	Java Logging API
2	Log4J
3	Apache Commons Logging
4	SLF4J

SLF4J stands for Simple Logging Facade for Java

Java Logging API

Sun MircoSystmes (Oracle Corporation) has introduced one logging framework as part of JSE(Java Standard Edition) API under `java.util.logging` package.

Sample API Classes

`java.util.logging.FileHandler`
`java.util.logging.Level`
`java.util.logging.LogManager`
`java.util.logging.Logger`

DURGA SOFTWARE SOLUTIONS
Tools Material

java.util.logging.XMLFormatter

Sample Java Logging Program:

```
import java.util.logging.Level;
import java.util.logging.Logger;
public class SampleProgram
{
    public static void main(String[] args)
    {
        Logger log = Logger.getLogger("Some Logging");
        log.info("Its an INFO Message");
        log.log(Level.SEVERE, "Its an INFO Message");
        log.info("Its an INFO Message");
    }
}
```

Output:

Jun 2, 2011 3:00:30 PM SampleProgram main

INFO: Its an INFO Message

Jun 2, 2011 3:00:30 PM SampleProgram main

SEVERE: Its an SEVERE Message

Jun 2, 2011 3:00:30 PM SampleProgram main

INFO: Its an INFO Message

JSE Logging API contains some drawbacks

- ✓ Does NOT contain meaningful logging levels
- ✓ Performance issues
- ✓ Less flexible to use

NOTE:-

Unlike other frameworks - Log4J, Java Logging API does NOT have a flexible API to use different configuration files like properties file, xml file etc.

The levels in descending order are:

- SEVERE (highest value)
- WARNING
- INFO
- CONFIG
- FINE
- FINER

DURGA SOFTWARE SOLUTIONS
Tools Material

- FINEST (lowest value)

Log4J solves most of these problems and provides many advantages.

LOG4J Framework

What is Log4J?

Log4J stands for Logging Framework for Java
It is an open source logging framework from Apache Software Foundation for Java Applications

Log4J is the most popular logging framework for Java Applications (for both Standalone Java Applications and Web applications).

Log4j is JDK 1.1.x compatible.

Log4J Framework Advantages:

- ✓ Contains meaningful logging levels
- ✓ Resolves some Performance issues
- ✓ Easy and flexible to use
- ✓ Supports both properties file configurations and XML configurations

Drawbacks of Log4J Framework:

- ✓ Reduces some application performance
- ✓ Tightly coupled
- ✓ Bit tough to migrate Log4J framework to other framework

Log4J Framework Features:

- log4j is optimized for speed.
- log4j is based on a named logger hierarchy.
- log4j is fail-stop but not reliable.
not ↗ when ever problem come it stop the execution.
- log4j is ~~not~~ thread-safe.
- log4j is not restricted to a predefined set of facilities.
- Logging behavior can be set at runtime using a configuration file. Configuration files can be property files or in XML format.
- log4j is designed to handle Java Exceptions from the start.
- log4j can direct its output to a file, the console, an java.io.OutputStream, java.io.Writer, a remote server using TCP, a

DURGA SOFTWARE SOLUTIONS
Tools Material

remote Unix Syslog daemon, to a remote listener using JMS, to the NT EventLog or even send e-mail.

- log4j uses 5 levels, namely DEBUG, INFO, WARN, ERROR and FATAL.
- The format of the log output can be easily changed by extending the Layout class.
- The target of the log output as well as the writing strategy can be altered by implementations of the Appender interface.
- log4j supports multiple output appenders per logger.
- log4j supports internationalization.

NOTE:-

To overcome this tightly couple problem, Spring Framework has introduced AOP (Aspect Oriented Programming). In AOP, we can do logging dynamically and declaratively at runtime.

That means Logging component won't touch our application program.

Log4J supports the following logging levels

ALL
DEBUG
INFO
WARN
ERROR
FATAL
OFF

→For normal log levels in Log4J Framework, the ascending log levels are.

DEBUG
INFO
WARN
ERROR
FATAL

Log4J Logging levels

Level	Description
FATAL	Severe errors that cause premature termination. Expect

DURGA SOFTWARE SOLUTIONS
Tools Material

	these to be immediately visible on a status console.
ERROR	Other runtime errors or unexpected conditions. Expect these to be immediately visible on a status console.
WARNING	Use of deprecated APIs, poor use of API, near errors, other runtime situations that are undesirable or unexpected, but not necessarily "wrong". Expect these to be immediately visible on a status console.
INFO	Interesting runtime events (startup/shutdown). Expect these to be immediately visible on a console, so be conservative and keep to a minimum.
DEBUG	detailed information on the flow through the system. Expect these to be written to logs only.
OFF	Switch off Logging completely Or Turns Off all logging Temporarily off the logging messages.
ALL	Supports all Logging levels. Turns ON all logging

NOTE:-

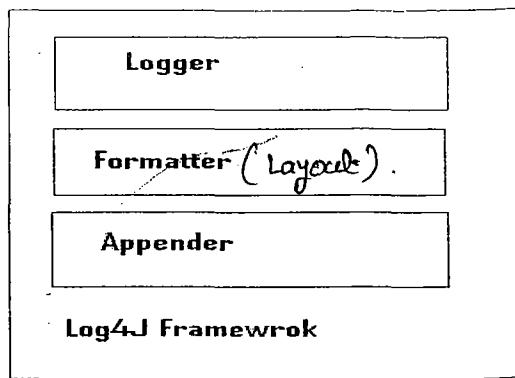
Initially, Apache Software foundation has developed Commons Logging API.

Log4J Framework internally uses Apache Commons Logging which is available in commons-logging-x.x.jar file.

Official web site for Apache Log4J Framework

DURGA SOFTWARE SOLUTIONS
Tools Material

<http://logging.apache.org/log4j/1.2/download.html>



Log4J Framework Architecture:

Log4J Framework is broken into three major parts

- ✓ Logger
- ✓ Formatter
- ✓ Handler (Appender)

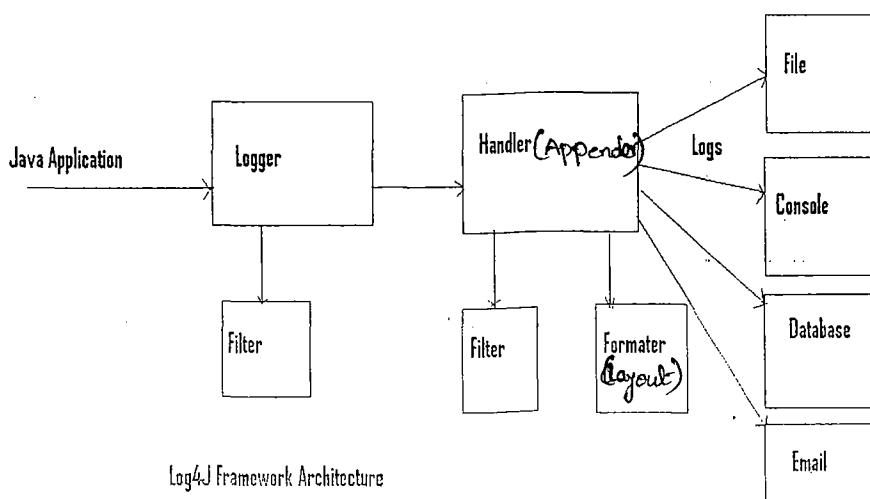
The Logger is responsible for capturing the message to be logged along with certain metadata and passing it to the logging framework.

After receiving the message, the framework calls the Formatter with the message.

The Formatter accepts the message object and formats it for output.

The framework then hands the formatted message to the appropriate Appender for disposition.

This might include a console display, writing to disk, appending to a database, or email.



Advantages of Log4j:

- Simpler log level hierarchy and log methods

DURGA SOFTWARE SOLUTIONS

Tools Material

- More handlers
- Mighty formatters
- Optimized logging costs
- Simpler log level hierarchy:

Steps to integrate Log4J Framework with Java Applications

- a) Download log4j-x.x.jar file and add it to your application classpath
- b) As per your project requirements, configure logging details in log4j.properties or log4j.xml file
- c) Use Logger to log messages in your application Java Programs

Configuration File Examples

- a) log4j.properties
- b) log4j.xml

Simple log4j.properties example

```
log4j.rootLogger=info, way2it
log4j.appender.way2it=org.apache.log4j.ConsoleAppender
log4j.appender.way2it.layout=org.apache.log4j.PatternLayout
log4j.appender.way2it.layout.conversionPattern=%d{yyyy-MM-dd
HH:mm:ss,SSS} %-5p - %m%n
```

- ✓ ConsoleAppender to display logs to a console
- ✓ PatternLayout
- ✓ Conversion
- ✓ Pattern

It is using the following conversion pattern

%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p - %m%n

```
log4j.rootLogger=INFO,C,F
#log4j.rootLogger=DEBUG,F

log4j.appenders.C=org.apache.log4j.ConsoleAppender
log4j.appenders.C.layout=org.apache.log4j.PatternLayout

log4j.appenders.F=org.apache.log4j.FileAppender
log4j.appenders.F.layout=org.apache.log4j.HTMLLayout
log4j.appenders.F.file=banking.html
log4j.appenders.F.layout.conversionPattern=%d{yyyy-MM-dd
HH:mm:ss,SSS} %-5p - %m%n
```

DURGA SOFTWARE SOLUTIONS
Tools Material

B) Two Appenders

ConsoleAppender
FileAppender

C) Conversion pattern

```
log4j.rootLogger=INFO,C,F
#log4j.rootLogger=DEBUG,F
log4j.appender.C=org.apache.log4j.ConsoleAppender
log4j.appender.C.layout=org.apache.log4j.PatternLayout
log4j.appender.F=org.apache.log4j.RollingFileAppender
log4j.appender.F.layout=org.apache.log4j.HTMLLayout
log4j.appender.F.file=banking.html
log4j.appender.F.layout.conversionPattern=%d{yyyy-MM-dd
HH:mm:ss,SSS} %-5p - %m%n
#Default is true
log4j.appender.F.append=true
```

Here it is using RollingFileAppender Appender to log data into a log file.

What is the major difference between FileAppender and RollingFileAppender?

FileAppender:

Whenever we log data to a file, it overwrites the existing log data with new log data.

That means first it cleans the existing log data and add new data

RollingFileAppender:

Whenever we log data to a file, it DOES NOT overwrite the existing log data with new log data.

That means, it simply appends the new logging data to the existing log data.

```
log4j.rootLogger=INFO,C,F
#log4j.rootLogger=DEBUG,F

log4j.appender.C=org.apache.log4j.ConsoleAppender
log4j.appender.C.layout=org.apache.log4j.PatternLayout

log4j.appender.F=org.apache.log4j.RollingFileAppender
log4j.appender.F.layout=org.apache.log4j.HTMLLayout
log4j.appender.F.file=banking.html
log4j.appender.F.layout.conversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS}
%-5p - %m%n
log4j.appender.F.MaxFileSize=10KB
# Keep one backup file
log4j.appender.R.MaxBackupIndex=1
#Default is true
log4j.appender.F.append=true
```

DURGA SOFTWARE SOLUTIONS
Tools Material

**In this example, we are using new properties like file size and file backup
log4j.appenders.F.MaxFileSize=5MB**

It defines the maximum limit of file size. Here our file size is 5MB. Once it reaches, it moves the

Entire log data into backup file and creates new empty log file.

log4j.appenders.R.MaxBackupIndex=1

It specifies how many backup files we want to use to store log data.

log4j.xml example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration PUBLIC "-//LOGGER"
"http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/xml/doc-
files/log4j.dtd">
<log4j:configuration>
<!--
    an appender is an output destination, such as the console or a file;
    names of appenders are arbitrarily chosen
-->
<appender name="stdout" class="org.apache.log4j.ConsoleAppender">
    <layout class="org.apache.log4j.PatternLayout">
        <param name="ConversionPattern"
            value="%d{ABSOLUTE} %5p %c{1}:%L - %m%n" />
    </layout>
</appender>
<!--
    loggers of category 'org.springframework' will only log messages of level "info"
or higher;
    if you retrieve Loggers by using the class name (e.g.
Logger.getLogger(AClass.class))
        and if AClass is part of the org.springframework package, it will belong to this
category
-->
<logger name="org.springframework">
    <level value="info"/>
</logger>
<!--
    everything of spring was set to "info" but for class
    PropertyEditorRegistrySupport we want "debug" logging
-->
<logger name="org.springframework.beans.PropertyEditorRegistrySupport">
    <level value="debug"/>
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
</logger>
<logger name="org.acegisecurity">
    <level value="info"/>
</logger>
<!-- the root category -->
<root>
    <!--
        all log messages of level "debug" or higher will be logged, unless defined
        otherwise
        all log messages will be logged to the appender "stdout", unless defined
        otherwise
    -->
    <level value="debug" />
    <appender-ref ref="stdout" />
</root>
</log4j:configuration>
```

TTCC

TTCC stands for Time Thread Category Component

TTCC is a message format used by log4j. It uses the following pattern:

%r [%t] %-5p %c %x - %m%n

Mnemonic	Description
%r	Used to output the number of milliseconds elapsed from the construction of the layout until the creation of the logging event.
%t	Used to output the name of the thread that generated the logging event.
%p	Used to output the priority of the logging event.
%c	Used to output the category of the logging event.
%x	Used to output the NDC (Nested Diagnostic Context) associated with the thread that generated the logging

DURGA SOFTWARE SOLUTIONS
Tools Material

	event.
%m	Used to output the application supplied message associated with the logging event.
%n	Used to output the platform-specific newline character or characters.
%d	To display current date and time

Note:-

Log4j has been ported by independent authors to C, C++, Python, Ruby, Eiffel and the much maligned C#.

Log4J Examples:

Example1)

```
package com.way2it;
import org.apache.log4j.Logger;
public class HelloWorld
{
    static Logger logger = Logger.getLogger("com.way2it.HelloWorld");
    static public void main(String[] args)
    {
        logger.debug("Hello world Log.");
    }
}
```

OR

```
package com.way2it;
import org.apache.log4j.Logger;
public class HelloWorld
{
    static Logger logger = Logger.getLogger(HelloWorld.class);
    static public void main(String[] args)
    {
        logger.debug("Hello world Log.");
    }
}
```

Can we use lower case logging levels in log4j.properties file?

Yes we can use

#Log4j properties

log4j.rootLogger=debug,bank

OR

DURGA SOFTWARE SOLUTIONS
Tools Material

```
#Log4j properties
log4j.rootLogger=DEBUG,bank
```

Do we need to use same the following properties file for Log4J or we can use different file names?

a) log4j.properties

b) log4j.xml

Because Log4J API contains org.apache.log4j .LogManager and defines these two files as show below.

```
package org.apache.log4j;
public class LogManager
{
    public static final String DEFAULT_CONFIGURATION_FILE =
"log4j.properties";
    public static final String DEFAULT_XML_CONFIGURATION_FILE =
"log4j.xml";
}
```

If we don't use one of these files, then we cant get output. We can observe some warning messages.

```
import org.apache.log4j.Logger;
public class Sample
{
    static Logger log = Logger.getLogger(Sample.class);
    public static void main(String[] args)
    {
        log.info("Hello 1");
        log.info("Hello 2");
        log.info("Hello 3");
    }
}
```

Output:

```
log4j:WARN No appenders could be found for logger (Sample).
```

```
log4j:WARN Please initialize the log4j system properly.
```

If we don't want to use log4j.properties or log4j.xml file, then we can use BasicConfigurator to

get the log messages with default format.

Default implementation of BasicConfigurator

```
package org.apache.log4j;
public class BasicConfigurator
{
    public static void configure()
    {
        Logger root = Logger.getRootLogger();
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
root.addAppender(new ConsoleAppender(new PatternLayout("%r  
[%t] %p %c %x - %m%n")));
}
public static void configure(Appender appender)
{
    Logger root = Logger.getRootLogger();
    root.addAppender(appender);
}
public static void resetConfiguration()
{
    LogManager.resetConfiguration();
}
}
```

By Default, BasicConfigurator is connected to ConsoleAppender and with basic pattern

%r [%t] %p %c %x - %m%n

The output contains

- ✓ Relative time-the number of milliseconds that elapsed since the start of the program until the invocation of the logging request,
- ✓ The name of the invoking thread between brackets
- ✓ The level of the request
- ✓ The logger name
- ✓ Finally the message.

Example:

```
import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.Logger;
public class Sample
{
    static Logger log = Logger.getLogger(Sample.class);
    public static void main(String[] args)
    {
        BasicConfigurator.configure();
        log.info("Hello 1");
        log.info("Hello 2");
        log.info("Hello 3");
    }
}
```

Output:

```
0 [main] INFO Sample - Hello 1
0 [main] INFO Sample - Hello 2
```

DURGA SOFTWARE SOLUTIONS
Tools Material

0 [main] INFO Sample - Hello 3
%r [%t] %p %c %x - %m%n

NOTE:-

BasicConfigurator.configure() being the simplest but also the least flexible.

For each Log4J logging levels, Logger class contains respective methods.

Logger class contains the following methods

debug() – DEBUG level

info() – INFO level

warn() – WARN level

error() – ERROR level

fatal() – FATAL level

Ordering of Log4J Log levels:

ALL < DEBUG < INFO < WARN < ERROR < FATAL < OFF.

If we want stop logs in Production or LIVE systems, how do it in Log4J Framework?

Changing logging level from existing value to OFF in log4j.properties file

log4j.rootLogger=OFF,C,F

log4j.appender.C=org.apache.log4j.ConsoleAppender

log4j.appender.C.layout=org.apache.log4j.PatternLayout

log4j.appender.C.layout.conversionPattern=%d{yyyy-MM-dd}

HH:mm:ss,SSS} %-5p - %m%n

log4j.rootLogger=OFF,C,F → this OFF value stop logs

RootLogger:

What is RootLogger? What is the importance of it?

RootLogger is root object for all remaining application logs.

It contains default log level. If you don't assign any log level to your application logger, then they inherit RootLogger log level or it's immediate root logger automatically.

Examples:

Logger name	Assigned level	Effective level
root	DEBUG	DEBUG
x	none	DEBUG
x.y	none	DEBUG
x.y.z	none	DEBUG

Logger name	Assigned level	Effective level
-------------	----------------	-----------------

DURGA SOFTWARE SOLUTIONS
Tools Material

root	DEBUG	DEBUG
x	ERROR	ERROR
x.y	INFO	INFO
x.y.z	INFO	INFO

Logger name	Assigned level	Effective level
root	DEBUG	DEBUG
x	ERROR	ERROR
x.y	none	ERROR
x.y.z	INFO	INFO

Logger name	Assigned level	Effective level
root	DEBUG	DEBUG
x	ERROR	ERROR
x.y	INFO	INFO
x.y.z	none	INFO

BasicConfigurator.configure()

It is equal to the following log4j.properties file

```
# Set root logger level to DEBUG and add an appender called A1.
log4j.rootLogger=DEBUG, A1
# A1 is set to be a ConsoleAppender.
log4j.appender.A1=org.apache.log4j.ConsoleAppender
# A1 uses PatternLayout.
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%-4r [%t] %-5p %c %x
%-om%n
```

PropertyConfigurator

If we want to configure properties programmatically, we can do it using

PropertyConfigurator class available in org.apache.log4j file.

Example:

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Properties;
import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
public class Sample
{
    static Logger log = Logger.getLogger(Sample.class);
    public static void main(String[] args)
        throws FileNotFoundException, IOException
    {
        String file = "mylog.properties";
        Properties logProperties = new Properties();
        logProperties.load(new FileInputStream(file));
        PropertyConfigurator.configure(logProperties);
        //BasicConfigurator.configure();
        PropertyConfigurator.configure(logProperties);
        log.debug("debug 1");
        log.info("info 2");
        log.error("eror 3");
    }
}
```

How to define different logging levels for different Project Modules?

Let us assume that we have the following two modules in our Banking application

- a) CreditCard Module
- b) DebitCard Module

log4j.properties

```
# CreditCardFileAppender - used to log messages in the creditcard.log file.
log4j.appender.CreditCardFileAppender=org.apache.log4j.FileAppender
log4j.appender.CreditCardFileAppender.File=creditcard.log
log4j.appender.CreditCardFileAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.CreditCardFileAppender.layout.ConversionPattern= %-4r [%t] %-5p %c %x - %m%n
# DebitCardFileAppender - used to log messages in the debitcard.log file.
log4j.appender.DebitCardFileAppender=org.apache.log4j.FileAppender
log4j.appender.DebitCardFileAppender.File=debitcard.log
log4j.appender.DebitCardFileAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.DebitCardFileAppender.layout.ConversionPattern= %-4r [%t] %-5p %c %x - %m%n
log4j.logger.com.way2it.creditcard=WARN,CreditCardFileAppender
log4j.logger.com.way2it.debitcard=DEBUG,DebitCardFileAppender
```

Project Modules

Java Programs

```
package com.way2it.creditcard;
import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;
import com.way2it.debitcard.DebitCardModule;
public class CreditCardModule
{
    static Logger logger = Logger.getLogger(CreditCardModule.class);
    public static void main(String[] args)
    {
        PropertyConfigurator.configure("log4j.properties");
        logger.debug("Sample debug message");
        logger.info("Sample info message");
        logger.warn("Sample warn message");
        logger.error("Sample error message");
        logger.fatal("Sample fatal message");
        SampleReport obj = new SampleReport();
        obj.generateReport();
    }
}
package com.way2it.debitcard;
import org.apache.log4j.Logger;
public class DebitCardModule
{
    static          Logger          logger      =
    Logger.getLogger(DebitCardModule.class);
    public void generateReport()
    {
        logger.debug("Sample debug message");
        logger.info("Sample info message");
        logger.warn("Sample warn message");
        logger.error("Sample error message");
        logger.fatal("Sample fatal message");
    }
}
```

Output:

After executing the program, the contents of **creditcard.log** file.

[main] WARN com.way2it.creditcard.CreditCardModule - Sample warn message

[main] ERROR com.way2it.creditcard.CreditCardModule - Sample error message

[main] FATAL com.way2it.creditcard.CreditCardModule - Sample fatal message

The contents of **debitcard.log** file.

DURGA SOFTWARE SOLUTIONS
Tools Material

```
[main] DEBUG com.way2it.debitcard.DebitCardModule - Sample debug message
[main] INFO com.way2it.debitcard.DebitCardModule Sample info message
[main] WARN com.way2it.debitcard.DebitCardModule - Sample warn message
[main] ERROR com.way2it.debitcard.DebitCardModule - Sample error message
[main] FATAL com.way2it.debitcard.DebitCardModule - Sample fatal message
```

How to configure log4j.xml file programmatically?

log4j.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j='http://jakarta.apache.org/log4j/'>
    <appender name="way2it"
        class="org.apache.log4j.ConsoleAppender">
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%-4r [%t] %-5p %c
%x - %m%on" />
        </layout>
    </appender>
    <root>
        <level value="debug" />
        <appender-ref ref=" way2it " />
    </root>
</log4j:configuration>
```

HelloWorld.java

```
package com.way2it.helloworld;
import org.apache.log4j.Logger;
import org.apache.log4j.xml.DOMConfigurator;
public class HelloWorld
{
    static Logger logger = Logger.getLogger(HelloWorld.class);
    public static void main(String[] args)
    {
        DOMConfigurator.configure("log4j.xml");
        logger.debug("Sample debug message");
        logger.info("Sample info message");
        logger.warn("Sample warn message");
        logger.error("Sample error message");
        logger.fatal("Sample fatal message");
    }
}
```

DURGA SOFTWARE SOLUTIONS
Tools Material

Output:

```
[main] DEBUG com.way2it.helloworld.HelloWorld - Sample debug message
[main] INFO com.way2it.helloworld.HelloWorld - Sample info message
[main] WARN com.way2it.helloworld.HelloWorld - Sample warn message
[main] ERROR com.way2it.helloworld.HelloWorld - Sample error message
[main] FATAL com.way2it.helloworld.HelloWorld - Sample fatal message
```

Top Eleven tips on logging in Java

- 1) Use isDebugEnabled() for putting debug log in Java , it will save lot of string concatenation activity if your code run in production environment with production logging level instead of DEBUG logging level.
- 2) Carefully choose which kind of message should go to which level for logging in Java, It become extremely important if you are writing server application in core java and only way to see what happen is java logs. If you log too much information your performance will be affected and same time if you don't log important information like incoming messages and outgoing messages in java logs then it would become extremely difficult to identify what happened in case of any issue or error because nothing would be in java logs.
- 3) Use either log4j or java.util.logging for logging in Java, I would recommend log4j because I have used it a lot and found it very flexible. It allows changing logging level in java without restarting your application which is very important in production or controlled environment. To do this you can have log4j watchdog which continuously look for log4j.xml in a particular directory and if finds loads it and reset logging in java.
- 4) By using log4j.xml you can have different logger configuration for different java classes as well. You can have some classes in INFO mode, some in WARN mode or ERROR mode. It's quite flexible to do this to customize java logging.
- 5) Another important point to remember is format of java logging, this you specify in logger. Properties file in case of java.util.logging API for logging to use which java logging Formatter. Don't forget to include Thread Name and fully qualified java class Name while printing logs because it would be impossible to find sequence of events if your code is executed by multiple threads without having thread name on it. In my opinion this is the most important tips you consider for logging in Java.

DURGA SOFTWARE SOLUTIONS
Tools Material

- 6) By carefully choosing format of java logging at logger level and format of writing log you can have generate reports from your java log files. Be consistent while logging messages, be informative while logging message, print data with message wherever required.
- 7) While writing message for logging in Java try to use some kind of prefix to indicate which part of your code is printing log e.g. client side, Database site or session side, later you can use this prefix to do a "grep" or "find" in Unix and have related logs at once place. Believe me I have used this technique and it helped a lot while debugging or investigating any issues and your log file is quite large. For example you can put all Database level log with a prefix "DB_LOG:" and put all session level log with prefix "SESSION_LOG:"
- 8) If a given logger is not assigned a level, then it inherits one from its closest ancestor. That's why we always assign log level to root logger in configuration file log4j.rootLogger=DEBUG.
- 9) Both no logging and excessive logging is bad so carefully planned what to log and on which level you log that messages so that you can run fast in production environment and at same time able to identify any issue in QA and TEST environment.
- 10) I found that for improving logging its important you look through your log and monitor your log by yourself and tune it wherever necessary. It's also important to log in simple English and it should make sense and human readable since support team may wan to put alert on some logging message and they may want to monitory your application in production.
- 11) if you are using SLFJ for logging in java use parametrized version of various log methods they are faster as compared to normal method.
`logger.debug("No of Orders " + noOfOrder + " for client : " + client);
// slower
logger.debug("No of Executions {} for clients:{}", noOfOrder , client);
// faster`

Nested Diagnostic Contexts

Uses Stack per user

Most real-world systems have to deal with multiple clients simultaneously. In a typical multithreaded implementation of such a system, different threads will handle different clients. Logging is especially well suited to trace and debug complex distributed applications. A common approach to differentiate the logging output of one client from another is to instantiate a new separate logger for each client. This promotes the proliferation of loggers and increases the management overhead of logging.

DURGA SOFTWARE SOLUTIONS

Tools Material

To uniquely stamp each request, the user pushes contextual information into the NDC, the abbreviation of Nested Diagnostic Context. The NDC class is shown below.

```
public class NDC {  
    // Used when printing the diagnostic  
    public static String get();  
    // Remove the top of the context from the NDC.  
    public static String pop();  
    // Add diagnostic context for the current thread.  
    public static void push(String message);  
    // Remove the diagnostic context for this thread.  
    public static void remove();  
}
```

The NDC is managed per thread as a **stack** of contextual information.

Log4J Thread-Safe:

Is Log4J Framework Thread-safe? Why Log4J has lot of performance issues?

Yes Log4j is thread-safe.

It's not just thread-safe, it uses synchronized methods everywhere. That means that for heavy logging log4j will induce performance bottleneck and possible deadlocks for EJBs.

How and when do loggers inherit level?

Following 3 rules could be applied:

- Root logger always has an assigned level.
- A logger which does not have an explicit log level specified inherits the parent's log level.
- If the parent logger is not initialized then the parent's parent is checked and so on, until the root logger.

So , in effect, every logger which does not have an explicit level, will inherit from its parent. The parent may in turn inherit for its own parent, if no explicit logging was specified for the parent, and so on, until the root logger, which is guaranteed to have a log level.

Where can i download log4j?

We can download Log4J Framework at the following log4j website.

<http://logging.apache.org/log4j/index.html>

What is the root logger? How to get it?

It is the root of all logging classes in the log4j architecture. It always exists and applications can retrieve it using LogManager.getLogger() API.

LogManager.getLogger()

How do we know which log requests sent to the logger will be logged?

DURGA SOFTWARE SOLUTIONS
Tools Material

A log request is said to be enabled (and logged), if the log level of the request is higher than or equal to the log level of the logger.

IF LOG.LEVEL >= LOGGER.LEVEL, then the LOG is assumed enabled.

For normal log levels in log4j the ascending log levels are DEBUG, INFO, WARN, ERROR and FATAL

Also to filter just the messages with a particular level, without the allowing the higher or lower levels to be logged is done using LevelMatchFilter

How many loggers of the same name can exist?

Only 1 logger with a specified name can exist at a time.

Multiple requests to getLogger API for a same logger name, returns exact same reference for each of the requests.

SLF4J

What is SLF4J? Why and when do we need to use SLF4J?

SLF4J Stands for Simple Logging Facade for Java

The Simple Logging Facade for Java or (SLF4J) serves as a simple facade or abstraction for various logging frameworks, e.g. java.util.logging, log4j and logback, allowing the end user to plug in the desired logging framework at deployment time.

Similarities and Differences with Log4j

Five of Log4j's six logging levels are used. FATAL has been dropped on the basis that inside the logging framework is not the place to decide when an application should terminate and therefore there is no difference between ERROR and FATAL from the logger's point of view.

Logger instances are created via the LoggerFactory, which is very similar in Log4j. For example,

```
private static final Logger LOG =  
    LoggerFactory.getLogger(Wombat.class);
```

In Logger, the logging methods are overloaded with forms that accept one, two or more values.[1] Occurrences of the simple pattern {} in the log message is replaced in turn with the values. This is simple to use yet provides a performance benefit when the values have expensive toString() methods. When logging is disabled at the DEBUG level, the logging framework does not need to evaluate the string representation of the values. In the following example, the values count or userAccountList only need to be evaluated when DEBUG is enabled; otherwise the overhead of the debug call is trivial.

```
LOG.debug("There are now " + count + " user accounts: " +  
    userAccountList); // slow
```

```
LOG.debug("There are now {} user accounts: {}", count,  
    userAccountList); // faster
```

DURGA SOFTWARE SOLUTIONS
Tools Material

Similar methods exist in Logger for isDebugEnabled() etc. to allow more complex logging calls to be wrapped so that they are disabled when the corresponding level is disabled, avoiding unnecessary processing.

Unlike Log4j, SLF4J offers logging methods that accept markers. These are special objects that enrich the log messages and are an idea that SLF4J has borrowed from logback.

FAQs

Log4J Interview Questions

Why we need logging in Java?

What are different logging levels in Java?

How to choose correct logging level in java?

How incorrect java logging affect performance?

What are different logging frameworks available for Java?

What is Log4J?

Why most of the people are using Log4J in their applications.

Advantages of Log4J Framework?

Drawbacks of Log4J Framework?

What is PropertyConfigurator?

How to use different log4j properties file name?

Is log4j.properties file mandatory?

How to use multiple log files?

What is ConsoleAppender? What is the use of it?

What is conversion pattern?

What is the meaning of the following conversion pattern?

%-4r [%t] %-5p %c %x - %m%n

How and where to specify log file size?

What is the default logging level for Log4J?

Who is the founder of Log4J Framework?

What is the root element of log4j.xml file?

What is FileAppender? What is the use of it?

What is RollingFileAppender? What is the use of it?

What are the major differences between FileAppender and RollingFileAppender?

How and where to define log file appending option?

How to define package level loggings?

What is the xml element to define conversion pattern in log4j.xml file?

How to input log4j.properties file to our java class in programmatic approach?

Drawbacks of Log4J programmatic approaches?

Advantages of Log4J Declarative approaches?

DURGA SOFTWARE SOLUTIONS
Tools Material

Why most of the people uses log4j.properties file for logging in their application?

What is DOMConfigurator?

When do we need to use DOMConfigurator?

How to input log4j.xml file to our java class in programmatic approach?

Which one is more preferable between log4j.properties and log4j.xml file?

Can we use lower case logging levels in log4j.properties file?

Is Log4J Thread-safe? Why?

How to solve Performance issues of Log4J?

How and when do loggers inherit level?

Junit

Testing:

Testing is nothing but checking the expected results with actual results.

TestCase:

TestCase is a TestPlan where expected results will be compared with actual results with specific input values. To test one functionality it is recommended to write more test cases with both wrong data and ~~right~~ data.

Unit Testing

The testing performed by the programmer on his own piece of code is called as UnitTesting

PeerTesting

The unit testing performed by the programmer on other programmers code is called PeerTesting.

- ✓ Unit Testing, Peer Testing are the responsibility of the programmer .The remaining testing 's like Integration Testing,Performance Testing ,Navigation Testing and etc are the responsibility of QA department.

TestSuite

TestSuite provides the environment to run all the TestCases.

What is Junit:

- ✓ **JUnit** is a program used to perform **unit testing** of virtually any software. JUnit testing is accomplished by writing test cases using Java, compiling these test cases and running the resultant classes with a JUnit Test Runner.
- ✓ I will explain a little bit about software and unit testing in general. What and how much to test depends on how important it is to know that the tested software works right, in relation to how much time you are willing to dedicate to testing. Since you are reading this, then for some reason you have decided to dedicate at least some time to unit testing.

Example

DURGA SOFTWARE SOLUTIONS
Tools Material

- ✓ I'm currently developing an SMTP server. An SMTP server needs to handle email addresses of the format specified in RFC documents. To be confident that my SMTP server complies with the RFC, I need to write a test case for each allowable email address type specified in the RFC. If the RFC says that the character "#" is prohibited, then I should write a test case that sends an address with "#" and verifies that the SMTP server rejects it. You don't need to have a formal specification document to work from. If I wasn't aiming for *compliance*, I could just decide that it's good enough if my server accepts email addresses consisting only of letters, numbers and "@", without caring whether other addresses succeed or fail.
- ✓ Another important point to understand is that, everything is better if you make your tests before you implement the features. For my SMTP server, I did some prototyping first to make sure that my design will work, but I did not attempt to satisfy my requirements with that prototype. I am now writing up my test cases and running them against existing SMTP server implementations (like Sendmail). When I get to the implementation stage, my work will be extremely well defined. Once my implementation satisfies all of the unit tests, just like Sendmail does, then my implementation will be completed. At that point, I'll start working on new requirements to surpass the abilities of Sendmail and will work up those tests as much as possible before starting the second implementation iteration. (If you don't have an available test target, then there sometimes comes a point where the work involved in making a dummy test target isn't worth the trouble... so you just develop the test cases as far as you can until your application is ready).
- ✓ Everything I've said so far has to do with unit testing, not JUnit specifically. Now on to JUnit...
- ✓ JUnit was originally written by **Erich Gamma and Kent Beck**.
- ✓

DURGA SOFTWARE SOLUTIONS
Tools Material

Installing JUnit:

✓ **Downloading :**

You can download JUnit 3.x from <http://www.junit.org/index.htm> in the zipped format.

✓ **Installation :**

Below are the installation steps for installing JUnit:

1. Unzip the JUnit3.x.zip file.
2. Add **junit-3.x.jar** to the CLASSPATH. or we can create a bat file "**setup.bat**" in which we can write "set CLASSPATH=.;%CLASSPATH%;junit-3.x.jar;". So whenever we need to use JUnit in our project then we can run this setup.bat file to set the classpath.

Test Coverage -- quantity of test cases

- ✓ With respect to test coverage, an ideal test would have test cases for every conceivable variation type of input, *not every possible instance of input*. You should have test cases for combinations or permutations of all variation types of the implemented operations. You use your knowledge of the method implementation (including possible implementation changes which you want to allow for) to narrow the quantity of test cases way down.

Example:

- ✓ I'll discuss testing the multiplication method of a calculator class as an example. First off, if your multiplication method hands its parameters (the multiplication operands) exactly the same regardless of order (now and in the future), then you have just dramatically cut your work from covering all permutations to covering all combinations
- ✓ You will need to cover behavior of multiplying by zero, but it will take only a few test cases for this. It is extremely unlikely that the multiplication method code does anything differently for an operand value of 2 different than it would for an

DURGA SOFTWARE SOLUTIONS
Tools Material

operand value of 3, therefore, there is no reason to have a test for "2 x 0" and for "3 x 0".

- ✓ All you need is one test with a positive integer operand and zero, like "46213 x 0". (Two test cases if your implementation is dependent upon input parameter sequence). You may or may not need additional cases to test other types of non-zero operands, and for the case of "0 x 0". Whether a test case is needed just depends upon whether your current and future code could ever behave differently for that case.

TestExpressions

- ✓ The most important thing is to write tests like

```
(expectedResult == obtainedResult)
```

Or

```
(expectedResult.equals(obtainedResult))
```

- ✓ You can do that without JUnit, of course. For the purpose of this document, I will call the smallest unit of testing, like the expression above, a *test expression*. All that JUnit does is give you flexibility in grouping your test expressions and convenient ways to capture and/or summarize test failures.

Class junit.framework.Assert

- ✓ The Assert class has a bunch of static convenience methods to help you with individual test expressions. For example, without JUnit I might code

```
if (obtainedResult == null || !expectedResult.equals(obtainedResult))  
  
    throw new MyTestException("Bad output for # attempt");
```

- ✓ With JUnit, you can code

```
Assert.assertEquals("Message for  
    first attempt", ID, obtainedID);
```

DURGA SOFTWARE SOLUTIONS

Tools Material

- ✓ There are lots of assertXxx() methods to take care of the several common Java types. They take care of checking for nulls. Both assert() failures and failure()s (which are effectively the same thing) throw junit.framework.AssertionFailedError Throwables, which the test-runners and listeners know what to do with. If a test expression fails, the containing test method quits (by virtue of throwing the AssertionFailedError internally, then the test-runner performs cleanup before executing the next test method in the sequence (with the error being noted for reporting now or later). Take a look at the API spec for junit.framework.Assert.

JUnit Failures vs. Errors

- ✓ JUnit test reports differentiate failures vs. errors. A **failure** is a test which your code has explicitly failed by using the mechanisms for that purpose (as described above). Generation of a failure indicates that your time investment is paying off-- it points you right to an anticipated problem in the program that is the target of your testing.
- ✓ A JUnit error, on the other hand, indicates an unanticipated problem. It is either a resource problem such as is normally the domain of Java unchecked throwables or it is a problem with your implementation of the test. When you run a test and get an error, it means you really need to fix something, and usually not just the program that you intended to test. Either a problem has occurred outside of your test method (like in test class instantiation, or the test setup methods described in following chapters), or your test method has thrown an exception.

Note:

With some 3.x versions of JUnit, there are/were lifecycle side-effects when Errors were produced. Unless you know otherwise, don't assume that cleanup methods will run after test failure. (This is not a concern with JUnit 4.x, which is reliable in this respect).

Interface junit.framework.Test

DURGA SOFTWARE SOLUTIONS
Tools Material

- ✓ An object that you can run with the JUnit infrastructure is a Test. But you can't just implement Test and run that object. You can only run specially created instances of TestCase. (Running other implementations of Test will compile, but produce runtime errors.)

Abstract class junit.framework.TestCase

It gives own platform to write Testcases.

- ✓ A test case defines the fixture to run multiple tests. To define a test case
 1. implement a subclass of TestCase
 2. define instance variables that store the state of the fixture
 3. initialize the fixture state by overriding setUp() → *Execute once for every Testcase.*
 4. clean-up after a test by overriding tearDown(). → *It also execute once after every Testcase.*
- ✓ setup() method is executed for every test case method execution. In this method we can perform Initialization activity.
- ✓ teardown() method is executed at the end of each Test Case method .In this method we can perform cleanup activity or uninitializtion.
- ✓ Every Test case class must and should extend the TestCase class
- ✓ Every Test case class should follow the following conventions

Coding Convention :

1. Name of the test class must end with "Test".
2. Name of the method must begin with "test".
3. Return type of a test method must be void.
4. Test method must not throw any exception.
5. Test method must not have any parameter

Class junit.framework.TestSuite

- ✓ A TestSuite is just an object that contains an ordered list of runnable Test objects. TestSuites also implement Test() and are runnable. To run a TestSuite is just to run all of the elemental Tests in the specified order, where by elemental tests, I mean Tests for a single method like we used in the Abstract class

DURGA SOFTWARE SOLUTIONS
Tools Material

junit.framework.TestCase .TestSuites can be nested. Remember that for every elemental Test run, three methods are actually invoked, setUp + test method + tearDown.

This is how TestSuites are used.

```
TestSuite s = new TestSuite();
s.addTest(new TC("tcm"));
s.addTest(new TD("tdm"));
s.addTestSuite(AnotherTestSuiteImplementation.class);
s.run(new TestResult());
```

Class junit.textui.TestRunner

A command line based tool to run tests.

```
java junit.textui.TestRunner [-wait] - TestCaseClass
```

TestRunner expects the name of a TestCase class as argument. If this class defines a static suite method it will be invoked and the returned test is run. Otherwise all the methods starting with "test" having no arguments are run.

When the wait command line argument is given TestRunner waits until the users types RETURN.

TestRunner prints a trace as the tests are executed followed by a summary at the end.

Examples

- ✓ If you want to work with Junit you need to write three classes
 1. Main Class(i.e our Actual Logic)
 2. TestCase class
 3. Use TestRunner class to run all TestCase classes

Calc.java

DURGA SOFTWARE SOLUTIONS

Tools Material

```
1 class Calc
2 {
3     public int add(int a,int b)
4     {
5         return a+b;
6     }
7     public int sub(int a,int b)
8     {
9         return a-b;
10    }
11    public int div(int a,int b)
12    {
13        return a/b;
14    }
15
16 }
17
```

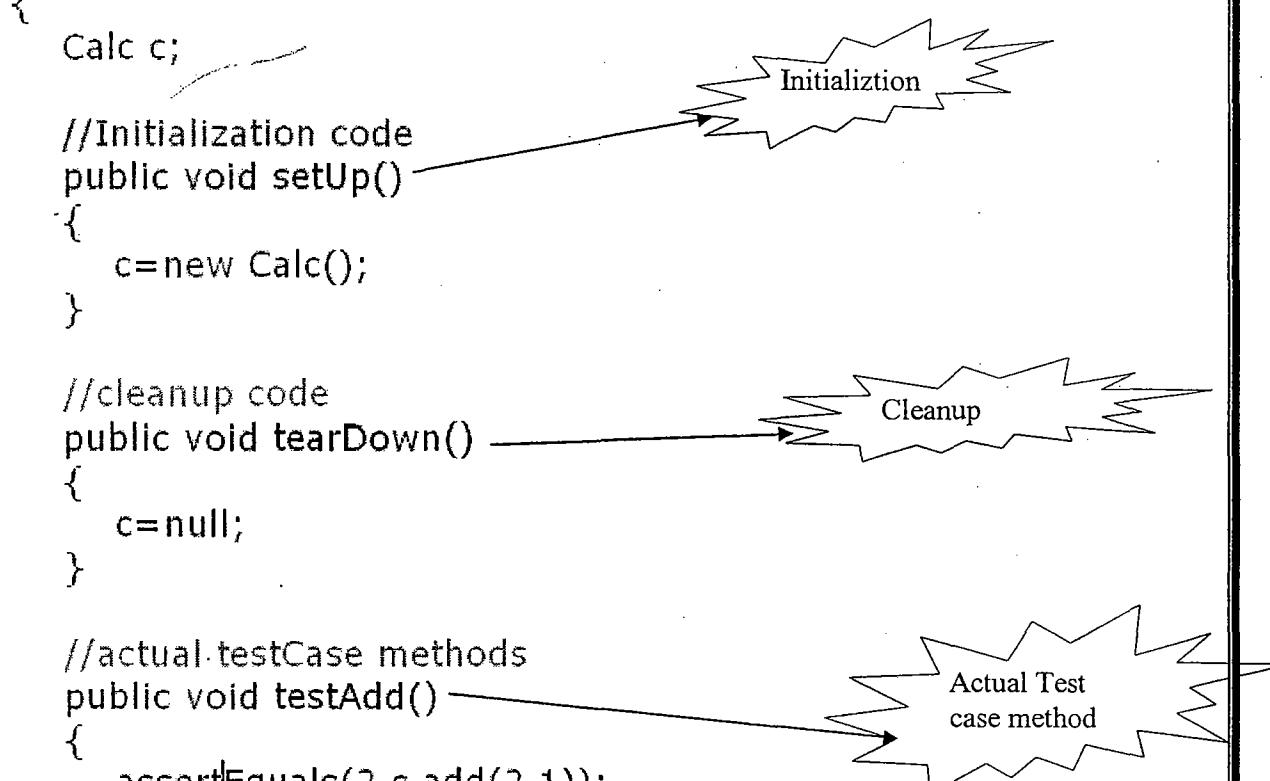
The diagram illustrates the flow of business logic. Three arrows originate from the method definitions in the code: 'add(int a,int b)', 'sub(int a,int b)', and 'div(int a,int b)'. These arrows point towards a central, jagged starburst shape. Inside this starburst, the text 'Business Logic' is written, indicating that these specific methods represent the core business logic of the application.

Write TestCase Class to Test the Functionality

```
import junit.framework.TestCase;
class CalcTest extends TestCase
{
    Calc c;
    //Initialization code
    public void setUp()
    {
        c=new Calc();
    }

    //cleanup code
    public void tearDown()
    {
        c=null;
    }

    //actual testCase methods
    public void testAdd()
    {
        assertEquals(2,c.add(2,1));
    }
}
```



Initialization

Cleanup

Actual Test
case method

**Compile and Running The
TestCases**

```
D:\JUNIT\JUnitEx>set classpath=D:\JUNIT\junit4.10\junit-4.10.jar;..
D:\JUNIT\JUnitEx>javac Calc.java
D:\JUNIT\JUnitEx>javac CalcTest.java
D:\JUNIT\JUnitEx>java junit.textui.TestRunner CalcTest
Time: 0
OK (1 test)

D:\JUNIT\JUnitEx>
```

DURGA SOFTWARE SOLUTIONS
Tools Material

If Test Case Fails

```
D:\JUNIT\JUnitEx>javac CalcTest.java
D:\JUNIT\JUnitEx>java junit.textui.TestRunner CalcTest
.F
Time: 0
There was 1 failure:
1) testAdd(CalcTest)junit.framework.AssertionFailedError: expected:<2> but was:<
3>
    at CalcTest.testAdd(CalcTest.java:21)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.
java:39)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAcces-
sorImpl.java:25)

FAILURES!!!
Tests run: 1, Failures: 1, Errors: 0

D:\JUNIT\JUnitEx>
```

JUnit with Eclipse

Preparation

- ✓ Create a new project "com.dsoft.first". We want to create the unit tests in a separate folder. Create therefore a new source folder "test" via right mouse click on your project, select properties and choose the "Java Build Path". Select the tab source code.
- ✓ Press "Add folder" then then press "Create new folder". Create the folder "test".
 - The creation of an separate folder for the test is not mandatory. But it is good advice to keep the test coding separate from the normal coding.

Create a Java class

Create a package "com.dsoft.first" and the following class.

```
package com.dsoft.first;
public class MyClass {
    public int multiply(int x, int y)
    {
        return x / y;
    }
}
```

Create a JUnit test

- ✓ Select your new class, right mouse click and select New ->JUnit Test case, change the source folder to JUnit. Select "New JUnit 4 test". Make sure you change the source folder to test.
- ✓ Press next and select the methods which you want to test.
- ✓ If you have not yet JUnit in your classpath, Eclipse will asked you if it should be added to the classpath.

Create a test with the following code.

```
package com.dsoft.first;
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
import org.junit.Test;  
import static org.junit.Assert.assertEquals;  
public class MyClassTest {  
    @Test  
    public void testMultiply() {  
        MyClass tester = new MyClass();  
        assertEquals("Result", 50, tester.multiply(10, 5));  
    }  
}
```

Run your test via Eclipse

- ✓ Right click on your new test class and select Run-As-> Junit Test.
- ✓ The test should be failing (indicated via a red bar). This is due to the fact that our multiplier class is currently not working correctly (it does a division instead of multiplication). Fix the bug and re-run test to get a green light.
- ✓ If you have several tests you can combine them into a test suite. All test in this test suite will then be executed if you run the test suite. To create a new test suite, select your test classes, right mouse click-> New-> Other -> JUnit -Test Suite
- ✓ Select next and select the methods you would like to have test created for.

Tip

This does currently not work for JUnit4.0 testcases. See Bug Report

- ✓ Change the coding to the following to make your test suite run your test. If you later develop another test you can add it to @Suite.SuiteClasses

```
package mypackage;  
import org.junit.runner.RunWith;  
import org.junit.runners.Suite;  
@RunWith(Suite.class)  
@Suite.SuiteClasses( { MyClassTest.class } )  
public class AllTests {  
}
```

Run your test via code

- ✓ You can also run your test via your own coding. The class "org.junit.runner.JUnitCore" provides the method runClasses() which allows you to run one or several tests classes. As a return parameter you receive an object of type "org.junit.runner.Result". This object can be used to retrieve information about the tests and provides information about the failed tests.
- ✓ Create in your "test" folder a new class "MyTestRunner" with the following coding. This class will execute your test class and write potential failures to the console.

DURGA SOFTWARE SOLUTIONS
Tools Material

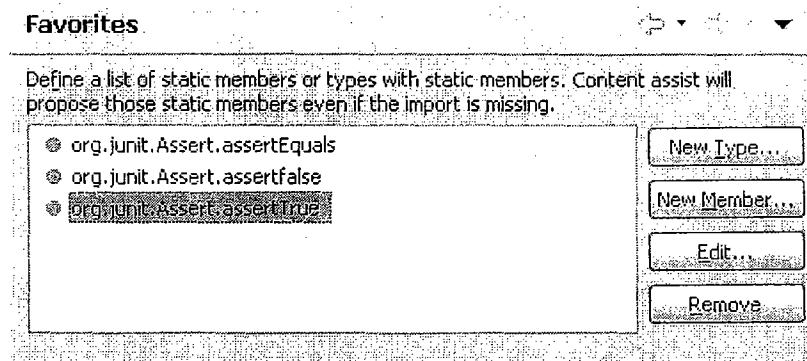
```
package com.dsoft;
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;
public class MyTestRunner {
    public static void main(String[] args) {
        Result result = JUnitCore.runClasses(MyClassTest.class);
        for (Failure failure : result.getFailures()) {
            System.out.println(failure.toString());
        }
    }
}
```

JUnit (more) in Detail

Static imports with Eclipse

JUnit uses a lot of static methods and Eclipse cannot automatically import static imports. You can make the JUnit test methods available via the content assists.

Open the Preferences via Window -> Preferences and select Java > Editor > Content Assist > Favorites. Add then via "New Member" the methods you need. For example this makes the assertTrue, assertFalse and assertEquals method available.



You can now use Content Assist (Ctrl+Space) to add the method and the import.

I suggest to add at least the following new members.

- org.junit.Assert.assertTrue
- org.junit.Assert.assertFalse
- org.junit.Assert.assertEquals
- org.junit.Assert.fail

3.2. Annotations

DURGA SOFTWARE SOLUTIONS
Tools Material

The following give an overview of the available annotations in JUnit 4.x

Table 1. Annotations

Annotation	Description
@Test public void method()	Annotation @Test identifies that this method is a test method.
@Before public void method()	Will perform the method() before each test. This method can prepare the test environment, e.g. read input data, initialize the class)
@After public void method()	Test method must start with test executed at the end of each test . Will perform the method before the start of all tests. This can be used to perform time intensive activities for example be used to connect to a database
@BeforeClass public void method()	Will perform the method after all tests have finished. This can be used to perform clean-up activities for example be used to disconnect to a database
@AfterClass public void method()	Will ignore the test method, e.g. useful if the underlying code has been changed and the test has not yet been adapted or if the runtime of this test is just to long to be included.
@Test(expected=IllegalArgumentException.class)	Tests if the method throws the named exception
@Test(timeout=100)	Fails if the method takes longer than 100 milliseconds

Assert statements

The following gives an overview of the available test methods:

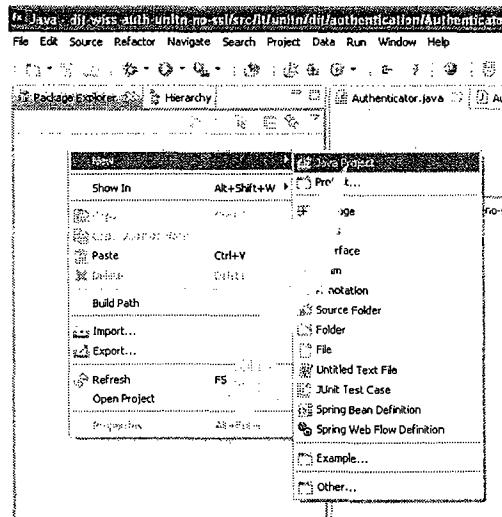
Table 2. Test methods

Statement	Description
fail(String)	Let the method fail, might be usable to check

DURGA SOFTWARE SOLUTIONS
Tools Material

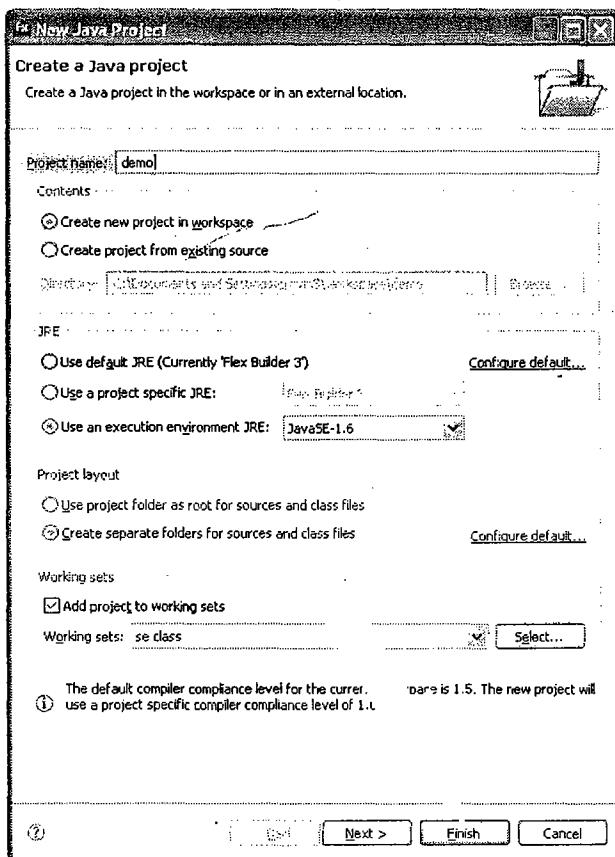
Statement	Description
	that a certain part of the code is not reached.
assertTrue(true);	True
assertEquals([String message], expected, actual)	Test if the values are the same. Note: for arrays the reference is checked not the content of the arrays
assertEquals([String message], expected, actual, tolerance)	Usage for float and double; the tolerance are the number of decimals which must be the same
assertNull([message], object)	Checks if the object is null
assertNotNull([message], object)	Check if the object is not null
assertSame([String], expected, actual)	Check if both variables refer to the same object
assertNotSame([String], expected, actual)	Check that both variables refer not to the same object
assertTrue([message], boolean condition)	Check if the boolean condition is true.

Working with MyEclipse



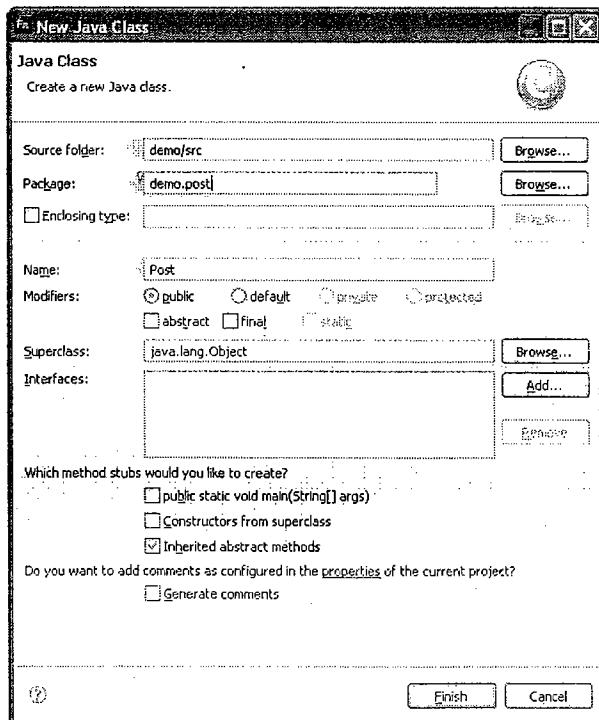
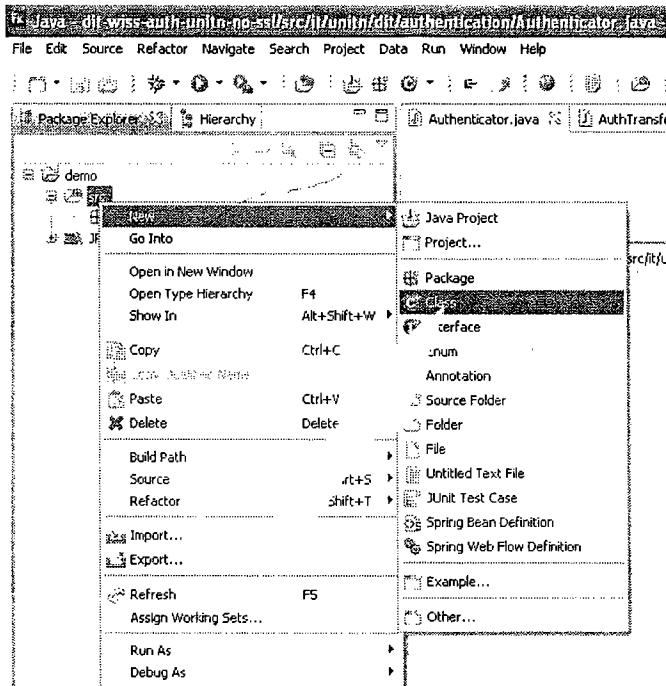
Create a Java Project

DURGA SOFTWARE SOLUTIONS
Tools Material



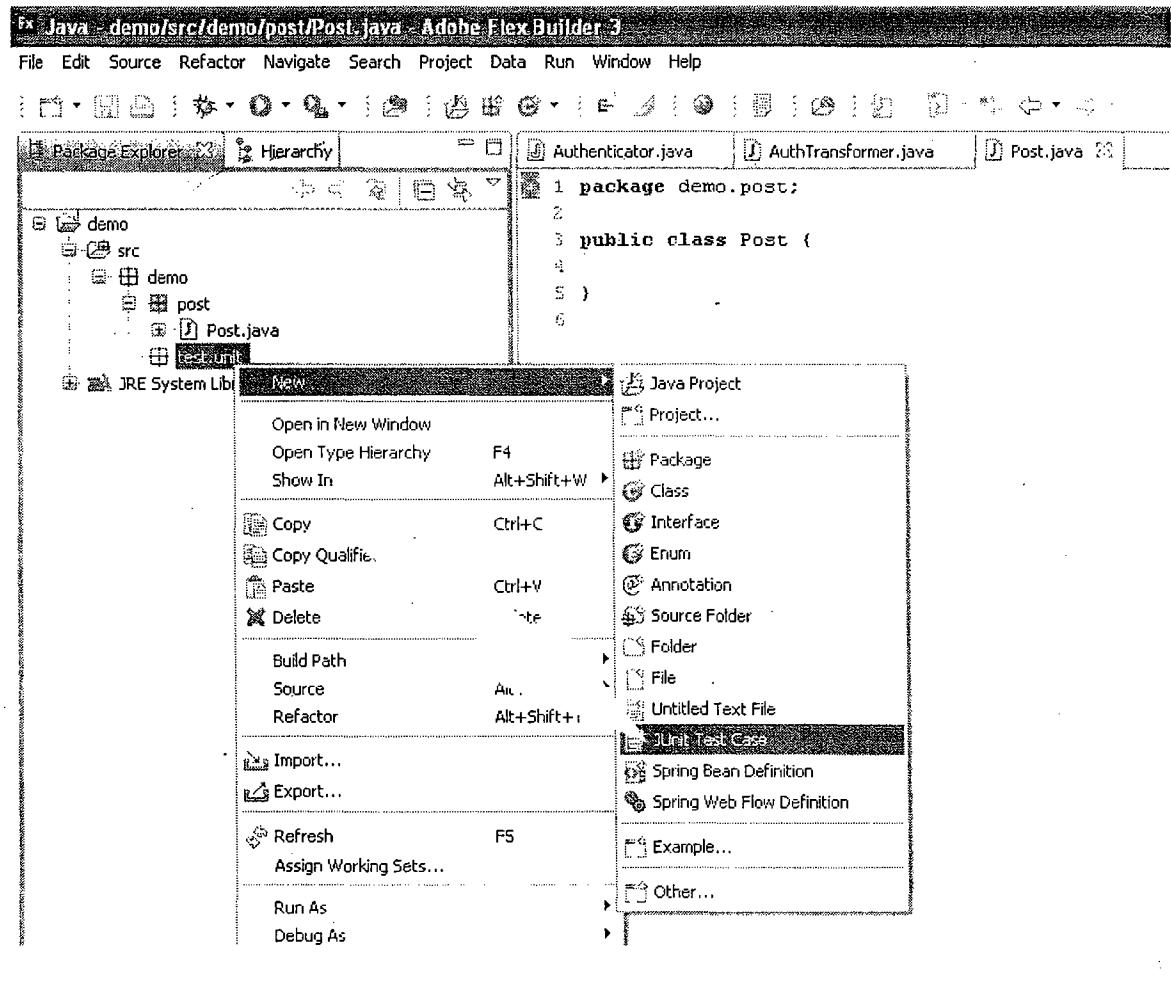
DURGA SOFTWARE SOLUTIONS
Tools Material

Lets Create a Model class



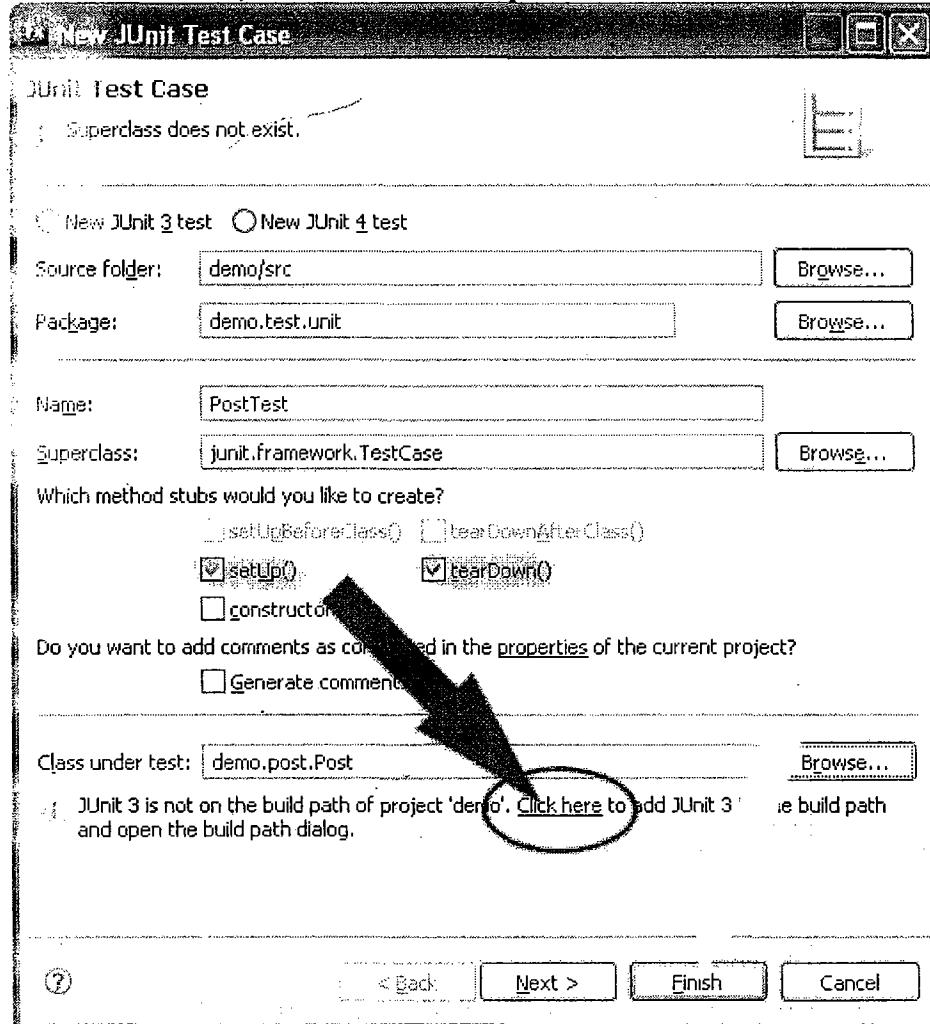
DURGA SOFTWARE SOLUTIONS
Tools Material

Add a JUNIT TestCase class



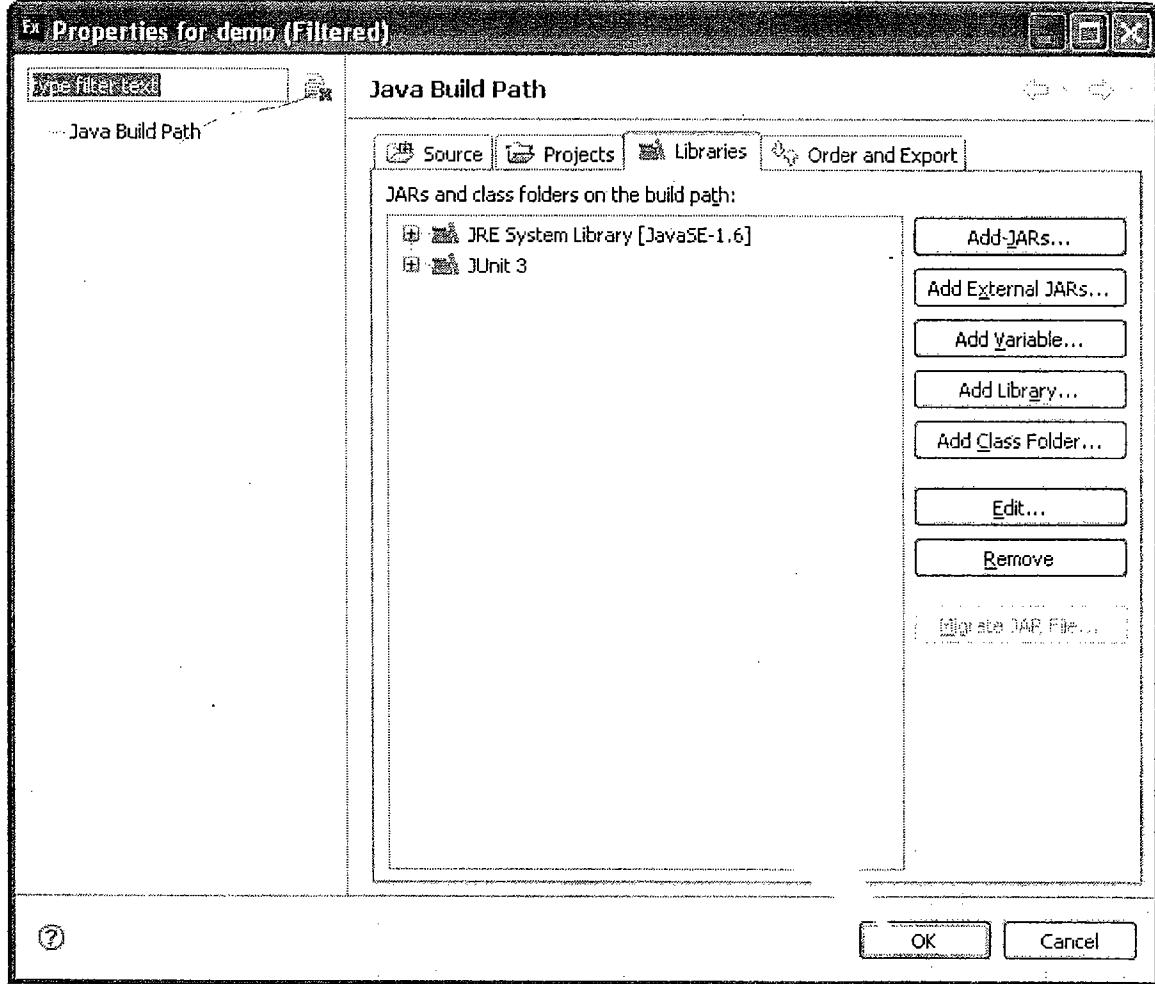
DURGA SOFTWARE SOLUTIONS
Tools Material

Call it PostTest, to follow a name pattern



DURGA SOFTWARE SOLUTIONS
Tools Material

Add Junit Library to class path



DURGA SOFTWARE SOLUTIONS Tools Material

Write Your own TestCases

Java - demo/src/demo/test/unit/PostTest.java - Adobe Flex Builder 3

File Edit Source Refactor Navigate Search Project Data Run Window Help

Package Explorer Hierarchy

```
demo
  src
    demo
      post
        Post.java
        PostTest.java
    test.unit
    JRE System Library [JavaSE-1.6]
    JUnit 4
```

Authenticator.java AuthTransformer.java Post.java PostTest.java

```
1 package demo.test.unit;
2
3 import demo.post.Post;
4 import junit.framework.TestCase;
5
6 public class PostTest extends TestCase {
7     private Post post;
8
9     protected void setUp() throws Exception {
10         post = new Post();
11     }
12
13     protected void tearDown() throws Exception {
14         post = null;
15     }
16
17 }
18
19 }
```

Test Your Functionalities

Java - demo/src/demo/post/Post.java - Adobe Flex Builder 3

File Edit Source Refactor Navigate Search Project Data Run Window Help

Package Explorer Hierarchy

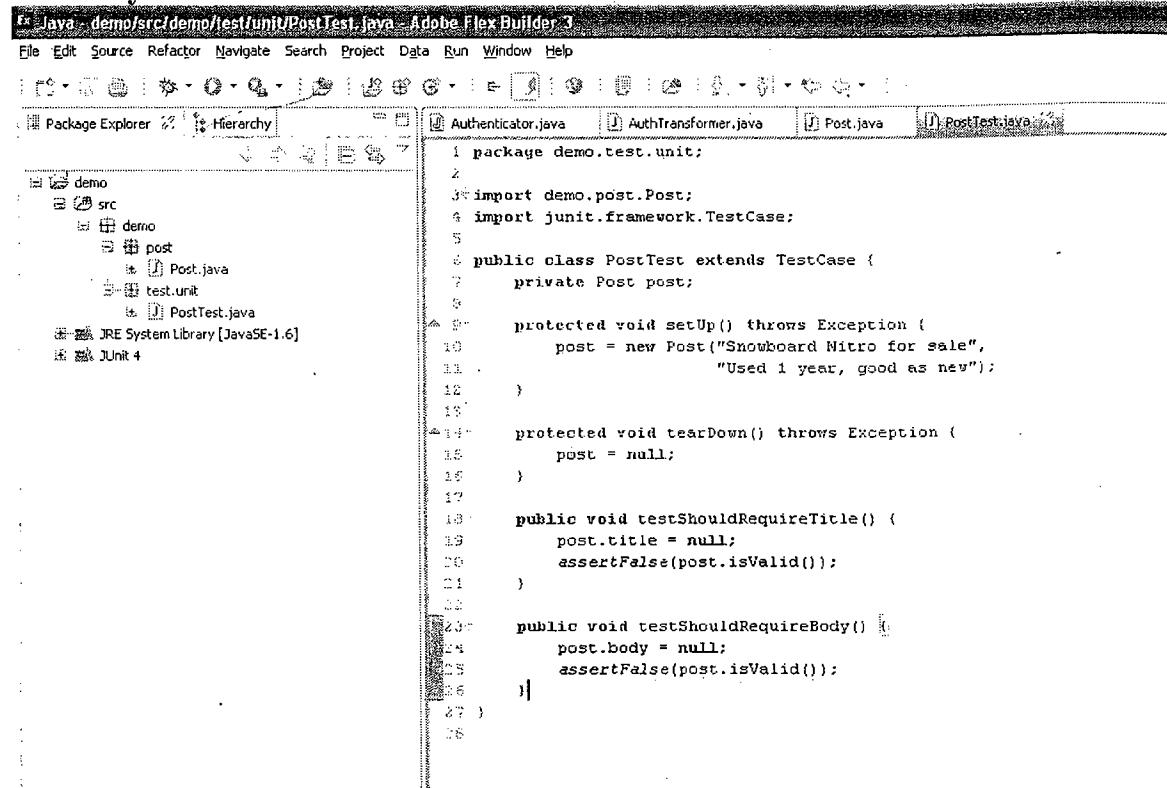
```
demo
  src
    demo
      post
        Post.java
        PostTest.java
    test.unit
    JRE System Library [JavaSE-1.6]
    JUnit 4
```

Authenticator.java AuthTransformer.java Post.java PostTest.java

```
1 package demo.post;
2
3 public class Post {
4     public String title; // required
5     public String body; // required
6
7     public Post() {
8         title = null;
9         body = null;
10    }
11
12    public Post(String aTitle, String aBody) {
13        title = aTitle;
14        body = aBody;
15    }
16
17    public boolean isValid() {
18        return title != null && body != null &&
19                      title.length() > 0 && body.length() > 0;
20    }
21
22 }
```

DURGA SOFTWARE SOLUTIONS
Tools Material

Add Any number of TestCase Methods



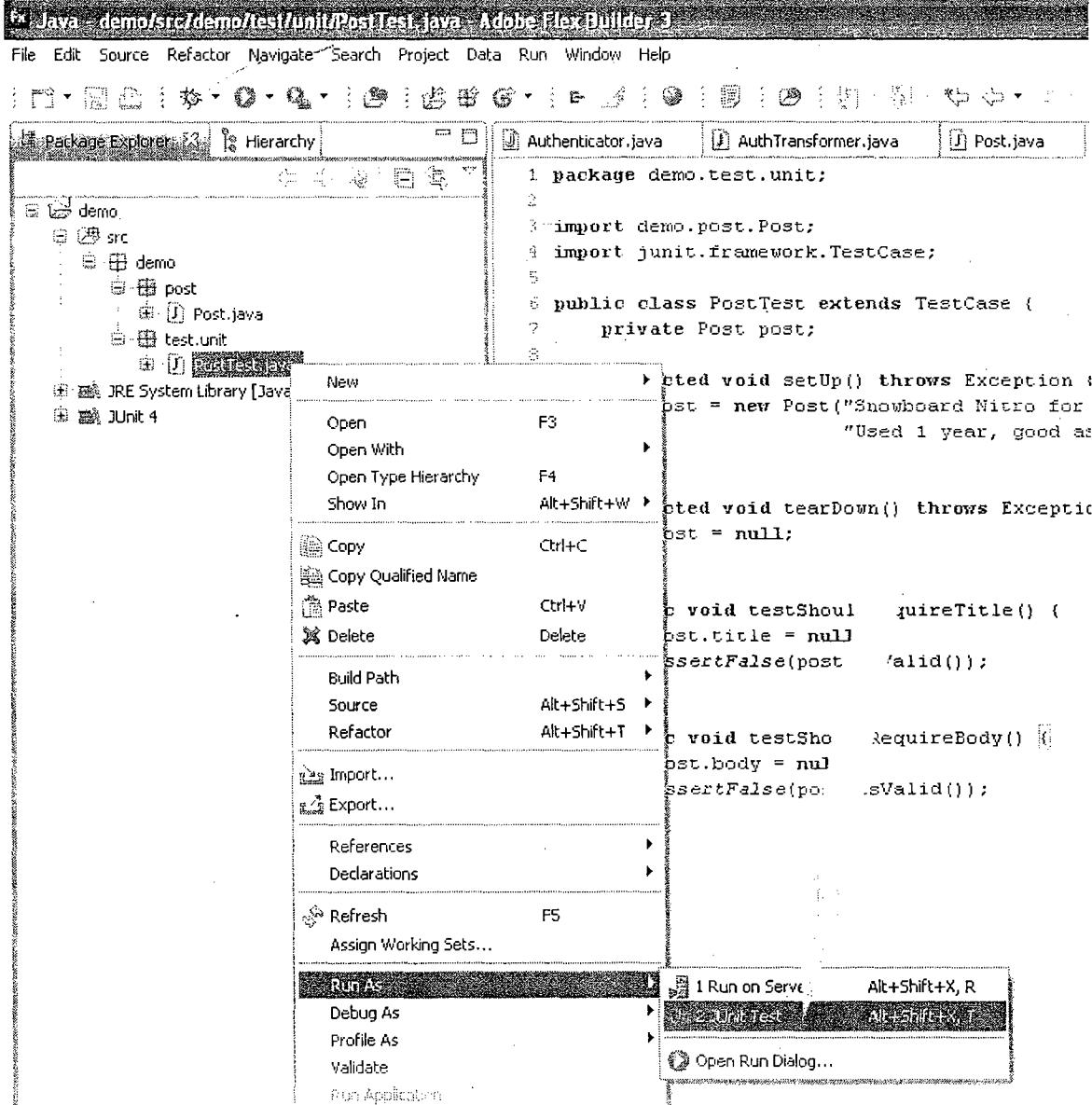
The screenshot shows the Adobe Flex Builder 3 IDE interface. The title bar reads "File: demo/src/demo/test/unit/PostTest.java - Adobe Flex Builder 3". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Data, Run, Window, Help. The toolbar has various icons for file operations. The left sidebar shows the "Package Explorer" with a project structure: demo > src > demo > post > Post.java, test.unit > PostTest.java. Below it are JRE System Library [JavaSE-1.6] and JUnit 4. The main workspace displays the Java code for PostTest.java:

```
1 package demo.test.unit;
2
3 import demo.post.Post;
4 import junit.framework.TestCase;
5
6 public class PostTest extends TestCase {
7     private Post post;
8
9     protected void setUp() throws Exception {
10         post = new Post("Snowboard Nitro for sale",
11                         "Used 1 year, good as new");
12     }
13
14     protected void tearDown() throws Exception {
15         post = null;
16     }
17
18     public void testShouldRequireTitle() {
19         post.title = null;
20         assertFalse(post.isValid());
21     }
22
23     public void testShouldRequireBody() {
24         post.body = null;
25         assertFalse(post.isValid());
26     }
27 }
```

DURGA SOFTWARE SOLUTIONS
Tools Material

Running TestCase

Here we go: Run As => JUnit Test



Analysing Test Results

Ex: Java - demo/src/demo/test/unit/PostTest.java - Adobe Flex Builder 3

File Edit Source Refactor Navigate Project Data Run Window Help

Package Explorer Hierarchy JUnit Post.java PostTest.java

Finished after 0,016 seconds

Runs: 2/2 Errors: 0 Failures: 0

testShouldRequireTitle testShouldRequireBody

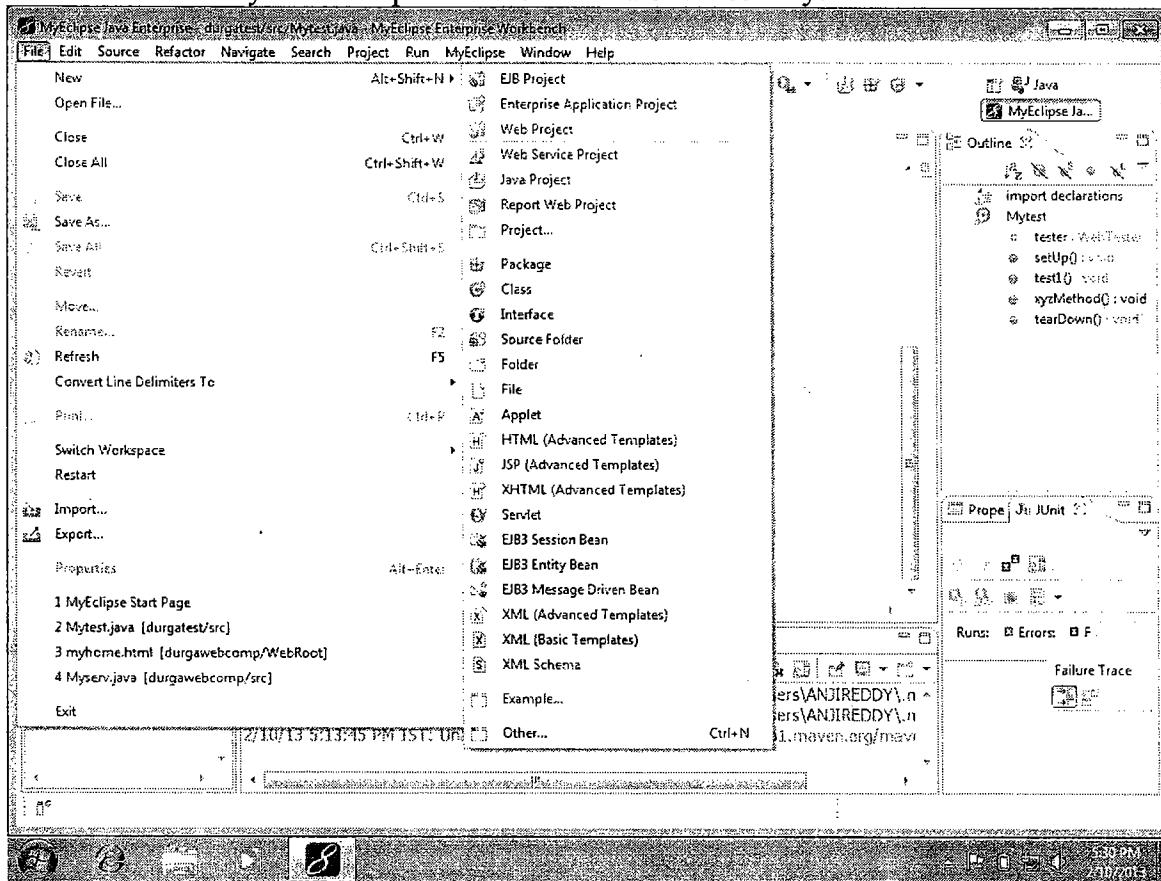
```
1 package demo.test.unit;
2
3 import demo.post.Post;
4 import junit.framework.TestCase;
5
6 public class PostTest extends TestCase {
7     private Post post;
8
9     protected void setUp() {
10         post = new Post("Some Title", "Useless Body");
11     }
12
13     protected void tearDown() {
14         post = null;
15     }
16
17     public void testShouldRequireTitle() {
18         post.title = null;
19         assertFalse(post.isTitleValid());
20     }
21
22     public void testShouldRequireBody() {
23         post.body = null;
24         assertFalse(post.isBodyValid());
25     }
26 }
27
28 }
```

Failure Trace

DURGA SOFTWARE SOLUTIONS
Tools Material

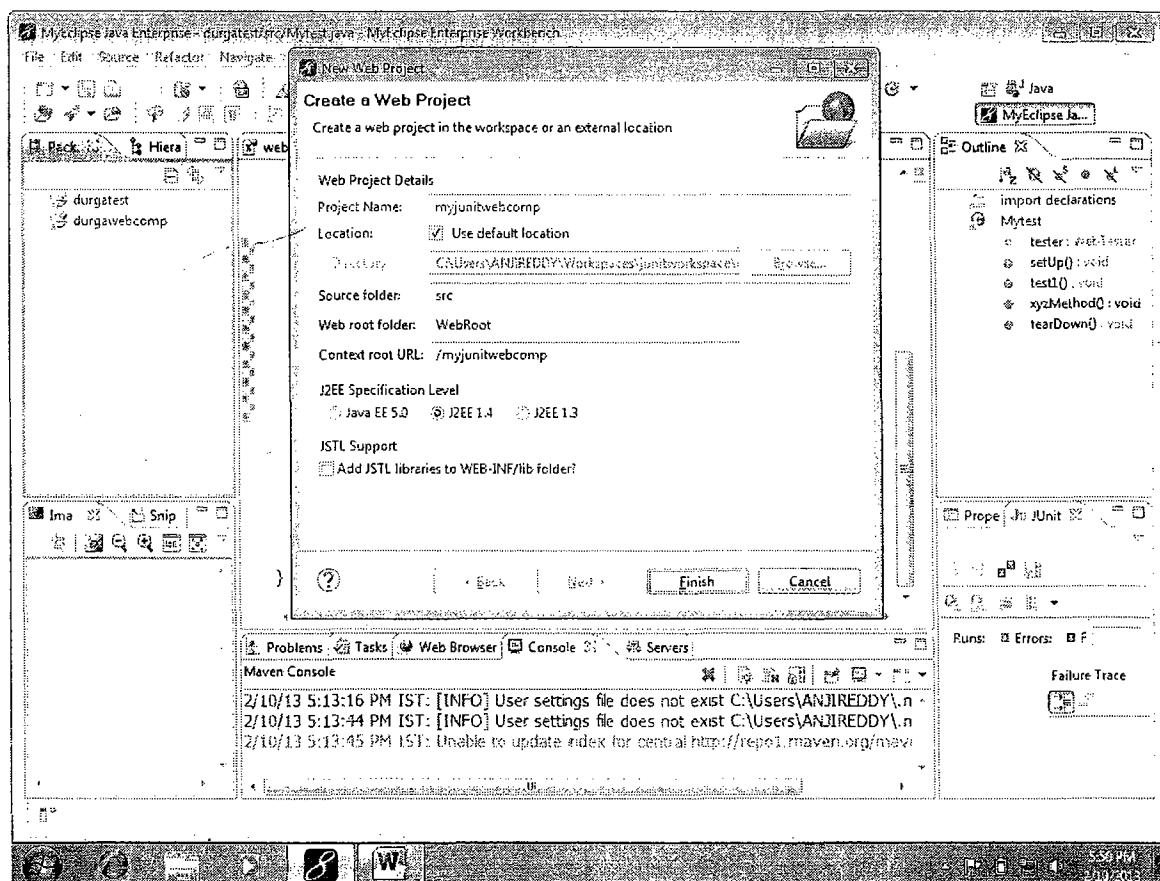
Junit webcomponentdevelopment and Testing Steps

- First of all we need to create new work space(work space is nothing but a small folder)in MyEclipse IDE.
 - File->New->webproject->enter the projectname(myjunitwebcomp)->ok->finish.
- then automatically webcomponet was created successfully.



DURGA SOFTWARE SOLUTIONS

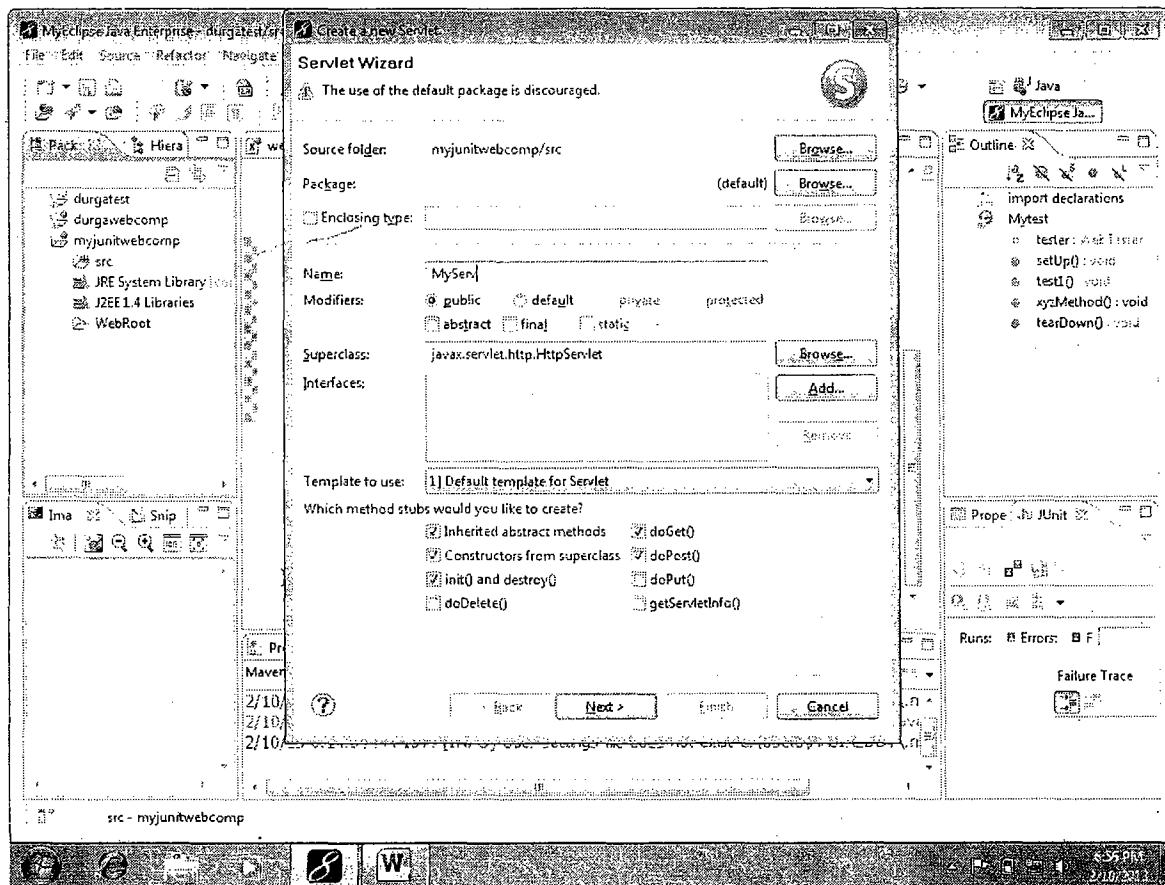
Tools Material



- Now we have to start project development code
→ Extract project folder (myjunitwebcomp) → right click on src folder → and take one servlet class and we can develop servlet component code.

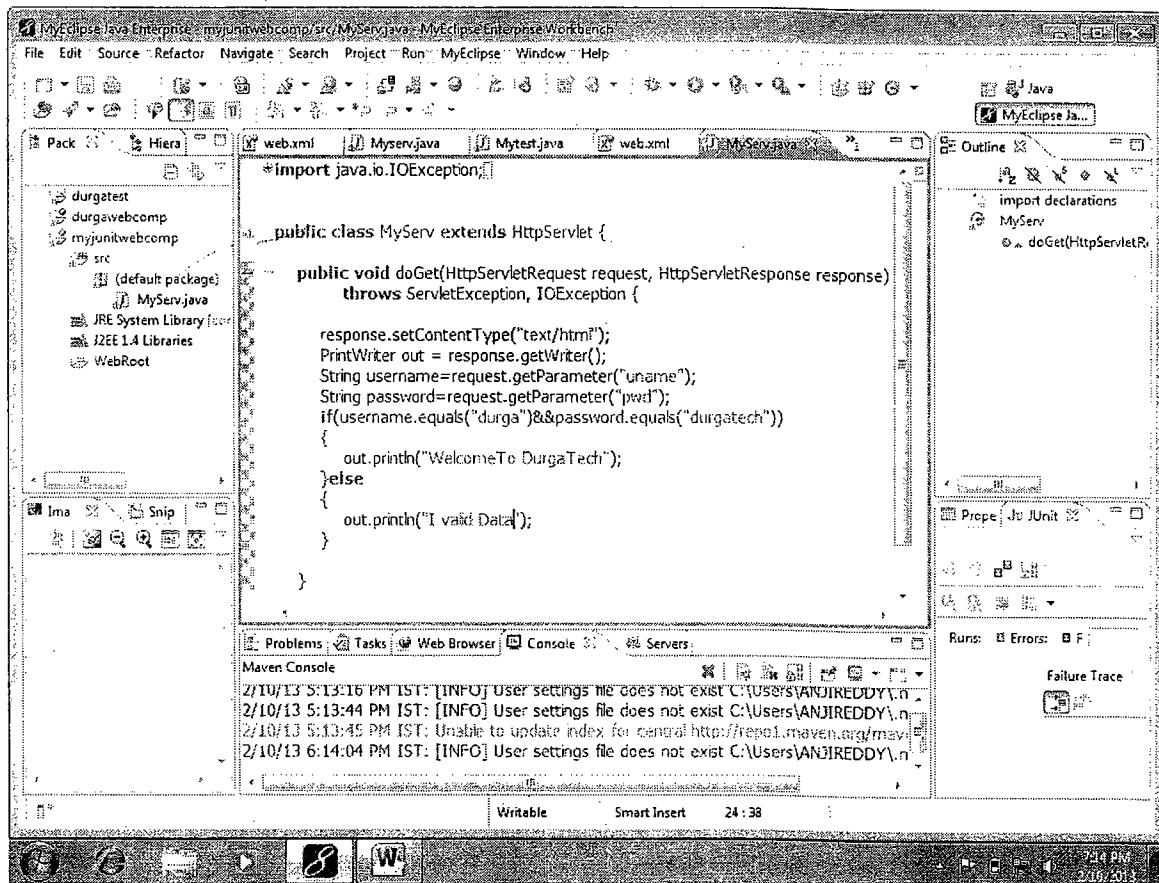
DURGA SOFTWARE SOLUTIONS

Tools Material



DURGA SOFTWARE SOLUTIONS

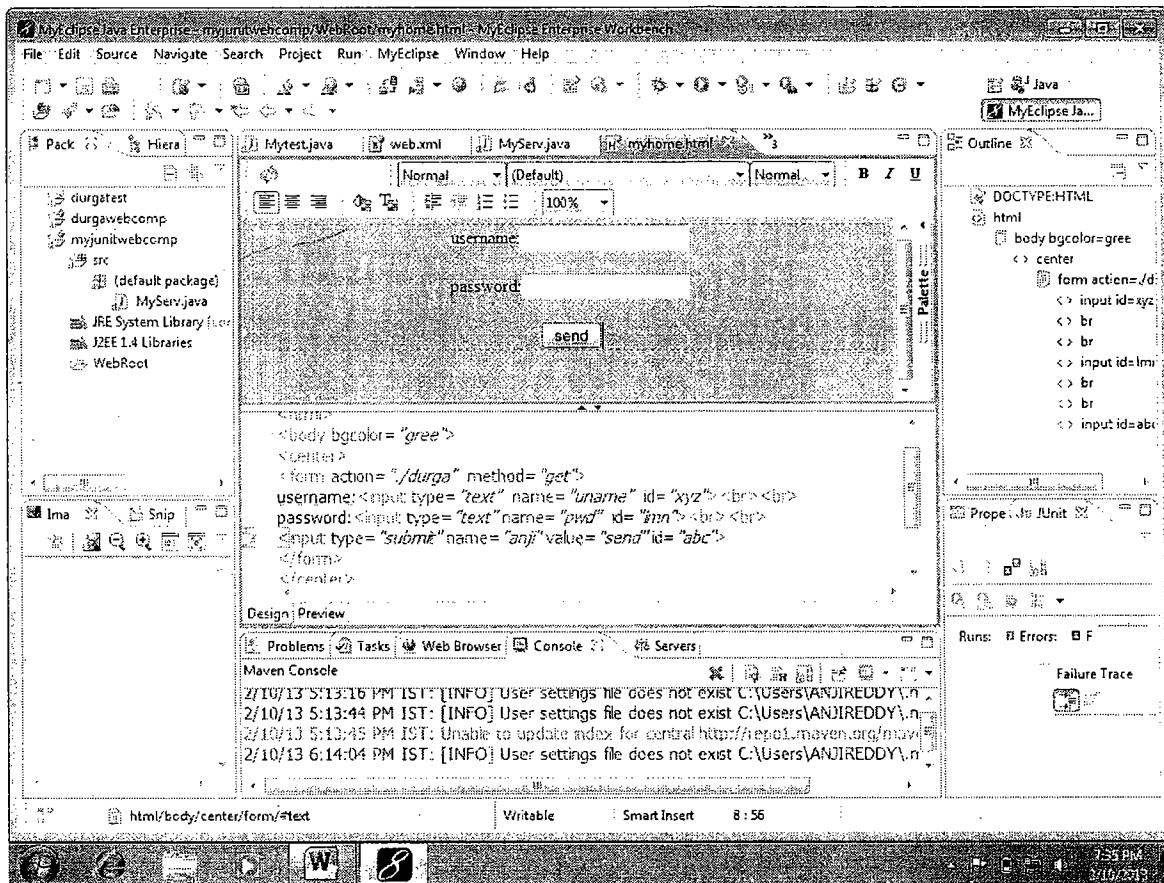
Tools Material



- Now by using html we can develop user details form
- Right click on projectfolder and create user details form and develop the code as follows.
- open the web.xml file and whater url we are passing in a myhome.html same url we can use in web.xml file also
- and finally configure the server and deployee the web application into server
- Now start the server, firt of all test the application manuually then it is success.
- After that we can go for junit testing and develop the testcases of the webcomponent

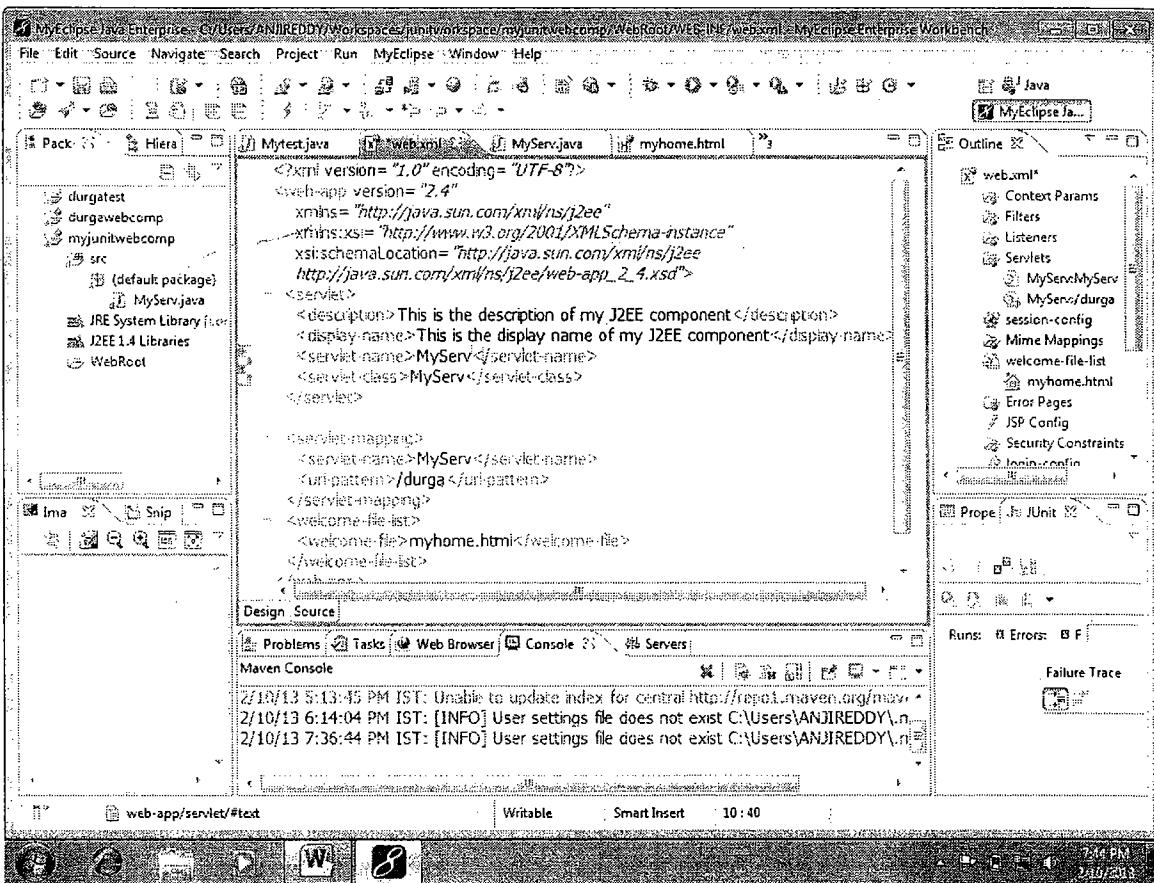
DURGA SOFTWARE SOLUTIONS

Tools Material



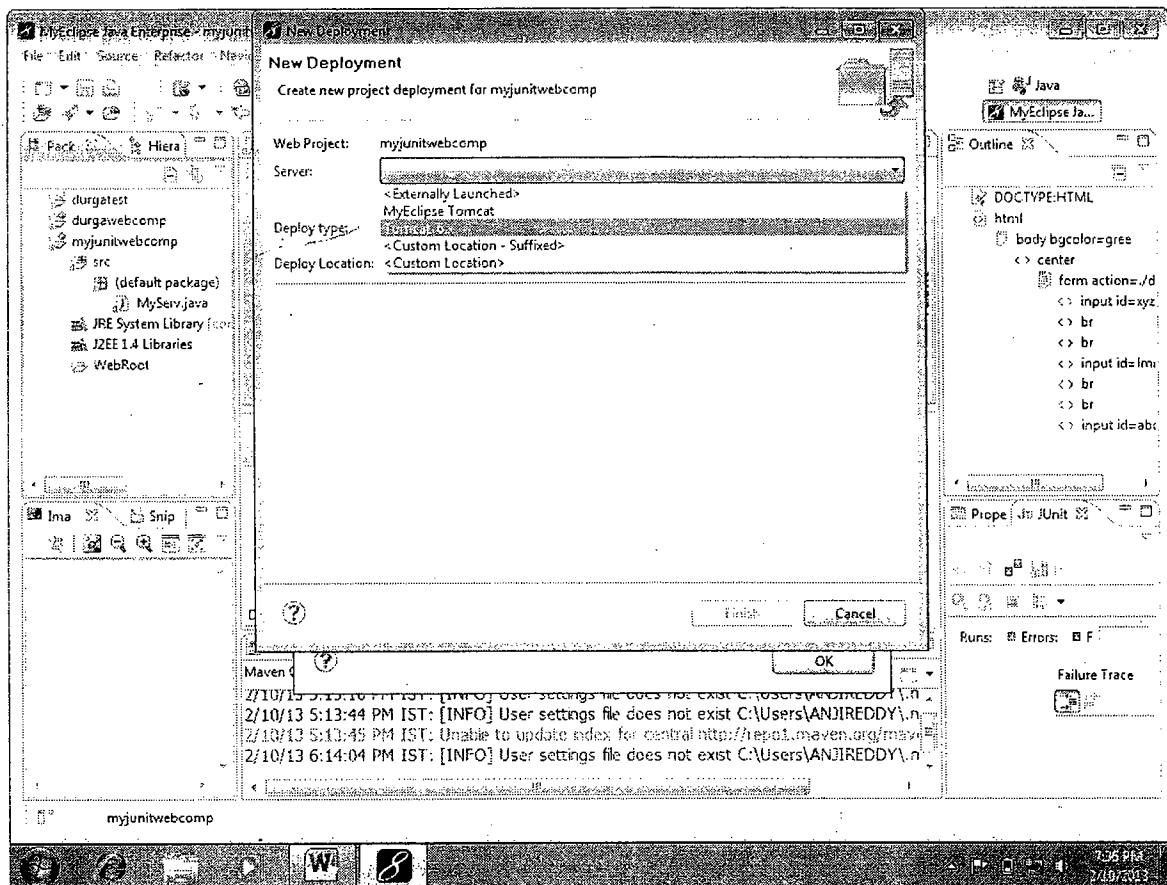
DURGA SOFTWARE SOLUTIONS

Tools Material



DURGA SOFTWARE SOLUTIONS

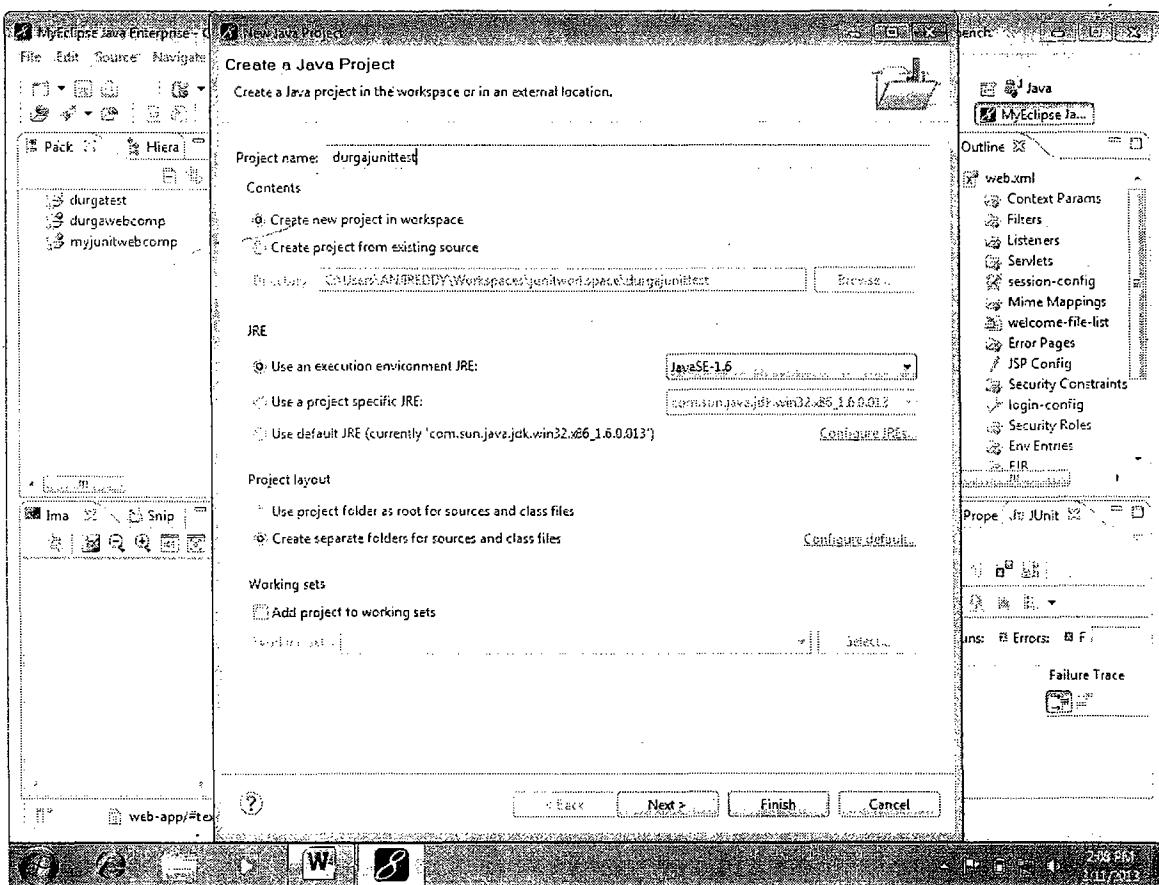
Tools Material



- Now we have to start junit component
- create normal java project and add all the junit related jar files as follows
- file->new->javaproject->enterprojectname(ex:durgajunitproject)->next->finish.
- right click on project->buildpath->cofigurebuildpath->addlibraries ->addexternaljarfiles->add all jar files related to junit.

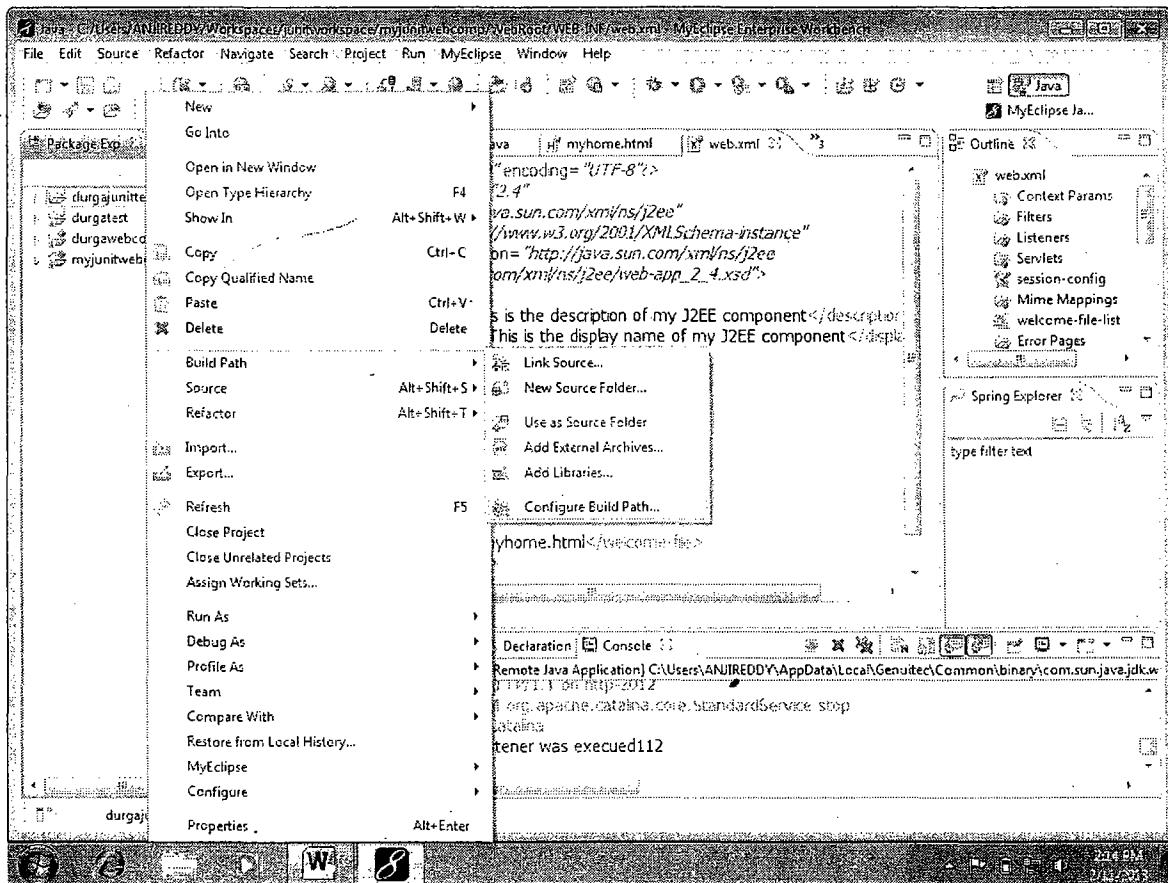
DURGA SOFTWARE SOLUTIONS

Tools Material



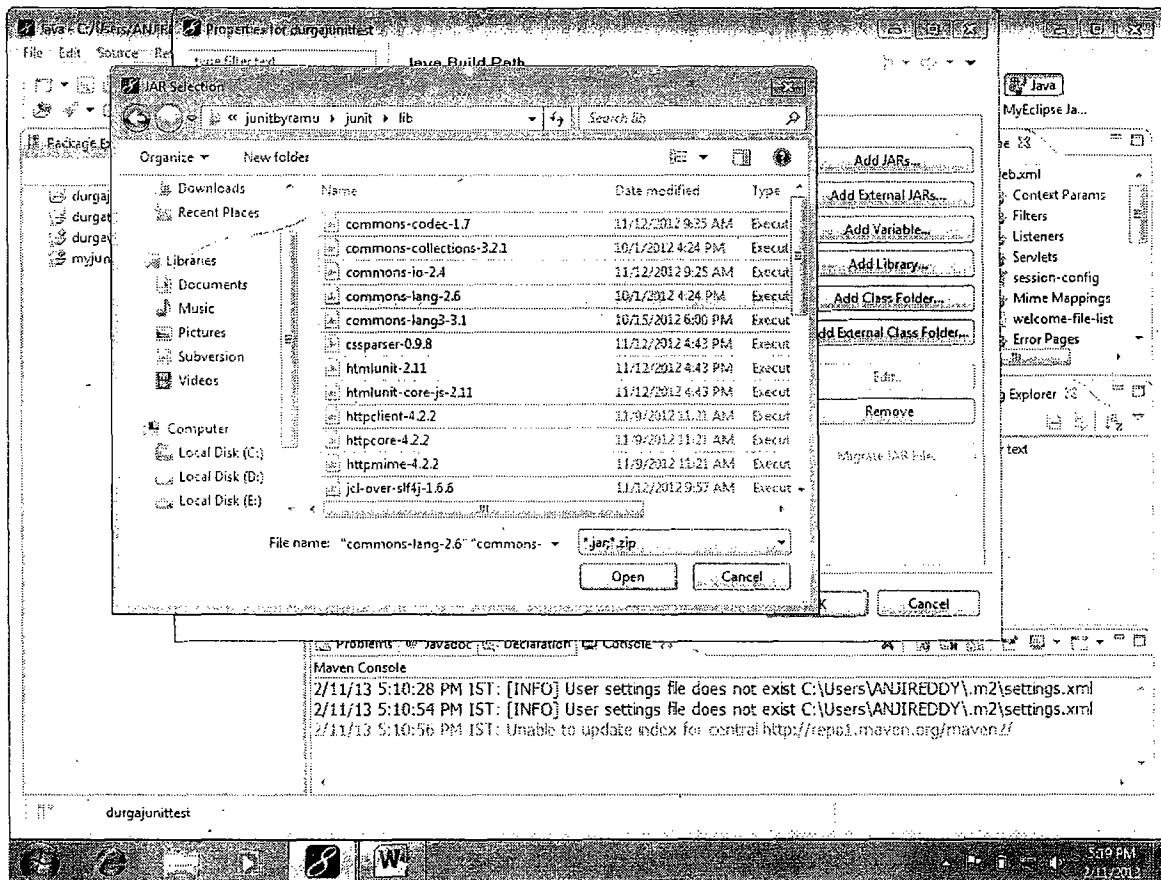
DURGA SOFTWARE SOLUTIONS

Tools Material



DURGA SOFTWARE SOLUTIONS

Tools Material



→ now test the application.

- the above jar files can be gathered by installing junit web slw.
- Gather above jar files from <webunit-homes> folder, <webunit-homes>\lib folder.
 ↓
 All jars
- we can gather above jar files by installing webunit-slw (webunit-3.1-release.zip file extraction).

DURGA SOFTWARE SOLUTIONS

Tools Material

The screenshot shows the MyEclipse Enterprise Workbench interface. The left pane displays a file tree with 'MyServer.java', 'myhome.html', and 'JUnitTestPlanComp.java'. The right pane shows the content of 'JUnitTestPlanComp.java'.

```
import org.junit.Before;
import org.junit.Test;

import net.sourceforge.jwebunit.junit.WebTester;
import junit.framework.*;
public class JUnitTestPlanComp extends WebTester {
    private WebTester tester;
    @Before
    public void setDurga() {
        tester=new WebTester();
        tester.setBaseUrl("http://localhost:2020/TestApp");
    }
    @Test
    public void durgaTest1(){
        tester.beginAt("myhome.html");
        tester.setTextField("name", "durga");
        tester.setTextField("pwd", "durgatech");
        tester.submit("click");
        tester.assertTextPresent("welcome to durgaTech");
    }
    @Test
    public void durgaTest2(){
        tester.beginAt("myhome.html");
        tester.setTextField("name", "durga");
        tester.setTextField("pwd", "durgatech");
        tester.submit("click");
        tester.assertTextPresent("Invalid Credential");
    }
}
```

The terminal window at the bottom shows the following output:

```
17:40:42.750 [main] DEBUG c.g.htmlunit.html.HtmlElement - Firing Event
17:40:42.750 [main] DEBUG c.g.htmlunit.html.HtmlElement - Firing Event
17:40:42.750 [main] DEBUG c.g.htmlunit.html.HtmlElement - Firing Event
```

A context menu is open on the right side of the screen, listing various Eclipse keyboard shortcuts.

*1: Submit the request.

DURGA SOFTWARE SOLUTIONS

Tools Material

The screenshot shows the MyEclipse Enterprise Workbench interface. The central part displays a Java file named `Myserv.java` containing JUnit test code. The code includes imports for `org.junit.Before`, `org.junit.Test`, `net.sourceforge.jwebunit.junit.WebTester`, and `junit.framework.*`. It defines a class `JunitTestPlanComp` that extends `WebTester`. The class has a private field `WebTester tester` and two methods: `setDurga()` and `durgaTest1()`. The `durgaTest1()` method uses `tester.beginAt("myhome.html")`, sets text fields for username ("durga") and password ("durgatech"), submits the form, and asserts that the page contains "welcomeTodurgatech". A failure trace is shown below, indicating an `java.lang.AssertionError: Expected text not found`.

```
import org.junit.Before;
import org.junit.Test;

import net.sourceforge.jwebunit.junit.WebTester;
import junit.framework.*;

public class JunitTestPlanComp extends WebTester {
    private WebTester tester;
    @Before
    public void setDurga()
    {
        tester=new WebTester();
        tester.setBaseUrl("http://localhost:2012/durgawebcomp");
    }
    @Test
    public void durgaTest1()
    {
        tester.beginAt("myhome.html");
        tester.setTextField("uname", "durga");
        tester.setTextField("pwd", "durgatech");
        tester.submit("click");
        tester.assertTextPresent("welcomeTodurgatech");
    }
}
```

In the bottom left corner of the code editor, there is a small icon of a person wearing a mask.

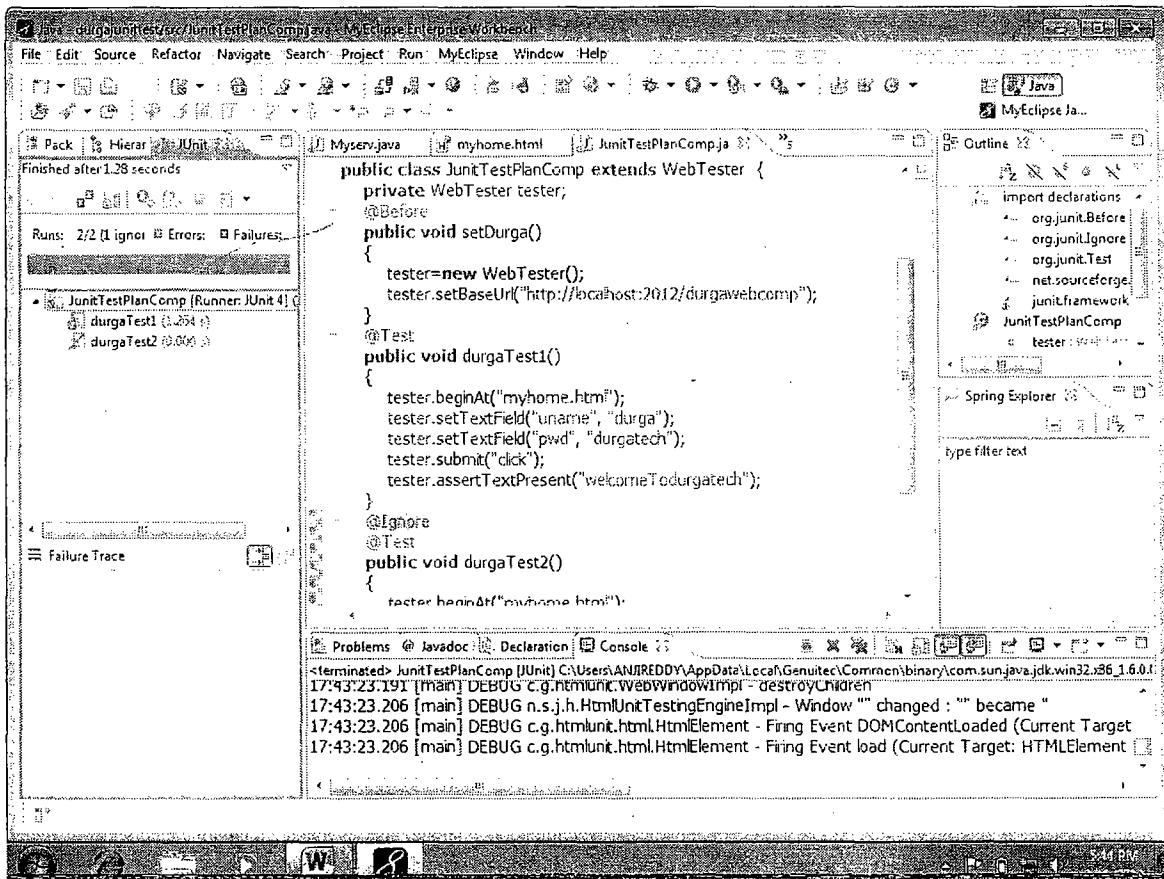
The Problems view at the bottom shows the following log entries:

```
<terminated> JunitTestPlanComp [JUnit] C:\Users\ANJIREDDY\AppData\Local\Genutec\Common\binary\com.sun.java.jdk.win32.x86.16.0.1
17:37:53.051 [main] DEBUG n.s.j.h.HtmlUnitTestingEngineImpl - Window "" changed : "" became "
17:37:53.051 [main] DEBUG c.g.htmlunit.html.HtmlElement - Firing Event DOMContentLoaded (Current Target
17:37:53.051 [main] DEBUG c.g.htmlunit.html.HtmlElement - Firing Event load (Current Target: HTMLElement
```

The above case is failure case.

DURGA SOFTWARE SOLUTIONS

Tools Material



This is an Ignore case

DURGA SOFTWARE SOLUTIONS
Tools Material

CVS

CONTENT:

1. Introduction
2. Why CVS?
3. Definition of CVS
4. Features of CVS
5. Terminology
 - i) Repository
 - ii) Sandbox
 - iii) Check out
 - iv) Commit (check in)
 - v) Update
 - vi) History
 - vii) Revision
6. Software Description
7. Software Installation
8. Working with CVS

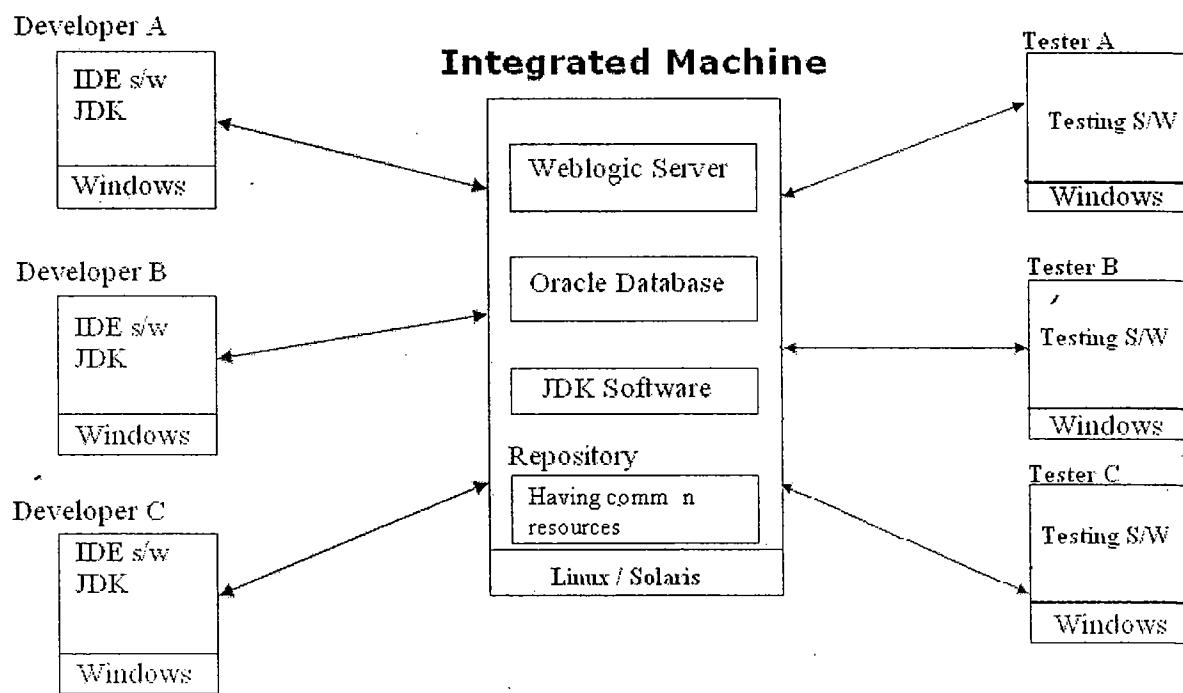
Server software CVS NT

IDE : MyEclipse

CVS (Version Control System)

Introduction:

In real time all developers and testers machines will be there running in windows environment but all these machines will be connected to a common machine of company called **integrated machine**. Generally this integrated machine resides in linux or solaris environment having high configuration and also contains the common software that are required for multiple projects of company.



The multiple projects of company will use the multiple logical databases that are created in database software of integrated machine on one per project basis.

DURGA SOFTWARE SOLUTIONS
Tools Material

During development mode of the project, the project will be maintained in the CVS Repository or SVN Repository to make the resources of the project visible and accessible for all developers of the project.

(code versioning)

Definition : CVS is a version control system. Using it, developers can record the history of their source files. This is also called as Source Code Management.

Dick Grune developed CVS as a series of shell scripts in July 1986.

- ❖ The CVS repository keeps track of various operations that are done in the files by developers by accessing the files (by developers by accessing the files) from CVS repository.
- ❖ CVS repository keeps track of various modifications done in files by different developers by generating versions.

Ex:

- Test.java (Original file)
- Test.java 1.1 (after first modification)
- Test.java 1.2 (after second modification)
- Test.java 1.3 (after third modification)

Benefits of Source Code Management: (versioning).

- ✓ All code changes are tracked.
- ✓ Avoid losing work due to simple mistakes (Allows you to roll back changes).
- ✓ Code changes across several developers can be synchronized.
- ✓ Makes it easy to backup your source code.
- ✓ You can work on several different copies of the same application at the same time.
- ✓ Supervisor can see how the code evolved over time.

Terminology:

DURGA SOFTWARE SOLUTIONS
Tools Material

- **Repository** : area on the server where the files are stored (**Project**).
- **Sandbox** : a local copy of the code which you work on and then commit to the repository
- **Checkout** : Process of collecting resource or project from CVS repository.
- **Commit** : Process of keeping resources back into CVS Repository after doing modifications is called as commit operation or Checkin operation.
- **Update** : getting code changes that have been committed since you checked out the project
- **Merge** : combining changes between two versions of the same file
- **History** : shows a list of commit messages, times and, who committed for a particular file
- **Revision** : cvs assigned version number for a file

Software Description:

Some CVS Repository softwares are

- ✓ CVSNT
- ✓ WinCVS
- ✓ Tortoise
- ✓ ClearCase
- ❖ The CVS repository software will be installed on the integrated machine and the IDE software of the developer machines will be configured to interact with CVS Repository.

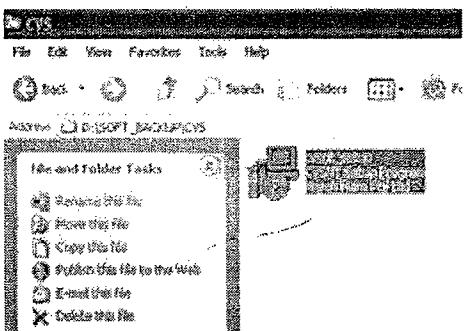
CVS NT:

Type : repository software
Version : 2.x
Download from : www.cvsnt.org

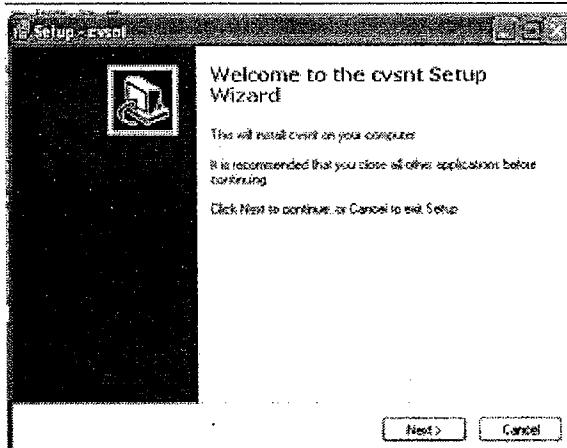
CVSNT INSTALLATION:

SERVER INSTALLATION STEPS

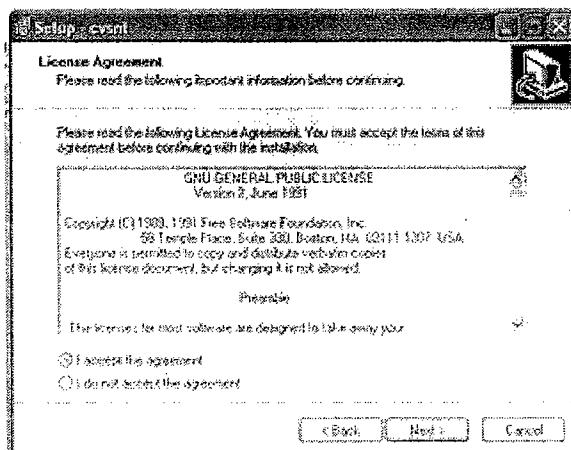
DURGA SOFTWARE SOLUTIONS Tools Material



CVSNT setup file. Double Click on the Setup file

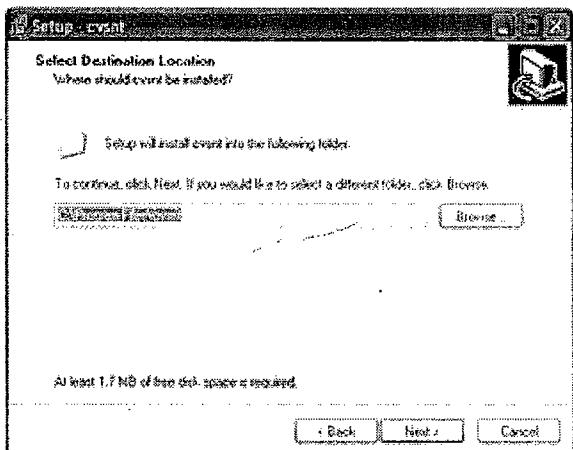


Click on Next

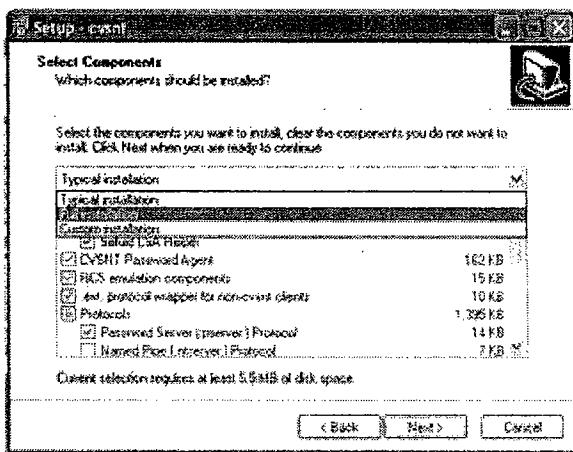


Accept the agreement and click on Next

DURGA SOFTWARE SOLUTIONS
Tools Material

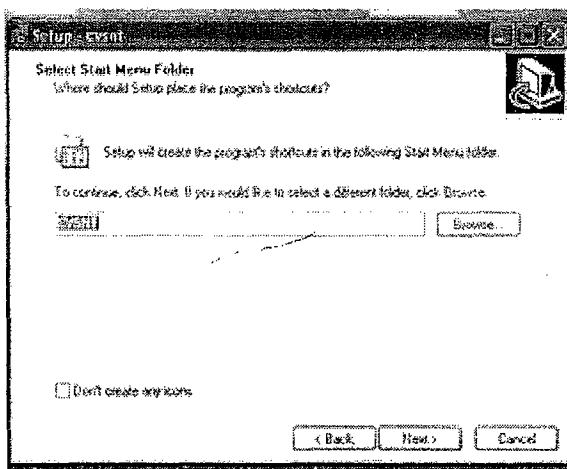


Select the Installation Directory(keep Default Only) and Click on Next

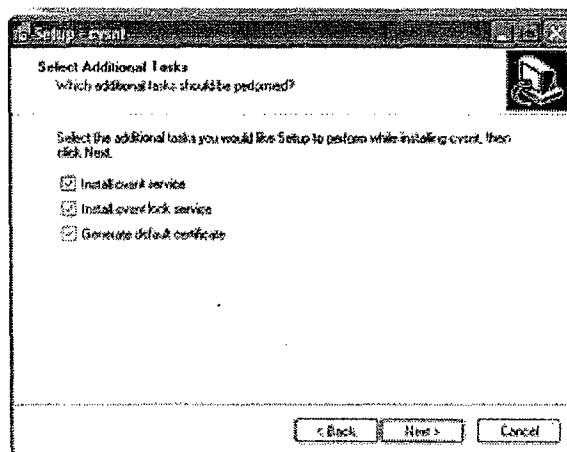


Select Full Installation and Click On Next

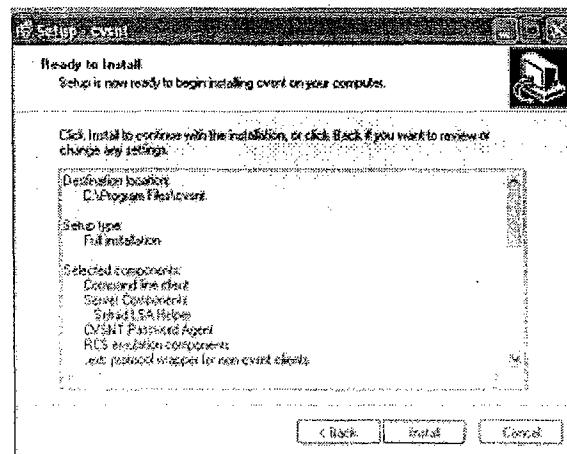
DURGA SOFTWARE SOLUTIONS
Tools Material



Click on Next



Select All and Click on Next



DURGA SOFTWARE SOLUTIONS
Tools Material

Click on Install



Click on Finish

Procedure to create CVS NT Repository for certain project / module :

Step-1: create a directory in computer's file system.

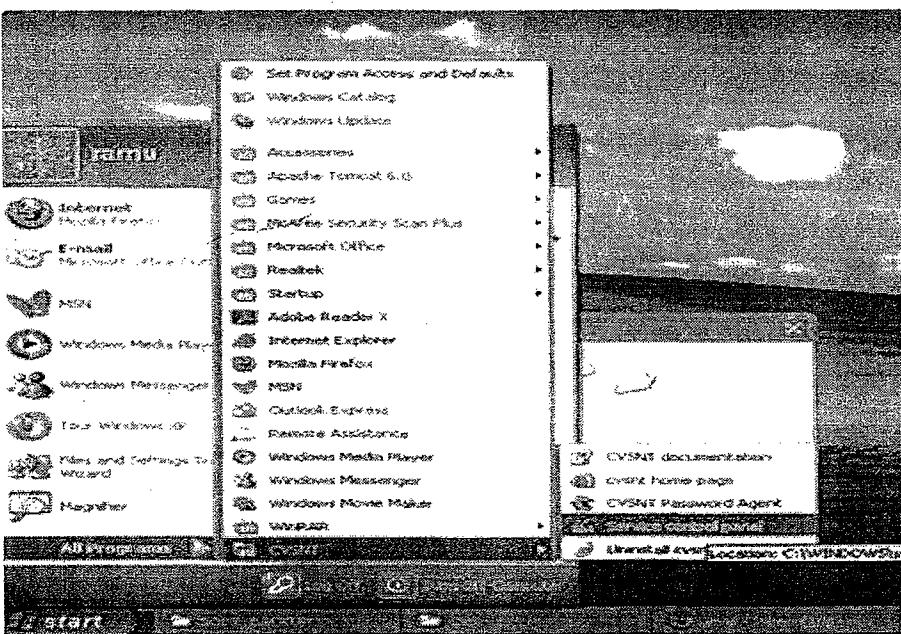
Ex: D:\CVSRep

Step-2: Start the service control panel of CVS NT software pointing to above folder (D:\CVSRep).

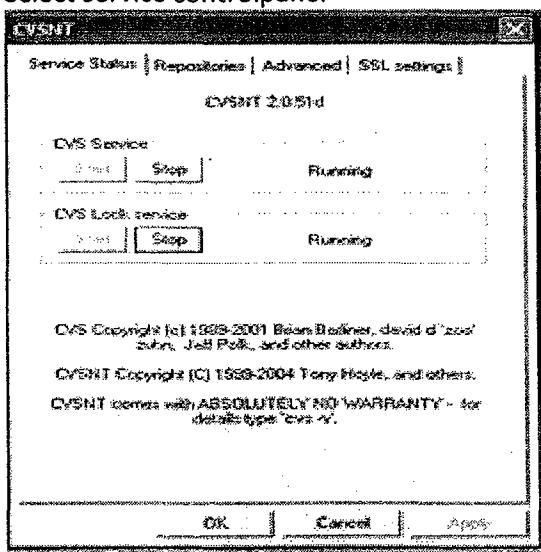
" Start → programs → CVS NT → Service Control Panel → click on **start CVS** service → click on **start CVS Lock Service** → repositories tab → add → location → browse & select D:\CVSRep → give logical name for repository (ex: /cvsrep) ('/' mandatory) → ok → click on yes on Do you want to initialize pop up window → apply → ok "

Creating The Repository

DURGA SOFTWARE SOLUTIONS Tools Material

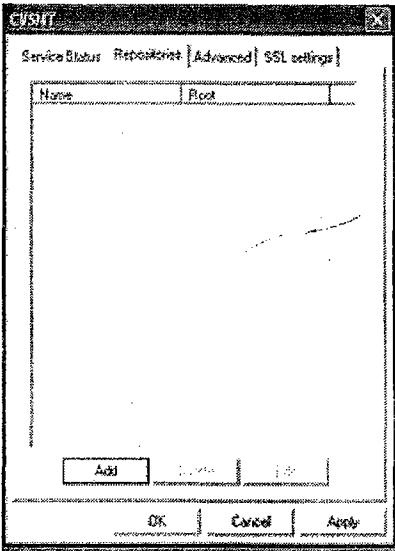


Select service controlpanel

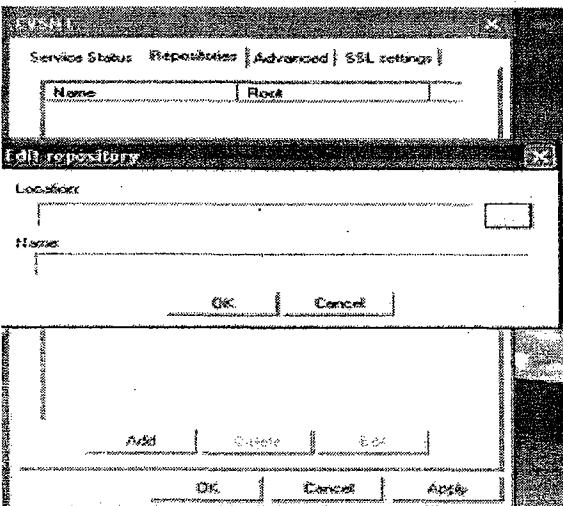


Start the CVS Service and CVS Lock Service and Go to Repositories tab

DURGA SOFTWARE SOLUTIONS
Tools Material

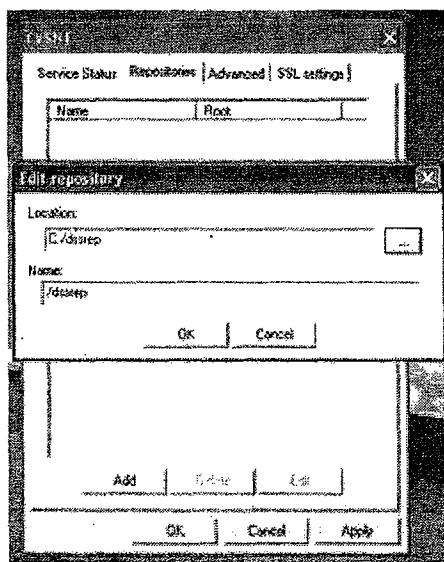
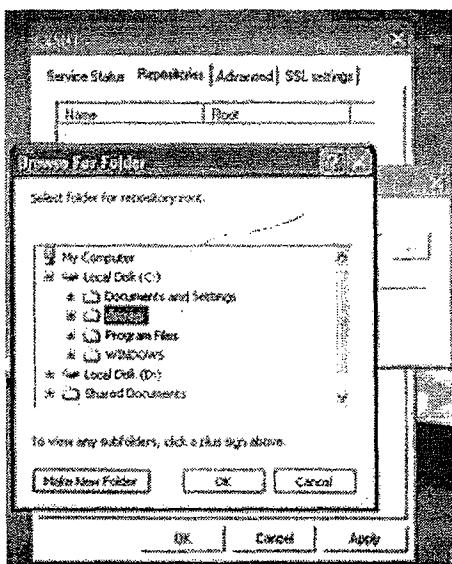


Click on Add Button

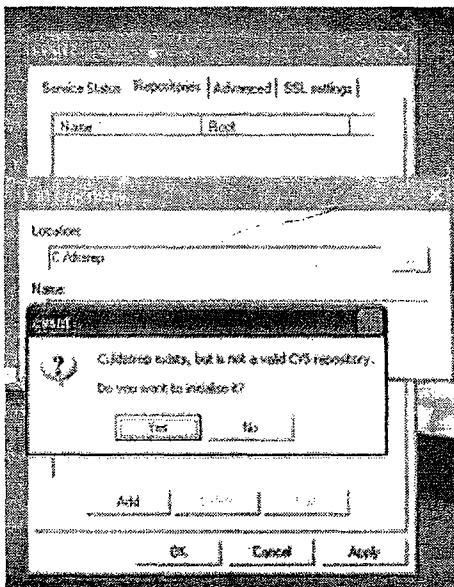


Select the repository Location and Logical Name

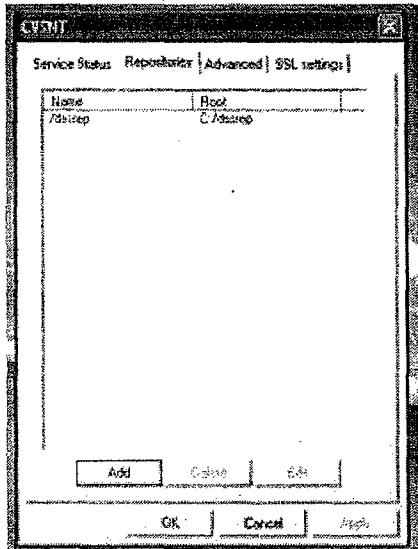
DURGA SOFTWARE SOLUTIONS
Tools Material



DURGA SOFTWARE SOLUTIONS
Tools Material



Click on Yes It will initialize the CVS Repository



Click on Apply

Click on Ok

The Repository will be created on the Server machine

Procedure to store project into CVS NT repository from MyEclipse IDE :

Step-1 : Launch MyEclipse having new work space for programmer.

Step-2 : Configure CVS NT repository with MyEclipse IDE.

“ Window menu → show view → other → CVS → select CVS editors repositories

→ ok

DURGA SOFTWARE SOLUTIONS
Tools Material

Go to repositories window → new → repository location

Give details for Host
Repository path
User
Password → finish "

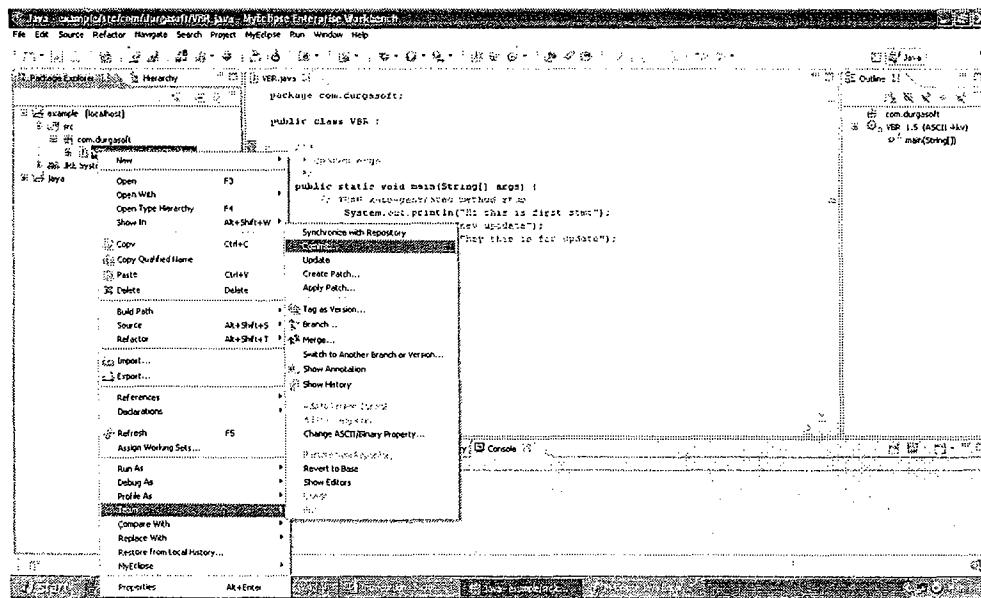
Step-3 : create a project in MyEclipse IDE.

Step-4 : Place the above project in CVSNT Repository (/cvsrep).

" Right click on project → team → share project → select cvsrep repository → next → finish. "

Check in : Keep java source file in CVS repository (/cvsrep) .

" Right click on Test.java → team → commit (checkin) → enter some comment → finish. "



❖ A new version will create for file when you perform commit operation.

Ex: Test.java (original file)

Test.java 1.1 (after first modification)

DURGA SOFTWARE SOLUTIONS

Tools Material

Test.java 1.2 (after second modification)

Check out : import/checkout project from cvs repository (for another user).

" File menu → import → cvs → projects from CVS → next → use an existing module → select your project → next → checkout as a project in the work space → next → finish. "

The screenshot shows the MyEclipse Java Enterprise Workbench interface. The code editor displays the following Java code:

```
public class VBR {
    public static void main(String[] args) {
        System.out.println("Hi this is first code");
        System.out.println("New update");
        System.out.println("Hey this is for update");
    }
}
```

The properties panel on the right shows the following settings:

Property	Value
derver	false
editable	true
lasted	October 24, 2011 1:16:46 PM
linked	
location	C:\Documents and Settings\user\My Documents
path	VBR.java

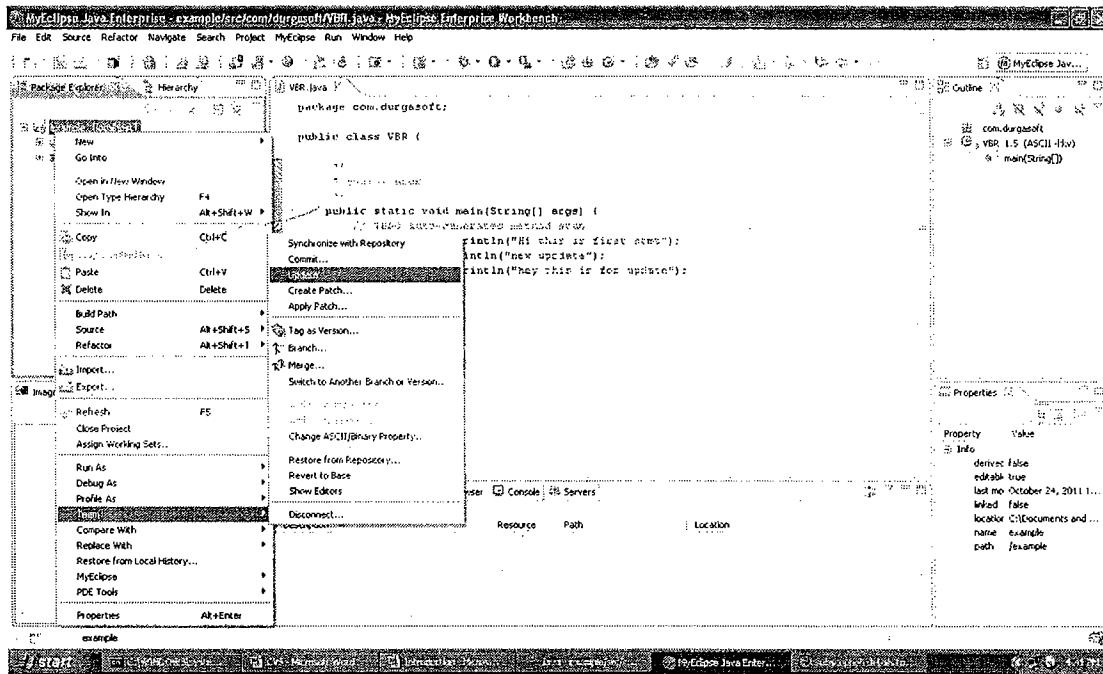
Update : This operation can be used for two purposes.

- To get the updated version of the source file.
- To update the content of current file in cvs repository.

" Right click on file → team → update. "

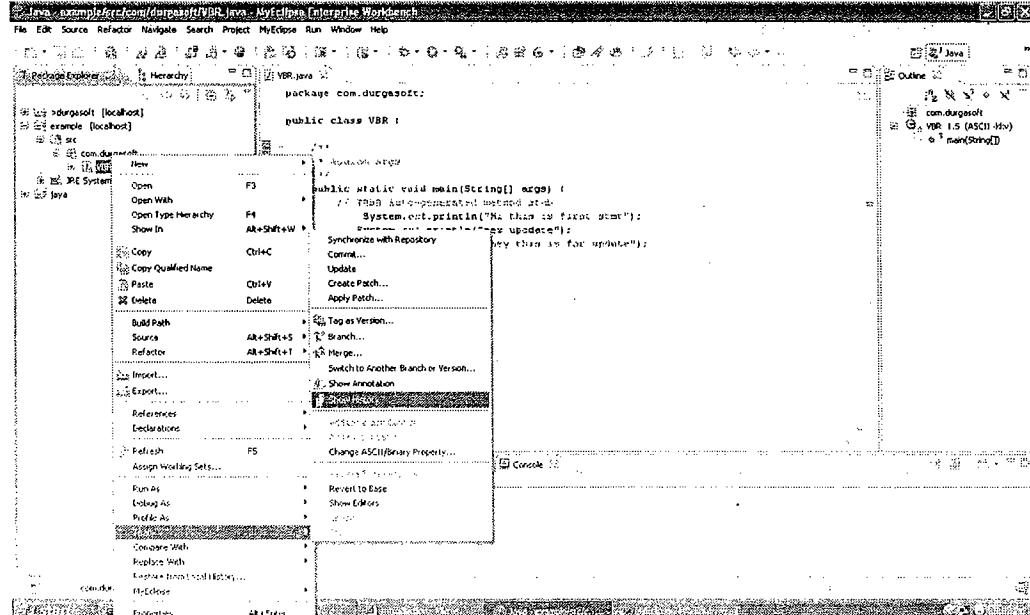
DURGA SOFTWARE SOLUTIONS

Tools Material



History : To replace current file content with one of the existing old version of the cvs repository .

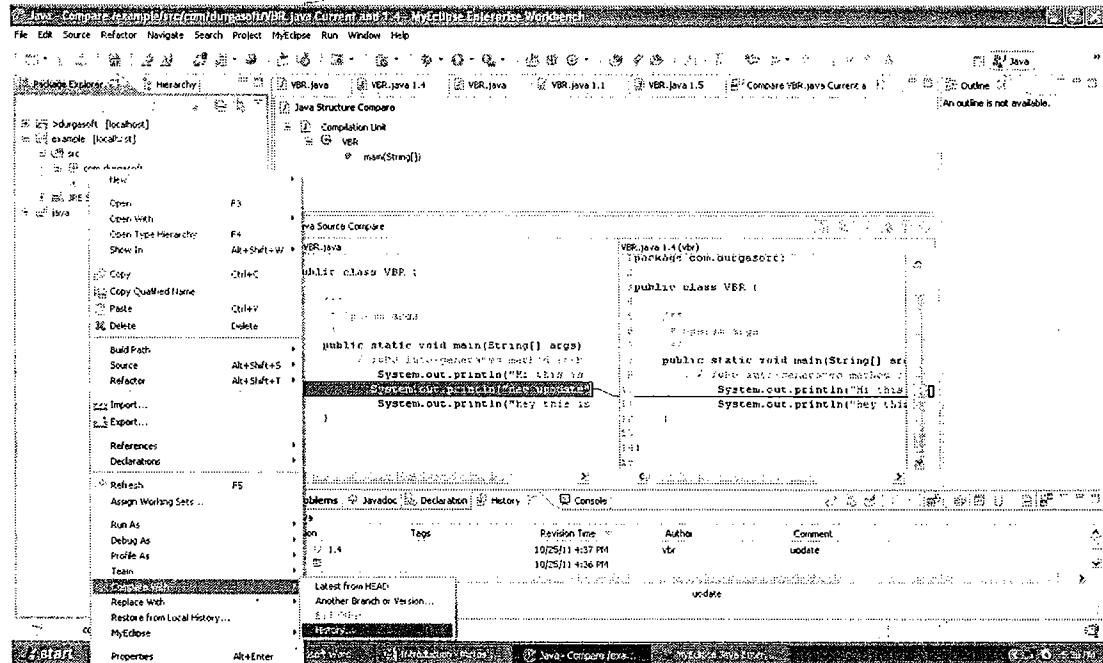
Click on .java file → team → show history → Go to history window → double click on version what ever you want.



DURGA SOFTWARE SOLUTIONS
Tools Material

Compare : To compare code of current version with the different versions in repository.

" Right click on source file → compare with → History → Go to history window select a version for compare "

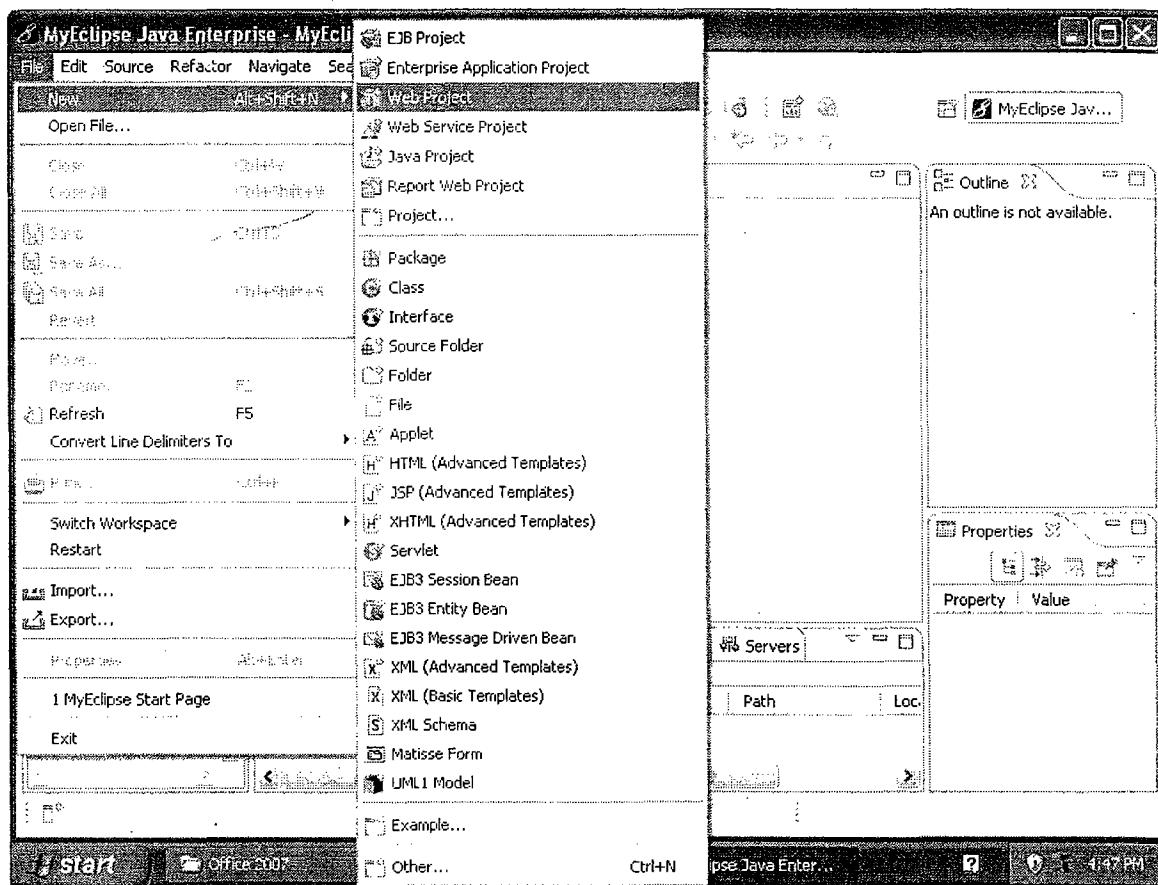


Steps to develop web application using cvs tool:

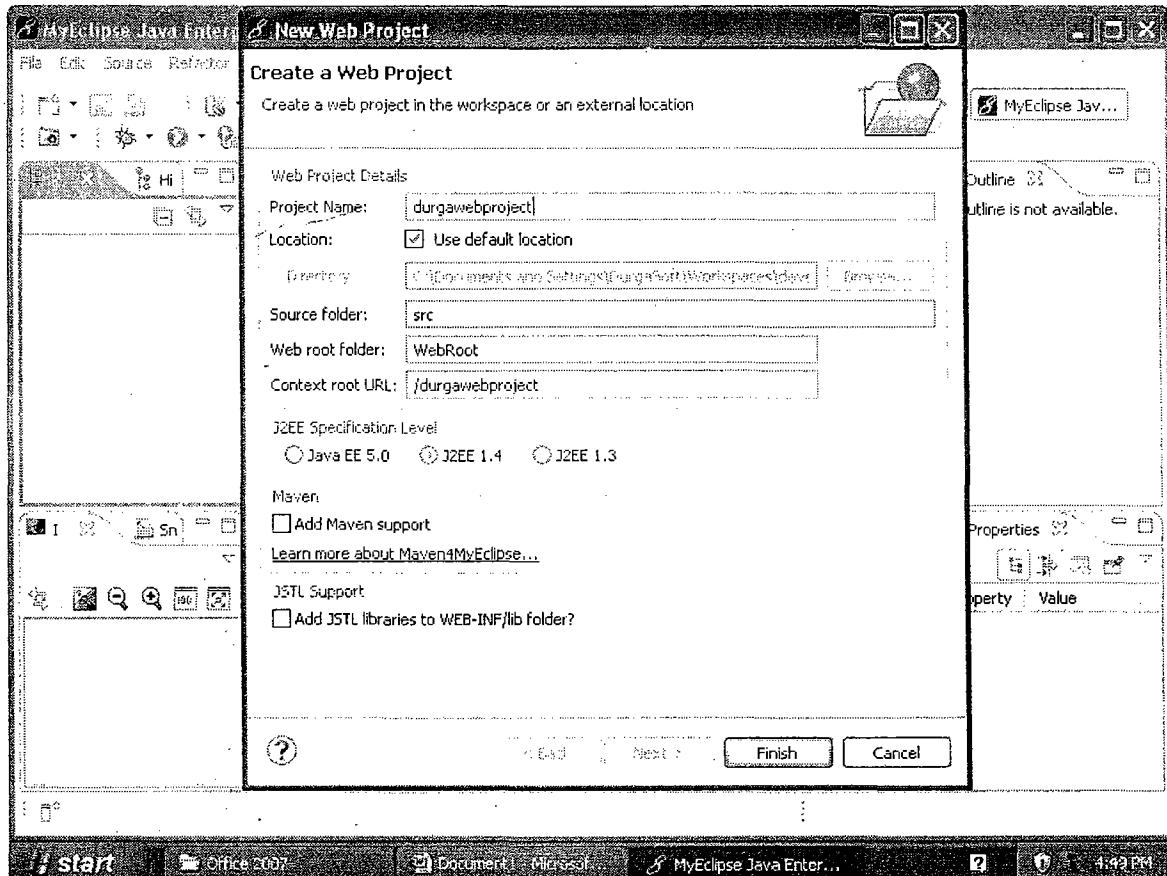
Step-1

→ First of all create the web project using myeclipse IDE as follows
File->new->webproject->enter the project name(durgawebproject)

DURGA SOFTWARE SOLUTIONS
Tools Material



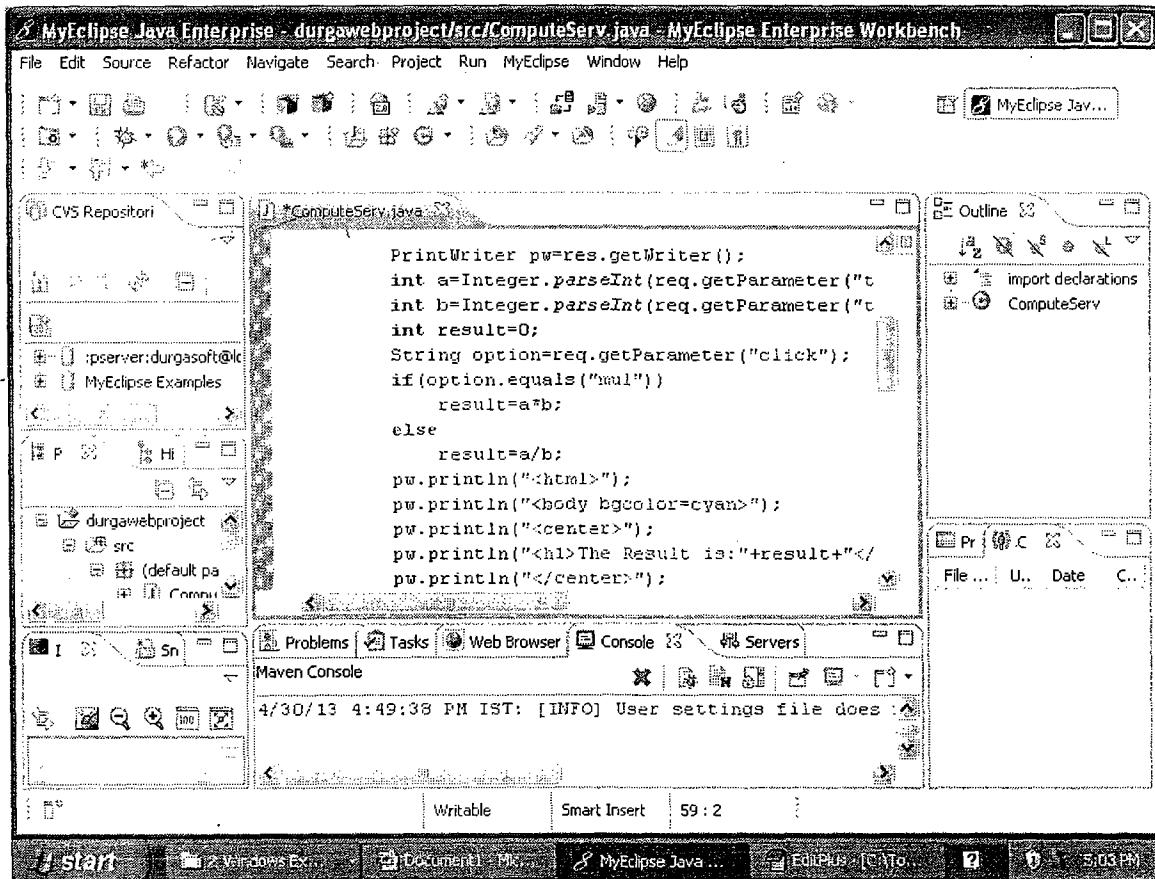
DURGA SOFTWARE SOLUTIONS
Tools Material



Step-2

- >Develop the web resources as follows.
- first of all develope the servlet in that servlet implement the some business logic.after that implementing the presentation pages.

DURGA SOFTWARE SOLUTIONS
Tools Material



Refer to servlet code as follows

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class ComputeServ extends HttpServlet
{
    public void doGet(HttpServletRequest req,HttpServletResponse
res)throws ServletException,IOException
    {
        System.out.println("your request is get=====!");
        process(req,res);
    }
    public void process(HttpServletRequest req,HttpServletResponse
res)throws ServletException,IOException
    {
        PrintWriter pw=res.getWriter();
        int a=Integer.parseInt(req.getParameter("t1"));
        int b=Integer.parseInt(req.getParameter("t2"));}
```

DURGA SOFTWARE SOLUTIONS
Tools Material

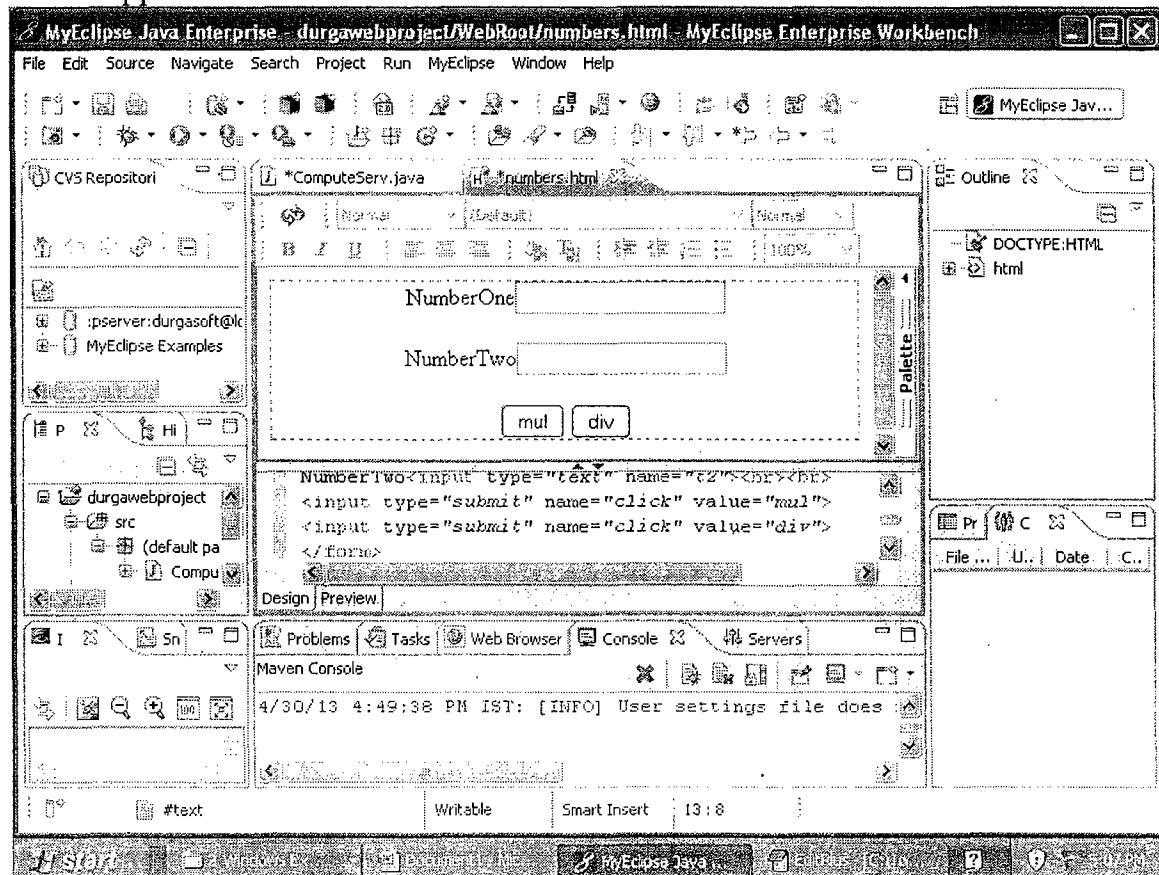
```
int result=0;
String option=req.getParameter("click");
if(option.equals("mul"))
    result=a*b;
else
    result=a/b;
pw.println("<html>");
pw.println("<body bgcolor=cyan>");
pw.println("<center>");
pw.println("<h1>The Result is:"+result+"</h1>");
pw.println("</center>");
pw.println("</body>");
pw.println("</html>");
pw.close();
}
public void doPost(HttpServletRequest req,HttpServletResponse
res) throws ServletException,IOException
{
    System.out.println("your request is
post=====!");
    process1(req,res);
}
public void process1(HttpServletRequest req,HttpServletResponse
res) throws ServletException,IOException
{
    PrintWriter pw=res.getWriter();
    int a=Integer.parseInt(req.getParameter("t1"));
    int b=Integer.parseInt(req.getParameter("t2"));
    int result=0;
    String option=req.getParameter("click");
    if(option.equals("mul"))
        result=a*b;
    else
        result=a/b;
    pw.println("<html>");
    pw.println("<body bgcolor=cyan>");
    pw.println("<center>");
    pw.println("<h1>The Result is:"+result+"</h1>");
    pw.println("</center>");
    pw.println("</body>");
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
        pw.println("</html>");  
        pw.close();  
    }  
}
```

Web.xml

```
<web-app>  
<servlet>  
<servlet-name>one</servlet-name>  
<servlet-class>ComputeServ</servlet-class>  
</servlet>  
<servlet-mapping>  
<servlet-name>one</servlet-name>  
<url-pattern>/compute</url-pattern>  
</servlet-mapping>  
</web-app>
```



Code for input page.

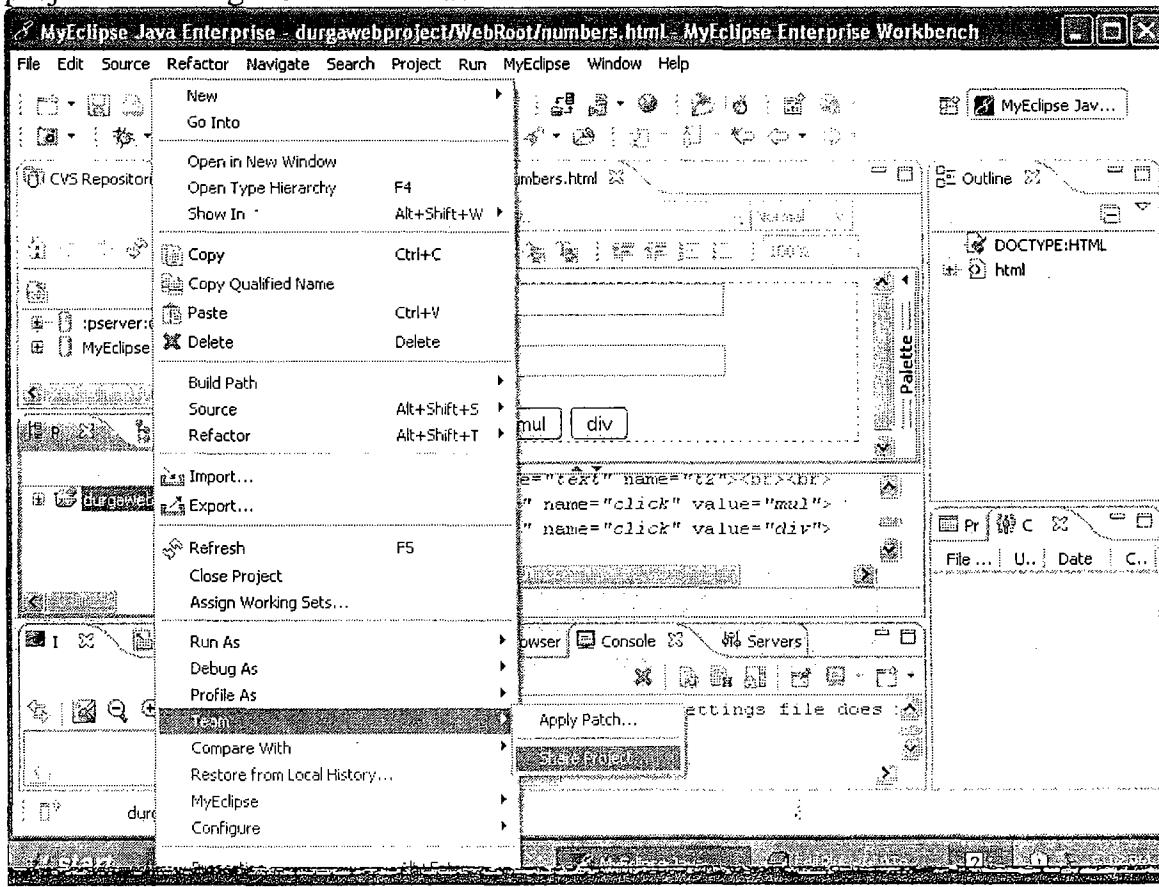
```
<html>
```

DURGA SOFTWARE SOLUTIONS
Tools Material

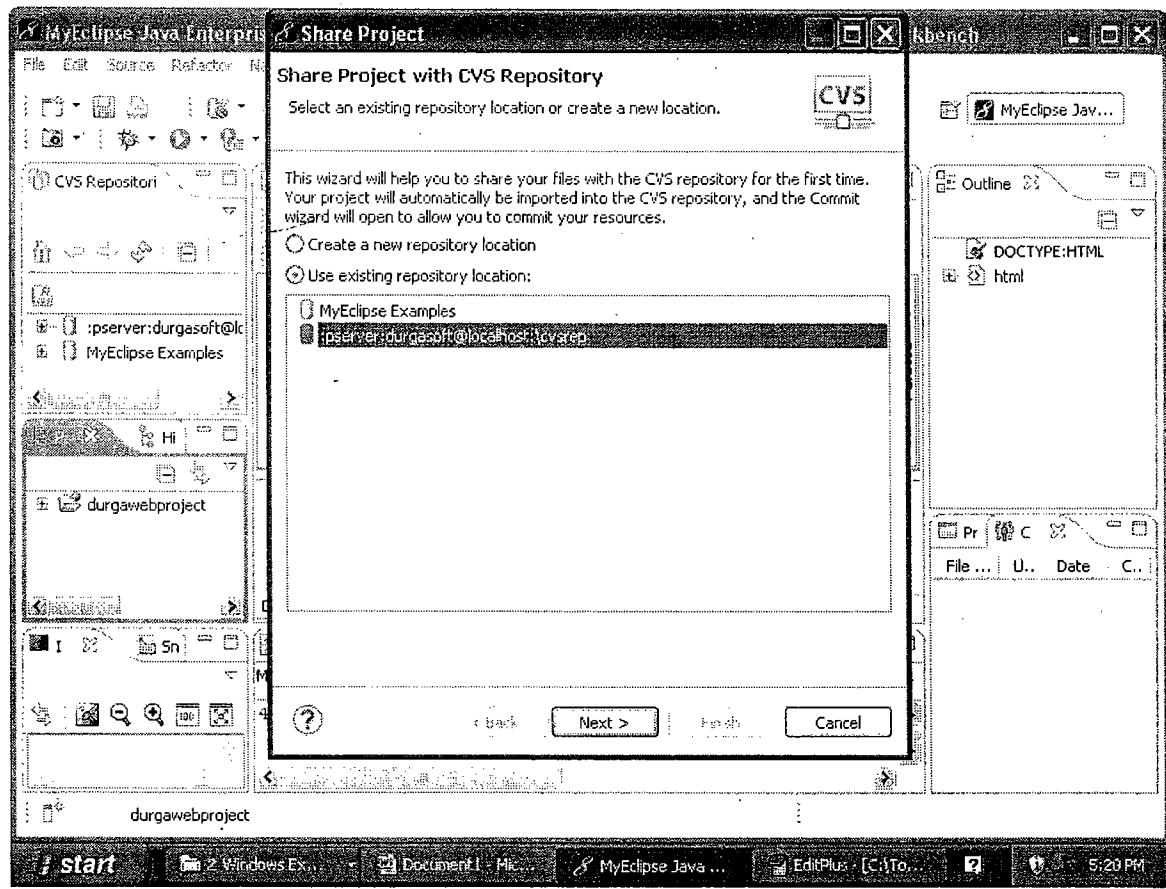
```
<body bgcolor="yellow">
<center>
<form action="./compute" method="post">
NumberOne<input type="text" name="t1"><br><br>
NumberTwo<input type="text" name="t2"><br><br>
<input type="submit" name="click" value="mul">
<input type="submit" name="click" value="div">
</form>
</center>
</body>
</html>
```

Step-3

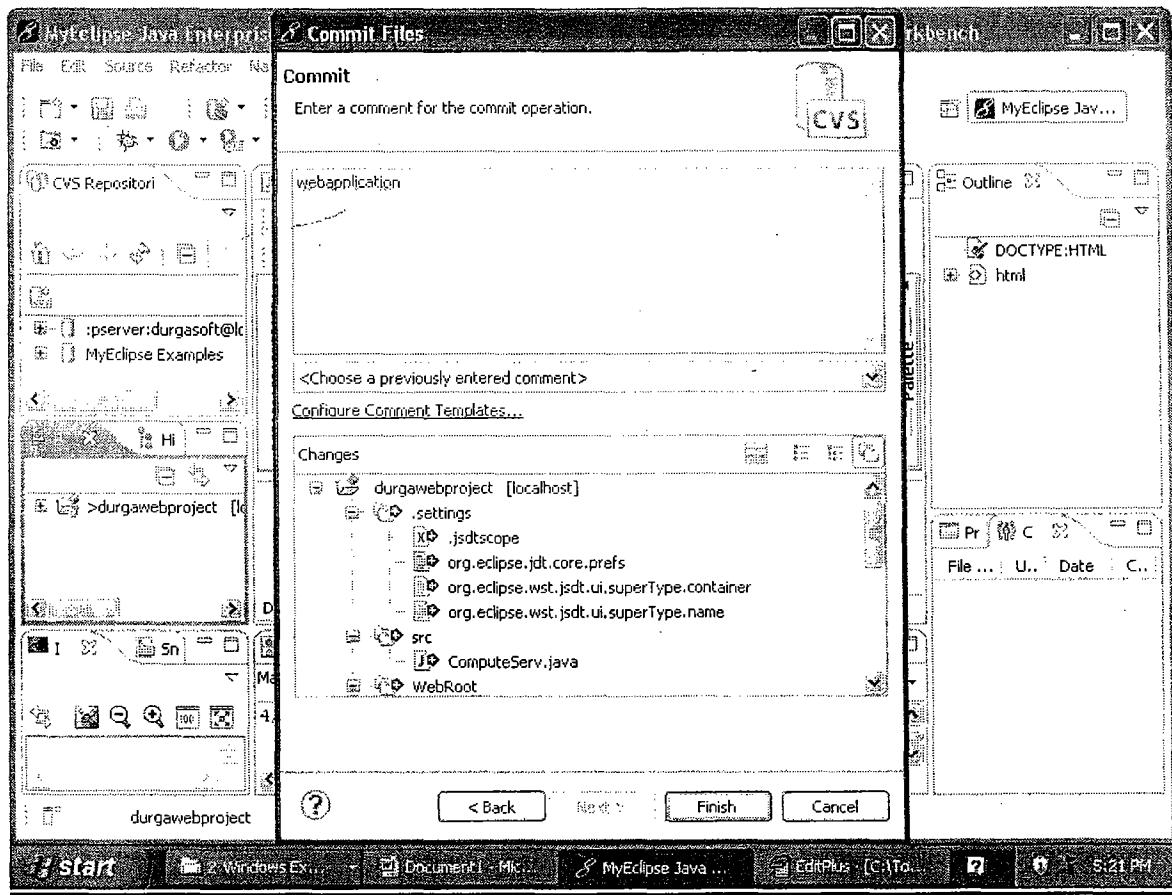
- >now share the project from client machine to repository location.
- Right click on project->team->shareproject->then automatically the project is sharing from client machine to server side machine.



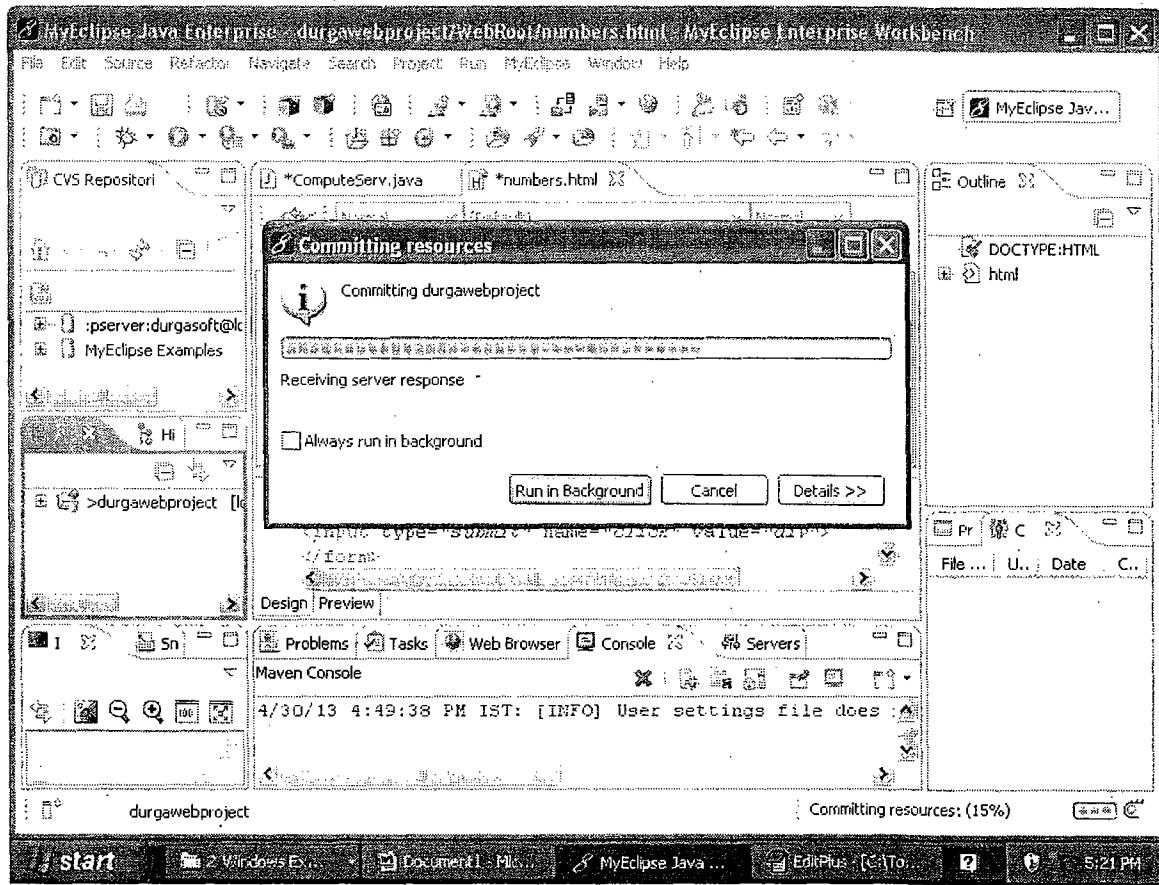
DURGA SOFTWARE SOLUTIONS
Tools Material



DURGA SOFTWARE SOLUTIONS
Tools Material



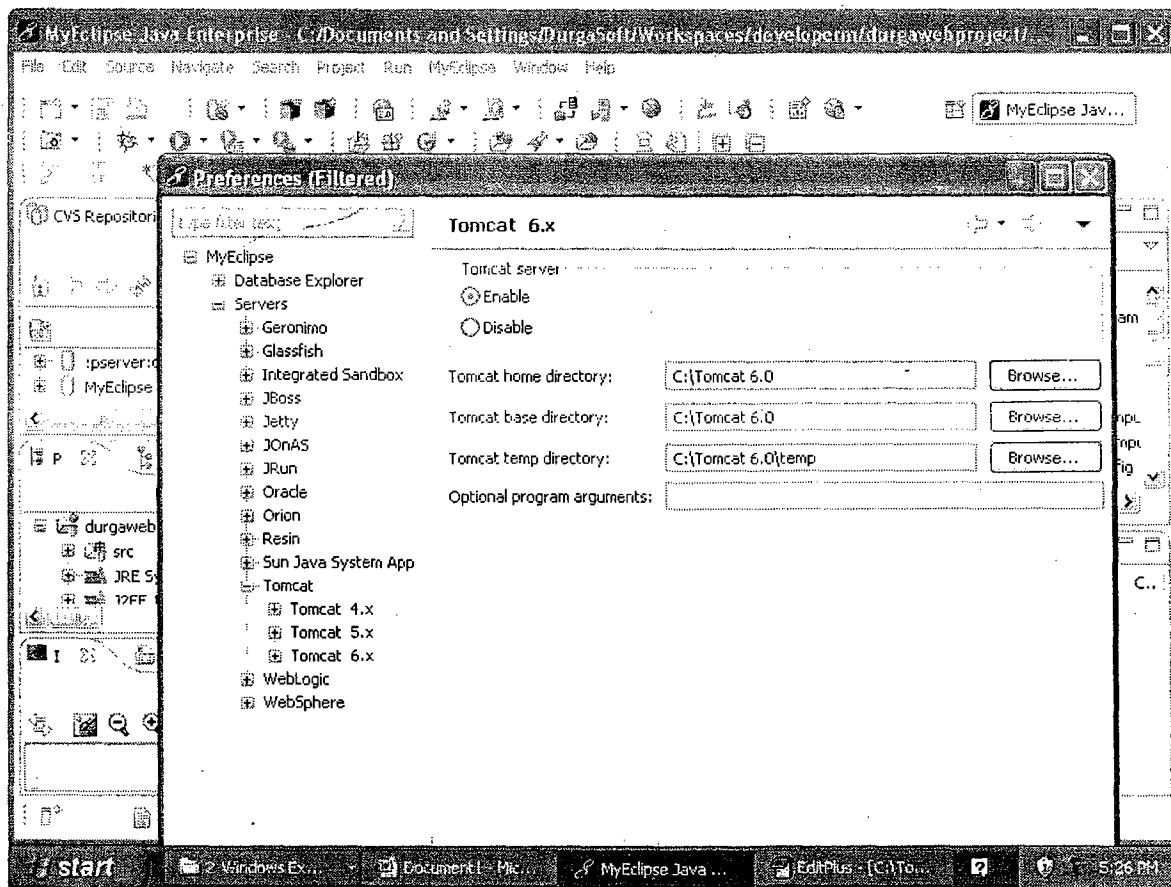
DURGA SOFTWARE SOLUTIONS
Tools Material



Step-4

Configure the server as follows.

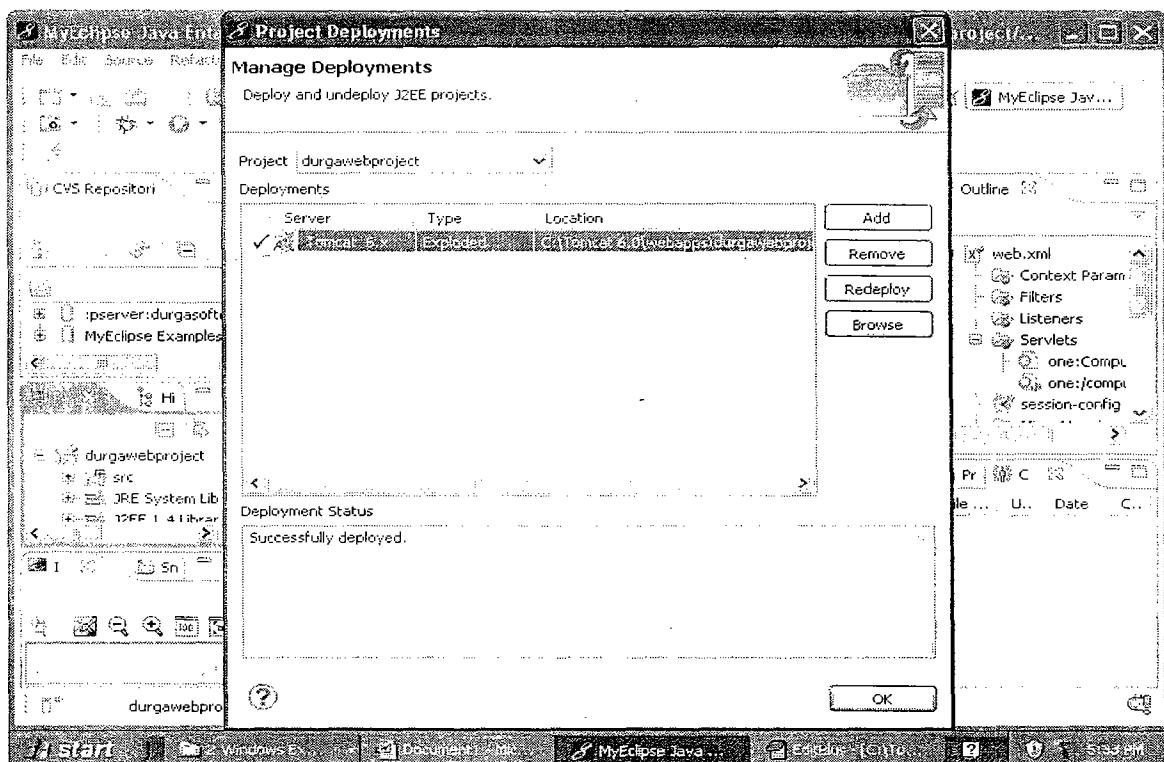
DURGA SOFTWARE SOLUTIONS
Tools Material



Step-5

Deploye the web application in tomcat server.

DURGA SOFTWARE SOLUTIONS
Tools Material



Step-6

Now start the server and testing the application.

Now open the internet explorer and make a request to web application.

DURGA SOFTWARE SOLUTIONS
Tools Material

MyEclipse Java Enterprise - MyEclipse Enterprise Workbench

File Edit Navigate Search Project Run MyEclipse Window Help

Problems Tasks Web Browser Console Servers

```
tomcat6Server [Remote Java Application] C:\Program Files\Genuitec\Commont\binary\com.sun.java.jdk.win32.x86_1.6.0_01\bin\javaw.exe (Apr 30, 2013 5:35:03 PM org.apache.catalina.startup.HostConfig deployDirectory)
INFO: Deploying web application directory querystringapp
Apr 30, 2013 5:35:03 PM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory ROOT
Apr 30, 2013 5:35:03 PM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory sample-project
Apr 30, 2013 5:35:03 PM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory userdefinedapp
Apr 30, 2013 5:35:03 PM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory welcome-project
Apr 30, 2013 5:35:03 PM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory welcome-service
Apr 30, 2013 5:35:03 PM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory welcome-service
Apr 30, 2013 5:35:03 PM org.apache.coyote.http11.Http11Protocol start
INFO: Starting Coyote HTTP/1.1 on http-2013
Apr 30, 2013 5:35:03 PM org.apache.jk.common.ChannelSocket init
INFO: JK: apc3 listening on /0.0.0.0:2013
Apr 30, 2013 5:35:03 PM org.apache.jk.server.JkMain start
INFO: JK running ID=0 timer=0/32 config=null
Apr 30, 2013 5:35:03 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 1569 ms
```

Start Taskbar MyEclipse Java ... EditPlus [C:\to... 5:35 PM

MyEclipse Java Enterprise - MyEclipse Web Browser - MyEclipse Enterprise Workbench

File Edit Navigate Search Project Run MyEclipse Window Help

CVS Repository Problems Tasks Web Browser Console Servers

numbers.html web.xml MyEclipse Web Browser

http://localhost:2013/

The Apache Software Foundation
The Apache http
If you're seeing this page via setup Tomcat succe
As you may have guessed by r
page. It can be found on the loc

Administration Status Tomcat Manager

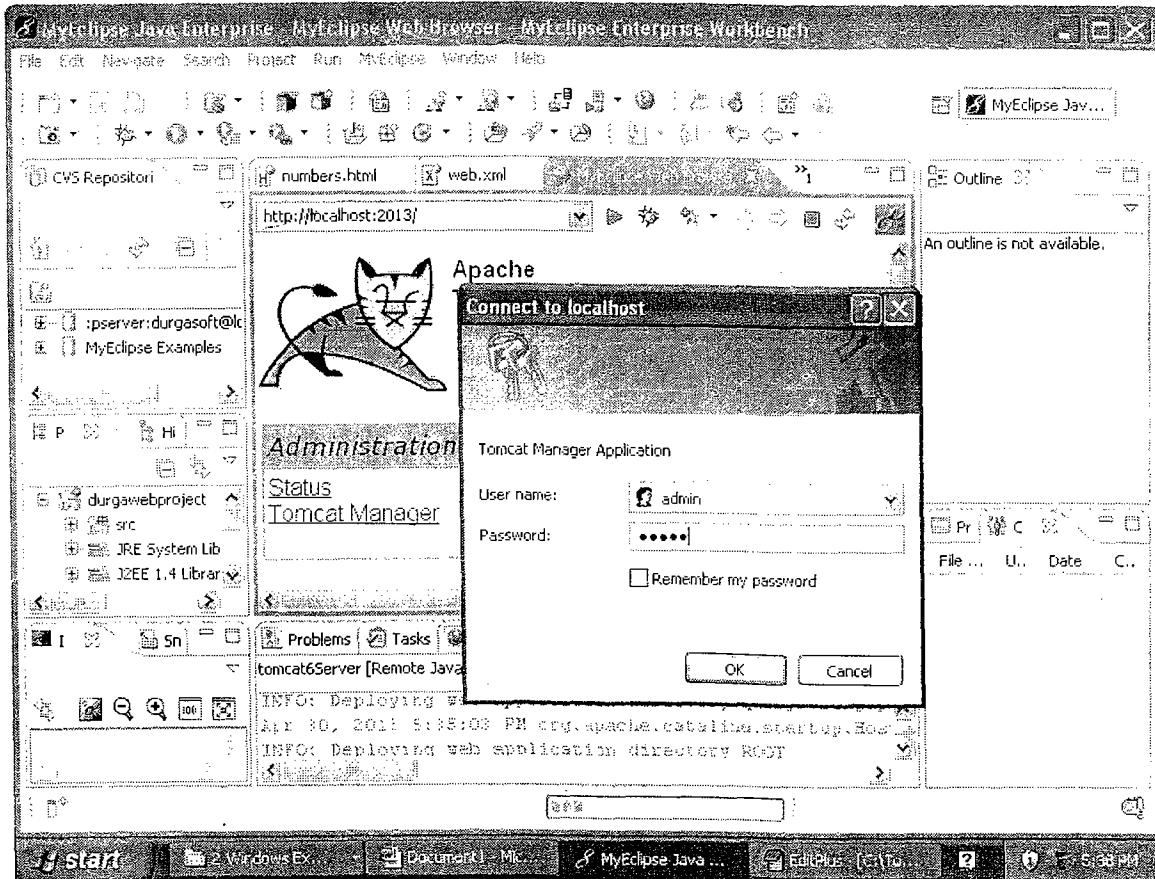
tomcat6Server [Remote Java Application]

```
INFO: Deploying web application directory querystringapp
Apr 30, 2013 5:35:03 PM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory ROOT
Apr 30, 2013 5:35:03 PM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory sample-project
Apr 30, 2013 5:35:03 PM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory userdefinedapp
Apr 30, 2013 5:35:03 PM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory welcome-project
Apr 30, 2013 5:35:03 PM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory welcome-service
Apr 30, 2013 5:35:03 PM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory welcome-service
Apr 30, 2013 5:35:03 PM org.apache.coyote.http11.Http11Protocol start
INFO: Starting Coyote HTTP/1.1 on http-2013
Apr 30, 2013 5:35:03 PM org.apache.jk.common.ChannelSocket init
INFO: JK: apc3 listening on /0.0.0.0:2013
Apr 30, 2013 5:35:03 PM org.apache.jk.server.JkMain start
INFO: JK running ID=0 timer=0/32 config=null
Apr 30, 2013 5:35:03 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 1569 ms
```

File ... U... Date C...

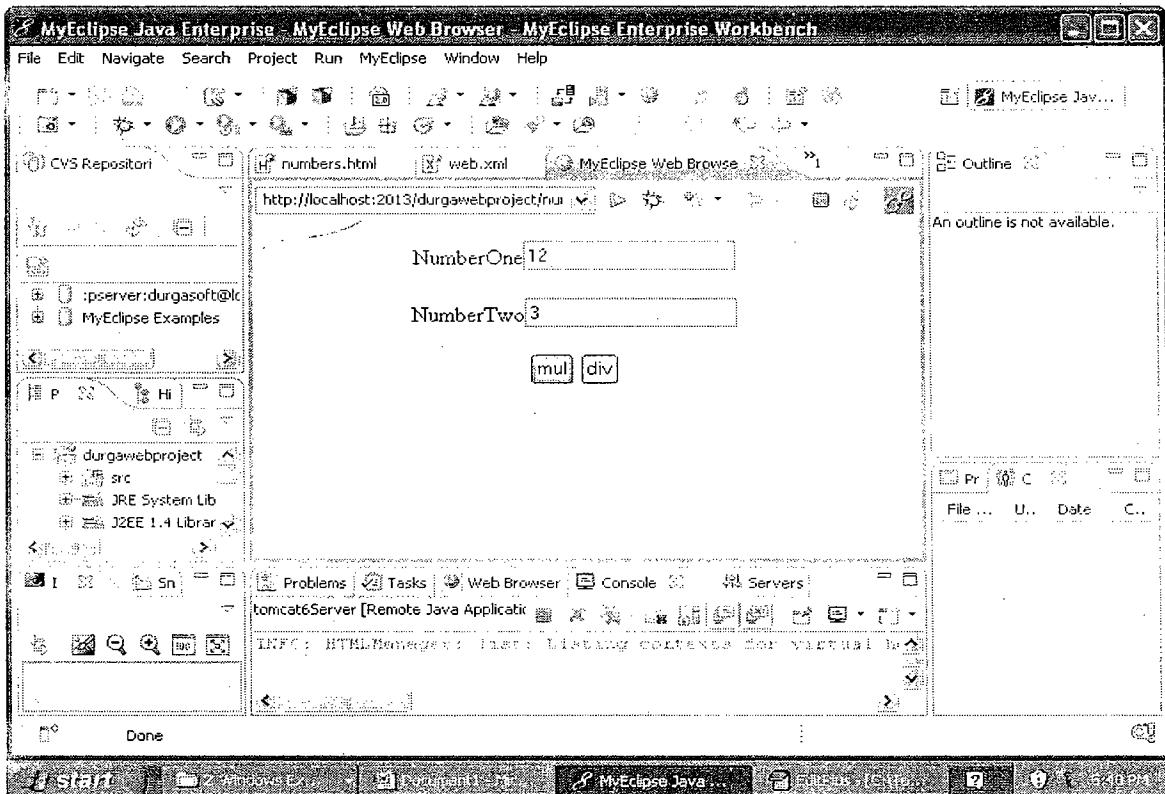
http://www.apache.org/

DURGA SOFTWARE SOLUTIONS
Tools Material

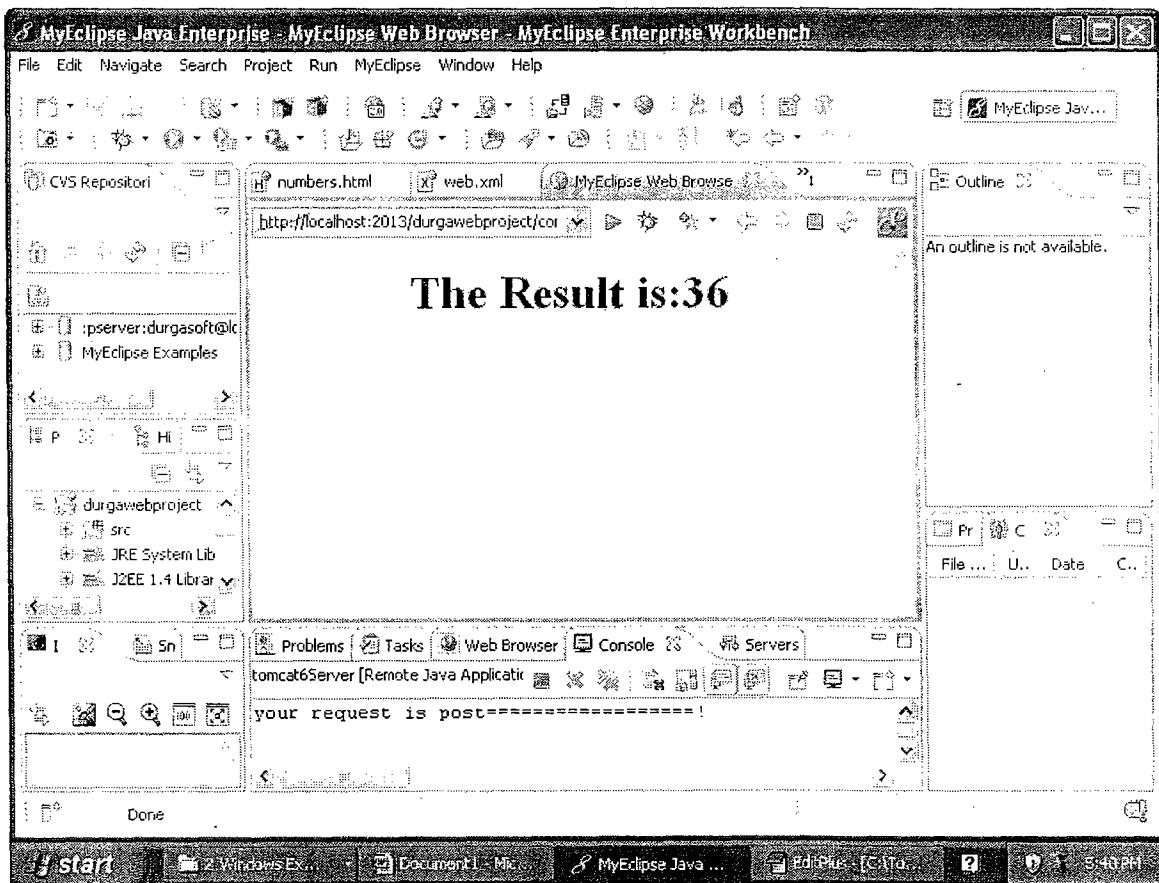


Result

DURGA SOFTWARE SOLUTIONS
Tools Material



DURGA SOFTWARE SOLUTIONS
Tools Material



Jasper Reports

Introduction:

Jasper Reports is a popular open-source reporting engine whose main purpose is to help creating page oriented, ready to print documents in a simple and flexible manner. Jasper Reports is written in 100% Java and can be embedded in any Java application. Jasper Reports has the ability to deliver rich content in various formats such as PDF, HTML, XLS, CSV, XML files, or directly on the screen or printer.

Jasper Reports is distributed under two licenses, an Apache-style license and the LGPL license. Due to its flexible licensing, JasperReports is the perfect candidate to complete the reporting features for any commercial or open-source application.

Steps To design Reports Based on Jasper Reports:

a) Preparing .jrxml file & compiling the .jrxml file:

Jasper Reports organizes data retrieved from a data source according to a report-design defined in a JRXML file. In order to fill a report with data, the report-design must be compiled first.

The compilation of the JRXML file representing the report-design is performed by the compileReport() method exposed by the JasperCompileManager class.

Through compilation, the report design is loaded into a report-design object that is then serialized and stored on disk (Jasper Report class). This serialized object is used when the application wants to fill the specified report-design with data. In fact, the compilation of a report-design implies the compilation of all Java expressions defined in the JRXML file representing the report design. Various verifications are made at compilation time, to check the report-design consistency. The result is a ready-to-fill report-design that will be used to generate documents on different sets of data.

DURGA SOFTWARE SOLUTIONS
Tools Material

Eg:

```
<!DOCTYPE jasperReport PUBLIC "-//JasperReports/DTD Report
Design//EN" "http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
<jasperReport name="Simple_Report">
    <detail> (body of the report directly).
        <band height="20">
            <staticText>
                <reportElement x="180" y="0" width="200" height="20"/>
                <text><![CDATA[welcometodurgasoftwareolutions]]></text>
            </staticText>
        </band>
    </detail>
</jasperReport>
```

- **<jasperReport>** - the root element.
- **<title>** - its contents are printed only once at the beginning of the report
- **<pageHeader>** - its contents are printed at the beginning of every page in the report.
- **<detail>** - contains the body of the report.
- **<pageFooter>** - its contents are printed at the bottom of every page in the report.
- **<band>** - defines a report section, all of the above elements contain a band element as its only child element.

b) Fill The Report:

In order to fill a report-design, one can use the `fillReportXXX()` methods exposed by the `JasperFillManager` class. Those methods receive as a parameter the report-design object, or a file representing the specified report-design object, in a serialized form, and also a JDBC connection to the database from which to retrieve the data to fill the report with. The result is an object that represents a ready-to-print document (`Jasper Print` class) and that can be stored on disk in a serialized

DURGA SOFTWARE SOLUTIONS
Tools Material

form (for later use), can be delivered to the printer or to the screen, or can be exported into a PDF, HTML, XLS, RTF, ODT, CSV, TXT or XML document.

As you can see, the main classes to use when working with JasperReports are:

`net.sf.jasperreports.engine.JasperCompileManager`
`net.sf.jasperreports.engine.JasperFillManager`
`net.sf.jasperreports.engine.JasperPrintManager`
`net.sf.jasperreports.engine.JasperExportManager`

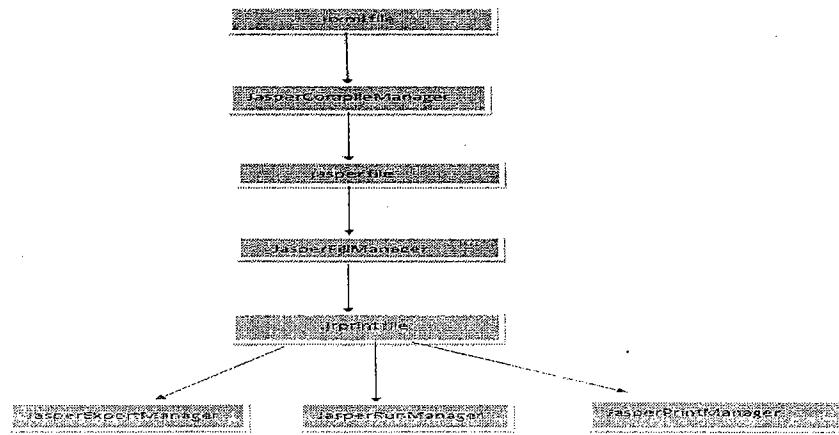
These classes represent a façade to the Jasper Reports engine. They have various static methods that simplify the access to the API functionality and can be used to compile an JRXML report design, to fill a report, to print it, or to export to other document formats (PDF, HTML, XML).

c) View, Print and Export Reports:

Generated reports can be viewed using the JasperViewer application. In its main () method, it receives the name of the file which contains the report to view.

Generated reports can be printed using the `printReport()`, `printPage()` or `printPages()` static methods exposed by the `JasperPrintManager` class.

After having filled a report, we can also export it in PDF, HTML or XML format using the `exportReportXXX()` methods of the `JasperExportManager` class.



Parameters:

DURGA SOFTWARE SOLUTIONS
Tools Material

Parameters are object references that are passed-in to the report filling operations. They are very useful for passing to the report engine data that it can not normally find in its data source. For example, we could pass to the report engine the name of the user that has launched the report filling operation if we want it to appear on the report, or we could dynamically change the title of our report.

An important aspect is the use of report parameters in the query string of the report, in order to be able to further customize the data set retrieved from the database. Those parameters could act like dynamic filters in the query that supplies data for the report.

Declaring a parameter in a report design is very simple and it requires specifying only its name and its class:

```
<parameter name="ReportTitle" class="java.lang.String"/>
<parameter name="MaxOrderID" class="java.lang.Integer"/>
<parameter name="SummaryImage" class="java.awt.Image"/>
```

Declare Parameter.

There are two possible ways to use parameters in the query:

1. The parameters are used like normal java.sql.PreparedStatement parameters using the following syntax:

```
SELECT * FROM Orders WHERE CustomerID = $P{OrderCustomer}
```

Using Parameter

2. Sometimes is useful to use parameters to dynamically modify portions of the SQL query or to pass the entire SQL query as a parameter to the report filling routines. In such a case, the syntax differs a little, like in the following example:

```
SELECT * FROM Orders ORDER BY $P!{OrderByClause}
```

↓

UN Parameter

We Supply Parameter values from java appn as Map<String, Object> placed On Data Source: fillxxxxx() method of FillManager.

DURGA SOFTWARE SOLUTIONS
Tools Material

JasperReports support various types of data sources using a special interface called JRDataSource.

There is a default implementation of this interface, the JRResultSetDataSource class, which wraps a java.sql.ResultSet object. It allows the use of any relational database through JDBC.

When using a JDBC data source, you could pass a java.sql.Connection object to the report filling operations and specify the query in the report definition itself (see the <queryString> element in the XML file) or could create a new instance of the JRResultSetDataSource by supplying the java.sql.ResultSet object directly.

With other types of data sources, things should not be different and all you have to do is to implement the JRDataSource interface, or use one of the implementations that are shipped with the JasperReports library to wrap in-memory collections or arrays of JavaBeans, CSV or XML files, etc.

Fields:

Report fields represent the only way to map data from the data source into the report generating routines. When the data source of the report is a java.sql.ResultSet, all fields must map to corresponding columns in the java.sql.ResultSet object. That is, they must have the same name as the columns they map and a compatible type.

We can define the following fields in our report design:

```
<field name="EmployeeID" class="java.lang.Integer"/>
<field name="LastName" class="java.lang.String"/>
<field name="FirstName" class="java.lang.String"/>
<field name="HireDate" class="java.util.Date"/>
```

} field declaration block
These names must match
with column names.

Expressions:

DURGA SOFTWARE SOLUTIONS
Tools Material

Expressions are a powerful feature of JasperReports. They can be used for declaring report variables that perform various calculations, for data grouping on the report, to specify report text fields content or to further customize the appearance of objects on the report.

Basically, all report expressions are Java expressions that can reference report fields and report variables.

In an XML report design there are several elements that define expressions: <variableExpression>, <initialValueExpression>, <groupExpression>, <printWhenExpression>, <imageExpression> and <textFieldExpression>.

In order to use a report field reference in an expression, the name of the field must be put between \$F{ and } character sequences.

For example, if we want to display in a text field, on the report, the concatenated values of two fields, we can define an expression like this one:

<textFieldExpression> → To display original values of column to depends on Expression.
\$F{FirstName} + " " + \$F{LastName}
</textFieldExpression> → column value1.

The expression can be even more complex:

```
<textFieldExpression>
$F{FirstName} + " " + $F{LastName} + " was hired on " +
(new SimpleDateFormat("MM/dd/yyyy")).format($F{HireDate}) + "."
                                         ↓
                                         column value2 will come.
</textFieldExpression>
```

To reference a variable in an expression, we must put the name of the variable between \$V{ and } like in the example below:

```
<textFieldExpression>
"Total quantity : " + $V{QuantitySum} + " kg."
</textFieldExpression>
```

→ To use column named in expression for displaying column values we must declare them as

DURGA SOFTWARE SOLUTIONS
Tools Material

There is an equivalent syntax for using parameters in expressions. The name of the parameter should be put between \$P{ and } like in the following example:

```
<textFieldExpression>  
"Max Order ID is : " + $P{MaxOrderID}  
</textFieldExpression>
```

Variable: (hold a value it comes through Expression).

A Report variable is a special object build on top of an expression. Variables can be used to simplify the report design by declaring only once an expression that is heavily used throughout the report design or to perform various calculations on the corresponding expressions.

In its expression, a variable can reference other report variables, but only if those referenced variables were previously defined in the report design. So the order in which the variables are declared in a report design is important.

As mentioned, variables can perform built-in types of calculations on their corresponding expression values like : count, sum, average, lowest, highest, variance, etc.

A variable that performs the sum of the Quantity field should be declared like this:

```
<variable name="QuantitySum"  
class="java.lang.Double" calculation="Sum">  
<variableExpression>$F{Quantity}</variableExpression>  
</variable>
```

what do you want perform

Type of variable

values

This variable QuantitySum holds
the sum of the Quantity column
values

For variables that perform calculation we can specify the level at which they are reinitialized. The default level is Report and it means that the variable is initialized only once at the beginning of the report and that it performs the specified calculation until the end of the report is reached. But we can choose a lower level of reset for our variables in order to perform calculation at page, column or group level.

DURGA SOFTWARE SOLUTIONS
Tools Material

For example, if we want to calculate the total quantity on each page, we should declare our variable like this:

```
<variable name="QuantitySum" class="java.lang.Double"  
resetType="Page" calculation="Sum">  
<variableExpression>$F{Quantity}</variableExpression>  
<initialValueExpression>new Double(0) </initialValueExpression>  
</variable>
```

Our variable will be initialized with zero at the beginning of each new page.

There are also the following built-in system variables, ready to use in expressions:

PAGE_NUMBER
COLUMN_NUMBER
REPORT_COUNT
PAGE_COUNT
COLUMN_COUNT

Predefined variable helping the programmers.

Charts:

JasperReports now has built-in support for charts. There is a new, ready-to-use chart component, although we already had images, text fields, subreports and other elements. This greatly simplifies the way charts are included inside reports, because previously the user had to completely rely on scriptlets in order to gather the chart data and render the chart using an image element in the report template.

Now with the new chart component, the user only has to make the desired visual settings and define the expressions that will help the engine build up the chart dataset in an incremental fashion during the iteration through the report data source.

JasperReports currently supports the following types of charts:

DURGA SOFTWARE SOLUTIONS

Tools Material

Pie, Pie 3D, Bar, Bar 3D, XY Bar, Stacked Bar, Stacked Bar3D, Line, XY Line, Area, XY Area, Scatter Plot, Bubble, Time series, High Low Open Close, Candlestick.

These types of charts use several types of datasets (each type of chart works with certain types of datasets): **Pie Dataset, Category Dataset, XY Dataset, Time Series, Time Period Values, XYZ Dataset, High Low Dataset.**

For all charts we can configure the following:

- border around all sides
- background color
- title
- title position (top, left, bottom, right)
- title font
- title color
- subtitle
- subtitle font
- subtitle color
- show/hide legend
- plot area background color
- plot area background transparency (alpha)
- plot area foreground transparency (alpha)
- plot orientation (vertical, horizontal)
- axis labels

For all datasets we can configure:

- increment type (detail, column, page, group, report)
- increment group
- reset type (none, column, page, group, report)
- reset group

Specific settings by chart type:

DURGA SOFTWARE SOLUTIONS
Tools Material

- **Pie 3D**
- **depth factor**
- **Bar, XY Bar, Stacked Bar**
- **hide/show labels**
- **hide/show tick marks**
- **hide/show tick labels**
- **Bar 3D, Stacked Bar 3D**
- **hide/show labels**
- **x offset (3D effect)**
- **y offset (3D effect)**
- **Line, XY Line, Scatter Plot, Time series**
- **hide/show lines**
- **hide/show shapes**
- **Bubble**
- **scale type (both axes, domain axis, range axis)**
- **High Low Open Close**
- **hide/show close ticks**
- **hide/show open ticks**
- **Candlestick**
- **hide/show volume**

1. Programs:

2. App1

3. -----FirstReport.jrxml-----
4. <?xml version="1.0" encoding="UTF-8"?>
5. <!DOCTYPE jasperReport PUBLIC "-//JasperReports//DTD Report Design//EN"
6. "http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
7. <jasperReport name="FirstReport">
8. <detail>
9. <band height="60">
 - i. <staticText>

DURGA SOFTWARE SOLUTIONS
Tools Material

```
1. <reportElement x="20" y="0" width="200"
   height="40" forecolor="#FF3300"
   backcolor="#FFFFFF"/>
2. <text><![CDATA[Welcome To
DurgaSoftwareSolutions]]></text>
ii. </staticText>
10. </band>
11. </detail>
12. </jasperReport>

13. -----FirstReport.java-----
14. import java.util.*;
15. import net.sf.jasperreports.engine.*;
16. public class FirstReport
17. {
18. public static void main(String[] args)
19. {
i. try{
1. System.out.println("compiling report starts.....");
2. JasperCompileManager.compileReportToFile("FirstRe
port.jrxml");
3. System.out.println("compilation done.....");
4. System.out.println("data filling starts.....");
5. JasperFillManager.fillReportToFile("FirstReport.jaspe
r",new HashMap(),new JREmptyDataSource());
6. System.out.println("data filling done.....");
7. System.out.println("generate the report byusing
exportmanager.....");
8. //JasperRunManager.runReportToHtmlFile("FirstRep
ort.jasper",new HashMap(),new
JREmptyDataSource());
9. JasperRunManager.runReportToPdfFile("FirstReport.
jasper",new HashMap(),new JREmptyDataSource());
10. System.out.println("generate the report byusing
exportmanager is done.....");
ii. }
iii. catch(Exception e){
1. e.printStackTrace();
iv. }
20. }
21. }
```

DURGA SOFTWARE SOLUTIONS
Tools Material

22. App2

1. <?xml version="1.0" encoding="UTF-8"?>
2. <jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports http://jasperreports.sourceforge.net/xsd/jasperreport.xsd" name="ThirdReport" language="groovy" pageWidth="595" pageHeight="842" columnWidth="535" leftMargin="20" rightMargin="20" topMargin="20" bottomMargin="20">
3. <property name="ireport.zoom" value="1.0"/>
4. <property name="ireport.x" value="0"/>
5. <property name="ireport.y" value="0"/>
6. <style name="Title" forecolor="#FFFFFF" fontName="Times New Roman" fontSize="50" isBold="false" pdfFontName="Times-Bold"/>
7. <style name="SubTitle" forecolor="#CCCCCC" fontName="Times New Roman" fontSize="18" isBold="false" pdfFontName="Times-Roman"/>
8. <style name="Column header" forecolor="#666666" fontName="Times New Roman" fontSize="14" isBold="true" pdfFontName="Times-Roman"/>
9. <style name="Detail" mode="Transparent" fontName="Times New Roman" pdfFontName="Times-Roman"/>
10. <style name="Row" mode="Transparent" fontName="Times New Roman" pdfFontName="Times-Roman">
 - i. <conditionalStyle>
 1. <conditionExpression><![CDATA[\$V{REPORT_COU NT} % 2 == 0]]></conditionExpression>
 2. <style mode="Opaque" backcolor="#F0EFEF"/>
 - ii. </conditionalStyle>
11. </style>
12. <parameter name="ReportTitle" class="java.lang.String"/>
13. <field name="ENO" class="java.math.BigDecimal"/>
14. <field name="ENAME" class="java.lang.String"/>
15. <field name="ESAL" class="java.math.BigDecimal"/>
16. <field name="EADDRESS" class="java.lang.String"/>
17. <field name="EMOBILE" class="java.lang.String"/>
18. <field name="EMAIL" class="java.lang.String"/>
19. <title>
 - i. <band height="136" splitType="Stretch">
 1. <textField>
 - a. <reportElement style="SubTitle" x="200" y="29" width="196" height="22"/>

DURGA SOFTWARE SOLUTIONS

Tools Material

- b. <textElement textAlignment="Center">
 - i.
 - c. </textElement>
 - d. <textFieldExpression class="java.lang.String">\$P{ReportTitle}</textFieldExpression>
2. </textField>
- ii. </band>
20. </title>
- i. <columnHeader>
 - ii. <band height="26" splitType="Stretch">
 - 1. <staticText>
 - a. <reportElement style="Column header" x="0" y="7" width="92" height="18" forecolor="#000000"/>
 - b. <textElement>
 - i.
 - c. </textElement>
 - d. <text><![CDATA[ENO]]></text>
 - 2. </staticText>
 - 3. <staticText>
 - a. <reportElement style="Column header" x="92" y="7" width="92" height="18" forecolor="#000000"/>
 - b. <textElement>
 - i.
 - c. </textElement>
 - d. <text><![CDATA[ENAME]]></text>
 - 4. </staticText>
 - 5. <staticText>
 - a. <reportElement style="Column header" x="184" y="7" width="92" height="18" forecolor="#000000"/>
 - b. <textElement>
 - i.
 - c. </textElement>
 - d. <text><![CDATA[ESAL]]></text>
 - 6. </staticText>
 - 7. <staticText>
 - a. <reportElement style="Column header" x="276" y="7" width="92" height="18" forecolor="#000000"/>
 - b. <textElement>

DURGA SOFTWARE SOLUTIONS

Tools Material

- i.
- c. </textElement>
- d. <text><![CDATA[EADDRESS]]></text>
- 8. </staticText>
- 9. <staticText>
 - a. <reportElement style="Column header" x="368" y="7" width="92" height="18" forecolor="#000000"/>
 - b. <textElement>
 - i.
 - c. </textElement>
 - d. <text><![CDATA[EMOBILE]]></text>
- 10. </staticText>
- 11. <staticText>
 - a. <reportElement style="Column header" x="460" y="7" width="92" height="18" forecolor="#000000"/>
 - b. <textElement>
 - i.
 - c. </textElement>
 - d. <text><![CDATA[EMAIL]]></text>
- 12. </staticText>
- iii. </band>
- 21. </columnHeader>
- 22. <detail>
 - i. <band height="18" splitType="Stretch">
 - 1. <frame>
 - a. <reportElement style="Row" mode="Opaque" x="0" y="0" width="555" height="18"/>
 - b. <textField isStretchWithOverflow="true">
 - i. <reportElement style="Detail" positionType="Float" x="0" y="0" width="92" height="18"/>
 - ii. <textElement>
 - 1.
 - iii. </textElement>
 - iv. <textFieldExpression class="java.math.BigDecimal"><![CDATA[A[\$F{ENO}]]></textFieldExpression>
 - c. </textField>
 - d. <textField isStretchWithOverflow="true">
 - i. <reportElement style="Detail" positionType="Float" x="92" y="0" width="92" height="18"/>
 - ii. <textElement>
 - 1.

DURGA SOFTWARE SOLUTIONS
Tools Material

```
iii. </textElement>
iv. <textFieldExpression
    class="java.lang.String"><![CDATA[$F{
    ENAME}]]></textFieldExpression>
e. </textField>
f. <textField isStretchWithOverflow="true">
    i. <reportElement style="Detail"
        positionType="Float" x="184" y="0"
        width="92" height="18"/>
    ii. <textElement>
        1. <font size="14"/>
    iii. </textElement>
    iv. <textFieldExpression
        class="java.math.BigDecimal"><![CDATA
        A[$F{ESAL}]]></textFieldExpression>
g. </textField>
h. <textField isStretchWithOverflow="true">
    i. <reportElement style="Detail"
        positionType="Float" x="276" y="0"
        width="92" height="18"/>
    ii. <textElement>
        1. <font size="14"/>
    iii. </textElement>
    iv. <textFieldExpression
        class="java.lang.String"><![CDATA[$F{
        EADDRESS}]]></textFieldExpression>
i. </textField>
j. <textField isStretchWithOverflow="true">
    i. <reportElement style="Detail"
        positionType="Float" x="368" y="0"
        width="92" height="18"/>
    ii. <textElement>
        1. <font size="14"/>
    iii. </textElement>
    iv. <textFieldExpression
        class="java.lang.String"><![CDATA[$F{
        EMOBILE}]]></textFieldExpression>
k. </textField>
l. <textField isStretchWithOverflow="true">
    i. <reportElement style="Detail"
        positionType="Float" x="460" y="0"
        width="92" height="18"/>
    ii. <textElement>
        1. <font size="14"/>
    iii. </textElement>
```

DURGA SOFTWARE SOLUTIONS

Tools Material

- iv. <textFieldExpression class="java.lang.String"><![CDATA[\$F{EMAIL}]]></textFieldExpression>
- m. </textField>
- 2. </frame>
- ii. </band>
- 23. </detail>
- 24. <columnFooter>
 - i. <band height="7" splitType="Stretch">
 - 1. <line>
 - a. <reportElement positionType="FixRelativeToBottom" x="0" y="3" width="555" height="1"/>
 - b. <graphicElement>
 - i. <pen lineWidth="0.5" lineColor="#999999"/>
 - c. </graphicElement>
 - 2. </line>
 - ii. </band>
- 25. </columnFooter>
- 26. <pageFooter>
 - i. <band height="25" splitType="Stretch">
 - 1. <frame>
 - a. <reportElement mode="Opaque" x="0" y="1" width="555" height="24" forecolor="#D0B48E" backcolor="#000000"/>
 - b. <textField evaluationTime="Report">
 - i. <reportElement style="Column header" x="513" y="0" width="40" height="20" forecolor="#FFFFFF"/>
 - ii. <textElement verticalAlignment="Middle">
 - 1.
 - iii. </textElement>
 - iv. <textFieldExpression class="java.lang.String"><![CDATA[" " + \$V{PAGE_NUMBER}]]></textFieldExpression>
 - c. </textField>
 - d. <textField>
 - i. <reportElement style="Column header" x="433" y="0" width="80" height="20" forecolor="#FFFFFF"/>
 - ii. <textElement textAlignment="Right" verticalAlignment="Middle">

DURGA SOFTWARE SOLUTIONS

Tools Material

```
1. <font size="10" isBold="false"/>
iii. <textElement>
iv. <textFieldExpression
    class="java.lang.String"><![CDATA[Pa
ge "+$V{PAGE_NUMBER}+"of"]]></textFieldExpression>
e. </textField>
f. <textField pattern="EEEEEE dd MMMMM
yyyy">
    i. <reportElement style="Column header"
        x="2" y="1" width="197" height="20"
        forecolor="#FFFFFF"/>
    ii. <textElement
        verticalAlignment="Middle">
        1. <font size="10" isBold="false"/>
    iii. </textElement>
    iv. <textFieldExpression
        class="java.util.Date"><![CDATA[new
        java.util.Date()]]></textFieldExpression>
g. </textField>
2. </frame>
ii. </band>
27. </pageFooter>
28. </jasperReport>
```

JasperExportManager.java

```
import java.io.*;
import net.sf.jasperreports.engine.*;
import net.sf.jasperreports.engine.export.*;
import net.sf.jasperreports.engine.export.oasis.*;
import net.sf.jasperreports.engine.export.ooxml.*;
import net.sf.jasperreports.engine.util.*;
import java.util.*;
import java.sql.*;
import com.lowagie.text.pdf.*;
import javax.servlet.http.*;
import javax.servlet.*;
import net.sf.jasperreports.view.*;
import net.sf.jasperreports.engine.JRResultSetDataSource;

public class JasperExportManager extends HttpServlet
{
    String filename;
    JRExporter exporter;
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
String option;
public void doGet(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException
{
    OutputStream outputStream = res.getOutputStream();
    option=req.getParameter("format");

    Map m=new HashMap();
    m.put("ReportTitle", "Employee Details");

    try{
        //filename="D:/jasperreports/ThirdReport.jrxml";

        filename=getServletContext().getRealPath("/")+"ThirdReport.jrxml";
        JasperCompileManager.compileReportToFile(filename);

        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott
","tiger");
        Statement st=con.createStatement();
        ResultSet rs=st.executeQuery("select * from emp");
        JRResultSetDataSource resultSetDataSource = new
JRResultSetDataSource(rs);

        //filename="D:/jasperreports/ThirdReport.jasper";

        filename=getServletContext().getRealPath("/")+"ThirdReport.jasper";
        JasperPrint
jasperPrint=JasperFillManager.fillReport(filename,m,resultSetDataSource);

        if(option.equalsIgnoreCase("pdf"))
        {
            res.setContentType("application/pdf");
            res.setHeader("Content-Disposition", "attachment;
filename=\"employee.pdf\"");
            exporter = new JRPdfExporter();
            exporter.setParameter(JRExporterParameter.JASPER_PRINT,
jasperPrint);
            exporter.setParameter(JRExporterParameter.OUTPUT_STREAM,
outputStream);
        }
    }
}
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
    exporter.setParameter(JRPdfExporterParameter.IS_ENCRYPTED,
Boolean.TRUE);

    exporter.setParameter(JRPdfExporterParameter.IS_128_BIT_KEY,
Boolean.TRUE);

    exporter.setParameter(JRPdfExporterParameter.USER_PASSWORD,
"durga");

    exporter.setParameter(JRPdfExporterParameter.OWNER_PASSWORD,
"reports");

    exporter.setParameter(
JRPdfExporterParameter.PERMISSIONS,
new Integer(PdfWriter.ALLOW_COPY |
PdfWriter.ALLOW_PRINTING));
}

else if(option.equalsIgnoreCase("csv"))
{
    res.setContentType("application/csv");
    res.setHeader("Content-Disposition", "attachment;
filename=\"employee.csv\"");
    exporter = new JRCsvExporter();
    exporter.setParameter(JRExporterParameter.JASPER_PRINT,
jasperPrint);
    exporter.setParameter(JRExporterParameter.OUTPUT_STREAM,
outputStream);
}

else if(option.equalsIgnoreCase("rtf"))
{
    res.setContentType("application/rtf");
    res.setHeader("Content-Disposition", "attachment;
filename=\"employee.csv\"");
    exporter = new JRRtfExporter();
    exporter.setParameter(JRExporterParameter.JASPER_PRINT,
jasperPrint);
    exporter.setParameter(JRExporterParameter.OUTPUT_STREAM,
outputStream);
}

else if(option.equalsIgnoreCase("odt"))
{

    res.setContentType("application/vnd.oasis.opendocument.text");
}
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
res.setHeader("Content-Disposition", "attachment;
filename=\"employee.odt\"");
exporter = new JROdtExporter();
exporter.setParameter(JRExporterParameter.JASPER_PRINT,
jasperPrint);
exporter.setParameter(JRExporterParameter.OUTPUT_STREAM,
outputStream);
}

else if(option.equalsIgnoreCase("ods"))
{

res.setContentType("application/vnd.oasis.opendocument.spreadsheet");
res.setHeader("Content-Disposition", "attachment;
filename=\"employee.ods\"");
exporter = new JROdsExporter();
exporter.setParameter(JRExporterParameter.JASPER_PRINT,
jasperPrint);
exporter.setParameter(JRExporterParameter.OUTPUT_STREAM,
outputStream);
}

else if(option.equalsIgnoreCase("xlsx"))
{
    res.setContentType("application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet ");
    res.setHeader("Content-Disposition", "attachment;
filename=\"employee.xlsx\"");
    exporter = new JRXlsxExporter();
    exporter.setParameter(JRExporterParameter.JASPER_PRINT,
jasperPrint);
    exporter.setParameter(JRExporterParameter.OUTPUT_STREAM,
outputStream);
}

else if(option.equalsIgnoreCase("docx"))
{
    res.setContentType("application/vnd.openxmlformats-
officedocument.wordprocessingml.document ");
    res.setHeader("Content-Disposition", "attachment;
filename=\"employee.docx\"");
    exporter = new JRDocxExporter();
    exporter.setParameter(JRExporterParameter.JASPER_PRINT,
jasperPrint);
    exporter.setParameter(JRExporterParameter.OUTPUT_STREAM,
outputStream);
}
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
    }

    else if(option.equalsIgnoreCase("xhtml"))
    {
        res.setContentType("application/xhtml+xml ");
        res.setHeader("Content-Disposition", "attachment;
filename=\"employee.xhtml\"");
        exporter = new JRXhtmlExporter();
        exporter.setParameter(JREporterParameter.JASPER_PRINT,
jasperPrint);
        exporter.setParameter(JREporterParameter.OUTPUT_STREAM,
outputStream);
    }

    else if(option.equalsIgnoreCase("html"))
    {
        res.setContentType("text/html ");
        res.setHeader("Content-Disposition", "attachment;
filename=\"employee.html\"");
        exporter = new JRHtmlExporter();
        exporter.setParameter(JREporterParameter.JASPER_PRINT,
jasperPrint);
        exporter.setParameter(JREporterParameter.OUTPUT_STREAM,
outputStream);
    }

    else if(option.equalsIgnoreCase("pptx"))
    {
        res.setContentType("application/vnd.openxmlformats-
officedocument.presentationml.presentation");
        res.setHeader("Content-Disposition", "attachment;
filename=\"employee.pptx\"");
        exporter = new JRPptxExporter();
        exporter.setParameter(JREporterParameter.JASPER_PRINT,
jasperPrint);
        exporter.setParameter(JREporterParameter.OUTPUT_STREAM,
outputStream);
    }

    else if(option.equalsIgnoreCase("xls"))
    {
        res.setContentType("application/vnd.ms-excel");
        res.setHeader("Content-Disposition", "attachment;
filename=\"employee.xls\"");
        ByteArrayOutputStream bout=new ByteArrayOutputStream();
        exporter = new JRXlsExporter();
    }
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
    exporter.setParameter(JRExporterParameter.JASPER_PRINT,  
jasperPrint);  
    exporter.setParameter(JRExporterParameter.OUTPUT_STREAM, bout);  
        outputStream.write(bout.toByteArray());  
    }  
  
    else if(option.equalsIgnoreCase("xml"))  
    {  
        res.setContentType("text/xml");  
        res.setHeader("Content-Disposition", "attachment;  
filename=\"employee.jrpxml\"");  
        exporter = new JRXmlExporter();  
        exporter.setParameter(JRExporterParameter.JASPER_PRINT,  
jasperPrint);  
        exporter.setParameter(JRExporterParameter.OUTPUT_STREAM,  
outputStream);  
    }  
  
    else if(option.equalsIgnoreCase("view"))  
    {  
        JasperViewer.viewReport(jasperPrint);  
    }  
  
    catch(Exception e){  
        e.printStackTrace();  
    }  
    try{  
        exporter.exportReport();  
    }  
    catch(Exception e){  
        e.printStackTrace();  
    }  
    finally{  
        if (outputStream != null) {  
try {  
            outputStream.close();  
        }  
        catch (IOException ex) {  
            ex.printStackTrace();  
        }  
    }  
    }  
}
```

DURGA SOFTWARE SOLUTIONS
Tools Material

I Reports Tool:

iReport is a program that helps users and developers that use JasperReports library to visually design reports.

Through a rich and very simple to use GUI, iReport provide all the most important functions to create nice reports in a little time.

iReport can help people that don't know JasperReports library to create complex reports and learn the XML syntax taking a look to the generated code.

iReport can help skilled report designer to compose very complex page saving a lot of time.

iReport is written in java. From the version 0.2.0 it was totally rewritten. For this reason there are two manuals for iReport. The new development direction was taken for a lot of reasons. iReport has lost a bit of efficiency leaving the win32 native GUI for a clear, pure swing interface. But this is the right direction...

1 Set Up To Launch iReport-3.7.5 Tool

Step1:

Start→Programs→Jasper Soft→iReports-3.7.5→iReport3.7.5

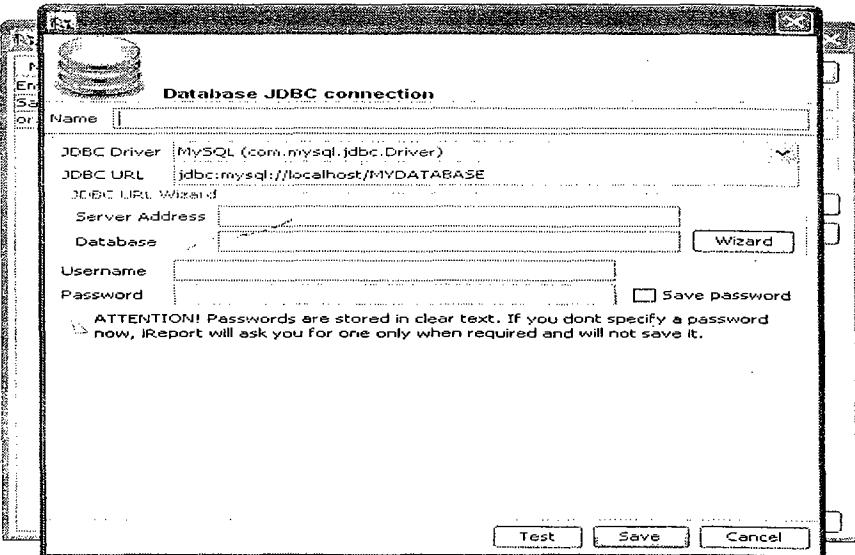
Step2:

Tools→Options→iReport→ClassPath→AddJar→Select ojdbc14.jar→Open→Ok

Step3:

Report DataSource  (symbol shown in documentation)
→new→DataBaseJdbcConnection→next→

DURGA SOFTWARE SOLUTIONS
Tools Material



(provide the names as shown in above)

Name:orads

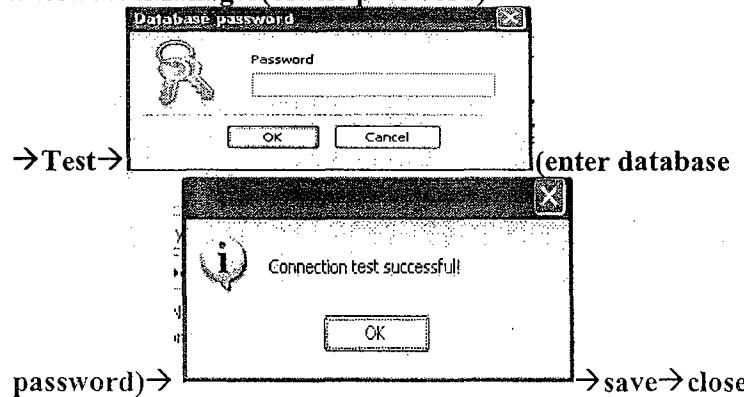
JdbcDriver:Oracle(oracle.jdbc.driver.OracleDriver)

JdbcUrl:jdbc:oracle:thin:@localhost:1521:xe

ServlerAddress:localhost

UserName:system(oracle username)

Password:Manager(oracle password)



Procedure to design report byusing iReport tool:

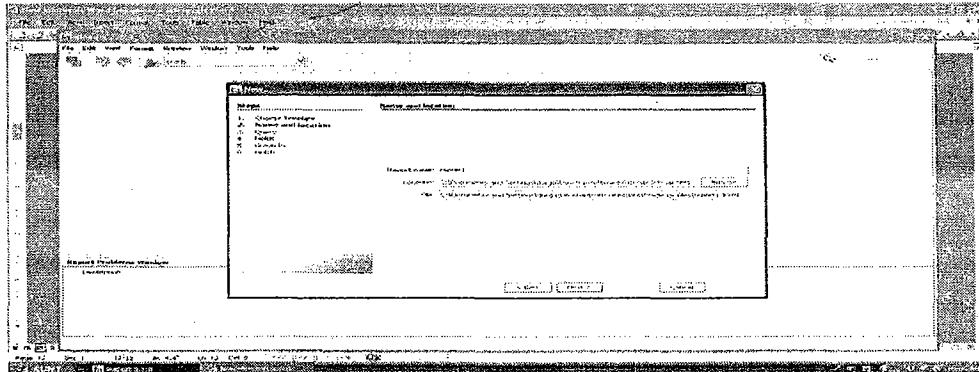
Step1:

Start→Programs→Jasper Soft→iReports-3.7.5→iReport3.7.5

DURGA SOFTWARE SOLUTIONS
Tools Material

Step2: launch report design template

File→new→Report→choose some template(eg:coffee landscape)→launch report wizard→



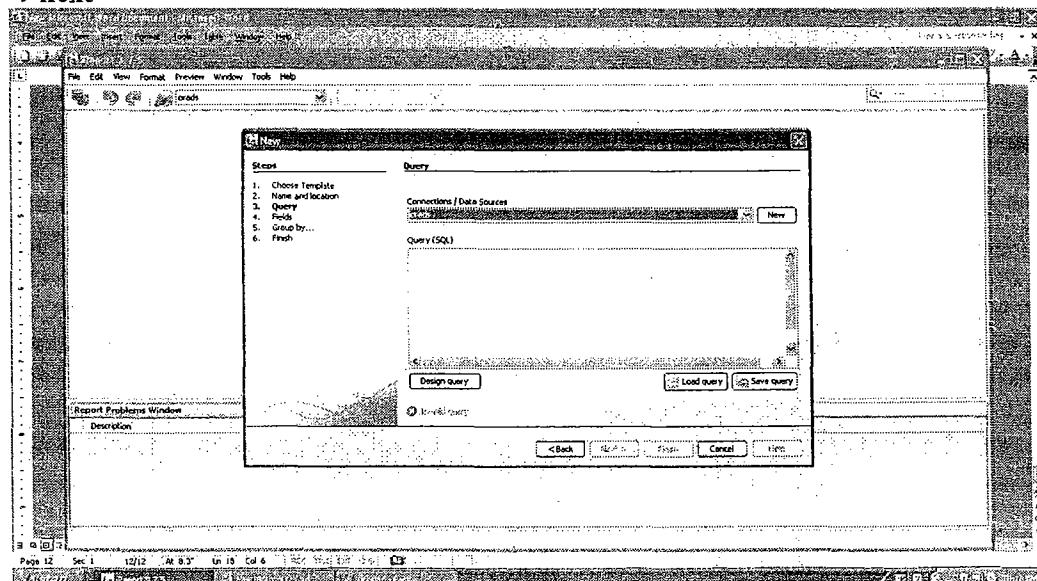
Provide the data in the fields:

Report name:Report1(some name)

Location: C:\Documents and Settings\durga\My Documents\Downloads (some location)

(in this location only .jrxml file will be saved)

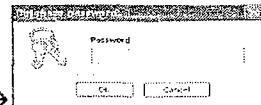
→next



Provide the in the fields:

Connections/datasources:orads

Query(sql):select * from emp(some sql query)→next→ enter the data base password→ok→select the filds we want to display on the

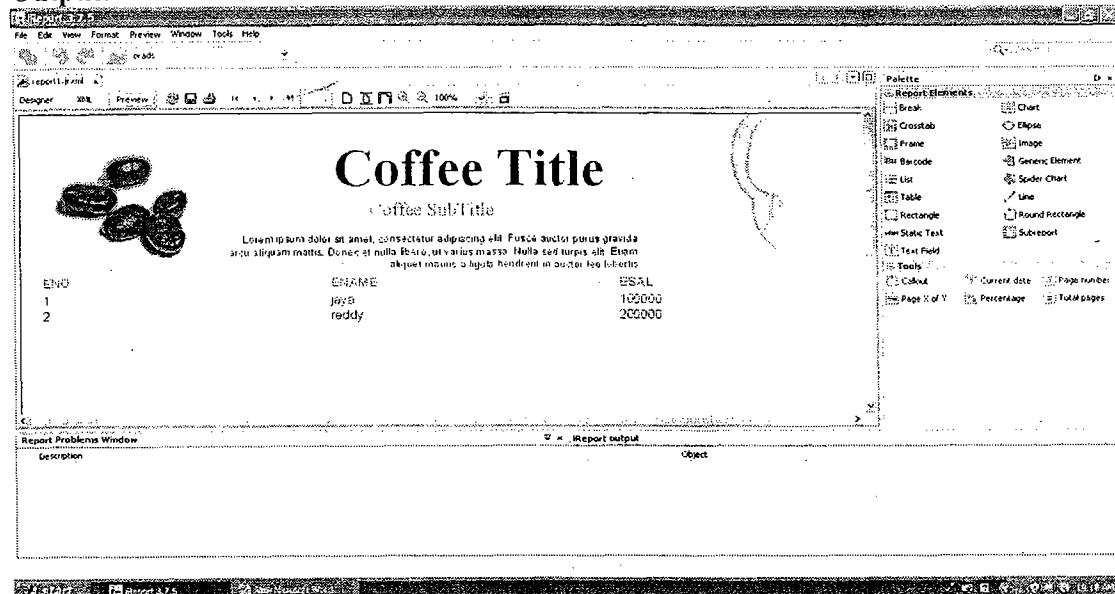


DURGA SOFTWARE SOLUTIONS

Tools Material

report → next → group1:some group(optional) → next → finish → we will see the congratulation message → preview.

Output:



(we can perform the modifications on the generated reports by using drag&drop operation, by using some iReports tools)
(we can generate the report in any style by using preview option)

Procedure To Design Graphical Report:

Step1: Start → Programs → Jasper Soft → iReports-3.7.5 → iReport3.7.5

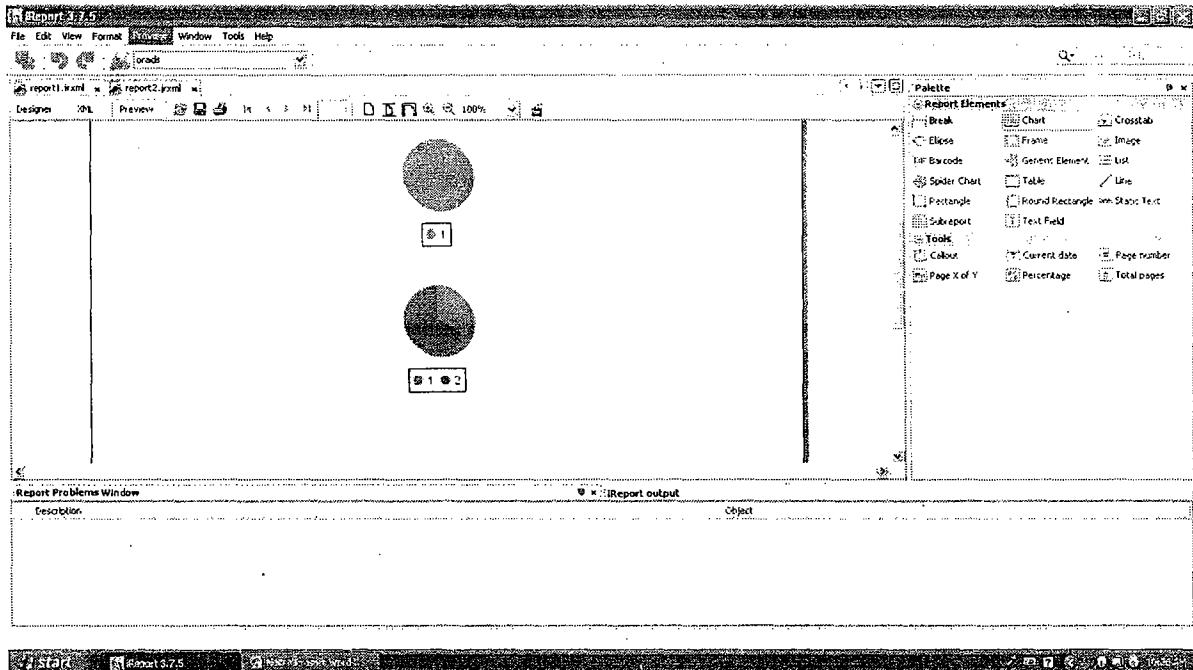
Step2:

File → new → report → BlankLetter → Launch Report Wizard → Report name:Report2(some name), Location: Some Location(hear the report will be .jrxml file will be generated) → file(it will be automatically generate) → next → Connection/data sources:orads, Query(SQL):select * from emp → next → select required fields → next → Group1(ESAL)(as your wish) → next (we will see the congratulation message) → finish

DURGA SOFTWARE SOLUTIONS
Tools Material

Step3:

select pallets → drag & drop chart to detail1 → select Pie(any one) → ok → dataset: Main report DataSet → next → select unique identifier: eno (primary key field) → Apply → select a numeric value (ESAL) → Apply → next → finish → preview output:



Standard procedure to work with jasper reports in real world:

Step1:

Use iReport tool to design the report and get the equivalent .jrxml file

Step2:

Design stand alone application/desktop application or web application using jasper reports API

- a) Use JasperCompileManager to compile .jrxml file
- b) Use FillManager to fill up the report with the data
- c) Use ExportManager (or) PrintManager (or) RunManager to generate (or) print the report

MAVEN

CONTENT:

1. Maven Overview
2. Maven
3. Maven Origins
4. What Maven Provide?
5. Mavens Principles
6. Declarative Execution
 - i) Maven's project object model (POM)
 - ii) Maven's build life cycle
7. Coherent Organization of Dependencies
 - i) Local Maven repository
 - ii) Central Maven repository
 - iii) Remote Maven Repository
 - iv) Locating dependency artifacts
8. Maven's Benefits
9. Maven Environment SetUp
 - i) System Requirement
 - ii) java installation verification
 - ii) set JAVA environment
 - iii) Download Maven Archive
 - iv) Extract Maven Archive and Configure Maven Environment Variables
 - vi) Add Maven bin directory location to system path and verify Maven installaton
10. Create Java Project
11. Build and Test Java Project
12. External Dependencies
13. Maven 2 Eclipse Plugin
14. Create web application using maven

DURGA SOFTWARE SOLUTIONS
Tools Material

15. Generate Documentation for Maven Project
16. Project Creation from Maven Templates
17. Team Collaboration with Maven
18. Migrating to Maven

MAVEN (Build Tool)

Maven Overview:

Maven provides a comprehensive approach to managing software projects. From compilation, to distribution, to documentation, to team collaboration, Maven provides the necessary abstractions that encourage reuse and take much of the work out of project builds.

Maven:

Maven is a project management framework. Maven is a build tool or a scripting framework.

Maven encompasses a set of build standards, an artifact repository model, and a software engine that manages and describes projects. It defines a standard life cycle for

building, testing, and deploying project artifacts. It provides a framework that enables easy reuse of

common build logic for all projects following Maven's standards. The Maven project at the Apache

Software Foundation is an open source community which produces software tools that understand a

common declarative Project Object Model (POM).

Maven 2, a framework that greatly simplifies the process of managing a software project.

Note: Maven is a declarative project management tool that decreases your overall time

DURGA SOFTWARE SOLUTIONS

Tools Material

to market by effectively leveraging cross-project intelligence. It simultaneously reduces your duplication effort and leads to higher code quality.

Maven Origin's:

Maven was borne of the practical desire to make several projects at the Apache Software Foundation (ASF) work in the same, predictable way. Prior to Maven, every project at the ASF had a different approach to compilation, distribution, and Web site generation. The ASF was effectively a series of isolated islands of innovation. While there were some common themes across the separate builds, each community was creating its own build systems and there was no reuse of build logic across projects. The build process for Tomcat was different than the build process for Struts, and the Turbine developers had a different site generation process than the Jakarta Commons developers.

This lack of a common approach to building software meant that every new project tended to copy and paste another project's build system. Ultimately, this copy and paste approach to build reuse reached a critical tipping point at which the amount of work required to maintain the collection of build systems was distracting from the central task of developing high-quality software. In addition, for a project with a difficult build system, the barrier to entry was extremely high; projects such as Jakarta Taglibs had (and continue to have) a tough time attracting developer interest because it could take an hour to configure everything in just the right way. Instead of focusing on creating good component libraries or MVC frameworks, developers were building yet another build system.

Maven entered the scene by way of the Turbine project, and it immediately sparked interest as a sort of Rosetta Stone for software project management. Developers within the Turbine project could freely move between subcomponents, knowing clearly how they all worked just by understanding how one of the components worked. Once developers spent time learning how one project was built, they did not have to go through the process again when they moved on to the next project. Developers at the ASF stopped figuring out creative ways to compile, test, and package software, and instead, started focusing on component development.

The same standards extended to testing, generating documentation, generating metrics and reports, and deploying. If you followed the Maven Build Life Cycle, your project gained a build by default. Soon after the creation of Maven other projects, such as Jakarta Commons, the Codehaus community started to adopt Maven 1 as a foundation for project management.

Many people come to Maven familiar with Ant, so it's a natural association, but Maven is an entirely different creature from Ant. Maven is not just a build tool, and not necessarily a replacement for Ant.

Whereas Ant provides a toolbox for scripting builds, Maven provides standards and a set of patterns in order to facilitate project management through reusable, common build strategies.

Maven Provides:

Maven provides developers to manage the following: Builds, Documentation, Reporting, Dependencies, SCM's, Releases.

DURGA SOFTWARE SOLUTIONS Tools Material

Maven provides a useful abstraction for building software in the same way an automobile provides an abstraction for driving. When you purchase a new car, the car provides a known interface; if you've learned how to drive a Jeep, you can easily drive a Camry. Maven takes a similar approach to software projects: if you can build one Maven project you can build them all, and if you can apply a testing plugin to one project, you can apply it to all projects. You describe your project using Maven's model, and you gain access to expertise and best-practices of an entire industry. Given the highly inter-dependent nature of projects in open source, Maven's ability to standardize locations for source files, documentation, and output, to provide a common layout for project documentation, and to retrieve project dependencies from a shared storage area makes the building process much less time consuming, and much more transparent.

Maven provides you with:

- ❖ A comprehensive model for software projects
- ❖ Tools that interact with this declarative model

An individual Maven project's structure and contents are declared in a Project Object Model (POM),

which forms the basis of the entire Maven system.

Maven Principles:

According to Christopher Alexander "patterns help create a shared language for communicating

insight and experience about problems and their solutions".

The following Maven principles were inspired by Christopher Alexander's idea of creating a shared language:

- ❖ Convention over configuration
- ❖ Declarative execution
- ❖ Reuse of build logic
- ❖ Coherent organization of dependencies.

Maven provides a shared language for software development projects.

Maven provides a structured build life cycle so that problems can be approached in terms of this structure.

DURGA SOFTWARE SOLUTIONS Tools Material

Each of the principles above enables developers to describe their projects at a higher level of abstraction, allowing more effective communication and freeing team members to get on with the important work of creating value at the application level.

The multiple projects of company will use the multiple logical databases that are created in database software of integrated machine on one per project basis.

During development mode of the project, the project will be maintained in the CVS Repository or SVN Repository to make the resources of the project visible and accessible for all developers of the project.

Declarative Execution:

Everything in Maven is driven in a declarative fashion using Maven's Project Object Model (POM) and specifically, the plugin configurations contained in the POM. The execution of Maven's plugins is coordinated by Maven's build life cycle in a declarative fashion with instructions from Maven's POM.

Maven's POM:

Maven is project-centric by design, and the POM is Maven's description of a single project. Without the POM, Maven is useless - the POM is Maven's currency. It is the POM that drives execution in Maven and this approach can be described as model-driven or declarative execution.

The POM is an XML document and looks like the following (very) simplified example:

```
<project>
<modelVersion>4.0.0</modelVersion>
<groupId>com.mycompany.app</groupId>
<artifactId>my-app</artifactId>
<packaging>jar</packaging>
<version>1.0-SNAPSHOT</version>
<dependencies>
<dependency>
<groupId>junit</groupId>
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>
</dependencies>
</project>
```

POM will allow you to compile, test, and generate basic documentation.

The POM contains every important piece of information about your project.

Maven's Super POM carries with it all the default conventions that Maven

encourages, and is the

analog of the Java language's java.lang.Object class.

POM contains the following key elements:

- ❖ **project** - This is the top-level element in all Maven pom.xml files.

- ❖ **modelVersion** - This required element indicates the version of the object model that the

POM is using. The version of the model itself changes very infrequently, but it is mandatory

in order to ensure stability when Maven introduces new features or other model changes.

- ❖ **groupId** - This element indicates the unique identifier of the organization or group that

created the project. The groupId is one of the key identifiers of a project and is typically

based on the fully qualified domain name of your organization. For example org.apache.maven.plugins is the designated groupId for all Maven plugins.

- ❖ **artifactId** - This element indicates the unique base name of the primary artifact being

generated by this project. A typical artifact produced by Maven would have the form

<artifactId>-<version>.<extension> (for example, myapp-1.0.jar). Additional artifacts such as source bundles also use the artifactId as part of their file name.

DURGA SOFTWARE SOLUTIONS
Tools Material

❖ **packaging** - This element indicates the package type to be used by this artifact (JAR, WAR, EAR, etc.). This not only means that the artifact produced is a JAR, WAR, or EAR, but also indicates a specific life cycle to use as part of the build process. The life cycle is a topic dealt with later in this chapter. For now, just keep in mind that the selected packaging of a project plays a part in customizing the build life cycle. The default value for the packaging element is jar so you do not have to specify this in most cases.

❖ **version** - This element indicates the version of the artifact generated by the project. Maven goes a long way to help you with version management and you will often see the SNAPSHOT designator in a version, which indicates that a project is in a state of development.

❖ **name** - This element indicates the display name used for the project. This is often used in Maven's generated documentation, and during the build process for your project, or other projects that use it as a dependency.

❖ **url** - This element indicates where the project's site can be found.

❖ **description** - This element provides a basic description of your project.

All POMs inherit from a parent (despite explicitly defined or not). This base POM is known as the Super POM, and contains values inherited by default. Maven uses the effective pom (configuration from super pom plus project configuration) to execute relevant goal. It helps developer to specify minimum configuration details in his/her pom.xml.

DURGA SOFTWARE SOLUTIONS

Tools Material

Although configurations can be overridden easily

Maven Build LifeCycle:

Maven models projects as "**nouns**" which are described by a POM. The POM captures the identity of a project.

Maven the "**verbs**" are goals packaged in Maven plugins which are tied to a phases in a build lifecycle.

A Maven lifecycle consists of a sequence of named phases: prepare-resources, compile, package, and install among other.

There is phase that captures compilation and a phase that captures packaging.

There are pre- and post- phases which can be used to register goals which must run prior to compilation, or tasks which must be run after a particular phase. When you tell Maven to build a project, you are telling Maven to step through a defined sequence of phases and execute any goals which may have been registered with each phase.

A build lifecycle is an organized sequence of phases that exist to give order to a set of goals. Those goals are chosen and bound by the packaging type of the project being acted upon.

There are three standard lifecycles in maven:

- clean
- default
- Site

The clean phase will be executed first, and then the dependency: copy-dependencies goal will be executed, and finally package phase will be executed.

Clean Life Cycle:

When we execute mvn post-clean command, Maven invokes the clean lifecycle consisting of the following phases.

- pre-clean
- clean
- post-clean

Maven clean goal (clean:clean) is bound to the clean phase in the clean lifecycle. Its clean:clean goal deletes the output of a build by deleting the build directory.

DURGA SOFTWARE SOLUTIONS
Tools Material

Thus when mvn clean command executes, Maven deletes the build directory.

We can customize this behavior by mentioning goals in any of the above phases of clean life cycle.

Default (or Build) Life Cycle:

This is the primary life cycle of Maven and is used to build the application. It has following 23 phases.

There are few important concepts related to Maven Lifecycles which are wroth to mention:

When a phase is called via Maven command,

Ex: **mvn compile**, only phases upto and including that phase will execute. Different maven goals will be bound to different phases of Maven lifecycle depending upon the type of packaging (JAR / WAR / EAR).

Site Lifecycle:

Maven Site plugin is generally used to create fresh documentation to create reports, deploy site etc.

Phases:

- pre-site
- site
- post-site
- site-deploy

command: C:\MVN\project>mvn site

Maven Build Profile:

A Build profile is a set of configuration values which can be used to set or override default values of Maven build.

Using a build profile, you can customize build for different environments such as Production v/s Development environments.

Profiles are specified in pom.xml file using its activeProfiles / profiles elements and are triggered in variety of ways.

DURGA SOFTWARE SOLUTIONS
Tools Material

Profiles modify the POM at build time, and are used to give parameters different target environments (for example, the path of the database server in the development, testing, and production environments).

Types of Build Profile Build profiles are majorly of three types

Type	Where it is defined
Per Project	Defined in the project POM file, pom.xml
Global	Defined in Maven global settings xml file (%M2_HOME%/conf/settings.xml)
Per User	Defined in Maven settings xml file (%USER_HOME%/.m2/settings.xml)

Profile Activation:

A Maven Build Profile can be activated in various ways.

- Explicitly using command console input.
- Through maven settings.
- Based on environment variables (User/System variables).
- OS Settings (for example, Windows family).
- Present/missing files.

Maven Plugin's:

Maven is actually a plugin execution framework where every task is actually done by plugins. Maven Plugins are generally used to :

- create jar file
- create war file
- compile code files
- unit testing of code
- create project documentation
- create project reports

DURGA SOFTWARE SOLUTIONS
Tools Material

A plugin generally provides a set of goals and which can be executed using following syntax:

Syntax: mvn [plugin-name]:[goal-name]

For example, a Java project can be compiled with the maven-compiler-plugin's compile-goal by running following command.

Command: mvn compiler:compile

Coherent Organization of Dependencies:

We are now going to delve into how Maven resolves dependencies and discuss the intimately connected concepts of dependencies, artifacts, and repositories. If you recall, our example POM has a single dependency listed for Junit:

```
<project>
<modelVersion>4.0.0</modelVersion>
<groupId>com.mycompany.app</groupId>
<artifactId>my-app</artifactId>
<packaging>jar</packaging>
<version>1.0-SNAPSHOT</version>
<dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>
</dependencies>
</project>
```

This POM states that your project has a dependency on JUnit, which is straightforward, but you may be asking yourself "Where does that dependency come from?" and "Where is the JAR?" The answers to those questions are not readily apparent without some explanation of how Maven's dependencies, artifacts and repositories work. In "Maven-speak" an artifact is a specific piece of software. In Java, the most common artifact is a JAR file, but a Java artifact could also be a WAR, SAR, or EAR.

DURGA SOFTWARE SOLUTIONS Tools Material

file. A dependency is a reference to a specific artifact that resides in a repository. In order for Maven to attempt to satisfy a dependency, Maven needs to know what repository to search as well as the dependency's coordinates. A dependency is uniquely identified by the following identifiers: groupId, artifactId and version.

At a basic level, we can describe the process of dependency management as Maven reaching out into the world, grabbing a dependency, and providing this dependency to your software project. There is more going on behind the scenes, but the key concept is that Maven dependencies are declarative.

In the POM you are not specifically telling Maven where the dependencies are physically located, you are simply telling Maven what a specific project expects.

Maven takes the dependency coordinates you provide in the POM, and it supplies these coordinates to its own internal dependency mechanisms. With Maven, you stop focusing on a collection of JAR files; instead you deal with logical dependencies. Your project doesn't require junit-3.8.1.jar, instead it depends on version 3.8.1 of the junit artifact produced by the junit group.

Dependency Management is one of the most powerful features in Maven. When a dependency is declared within the context of your project, Maven tries to satisfy that dependency by looking in all of the remote repositories to which it has access, in order to find the artifacts that most closely match the dependency request. If a matching artifact is located, Maven transports it from that remote repository to your local repository for project use.

Repositories:

A maven repository is a place i.e. directory where all the project jars, library jar, plugins or any other project specific artifacts are stored and can be used by Maven easily.

Maven repositories are of three types:

- Local
- Central

DURGA SOFTWARE SOLUTIONS
Tools Material

- Remote

Local Repository:

- Maven local repository is a folder location on your machine. It gets created when you run any maven command for the first time.
- Maven local repository keeps your project's all dependencies (library jars, plugin jars etc).
- When you run a Maven build, then Maven automatically downloads all the dependency jars into the local repository.
- It helps to avoid references to dependencies stored on remote machine every time a project is build.

Maven local repository by default get created by Maven in %USER_HOME% directory. To override the default location, mention another path in **Maven settings.xml file available at %M2_HOME%\conf directory.**

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
  http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <localRepository>C:/maven_repo</localRepository>
</settings>
```

When you run Maven command, Maven will download dependencies to your custom path.

Central Repository:

Maven central repository is repository provided by Maven community. It contains a large number of commonly used libraries.

When Maven does not find any dependency in local repository, it starts searching in central repository using following **URL: <http://repo1.maven.org/maven2/>**

Key concepts of Central repository

- This repository is managed by Maven community.
- It is not required to be configured.
- It requires internet access to be searched.

DURGA SOFTWARE SOLUTIONS

Tools Material

To browse the content of central maven repository, maven community has provided a **URL:**<http://search.maven.org/#browse>. Using this library, a developer can search all the available libraries in central repository.

Remote Repository:

Sometime, Maven does not find a mentioned dependency in central repository as well then it stopped build process and output error message to console. To prevent such situation, Maven provides concept of **Remote Repository** which is developer's own custom repository containing required libraries or other project jars. For example, using below mentioned POM.xml, Maven will download dependency (not available in central repository) from Remote Repositories mentioned in the same pom.xml.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.companyname.projectgroup</groupId>
  <artifactId>project</artifactId>
  <version>1.0</version>
  <dependencies>
    <dependency>
      <groupId>com.companyname.common-lib</groupId>
      <artifactId>common-lib</artifactId>
      <version>1.0.0</version>
    </dependency>
    <dependencies>
      <repository>
        <id>companyname.lib1</id>
        <url>http://download.companyname.org/maven2/lib1</url>
      </repository>
      <repository>
        <id>companyname.lib2</id>
        <url>http://download.companyname.org/maven2/lib2</url>
      </repository>
    </repositories>
  </project>
```

Maven Dependency Search Sequence:

When we execute Maven build commands, Maven starts looking for dependency libraries in the following sequence:

- ❖ **Step 1** - Search dependency in local repository, if not found, move to step 2 else if found then do the further processing.

DURGA SOFTWARE SOLUTIONS
Tools Material

- ❖ **Step 2** - Search dependency in central repository, if not found and remote repository/repositories is/are mentioned then move to step 4 else if found, then it is downloaded to local repository for future reference.
- ❖ **Step 3** - If a remote repository has not been mentioned, Maven simply stops the processing and throws error (Unable to find dependency).
- ❖ **Step 4** - Search dependency in remote repository or repositories, if found then it is downloaded to local repository for future reference otherwise Maven as expected stop processing and throws error (Unable to find dependency).

Maven Benefits:

Organizations and projects that adopt Maven benefit from:

- ❖ **Coherence:** Maven allows organizations to standardize on a set of best practices. Because

Maven projects adhere to a standard model they are less opaque. The definition of this term

from the American Heritage dictionary captures the meaning perfectly: "Marked by an orderly, logical, and aesthetically consistent relation of parts."

- ❖ **Reusability:** Maven is built upon a foundation of reuse. When you adopt Maven you are effectively reusing the best practices of an entire industry.
- ❖ **Agility:** Maven lowers the barrier to reuse not only for build logic, but also for software components. Maven makes it easier to create a component and then integrate it into a multi-project build. Developers can jump between different projects without the steep learning curve that accompanies custom, home-grown build systems.
- ❖ **Maintainability:** Organizations that adopt Maven can stop "building the build", and focus on

DURGA SOFTWARE SOLUTIONS
Tools Material

building the application. Maven projects are more maintainable because they follow a common, publicly-defined model.

Maven Environment SetUp:

Maven is Java based tool, so the very first requirement is to have JDK installed in your machine.

System Requirement

JDK	1.5 or Above
Memory	no minimum requirement.
Disk Space	no minimum requirement.
Operating System	no minimum requirement.

verify Java installation in your machine:

Now open console and execute the following **java** command.

Operating System	Task	Command
Windows	Open Command Console	c:\> java -version
Linux	Open Command Terminal	\$ java -version

Let's verify the output for all the operating systems:

Operating System	OutPut
Windows	java version "1.6.0_21" Java(TM) SE Runtime Environment (build 1.6.0_21-b07) Java HotSpot(TM) Client VM (build 17.0-b17, mixed mode, sharing)
Linux	java version "1.6.0_21" Java(TM) SE Runtime Environment (build 1.6.0_21-b07) Java HotSpot(TM) Client VM (build 17.0-b17, mixed mode, sharing)

Create Java Project:

To create your first project, you will use Maven's **Archetype** mechanism. An archetype is defined as an original pattern or model from which all other things of

DURGA SOFTWARE SOLUTIONS
Tools Material

the same kind are made. In Maven, an archetype is a template of a project, which is combined with some user input to produce a fully functional Maven project.

Archetype:

Archetype is a Maven project templating toolkit. An archetype is defined as *an original pattern or model from which all other things of the same kind are made*. The name fits as we are trying to provide a system that provides a consistent means of generating Maven projects. Archetype will help authors create Maven project templates for users, and provides users with the means to generate parameterized versions of those project templates.

Archetype ArtifactIds	Description
maven-archetype-archetype	An archetype which contains a sample archetype.
maven-archetype-j2ee-simple	An archetype which contains a simplified sample J2EE application.
maven-archetype-mojo	An archetype which contains a sample a sample Maven plugin.
maven-archetype-plugin	An archetype which contains a sample Maven plugin.
maven-archetype-plugin-site	An archetype which contains a sample Maven plugin site.
maven-archetype-portlet	An archetype which contains a sample JSR-268 Portlet.
maven-archetype-quickstart	An archetype which contains a sample Maven project.
maven-archetype-simple	An archetype which contains a simple Maven project.
maven-archetype-site	An archetype which contains a sample Maven site which demonstrates some of the supported document types like APT, XDoc, and FML and demonstrates how to i18n your site.
maven-archetype-site-simple	An archetype which contains a sample Maven site.
maven-archetype-	An archetype which contains a sample Maven Webapp project.

DURGA SOFTWARE SOLUTIONS
Tools Material

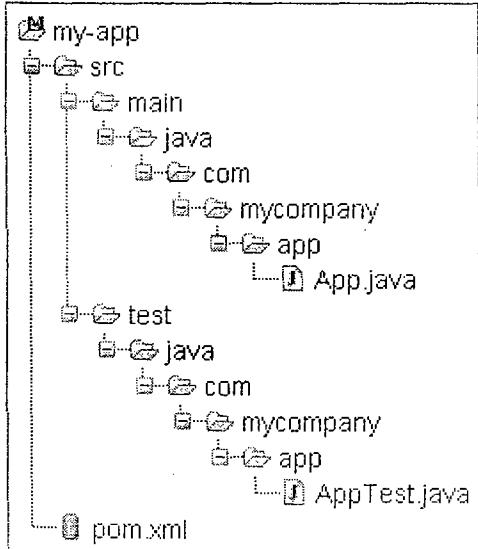
webapp

To create the Quick Start Maven project, execute the following:

C:\mvnbook> mvn archetype:create -DgroupId=com.mycompany.app \ - DartifactId=my-app

First, you will notice that a directory named my-app has been created for the new project, and this directory contains your pom.xml, which looks like the following:

```
<project>
<modelVersion>4.0.0</modelVersion>
<groupId>com.mycompany.app</groupId>
<artifactId>my-app</artifactId>
<packaging>jar</packaging>
<version>1.0-SNAPSHOT</version>
<name>Maven Quick Start Archetype</name>
<url>http://maven.apache.org</url>
<dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>
</dependencies>
</project>
```



Using Project Inheritance:

One of the most powerful features in Maven is *project inheritance*. Using project inheritance allows you to do things like state your organizational information, state

DURGA SOFTWARE SOLUTIONS

Tools Material

your deployment information, or state your common dependencies - all-in a single place.

Being the observant user, you have probably taken a peek at all the POMs in each of the projects that make up the Proficio project and noticed the following at the top of each of the POMs:

```
[...]
<parent>
<groupId>com.devzuz.mvnbook.proficio</groupId>
<artifactId>proficio</artifactId>
<version>1.0-SNAPSHOT</version>
</parent>
[...]
```

If you look at the top-level POM for Proficio, you will see that in the dependencies section there is a declaration for JUnit version 3.8.1. In this case the assumption being made is that JUnit will be used for testing in all our child projects. So, by stating the dependency in the top-level POM once, you never have to declare this dependency again, in any of your child POMs. The dependency is stated as following:

```
<project>
[...]
<dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>
</dependencies>
[...]
</project>
```

What specifically happens for each child POM, is that each one inherits the dependencies section of the top-level POM. So, if you take a look at the POM for the proficio-core module you will see the following (Note: there is no visible dependency declaration for JUnit):

```
<project>
<parent>
<groupId>com.devzuz.mvnbook.proficio</groupId>
<artifactId>proficio</artifactId>
<version>1.0-SNAPSHOT</version>
</parent>
<modelVersion>4.0.0</modelVersion>
<artifactId>proficio-core</artifactId>
<packaging>jar</packaging>
<name>Maven Proficio Core</name>
<dependencies>
<dependency>
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
<groupId>com.devzuz.mvnbook.proficio</groupId>
<artifactId>proficio-api</artifactId>
</dependency>
<dependency>
<groupId>org.codehaus.plexus</groupId>
<artifactId>plexus-container-default</artifactId>
</dependency>
</dependencies>
</project>
```

In order for you to see what happens during the inheritance process, you will need to use the handy **mvn help:effective-pom command**.

This command will show you the final result for a target POM. After you move into the proficio-core module directory and run the command, take a look at the resulting POM; you will see the JUnit version 3.8.1 dependency:

POM.xml

```
<project>
[...]
<dependencies>
[...]
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>
[...]
</dependencies>
[...]
</project>
```

Managing Dependencies:

When you are building applications you typically have a number of dependencies to manage and that number only increases over time, making dependency management difficult to say the least. Maven's strategy for dealing with this problem is to combine the power of project inheritance with specific dependency management elements in the POM.

When you write applications which consist of multiple, individual projects, it is likely that some of those projects will share common dependencies. When this happens it is critical that the same version of a given dependency is used for all your projects, so that the final application works correctly.

You don't want, for example, to end up with multiple versions of a dependency on the classpath when your application executes, as the results can be far from desirable. You want to make sure that all the versions, of all your dependencies, across all of your projects are in alignment so that your testing accurately reflects what you will deploy as your final result. In order to manage, or align, versions of

DURGA SOFTWARE SOLUTIONS

Tools Material

dependencies across several projects, you use the dependency management section in the top-level POM of an application.

To illustrate how this mechanism works, let's look at the dependency management section of the Proficio top-level POM:

```
<project>
[...]
<dependencyManagement>
<dependencies>
<dependency>
<groupId>com.devzuz.mvnbook.proficio</groupId>
<artifactId>proficio-model</artifactId>
<version>${project.version}</version>
</dependency>
<dependency>
<groupId>com.devzuz.mvnbook.proficio</groupId>
<artifactId>proficio-api</artifactId>
<version>${project.version}</version>
</dependency>
<dependency>
<groupId>com.devzuz.mvnbook.proficio</groupId>
<artifactId>proficio-store-memory</artifactId>
<version>${project.version}</version>
</dependency>
<dependency>
<groupId>com.devzuz.mvnbook.proficio</groupId>
<artifactId>proficio-store-xstream</artifactId>
<version>${project.version}</version>
</dependency>
<dependency>
<groupId>com.devzuz.mvnbook.proficio</groupId>
<artifactId>proficio-core</artifactId>
<version>${project.version}</version>
</dependency>
<dependency>
<groupId>org.codehaus.plexus</groupId>
<artifactId>plexus-container-default</artifactId>
<version>1.0-alpha-9</version>
</dependency>
</dependencies>
</dependencyManagement>
[...]
</project>
```

Using Snapshots:

While you are developing an application with multiple modules; it is usually the case that each of the modules are in flux. Your APIs might be undergoing some change or your implementations are undergoing change and are being fleshed out, or you may be doing some refactoring. Your build system needs to be able to deal easily with this real-time flux, and this is where Maven's concept of a *snapshot* comes into play. A snapshot in Maven is an artifact that has been prepared using the most recent

DURGA SOFTWARE SOLUTIONS
Tools Material

sources available. If you look at the top-level POM for Proficio you will see a snapshot version specified:

```
<project>
[...]
<version>1.0-SNAPSHOT</version>
<dependencyManagement>
<dependencies>
<dependency>
<groupId>com.devzuz.mvnbook.proficio</groupId>
<artifactId>proficio-model</artifactId>
<version>${project.version}</version>
</dependency>
<dependency>
<groupId>com.devzuz.mvnbook.proficio</groupId>
<artifactId>proficio-api</artifactId>
<version>${project.version}</version>
</dependency>
<dependency>
<groupId>org.codehaus.plexus</groupId>
<artifactId>plexus-container-default</artifactId>
<version>1.0-alpha-9</version>
</dependency>
</dependencies>
</dependencyManagement>
[...]
</project>
```

Specifying a snapshot version for a dependency means that Maven will look for new versions of that dependency without you having to manually specify a new version. Snapshot dependencies are assumed to be changing, so Maven will attempt to update them. By default Maven will look for snapshots on a daily basis, but you can use the -U command line option to force the search for updates.

Controlling Snapshots:

Snapshots were designed to be used in a team environment as a means for sharing development versions of artifacts that have already been built. Usually, in an environment where a number of modules are undergoing concurrent development, the build involves checking out all of the dependent projects and building them yourself. Additionally, in some cases, where projects are closely related, you must build all of the modules simultaneously from a master build. While building all of the modules from source can work well and is handled by Maven inherently, it can lead to a number of problems:

DURGA SOFTWARE SOLUTIONS
Tools Material

- It relies on manual updates from developers, which can be error-prone. This will result in local inconsistencies that can produce non-working builds
 - There is no common baseline against which to measure progress
 - Building can be slower as multiple dependencies must be rebuilt also
 - Changes developed against outdated code can make integration more difficult
- As you can see from these issues, building from source doesn't fit well with an environment that promotes continuous integration. Instead, use binary snapshots that have been already built and tested.

In Maven, this is achieved by regularly deploying snapshots to a shared repository, such as the internal repository set up in section 7.3. Considering that example, you'll see that the repository was defined in proficio/trunk/pom.xml:

```
[...]
<distributionManagement>
<repository>
<id>internal</id>
<url>file://localhost/C:/mvnbook/repository/internal</url>
</repository>
[...]
</distributionManagement>
```

Now, deploy proficio-api to the repository with the following command:

C:\mvnbook\proficio\trunk\proficio-api> mvn deploy

You'll see that it is treated differently than when it was installed in the local repository. The filename that is used is similar to proficio-api-1.0-20070726.120139-1.jar. In this case, the

version used is the time that it was deployed (in the UTC timezone) and the build number. If you were to deploy again, the time stamp would change and the build number would increment to 2.

This technique allows you to continue using the latest version by declaring a dependency on 1.0-

SNAPSHOT, or to lock down a stable version by declaring the dependency version to be the specific equivalent such as 1.0-20070726.12013-1. While this is not usually the case, locking the version in this way may be important if there are recent changes to the repository that need to be ignored temporarily.

The -U argument in the prior command is required to force Maven to update all of the snapshots in

the build. If it were omitted, by default, no update would be performed. This is because the default

policy is to update snapshots daily – that is, to check for an update the first time that particular

dependency is used after midnight local time.

C:\mvnbook\proficio\trunk\proficio-core> mvn -U install

We can also change the interval by changing the repository configuration. To see this, add the following configuration to the repository configuration you defined above in proficio/trunk/pom.xml:

DURGA SOFTWARE SOLUTIONS
Tools Material

```
[...]
<repository>
[...]
<snapshots>
<updatePolicy>interval:60</updatePolicy>
</snapshots>
</repository>
[...]
```

In this example, any snapshot dependencies will be checked once an hour to determine if there are updates in the remote repository. The settings that can be used for the update policy are never, always, daily (the default), and interval:*minutes*.

Code Compile:

Compile the source code using the following command:

C:\mvnbook\my-app> mvn compile

Compiling Test Sources and Running Unit Tests:

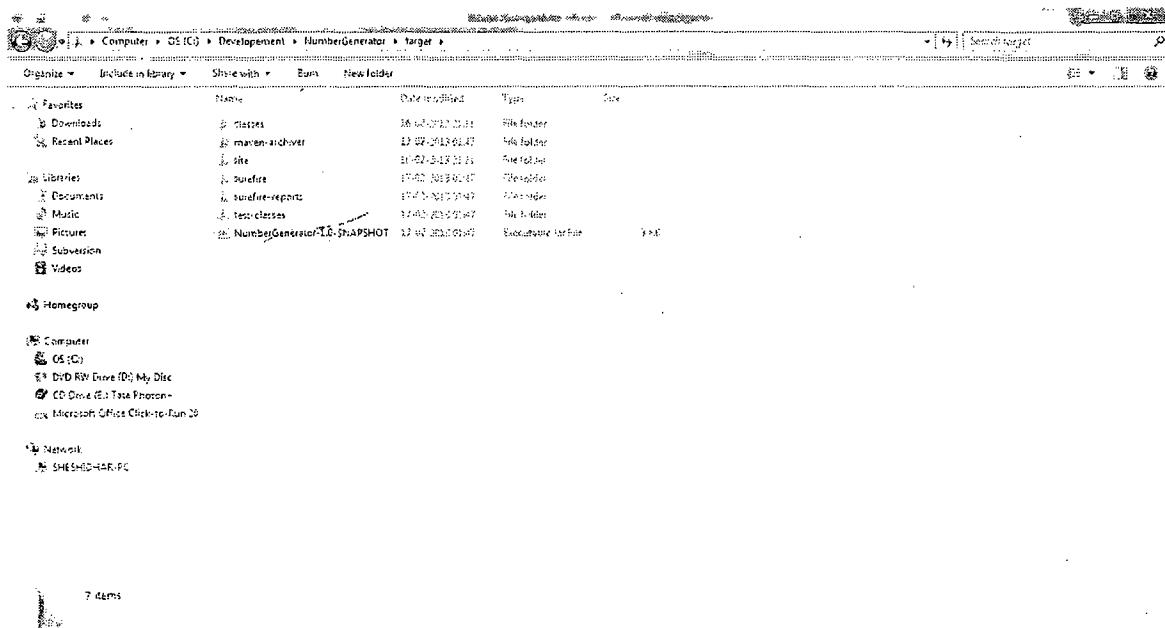
mvn test will always run the **compile** and **test-compile** phases first

Packaging and Installation to Your Local Repository:

Making a JAR file is straightforward and can be accomplished by executing the following command:

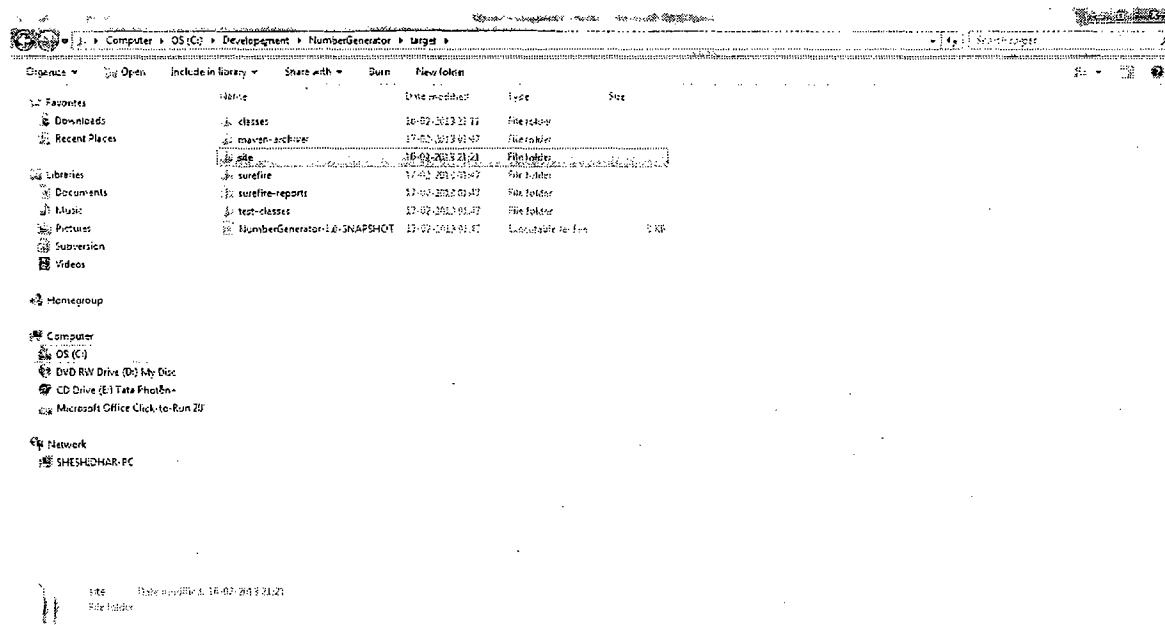
C:\mvnbook\my-app> mvn package

DURGA SOFTWARE SOLUTIONS Tools Material



create a basic Web site for your project, simply execute the following command:

C:\mvnbook\my-app> mvn site



DURGA SOFTWARE SOLUTIONS

Tools Material

Computer > OS (C) > Development > NumberGenerator > target > site				
Organize	Name	Size	Type	Size
Favorites	jboss	16.02.2011 12:23	File/folder	
Downloads	jsf	16.02.2011 22:13	File/folder	
Recent Places	jsf/images	16.02.2011 22:13	File/folder	
Sig Libraries:	jsf/dependencies	16.02.2011 22:13	Folder > HTML Doc.	1KB
Documents	jsf/dependency-info	16.02.2011 22:13	Folder > HTML Doc.	1KB
Music	jsf/distribution-management	16.02.2011 22:13	Folder > HTML Doc.	1KB
Pictures	jsf/index	16.02.2011 22:13	Folder > HTML Doc.	1KB
Subversion	jsf/integration	16.02.2011 22:13	Folder > HTML Doc.	0KB
Videos	jsf/issue-tracking	16.02.2011 22:13	Folder > HTML Doc.	0KB
Homegroup	jsf/licence	16.02.2011 22:13	Folder > HTML Doc.	0KB
Computers	jsf/mail-lists	16.02.2011 22:13	Folder > HTML Doc.	0KB
OS (C)	jsf/plugin-management	16.02.2011 22:13	Folder > HTML Doc.	0KB
DBD RH Drive (D): My Disc	jsf/plugins	16.02.2011 22:13	Folder > HTML Doc.	0KB
CD Drive (E:) Tata Photon-	jsf/project-info	16.02.2011 22:13	Folder > HTML Doc.	0KB
Microsoft Office Click-to-run 20	jsf/project-summary	16.02.2011 22:13	Folder > HTML Doc.	0KB
	jsf/source-repositories	16.02.2011 22:13	Folder > HTML Doc.	0KB
	jsf/team-list	16.02.2011 22:13	Folder > HTML Doc.	0KB

16 items

http://cds.sarvagya.org:8080/maven-snapshots/NumberGenerator/1.0-SNAPSHOT/NumberGenerator-1.0-SNAPSHOT-site/target/site/project-summary.html

Most Visited Getting Started Latest Headlines http://utils.babylon.co... Bookmarks

Search

Project Information

Name	NumberGenerator
Description	A simple maven project to generate random numbers.
Homepage	http://maven.apache.org

Project Organization

This project does not belong to an organization.

Build Information

GroupID	com.mkyong
ArtifactID	NumberGenerator
Version	1.0-SNAPSHOT
Type	JAR
JDK Rev	

Copyright © 2013. All Rights Reserved

DURGA SOFTWARE SOLUTIONS Tools Material

The screenshot shows a web browser window with the URL <http://dile.babylon.co.in:8080/maven-sites/NumbGenerator/target/site/plugins.html>. The page title is "NumberGenerator". On the left, there's a sidebar with "Project Documentation" and a "Maven" logo. The main content area has a section titled "Project Build Plugins" which lists various Maven plugins with their versions:

Plugin	Version
org.apache.maven.plugins:maven-clean-plugin	2.4.1
org.apache.maven.plugins:maven-compiler-plugin	3.9.2
org.apache.maven.plugins:maven-deploy-plugin	2.7
org.apache.maven.plugins:maven-install-plugin	2.3.1
org.apache.maven.plugins:maven-jar-plugin	2.3.2
org.apache.maven.plugins:maven-resources-plugin	2.5
org.apache.maven.plugins:maven-site-plugin	3.0
org.apache.maven.plugins:maven-eclipse-plugin	2.10

Below this, there's a section titled "Project Report Plugins" with the note: "There are no plugins reports defined in the Reporting part of this project." At the bottom right, it says "Copyright © 2013. All Rights Reserved."

Deploying your Application:

Currently Maven supports several methods of deployment, including simple file-based deployment, SSH2 deployment, SFTP deployment, FTP deployment, and external SSH deployment. In order to deploy, you need to correctly configure your **distributionManagement** element in your POM, which would typically be your top-level POM, so that all child POMs can inherit this information. Here are some examples of how to configure your POM via the various deployment mechanisms.

➤ Deploying to the File System:

To deploy to the file system you would use something like the following:

```
<project>
[...]
<distributionManagement>
<repository>
<id>proficio-repository</id>
<name>Proficio Repository</name>
<url>file://${basedir}/target/deploy</url>
</repository>
</distributionManagement>
[...]
</project>
```

DURGA SOFTWARE SOLUTIONS
Tools Material

➤ Deploying with SSH2:

To deploy to an SSH2 server you would use something like the following:

```
<project>
[...]
<distributionManagement>
<repository>
<id>proficio-repository</id>
<name>Proficio Repository</name>
<url>scp://sshserver.yourcompany.com/deploy</url>
</repository>
</distributionManagement>
[...]
</project>
```

➤ Deploying with SFTP:

To deploy to an SFTP server you would use something like the following:

```
<project>
[...]
<distributionManagement>
<repository>
<id>proficio-repository</id>
<name>Proficio Repository</name>
<url>sftp://ftpserver.yourcompany.com/deploy</url>
</repository>
</distributionManagement>
[...]
</project>
```

Deployment Automation:

In normally a deployment process consists of following steps

- Check-in the code from all project in progress into the SVN or source code repository and tag it.
- Download the complete source code from SVN.
- Build the application.
- Store the build output either WAR or EAR file to a common network location.
- Get the file from network and deploy the file to the production site.
- Updated the documentation with date and updated version number of the application.

DURGA SOFTWARE SOLUTIONS
Tools Material

Problem Statement :

There are normally multiple people involved in above mentioned deployment process. One team may handle check-in of code, other may handle build and so on. It is very likely that any step may get missed out due to manual efforts involved and owing to multi-team environment. For example, older build may not be replaced on network machine and deployment team deployed the older build again.

Solution :

Automate the deployment process by combining

- Maven, to build and release projects,
- SubVersion, source code repository, to manage source code,
- and Remote Repository Manager (Jfrog/Nexus) to manage project binaries.

We'll be using Maven Release plug-in to create an automated release process.

For Example: durga-soft-api project POM.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>bus-core-api</groupId>
  <artifactId>durga-soft-api</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <scm>
    <url>http://www.svn.com</url>
    <connection>scm:svn:http://localhost:8080/svn/jrepo/trunk/
      Framework</connection>
    <developerConnection>scm:svn:${username}/${password}@localhost:8080:
      common_core_api:1101:code</developerConnection>
  </scm>
  <distributionManagement>
    <repository>
      <id>Core-API-Java-Release</id>
      <name>Release repository</name>
      <url>http://localhost:8081/nexus/content/repositories/
        Core-Api-Release</url>
    </repository>
  </distributionManagement>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-release-plugin</artifactId>
        <version>2.0-beta-9</version>
      
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
<configuration>
<useReleaseProfile>false</useReleaseProfile>
<goals>deploy</goals>
<scmCommentPrefix>[durga-soft-api-release-checkin]-<
/scmCommentPrefix> .
</configuration>
</plugin>
</plugins>
</build>
</project>
```

In Pom.xml, following are the important elements used:

Element	Description
SCM	Configures the SVN location from where Maven will check out the source code.
Repositories	Location where built WAR/EAR/JAR or any other artifact will be stored after code build is successful.
Plugin	maven-release-plugin is configured to automate the deployment process.

Maven Release Plug-in:

The Maven does following useful tasks using *maven-release-plugin*.

mvn release:clean

It cleans the workspace in case the last release process was not successful.

mvn release:rollback

Rollback the changes done to workspace code and configuration in case the last release process was not successful.

mvn release:prepare

Performs multiple number of operations:

- Checks whether there are any uncommitted local changes or not
- Ensures that there are no SNAPSHOT dependencies
- Changes the version of the application and removes SNAPSHOT from the version to make release
- Update pom files to SVN.

DURGA SOFTWARE SOLUTIONS
Tools Material

- > Run test cases
- > Commit the modified POM files
- > Tag the code in subversion
- > Increment the version number and append SNAPSHOT for future release
- > Commit the modified POM files to SVN.

`mvn release:perform`

Checks out the code using the previously defined tag and run the Maven deploy goal to deploy the war or built artifact to repository

Maven Web Application:

create a simple java web application using **maven-archetype-webapp** plugin.

```
C:\anyfolder>mvn archetype:generate -DgroupId=com.durga.scjp -  
-DartifactId=DurgaWeb  
-DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false
```

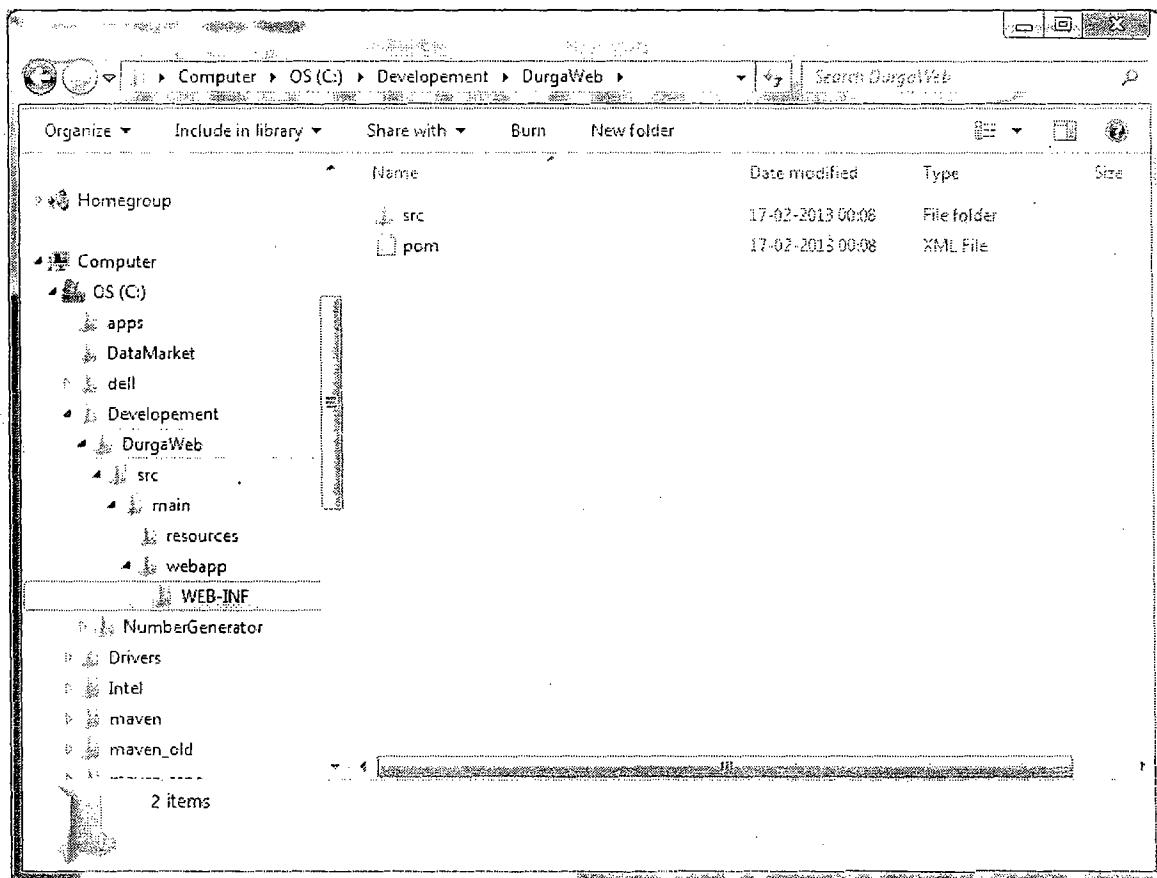
Maven will start processing and will create the complete web based java application project structure.

O/P can be seen at console.

```
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'archetype'.
[INFO] -----
[INFO] Building Maven Default Project
[INFO] task-segment: [archetype:generate] (aggregator-style)
[INFO] -----
[INFO] Preparing archetype:generate
[INFO] No goals needed for project - skipping
[INFO] [archetype:generate {execution: default-cli}]
[INFO] Generating project in Batch mode
[INFO] -----
[INFO] Using following parameters for creating project
from Old (1.x) Archetype: maven-archetype-webapp:1.0
[INFO] -----
[INFO] Parameter: groupId, Value: com.companyname.automobile
[INFO] Parameter: packageName, Value: com.companyname.automobile
[INFO] Parameter: package, Value: com.companyname.automobile
[INFO] Parameter: artifactId, Value: trucks
[INFO] Parameter: basedir, Value: C:\MVN
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\MVN\trucks
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 16 seconds
[INFO] Finished at: Tue Jul 17 11:00:00 IST 2012
[INFO] Final Memory: 20M/89M
[INFO] -----
```



Maven uses a standard directory layout. Using above example, we can understand following key concepts

Folder Structure	Description
DurgaWeb	contains src folder and pom.xml

DURGA SOFTWARE SOLUTIONS
Tools Material

src/main/webapp	contains index.jsp and WEB-INF folder.
src/main/webapp/WEB-INF	contains web.xml
src/main/resources	it contains images/properties files .

POM.xml:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.companyname.automobile</groupId>
  <artifactId>DurgaWeb</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>trucks Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build>
    <finalName>DurgaWeb</finalName>
  </build>
</project>
```

Maven also created a sample JSP Source file

C:\ >Development> DuregaWeb> src > main > webapp > you will see index.jsp.

```
<html>
<body>
<h2>Hello World!</h2>
</body>
</html>
```

Build Web Application:

DURGA SOFTWARE SOLUTIONS
Tools Material

Let's open command console, go the C:\Development\DurgaWeb\ directory and execute the following **mvn** command.

C:\Development\Development>mvn clean package

Maven will start building the project.

```
[INFO] Scanning for projects...
[INFO] -----
[INFO] Building trucks Maven Webapp
[INFO] task-segment: [clean, package]
[INFO] -----
[INFO] [clean:clean {execution: default-clean}]
[INFO] [resources:resources {execution: default-resources}]
[WARNING] Using platform encoding (Cp1252 actually) to
copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO] [compiler:compile {execution: default-compile}]
[INFO] No sources to compile
[INFO] [resources:testResources {execution: default-testResources}]
[WARNING] Using platform encoding (Cp1252 actually) to
copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory
C:\Development\Development\src\test\resources
[INFO] [compiler:testCompile {execution: default-testCompile}]
[INFO] No sources to compile
[INFO] [surefire:test {execution: default-test}]
[INFO] No tests to run.
[INFO] [war:war {execution: default-war}]
[INFO] Packaging webapp
[INFO] Assembling webapp[trucks] in [C:\Development\Development\target\trucks]
[INFO] Processing war project
[INFO] Copying webapp resources[C:\Development\Development\src\main\webapp]
[INFO] Webapp assembled in[77 msecs]
[INFO] Building war: C:\Development\Development\target\trucks.war
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 3 seconds
[INFO] Finished at: Tue Jul 17 11:22:45 IST 2012
[INFO] Final Memory: 11M/85M
[INFO] -----
```

Deploy Web Application:

Now copy the DurgaWeb.war created in C:\ > Development > DurgaWeb > target > folder to your webserver webapp directory and restart the webserver.

DURGA SOFTWARE SOLUTIONS
Tools Material

Test Web Application:

Run the web-application using URL : `http://<server-name>:<port-number>/trucks/index.jsp`
Verify the output.

Maven Eclipse Integration:

Eclipse provides an excellent plugin m2eclipse which seamlessly integrates Maven and Eclipse together.

Some of features of m2eclipse are listed below:

- You can run Maven goals from Eclipse.
- You can view the output of Maven commands inside the Eclipse using its own console.
- You can update maven dependencies with IDE.
- You can Launch Maven builds from within Eclipse.
- It does the dependency management for Eclipse build path based on Maven's pom.xml.
- It resolves Maven dependencies from the Eclipse workspace without installing to local Maven repository (requires dependency project be in same workspace).
- It automatic downloads required dependencies and sources from the remote Maven repositories.

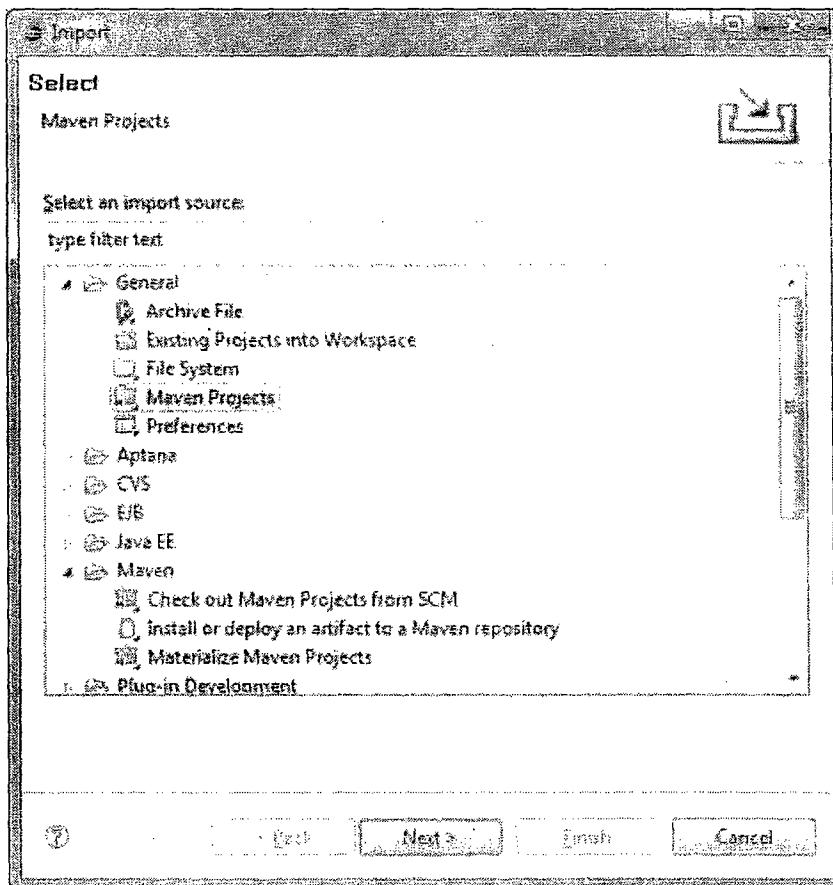
Use any of the following links to install m2eclipse:

Eclipse	URL
Eclipse 3.5 (Gallileo)	http://www.sonatype.com/books/m2eclipse-book/reference/ch02s03.html
Eclipse 3.6 (Helios)	http://www.sonatype.com/books/m2eclipse-book/reference/install-sect-marketplace.html

DURGA SOFTWARE SOLUTIONS
Tools Material

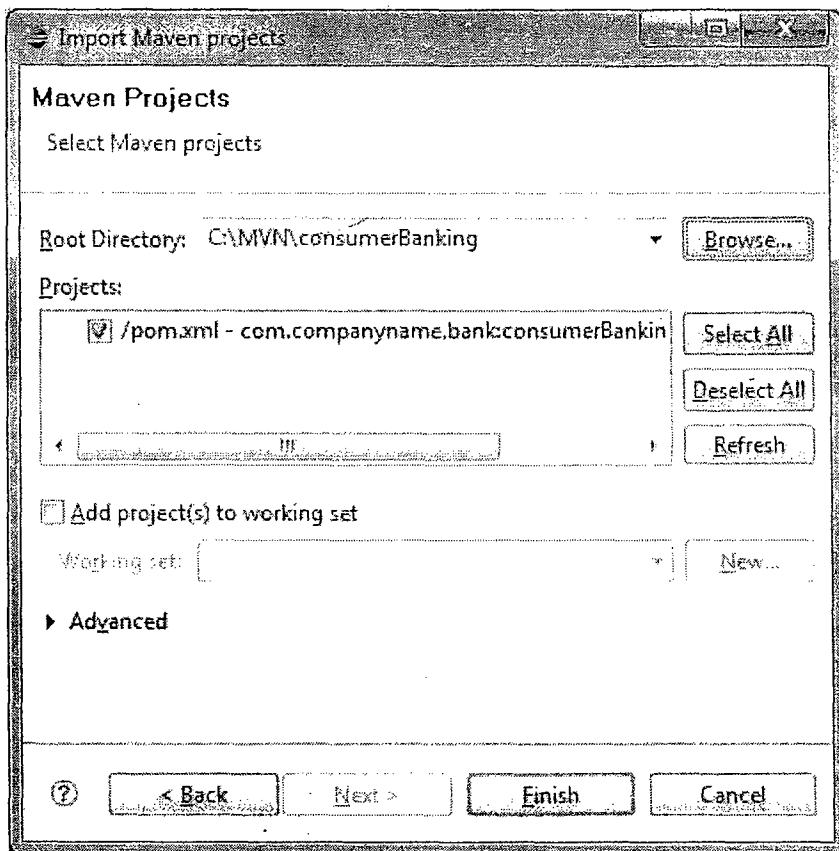
Import a maven project in Eclipse:

- Open Eclipse.
- Select File > Import > option.
- Select Maven Projects Option. Click on Next Button.



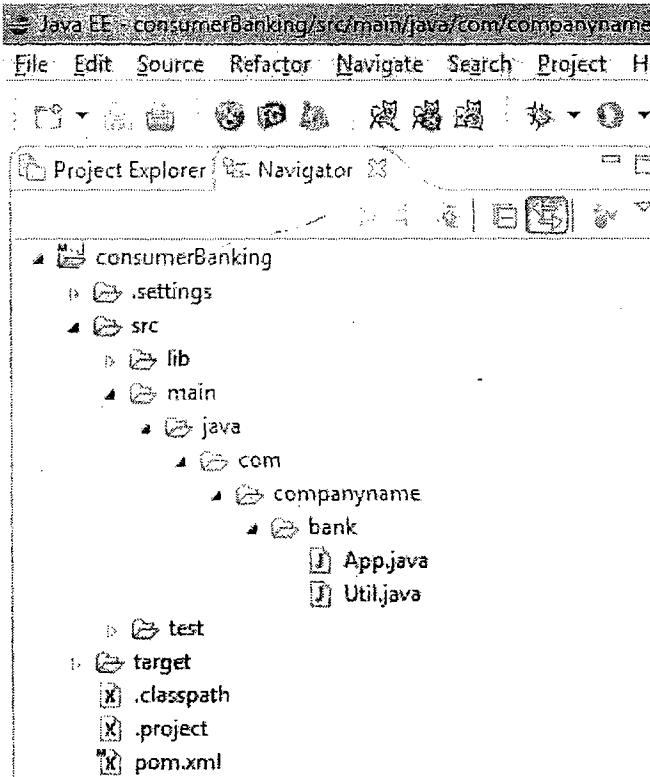
- Select Project location, where a project was created using Maven. We've create a Java Project consumerBanking. See Maven Creating Project to see how to create a project using Maven.
- Click Finish Button.

DURGA SOFTWARE SOLUTIONS
Tools Material



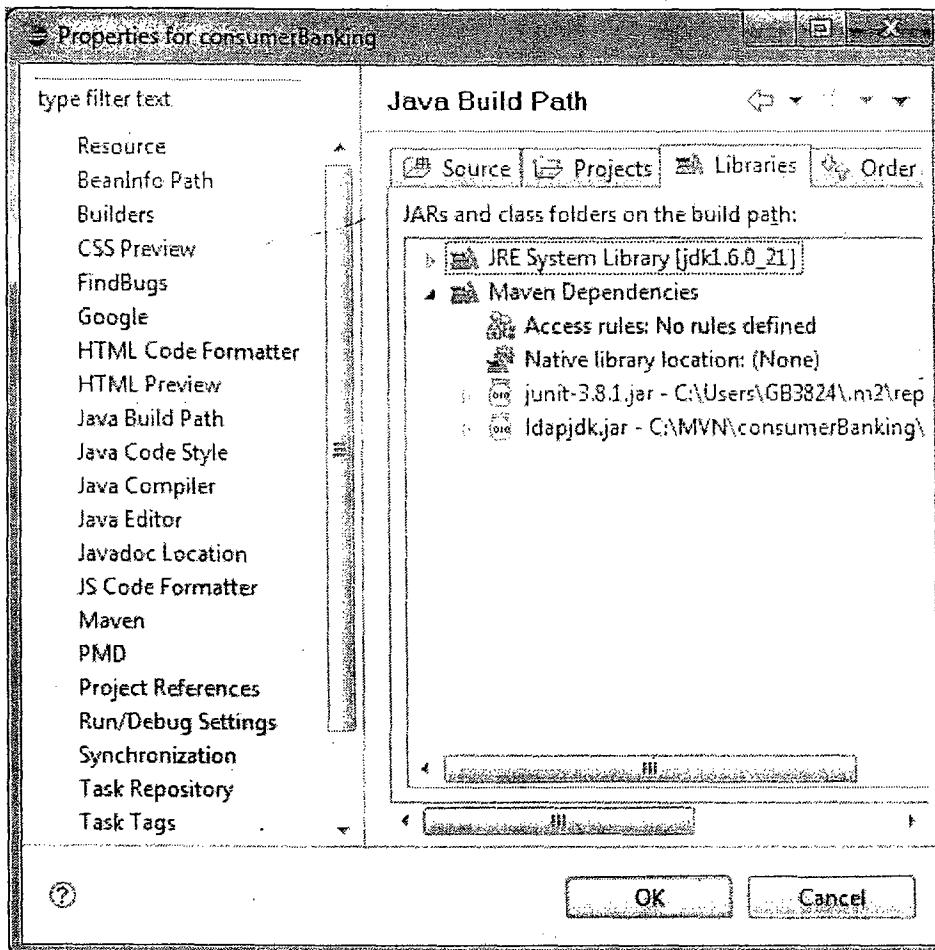
Now, you can see the maven project in eclipse.

DURGA SOFTWARE SOLUTIONS
Tools Material



Now, have a look at consumerBanking project properties. You can see that Eclipse has added Maven dependencies to java build path.

DURGA SOFTWARE SOLUTIONS
Tools Material



Now, Its time to build this project using maven capability of eclipse.

- Right Click on consumerBanking project to open context menu.
- Select Run as option
- Then maven package option
- Maven will start building the project. You can see the output in Eclipse Console.

```
[INFO] Scanning for projects...
[INFO] -----
[INFO] Building consumerBanking
[INFO]
[INFO] Id: com.companyname.bank:consumerBanking:jar:1.0-SNAPSHOT
[INFO] task-segment: [package]
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
[INFO] -----
---
[INFO] [resources:resources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:compile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:testCompile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [surefire:test]
[INFO] Surefire report directory:
C:\MVN\consumerBanking\target\surefire-reports
```

T E S T S

```
-----
Running com.companyname.bank.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.047
sec
```

Results :

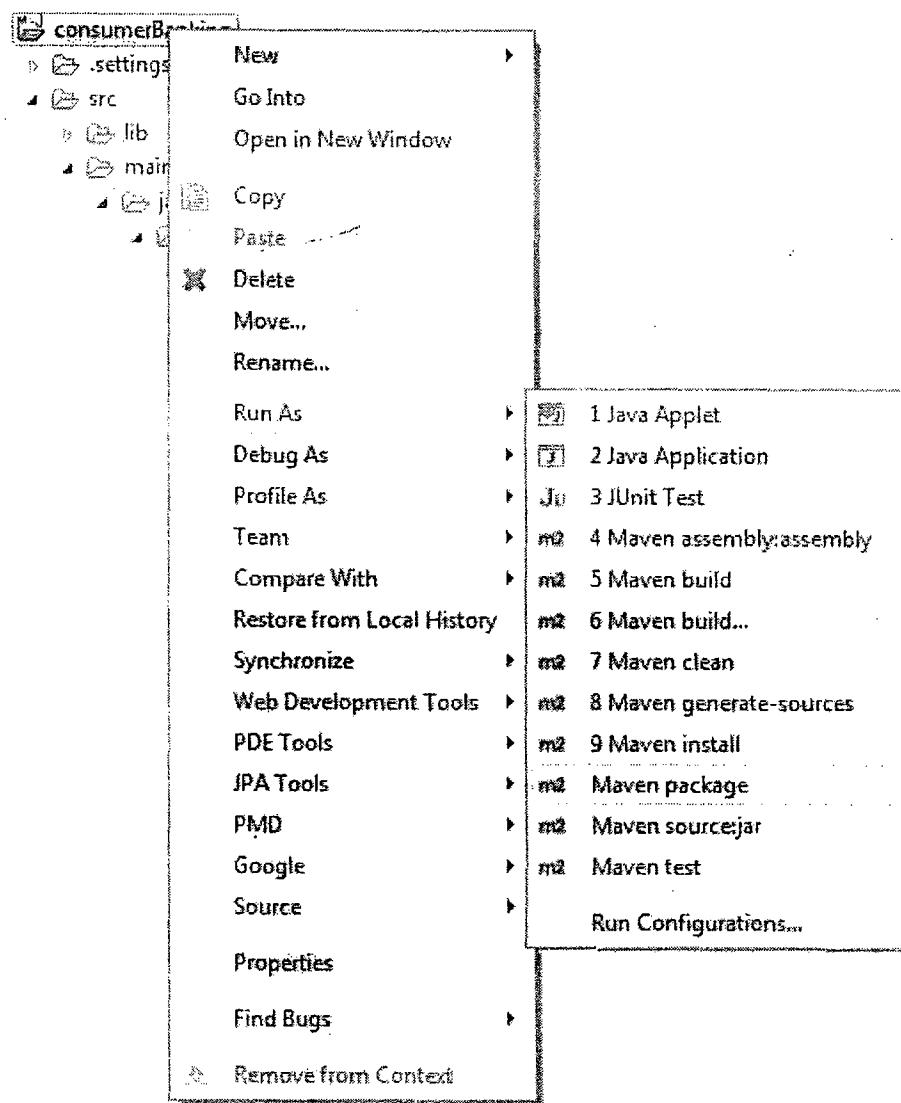
```
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO] [jar:jar]
[INFO] -----
```

```
---
[INFO] BUILD SUCCESSFUL
[INFO] -----
```

```
---
[INFO] Total time: 1 second
[INFO] Finished at: Thu Jul 12 18:18:24 IST 2012
[INFO] Final Memory: 2M/15M
[INFO] -----
```

```
---
```

DURGA SOFTWARE SOLUTIONS
Tools Material



Now, right click on App.java. Select Run As option. Select As Java Application. You will see the result

O/P: Hello World!

DURGA SOFTWARE SOLUTIONS
Tools Material

Maven Programs

DURGA SOFTWARE SOLUTIONS
Tools Material

- 1) Struts Application
- 2) Hibernate Application
- 3) Spring using JDBC
- 4) Sample Web Application.

Struts Application

Steps to Struts Application using Maven

Generate a simple webapp with archetype:generate:

```
C:\Development>mvn archetype:generate -  
DarchetypeArtifactId=maven-archetype-webapp -  
DgroupId=maven-tutorial -DartifactId=struts1app
```

```
[INFO] Scanning for projects...  
[INFO] Searching repository for plugin with prefix: 'archetype'.  
[INFO] -----  
[INFO] Building Maven Default Project  
[INFO]   task-segment: [archetype:generate] (aggregator-style)  
[INFO] -----  
[INFO] Preparing archetype:generate  
[INFO] No goals needed for project - skipping  
[INFO] Setting property: classpath.resource.loader.class =>  
'org.codehaus.plexus.velocity.ContextClassLoaderResourceLoader'.  
[INFO] Setting property: velocimacro.messages.on => 'false'.  
[INFO] Setting property: resource.loader => 'classpath'.  
[INFO] Setting property: resource.manager.logwhenfound => 'false'.  
[INFO] [archetype:generate {execution: default-cli}]  
[INFO] Generating project in Interactive mode  
Define value for version: 1.0-SNAPSHOT: :  
Confirm properties configuration:  
groupId: maven-tutorial  
artifactId: struts1app  
version: 1.0-SNAPSHOT  
package: maven-tutorial  
Y: :  
[INFO] -----  
[INFO] Using following parameters for creating OldArchetype: maven-  
archetype-webapp:1.0  
[INFO] -----  
[INFO] Parameter: groupId, Value: maven-tutorial  
[INFO] Parameter: packageName, Value: maven-tutorial
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
[INFO] Parameter: package, Value: maven-tutorial
[INFO] Parameter: artifactId, Value: struts1app
[INFO] Parameter: basedir, Value: C:\maven
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] ***** End of debug info from resources from
generated POM *****
[INFO] OldArchetype created in dir: C:\Development\struts1app
[INFO]
-----
[INFO] BUILD SUCCESSFUL
[INFO]
-----
[INFO] Total time: 20 seconds
[INFO] Finished at: Mon Sep 07 10:36:45 CDT 2009
[INFO] Final Memory: 8M/14M
[INFO]
```

C:\Development>

Maven generates a directory structure like this:

struts1app
struts1app/pom.xml
struts1app/src
struts1app/src/main
struts1app/src/main/resources
struts1app/src/main/webapp
struts1app/src/main/webapp/index.jsp
struts1app/src/main/webapp/WEB-INF
struts1app/src/main/webapp/WEB-INF/web.xml

Simple POM:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>maven-tutorial</groupId>
<artifactId>struts1app</artifactId>
<packaging>war</packaging>
<version>1.0-SNAPSHOT</version>
<name>struts1app Maven Webapp</name>
<url>http://maven.apache.org</url>
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
<build>
  <finalName>struts1app</finalName>
</build>
</project>
```

Create **settings.xml** in **C:\Users\{Username}\.m2**, add **Java.net** repository for Java EE dependencies:

```
<?xml version="1.0" encoding="UTF-8"?>
<settings>
  <profiles>
    <profile>
      <id>DefaultProfile</id>

      <activation>
        <activeByDefault>true</activeByDefault>
      </activation>

      <repositories>
        <repository>
          <id>maven2-repository.dev.java.net</id>
          <name>Java.net Repository for
Maven</name>
<url>http://download.java.net/maven/2/</url>
          <layout>default</layout>
        </repository>
      </repositories>
    </profile>
  </profiles>
```

DURGA SOFTWARE SOLUTIONS
Tools Material

</settings>

Add Java EE and Struts dependencies in pom.xml. Note that the Java EE dependency has scope provided, meaning that the web app container provides the jars, therefore we don't need to bundle them with our war file.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>maven-tutorial</groupId>
  <artifactId>struts1app</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>struts1app Maven Webapp</name>
  <url>http://maven.apache.org</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>

    <dependency>
      <groupId>javax</groupId>
      <artifactId>javaee-api</artifactId>
      <version>6.0-SNAPSHOT</version>
      <scope>provided</scope>
    </dependency>

    <dependency>
      <groupId>struts</groupId>
      <artifactId>struts</artifactId>
      <version>1.2.9</version>
    </dependency>
  </dependencies>

  <build>
    <finalName>struts1app</finalName>
  </build>
```

DURGA SOFTWARE SOLUTIONS
Tools Material

</project>

Create a directory named java under main, create the Struts form and action classes:

**src/main/java
src/main/java/com
src/main/java/com/dss
src/main/java/com/dss/HelloAction.java
src/main/java/com/dss/HelloForm.java**

HelloForm.java

```
package com.dss;  
  
import org.apache.struts.action.ActionForm;  
  
public class HelloForm extends ActionForm {  
    private String name;  
  
    public String getName() {  
        return this.name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```

HelloAction.java

```
package com.dss;  
  
import java.io.IOException;  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
  
import org.apache.struts.action.*;  
  
public class HelloAction extends Action {
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
public ActionForward execute(ActionMapping map,
    ActionForm form,
    HttpServletRequest req,
    HttpServletResponse resp)
throws IOException, ServletException {
HelloForm f = (HelloForm) form;
req.setAttribute("name", f.getName());
return map.findForward("hello");
}
```

Create a directory named jsp under webapp, and create 2 jsp files:

index.jsp:

```
<%@ page contentType="text/html;charset=UTF-8"
language="java" %>
<%@ taglib uri="http://struts.apache.org/tags-html"
prefix="html" %>

<html>
<body>
<html:form action="/hello" method="post">
    Enter Name: <html:text property="name"/>
    <br/>
    <input type="submit" name="submit" value="Go"/>
</html:form>
</body>
</html>
```

hello.jsp:

```
<%@ page contentType="text/html;charset=UTF-8"
language="java" %>
<%@ taglib uri="http://struts.apache.org/tags-bean"
prefix="bean" %>

<html>
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
<body>
<h2>Hello, <bean:write name="name"/>!</h2>
</body>
</html>
```

Change the contents of the existing index.jsp to:

```
<html>
<head>
<meta http-equiv="refresh" content="0;URL=index.do">
</head>
<body>
</body>
</html>
```

Create struts-config.xml under WEB-INF:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE struts-config PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration
1.3//EN"
"http://struts.apache.org/dtds/struts-config_1_3.dtd">

<struts-config>
  <global-forwards>
    <forward name="index" path="/jsp/index.jsp"/>
  </global-forwards>

  <form-beans>
    <form-bean name="helloForm"
type="com.dss.HelloForm" />
  </form-beans>

  <action-mappings>
    <action
      path="/index"
      name="helloForm"
      type="org.apache.struts.actions.ForwardAction"
      parameter="/jsp/index.jsp"
      validate="false">
    </action>
  </action-mappings>
</struts-config>
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
<action path="/hello"
       name="helloForm"
       scope="request"
       type="com.dss.HelloAction"
       validate="false">
    <forward name="hello" path="/jsp/hello.jsp" />
</action>
</action-mappings>
</struts-config>
```

Change the contents of WEB-INF/web.xml:

```
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
    <display-name>Archetype Created Web Application</display-
name>

    <servlet>
        <servlet-name>action</servlet-name>
        <servlet-class>org.apache.struts.action.ActionServlet</servlet-
class>
        <init-param>
            <param-name>config</param-name>
            <param-value>/WEB-INF/struts-config.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>action</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
</web-app>
```

The final structure of the project should look like:

```
struts1app
struts1app/pom.xml
struts1app/src
struts1app/src/main
struts1app/src/main/java
struts1app/src/main/java/com
struts1app/src/main/java/com/dss
struts1app/src/main/java/com/dss/HelloAction.java
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
struts1app/src/main/java/com/dss/HelloForm.java
struts1app/src/main/resources
struts1app/src/main/webapp
struts1app/src/main/webapp/index.jsp
struts1app/src/main/webapp/jsp
struts1app/src/main/webapp/jsp/hello.jsp
struts1app/src/main/webapp/jsp/index.jsp
struts1app/src/main/webapp/WEB-INF
struts1app/src/main/webapp/WEB-INF/struts-config.xml
struts1app/src/main/webapp/WEB-INF/web.xml
```

Finally, build with "mvn package".

**Maven will generates struts1app.war and place it in target folder.
Take the war file and deploy it in server.**

```
=====
=====
```

Hibernate with Maven Application

```
CREATE TABLE DBUSER (
    USER_ID      NUMBER (5)  NOT NULL,
    USERNAME     VARCHAR2 (20) NOT NULL,
    CREATED_BY   VARCHAR2 (20) NOT NULL,
    CREATED_DATE DATE        NOT NULL,
    PRIMARY KEY ( USER_ID )
);
```

Use Maven archetype:generate to create a standard project structure.

```
C:\Development>mvn archetype:generate -DgroupId=com.mkyong -
-DartifactId=HibernateExample -DarchetypeArtifactId=maven-archetype-
quickstart -DinteractiveMode=false
```

Maven to Eclipse IDE

To convert the generated Maven based project to Eclipse project, and import it

DURGA SOFTWARE SOLUTIONS
Tools Material

into your Eclipse IDE.

mvn eclipse:eclipse

Add Hibernate and Oracle Dependency

Update **pom.xml** file, and add all related dependencies.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.dss.common</groupId>
    <artifactId>HibernateExample</artifactId>
    <packaging>jar</packaging>
    <version>1.0</version>
    <name>HibernateExample</name>
    <url>http://maven.apache.org</url>

    <repositories>
        <repository>
            <id>JBoss repository</id>
            <url>http://repository.jboss.org/nexus/content/groups/public/</url>
        </repository>
    </repositories>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <!-- ORACLE database driver -->
        <dependency>
            <groupId>com.oracle</groupId>
            <artifactId>ojdbc14</artifactId>
            <version>10.2.0</version>
        </dependency>
    </dependencies>

```

DURGA SOFTWARE SOLUTIONS

Tools Material

```
</dependency>

<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>3.6.3.Final</version>
</dependency>

<dependency>
    <groupId>javassist</groupId>
    <artifactId>javassist</artifactId>
    <version>3.12.1.GA</version>
</dependency>

</dependencies>
</project>
```

Hibernate Mapping file (hbm) + Model

Create a Hibernate XML mapping file and Model class for table "DBUSER".

Create following "DBUser.hbm.xml" file and put it under "**src/main/resources/com/dss/user**".

Note: Create the folder if it does not exists.

File : DBUser.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate
Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.dss.user.DBUser" table="DBUSER">
        <id name="userId" type="int">
            <column name="USER_ID" precision="5" scale="0" />
            <generator class="assigned" />
        </id>
        <property name="username" type="string">
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
<column name="USERNAME" length="20" not-null="true" />
</property>
<property name="createdBy" type="string">
    <column name="CREATED_BY" length="20" not-null="true"
/>
</property>
<property name="createdDate" type="date">
    <column name="CREATED_DATE" length="7" not-null="true"
/>
</property>
</class>
</hibernate-mapping>
```

**Create a "DBUser.java" file and put it under
"src/main/java/com/dss/user/"**

File : DBUser.java

```
package com.dss.user;

import java.util.Date;

public class DBUser implements java.io.Serializable {

    private int userId;
    private String username;
    private String createdBy;
    private Date createdDate;

    public DBUser() {
    }

    public DBUser(int userId, String username, String createdBy,
                 Date createdDate) {
        this.userId = userId;
        this.username = username;
        this.createdBy = createdBy;
        this.createdDate = createdDate;
    }

    public int getUserId() {
        return this.userId;
    }

    public void setId(int userId) {
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
        this.userId = userId;
    }

    public String getUsername() {
        return this.username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getCreatedBy() {
        return this.createdBy;
    }

    public void setCreatedBy(String createdBy) {
        this.createdBy = createdBy;
    }

    public Date getCreatedDate() {
        return this.createdDate;
    }

    public void setCreatedDate(Date createdDate) {
        this.createdDate = createdDate;
    }

}
```

Hibernate Configuration File

Create a Hibernate configuration file “**hibernate.cfg.xml**” and put it under the root of resources folder, “**src/main/resources/hibernate.cfg.xml**”, and fill in your Oracle database details. And map to above Hibernate mapping file – “**DBUser.hbm.xml**”.

File : hibernate.cfg.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property
name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver
        </property>
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
<property
name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:XE</pr
operty>
<property name="hibernate.connection.username">system</property>
<property name="hibernate.connection.password">admin</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</proper
ty>
<property name="show_sql">true</property>
<mapping resource="com/dss/user/DBUser.hbm.xml"></mapping>
</session-factory>
</hibernate-configuration>
```

7. Hibernate Utility

Create a classic “HibernateUtil.java” class to take care of Hibernate session management. And put under “src/main/java/com/dss/util/HibernateUtil.java”

File : HibernateUtil.java

```
package com.dss.util;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {

    private static final SessionFactory sessionFactory =
buildSessionFactory();

    private static SessionFactory buildSessionFactory() {
        try {
            // Create the SessionFactory from
hibernate.cfg.xml
            return new
Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            // Make sure you log the exception, as it might
be swallowed
            System.err.println("Initial SessionFactory
creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }

    public static void shutdown() {
        // Close caches and connection pools
        getSessionFactory().close();
    }
}
```

DURGA SOFTWARE SOLUTIONS
Tools Material

Hibernate Coding

Update "App.java", to code Hibernate to save a dummy user record into a table "**DBUSER**".

File : App.java

```
package com.dss;

import java.util.Date;
import org.hibernate.Session;
import com.dss.util.HibernateUtil;
import com.dss.user.DBUser;

public class App {
    public static void main(String[] args) {
        System.out.println("Maven + Hibernate + Oracle");
        Session session =
        HibernateUtil.getSessionFactory().openSession();

        session.beginTransaction();
        DBUser user = new DBUser();

        user.setUserId(100);
        user.setUsername("superman");
        user.setCreatedBy("system");
        user.setCreatedDate(new Date());

        session.save(user);
        session.getTransaction().commit();
    }
}
```

Spring Using JDBC using Maven Example:

1)Create a table CUSTOMER in oracle DB.

CREATE TABLE CUSTOMER

DURGA SOFTWARE SOLUTIONS
Tools Material

```
( CUST_ID NUMBER(5) NOT NULL ,  
    NAME VARCHAR2(20) NULL ,  
    AGE NUMBER(2) NULL ,  
    CONSTRAINT PK_PERSON PRIMARY KEY (CUST_ID)  
);
```

Use Maven archetype:generate to create a standard project structure.

```
C:\Development>mvn archetype:generate -DgroupId=com.dss -  
DartifactId=SpringJDBCExample -DarchetypeArtifactId=maven-archetype-  
quickstart -DinteractiveMode=false
```

Add Hibernate and Oracle Dependency

Update **pom.xml** file, and add all related dependencies.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">  
  
<modelVersion>4.0.0</modelVersion>  
  
<groupId>com.dss.common</groupId>  
  
<artifactId>SpringJDBCExample</artifactId>  
  
<packaging>jar</packaging>  
  
<version>1.0-SNAPSHOT</version>  
  
<name>SpringJDBCExample</name>  
  
<url>http://maven.apache.org</url>  
  
<dependencies>  
  
<dependency>  
  
<groupId>junit</groupId>  
  
<artifactId>junit</artifactId>
```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
<version>3.8.1</version>

<scope>test</scope>

</dependency>

<!-- Spring framework -->

<dependency>

    <groupId>org.springframework</groupId>

    <artifactId>spring</artifactId>

    <version>2.5.6</version>

</dependency>

<dependency>

    <groupId>com.oracle</groupId>

    <artifactId>ojdbc14</artifactId>

    <version>10.2.0</version>

</dependency>

</dependencies>

</project>
```

Customer model

Add a customer model to store customer's data.

```
package com.dss.customer.model;

import java.sql.Timestamp;

public class Customer
{
    int custId;
    String name;
    int age;
    //getter and setter methods
}
```

DURGA SOFTWARE SOLUTIONS
Tools Material

Data Access Object (DAO) pattern

Create CustomerDao.java under
\src\main\java\com\mkyong\customer\dao\

Customer Dao interface.

```
package com.mkyong.customer.dao;

import com.mkyong.customer.model.Customer;

public interface CustomerDAO
{
    public void insert(Customer customer);
    public Customer findByCustomerId(int custId);
}
```

Create JdbcCustomerDAO.java under
src\main\java\com\mkyong\customer\dao\impl\

Customer Dao implementation, JdbcCustomerDAO.java use JDBC to issue a simple insert and select statement.

```
package com.dss.customer.dao.impl;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import javax.sql.DataSource;
import com.dss.customer.dao.CustomerDAO;
import com.dss.customer.model.Customer;

public class JdbcCustomerDAO implements CustomerDAO
{
    private DataSource dataSource;

    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
    }

    public void insert(Customer customer) {

        String sql = "INSERT INTO CUSTOMER " +
                    "(CUST_ID, NAME, AGE) VALUES (?, ?, ?)";
        Connection conn = null;

        try {
            conn = dataSource.getConnection();
            PreparedStatement ps =
conn.prepareStatement(sql);
            ps.setInt(1, customer.getCustId());
            ps.setString(2, customer.getName());

```

DURGA SOFTWARE SOLUTIONS
Tools Material

```
        ps.setInt(3, customer.getAge());
        ps.executeUpdate();
        ps.close();

    } catch (SQLException e) {
        throw new RuntimeException(e);

    } finally {
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {}
        }
    }
}

public Customer findByCustomerId(int custId){

    String sql = "SELECT * FROM CUSTOMER WHERE CUST_ID = ?";
    Connection conn = null;

    try {
        conn = dataSource.getConnection();
        PreparedStatement ps =
    conn.prepareStatement(sql);
        ps.setInt(1, custId);
        Customer customer = null;
        ResultSet rs = ps.executeQuery();
        if (rs.next()) {
            customer = new Customer(
                rs.getInt("CUST_ID"),
                rs.getString("NAME"),
                rs.getInt("Age")
            );
        }
        rs.close();
        ps.close();
        return customer;
    } catch (SQLException e) {
        throw new RuntimeException(e);
    } finally {
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {}
        }
    }
}
}
```

DURGA SOFTWARE SOLUTIONS
Tools Material

Spring bean configuration

Create the Spring bean configuration file for customerDAO and datasource.
Create File *Spring-Customer.xml* under *src/main/resources/customer/*

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-
                           2.5.xsd">

    <bean id="customerDAO"
          class="com.dss.customer.dao.impl.JdbcCustomerDAO">
        <property name="dataSource" ref="dataSource" />
    </bean>

</beans>
```

Create File *Spring-Datasource.xml* under *src/main/resources/database*

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-
                           2.5.xsd">

    <bean id="dataSource"
          class="org.springframework.jdbc.datasource.DriverManagerDataSource">

        <property name="driverClassName"
                  value="oracle.jdbc.driver.OracleDriver" />
        <property name="url"
                  value="jdbc:oracle:thin:@localhost:1521:XE" />
        <property name="username" value="root" />
        <property name="password" value="password" />
    </bean>

</beans>
```

Create File *Spring-Module.xml* under *src/main/resources/*

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-
                           2.5.xsd">

    <import resource="database/Spring-Datasource.xml" />
    <import resource="customer/Spring-Customer.xml" />

</beans>
```

DURGA SOFTWARE SOLUTIONS
Tools Material

Run App.java

```
package com.dss.common;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.dss.customer.dao.CustomerDAO;
import com.dss.customer.model.Customer;

public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context =
            new ClassPathXmlApplicationContext("Spring-Module.xml");

        CustomerDAO customerDAO = (CustomerDAO)
context.getBean("customerDAO");
        Customer customer = new Customer(1, "durga", 28);
        customerDAO.insert(customer);
        Customer customer1 = customerDAO.findById(1);
        System.out.println(customer1);

    }
}
```

Steps to Generate Sample Web Application using Maven

Using webapp archetype generate the generic Web Application Structure.

```
C:\Development>mvn archetype:create -  
DarchetypeGroupId=org.apache.maven.archetypes  
-DarchetypeArtifactId=maven-archetype-webapp -DarchetypeVersion=1.0  
-DgroupId=com.dss -DartifactId=ServletMavenDSS -Dversion=1.0-SNAPSHOT
```

C:\Development>

Maven generates a directory structure like this:

```
ServletMavenDSS
ServletMavenDSS /pom.xml
ServletMavenDSS/src
ServletMavenDSS/src/main
ServletMavenDSS/src/main/resources
ServletMavenDSS/src/main/webapp
ServletMavenDSS/src/main/webapp/index.jsp
ServletMavenDSS/src/main/webapp/WEB-INF
ServletMavenDSS/src/main/webapp/WEB-INF/web.xml
```

DURGA SOFTWARE SOLUTIONS
Tools Material

The project's pom.xml file is shown below

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.dsstest</groupId>
  <artifactId>ServletMavenDSS</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>mywebtest Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build>
    <finalName>ServletMavenDSS</finalName>
  </build>
</project>
```

Type Command:

C:\Development> mvn install

Maven will compile all the source files place all the class files in target folder, Unittesting will be done, and generate war file place in target folder.

Take a war file and deploy it in server. It will display.
O/P: Hello World!

Thank you.....

DURGA SOFTWARE SOLUTIONS
Tools Material

SVN

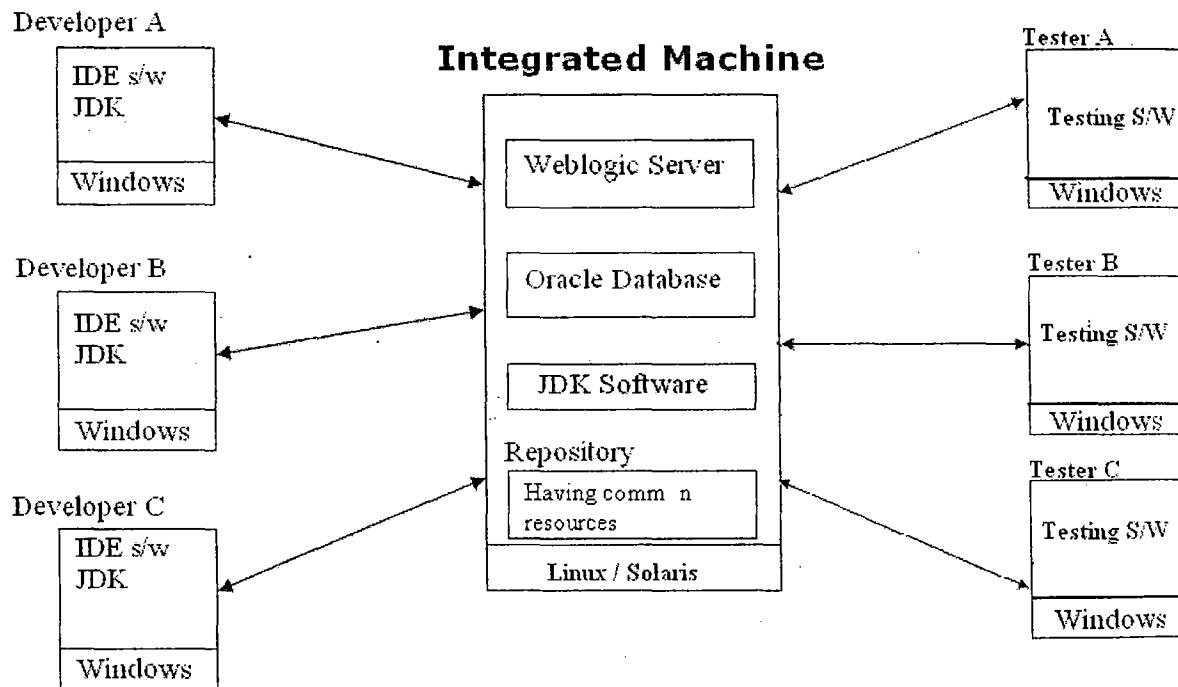
CONTENT:

9. Introduction
 10. Why SVN?
 11. Definition of SVN
 12. Evolution of SVN
 13. SVN Architecture
 14. Repository
 15. Subversion's Component.
 16. Terminology
 - viii) Repository
 - ix) Sandbox
 - x) Check out
 - xi) Commit (check in)
 - xii) Update
 - xiii) History
 - xiv) Revision
 17. Software Description
 18. Repository Operations
 19. Software Installation
 20. Working with Tortoise
- Server software SVN
- Procedure to create a Repository for project/Module
- IDE : Eclipse
- SVN Plug-in configuration with eclipse.

SVN (Version Control System)

Introduction:

In real time all developers and testers machines will be there running in windows environment but all these machines will be connected to a common machine of company called **integrated machine**. Generally this integrated machine resides in linux or solaris environment having high configuration and also contains the common software that are required for multiple projects of company.



The multiple projects of company will use the multiple logical databases that are created in database software of integrated machine on one per project basis.

DURGA SOFTWARE SOLUTIONS

Tools Material

During development mode of the project, the project will be maintained in the CVS Repository or SVN Repository to make the resources of the project visible and accessible for all developers of the project.

Definition : **SVN** is a version control system. Using it, developers can record the history of their source files. This is also called as Source Code Management.

CollabNet Inc. developed SVN in 2000.

- ❖ The SVN repository keeps track of various operations that are done in the files by developers by accessing the files by developers by accessing the files from SVN repository.
- ❖ Subversion can use the HTTP-based WebDAV/DeltaV protocol for network communications, and the Apache web server to provide repository-side network service. This gives Subversion an advantage over CVS in interoperability, and provides various key features for free: authentication, path-based authorization, wire compression, and basic repository browsing.
- ❖ Depending on your Operating System, you might choose the 32-bit or 64-bit versions and download.
- ❖ SVN repository keeps track of various modifications done in files by different developers by generating versions.

Ex:

- Test.java (Original file)
- Test.java 1.1 (after first modification)
- Test.java 1.2 (after second modification)
- Test.java 1.3 (after third modification)

Goals:

There are three basic goals of a version control system (VCS):

- ❖ We want people to be able to work simultaneously, not serially. Think of your team as a multi-threaded piece of software with each developer running in his own thread. The key to high performance in a multi-threaded system is to maximize concurrency. Our goal is to never have a thread which is blocked on some other thread.

DURGA SOFTWARE SOLUTIONS

Tools Material

- ❖ When people are working at the same time, we want their changes to not conflict with each other.

Multi-threaded programming requires great care on the part of the developer and special features such as critical sections, locks, and a test-and-set instruction on the CPU. Without these kinds of things, the threads would overwrite each other's data. A multi-threaded software team needs things too, so that developers can work without messing each other up. That is what the version control system provides.

- ❖ We want to archive every version of everything that has ever existed – ever. And who did it. And when. And why.

Benefits of Source Code Management:

- ✓ All code changes are tracked.
- ✓ Avoid losing work due to simple mistakes (Allows you to roll back changes).
- ✓ Code changes across several developers can be synchronized.
- ✓ Makes it easy to backup your source code.
- ✓ You can work on several different copies of the same application at the same time.
- ✓ Supervisor can see how the code evolved over time.

Evolution of SVN:

In early 2000, CollabNet, Inc. began seeking developers to write a replacement for CVS. CollabNet offers a collaboration software suite called CollabNet Enterprise Edition (CEE) of which one component is version control. Although CEE used CVS as its initial version control system, CVS's limitations were obvious from the beginning, and CollabNet knew it would eventually have to find something better. Unfortunately, CVS had become the de facto standard in the open source world largely because there wasn't anything better, at least not under a free license. So CollabNet determined to write a new version control system from scratch, retaining the basic ideas of CVS, but without the bugs and misfeatures.

When CollabNet called, Karl immediately agreed to work on the project, and Jim got his employer, Red Hat Software, to essentially donate him to the project for an indefinite period of time. CollabNet hired Karl and Ben Collins-Sussman, and detailed design work began in May. With the help of some well-placed prods from Brian Behlendorf and Jason

DURGA SOFTWARE SOLUTIONS Tools Material

Robbins of CollabNet, and Greg Stein (at the time an independent developer active in the WebDAV/DeltaV specification process), Subversion quickly attracted a community of active developers and welcomed the chance to finally do something about it. After fourteen months of coding, Subversion became "self-hosting" on August 31, 2001. That is, Subversion developers stopped using CVS to manage Subversion's own source code, and started using Subversion instead.

Subversion's Architecture:

On one end is a Subversion repository that holds all of your versioned data. On the other end is your Subversion client program, which manages local reflections of portions of that versioned data (called "working copies"). Between these extremes are multiple routes through various Repository Access (RA) layers. Some of these routes go across computer networks and through network servers which then access the repository. Others bypass the network altogether and access the repository directly.

Subversion Installation:

Subversion is built on a portability layer called APR—the Apache Portable Runtime library. The APR library provides all the interfaces that Subversion needs to function on different operating systems: disk access, network access, memory management, and so on. While Subversion is able to use Apache as one of its network server programs, its dependence on APR does not mean that Apache is a required component. APR is a standalone library useable by any application. It does mean, however, that like Apache, Subversion clients and servers run on any operating system that the Apache httpd server runs on: Windows, Linux, all flavors of BSD, Mac OS X, Netware, and others.

DURGA SOFTWARE SOLUTIONS Tools Material

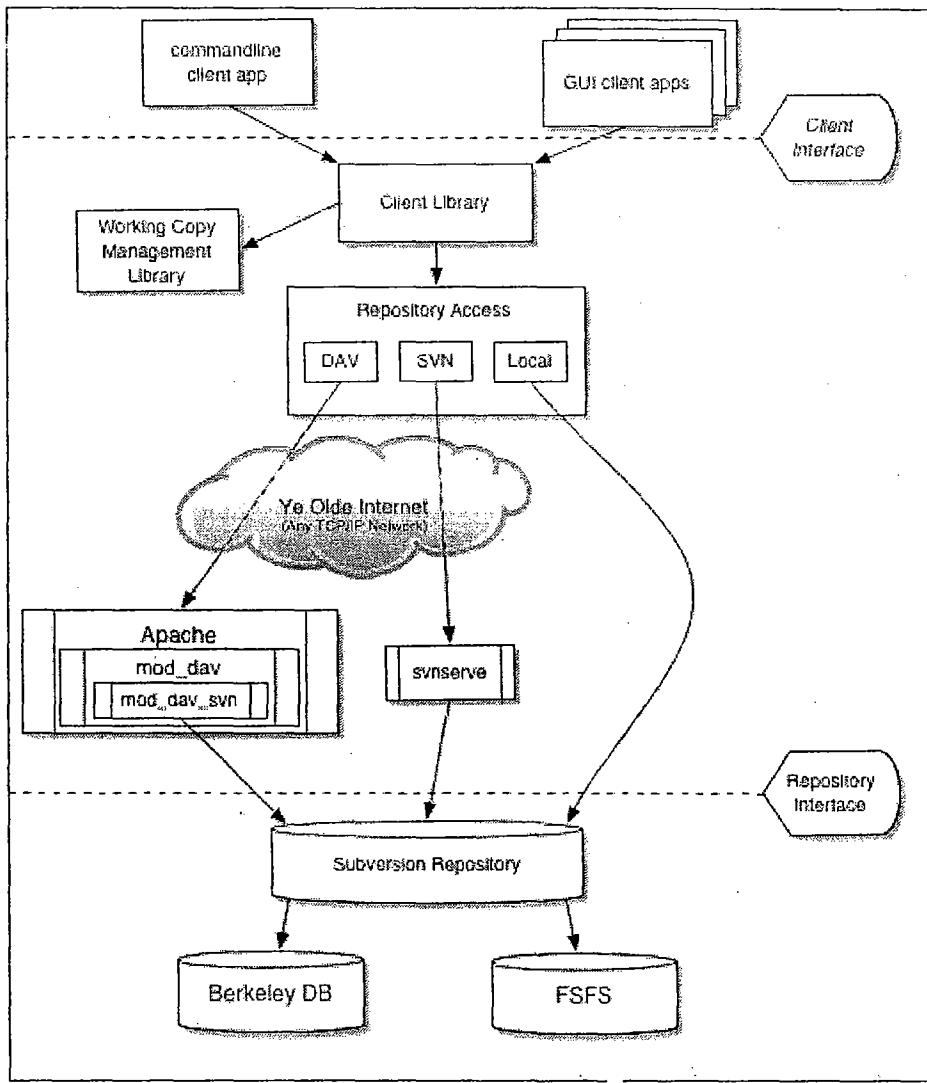
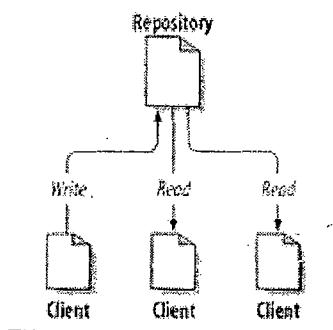


Figure 1.1 Subversion's Architecture

Repository:

Subversion is a centralized system for sharing information. At its core is a repository, which is a central store of data. The repository stores information in the form of a filesystem tree—a typical hierarchy of files and directories. Any number of clients connect to the repository, and then read or write to these files. By writing data, a client makes the information available to others; by reading data, the client receives information from others. Figure illustrates this.

DURGA SOFTWARE SOLUTIONS Tools Material



What makes the Subversion repository special is that it remembers every change ever written to it: every change to every file, and even changes to the directory tree itself, such as the addition, deletion, and rearrangement of files and directories.

When a client reads data from the repository, it normally sees only the latest version of the filesystem tree. But the client also has the ability to view previous states of the filesystem.

For example, a client can ask historical questions like, "What did this directory contain last

Wednesday?" or "Who was the last person to change this file, and what changes did he make?". These are the sorts of questions that are at the heart of any version control system: systems that are designed to record and track changes to data over time.

Versioning Models:

The core mission of a version control system is to enable collaborative editing and sharing of data. But different systems use different strategies to achieve this.

Problem of File-Sharing:

All version control systems have to solve the same fundamental problem: how will the system allow users to share information, but prevent them from accidentally stepping on each other's feet? It's all too easy for users to accidentally overwrite each other's changes in the repository.

Consider the scenario, Suppose we have two co-workers, Harry and Sally.

They each decide to edit the same repository file at the same time. If Harry saves his changes to the repository first, then it's possible that (a few moments later) Sally could accidentally

DURGA SOFTWARE SOLUTIONS
Tools Material

overwrite them with her own new version of the file. While Harry's version of the file won't be lost forever (because the system remembers every change), any changes Harry made won't be present in Sally's newer version of the file, because she never saw Harry's changes to begin with. Harry's work is still effectively lost—or at least missing from the latest version of the file—and probably by accident. This is definitely a situation we want to avoid!

Lock-Modify-Unlock Solution:

Many version control systems use a lock-modify-unlock model to address the problem of many authors clobbering each other's work. In this model, the repository allows only one person to change a file at a time. This exclusivity policy is managed using locks. Harry must "lock" a file before he can begin making changes to it. If Harry has locked a file, then Sally cannot also lock it, and therefore cannot make any changes to that file. All she can do is read the file, and wait for Harry to finish his changes and release his lock. After Harry unlocks the file, Sally can take her turn by locking and editing the file. Figure demonstrates this simple solution.

DURGA SOFTWARE SOLUTIONS Tools Material

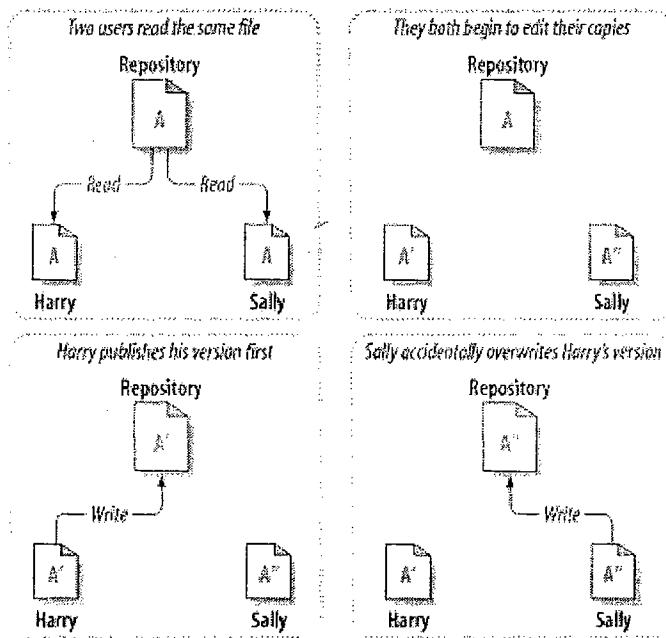


Figure 2.2 The problem to avoid

The problem with the lock-modify-unlock model is that it's a bit restrictive, and often becomes a roadblock for users:

- **Locking may cause administrative problems.**

Sometimes Harry will lock a file and then forget about it. Meanwhile, because Sally is still waiting to edit the file, her hands are tied. And then Harry goes on vacation. Now Sally has to get an administrator to release Harry's lock. The situation ends up causing a lot of unnecessary delay and wasted time.

- **Locking may cause unnecessary serialization.**

What if Harry is editing the beginning of a text file, and Sally simply wants to edit the end of the same file? These changes don't overlap at all. They could easily edit the file simultaneously, and no great harm would come, assuming the changes were properly merged together. There's no need for them to take turns in this situation.

- **Locking may create a false sense of security.**

Pretend that Harry locks and edits file A, while Sally simultaneously locks and edits file

DURGA SOFTWARE SOLUTIONS

Tools Material

B. But suppose that A and B depend on one another, and the changes made to each are semantically incompatible. Suddenly A and B don't work together anymore. The locking system was powerless to prevent the problem—yet it somehow provided a false sense of security. It's easy for Harry and Sally to imagine that by locking files, each is beginning a safe, insulated task, and thus not bother discussing their incompatible changes early on.

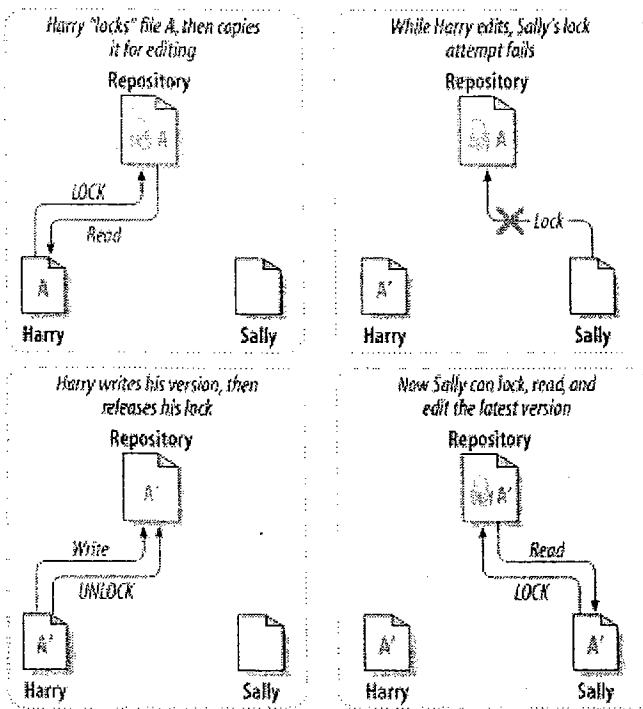


Figure 2.3 lock-modify-unlock solution

Copy-Modify-Merge Solution:

Subversion, and other version control systems use a copy-modify-merge model as an alternative to locking. In this model, each user's client contacts the project repository and creates a personal working copy—a local reflection of the repository's files and directories.

Users then work in parallel, modifying their private copies. Finally, the private copies are merged together into a new, final version. The version control system often assists with the merging, but ultimately a human being is responsible for making it happen correctly. Here's an example. Say that Harry and Sally each create working copies of the same project,

DURGA SOFTWARE SOLUTIONS
Tools Material

copied from the repository. They work concurrently, and make changes to the same file A within their copies. Sally saves her changes to the repository first. When Harry attempts to save his changes later, the repository informs him that his file A is out-of-date. In other words, that file A in the repository has somehow changed since he last copied it. So Harry asks his client to merge any new changes from the repository into his working copy of file A. Chances are that Sally's changes don't overlap with his own; so once he has both sets of changes integrated, he saves his working copy back to the repository. **Figure 2.4** and **Figure 2.5** show this process.

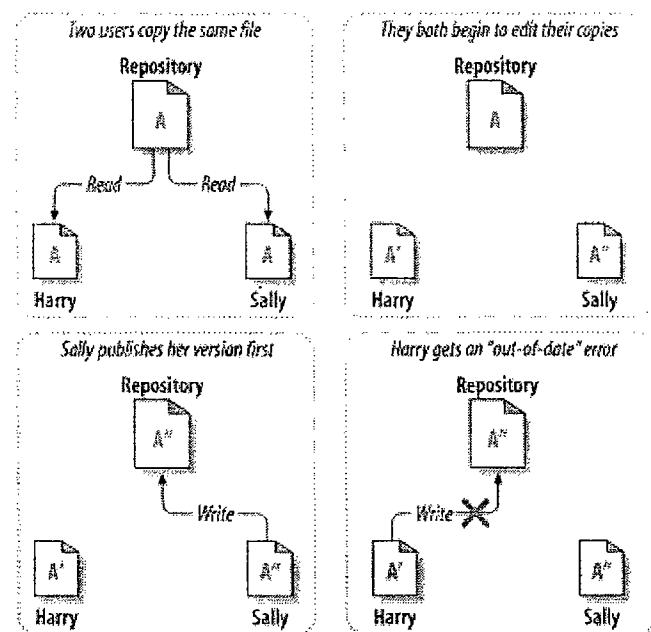


Figure 2.4 copy-modify-merge solution

DURGA SOFTWARE SOLUTIONS

Tools Material

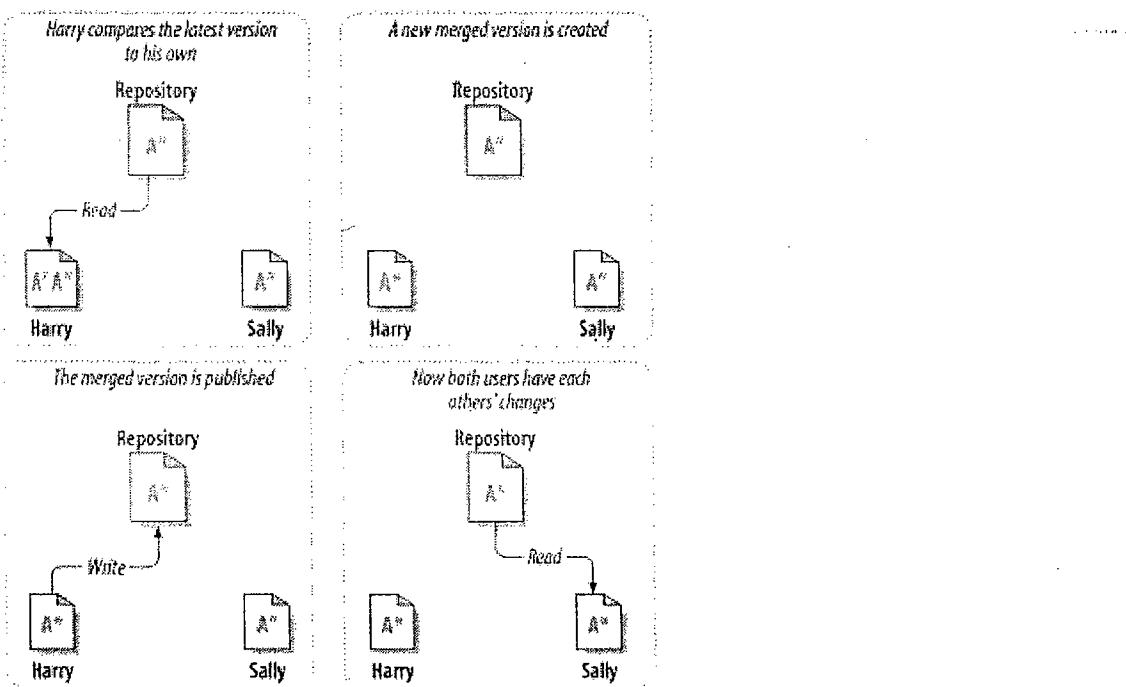


Figure 2.5 copy-modify-merge solution.

But what if Sally's changes do overlap with Harry's changes? What then? This situation is called a conflict, and it's usually not much of a problem. When Harry asks his client to merge

the latest repository changes into his working copy, his copy of file A is somehow flagged as

being in a state of conflict: he'll be able to see both sets of conflicting changes, and manually

choose between them. Note that software can't automatically resolve conflicts; only humans

are capable of understanding and making the necessary intelligent choices. Once Harry has

manually resolved the overlapping changes—perhaps after a discussion with Sally—he can

safely save the merged file back to the repository.

The copy-modify-merge model may sound a bit chaotic, but in practice, it runs extremely smoothly. Users can work in parallel, never waiting for one another. When they work on the same files, it turns out that most of their concurrent changes don't overlap at all; conflicts are infrequent. And the amount of time it takes to resolve conflicts is far less than the time lost by a locking system.

Subversion's Components:

Svn

DURGA SOFTWARE SOLUTIONS
Tools Material

The command-line client program

svnversion

A program for reporting the state (in terms of revisions of the items present) of a working copy

svnlook

A tool for directly inspecting a Subversion repository

svnadmin

A tool for creating, tweaking, or repairing a Subversion repository

mod_dav_svn

A plug-in module for the Apache HTTP Server, used to make your repository available to others over a network

svnserve

A custom standalone server program, runnable as a daemon process or invokable by SSH; another way to make your repository available to others over a network

svndumpfilter

A program for filtering Subversion repository dump streams

svnsync

A program for incrementally mirroring one repository to another over a network

svnrdump

A program for performing repository history dumps and loads over a network

Basic Work Cycle:

The most common things that you might find yourself doing with Subversion in the course of a day's work.

DURGA SOFTWARE SOLUTIONS

Tools Material

- Update your working copy
 - svn update
- Make changes
 - svn add
 - svn delete
 - svn copy
 - svn move
- Examine your changes
 - svn status
 - svn diff
 - svn revert
- Merge others' changes
 - svn merge
 - svn resolved
- Commit your changes
 - svn commit

Access Subversion Repositories:

Subversion repositories can be accessed through many different methods—on local disk, or through various network protocols.

Schema	Access Method
<code>file:///</code>	direct repository access (on local disk).
<code>http://</code>	access via WebDAV protocol to Subversion-aware Apache server
<code>Svn://</code>	access via custom protocol to an svnserve server
<code>https://</code>	same as <code>http://</code> , but with SSL encryption.
<code>Svn+ssh://</code>	same as <code>svn://</code> , but through an SSH tunnel

Working Copies Track the Repository:

DURGA SOFTWARE SOLUTIONS

Tools Material

A Subversion working copy is an ordinary directory tree on your local system, containing a collection of files. You can edit these files however you wish, and if they're source code files, you can compile your program from them in the usual way. Your working copy is your own private work area: Subversion will never incorporate other people's changes, nor make your own changes available to others, until you explicitly tell it to do so. You can even have multiple working copies of the same project.

After you've made some changes to the files in your working copy and verified that they work properly, Subversion provides you with commands to "publish" your changes to the other people working with you on your project (by writing to the repository). If other people publish their own changes, Subversion provides you with commands to merge those changes into your working directory (by reading from the repository).

Given this information, by talking to the repository, Subversion can tell which of the following four states a working file is in:

Unchanged, and current

The file is unchanged in the working directory, and no changes to that file have been committed to the repository since its working revision. An **svn commit** of the file will do nothing, and an **svn update** of the file will do nothing.

Locally changed, and current

The file has been changed in the working directory, and no changes to that file have been committed to the repository since its base revision. There are local changes that have not been committed to the repository, thus an **svn commit** of the file will succeed in publishing your changes, and an **svn update** of the file will do nothing.

Unchanged, and out-of-date

The file has not been changed in the working directory, but it has been changed in the repository. The file should eventually be updated, to make it current with the public revision. An **svn commit** of the file will do nothing, and an **svn update** of the file will fold the latest changes into your working copy.

DURGA SOFTWARE SOLUTIONS
Tools Material

Locally changed, and out-of-date

The file has been changed both in the working directory, and in the repository. An **svn commit** of the file will fail with an “out-of-date” error. The file should be updated first; an **svn update** command will attempt to merge the public changes with the local changes. If Subversion can’t complete the merge in a plausible way automatically, it leaves it to the user to resolve the conflict.

svn status command will show you the state of any item in your working copy.

Revision Keywords:

The Subversion client understands a number of revision keywords. These keywords can be used instead of integer arguments to the --revision switch, and are resolved into specific revision numbers by Subversion:

HEAD

The latest (or “youngest”) revision in the repository.

BASE

The revision number of an item in a working copy. If the item has been locally modified, the “BASE version” refers to the way the item appears without those local modifications.

COMMITTED

The most recent revision prior to, or equal to, BASE, in which an item changed.

PREV

The revision immediately before the last revision in which an item changed. (Technically, COMMITTED-1).

Note: PREV, BASE, and COMMITTED can be used to refer to local paths, but not to URLs.

Examining History:

There are several commands that can provide you with historical data from the repository:

svn log

Shows you broad information: log messages with date and author information attached

DURGA SOFTWARE SOLUTIONS Tools Material

to revisions, and which paths changed in each revision.

svn diff

Shows you the specific details of how a file changed over time.

The three distinct uses of **svn diff**:

- Examine local changes
- Compare your working copy to the repository
- Compare repository to repository

svn cat

To examine an earlier version of a file and not necessarily the differences between two files,

svn list

Displays the files in a directory for any given revision.

Branching and Merging

Branching, tagging, and merging are concepts common to almost all version control systems.

Branching is a fundamental part of version control.

Branch:

The basic concept of branch—namely, a line of development that exists independently of another line, yet still shares a common history if you look far enough back in time. A branch always begins life as a copy of something, and moves on from there, generating its own history

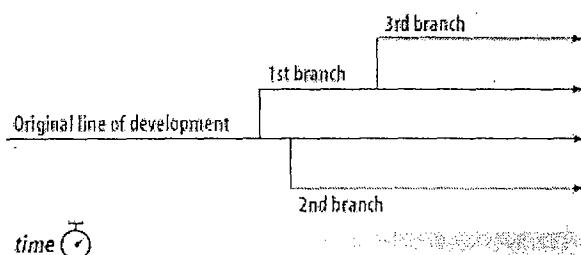


Figure Branches of development

DURGA SOFTWARE SOLUTIONS Tools Material

Subversion has commands to help you maintain parallel branches of your files and directories. It allows you to create branches by copying your data, and remembers that the copies are related to one another. It also helps you duplicate changes from one branch to another. Finally, it can make portions of your working copy reflect different branches, so that you can "mix and match" different lines of development in your daily work.

Creating a Branch:

Creating a branch is very simple: you make a copy of the project in the repository using the **svn copy** command. Subversion is not only able to copy single files, but whole directories as well.

There are two different ways to make a copy. We'll demonstrate the messy way first, just to make the concept clear.

To begin, check out a working copy of the project's root directory, /calc:

```
$ svn checkout http://svn.example.com/repos/calc bigwc
```

```
A bigwc/trunk/  
A bigwc/trunk/Makefile  
A bigwc/trunk/integer.c  
A bigwc/trunk/button.c  
A bigwc/branches/ .
```

Checked out revision 340.

Making a copy is now simply a matter of passing two working-copy paths to the **svn copy** command:

```
svn copy is able to operate directly on two URLs.  
$ svn copy http://svn.example.com/repos/calc/trunk \  
http://svn.example.com/repos/calc/branches/my-calc-branch \  
$ svn status  
A + branches/my-calc-branch  
Committed revision 341.
```

Cheap Copies:

Subversion's repository has a special design. When you copy a directory, you don't need to worry about the repository growing huge—Subversion doesn't actually duplicate any data. Instead, it creates a new directory entry that points to an existing tree. If you're a Unix user, this is the same concept as a hard-link. From there, the copy is said to be "lazy". That is, if you commit a change to one file within the copied directory, then only that file changes—the rest of the files continue to exist as links to the original files in the original directory.

DURGA SOFTWARE SOLUTIONS

Tools Material

It takes a very tiny, constant amount of time to make a copy of it. In fact, this feature is the basis of how commits work in Subversion: each revision is a “cheap copy” of the previous revision, with a few items lazily changed within.

Creating the Layout, and Importing Initial Data:

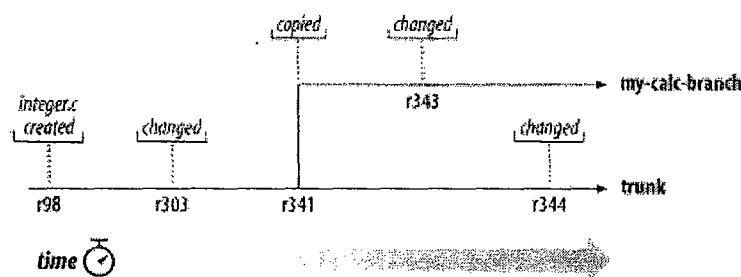
After deciding how to arrange the projects in your repository, you'll probably want to actually populate the repository with that layout and with initial project data.

.svn directory:

Every directory in a working copy contains an administrative area, a subdirectory named .svn. Usually, directory listing commands won't show this subdirectory, but it is nevertheless an important directory. Whatever you do, don't delete or change anything in the administrative area! Subversion depends on it to manage your working copy.

The branching of one file's history:

Things get interesting when you look at the history of changes made to your copy of integer.c:



Copying Changes Between Branches:

For projects that have a large number of contributors, it's common for most people to have working copies of the trunk. Whenever someone needs to make a long-running change that is likely to disrupt the trunk, a standard procedure is to create a private branch and commit changes there until all the work is complete.

Key Concepts Behind Branches:

There are two important lessons that you should remember.

DURGA SOFTWARE SOLUTIONS Tools Material

1. Unlike many other version control systems, Subversion's branches exist as normal filesystem directories in the repository, not in an extra dimension. These directories just happen to carry some extra historical information.
2. Subversion has no internal concept of a branch—only copies. When you copy a directory, the resulting directory is only a “branch” because you attach that meaning to it. You may think of the directory differently, or treat it differently, but to Subversion it's just an ordinary directory that happens to have been created by copying.

Merging:

The term “**merge**” somehow denotes that branches are combined together.

The **svn merge** command is almost exactly the same. Instead of printing the differences to your terminal, however, it applies them directly to your working copy as local modifications.

Svn diff-and-apply, Compares two repository trees and differences are applied to a working copy.

The command takes three arguments:

1. An initial repository tree (often called the left side of the comparison),
2. A final repository tree (often called the right side of the comparison),
3. A working copy to accept the differences as local changes (often called the target of the merge).

Once these three arguments are specified, the two trees are compared, and the resulting differences are applied to the target working copy as local modifications.

When the command is done.

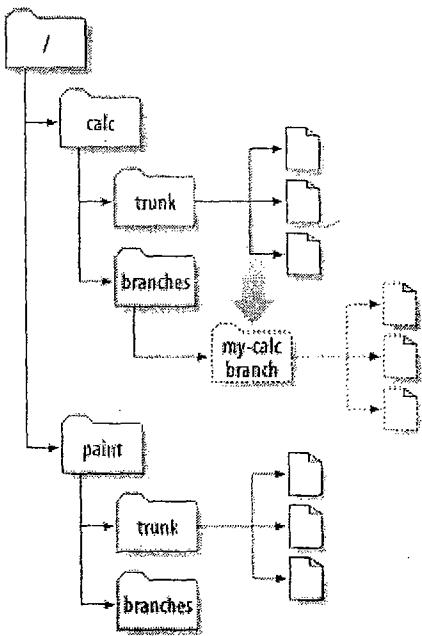
The syntax of **svn merge** allows you to specify the three necessary arguments rather flexibly.

Here are some examples:

```
$ svn merge http://svn.example.com/repos/branch1@150 \
http://svn.example.com/repos/branch2@212 \ my-working-copy
$ svn merge -r 100:200 http://svn.example.com/repos/trunk my-working-copy
$ svn merge -r 100:200 http://svn.example.com/repos/trunk
```

The first syntax lays out all three arguments explicitly, naming each tree in the form URL@REV and naming the working copy target. The second syntax can be used as a shorthand for situations when you're comparing two different revisions of the same URL. The last syntax shows how the working-copy argument is optional; if omitted, it defaults to the current directory.

DURGA SOFTWARE SOLUTIONS
Tools Material



Common Branching Patterns:

Branch Releases:

Most software has a typical lifecycle: code, test, release, repeat. There are two problems with this process. First, developers need to keep writing new features while quality-assurance teams take time to test supposedly-stable versions of the software. New work cannot halt while the software is tested. Second, the team almost always needs to support older, released versions of software; if a bug is discovered in the latest code, it most likely exists in released versions as well, and customers will want to get that bugfix without having to wait for a major new release.

Here's where version control can help. The typical procedure looks like this:

- Developers commit all new work to the trunk.

Day-to-day changes are committed to /trunk: new features, bugfixes, and so on.

- The trunk is copied to a "release" branch.

When the team thinks the software is ready for release (say, a 1.0 release), then /trunk might be copied to /branches/1.0.

- Teams continue to work in parallel.

One team begins rigorous testing of the release branch, while another team continues new work (say, for version 2.0) on /trunk. If bugs are discovered in either location, fixes are ported back and forth as necessary. At some point, however, even that process stops. The branch is "frozen" for final testing right before a release.

DURGA SOFTWARE SOLUTIONS

Tools Material

- The branch is tagged and released. When testing is complete, /branches/1.0 is copied to /tags/1.0.0 as a reference snapshot. The tag is packaged and released to customers.
- The branch is maintained over time. While work continues on /trunk for version 2.0, bugfixes continue to be ported from /trunk to /branches/1.0. When enough bugfixes have accumulated, management may decide to do a 1.0.1 release: /branches/1.0 is copied to /tags/1.0.1, and the tag is packaged and released.

This entire process repeats as the software matures: when the 2.0 work is complete, a new 2.0 release branch is created, tested, tagged, and eventually released. After some years, the repository ends up with a number of release branches in “maintenance” mode, and a number of tags representing final shipped versions.

Tags:

Another common version control concept is a tag. A tag is just a “snapshot” of a project in time. In Subversion, this idea already seems to be everywhere. Each repository revision is exactly that—a snapshot of the filesystem after each commit.

However, people often want to give more human-friendly names to tags, like release-1.0. And they want to make snapshots of smaller subdirectories of the filesystem. After all, it’s not so easy to remember that release-1.0 of a piece of software is a particular subdirectory of revision 4822.

Syntax:

```
$ svn copy http://svn.example.com/repos/calc/trunk \
http://svn.example.com/repos/calc/tags/release-1.0 \
-m "Tagging the 1.0 release of the 'calc' project."
Committed revision 351.
```

In Subversion, there’s no difference between a tag and a branch. Both are just ordinary directories that are created by copying. Just as with branches, the only reason a copied directory is a “tag” is because humans have decided to treat it that way: as long as nobody ever commits to the directory, it forever remains a snapshot. If people start committing to it, it becomes a branch.

Branch Maintenance:

Subversion is extremely flexible. Because it implements branches and tags with the same underlying mechanism (directory copies), and because branches and tags appear in normal filesystem space, many people find Subversion intimidating. It’s almost *too* flexible. In this section, we’ll offer some suggestions for arranging and managing your data over time.

DURGA SOFTWARE SOLUTIONS
Tools Material

Repository Layout:

There are some standard, recommended ways to organize a repository. Most people create a trunk directory to hold the “main line” of development, a branches directory to contain branch copies, and a tags directory to contain tag copies. If a repository holds only one project, then often people create these top-level directories :

/trunk
/branches
/tags

If a repository contains multiple projects, admins typically index their layout by project (see Section 5.4.1 to read more about “**project roots**”):

/paint/trunk
/paint/branches
/paint/tags
/calc/trunk
/calc/branches
/calc/tags

Terminology:

- **Repository** : area on the server where the files are stored
- **Sandbox** : a local copy of the code which you work on and then commit to the repository
- **Checkout** : Process of collecting resource or project from SVN repository.
- **Commit** : Process of keeping resources back into SVN Repository after doing modifications is called as commit operation or Checkin operation.
- **Update** : getting code changes that have been committed since you checked out the project
- **Merge** : combining changes between two versions of the same file
- **History** : shows a list of commit messages, times and, who committed for a particular file
- **Revision** : SVN assigned version number for a file

DURGA SOFTWARE SOLUTIONS
Tools Material

Software Description:

Some SVN Repository softwares are

- ✓ CVSNT
- ✓ WinCVS
- ✓ Tortoise
- ✓ ClearCase
- ❖ The SVN repository software will be installed on the integrated machine and the IDE software of the developer machines will be configured to interact with SVN Repository.

TORTOISE SVN :

Type : repository software

Version : 1.7.x

Download from : <http://tortoisessvn.net/downloads.html>

REPOSITORY OPERATIONS:

Atomic Commits : Atomic Commits means that, If an operation on the repository is interrupted in the middle, the repository will not be left in an inconsistent state. Are the check-in operations atomic? Are the check-in operations atomic, or can interrupting an operation leave the repository in an intermediate state?

CVS	Commits are not atomic
SVN	Commits are atomic

Files and Directories Moves or Renames: Does the system support moving a file or directory to a different location while still retaining the history of the file?

CVS	No. Renames are not supported and a manual one may break history in two.
SVN	Yes. Renames are supported.

Files and Directories Copies: Does the version control system supports copying files or directories to a different location at the repository level, while retaining the history?

CVS	No. copies are not supported.
SVN	Yes. It's a very cheap operation ($O(1)$) that is also utilized for branching.

DURGA SOFTWARE SOLUTIONS
Tools Material

Remote Repository Replication: Does the system support cloning a remote repository to get a functionally equivalent copy in the local system? That should be done without any special access to the remote server except for normal repository access.

CVS	No
SVN	Indirectly, by using the SVN::Mirror script [2] or the svn-push utility [3].

Propagating Changes to Remote Repositories: Can the system propagate changes from one repository to another?

CVS	No
SVN	Yes, using either the SVN::Mirror script [2] or the svn-push utility [3].

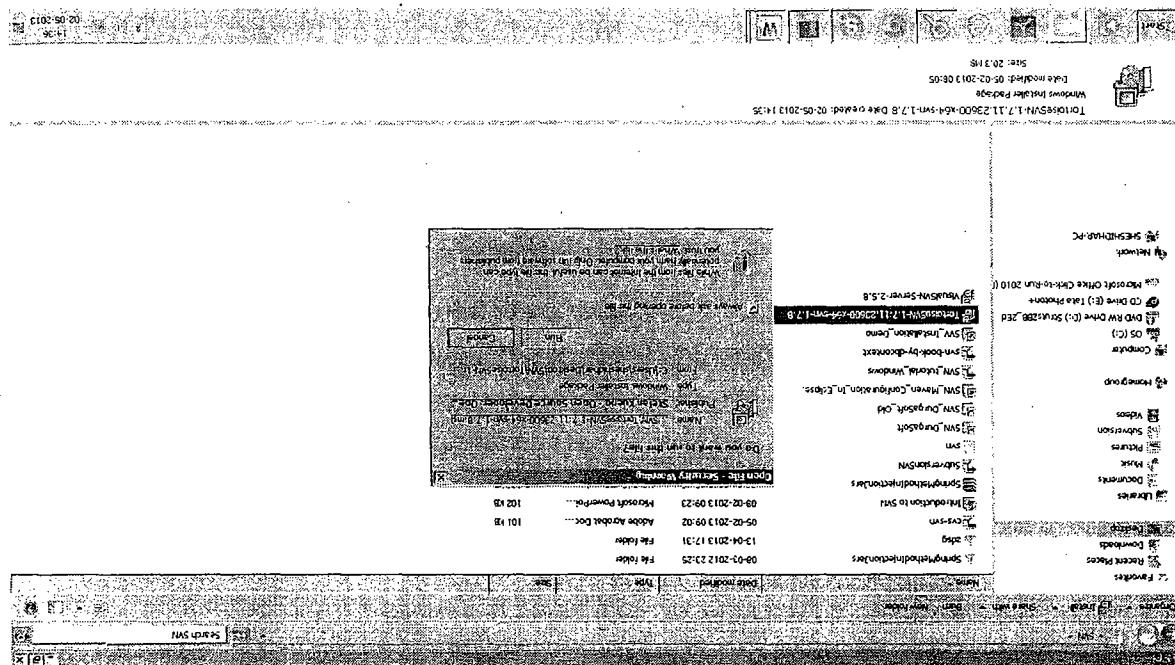
Repository Permissions: Is it possible to define permissions on access to different parts of a remote repository? Or is access open for all?

CVS	Limited. Pre-commit hook scripts' can be used to implement various permissions systems.
SVN	Yes. The WebDAV-based service supports defining HTTP permissions various directories of the repository.

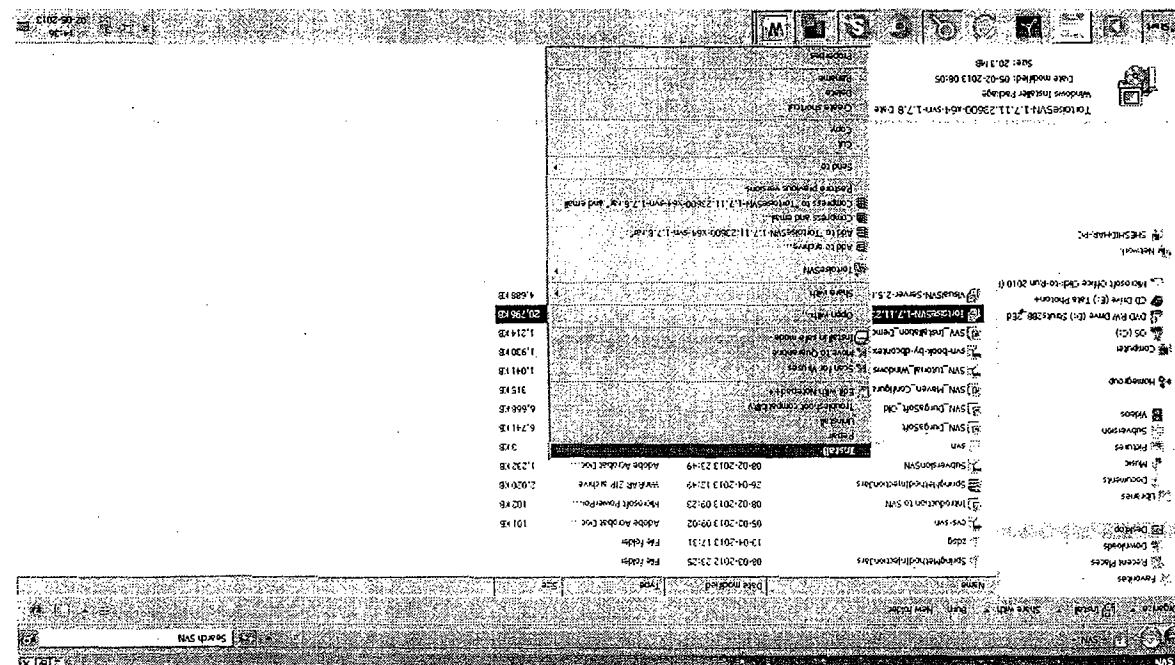
TORTOISE SVN INSTALLATION:

SERVER INSTALLATION STEPS

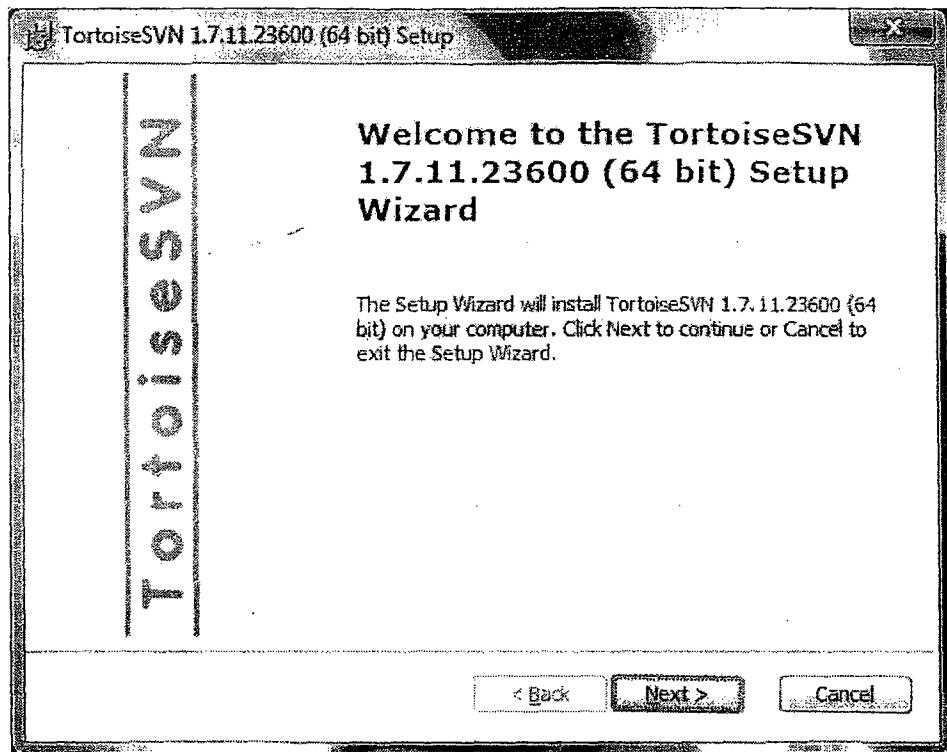
Click on Run.



Right click on TortoiseSVN-1.7.11.23600-x64-svn-1.7.8 click on install.

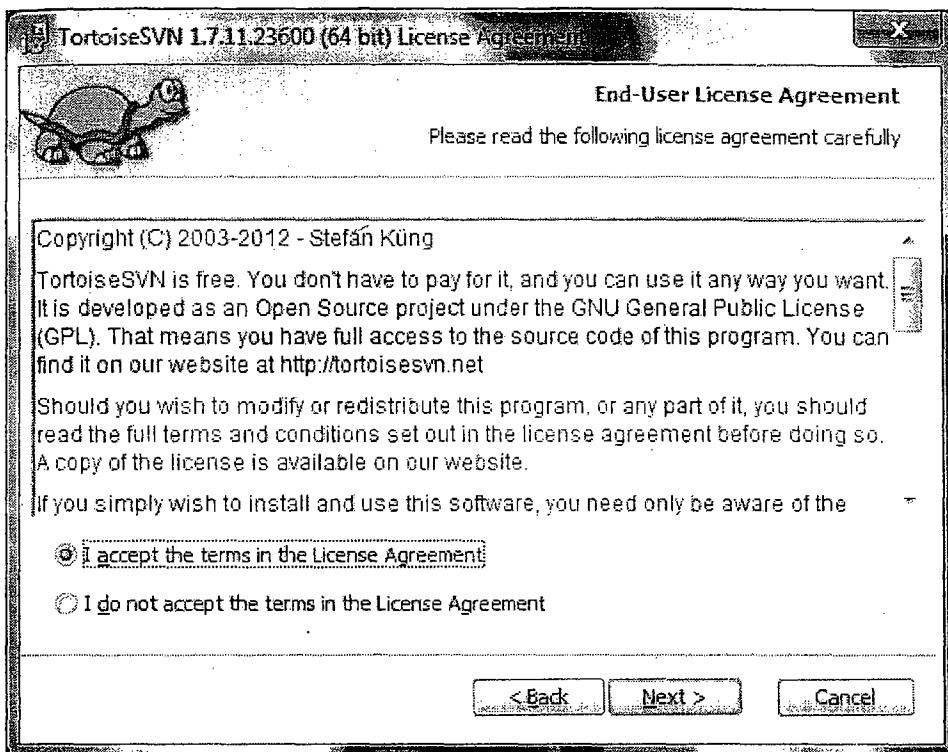


DURGA SOFTWARE SOLUTIONS
Tools Material



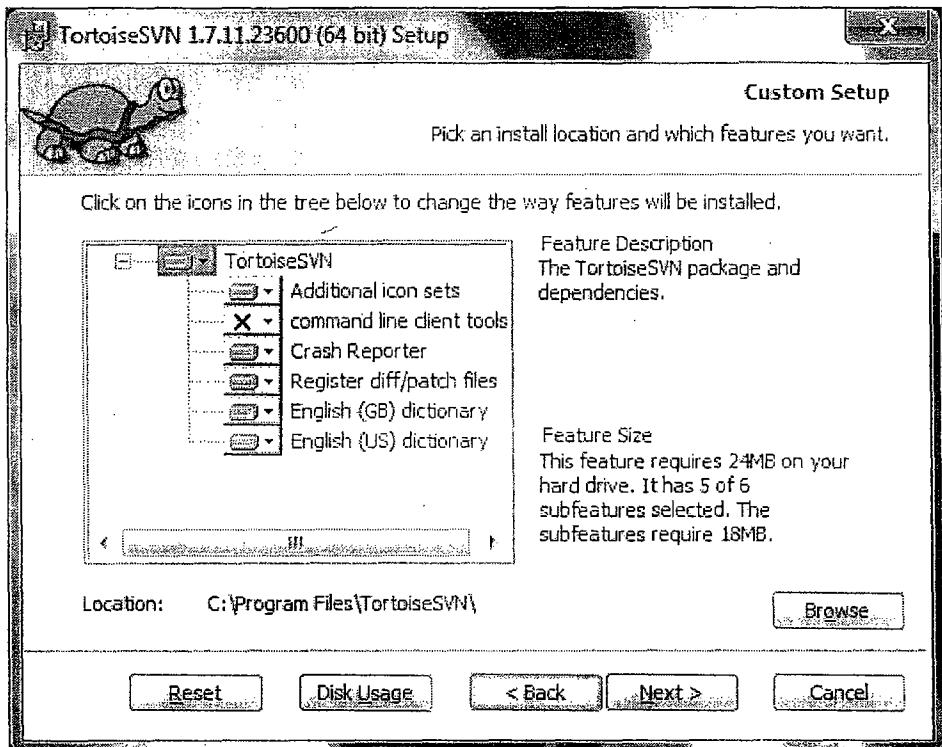
Click on Next

DURGA SOFTWARE SOLUTIONS
Tools Material

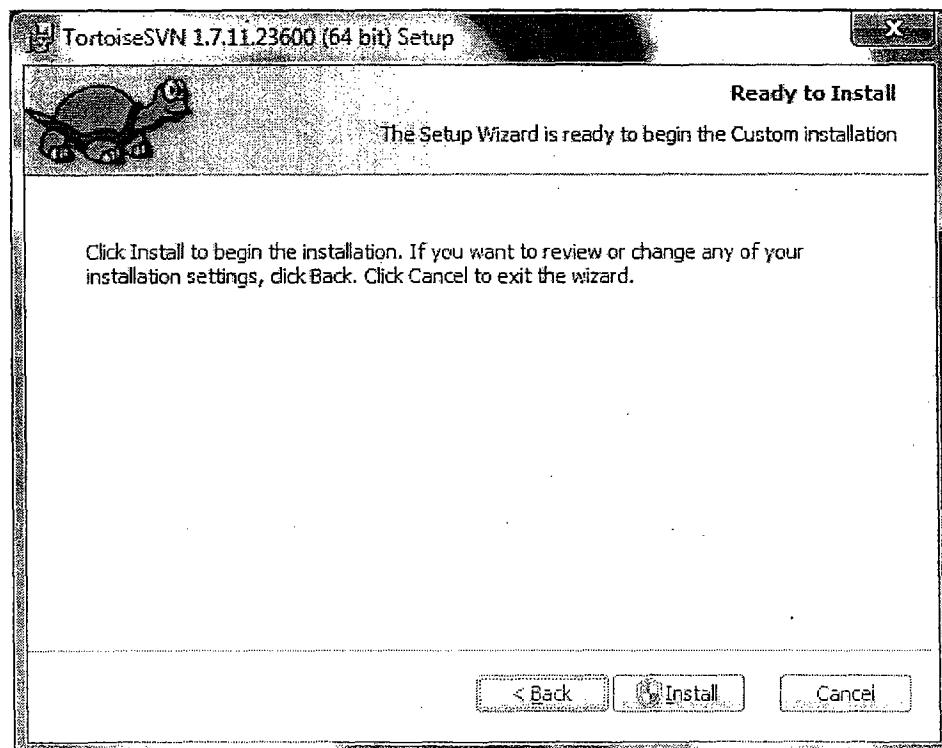


Accept the agreement and click on Next

DURGA SOFTWARE SOLUTIONS
Tools Material

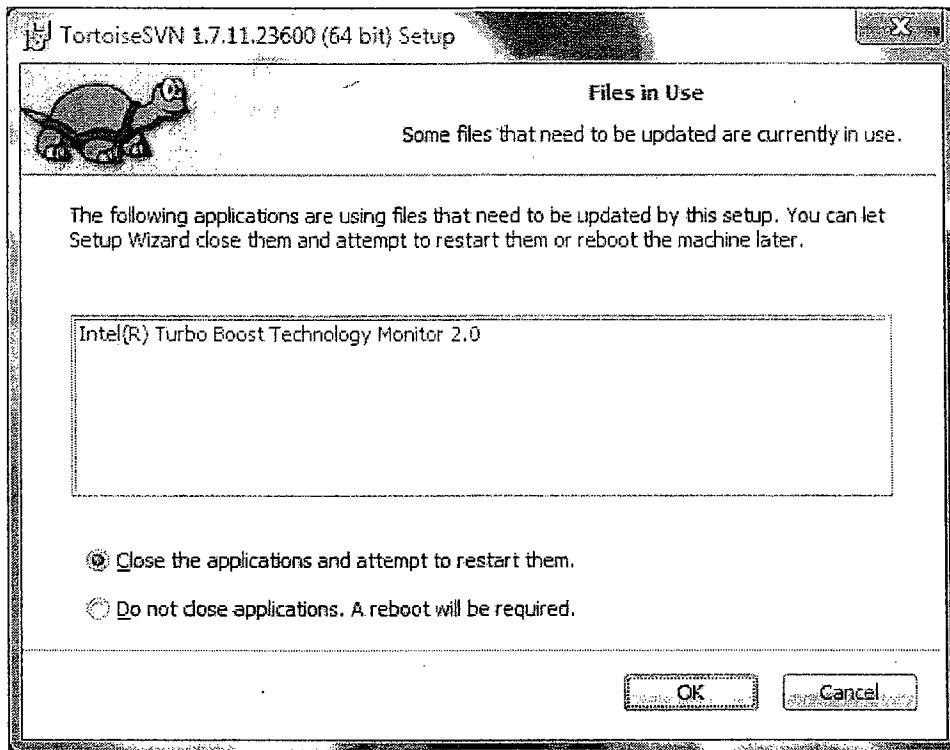


Select the Installation Directory(keep Default Only) and Click on Next



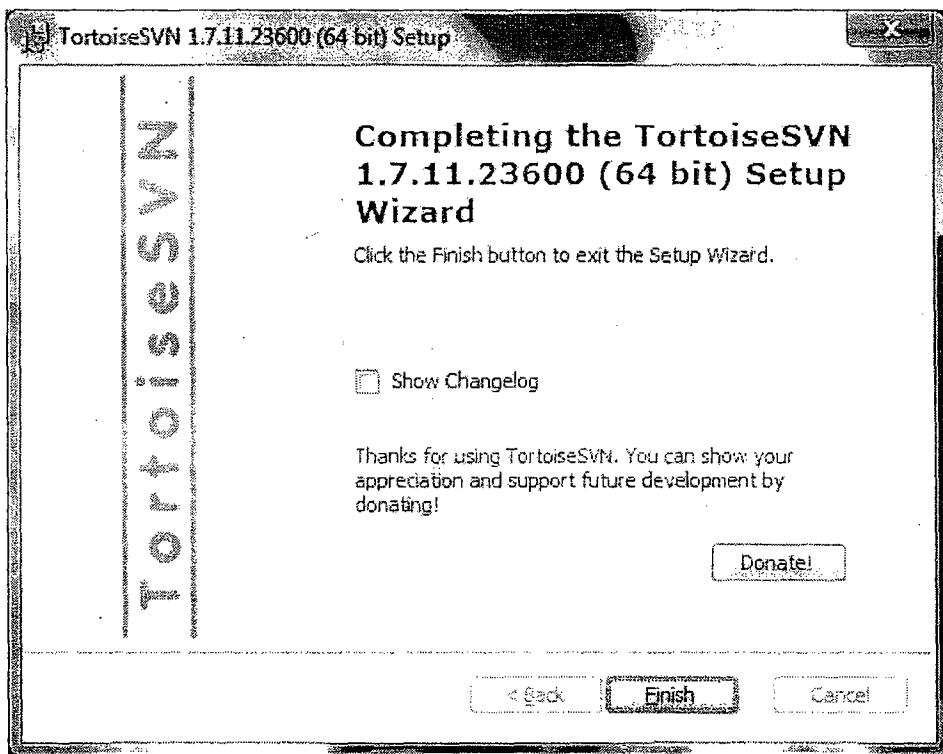
DURGA SOFTWARE SOLUTIONS
Tools Material

Click On Install



Click on OK

DURGA SOFTWARE SOLUTIONS
Tools Material



Click on Finish

After installing TortoiseSVN restart the system.

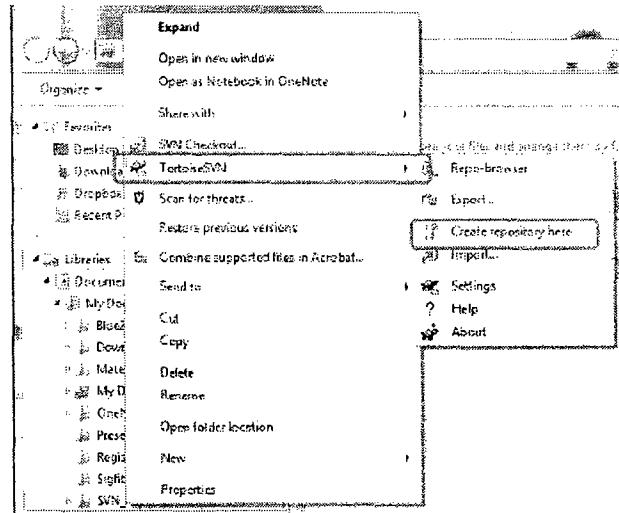
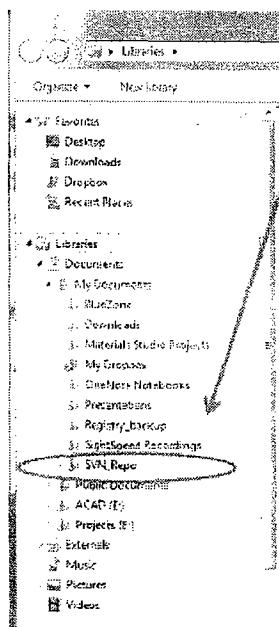
Procedure to create SVN Repository for certain project / module :

DURGA SOFTWARE SOLUTIONS

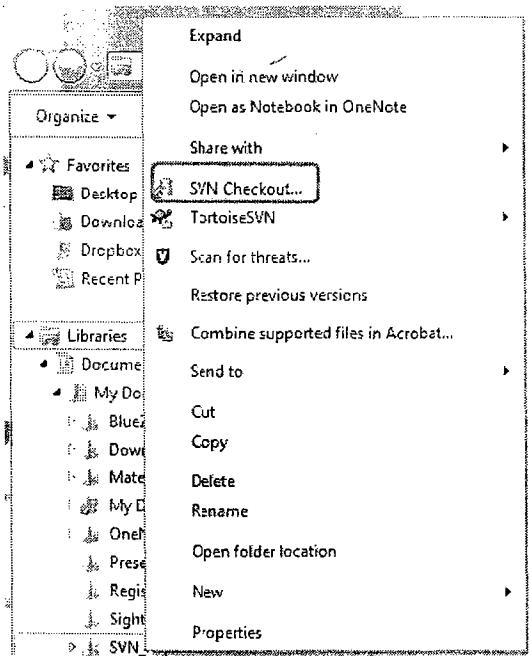
Tools Material

Create a local repository

- Create a new directory somewhere (perhaps under "My Documents"). I called mine "SVN_Repo."
- Right-click on the SVN_Repo folder. The menu that pops up will have a couple of new items: the "SVN Checkout..." and "TortoiseSVN." Choose "TortoiseSVN."
- Click on "Create repository here" from the menu that pops up.

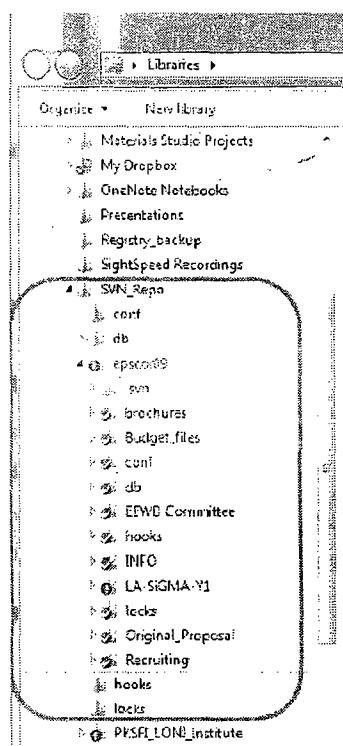


Do the first “Check Out”



- Once the repository is successfully created, right-click on the “SVN_Repo” directory again, and choose “SVN Checkout...”
- The program will ask you for the repo address. For the LA-SiGMA project, it is <https://svn.cct.lsu.edu/repos/proposals/epscor09>
- Enter the user-id and password provided by CCT.
- The checkout process now starts, and creates an exact replica of the SVN file system on your local drive. You should do this step while connected to a fast connection, because our repo has become quite large.
- You should allow a fair amount of time. [I did this at home, on a 54 Mbps wireless, and it took about 2.5 hours.]

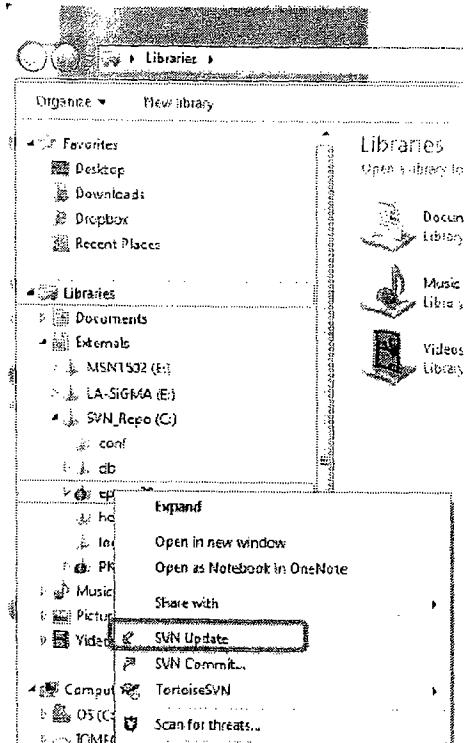
Directory Structure



- After a successful checkout, you will see that several folders have appeared under "SVN_Repo." Mine is shown on left.
- Note the red ! (exclamation) and the green check marks appearing on the directory icons.
- The red symbol on the top level directory (epscor09) means that one or more of the subdirectories contain a file that I have changed, making my local version different from the copy on the SVN.
- The green check marks indicate that all the files in those folders have not been changed locally since the last update.

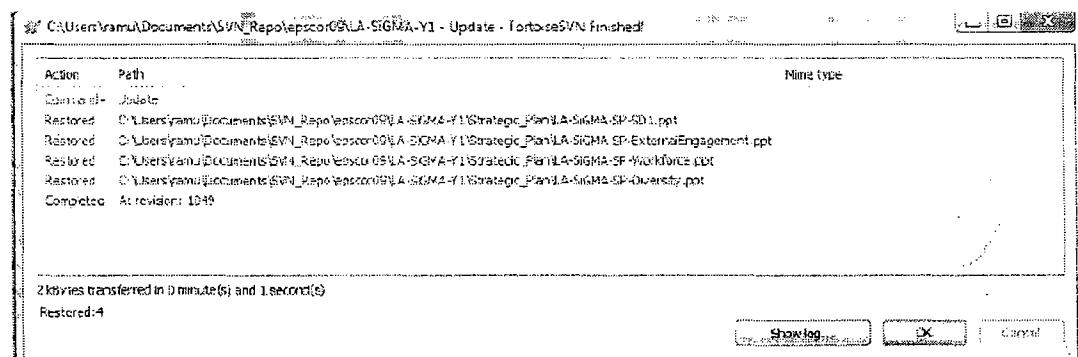
Working with the SVN: Update and Commit

- When working with the SVN on a daily basis, there are just two things to remember: update, and commit.
- Before you start working on any file in the SVN, it is important to make sure you have the most up-to-date versions of all files. For this, you would do an "update" from the SVN menu: right click on the directory icon, and choose "SVN Update."
- The update process copies only the files that have changed since the last update.
- The commit process copies files you may have changed locally to the central SVN (so that others can get it by updating).



DURGA SOFTWARE SOLUTIONS
Tools Material

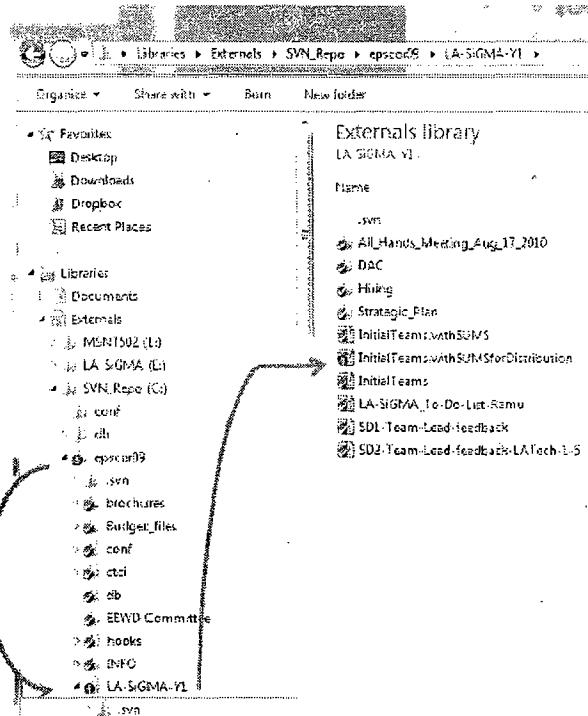
The screen-shot of an update process



I had deliberately deleted four files from my local SVN repository prior to this update. You can see that the update process restored those files from the central SVN. Note that we are now at revision # 1049.

Committing changes - 1

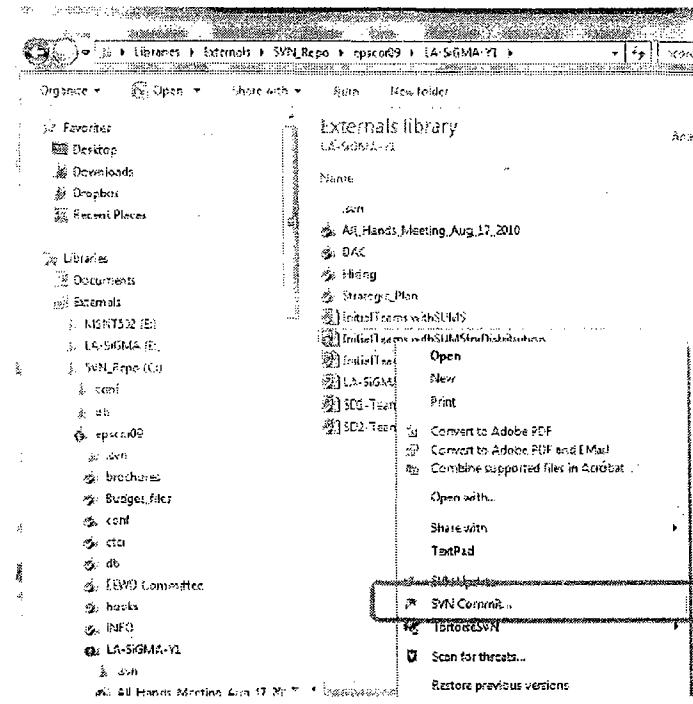
- If you make changes to a file, or if you create a new file, it needs to be "committed" (not in the psychiatric sense in most cases) so that others can see it.
- In the screen shot to the right, I have deliberately modified one of the files. Note that it is marked by a red exclamation mark all the way up the directory tree to the top level.
- Note that all the other files (and sub-folder) have green check marks, indicating that their contents are consistent with the central SVN.
- If you create a new file, it will not have any mark on it, which is a clue that the central SVN knows nothing about that file's existence.



DURGA SOFTWARE SOLUTIONS
Tools Material

Committing Changes - 2

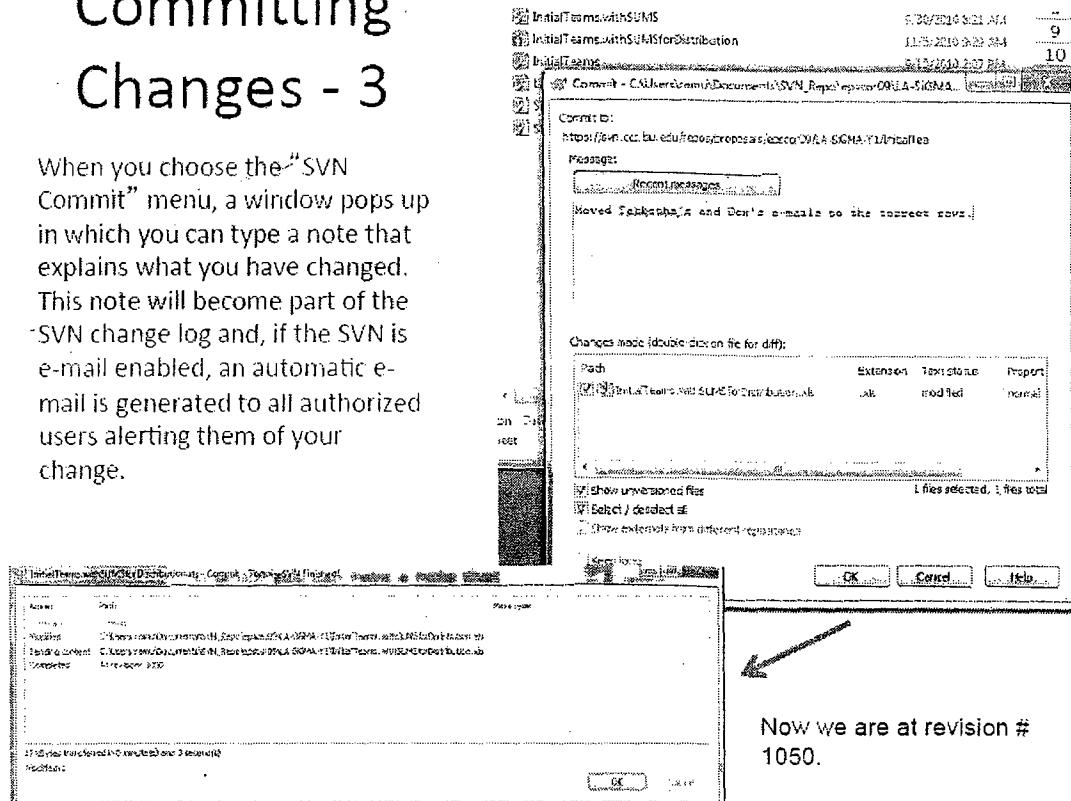
The screen shot shows that I am about to commit the file that I changed.



DURGA SOFTWARE SOLUTIONS
Tools Material

Committing Changes - 3

When you choose the "SVN Commit" menu, a window pops up in which you can type a note that explains what you have changed. This note will become part of the SVN change log and, if the SVN is e-mail enabled, an automatic e-mail is generated to all authorized users alerting them of your change.

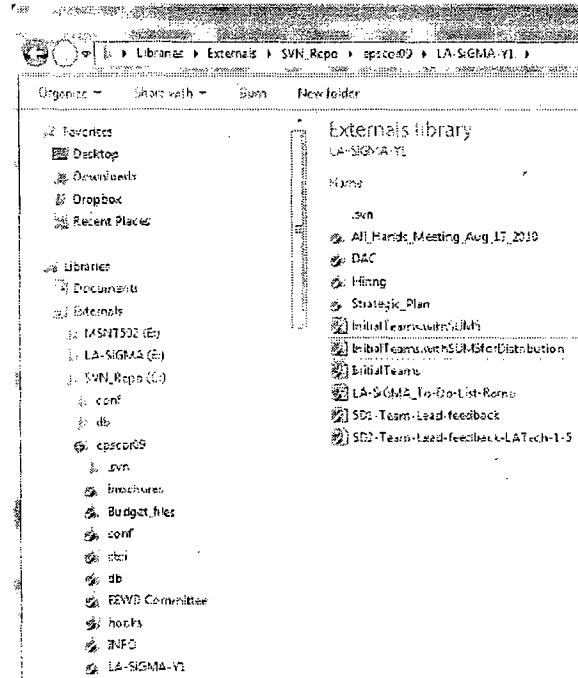


DURGA SOFTWARE SOLUTIONS
Tools Material

One more slide on “commit”

If all goes well with the “commit,” when I refresh my directory, the red exclamation mark will change to a green check mark, telling me that I have achieved a harmonious union with the great mother SVN.

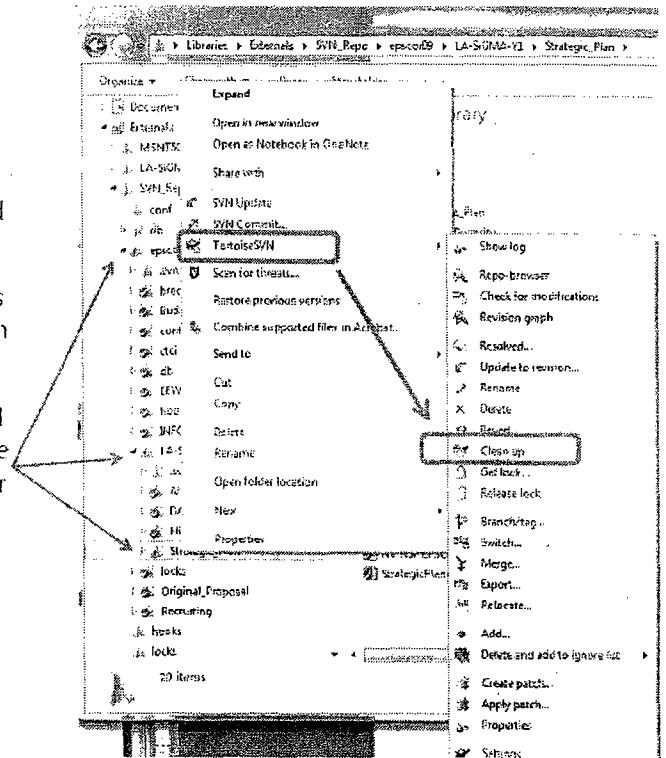
You may note that there is still a red exclamation mark on the high level directory “epscor09.” This never goes away on my desktop machine. I think it is an artifact of my 64-bit client not interacting with the central SVN. I run the 32-bit client application on my laptop and it does not have this problem.



DURGA SOFTWARE SOLUTIONS
Tools Material

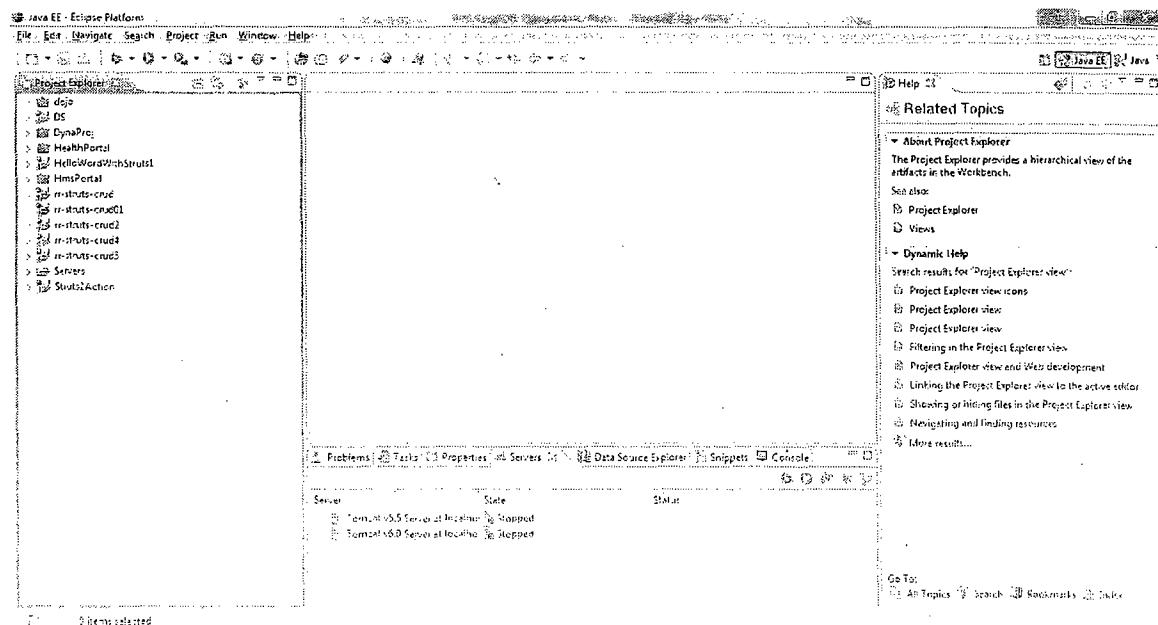
Clean up

- Sometimes, things go wrong, and conflicts arise.
- One way to resolve the conflict is to use the “Clean up” menu from “TortoiseSVN.”
- If that does not work, you should copy the conflicted files (note the yellow triangle marks) to another directory, delete them from the local SVN Repo, and do an “Update.”



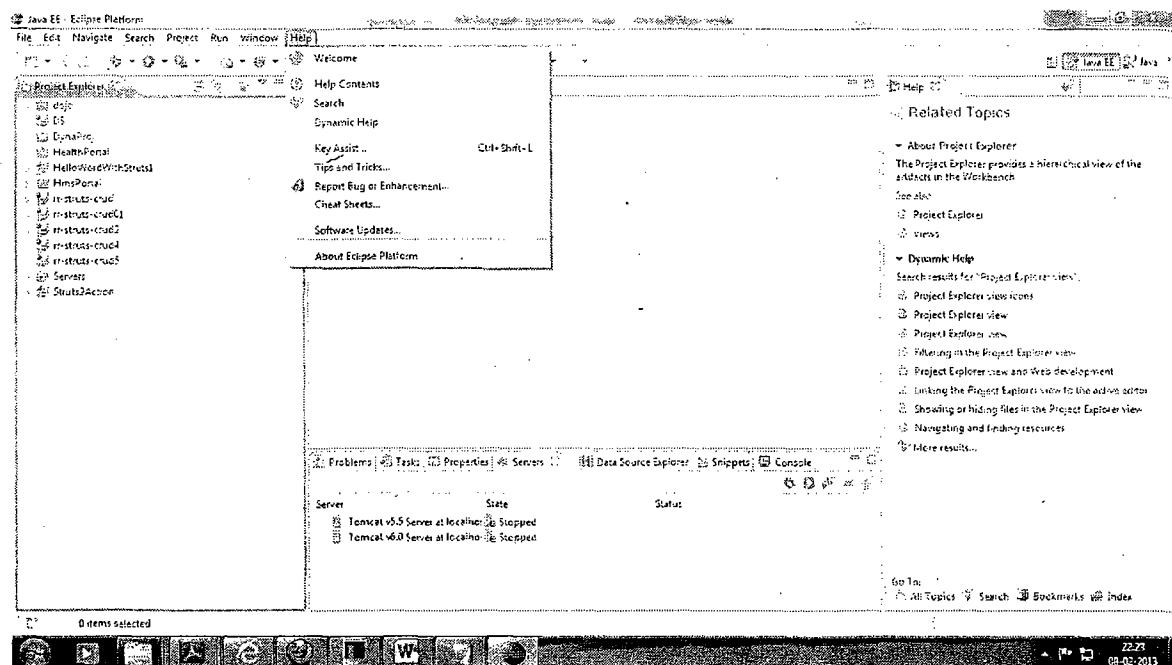
Procedure to configure SVN Plugin into Eclipse IDE :

Step-1 : Launch Eclipse having new work space for programmer.



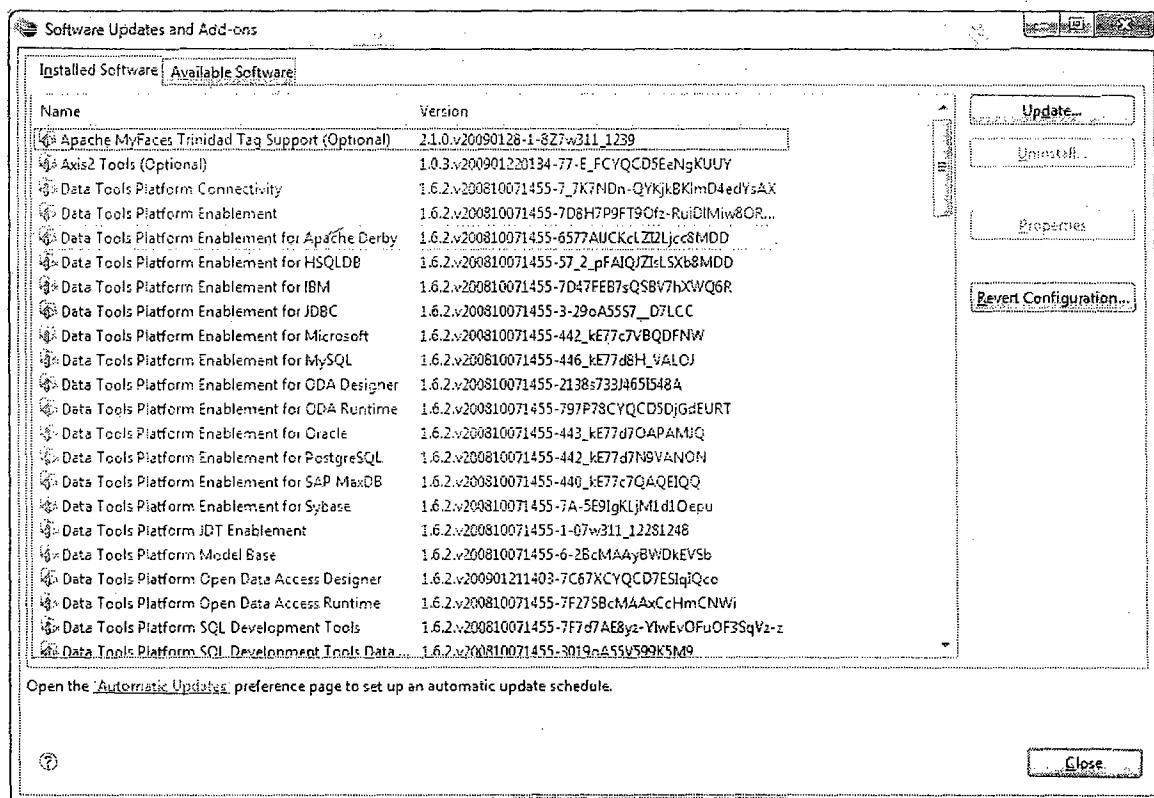
DURGA SOFTWARE SOLUTIONS Tools Material

Step-2 : Click on Software Updates under Help



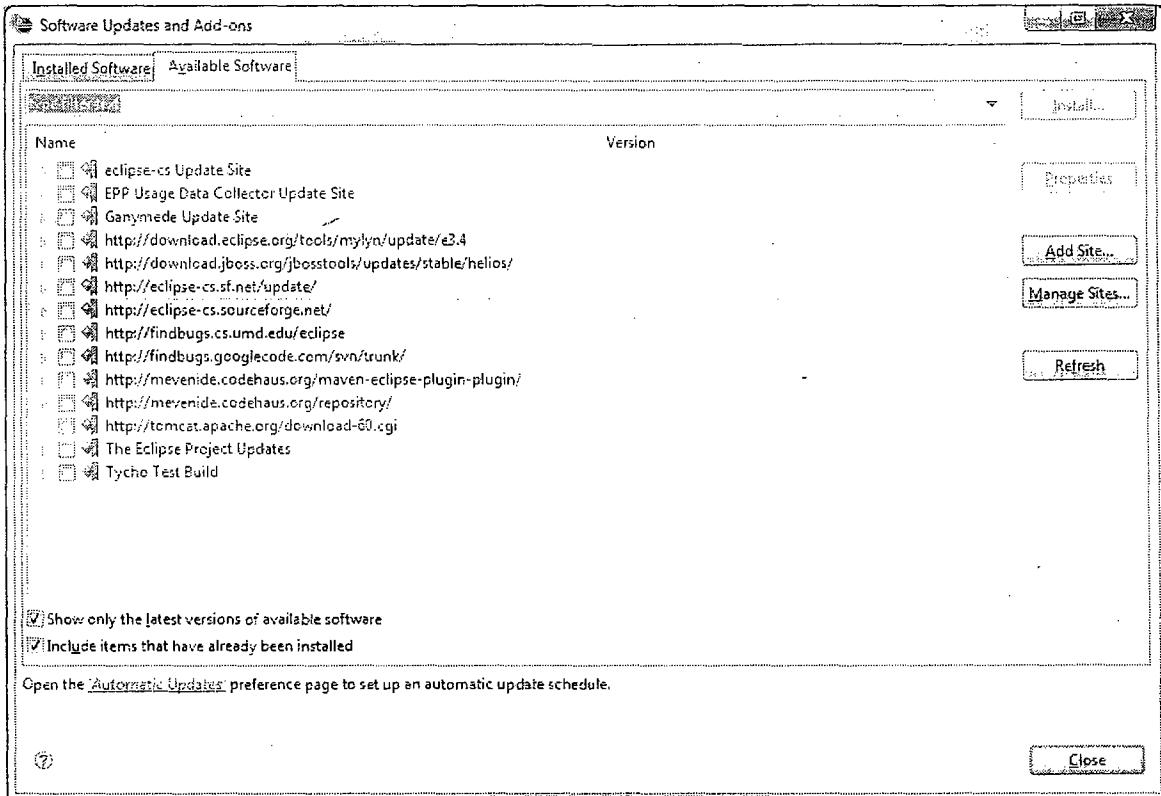
Step-3:

DURGA SOFTWARE SOLUTIONS Tools Material



Step-4:

DURGA SOFTWARE SOLUTIONS Tools Material

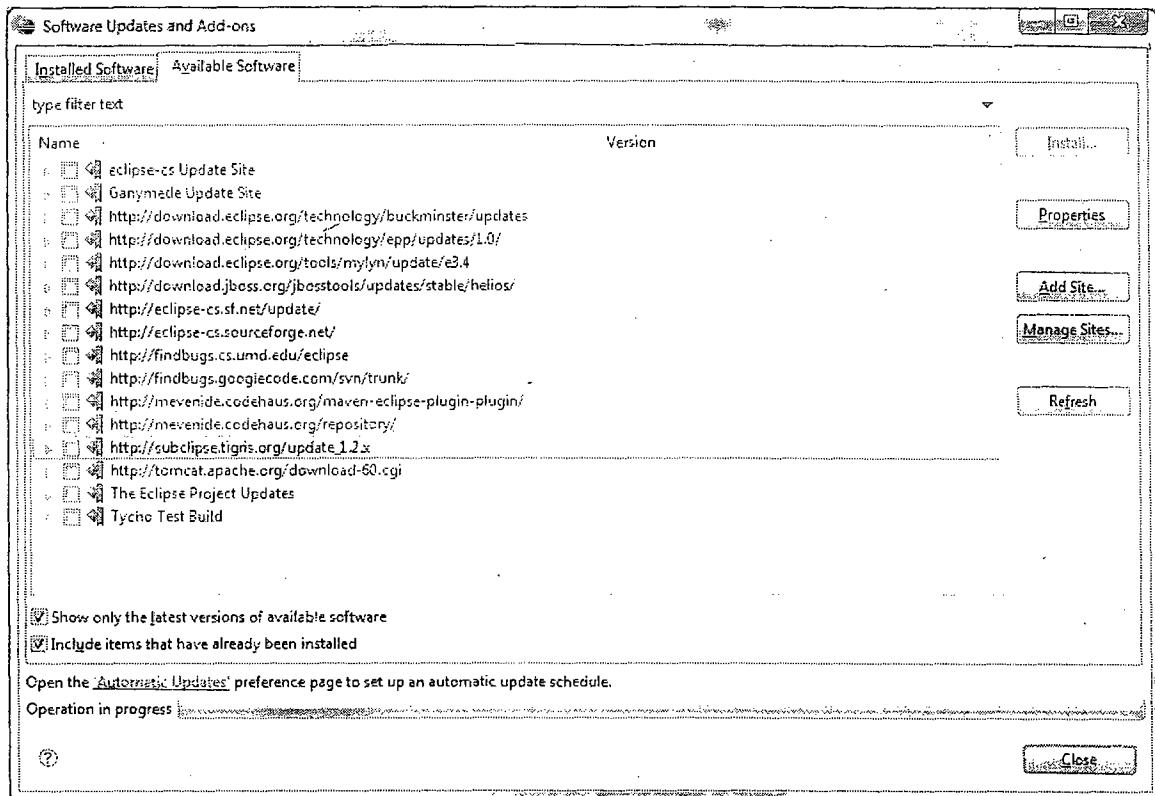


Step-5:

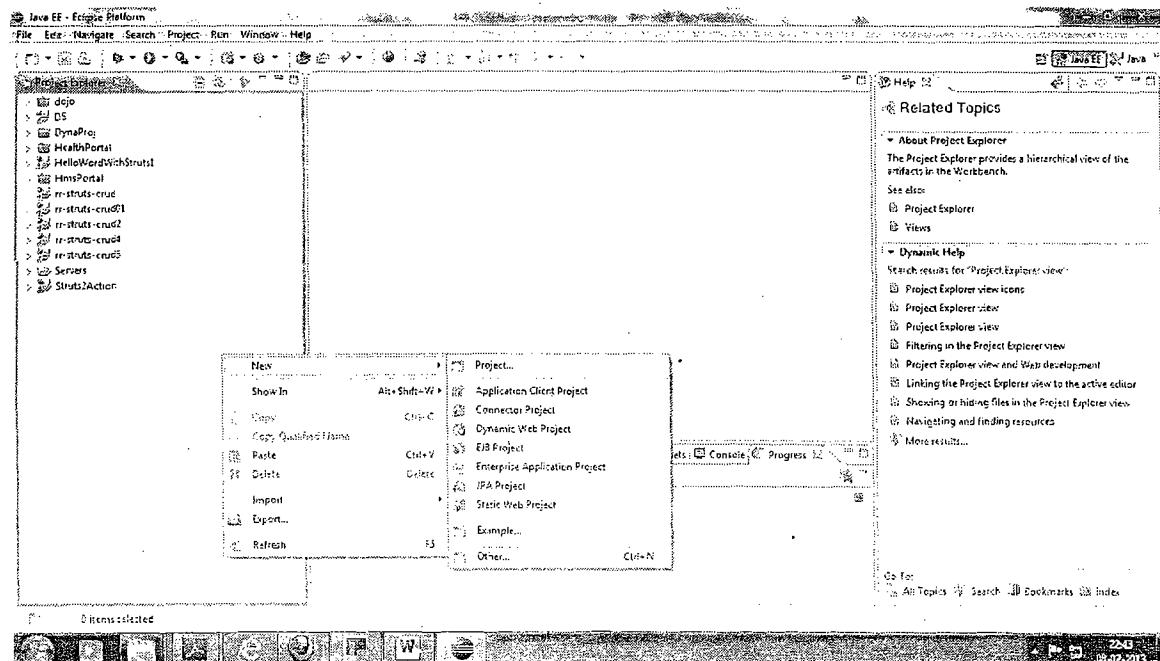
Click on Add Site and enter http://subclipse.tigris.org/update_1.2.x url

and click on install. After installing the SVN and restart the Eclipse.

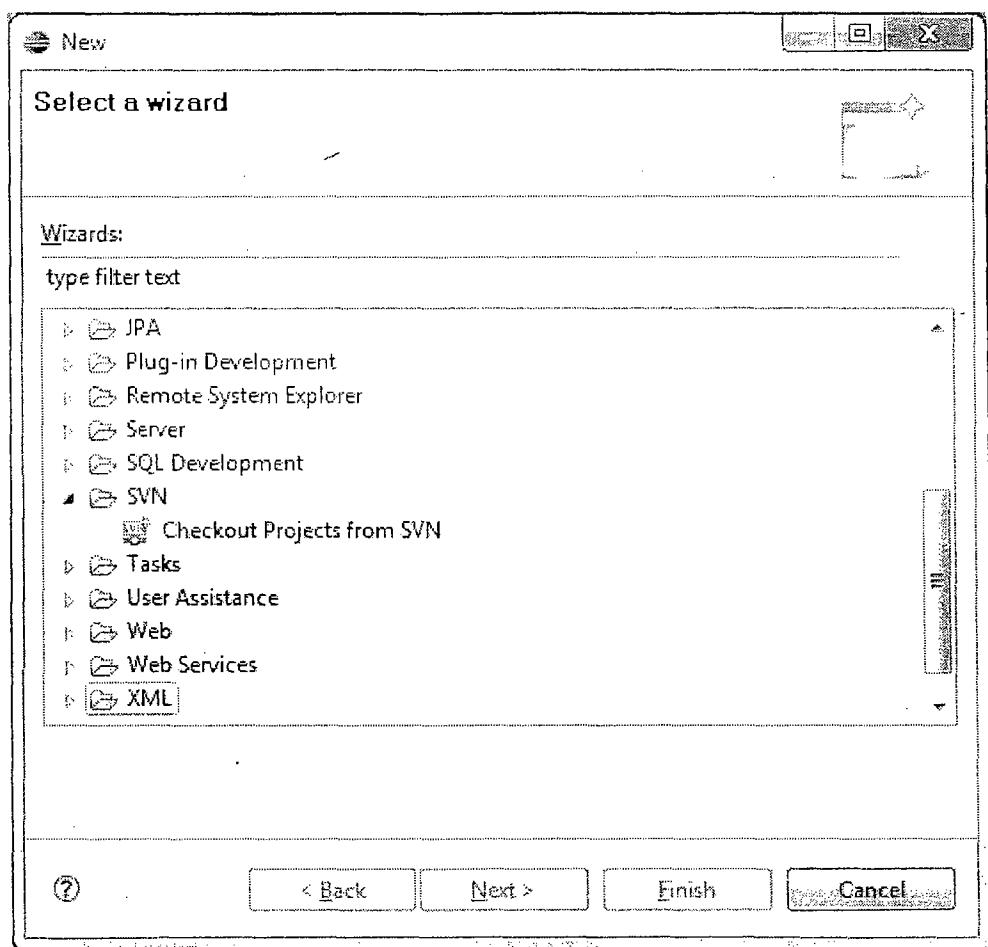
DURGA SOFTWARE SOLUTIONS Tools Material



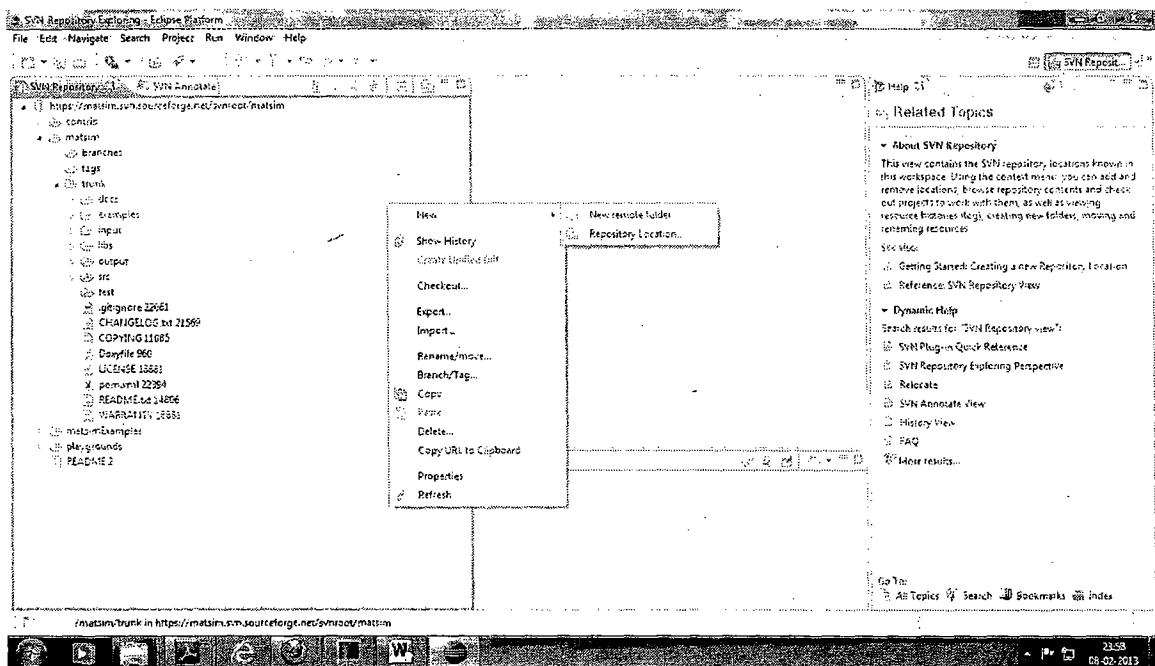
Step-6: To confirm SVN is configured



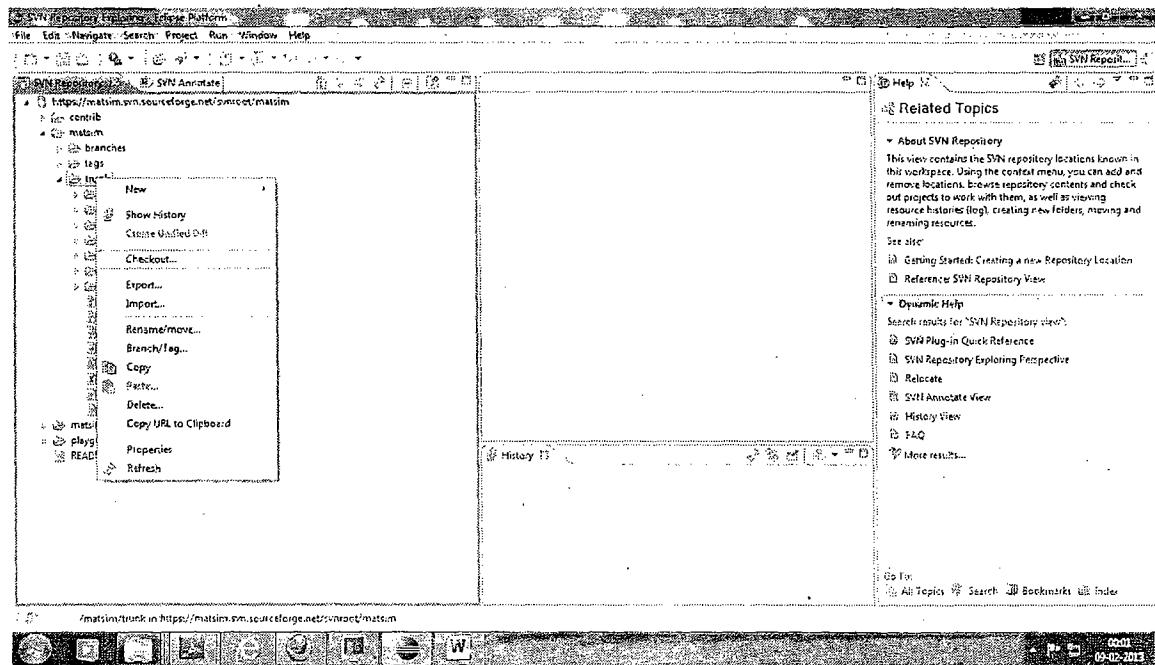
DURGA SOFTWARE SOLUTIONS
Tools Material



DURGA SOFTWARE SOLUTIONS Tools Material



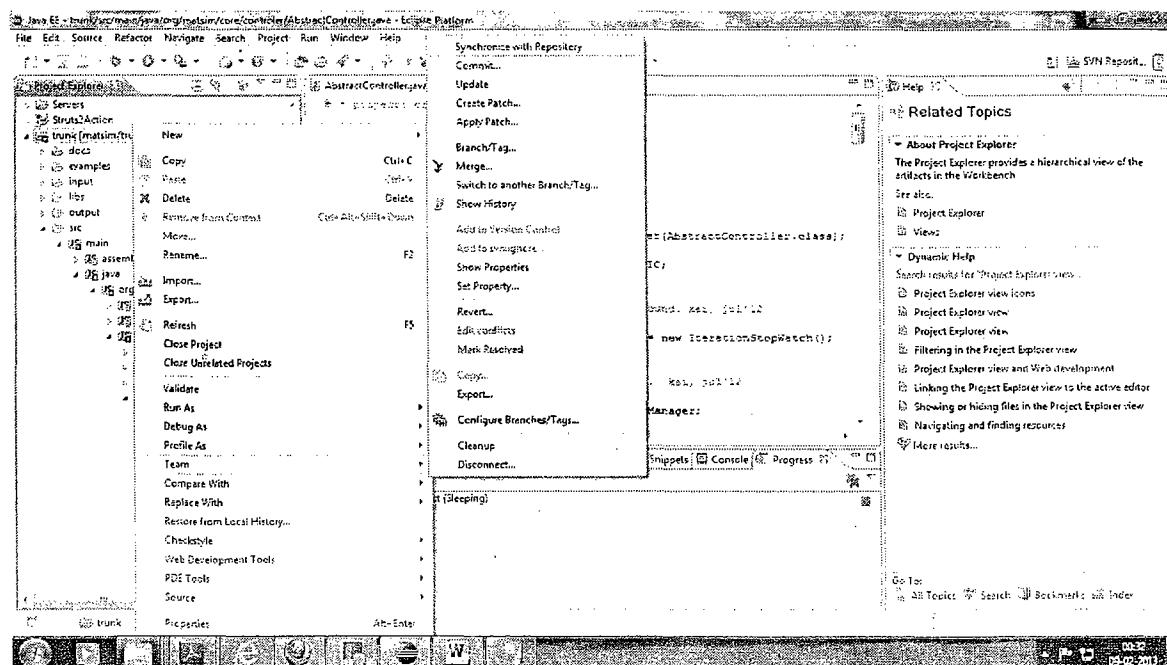
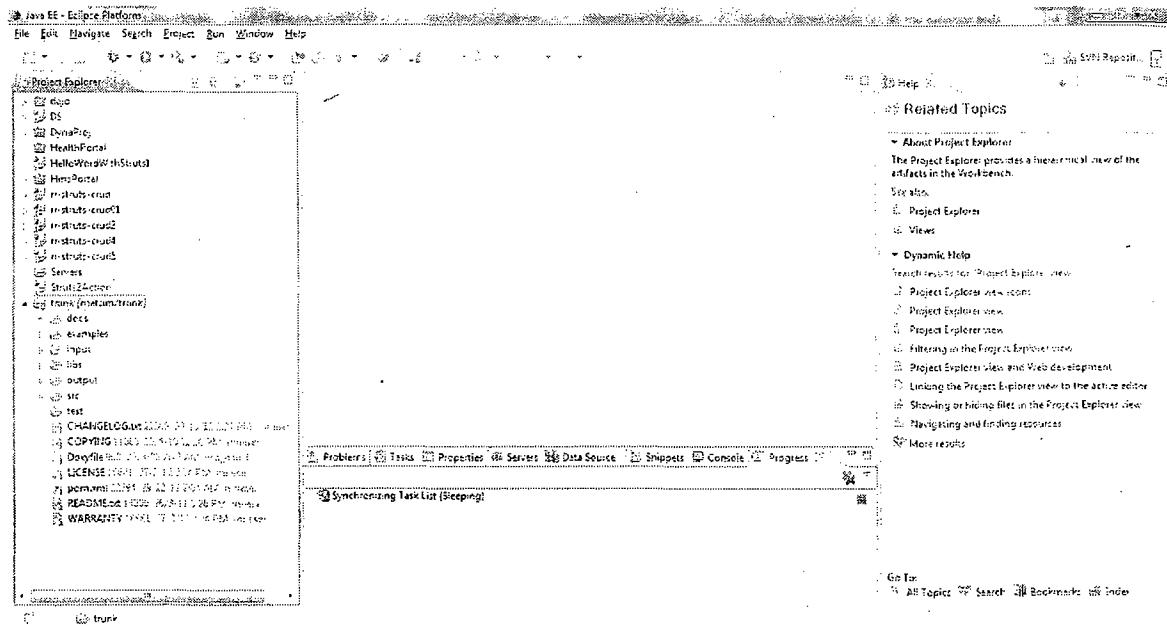
Check out : Check out code from Repository.



DURGA SOFTWARE SOLUTIONS

Tools Material

Checkout code from Repository

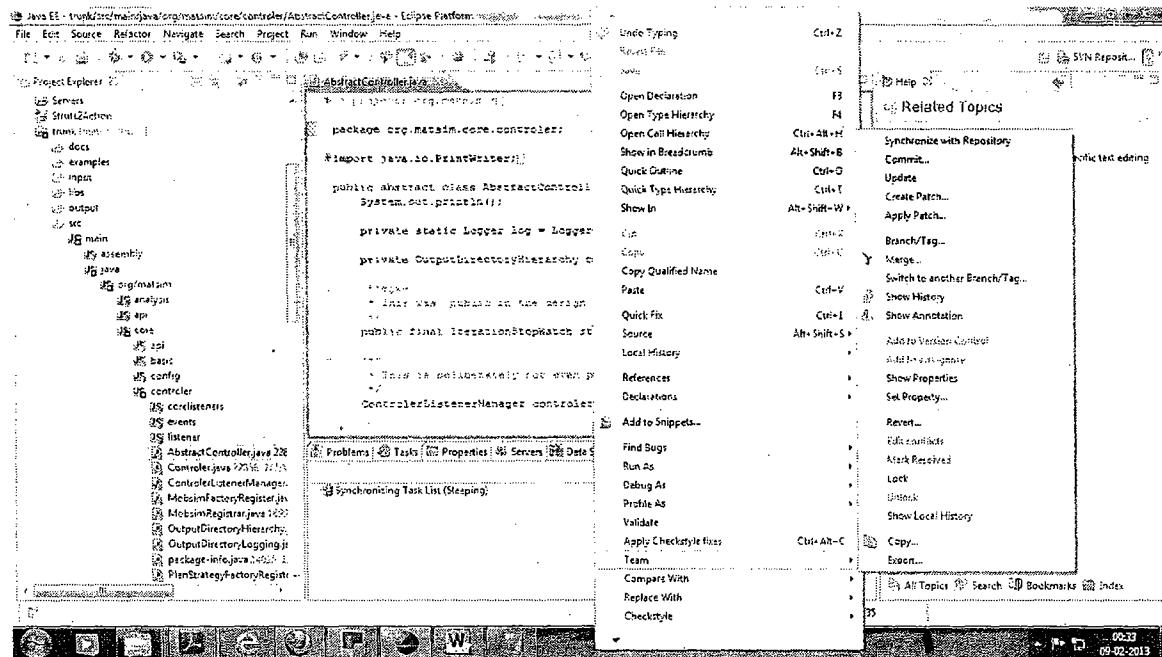


DURGA SOFTWARE SOLUTIONS

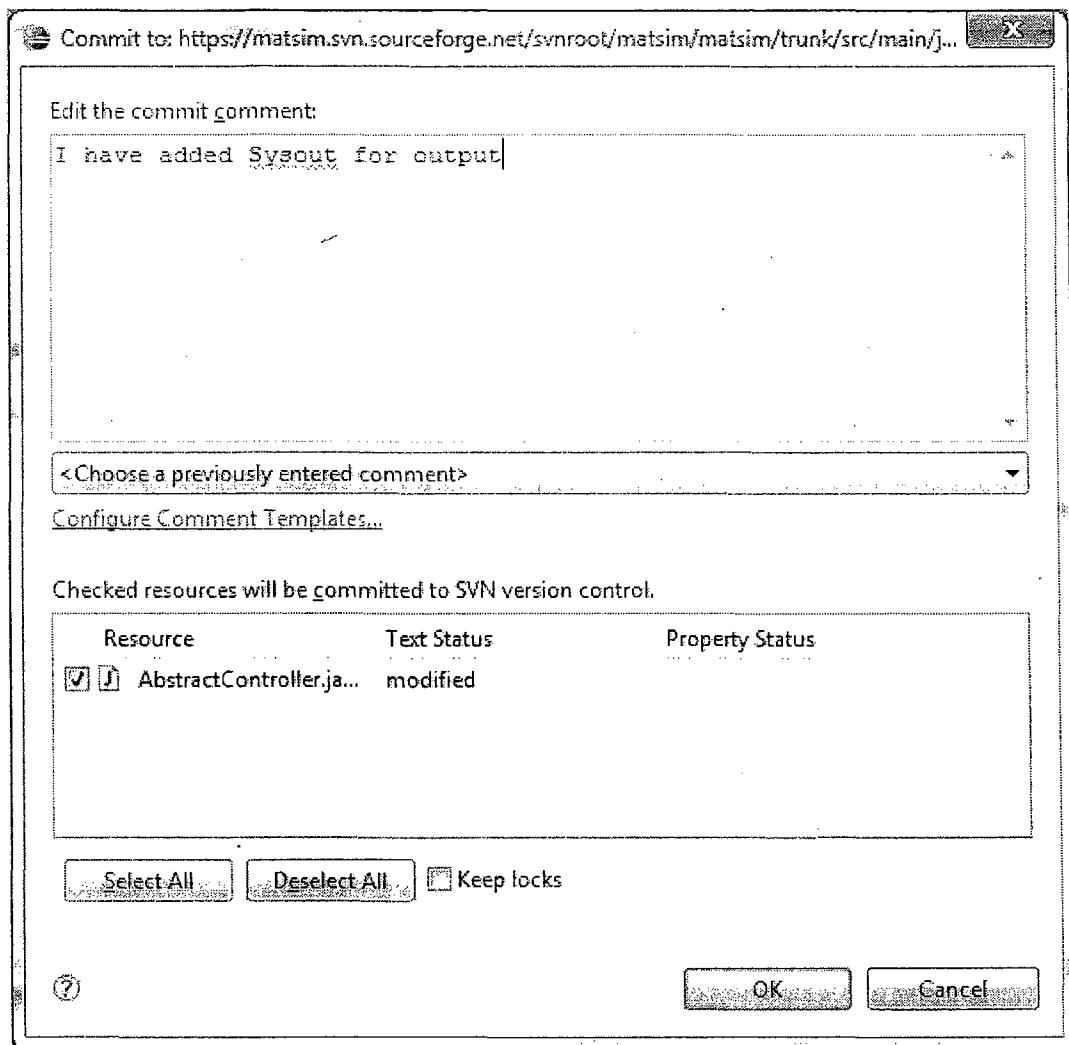
Tools Material

Check In:

Right click on AbstractController.java → team → commit (checkin) → enter some comment → finish.



DURGA SOFTWARE SOLUTIONS
Tools Material



We have to write the comments for which purpose file is modified.

Click on OK. We have to provide user name and password then the latest changes will be updated into the repository.

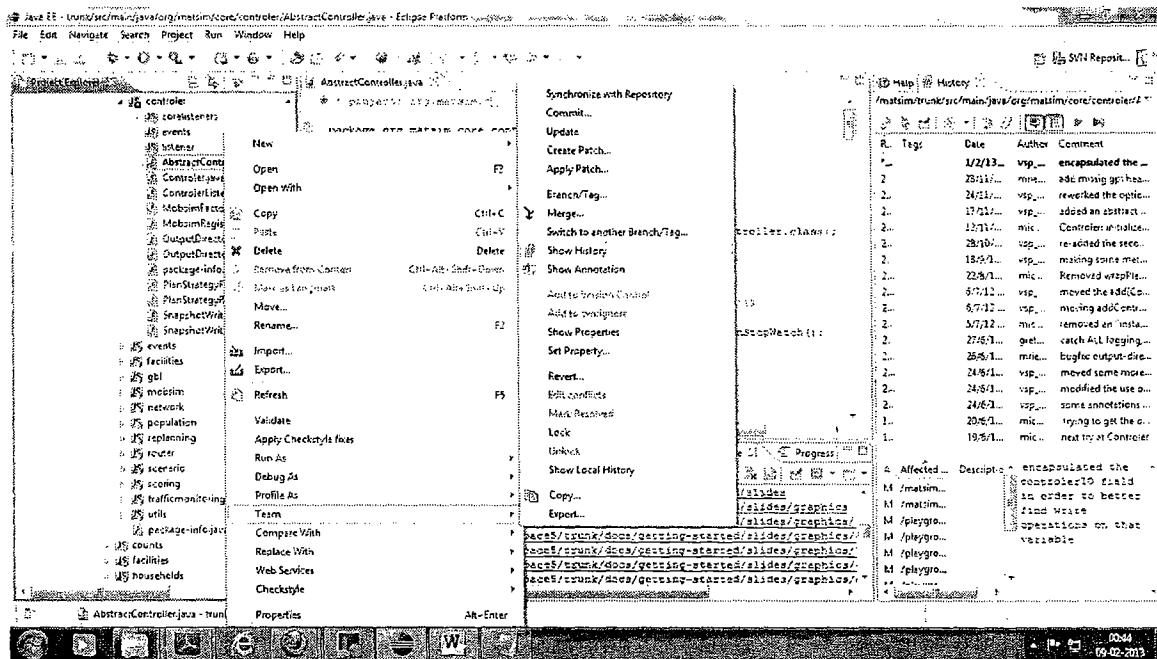
DURGA SOFTWARE SOLUTIONS

Tools Material

- ❖ A new version will create for file when you perform commit operation.

History : To replace current file content with one of the existing old version of the cvs repository.

- ❖ Click on .java file → team → show history → Go to history window → double click on version what ever you want.



DURGA SOFTWARE SOLUTIONS

Tools Material

Java EE - trunk/java/main/java/org/matsim/core/controllers/AbstractController.java - Eclipse Platform

File Edit Navigate Search Project Run Windows Help

SVN Report

Revision Date Author Comment

21845	10/11/12 10:20 PM	vsp_nigel	added an abstract method <code>getControllerId()</code> in AbstractController. Fixed the resulting compile errors. Implemented a <code>getTerminalIdAndConfig()</code> for the standard controller.
22095	10/11/12 10:30 PM	michaelz	add missing gpl header
22096	10/11/12 10:37 PM	michaelz	bugfix: output-directory must be initialized as early as possible, as some strategy modules (e.g. study) already try to write to it when being initialized.
22098	10/11/12 10:40 PM	gretchen	catch ALL logging, also system and build information
22174	10/11/12 10:56 PM	michaelz	Controller initializes all time resources, minStrategy, etc after ...
22289	10/12/12 12:42 AM	vsp_nigel	encapsulated the controllerID field in order to better find write operations on that variable
20902	10/9/12 9:51 PM	vsp_nigel	missing some methods of AbstractController were public
20934	10/9/12 9:57 PM	vsp_nigel	modified the use of config in AbstractController such that it is no longer a private field that may be shadowed in downstream classes
20935	10/9/12 9:58 PM	vsp_nigel	removed some more helper methods up into AbstractController
20936	10/9/12 10:39 PM	vsp_nigel	moved the addCoreControllerListener functionality to the AbstractController
20938	10/9/12 10:39 PM	vsp_nigel	moving addControllerListener from Controller to AbstractController
19972	10/6/12 8:01 PM	michaelz	test by a Controller
21524	10/10/12 11:51 PM	vsp_nigel	re-added the second config dump before iterations start. Has gotten lost somewhere ...
20154	5/7/12 7:07 PM	michaelz	removed an "instanceof" condition with unless purpose from VehicularSeparationHandler we now have people who want to to caravans with the transit module, so transit drivers may want them
20447	22/02/12 8:59 PM	michaelz	Removed wrapPlanAlgo from AbstractController
21564	24/11/12 4:43 PM	vsp_nigel	rewrote the section <code>core/MSATransitAdmission</code> a bit. Debugging this mess is necessary to make the config file dump follow the move of the config file reading into the constructor. Also made the some annotations that suppress compiler warnings at my setup
20033	10/6/12 9:39 PM	vsp_nigel	trying to get the out of memory exception on the clients to disappear again
19921	10/6/12 11:09 PM	michaelz	

A Affected ... Description

- M /mainline...
- M /mainline...
- I4 /playground...
- M /playground...
- M /playground...
- M /playground...
- M /playground...

* encapsulated the controllerID field in order to better find write operations on that variable

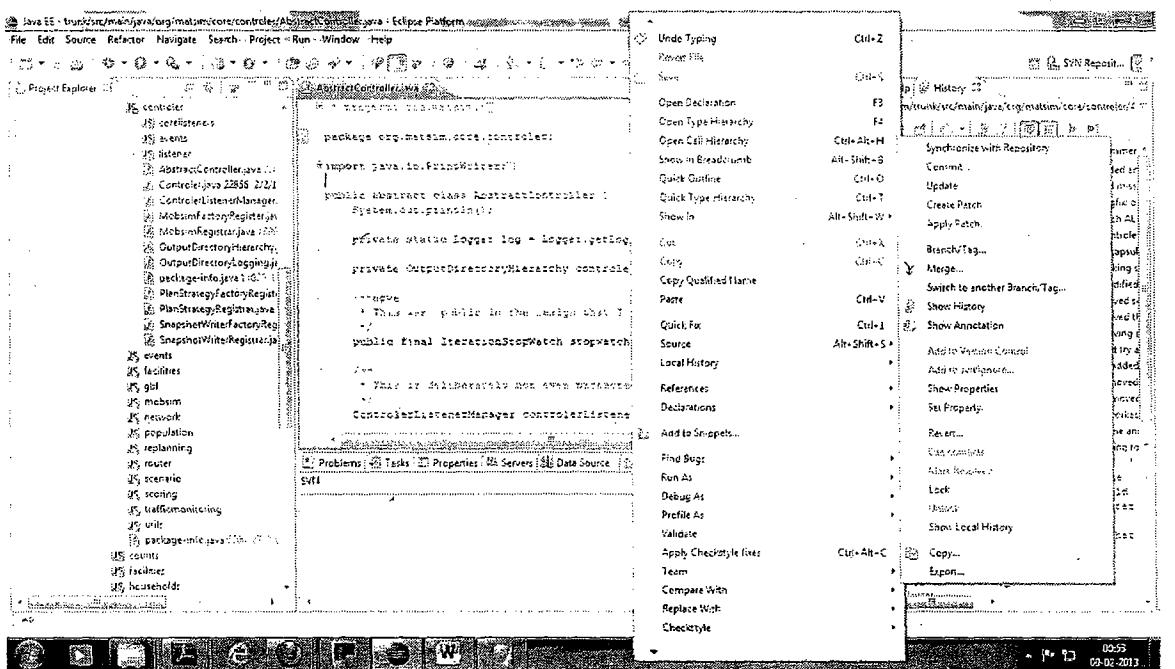
Update : This operation can be used for two purposes.

- To get the updated version of the source file.
- To update the content of current file in cvs repository.

" Right click on file → team → update. "

DURGA SOFTWARE SOLUTIONS

Tools Material



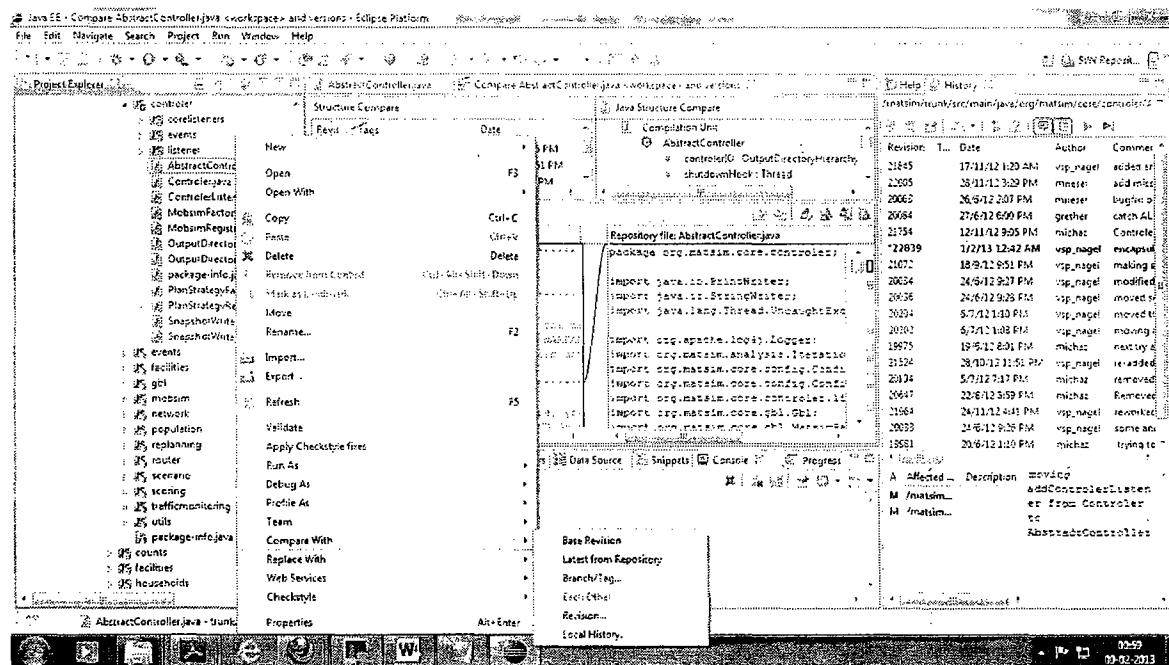
Compare : To compare code of current version with the different versions in repository.

DURGA SOFTWARE SOLUTIONS

Tools Material

“ Right click on source file → compare with → Revision → Go to Revision window

select a version for compare.”



Compare : To compare code of current version with the latest version in repository.

“Right click on source file → compare with → Latest From Repository” for compare.

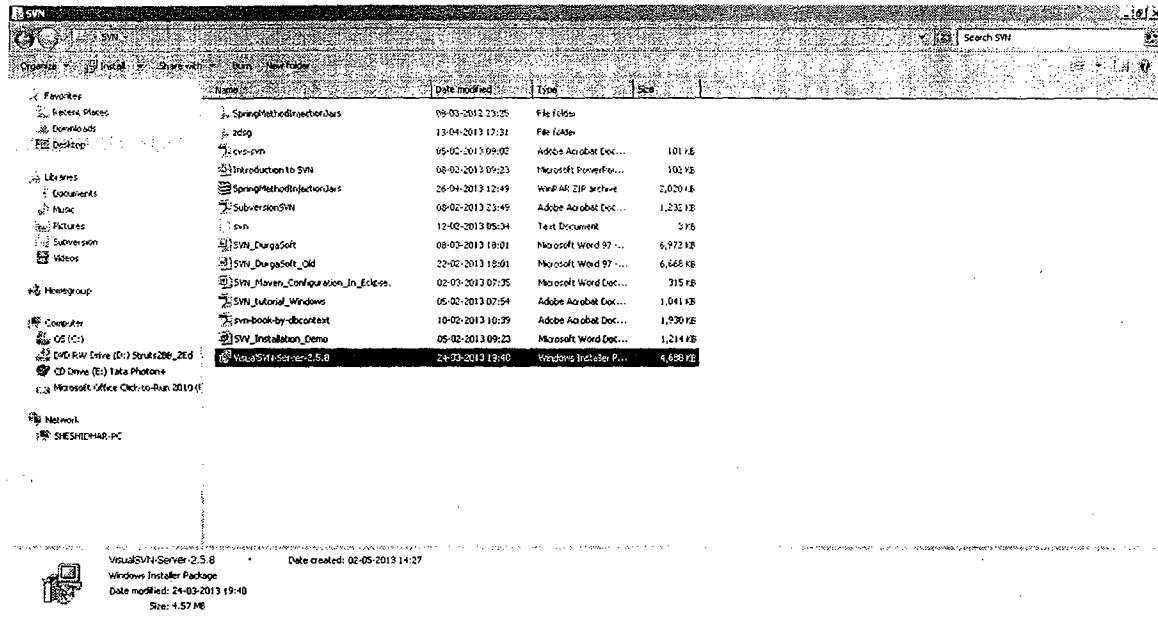
Compare : To compare code of current version with the Base version in repository.

“Right click on source file → compare with → Base Revision” for compare.

DURGA SOFTWARE SOLUTIONS
Tools Material

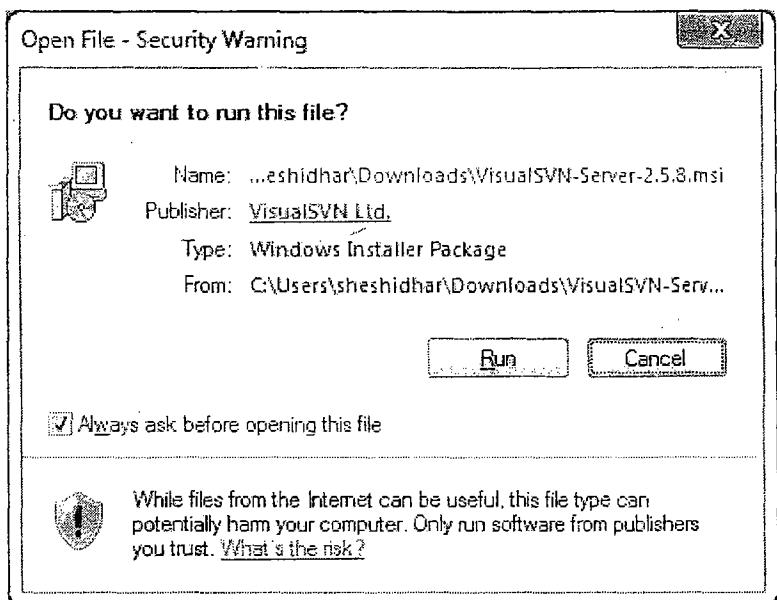
SVN Server Windows Installation:

Download VisualSVN Server from the below url:<http://www.visualsvn.com/server/download/>

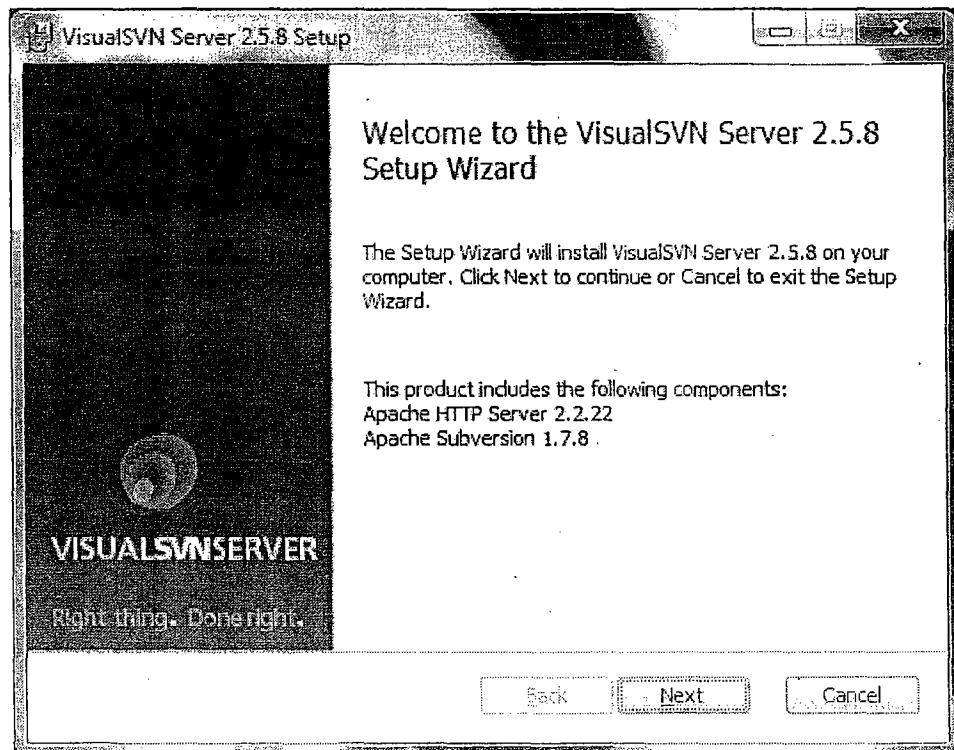


Right click on VisualServer.exe and Right click and click on Install

DURGA SOFTWARE SOLUTIONS
Tools Material

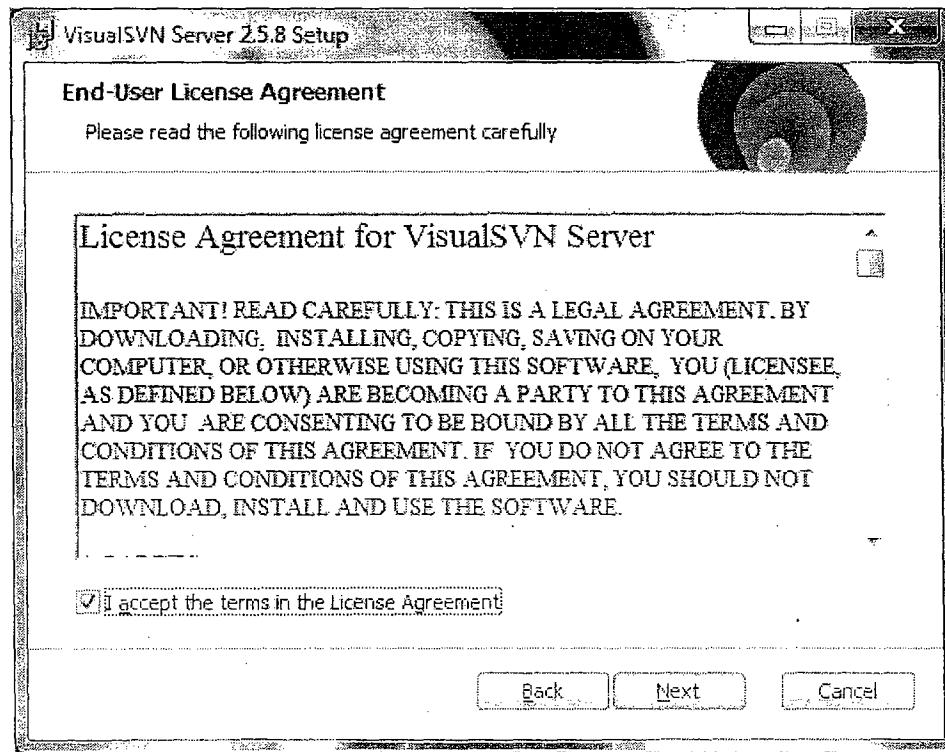
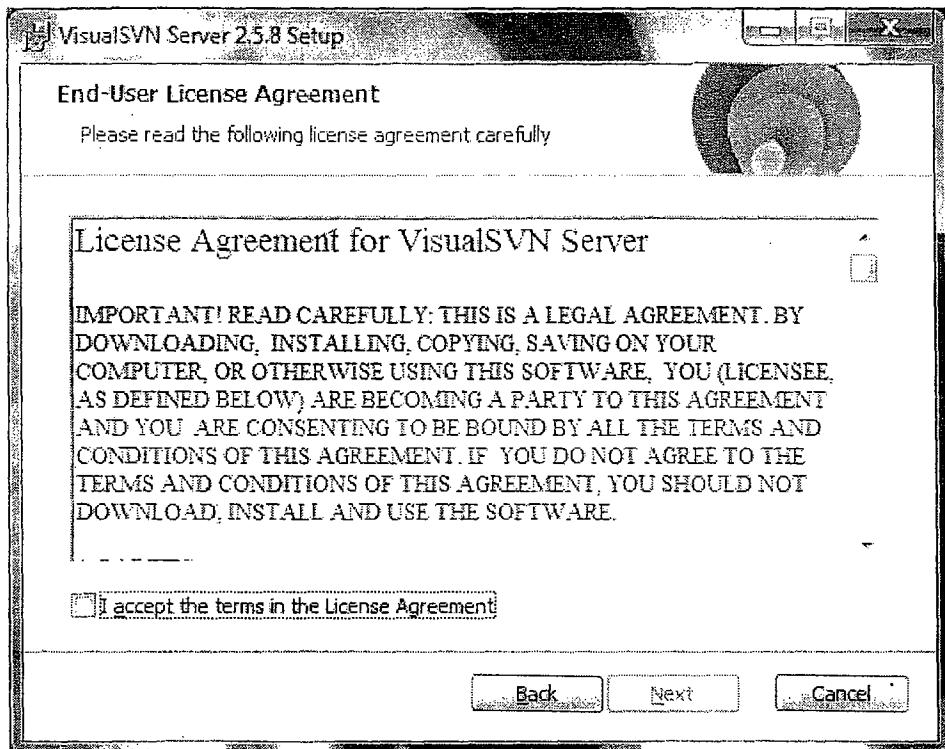


Click on Run



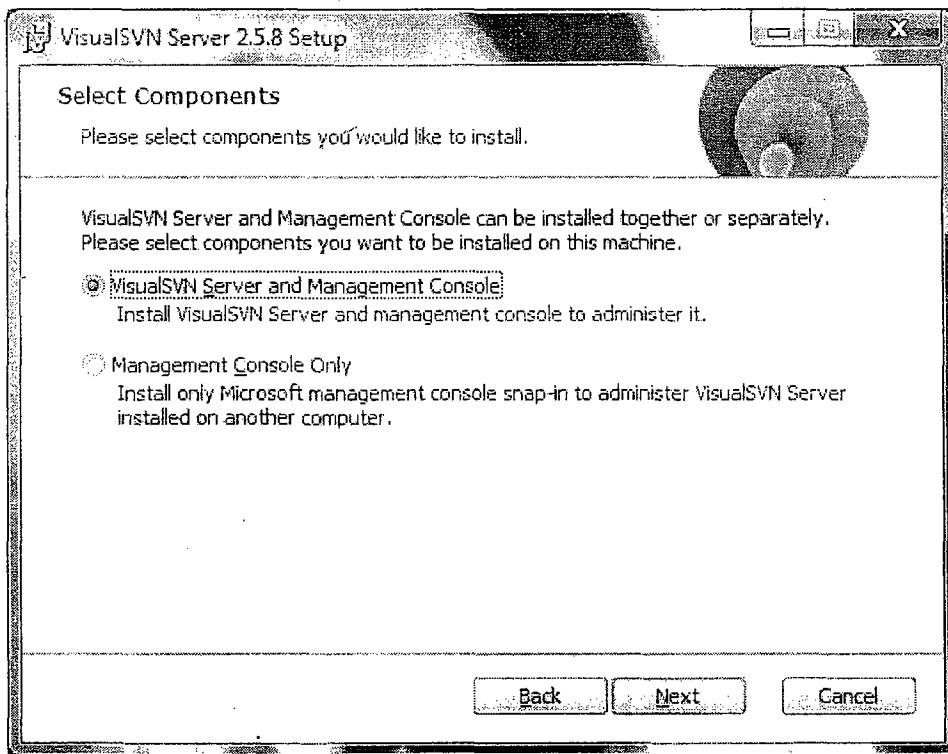
Click on Next

DURGA SOFTWARE SOLUTIONS
Tools Material



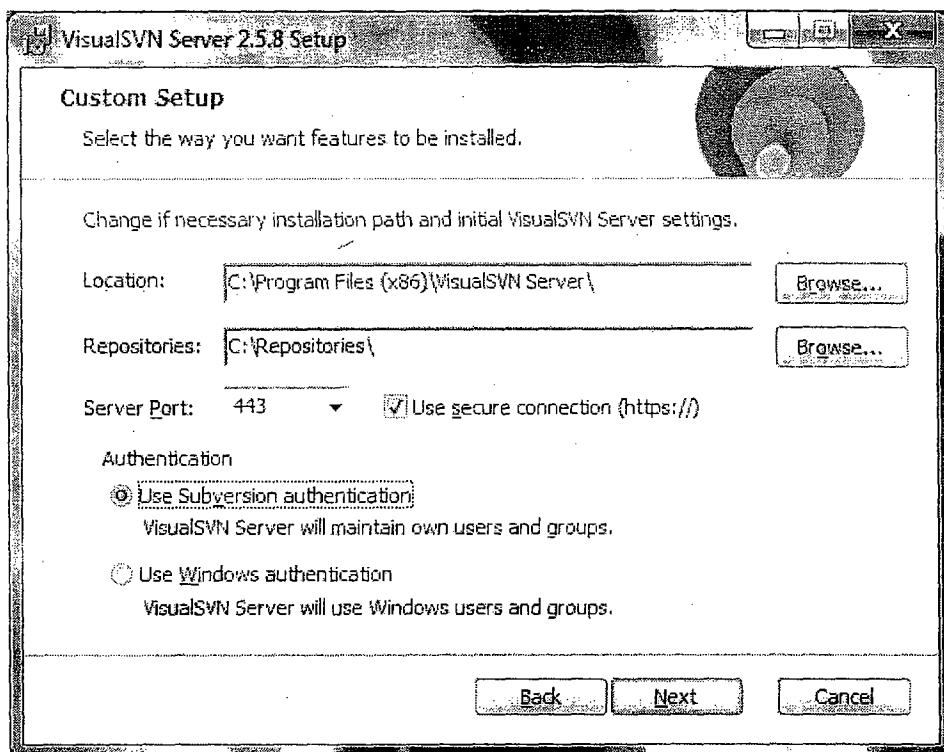
DURGA SOFTWARE SOLUTIONS
Tools Material

Select accept the license Agreement and click Next



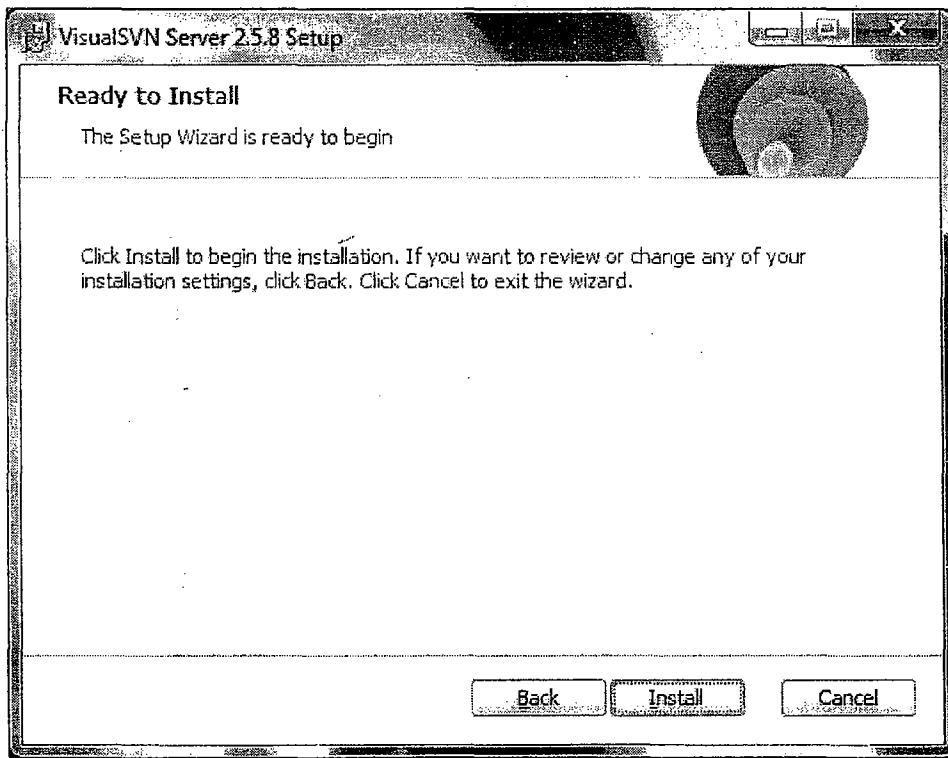
Click Next

DURGA SOFTWARE SOLUTIONS
Tools Material



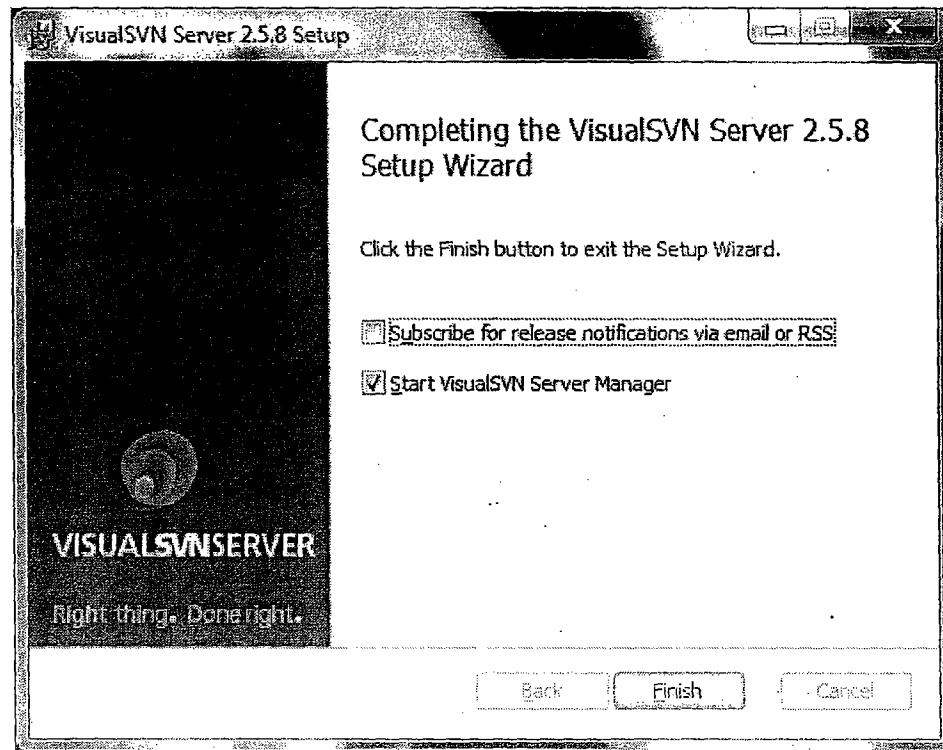
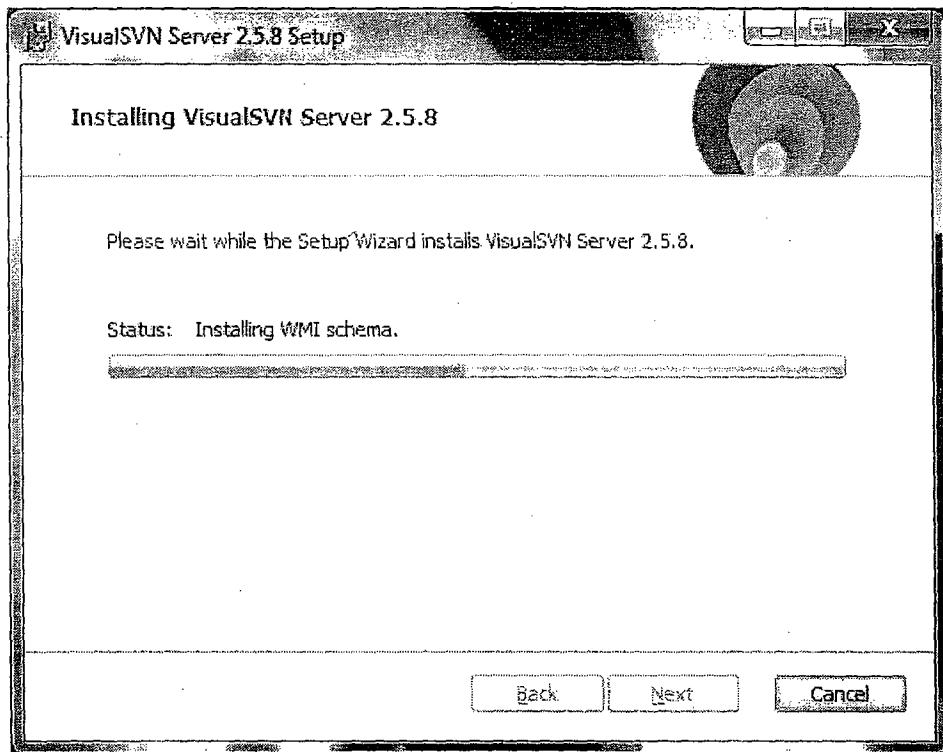
Click Next

DURGA SOFTWARE SOLUTIONS
Tools Material



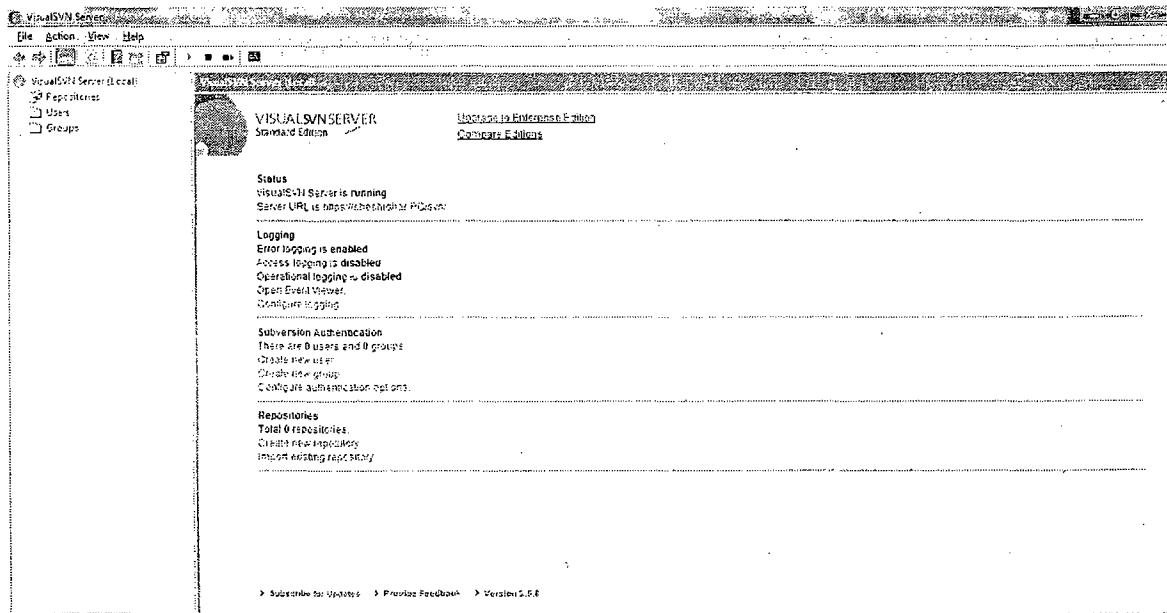
Click Install

DURGA SOFTWARE SOLUTIONS
Tools Material



DURGA SOFTWARE SOLUTIONS
Tools Material

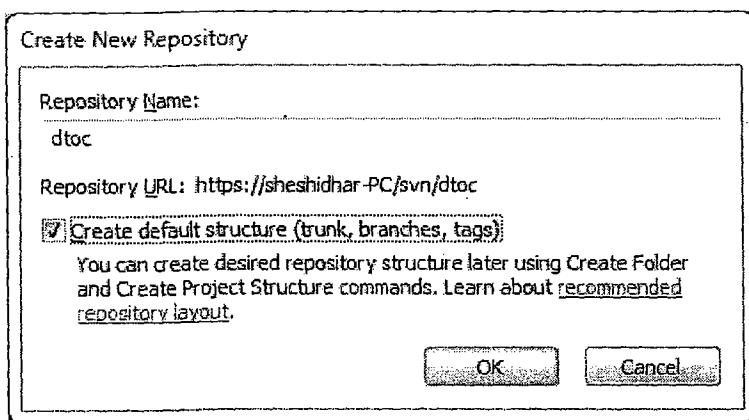
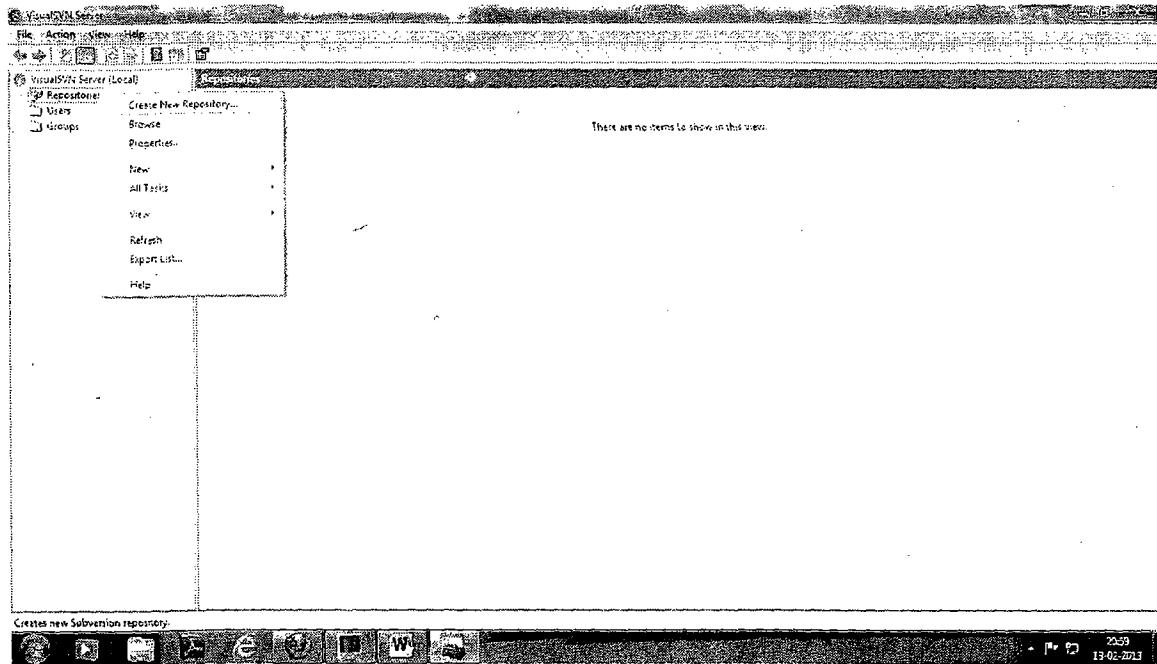
Click Finish



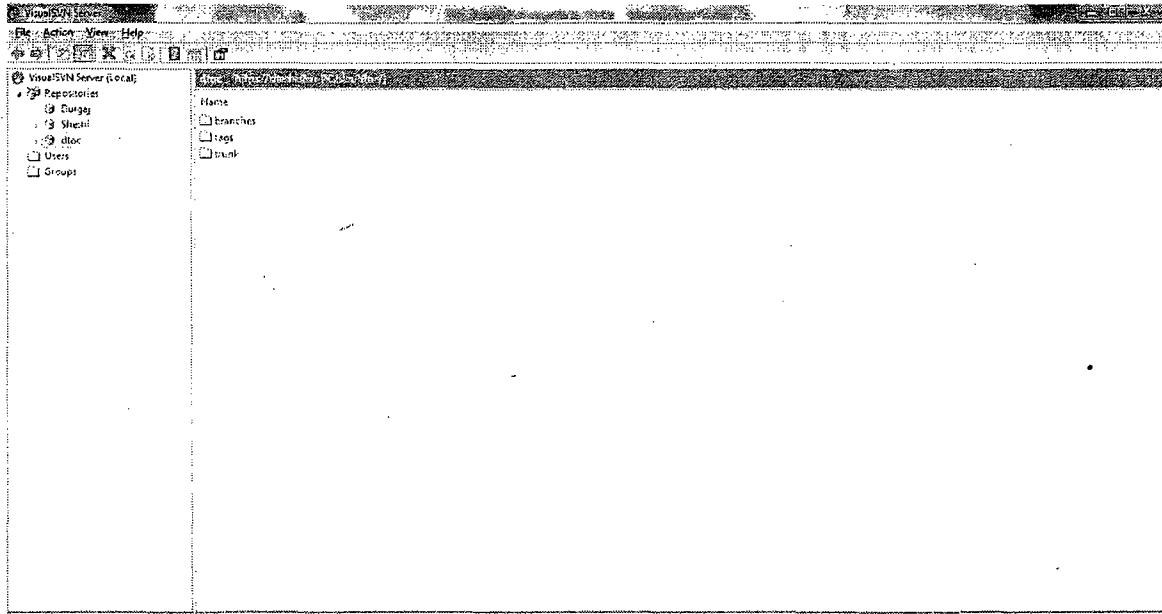
SVN Tortoise Server screen.

Steps to create Repository:

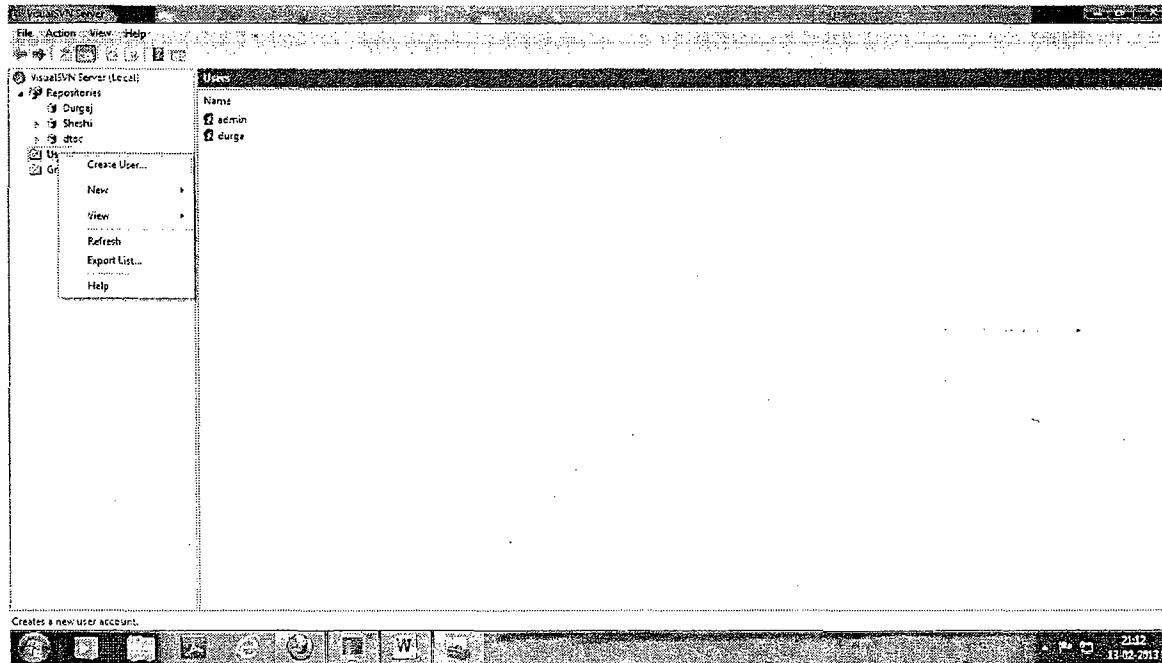
DURGA SOFTWARE SOLUTIONS
Tools Material



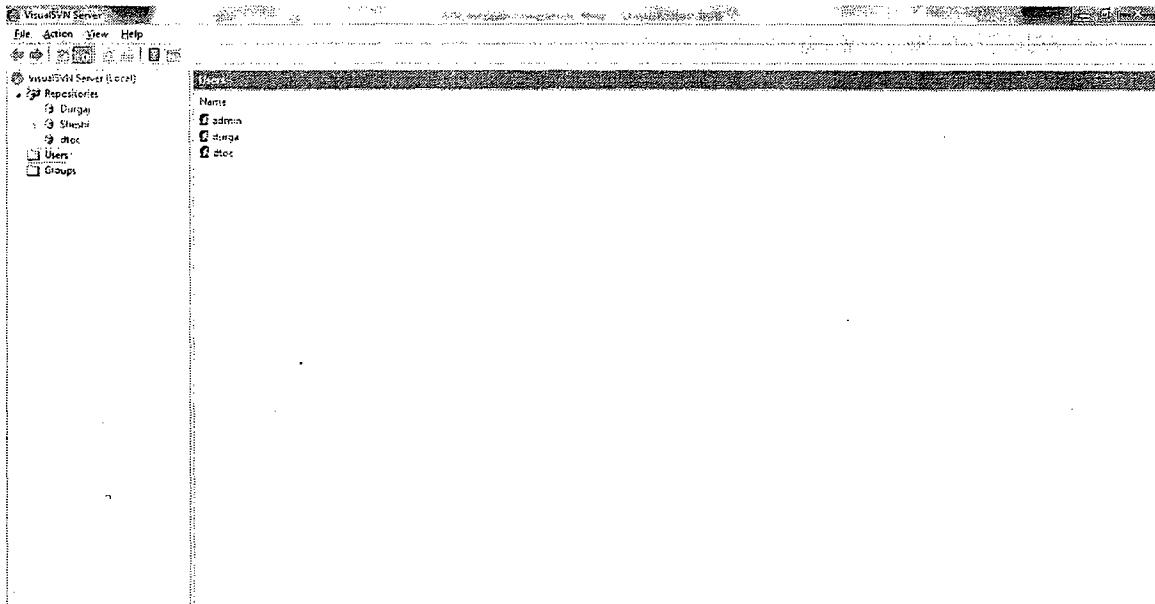
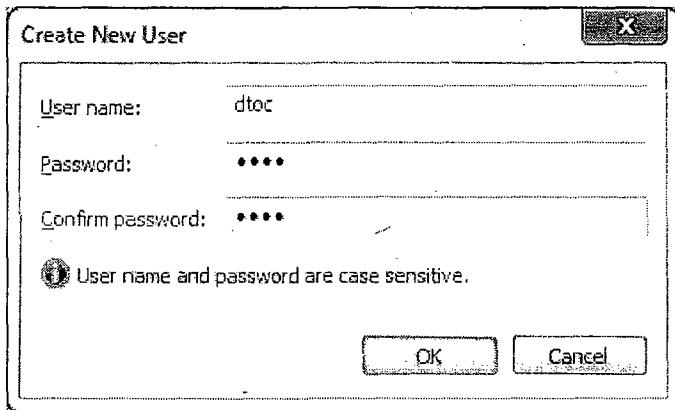
DURGA SOFTWARE SOLUTIONS Tools Material



Steps to Create User:

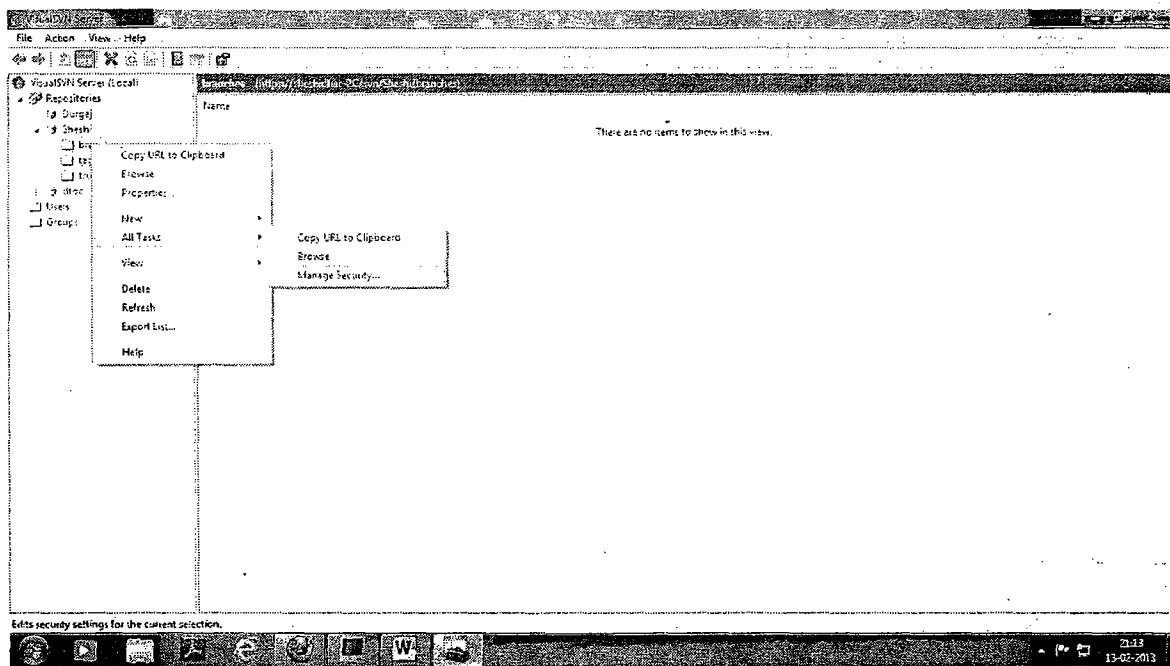


DURGA SOFTWARE SOLUTIONS
Tools Material



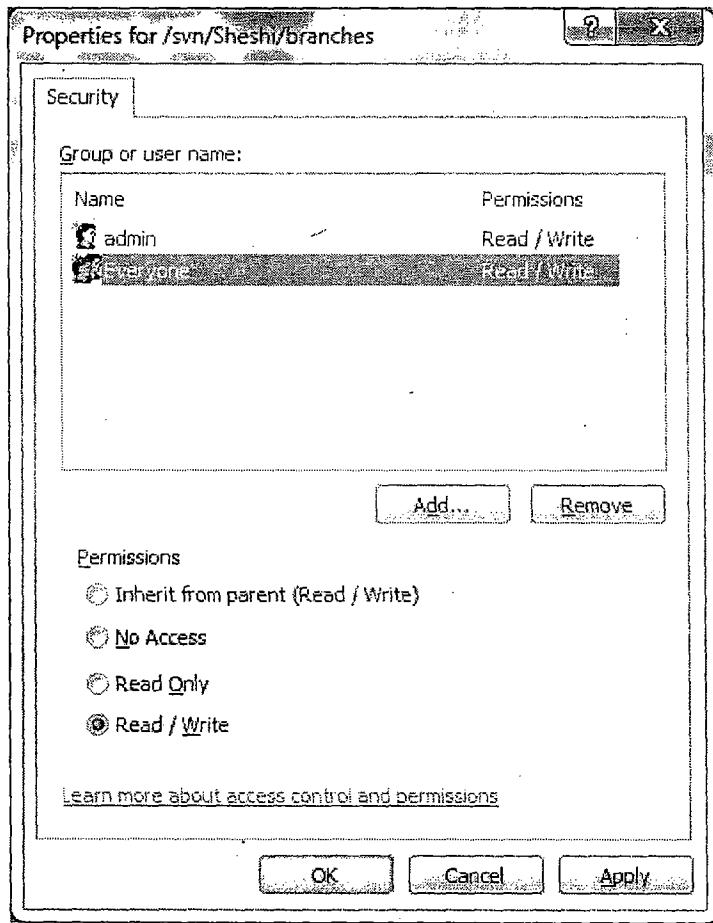
DURGA SOFTWARE SOLUTIONS
Tools Material

Steps: Adding User read/write permissions to access Branch/Trunk



Repository url: <https://localhost/svn/Sheshi/branches/>

DURGA SOFTWARE SOLUTIONS
Tools Material



SVN Plugin Configuration in Eclipse:

In Eclipse, select the menu "Help > Install New Software".

Enter the Update Site, depending on your version of subversion you installed:

If you installed Subversion 1.6: http://subclipse.tigris.org/update_1.6.x

If you installed Subversion 1.7: http://subclipse.tigris.org/update_1.8.x

Select the following items to be installed:

- Subclipse
- Subversion Client Adapter
- Subversion JavaHL Native Library Adapter
- SVNKit Client Adapter

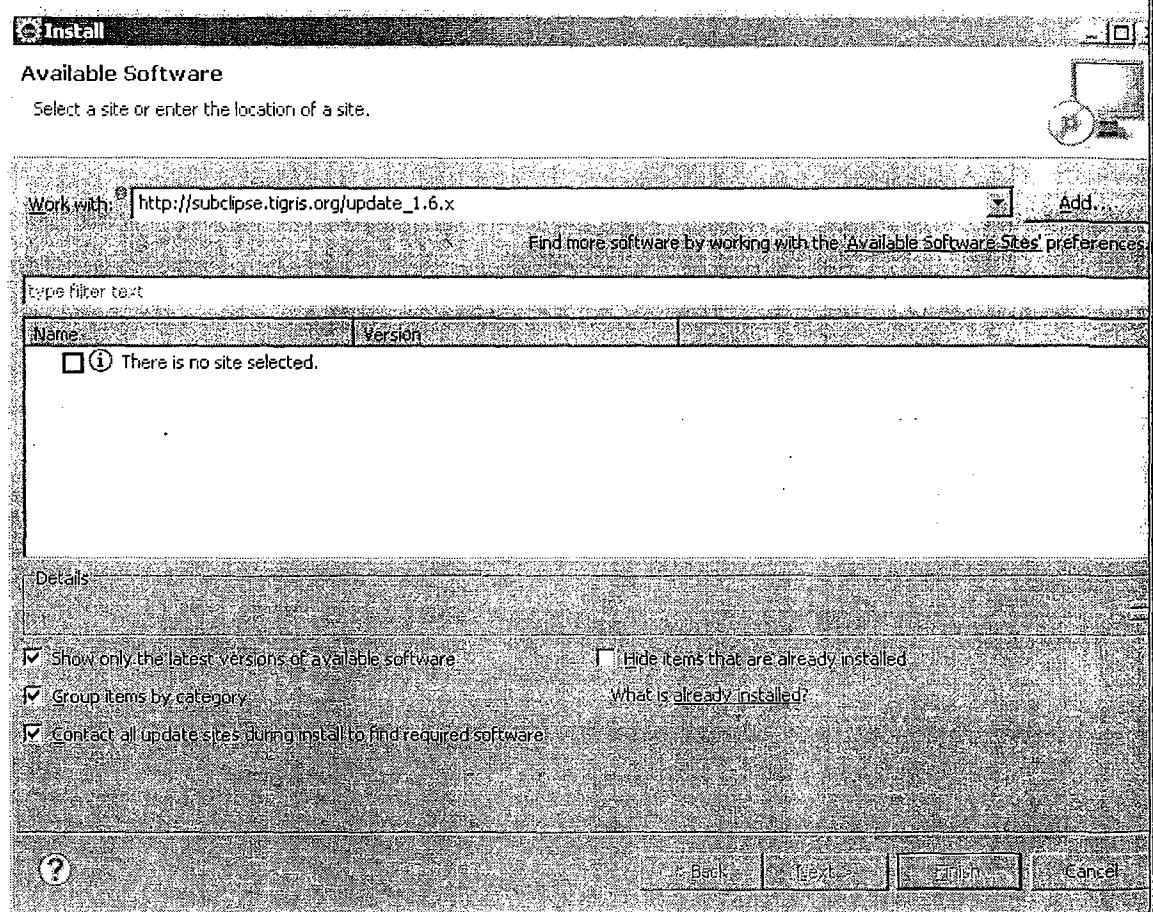
After the installation, restart Eclipse.

DURGA SOFTWARE SOLUTIONS

Tools Material

To verify that the integration works, try the following steps:

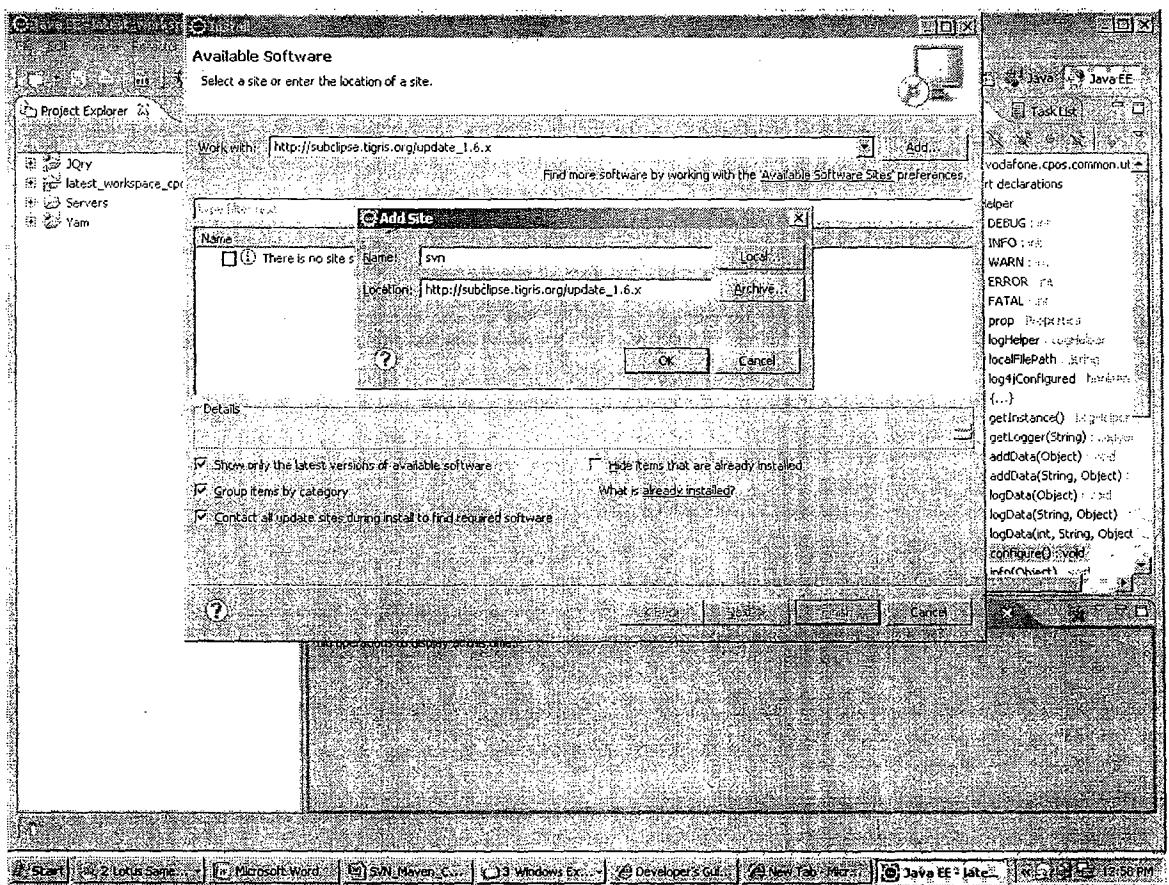
- Open the "SVN Repository Exploring" perspective (menu Window > Open Perspective > Other)
- Add the following SVN Repository Location:
<https://matsim.svn.sourceforge.net/svnroot/matsim>
To add it, click on the yellowish icon with "svn" and the plus sign.
- Test if you can browse to the sub-directories, e.g. matsim/trunk.



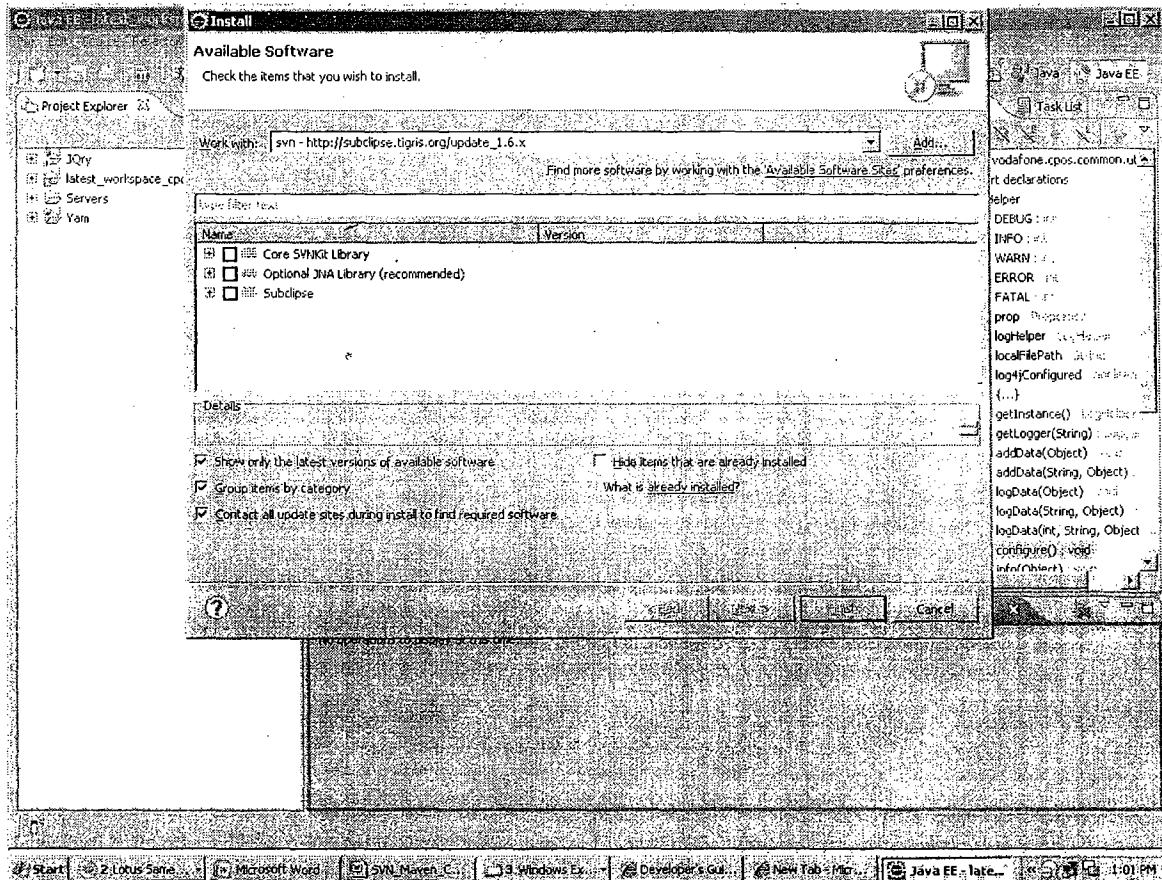
Click Add

DURGA SOFTWARE SOLUTIONS

Tools Material

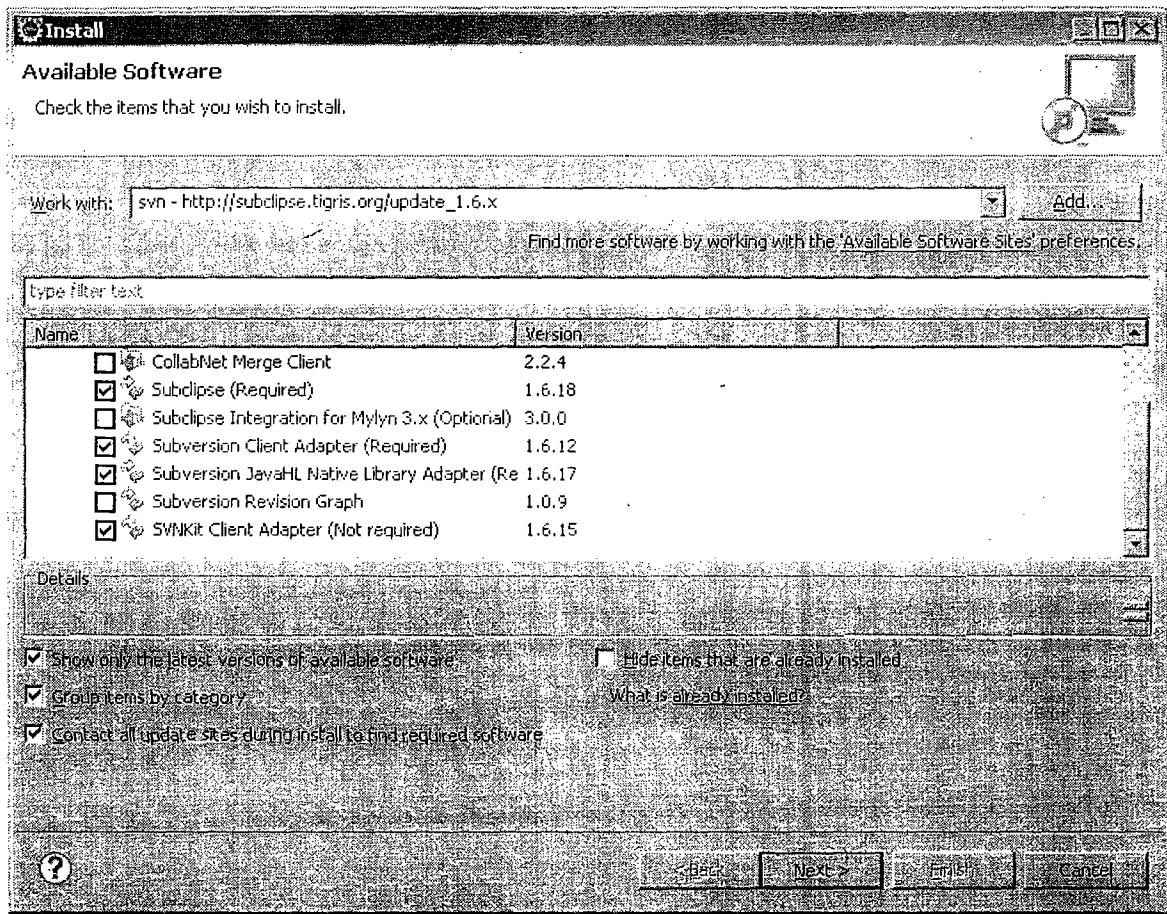


DURGA SOFTWARE SOLUTIONS Tools Material



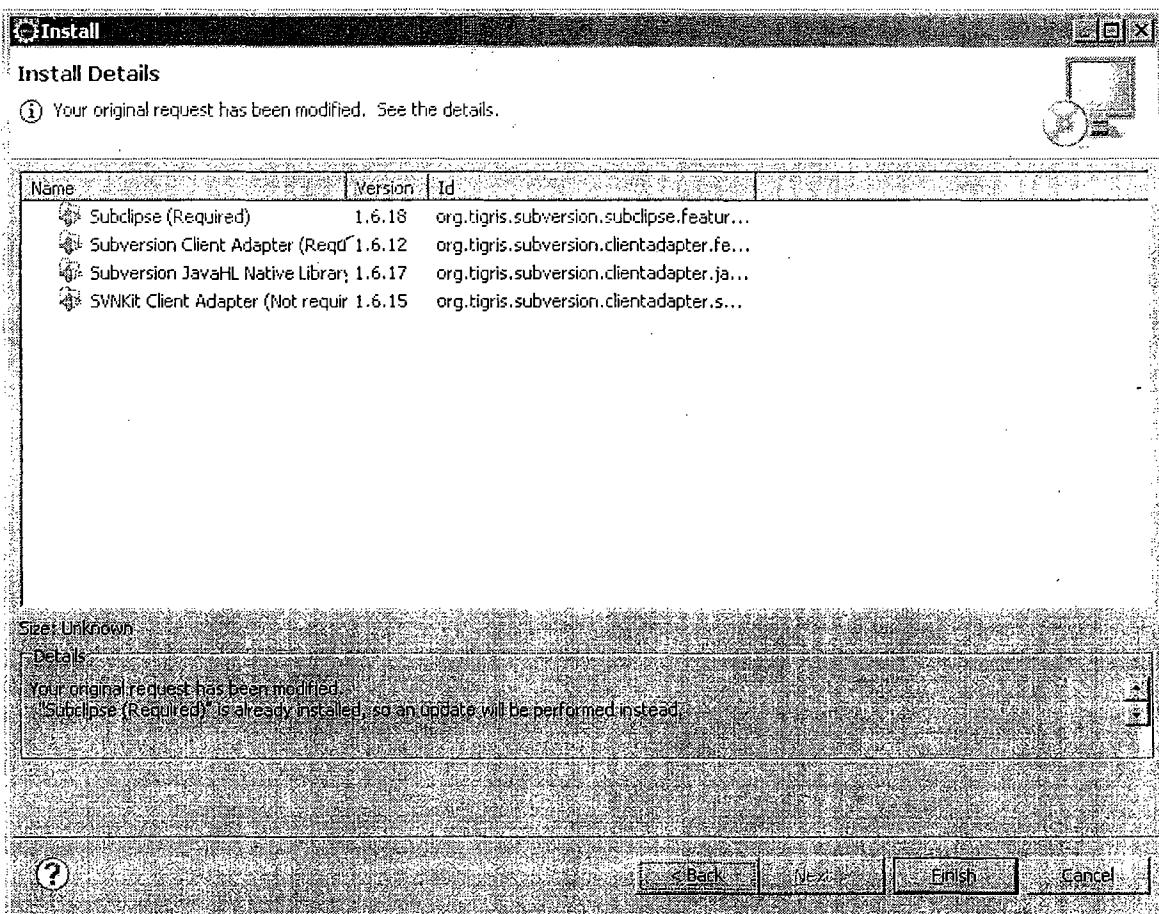
Expand Subclipse:

DURGA SOFTWARE SOLUTIONS
Tools Material



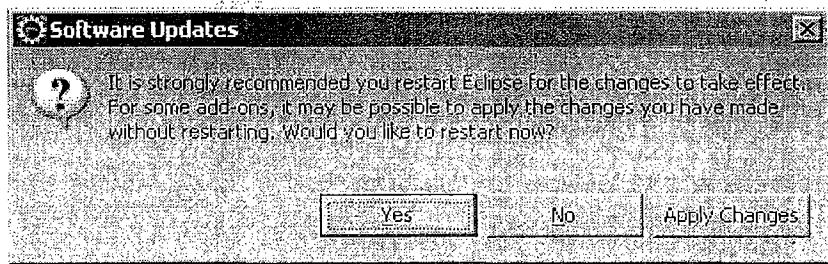
Click on next

DURGA SOFTWARE SOLUTIONS
Tools Material



Click Finish

After once installed it will prompts to restart the start system



Thank you.....

>>>>>>>Log4J Apps>>>>>>>>>>>>

App1

-----SelectTest.java-----

```
1 //SelectTest.java
2 import java.sql.*; // jdbc api
3 import org.apache.log4j.*; // log4j api
```

```
4 public class SelectTest
5 {
```

```
6     public static void main(String[] args) throws Exception
7     {
```

```
8         //logger obj is ready pointing to SelectTest class
9         Logger logger=Logger.getLogger(SelectTest.class);
```

```
10        //layout object
11        SimpleLayout layout=new SimpleLayout();
```

```
12        //appender object
13        ConsoleAppender appender=new ConsoleAppender(layout);
```

```
14        //add appender to logger
15        logger.addAppender(appender);
```

```
16        //set logger level for retrieving log messages
17        logger.setLevel((Level)Level.DEBUG);
```

```
19        // write jdbc code here
20        try
21        {
```

```
22            Class.forName("oracle.jdbc.driver.OracleDriver");
23            logger.debug("Proj1.mod1.App1.SelectTest.Driver is Loaded");
```

```
25            Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
26            logger.info("Proj1.mod1.App1.SelectTest.Conection established");
```

```
28            Statement st=con.createStatement();
29            logger.debug("Proj1.mod1.App1.SelectTest.Satement obj is ready");
30            INFO
```

```
32            ResultSet rs=st.executeQuery("select * from emp1");
33            if(rs==null)
```

```
34            {
35                logger.warn("Proj1.mod1.App1.SelectTest.empty ResultSet has come");
36            }
37            else
```

```
38            {
39                INFO
40                logger.debug("Proj1.mod1.App1.SelectTest.ResultSet geneated");
41            }
```

```
43            while(rs.next())
44            {
```

```
45                System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
46            }INFO
47            //while
```

```
48            logger.debug("Proj1.mod1.App1.SelectTest.ResultSet obj processed");
```

```
50            rs.close();
51            st.close();
52            con.close();
53            logger.debug("Proj1.mod1.App1.SelectTest.all jdbc streams are closed");
54        }
```

```
55        catch(SQLException se)
```

-> Basic example using SimpleLayout &
ConsoleAppender.

```

25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124

{
    logger.error("Proj1.mod1.App1.SelectTest.DB Exception ");
    se.printStackTrace();
}
catch(Exception e)
{
    logger.fatal("Proj1.mod1.App1.SelectTest.Unknown problem");
    e.printStackTrace();
}
logger.debug("Proj1.mod1.App1.SelectTest.end of the Application");

} //main
} //class
//add log4-version.jar , odbc14.jar files to classpath
//>javac SelectTest.java
//>java SelectTest
-----
-----App2 : Appln using HTMLLayout & WriterAppender.
-----SelectTest.java-----
//SelectTest.java
import java.sql.*; // jdbc api
import org.apache.log4j.*; // log4j api
import java.io.*;
public class SelectTest
{
    public static void main(String[] args) throws Exception
    {
        //logger obj is ready pointing to SelectTest class
        Logger logger=Logger.getLogger(SelectTest.class);
        //layout object
        HTMLLayout layout=new HTMLLayout();
        //appender object
        FileOutputStream fos=new FileOutputStream("log1.html",true);
        WriterAppender appender=new WriterAppender(layout,fos);
        //add appender to logger
        logger.addAppender(appender);

        //set logger level for retrieving log messages
        logger.setLevel(Level.DEBUG);

        // write jdbc code here
        try
        {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            logger.debug("Proj1.mod1.App1.SelectTest.Driver is Loaded");

            Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
            logger.info("Proj1.mod1.App1.SelectTest.Conection established");

            Statement st=con.createStatement();
            logger.debug("Proj1.mod1.App1.SelectTest.Statement obj is ready");

            ResultSet rs=st.executeQuery("select * from emp1");
            if(rs==null)
            {
                logger.warn("Proj1.mod1.App1.SelectTest.empty ResultSet has come");
            }
            else
            {
                logger.debug("Proj1.mod1.App1.SelectTest.ResultSet geneated");
            }
        }
    }
}

```

```

25
26     while(rs.next())
27     {
28         System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
29     }//while
30     logger.debug("Proj1.mod1.App1.SelectTest.ResultSet obj processed");
31
32     rs.close();
33     st.close();
34     con.close();
35     logger.debug("Proj1.mod1.App1.SelectTest.all jdbc streams are closed");
36 }
37 catch(SQLException se)
38 {
39     logger.error("Proj1.mod1.App1.SelectTest.DB Exception ");
40     se.printStackTrace();
41 }
42 catch(Exception e)
43 {
44     logger.fatal("Proj1.mod1.App1.SelectTest.Unknown problem");
45     e.printStackTrace();
46 }
47 logger.debug("Proj1.mod1.App1.SelectTest.end of the Application");
48
49 } //main
50 } //class

```

//add log4j-version.jar , odbc14.jar files to classpath

//>javac SelectTest.java

//>java SelectTest SelectTest.java AppIn3) Applying using Propertiesfile Log4j configuration.

//SelectTest.java

```

import java.sql.*; // jdbc api
import org.apache.log4j.*; // log4j api

```

public class SelectTest

{

public static void main(String[] args) throws Exception

{

//logger obj is ready pointing to SelectTest class

Logger logger=Logger.getLogger(SelectTest.class);

//locate the properties file where log4j cfgs are placed

PropertyConfigurator.configure("log.properties"); It points to property file

// write jdbc code here

try

{

Class.forName("oracle.jdbc.driver.OracleDriver");

logger.debug("Proj1.mod1.App1.SelectTest.Driver is Loaded");

Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
logger.info("Proj1.mod1.App1.SelectTest.Connection established");

Statement st=con.createStatement();

logger.debug("Proj1.mod1.App1.SelectTest.Statement obj is ready");

ResultSet rs=st.executeQuery("select * from emp");

if(rs==null)

{
logger.warn("Proj1.mod1.App1.SelectTest.empty ResultSet has come");

```

187     }
188     else
189     {
190         logger.debug("Proj1.mod1.App1.SelectTest.ResultSet geneated");
191     }
192
193     while(rs.next())
194     {
195         System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
196     }//while
197     logger.debug("Proj1.mod1.App1.SelectTest.ResultSet obj processed");
198
199     rs.close();
200     st.close();
201     con.close();
202     logger.debug("Proj1.mod1.App1.SelectTest.all jdbc streams are closed");
203 } //try
204 catch(SQLException se)
205 {
206     logger.error("Proj1.mod1.App1.SelectTest.DB Exception ");
207     se.printStackTrace();
208 }
209 catch(Exception e)
210 {
211     logger.fatal("Proj1.mod1.App1.SelectTest.Unknown problem");
212     e.printStackTrace();
213 }
214 logger.debug("Proj1.mod1.App1.SelectTest.end of the Application");
215 } //main
216 } //class
217 //add log4j-version.jar , odbc14.jar files to classpath
218 //>javac SelectTest.java
219 //>java SelectTest
220 -----log.properties-----
221 #for ConsoleAppender,SimpleLayout (# indicates comment)
222 #log4j.rootLogger=INFO,S -> alias name representing logger obj.
223 #log4j.appender.S=org.apache.log4j.ConsoleAppender //adding appender to logger obj (S).
224 #log4j.appender.S.layout=org.apache.log4j.SimpleLayout //adding layout to logger obj (S).
225
226
227 # for FileAppender,HTMLLayout
228 #log4j.rootLogger=DEBUG,S
229 #log4j.appender.S=org.apache.log4j.FileAppender
230 #log4j.appender.S.File=mymsgs.html
231 #log4j.appender.S.Append=true
232 #log4j.appender.S.layout=org.apache.log4j.HTMLLayout
233
234 # for RollingFileAppender,HTMLLayout
235 #log4j.rootLogger=DEBUG,R
236 #log4j.appender.R=org.apache.log4j.RollingFileAppender
237 #log4j.appender.R.File=mylog1.html
238 #log4j.appender.R.MaxFileSize=5KB -> MaxSize of current log file.
239 #log4j.appender.R.MaxBackupIndex=10 -> Max Backup file that are allowed.
240 #log4j.appender.R.layout=org.apache.log4j.HTMLLayout
241 -for XML Layout : log4j.appender.R.layout=org.apache.log4j.xml.XMLLayout.
242
243 #for DailyRollingFileAppender,PatternLayout
244 log4j.rootLogger=DEBUG,R
245 log4j.appender.R=org.apache.log4j.DailyRollingFileAppender
246 log4j.appender.R.File=myfile.txt
247 log4j.appender.R.DatePattern='yyyy-MM-dd-HH-mm'
248 log4j.appender.R.layout=org.apache.log4j.PatternLayout

```

```

49 log4j.appenders.R.layout.ConversionPattern=%p %r [%t] %c %m %d %n
50
51 App3 : Appln using XML file based log4j configurations.
52 -----details.xml-----
53 <?xml version="1.0" encoding="UTF-8" ?>
54 <!DOCTYPE log4j:configuration SYSTEM "log4j.dtd"><log4j:configuration
55 xmlns:log4j="http://jakarta.apache.org/log4j/">
56     ↗ Appenders logical name.
57     <appender name="abc" class="org.apache.log4j.ConsoleAppender">
58         <layout class="org.apache.log4j.SimpleLayout"/>
59     </appender>
60     <root>
61         <priority value ="warn" /> Logger level to retrieve the log message.
62         <appender-ref ref="abc"/> refer line 256.
63     </root>
64 </log4j:configuration>
65 -----ExternalXmlTest.java-----
66 import org.apache.log4j.Logger;
67 import org.apache.log4j.xml.DOMConfigurator;
68 public class externalxmltest
69 {
70     static Logger logger = Logger.getLogger(externalxmltest.class);
71     public static void main(String args[])
72     {
73         DOMConfigurator.configure("details.xml");
74
75         logger.debug("Here is some DEBUG");
76         logger.info("Here is some INFO");
77         logger.warn("Here is some WARN");
78         logger.error("Here is some ERROR");
79         logger.fatal("Here is some FATAL");
80     }
81 }

```

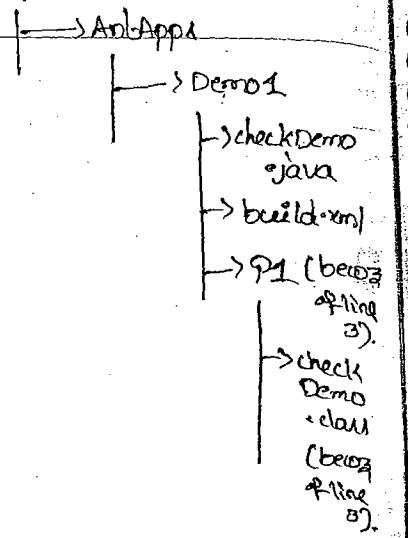
The possible values for log4j.appenders.R.DatePattern (property)

- yyyy-MM : Roll log file on the front of each Month.
- yyyy-WW : " " " at the start of each week.
- yyyy-MM-dd : " " " at the midnight everyday.
- yyyy-MM-dd-a : Roll log file at the midnight & Midday everyday.
- yyyy-MM-dd-HH : " " " " start-of every hour.
- yyyy-MM-dd-HH-mm : Roll log file at the beginning of every minute.

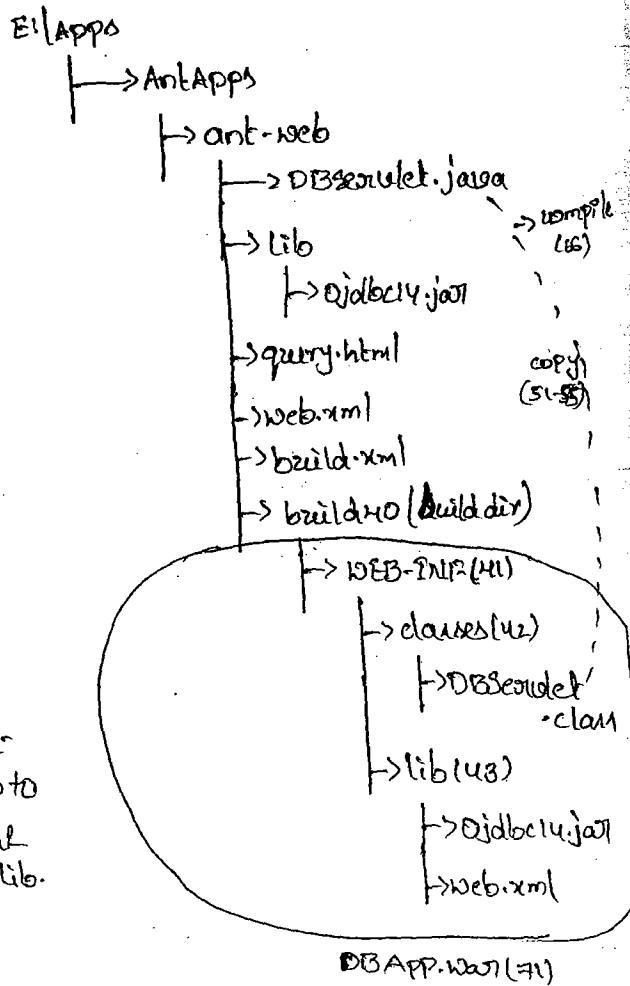
1 >>>>>>>>> Ant Build files >>>>>>>>>>>>

2 -----

3 File1
 4 ----- build.xml logical name & default target to execute
 5 <!- Build Script for compiling and running Basic Java App -->
 6 <project name="test" default="exec" basedir=".">
 7 <!- variables declaration --> (c)
 8 <property name="src" value="."/> Everything of current
 9 <property name="pack" value="p1"/> directory should be
 10
 11 <!- target for compilation --> managed to locate.
 12 <target name="compile">
 13 (e) <javac srcdir="\${src}" destdir="\${src}"/>
 14 </target>
 15 Refer no: 8 Place where to generate
 16 <!- target for execution --> .class files of pkg p1.
 17 (d) <target name="exec" depends="compile">
 18 (f) <java classname="\${pack}.CheckDemo"/>
 19 </target>
 20 </project> refer ① bcoz of this execution done to p1.CheckDemo
 21 excludes p1.CheckDemo.class
 22
 23 f → execute the appn.
 24
 25 File2
 26 (b) ----- build.xml -----
 27 <!- building and deploying application using ant tool -->
 28 <project name="Dep_Web" basedir=". default="deploy">
 29 (c)
 30 <!- Variable Declaration -->
 31 <property name="home" value="D:/Tomcat5.5"/>
 32 <property name="deploy.dir" value="\${home}/webapps"/>
 33 <property name="build.dir" value="build"/>
 34 <property name="web-inf" value="\${build.dir}/WEB-INF"/>
 35 <property name="warname" value="DBApp"/>
 36
 37 <!- construct staging dir structure -->
 38 <target name="prepare">
 39
 40 <mkdir dir="\${build.dir}"/>
 41 <mkdir dir="\${web-inf}"/>
 42 <mkdir dir="\${web-inf}/classes"/>
 43 <mkdir dir="\${web-inf}/lib"/>
 44 destination folder
 45 <copy todir="\${web-inf}"/>
 46 <fileset dir="."> source folder
 47 <include name="web.xml"/>
 48 </fileset>
 49 </copy>
 50 destination folder
 51 <copy todir="\${web-inf}/lib">
 52 <fileset dir="lib"> source folder from current
 53 <include name="classes111.jar"/> directory lib to
 54 </fileset> destination
 55 </copy>
 56 destination folder
 57 <copy todir="\${build.dir}"/>
 58 <fileset dir="."> source folder
 59 <include name=".html"/>
 60 </fileset>
 61 </copy>
 62 </target>



E:\App\AntApp\Demo1>ant
 (a)



E:\App\AntApp\ant-web> ant

(a)

```

63
64 <!-- compile java sources -->
65 <target name="compile" depends="prepare">
66   <javac srcdir="." destdir="${web-inf}/classes"/>
67 </target>           ↓ compiles .java files of current directory & generates .class file in
68                                         WEB-INF / class folder.
69 <!-- Create binary distribution (war file)-->
70 <target name="war" depends="compile"> (e)
71   <war basedir="${build.dir}" webxml="${web-inf}/web.xml" warfile="${warname}.war" /> It creates
72 </target>           ↓
73
74 <!-- Deploy war file to tomcat server-->
75 <target name="deploy" depends="war"> (d)
76   <copy file="${warname}.war" todir="${deploy.dir}"/>
77   <delete file="${warname}.war"/>
78 </target>           ↓
79 </project>           deletes war file of current directory.
80
81
82
83 File3
-----build-struts.xml-----
84
85 <!--building and deploying struts application using ant tools -->
86
87 <project name="Dep_Struts" basedir="." default="deploy">
88
89 <!-- Variable Declaration -->
90 <property name="home" value="D:\Tomcat5.5"/>
91 <property name="deploy.dir" value="${home}/webapps"/>
92 <property name="build.dir" value="build"/>
93   <property name="base.dir" value=". "/> (11)
94   <property name="web-inf" value="${build.dir}/WEB-INF"/>
95 <property name="warname" value="DemoWeb"/>
96
97 <!-- classpath for Struts 1.3 -->
98 <path id="compile.classpath">
99   <pathelement path="lib/struts-core.1.3.8.jar"/>
100  <pathelement path="lib/servlet-api.jar"/>
101 </path>
102
103 <!-- construct staging dir structure -->
104 <target name="prepare">
105   <mkdir dir="${build.dir}"/>
106   <mkdir dir="${web-inf}"/>
107   <mkdir dir="${web-inf}/classes"/>
108   <mkdir dir="${web-inf}/lib"/>
109
110   destination folder
111 <copy todir="${web-inf}">
112   <fileset dir="${base.dir}"> source folder.
113     <include name="*.xml"/>
114     <include name="*.tld"/>
115   </fileset>
116 </copy>
117
118 <copy todir="${web-inf}/lib">
119   <fileset dir="jars">
120     <include name="*.jar"/>
121   </fileset>
122 </copy>
123
124 <copy todir="${build.dir}">
125   <fileset dir=".">
126

```

WEB-INF / class folder.

target 35

DB App. war file

representing the
Web application.

} logic to add jar files to classpath.

```

125 <include name="*.jsp"/>
126 </fileset>
127 </copy>
128 </target>
129
130 <!-- Normal build of application -->
131 <target name="compile" depends="prepare">
132   <javac srcdir="src" destdir="${web-inf}/classes">
133     <classpath refid="compile.classpath"/>
134   </javac>           ↪ ref 98 to 101
135 </target>
136
137 <!-- Create binary distribution (war-file)-->
138 <target name="war" depends="compile">
139   <war basedir="${build.dir}" warfile="${warname}.war" webxml="${web-inf}/web.xml"/> // creates
140 </target>           ↪ from base directory      ↪ war-filename      ↪ WEB-INF
141           ↓                               ↓                               ↓
142 <!-- Copy war file to tomcat -->
143   <target name="deploy" depends="war">
144     <copy file="${base.dir}/${warname}.war" todir="${deploy.dir}">           ↪ copy the current directory war file
145       <delete file="${base.dir}/${warname}.war"/>                                → to destination directly.
146   </target>
147 </project>           ↓
148           delete war file from current directory.
149

```

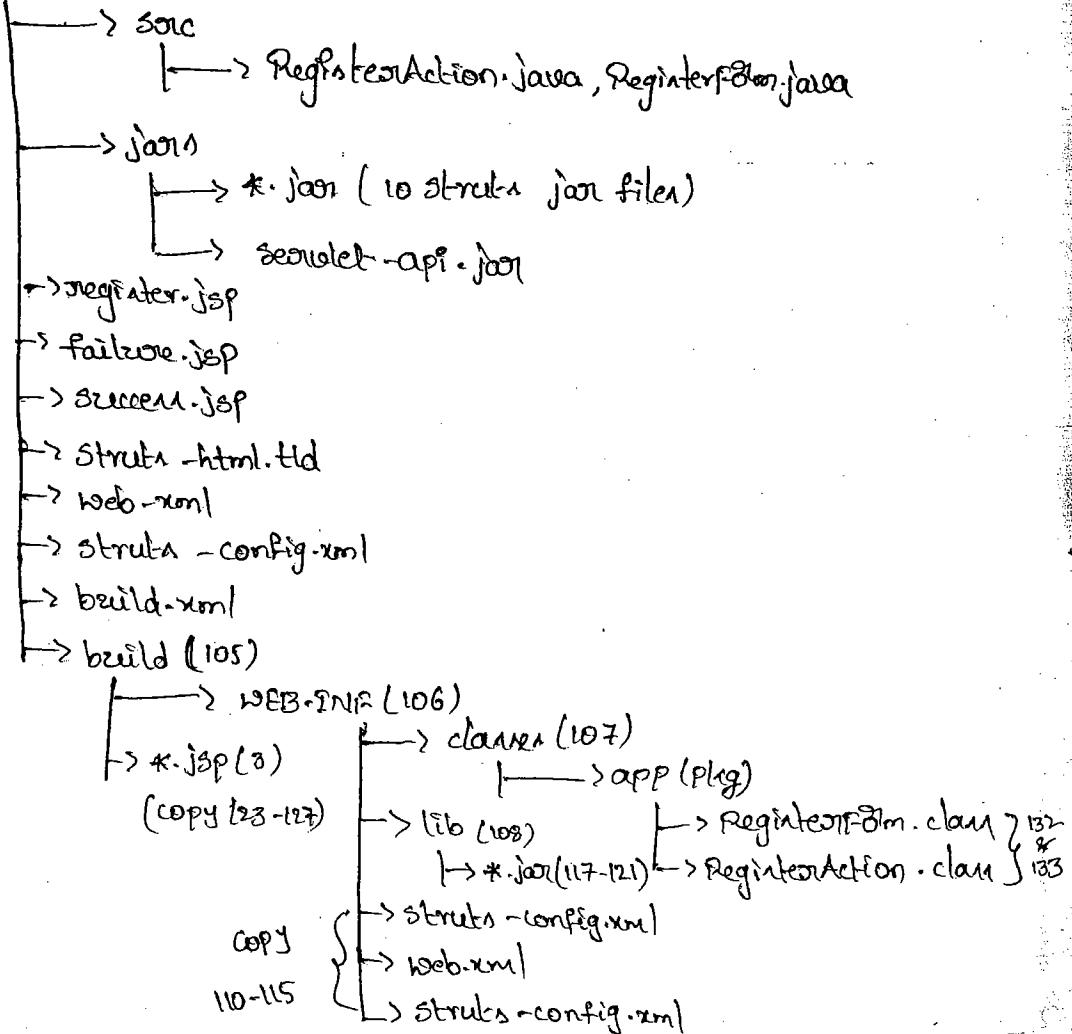
Handwritten notes:

- Annotations for target "war": "creates war file", "WEB-INF", "warfilename", "web.xml", "warfilename".
- Annotation for target "deploy": "copy the current directory war file to destination directly".
- Annotation for project end: "delete war file from current directory".

E:\APP\

| -> AntApp1

| -> ant-struts



1 >>>>> Apps on junit >>>>>>>

2
3 App1
4 ----- Test1.java Actual class to test. (fixture).

5 public class Test1
6 {
7 public static int add(int x,int y)
8 {
9 return x+y;
10 }
11 public static int sub(int x,int y)
12 {
13 return x-y;
14 }

15 }
16 }
17 ----- TestMath.java -----

18 // TestMath.java
19 import junit.framework.*;

20
21 public class TestMath extends TestCase -> Mandatory.

22 {
23 public void setUp()
24 {
25 System.out.println("setUp");
26 }

27 public void testAdd()
28 {
29 int num1=3;
30 int num2=2;
31 int total=5; expected Result
32 int sum=Test1.add(num1,num2); Actual Result
33 assertEquals(total,sum);
34 }
35 ↓
36 expected Result → Actual Result.

37 public void testAdd1()
38 {
39 int num1=13;
40 int num2=12;
41 int total=25; expected Result
42 int sum=Test1.add(num1,num2); Actual Result
43 assertEquals(total,sum);
44 }
45 ↓
46 expected Result → Actual Result.

47 public void tearDown()
48 {
49 System.out.println("tearDown");
50 }

51 public void testSub()
52 {
53 int num1=3;
54 int num2=2;
55 int res1=1; Expected Result
56 int res2=Test1.sub(num1,num2); Actual Result.
57 assertEquals(res1,res2);
58 }
59 ↓
60 Here actual computation is done.

E:\ APPS

→ junitApps

→ APP1

→ Test1.java
→ TestMath.java
→ AllTests.java

E:\ APP1\ junitApps\ APP1> javac *.java
> java AllTests

Testcase1.

Testcase2.

Testcase3

AllTest.java Test write (to run all the testcases).

import junit.framework.*;

```

63 public class AllTests {
64     public static TestSuite suite() {
65         TestSuite suite = new TestSuite("All JUnit Tests");
66         suite.addTestSuite(TestMath.class); → Testcase is added
67         return suite;
68     }
69
70     public static void main (String[] args) {
71         junit.textui.TestRunner.run(suite()); → Run all the testcases that are added to
72     }                                     → testsuite & generated report.
73 }   pkg name class method
74
75 App2(Unit Testing of webapplication using httpunit)
76 -----webapplication-----
77 -----index.jsp-----
78 <form action=verify.jsp>
79 UserName:<input type=text name=username><br>
80 Password:<input type=text name=password><br>
81 <input type=submit value>Login>
82 </form>
83 -----verify.jsp-----
84 <%
85 String uname=request.getParameter("username");
86 String pass=request.getParameter("password");
87
88 if(uname.equals("durga") && pass.equals("tech"))
89 out.print("Valid credentials");
90 else
91 out.print("Invalid Credentials");
92 %>
93 -----web.xml-----
94 <web-app/>
95 -----TestCase1.java-----
96 import junit.framework.TestCase;
97
98 import com.meterware.httpunit.WebConversation;
99 import com.meterware.httpunit.WebForm;
100 import com.meterware.httpunitWebResponse;
101
102
103 public class TestCase1 extends TestCase {
104
105     public void testGoodLogin() {
106         try {
107             WebConversation conversation = new WebConversation(); → BAN def to create other obj.
108
109             WebResponse response = conversation.getResponse( "http://localhost:2020/HelpUnitApp/index.jsp" );
110             // System.out.println( response );
111
112             WebForm loginForm = response.getForms()[0];
113             loginForm.setParameters("username", "durga"); → gives access to form of index.jsp.
114             loginForm.setParameters("password", "tech"); } → sets request parameter values.
115             response = loginForm.submit(); → submit the request from form page to verify.jsp.
116
117             // System.out.println( response.getText() );
118             String resp=response.getText(); → Gathers the response given by verify.jsp.
119             System.out.println(resp);
120
121             assertEquals("Valid credentials",resp);
122
123         } catch (Exception e) {
124

```

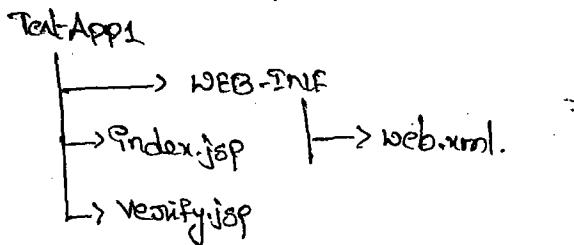
```

25 System.err.println( "Exception: " + e );
26 }
27 } //method
28 public void testBadLogin() {
29 try {
30     WebConversation conversation = new WebConversation();
31
32     WebResponse response = conversation.getResponse( "http://localhost:2020/HttpUnitApp/index.jsp" );
33
34     WebForm loginForm = response.getForms()[0];
35
36     loginForm.setParameter( "username", "durga1" );
37     loginForm.setParameter( "password", "tech" );
38
39     response = loginForm.submit();
40     //System.out.println( response.getText() );
41     String resp=response.getText();
42     assertEquals("Invalid Credentials",resp);
43 } catch (Exception e) {
44     System.err.println( "Exception: " + e );
45 }
46 } //method
47 } //class
48

```

Steps to execute AppIn 2:

Step1: Deploye web appln in tomcat server & make sure that it is running perfectly. (Develop it as web appln in MyEclipse).



Step2: Make sure that HttpUnit s/w is installed.

Download & extract httpunit-1.7.zip file.

Step3: Create Java Project in MyEclipse IDE & add following jar files collected from Junit s/w.

httpunit.jar → JS-1.6R5.jar , jtidy-4Aug2000r7-dec.jar.

Note: Also add JUnit 3 (8) JUnit Library.

Step4: Add Testcase class to the Project using HttpUnit API. Refer Testcase1.jar of Previous Page.

Step5: Add TestBuite class to project selecting the above Testcase class.

Step6: Run the TestBuite class. Right click → Run as → Junit-Test.
on the project

```

1 >>>>>>>>> Apps on jasper reports>>>>>>>>>
2 -----
3 App1
4 -----FirstReport.jrxml----- Design the report.
5 <?xml version="1.0"?>
6 <!DOCTYPE jasperReport
7 PUBLIC "-//JasperReports//DTD Report Design//EN"
8 "http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
9 <jasperReport name="FirstReport"> -> logical name.
10 <detail>
11   <band height="20">
12     <staticText>    To decide x & y coordinate.
13       <reportElement x="20" y="0" width="200" height="20" forecolor="#FF3300" backcolor="#FFFFFF"/>
14         <text><![CDATA[Welcome to Jasper Reports ]]></text>
15     </staticText>
16   </band>
17 </detail>
18 </jasperReport>
19 -----FirstReportCompile.java-----
20 import net.sf.jasperreports.engine.JasperCompileManager;
21 public class FirstReportCompile
22 {
23   public static void main(String[] args)
24   {
25     try
26     {
27       System.out.println("Compiling report...");
28       JasperCompileManager.compileReportToFile("FirstReport.jrxml");
29       System.out.println("Done!");
30     }
31     catch(Exception e)
32     {
33       e.printStackTrace();
34     }    > javac FirstReportCompile.java
35   }    > java FirstReportCompile -> gives FirstReport.jasper file
36 } -----FirstReportFill.java-----
37 import java.util.HashMap;
38 import net.sf.jasperreports.engine.JREmptyDataSource;
39 import net.sf.jasperreports.engine.JasperFillManager;
40 import net.sf.jasperreports.engine.JasperRunManager;
41 public class FirstReportFill
42 {
43   public static void main(String[] args)
44   {
45     try
46     {
47       System.out.println("Filling report...");
48       JasperFillManager.fillReportToFile("FirstReport.jasper", new HashMap(), new JREmptyDataSource());    > given FirstReport.jrPrint file
49       new HashMap(), new JREmptyDataSource());
50
51
52 //JasperRunManager.runReportToHtmlFile("FirstReport.jasper",new HashMap(),new
53 JREmptyDataSource()); -> gives report in the .html file
54 JasperRunManager.runReportToPdfFile("FirstReport.jasper",new HashMap(),new JREmptyDataSource());    L -> gives the report in the .pdf file.
55 System.out.println("Done!");
56 }
57 catch(Exception e)
58 {
59   e.printStackTrace();
60 }
61 }

```

> FirstReportCompile.java
 > FirstReportFill.java
 > JavaC * .java.
 > java FirstReportCompile.
 > java FirstReportFill.

Jar files in classpath

Jasperreports-1.3.7.5.jar.
 Commons-beanutils-1.8.0.jar.
 Commons-digester-1.7.jar.
 Commons-logging-1.0.4.jar.
 FreeMarker-2.1.7.jar.
 Freechart-1.0.12.jar
 Commons-collections-2.1.1.jar

```

62 } >javac FirstReportFill.java
63 } >java FirstReportFill
64 Generates the report.

```

App2

TestServlet.java (Servlet Program)

```

67 import java.io.*;
68 import java.util.HashMap;
69 import javax.servlet.*;
70 import javax.servlet.http.*;
71 import net.sf.jasperreports.engine.*;
72 public class TestServlet extends HttpServlet
73 {
74
75 protected void doGet(HttpServletRequest request, HttpServletResponse
76 response) throws ServletException, IOException
77 {
78 ServletOutputStream sos =
79 response.getOutputStream();
80
81 response.setContentType("application/pdf");
82 response.setHeader("Content-disposition", "inline;filename=\\\"Report.pdf\\\"");
83 try
84 {
85 JasperPrint jp=JasperFillManager.fillReport(getServletContext().getRealPath("/")+"FirstReport.jasper",new
86 HashMap(),new JREmptyDataSource());
87 byte[] pdfasbytes = JasperExportManager.exportReportToPdf(jp);
88
89 //for bytes representing the report content.
90 sos.write(pdfasbytes);
91
92 //It displays the bytes as downloadable file in
93 //browser window.
94 sos.flush();
95 sos.close();
96
97 }
98 catch (JRException e)
99 {
100 e.printStackTrace();
101 }
102
103 }
104
105 //class
106
107 -----web.xml-----
<web-app>
<servlet>
<servlet-name>abc</servlet-name>
<servlet-class>TestServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>abc</servlet-name>
<url-pattern>/testurl</url-pattern>
</servlet-mapping>
</web-app>

```

App3

catalog.xml-----

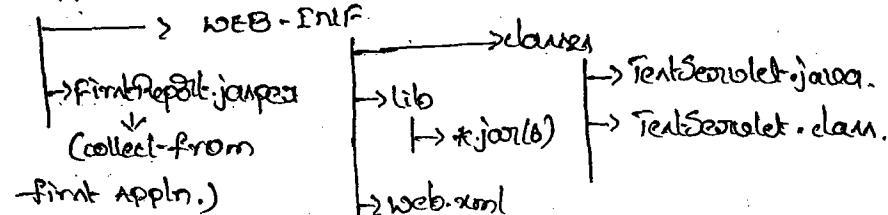
```

<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE jasperReport PUBLIC "-//JasperReports//DTD Report Design//EN"
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
<jasperReport name="PDFReport" pageWidth="975">
<reportFont name="Arial_Normal" isDefault="true" fontName="Arial" size="12" isBold="false"
isItalic="false" isUnderline="false" isStrikeThrough="false" pdfFontName="Helvetica"
pdfEncoding="Cp1252" isPdfEmbedded="false"/>
<reportFont name="Arial_Bold" isDefault="false" fontName="Arial" size="12" isBold="true" isItalic="false"

```

↳ declaring → some logical name

JApp2



jar files in classpath: servlet-api.jar,
jasperreports-3.7.5.jar.

jar files in WEB-INF/lib folder:

commons-beanutils-1.8.0.jar, commons-digester-1.7.jar,
commons-collections-2.1.1.jar, commons-logging-1.0.4.jar,
ff4j-2.1.7.jar, jasperreports-3.7.5.jar.

```

isUnderline="false" isStrikeThrough="false" pdfFontName="Helvetica-Bold" pdfEncoding="Cp1252"
isPdfEmbedded="false"/>
121 <reportFont name="Arial_IItalic" isDefault="false" fontName="Arial" size="12" isBold="false" isItalic="true" 181
    isUnderline="false" isStrikeThrough="false" pdfFontName="Helvetica-Oblique" pdfEncoding="Cp1252" 182
    isPdfEmbedded="false"/>
122
123 <parameter name="ReportTitle" class="java.lang.String"/> → parameter declaration. 183
124 <queryString><![CDATA[SELECT CatalogId, Journal, Publisher, Edition, Title, Author FROM 184
    Catalog1]]></queryString> → Query that selects the details from table. 185
125
126 <field name="CatalogId" class="java.lang.String"/> } field declaration must match with 186
127 <field name="Journal" class="java.lang.String"/> column names in DB table. 187
128 <field name="Publisher" class="java.lang.String"/>
129 <field name="Edition" class="java.lang.String"/>
130 <field name="Title" class="java.lang.String"/>
131 <field name="Author" class="java.lang.String"/>
132
133 <title>
134     <band height="50"> → allow to define content in various position of the report. 196
135         <textField> → (Some text will be displayed) Allow to specify text content of the 197
136             <reportElement x="350" y="0" width="200" height="50" /> → specifies location of the content. 198
137             <textFieldExpression class="java.lang.String">$P{ReportTitle}</textFieldExpression> 199
138         </textField>                                ↓ this parameter value become title of 200
139     </band>                                         the report (refer line no: 123 & 293, 201
140 </title>                                         294. 202
141
142 <pageHeader>
143     <band>
144     </band>
145 </pageHeader>
146
147 <columnHeader>
148     <band height="20">
149         <staticText> → denotes the position of static text. 209
150             <reportElement x="0" y="0" width="100" height="20"/> 210
151             <textElement> → denotes the style of static content. 211
152                 <font isUnderline="false" reportFont="Arial_Bold"/> 212
153             </textElement>                                ↳ refer line no: 120. 213
154             <text><![CDATA[CATALOG ID]]></text> 214
155         </staticText>                                ↳ Actual data to display. 215
156         <staticText>
157             <reportElement x="125" y="0" width="100" height="20"/> 216
158             <textElement> → Allow to specify styles of the content. 217
159                 <font isUnderline="false" reportFont="Arial_Bold"/> 218
160             </textElement>
161             <text><![CDATA[JOURNAL]]></text> 219
162         </staticText>
163         <staticText>
164             <reportElement x="250" y="0" width="150" height="20"/> 220
165             <textElement>
166                 <font isUnderline="false" reportFont="Arial_Bold"/> 221
167             </textElement>
168             <text><![CDATA[PUBLISHER]]></text> 222
169         </staticText>
170
171
172         <staticText>
173             <reportElement x="425" y="0" width="100" height="20"/> 223
174             <textElement>
175                 <font isUnderline="false" reportFont="Arial_Bold"/> 224
176             </textElement>
177             <text><![CDATA[EDITION]]></text> 225

```

```

178 </staticText>
179 <staticText>
180   <reportElement x="550" y="0" width="200" height="20"/>
181   <textElement>
182     <font isUnderline="false" reportFont="Arial_Bold"/>
183   </textElement>
184   <text><![CDATA[TITLE]]></text>
185 </staticText>
186
187 <staticText>
188   <reportElement x="775" y="0" width="200" height="20"/>
189   <textElement>
190     <font isUnderline="false" reportFont="Arial_Bold"/>
191   </textElement>
192   <text><![CDATA[AUTHOR]]></text>
193 </staticText>
194
195 </band>
196 </columnHeader>
197
198 <detail>
199   <band height="20">
200     <textField>
201       <reportElement x="0" y="0" width="100" height="20"/>
202       <textFieldExpression class="java.lang.String"><![CDATA[$F{CatalogId}]]></textFieldExpression>
203     </textField>
204
205     <textField pattern="0.00">
206       <reportElement x="125" y="0" width="100" height="20"/>
207       <textFieldExpression class="java.lang.String"><![CDATA[$F{Journal}]]></textFieldExpression>
208     </textField>
209
210     <textField pattern="0.00">
211       <reportElement x="250" y="0" width="150" height="20"/>
212       <textFieldExpression class="java.lang.String"><![CDATA[$F{Publisher}]]></textFieldExpression>
213     </textField>
214
215     <textField>
216       <reportElement x="425" y="0" width="100" height="20"/>
217
218       <textFieldExpression class="java.lang.String"><![CDATA[$F{Edition}]]></textFieldExpression>
219     </textField>
220
221     <textField pattern="0.00">
222       <reportElement x="550" y="0" width="200" height="20"/>
223       <textFieldExpression class="java.lang.String"><![CDATA[$F{Title}]]></textFieldExpression>
224     </textField>
225
226     <textField>
227       <reportElement x="775" y="0" width="200" height="20"/>
228       <textFieldExpression class="java.lang.String"><![CDATA[$F{Author}]]></textFieldExpression>
229     </textField>
230
231   </band>
232 </detail>
233 <columnFooter>
234   <band>
235   </band>
236 </columnFooter>
237
238 <pageFooter>
239   <band height="15">

```

↓
Display column value. (refer line no: 126)

```

240 <staticText>
241   <reportElement x="0" y="0" width="40" height="15"/>
242   <textElement>
243     <font isUnderline="false" reportFont="Arial_Italic"/>
244   </textElement>
245   <text><![CDATA[Page #]]></text>
246 </staticText>
247 <textField>
248   <reportElement x="40" y="0" width="100" height="15"/>
249   <textElement>
250     <font isUnderline="false" reportFont="Arial_Italic"/>
251   </textElement>
252   <textFieldExpression
253     class="java.lang.Integer"><![CDATA[$V{PAGE_NUMBER}]]></textFieldExpression>
254 </textField>
255 </band>
256 </pageFooter>
257 <summary>
258   <band>
259   </band>
260 </summary>
261
262 </jasperReport>
263 -----catalog.jsp----- (generates dropPDF in the form of pdf document-).
264 <%@ page import="java.io.* , java.util.* , java.sql.* , net.sf.jasperreports.engine.* ,
265           net.sf.jasperreports.engine.design.JasperDesign,net.sf.jasperreports.engine.xml.JRXmlLoader"
266 %>
267
268 <%
269 InputStream input=new FileInputStream(new File("c:/JasperReports/catalog.xml")); can also be located
270 JasperDesign design = JRXmlLoader.load(input); from reso appn by using
271           It loads xml file. similar code of line no: 85.
272 JasperReport report = JasperCompileManager.compileReport(design);
273 Map parameters = new HashMap(); parameters.put("ReportTitle", "PDF JasperReport"); supplies Parameter value dynamically at
274 parameters.put("ReportTitle", "PDF JasperReport"); runtime. (refer line no: 137).
275
276 // get DB connection
277 Class.forName("oracle.jdbc.driver.OracleDriver");
278 Connection conn=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
279
280 // generate Report
281 JasperPrint print = JasperFillManager.fillReport(report,
282           parameters, conn);
283 OutputStream output=new FileOutputStream(new File("c:/JasperReports/catalog.pdf")); (
284 JasperExportManager.exportReportToPdfStream(print, output); → completes the dropPDF generation.
285 %>
286 -----catalog-excel.jsp-----
287 <%@ page import="java.io.* , java.util.* , java.sql.* , net.sf.jasperreports.engine.* ,
288           net.sf.jasperreports.engine.design.JasperDesign,net.sf.jasperreports.engine.xml.JRXmlLoader,net.sf.jasper
289           reports.engine.export.*"%
290 <%
291 InputStream input=new FileInputStream(new File("c:/JasperReports/catalog.xml"));
292 JasperDesign design = JRXmlLoader.load(input);
293
294 JasperReport report = JasperCompileManager.compileReport(design);
295 Map parameters = new HashMap();
296 parameters.put("ReportTitle", "Excel JasperReport");
297
298 // get DB connection

```

299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330

```

299 Class.forName("oracle.jdbc.driver.OracleDriver");
300 Connection conn=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
301
302
303 JasperPrint print = JasperFillManager.fillReport(report, parameters, conn);
304 OutputStream ouputStream=new FileOutputStream(new File("c:/JasperReports/catalog.xls"));
305 ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream(); → An Empty Stream To create
306
307
308 JRXLSExporter exporterXLS = new JRXMLExporter(); → internal obj creation of Exporter.
309 exporterXLS.setParameter(JRXLSExporterParameter.JASPER_PRINT, print); ↗ refer line no: 303.
310 exporterXLS.setParameter(JRXLSExporterParameter.OUTPUT_STREAM, byteArrayOutputStream);
311 exporterXLS.exportReport(); ↗ refer 305.
312 → exports entire report to byteArrayOutputStream obj.
313 ouputStream.write(byteArrayOutputStream.toByteArray());
314 ouputStream.flush(); ↗ writes the report data to catalog.xls file
315 ouputStream.close();
316 ↗ refer line no: 304.
317 %>
318 -----web.xml-----
319 <web-app>
320
321 → By using JasperExportManager we can export report directly to pdf, html, xml
322 format. To export the content to other format like xls, xls, csv we special
323
324 exports like JRXMLExporter, JRCSVExporter.
325
326
327
328
329
330

```

c:\ JasperReports

→ catalog.xml (rpt file)

Report-App



jar files in classpath: Same as Appn 2.

jar files in WEB-INF lib folder: Appn 2 jars of jidbcon.jar, poi-3.6.jar

request url: http://localhost:2020/Report-App/catalog.jsp
/catalog-excel.jsp

look for report files in c:\ JasperReports folder.

63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

1 >>>>>>>>NSIS Installer code >>>>>>
2 -----webinstaller.nsi-----
3 ;Include Modern UI
4 ; comment
5 !include "MUI2.nsh" (like headerfile).
6
7 ;----- il comment-
8 Name "WebApp SetUp" → logical name.
9 outFile "MyJavaSoftware.exe" → The setup file name to be generated.
10
11 ;Default installation folder
12 InstallDir "C:\Program Files" (which is the folder to install all related slw's). ↑ like JRE, Tomcat
13
14 ;Request application privileges for Windows Vista
15 RequestExecutionLevel user
16
17 ;
18 ;Interface Settings
19
20 !define MUI_ABORTWARNING (If any disturbances come on setup file creation (meould from that purpose))
21
22 ;
23 ;Pages
24
25 ;Page showing license agreement
26 !insertmacro MUI_PAGE_LICENSE "${NSISDIR}\Docs\Modern UI\License.txt" (to launch License page)
27
28 ;page showing options to be installed
29 !insertmacro MUI_PAGE_COMPONENTS
30
31
32 ;!insertmacro MUI_PAGE_DIRECTORY (Directry to install the slw).
33
34 ;needed,else no section will be executed
35 !insertmacro MUI_PAGE_INSTFILES
36
37 ;for uninstaller
38 ; !insertmacro MUI_UNPAGE_CONFIRM
39 ; !insertmacro MUI_UNPAGE_INSTFILES]*
40
41 ;
42 ;setting header
43 !insertmacro MUI_LANGUAGE "English"
44
45 ;
46 ;Install JRE (setup to install JRE)
47
48 Section "JRE 1.6" SecDummy
49 SetOutPath "$INSTDIR" → refer line no:12
50 file jre.exe
51 execWait jre.exe
52 SectionEnd
53
54 ;
55 ;Install Tomcat 6.0 (setup to install tomcat).
56
57 Section "Tomcat 6.0" SecTomcat
58 SetOutPath "$INSTDIR"
59 file tomcat.exe
60 execWait tomcat.exe
61 SectionEnd
62

```

```

63
64 ;-----
65 ;Deploy your application
66
67 section "Deploy Date App" secDeploy
68 setoutpath "D:\Tomcat 6.0\webapps\" } Here we copy DateApp folder to webapps folder.
69 file DateApp.war
70 sectionEnd
71
72 ;-----
73 ;make it available in startup (Procedure to create one link from start menu).
74
75 section "Create Startup" secStartup
76 StrCpy $R1 "http://localhost:2020/DateApp" This given url becomes $R1 variable value.
77 CreateDirectory $SMPROGRAMS\MyApp → create one special directory called MyApp.
78 CreateShortCut $SMPROGRAMS\MyApp\MyDateApp.lnk '$R1' →
79 sectionEnd
80
81 ;-----
82 ;Launch IE and show the page
83
84 section "Launch DateApp" secLaunch
85 StrCpy $R1 "http://localhost:2020/DateApp"
86 ExecWait "$PROGRAMFILES\Internet Explorer\iexplore.exe" "$R1" → Launches Internet Explorer having
87 $R1 supplied url.
88 sectionEnd
89
90
91
92
93
94
95
96
97
98
99

```

This handwritten note provides a detailed explanation of the script logic:

- Section 67:** Deploy the DateApp.war file to the D:\Tomcat 6.0\webapps\ folder.
- Section 75:** Create a startup entry in the Windows Start Menu pointing to the deployed URL (`http://localhost:2020/DateApp`). This URL is stored in the variable `$R1`.
- Section 84:** Launch Internet Explorer (iexplore.exe) with the URL stored in `$R1`.

```

1 // for compiling and executing java app
2 -----pom.xml-----
3 <project xmlns="http://maven.apache.org/POM/4.0.0"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
6
7   <modelVersion>4.0.0</modelVersion> over Organization Project info (userdefined)
8   <groupId>SampleApp</groupId>
9   <artifactId>CoreJava</artifactId>
10  <version>1</version>
11
12 <!--project-->
13
14 <build>
15   <finalName>MyJar</finalName> It is predefined property pointing to current directory.
16   <directory>${project.basedir}/myoutput</directory> build directory in typepath.
17   <outputDirectory>${project.build.directory}/classes</outputDirectory>
18   <sourceDirectory>${project.basedir}/src</sourceDirectory>
19 <plugins>
20   <plugin> The Provider company name of plugin
21     <groupId>org.codehaus.mojo</groupId>
22     <artifactId>exec-maven-plugin</artifactId> plugin name
23     <version>1.1.1</version> plugin version.
24   <executions>
25     <execution>
26       <phase>test</phase> -> phase name of default life cycle (refer page no: 301).
27       <goals>
28         <goal>java</goal> -> Goal name of plugin (refer page No: 304).
29       </goals>
30     <configuration>
31       <mainClass>p1.Hello</mainClass> -> class to execute.
32       <arguments>
33         <argument>2</argument> -> command line arguments values required for appln.
34         <argument>5</argument>
35       </arguments>
36     </configuration>
37   </execution>
38 </executions>
39 </plugin>
40 </plugins>
41 </build>
42 <properties> -> optional here
43   <!--user defined property here using userdefined tags-->
44   <pack>durga</pack>
45 </properties>
46 </project>
47
48 E:\AppA\maven> mvn test
49   ↓
50   executes upto
51   test-phase of
52   default-
53   life cycle.
54
55
56 Hello.java : classes/*
57   |-----> P1/*
58   |-----> Hello.class*
59   |
60   Package P1;
61   public class Hello
62   {
63     public static void main (String[] args)
64   }

```

E:\AppA\maven> mvn test

↓
executes upto
test-phase of
default-
life cycle.

*: Maven generated content

```

62 // for building and deploying web application
63 -----pom.xml-----
64 <project xmlns="http://maven.apache.org/POM/4.0.0"
65   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
66   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
67     <modelVersion>4.0.0</modelVersion>
68     <groupId>org.durga</groupId>
69     <artifactId>JspApp</artifactId>
70     <packaging>war</packaging> (war, jar, pom etc) what we are expecting when delivery the proj.
71     <version>1.0</version>
72     <name>JspApp Maven Webapp</name>
73     <!--<url>http://maven.apache.org</url>-->
74     <!--<dependencies>
75       <dependency>
76         <groupId>junit</groupId>
77         <artifactId>junit</artifactId>
78         <version>3.8.1</version>
79         <scope>test</scope>
80       </dependency>
81     </dependencies>-->
82     <build>
83       <finalName>JSPApp</finalName>
84       <directory>${project.basedir}/test</directory>
85       <outputDirectory>${project.build.directory}/classes</outputDirectory>
86       <sourceDirectory>${project.basedir}/allResource</sourceDirectory>
87
88     <plugins>
89       <plugin>
90         <groupId>org.apache.maven.plugins</groupId> company name (we can gather from https://repoMaven https://maven.apache.org/repo)
91         <artifactId>maven-compiler-plugin</artifactId>
92         <version>2.3.2</version>
93         <executions>
94           <execution>
95             <phase>compile</phase> refer page no: 301.
96             <goals>
97               <goal>compile</goal> refer page no: 304.
98             </goals>
99           </execution>
100         </executions>
101       </plugin>
102       <!-- compiles all the java files of allResource folder. -->
103       <plugin>
104         <groupId>org.apache.maven.plugins</groupId>
105         <artifactId>maven-antrun-plugin</artifactId> refer page no: 304.
106         <version>1.3</version>
107         <executions>
108           <execution>
109             <phase>generate-resources</phase> refer page no: 300.
110             <goals>
111               <goal>run</goal> refer page no: 304.
112             </goals>
113             <configuration> -->
114               <tasks>
115                 <copy todir=".src/main/webapp"> copy index.jsp to webapp folder.
116                   <fileset dir="allResource">
117                     <include name="index.jsp"/>
118                   </fileset>
119                 </copy>
120                 <copy todir=".src/main/webapp/WEB-INF"> copies web.xml file to WEB-INF folder
121                   <fileset dir="allResource">
122                     <include name="web.xml"/>
123                   </fileset>
124                 </copy>
125               </tasks>
126             </configuration>
127           </execution>
128         </executions>
129       </plugin>
130     </plugins>
131   </build>
132 </project>

```

What we are expecting when delivery the proj.
(like creating war file).

optional

company name (we can gather from ~~https://repo~~^{Maven} https://maven.apache.org/repo)

refer page no: 304

refer page no: 301.

refer page no: 304.

compiles all the java files of allResource folder.

refer page no: 304.

refer page no: 300.

refer page no: 304.

copy index.jsp to webapp folder.

copies web.xml file to WEB-INF folder
from allResource folder.

```

123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162

```

123
 124
 125
 126
 127
 128
 129
 130
 131
 132
 133
 134
 135
 136
 137
 138
 139
 140
 141
 142
 143
 144
 145
 146
 147
 148
 149
 150
 151
 152
 153
 154
 155
 156
 157
 158
 159
 160
 161
 162

</fileset>
 </copy>
 <copy todir=".target/classes">
 <fileset dir="allResource">
 <include name="DBConnector.java"/>
 </fileset>
 </copy>
 <copy todir=".src/main/java">
 <fileset dir="allResource">
 <include name="DBConnector.java"/>
 </fileset>
 </copy>
 <copy todir=".src/main/webapp/WEB-INF/lib">
 <fileset dir="allResource">
 <include name="classes111.jar"/>
 </fileset>
 </copy>
 </tasks>
 </configuration>
</execution>
</executions>
</plugin>
</plugins>
</build>
</project>

copied DBConnector.java file to classes folder from allResource folder.
 copied DBConnector.java file to java folder from allResource folder.
 copied classes111.jar file to WEB-INF/lib folder.
 default goal is "deploy" - for this appln.
 while configuring the plugin if the goal phase is not specified default goal will be executed at tool supplied phase

E:\App\ Maven1
 AllResource
 pom.xml
 src*
*
 test*
*
 index.jsp
 DBConnector.java
 classes111.jar
 web.xml
 JSPApp.war*
*

*: They will be generated automatically by using some plugin's in Maven tool.

E:\App\ Maven1> mvn tomcat:deploy

→ Maven Plugin can prepare the war file of web app, only when the resources are arranged according to common directory structure.

Maven 3.0.3

Download & Installation:-

- Download Maven 3.0.3 (Binary zip) from
<http://maven.apache.org/download.html>
- Extract it to your choice drives or directory(Say C:)
- Set the following variable in the environment variable
 - path=C:\apache-maven-3.0.3\bin; othervalues
 - JAVA_HOME=C:\Program Files\Java\jdk1.6.0_11
- Open CMD and type the following
 - C:\>mvn -version
- You will get java version, and maven version, which shows maven is installed successfully in your machine

What is Maven?

It is a build tool that can be used for building and managing any Java-based project.

Maven allows a project to build using its project object model (POM) and a set of plugins that are shared by all projects using Maven, providing a uniform build system.

As a developer, we need to write the entries on a file called POM.xml, which contains all the information about

- which project to build(jar,ear,war)
- how to build the project(where are the source file, where it needs to be copied)
- what are the dependent jar file needed to build the project, and whether that jar file is needed only to compile the resources or needed to execute also

MAVEN REPOSITORY (Local Repository).

Maven maintains a repository where all the dependencies of the project will be stored. Dependencies means needed jar files and plugins

By default the repository will be created under "C:\Documents and Settings\<Your_User>\.m2" directory

The repository will be created after you execute pom.xml or any plugins

You can change the location of the repository by modifying C:\apache-maven-3.0.3\conf\settings.xml

<localRepository>d:/mavenRepository</localRepository>
Will set the repository in D:

Life Cycle of Maven

Maven has 3 Life Cycle. Each Life Cycle contains number of Phases

- Clean
- default
- site

Clean:-This Life Cycle contains 3 phases

Pre-clean	Executes processes needed prior to the actual project cleaning
clean	Remove all the file generated in previous build
Post-clean	Executes processes needed to finalize the project cleaning

Default: This Life Cycle contains 23 phases

1	validate	validate the project is correct and all necessary information is available.
2	initialize	initialize build state, e.g. set properties or create directories.
3	generate-sources	generate any source code for inclusion in compilation.
4	process-sources	process the source code, for example to filter any values.
5	generate-resources	generate resources for inclusion in the package.
6	process-resources	copy and process the resources into the destination directory, ready for packaging.
7	compile	compile the source code of the project.
8	process-classes	post-process the generated files from compilation, for example to do bytecode enhancement on Java classes.
9	generate-test-sources	generate any test source code for inclusion in compile
10	process-test-sources	process the test source code, for example to filter any values.
11	generate-test-resources	create resources for testing.
12	process-test-resources	copy and process the resources into the test destination directory.
13	test-compile	compile the test source code into the test destination

DURGA SOFTWARE SOLUTIONS

MAVEN

		directory
14	process-test-classes	post-process the generated files from test compilation, for example to do bytecode enhancement on Java classes. For Maven 2.0.5 and above.
15	test	run tests using a suitable unit testing framework. These tests should not require the code be packaged or deployed.
16	prepare-package	perform any operations necessary to prepare a package before the actual packaging. This often results in an unpacked, processed version of the package. (Maven 2.1 and above)
17	package	take the compiled code and package it in its distributable format, such as a JAR.
18	pre-integration-test	perform actions required before integration tests are executed. This may involve things such as setting up the required environment.
19	integration-test	process and deploy the package if necessary into an environment where integration tests can be run.
20	post-integration-test	perform actions required after integration tests have been executed. This may include cleaning up the environment.
21	verify	run any checks to verify the package is valid and meets quality criteria.
22	install	install the package into the local repository, for use as a dependency in other projects locally.
23	deploy	done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

Site: It contains 4 phases

Pre-site	executes processes needed prior to the actual project site generation
site	generates the project's site documentation
Post-site	executes processes needed to finalize the site generation, and to prepare for site deployment
Site-deploy	deploys the generated site documentation to the specified web server

Sample POM.xml

```
<project xmlns="...maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sathyasathya</groupId>
  <artifactId>JspApp</artifactId>
  <packaging>war</packaging>
  <version>1.0</version>
  <name>JspApp Maven Webapp</name>
  [our organization / our project related user defined content.]
  <build>
    <finalName>JSPApp</finalName>
    <directory>test</directory> [refer below.]
    <outputDirectory>test/classes/${pack}</outputDirectory> [→ where to store the o/p content.]
    <sourceDirectory>src</sourceDirectory> [source directory.]
    <plugins>
      <plugin>
        </plugin>
      </plugins> [Here we configure All the Maven supplied plugin's to perform various activities.]
    </build>
    <dependencies>
      <dependency>
        </dependency>
      </dependencies> [Here we configure various jar files related to our appn.]
    <properties>
      <!--user defined variable-->
      <pack>sathyasathya</pack> [user defined properties to define values & to mix those values in pom.xml.]
    </properties>
  </project>
```

Note: - groupId, ArtifactId, Version are just like package in java. They are given to uniquely identify each plugins, project, dependencies. For each dependency you can see a separate directory in the repository, holding that jar files.

Note:-

- All these phases are executed serially.
- To perform each operation, we need to take the support of a plugins.
- If you want to perform any operation in a particular phase, while configuring plugins specify the phase and goal, so that when maven executes that phase, it will perform the operation as specified by plugins.

- If you execute the pom.xml using mvn package, then it will execute all the phases from 1 to 17, but does not deploy the war.

Dependencies:

Some times our java class depends on certain jar files, In these case we can use those jar files using 2 ways

1> Manually arrange jar file in your web-resources

2> Just give information to Maven about the jar file, Maven will download the jars from local repository (If its not available in local repository, maven will download the same from internet repository)

To gather jar from maven repository (Local or Internet), we need to include the following code In pom.xml

```
<dependencies>
  <dependency> fixed
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

→ organisation name
→ represents jar file name
→ Represents version of jar
→ scope of the dependency

Scope:

It defines how project uses that dependency. It is of 6 type

1>compile:- This is the default scope, used if none is specified. It means the jar file is needed to compile the resources, and hence added to classpath dynamically.

2>Provided:- This indicates that the jar file will be provided by underlying server/container /JDK at runtime.

E.g:- If resource is servlet/jsp, Maven needs those jar only to compile those resources, while execution those jar will be provided by Servers.

3>runtime:- This indicates that the dependency is not required for compilation, but is for execution. It is in the runtime path but not the compile classpath.

4>test:- This scope indicates that the dependency is not required for normal use of the application, and is only available for the test compilation and execution phases.

5>system:- This scope is similar to provided except that you have to provide the JAR which contains it explicitly. The artifact is always available and is not looked up in a repository.

6> import:- (only available in Maven 2.0.9 or later)

This scope is only used on a dependency of type pom in the <dependencyManagement> section. It indicates that the specified POM should be replaced with the dependencies in that POM's <dependencyManagement> section. Since they are replaced, dependencies with a scope of import do not actually participate in limiting the transitivity of a dependency.

List of Important Plugins and goals

Plugin Name	Goals Available	Purpose
maven-compiler-plugin	compile,testCompile	Compile java file and testCase file
exec-maven-plugin	exec,java	Compiles and executes java classes
maven-antrun-plugin	run	Execute ant tasks
tomcat-maven-plugin	Start,stop,deploy, redeploy,undeploy	Deploys the project into tomcat server
jboss-maven-plugin	--do--	Deploys the project into jboss

How to Gather dependency details of any jar file

Search for your jar here <http://mirrors.ibiblio.org/pub/mirrors/maven/>

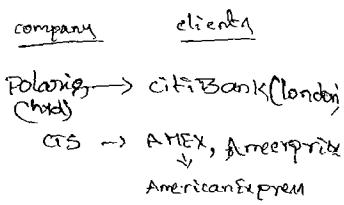
Navigate inside , search and open pom.xml, which contains dependency details

Info About all the plugins <http://mvnrepository.com/plugins.html>

REALTIME TOOLS

4/7/13

- (1) Project Release process in Realtime.
- (2) Log4j (Logging for Java).
- (3) Ant (Another Neat Tool).
- (4) Maven (Alternate for Ant tool).
- (5) Junit & Http Unit.
- (6) CVS.
- (7) SVN.
- (8) Installer (NSIS).
- (9) Converting Java Appn to an exe file.
- (10) Encrypting .class file & preventing from its Decompilation.
- (11) Encrypting Passwords.
- (12) Debugging Java Appn using Jdb tools & IDEs.



Testing Valulabs
AppLab

webx
Teamviewer
sify

Project Release Process :

- During project release & after project release the SW company maintains 2 teams.
 - (i) Onsite Team
 - ⇒ Workers at client organization location to receive, install & maintain the project.
 - ex: Team at ~~██████████~~ Citibank London
 - (ii) Offshore Team
 - ⇒ Team at SW company supporting onsite team.
 - ex: Team at Polarix hyd

By using Telnet Windows machine communicate with integrated Machine

Offshore Location (POLARIS) (hyd).

Onsite Location (citiBank) (London).

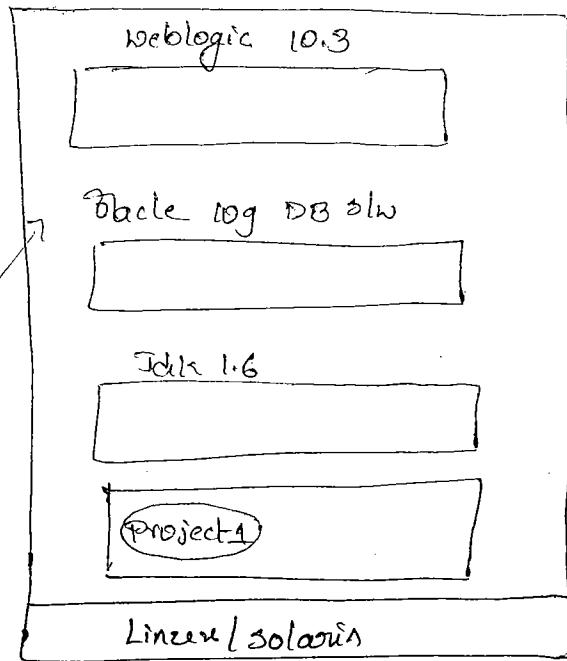
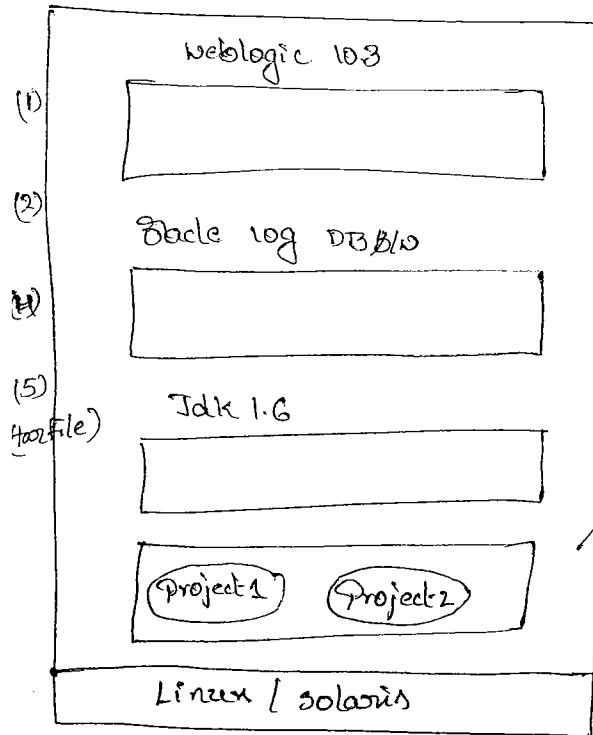
Integrated Machine / Development Box

contracts mail to
(11) company.

(3)

production Box

(9)



FTP Appln

UAT → User Acceptance Test.

tar file → Table archive it is similar to zip file in windows.

→ All Project 1 in B/w company will be maintained in a Integrated machine or development box. This generally runs in Linux / solaris environment.

→ The Machine in client organization location where the released project will be installed is called an Production box.

With respect to the diagram

①: On released Project both Pinhole Testing & 3rd Party testing will be completed

- ②: onsite team will be formed & will be sent to onsite location (client organization location).
- ③: This onsite team keeps Production box ready having the same setup of Integrated Machine. (including versions).
- ④: offshore team will be formed at SW company.
- ⑤:
 - Tool file will be prepared based on the project of Integrated Machine
 - ↓
 - Taped Archive

having all the documents of the Project.
- ⑥: Offshore team prepares dispatch central document having clear-cut instructions for onsite engineers to receive & install Project.
- ⑦: P.M prepares release mail having description about the Project to be released. & sends that one to multiple parties like HR Manager, P.L, BDM, Delivery Manager, client Representative & etc.
Business Development Manager
- ⑧: Offshore team uses FTP appn to send the project to client organization having DC document.
- ⑨: onsite team follows the clear-cut instructions of DC document & installs the Project in Production box.
- ⑩: onsite engineer creates few dummy users in the Project & allows client organization people to test the Project. Then it is called UAT (User Acceptance Test).
- ⑪: On successful installation of Project the client organization sends congrats mail to SW Company.

There are 2 types of Project Release

(i) Main Release

(complete project will be released).

(ii) Patch Release

(partial project & the code related bugfixing & enhancements will be released).

Ex: windows xp contains main release.

Service pack1 , Service pack2 releases are called patch releases.

Note: During Patch release all the above discussed steps will be there except step7 & step11.

5.7.1.8

Logging:

→ The thing that keeps track of flow of execution in the appn is called log msg.

→ It is always recommended to record log msg's in different destination so that we can utilize them towards debugging of appn's & bug fixing.

→ Performing logging operations by using S.O.PU gives following limitations.

(1) Log's can't be written to different destinations like files, DB's except console monitor.

(2) Doesn't allow to categorize the msg's.

(3) Doesn't allow to format the msg's.

(4) Possibility of losing msg's after certain period.

(5) S.O.PU writes log msg's as single threaded process. so it can write multiple log msg's at a time.

To overcome these problems any Thirdparty supplied (or) other logging api's:

Logging API (Jdk)

Annotations (Jdk)

Apache commons - logging (Apache)

Log4j (Apache) (Internally uses commons - logging).
good to use.

Log4j:

Type: Logging API / tool / framework for Java.

Version: 1.4.x (compatible with Jdk 1.4+)

Vendor: Apache foundation.

OpenSource.

To download slf: Download slf as zip file from www.apache.org.

Jar file that represents log4j API: log4j-<version>.jar

→ ApplicationServer slf like weblogic gives jar file representing Log4j API
(log4j-1.2.8.jar) By adding this jar file to classpath we can
configure log4j in our computer.

Log4j Features:

- Allows to categorize log msg's.
- Allows to ~~write~~ format log msg's.
- Allows to write log msg's to different destinations.
- Allows to specify priority levels for log msg's.
- Allows to filter log msg's while retrieving them.
- can write log msg's simultaneously (log4j is not thread-safe).

Thread-safe is advantage to log4j. so multiple threads can share the log msg simultaneously.)

- log4j is the enhancement of apache commons logging api. simplifying the programming of logging.
- Most of the Technologies & services still are using apache commons logging api, log4j internally to generate log msg's.
- ~~log~~ the log4j generate log msg's are very useful for offshore engineers while fixing bugs. These offshore engineers collect log msg's from onsite engineers.
- log4j allows to create 5 categories of log msg's. having priority levels debug < info < warn < error < fatal.
- Use debug level to place the regular conformation statements.
- Use info level to place important actions (like connection obj creation), related log msg's.
- Use warn level to place log msg's related to deprecated api, poor api utilization.
- Use error level while handling known exceptions in the catch blocks.
- Use fatal level while handling unknown exceptions in the catch block.
- There are 3 important obj's in the programming of log4j.
 - (i) Logger obj.
 - (ii) Appender obj.
 - (iii) Layout- obj (formatter obj).

{imapAppend()
smtp
↳ send log msg to email.

Logger obj:

- Enables logging operation on certain java class.
- provides methods to generate different levels of log msg's.
- Allows to specify logger level to filter log msg's while retrieving log msg's.

Logger logger = Logger.getLogger(Selectient.class);

↓
logger obj

↓
Static factory method.

↳ the class on which you want to enable logging.

logger.debug("msg1"); // debug level log msg.

logger.info("msg2"); // Info " " "

logger.warn("msg3"); // warn " " "

logger.error("msg4"); // error " " "

logger.fatal("msg5"); // fatal " " "

to filter (S)

retrieve the log

msg specifying

the logger levels.

logger.setLevel((Level)Level.INFO);

↓
specifies logger levels to retrieve the log msg's.

- If the given logger level is "INFO" then it retrieves only Those log msg's whose logger levels are greater than & equal to Info.

Appender obj:

- It specifies the destination explicitly to record

6/7/13

Procedure Logger obj to Perform logging operations:

|| get logger obj

```
Logger logger = Logger.getLogger(SelectTest.class);
```

|| create layout obj

```
SimpleLayout layout = new SimpleLayout();
```

|| create Appender obj

```
ConsoleAppender appender = new ConsoleAppender(layout);
```

|| add Appender obj to logger obj.

```
logger.addAppender(appender);
```

|| specify the logger level to retrieve the log msgs.

```
logger.setLevel((Level)Level.INFO);
```

|| write log msgs

```
logger.debug("msg1");
```

```
logger.info("msg2");
```

....

Note: we always place log msgs along with Application code.

→ logger API means org.apache.log4j & its subpackages.

→ For example appln on Log4j refer Page no: 277 to 281 of the booklet.

→ The default logger level to retrieve the log msgs is "DEBUG".

→ For detailed information on Log4j refer page nos: (18) to (44) of current booklet.

→ logger.setLevel((Level)Level.OFF);

↳ used to stop logging operations on the appln.

- The standard slogan of the industry is don't hardcode any values ⁱⁿ our app. That are changeable in the feature. It is recommended to pass these values to app. → From outside the app. by using either Properties file or XML file.
- log4j supports both properties file & XML file to specify their configuration.
- For this we need to use PropertyConfigurator & DOMConfigurator class.

Q7/3

→ property names fixed, values are changeable

ex: Log4j.rootLogger = INFO, S → alias name.
 ↓ ↓
 Property name value

- Most regularly used Appender, Layout classes of realworld are DailyRollingFileAppender, patternLayout.
- While supplying Log4j configuration from XML file we need to use DOMConfigurator class.

→ For example App. refer pgno: 281 of booklet.

- When Java web app. uses Log4j API, the Log4j API related jar file should be added to classpath & should be added to WEB-INF/lib folder of web app.
- Server based app. except the Log4j properties based properties file, XML file in underlying server home directly & also generates log file in the same home directory.

SLF4J: (Simple Logging facade for Java).

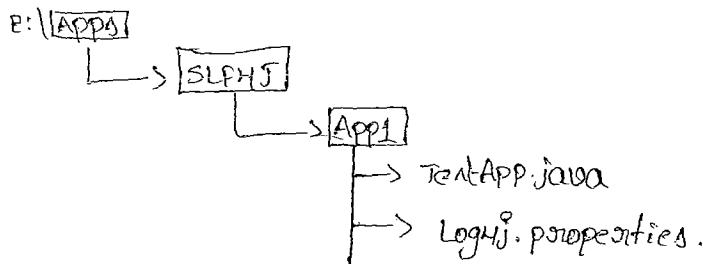
- It is a framework, which internally uses other logging API's based on the dependent jar files we supply.
- It can use the following API's internally.
 - Log4j.
 - JDK Logging API.
 - Apache-commons Logging API.

→ Then slf4j comes as SLF4J-1.7.5.zip file & the fatal level of logging has been removed in the SLF4J programming.

→ For related info on SLF4J refer page no: (42) & (43) of booklet.

Example Appn on SLF4J:

Deployment Directory Structure:



Step(i): Install SLF4J slf (SLF4J-version.zip file)

Step(ii): keep following jar files in the classpath.

- (i) SLF4J-api-1.7.5.jar
- (ii) SLF4J-log4j12-1.7.5.jar } get from slf4j slf,
- (iii) log4j-1.2.15.jar → from Log4j slf.

Log4j.properties:-

log4j.rootLogger = INFO, R

log4j.appenders.R = org.apache.log4j.RollingFileAppender

log4j.appenders.R.File = ~~d:\~~ d:\mylog.html

log4j.appenders.R.layout = org.apache.log4j.HTMLLayout

TextApp.java :

```
import org.slf4j.Logger;
public class TextApp {
    private static final Logger logger = LoggerFactory.getLogger("Text");
    public static void main (String[] args) {
        logger.debug("This is debug");
        logger.info("This is Info");
    }
}
```

```
logger.warn("This is warn");
logger.error("This is error");
```

3

3

Note: The above steps appn uses Log4j internally to generate the log msg's.

9/7/18

ANT: (Another Neat Tool)

→ Instead of executing certain commands in a sequence for multiple times by remembering their names & folder if it is ~~not~~ recommended to place them in a batch file & run the batch file for multiple times.

ex: run.bat

```
date
time
ver
>run
>run
>run
```

→ similarly to automate the whole process of appn development, deployment & execution operations we can use build tools. Like ANT, MS-build, Maven & etc.

→ ANT is a build tool that can automate the complicated & repetitive tasks of appn development, deployment & execution.

→ This build tool is very useful, towards developing & executing complex Appn's & towards module level integration & project level integration.

→ we write this automation script of ANT tool by using XML file called build file.

→ ANT uses build.xml as default build file & also allows to specify other filename as build filename.

→ To configure ANT tool keep the location of ant.bat file in path environment variable.

variable name: PATH.

variable value : `c:\oracle\middleware\models\big.apache.ant-1.7.0\bin`
 ↓
 weblogichome <other values>

→ ok → ok → ok.

with this ANT tool configuration successfully.

ANT:

Type: build tool for Java

Version: 1.7.*

Vendor: apache foundation

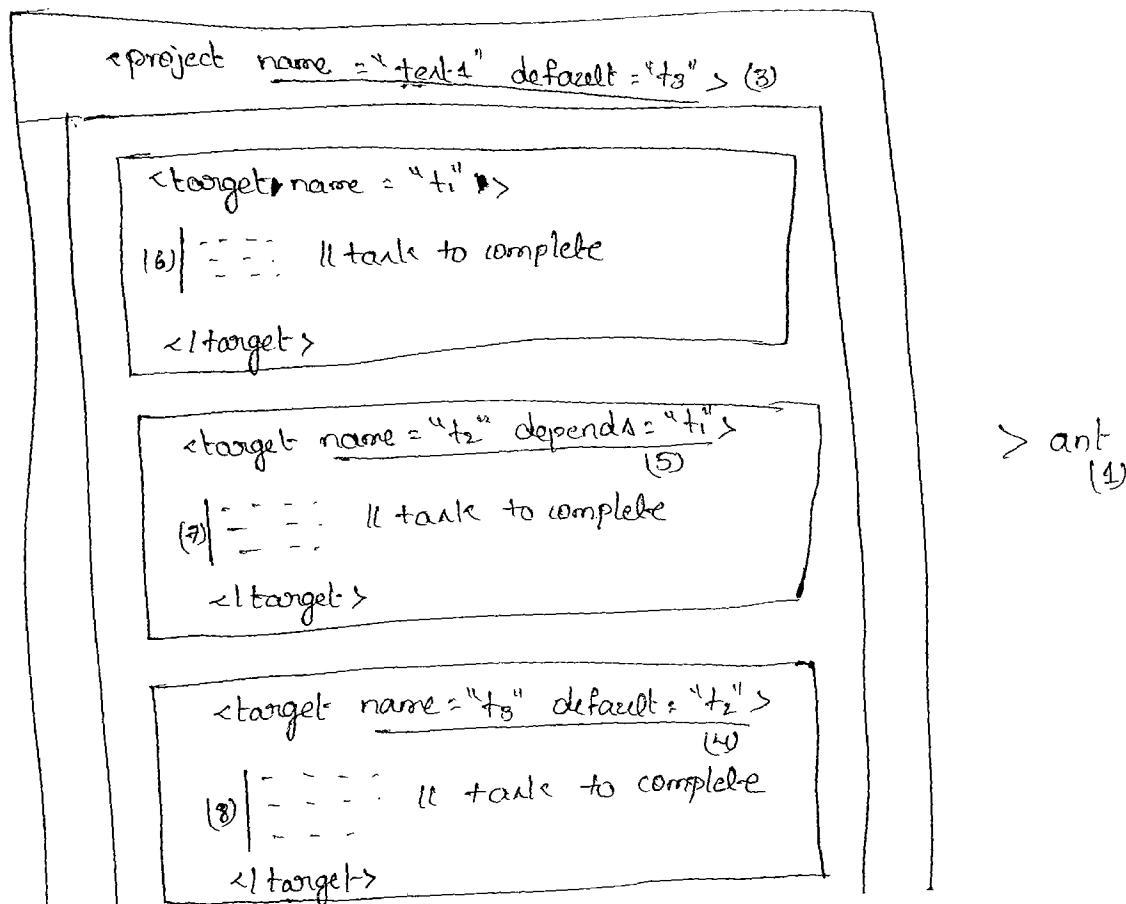
Open source SW

To download SW: Download it as zip from www.apache.org website.

→ For related info on ANT tool refer Pages no: ② to ⑯ of the booklet.

Structure Of Build file:

build file (xml file) (build.xml) (2)



- Each build file contains one project. Each project contains one or more targets specifying their dependency order.
- Each target contains one or more tasks to complete, to complete these tasks we use the ANT supplied <xml> tags as like <Makefile>, <javac>, <javas & <echo> (or) <mkfile>
- For example Applying on ANT tool refer page no: 282 to 284
- cmd prompt > ant (takes build.xml file of current directory as build file).
- cmd prompt > ant -buildfile myfile.xml (takes myfile.xml of current directory as build file).
- We can make Ant tool executing only specific targets of build file.
> ant -buildfile myfile.xml compile
 ↳ It executes only compile target of build file.
- We can give inputs to build.xml file from properties file.
Ex: myfile.properties
 src = .
 pack = pack1
- We can specify properties file in build file by using
 <property file = "myfile.properties" /> (In myfile.xml)
- *→ If you keep same property names in both xml, properties file having two different values. The value kept in properties file will be effected.
 ↳ <property file = "myfile.properties" />
 <property name = "pack" value = "P" />

Q → what is the difference b/w working with batch file & working with Ant build tool?

Ans: Both can be use to automate build process (development, deployment & execution). But in batch file we can't specify dependence b/w operations, but it is possible with Ant tool.

10/7/13

→ In Jdk env, tool.jar is added to classpath.

Directory Structure:

Refer appn on Page no: (282)

E:\ Apps

 └→ AntApps

 └→ ant-web

 copy (57-61)

 |

 DBSelect.java - copy (68) - - -

 lib
 └→ aidloc4u.jar - copy (51-55) - - -

 query.html

 web.xml

 build.xml

 build (40)

copy

(45-49)

 |

 |

 |

 |

 |

 |

 |

 |

 |

 |

 |

 |

 |

 |

 |

 |

WEB-INF (41)

classes (42)

 └→ DBSelect.class

 lib (43)

 └→ aidloc4u.jar

 web.xml

DBAPP.war (71)

 └→ DBApp.war (71)

Note: Netbeans IDE internally uses ant build tool while running the appln. upto myEclipse 6 ant tool is utilized to run the appln from myEclipse 8 onwards "Hadoop" is used.

→ To run the build file Through Eclipse/ myEclipse IDE's

refer page no: (12) to (14) of the booklet.

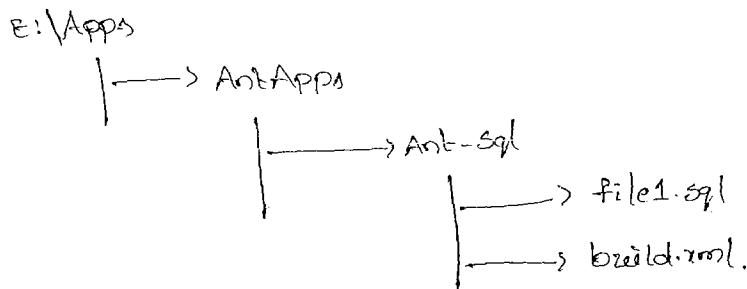
→ How to Perform DB operation directly from build file?

11/7/13

→ To automate various ~~complicated~~ & repetitive operations of DB Designing

& DB data creation we ~~can~~ can write build script using Ant tool.

That we need to use `<sql>` tag as shown below.



file1.sql:

1 insert into student values (908,'maja','hyd');
2 select * from student.

build.xml:

```
<project name="test" default="t1" basedir=".">
<target name="t1">
<sql driver="oracle.jdbc.driver.OracleDriver"
      url = "jdbc:oracle:thin:@localhost:1521:ORCL"
      user="scott"
      password="tiger"
      src = "file1.sql"
      print="yes"
      output = "output.txt">
<classpath><path element location="c:/Oracle/product/10.2.0/db_1/jdbc/lib/ojdbc14.jar"/></classpath>
</sql>
</target>
</project>
```

- To fix the problem of ^{bugs in code} issues first we need to analyze the problem & for that we can use debugger which performs debugging by showing flow of execution in the code.
- Debugging process doesn't solve or fix the problems it allows to analyze the problems.
- Debugging is useful for following activities.
- ⇒ To analyze code flow.
 - ⇒ To analyze bugs or errors.
 - ⇒ To know internal flow of appn's.
 - ⇒ To enable watch monitoring on the application data.
 - ⇒ To know the container level underlying execution & etc.
- To perform debugging in CUI environment use the Jdk supplied "jdb" tool.
- To perform debugging in IDE environment, use the IDE supplied built-in debugger.
- While performing debugging we need to set break points.
- Break point is logical position in debugging environment from where the debugging starts.
- Two types of break points.
- (1) Line Break Point (Based on line no's).
 - (2) Method Break Point (Based on Method name).

Example Appn :

```
1 TestAPP.java  
public class TestAPP  
{  
    public void m1()  
{
```

```

s.o.p ("TestApp : z());
for (int i=1; i<25; i++)
{
    s.o.println ("hello -->" + i);
}
}

```

```

public void y()
{
    s.o.println ("TestApp : y()");
    int a=10;
    int b=20;
    s.o.p ("sum is " + (a+b));
    z();
}

```

```

public void x()
{
    s.o.println ("TestApp : x()");
    y();
    s.o.println (new java.util.Date());
}

```

```

public void main (String args[])
{
    try {
        Thread.sleep (20000);
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}

```

s.o.println ("start of TestApp : main()");

TestApp t1 = new TestApp();

t1.x();

s.o.println ("end of TestApp : main()");

> javac TestApp.java

> gdb

initializing jdb...

> run TestApp

run TestApp

*1: Required to get the chance
of setting breakpoints once
the appln execution is
started.

> VH started:
> Stop in TestApp.java
set breakpoint TestApp.java
> Start 0
:
main[1] step

> :
main[1] step
> :
main[1] other commands

Step → executes current instruction
Step up → Executes until the current method returns to its column

Step → executes current line.

next → step one line (step over calls)

For line breakpoints:

Stop at TestApp: 20
↓
classname line no.

12/3/13

② Procedure to perform java appn debugging & Add Watch operations in ~~IDE~~

NetBeans IDE:

Step1: Create Java project & add one Java appn to source packages folder.

TestApp.java:

Same as Previous appn

Step2: Create Break point on any line no that contains code.

(By double click on line no break point is created on that line no.)

Step3: Debug the appn

right click in TestApp.java → ~~Run~~ file

OR stepinto, stepover & other options of debug tool bar.

→ To enable watch on variables & expressions.

Windows menu → Debugging → variables → create new watch → add variables ↓
like a, b

↓
Expressions

↓
like a, ab, abc etc

→ while performing debugging we can also enable analysis on callstack,
loaded classes, Breakpoints, Threads & etc.

Procedure to debug java web app by using NetBeans IDE:

Step1: Create Java web project.

Step2: Add JSP program to the WebPages folder.

Right click on WebPages folder → JSP → JSP filename: Test.

Test.jsp:

```
<b> how are u </b>
<br>
<% out.println("Output P");%
    for (int i=1; i<=10; ++i)
        {
            out.println("Durga" + i);
        }
%>
```

Step3: Create Break point in JSP program (double click on any line no Breakpoint will be created successfully).

Step4: Debug the JSP program.

Right click on Test.jsp on project → Debug → Server side Java →
Debug.

Procedure to debug javaScript code using NetBeans IDE:

Step 1: Add html file to webpages folder of the project.

Right click on webpages folder → new → HTML file name: Demo

Demo.html :

```
<html>
  <head>
    <script language="javascript">
      for (var i = 1; i <= 5; ++i)
        document.writeln("Hello " + i);
    </script>
  </head>
  <body>
    From Demo.html
  </body> </html>
```

Step 2: Debug the appn. (Perform this step after creating Break point).

Right click on webProject → Debug → Open browser side Java with client side

JavaScript in the selected browser → Debug

Procedure to Perform Debugging of java appn in MyEclipse IDE:

Step 1: Create Java Project having Java appn.

Step 2: Create Break point.

Step 3: Debug the appn.

Right click in TestApp.java → Debug As → java appn.

→ To enable AddWatch operation on variables & Expressions use window menu →

ShowView → Variables, Expressions option.

Procedure to perform debugging on Java web applet using MyEclipse IDE.

Step1: Create web project

New → web project → Project Name: TestWebProj1 → finish.

Step2: Add some code in default index.jsp

index.jsp:

```
<body> hello how are u </body>  
<br><br>  
<% for (int i=1; i<=5; ++i){%>  
    { out.println("Durga" + i);  
}%>
```

Step3: Create breakpoint in source code.

Step4: Configure any server with MyEclipse IDE that supports Debugging.

Window Hence → Preferences → MyEclipse → servers → Tomcat →
Tomcat 6 → Enable → Tomcat home directory: D:\Tomcat 6.0
→ Apply → ok.

Step5: Debug the appn.

Right click on Projects → Debug as → MyEclipse server appn → Tomcat 6.x
→ OK.

Step6: Test the project

url: http://localhost:2020/TestWebProj1/index.jsp

→ Every version of Eclipse is identified with nickname like Eclipse Helios, Eclipse Glynnaide, Eclipse Indigo & Eclipse Kepler etc

→ Debugging of Java appn & web appn in Eclipse IDE is much similar to MyEclipse IDE.

UNIT Testing (JUNIT) (written by Kent Beck, Gamma)

Test-Unit Testing

→ For basics of unit testing, Testcase, peer testing, & etc refer page no's 45

→ 49

13/11/13

Junit:

Type: Unit Testing framework.

Version: 3.x (compatible with jdk1.4+).

4.x (compatible with jdk1.5+). (Supports Annotations).

Created: KentBeck, Erich Gamma.

Vendor: To download site: download it as zip file from www.junit.org website.

Installation: Extract the zip file to a folder.

Jar file that represents junit api: junit-<version>.jar
(<junit-home>-folder).

Example Test Scenarios:

Scenario1: Testing the login App

Testcase1: Test with valid data.

Testcase2: Test with invalid data.

Testcase3: Test with no data.

Scenario2: Testing the simple Payment logic

Testcase1: Test with given values.

$$P=10000, t=2, \pi=2.$$

Testcase2: Test with given values.

$$P=40000, t=10, \pi=3.$$

Testcase3: Test with zero's.

$$P=0, t=0, \pi=0.$$

Testcase4: Test with -ve values.

Testcase5: Test with no data.

- Before arrival of Junit or other unit testing tools people use to write their testcases on piece of paper manually. But in Junit we can write them test-cases in a class in the form of `testXXX()` methods. In these methods we can compare expected results with generated results by using `assertEquals()` method.
- For related info on testcases & assertXXX() methods refer Page no's 50 to 58.
- Every Junit-based unit-testing contain 3 resources.
 - (i) Actual class to test.
 - (ii) class having Test-cases.
 - (iii) Test-Suite to run the Test-cases.

Procedure to Perform Junit-Based Unit Testing through manual Process:

Step1: Keep the above said 3 resources ready (refer appn(1) of Page no's: 285 & 286).

Step2: Make sure that `Junit-home\junit-version.jar` file is added to classpath.

Step3: compile all .java files & execute Test-Suite class.

→ We can avoid developing Test-Suite class separately by executing Test-Runner class directly from command prompt.

> `java junit.textui.TestRunner TestMath`

Procedure to develop Junit-based unit testing application by using Eclipse IDE:

Step1: Create Java Project.

Step2: Add Junit-3 libraries to the project.

Right click on project → BuildPath → Add Libraries → Junit →

Junit Library version: Junit-3 → Finish.

Step3: Develop actual class to test.

Now →

payment.java:

Public class Payment

{ public float calcIntAmt (float p, float t, float r) throws Exception

{ if (p<0 || t<0 || r<0)

 throw new Exception ("Provide proper inputs");

 float famt = (p*t*r) / 100.0f;

 return famt;

}

}

Steps: Add Test-case class to project.

Right click on src folder → New → Other → Java → JUnit →

JunitTestCase → Next → ② New Junit 3 test, name: MyTests,

setUp(), tearDown(), class under test: payment → Next

payment

calcIntAmt →

MyTests.java:

import junit.framework.TestCase;

Public class MyTests extends TestCase

{ Payment p;

Protected void setUp() throws Exception

{ p = new Payment();

Protected void tearDown() throws Exception

{ p=null;

} TestCase1 (with valid data)

Public void testCalcIntAmt() throws Exception

{ assertEquals("From Testcase1", 4000.0f, p.calcIntAmt(10000, 2, 20));

}

} Test-case2 (with valid data)

Public void testCalcIntAmt2() throws Exception

{ assertEquals("From Testcase2", 8000, p.calcIntAmt(20000, 2, 20), 0.1f);

}

} Test cases (with invalid data)

Public void testCalcIntAmt2()

{ try

{ p.calcIntAmt(0.0f, 0.0f, 0.0f);

} catch (Exception e)

{ fail ("Testcase3 failed");

}

Step5: Add test-Suite.

Right-click on src folder → New → Other → Java → JUnit → JUnit-Test Suite → Next → Select MyTest → Finish.

Steps: Run the TestSuite class.

Right click in AllTest.java → JunitTest

Q) what is the difference b/w JUnit-Error & JUnit-Failure?

- JUnit-Error will be raised if Business method of actual class (fixtore) throw Exception during execution.
- JUnit-Failure raises when actual results are not matching with expected results.

16/7/13

- JUnit3 doesn't supports annotations based programming.
- JUnit4 supports annotations based programming.

@Test.

@Before.

@After.

@BeforeClass.

@AfterClass.

@Test (expected = < class name>).

@~~Test~~ (timeout = < some time>).

~~Test~~

@Ignore

→ For related info on these annotations refer page no: 57 of booklet.

Procedure to develop JUnit4 based appln by using MyEclipse IDE:

Step1: Create Java Project in MyEclipse IDE.

Step2: Add actual class to test- to the src folder. (payment.java same as previous's appln).

Note: Add following business method in Payment.java.

```

public Date generateSynDate( boolean flag )
{
    if (flag == true)
        return new Date();
    else
        return null;
}

```

Step3: Add Junit Libraries to the project.

Right click on project → BuildPath → Add Libraries → Junit → Junit Library version: [Junit] → finish.

Step4: Add Test-case to classes to the project.

Right click on the Project → New → Other → Java → Junit → Junit-test → Next → ① New Junit test, name: MyTests, select All the 4 methods, class under test: Payment → Next → Payment → finish.
 calculateAmount()
 generateSynDate()

MyTest.java:

```

Public class MyTests {
    static Payment p;
    @BeforeClass
    Public static void setUpBeforeClass() throws Exception
    {
        S.O..println("MyTests: setUpBeforeClass");
        p = new Payment();
    }
    @AfterClass
    Public static void tearDownAfterClass() throws Exception
    {
        S.O.P ("MyTests: tearDownAfterClass");
        p = null;
    }
    @Before
    Public void setUp() throws Exception
    {
        S.O.P ("MyTests: SetUp()");
    }
}

```

```

@After
public void tearDown() throws Exception {
    2. System.out.println("MyTests: tearDown()");
    3.
}

@SuppressedWarnings ("deprecation")
@Text-
public void testCalcIntraAmount() throws Exception {
    2. assertEquals(4000.0f, p.calcIntraAmount(10000.0f, 20), 0.5f);
    3.
}

```

3
4

Step5: Add TestSuite to the project.

Right click on project → new → other → Junit → JunitSuite → Next → finish. → Add following annotations on the top of ~~alltests~~ AllTests class.

```

import junit.framework.Test;
import junit.framework.TestSuite;
import org.junit.runners.Parameterized;
import org.junit.runners.Suite;

@Suite.SuiteClasses({MyTests.class})
public class AllTests {
    one method suite() will come here.
}

```

Step6: Run the test suite

Rightclick on the AllTests.java → Run as → Junit Test.

→ To perform unit testing on web appn we can use either Junit or HttpUnit.

→ HttpUnit is a separate SW to install it provides environment to create a simulator for browser window to send request & to gather result & to compare actual result's with expected result's.

→ P8 example appn on HttpUnit based unit testing after appln given in

Page no's: 286 & 287

→ writing web based unit testing TestApp1 web appn of 286 partly by using Junit & Jwebunit tool

Step1: Make sure that TestApp1 appn is in editing mode.

Step2: Create Java Project. Add all the jar files of <junit-home>/lib folder to the build path of the project. (i) Add junit 3 & junit 4 Libraries to the Project.

Step3: Add following additional jar files to the buildpath of the project.
(refer page no : 77).

Step4: Add Testcase class having logic to perform unit testing.
(refer page no: 78 @ 79).

Step5: Add TestSuite class & run the TestSuite class.

AllTests.java:

```
import junit.framework.Test;
```

```
@RunWith( Suite.class)
```

```
@suite . SuiteClasses (MyTeston.class)
```

```
public class AllTests {
```

```
}
```

↳ class having Test-cases.

17/7/13

CVS (Code Versioning)

→ In project development, it's always team work. That means all members of the team will want to share project resources & should modify them if needed. In this process there should be one some monitoring mechanism to keep track of various modifications done to various resources simultaneously & concurrently by various programmers. For this we need to use versioning mechanism to keep track of modifications.

Test.java (Original)

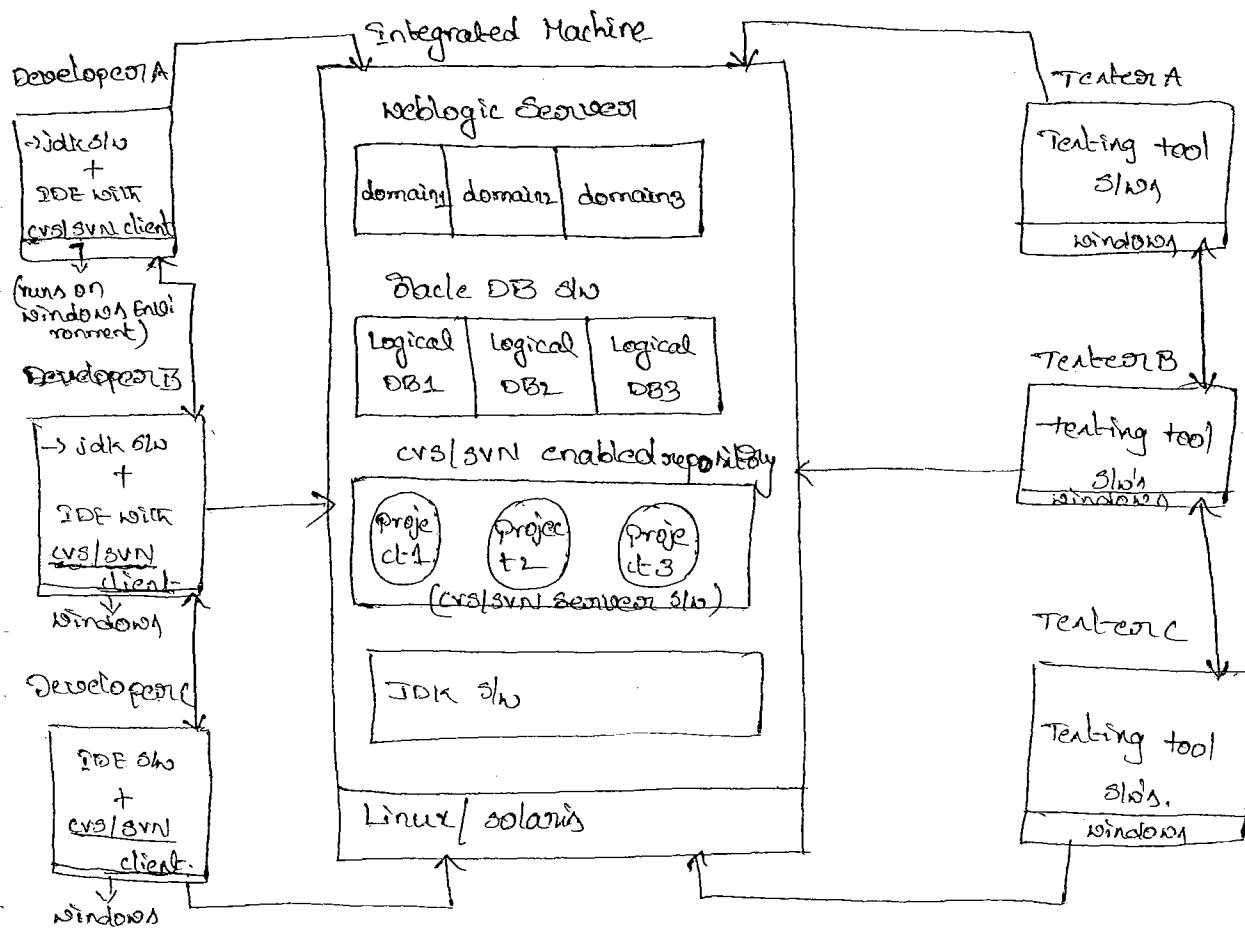
Test.java 1.1 (modification 1) (Done by siva).

Test.java 1.2 (modification 2) (Done by siva).

Test.java 1.3 (modification 3) (Done by siva).

Understanding SAM / SCM :

SAM - Site Asset Management , SCM → Site configuration Management.



- > Integrated Machine is the common ~~host~~ server machine in the company where all the projects & s/w's will managed. This machine generally runs in Linux & Solaris environment containing several s/w's with multiple domains, DB s/w's with multiple DB's & CVS server s/w with multiple repositories for multiple projects.
- The directory where each project will be saved in the integrated machine.
- > Set of administrators like DBA's, Linux & Solaris administrators, Appln server s/w Administrators together maintains integrated machine. These administrators are called librarians.
- > Developers, Testers machines will be there in windows environment & they interact with integrated machine through FTP appln & Telnet appln.
- > The process of taking content from CVS repository is called checkout operation.
- > The process of keeping content in CVS repository is called checkin / commit operation. For every checkin operation one ~~one~~ version will be generated for resource as discussed above.
- > Every IDE comes with CVS/SVN client s/w & ~~already~~ allows to perform both checkin, checkout operations.
- > TL & PL creates basic Project directory structure in a repository of integrated machine & asks all the developers to check out the project to their machine after adding more content to project & modifying existing resources. Each developer performs checkin operation back to repository.
- > When TL & PL points more versions for single resource. (like Struts configuration file) Then he merges all the versions into single version with proper analysis by using ~~the~~ awk, sed, diff & etc. comparison tools.

List of CVS Sevices soft's:

WinCVS

CVSNT

ClearCase

Tortoise & etc.

List of SVN Services soft's:

SVN → Sub Revision.

UbuntuSVN

VisualSVN

Tortoise SVN

Smart SVN & etc.

→ SVN is the enhancement of CVS soft.

Note:
→ Every Java IDE maintains CVS, SVN client-1.

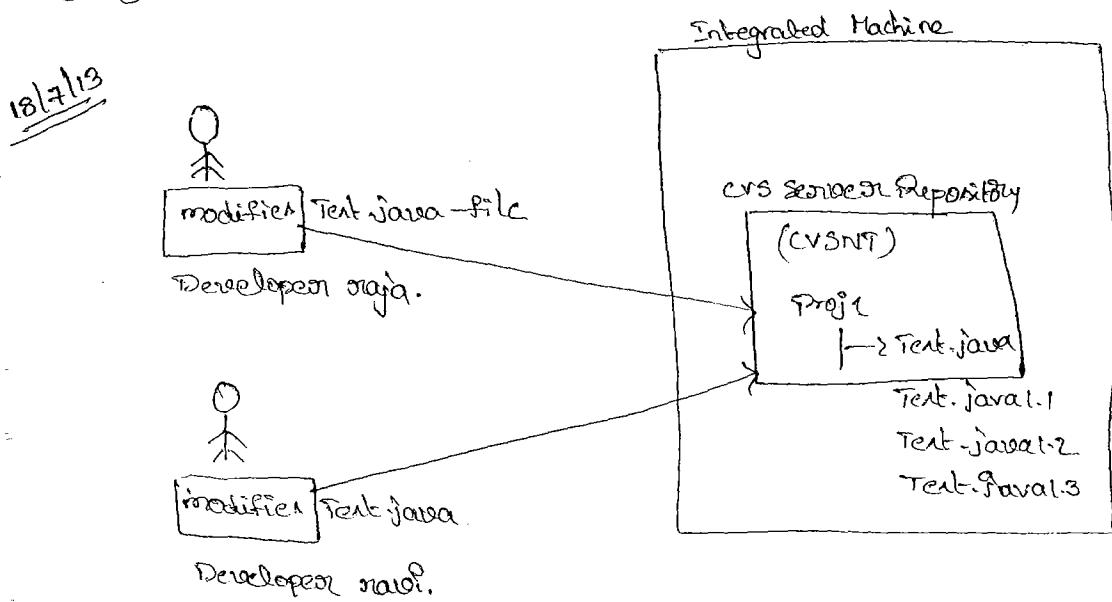
→ For related info of SCM refer Page No's: 82 to 84.

Procedure to Install CVSNT CVS Service soft & creating one repository

In That:

refer Page No: 84 to 91.

(Programmers are not responsible for this work)



Procedure to Perform checkin & checkout operations on cvs Repository based on the above diagram setup:

Step1: Create 2 windows user having names raja, mavi.

username pwd (Windows level user)

raja raja.

mavi mavi.

Step2: Launch MyEclipse IDE choosing separate workspace folder for raja user. (E:\workspace\raja)

Step3: Make sure that CVSNT service is in running mode.

Start → Programs → CVSNT → Service control panel → CVS Service: [start]

Step4: Configure the above CVSNT Repository with MyEclipse IDE (for raja user).

(a) Window Hence → Show view → others → CVS → Select CVS editor, repositories.

(b) Right click in CVS Repositories tab → New → Repository Location →

Host: localhost , Repository path: [DURGAREP] (created in previous class).

(Select from Repository tab of CVSNT SW), User: raja, Password: raja
→ finish.

Step5: Create Java Project in MyEclipse IDE. & add one .java file.

Test.java:

Public class Test-

```
{  
    public static void main(String[] args)  
    {  
        System.out.println("Hello");  
    }  
}
```

Step6: Share Project by keeping in repository.

Right click on project → Team → Share Project → select DURGAREP repository

→ Next → Next → Finish → Enter a comment for the commit operation.

pro4 - checkin

Step7: modify Test.java & Perform checkin/ commit operation

Right click on Test.java → team → commit → Enter a comment for the commit operation.

Test.java → change1

→ finish.

Step8:

Perform the following additional operations

(a) compare local Test.java with repository history.

Right click on Test.java → compare with → History.

(b) compare with Head version of repository.

The latest content placed in repository.

Right click on Test.java → compare with latest from Head.

(c) synchronize current file (Test.java) with repository.)

Right click on Test.java → team → synchronize with repository.

(d) Even though you look Test.java file content you can get it from repository.

Right click on Test.java → Replace with → History → select revision
(based on comment) → Replace.

(e) To see the history of a file having various versions

Right click on Test.java → team → show history.

Note: Perform all the above operations as Java user operations.

Step9: Launch another copy of Eclipse with different workspace for mavi user.
E:\maviravi

Step10: configure CVS Repository in their workspace for mavi user

Windows Menu → Show View → Other → CVS → CVS Repositories →
right click in repositories window → New → Repository Location →

location:

[Host: localhost, Repository path: /DURGAREP]

Authentication

[User: srujan, Pass: srujan] → finish.

Step 11: Make srujan user → checkout the project from repository.

file menu → Import → CVS → projects from CVS → select DURGAREP → next → ② use existing module → ^{select} proj1 → next → ③ checkout as a project in the workspace → next → Select HEAD → Finish.

Step 12: Perform few commit operations on Test.java from srujan user.

Step 13: Go to srujan workspace folder [His IDE window] & get updates done by srujan user.

Right click on test.java → team → update.

→ For above steps based screenshots refer page no's: (92) to (111).

SVN (Sub Versioning)

- SVN is complete alternate for CVS tool basic Source code Management.
The shell script used inside CVS SW are having multiple bug's, so to ~~keep~~
overcome those bug's, another versioning tool is given i.e. SVN.
- For differences b/w CVS & SVN refer supplementary handout given on 18/7/13
~~also~~
& refer page no's: 228 & 229.

19/7/13

- List of SVN Server SW.

Procedure to Install:

Step1: Download ubersvn SW by using following link.

www.wandisco.com/ubersvn/download

* also generate the key.

Step2: Install ubersvn (submit the above generated key).

Step3: Create one admin user choosing Username & Password. This is also
should be done properly Installation.

Step4: choose base url, ubersvn port no's & skip port during Installation.

To place projects in ubersvn Repository without using IDE we need to use
SVN SW.

Procedure to install tortoise SVN:

Step1: Download tortoise svn from following link <http://tortoisenv.net/downloads.html>

Step2: Install the .exe.

Procedure to use ubersvn & tortoise svn to gather to perform versioning operation:

Step1: Launch ubersvn

Start → programs → ubersvn → Launch ubersvn → Enter admin username & password (creating during installation).

Step2: Add user (programmer)

Users tab → add ... full name Raja U.W Raja, pwd raja, pwd: raja, Email raja@x.com → create.

Take one more user like above user tab process another user name: rai, Pwd rai@x.com → create.

Create directly acting as repository:

Step1: Repositories tab → add → Repository name dwrgarep1 → Next → Done

Step2: Create team & add users to team.

Teams tab → add → Team name: dteam, Repositories: dwrgarep1, click on Add button, user Raja add button, rai? click on add button → create.

Step3: Keep a project in ubersvn repository.

Right click on Project Folder (any technology based application) →

Tortoise SVN → Import url of repository <http://127.0.0.1:9880/dwrgarep1>

→ Import remote Project → checkin → ok.

↓
(Copy from Repository tab of ubersvn)

→ U-N: maja pwd: maja → ok.

step4: Take separate folder for maja & perform checkout operation on the Project.

E:\majawork\ (get into that folder) (open that folder) → Right click → SVN checkout → URL of repository http://127.0.0.1:9880/dwsgame → Revision Head Revision → ok. → U-N: maja pwd: maja → k
→ observe green color checkmarks for the files that are gathered from repository.

Note: [If the green color checkmarks for the files that are gathered from repository] *

If the green color checkmark is not coming

(a) R.c. on empty area → tortoise SVN → settings → Icon overlays → Network drives.

(b) task manager → file → New task → open explorer.exe → ok.

steps: Modify any file content of checkedout project & commit it.

R.c. on Resource file → SVN commit → Message changes → maja → ok → U-N maja pwd: maja → ok → ok.

Perform some following operation from maja work folder:

R.c. on file → tortoise SVN diff with previous version → show log file

R.c. on file → tortoise SVN → check for modifications.

R.c. on file → tortoise SVN → Revision graph.

EclipseMarket place for plugin to install SVN in Eclipse working with Eclipse IDE:

Step1: Launch Eclipse IDE for Maya user with diff workspace.

Step2: Add SVN plugin to Eclipse IDE (either using Install SW option (B) using Eclipse Market place.

Help menu → Install SW → work with SVN - http://subclipse.tigris.org/
update-1.8.x.

Step4: launch SVN Repository window.

Window menu → Show View → Other → SVN → SVN Repositories → ok.

Step5: Link Eclipse IDE with user SVN Repository.

SVN Repositories window → New → Repository Location . url : http://127.0.0.1:9880

/dungarep1 → finish → u.n: maja pwd: maja → ok.

Step6: checkin total project to repository

R.c. on Project (Testproj1) → team → ShareProject → SVN → Next → dungarep1 → Next → Next → Initial Import → finish → u.n: maja, pwd: maja → ok.

Step7: modify test.java for multiple times & perform multiple commit operations

R.c. on test.java → team → Commit → change maja → ok.

Step8: We can also perform , replace with compare with operations on test.java

as discussed in cvs class.

Step1: launch Eclipse IDE with diff workspace for maja user

Step2: checkout project from repository from maja user.

→ file menu → Import → SVN → checkout projects from SVN → create new repository location → url http://127.0.0.1:9880/dungarep1 → Next → u.n: maja, pwd: maja → ok → testproj1 → Next → checkout as a Project in the workspace → Next → finish.

Step3: Perform one to checkin / commit operation on test.java from maja user.

Step4: Make maja user get in modifications done by saroj user.

R.c. on test.java (maja Eclipse IDE window) → team → update to Head.

30/7/19

Jasper Report

→ while developing project report generation is very important aspect to analyse & summarize the data.

ex: Attendance Report, Sales report, Inventory report & etc.

→ There are lot of 3rd party APIs/Tools to generate reports in various formats.
like XML, HTML, PDF, MS-Excel & etc document.

→ The report generation tools are

- (1) Crystal Reports.
- (2) Data Reports.
- (3) Accrete Reports.
- (4) Jasper Reports.
- (5) iReport.
- (6) Display Tags (only web level).

Jasper Reports:

Vendor: JasperSoft

Type: Report Tool for Java Environment.

Version: 3.x

To download it: Download as zip file from www.jaspersoft.com
↓
Jasperreports-3.7.5-project.zip

<JasperReports-home>\dist\ Jasperreports-3.7.5 jar file represents the
JasperReports API.

→ JasperReport is an open source Java based reporting tool given by JasperSoft
having the ability to ^{design the} generate report, to fillup the report with data & to
generate the report in various format.

Report Page

Page Header

Date: 20/07/2018

Title

Attendance Report

Column Headers

Sno	Sname	Sadd	Attendance %.
-----	-------	------	---------------

Details

101	maja	hyd	89%
353	navi	vizag	84%

Column Footer

Note: This is attendance of Academic year 2012-13

Page Footer

Page 1 of 10

④ All copy rights are reserved.

Procedure to develop reporting Appn by using Jasper reports:

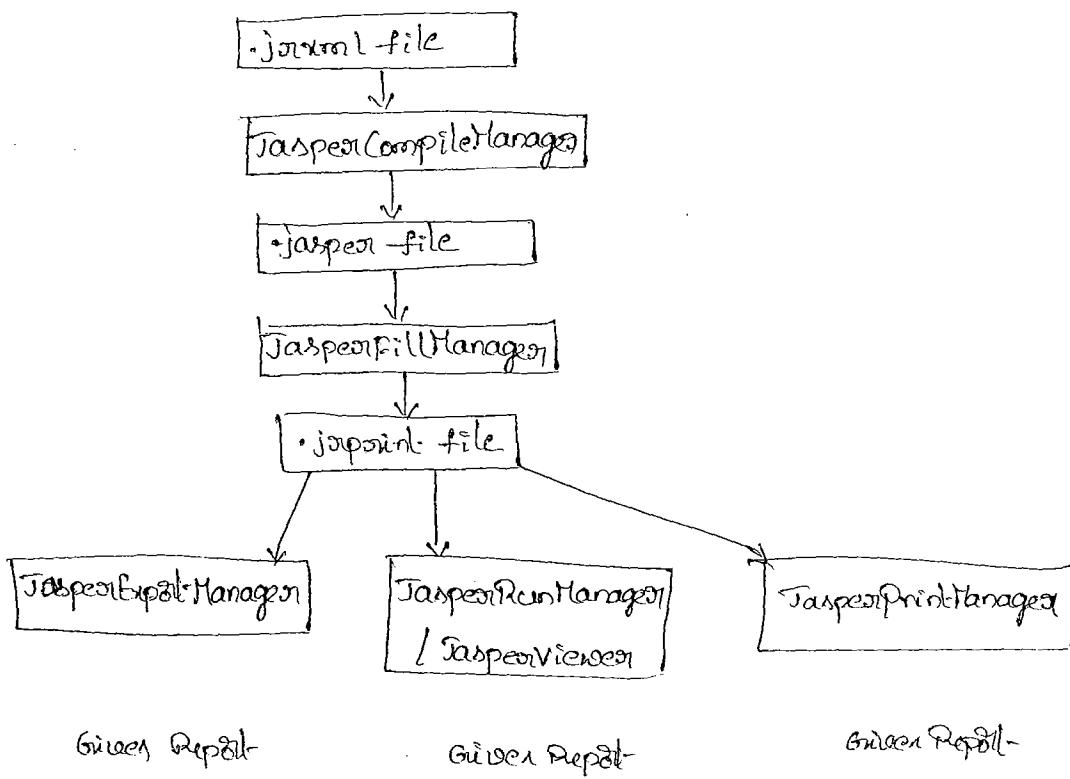
Step1: Prepare .jrxml file to design the report.

Step2: Compile the report by using ^{Jasper}*compile manager.

Step3: Fill the report with data by using jasperfill manager.

Step4: View & print or Export report by using Jasper viewer or JasperPrint Manager
(④ JasperExport Manager).

• jasper file represents only the designing of the report. whereas an .jprint file represents report with data



- For detailed info refer page no's: 112 to 184.
 - We can use `compileXXX()` methods on `JasperCompileManager` class to compile a `.jrxml` file.
 - We can also use `fillXXX()` methods of `JasperFillManager` to fillup the report with data.
 - We can use `printXXX()` methods of `JasperPrintManager`, (2) `ExportXXX()` methods of `JasperExportManager`, (3) `runXXX()` methods of `JasperPrintManager` to generate the report in different format.
 - For example appn on Jasper report refer page no's: 288 to 290. &
Page no's: 121 to 133.
 - Example appn to generate the report- without explicitly generating .jasper, .jreport files :
- FinalReport.jrxml:
- refer page NO: 288.

FirstReport.java:

```

import java.util.HashMap;
import net.sf.jasperreports.engine.*;
import net.sf.jasperreports.view.*;

public class FirstReport
{
    public static void main(String[] args) throws Exception
    {
        JasperReport jsr = JasperCompileManager.compileReport("firstreport.jrxml");
        JasperPrint jp = JasperFillManager.fillReport(jsr, new HashMap(), new JREmptyDataSource());
        JasperViewer.viewReport(jp); // display the report.
        System.out.println("Done");
    }
}

```

→ while developing .jrxml file we can place Parameters, fields, Expressions & variables.

→ For related info refer Page nos: 114 to 121.

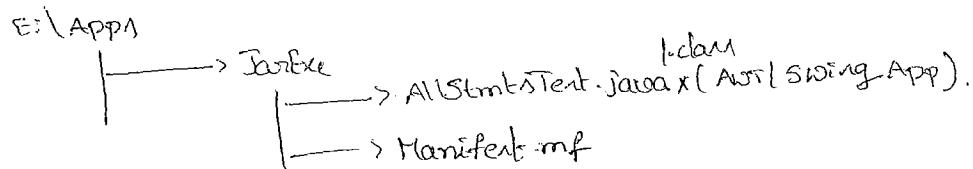
22/7/13

→ JasperSoft has given a tool called iReport to simplify designing of report & to generate .jrxml file dynamically.

→ For information of iReport along with examples refer ~~apple~~ page nos: (134) to (138)

Converting A Java Appn Into An Exe file

Step 1: keep the Appn ready with Manifest file (GUI Appn).



Manifest.mf :-

Manifest-Version: 1.0

Main-class: AllStartTest

space must be replaced.

Step2: Prepare jar file representing the above appn including manifest file.

E:\App\source> jar cvf AllStartTest.jar

E:\App\source> jar cvf manifest.mf App1.jar

Here m means include manifest file while creating jar file.

Step3: Install jar2exe s/w by downloading it from following url

<http://www.rogelab.com/download/noc/jar2exe> (as jar-free.zip file).

This file extraction gives setup file to install the s/w.

Step4: Use the above tool to convert App1.jar file to exe file.

Launch jar2exe tool → next → browse & select the above App1.jar → next → next → ① windows GUI Appn. → Browse & select splash image → next → ② Encrypt & hide class files → next → Add → Browse & select App1.jar file as dependent jar → next → choose icon → next → finish.
→ Launch App1.exe file (just double click on it)

→ when we deliver project in the exe file the person who receives the exe file must keep all dependent s/w's ready & manually on his own. To overcome this

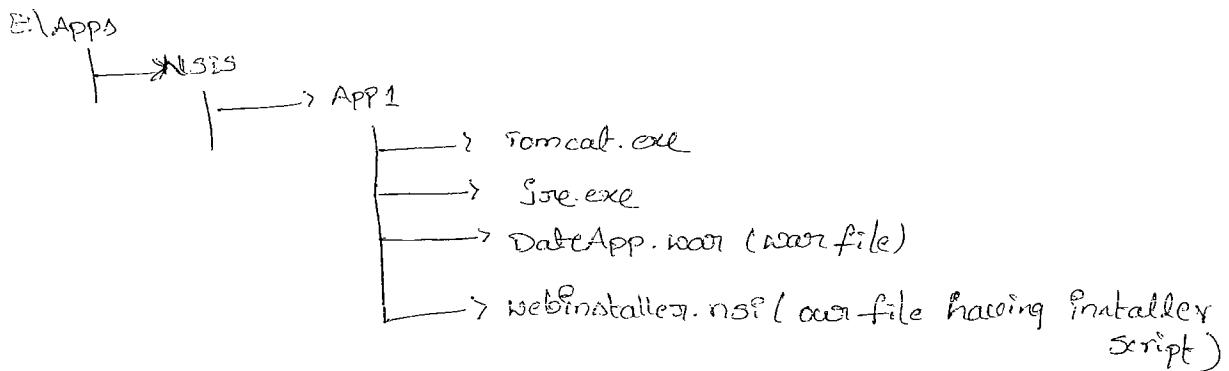
problem we can use NSIS tool to prepare setup file having Appn & dependent s/w's.

(NULLSOFT Installation System)

Ex:-

Step1: Download & install NSIS s/w. nsis-2.45-setup

Step2: Keep the following resources ready.



Step3: Develop .nsi file ~~(files)~~.

webinstaller.nsi

Refer Page no: 294 & 295.

Step4: Run the .nsi file.

Right click on webinstaller.nsi → compile NSIS Script- (given) ~~setup file~~
myjavaSoft.war.)

Step5: launch the setup file

MAVEN

31/7/13

→ It is a build tool given as alternate & enhancement of ant tool.

→ Ant tool is just build tool to build the projects. but the Maven tool can be used in multiple angles like as build tool, as project management tool, as source code Management tool & etc.

Q What is the diff b/w Ant tool & Maven tool?

Ant

→ Doesn't give built-in plugin's to complete → gives the requirement.

→ Ant is Procedural (Everything should be specified manually).

→ NO Lifecycle.

→ Build Scripts are not reusable.

→ Doesn't support Inheritance & multi-modules.

→ Default build file is build.xml

→ It is just build tool.

→ For related info refer Page no's: 139 to 205 & refer page no's: 296 to 307

→ How to configure The Maven tool nicely to use?

Refer Page no: 299.

→ For example appn on maven refer Page no's: (296) to (298) & (181) to (204).

Maven

→ Maven is declarative (plugin's take care of so many activities).

→ Lifecycle with multiple Phases

→ Build script's are reusable.

→ ~~support~~.

→ Default build file is ~~build.xml~~ pom.xml

→ It is build tool, Project Management tool & Sourcecode Management tool & etc.

Sub 219

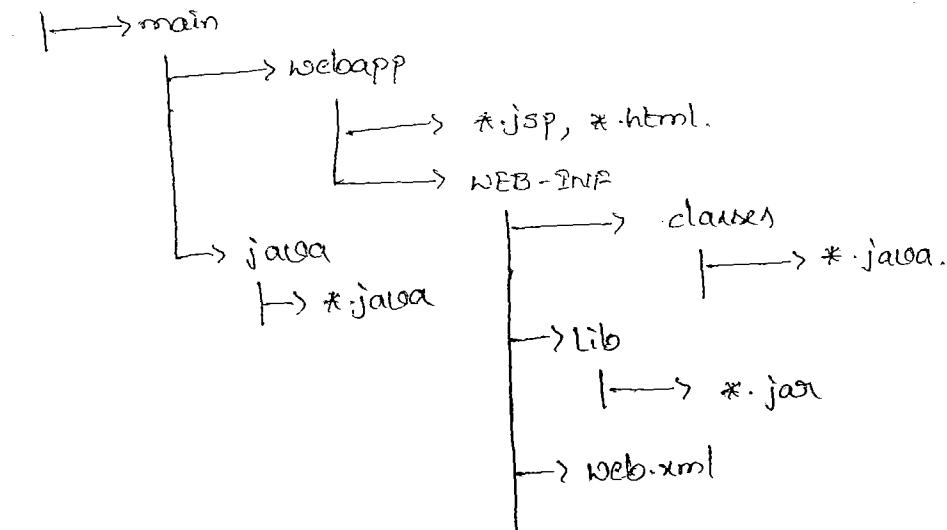
→ The Maven supplied maven-ant-tasks plugin allows to perform ant operations through pom.xml file.

→ For example appn That deploys web appn in Tomcat- Source refer page NO:

(297) & (298) of The Booklet.

→ The common directory structure of Maven webappn is

SRC



Note: The content arranged in the above webappn can be used by Maven plugin to generate war file dynamically.

~~Eclipse~~

Procedure to develop Hibernate appn with the support of Maven by using Eclipse IDEs.

Note: In Eclipse Kepler x Maven ~~plugin~~ ~~is built-in~~ in ~~plugins~~.

In other flavours of Eclipse ~~the~~ ^{This} plugin should be added Separately (Manually).

Step1: Add Eclipse plugin to work with Maven.

Step2: HelpMenu → Install new SW → Add → Mvnplugin ,
Name:

Location: <http://download.eclipse.org/technology/m2e/releases> →

M2E - Maven Integration for Eclipse → next.

Step3: Create Java Project having name "HBproj1".

Step4: Convert into Maven Project.

Right click on project → configure → convert to Maven Project →

Group Id: org.hibernate , Artifact Id: HBproj1 , Revision: 0.0.1-SNAPSHOT ,

Plug-in: maven (Below of this pom.xml will come in our project).

Steps: Gather Hibernate related jars information from Mavenrepository.com

Mavenrepository.com → Hibernate → Hibernate Annotations → org.hibernate →
hibernate-core → Select Version 4.1.2.Final → gather info

```
<dependency>
  <groupId> ... </groupId>
  <artifactId> ... </artifactId>
  <version> ... </version>
</dependency>
```

Step5: Add dependencies to pom.xml

pom.xml → dependencies tab → Add → GroupId: org.hibernate ,
ArtifactId: hibernate-core , version: 4.1.2.Final , scope: runtime → ok
(8)

pom.xml → dependencies tab → Add → Enter groupid, ArtifactId.
[org.hibernate] → Search
& select → ok .

Step6: Fill up a ~~src~~ folder with hibernate4.x appn ~~resources~~.

Step7: Add ejbclient.jar file manually to buildpath of the project.

Step8: Run the hibernate client appn.

→ For above steps based procedure refer page NO: (191) to (197)

Step9: For Haven based struts appn development using Eclipse IDE refer

page NO: (183) to (191).

→ Developing Example appn to demonstrate Inheritance to Haleden.

Step1: keep The above HBProj1 project ready as base project.

Step2: Add The following packaging tag in ~~POM~~. pom.xml after <version> tag of ~~<parent>~~ <project> tag

<packaging> pom </packaging> (^{For} ~~use~~ parent project).

Note: R.c.on Project → Haleden → update Project.

Step3: Create another java project as child project (HBProj2) → Next → finish.

Step4: convert child Project to Haleden Project.

R.c.on project → configure → convert to Haleden Project →

Group Id: dig.durga → finish.

Step5: Add <parent> tag in pom.xml of child Project having HBProj1 details.

```
<parent>
  <groupId> dig.durga </groupId>
  <artifactId> HBProj1 </artifactId>
  <version> 0.0.1-SNAPSHOT </version>
</parent>
```

All these details are gathered from pom.xml of HBProj1.

Note: Do observe that HBProj2 ~~gets~~ All the jars that are there in HBProj1 Project.

Procedure to develop MultiModule project using Maven & Eclipse (Helios)

Step1: Create Maven Project.

File menu → new → other → Maven → Maven Project → next → next →

Create a simple project (if our project is ~~not~~ a webapp then select this one).
Packaging: POM

→ GroupId: dig.durga , artifactId: EmpProj1x → next → finish.

Step2: Add Mod1 as Maven Module to the Project.

File menu → new →

R.C. on EmpProj1 → new → Other → Maven → Maven module → next →

Create a simple project, Module Name: mod01 → next →

GroupId: com.dss, ArtifactId: mod01, packaging: jar → next → finish.

Step3: Add java class to mod01.

R.C. on Project → New → class → package: p1, name: calc → finish [X]

Step2: keep pom.xml source code of above project in open mode.

Step3: Add Mod1 module to Project.

R.C. on Project → New → Other → Maven → Maven Module → next →

Create a Simple Project, Module Name: Mod1 → next → GroupId: dg.durga

, packaging: jar → next → finish.

Step4: Add one java class to Project having B-Method.

R.C. on Mod1 module → New → class → package: p1, name: calc → finish.

Calc.java:

Package p1;
Public class Calc {

 Public int add (int x, int y) {

 Return x+y;

}

3

Step5: Add Mod2 Module to Project as web module.

R.C. on Project → New → Other → Maven → Maven Module → next →

Module Name: mod2 → next → select Maven-archetype-webapp → next → finish.

Step6: open pom.xml of mod2 & write <dependencies> tag to use the mod1 module related jar file.

In pom.xml of mod2 module write the following additional dependency > 

```
<dependencies>
<dependency>
    <groupId> org-dcrga </groupId>
    <artifactId> mod1 </artifactId>
    <version> 0.0.1-SNAPSHOT </version>
}
collect from pom.xml
of Mod1 module
</dependency>
</dependencies>
```

↳ mod2\src\main\webapp\index.jsp

Write following code in the index.jsp of mod2 module.

index.jsp:

```
<%@page import="p1.Calc"%>
<% out.println("Result is " + new Calc().add(23,45)); %>
```

Step7: Perform Maven build on the project.

R.C. on Project → Run As → Maven build → Goals: clean install,
 update Snapshot, skip Tests → Apply → Run.

Note: If the above step gives MojoFailureException perform the following activity.
window menu → Preferences → java → Installed JRE → Java → Edit → Add External JAR select tools.jar file from <java-home>\lib folder → finish → ok. → Perform Maven build once again & get mod2.war file in mod2 module target folder.

Step8: Deploy mod2.war file in Tomcat server. (copy mod2.war file to <Tomcat-home>\webapps\ folder).

Step9: Test the Appln.

http://localhost:2020/mod2

~~Encryption~~ Password Encryption:

Procedure to encrypt password by using diff algorithms :

Step1: Download & extract janypl-1.9.0-dist.zip file.

Step2: Add <janypl-home>/lib/janypl-1.9.0.jar file to classpath.

Step3: Develop The appln. Refer The supplementary handout given on 27/7/13.

Step4: compile & execute The appln.

Procedure to Encrypt the .class file content of jar file :

Step1: Create jar file having ur appln. But this jar file must have one class having main().

javachat1.jar

Step2: Download proguard4.1obeta1.jar file & extract it.

Step3: Prepare one .pro script file specifying the instructions.

Script.pro :

-injars javachat1.jar

-outjars javachat1-out.jar

-libraryjars <java-home>/lib/alt.jar

-printmapping proguard.map (generates a map file)

-overloadaggressively

-repackageclasses ''

-allowaccessmodification (it allows to make every member variable as public variable).

-keep public class Javachat1 {

 public static void main (java.lang.String []);

}

The process of encrypting .class file content of jar file & optimizing the content by removing unused code is called obfuscation.

Script.pro:

Step 4: Run the above created Script.pro file to get encrypted classes based jar file.

> java -jar <proguard-home>\lib\proguard.jar @Script.pro

Specify the final location folder of proguard file.

→ The above command gives Javachat1-out.jar file having encrypted .class files.

Steps: Compare the .class file source code of both Javachat1.jar & Javachat1-out.jar